



2008

AS & A2 course texts

2008 Specification

A Level **Computing**

Roger Blackford

Chris Leadbetter

Welcome to this latest edition of the B & L notes that accompany the OCR course. The look of the files is somewhat different this time and we hope that you are pleased with the result. Hopefully it looks somewhat more impressive, though the content still works on the same principle, that the bullet points in the specification are covered in strict chronological order and we try to give a clear explanation of the expectations on a candidate when they go in to the examination room. The new look is down to James and Roger and I thank him for all his efforts which are certainly above and beyond...

The way you decide to use the content is entirely up to you within your centre, we simply ask that the contents remain within the centre that has purchased the license.

Any queries that you may have please don't hesitate to contact us at c_leadbetter@hotmail.com just be aware that at certain times of year the reply may be a bit delayed for obvious reasons.

No more to be said except to hope that you find the contents of use and to wish you Happy Teaching/Learning.

Chris Leadbetter

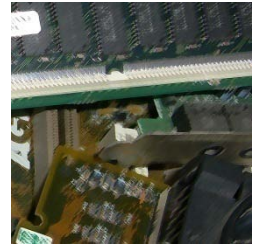
Section			Title	Page
Book 1 / AS			Computer Fundamentals	7
1	1		<i>Components of a Computer System</i>	7
1	1	a&b	Define Terms and State the Purpose of Peripheral Hardware Devices	7
1	1	a&c	Define Terms and Explain the Use of Different Types of Software	8
1	1		Example Questions	9
1	2		<i>Software</i>	10
1	2	a	Describe the Stages of the Systems Life Cycle	10
1	2	b	Explain the Importance of Defining a Problem Accurately	11
1	2	c	Describe the Function and Purpose of a Feasibility Study	12
1	2	d	Explain the Importance of Determining the Information Requirements of a System and Describe Different Methods of Fact Finding	13
1	2	e	Describe what is Involved when Analysing Requirements of a System, Explaining the Nature of the Requirements and its content	15
1	2	f	Describe a Design Specification, Including Input Design, Diagrammatic Depiction of the Overall System, Processing, Data Structure Design and Output Design	17
1	2	g	Explain the Importance of Evaluating the System, and how to Identify the Criteria Used for Evaluation	19
1	2	h	Explain the Content and Importance of Different Types of Documentation at Different Stages of the System Life Cycle, Including the Technical and User Manuals	20
1	2	i	Explain the Importance of System Testing and Installation Planning	22
1	2	j	Explain the Purpose of Maintaining the System, and Explain the Need for System Review and Reassessment	23
1	2	k	Describe Prototyping to Demonstrate how a Solution will Appear	24
1	2	l	Describe the Spiral and Waterfall Models of the System Life Cycle	25
1	2	m	Identify the Features of Common Applications Found in Business, Commercial and Industrial Applications	26
1	2	n	Identify and Justify Generic Applications Software for Particular Application Areas	27
1	2	o	Identify and Justify Application Areas for which Custom-written Applications Software is Appropriate	30
1	2	p	Describe the Characteristics of Knowledge Based Systems	31
1	2	q	Describe the Purpose of Operating Systems	32
1	2	r	Describe the Characteristics of Different Types of Operating System and their Uses	33
1	2	s	Describe a Range of Applications Requiring Batch Processing and Applications in Which a Rapid Response is Required	35
1	2	t	Identify and Describe the Purpose of Different Types of User Interface	36
1	2	u	Discuss the Importance of Good Interface Design	39
1	2	v	Identify and Describe the Purpose of a Range of Utilities	40
1	2		Example Questions	41
1	3		<i>Data: Its Representation, Structure and Management in Information Systems</i>	48
1	3	a	Express Numbers in Binary, Binary Coded Decimal (BCD), Octal and Hexadecimal	48
1	3	b	Describe and Use two's Complement and Sign and Magnitude to Represent negative Integers	53
1	3	c	Perform Integer Binary Arithmetic: Addition and Subtraction	55
1	3	d	Explain the Use of Code to Represent a Character Set (ASCII, EBCDIC and UNICODE)	57
1	3	e	Describe Manual and Automatic Methods of Gathering and Inputting Data into a System	58
1	3	f	Explain the techniques of Validation and Verification, and Describe validation Tests that can be Carried out on Data	61
1	3	g	Describe Possible Forms of Output, Stating the Advantages	64
1	3	h	Explain the Procedures Involved in backing up Data and Archiving, Including the Difference Between data that is Backed up and Data that is Archived	66
1	3		Example Questions	67
1	4		<i>Hardware</i>	72
1	4	a	Describe the Function and Purpose of the Control Unit, Memory unit and ALU as Individual Parts of the Computer	72
1	4	b	Explain the Need for and the Use of Registers in the Functioning of the Processor	73
1	4	c	Explain the Need for and Describe the Use of Buses to Convey Information	74
1	4	d	Describe the Connectivity of Devices	75
1	4	e	Describe the Differences Between Types of Primary memory and Explain Their Uses	76
1	4	f	Describe the Basic Features, Advantages, Disadvantages and Uses of Secondary Storage Media	78
1	4	g	Describe the Transfer of Data Between Different Devices and Primary Memory, Including the Use of Buffers	81
1	4	h	Describe a Range of Common Peripheral Devices in Terms of Their Features, Advantages, Disadvantages and Uses	83
1	4	i	Describe and Justify the Appropriate Peripheral Hardware for a Given Application	87
1	4		Example Questions	88
1	5		<i>Data Transmission</i>	93
1	5	a	Describe the Characteristics of a LAN and a WAN	93
1	5	b	Show an Understanding of the Hardware and Software Needed for a LAN or WAN	94
1	5	c	Describe the Different Types of Data Transmission	95
1	5	d	Explain the Relationship Between Bit Rates and the Time Sensitivity of Information	97
1	5	e	Recognise that Errors Can Occur in Data Transmission and Explain Methods of Detecting and Correcting These Errors	98
1	5	f&g	Describe Packet Switching and Circuit Switching, Explaining the Difference in Use	99
1	5	h	Define the Term Protocol and Explain the Importance of a Protocol in the Transmission of Data	100
1	5	i	Describe the Need for Communication Between Devices and Between Computers, and Explain the Need for Protocols to Establish Communication Links	101

1	5	j	Explain the Need for Both Physical and Logical Protocols and the Need for Layering in an Interface	102
1	5		Example Questions	103
1	6		<i>Implications of Computer Use</i>	107
1	6	a	Discuss Changing Trends in Computer Use and their Economic, Social, Legal and Ethical Effects on Society	107
1	6	b	Explain Changes to Society Brought About by the Use of Computer Systems	109
1	6	c	Discuss the Effects on Privacy and Confidentiality of Data Held in Computer Systems and Steps Which can be Taken to Protect Confidentiality	110
1	6	d	Understand the Need for Legislation Governing Computer System Use	111
1	6		Example Questions	
BOOK 2 / AS		Programming Techniques and Logical Methods		114
2	1		Designing Solutions to Problems	114
2	1	a	Discuss the Importance of Good Interface Design	114
2	1	b	Design and Document Data Capture Forms, Screen Layouts, Report Layouts or Other Forms of Input and Output (eg: Sound) Given a Problem	116
2	1	c	Determine the Requirements of a Program	129
2	1	d	Explain the Advantages of Designing a Solution to a Problem by Splitting it up into Smaller problems (top-down/modular design)	130
2	1	e	Produce and Describe Top-Down/Modular Designs Using Appropriate Techniques, Including Structure Diagrams, Showing Stepwise Refinement	131
2	1	f	Produce Algorithms to Solve Problems	132
2	1	g	Describe the Steps of an Algorithm Using a Program Flowchart	134
2	1	h	Describe the Steps of an Algorithm Using Pseudo-Code	136
2	1	i	Understand and Implement Algorithms and Evaluate them by Commenting on their Efficiency, Correctness and Appropriateness for the Problem to be Solved	137
2	1	j	Describe the Use of Rapid Application Development (RAD) as a Design Strategy, Including Prototyping and Iterative Development	138
2	1		Example Questions	140
2	2		<i>The Structure of Procedural Programs</i>	144
2	2	a	Define and Correctly use the Following Terms as they Apply to Procedural Programming; Statement, Subroutine, Procedure, Function Parameter/argument, Sequence, Selection, Iteration/repetition, Loop	144
2	2	b	Identify the Three Basic Programming Constructs used to Control the Flow of Execution	148
2	2	c	Understand and Use Selection of Pseudo-code and the Procedural Programming Language, Including the use of IF Statements and CASE/SELECT Statements	149
2	2	d	Understand and Use Iteration in Pseudo-code and a Procedural Programming Language, including the use of count-controlled loops	153
2	2	e	Understand and Use Nested Selection and Iteration Statements	156
2	2	f	Understand, Create and Use Subroutines, including the Passing of Parameters and the Appropriate use of the Return Value of Functions	158
2	2	g	Identify and Use Recursion to Solve Problems; Show an Understanding of the Structure of a Recursive Subroutine Including the Need for a Stopping Condition	163
2	2	h	Trace the Execution of a Recursive Subroutine Including Calls to Itself	166
2	2	i	Discuss the Relative Merits of Iterative and Recursive Solutions to the Same Problem	168
2	2		Example Questions	169
2	3		<i>Data Types and Data Structures</i>	174
2	3	a	Define Different Data Types, Boolean, Character and String; Select and use them Appropriately in the Solutions to Problems	174
2	3	b	Define and Use Arrays for Solving Simple Problems, Including Initialising Arrays, Reading Data into Arrays and Performing Simple Serial Search on a one-dimensional Array	175
2	3	c	Explain the Advantages and Disadvantages of Different Data Types and Data Structures for Solving a Given Problem	179
2	3	d	Design and Implement a Record Format	182
2	3	e	Define Different Modes of File Access: Serial, Sequential, Indexed Sequential and Random; and Justify a Suitable Mode of File Access for a Given Example	185
2	3	f	Store, Retrieve and Search for Data in Files	187
2	3	g	Estimate the Size of a File from its Structure and the Number of Records	188
2	3	h	Use the Facilities of a Procedural Language to Perform File Operations	189
2	3		Example Questions	194
2	4		<i>Common Facilities of Procedural Languages</i>	196
2	4	a	Understand and Use Assignment Statements	196
2	4	b	Understand Arithmetic Operators Including Operators for Integer Division	197
2	4	c	Understand a Range of Operational Operators & use these to Construct Expressions	198
2	4	d	Understand the Boolean Operators AND, OR and NOT and use these to Construct Expressions	199
2	4	e	Understand the Effects of the Precedence of Standard Operators and the use of Parentheses to Alter the Order of Precedence	200
2	4	f	Evaluate Expressions Containing Arithmetic, Relational and Boolean Expressions	202
2	4	g	Understand and use a Range of Operators and Built in Functions for String Manipulation, Including Location, Extraction, Comparison, Concatenation, Determining the length of a String and Converting Between Characters and their ASCII Code	204
2	4	h	Understand that Relational Operations on alphanumeric Strings Depend on Character Codes of the Characters and Explain the Results of this Effect	206
2	4	i	Input and Validate Data	207
2	4	j	Output Data onto Screen/File/Printer, Formatting the Data for Output as Necessary	208
2	4		Example Questions	209
2	5		<i>Writing Maintainable Programs</i>	212

2	5	a	Define, Understand and use the Following Terms Correctly as they Apply to Programming: Variable, Constant, Identifier, Reserved word/ keyword	212
2	5	b	Explain the Need for Good Program-writing Techniques to Facilitate the Ongoing Maintenance of Programs	213
2	5	c	Declare Variables and Constants, Understanding the Effect of Scope and Issues Concerning the Choice of Identifier (Including the need to avoid reserved words/keywords)	214
2	5	d	Select and Use Meaningful Identifier Names and Use Standard Conventions to Show the Data Types and Enhance Readability	217
2	5	e	Use Declared Constants to Improve Maintainability	218
2	5	f	Initialise Variables Appropriately, Before Using them	219
2	5	g	Create Appropriately Modularized Programs, Making Effective use of Subroutines to Improve Maintainability	220
2	5	h	Annotate the Code with Comments so that the Logic of the Solution can be Followed	222
2	5	i	Use Indentation and Formatting to show Clearly the Control Structures Within the Code	224
2	5		Example Questions	225
2	6		<i>Testing and Running a Solution</i>	226
2	6	a	Describe Types of Errors in Programs and Understand how and when these may be Detected	226
2	6	b	Identify why/where an Error may Occur in an Algorithm and State how the Algorithm may be Corrected	227
2	6	c	Describe Testing Strategies Including White Box Testing, Black Box Testing, Alpha Testing, Beta Testing and Acceptance Testing	229
2	6	d	Select Suitable Test Data for a Given Problem, Including normal, Borderline and Invalid Data	230
2	6	e	Perform a Dry Run on a Given Algorithm using a Trace Table	231
2	6	f	Describe the use of a Range of Debugging Tools and Facilities Available in Procedural Programming Languages Including Translator Diagnostics, Break Points, Stepping, and Variable Checks	232
2	6	g	Describe the Purpose of an Installation Routine in the Delivered Version of a Program	233
2	6		Example Questions	234
BOOK 3 / A2			Advanced Computing Theory	237
3	1		<i>The Function of Operating Systems</i>	237
3	1	a	Describe the Main Features of Operating Systems- ie; Memory Management, Scheduling Algorithms	237
3	1	b	Explain how Interrupts are Used to Obtain Processor Time and how Processing of Interrupted Jobs may Later be Resumed	241
3	1	c	Define and Explain the Purpose of Scheduling, Job Queues, Priorities and how they are used to Manage job Throughput	244
3	1	d	Explain how Memory is Managed in a Typical Modern Computer System	249
3	1	e	Describe Spooling and How it is Used	252
3	1	f	Describe the Main Components of a Typical Desktop PC Operating System, Including the File Allocation Table (FAT) and how it is used, and the Purpose of the Boot File	253
3	1		Example Questions	256
3	2		<i>The Function and Purpose of Translators</i>	259
3	2	a	Describe the Need for and use of Translators to Convert Source Code to Object Code	259
3	2	b&c	Understand the Relationship Between Assembly Language and Machine Code. Describe the use of an Assembler in Producing Machine Code	260
3	2	d	Describe the Difference Between Interpretation and Compilation	262
3	2	e	Describe the Purpose of Intermediate Code in a Virtual Machine	263
3	2	f,g,h	The Stages of Translation; Lexical Analysis, Syntax Analysis, and Optimisation	268
3	2	i	Describe the use of Library Routines	272
3	2		Example Questions	273
3	3		<i>Computer Architectures</i>	279
3	3	a	Describe Classic Von Neumann Architecture, Identifying the need for, and the uses of, Special Registers in the Functioning of a Processor	279
3	3	b	Describe, in Simple Terms, the Fetch/Decode, Execute Cycle, and the Effects of Stages of the Cycle on Specific Registers	281
3	3	c	Discuss co-Processor, Parallel Processor and Array Processor Systems, their uses, Advantages and Disadvantages	282
3	3	d	Describe and Distinguish Between Reduced Instruction Set Computer (RISC) and Complex Instruction Set Computer (CISC) Architectures	284
3	3		Example Questions	285
3	4		<i>Data Representation</i>	288
3	4	a	Demonstrate an Understanding the Floating Point Representation of a Real Floating Point Binary Number	288
3	4	b	Normalise a Real Binary Number	289
3	4	c	Discuss the Trade-off Between Accuracy and Range when Representing Numbers	290
3	4		Example Questions	291
3	5		<i>Data Structures and Data Manipulation</i>	293
3	5	a	Explain how Static Data Structures may be used to Implement Dynamic Data Structures	293
3	5	b	Describe Algorithms for the Insertion, Retrieval and Deletion of Data Items Stored in Stack Queue and Tree Structures	295
3	5	c	Explain the Difference Between Binary Searching and Serial Searching, Highlighting the Advantages and Disadvantages of Each	300
3	5	d	Explain how to Merge Data Files	303
3	5	e	Explain the Differences Between the Insertion and Quick Sort Methods, Highlighting the Characteristics, Advantages and Disadvantages of Each	305
3	5		Example Questions	311
3	6		<i>High Level Language Programming Paradigms</i>	314
3	6	a	Identify a Variety of Programming Paradigms	315
3	6	b	Explain with Examples, the Terms Object-Oriented, Declarative, and Procedural as Applied to High Level Languages, Showing an Understanding of Typical uses	317

3	6	c	Discuss the Concepts, and using Examples, Show an Understanding of, Data Encapsulation, Classes and Derived Classes, and Inheritance when Referring to Object-Orientated Languages	323
3	6	d	Understand the Purpose of the Unified Modelling Language (UML)	327
3	6	e&f	Create and Interpret Class, Object, use Case and Communication Diagrams, and Interpret State, Sequence and Activity Diagrams	328
3	6	g	Discuss the Concepts and, Using Examples, Show and Understanding of Backtracking, Instantiation, Predicate Logic and Satisfying Goals when Referring to Declarative Languages	333
3	7		<i>Programming Techniques</i>	339
3	7	a	Explain how Functions, Procedures and their Related Variables may be Used to Develop a Program in a Structured way, Using Stepwise Refinement	339
3	7	b	Describe the use of Parameters, Local and Global Variables as Standard Programming Techniques	341
3	7	c	Explain how a Stack is Used to Handle Procedure Calling and Parameter Passing	349
3	7	d	Explain the Need for, and be able to Create and Apply, BNF (Backus-Naur Form) and Syntax Diagrams	355
3	7	e	Explain the Need for Reverse Polish Notation	361
3	7	f	Convert Between Reverse Polish Notation and Infix Form of Algebraic Expressions using Trees and Stacks	362
3	8		<i>Low Level Languages</i>	365
3	8	c	Discuss the Concepts and, Using Examples, Show an Understanding of Mnemonics, Opcode, Operand and Symbolic Addressing in Assembly Language to Include Simple Arithmetic Operations, Data Transfer and Flow Control	365
3	8	a	Explain the Concepts and, Using Examples, Demonstrate an Understanding of the use of the Accumulator, Registers, and Program Counter	367
3	8	b	Describe Immediate, Direct, Indirect, Relative and Indexed Addressing of Memory when Referring to Low Level Languages	374
3	9		<i>Databases</i>	376
3	9	a	Describe Flat Files and Relational Databases, Explaining the Difference Between them	376
3	9	b	Design a Simple Relational Database to the Third Normal Form (3NF), Using Entity-Relationship (E-R) Diagrams and Decomposition	379
3	9	c	Define and Explain the Purpose of, Primary, Secondary and Foreign Keys	389
3	9	d	Describe the Structure of a DBMS, Including the Function and Purpose of the Data Dictionary, Data Description Language (DDL) and Data Manipulation Language (DML)	390
BOOK 4 / A2			Project	393
4	0		<i>Computing Project – Introduction</i>	393
4	0	1	General Description	393
4	0	2	Project Selection	394
4	0	3	Problems that May Arise	396
4	0	4	Specification Requirements	397
4	0	5	Format of the Chapters	398
4	a		<i>Computing Project- Definition, Investigation & Analysis</i>	399
4	a	i	Definition- Nature of the Problem to be Investigated	399
4	a	ii	Investigation and Analysis	401
4	b		<i>Computing Project- Design</i>	404
4	b	i	Nature of the Solution	404
4	b	ii	Algorithms	407
4	b	iii	Test Strategy	408
4	c		<i>Computing Project- Software Development and Testing</i>	409
4	c	i	Software Development	409
4	c	ii	Testing	410
4	d		<i>Computing Project- Documentation</i>	411
4	e		<i>Computing Project- Evaluation</i>	413
4	e	i	Discussion of the Degree of Success in Meeting the Original Objectives	413
4	e	ii	Evaluate the User's Response to the System	414
4	e	iii	Desirable Extensions	415

1.1 Components of a Computer System



1.1.a and b *Define terms and state the purpose of peripheral hardware devices*

It is important to explain some terms so that we can use them throughout the rest of the work.

Hardware

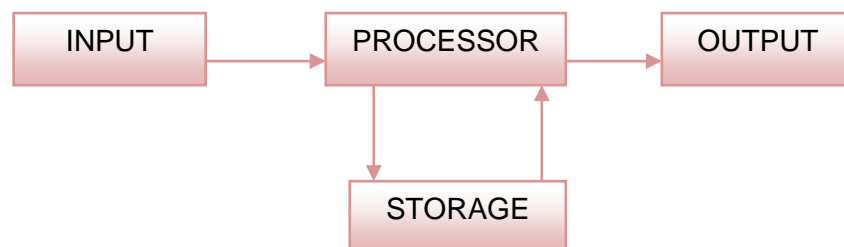
The hardware consists of the physical parts of the computer system.

Peripheral device

These are hardware devices that are outside the computer processor. Typically they are connected to the computer by cables. A printer is an obvious example because it is clear that it is separate from the actual computer itself, but a hard disk drive is also a peripheral, even if it is in the same box as the computer because it is outside the thing that does all the work, the processor itself.

Most peripheral devices are one of three types:

1. An input device is any piece of hardware that allows data to be transmitted to the computer processor. No computer, however powerful, is of value unless it is possible to give it instructions and data to be used when those instructions are carried out.
2. An output device is any piece of hardware that allows the computer processor to convey information or the results of its processing. There is no point in having a computer system carrying out processing of any sort unless the results of that processing can be either reported or used to affect future input to the system.
3. Storage is any piece of hardware which can store data outside the processor in a form which is suitable for input back into the processor. This is necessary because data must be saved for future use if the processor is switched off and for transfer from one machine to another.



The diagram shows a typical set of hardware in a computer system. The arrows show the direction of flow of data and information. Notice that the storage device can be both an input device and an output device, this is what makes it different from both.

Descriptions of the various types of hardware will be given in section 1.4

1.1.a and c *Define terms and explain the use of different types of software*

Software

Software is the set of instructions which makes the computer system do something useful. These sets of instructions are collected together in workable groups known as programs. Without programs of instructions computers would not be able to function because they would not know what to do.

It is important to distinguish between two different types of software.

1. Systems Software: This is sets of instructions that make the hardware of the computer available for use. Included are programs that allow human beings to communicate with the computer, programs that control the movement of data around the computer, programs that turn instructions in human understandable form into instructions that a computer can recognise and many others which will be studied in section 1.2.

2. Applications packages: A piece of applications software is a program, or set of instructions, designed to make the computer carry out some task. It differs from systems software in that it makes the computer do something that is useful for the user. Word word-processing software, or PowerPoint presentation software are both examples of applications software because they both allow the user to produce something that is of general use. Applications software will be covered in more detail in section 1.2 and throughout much of the rest of the specification.

1.1 Example Questions

This section contains example questions based on the work in the first section of module 1. The work is of a very basic level because of its position in the syllabus, and candidates should not be worried about the very fundamental topics involved.



Typical examiner's comments follow each question.

1. Explain the difference between hardware and software. (2)

A. -Hardware comprises the physical parts of the computer system, while...

-the software are the programs that make the machine produce useful results.

Notice that the answer suggested provides a link between the two terms. This was expected because of the use of the words 'explain the difference' in the question. There are two marks available for the question so make sure that you have said two distinct things. Make sure that the second point is not just the opposite of the first. In this example there would be no mark for saying that software are the other parts that are not physical. Normally a question will have more mark points available than there are marks available for the question, but as this is a simple illustrative question there are only two points that can be made.

2. Give **two** reasons why a computer system would need to have some type of external storage device. (2)

A. -In order to store files when the power to the processor is switched off.

-In order to store files which are too large to be stored in the processor itself, until they are needed.

-To allow for the transfer of data from one machine to another.

Notice that the question asks for reasons why the system needs to have storage, NOT for what is stored or in what sort of device it is stored. Be careful to answer the question asked. Note that there are two indicators in the question that you should give two answers, the word in bold and the number of marks. There is a golden rule that should be adopted in exams, don't make the question any more difficult than it already is, so as the question does not ask for a comparison or an explanation, don't give them. Notice that there are more than two acceptable answers. If you can think of three then give them, the examiner will choose the two best, but never give more than one extra answer because you then demonstrate that you are not sure and the examiner will choose your first two answers, right or wrong.

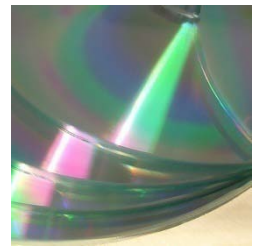
3. Describe the difference between system software and application software. (2)

A. -System software controls how the computer system works while...

-application software allows the user to make the computer do something useful.

Again, this is a comparison type of question so the two parts of the answer should be related.

1.2 Software



1.2.a *Describe the stages of the systems life cycle*

When a problem solution is being produced it is important to follow a specific set of steps in a set order so that steps are not missed out and that assumptions are not made when they should not be.

One of these methodologies of how to solve problems is called the systems life cycle. It is not the only methodology and it is sometimes not the best of the different methodologies, however, it is the most common way of solving problems using computers. It is important to understand the different steps and the logic of the order in which they are presented because the project reports that candidates are going to produce in module 4 of the full A level should be based on the systems life cycle.

Sections 1.2.b to 1.2.i describe the different stages in the standard model and how they fit together logically.

1.2.b *Explain the importance of defining a problem accurately*

It is important from the outset to ensure that when a computer system is being designed all those that are involved are agreed about the aims of the system.

There will normally be a person, a company or an organisation, that decides that a task would benefit from computerisation. This belief normally arises because there is a problem which cannot be solved by previously used methods or similar problems seem to have been solved by computerisation elsewhere. Unfortunately, the owner of the problem probably understands the problem itself quite well, but does not understand the consequences of using computers to try to solve the problem, or, indeed, whether such a solution is even possible. For this reason it is necessary for the organisation to employ a specialist who does understand computers and computerised solutions to problems. This is the systems analyst. Unfortunately, it is probable that the systems analyst is not expert in the area of the problem.

The system analyst's job is to solve the problem by planning and overseeing the introduction of computer technologies. The owner of the problem will be happy if the analyst can introduce a computer solution. The problem arises when the analyst, who doesn't know very much about the business, solves the problem that they think needs solving, while the owner of the problem expects a very different problem to be solved.

It seems obvious that if someone is doing something for someone else that they should make sure that they know what is required. If you are asked to answer questions 1 to 5 and you do 6 to 10 instead there is an obvious breakdown in communication, but this is what can happen if you only listen to the instruction, "Do the 5 questions". The method most often used to overcome this problem is for there to be discussions between all the interested parties, and then for a list of objectives to be written up. The solution to the problem is the successful implementation of all the objectives. The different people involved then agree to the list of the objectives. The success, or otherwise, of the project depends on the completion of those objectives.

The definition of the problem is the most important part of the analysis because if it is not done correctly the wrong problem may be solved. This is quite common and has led to many failed projects.

1.2.c *Describe the function and purpose of a feasibility study*

When the organisation and the systems analyst have agreed the definition of the problem, a decision must be made about the value of continuing with the computerised solution. The organisation may be convinced that there is a problem to be solved, and that its solution will be worth the effort and the expense. However, the systems analyst is being paid to look at the problem, and its solution, from another point of view. The analyst is an expert in computer systems and what is possible with computer systems. This analyst must consider the problem from the point of view of the computerised part of the solution and make a report to the organisation saying whether the solution is possible and sensible. This report is called the feasibility study because it says whether the solution is feasible.

The feasibility study will consider the problem, and the proposed solution, from a number of points of view.

- Is the solution technically possible? A car firm may decide that if robots are used on the production line to assemble the different parts of engines then quality of work may improve. However, if it is not possible to manufacture a robot arm of the correct dimensions to fit into the small areas under a car bonnet, it doesn't matter how big an improvement there would be in the quality of the finished product, it would simply not be technically possible.
- Is the solution economic to produce? Perhaps the robots exist that can be programmed to assemble this engine and the benefits will be worthwhile. However, if the cost of the robots and the program to control them is so great that it puts the car manufacturer out of business, the introduction of the robots is not sensible.
- Is the solution economic to run? It has been decided that there are tremendous benefits in producing a robot production line and the robots are very cheap to buy, but they have so many electric motors, and they are so slow at assembling the engines that it is cheaper to employ people to do the job.
- What is the effect on the human beings involved? If the car plant is the only major employer in the region, and the introduction will put much of the workforce out of work, then the employer may decide that the human cost is too great, certainly the Government would.
- Is the workforce skilled enough? It will be necessary to employ highly skilled technicians to look after the robots. If there are no workers with the necessary skills, then the computerisation is not feasible.
- What effect will there be on the customer? If the customer is likely to be impressed with the introduction of the new systems, it may be worth the expense. However, if the customer will notice no difference, what is the point?
- Will the introduction of the new systems be economically beneficial? Put bluntly, will the introduction of robots increase the profits made by the firm?

Often, the first stage of problem definition is included as one of the stages of the feasibility study.

1.2.d *Explain the importance of determining the information requirements of a system and describe different methods of fact finding, including questionnaires, observation and structured interviews, highlighting the advantages and disadvantages of each method*

If the feasibility study shows a positive result which is accepted by the company, the next stage will be for the analyst to collect as much information about the process as possible.

When a computer system is being designed, it is necessary to ensure that the designer of the system finds out as much as possible about the requirements of the system. We have already mentioned the importance of defining the problem but further difficulties can arise. Imagine an organisation that commissions an analyst to design a new payroll program. The analyst and the boss of the organisation agree that the software needs to be able to communicate with the relevant banks, and to deduct tax at source and other important details. The analyst designs the solution and the software is written. It is the best payroll program ever produced and is installed into the computer system ready to do the first monthly payroll. Unfortunately, many of the workforce is employed on a weekly basis. A simple question to one of those paid in this way would have highlighted this for the analyst and would mean that the costly mistake was avoided. When a system's requirements are being studied there is no room for errors caused by lack of information. It is important, at this stage, to ensure that we all understand what is meant by 'information' in this context. The information being collected is information that the analyst needs in order to understand the problem, not information like the workers' bank account numbers which would be needed to run the computer solution when it is finished.

Once the feasibility study has been accepted the analyst needs to collect as much information about the problem as possible. Obvious methods of collecting information are to ask different types of people. It may not be feasible to ask everyone in the organisation for their views, but a representative sample of workers, management, customers should be given the chance to supply information. After all, the part time worker on the production line probably knows more about the business than the analyst. There are three accepted methods of finding out people's views:

1. By interview. Interviews are particularly important because they allow the interviewee to talk at length, and also allow the interviewer to leave a prepared script so that interesting or unexpected responses can be followed up. However, they are very time consuming and consequently restricting in the number of people whose views can be sought
2. By using questionnaires. Questionnaires make it possible to find out the views of a large number of people very quickly, but because the questions are pre-determined the person who is supplying the answers may find difficulty in putting their point of view across. Also, questionnaires are very difficult to design. Many people have difficulty completing them, particularly if the instructions are not very clear.
3. A compromise is to hold a group meeting. This allows a number of people to discuss points and make their views known and yet cuts down on the amount of time spent in interviews getting the same answers over and over again. The problem with meetings is that one or two people tend to dominate, not allowing all the members of the group to give their opinions.

Often the views of the people connected with the problem are clouded by years of familiarity, so it is important for the analyst to also gain first-hand knowledge of any existing systems. There are two accepted ways of doing this:

1. To observe the current system in action. Care must be taken to take into account the fact that if the workers know that they are being observed it is unlikely that they will behave in their normal manner. This is the same effect as a television camera has on otherwise fairly sensible individuals.
2. Collect printed documentation and study it in order to find out what data is required by the system and in what form information is output.

1.2.e *Describe what is involved when analysing the requirements of a system, explaining the nature of the requirements specification and its content: including current data structures, inputs, outputs and processing represented in diagrammatic form (data flow diagrams, system flowcharts), identify inefficiencies/problems in the current system*

The planning of any system design must start by deciding what the requirements are of the system. A system may need to store data for future reference or processing. However, simply being aware that the system may need to store data is not enough.

- Decisions need to be made about the types of data to be held as this will dictate the form that the data will be stored in and the amount of storage space required for each set of data. Note that this is referring to knowledge about the type of data not the data itself.
- Calculations as to the number of sets of data that are going to be held have to be made because the volume of storage will make some storage devices more sensible than others. Also, the volume of storage can affect the structures that will be used to store the data.
- Decisions need to be made about the relative importance of the different ways of accessing the data. Is it going to be necessary to access individual items of data or will all the data be accessed at the same time? Will the data be changed regularly or is it fairly static?
Again, such decisions will dictate the type of storage devices that will be appropriate to the solution.

When decisions are made as to the direction that a particular system is going to take, it is normal to produce a list of tasks that need to be carried out to complete the system. These tasks should be in a specific order. It would not make sense to consider inputting data into the system before designing the data structure into which the data is to be put. This list is called a priority list or a Task Model and it forms the basis of the system design. Each of the tasks that are in the list should then be considered separately to decide the important points about each one. An example would be that the file of information on stock being held in a business must allow

- for 1000 items (immediately putting a lower limit on the size of appropriate storage)
- for direct searching for information on a single item (means that some storage media are not suitable, and that some data structures cannot be used)
- another file, of manufacturers, to be accessed from each record in the stock file to facilitate reordering (forces one field to act as a link between the two files).

Plenty of other facts must be considered and decisions made, but a set of constraints has already been placed on the design. This is known as the design specification and it should be agreed between the analyst and the organisation before the design is implemented.

Data flow diagrams

These are diagrams that are used to describe systems. Boxes are used to stand for input, processes, storage, and output. Arrows show the direction of communication around the system, and communications outside the system. As the name implies, these diagrams are used to show the directions of flow of data from one part of a system to another. Data flow diagrams can have complex shapes for the boxes that are used, but the important thing is not the shapes of the boxes, rather the logical train of thought that has been used to produce the diagram. Rectangular boxes, though not always used, are perfectly acceptable for all elements in a data flow diagram. Such diagrams are intended to show how the processes and the data interrelate, not the details of any of the programming logic.

Systems Flowcharts

A systems flowchart is a diagram that shows how the elements of a system are associated with each other and how they interact with each other. Included in the detail would be the types of hardware being used the direction of the data flow and the inputs and outputs from the system.

Students would be expected to be able to follow the logic in a data flow diagram and to describe the logical flow of data through a system using a diagrammatic representation, but there will be no requirement to use a particular notation.

At this stage the analyst will also identify the hardware that will be necessary for the system to operate as the user requires, and the software characteristics that will need to be satisfied for the system to operate.

1.2.f *Describe a design specification including input design, diagrammatic depiction of the overall system, processing, data structure design and output design*

In the design specification the analyst will make decisions about the input to the system including the types of input, the hardware to be used, the methods for checking the input data for accuracy and the way the system will communicate with the user. Methods for storing the data collected and methods of manipulation will also be explained as will the output from the system.

Input Design

All systems require input. The way that the data is input to the system depends on a number of factors.

- The data that is required. Is it graphical/textual/physical in nature? Is the data already in existence or does it need to be collected first?
- The hardware that is available. Is the data to be entered via a keyboard by an operator, or is there an automatic way to enter the data?
- The experience of the operator.
- The design of the user interface.

This section relates to section 1.2.t which describes different forms of communication between the computer and the outside world, which should be understood to fully comprehend the importance of the input decisions to be made.

Output Design

The results that are produced by the system must be presented in a way that is appropriate for the application. If the system is designed to produce bank statements for customers, then it would not be sensible to have an audio output. Similarly, a burglar alarm system would not serve the purpose for which it had been designed if the output is a message on a computer screen saying that someone has broken in as there is probably no-one in the house to read it, which is why the alarm was needed in the first place.

The decision about the type of output will depend greatly upon the same factors as the input, namely, the hardware available, the form that the output needs to be in, the experience of the operator, indeed, whether there will be an operator present.

Equally important to giving enough information, is the danger of providing too much. In order for users to be able to understand the information presented, various tricks can be used.

- Information can be arranged on a monitor screen by always putting the same sort of information in the same place. The operator will then quickly become accustomed to the relative importance of different areas of the screen.
- Information can be colour coded. Important information may appear in red while that which is less important is in black. Notice that this implies some form of decision making on the part of the processor to determine what is important in the first place. It is also necessary to be very careful about choice of colours. People who are colour blind commonly find difficulty distinguishing between red and green, so choosing these colours to stand for information that is dangerously near a limit or at a safe level, could be disastrously wrong. Similarly, colour combinations have to be chosen carefully. Blue writing on a black background is almost impossible to see, as is yellow writing in some lighting conditions.
- Video reversal can be used to highlight a particular piece of information effectively. This is when the normal writing on the screen is black on a white background, but the piece that needs to stand out is shown as white on a black background.
- Very important pieces of information may be shown as a dialogue box obscuring the rest of the screen until it is dealt with.
- A printer may be reserved for special messages so that a hard copy of the information is preserved. Again, the fact that the information appears on that printer means that it has a particular importance.
- Information can be made to flash, or can be printed in a different size, anything that makes the operator's eye go to that part of the screen.
- It is important not to put too much on a single screen. It is better to use several screens with the contents clearly shown.
- If more than one screen is used, it is important to be able to move both forwards and backwards through the screens easily.

Data Structure Design

The data used in a computer solution will need to be stored somewhere. The storage is not that important. What is important is getting it back again when it is needed. A number of factors contribute to the access to the data one of which is the design of the way that the data is stored. Descriptions of the different ways to organise the data will be covered in section 3.5, sufficient at the moment to realise that there are different organisations and that the analyst will need to make a decision about which to use.

1.2.g *Explain the importance of evaluating the system, and how to identify the criteria used for evaluation*

Any system must match certain criteria if it is to be considered successful. This does not only apply to a computer system but any system. For example, a car must satisfy various criteria before being able to be called a car

- it must move under its own power
- it must be possible to steer it
- it must have 3 or 4 wheels
- it must have seats.

There would be many other facts which would have to be true before most of us would recognise that this system is a car. However, some assumptions have been made in just the four criteria mentioned here. A toy, radio controlled, car fits all these criteria, but it was not the sort of system that we had in mind when designing the criteria. Perhaps the second bullet point should have specified that it has to be controlled from within the car itself, or should there be a new criteria that gives a minimum size for the vehicle? When systems are being designed, this list of criteria, that the finished system must match, is of paramount importance. It must also be agreed between the designer of the system and the commissioner of the design, so that there will be no unfortunate misunderstandings. We can imagine a situation where the Ford motor company commission a new car design, only to find that it is 30 cms long when delivered.

Note that these criteria are decided before the system is created. They are used to decide how well the system works. In other words, does it do what it is meant to? This question can only be answered if it is known what the system was meant to do in the first place.

1.2.h *Explain the content and importance of different types of documentation at different stages of the system life cycle, including the technical and user manuals*

The documentation of a system consists of all the text and graphics that explain how the system is to be, or was, produced, how it should be used, and how it can be maintained.

Documentation is created at different stages in the life of a system and for different people to use. Some of the different types of documentation are met in this section and others will be encountered later in the course. Indeed, much of the project work that is done in module 4 consists of producing the documentation for a problem solution.

Requirements Specification

This is a list of the requirements of the customer for whom the system is being designed. It consists, largely, of the criteria that will be used for the evaluation of the finished system. It is usual for the systems analyst and the customer to sign the list of requirements so that there is no confusion when the work is finished.

Design Specification

Taking the requirements specification and working out the stages necessary to produce the required end product is known as the design specification. This will include the different stages, often shown in diagrammatic form and also the criteria for each stage of the solution. For example, one part of a solution may be the production of a file of data. The ways that this file of data relates to the other parts of the system, and the specification of the file (What is the key field? How many records will there be? What type of medium should it be stored on?) make up the design specification.

The design specification will tend to alter as the solution is being produced, but it is very important to keep it up to date so that it can be used as a reference by all those involved in the work.

Program Specifications

These will include detailed algorithms showing the method of solution for some of the parts of the problem. These algorithms may well be in the form of flow diagrams or pseudo code. The language to be used, the data structures necessary, and details of any library routines to be used will also be in the program specification.

Technical Documentation

The technical documentation will include the program specifications, the coded program itself, details of hardware configurations. Generally, anything that will aid a technician in maintaining or updating a system. Indeed, technical documentation is otherwise known as maintenance documentation. This type of documentation is not intended to be accessible to the user of the system, who does not need any of these details to be able to use the system correctly.

The technical document will grow along with the system solution itself. Details will be added as the problems are solved. This is particularly important as the people who are working on producing the solution will refer to it throughout their work to ensure that their part of the solution matches everyone else's. For example if one person is working on the barcode reading part of the program and calls the barcode BAR_CODE while the person who is planning the data files calls the barcode BCODE, the system cannot work.

There are three types of maintenance that need to be supported by the technical documentation: (For example, imagine a new till system in a supermarket)

1. Corrective maintenance is when an error is found in the normal running of the system which needs to be corrected. For example, the workers find that if they try to scan bananas after a tin of tuna, the system fails to register the bananas. This is probably a problem with the way a particular value of a barcode is stored which would not have been discovered during normal testing as it would be impossible to test every combination of goods to pass through the terminal.

2. Perfective maintenance is when the system is running and is working properly but it is found that something could be done better. For instance, the workers find that when the terminal recognises a two for one special offer it is printed on the receipt as soon as the second item goes through the reader. This causes confusion because shoppers cannot see easily that they have received the discounts. It is decided that all the discounts should be printed at the end of the receipt to make them more visible to the shoppers.

3. Adaptive maintenance is when the system is working perfectly well, but there is some external change which means that the system has to be altered. In the supermarket, the rate of VAT on fresh vegetables is altered by the government. This would mean that the calculations carried out by the system would need to be altered to show the correct amount of VAT on the till receipt. Another factor may be the need to change some of the hardware in the system because the previous hardware was out of date. When this is done the software will need to be altered to accommodate this. For instance the receipt printer used to print up to 40 characters on a line, the new printers will only allow 35 characters on a line.

User Documentation

This is the documentation for the person who will actually be using the system. It contains those details that are needed to make the system operate as it should. Items normally included in the user documentation will be

- how to install the system onto specific hardware
- methods of input of data
- examples of valid input
- examples of output screens
- error messages and what to do when they appear

Most importantly the user guide should reflect the requirements specification that has been agreed between the client and the systems analyst. This specification makes clear what is expected of the finished software, so the user guide should explain how to use the software to carry out these tasks.

The user documentation will be built up as the solution progresses so that when it is finished the user documentation will have all the detail necessary and will only require the analyst to arrange it in a sensible form and to add details like an index/contents page/glossary of terms used/methods to be used to create back up and archive files (these terms are fully explained in section 1.3.h).

Some user documentation is part of the software and can be called up onto the screen when it is needed. Typical of this type of documentation are help screens brought up by the ? key or by pressing a special on screen button. Often such documentation has a search facility which must be programmed after the file is produced. The software engineers will add to this file as an integral part of the software that they are producing.

This type of documentation is called on-screen help. Be careful not to confuse this with on-line help which is the use of the internet to access a site devoted to the software which is being used in order to send queries to the company that has produced it and to do things like down loading of patches to cover any problems discovered during the normal operation of the software.

1.2.i *Explain the importance of system testing and installation planning*

Any system needs to be tested to ensure that it works. This seems to be a fairly obvious statement, but in reality such testing is impossible in all but the simplest of systems because it simply is not possible to test every conceivable input to, or logical construction in, the system. This difficulty means that testing that is to be done must be carefully planned and that it should relate directly to the criteria referred to earlier in this chapter.

As far as the systems analyst is concerned the most important type of testing is functional testing. This tests the system to see if it will successfully do all the things that were specified as necessary in the requirements specification, in other words does it work? The analyst is looking to provide proof that what they have done meets the criteria because if they can show that the system does what they were asked to do they will get paid. There are many different ways of testing a system, some of which will be covered later in the course, specifically in section 2.6.c.

When the system has been completed, tested and accepted by the client, it has to be implemented so that it is performing the tasks for which it was designed. Initially, this involves

- ensuring that the correct hardware is available
- installing the new software on the hardware
- arranging for staff to be trained in the use of the new system
- inputting the data to the data files, either manually or by downloading them from the original system.

The system handover, itself can be done in a number of ways:

- Parallel running. Until the system can be considered fault free, the old and new systems are run side by side, both doing the same processing. This allows results to be compared to ensure that there is no problem with the new system. Such a system is 'safe' and also allows staff training to be carried out, but it is obviously very expensive because of the need to do everything twice. Parallel running is used in situations where the data is so valuable that there must be no possibility of failure.
- Pilot running. This is where the software is fully implemented but only in one area of the organisation, perhaps one branch of a supermarket. It means that the software can be run in a real situation to help spot any errors; the staff can be trained in using the new system on some sort of rolling program, perhaps the staff at other branches are sent there to work the new system for a week so that they know what to expect when the system comes to their store. Also, if there are any major faults then only one branch of the business is affected.
- Big bang, or direct change. The old system is removed and the new system replaces it completely and immediately.
- Phasing. Parts of a system are replaced while the remaining parts are covered by the old system. This allows for some testing of the new system to be done, and for staff training to take place, but also allows for a back-up position if the new version does not work as anticipated.

1.2.j *Explain the purpose of maintaining the system, and explain the need for system review and reassessment, understanding that software has a limited life span*

Systems may be designed for a well defined purpose and may realise that purpose, hence they would be considered successful. However, the original reasons for a particular system to be created may change, the hardware may alter, the law governing a system may change. For many reasons the original system may no longer be satisfactory. One solution would be to produce a totally new system. Another would be to adapt the present one so that it can be used in the new circumstances. This situation is reviewing the system and is the main reason for technical documentation.

While the system is running it will need attention because of faults being discovered, this again needs a technician with a set of maintenance documentation.

Computing and computer applications is a subject that changes so regularly through improvements in technology, new ideas, different legal frameworks trying to keep pace, that, in reality, a system should never be considered to be finished. Rather than being a linear process with a beginning, middle and an end, it should be thought of as a circular process, continually returning to previous stages to fine tune them and take advantage of changing circumstances.

These points should be considered in conjunction with comments on technical documentation in section 1.2.h

1.2.k *Describe prototyping to demonstrate how a solution will appear*

When a piece of software is being designed it is often useful to design a small part of it so that that part can be considered and decisions made about it without the rest of the system getting in the way. This is particularly important when designing parts of the system which the client/end user will be using. In a data handling example it is important for the end user to be happy with the input and output screens. The methods of storing the data that is input or the ways in which the data will be manipulated by the system are not of interest to the user. Consequently it is useful to develop an input screen which the user can see and practice using despite the fact that it is not attached to anything, the data that is input doesn't actually do anything. This screen is called a prototype and the action of the end user in trying out inputting data and perhaps suggesting changes to the way the data is input is called prototyping.

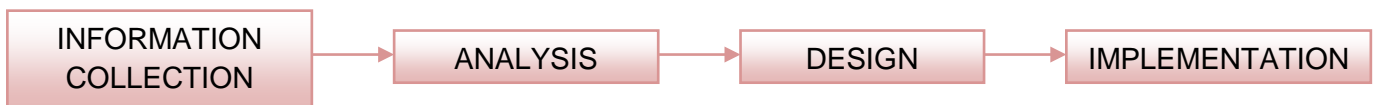
This concept of prototyping is very important because often the end user will not have a clear idea of what they require until they can actually see it. If the analyst sticks to the system life cycle completely then nothing will actually be produced until the analysis and design is completed and then the implementation can begin. However, the prototype screen can be produced very quickly and used to check that the end user is happy with the details of input and output before things go too far.

1.2.1 Describe the spiral and waterfall models of the system life cycle

The systems life cycle should not be thought of as a linear process, in other words a process where each part is finished before moving on to the next part and never having to go back. Under the heading of prototyping we have already stated that the user is probably not clear as to their vision of the finished product. This is why part of the design process is to produce a prototype screen. The comments about this screen will almost certainly include problems which need to be corrected, in other words a return to the information collection and analysis sections so that the screen can then be redesigned. The process has now become something of a repeating process.

The best way to show this is to imagine the systems life cycle in diagrammatic form.

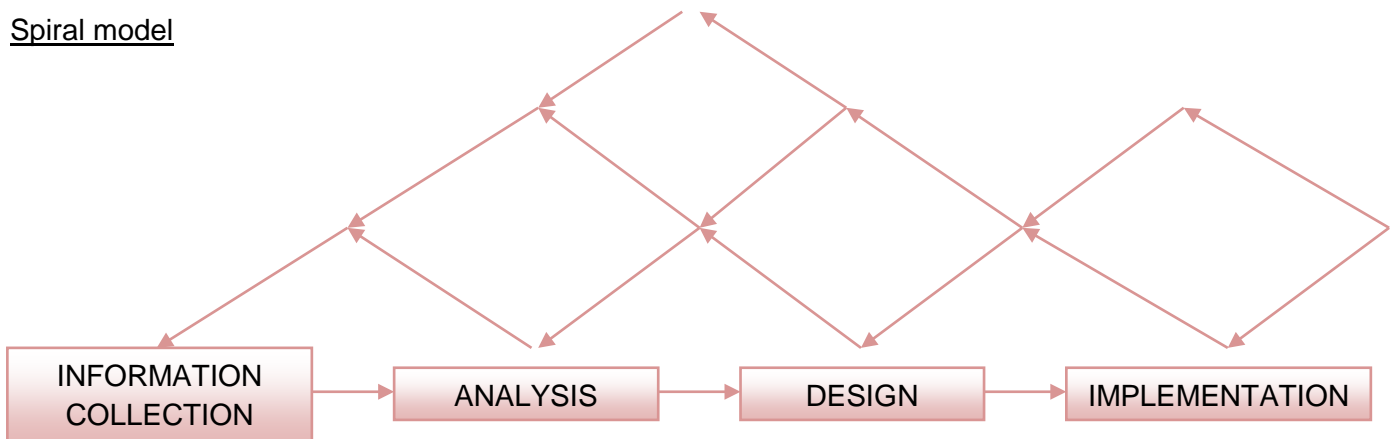
Many people believe that it is a linear process:



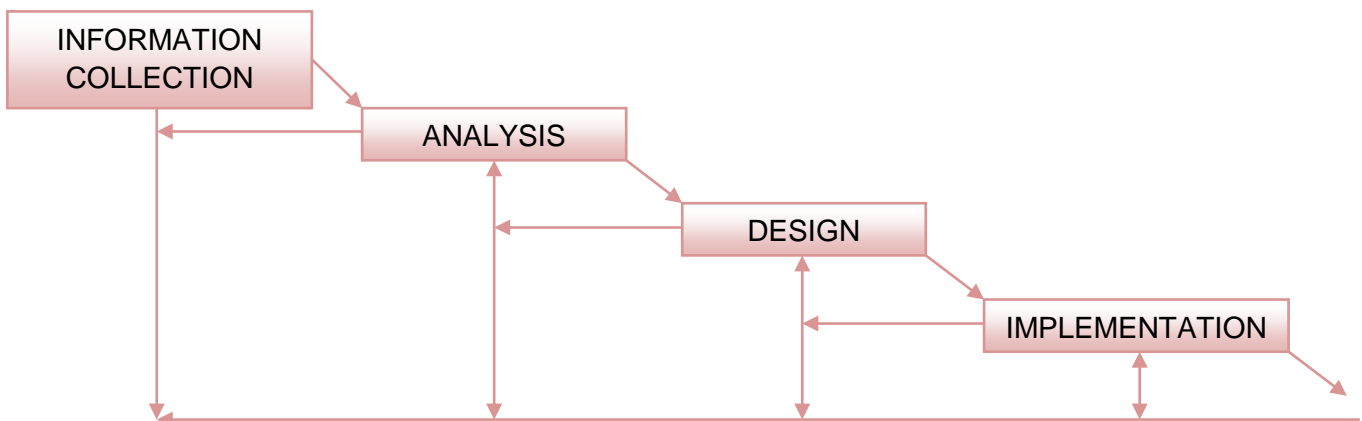
This is only true if the problem solution is very simple, so simple that there is probably no need to use the systems life cycle anyway.

More realistic are the spiral and waterfall models-

Spiral model



Waterfall model:



With these models the idea is that the analyst has to keep returning to previous stages in order to take account of problems that arise in later stages of the production of the problem solution.

1.2.m *Identify the features of common applications found in business, commercial and industrial applications*

Stock Control

As the name implies, stock control systems are used to keep track of the stock being held by an organisation.

There are typically three areas that need to be handled by a stock control program.

- To keep details of the individual items that the organisation holds in its stock. These details will differ depending on the item being held and the information about it that the organisation decides is important. A consignment of towels in the stock of a shop will need to have the colour of the towels stored, whereas the information stored about bulldozers stored by a plant hire firm would not include colour as it is irrelevant to their function. What would be included would be the horse power of the machines, which obviously does not apply to the towels.
- To store information related to the product though not directly about the product itself. Examples would be storing details of the supplier, or where the goods are being stored in the shop/warehouse.
- To do with the use of the goods. Who has hired the bulldozer out? Where is it? When is it due back? What sort of condition was it in when it was hired out?

Stock control systems are all to do with keeping track of stock, recording stock levels and the condition of the stock and keeping track of where it is going.

Order Processing

Many stock control systems are charged with the need to record the number or amount of the goods that are in stock. If the quantity in stock falls below a pre-determined limit, the system should automatically institute a reordering process. This would be done by using the details of the supplier and possibly including the human management in the process to allow for variations caused by season/fashion/.. Copies of the orders would be sent to the accounts department so that when the invoice for the goods comes in it can be matched up with the order. When the goods are delivered, the ordering system is informed so that the live order can be shown to have been fulfilled, and the accounts department is told that the order has arrived and that clearance has been given for payment to take place.

If the organisation is the one that is filling the order, the operation is similar, but in reverse.

Payroll

The payroll is a perfect example of a batch process (described in 1.2.r). All the records need to be processed during the same run because all the workers need to be paid. Each record undergoes the same type of processing, working out the number of hours worked, multiplying by the hourly rate and then doing the tax calculations. The process requires no human intervention during the processing.

The traditional picture of the payroll application is that the master file of workers is held on a tape, as is the transaction file that holds the workers' details for that week. The two are arranged in the same order and then run together and the results stored on a new master file, again held on a tape. This gives rise to the ancestral filing system form of backing up of files which will be described in 1.3.h. In this system, the old files are kept so that if the new versions are corrupted they can be reproduced using the old master and transaction files. While easy to understand when the files are stored on tapes, the file is more likely to be held on a disk with an index to allow fast access to individual records, this would imply that the records are overwritten and that there is no new version of the master file created.

Process Control

As the name implies, this is the use of a computer to automatically control a process. The computer receives information about the process from sensors which allow it to make decisions. The results of these decisions are actions that are carried out. The next set of input from the sensors not only tells the system about the current state of the process but also allows the computer to compare with the last set of inputs to decide whether the actions that it took last time had any impact. This process is known as feedback.

Point Of Sale Systems

A computer used at a point of sale needs to carry out three actions. The first is to identify the goods being bought, the second is to carry out whatever processing is required and to produce a satisfactory output, and finally to arrange for payment.

The identification of the goods can be done in a number of ways, but the standard method is to read a code from a barcode. This code is then validated (see section 1.3.f) and then sent to the processor. The processor uses this barcode as the key field when searching the product file. When the record is found the contents of the record are used to produce a printout for the customer (till receipt), and to accumulate the total value of all the goods that have been bought. Finally, the payment can be made electronically by sending details of the customer account to the bank or credit card company from where payment will be made to the store.

Marketing

When a product or service is developed it is important to make the potential customers aware of it. Marketing is the term given to this process. In computer terms it would include the use of systems to produce advertising literature, the promotion of the product on the World Wide Web and the use of techniques such as direct mailing by using purchased files of people who may reasonably be expected to have some interest and then mail merging to produce 'junk mail'.

Computer Aided Design (CAD)

This is simply the use of a computer system to design a commodity. It may be a house, or a carburettor to fit a particular engine, or it may be a new traffic flow system around a town. The software can be used to do calculations. Will the roof be well enough supported by walls made from that particular product? It can be used to make decisions about the way the product is manufactured. Will the robot tool be able to manoeuvre into a position to be able to screw the two halves together? It can be used to cost a solution, having decided on the design of the carburettor the prices of the individual components can be used along with the known costs of production to calculate the cost of each unit. It can be used to make predictions, what will happen if this street is made one way?

Computer Aided Manufacture (CAM)

CAM is the use of a computer to help with the manufacturing process. If the two principles of CAD and CAM are combined then it should be possible to produce a seamless process whereby the computer produces the design which is electronically sent to another system where the design is produced by a computer controlled robot.

1.2.n *Identify and justify generic applications software for particular application areas*

To begin with it is necessary to provide a few definitions so that we all know where we are starting from.

Applications software: This is a program or programs that do a task that would have to be carried out even if there was no computer. For instance a word processor is an applications program because the letter to the supplier has to be written anyway.

Applications package: This is basically an applications program together with its associated documentation (so that you know how to use it!) The project that you are going to do in module 4 will contain software to solve the problem (an applications software) together with the User Guide (this makes it an applications package. The definition is sometimes widened to mean a set of applications programs that can communicate with each other would make up a package.

Generic applications software/program/package: This is software which can solve a number of problems. For instance a word processor is a generic piece of software because it can be used to solve lots of different problems, whereas a piece of software which is designed to solve quadratic equations is not generic because it has a specific use and cannot be adapted for other uses.

There is nothing frightening about this section. The intention is simply that candidates should be able to suggest sensible generic software for given applications. The software automatically divides itself into specific uses. The student should be able to isolate the important characteristics of an application from the description and then decide which of the generic software best fits the application characteristics.

Word Processing

Used for applications where the user has the need to communicate with others using text. Writing letters, mail merging, preparing text documents for use in other software packages, are all typical uses of a word processing package.

Spreadsheets

Spreadsheets are a type of software that allows data to be stored. If this were all that a spreadsheet was to do then there would be other, more satisfactory types of software available. A spreadsheet is particularly useful because it can store different types of data, including numerical data, and that it can perform calculations on the areas (cells) where the numerical data is being stored. Spreadsheets should be considered for any example where data is stored and calculations need to be carried out on it. Examples would be profit and loss accounts, budgeting, payrolls (although other, more specialised, payroll software would be used in a large scale application), indeed any example that requires the manipulation of figures to give accurate results or forecasts or predictions.

Desktop Publishing (DTP)

This type of software is characterised by the ability to produce a page of printed output that has been designed by using advanced layout techniques. The page may well contain text, graphics, tables and many other types of output each one of which may be better produced using a word processor or a drawing package or a spreadsheet. The value of DTP is that it contains powerful tools for arranging these individual items on the page, the printout of which can be used directly as the starting point for a printing process. Typically DTP software is used for the production of leaflets, posters, proof copies of books and magazines. Many word processors now have features which previously would only have been found in a DTP package,

for instance the ability to produce text in columns or to surround graphics with text, and the distinction between a DTP package and a word processor has become less clear cut.

Presentation Software

The growth of the use of presentation software has followed the development of portable computer systems. If a salesman is to do a presentation to a group of people, he will take a computer to the meeting with a previously prepared presentation on it. The software allows for the preparation of a show which typically follows a storyboard of individual screens. The software allows morphing from one screen to another and also allows animation and full use of text and graphics within individual screens. If required, a soundtrack can be added to complement the pictures being shown. Ideally, such a presentation should be output via some device that would be designed for the audience expected to be watching. This would mean that a single monitor would be fine for some automated display in a department store, but would require something more akin to a projection screen for a larger audience. This could be accomplished using an over head projector linked to the computer or an RGB projector.

The use of presentation software is now so prevalent that there is a back lash against its use, with audiences being turned off by 'yet another PowerPoint!'. The reason for this is that presenters have not been sensible or discerning in its use. Sometimes presentation software is the most appropriate means of providing a presentation because of its advanced features, but too much use when there is no reason is counterproductive.

Drawing Packages

This is a package that produces graphics output. Often such output is exported to a DTP package for inclusion in some publication, or to a piece of presentation software for inclusion in a display sequence. Another use for such output is to enliven a page on the world wide web. There are many different forms of graphics package split into groups dependent on the way that the graphic is produced. The two most common are bitmap graphics where each pixel is treated separately, and vector graphics where the lines on the drawing are created mathematically. Different software packages create the graphics in different ways; for instance Paintbrush creates pictures in bitmap form while Draw uses vectors. The simple way of describing the difference is in sizing the drawing. If a drawing held as a bitmap is increased in size then each of the pixels is increased in size and hence becomes more obvious. If a drawing stored using vector graphics is increased in size the only things that change are the mathematical formulae for producing the lines, which will produce a picture of comparable quality whatever the size.

1.2.o *Identify and justify application areas for which custom-written applications software is appropriate*

A Custom package is one that has been specially written to solve a specific problem.

There are many reasons why this may be considered necessary rather than just going to the shops and buying a piece of software off the shelf. After all, if a small company wanted to keep a database of customers and orders there are plenty of pieces of software which would be appropriate to use. However, it is possible that the requirements of the company are unusual and there is no available piece of software which will satisfy all the requirements. It may be that the requirements are so simple that all the superfluous parts of the available generic software would never be used and would interfere with the purpose that the company has for it. The difficulty with using generic applications software which is tailored to solve the problem is that the compromises that need to be made to come up with a working solution tend to be made on the part of the problem because the software is already written. In other words the problem solution needs to be made to fit the software that is available instead of the software being made to solve exactly the problem that the company has.

The decision about whether to write new software is a very important one. We have said that compromises to the solution will probably be necessary if pre written software is to be used, but the compromises would have to be major to overcome the advantages of using generic applications software. The advantages of using pre written software are that it is much cheaper because the development costs are shared amongst everyone who buys it; it will be available for use much quicker because the long writing process has already been done, it is simply a matter of tailoring it to the specific needs; it has been in widespread use for some time so most of the bugs are missing; it is probably a part of a package of software so will be compatible with other software used by the company; finally, there will be training materials and even a lot of people who already know how to use it.

However, there are many applications that are either new applications or simple one off applications that do not fit into the available software types, where the use is so specialised that the software will need to be written (or at least tailored) for that particular use. Uses that would require such specialist software are legion, but would include many uses of control software. Robots used on production lines tend to be one-off machines, designed for that particular purpose, consequently, the software that would drive them would, similarly, be one-off. A stock control system in a warehouse would have standard modules, but would have other sections which would be for that warehousing system alone.

The important consideration is how different from the standard design is the system for which we want the software, the further away from the standard, the more difficult it is to use standard software, and the more likelihood there is that the system will require software written specially for it.

1.2.p *Describe the characteristics of knowledge based systems*

A knowledge based system is a computer system where all the expert human knowledge covering a particular topic is brought together and made available to the user through a computer system which uses the facts it has been given in order to answer queries from outside the system. Knowledge based systems are often called expert systems because they contain all the knowledge of the expert, however, they tend to lack one thing that the human expert possesses, intuition. They have to follow the same rules every time and consequently they should give the same answer to the same queries every time, while the human can recognise subtle differences that are not part of the accepted knowledge surrounding a topic. It is for this reason that such systems are still capable of being beaten by a human expert when playing chess and if they cannot even guarantee a win in what is a terribly narrow field (basically restricted to 64 squares on a board) it gives an indication of their limitations.

Knowledge based systems comprise 4 parts:

- There is a knowledge base. This knowledge base contains all the information that has been supplied to the system by the experts who were involved when it was set up.
- The access to this knowledge can be very haphazard unless the system follows certain rules. That is why all expert, or knowledge-based, systems include rule bases which determine how the data within the system relate to each other.
- There must be an algorithm, or set of algorithms, for determining how the rules in the rule base should be applied to the knowledge in the knowledge base. These algorithms are known as the inference engine. The inference engine gives a method for searching and querying the knowledge base using the rules in the rule base.
- Finally, there must be a user interface which allows the user to interrogate the knowledge base. The user interface will pass requests on to the inference engine which, in turn, interrogates the knowledge base. The user interface will prompt the user, often by producing a series of questions each of which will have a small number of possible answers, and each of which will have the effect of accessing smaller and smaller areas of the knowledge base. The user interface will also allow the user to ask for explanations of the reasoning behind the advice that has been given and the results will give conclusions with the probabilities of those conclusions being correct.

The last point made above is important. In many cases the result of an enquiry cannot be given with any degree of certainty. Typically, a knowledge based system will give the results together with a probability of whether the outcome predicted will happen, or prove to be correct. These probabilities have different importance dependent on the application in question. Being able to suggest illnesses of patients with 95% success is actually quite high and would be considered a useful tool for diagnosis. Being able to suggest materials for the hinges of a cargo door, which will withstand the pressures of takeoff, in a new aeroplane with 95% success would mean that 1 in 20 planes would be in danger of falling out of the sky. (This is precisely what happened with the McDonnell Douglas plane which crashed on takeoff at Orly airport near Paris killing over 300 people. The hinges of the cargo door blew out. All the other DC 10s had to have the hinges changed.)

So, what are knowledge based systems used for? Basically, the different applications are numerous. Consider your project that you will be doing next year: the probability is that whether it is based on a database, a spreadsheet, a website, or almost anything else, it can be thought of as a knowledge based system. It will probably have a large amount of data in it which can be accessed according to information input by the user and the data will be searched by some sort of search engine using rules in order to give the user a response. A classic definition of a knowledge based system.

Human knowledge encompasses such a massive area of data that it is not reasonable to try to distil all human knowledge into one computer database. Because of this problem of volume, each knowledge-based system is restricted to a narrow area of knowledge. It may be geological patterns in the oil exploration industry, medical diagnosis in the doctor's waiting room or personnel information in a company, all are well delineated areas of expertise where there are no grey areas of whether the data should be included or not.

1.2.q Describe the purpose of operating systems

An operating system is a set of programs designed to run in the background on a computer system, giving an environment in which application software can be executed.

Most operating systems comprise a large set of programs, only some of which are stored in the processor memory all the time. Many of the routines available in the O.S. are stored on the hard drive so that they can be accessed when required. This not only saves space in the processor memory but also means that the O.S. can be easily changed to a different one.

When you are using an applications package you are not communicating with the computer hardware, you are communicating with the operating system. Without an operating system, no matter how many programs you have, the computer is useless. The operating system sits between the hardware and the application program or user.



There are many different types of operating system (OS), each of which will make the computer behave differently and will be designed to be appropriate to a given situation. We will look at some of the different types of OS in the next section and again in other parts of the syllabus, but they all have some things in common:

- The OS controls the operations of the hardware, particularly; it controls the flow of data around the system. For example it controls the storage devices and fetches data that the computer can use and is then in control of putting it away again. It controls the characters you type in and arranges for them to be displayed on the screen, or the position of the cursor controlled by a mouse.
- It provides a platform on which the applications software can run.
- The operating system controls access to the computer system so that you can put passwords on your files to stop other people seeing them. It also knows what rights different people have and makes sure they stick to them.
- It provides a means of communication between the human being/ outside world and the computer. This is called the Human Computer Interface (HCI) and we shall study these in some detail in later sections, starting with section 1.2.t
- It provides a number of pieces of software which can be used to look after the system. One which we are all familiar with is anti-virus software. These are known as utility software and we shall be looking at them in more detail in section 1.2.v

1.2.r *Describe the characteristics of different types of operating system and their uses: batch, real time, single user, multi user, multi tasking and distributed systems*

Batch Processing

When computing was still a new science, there were not enough machines to satisfy the demand for processor time from students in universities who wanted lots of calculations done, firms who wanted their payroll worked out, and many others. The big problem was the 'speed mismatch' between the user sitting at the keyboard who was very slow, and the machine which was very fast. There are two simple solutions to this problem, one is to buy more machines and the other is to make the machines work more effectively by taking away the slowest part of the system – the human being. Nowadays we might well opt to buy more machines, but this used not to be an option. This problem gave rise to the development of batch processing.

A batch processing operating system is one that does not allow for interaction between the user and the processor during the execution of the work. Lots of programs (or more likely these days, lots of data to be run through the same program) that need to be run are collected together (to form a batch) and they are sent to the computer. The batch operating system then controls their passage through the computer.

Nowadays, batch processing tends to be used where:

- there are large amounts of data to be processed,
- the data is very similar in nature and...
- it requires similar processing,
- the computer system has identifiable times when it is not being used, and so has available processor time
- the application does not require human intervention.

Typical examples of applications which would be done using batch processing include production of bank statements from customer files, production of gas (electricity, telephone) bills from customer records, the compilation of high level language programs where a number of users want programs compiled.

Real-time

A real-time O.S. is one which can react quickly enough to affect the next input, or process to be carried out.

Most real-time systems are based on control of some process in the real world or on information handling. A chemical plant has a reaction vessel where the temperature is critical to the result of the process. The temperature is monitored by a computer which accepts input from a sensor and then makes a decision whether to adjust the heating elements in the vessel. In this example, it would not be sensible for the computer to be running any OS that is not real-time because if there was a delay in the decision making process, it might mean that the reaction is corrupted in some way. A robot trolley is controlled by a processor which takes input from a sensor following a black line on the floor, and makes decisions concerning steering to keep the trolley on the black line. If the processor was not controlled by a real-time OS, the trolley would very soon leave the black line because it would not be steering quickly enough. A catalogue shop processes orders according to the code for a product which is input and the system then compares the code with information in its files. When it finds the correct code it can report to the user the number of that item that there are in the store. If there was only one left of a certain item, it would be necessary to record the fact that a shopper had bought it before the next shopper has their request dealt with otherwise the second person might be sold the same item. Because the information on the system must be processed immediately the OS needs to be ready to handle input as soon as it comes in. This means that it cannot be using up some of its slack time doing other tasks while it is waiting to be asked to do something. This implies that the computer will not be using its full potential as far as processing is

concerned. When this happens it is said to display a high rate of redundancy. Real-time systems tend to display a high rate of redundancy.

Single User

As the term implies, a single user OS is specifically one that is used to control a system which has only one user, and their programs, at any one time. A perfect example of a single user system is the one that you may have at home. Only one person uses it at a time. Note that it does not mean a system that only ever has one user!

Multi-user

Again, as the name implies, this type of OS services more than one user simultaneously.

There are two types of multi-user OS:

1. A network system comprises a number of computers linked together for the purposes of communication and sharing of resources. Normally one of the machines is used to control the rest of the system; this machine is called the server. Networks are important because they allow hardware and software to be shared and also mean that only a single copy of the information on a system is needed, and so it can be kept up to date relatively easily.
2. A time-sharing system has a single (normally powerful) computer which is connected up to a number of terminals. These terminals are not computers as in the case of the network system, but have a very limited amount of processing power. Again, such a system allows communication between users on the system and also allows sharing of hardware and software across the system.

At the moment it is difficult to tell the difference between the two types of multi-user system, but we shall return to this in sections 1.5 and 3.3.

Multi-tasking

This is a type of OS that allows several applications to be available simultaneously. On a simple single user system you will probably be used to having a number of things running at the same time. Perhaps one window shows a spreadsheet while another shows a word processing application because you are copying a spreadsheet into the word processed document. You are connected to the Internet in case an email comes in and you are playing music while you work. It appears that more than one task is running simultaneously. They aren't, they just appear to be. The OS is able to switch from one task to another so quickly that it just seems that they are all being done at once. This is multi-tasking. The OS that most of us use on our own computer systems, Windows, is a multi-tasking OS.

Distributed

A distributed system is one that allows software and data files to be distributed around a system. An ordinary network will have a server both controlling the network and access to the hard drive which is connected to the server. A distributed system might store the word processing software on one computer's hard drive, while the files of work are stored somewhere else, and the spreadsheet software is stored on a third disk drive. This can speed access to files because there is no single bottle neck which all the information must pass through.

1.2.s *Describe a range of applications requiring batch processing and applications in which a rapid response is required*

Batch Processing

A batch processing system is used when the output does not have to be produced immediately. Other factors are: that the application will tend to use a large amount of data; that processing will tend to be of the same type for each set of data and that human intervention is not necessary. In 1.2.r we saw that the perfect example of batch processing is the payroll system, and nothing has changed because this is a different section. If you would prefer not to use the payroll example in an exam question then use the production of bills or bank statements, but be very careful in choosing anything else because it must be clear that there are good reasons for using batch processing. For example, simply to say that an ATM machine uses batch processing is not enough. Requests for bank statements are batched for later processing, but checking of the PIN for a customer must be carried out in real time, so be careful to be quite specific and to justify your choice: "because the statements must be sent out by post for security reasons and therefore they can be processed while the computer is not doing other things."

Rapid Response Processing

In module 1 this was referred to as real time processing. Real time processing can be thought of as being used in process control where the results of the process are used to inform the next input. A good example would be the control of a robotic machine on the production line. The other example is in information systems where it becomes necessary to update the file of information before the next enquiry is dealt with. The classic example is the airline (or theatre) booking system. If a customer enquires whether there is a seat available on the flight, and in response to a positive reply decides to buy the ticket, then the number of tickets now available must be updated before the next person makes an enquiry, otherwise the second person may be sold the same seat.

In general terms, an examination question will expect the student to be able to decide which of these two types of processing are most appropriate in a given situation. It is not sensible to list a number of different applications for each type and to expect students to learn them. The application in a question may not be on the list and the student would not be demonstrating an understanding of the concepts even if it was. Far more sensible is to learn to recognise the characteristics of each of the types of processing in a given application.

1.2.t *Identify and describe the purpose of different types of user interface, suggesting the characteristics of different types of interfaces which make them appropriate for use by different types of user*

A computer is used by a person who needs to communicate with the machine in order to instruct it as to their wishes. The person also receives responses from the computer. The means of communication between the user and the machine is known as the user interface (or Human Computer Interface, HCI) and consists of both hardware and software.

Basically, if you are sitting at a keyboard and typing instructions in to a computer and then the computer is giving responses to you by printing details out on a screen, then that is an interface. The keyboard and the screen are the hardware and the software that allows the computer to understand typed instructions and turn its responses into something that you will understand on the screen is the other part of the interface. Change the screen for a printer and the basic interface is the same except the change of hardware probably means that it is a different use. For example it was originally a system for making enquiries about tickets, the new version is a system which will accept the input in the same way but will print the ticket

There are many different types of interface dependent on the application it is being used with, the conditions in which it is to be used and the knowledge/skills of the user.

There are a number of different types of HCI, five of which you will be expected to be able to describe and suggest sensible uses for:

1. Form based

If the majority of the input to a system is of a standard type, in other words the computer knows what sort of input to expect, then a typical interface will produce a form on the screen to be filled in. This sort of interface would be used where an operator is inputting information while asking a customer questions over the telephone. The interface

- prompts the operator to ask each of the questions in turn
- makes the operator input the information in the correct order and ensures that nothing is missed out
- ensures that the information is input in the correct format by having specific areas to input the data
- makes the checking of the information easier.

The characteristics of a form based interface are that

- it has specified areas for the data. For example, boxes for input of coded material like the date or the sex of the customer, and areas to be filled in with textual information
- it has a cursor which moves to the next box to be filled in, sometimes the box is highlighted to make it clear to the operator where the data is to be inserted
- some of the boxes are more important than others and the cursor will not move on until some data has been supplied
- it checks that what has been input is sensible for that box before moving on to the next.

A typical area of use for a forms based interface is when a person orders something from a supplier on the Internet they will be expected to fill in a number of personal details and payment details. These details are normally typed into a series of boxes. If a box is missed out then it is probable that a message will come on the screen to inform the user that some of the information has not been given and if the wrong data (e.g. 18/11/90 is typed in to the box asking for name) is offered the interface will reject it.

2. Menu based

Menu based interfaces are used in situations where the operator tends not to know what the options are that are available. Examples of this would be information systems for tourists or users of a particular service. A good example is the use of the handset menus for digital television. A list of choices is made available followed by a further set of choices based on the first choice, and so on until the result is obtained. Imagine a system at a train station in a popular holiday location. The first screen may ask for the general area of interest (accommodation, trips, shopping, entertainment..), once the choice of accommodation has been made the next screen may offer hotels, guest houses, bed and breakfast, self catering. The next screen may offer different price bands, and finally a list of all the available properties that match the previous choices. Input is often done using a touch screen because of the location of such systems or by something like a hand set or a pointing device because there is a very restricted number of choices and because the people who use them are often in no way computer literate, meaning that simple systems are essential.

3. Graphical

Graphical interfaces are called GUI (graphical user interface) or WIMP (windows, icons, menus, pointer). The terms describe what the user sees on the screen. There are many different types, but the user would expect to be able to view different applications or files on the screen, this is done by putting each into its own boarded area known as a window. The user will expect to be able to select options by use of menus of choices (note that this is not a menu based interface as so much more is offered) and by using small pictures which represent the different options available. Choices are selected by the user by using some sort of pointing device to indicate choice; typically this would be a mouse. The whole principle of a GUI is to make the use of the system as simple as possible by hiding all the complicated bits! A good example of this is the use of an icon to bring a piece of software onto the screen. In reality, the loading of the software to that area of the screen is a complicated process involving a lot of computer instructions. When the icon is chosen the computer is simply told to run those instructions so the software will appear. The icon has hidden all the difficult bits from the user.

4. Natural language

Sometimes referred to as a conversational interface, the computer will ask questions which elicit a response which gives the user the impression that they are talking to the computer. The trick is that the system restricts itself to questions to which the only sensible answers are the ones that it knows. If the user leaves the expected responses, a message is produced which makes clear that a further attempt is required. It is often assumed that a natural language interface is very close to a spoken language, probably true, but it will likely be typed in to the machine rather than actually being spoken. Typical examples would be the search engines used on the Internet or large information systems and a true spoken version might be used by a disabled person to communicate with the machine.

5. Command line

Or 'command based interface' is one where the user types a series of commands at the keyboard which tell the computer what their intentions are. The user needs to know what the possible commands are, and also needs to understand the way files are stored on the system. The characteristics of a command based interface are

- the user needs to know what commands are available
- the user needs to understand the commands
- the user needs to understand the way that material is stored in the computer system

Because of the above points there are two very important characteristics about a command based interface:

1. The system is very much more open than in the other types of interface. Other interfaces restrict the options that the user has available to them whereas the command line interface allows anyone with a knowledge of the commands access to the entire operating system and hence to the workings of the entire computer. This can be particularly important for the system manager because other users can only be allowed to have access to specific parts of the system.
2. Command based interfaces can only be used by computer literate people because you not only have to understand the commands and their uses but you also need to understand something about how the computer operates, particular about how information is stored.

There are many other points to be made about interfaces, especially the fact that the second part of the interface consists of the hardware necessary to put the software interface into operation. These points will be made when the appropriate stage is reached in this course.

1.2.u *Discuss the importance of good interface design*

However complex the software, however expensive and powerful the hardware, the system is unusable if there is no intuitively simple-to-use interface with the human being who is in control of the system or for whom the system is producing results. The Human Computer Interface (HCI) must be unambiguous and must allow the user to input all the data that the user thinks important. The HCI must also produce the output in an easily understandable form.

The intended user of the interface must be taken into account. The interface design for a pre-school playgroup to teach the children about numbers is going to be very different from the design of an interface for the manager of a chemical plant to keep a check on the reactions around the plant. The interface that this manager would use to study a particular reaction (perhaps tables of figures, or graphs showing the different parameters of the reaction) is very different from the interface the same person would use if they wanted to see the flow of a chemical around the whole site. This would probably be in the form of a diagrammatic representation of the site.

The circumstances under which the interface is to operate are also important. In section 1.3.g we will study the job of a nurse looking after a group of patients. In those circumstances an audible warning that a problem may be occurring is far more sensible than a visual one which may be ignored if the nurse's attention is drawn away from the screen.

Finally, the designer of the interface needs to take into account the purpose for which the interface was designed, namely the transfer of data into the system and the communication of information from the system to the user. For instance if the user is using the computer to play a game, then the paramount importance of the interface is that it adds to the enjoyment of the user.

Consideration of the purpose of the interface:

- who it is for
 - what information needs to be conveyed
 - the circumstances under which the interface must operate
 - the effectiveness of the communication
 - the enjoyment obtained from using it
- are all important when the interface is being designed.

1.2.v *Identify and describe the purpose of a range of utilities*

In 1.2.q we said that one of the purposes of an operating system was to supply utility software. These are short programs that carry out tasks within the computer that are necessary to the smooth running of the computer. It is necessary that candidates are aware of a few of these utility programs and their uses. There is a warning here that, while many routines can be described as utility programs, there are a number that fall into a grey area between being part of the system and being part of application packages. For the benefit of the course it is sensible to always use standard pieces of software about which there can be no argument.

- Data transfer programs. These are utilities that are written to control the movement of data from one piece of hardware to another. An example would be sending data from the processor to the printer for printing out.
- Hardware drivers. Any piece of hardware needs to be controlled and set up for communication with the processor. The drivers are the programs that set the rules for these communications, for example the printer driver will contain details of all the font options available.
- File handling. Sets of data on a computer system are known as files, and a set of programs is necessary in order to handle the storage and use of these files. First there are programs to store and retrieve the files in the first place. Then there are programs which allow files to be altered, or even deleted completely. Sometimes the contents of two files need to be combined, this is known as a merge. Finally, the contents of a file may need to be put into some sort of order, sorting. Candidates should not consider these to be different utilities as a file handling utility will normally do all these tasks. Consequently, a question which asked for two examples of utility software would not be answerable by describing two file handling routines.
- An automatic backup utility. All files on a computer system need to be protected from being damaged. This is a simple routine which copies the contents of files to some other location so that if the original is damaged there is a replacement copy available. The important point here is that the routine is carried out automatically. Simply saying a back up is not a satisfactory response for a question which asks for an example of a utility because back ups can be done manually as a simple file process. A good example of a utility program would be an automatic incremental back up which would simply back up the changes to the file since the previous back up was made.
- Anti-virus software. Computers can contract a virus, which is a small program which can reproduce itself and ultimately take over spaces in memory thereby rubbing out other data and programs. A piece of anti-virus software is a utility program which is continually looking out for the characteristics of a virus and which deletes any that it finds.
- Compression software. This software reduces the sizes of files by cutting out much of the duplication of data in the stored material. This is particularly useful because it speeds up the electronic transfer of files over email connections.

1.2 Example Questions



Questions from the first half of this section may be a combination type question which will cover a number of the sections rather than be taken from a single specific section. The first question is an example of this type of question based on the systems life cycle.

1. a) State **three** different stages in the systems life cycle. (3)
- b) Describe the purpose of each of the stages you have chosen from the systems life cycle. (6)

A. a) One mark for each of three of:

Problem identification/definition

Feasibility study

Information collection

Analysis of information collected

Design

Implementation/installation/Testing

Evaluation

Documentation

Maintenance/system review

- b) Two marks per -, maximum of three – which must match (a), max 6 marks:

-Problem definition: Important for the analyst and end user to agree on the problem/Each will have their own idea as to what the problem is/Danger of wrong problem being solved if no agreement made/Each has a different expertise which are both relevant to the problem solution.

-Feasibility study: Preliminary investigation in order to determine whether a solution is desirable or even possible/looks into aspects like technical aspects/economic aspects/resources both human and software/social consequences of the solution/cost benefit analysis

-Information collection: Marks for descriptions of interviews/
questionnaires/document collection/group meetings/observation of present system

-Analysis: Of information collected/in order to establish requirements/to understand present system/to establish data handling requirements and methods

-Design: Of interface/software/data structures/error checking

-Implementation/installation/testing: Comment about software creation/methods and problems of installation/test plan related to agreed objectives/types of test carried out

-Evaluation: In relation to the agreed objectives/criteria used for evaluation/degrees of success/evaluation from different perspectives of those involved

-Documentation: Need to provide documentation for the user/ for maintenance in future/importance of ensuring that documentation is produced throughout process, not as an add on

-Maintenance: Importance of perfective/adaptive/corrective maintenance.

Notice: the answers in part (a) could be simple one word answers, no explanation is necessary for answers to questions that start with the simple keyword 'state'. Notice the large number of possible responses available for part (b). There are 9 sections to choose from and each has at least 4 possible responses, making 36 altogether for only 6 marks. This is an example of a question which the examiner can aim at different levels of ability. If the question asked for one comment about each of 6 stages in the life cycle then most candidates can be expected to be able to give one fact about each stage. Asking for 2 things about each of 3 stages is a bit more difficult. Ask for 3 things about each of 2 stages and the question is now rather difficult to get full marks on and the sort of question that asks for 4 things about a feasibility study is now very restricted and 4 marks would be aimed at the more able candidates. In this sort of question it is essential to take your cue from the number of marks available for the question, to inform you of how detailed the answer should be.

Other questions on the systems life cycle part of the syllabus will be used to test the same knowledge but could be expected to be aimed at individual sections and will tend to be of the form: 'Explain why a feasibility study would be a sensible stage in the construction of a computerised solution of a problem' or 'Given...application, explain why the social and economic factors in the feasibility study would be particularly important if a project is suggested which would automate one of the production lines'.

These can largely be divided into two types of questions: the first is an example of a question which is based on book work, generic answers are expected, while the second is very much based around the particular application and the answers should be related to that application. The first is an example of an AO1 question while the second is an AO2 question.

2. Describe how CAD/CAM can be used to produce prototype designs in a manufacturing process. (4)

A. -CAD is used to design the item

-The software can be used to carry out tests on the finished design

-Completed designs sent electronically to CAM software

-which controls robot machinery to produce the item.

Notice: Standard answers are all that will be expected from questions about the standard software packages. The facts that are produced in the text will be more than adequate for the needs of the examination.

3. A firm produces widgets for sale to the brewing industry.

A brewing company may come to the firm with a proposal for a widget to fit a particular container that will need to be produced by the firm, which will then ship the finished product, in batches, and invoice the brewing company. Sometimes the sales team will be sent to try to persuade the brewing firm of the advantages of the company's widgets.

Explain how the company can use commonly available software in the running of its business. (6)

A. -CAD can be used to design the widget

-CAM can be used to produce prototype/help set up production line

-Spreadsheets can be used to keep track of the firm's accounts

-Stock control software can be used to control the stock being stored in the warehouse

-Order processing software can be used to keep track of new orders when they arrive

-Payroll software can be used for paying the workforce

-Presentation software can be used by the salesman to impress the brewery firm.

Notice: This is another variant on a type of question that can be asked for section 1.2.m and n. It is possible to make almost any software fit the given scenario when it is that wide in scope. The important thing is not to be able to pick particular pieces of software but to give a reasonable explanation of why that software would be used in the given situation.

4. Discuss the reasons that a solicitors' practice would have for choosing an already existing word processing software package if it was decided to change the word processor currently being used because it was perceived to be out of date. (4)

A. The office manager would be keen to go ahead with the purchase as soon as possible because all the decision making would have been done and the need has already been established. When making the decision to change, current packages available would have been evaluated so there is probably some confidence in what is available. The office is likely to be a relatively small operation with few users, hence the relative costs of buying custom software compared to off-the-shelf software would be very high as would the costs of the staff training.

Notice: A very simple section of the specification but a difficult question. A question which starts with 'Discuss' is expecting a certain amount of analysis of the situation, not just hard facts. Also expected is a linking of the question to the scenario given. Notice that there is an indication in the number of marks for the question of the degree of depth of the answer. There were plenty of other points that could have been made in answer to this question but plenty have already been stated to get the four marks, and four marks means four minutes to answer the question. When you consider reading and thinking time I've certainly used up my four minutes.

5. In the same scenario, State **two** reasons why custom made software might be more appropriate than such readily available software. (2)

A. -This application may need more specialised routines than are available in the readily available software

-Many of the routines in the readily available software will not be relevant

Notice: A far simpler question than the first one, despite the fact that this is about the type of software which is generally considered more difficult for candidates because it tends to be outside their experience. It simply needs two statements of fact, the standard answers in the text, with no need to describe or explain, though notice that there is some relating to the scenario. Also note that the phrase 'off-the-shelf' software is not used because it is not mentioned in the specification.

6. Describe what is meant by a knowledge-based system. (4)

- A
- A knowledge-based system is one that has all human knowledge on a restricted topic
 - stored in its knowledge base
 - together with a number of rules for applying that knowledge/a rule base
 - and an inference engine which uses rules in the rule base to search the knowledge base
 - An HCI in order to communicate with the user.

Notice: A very definition based question. The only other type of question that would be asked is one that asks for a description of how a system would be used in a given application.

There are some areas of the specification where the topic stated is so narrow that there really is only one possible question that can be asked. It will look different from one exam paper to the next but it will be exactly the same question. Question 5 is from one of those sections and question 6 is close to being one.

7. State **three** purposes of an operating system as part of a computer system. (3)

- A.
- To control the hardware
 - To provide a platform for application software to run on
 - To provide an HCI

Notice: There are many possible responses to the question (if in doubt refer to section 1.2.q). It is sensible to ensure that there is no confusion to ensure that candidates have 3 or 4 specific answers that they will use when this sort of question comes up so that they do not start making things up. Also this is an ideal question to use to show the different style of answer required for each of the keywords that candidates might meet in an exam. Give a class the same question except that 'state' is changed to 'describe' and the mark allocation is 6 and discuss how this alters the responses that the candidate should give.

8. a) Distinguish between a *multi-tasking* and a *multi-access* operating system. (2)

- A.
- A multi-tasking operating system is one where the user of the machine is given the impression that they can carry out more than one task at a time.
 - A multi-access operating system is one where it is possible for more than one user to access the system apparently at the same time.

Notice that there are a large number of points that could have been made about both these operating systems, but most of them would not answer the question. It is important when answering a question starting with 'distinguish' to choose facts that show a comparison.

- b) State what is meant by a *distributed system*, and give an advantage of this type of multi-access system over a simple network of machines. (2)

A. - A distributed system is one which uses many storage locations on different machines to store software and files.

- Access to files can be speeded up because more than one file command can be carried out at a time.

Notice: When an advantage is asked for it is normal to state in the question, either explicitly or implicitly, with what the comparison should be made. Be careful to give an advantage using this comparison and not a more generalised one.

9. A company payroll system uses a personnel file, among others.

a) Explain the difference between batch processing and rapid response processing. (2)

b) Explain how batch processing and rapid response processing can both be sensibly used in the context of the personnel file being used to calculate the payroll. (4)

A. a) -Batch processing is the processing of data by collecting the data and then processing it in a single batch.

-Rapid response processing is processing where the result of the processing is needed immediately/real time processing.

b) -Payroll needs to be run once per week and...

-requires no human intervention, consequently perfect for batch processing.

-Enquiries by employees need to be handled on a one off basis...

-with the result of the enquiry being made available to the employee immediately.

Notice: The use of different operating systems at different times to perform tasks within the same application area provides a very popular question.

10. A computer operator takes phone calls from the public who ring up asking whether a particular item in a catalogue is available. The operator needs to type in a series of responses to questions put to the caller, so that the computer can check the file and determine whether there are any of that item available. Design a screen interface that would be suitable for the operator to use. (4)

A. -Form type interface

-Catalogue number

-Space for the description of goods which will be filled in by the computer itself

-Spaces for computer to produce availability and price

-Laid out with spaces for input.

Notice: This question is more likely to appear in module 2 examination because it is more practical, but it is a good exercise to do to check the understanding of a form based interface. If a candidate can do this question then they should be able to answer more standard types of questions. What is just as important here are the things that would not be on the screen. The question makes it quite clear that there is no ordering going on, so spaces for name and address, or method of payment, are not only going to score no marks, but will probably be penalised because they demonstrate that the candidate

has not understood the question. In this type of question it is important to demonstrate that you have taken the situation into account.

11. The technician responsible for maintaining the system in question 10 uses a command line interface.

a) Explain what is meant by a *command line interface*. (2)

b) Give **two** advantages and **one** disadvantage to the technician of using a command line interface rather than a menu based interface. (3)

A. a)-Series of commands typed at a screen prompt...

-which give specific instructions to the computer.

b)Advantages:

-Entire system is available to the technician

-Access to the particular part of the system required is gained more quickly than using other types of interface.

Disadvantage:

-The technician needs to know the commands that are available

-The technician needs to understand the way the system is designed so that it can be navigated efficiently.

Notice: The language used in this answer is not the sort of language that a candidate will use in an examination. Don't worry about this. Answers like "so that you can get around the system" are perfectly acceptable.

12.a)Explain the need to have a driver for a printer. (2)

A. -The printer will need to be able to communicate with the computer.

-The driver tells them how to communicate...

-it contains the rules for communication.

b) Give **three** different utility programs which would be part of a single user operating system, and state what each would be used for. (6)

A. -Data transfer...

-to control the transfer of data between pieces of hardware.

-File deletion...

-to delete a file which is no longer required to be stored.

-File retrieval...

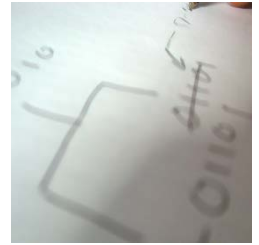
-to retrieve a file from storage into the memory of the computer.

-Automatic back up program...

-to automatically make back up copies of files in case of damage to the original.

Notice: that four were given in the answer, although only three were requested. In this case it was fortunate that an extra answer was given because the second and third examples are basically the same thing, they are file handling routines. The question specifically stated 'different'. This answer would have been awarded full marks on the basis that there were three full answers, but if an extra answer of anti-virus software had been added to the end then the examiner would be thinking that the candidate could not decide because too many possibilities were given. In cases like this the examiner would mark the first three responses and the candidate would be penalised.

1.3 Data: Its Representation, Structure and Management in Information Systems



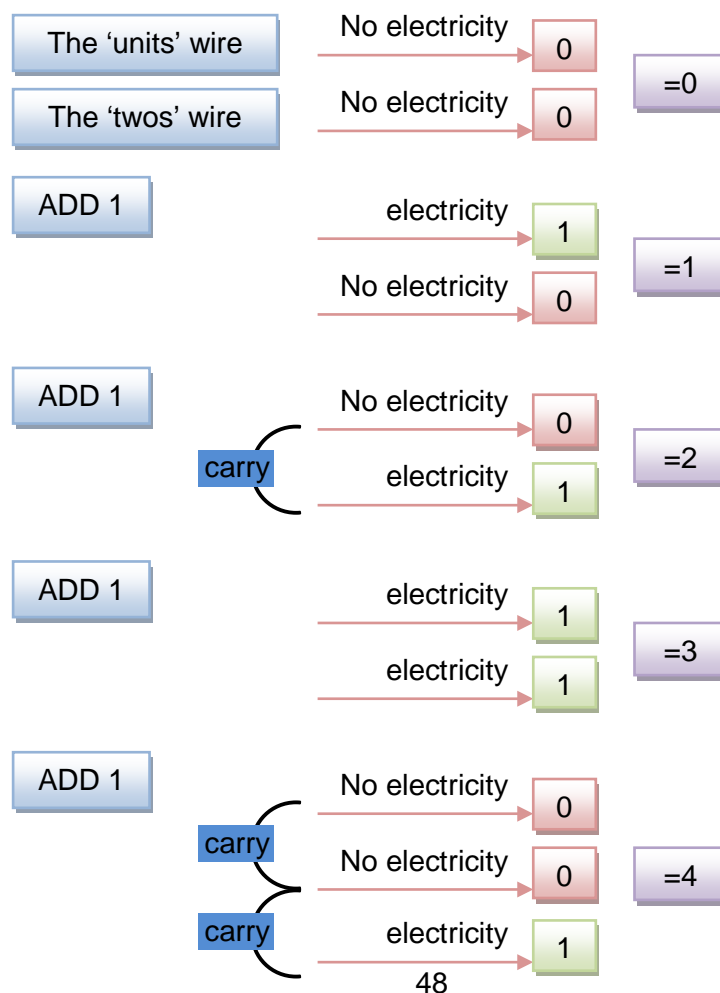
1.3.a Express numbers in binary, binary coded decimal (BCD), octal and hexadecimal

Counting is one of the first skills that a young child masters, and none of us consider counting from 1 to 100 difficult. However, to count, we have to learn, by heart, the meanings of the symbols 0,1,2...9 and also to understand that two identical symbols mean totally different things according to their 'place' in the number. For instance, in 23 the 2 actually means $2 * 10$. But why multiply by 10? Why not multiply by 6? The answer is simply that we were taught to do that because we have 10 fingers, so we can count on our fingers until we get to the last one, which we remember in the next column and then start again.

We don't need to count in tens. The ancient Babylonians counted in a system, which is similar to counting in sixties. This is very difficult to learn because of all the symbols needed, but we still use a system based on sixties today: 60 minutes = 1 hour; 60 seconds = 1 minute; $6 * 60$ degrees = 1 revolution.

Instead of increasing the number of symbols in a system, which makes the system more difficult, it seems reasonable that if we decrease the number of symbols the system will be easier to use.

A computer is an electronic machine. Electricity can be either on or off. If electricity is not flowing through a wire the lack of electricity can stand for 0. If electricity is flowing, then it stands for 1. The difficulty is what to do for the number 2. We can't just pump through twice as much electricity, what we need is a carry system, just like what happens when we run out of fingers. What we need is another wire.



The computer can continue like this for ever, just adding more wires when it gets bigger numbers.

This system, where there are only two digits, 0 and 1, is known as the binary system.

Each wire, or digit, is known as a binary digit. This name is normally shortened to BIT.

So each digit, 0 or 1, is one bit.

A single bit has very few uses so they are grouped together. A group of bits is called a BYTE. Usually a byte has 8 bits in it. (In section 1.3.d we will see that bytes can be of different sizes, but for now we will say that there are 8 bits in a byte.)

The first thing we must be able to do with the binary system is to change numbers from our system of 10 numbers (the denary system) into binary, and back again. There are a number of methods for doing this, the simplest being to use the sort of column diagrams, which were used in primary school to do simple arithmetic

Thousands	Hundreds	Tens	Units
-----------	----------	------	-------

except, this time we are using binary, so the column headings go up in twos instead of tens

← 32s	16s	8s	4s	2s	units
-------	-----	----	----	----	-------

To turn a denary number into a binary number simply put the column headings, start at the left hand side and follow the steps:

- If the column heading is less than the number, put a 1 in the column and then subtract the column heading from the number. Then start again with the next column on the right.
- If the column heading is greater than the number, put a 0 in the column and start again with the next column on the right.

Note: You will be expected to be able to do this with numbers up to 255, because that is the biggest number that can be stored in one byte of eight bits.

e.g. Change 117 (in denary) into a binary number.

Answer: Always use the column headings for a byte (8 bits)

128	64	32	16	8	4	2	1
-----	----	----	----	---	---	---	---

Follow the algorithm.

128 is greater than 117 so put a 0 and repeat.

128	64	32	16	8	4	2	1
0							

64 is less than 117 so put a 1.

128	64	32	16	8	4	2	1

0 1

Take 64 from 117 = 53, and repeat.

32 is less than 53, so put a 1.

128	64	32	16	8	4	2	1
0	1	1					

Take 32 from 53 = 21, and repeat.

If you continue this the result (try it) is

128	64	32	16	8	4	2	1
0	1	1	1	0	1	0	1

So 117 (in denary) = 01110101 (in binary).

To turn a binary number into denary, simply put the column headings above the binary number and add up all the columns with a 1 in them.

e.g. Change 10110110 into denary.

Answer:

128	64	32	16	8	4	2	1
1	0	1	1	0	1	1	0

So 10110110 = 128 + 32 + 16 + 4 + 2 = 182 (in denary).

This principle can be used for any number system, even the Babylonians' sixties if you can learn the symbols!

e.g. If we count in eights (called the OCTAL system) the column headings go up in 8's.

512	64	8	1

So 117 (in denary) is 1 lot of 64, leaving another 53.

53 is 6 lots of 8 with 5 left over. Fitting this in the columns gives

512	64	8	1
0	1	6	5

So 117 in denary is 165 in octal.

Why bother with octal?

Octal and binary are related. If we take the three digits of the octal number 165 and turn each one into binary using three bits each we get

$$1 = 001 \quad 6 = 110 \quad 5 = 101$$

Put them together and we get 001110101 which is the binary value of 117 which we got earlier.

Another system is called HEXADECIMAL (counting in 16's)

This sounds awful, but just use the same principles.

256	16	1

So 117 (in denary) is 7 lots of 16 (112) plus an extra 5. Fitting this in the columns gives

256	16	1
0	7	5

Notice that 7 in binary is 0111 and that 5 is 0101, put them together and we get 01110101 which is the binary value of 117 again. So binary, octal and hexadecimal are all related.

There is a problem with counting in 16's instead of the other systems. We need symbols going further than 0 to 9 (only 10 symbols and we need 16!).

We could invent 6 more symbols but we would have to learn them, so we use 6 that we already know, the letters A to F. In hexadecimal A stands for 10, B stands for 11 and so on to F stands for 15.

So the hexadecimal number BD stands for 11 lots of 16 and 13 units

$$= 176 + 13$$

$$= 189 \text{ (in denary)}$$

Note: B = 11, which in binary = 1011

D = 13, which in binary = 1101

Put them together to get 10111101 = the binary value of 189.

Binary Coded Decimal

Some numbers are not proper numbers because they don't behave like numbers. A barcode for chocolate looks like a number, and a barcode for sponge cake looks like a number, but if the barcodes are added together the result is not the barcode for chocolate cake. The arithmetic does not give a sensible answer. Values like this that look like numbers but do not behave like them are often stored in binary coded decimal (BCD). Each digit is simply changed into a four bit binary number which are then placed after one another in order.

e.g. 398602 in BCD

Answer: 3 = 0011 9 = 1001

8 = 1000 6 = 0110

0 = 0000 2 = 0010

So 398602 = 001110011000011000000010 (in BCD)

Note: All the zeros are essential otherwise you can't read it back.

1.3.b Describe and use two's complement and sign and magnitude to represent negative integers

If a computer system uses a byte to store a number in the way that was suggested in 1.3.a there are three problems that arise:

1. The biggest number that can be represented is 255 because there aren't enough bits to store bigger numbers. This is easily solved by using more than one byte to represent a number. Most computer systems use either two or four bytes to store a number. There is still a limit on the size that can be represented, but it is now much larger.

2. The second problem is not so easy to solve, how to represent fractions. This will be looked at in section 3.4.

3. Negative numbers cannot be stored. This is fairly simple to solve and we will look at two ways of representing negative integers (negative whole numbers) now.

The example we used for binary storage was 117 which becomes 01110101 in binary. If we want to store +117 or -117, these numbers need a second piece of data to be stored, namely the sign.

Method 1: Sign and Magnitude.

Use the first bit in the byte (this is called the most significant bit (MSB)) to represent the sign (0 for + and 1 for -) instead of representing 128. This means that

$$+117 = 01110101 \quad \text{and} \quad -117 = 11110101$$

Notes: The range of numbers possible is now -127 to +127 because we only have 7 bits to store the size of the integer instead of having 8.

The byte does not represent just a number but also a sign, this makes arithmetic difficult because it is difficult to tell the difference between the MSB and all the other bits. They look the same and the danger is that if you try to add two negative numbers, when it comes to the MSB to say

$$1 + 1 = 0 \text{ carry } 1$$

doesn't make any sense.

Method 2: Two's Complement

The MSB stays as a number, but is made negative. This means that the column headings are:

-128	64	32	16	8	4	2	1
------	----	----	----	---	---	---	---

+117 does not need to use the MSB, so it stays as 01110101.

Whereas:

$$-117 = -128 + 11$$

$$= -128 + (8 + 2 + 1)$$

fitting this into the columns gives 10001011

Two's complement seems to make everything more complicated for little reason at the moment, but later it becomes essential because it makes the arithmetic easier.

One way of calculating the two's complement value of a negative number is shown above, however there is a nice algorithm which most people find easier:

- Work out the binary value of the positive number (make sure you write down the whole number including leading zeros)
- Change all the digits, 0 for 1 and 1 for 0
- Add 1

Taking our example of -117:

Stage 1 is work out the positive binary value:

01110101

Stage 2 is to change all the digits:

10001010

Stage 3 is to add 1:

10001011

..which matches the value we got using the other method.

1.3.c Perform integer binary arithmetic: addition and subtraction

The syllabus requires candidates to be able to add two binary integers, and also be able to take one away from another. The numbers and the answers will be limited to one byte.

Addition

There are four simple rules

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$

and the difficult one

$$1 + 1 = 0 \text{ (Carry 1)}$$

e.g. Add together the binary equivalents of 91 and 18

Answer:

$$\begin{array}{r} 91 = \quad 01011011 \\ 18 = \quad 00010010 \quad + \\ \hline 01101101 \\ \hline \quad 1 \quad 1 \end{array} = 109$$

Subtraction

This is where two's complement is useful. To take one number away from another, simply write the number to be subtracted as a two's complement negative number and then add them up.

e.g. Work out $91 - 18$ using their binary equivalents.

Answer: $91 = 01011011$

-18 as a two's complement number is

$$\begin{aligned} & -128 + 110 \\ & = -128 + (+64 + 32 + 8 + 4 + 2) \\ & = 11101110 \end{aligned}$$

Now add them

$$\begin{array}{r} 01011011 \\ 11101110 \quad + \\ \hline 1/01001001 \\ \hline 1111111 \end{array}$$

But the answer can only be 8 bits, so cross out the 9th bit, giving:

$01001001 = 64 + 8 + 1 = 73.$

Notes: Lots of carrying here makes the sum more difficult, but the same rules are used.

One rule is extended slightly because of the carries, $1+1+1 = 1$ (carry 1)

Things can get harder but this is as far as the syllabus goes.

1.3.d *Explain the use of code to represent a character set (ASCII, EBCDIC and UNICODE)*

There are other types of data that have to be stored in computers besides numbers, for instance, the letters of the alphabet.

The complete set of characters that the computer can recognise and use is known as its character set. Each of the characters in the character set must have its own binary value which is its code by which the computer recognises it.

These codes are stored as binary numbers. For instance:

A could be stored as 000 B as 001 and so on.

Unfortunately, there are only 8 possible codes using 3 bits, so we could store the letters A to H but not the rest, and what about the lower case letters and punctuation and...? The computer can store as many characters as necessary simply by using more and more bits for the code.

The size of the character set and the characters that the computer needs to be able to recognise depend a lot on what the computer is meant to be able to do. Some systems don't need to be able to recognise a lot of characters so they only use a few bits for each character. A good example of this is an ATM (cash machine). There are very few characters that are needed, for instance the alphabet is not needed. It is an interesting exercise to study the input requirements of a cash machine and to determine how many bits are necessary to code that many characters.

The number of bits needed to store one character is called a byte which is usually said to have 8 bits because most systems use 8 bits to store the code for each character.

A good way of thinking about the character set for a normal computer is to have a look at the characters that are available on its keyboard.

If we use 8 bits for a byte we now have enough codes, but another problem arises. If my computer stores A as 01000001 and your computer stores A as 01000010 then the computers cannot communicate because they cannot understand each other's codes. In the 1960's a meeting in America agreed a standard set of codes so that computers could communicate with each other. This standard set of codes is known as the ASCII set. Most systems use ASCII so you can be fairly sure that when you type in A it is stored in the computer's memory as 01000001. However, you can't be sure because some systems use other codes.

EBCDIC: A less common code developed by IBM for use with its main frame computers and is sometimes used by other larger scale computer systems. Like ASCII it uses 8 bit codes for characters but differs both in the codes used and the characters that are represented.

UNICODE: is a 16 bit code that can represent over 65000 characters. This means that all the characters used by the world's languages can be represented which makes it very useful in, for instance, document handling where a single document needs to be written in English, Arabic and Chinese, say. It also allows the localisation of software where standard software can be adapted for use by different cultures by modifying the layout and features of the interface. Both Microsoft and Apple have added, or have announced an intention to add, Unicode support to their operating systems.

Section Notes

All the characters that a system can recognise are called its character set.

ASCII uses 8 bits so there are 256 different codes that can be used and hence 256 different characters. (This is not quite true because standard ASCII uses 7 bits for the character and saves the eighth one as a means of checking the rest. We will be looking at how this is done in section 1.5.e)

There are lots of different character sets available to the computer for different tasks. Two to consider when thinking about how many bits are needed in the standard byte are:

ANSI which includes lots of graphical symbols, lines and shapes;

The standard Chinese alphabet character set. A Chinese keyboard has thousands of different characters.

A problem will arise when the computer retrieves a piece of data from its memory.

Imagine that the data is 01000001. Is this the number 65, or is it A?

They are both stored in the same way, so how can it tell the difference?

The answer is that characters and numbers are stored in different parts of the memory, so it knows which one it is by knowing whereabouts it was stored.

1.3.e *Describe manual and automatic methods of gathering and inputting data into a system*

All computer systems need to have data input to them otherwise they have nothing to process. The methods of collecting the data can be divided into two types: automatic and manual data collection.

Automatic Data Collection

The most obvious type of automatic data collection is in a control system where the computer collects its view of the outside world from sensors that give information about the physical environment. The data collection done by a sensor is continuous, but the reading of the data is within a fixed time period (the processor does not want to know the temperature in the room all the time, but perhaps every 5 minutes. This gives the previous decision, about whether or not to turn the heating on, long enough to have had some effect.) The use of only some of the available data is known as sampling.

Many sensors that measure physical values are analogue sensors while the data required by the processor needs to be digital. Analogue data is physical data that creates a signal which consists of continuously changing voltage (for example, a thermistor increases the voltage output as the temperature which it is measuring increases). This signal must be changed into a stream of 0s and 1s that the computer can recognise. This is done by an analogue to digital converter.

When data is collected off-line, often by sensors in remote locations and then stored until ready for input to the system at a time that is convenient to the system, it is known as data logging. A typical data logger will be in the form of a disk drive, on which the data is stored until a set of data has been collected and the data can be entered into the system in one go. Obviously, this would not be suitable data input for a system which was controlling the central heating in a house, but a remote weather station on a mountain top where different readings are taken every 10 minutes and then radioed back to the weather centre once every 24 hours would need just such a device to store the data until it was required.

There are many different types of sensor depending on the variable type that they are intending to measure. Whatever the value to be measured, if it is a physical value there is almost certainly a type of sensor to measure it. Be careful not to confuse the use of sensors in a computer system with the idea of a mechanical control system. There is a difference between a thermistor (temperature sensor) collecting data about the temperature of a room and passing it to a computer which can then make decisions about the need for heating or cooling according to an algorithm that it works through and a switching device like a bimetal strip which bends as it cools and when it has moved enough it trips a switch to turn on a heater. This is a mechanical switch with no decision making element in it.

Bar codes

Less obvious forms of automatic data collection are barcodes in a supermarket. The code is translated into a series of dark coloured bars on a light background so that the data can be input to the machine without any further preparation. Candidates should not worry about having to explain the dynamics of a bar code but they should be aware of the need for checking the input by use of a check digit contained within the code. The bar code question will almost certainly be contained within another question, either about automatic data collection or about stock systems in warehouses or supermarkets.

Automatic data collection can be considered to be any data collection that does the two stages of data collection and data input to the system without going through the intermediate phase of data preparation to make it suitable for computer use.

Optical Mark Recognition/Reading (OMR)

Another good example is the school register which is taken by making marks on a sheet of paper and that can then be read directly into the computer with no human intervention by an optical mark reader (OMR). The essential point about OMR is that information is read by translating the position of the mark on the paper into a meaning, so that two marks side by side on the paper mean different things because of where they are rather than what they look like. The school register is only one of many applications. The basis of a question could be about the information contained and how it is contained and read, or it will be about the suitability for particular uses.

Voice Recognition

Another method of automatically entering data is by voice recognition, which is particularly important for disabled users. It is done by using a microphone to communicate with the computer. The sound waves produced are then sampled and digitised. The digital values produced are then compared with a library of digital values stored on the computer and interpreted by the computer. Two problems that arise are the need to train the computer to a particular voice and problems with background sound distorting the original meaning. The smaller the vocabulary of the system and the masking of background noise, the more effective it is which is why voice recognition is particularly useful to simple menu commands in telephone enquiry systems.

Magnetic Ink Character Recognition

A form of data input which is almost uniquely used by banks is the magnetic ink characters that are printed on the bottom of cheques before being sent to the account holder. The magnetic ink is particularly easy for the computer to read and contains enough information to identify the bank and the account at that bank. All of this can be done with no further human intervention after the original printing of the cheque book. However, the data that is written on the cheque by the customer (who it is made out to and for how much) is not ready for input and hence requires some human intervention to make it useable.

Chip and Pin Technology

Personal information is extremely sensitive and valuable. It needs to be protected as far as possible. Personal information and access to it, on a credit or debit card was always controlled/stored on a magnetic strip on the back of the card. These strips are still used on things like car park tickets where the information is not very valuable. However, personal information on cards now tends to be stored in a computer chip embedded into the card and requiring an identification code (PIN) to gain access to it.

Image Capture

A 'posh' phrase for taking and storing pictures. With phones now used as cameras (even video cameras) and all sorts of devices being able to take a picture (capture an image), even credit cards with a picture chip in them, it no longer needs explanation as to how the user takes the picture. The picture is stored electronically. This can use up a large amount of memory to store images to any degree of detail, so the electronic versions of the images are compressed before they are stored in order to save space on the storage device. There are many ways to take/capture/compress/store the images, so for the sake of standardisation, most manufacturers/users have agreed on accepted ways of doing it. There are two in particular which are widely used, they are called jpeg format (a standard way of storing still pictures) and mpeg (a standard way of storing moving pictures).

Scanner

A scanner is a device that shines a strong light at a source document and then reads the intensity of the reflected light. The surface of the document is divided into small rectangles, or pixels, and the light intensity is measured of each pixel, it is then reported to the computer as a bit map. Scanners can be of different sizes, typical is an A4 sized flat bed scanner where the document is placed on a sheet of glass which is then scanned line by line, or a hand held scanner which can be rolled across the image a number of times collecting a band each time, these bands of image can then be matched up by the software to produce the complete document.

Video Capture Card

A video picture is made up of a series of images which are changed approximately 26 times per second in order to fool the brain into thinking that the images are moving. A video capture card is an interface board which fits into one of the expansion slots in a processor that allows the processor to store the values of the screen pixels for a specific picture. In other words it allows the action to be frozen. A typical example of the use of a video capture card is the market stall that uses a video camera to take an image of a customer and then to select one image to print onto the T shirt.

Digital Camera

Works in a similar way to an optical camera but does not store the image on film. Instead, the image is stored electronically enabling the user to download it into a computer and to manipulate the image and print out the images if desired.

Manual Data Entry

The most obvious is the form, designed to collect data, which needs to be input to the computer. An operator reads the data on the form and then types it into the computer via a keyboard. An extra stage has been added here, the data has had to be typed in, in other words the original data was not in a form acceptable to the computer.

Touch Screens

When the method of input does not rely on a keyboard then one of the stages has been removed. Often touch screens are used. They tend to be used when there are a finite number of possible choices and the user is restricted to what is available on the screen. Use of this technology is most often controlled by the environment: Danger from the weather so a keyboard cannot be used; susceptibility to vandalism so a mouse is not sensible...

Optical Character Recognition/Reading (OCR)

Computer systems are available that will read individual characters and input them without the data having to be transcribed, this would count as automatic data collection (it is known as optical character reading (OCR)). OCR has become very much more reliable as algorithms to recognise the characters have improved. Rather like the sound recognition methods, the computer samples the letter that is scanned and compares it to a library of similar images, selecting the best fit. Major improvements in reliability have come about because of improvements to the algorithms by increasing the sampling, expanding the library of characters to compare and also to improve predictive algorithms so that combinations of characters can be accepted/rejected dependent on meaning.

Questions on this part of the syllabus will be suggesting suitable input methods for particular situations, and offering advantages and disadvantages for particular forms of data input in different situations, or, given a situation the candidate is expected to be able to offer sensible suggestions of methods.

1.3.f *Explain the techniques of validation and verification, and describe validation tests that can be carried out on data*

When data is input to a computer system, it is only valuable data if it is correct. If the data is in error in any way, then no amount of care in the programming will make up for the erroneous data and the results produced can be expected to be unreliable. There are three types of error that can occur with the data on entry. The first is that the data, while reasonable, is wrong. If your date of birth is written down on a data capture form as 18th of November 1983, it will (except in very rare cases) be wrong. It can be typed into the computer with the utmost care as 181183, it can be checked by the computer to make sure that is a sensible date, and will then be accepted as your date of birth despite the fact that it is wrong. There is no reason for the computer to imagine that it may be wrong, quite simply, when you filled out the original form, you made a mistake. The second type of error is when the operator typing in the data hits the wrong key and types in 181193, or the equivalent. In this case an error has been made that should be able to be spotted if a suitable check is made on the input. This type of data checking is called verification. The third type of error is when something is typed in which simply is not sensible. If the computer knows that there are only 12 months in a year then it will know that 181383 must be wrong because it is not sensible to be born in the thirteenth month. If a sixth former's birthday is typed in as 161183 then it is not sensible because the student would be too old. Checks on the sensibility of the data are called validation checks.

Faulty Data

There is very little that can be done about faulty data except to let the owner of the data check it visually on a regular basis. The personal information kept on the school administration system about you and your family will be printed off at regular intervals so that your parents can check to ensure that the stored information is still correct.

Verification

Verification means checking the input data with the original data to make sure that there have been no transcription errors. The standard way to do this is to input the data twice to the computer system. The computer then checks the two sets of data (which should be the same) and if there is a difference between the two sets of data, the computer knows that one of the inputs is wrong. It won't know which one is wrong but it can now ask the operator to check that particular input. Another method is the one which you use when typing data into the computer. You will look at the screen from time to time and check that you have not made a mistake by reading your input. This visual check is a form of verification.

Validation

Validation is a check on DATA INPUT to the system by comparing the data input with a set of rules that the computer has been told the data must follow. The words data input have been typed in capitals in order to make them stand out. One of the biggest mistakes that students make is to confuse validation with another type of checking mechanism used when data is passed around a system. This will be introduced in section 1.5.e.

Validation is carried out by the computer system itself, using rules about what type of data to expect. These rules are called validation rules. If the data does not match up with the rules, then there must be an error. There are many different types of validation rule that can be used to check input in different applications:

1. Range check. A mathematics exam is out of 100. A simple validation rule that the computer can apply to any data that is input is that the mark must be between 0 and 100 inclusive. Consequently, a mark of 101 would be rejected by this check as being outside the acceptable range.
2. Character check. A person's name will consist of letters of the alphabet and sometimes a hyphen or apostrophe. This rule can be applied to input of a person's name so that dav2d will immediately be rejected as unacceptable.
3. Format check. A particular application is set up to accept a national insurance number. Each person has a unique national insurance number, but they all have the same format of characters, 2 letters followed by 6 digits followed by a single letter. If the computer knows this rule then it knows what the format of a NI number is and would reject ABC12345Z because it is in the wrong format, it breaks the rule.
4. Length check. A NI number has 9 characters, if more or fewer than 9 characters are keyed in then the data cannot be accurate.
5. Existence check. A bar code is read at a supermarket check out till. The code is sent to the main computer which will search for that code on the stock file. As the stock file contains details of all items held in stock, if it is not there then the item cannot exist, which it obviously does, therefore the code must have been wrongly read.
6. Presence check. This is widely used with a form based HCI (section 1.2.t). Some of the fields/cells that are to be filled in on the screen are more important than others and must be filled in. For instance, a person's surname is essential to a particular application but giving their first name is optional. The computer would check that the surname actually had some value in it, whatever it was. This is known as a presence check. Note that the presence check would not be used on the first name in this example.

Existence and presence checks are often confused, be careful.

7. Check digit. When the code is read on the item at the supermarket, it consists of numbers. One number is special, it is called the check digit. If the other numbers have some arithmetic done to them, using a simple algorithm, the answer should be this special digit. When the code is read at the check out till, if the arithmetic does not give the check digit it must have been read wrongly; it is at this point that the beeping sound would normally be heard if everything is alright.

For example, suppose the code of the item to be entered is 1375, a check digit may be calculated by giving each digit a weight of 5, 4, 3 and 2 and summing the results.

$$5 \times 1 = 5$$

$$4 \times 3 = 12$$

$$3 \times 7 = 21$$

$$2 \times 5 = 10 \quad \text{the sum is } 5 + 12 + 21 + 10 = 47$$

The next step is to divide this sum by a suitable number and note the remainder. It is normal to use 11 although any prime number will work.

47 divided by 11 gives a remainder of 3.

This would be called a modulus 11 check digit because this type of arithmetic where we are interested in the remainder after a division sum is done is called modulo division.

This means that the number stored as a bar code is 13743.

A problem can arise using modulus 11 if the remainder is 10.

When the calculation gives a check digit of 10, this is 2 digits, not 1! When this happens X is used as the check digit. A good example of the use of modulus 11 is the ISBNs used on books.

1.3.g *Describe possible forms of output, stating the advantages and disadvantages of each with reference to the target audience*

When data has been processed by a computer system it is necessary to report the results of the processing. There are a number of different ways that the results can be reported to the user.

Graphs

Graphs show trends very clearly. Different types of graph can illustrate different characteristics, and when two variables need to be compared, a visual representation can be very useful. However the importance of the scales is paramount because otherwise a very misleading picture can be given. Also, the specific values are not easily read from a graph, indeed, in a continuous distribution, it is simply not possible to take reliable readings to any degree of accuracy.

Reports

A report is a hard copy printout of the values of variables. This has the advantage of producing the actual figures according to the values specified by the user. However, the figures themselves may need skill to interpret their significance and the value of figures in a vacuum is often hard to justify.

Interactive Presentations

The previous forms have relied on the format of the report being decided without the luxury of being able to see what the figures look like in the first place. If the system allows the user to decide the type and range of output required during the run, then there is some positive user involvement leading to an interactive presentation where the user can adjust the output to suit the example.

Sound

Many applications do not lend themselves to a standard, visual, printout. Sound can be used for output from some systems. Obvious examples would be voice synthesis for reporting to blind people and an alarm system to protect property against burglars.

Video

Video is a visually satisfying form of output that takes large amounts of memory to produce because the nature of the medium requires large quantities of pictures to produce the feel of continuous motion. Video is useful for demonstration of techniques where there is little value in pages of instruction if a simple video can illustrate something better.

Images

Images, or pictures, can be used to enhance understanding. These may be created using graphics packages, may be scanned into the computer or imported from a camera. By using a number of slightly different images, animation can be created.

Animations

Provide a good stimulus for an audience and lead from one slide to another when making a slide based presentation. Animation takes considerably less processing power than other forms of motion, unless the image being animated is complex. Animation is used so often that it can come across as being a boring technique that has just been added for 'gloss'.

Care needs to be taken when using animation as it can detract from the presentation if it involves complex changes.

A simple animation can be created by using just two images. For example, storing two pictures of a Christmas tree, one plain green and the other containing small coloured circles, and switching quickly between them can create the illusion of flashing lights.

Output According to Target Audience

Imagine an intensive care ward at a hospital. There are six beds, each with a patient who is being monitored by a computer. The outputs are available for a variety of users. There is a nurse at a desk at one end of the ward. The nurse has other duties, but is expected to make the rounds of the patients to check on their progress at regular intervals. Doctors come round the ward twice a day to check on the patients and make any adjustments to their medication.

If a patient is sensed by the computer system to have suffered a relapse while the nurse is sitting at the desk, a sensible output would be sound, some sort of alarm to bring the notice of the nurse to the fact that something is wrong. This may be accompanied by a flashing light, or some other device, to quickly draw attention to the particular one of the six patients needing attention. When the nurse goes around the patients to make a visual check of their conditions, it is not necessary to know exact figures of heart rate or blood sugar, a quick glance at a screen showing a scrolling graph of the state of the patient's vital signs over the last 20 minutes will be perfectly adequate. If the graph looks in any way abnormal, it may be necessary to get a printout of the actual values of the variables for that patient to determine what action, if any, needs to be taken. The doctor may well want to see a printout of all the variable values for the last twenty four hours, particularly if there is something happening to the patient which is difficult to understand, such historical data can hold the clue to present symptoms. The doctor may change the medication or the parameters within which the patient can be considered to be stable, this will involve the nurse resetting values on the scales of the graphical output, or even resetting the parameters for setting off the audible alarm. This involves the nurse in using an interactive presentation with the system. Once a week the nurse takes a first aid class at the local sixth form college. There are too many students for a one to one presentation all the time, so the college computer system has been loaded with demonstration software showing an animation of the technique for artificial respiration.

When considering output, always consider the importance of timeliness and relevance. Data tends to have a limited life span, which can be different for the same data in different situations. The data on heart rate from 3 hours ago is not going to be of importance to the nurse looking after the patient, but it may be of great value to the doctor in providing a clue as to the reason for a sudden change in condition. Some data is not relevant to particular situations, however up to date it is. The fact that a patient has blue eyes has no bearing on their physical state and consequently should not be considered relevant to this example, although it may well be in other circumstances.

1.3.h *Explain the procedures involved in backing up data and archiving, including the difference between data that is backed up and data that is archived*

Backing up data

Data stored in files is very valuable. It has taken a long time to input to the system, and often, is irreplaceable. If a bank loses the file of customer accounts because the hard disk crashes, then the bank is out of business.

It makes sense to take precautions against a major disaster. The simplest solution is to make a copy of the data in the file, so that if the disk is destroyed, the data can be recovered. This copy is known as a BACK UP. In most applications the data is so valuable that it makes sense to produce more than one back up copy of a file. Some of these copies will be stored away from the computer system in case of something like a fire which would destroy everything in the building.

The first problem with backing up files is how often to do it. There are no right answers, but there are wrong ones. It all depends on the application. An application that involves the file being altered on a regular basis will need to be backed up more often than one that is very rarely changed (what is the point of making another copy if it hasn't changed since the previous copy was made?). A school pupil file may be backed up once a week, whereas a bank customer file may be backed up hourly.

The second problem is that the back up copy will rarely be the same as the original file because the original file keeps changing. If a back-up is made at 9.00am and an alteration is made to the file at 9.05am, if the file now crashes, the back up will not include the change that has been made. It is very nearly the same, but not quite. Because of this a separate file, of all the changes that have been made since the last back up, is kept. This file is called the transaction log and it can be used to update the copy if the original is destroyed. This transaction log is very rarely used. Once a new back up is made the old transaction log can be destroyed. Speed of access to the data on the transaction log is not important because it is rarely used, so a transaction log tends to use serial storage of the data and is the best example of a serial file if an examination question asks for one.

Archiving data

Sometimes data is no longer being used. A good example would be in a school when pupils leave at the end of the sixth form. All their data is still on the computer file of pupils, taking up valuable space. It is not sensible to just delete it, there are all sorts of reasons why the data may still be important. For instance a past pupil may ask for a reference. If all the data has been erased it may make it impossible to write a sensible reference. Data that is no longer needed on the file but may be needed in the future should be copied onto a long term storage medium and stored away in case it is needed. This is known as producing an ARCHIVE of the data. (Schools normally archive data for 7 years before destroying it).

Note: Archived data is NOT used for retrieving the file if something goes wrong, it is used for storing little used or redundant data in case it is ever needed again, so that space on the hard drive can be freed up.

1.3 Example Questions



1. a) Express the number 113 (denary) in

(i) binary

(ii) in BCD

using an appropriate number of 8-bit bytes in each case.

(4)

b) Using the answer obtained in part (a) show how 113 (denary) can be expressed in

(i) octal

(ii) hexadecimal.

(4)

A. a) (i)

128	64	32	16	8	4	2	1
0	1	1	1	0	0	0	1

= 01110001.

(ii) 1 = 0001

1 = 0001

3 = 0011

Therefore 113 = 0000000100010011 (BCD)

b) (i) 113 = 001 110 001 in binary

= 1 6 1 in octal.

(ii) 113 = 0111 0001 in binary

= 7 1 in hexadecimal.

Notice: (i) and (ii) It was necessary to show a method of working out. Many students have calculators that will automatically change from one number system to another, so it is necessary to show that you know what you are doing, even if you use a calculator to check the results. Also, the question stated that the appropriate number of bytes be used, in part (i) this is obviously 8 bits and is an easy mark, but in part (ii) it is necessary to add a set of zeros to the front of the answer to make it a whole number of bytes.

(iii) and (iv) the question stated that the first answer had to be used, so one of the two marks is going to be given for showing the relationship between binary and each of these two representations. Notice that for the octal answer it was necessary to add a 0 to the front of the binary number to give 9 bits (3 lots of 3).

2. Explain how the denary number -27 can be represented in binary using

(i) sign and magnitude

(ii) two's complement

notation, using a single, 8-bit, byte for each answer.

(4)

A. (i)

+/-	64	32	16	8	4	2	1
1	0	0	1	1	0	1	1

$= 10011011$

(ii) $-27 = -128 + 101$

$$= -128 + (+64 + 32 + 4 + 1)$$

-128	64	32	16	8	4	2	1
1	1	1	0	0	1	0	1

$= 11100101$

Notice that the question said "Explain...", so just writing the answer down is not acceptable. There will be a mark for showing the column headings, particularly the value of the MSB in each case.

3. Add together the binary equivalents of 34 and 83, using single byte arithmetic, showing you working.

(3)

A.

$$\begin{array}{r} 34 = 00100010 \\ 83 = 01010011 + \\ \hline 01110101 = 117 \\ \hline 1 \end{array}$$

Notice: that the question asked for the working. The part that shows that you are capable of doing the arithmetic is the carry, don't miss it out. In a question like this try to ask yourself what evidence the examiner is expecting for the mark. The other marks are for using 8 bits for each value, and for the answer.

4. Describe how characters are stored in a computer.

(3)

A. -Each character is given a code...

-as a binary number

-Each character code occupies one byte of data

-Typically one byte is eight bits

-Most computers use a standard set of characters like the ASCII set.

Notice: There are lots of marks available here. The question has been left deliberately open so that there are plenty of mark points available.

5. a) State **two** methods of data entry used by banks in their cheque system.

(2)

b) Explain why banks find the use of your two examples suitable for this application.

(4)

A. a)-Keying in of data, either to a disk for later entry, or directly onto the cheque in machine readable form.

- MICR

b)-Keying in can be used to place the details of the payee onto the cheque together with the amount,

- in machine readable form,

- or to store the data on a disk for future use.

- MICR is already in machine readable form

- placed on cheques at time of printing the cheque book

- means that complex figures like account number do not have to be keyed in which would invite human error.

Notice: Although the answer is given as bullet points, the response expected would be a prose explanation.

6. A small stall is to be opened, as part of a fairground, where the customer can have their likeness printed on to the front of a sweatshirt. Describe two possible methods of capturing the image to be printed.

(4)

A. -Digital camera

- connected to the computer which uses

- software to crop and present the image to the printer

- Image stored in jpeg format

- Camcorder where the image is sent to..

- a video capture card which produces a still image on a screen in the same way as a digital camera does

- where it will be stored in mpeg format

(1 per —, max 2 per method, max 4)

Notice: Digital camera and Camcorder are not mentioned in the specification. However, the examiner expects you to have a working knowledge of other input types/devices not stated in the specification. The specification actually says 'including...', in other words any input method may be alluded to in the question, although, in practice, those used in the exam will be common and relevant to a simple situation.

7. A mail order firm receives orders from customers on paper order forms. The data from these forms are keyed into the computer system by operators. The data that is to be keyed in includes the 5 digit article number, the name of the customer and the date that the order has been received.
- a) Explain how the data input would be verified. (3)
- b) Describe three different validation routines that could be performed on the data. (6)

A. a)-Two operators would...

-independently key in the data

-The two copies of the data are then compared..

-by the software..

-and errors are reported to the operators.

b)-Article number can have a length check carried out on it...

-if there are not 5 characters then the article number must be wrong.

-Name of customer can be checked with a character check...

-any characters other than letters or hyphen or apostrophe must mean that the check has been failed.

-The date can be subject to a range check (actually a number of range checks)...

-the first two characters must be less than 32.

Notice: Do not use the verification technique of printing out the data so that the customer can check it. This is a mail order company and hence this would be impossible. There are many possible alternatives for part (b). Choose them carefully so that there are three different ones and so that all three pieces of data are used. Notice that there are two marks for each one, meaning that the examiner wants to know what the type of check is, and also the rule that has to be followed by the data in order to be treated as valid.

8. A reaction vessel in a chemical plant is monitored, along with many others, by a computer system using a number of sensors of different types. Describe **three** different types of output that would be used by such a system, stating why such a use would be necessary. (6)

A. -Graphs (of the temperature, pressure...) showing the general state of the reaction vessel..

-to show the operator the trends in the vessel, for example shows clearly whether the temperature is increasing.

Report (of temperature)...

-while the graph shows a trend, the report gives precise figures.

-Sound...

-an alarm would sound if the temperature went past a safe limit.

-Hard copy printout...

-to allow investigators to study problems that may cause a shutdown or unacceptably poor product.

Notice: While knowledge of a chemical reaction is not part of a computing syllabus, it is reasonable to expect students to realise that heat, or some other sensible parameter, would play an important part in the reaction.

9. A library keeps both a book file and a member file. The library does a stock take twice a year and orders new books only once a year. Members can join or cancel their membership at any time.

a) Describe how the library can implement a sensible system of backing up their files.

(4)

b) Explain the part that would be played by archiving in the management of the files.

(4)

A. a)-Book file needs to be backed up twice (or three) times a year...

-when the stock take or book purchasing has made the file alter.

-Member file needs backing up daily (at least weekly) because of constant changes...

-would also need to keep a transaction log for the member file.

-Back up copies would be stored away from the building with the computer system in it to ensure that a copy of files survived in case of fire.

-Multiple copies of the book file would be made

-Member file copies and transaction logs may be kept for a number of back up periods.

b)-When books are discovered to be missing, or if a book is replaced by a more up-to-date edition, the old records should be kept...

-but they are no longer live so are taken from the hard disk...

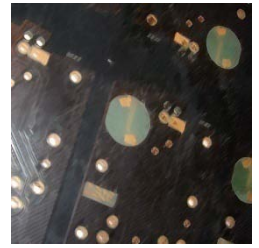
-after a copy (archive) has been made.

-When members leave the library the data should be archived...

-also when they have not taken a book out for a long period of time their record can be considered to be dormant.

Notice: Backing up and archiving are topics which cause confusion among candidates. There really is no need for such confusion if you remember that archiving has nothing to do with recovery after a disk failure. Note that there are plenty of mark points available in this question.

1.4 Hardware



1.4.a *Describe the function and purpose of the control unit, memory unit and ALU as individual parts of a computer*

The first thing to be said in this section is what is not necessary. The three topics that are listed in the heading would normally lead to the fetch-execute cycle, this does not come into this module. Likewise, any complicated diagrams showing how the various parts of the processor are connected up and communicate with each other do not appear in the syllabus until the A2 section, next year. The requirements of this section are very simple; they are restricted to a basic understanding of what these three parts of the processor are meant to do.

Control Unit

All computers follow instructions that are given to it in a program. These instructions are in a particular order in the program, and following them, and carrying them out, will mean that the computer has accomplished whatever task it was meant to do. Something, in the computer, has to manage the instructions and make sure that all the other parts of the processor do what they should be doing. This is the job of the control unit.

The control unit has three jobs

1. It has to decide which instruction to carry out next and then go and get it.
2. It has to decipher the instruction, in other words it has to work out what needs to be done to carry the instruction out.
3. It has to tell other parts of the processor what they should do so that the instruction will be carried out.

Arithmetic Logic Unit (ALU)

The ALU is where things are actually done in the processor.

1. The ALU contains circuitry that allows it to add numbers together (do arithmetic).
2. It allows for logic decisions to be made (If the value is negative then ..., the part of the processor that makes the decision is the ALU).
3. The third task of the ALU is to act as the gateway between the processor and the other parts of the computer system. All input and output to and from the processor goes into the ALU and waits there while the control unit decides what to do with it.

Memory Unit

The third part of the processor is where everything that the processor is going to use is stored. This includes all the program instructions and all the data needed to carry out those instructions. One of the jobs of the control unit is to be able to find the information stored there when it is to be used.

The parts of the operating system, which the computer is using at the time also, need to be stored in memory.

1.4.b *Explain the need for and the use of registers in the functioning of the processor*

The three parts of the processor use a series of special registers in order to process the instructions in a program. The ones that are specified in the specification are:

Register	Meaning
PC	Program Counter
CIR	Current Instruction Register
MAR	Memory Address Register
MDR	Memory Data Register
Accumulator	Holds results

We need to understand the purpose of each of these registers, though at the moment we will be working in something of a vacuum because we don't need to understand how they fit together or interrelate with each other until we meet them again in sections 3.3.a and 3.3.b.

- The program counter keeps track of whereabouts in the computer's memory the next instruction can be found so that a copy of the instruction can be placed in the current instruction register. Sometimes the program counter is called the Sequence Control Register (SCR) as it controls the sequence in which instructions are executed.
- The current instruction register holds the instruction that is to be executed while it is being executed.
- The memory address register is used to hold the position in memory (the address) that contains either the next piece of data or an instruction that is to be used.
- The memory data register acts like a buffer and holds anything that is copied from the memory ready for the processor to use it.
- The central processor contains the arithmetic logic unit (also known as the arithmetic unit) and the control unit. The arithmetic logic unit (ALU) is where data is processed. This involves arithmetic and logical operations. Arithmetic operations are those that add and subtract numbers, and so on. Logical operations involve comparing binary patterns and making decisions.
- The control unit fetches instructions from memory, decodes them and synchronises the operations before sending signals to other parts of the computer.
- The accumulator is in the arithmetic unit, the program counter and the instruction registers are in the control unit and the memory data register and memory address register are in the processor.

1.4.c *Explain the need for and describe the use of buses to convey information*

A bus is a path down which information can pass. If you think of a bus as being a set of wires which is reserved for a particular type of information you won't go far wrong, although it doesn't have to be wires.

There are three different types of bus that we are interested in. They will all look the same but the difference is in what they are used to convey.

1. Data bus. This is used to carry the data that needs to be transferred from one part of the hardware to another.

2. Address bus. This carries the address of the location to which the data in the data bus should be delivered. The address and the data travel in tandem to each of the locations in turn until the component being visited recognises its address, it then grabs the data that is being transported.

3. Control bus. In section 1.4.a it was stated that one of the components of the processor was the control unit, whose job is to control and coordinate the operations of the rest of the processor. It can only do this if it can send commands to the different components of the system. The control bus is used to carry these control signals.

1.4.d *Describe the connectivity of devices*

Candidates should be aware that there are a number of ways of connecting different devices/components together. To start with there are physical and non-physical means of connection.

1. Physical means

Basically, connecting two devices using a cable. This may be:

Simple copper cable, normally arranged in pairs with an earth wire. This may be in the form of coaxial cable or may consist of a pair of wires twisted together to take the signals (twisted pair or TP cable). A reliable though very restricting form of communication medium because of the need to have a physical presence for the cable and the speed of data transmission is limited. However, communications are secure as long as the owner has control of the cable. There are many different standards of cable categorised by labelling them as one of the types CAT1 to CAT7. The differences are not important to us, save the fact that the different categories have different characteristics of rate of transmission of data, distance over which the signal quality remains reliable and the cost of the cable. As with all things, the choice of cable is normally based around a compromise between performance and cost.

Fibre-optic cable has all the benefits and drawbacks of copper cables except that rates of data transmission are far greater. Communication is by use of light beams sent down a fine glass filament. Many of these filaments can be wrapped together to provide many channels for information.

2. Non-Physical means

Wireless communication is more popular now because it removes the restrictions placed on location of devices because of the position of the cable. Wireless communication can be used anywhere within the wireless footprint. This has tremendous advantages like a student being able to pick up communications on a lap top anywhere in a school, but also has two major disadvantages. The first is that control over who can access the data sent is very difficult: if the student can access data anywhere in the school, so can a person who does not have access rights, so security is that much harder and needs to be thought of very seriously. The second problem is an, as yet, unproven worry that the high intensity radio signals used may cause health problems.

When thinking about wireless communication think of the different forms it may take: radio communication to blanket an area; microwave or infra red communication which overcome problems of interception because the signals travel in straight lines but are once again limited to the locations between which data can be sent; satellite communication is often used over long distances, the signals being sent by using lasers or some other means of a narrow beam of data.

One measure of the effectiveness of the means of communication is the rate at which data can be sent along the medium. Note that this is not the speed, but the rate: the volume of data in a given time period. This is measured in bits per second and is known as the baud rate.

One way of increasing the volume of data is to send more than one set of data at the same time. The different sets of data do not get mixed up because they use the same principal as radio transmissions. Each set of data is sent at a different frequency, just like radio stations are sent on different frequencies. The radio stations don't get mixed up and neither do the signals on the communication medium that the computer uses. However, there is a limit to the number of different signals that can be sent at the same time before they do interfere with each other. This is known as the bandwidth of the medium. The higher the bandwidth the more sets of data can be sent simultaneously. Copper cable has a low band width while wireless communications have a higher bandwidth and fibre optics potentially have a very high bandwidth.

1.4.e *Describe the differences between types of primary memory and explain their uses*

The first question that needs to be answered is “What is primary Memory?”

In section 1.4.a the concept of the processor needing to store data of all types in the memory was introduced. However, as those who have used a computer will know, there are plenty of other places that a computer can store data or programs e.g. a disk drive or a CDROM. These will be discussed in section 1.4.f. In 1.4.a it was stated that any data or program instructions that the processor was to follow had to be in the processor memory. Anything that is being held elsewhere, like on the surface of a disk drive, cannot be used by the processor until it is moved into the processor’s memory. Because it is so important it is called the primary memory. The primary memory of the computer is the memory that is advertised as being “1 Gbyte of RAM”. This simply describes the capacity of the memory.

There are different types of memory in a processor:

1. ROM (Read Only Memory)

ROM is memory that cannot be altered. This means that even switching the computer off will not affect the contents of the ROM. There is very little that needs to be stored in ROM so it tends to be very small. In the past the whole operating system was stored on ROM, but that means that if you want to use your computer in a different way or you want to install the latest edition of an operating system to replace the one in use, you can’t because the computer can’t delete the old one. When the computer is turned on it would be there. For this reason the operating system is stored on the hard drive of a computer along with all the other programs that may be used. A problem arises because when the computer is switched on it needs the operating system to be able to do anything useful, so a small program is stored on ROM whose job it is to go and get the operating system from the hard drive so that the computer can work. This little program is called the bootstrap, and the process of retrieving the operating system is known as booting.

Some processors are used in applications where the processing follows a standard pattern all the time. Examples would be the processor controlling a washing machine or a burglar alarm system. The program instructions are standard each time it is used. Also, it is important that the user cannot alter the instructions, consequently, it is sensible to store them on ROM so that they cannot be interfered with. In this case the ROM is sizable compared to the rest of the primary memory and the processor is said to be embedded.

2. RAM (Random Access Memory)

RAM stores the programs that are being used by the computer (including the operating system) and the data that is being used with those programs. When the computer is switched off all this data is lost. RAM is said to be volatile memory because it is so easily changed, whereas ROM is non-volatile because it cannot change.

Notes

1. There are many other types of primary memory that you may have heard of (PROM, EPROM, EAPROM, SRAM, DRAM...). They are all a type of either RAM or ROM and you have no need to learn the details about any of them.

2. A typical question will ask for an example of what is stored in ROM and RAM. The safest answers are the bootstrap being stored in ROM and user software and data being stored in RAM. Problems arise if a student answers that the operating system is stored in ROM because it can be, but in most micro systems it isn’t for the reasons given above. The BIOS (if you don’t know what that is, don’t worry because there is no reason why you should at the moment) is another problem. The BIOS itself includes user defined parameters and hence is not stored in ROM entirely. In fact the BIOS tends to be stored in a special type of

RAM which is refreshed using battery power when the system is switched off. This is past the level of this course and students are advised not to use this as an example of storage in ROM.

1.4.f *Describe the basic features, advantages, disadvantages and uses of secondary storage media*

Primary memory is memory within the processor. It is here that the computer stores data that are in current use because the control unit does not have direct access to data that is stored anywhere outside the processor. However, the storage that is available in the processor is limited in size and volatile. What is needed is something that is less temporary in nature and that does not have the same restrictions as far as size is concerned. This will be memory outside the processor. It is called secondary storage.

There are a number of different types of secondary storage that can be categorised according to

- Means by which the data is stored, optically, magnetically or on solid state devices
- The technique used for storage of the data, sequential storage or direct access storage
- The capacity of the medium, how much can be stored on it
- Portability of the medium, can it be moved around easily
- Access times to the data stored.

1. Magnetic tape

Magnetic tape is still widely used, particularly for long term storage of archive material and for back up copies of large files. The big disadvantage of tape is that the access to the data stored is, by necessity, sequential which makes it largely unsuitable for most data handling applications. However, large volumes of data can be held and the medium is freely transportable. It is these qualities which make it valuable for producing and storing back up and archive copies of files stored on a computer system.

2. Magnetic hard disk

Data is stored using small areas on the surface of the disk which are magnetised in one of two states, one standing for 0 and the other for 1. The hard disk is made of a rigid material and the read/write heads skim very near to the surface, it is contained in a sealed unit. The storage density can be very great, consequently the amount that can be stored on a single surface is very great. This means that it is important to have strategies to be able to find material on the surface of the disk. The surface is divided up into small areas in order to make storage and searching easier to do and so that access times are much faster. The process of dividing the surface of a disk up into more manageable areas is called formatting. It is carried out by a format program which is another example of a utility program others of which were described in section 1.2.v. The hard drive is likely to be the main secondary storage for a computer system, having very large storage capacities. Although they can be portable, it is more likely that they will be fixed to the chassis of the machine because of the tolerances that they have to work to.

3. CD ROM

A CD ROM is different from the storage devices so far mentioned because it is not magnetic. A CD ROM is an optical storage device, using the reflection of a laser off a pitted surface to store information. Large quantities of data can be stored on the surface and it is completely portable from one machine to another. Most computer systems can be relied on to have a CD drive, which makes the data stored truly portable between machines and a CD ROM is not alterable by the user. For these reasons, manufacturers have tended to use CD ROMs to produce software and large data files like encyclopaedias, although it is more likely that such data rich systems will now be produced on DVD.

4. CD RW

The same basic principles apply to a CD RW as to a CD ROM except that the contents can be both erased and rewritten. This makes them particularly useful for transporting files and also for backing up and archiving files from smaller systems.

5. DVD ROM and RW

DVDs have the same properties as CDs but they can hold larger amounts of information.

6. Solid State Technology

This includes things like USB sticks/wands/cards. The storage is done on a circuit which reacts in the same way as primary memory and hence provides very fast access to the data. The device is fully portable and very small. They are often kept on key rings and are activated by simply plugging them into the computer's USB port.

7. Others

There are other types of secondary storage and other forms become available regularly. Candidates should try to keep up to date with types of storage that are in common usage as the specification does not mention any by name.

Speed of access to data.

Notice that there are no figures quoted for access times. There is little point in doing this because access times vary according to the manufacturer and the point of development so far reached. If the author were to quote a number, not only would it be wrong as far as this work is concerned because there is such a wide range, but it would certainly be out of date by the time you read it. In general terms tape storage is the slowest access because of the way that the data is stored sequentially. Access times from CD and DVD and hard drives are faster, but because there is such a range of both types of drive available, it is impossible to say that one is faster than the other. Solid state devices provide the fastest access times because there is no mechanical motion, no moving parts.

Capacity

Again, much depends on the type of drive or size of medium being used. Tapes come in different sizes, but tend to be comparable in capacity with smaller hard drives. Of the different types of disk a CD stores enough information to hold about 70 minutes of music while a DVD holds about 4 hours of visual material. A solid state device can hold enough music so that you could listen to it for a year and not hear the same music twice while a large hard disk will hold the music library for a radio station.

Uses

Obviously, no list of uses will be complete. However, there are some obvious uses for each storage type which take into account the advantages and disadvantages of each.

Tape is used for making backups of files held on computer systems. This makes use of the fact that it can store a large amount of data, but the disadvantage of the access being slow does not matter because it is rare that a backup file would be used anyway.

A solid state device has the advantage that it can be written onto and taken away from the computer. Because of this it can be used for storing personal confidential files. Add to this the fact that all computers can be relied on to have a USB port and it becomes a sensible way of transferring information from one machine to another.

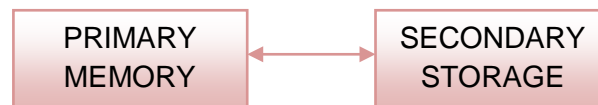
A hard drive has the advantages of being fast to access and also stores massive amounts of data. These advantages mean that it is going to be used for storing software and user files. The disadvantage of being attached to one machine is not important if the same users always use the same machines, some other method of sharing will have to be devised if this is not true.

A CDROM or DVD ROM cannot be altered (unless it is a special re-writable). This disadvantage can be turned into an advantage if the owner does not want the contents of the storage to be altered. Examples of files stored on these media are software for import to a system and large reference files like encyclopaedias.

1.4.g *Describe the transfer of data between different devices and primary memory, including the use of buffers and interrupts*

There are two places where data can be stored in a computer system, the primary memory and secondary storage. Sometimes it is necessary to transfer data from the primary memory to secondary storage, perhaps because the computer is going to be switched off in which case the data in primary memory would be lost. Sometimes it is necessary to transfer data from secondary storage to primary memory, perhaps because the processor wants to use some data held in the secondary storage and, consequently, needs to move the data into the primary memory first. Whichever direction the data is to be transferred the method of transfer must be planned.

Normally the data would be transferred through a wire connecting the two storage areas together. The type of wire and the different rules for data transfer are explained in section 1.5.c, sufficient for the moment is to have a picture of data travelling freely in both directions.



Unfortunately, we know that things can't be that simple. We know that primary memory is part of the processor, and that anything that goes in or out of the processor must go through the ALU, so the diagram should look like this



This causes a problem. The primary memory operates at great speed because it is part of the processor, while the secondary storage is probably some sort of disk or tape which (while it seems incredibly quick to us) is very slow at reading or writing the data, in comparison. This means that the processor should be able to get on with something else because the secondary storage is so slow. Unfortunately, it can't because it needs the ALU that is being used for the transfer of the data. This problem is overcome by the use of a buffer.

Buffers

The problem is caused because the secondary storage device is so slow compared to the processor, the solution is to put a small amount of fast memory in to the system between the ALU and the secondary storage device.

This means that the processor can send data very quickly (or receive it), and then get on with something else while the storage device takes its time in reading (or sending) the data. This small amount of memory between the two parts of the system is called a buffer.

Stage One: Filling the buffer from the processor



Stage Two: Emptying the buffer to storage



Interrupts

This system is fine if the buffer can hold all the data that needs to be sent to the storage device. However, if there is more data than will fit in the buffer, the storage device has to be able to tell the processor (actually the control unit) that it has used up all the data in the buffer and that it is ready for the next lot. This forces the processor to stop what it is doing and fill the buffer up with more data. Because this message from the secondary device makes the processor stop what it is doing it is called an interrupt. However, the processor is really in control of the system and cannot be ordered about without having some say in it. What really happens is that the interrupt arrives and the processor stores it and carries on with what it is doing until the thing which the interrupt wants to be done is the most important thing that the processor needs to do. It is only then that the processor will stop what it was doing and carry out the interrupt. The whole procedure is then repeated.

Notes

A buffer is a small amount of fast memory outside the processor that allows the processor to get on with other work instead of being held up by the secondary device.

An interrupt is the message which the secondary device can send to the processor which tells the processor that all the data in the buffer has been used up and that it wants some more.

The example used here was communication between primary and secondary storage, but the same is true for any communication between the processor and a peripheral device, e.g. a printer or a keyboard.

The system is not really quite as simple as suggested here, but further complications can wait until later in the course.

1.4.h *Describe a range of common peripheral devices in terms of their features, advantages, disadvantages, and uses*

Any hardware device that is part of the computer system but is not part of the processor itself is called a peripheral device.

Peripheral devices can be categorised under four headings:

1. Communication devices

These devices allow for communication between machines and will be covered in detail in section 1.5.b.

2. Storage devices

These devices, which provide for secondary storage in a computer system, have been described in section 1.4.f.

3. Input devices

There are too many different input devices to expect familiarity with all of them. However, by inclusion in the syllabus there are some that must be understood. While this list is not exhaustive, specific questions will not be asked about other input devices with the following exceptions. There are some devices that are so common that any candidate taking an exam in computing can be expected to have experience of using them, e.g. mouse, keyboard. Additionally, candidates should be encouraged to learn about a variety of contemporary devices so that they can use them when answering questions that ask for a hardware configuration for a specific application.

a) Keyboard. A standard keyboard uses keys that stand for the different characters that the computer recognises in its character set. Most keyboards contain the letters of the alphabet, but not all do, for instance most calculator keyboards are very different, as are the keyboards for use at cash machines. Indeed, those used at cash machines have the added sophistication that the different keys stand for different characters/values at different times through the application. The characters needed for specialist use machines are determined by the use to which the machine is to be put. Keyboards are the most common form of input device to a system because they are universally available and understood.

The common keyboard is known as the QWERTY keyboard because those are the first six characters on the top line. The design is not very good for two reasons. First, the arrangement of characters comes from the original typewriter whose keyboard was arranged to be the most difficult to use in order to slow typists down so that they did not jam the mechanism of the old machines. The second is that the keyboard itself is difficult to use comfortably because of the way that the keys are arranged in rigid rows, making it awkward to keep your arms comfortable while using it. The first problem is very difficult to solve because of all the experienced operators that can use the present keyboard so well. Retraining them to use a different arrangement of keys would not be feasible. Various attempts have been made to address the second problem by arranging the keys in curves that fit the palm of the hand rather than in straight lines. These are called natural or ergonomic keyboards.

One problem with normal keyboards is that they are particularly prone to damage from dirt or liquids because of the gaps between the keys. A different type of keyboard, where the keys are simply part of a continuous surface which has areas on it, which are sensitive to pressure, can overcome this problem. Called touch-sensitive keyboards, or concept keyboards, they are ideal for use outside because rain will not damage them like a normal keyboard.

A type of keyboard not yet mentioned is a musical keyboard. Normally arranged like a piano keyboard these need a special piece of hardware to allow them to work properly, known as a MIDI (musical

instrument digital interface) this connects the musical keyboard to the processor and allows data to be passed between the instrument and the processor.

b) Mouse. A mouse is a device designed to be used with a pointer on the screen. It is particularly useful because it mimics the natural human reaction of being able to point at something. A mouse is really two input devices in one. One is the movement around the screen created by actually moving the mouse in a physical way, and the other is the ability to select which is done by using a switch (the mouse buttons). Variations have been developed which use the same basic principles but are designed for particular applications. An example is the tracker ball used in many laptop computers. This is like an upside down mouse where the user moves the ball directly rather than moving an object around a flat surface. This is necessary because when using a laptop there may not be a flat surface available. A mouse is a special form of input device because it uses both analogue and digital inputs. The analogue input of the movement of the ball and the digital input from the switch.

c) Barcode readers. A barcode reader is a laser scanner that reads the reflected laser light from a series of dark and light coloured lines of varying thickness. The different widths of pairs of lines make up a code that can be converted into a number. This number can then be used as the key field relating to a file of items that have been bar coded. The details of the contents of the barcodes are not of importance to us in this section, except to say that barcodes can easily be misread by the system, so one of the digits in the number is used to check that the rest of the code has been read properly. This digit is called the check digit, and has been explained in detail in section 1.3.f. Barcodes are particularly useful because they do not rely on human beings to input the data, although, if the barcode is damaged so that the laser scanner cannot read it properly, the digits represented by the code are printed underneath so that they can be input by a user at a keyboard. Barcodes are used where the data does not change, and so can be printed on original packaging. Also see section 1.3.e.

d) MICR (magnetic ink character reader). This is a device that reads characters that are printed on an original document at the time of it being created. The characters are printed using magnetic ink. The value is that the characters are readable by humans and by machines. The only common use for such characters is the data printed on the bottom of cheques containing account identification. Also see section 1.3.e.

e) OCR (optical character reader). This is a device that reads characters and can distinguish between the different characters in a given character set. It works by comparing the shape of a scanned character with a library of shapes that it is intended that it should recognise. OCR tends to be an unreliable form of input and works more effectively when it is restricted to having to recognise a standard character set produced by printing rather than by using hand writing. OCR is used for reading post codes on printed documents and also for reading documents for blind people, the contents of which can be output using a voice synthesizer. Also see section 1.3.e.

f) OMR (optical mark reader). This device can recognise the presence of a mark on a sheet of paper. The position of the mark conveys information to the machine. For example a school register may consist of a list of names of pupils in a class together with two columns of small rectangles, one for present and one for absent. The same action (shading in a rectangle) stands for both being present and being absent. The difference is the position that the mark occupies on the paper. Printing in the sensitive areas of the sheet is done using a special type of ink which the optical scanner does not see, that is why OMR documents tend to be printed in a light blue or pink colour. The other standard use for OMR documents is as multi choice examination answer sheets. Also see section 1.3.e.

The big advantage of both OCR and OMR is that data can be input to a computer system without having to be transcribed first, thereby cutting down the number of errors on data input.

g) Scanners. A scanner is a device that converts a document into a series of pixels (picture elements – these are small squares that, when put together, form a picture). The larger the number of pixels, or conversely the smaller each individual pixel, the better the definition of the final picture. There are different types of scanner, but all use the same principle to create the image. A typical use for a scanner would be to input a picture of a house so that it could be included with the details of a house that is for sale in an estate agent's publication.

A scanner is an input device, not to be confused with a plotter which is an output device.

h) Graphics Tablet. A graphics tablet is a flat surface on which a piece of paper is placed. The user can then draw on the paper and the tablet will sense where the pencil is pointing and transfer the line to the screen.

i) Microphones. Used to input sound to a computer system.

4. Output Devices

There are too many output devices to be able to write notes on all of them. Again, the same thing is true about output as is true about input, that it is important to know about those devices stated in the syllabus and also a range of devices that will allow for sensible decisions about peripheral devices to be made for a given scenario in a question.

a) Screens. Monitor screens are categorised according to the obvious colour/monochrome, also according to the number of pixels that there are on the screen. The more pixels there are, the better the picture will be, this is known as the screen resolution. This is being typed using a very low resolution, monochrome screen. If you consider the contents, there is no reason for any further sophistication to be necessary. However, a computer system running a modern game program will need colour and many more pixels in order to produce a satisfactory picture. The more pixels that there are on the screen, the higher the resolution is said to be.

A particular type of screen, called a touch screen, acts as both an input device and an output device. Information is output by the system onto the screen and the user is invited to answer questions or make choices by pointing at a particular area of the screen. The device can sense where the user is pointing and can report the position to the processor. The processor can then deduce what the user's reply was according to the position that was pointed to. Touch screens are particularly useful in areas where keyboards are not appropriate, e.g. where the device may suffer from vandalism. They are also useful for users who would find difficulty using other input devices, e.g. very young children who want to be able to draw on a screen.

b) Printers. A printer is a device which provides the user with an output from the system which is permanent. This output is known as hard copy, so a printer is a device which produces hard copy. There are many different types of printer and the student should be aware of a number of them, their uses, advantages and disadvantages. However, there is no need to understand how they work.

The first type is a dot matrix printer. These tend to be slow, and the output is particularly poor quality. The big advantage is that the output is produced by using pins to strike at the surface of the paper. Because of the physical nature of the way that the printout is produced, it is possible to obtain multiple copies by using carbon paper or self carbonating paper. A good example of this is the receipt that a shopper is presented with if buying something using a credit card, there are two copies produced, back to back, one for the shop to keep and one for the buyer to take away with them. Dot matrix printers are noisy and uncommon now but still exist for the specialised use of producing multiple copies.

Ink jet printers, which produce output by spraying ink on to the paper could not produce the two copies that the dot matrix can, but it can produce much better quality and in colour, at low cost. This makes ink jet printers ideal for home use.

Laser printers can produce very high quality work at high speed. The cost is more than with the other types but used where it is necessary to give a good impression, for instance sending letters from a solicitor's office to clients. These are the most common type of printer, capable of quality and speed of copy at a relatively low price.

Plotters are a type of printer designed for drawing lines and geometric designs rather than for producing characters. The image is created by pens being moved across a piece of paper, under the command of the processor. Plotters tend to be used for drawing blueprints, perhaps in an architect's office to produce detailed drawings of buildings for builders to follow.

c) Speakers. Used to output sound from a computer system.

Sensors and actuators are used for automatic input and control. Sensors have been discussed in section 1.3.e. Actuators are devices that can be controlled by a computer processor and have the ability to transform the decisions of the computer into physical decisions like turning valves on and off.

There are many other peripheral devices and, as has been mentioned, a knowledge of some others will not come amiss, however that is enough to be able to answer questions in the exam. The questions will normally take the form of presenting a scenario and then asking for a description of the hardware required. The important thing to remember is how the marks will be awarded. There will not be a mark for every device mentioned, but the candidate will be expected to give sensible suggestions for each of the four areas of peripherals mentioned at the start of this section. In other words the mark will not be for a keyboard or a mouse, but for suggesting sensible methods of input to the system.

1.4.i *Describe and justify the appropriate peripheral hardware for a given application*

This section deals with the needs of the application rather than the peripheral devices themselves. For details of the devices the reader's attention is drawn to sections 1.3.e and 1.4.h..

When peripheral devices are chosen, a number of factors need to be taken into account.

- Who are the people that are going to use the application? Specifically, consideration must be given to their age (can they read is an obvious question), their ability with computer systems, their understanding of the software in use (do they need to be given instruction how to use it or do they know what is expected of them?), any physical disabilities which may make some hardware impossible to use.
- Is the system automated in any way? If so the peripherals may include sensors and actuators.
- Under what circumstances will the system be used? If the input and output devices are to be used in the open air then the environment will dictate some restrictions which would not be necessary if the system was being used in a computer room.
- The software that is being used. The software will dictate the type of input required and the type of output that will be produced, consequently it will also have an influence on the peripherals that are suitable for that input and output.

When answering questions drawn from this section, compare the requirements of the system with the characteristics of the available hardware devices, and be prepared to state why a particular choice has been made.

It is also important to think in terms of 4 parts to the answer: input, output, storage and communication. The last of which may not be necessary dependent on the application.

1.4 Example Questions



1. State **three** functions of the arithmetic logic unit. (3)
 - A. 1. Any arithmetic is carried out in the ALU. (This normally takes the form of addition, and the result is stored in the ALU temporarily before being returned to the memory)
 2. Logical decisions can be made by the processor. (The processor can use the ALU to decide whether one value is greater than another, or if one is negative, and then make decisions based on the answers)
 3. Communication with peripheral devices is carried out via the ALU. (All data which is to be used by the processor has to be stored in the memory. Any item of data which is bound for the memory must be stored temporarily in the ALU while the processor decides where it is going to be stored)

Notice: The question asked for three answers. The sensible response would seem to be to number them. The expected answers are the three single line responses. The contents of the brackets give extra information which would not be expected in this question. However, if the question had said 'Describe...' or 'Explain...' instead of 'State...', then the question would have been worth 6 or even 9 marks, and some extra detail, for instance the contents of the brackets, would have been necessary.

2. Explain the purpose of:
 - (i) the program counter
 - (ii) the current instruction registerin the functioning of the processor. (4)

- A. (i) -Stores the address/location in memory...
 - of the next instruction to be carried out
- (ii) -Stores the instruction itself...
 - while it is being carried out.

Notice: Difficult to make more than one mark out of either of these. It means that a fairer question is likely to be a single mark 'State...' question for each one. The problem is that candidates don't know about the fetch execute cycle yet and hence cannot be expected to link the functions of the registers together yet, just know that they are special and have a very basic idea as to what makes them special.

3. Describe **two** different types of bus used in a processor. (4)

A. -Data bus

-used to transmit data from component to component

-Address bus

-carries address of data destination

-contents must be synchronised with the correct contents of the data bus

-Control bus

-carries control commands from control unit to different parts of the processor

Notice: Again, there is very little that can be asked beyond the bare facts in the specification because the candidates' knowledge is strictly limited at this stage.

4. Explain what is meant by bandwidth and why it is important when deciding what type of connectivity is used. (4)

A. -Bandwidth is a measure of the number of different channels of data...

-that can be transmitted at any one time...

-without causing interference

-Each type of media has a different bandwidth

-Each type of media costs a different amount

-Decision needs to be made according to the volume of data that needs to be sent in a given period of time.

5. a) State **two** ways that RAM and ROM memory differ. (2)

b) Explain what types of data would be stored in each of RAM and ROM memory, giving reasons for your answers. (4)

A. a) -RAM is volatile (will lose its contents when the power is switched off), while ROM is not.

-The contents of RAM can be altered, whereas the data stored in ROM cannot be altered.

-ROM is normally smaller than RAM

b) -RAM would contain user files and software that is in current use.

-RAM is used for these files because they have to be alterable, and the user will want to use different software from time to time, implying that the original software will need to be erased so that it can be replaced. Losing these files when the machine is switched off is not a problem because they will have been saved on secondary storage.

-ROM is used to store the bootstrap program.

-ROM is used because when the computer is initially switched on there must be a program present that can be used to load up the rest of the operating system from secondary storage. This program must not be altered because without it the computer cannot start to work.

Notice: the way that the answers have been phrased in part (a). The question asked for a comparison of ROM and RAM so both need to be mentioned when you are giving an answer. To say that RAM is volatile is true, but does not answer the question until you say that ROM is not.

Part (b) is asking for reasons to be given. Notice that the reasons given related to the answers given in part (a). The part (b) answer is not, "user files because there is more space in RAM", although that could be right if phrased a little more carefully.

6. A student has a home computer system.

State what storage devices would be used on a home computer system and justify the need for each one. (8)

- A. -Hard disk...

-to store software and user files (e.g. word processor and essays)

-Memory stick...

-to enable work to be transported between school and home so that it can be continued in the evening

-CD ROM...

-to allow the import of new games to the PC.

-CD RW...

-to back up the files, so that important work is not lost if the hard disk crashes.

Notice: The question does not specify how many devices should be mentioned. However, each device has to be named and justified (2 marks) and there are 8 marks for the question, so four devices seems sensible. What happened to tape? After all that is the first secondary storage that was mentioned in the text. Tape is not sensible in this example, there is not a big enough volume of data to make a tape worthwhile, and the old cassette tape is an antique by now. Consequently, a tape device does not fit the application given.

Also notice the way that the justification for each of the devices was linked to the application, for a memory stick, the answer did not just say that it could be used to transport files, but it gave a good reason for wanting to do so in this case.

7. Describe how buffers and interrupts can assist in the transfer of data between primary memory and a secondary storage device. (4)

- A. I will describe transfer from primary memory to a secondary storage device.

-Buffer is an area of fast access storage..

-which can be filled by the processor and then emptied at slower speed by the secondary storage device...

-allowing the processor to continue with other tasks.

-When the secondary device has used the contents of the buffer...

-it needs to tell the processor that it requires more data...

-this is done by sending a message to the processor, called an interrupt.

Notice: The answer describes transfer of data from the processor to the peripheral. Transfer in the other direction is equally valid and gives rise to a similar answer. However, if the two answers are mixed together then it produces a confused, and often, wrong answer. This is why the first statement is important as it has set the parameters by which the question will be answered. You don't have to say this in your answer, but it makes clear what is being done.

Also, there are more mark points than there are marks available. Hopefully, you are getting used to this because it is generally true in most questions. In this question there are obviously two points for interrupts and two for buffers, but in order to get the marks you must make sure that the definition appears somewhere. So, for the interrupt, you need to earn the last mark point, otherwise you haven't defined what an interrupt is.

8. A department store decides to place a computer system by the main entrance to the store so that customers can find out whereabouts in the store items are available. The different departments remain in the same places, but the articles available in each department change on a regular basis. State a sensible hardware design for such a computer system, giving reasons for your choices of hardware. (6)

A. Input:

-Touch screen

-Some protection against vandalism/restricts user access to contents of menu system displayed/simple to use because it uses human reaction of pointing so no training necessary

Output:

-(Touch screen)/printer

-Touch screen outputs choices for user to select from/printer available for producing hardcopy as a permanent record for the customer

Storage:

-Hard drive/CDROM

-Hard drive necessary to store details of the products on sale because of the large number of changes that occur in this file. CDROM used to keep the store plan and the location of the departments as these do not change.

Notice: The large number of possible mark points. Also the way that the marks are split up. In this type of question you can't expect a lot of marks for simply writing down a long list of peripheral devices, the marks are split up according to the four types of peripheral mentioned in section 1.4.h. In reality, this question would probably have talked about such systems being available throughout the store, but this would mean that they would be able to communicate with one another. As communication comes in the next section it was left out of this example question, but such a question would normally have 8 marks, 2 marks for each of the types of peripheral.

9. A company has a workforce of around 2000. Some work in the office using the computer system for administrative tasks, while others use the computer system on the production line for giving details of orders that need to be manufactured.

Select appropriate peripheral hardware for these two application areas, giving reasons for your choices. (12)

A. Office:

- Keyboard/mouse for input because...
- input is likely to be character based owing to typical office tasks.
- Monitor/laser printer for output because...
- output needs to be checked and then high quality printout for documents to be sent outside firm.
- Hard drive for storage because...
- of need to store and retrieve documents out of sequence.

Communication:

- Some form of external communication in order to pay workers through banks/electronic communications with suppliers
- Using something like broadband/modem/...

Factory:

- Touch screen for input because...
- dirty environment could damage other forms of input device.
- Plotter for output in order to...
- produce design quality drawings for use in manufacturing process.
- Hard drive for storage in order to...
- store number of high quality drawings.

Communication:

- Necessary for communication, either between machines or to some central resource/LAN...
- requiring cabling/network cards.

Notes: The standard form of the hardware question requiring answers for input/output/storage/ and communication where appropriate. In reality this is too big a question to be asked in an examination, the normal question will be out of 3,4,6 or 8 dependent upon whether the question says state or describe and whether communications play a sensible part in the application.

1.5 Data Transmission



1.5.a *Describe the characteristics of a LAN and a WAN*

All the systems that have been mentioned so far have been individual computers, sometimes with more than one user, but single processors. This means that the systems we have discussed so far are not connected to other machines.

Imagine a classroom with 20 computers in it. Every time the lesson ends you would need to store your files on secondary storage. It would be possible to store the files on a memory stick and take them away with you, but the likelihood is that the files will be stored on the hard disk. This means that the next time you want to use those files you need to sit at the same computer. It would be much more sensible to have a system that allowed access to the same files through any of the 20 computers. To allow this the computers need to be connected up to each other. When computers are connected together to share files they make a network.

A network of 20 computers in a school classroom is obviously on a small scale, not because 20 is a small number, but because the communication is made easier because of the short distances involved. If a business with head offices in London and factories in Leeds and Manchester wanted to connect the computers on the three sites up there is an obvious problem of distances to be overcome. Generally, networks over small distances are called Local Area Networks (LAN) while those over great distances are Wide Area Networks (WAN). The school will have control over the medium used (probably wireless or cable) to link the machines together whereas the business will probably use the telephone system or some other communications system on which it can rent time. A third difference is that it would be very difficult to break into the files on a LAN because it is all contained on one site whereas the data transferred on the WAN is far more subject to interception.

Whether the network is a WAN or a LAN it will have the advantage of offering the users the chance to communicate with one another, to share information centrally, to share copies of software and to allow multiple access to files of data. In a LAN there is the added benefit of being able to share hardware, so the classroom with 20 computers may only have 3 printers. Also those printers may be different types that could be used for different tasks. This means that the type of printer used is dependent on the job that the user wants it to do rather than the type of printer that happens to be connected to the computer on which the work is being done.

To summarise:

- Computers can be linked together to form networks
- If the distances are short the network is called a LAN, if longer the network is a WAN
- Networks allow computers to communicate
- Networks allow the sharing of both hardware and software.

1.5.b *Show an understanding of the hardware and software needed for a LAN and for accessing a WAN*

Network Basics

In order to connect computers in a network there are a number of essential things necessary:

- Each computer needs to be able to communicate with others. In order to do this there needs to be a special piece of hardware attached to the processor, called a network card. It is through this network card that the computer can communicate.
- The computers need to be connected in some way (also see section 1.4.d). The standard method in schools is to use a cable although wireless communication is becoming more popular despite problems with security and possible health risks. The cable is likely to be coaxial, like a television aerial. The cable has a limitation in that the signal gradually deteriorates as it is sent down the cable which means that the maximum length of the cable is about 300 metres. This maximum depends on a number of things, not least the quality of the cable used, but all cable is ultimately going to be limited in length. The same limitations apply to wireless communication in such environments, though the restriction on movements is not the same with a wireless system as if all the components of the network need to be hard wired.
- With a number of computers attached to the network, communications are going to get complicated. There is a need for something to control the signals being transmitted. This job is often carried out by a computer whose job is to control the network, it is known as a server or a network server. One of the jobs of the server is to control access to the files held on a hard disk, because of this it is sometimes known as the file server. Some networks have a number of different servers controlling different parts of the system, for example, there may be a printer server to control use of the printers. On many LANs these server functions are all carried out by the single network server. Some, very basic, networks do not have a server, all the computers enjoying equal status. These are known as peer to peer networks but they lack many of the features which would be expected on a normal network, like data security measures.
- The communications around such a system are obviously quite difficult to control. It is necessary to have a set of instructions that the network must follow, this is known as network software or a network operating system.

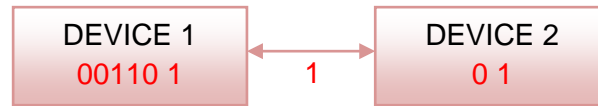
A wide area network differs because the distances involved tend to be far greater than for a LAN.

Consequently it is not possible to connect all the computers by a cable. Actually the cables already exist to do this, the telephone system. So to connect two computers over a long distance, they can simply ring each other up! Unfortunately, the type of signals produced by a computer are different to the type of signal that can be sent down a telephone line. In order to send a computer signal down a telephone line it must be altered first. This is done by another piece of hardware called a modem. A similar modem must be present at the other end of the line in order to turn the signal back again. The medium itself will probably be fibre optic cable providing a broadband service. These terms have all been mentioned before in section 1.4.d.

1.5.c Describe the different types of data transmission

Serial and Parallel transmission of data

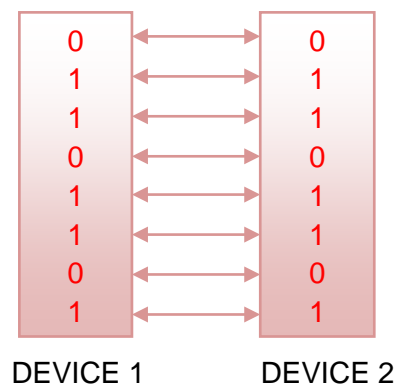
Data needs to be transmitted between devices in a computer system. The easy way to do this is by using a single wire that links the two devices. Data is transmitted in the form of bits down the wire, so an 8 bit byte that stands for a single character, will be transmitted in 8 parts, one signal for each bit.



The diagram shows the data byte 01101101 being transmitted. As there is only one wire, only one bit can be transmitted at a time. This is known as SERIAL transmission of data.

Serial transmission has the advantages of being simple and reliable because the next bit is not transmitted until the current one has arrived at its destination so they cannot get mixed up. However, because only one bit can be transmitted at a time, the speed of transmission is slow.

If the devices are connected by more than one wire, then more bits can be sent at once. A sensible number of wires would be 8, because then a whole byte can be sent at once instead of having to send one bit at a time.



This type of data transfer is called PARALLEL transmission. Parallel transmission of data is obviously faster than serial because all the bits are travelling at the same time, but because of the fine tolerances in the transmission, it is less reliable as the bits can become muddled up.

Modes of Transmission

If data is to be transmitted between devices there are three different modes of transmission possible.

a) Simplex mode

If data can only travel in one direction then it is known as a SIMPLEX transmission. A good example of a simplex transmission of data is teletext information which is passed to a television receiver, but there is no way to send data in the other direction.



b) Duplex mode

When data can pass in both directions at the same time it is known as DUPLEX transmission of data. An example of duplex transmission would be a telephone conversation as both users can speak and be heard at the same time.



c) Half duplex mode

When data can pass in both directions, but only in one direction at a time, the transmission is called HALF DUPLEX. An example of half duplex is a CB radio system in which each handset can either be set to receive mode or send mode.



1.5.d *Explain the relationship between bit rates and the time sensitivity of the information*

Information needs to be sent to devices in a computer system from other devices. For example, a picture stored in the memory of the computer needs to be sent across a network to another computer. Remember that the more pixels that there are and the more colours that can be represented, the better the picture will be. However there is a limit to the amount of information that can be transmitted in a given time across the network, so a decision needs to be made. One choice is to have as much detail as possible and accept that it will take a relatively long time to transmit it. The other is to limit the resolution of the picture, meaning that there is less information to send so that the message will be sent more quickly.

The number of bits that can be sent in one second is known as the BIT RATE. The units used to measure the bit rate are BAUD. 1 baud = 1 bit per second.

Note that text can be sent much more quickly than other forms of information because it needs far fewer bits (1 byte per character) than other types of data. When data other than text is being transmitted, e.g. on the internet, it is important to limit the amount of data that needs to be sent or the time it takes to download the data can be unreasonably long. The data can be limited by such simple things as reducing the size of pictures so that they only take up a small part of the screen, or that they are restricted to a few colours. Speeding up the transmission of the information by reducing the amount of data that is sent is known as compression.

The fact that it might take a noticeable amount of time to send a picture or other information from one place to another does not matter if the user waiting for it does not mind, but what about a video? This does not matter either. If it takes an hour to download a 5 minute video it does not matter because it can be stored as it is being received and then watched properly when it has all been down loaded. The problem arises when the video needs to be watched as it is being down loaded. If this is to be done, one of two things will be true: Either the video will be very difficult, and unrealistic, to watch because the frames will not follow on from one another easily and the video will look very jumpy or the bit rate of the communication medium will be large enough to send the data quickly enough so that the data can be used in real time.

It is this ability to use the data in real time that we mean by the phrase time sensitivity. If the use of data is affected by the time that it is used it is time sensitive.

1.5.e *Recognise that errors can occur in data transmission and explain methods of detecting and correcting these errors*

When data, of whatever type, is being transmitted from one part of a computer system to another, it is transmitted as a series of binary digits. Any data that is transmitted is going to be made up of a very large number of bits. Consequently, there are bound to be occasions when the data is not transmitted correctly. Luckily there are only two possible mistakes that can occur, either a 1 is received as a 0, or a 0 is received as a 1. Mistakes occur rarely, but when they do occur they can be very serious, as the information is no longer correct. This makes it very important that there should be methods for checking the data when it has been transmitted.

a) Echoing Back

The simplest way of checking the transfer of the data is to send it back again. If the data that is sent back is the same as the data that was sent in the first place then the original data must have reached its destination correctly, if not then it needs to be sent again. This is known as ECHOING BACK. Echoing back is very effective, but suffers from having to send data twice, thus taking longer than necessary, and needing to be a duplex, or half duplex, system to allow data transfer in both directions.

b) Parity

All data is transmitted as bits (0s and 1s). The Number of 1s in a byte must always be either an odd number or an even number. If two devices that are communicating data decide that there will always be an odd number of 1s, then if a byte is received that has an even number of 1s, an error must have occurred. E.g. the byte 01011000 has 3 ones in it. 3 is an odd number, so it fits the rule that it must have an odd number of ones. When it is sent there is an error in transmission so that the first bit is received as a one. So, the byte received is 11011000. This has 4 ones in it, which is an even number, so there must be an error. The receiving device would ask for it to be sent again.

Notes:

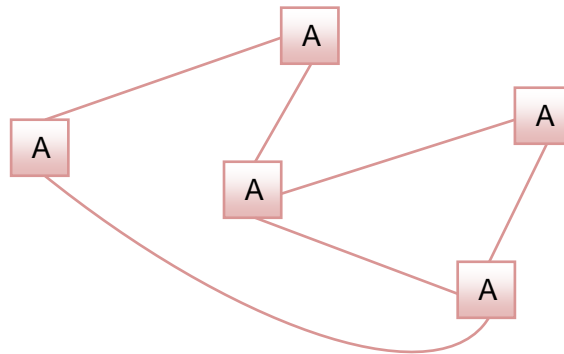
- If two mistakes are made in the same byte they will cancel each other out and the faulty data will be accepted. This problem can be overcome, and in the same way, a clever way of correcting error mistakes can be implemented. This method is not part of this course.
- Earlier in this course it was said that a byte was the number of bits necessary to hold a character code. Specifically, an ASCII character uses 8 bits in a byte, giving 256 different characters. This is not true because one of the bits has to be reserved for a parity bit, the bit that can change to make sure that the number of ones is always odd. This means that there are 128 different characters possible.
- The implication in all the above work is that odd parity is always used. Even parity can equally well be used, whichever has been agreed between the two devices.
- Parity is not only used during data transfer between devices, but also when data is transferred between different parts of the CPU.
- It is important to remember that parity does not depend on odd or even numbers but is an odd or even number of 1's in a byte.

c) Check Sum

Data will normally be sent from one place to another as a block of bytes rather than as individual bytes. The computer can add numbers together without any trouble, so another checking procedure is to add all the bytes together that are being sent in the block of data. This can be done whatever the bytes are representing because they are always binary numbers and so can be added together. The carry, out of the byte, is not taken into account, so the answer is an 8 bit number, just like the bytes. This answer is calculated before the data is sent, and then calculated again when it is received, and if there are no errors in the transmission, the two answers will match.

1.5.f and g *Describe packet switching and circuit switching explaining the difference in their use*

When a message is being sent from one machine to another, particularly over a wide area network, the message may have to pass through other machines first. This may be forced on the system because there is no direct route from one machine to another.



In the network shown, it would be easy to send a message from A to D or from A to B because A is directly connected to both of them. However, sending a message from A to C is much more difficult because there is no direct route. There are two ways that the message can be sent

a) Packet switching

The message is split into a number of equal sized packets. Each packet has a label saying where it is meant to be going and what number packet it is. These packets are sent along communication lines towards the destination. Each time a packet reaches a node on the network the node decides which direction to send it on. So, one packet in the message from A reaches node D. The obvious route to take is the one directly to C, but it is already in use for another message, so D decides to send it to E instead. The next packet arrives at D and, this time, the line to C is free, so the packet is sent direct to C. When the message has all arrived at C it has to be reassembled in the correct order.

b) Circuit switching

Before the message is sent, the network reserves a route from A to C. The message can then be sent directly from A to C. It will still be divided into packets for the purposes of transmission but the packets will be sent in order and hence will arrive in order and so will not need to be reordered when it gets there although the packets will need to be reassembled to produce the full message.

Packet switching allows optimum use of the connections around the network because as many routes are in use at one time as possible, whereas circuit switching means that the whole message is kept together so it does not need to be reordered at the destination.

Packet switching also allows for greater security of the messages that are being sent because the different parts of the message are being sent in different directions along different routes and hence they are far more difficult to intercept. Although circuit switching means that the message is kept together it also means that a large proportion of the network will be reserved for a period of time hence hindering messaging across the network.

1.5.h *Define the term protocol and explain the importance of a protocol to the transmission of data*

When data is being transferred from one place to another in a computer system there must be rules set up as to how the transfer is going to be done. Typical rules would be

- A rule about the wire connecting the two parts of the system. Errors would occur if one device had a serial connection and the other was expecting a parallel connection
- If one device sent data at a particular bit rate and the other device read what it received at a different rate, the message would never be received correctly.
- If one device used even parity and the other device used odd then no correctly sent byte of information would ever be accepted.

The set of rules that needs to be set up to allow the transfer of data to be carried out is known as a protocol. There are a number of protocols that you may have heard of e.g. http, tcp (ip) but don't worry about them, you do not need to know specific examples of protocols, but you should be able to discuss some of the rules that go to make up a particular protocol.

1.5.i *Describe the need for communication between devices and between computers, and explain the need for protocols to establish communication links*

In the early years of personal computing it was not uncommon for people who had bought a particular computer to have to buy anything that they wanted to use on the system from the same manufacturer. This provided a ready built market for manufacturers but meant that the customer had very little choice. It also meant that if a customer wanted a particular type of printer, for instance, which the manufacturer did not supply then they would have to do without.

A major step forward was the idea of standardisation so that the hardware supplied by one manufacturer would be compatible with hardware from another. In order for this to happen the different rules for communication that these peripheral devices used had to be understood by the computers and these computers would then be able to tailor their messages to the peripherals in much the way that was suggested in 1.5.h. In order to understand the requirements of a peripheral it is necessary to give the computer its 'details'. This is done by running a piece of utility software called a hardware driver which tells the computer about the protocol to be used.

There is another aspect that should be considered and that is the need for the two devices (like the computer and printer) to tell each other that they are ready to communicate. This is another of the parts of the protocol and is called the handshake.

1.5.j *Explain the need for both physical and logical protocols and the need for layering in an interface*

The interface between two devices is the connection between them through which the data can be sent.

This connection between the devices can take the form of the physical connection, whether a wired or wireless method is used. If wired, is it serial or parallel? If wireless, is it radio communication or microwave? What frequency is the communication going to be on?

However, it also includes many other things like how is error correction going to be carried out? How are messages routed from one place to another? How is the flow of data going to be controlled? How is the data transfer going to be synchronised so that data does not become mixed up, and many other things that need to be agreed upon. These are the logical parts of the protocol.

Think of the physical parts of the protocol being the rules about providing the means of connection while the logical part is the rules about the form that the message itself will take.

This is a very complicated idea, but, in essence, there are two ways of setting up this interface. One way is to try to do the whole thing in one go, and the other way is to divide the interface into sections and do one bit at a time. For instance, first set up the rules about how the devices are to be connected physically, and later on worry about how to spot errors that have been made in the data transfer.

This connection between the devices can take the form of the physical connection, whether wires are used or some other form of connecting them together. However, it also includes many other things like how error correction is going to be carried out, how messages are routed from one place to another, how the flow of data is going to be controlled, how the data transfer can be synchronised so that data does not become mixed up, and many other things that need to be agreed upon.

This is known as layering the interface because the most basic layer has to be done first and then more layers may be put on top.

The main advantage of layering the interface is that changes may be made to one layer without having to change all the layers. It also allows manufacturers to develop devices for particular layers, knowing that they will be compatible with other devices at that level. Layers of the protocol can be altered without having to change the whole protocol, simply the layer that needs to be changed and the links linking the layer to those above and below it.

1.5 Example Questions



1. Explain the difference between a wide area network (WAN) and a local area network (LAN). (2)

A. -In a WAN the computers on the network are remote from each other, while in a LAN the computers are close to each other.

-The method of communication between nodes is likely to be different, with a LAN typically being hard wired and a WAN being connected by using phone lines.

Notice: The use of the terms LAN and WAN are perfectly acceptable without reference to what they stand for, as it has been stated in the question. Even if it hadn't been stated in the question, these sets of initials are stated in the syllabus so there is no problem in using them without explanation. However, be careful when using acronyms like this as what might be obvious to you may not have been heard of by the examiner, so a brief explanation on first mention is generally a good idea.

Two mark points available, note that they are not given for saying that the WAN is remote while the LAN is confined to a small area. This is really the same thing said the other way around. Be very careful about saying the same thing twice and thinking you have both marks where you have really only said one thing. Another example of this is when a question asks for advantages and disadvantages of two things. The same thing given as an advantage of one will not be given a second mark as the disadvantage of another.

2. a) State three pieces of hardware that are needed to create a LAN from a set of stand-alone computers. (3)

b) Explain why the communication over a WAN differs from that across a LAN and state how the hardware necessary for communication would differ from that used in part (a). (3)

A.a)-Network cards in each of the computers

-Cable to carry the signals from one point to another on the network

-A network server to control access to peripheral devices and to files on the shared hard drive.

b)-The distances between the computers are further than on a LAN...

-The cable used to connect up a LAN will only transfer signals over limited distances

-Other communication medium needed, e.g. modem and a telephone line.

3. Explain the difference between

(i) simplex

(ii) half duplex

(iii) duplex

transmission of data, giving an example of the use of each.

(6)

A. (i) –Simplex is the transmission of data in only one direction

-An example is the transmission of a television picture from the transmitter to the receiving aerial, there is no signal sent back.

(ii) –Half duplex is a means of transmitting data in both directions, but only in one direction at a time.

-An example of half duplex transmission is a walkie talkie where the handset needs to be set to either send or receive.

(iii) –Duplex transmission of data is the ability to send messages over the same link, in both directions, at the same time.

-An example is the use of the telephone system.

Notice: Nice, standard, question requiring a standard answer. As with so many of these questions, if you try to make up your own examples you could be making a big mistake.

4. Explain why the bit rate is more important when sending a video from one device to another, than it is when sending a page of text. (3)

A. -Video requires more data because each pixel must be defined

-Page of text has less data because each character is defined by its own byte of data

-Bit rate is the rate at which a given amount of data can be transmitted which becomes more important when the value of the data is dependent on the time at which it is put to use.

Notice: This answer is not quite right because it is possible to imagine a video which is being downloaded for use in three days time. In a situation like that it does not matter how long it takes and hence how small the bit rate is. Also this answer does not take any account of compression algorithms which may speed up the process of communication sufficiently. However, in principle this answer is true, and, considering there are only three marks, any attempts to complicate the answer must be counter-productive.

5. The following bytes of data are received by one device after being transmitted from another.

01001101

10001000

10101011

00011011

An automatic checking technique is used to check that the data has been transmitted without error.

- a) State which byte has been received incorrectly, explaining how you arrived at your answer. (3)
- b) Explain why it is possible that a byte of data could still be incorrect despite passing the test that you used in part (a). (1)
- c) Calculate the value of the checksum for this data. (3)

A. a)-10101011

-Bytes should pass a parity check

-Other bytes are of even parity while this is odd

- b) -If two bits in the same byte are in error the effect of each will cancel out the other.

c) 01001101

10001000

10101011

00011011

10011011

10 1110111

Mark points:

-8 bits correct

-Further bits have been rejected in order to limit to 8 bits

-Method shown (carries)

Notice: This is a typical way of asking about parity. The other type of question asks for a description of parity which is far more difficult to answer and hence would be aimed at about a C grade candidate whereas part (a) should be accessible to a grade E candidate. Part (b) is a standard answer while part (c) is a convenient way of asking a question from 1.3.c and hence is a common question because it covers two areas of the specification at the same time. Notice also that it is necessary to show the working out for full marks, there being other methods that can be used which do not involve using the binary addition stated in the specification.

6. Explain the difference between packet switching and circuit switching. (2)

A. -Packet switching sends message in equally sized pieces in any order, while in circuit switching the message is sent in equal sized packets in order meaning that it does not need to be reordered at destination.

-Packet switching involves the individual packets being sent onto the network before a route is established, while circuit switching requires a route to be established before transmission.

Notice: A difficult question. The immediate reaction is to say that circuit switching needs a route to be established before transmission. Packet switching does not need a route to be established. This is the same point from two perspectives, and hence is only worth one mark. The second mark is much more difficult to find. There is a common misconception that 'packets' is a suitable difference between the two methods, whereas, in reality, the message is divided into packets however it is sent. This was, unfortunately, strengthened because of the previous version of this CD, hopefully the problems are now sorted out.

7. A computer is to use a printer to provide hardcopy output of jobs. In order for the data to be transmitted and received properly a protocol must be set up between the two pieces of hardware.

State two parts of the protocol which would be essential in this example, giving reasons why they are necessary. (4)

A. -Baud rate...

-necessary because if the computer transmits at a different speed to that which the printer can receive the message will become jumbled.

-Parity type...

-because if the computer transmits using odd parity and the printer checks using even parity correct bytes will never be accepted.

Notice: There are many possible answers to this type of question. You may know more than the ones stated in the text of this chapter, be warned, stay with the ones in the text then there is no chance of misunderstanding. Some students, on sight of the word protocol, desperately want to write down TCP/IP or http. Be very careful. It would be difficult to think of a question, at this level, which could possibly have those as answers.

8. Explain what is meant by a handshake. (3)

A. -A pair of signals...

-sent between two hardware devices...

-to ensure that they are ready to communicate.

Notice: An easy, definition, type of question, but what other question on this topic can be asked?

9. Explain why protocols are arranged in layers. (3)

A. -Allows different rules within a protocol to be set up independently

-Allows manufacturers to design devices for particular layers

-Individual layers can be altered by changing the layer and links to other layers.

1.6 Implications of Computer Use



1.6.a *Discuss changing trends in computer use and their economic, social, legal and ethical effects on society*

The history of the computer has been remarkably short for a device that now pervades every aspect of our society and affects the lives of so many people in a serious way.

The first effective, useful, stored program machine is now acknowledged to have been the Colossus which was a computer that was designed to break the German Enigma codes during the war. Confusion over which computer was the first has been rife because of contrary claims and a lack of definition of what constitutes a computer. Different centres around the world fiercely put forward their claims to have invented the first computer, an argument that was settled by the lifting of the official secrets act from Colossus in the 1970s.

Such a seemingly mundane argument over something that only carried prestige for the eventual winner of the argument, not money or power, was a symptom of the jealous way that people who were involved with computers, and who were able to use them, guarded their privileges from others.

This mentality gave rise to the concept of 'Fortress Computing' where computer competent people kept others away from the systems. There was some justification for this in order to use a scarce resource as effectively as possible, but there was undoubtedly an element of such workers protecting their position by making computer use seem somewhat mystical. The results of this are still seen today in the attitudes of many middle aged, and older people, who believe that using a computer is beyond them. This is not their fault, but something with which they were brain washed when they were younger.

The use of computers was largely confined to this technical elite in universities, and in larger businesses, where information systems were needed to produce payrolls and other information hungry examples. The revolution in computer use came in the late 1970s with the invention of a cheap, small, relatively easy to use machine called the Sinclair ZX79. There was very little it could do, for instance, it had 1/4K memory! But it did mean that ordinary people could own their own computer. Also, the phenomenal success of the ZX79, ZX80, and then the Sinclair Spectrum proved that there was a market. Until these machines, no one thought that ordinary people would want a computer, indeed in the 1950's, the home secretary announced to the House of Commons that there were two computers in England and he could not imagine that there would ever be a need for any more. Suddenly many manufacturers jumped on the band wagon and there was a plethora of machines. Each machine needed its own software and many manufacturers started to go out of business. There was no real market leader, although Acorn, in this country, were very popular because their machines were being used in schools, so parents wanted to buy one for their children to use for school work. In the 1980s IBM, the world's biggest computer manufacturer at the time, decided that they would produce a small machine. It was not successful, but many other, smaller, manufacturers decided to copy the way that it was built. Such machines were called IBM clones. Gradually, as the IBM clones grabbed a larger and larger share of the market, the other companies went to the wall. It certainly helped that Microsoft operating systems were installed in the cloned machines, not because they were the best operating systems, but because it meant that these machines could communicate easily and that files were compatible. Nowadays, we have a society where, world-wide, there is a higher number of computers in homes than there are refrigerators or baths.

The effect of this has been enormous on society, and the effects are what are discussed in this chapter.

35 years ago the proportion of the world's population that had access to a computer system was tiny, nowadays it is very high, and increasing daily. However, there are still large numbers of the population, even in a rich country like Britain, that do not have access on a regular basis. There are many reasons for this but one in particular is age. Older people, as mentioned above, are often not confident with computers and hence tend not to use them. There is a distinct danger that this will lead to another form of class-divided society, not based on accident of birth, or based on wealth (or lack of it), but based on whether a person has access to the information revolution.

Imagine two people who live in a village. The village has a grocer's shop and newsagent. The last bank in the village has closed because it was too costly for the bank to maintain the branch, which was losing money, because so many of the villagers had changed to running their account on the Internet. A person living in this village, who has a computer connected to the Internet, will be able to pay their bills, order goods from the supermarket 20 miles away at a lower cost than buying them from the grocer, will have a wider variety of choice from the supermarket than they could get locally. They will be able to book rail tickets, at a discount, through the Internet, which will also allow them to pick up holiday bargains and cheap flights. The other person does not have access to a computer. These two people are very different in their life styles and the social changes that have occurred, not just to one of them, because of the computer, are immense.

When we add to this the fact that one of these people used to be employed by the bank to run the branch in the village, but is now out of work, while the other has been told to go on a training course to learn a new piece of software so that they can work from home and that they get a pay rise because the skills that they are using make them more qualified, the scale of how much our lives are being affected by the use of new technology becomes even more evident.

Note that many of the changes made to our life styles have been forced rather than being made by choice, people like to be sociable around the village and meeting and talking in the village shop. People like to be able to discuss things with a travel agent or to go to work surrounded by colleagues and to form social relationships with them. The managers of the company may well prefer to have all the workers together in a single office building if only to monitor their work better, but how many businesses can afford that nowadays?

Legal changes have also been immense. For example what would have been the point of having legislation to force full public disclosure of records if the records are not kept on computer, after all no one would be aware of the existence of the records to start with. When the original data protection legislation was passed by parliament in 1984, records kept on paper were exempt. You could say what you liked about someone as long as it wasn't on a computer! (Not quite true, but almost).

What about the effects of the use of computers on other societies? The contents of the Internet are available anywhere in the world. The majority of the content is aimed at Western, well-off society. What right do we in this country have to try to impose our views on other societies? What effect does this content have on the people who look at it?

1.6.b *Explain changes to society brought about by the use of computer systems*

Computer availability and acceptance has brought about large changes, not just in the ways that we live, but also in the way we use our leisure time and in work practices.

Children, spend long hours of their leisure time in front of computer terminals playing computer games. This is a very insular way of passing the time which is beginning to worry educational psychologists because children who behave like this are in danger of not forming the social bonds, and learning about the social interaction, necessary in later life. Some of the games are particularly violent, or antisocial. Although there is no hard evidence of a link between playing such games and antisocial behaviour of the child themselves, there is a growing belief that there is such a link. Long hours spent in front of the monitor screen at an early age is becoming suspect in the amount of RSI and other injuries being suffered by young adults, many years before their parents generation would have shown such symptoms.

Work practices are changing in a number of ways.

- Computers have meant that some repetitive tasks have been taken away from human beings. A robot, on a car production line, can not only work more consistently than its human counterparts and do the jobs of many people, it does not need paying. Many jobs have been lost by the human work force in this way. However, companies that have invested in this way have tended to become more profitable which has the effect of safeguarding the jobs that are still offered by the company, and also leads to expansion and the creation of additional jobs.
- Additional jobs are created. Every time a robot is placed on a production line, someone has to build the robot, someone has to write the software to control it, and someone has to maintain it. The problem is that the majority of jobs created require more skills than the jobs that have been lost. This in turn means a 'skilling up' of the workforce. This change in working practices is another example of the technology creating two classes of people (Section 1.6.a), those able to learn additional skills and those not.
- The use of technology allows people, whose jobs require a desk and who can communicate satisfactorily with their colleagues without needing to be in the same physical environment, to have their desks wherever they want by using electronic means of communicating. This means that people can work from home for at least part of their week. The implications are far reaching. Less pollution because of fewer commuters and pleasant working conditions are both positive factors. Less positive is the ability of the electronic systems to report the working practices of the workers to their boss, and the lack of social contact which is so important in maintaining job satisfaction.
- In the past it would have been quite normal for a person to start a job after leaving full-time education and to stay in that job until they retired. Although this may be seen by some as being an uninviting prospect, it does provide for security and gives automatic prestige to even the lowliest of workers because of their seniority within the business bringing invaluable experience. This type of work practice is becoming less and less common as modern technology changes products, manufacturing techniques and the life cycle of businesses.
- Electronic communications have changed the way that business is done. Communication can be immediate, and the worker is never far from being contacted and consequently can be at work at all hours of the day. A good example is the world's stock markets. Twenty years ago, each country had its own stock market, the performance of which might affect others when they opened, but they tended to be fairly insular in their dealings. Nowadays, the stock markets of the world are all interconnected. As a stock-holder, it is possible to buy and sell stock at any time of the day or night simply by using one of the markets open at the time, no longer is it necessary to wait for the national market to open the following day.

1.6.c *Discuss the effects on privacy and confidentiality of data held in computer systems and steps which can be taken to protect confidentiality*

The expansion of computer systems around the world has given rise to more information being stored about individuals than ever before. The problem is that, to get another copy of a person's information, the only thing that needs to be done is to press a button. Distance is no object either. Using electronic communications it is possible to send a person's information from one side of the world to another in milliseconds and then to store that information in a system that the owner of the information is unaware of.

There would be no problem if it could be ensured that all the users of the information were going to use it for the right reasons and that the information was always accurate. However, these things cannot be guaranteed. Consequently it has become important to protect individuals from misuse of their information by legislation and by measures that restrict both the access to the data and the passing on of the data to other users.

Unfortunately the use of electronic information does not recognise national boundaries and some authorities are not as anxious to uphold the rights of the individual as others. A typical effect of this passing on of personal information is the junk mail that comes through the letter box or to your inbox. An interesting experiment is to change your initials every time you have to give your name, or to have different email addresses. Within a relatively short space of time junk mail will begin to arrive and by reference to the initials or to which mailbox it is delivered it is possible to work out who has sold your details on and to whom they have been sold.

The ease with which data can be accessed remotely when it is held in computer systems and the enormous volumes of data now held have meant that measures need to be taken to protect it. Measures like use of passwords and user access rights are important as are devices like firewalls (a piece of hardware designed to deny access to a system from someone trying to illegally access it from the outside or software that will do the same job).

1.6.d *Understand the need for legislation governing computer system use*

Most countries have now enacted legislation in attempts to overcome the problems that were mentioned in section 1.6.c. The purpose of such legislation is to protect citizens from something that by its very nature can be insidious because it is difficult to understand the consequences of unrestricted use of personal information.

Legislation protecting personal information must not be confused with copyright law. Copyright law protects intellectual property, in the case of a computer system it would protect the person who has written the software rather than the user or the person whose details are being stored. Each country will pass laws that are particular to that country, but they largely follow the same model that we have in Britain.

Any personal data that is stored in a computer system must:

- be accurate and up to date.
- be relevant to the task for which it is intended to be used and used for that original purpose.
- not be kept longer than is necessary.
- be kept securely. Access to the data must be strictly controlled.
- be available, on request, to the subject of the stored data.

Other measures that could be expected would be some control over the passing on of data to other organisations, exceptions being for reasons of national security or because it is thought to be in the individual's interest.

In Britain the Data Protection Act that was passed in 1984 only applied to data that was read magnetically, in other words was stored on a tape or a disk. It did not apply to data stored on a CDROM because that is stored optically. This has now been changed by the latest version of the act which do not specify how the data is stored but simply that it is being stored. This shows that, just like everything else in the world of computers, things change very quickly and that human systems often find difficulty in keeping up with the changes.

Attempts have been made to harmonise legislation across boundaries. The European Union passes their own directives about data, but it is difficult to get agreement when individual parliaments still have sovereignty.

1.6 Example Questions



1. Discuss the implications for customers of a travel agent closing down their agencies and doing all their business on the Internet.

(6)

A. -Lower overheads for the business...

-meaning that customers can expect to pay lower prices.

-Information will always be up to date because up-dated centrally..

-but customer may find more difficulty in finding the information required because there is no assistant to help.

-Customers who have the Internet will be able to access late bargains

-Customers without the Internet may find difficulty in booking any holiday.

-Customers with disabilities, particularly blindness may be more disadvantaged than most

-Complaints, if things go wrong, will be harder to make.

Notice: There are other points to make but only 6 marks so don't waste too much time. When you think you have earned the marks, stop. The question is quite specific about it being from the point of view of the customer, so don't start to talk about workers losing their jobs. It may be true, but does not answer the question. Although the mark scheme would be set out in this way, the question does use the keyword 'discuss', so don't just write notes, the examiner is looking for a coherent argument which contains six of the sensible points.

2. Describe **two** ways in which a computer system in the home can alter the interaction of the family with the outside world.

(4)

A. -Children play computer games...

-stops them going out and consequently they are safer, but...

-affects their social development.

-Parents may use the computer to work from home...

-cuts down the time spent commuting to work and allows for more leisure time...

-but may increase stress and worry about work because of reduced contact with colleagues.

Notice: Sensible to use obviously different members of the family. Need to say at least two things for each one because the question uses the keyword 'describe'. If you were thinking of saying that they could send emails rather than write letters it may be worth a mark but is a little trivial for both marks.

3. State **two** reasons why it is necessary to protect the confidentiality of information stored about people. (2)

A. -Data can be misused...

-by insurance companies to alter an insurance premium,

-by thieves who want to know which addresses have large premiums and are hence worth burgling

-by employers who can check on employees moonlighting.

Notice: Again, many different possible answers. This would be a nice starter question for a paper, as it expects simple one line answers.

4. Describe **three** measures that would be expected in a country's data protection legislation. (6)

A. Any three of the measures listed in section 1.6.d.

Notice: This is a banker question. It is such an important area of the syllabus it is likely to be asked on a regular basis, the problem for the examiner is that there are only a few ways of asking the question. Candidates should learn the measures stated in any data protection legislation and use them to answer the question.

5. Describe the consequences of allowing upper school pupils to use the Internet to find information for project work. (4)

A. -There is a danger that those pupils who have access to the Internet outside school will use it anyway, and...

-therefore it is only right to allow access in school.

-Pupils may find inappropriate sites...

-which means that the teacher has to censor the material in some way.



2.1 Designing Solutions to Problems

2.1.a *Discuss the importance of good interface design*

Any interface must clearly ask for relevant input or report the results clearly to the user. This input may be by means of questions, forms speech and so on. The output may be on a screen, in the form of a written report, use graphics or sound and so on.

Whatever the format of the interface it must not be confusing or ambiguous. Users should instinctively know what is required of them or there must be clear instructions explaining what they need to do. There should also be on-screen help available whenever possible.

Also, the layout should be natural to use; for example, if an input has to be copied from a printed form, the form on the screen should be similar and all values should be in the same order on the screen and the completed form. The order of data should be natural. For example, an address should be in the correct order such as street, town, county and postcode; not postcode, street, town, county. A name should consist of a title, surname and forenames. Wherever possible guide the user towards correct data entry. This can be done by using drop down list boxes or radio buttons.

If input requires more than one screen, make sure all but the first and last screens allow the user to move to previous or next screens; the first screen should allow the user to move to the next screen and the last screen should allow the user to move to the previous screen; all screens should allow the user to quit (without saving). When all the required data has been entered, the user should be able to save. When data is entered, it should be validated and a clear error message produced if necessary.

Many candidates give long discussions on font, colour and so on. Although this is important take care not to over emphasise these. A good interface will use few colours and very few different fonts. Emphasis should be on size, the use of bold, italic and so on. Don't try using too many styles at the same time. Text that is underlined and is in bold italics is extremely difficult to read and very tiring on the eyes. Look at magazines such as newspapers and magazines to see the use of few fonts but many sizes and styles.

When designing the user interface you should remember the following:

However complex the software, however expensive and powerful the hardware, the system is unusable if there is no intuitively simple to use interface with the human being who is in control of the system or for whom the system is producing results. The Human Computer Interface (HCI) must be unambiguous, allow the user to input all the data that the user thinks important, and produce the output in an easily understandable form.

The intended user of the interface must be taken into account. The interface design for a preschool playgroup to learn about numbers is going to be very different from the design of an interface for the manager of a chemical plant to keep a check on the reactions around the plant. The interface that this manager would use to study a particular reaction (perhaps tables of figures, or graphs showing the different parameters of the reaction) is very different from the interface the same person would use if they wanted to see the flow of a chemical around the whole site. This would probably be in the form of a diagrammatic representation of the site. The diagram itself would supply valuable information about positions in the plant

before any current information was put up. The circumstances under which the interface is to operate are also important. In the last chapter mention was made of the nurse looking after a group of patients. In those circumstances an audible warning that a problem may be occurring is far more sensible than a visual one which may be ignored if the nurse's attention is drawn away from the screen. Finally, the designer of the interface needs to take account of the purpose for which the interface was designed, namely the transfer of data into the system and the communication of information from the system to the user. If the user is using the computer to play a game, then the paramount importance of the interface is that it adds to the enjoyment of the user.

Consideration of the purpose of the interface:

- who it is for
- what information needs to be conveyed
- the circumstances under which the interface must operate
- the effectiveness of the communication
- the enjoyment obtained from using it

...are all important when the interface is being designed.

2.1.b *Design and document data capture forms, screen layouts, report layouts or other forms of input and output (eg: sound) for a given problem*

In order to do this you should consider the following important issues first.

Short Term and Long Term Memory

The information produced by a system can be of such a large volume that it is not possible for all the information to be stored for future use by the user. Some of the information is presented via the user computer interface, perhaps in the form of a scrolling graph. Such data is very transitory in nature because the screen display is being constantly updated. Such data is part of the short term memory. The long term memory is that which stores the data for future reference.

Visual Perception

This includes all the information that is presented by the system in such a way that it can be seen with the eyes. This means that the information will be presented on a screen or on a hardcopy produced by a printer. The size of individual pieces of information will be important as will the contrast used and the type of font in the case of text.

Colour

Colour is an important part of any HCI. The contrasting use of colours can highlight the more important information, or can be used to distinguish one type of information from another. The different levels of contrast between colours are necessary if the individual items of information are to stand out. Black on white provides the highest possible contrast, while dark blue on black is very difficult to decipher.

Layout

The layout of the data on the screen is important. The eye naturally reads from left to right and from top to bottom. This means that more important information should be positioned toward the top and left of the screen. The volume of information on the screen at any one time is also important because there is a limit to the amount that the eye can follow, and the brain distinguish, in one sweep of the screen content. If the information should be seen in an order, then the correct screen order is from top left to bottom right corners, as the eye naturally reads.

Content

The content of the information presented is important because a user will soon begin to ignore items of information that are constantly being put on the screen despite not being necessary. Similarly, if a method is used to show that a piece of information needs urgent attention, while the operator does not perceive the urgency, then all such highlighted information may be begun to be ignored in the same way.

Data may be captured in many ways, one of which is by means of forms. Form design requires a lot of thought, it is not simply a list of headings. The order of the headings is important as is the need for clear instructions on how to complete the form. It is worth collecting some forms and noting the order of the questions and the clarity of the instructions. Make a note of a few that you think are good and list your reasons for choosing them. This will help you to design your own forms.

Users should find that the order of entering data on a form is natural. If the full name and address are required, then keep the forenames and surname close together, do not have surname, address and then forename. Make sure there is sufficient space for the data required and that the instructions for completing the form are clear. For example, if the date of birth is required, a good layout is

Date of Birth:	D	D	M	M	Y	Y
----------------	---	---	---	---	---	---

Now design a form to collect personal data about pupils in a Primary school.

First decide what data are needed. Clearly, forename(s), surname and address are needed. But you will also need the date of birth of the pupil and details of parents/guardians and emergency contact details.

Now you must decide on whether to use free format or indicate the maximum number of characters for each piece of information. Finally, what should be the size of the form? Should the form be printed on both sides? Should it be in landscape or portrait orientation?

One solution is given here but there are many good solutions. You will be marked for content, order of data entry and clarity of instructions. In this solution, A5 paper is used and the form is on both sides of the paper which is in landscape orientation. Side 1 contains the pupil's details and side 2 the parents'/guardians' details.

HIGH TOWN PRIMARY SCHOOL	
Please complete using BLOCK LETTERS	
PUPIL'S DETAILS	
SURNAME:	<input type="text"/>
FORENAME(S):	<input type="text"/>
ADDRESS:	
STREET:	<input type="text"/>
TOWN:	<input type="text"/>
COUNTY:	<input type="text"/>
POST CODE:	<input type="text"/>
DATE OF BIRTH:	<input type="text"/>
SEX(Tick one):	<input type="checkbox"/> Boy <input type="checkbox"/> Girl

PARENT/GUARDIAN	
TITLE (DR, MISS, MR, MRS, MS, ETC):	<input type="text"/>
SURNAME:	<input type="text"/>
INITIALS	<input type="text"/>
ADDRESS(If different from pupil's):	
STREET:	<input type="text"/>
TOWN:	<input type="text"/>
COUNTY:	<input type="text"/>
POST CODE:	<input type="text"/>
TELEPHONE:	<input type="text"/>
Please include area code.	
EMERGENCY	
NAME:	<input type="text"/>
TELEPHONE 1:	<input type="text"/>
TELEPHONE 2:	<input type="text"/>

When the data is entered into a computer system, it may be done by the system asking a series of questions or by the user completing a form on the screen. In either case, be careful that the screen does not become cluttered and that it is possible for the user to correct errors. If the system asks a series of questions, it is easy for the screen to become cluttered. It can also be difficult to correct early errors. The following is a poor design as the screen scrolls and it is difficult to correct early errors. (The italicised data are entered by the user.)

A Amend a record
B Add a record
C Delete a record
D Quit

Choice: A

Enter key of record to be amended: SY14

Surname: SMYTH

Do you wish to change this surname? Y

What is the new surname? SMYTHE

Forenames: WILFRED BRIAN

Do you wish to change the forenames? N

Address: 17 Sun Street
New Town
Blackshire
NT1 7SS

Do you wish to change line 1? N

Do you wish to change line 2? N

Etc.

When designing screen layouts, you should consider very carefully the experience of the user. Most users for whom you will be writing solutions will be novices, or knowledgeable but intermittent users. We will consider the needs of the novice user and see how we can modify these for the knowledgeable user who will only use the solution occasionally.

- The user will not be confident of what the system requires; this means that you must make sure that the computer makes it clear what is to be done. Thus, all responses should be brief so that users can remember what is needed.
- Responses should be what is expected and consistent.
- The layout of the screen should have a consistent pattern so that the user can expect to find similar information in the same part of the screen throughout the use of the software.
- Input should be obvious so that no, or very little, training is necessary.
- Any messages to the user should be clear, unambiguous and jargon free. Use simple English in which the grammar is correct and sentences are short and to the point. A message to a novice user is completely different to that to an expert.
- Do not give the user too many options from which to choose (although the more experienced user can cope with a few more than the novice). Consider using short menus with sub-menus so that the user is guided through the choices.
- Remember that novice users vary considerably so that some are quicker than others. Let the user define the pace. If there is a message for the user, make sure that the user can decide when to move on.
- The user should know naturally when a response is required. Do not create a situation where the user is left trying to work out what to do next.

We shall now look at a piece of software called Roberta's Diamond. This is an adventure game for Key Stage 2 pupils. It involves pupils moving through a house and garden to find a stolen diamond. In order to move, pupils must solve mathematical problems; some of these are easy but some need further investigation. This means that pupils must be able to save their positions so that they can continue when they have solved the problem.

Fig 2.1.b.1 shows the initial screen. Most of the screen is taken up by the title but command buttons, down the left hand side, clearly state what can be done. Here there are only four options.

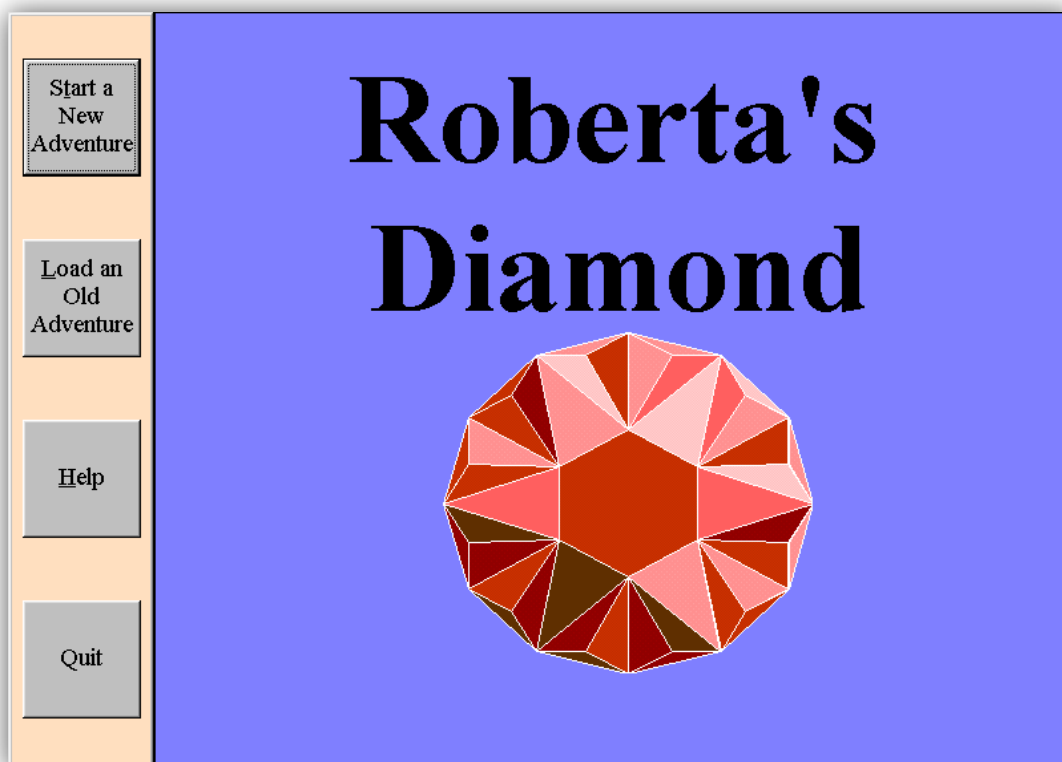


Fig. 2.1.b.1

If pupils choose *Start a New Adventure*, the screen in Fig. 2.1.b.2 appears. Notice that the left hand window has hidden two of the original buttons and introduced a new button.

To the right of the command buttons are two more windows. The upper describes the position in the game and the lower the user input. This is the format used throughout the software. In this case users are told where they are and then asked a simple question with only two possible answers.

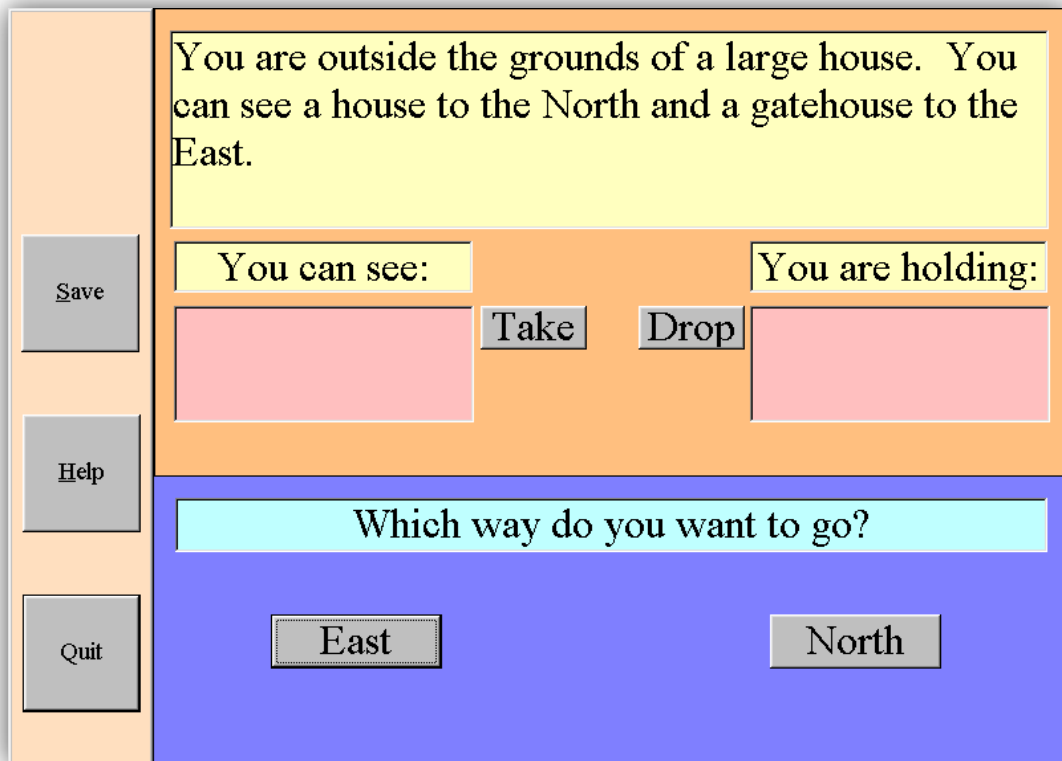


Fig. 2.1.b.2

A similar screen is shown in Fig. 2.1.b.3. In this screen the centre of the screen shows the contents of two list boxes from which the user can take objects. Take and drop are obvious but a user may click one of these buttons without first choosing what to take or drop.

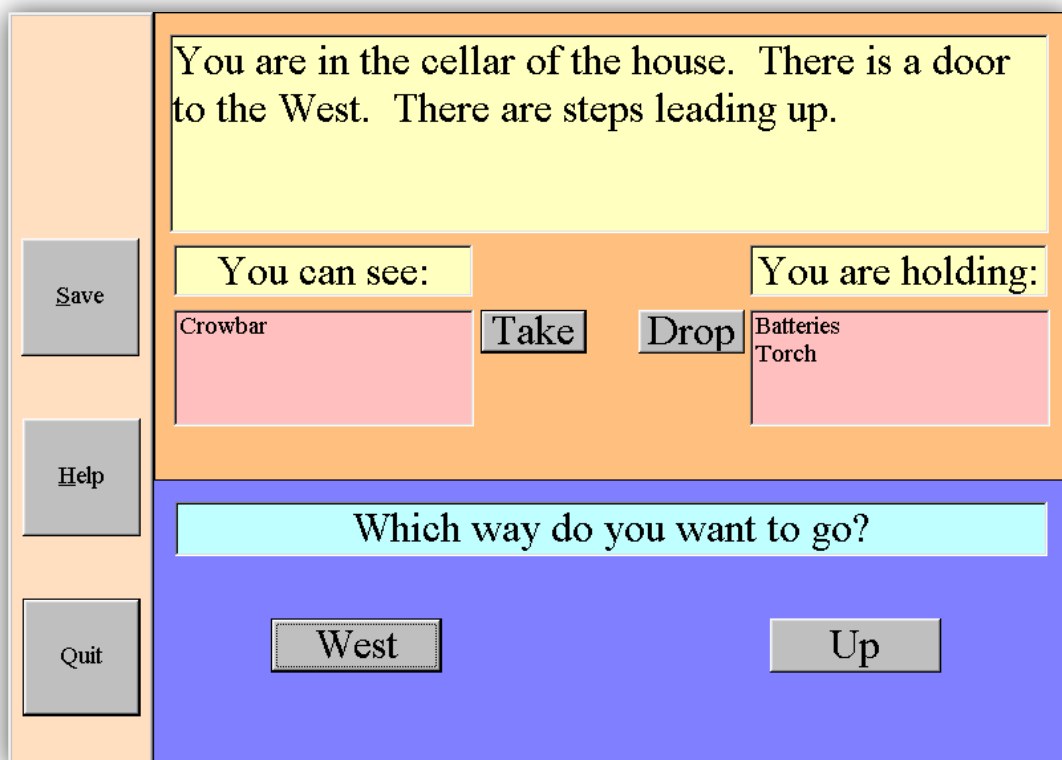


Fig. 2.1.b.3

If a player does click either *Take* or *Drop*, an error message appears. This is shown in Fig. 2.1.b.4. Notice that this window has a red background to draw the player's attention. It has a simple error message and is positioned at the bottom of the screen which is where nearly all input occurs. There are two reasons for this. Firstly, this is where the user will naturally look to input data and, secondly, it prevents the user

entering anything until they have completed the Take or Drop correctly. (All windows, except the error message one, are disabled at this point so the user has to click the O.K. button in order to continue.)

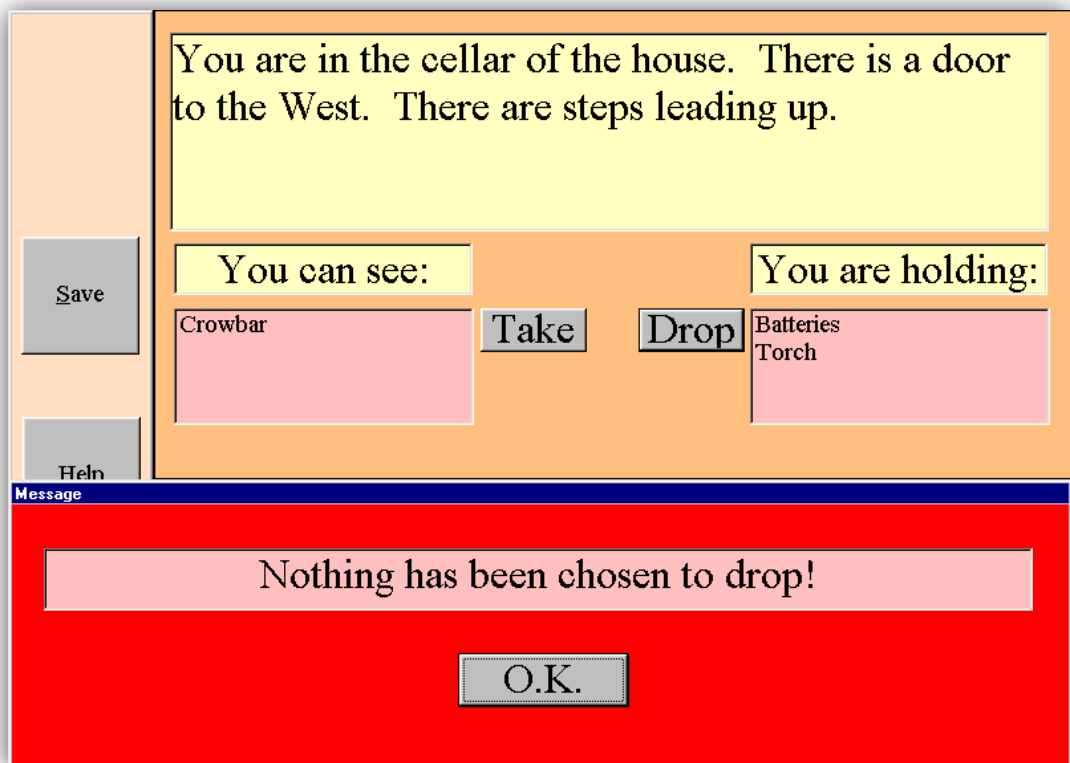


Fig. 2.1.b.4

In order to move, users must solve mathematical problems. One of these is shown in Fig. 2.1.b.5. Notice that this fills the screen so that the user's attention is concentrated on the problem. Further, the upper part of the screen gives details of the problem while the lower part is for input. This is consistent with the other screens. This consistency is very important as it trains the user to look at the correct part of the screen for information and for entering data. Users should not have to keep searching screens to find out what to do next. In order to maintain this consistency, you must try to think of all the possible scenarios that may occur during the running of your solution. You will then be able to design consistent screens.

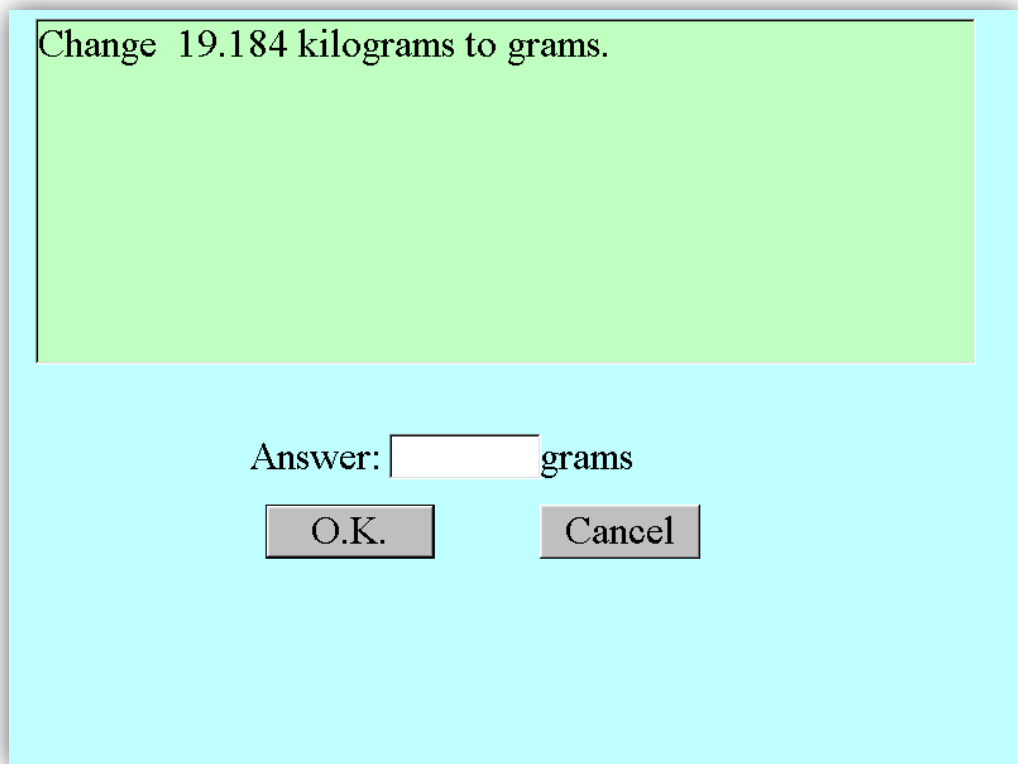


Fig 2.1.b.5

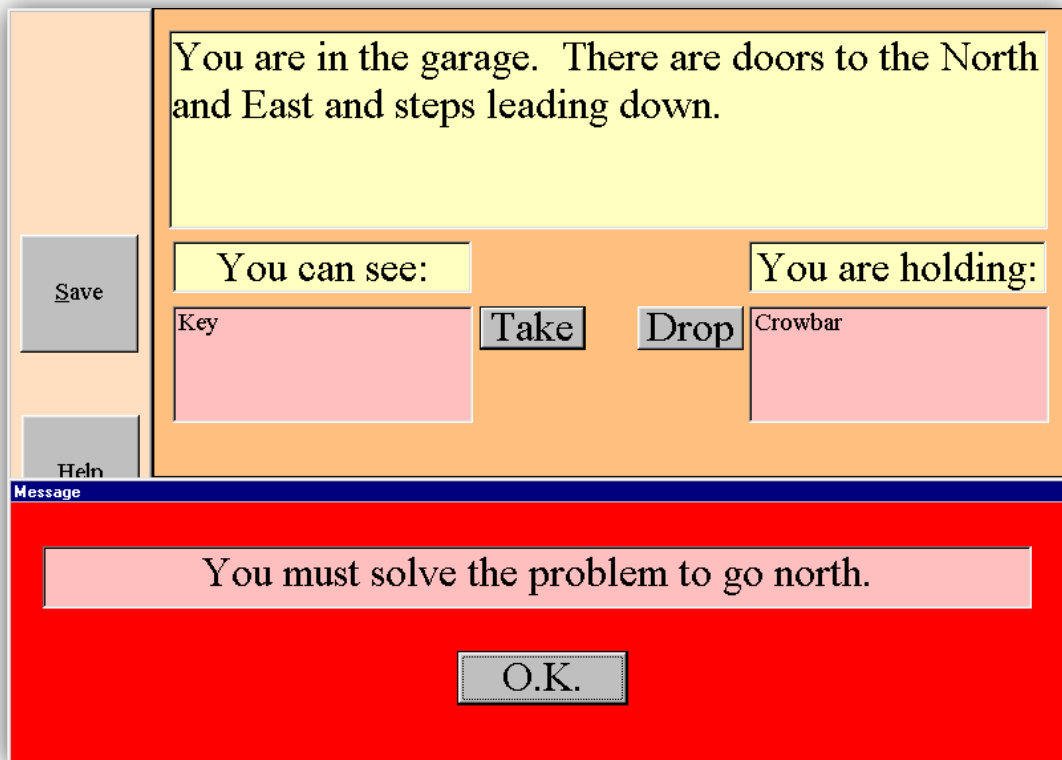


Fig 2.1.b.6

Fig. 2.1.b.6 shows the message that appears when a problem has not been solved. As before, the background is red and it is displayed at the bottom of the screen.

Sometimes a problem needs a diagram. Diagrams are consistently placed in the centre of the screen, between the description and the input window. This is shown in Fig 2.1.b.7 and Fig. 2.1.b.8. Notice that in Fig. 2.1.b.8, it is easy to move a disk. With all the screens, if the input has to be typed in, the user may click the O.K. button or press the enter key on the keyboard. This means that users can use the most natural method for them.

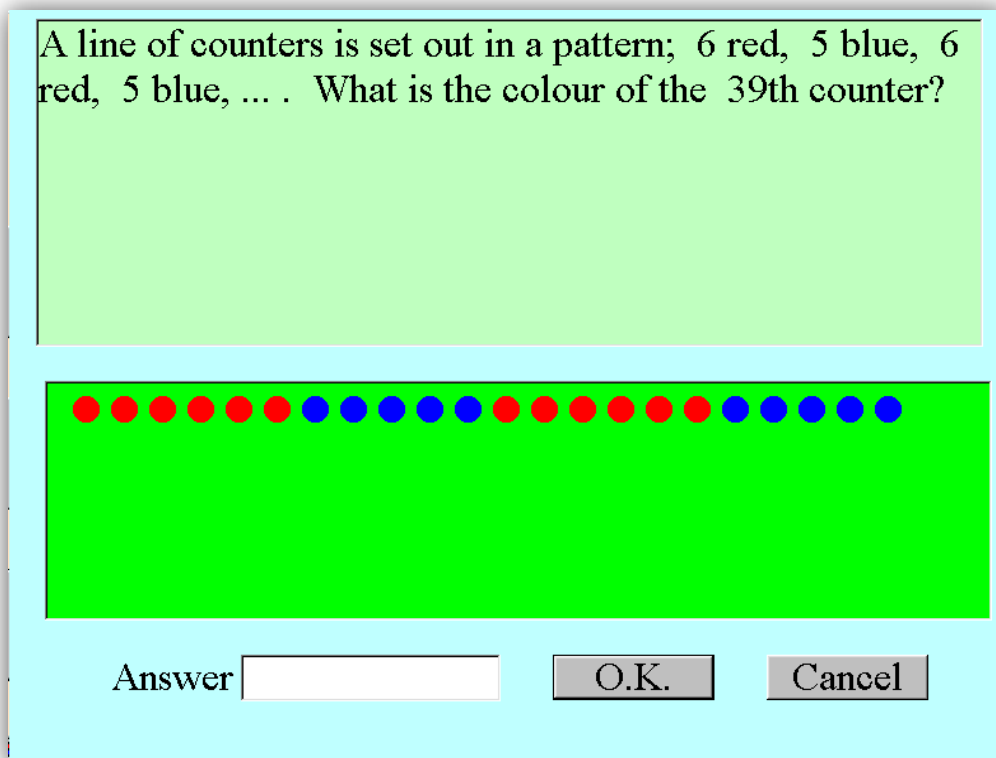


Fig 2.1.b.7

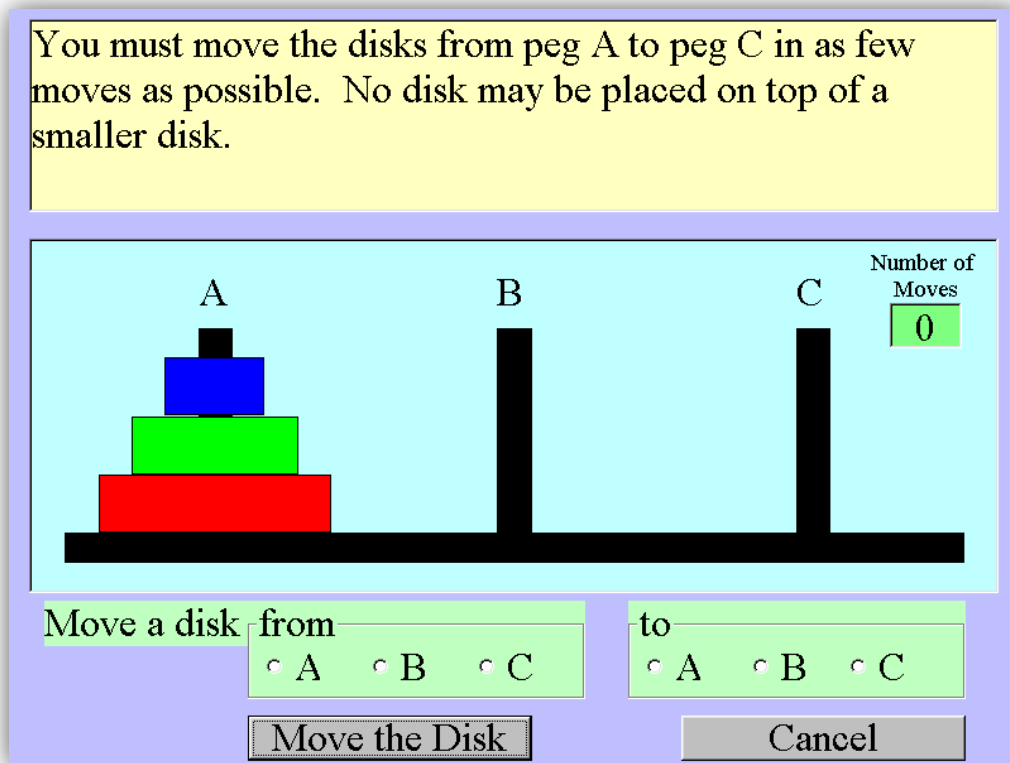


Fig. 2.1.b.8

Now we will consider the design of reports. It is important that reports are appropriate to the intended reader. For example, suppose a report is required that shows the expenses for four people. Fig. 2.1.b.1 shows the report in the form of a spreadsheet. This enables the user to compare expenses very easily.

B & L Publications					
Monthly Expenses for August 2008					
Printed 3rd September 2008					
	Travelling	Telephone	Postage	Meals	Totals
A. Beans	176.48	83.49	3.45	89.76	353.18
C. Doncaster	298.31	97.52	4.95	187.95	588.73
E. Franks	158.23	76.48	2.50	76.45	313.66
G. Harris	167.41	85.93	3.74	0.00	257.08
Totals	800.43	343.42	14.64	354.16	1512.65

Fig. 2.1.b.1

This is useful if the user wishes to make comparisons or needs detailed analysis. However, this can be very confusing for some people; also it contains too much detail for the needs of some readers. A graph may be more appropriate in these circumstances and the data is shown in graph form in Fig. 2.1.b.2.

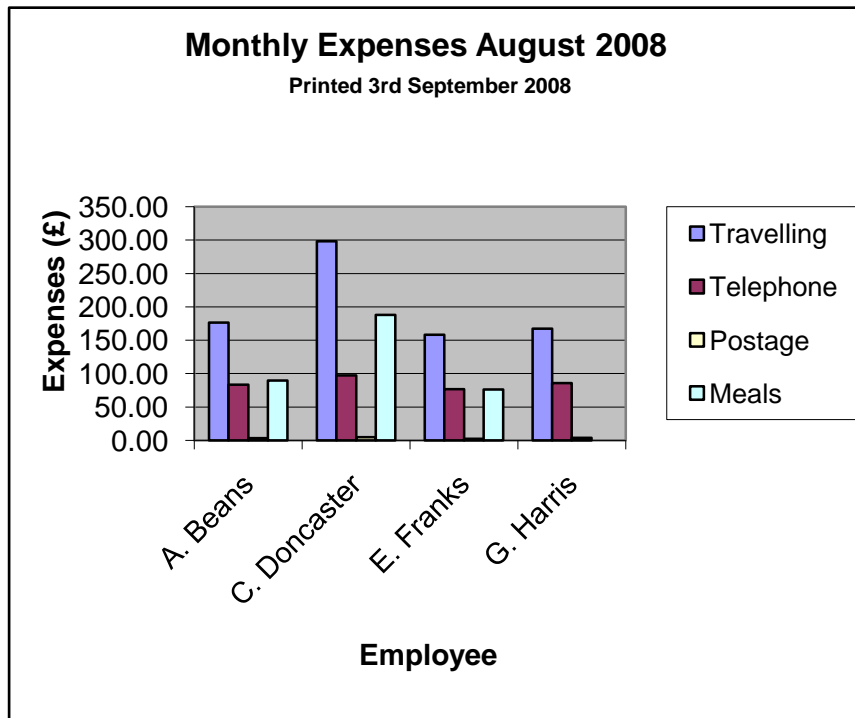


Fig. 2.1.b.2

However, in some circumstances the report shown in Fig. 2.1.b.3 is more appropriate.

What should you consider when producing a paper-based report?

- All reports should have a title and any appropriate sub-titles together with an indication of the period to which the report appertains.
- If the report is in columns, the columns should be clearly labelled.
- If any abbreviations are used, they should be meaningful to the end-user.
- It is important to group information according to the needs of the end-user and to make sure that field sizes are sufficiently large.
- All pages should be numbered but be careful using the system <page number> of (number of pages> (e.g. 63 of 67) as this can cause problems if a page is deleted or inserted. If this happens the whole of the report will have to be re-printed.
- Make sure that the report has aesthetic appeal.

B & L Publications	
Monthly Expenses for August 2008	
Printed 3 rd September 2008	
A. Beans	
Travelling	£176.48
Telephone	£ 83.49
Postage	£ 3.45
Meals	£89.76
Total	<u>£353.18</u>
C. Doncaster	
Travelling	£298.31
Telephone	£97.52
Postage	£ 4.95
Meals	<u>£187.95</u>
Total	<u>£588.73</u>
E. Franks	
Travelling	£158.23
Telephone	£ 76.48
Postage	£ 2.50
Meals	<u>£ 76.45</u>
Total	<u>£313.66</u>
G. Harris	
Travelling	£167.41
Telephone	£ 85.93
Postage	£ 3.74
Meals	<u>£ 0.00</u>
Total	<u>£257.08</u>
Totals	
Travelling	£800.43
Telephone	£343.42
Postage	£ 14.64
Meals	<u>£354.16</u>
Total	<u>£1512.65</u>

Fig. 2.1.b.3

When designing screen output, you will need to make similar decisions to those made for printed output. However, remember that screen output is more transient, it is not permanent like a printout. What needs to be considered for screen output?

- Remember that the size of the screen restricts the amount of information that can be displayed. Do not try to put too much on the screen at once.
- Use linked screens if there is a lot of information. The first screen should be linked to the next screen. The second and further screens should be linked to their next and previous screens and possibly to the first screen.
- Large characters and different fonts can be useful for headings as can the judicious use of colour.
- Do not overdo the use of different fonts and colours; the eye cannot cope with too many changes.

Now design a report that displays the expenses of the four employees of B & L Publications. The report should group the data by type of expense not by employee, as in the text. You may use the data given in Fig. 2.1.b.1 to show how the report will look when data is entered. (8)

Marks will be awarded for the following points.

- The name of the company is shown
- A heading showing the purpose of the report
- The period to which the report appertains
- The date the report was produced.
- Correct grouping
- Suitable indentation
- Sub-totals
- Grand total
- Easy to read and understand

A typical layout is shown in Fig. 2.1.b.4 and again in Fig. 2.1.b.5 but with data.

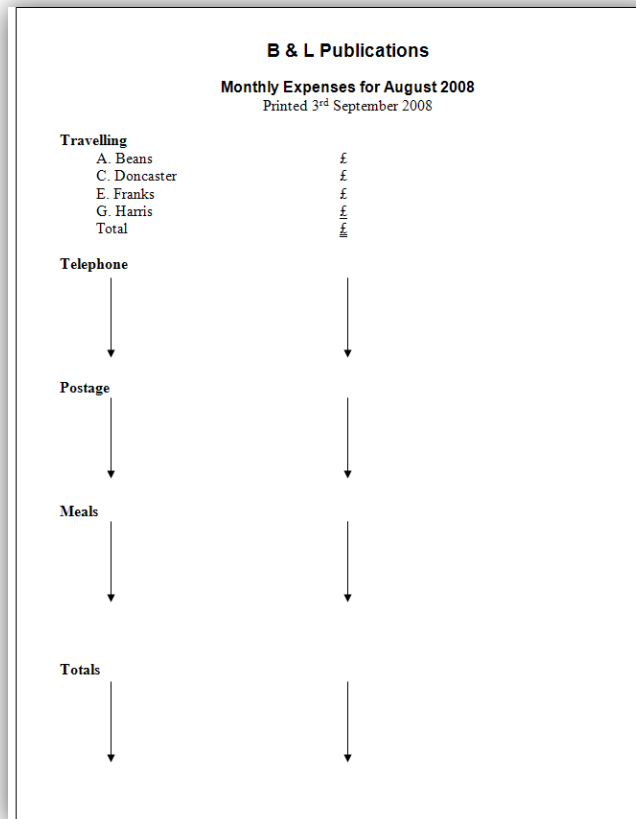


Fig. 2.1.b.4

Fig 2.1.b.4 is the design view.

B & L Publications
Monthly Expenses for August 2008
Printed 3rd September 2008

Travelling	
A. Beans	£176.48
C. Doncaster	£298.31
E. Franks	£158.23
G. Harris	£167.41
Total	<u>£800.43</u>
Telephone	
A. Beans	£ 83.49
C. Doncaster	£ 97.52
E. Franks	£ 76.48
G. Harris	£ 85.93
Total	<u>£343.42</u>
Postage	
A. Beans	£ 3.45
C. Doncaster	£ 4.95
E. Franks	£ 2.50
G. Harris	£ 3.74
Total	<u>£14.64</u>
Meals	
A. Beans	£ 89.76
C. Doncaster	£187.95
E. Franks	£ 76.45
G. Harris	£ 0.00
Total	<u>£354.16</u>
Totals	
A. Beans	£353.18
C. Doncaster	£588.73
E. Franks	£313.66
G. Harris	£257.08
Total	<u>£1512.65</u>

Fig. 2.1.b.5

We will now design a set of screens to show the information on the report of the previous example.

There are many possible answers to this question. The examiner will look for the following points and give a mark for each to a maximum.

- Title screen
- Screen for each expense and for totals
- Title screen allows movement to next screen
- Each screen, except title screen allows movement backwards and forwards (except last screen)
- Screen of contents allowing movement to each screen of expenses
- Name of business displayed
- Period displayed
- Date of publication displayed
- Every screen is well laid out
- Every screen has a heading with purpose and expense name
- There is an exit button

Figs. 2.1.b.6 to 2.1.b.8 show some of the screens. In this answer a table of contents is used to enable a user to jump to a particular screen. This is not the only solution.



Fig. 2.1.b.6

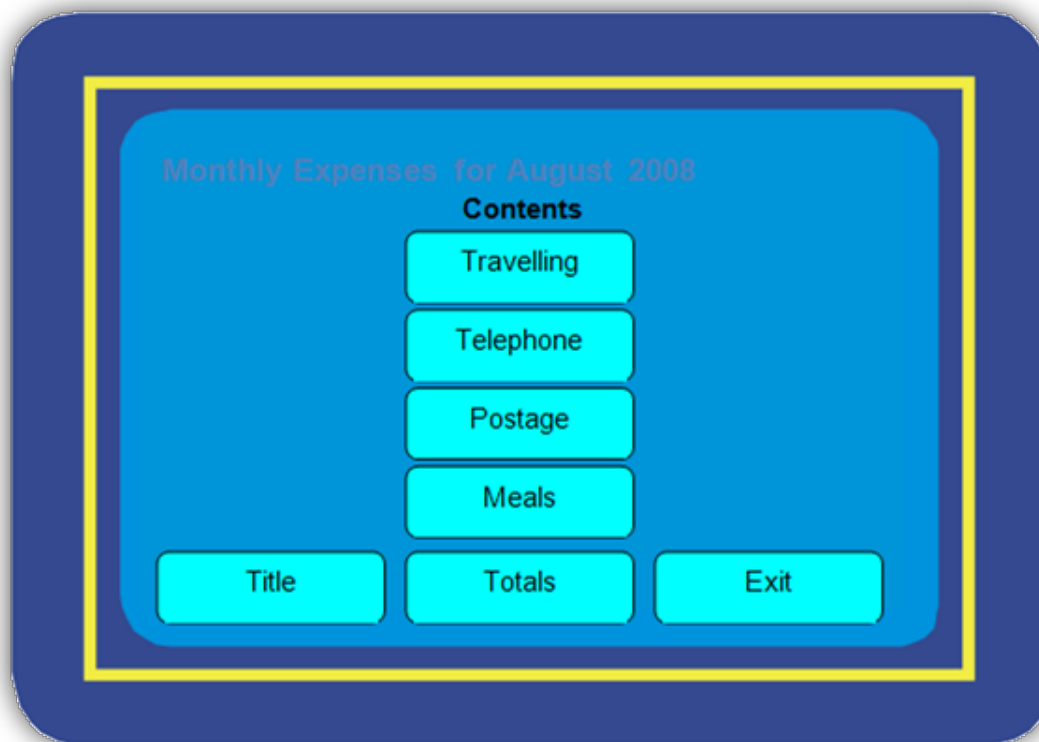


Fig. 2.1.b.7



Fig. 2.1.b.8

2.1.c *Determine the data requirements of a program.*

This section should be read in conjunction with section 2.3.a.

Whenever a program is written the programmer will have to reserve space for the data that is to be used. When doing this it is necessary to define the type of data that is to be stored. There are several data types used in programming and the types available depend on the language being used. However, the following cover all the types you are likely to need.

Integer numbers

These are whole numbers, positive or negative, and are usually used for counting. A typical example is the number of items in stock or the reorder level in a stock control system.

Real numbers

these are numbers that may have a decimal point and provide a much larger range of values than do integers. They are used when requiring measurements (which can be extremely small in atomic physics or very larger in astronomy). They are always used if the value may involve a decimal part.

Currency

These are monetary values such as the cost price of an item for sale. Note that some systems do not have this data type and they have to use real if amounts are to be stored in £'s or integer if amounts are in pence.

String/char/character/text

These are all the same and are used to store any characters as a group. Note that some languages use char for a single character and string to represent a number of characters in a group. Also, some systems allow the user to use char[n] for a group of up to n characters.

2.1.d *Explain the advantages of designing a solution to a problem by splitting it up into smaller problems (top-down/modular design).*

When a piece of software needs to be produced, the problem to be solved is likely to be far more complex than the calculation of the perimeter of a rectangle. An algorithm like that is easy for a human being to be able think about because it is so short, however, most useful problems involve such complex algorithms that it is not possible to consider them in one go. For this reason problems are normally divided up into a number of smaller problems, each one of which can be solved separately, and then combined to give a solution to the whole problem.

Consider that you have been asked to produce a solution to the problem of computerising a set of traffic lights in the centre of a town. The problem is a very complex one, but if it is divided into:

- How is the data going to be collected from sensors and stored in the system?
- How is the decision going to be made about the changing of the lights?
- How is the data collected from the pedestrian crossing and then stored..
- and processed?
- What outputs are necessary, and how are they controlled?

Then each problem becomes more manageable.

This sort of approach to problem solving is called a top-down approach, in that we started with an original problem (the top) and split the problem into smaller and smaller parts until they became manageable. When each of the individual problems has been solved the solutions are combined to produce a solution to the whole problem.

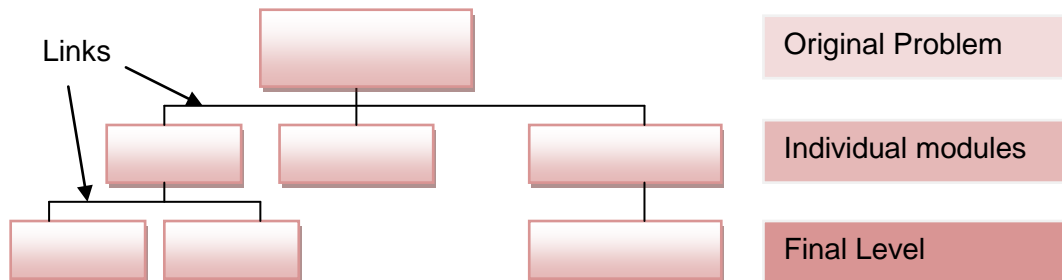
These individual parts of the solution are known as modules.

There are other advantages in this sort of approach.

- More than one person can be engaged on solving parts of the same problem simultaneously, consequently the problem should be solved more quickly.
- Different people are good at different things, so our company uses Vicki to solve the problem of collecting the information from the sensors, because Leon is not very good at that, whereas he is an expert concerning controlling things using a computer, so he does the processing module.
- Last month, our company produced a similar solution for a set of traffic lights in a different town. The sensors used were different, so Vicki still has to produce her module, and the number of roads is different so Leon will still need to come up with a solution for his module. However, the lights and other output devices being controlled are the same, so we might as well use the same module that we used last time. This happens often, that one of the modules is the same as one used previously. Because of this, software firms store the modules that they have produced so that they can be used again if the chance arises. Collections of modules like this are known as software libraries.
- Because the algorithms for the modules are so much shorter than for the whole problem, it is likely that the person producing a module will make fewer mistakes. Also, because the module is quite short, it is much easier to spot any errors that have been made, and correct them. This means that the finished software should be more reliable when it is being used.

2.1.e Produce and describe top-down/modular designs using appropriate techniques including structure diagrams, showing stepwise refinement.

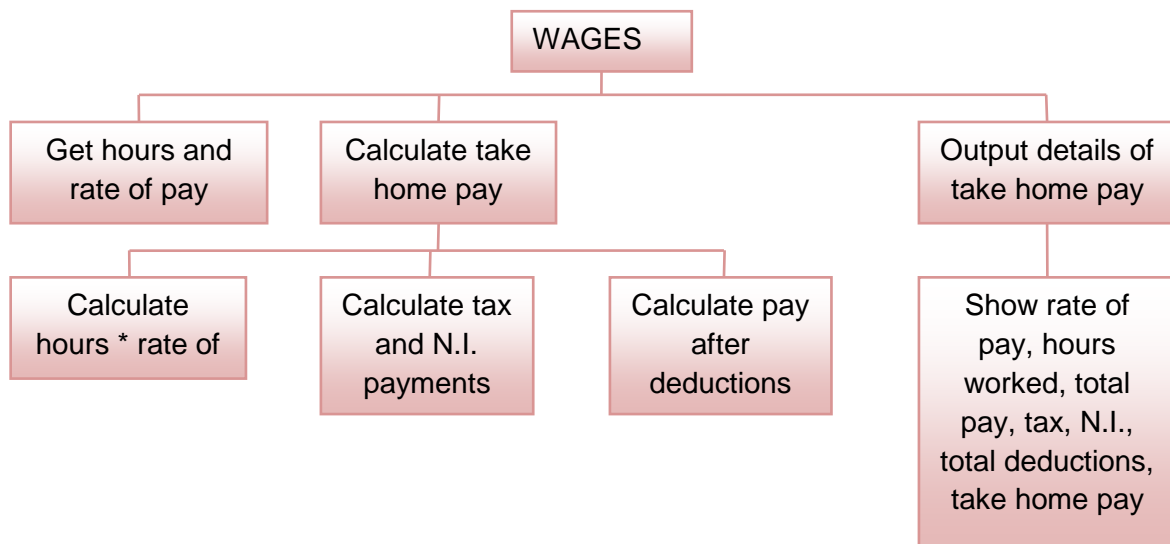
A diagram can be used to show this top-down design. It starts with the original problem as the highest level. The next, and subsequent, levels show how the problems in the previous levels are split up to form smaller, more manageable, problems. This continues until each of the blocks in the lowest levels is a self contained, easily solvable problem. These individual problems can then be solved and combined according to the links that have been used. If the links between the different blocks are then used correctly, the result will be a solution to the original problem. Imagine a diagram as a method for showing the individual modules that go to make up a solution to a problem and the relationships between them.



Some of the modules may involve repetition, this is perfectly satisfactory.

The links between the modules can be conditional. For example if the result of a module was a mark out of 100, then there could be two possible print routines, one for failure if the mark was below a certain level, and one for pass if it was above the pass mark.

Example: Given the rate of pay per hour and the number of hours worked, calculate the net pay after tax



and N.I. contributions. You may assume the tax rates are known.

This shows that the calculating of the take home pay is done in three steps. First the total pay is found, then the tax and N.I. contributions are calculated then the pay after deductions. The output has been further explained.

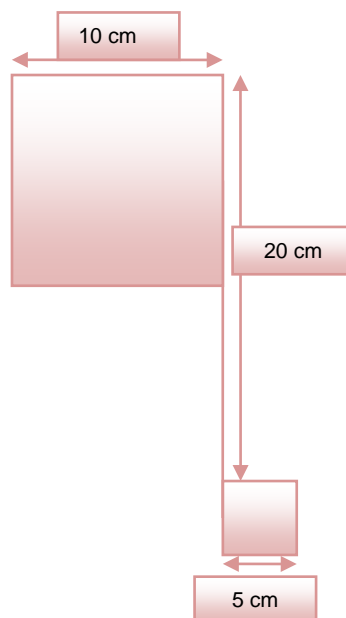
2.1.f Produce algorithms to solve problems

In order to be able to solve a problem, someone has to know what to do. In GCSE you were given instructions to solve, for example, simple equations. You were given instructions telling you how to isolate the unknown. These were algorithms! An algorithm is a set of instructions that will either solve a problem or, if there is no solution, will result in informing the user that there is no solution. Algorithms take many forms and some of these will be explained in future Sections. Here you are given a couple of simple examples.

Example: A toy can move over the floor according to commands that it is given through a keypad. As it moves it draws a line. It can obey the following instructions.

Instruction	Meaning
Forward n	Move forward n cm
Backward n	Move backwards n cm
Left n	Turn left n degrees
Right n	Turn right n degrees
Repeat n	Repeat the rest of the instruction n times

Write an algorithm for the toy to draw the diagram shown. Assume that the toy starts in the bottom left hand corner.



Repeat 4 Forward 5 Right 90
Forward 25
Repeat 3 Left 90 Forward 10

It is usual to make sure that the toy is back where it started and facing its initial direction. Rewrite this algorithm to do this.

Example: Write an algorithm to produce a cup of tea.

One solution is as follows. This is not the only solution and may not be the best solution but it works.

```
Put water in the kettle
IF it is an electric kettle
    Switch the kettle on
ELSE
    Put it on lit gas
ENDIF
Set up the cup and saucer
Put a spoon on the saucer
Get a teapot
Put tea in the teapot
IF you take milk
    Add milk to cup
ENDIF
IF you take sugar
    Put sugar to cup
ENDIF
WHILE water has not boiled
    Wait
ENDWHILE
Add water to teapot
Wait 3 minutes
Pour tea into cup
IF there is sugar in the cup
    Stir the tea
ENDIF
REPEAT
    Wait
UNTIL tea cool enough to drink
Drink tea
```

Notice the four IF statements; these are used when a decision needs to be made.

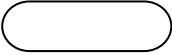
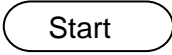

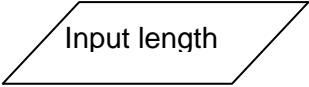

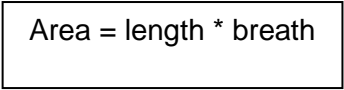
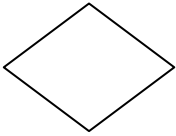
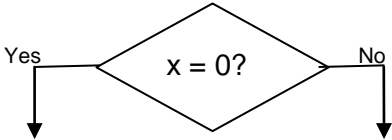

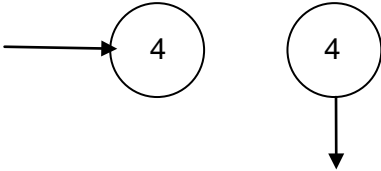
Also note the While ... Endwhile and Repeat ... Until loops.

All these constructs are used when writing algorithms and when programming.

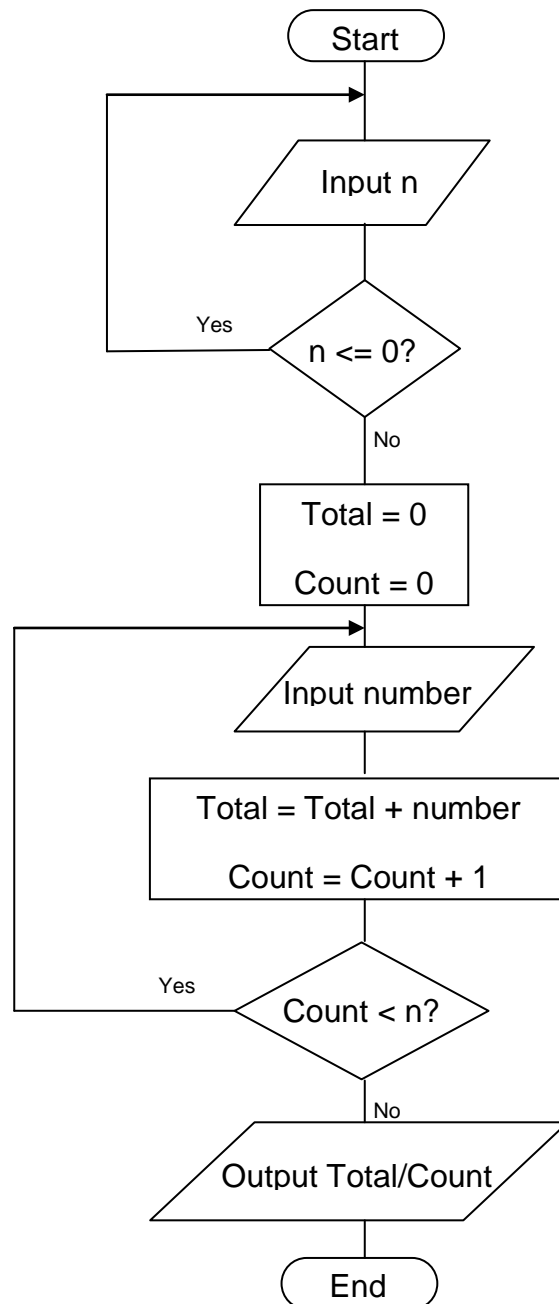
2.1.g Describe the steps of an algorithm using a program flowchart.

There are many ways of representing algorithms; which you choose to use is often a personal decision but you must be able to produce and use all the different methods. In the previous Section we saw the use of something like a programming language and ordinary English. Diagrams are often very helpful and in this Section you will see the use of program flowcharts.

Program flowcharts use the following symbols:

Symbol	Meaning	Example
	Start/Stop	
	I/O	
	Calculation	
	Decision	
	Connector	

Example: Look at the following flowchart and state what it will enable you to work out.



It tells you to read a number n and then to read n numbers, keeping a count and a running total. It then outputs the average (mean) of the numbers. Why does it first check the value of n ? What would happen if $n = 0$ or $n = -5$?

2.1.h Describe the steps of an algorithm using pseudo-code

This method will improve with practice and with the use of a high-level language. However, it is similar to structured English and uses a few constructs some of which are shown in this example which is the flowchart from the previous Section written using pseudo-code.

```
n = 0
WHILE n <= 0
    INPUT n
ENDWHILE
Total = 0
Count = 0
REPEAT
    INPUT Number
    Total = Total + Number
    Count = Count + 1
UNTIL Count = n
OUTPUT Total/Count
```

Try this out using the data

0, -5, 5, 2, 5, 4, 6, 3

You should get an output of 4.

Try writing this algorithm again but using a REPEAT loop instead of the WHILE loop and a WHILE loop instead of the REPEAT loop. Try out your solution.

Here's another example; try it out with the following two separate sets of data:

- (i) 1, -1, -6
- (ii) 2, 3, 5

```
INPUT a, b, c
d = b * b - 4 * a * c
IF d < 0 THEN
    OUTPUT "There are no real roots"
ELSE
    sr = SQRT(d)
    x1 = (-b + sr) / (2 * a)
    x2 = (-b - sr) / (2 * a)
    OUTPUT x1 and x2
ENDIF
END
```

Many more examples of these methods will appear in future Sections.

In both these examples make particular note of the indentation; this helps with reading and understanding the algorithm. The indentation shows the start and end of conditional statements. That is, statements that will only be done if a condition is TRUE. Indentation also shows the start and end of statements that are repeated.

It is sometimes useful to number the steps of an algorithm. This will be more apparent in later algorithms that have more complexity.

2.1.i Understand and implement algorithms and evaluate them by commenting on their efficiency, correctness and appropriateness for the problem to be solved

There are often many different algorithms that may be used to solve a problem. However, some are better than others. An algorithm should be efficient in that it does not keep working out the same thing many times. Look at this form of the algorithm given in the previous Section:

```
INPUT a, b, c
IF b * b - 4 * a * c < 0 THEN
    OUTPUT "There are no real roots"
ELSE
    x1 = (-b + SQRT(b * b - 4 * a * c)) / (2 * a)
    x2 = (-b - SQRT(b * b - 4 * a * c)) / (2 * a)
    OUTPUT x1 and x2
ENDIF
END
```

This is not very efficient because it expects the expression $b * b - 4 * a * c$ to be evaluated 3 times! In this, and the original, $2 * a$ has to be evaluated twice. It is much better to rewrite the algorithm to avoid this.

Now consider this algorithm:

```
INPUT n
REPEAT
    INPUT Number
    Total = Total + Number
    Count = Count + 1
UNTIL Count = n
OUTPUT Total/Count
```

This algorithm first trusts the user not to input an invalid number. It also assumes that Total and Count start at zero. This is very bad practice. Some languages assign the value zero to numbers when they are first introduced in a program but many languages do not. Always initialise any variables either by stating their starting values or by assigning them a value before using them on the right-hand side of statements.

Example: Write an algorithm to input the lengths of three sides of a triangle called a, b and c. Calculate and output the area of the triangle using the following formulae.

$$\text{area} = \sqrt{s(s-a)(s-b)(s-c)}$$

where

$$s = (a + b + c)/2$$

Use the following sets of data to test your algorithm.

(i) 3, 4, 5 (Area = 6)

(ii) 4, 8, 10 (Area = 15.199)

(iii) 4, 10, 16 (Error)

(iv) What happens if you enter a negative length?

(v) Is your algorithm efficient?

Many more algorithms will appear in future Sections, try to improve them!

2.1.j Describe the use of Rapid Application Development (RAD) as a design strategy, including prototyping and iterative development, and state its advantages and disadvantages

With technology changing rapidly it is important that projects are also developed quickly. One of the problems with traditional development methods is that they often restrict what can be done and solutions have a linear development. Consider the traditional software life cycle. Rapid Application Development (RAD) uses the spiral model to develop solutions. This method allows the developer to partially develop a solution and to continually refine the solution until the user is happy with the solution.

A simple way of developing an initial solution is to first develop a prototype. This is a partial solution that can be used to check with the end user that all that is needed has been clearly defined. Consider developing a database. This can be very sophisticated and have considerable flexibility and functions. A prototype could be developed using a visual programming language such as Visual Basic. The various forms could be developed together with buttons allowing the user to add, modify and delete data. In fact, when a button is clicked to code behind it 'pretends' to do what the developer thinks the user wants. It does not actually do it. This means that designs can be quickly changed according to the user's requirements.

The above screens have been designed in a few minutes. You can now take them to the user for comments. You should be able to easily criticise these screens. However, because they were created with VB, it is possible to make the buttons 'work'. If the user clicks on Cancel the forms should disappear (and a menu should appear, say). If the user clicks on Clear, the boxes should clear. Clicking Next -> would hide the first screen and show the second. Clicking <- Previous would close the second screen and open the first screen. Finally, clicking Add would add the data to the database. Thus it appears that the first part of the solution is working; in fact, it is only simulating the database. The advantage is that changes can be quickly and easily made until the user is happy with the results.

Comment on the design explaining what needs to be changes. As a hint, look at the title of the screens, where are the instructions telling the user what to do, consider the layout of the buttons and the colour scheme. Also consider the minimise, resize and close buttons at the top of the screen. Should the user be able to use them in the final database solution?

The rest of the database can be developed in this way so that the user can get a feel for the final product. However, this prototype doesn't actually work! Consider the prototype for a car body, usually made in wood!

When the prototype has been fully developed, the developer must create the real solution. This may be in the same language as the prototype but usually uses different software. For example, the creation of a database may well use database software rather than VB.

When will RAD work? There are many schools of thought but the following may give you an idea of different approaches.

- The success depends on the users and the developers working well together. It also relies on the developer being able to make decisions without having to keep referring to line managers.
- It is essential that testing is considered to be part of the iterative cycle as this will make sure that the developer will deliver what the user requires.
- Each stage must be considered for its fitness to do the job. Less reliance is placed on the requirements specification which may be incorrect or incomplete.
- If something is changed it is assumed that the developer can revert back to a previous part of the solution.
- It is assumed that incremental delivery is acceptable.

There are both advantages and disadvantages to using RAD.

Advantages:

- A good idea as to the format of the solution can be developed very quickly.
- There is plenty of interaction between developer and user.
- This will lead to a better understanding of what is required by the user.
- The user is continually involved in the development leading to a more accurate solution with less need for maintenance due to errors and omissions.
- This will reduce the cost of the project because the end product will satisfy the user's needs.

Disadvantages:

- It is essential that the user and developer can communicate well.
- The user and developer must be amicable.
- The user can find time to keep seeing the developer and is fully involved.
- When the prototype is what the user wants in the end product, the user expects the product to be available almost immediately.
- The finished product is not a working solution.

2.1 Example Questions



1 A user wishes to use a database to keep details of customers. State **three** features that the designer of the software should consider in order to produce a good interface.

- A
- It should be natural to know what to do next.
 - On-screen instructions should ensure this is the case.
 - Navigation between screens should be clear.
 - On screen forms should have the same design as input forms.
 - Each screen has a similar layout.
 - Few fonts are used.
 - The skills of the user.

(3)

Also see the end of Section 2.1(a)

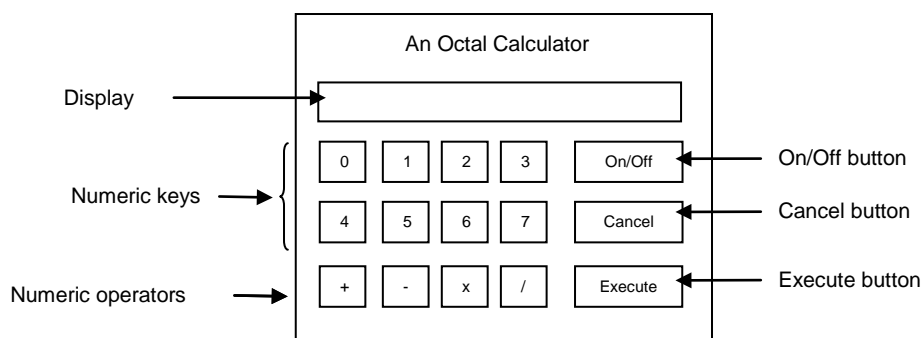
2 A touch screen interface is required for an octal calculator that can perform the four arithmetic functions (addition, subtraction, multiplication and division). It is intended that a user enters the first number, chooses the operation, enters the second number and finally indicates that the calculation is to be executed.

Design a suitable interface.

A First list what will be required:

- A title.
- A box to display the numbers
- Buttons to enter the digits 0 to 7
- A button for each operation.
- A cancel button.
- An on/off button.
- An execute button.

The next step is to draw a labelled diagram.



Marks are given for the contents of the screen and for the layout.

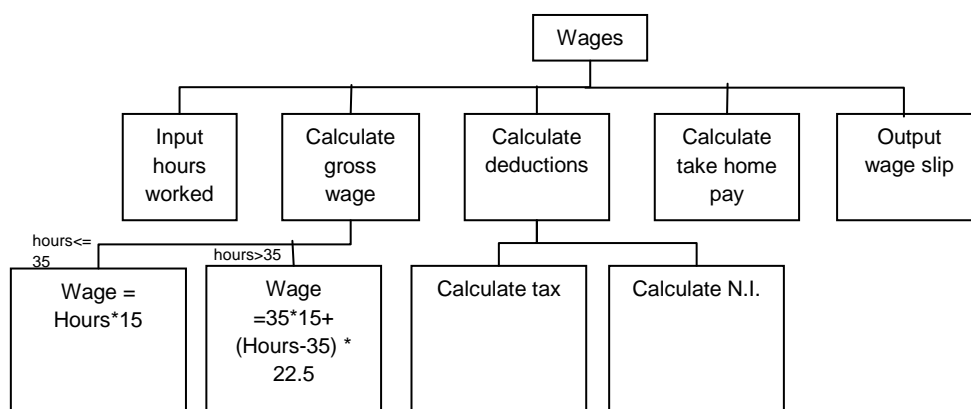
(7)

Diagrams should be labelled but can be drawn freehand.

3 Give **four** advantages of using top-down design to solve a problem.

- A
- More than one person can be engaged on solving parts of the same problem.
 - The problem should be solved more quickly.
 - Different people are good at different things, so each programmer can do what they do best.
 - Modules can be used on many problems.
 - Fewer errors because modules are relatively short.
 - Easier to find errors and correct them in shorter modules.
 - This means that the finished software should be more reliable when it is being used.

4 Consider the following top-down diagram, and then explain what it shows.



- A
- It shows how to calculate wages.
 - It inputs hours worked
 - Calculates gross wage
 - If hours ≤ 35 wage = hours * 15
 - Else it is hours * 35 + (hours - 35) * 22.5
 - Calculates deductions
 - Calculates tax
 - and N.I.
 - Calculates take home pay
 - and outputs a wage slip

(7)

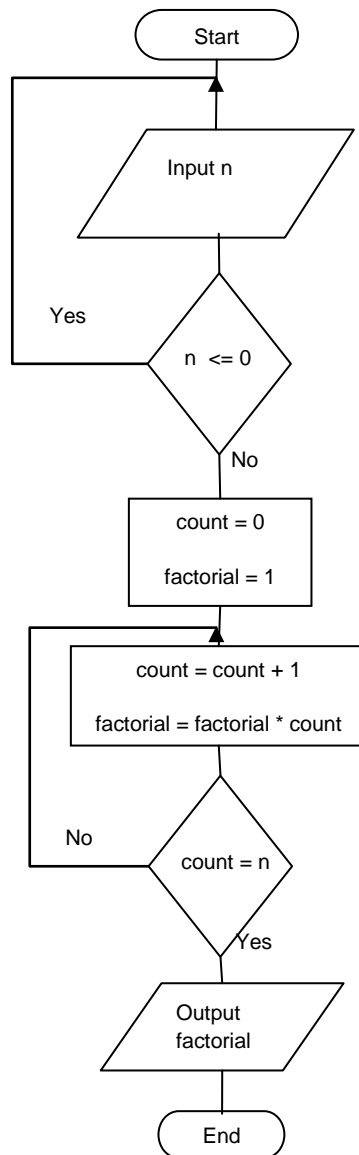
5 Draw a program flowchart to solve the following problem.

Read the value of an integer n and output n and $n!$ where

$$n! = 1 \times 2 \times 3 \times 4 \times \dots \times n$$

For example $5! = 1 \times 2 \times 3 \times 4 \times 5$

A



(8)

6 Repeat the above question using pseudo-code.

A There are more than one solution to this, but one solution is

```
REPEAT
  READ n
UNTIL n > 0
count = 0
factorial = 1
REPEAT
  count = count + 1
  factorial = factorial * count
UNTIL count = n
OUTPUT factorial
```

(8)

7 Explain the meaning of

(i) a prototype,

(ii) iterative development

in Rapid Application Development (RAD).

A (i) A prototype is a representation of a solution that shows what the final solution will look like. It has very little implementation of the solution. It is useful in RAD as it can be very quickly produced.

(2)

(ii) The procedure is iterative because the developer keeps returning to the user for comments. In the light of these comments the prototype can be quickly amended and the user consulted again. This procedure can be repeated until the solution required is clearly defined. At this stage the real solution can be designed and produced.

(4)



2.2 The Structure of Procedural Programs

2.2.a Define and correctly use the following terms as they apply to procedural programming: statement, subroutine, procedure, function, parameter/argument, sequence, selection, iteration/repetition, loop

This part of the Specification refers to procedural languages. These are languages that require the programmer to specify precisely how a computer is to solve a problem. They consist of a number of instructions that must be obeyed in the order specified. There are a large number of these languages which have been designed to solve different types of problem. For example, COBOL (Common Business Oriented Language) uses terms that are familiar to administrators; FOTRAN (FORmula TRANslation) was developed to solve mathematical/scientific problems; PASCAL (named after the mathematician Pascal) was developed as a teaching language; BASIC (Beginners All-purpose Symbolic Instruction Code) was developed so that as many people as possible could write computer programs and there are many others. All these languages have developed over the years and have more and more facilities.

This text does not teach you how to program in a particular language. Each Centre can choose its own language for use in the examination and in the project. There are numerous books on the market which provide all that is necessary to learn programming. However, many examples of programs will be given in this Section and we shall use the Visual Basic.NET programming language. These examples will be used to explain the theory of programming in a procedural language.

A *statement* in a programming language is simply an instruction that can be executed. Each statement can be simple or complex. The simplest statement is the assignment statement which takes the form

`<variable> = <arithmetic expression>`

or

`<variable> = <logical expression>`

In this notation, anything in angle brackets has to be replaced by a valid expression in the programming language being used.

A typical arithmetic expression is

`mass / volume`

and a typical logical expression is

`count < 10`

Thus the following are valid assignment statements.

`density = mass / volume`

`notFinished = count < 10`

A *subroutine* is a self-contained block of code that has a name and is executed by another program. It comes in two forms. The first is called a *procedure*. This can receive values from another program and can return no, one or many values back to that program. The other form is called a *function*; this can also receive values from another program but returns only one value back. Much more detail about subroutines, procedures and functions will be given in later Sections.

When passing values between the main program and a subroutine, the values must be given variable names. When the main program calls a subroutine, the values being passed to the subroutine are called *arguments*. In the subroutine, the values are called *parameters*.

In a program, a *sequence* is a set of instructions that are executed in the order in which they are written. There are no branches or loops.

E.g. Write a set of instructions to find the average of three numbers:

```
Dim num1, num2, num3, average As Double

num1 = InputBox("Enter first number", "First Number")
num2 = InputBox("Enter second number", "Second Number")
num3 = InputBox("Enter third number", "Third Number")

average = (num1 + num2 + num3) / 3

MsgBox("The average is " & CStr(average), , "Average")
```

At this stage you do not need to know the exact meaning of each instruction, just understand that each instruction is executed one after the other until the end is reached. All code in this book will be in Visual Basic.NET, but similar examples can be found in any high-level procedural language.

All high-level languages allow *selection* which means that the computer has to decide which instructions to do according to some criterion. There are several forms of selection (which will be explained later) but here is a simple example.

E.g. Write a program to read two numbers and to output which is the larger or that they are equal:

```
Dim num1, num2 As Double

num1 = InputBox("Enter first number.", "first Number")
num2 = InputBox("Input second number", "Second Number")

If num1 < num2 Then
    MsgBox("The second number is the larger.", , "Result")
Else
    If num1 > num2 Then
        MsgBox("The first number is the larger.", , "Result")
    Else
        MsgBox("The two numbers are equal", , "Result")
    End If
End If
```

In this sequence of instructions, if the first number is less than the second, then the instruction

```
MsgBox("The second number is the larger.", , "Result")
```

will be executed. The remaining instructions between the words If and EndIf will be ignored.

If the first number is greater than the second, the the instruction

```
MsgBox("The first number is the larger.", , "Result")
```

will be executed. The remaining instructions between the words If and EndIf will be ignored.

Finally, if neither of these conditions is true, the two numbers are equal and the instruction

```
MsgBox("The two numbers are equal", , "Result")
```

will be executed. The remaining instructions between the words If and EndIf will be ignored.

Finally we shall see an example of a *loop*. This is a set of instructions that are executed a finite number of times according to some condition. You have seen examples of loops when you looked at algorithms in the previous Section. The following example will show a different form of loop as well as the REPEAT loop already discussed with algorithms. Because the instructions are repeated, these loops are called *iteration/repetition* constructs.

E.g. Write a program to read an integer, n, and then to output the squares of the first n integers.

Here is the code, notice VB calls it a DO loop not a REPEAT loop, but it works the same. Notice the new loop called a FOR loop.

```
Dim num As Integer

Dim square As Integer

Dim count As Integer

Do

    num = InputBox("Enter a positive integer.", _
        "Number of repetitions")

Loop Until num > 0

lstResults.Items.Clear()

For count = 1 To num

    square = count * count

    lstResults.Items.Add(CStr(count) & " " & CStr(square))

Next count
```

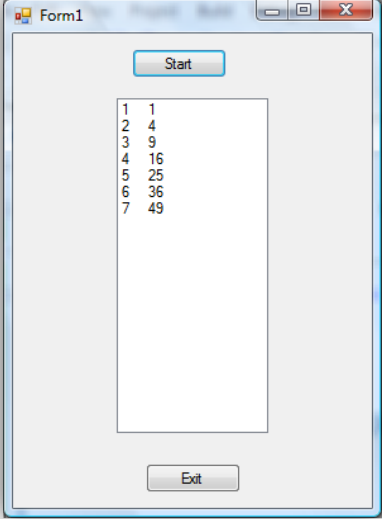
The DO loop makes the computer keep asking for the value of num until the user enters a positive number. What would happen if the user entered 6.8 or 5.3? You will learn later how to avoid this problem.

The computer then repeats the two instructions

```
square = count * count
```

```
lstResults.Items.Add(CStr(count) & " " & CStr(square))
```

the required number of



A screenshot of a Windows application window titled "Form1". The window contains a "Start" button at the top and an "Exit" button at the bottom. In the center, there is a list box displaying the following text:

1	1
2	4
3	9
4	16
5	25
6	36
7	49

times. A typical result is:

More details of these constructs will be given in the following Sections.

2.2.b Identify the three basic programming constructs used to control the flow of execution, ie sequence, selection and iteration

Read the following code and identify

- i. a sequence,
- ii. repetition,
- iii. selection,

```
1. Dim num As Double

2. Dim posCount, negCount As Integer

3. num = InputBox("Enter a number (zero to end)", "Next Number")

4. negCount = 0

5. posCount = 0

6. Do While num <> 0

    i. If num < 0 Then
    ii.     negCount = negCount + 1
    iii. Else
    iv.     posCount = posCount + 1
    v. End If
    vi. num = InputBox("Enter a number (zero to end)", _
                        "Next Number")

7. Loop

8. With lstResult.Items

    i. .Clear()
    ii. .Add("The number of negative numbers is " & CStr(negCount))
    iii. .Add("The number of positive numbers is " & CStr(posCount))

9. End With
```

- (i) The instructions 1 to 5 are executed in sequence as are the instructions 8 to 9
- (ii) The instructions 6 and 7 create a loop so that all the instructions from 6 to 8 are repeated.
- (iii) The instructions i to v inside the 6-7 loop form a selection construct.

2.2.c Understand and use selection in pseudocode and a procedural programming language, including the use of IF statements and CASE/SELECT statements

You have seen the IF statement used in pseudocode and in VisualBasic.NET (a procedural language). In this Section we will put the two together to show good programming practice. We shall also look at another selection construct called the CASE/SELECT construct.

E.g: Write an algorithm and a program to read an examination mark and to output the Grade according to the following rules

MARK	GRADE
0 – 39	FAIL
40 – 59	PASS
60 - 79	CREDIT
80 -100	DISTINCTION

All marks are out of 100.

There are many solutions to this problem; probably the following is the simplest to write:

```
READ mark
IF mark < 40 THEN grade = "FAIL"
IF mark >= 40 AND mark < 60 THEN grade ="PASS"
IF mark >= 60 AND mark < 80 THEN grade ="CREDIT"
IF mark >= 80 THEN grade = "DISTINCTION"
OUTPUT grade
```

The equivalent VB code is:

```
Dim mark As Integer

Dim grade As String


mark = InputBox("Enter the mark out of 100. ", "Examination Mark")


If mark < 40 Then grade = "FAIL"

If mark >= 40 And mark < 60 Then grade = "PASS"

If mark >= 60 And mark < 80 Then grade = "CREDIT"

If mark >= 80 Then grade = "DISTINCTION"


MsgBox("The grade is " & grade, , "GRADE")
```

What happens if the user enters 101 or -40 as the mark?

Clearly this program wants improving; look at this version:

```
REPEAT
    READ mark
UNTIL mark >= 0 AND mark <= 100
IF mark < 40 THEN grade = "FAIL"
IF mark >= 40 AND mark < 60 THEN grade = "PASS"
IF mark >= 60 AND mark < 80 THEN grade = "CREDIT"
IF mark >= 80 THEN grade = "DISTINCTION"
OUTPUT grade

    Dim mark As Integer

    Dim grade As String

Do

    mark = InputBox("Enter the mark out of 100. ", _
                    "Examination Mark")

Loop Until mark >= 0 And mark <= 100

If mark < 40 Then grade = "FAIL"

If mark >= 40 And mark < 60 Then grade = "PASS"

If mark >= 60 And mark < 80 Then grade = "CREDIT"

If mark >= 80 Then grade = "DISTINCTION"

MsgBox("The grade is " & grade, , "GRADE")
```

Now look at this algorithm:

```
REPEAT
    READ mark
UNTIL mark >= 0 AND mark <= 100
IF mark < 40 THEN
    grade = "FAIL"
ELSE
    IF mark >= 40 AND mark < 60 THEN
        grade = "PASS"
    ELSE
        IF mark >= 60 AND mark < 80 THEN
            grade = "CREDIT"
        ELSE
            grade = "DISTINCTION"
        ENDIF
    ENDIF
ENDIF
ENDIF
```

and this program:

```
Dim mark As Integer
Dim grade As String

Do

    mark = InputBox("Enter the mark out of 100. ", "Examination Mark")

Loop Until mark >= 0 And mark <= 100

If mark < 40 Then

    grade = "FAIL"

Else

    If mark >= 40 And mark < 60 Then

        grade = "PASS"

    Else

        If mark >= 60 And mark < 80 Then

            grade = "CREDIT"

        Else

            grade = "DISTINCTION"

        End If

    End If

End If

MsgBox("The grade is " & grade, , "GRADE")
```

Which is the more efficient?

In the first solution, every IF statement has to be checked. In the second solution, as soon as a TRUE answer is found, the assignment of grade is made and all other IF statements are ignored.

However, if we had more comparisons to make, this solution could become very difficult to understand. Here is an alternative solution:

```
REPEAT
    READ mark
UNTIL mark >= 0 AND mark <= 100
SELECT CASE mark
    CASE 0 TO 39
        grade = "FAIL"
    CASE 40 TO 59
        grade = "PASS"
    CASE 60 TO 79
        grade = "CREDIT"
    CASE >= 80
        grade = "DISTINCTION"
END SELECT
OUTPUT grade
```

```
Dim mark As Integer

Dim grade As String

Do

    mark = InputBox("Enter the mark out of 100. ", _
        "Examination Mark")

Loop Until mark >= 0 And mark <= 100

Select Case mark

    Case Is < 40

        grade = "FAIL"

    Case 40 To 59

        grade = "PASS"

    Case 60 To 79

        grade = "CREDIT"

    Case Is >= 80

        grade = "DISTINCTION"

End Select

MsgBox("The grade is " & grade, , "GRADE")
```

This version is efficient and easy to read.

More details of good programming practice will be given in Section 2.5 and testing strategies are in Section 2.6. The meaning of further programming terms are given in Sections 2.3 and 2.4.

2.2.d *Understand and use iteration in pseudocode and a procedural programming language, including the use of count-controlled loops (FOR-NEXT loops) and condition controlled loops (WHILE-ENDWHILE and REPEAT-UNTIL loops)*

One iterative construct we have not looked at is the FOR-NEXT construct. There are two forms of this construct. The first takes the form:

```
FOR <variable> = <start value> TO <end value>
    <series of program statements>
NEXT
```

The NEXT part of the construct can be followed by the name of the loop variable defined in the FOR part of the statement. An example is:

```
FOR count = 1 TO 10
    OUTPUT count, count * count
NEXT
```

Or:

```
FOR count = 1 TO 10
    OUTPUT count, count * count
NEXT count
```

In these examples the first value of count is 1, it then increments in steps of 1 until it reaches 10, which is used. The computer then exits the loop. Here is the corresponding VB:

```
Dim count As Integer

For count = 1 To 10
    lstResults.Items.Add(CStr(count) & "    " & CStr(count * count))
Next
```

The alternative version is:

```
Dim count As Integer

For count = 1 To 10
    lstResults.Items.Add(CStr(count) & "    " & CStr(count * count))
Next count
```

A typical result is shown below:

A screenshot of a Windows application window titled 'Form1'. The window contains a 'Start' button at the top and an 'Exit' button at the bottom. In the center, a list box displays the following text:

1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

The incremental steps need not be 1, that is the default value. For example to output the squares of the even numbers between 2 and 10 use:

```
Dim count As Integer

For count = 2 To 10 Step 2

    lstResults.Items.Add(CStr(count) & "    " & CStr(count * count))

Next count
```

A screenshot of a Windows application window titled 'Form1'. The window contains a 'Start' button at the top and an 'Exit' button at the bottom. In the center, a list box displays the following text:

2	4
4	16
6	36
8	64
10	100

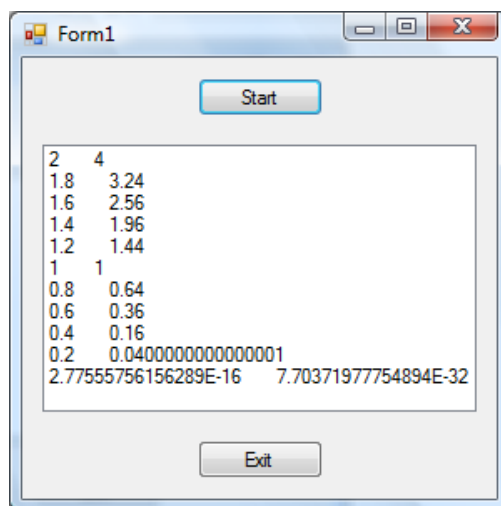
It is also possible to go up, or down, in fractional steps. Look at the result of running this code:

```
Dim count As Double

For count = 2 To 0 Step -0.2

    lstResults.Items.Add(CStr(count) & "      " & CStr(count * count))

Next count
```



Particularly note the last two answers. This is because you are using fractional numbers. IN Unit 3 you will learn that the number 0.0 cannot be held in a computer. The actual value will be very close to 0 but will not equal it. Great care needs to be taken when using fractional numbers as they are not held exactly as can be seen by the last but one answer.

Examples of WHILE-ENDWHILE and REPEAT-UNTIL loops have already been given in the text.

2.2.e Understand and use nested selection and iteration statements

Look at the following code which was given in Section 2.2(c):

```
If mark < 40 Then
    grade = "FAIL"
Else
    If mark >= 40 And mark < 60 Then
        grade = "PASS"
    Else
        If mark >= 60 And mark < 80 Then
            grade = "CREDIT"
        Else
            grade = "DISTINCTION"
        End If
    End If
End If
```

It consists of three IF statements which are nested inside one another. This is emphasised by indenting the code to make it clear that the statements are nested. The IF statements must not overlap.

Loops can also be nested

E.g. Read a student's name and the mark for each examination. Output the average mark for each student. The data is to take the form student's name followed by the examination marks. The list of marks for each student is terminated by the word "END" and the list of marks for each student is terminated by a negative mark.

The test data to be used is:

```
ROGER, 56, 48, 30, 89, -1
MARY, 78, 67, 97, -1
BRIAN, 67, 56, 78, 88, 45, -1
END
```

The algorithm is:

```
READ name
WHILE name <> "END"
    sum = 0
    count = 0
    READ mark
    WHILE mark >= 0
        sum = sum + mark
        count = count + 1
        READ mark
    ENDWHILE
    average = sum / count
    OUTPUT "The average mark for ", name, " is ", average
    READ name
ENDWHILE
```


The VB code for this is:

```
Dim mark, sum, count As Integer

Dim average As Double

Dim name As String

name = InputBox("Enter name of stdent (END to terminate.)", "Name")

Do While name.ToUpper <> "END"
    sum = 0
    {
        count = 0
        mark = CInt(InputBox("Enter next mark (-1 to end).", "Mark"))
        Do While mark >= 0
            sum = sum + mark
            count = count + 1
            mark = CInt(InputBox("Enter next mark (-1 to end).", _
                                "Mark"))
        Loop
    }
    average = sum / count

    MsgBox("The average mark for " & name & " is " & _
           CStr(average), , "Average Mark")

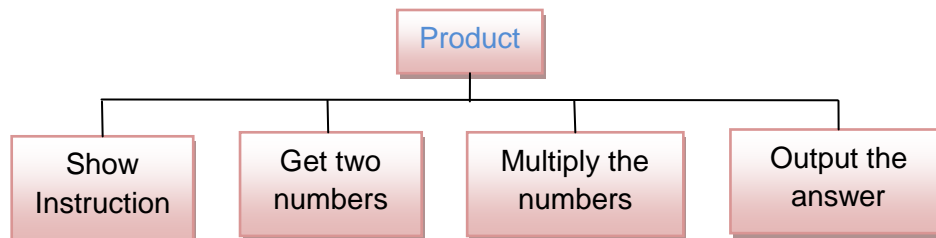
    name = InputBox("Enter name of stdent (END to terminate.)", _
                    "Name")
Loop
```

Notice that, in both the algorithm and the code, one loop is completely inside the other loop.

2.2.f Understand, create and use subroutines (procedures and functions), including the passing of parameters and the appropriate use of the return value of functions

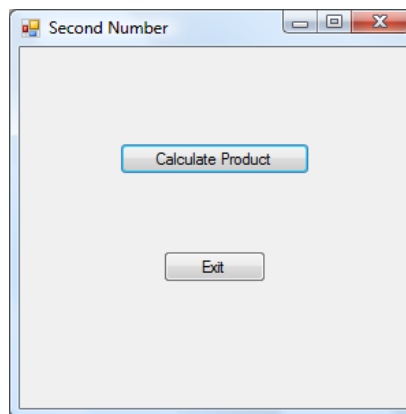
Procedures are used to split a problem into simpler problems in order to solve a complex problem. A procedure has a name, performs a task and is a program in its own right.

Consider a problem that expects the user to enter two numbers and to output their product. One possible solution is shown by the diagram below.



One solution is shown below.

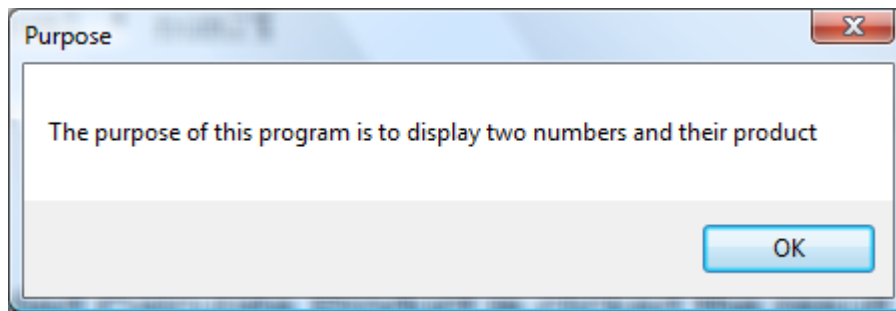
Interface:



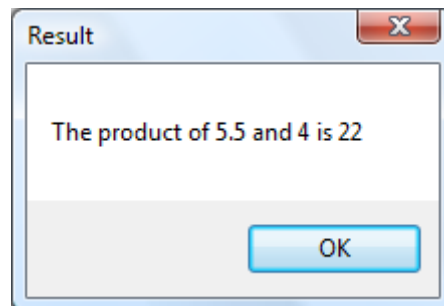
Code:

```
Private Sub btnCalculate_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles btnCalculate.Click  
  
    Dim num1, num2 As Double  
  
    Dim product As Double  
  
    MsgBox("The purpose of this program is to display two numbers and their  
product", , "Purpose")  
  
    num1 = 5.5  
    num2 = 4.0  
    product = num1 * num2  
  
    MsgBox("The product of " & num1 & " and " & num2 & " is " _  
        & product, , "Result")  
  
End Sub
```

When the button labelled Calculate Product is clicked the result is:



followed by:



Now we shall write a procedure called purpose to output the purpose of the program and another procedure called result to display the result of the calculation. The interface is the same as before.

The code is (the line numbering is so that the code can be explained, it is not part of the code):

```
1 Private Sub btnCalculate_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles btnCalculate.Click  
  
2     Dim num1, num2 As Double  
3     Dim product As Double  
  
4     ExplainPurposeOfProgram()  
  
5     num1 = 5.5  
6     num2 = 4.0  
  
7     product = num1 * num2  
  
8     OutputResult(num1, num2, product)  
  
9 End Sub
```

```

10  Sub ExplainPurposeOfProgram()

11      MsgBox("The purpose of this program is to display two numbers and
          their product", , "Purpose")

12  End Sub

13  Sub OutputResult(ByVal firstNumber, ByVal secondNumber, ByVal result)

14      MsgBox("The product of " & firstNumber & " and " & secondNumber _
          & " is " & result, , "Result")

15  End Sub

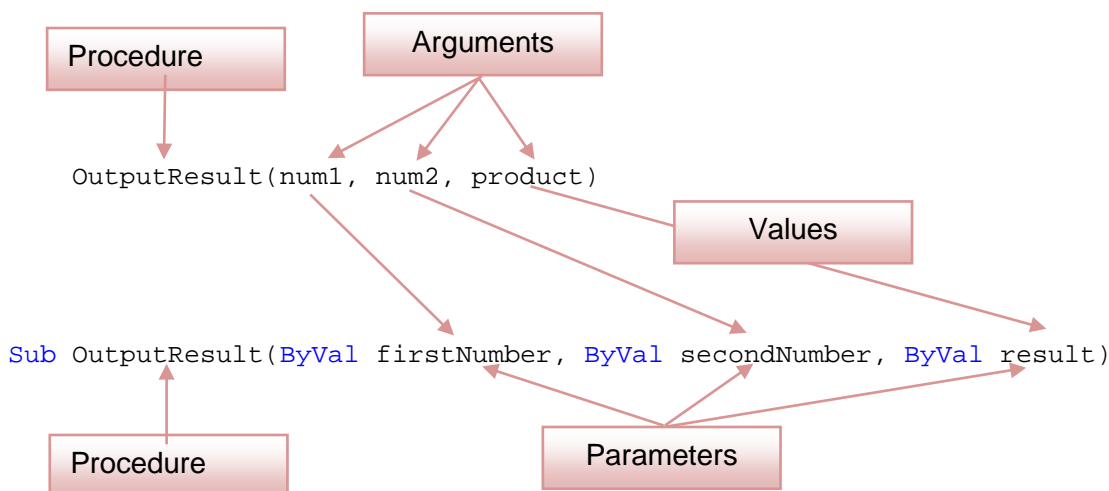
```

The meaning of this code will be explained in detail in future Sections of this Module. Here we concentrate of calling and using procedures.

Line 4 calls the procedure called `ExplainPurposeOfProgram`. This passes control to line 10 and the instruction at line 11 is executed. Line 12 passes control to the instruction immediately after the calling instruction; that is, to line 5.

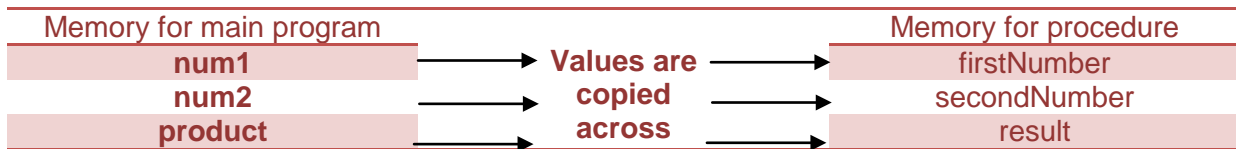
Instructions on lines 5, 6 and 7 are now executed. The instruction on line 8 calls the procedure `OutputResult`. This is slightly different to the previous call as the name is followed by three **arguments**. Now look at line 13 which is executed next. The name is followed by three **parameters**. The word `ByVal` will be explained in a moment.

The computer pairs off the arguments with the parameters:



When the instruction at line 8 is executed, the values of `num1`, `num2` and `product` are passed to the variables `firstNumber`, `secondNumber` and `result` respectively. This is called *passing by value*. Hence the word `ByVal` in front of the parameters.

Diagrammatically, this can be represented as shown below.



The memory for main program can only be accessed by the main program, not by the procedure. The memory for the procedure can only be accessed by the procedure, not the main program.

So far we have only passed values one way. Now look at the following code:

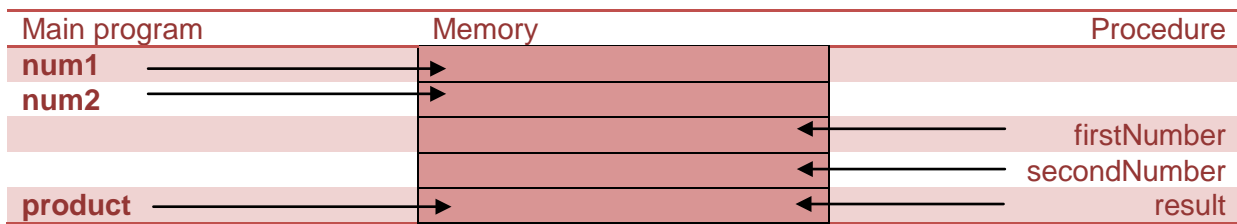
```

1 Private Sub btnCalculate_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnCalculate.Click
2     Dim num1, num2 As Double
3     Dim product As Double
4     ExplainPurposeOfProgram()
5     num1 = 5.5
6     num2 = 4.0
7     calcProduct(num1, num2, product)
8     OutputResult(num1, num2, product)
9 End Sub
10 Sub ExplainPurposeOfProgram()
11     MsgBox("The purpose of this program is to display two numbers and
        their product", , "Purpose")
12 End Sub
13 Sub OutputResult(ByVal firstNumber, ByVal secondNumber, ByVal result)
14     MsgBox("The product of " & firstNumber & " and " & secondNumber _
        & " is " & result, , "Result")
15 End Sub
16 Sub calcProduct(ByVal firstNumber, ByVal secondNumber, ByRef result)
17     result = firstNumber * secondNumber
18 End Sub

```

This version gives exactly the same results as the previous version, but the calculation of the product is done in the procedure `calcProduct` instead of in the main body of the program. In lines 5 and 6, the values of `num1` and `num2` are assigned as before. However, line 7 calls the procedure `calcProduct` which is defined in lines 16 to 18. Notice the change in the definition of the parameters in line 16. The first two are the same, but the third is defined as `ByVal`. This means that the address of this parameter is sent to the procedure not the value.

Thus, the variable `product` in the main program and the variable `result` in the procedure use the same memory as shown below.



Notice how `product` and `result` point to the same memory location.

If only one value needs to be returned to the main program, a function can be used.

The following code is an alternative solution to the above problem. `calcProduct` is replaced by the function `calcProduct`. This function only requires two parameters, the two numbers. The result is returned via the function name. This can be seen in line 7.

```

1  Private Sub btnCalculate_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnCalculate.Click

2      Dim num1, num2 As Double
3      Dim product As Double
4      ExplainPurposeOfProgram()
5      num1 = 5.5
6      num2 = 4.0
7      product = calcProduct(num1, num2)
8      OutputResult(num1, num2, product)
9  End Sub

10 Sub ExplainPurposeOfProgram()
11     MsgBox("The purpose of this program is to display two numbers and
        their product", , "Purpose")
12 End Sub

13 Sub OutputResult(ByVal firstNumber, ByVal secondNumber, ByVal result)
14     MsgBox("The product of " & firstNumber & " and " & secondNumber _
        & " is " & result, , "Result")
15 End Sub

16 Function calcProduct(ByVal firstNumber, ByVal secondNumber)
17     Return firstNumber * secondNumber
18 End Function

```

Notice how the result is returned by the function in line 17 and used by the main program in line 7.

Any subroutine (procedure or function) can contain many statements just as any other program. Each subroutine does a specific task. Thus a complex task can be broken down into simple sub-tasks, each of which can be replaced by a subroutine.

2.2.g Identify and use recursion to solve problems; show an understanding of the structure of a recursive subroutine including the need for a stopping condition

Consider calculating 'factorial 8' This written as 8! and is defined as:

$$8! = 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1$$

Now consider factorial 7, written as:

$$7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1$$

Comparing the two definitions we see that:

$$8! = 8 \times 7!$$

0! and 1! are both defined as 1. Thus, one solution is the one shown below which uses a simple loop to calculate the value of a number entered by the user:

```
1      Do
2          num = InputBox("Enter an integer number.", "Input Data")
3      Loop Until num >= 0

4      If num = 0 Or num = 1 Then
5          fact = 1
6      Else
7          fact = 1
8          count = num
9          Do
10             fact = fact * count
11             count = count - 1
12         Loop Until count = 0
13     End If

14     MsgBox("Factorial " & num & " is " & fact, , "Result")
```

Lines 1 to 3 ensure that a positive number is entered. Lines 4 and 5 check for a 0 or 1, and then set fact = 1. If the number entered is not 0 or 1, then lines 7 to 12 are executed. fact 1 and count are initialised and a loop calculates factorial n.

Now look at 8! again.

$$8! = 8 \times 7!$$

This means that if we can find 7! we only need to multiply it by 8 to find 8!

Similarly, if n is a positive integer,

$$n! = n \times (n - 1)!$$

In programming terms, we can write:

$$\text{factorial}(n) = n * \text{factorial}(n - 1)$$

Let's define a function called factorial; it would look something like this

```
Function factorialn(ByVal number As Integer)
    Return number * factorialn(number - 1)
/
End Function
```

Why won't this work?

It will go on forever because there is nothing to tell the computer when to stop. In this case, we know that $1! = 1$, so as soon as the computer has a value of 1, it knows that it can return 1 as the result. Details of how this works will be given in the next Section.

The corrected version is:

```
Function factorialn(ByVal number As Integer)
    If number = 1 Then
        Return 1
    Else
        Return number * factorialn(number - 1)
    End If
End Function
```

and the main program (excluding variable declarations) is:

```
Do
    num = InputBox("Enter an integer number.", "Input Data")
Loop Until num >= 0

If num = 0 Or num = 1 Then
    fact = 1
Else
    fact = factorialn(num)
End If

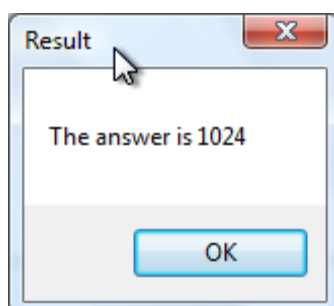
MsgBox("Factorial " & num & " is " & fact, , "Result")
```


If the input is 5, what is the output of the following program?

Now read the following code and state what it does. Write down the output when the number entered is 10.

```
Private Sub btnCalculate_Click(ByVal sender As System.Object,  
    ByVal e As System.EventArgs) Handles btnCalculate.Click  
  
    Dim num As Integer  
  
    Dim power As Integer  
  
    Do  
  
        num = InputBox("Enter a positive integer", "Input")  
  
    Loop Until num >= 0  
  
    power = power2(num)  
  
    MsgBox("The answer is " & power, , "Result")  
  
End Sub  
  
Function power2(ByVal number As Integer)  
  
    If number = 0 Then  
        Return 1  
    Else  
        Return 2 * power2(number - 1)  
    End If  
End Function
```

This code outputs 2^n if n is the input. Thus, if 2 is input the output is:



2.2.h Trace the execution of a recursive subroutine including calls to itself

Read the following algorithms for the function called Mystery and the recursive function Search (which is called by Mystery).

```
Private Sub btnCalculate_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnCalculate.Click

    Dim num As Integer

    Dim answer As Integer

    num = InputBox("Enter a positive integer", "Input")

    answer = find(1, num, num)

    MsgBox("The answer is " & answer, , "Result")

End Sub

Function find(ByVal first As Integer, ByVal last As Integer, ByVal number As
Integer)

    Dim mid As Integer

    Dim value As Integer

    mid = Int((first + last) / 2)

    If (mid * mid <= number) And (number < (mid + 1) * (mid + 1)) Then

        value = mid

    Else

        If mid * mid > number Then

            value = find(first, mid - 1, number)

        Else

            value = find(mid + 1, last, number)

        End If

    End If

    Return value

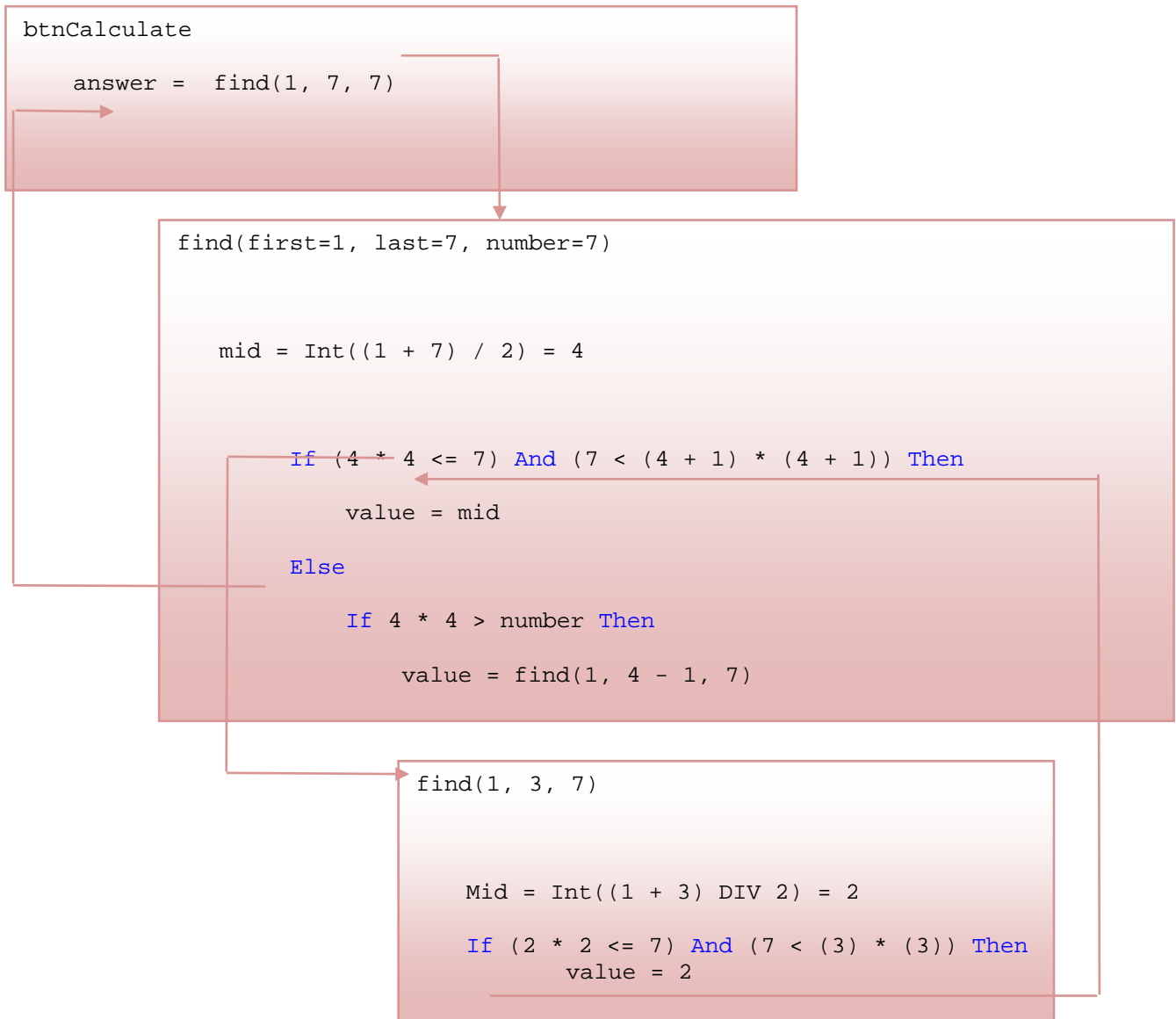
End Function
```

Int(n) returns the largest integer not greater than n.

Int(2.4) = 2 and Int(5.9) = 5.

If the input is 7, the following diagram shows each step of the algorithm.

Each time the recursive function is called, new storage has to be allocated to first, last, number, mid and value. This is because each time the function is called it has to have its own values for the variables.



Notice how a function returns to the instruction immediately after the one that called it.

2.2.i Discuss the relative merits of iterative and recursive solutions to the same problem

Any recursive algorithm can be written as an iterative program. Look again at Section 2(g) where the solution to the factorial problem was written as an iterative and as a recursive algorithm.

Recursion can be proved mathematically to produce a correct solution (or otherwise). Iteration can also be proved to be correct but sometimes is more difficult.

Iteration has the advantage over recursion in that the storage required is fixed. In recursion, the storage required varies according to the number of times the recursive function is called. Recursion leads to simpler algorithms than iteration.

Many problems are much easier to solve recursively as was shown with finding $n!$. This is because often the problem is defined recursively as you will see in A2.

Consider the Fibonacci sequence in which the first two terms are 1 and the rest are the sum of the previous two terms. This can be expressed as:

$$\text{fibonacci}(n) = \begin{cases} \text{fibonacci}(n-1) + \text{Fibonacci}(n-2) & \text{if } n > 2 \\ 1 & \text{if } n = 2 \\ 1 & \text{if } n = 1 \end{cases}$$

The program can be written as:

```
Private Sub btnCalculate_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnCalculate.Click
    Dim num As Integer
    num = InputBox("Enter a positive integer.", "Input")

    MsgBox("The " & num & " th number is " & fibonacci(num), , _
        "Result")
End Sub

Function fibonacci(ByVal number As Integer)

    If number = 1 Or number = 2 Then
        Return 1
    Else
        Return fibonacci(number - 1) + fibonacci(number - 2)
    End If
End Function
```

Compare this solution with an iterative solution shown below.

```
Private Sub btnCalculate_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnCalculate.Click
    Dim num As Integer
    Dim num1, num2, num3 As Integer
    Dim count As Integer = 2

    num = InputBox("Enter a positive integer.", "Input")
    If num = 1 Or num = 2 Then
        MsgBox("The " & num & " th number is 1", , "Result")
    Else
        num1 = 1
        num2 = 1
        While count <> num
            num3 = num1 + num2
            num1 = num2
            num2 = num3
            count = count + 1
        End While
        MsgBox("The " & num & " th number is " & num3, , "Result")
    End If
End Sub
```

The recursive solution follows the definition very closely and is much shorter. The iterative solution involves turning the definition into repetition.

2.2 Example Questions



1 Define the following terms.

- (i) Statement; (3)
- (ii) Procedure. (3)
- (iii) Parameter. (2)
- (iv) Selection. (2)
- (v) Loop. (2)

- A
- (i) - An instruction in a program or algorithm.
 - It may be simple like an assignment statement
 - `total = number1 + number2`
 - or more complex like a while... statement
 - `while number < 0`
 - `input number`
 - `endwhile` (3)
 - (ii) - A group of instructions that do a specific task.
 - It has a name
 - which is used to call it by another program/procedure.
 - It can receive values from the calling program
 - and may return 0, 1 or more values to the calling program. (3)
 - (iii) - A parameter is a variable in a subroutine
 - that receives its value from the calling program. (2)
 - (iv) - Selection is when alternative pieces of code can be chosen
 - according to some condition. (2)
 - (iv) - A loop is a group of instructions
 - in which the last instruction may return control to the first instruction. (2)

2 In the following program, identify

- (i) a sequence; (1)
- (ii) selection; (1)
- (iii) iteration. (1)

```
lstResult.Items.Clear()  
number = InputBox("Enter a number (0 to end).", "Input")  
  
While number <> 0  
    If number < 0 Then  
        lstResult.Items.Add("The number is negative.")  
    Else  
        lstResult.Items.Add("The number is positive.")  
    End If  
    number = InputBox("Enter a number (0 to end).", "Input")  
End While
```

A

```

lstResult.Items.Clear()
number = InputBox("Enter a number (0 to end).", "Input") } Sequence

While number <> 0
Selection(1) { If number < 0 Then
               lstResult.Items.Add("The number is negative.")
               Else
               lstResult.Items.Add("The number is positive.")
               End If
               number = InputBox("Enter a number (0 to end).", "Input")
End While } Iteration(1)

```

3 The table shows some abbreviations.

Abbreviation	Meaning
m	metre
kg	kilogram
s	second
A	ampere
K	kelvin

Write an algorithm to input an abbreviation and to output its meaning. If the abbreviation is not in the table, a suitable message should be output. (5)

A

- INPUT abbreviation (1)
- SELECT CASE abbreviation (1)
 - CASE "m"
 - OUTPUT "metre"
 - CASE "kg"
 - OUTPUT "kilogram"
 - CASE "s"
 - OUTPUT "second"
 - CASE "A"
 - OUTPUT "ampere"
 - CASE "K"
 - OUTPUT "kelvin"
- CASE ELSE (1)
 - OUTPUT "The abbreviation is unknown"
- END SELECT (1)

- 4 The following algorithm inputs two integer numbers and outputs all the integers from the first number to the last number.

```

INPUT firstNumber, lastNumber
FOR number = firstNumber TO secondNumber
    OUTPUT number
NEXT number

```

Rewrite this algorithm using a

- (i) WHILE – ENDWHILE loop; (3)
(ii) REPEAT – UNTIL loop. (3)
- (i) INPUT firstNumber, lastNumber }
 number = firstNumber } (1)
 WHILE number <= lastNumber }
 OUTPUT number }
 number = number + 1 } (1)
 ENDWHILE (1)
- (ii) INPUT firstNumber, lastNumber }
 number = firstNumber } (1)
 REPEAT }
 OUTPUT number }
 number = number + 1 } (1)
 UNTIL number > lastNumber } (1)

- 5 Write an algorithm to input a number. If the number is zero, stop. Otherwise, if the number is positive, output the number; if the number is negative, square the number, and output the square root of the answer. Then read another number. (5)

```

A INPUT number (1)
WHILE number <> 0 (1)
    IF number > 0 THEN
        OUTPUT number
    ELSE
        OUTPUT SQRT(number x number)
    ENDIF
    INPUT number (1)
ENDWHILE (1)

```

- 6 Read the following program and subroutine.

```

Dim num1, num2 As Double

num1 = InputBox("Enter first number.", "First Number")
num2 = InputBox("Enter second number", "Second Number")

mystery(num1, num2)

MsgBox("The numbers are " & num1 & " and " & num2, , "Result")

Sub mystery(ByRef firstNumber As Double, ByRef secondNumber As Double)

    Dim temp As Double
    If firstNumber > secondNumber Then
        temp = firstNumber
        firstNumber = secondNumber
        secondNumber = temp
    End If

```

End Sub

(a) State the output when the input is

(i) 3 and 7; (1)

(ii) 8 and 4. (1)

(b) State the purpose of the subroutine. (1)

A (a) (i) 3, 7 (1)

(ii) 4, 8 (1)

(b) To sort two numbers into ascending order. (1)

7 The following function is supposed to return the sum of the first n integers.

Explain what is wrong with it and correct the error.

```
Function sum(ByVal n As Integer)
    Return n + sum(n - 1)
End Function
```

- A - The function does not have a stopping condition
- therefore it will go on until there is no memory left

The corrected version is

```
Function sum(ByVal n As Integer)

    If n = 1 Then
        Return 1
    Else
        Return n + sum(n - 1)
    End If
End Function
```

8 Read the following algorithm.

```
FUNCTION mystery(m, n)

    r = m - INT(m / n) * n

    IF r = 0 THEN
        RETURN n
    ELSE
        RETURN mystery(n, r)
    END IF
END FUNCTION
```

INT(number) returns the integer part of number.

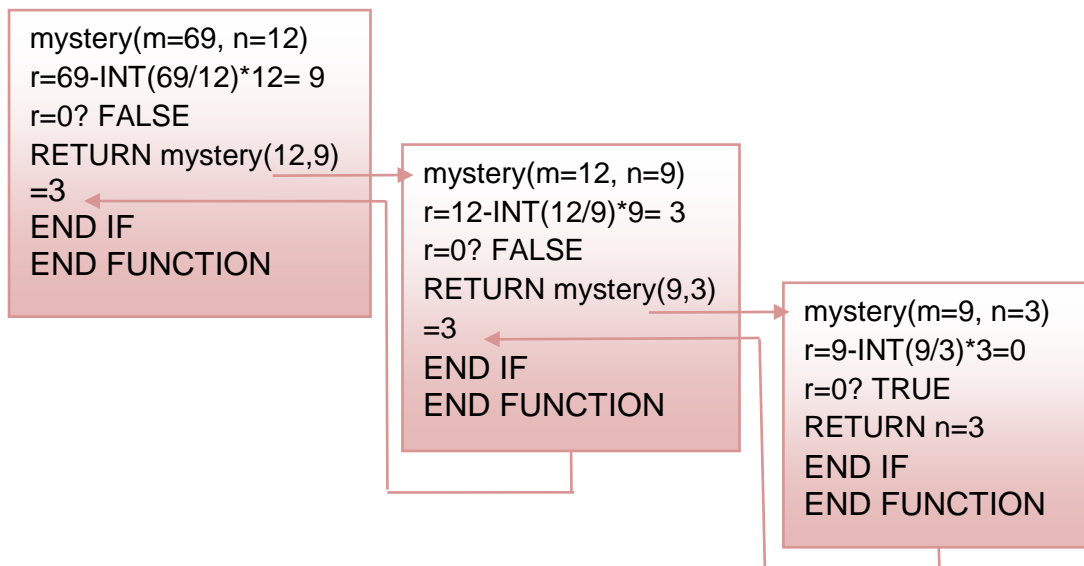
For example INT(4.2) = 4 and INT(5.8) = 5.

The function mystery is called with

mystery(69,12)

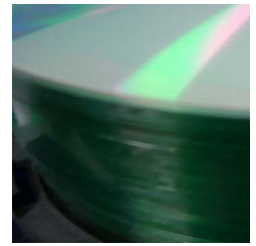
Produce a diagram showing each step of the algorithm and the function calls. (7)

A



(7)

2.3 Data Types and Data Structures



2.3.a *Define different data types, eg numeric (integer, real), Boolean, character and string; select and use them appropriately in the solutions to problems*

The computer needs to use different types of data in the operation of the system. All of these different types of data will look the same because they all have to be stored as binary numbers. The computer can distinguish one type of data from another by seeing whereabouts in memory it is stored.

Numeric data

There are different types of numbers that the computer must be able to recognise.

Numbers can be restricted to whole numbers; these are called INTEGERS and are stored by the computer as binary numbers using a whole number of bytes. It is usual to use 1 byte (called byte), 2 bytes (called short integers), 4 bytes (called integers) or 8 bytes (called long integers); the difference being simply that the more bytes used the larger the range of numbers that can be stored.

Integers are used for counting. For example, the number of students in a class would be of type Integer. Another type of number is a fraction (or decimal). This is stored by changing the position of the point so that it is always at the front of the number and then storing the fraction that is left (there is no whole part because the point comes first). However, 2.3 and .23 will now both be stored as .23 so we need an extra piece of data to be stored, the number of places the point has had to move. Fractions, then, are stored in two parts: the fraction part (called the mantissa) and the number of places the point has had to move (called the exponent). Because the first stage is always to move the point to the front of the number, and it is like the point "floating to the top", fractions are stored in FLOATING POINT form. Obviously there is a lot missing from this explanation, but that is more than enough for this part of the course. Simply remember that fractions are stored in binary with the point at the front, and then a separate part is the number of places the point has had to move to get it to the front.

Floating point numbers are used when fractions are required. For example, the mass of an object.

Boolean data

Sometimes the answer to a question is either yes or no, true or false. There are only two options. The computer uses binary data which consists of bytes of information, so it seems reasonable that the answer to such questions can be stored as a single byte. The computer stores True (Yes) as -1 and False (No) as 0. Data which can only have two states like this is known as BOOLEAN data.

A simple example of its use would be in the control program for an automatic washing machine. One of the important pieces of information for the processor would be to know whether the door was shut. Boolean variable could be set to 0 if it was open and to -1 if it was shut. A simple check of that value would tell the processor whether it was safe to fill the machine with water.

Characters

A character can be anything, which is represented in the character set of the computer by a character code. Modern systems use 2 bytes to represent a single Unicode character.

A character is useful when storing the gender of a person, "F" for female and "M" for male.

Strings

A string is a collection of characters which is given a name. The name can be used to reference the string of characters.

A string could store the name of a student.

2.3.b Define and use arrays (one- and two-dimensional) for solving simple problems, including initialising arrays, reading data into arrays and performing simple serial search on a one-dimensional array

Data stored in a computer is stored at any location in memory that the computer decides to use. This means that similar pieces of data can be scattered all over memory. This, in itself, doesn't matter to the user, except that to find each piece of data it has to be referred to by a variable name.

e.g. If it is necessary to store the 20 names of students in a group then each location would have to be given a different variable name. The first, Jane, might be stored in location NAME, the second, John, might be stored in FORENAME, the third, Penny, could be stored in CHRNAME, but I'm now struggling, and certainly 20 different variable names that made sense will be very taxing to come up with. Apart from anything else, the variable names are all going to have to be remembered.

Far more sensible would be to force the computer to store them all together using the variable name NAME. However, this doesn't let me identify individual names, so if I call the first one NAME(1) and the second NAME(2) and so on, it is obvious that they are all peoples' names and that they are distinguishable by their position in the list. Lists like this are called ARRAYS.

Because the computer is being forced to store all the data in an array together, it is important to tell the computer about it before it does anything else so that it can reserve that amount of space in its memory, otherwise there may not be enough space left when you want to use it. This warning of the computer that an array is going to be used is called INITIALISING the array. Initialising should be done before anything else so that the computer knows what is coming.

Initialising consists of telling the computer:

- what sort of data is going to be stored in the array so that the computer knows what part of memory it will have to be stored in
- how many items of data are going to be stored, so that it knows how much space to reserve
- the name of the array so that it can find it again.
- and the initial values of each name (which may be NULL)

Different programming languages have different commands for doing this but they all do the same sort of thing, a typical command would be

```
DIM surname(20) As String
```

DIM is a command telling the computer the name of the array, the number of names in the array and the data type to be used.

'Surname' is the name of the array, as String tells the computer that the data is going to be characters.(20) tells it that there are going to be up to 20 pieces of data.

Notes: Just because the computer was told 20 does not mean that we have to fill the array, the 20 simply tells the computer the maximum size at any one time.

Note that different languages treat the size of an array differently. Visual Basic.NET would allocate 21 elements to the array because the first element is numbered 0.

Other languages allow the programmer to state the numbers of the first and last elements. For example:

```
DIM surname(11:30) As String
```

would allocate 20 elements numbered surname(11) to surname(30).

This is a one-dimensional array. It is possible to have two-dimensional array, indeed, most languages allow multi-dimensional arrays (eg Visual Basic.NET).

An example of the definition of a two dimensional array is:

```
DIM table(10,20) As Integer
```

In Visual Basic.NET this would create a table with 11 rows and 21 columns.

Suppose we wish to initialise the array surname() with blanks. The code is:

```
Dim surname(20) As String
Dim i As Integer

For i = 0 To 20
    surname(i) = " "
Next i
```

To initialise an array of integers to zero use

```
Dim numbers(20) As Integer
Dim i As Integer

For i = 0 To 20
    numbers(i) = 0
Next i
```

Now suppose you wish to read the values for a one-dimensional array as they are input by the user. Suppose we know that the maximum number of values is 21 but there may be less. The array is to hold the surnames of the students in a class.

First we need to declare an array to hold 21 strings (representing the surnames), then we must stop trying to read names when the maximum is reached or the last one has been read. To indicate the end of the list we shall use the surname END.

The following code solves the problem:

```
Dim surnames(20) As String

Dim name As String

Dim i As Integer

i = 0

name = InputBox("Enter the next name (END to finish).", _
                "Input names")

While i <= 20 And name <> "END"

    surnames(i) = name

    name = InputBox("Enter the next name (END to finish).", _
                    "Input names")

    i = i + 1

End While
```

Now suppose you wish to initialise a two-dimensional array to represent the multiplication tables from 0 to 10. The following code will do this:

```
Dim multiplicationTable(10, 10) As Integer

Dim row, column As Integer

For row = 0 To 10

    For column = 0 To 10

        multiplicationTable(row, column) = row * column

    Next column

Next row
```

Notice the nested FOR loops. The effect is that row takes the value 0 then column takes each of the values 0 to 10. Next row takes the value 1 and column takes the values 0 to 10 again. This is repeated until row = 10 and column = 10 which is the last cell in the multiplicationTable. Each cell has the value row * column hence the multiplication table is produced.

Now suppose you want to search a one-dimensional array for a particular value. The simplest algorithm is:

```
Look at each element in the array
If the element is the required one
    output its position
If the end of the list is reached without finding item
    output element is not in list
```

This states what to do but is not detailed enough to write code. Now look at this algorithm:

```
count = 0
found = FALSE
READ name of item to be found
WHILE NOT End-Of-List and element not found
    IF list(count) = name THEN
        OUTPUT "Position is ", count
        found = TRUE
    ELSE
        count = count + 1
    END IF
END WHILE
IF NOT found THEN
    OUTPUT name, " is not in list"
END IF
END
```

The following code assumes that n names have been entered into the array called list:

```
Dim list(20), name As String

Dim n, count As Integer

Dim found As Boolean

count = 0

found

name = InputBox("Enter the name to be found", "Enter Name")

Do While count <= n And Not found

    If list(count) = name Then

        MsgBox(name & " is in position " & count, , _
                "Result of Search")

        found = True

    Else

        count = count + 1

    End If

Loop

If Not found Then

    MsgBox(name & " is not in the list", , "Error")

End If

End Sub
```

The above algorithm and code assume that there is at least one name in the list. As an exercise modify the algorithm and code to cope with an empty array.

2.3.c Explain the advantages and disadvantages of different data type and data structures for solving a given problem

Suppose you wish to store the names of all the students in a class. One way is to store the names as name1, name2, name3, The trouble here is that there will have to be an input instruction written for every name. This is cumbersome. Further, the number of names will have to be fixed. An alternative solution is to use an array. As long as the array is made big enough, it will hold all the names. Look at this algorithm:

```
INPUT the number of names = numNames
FOR count = 1 TO numNames
    INPUT names(count)
NEXT count
```

This requires names to be declared as a one-dimensional array of some size. As the maximum number in a class is not likely to exceed 100, dimension it as:

```
DIM names(100) AS STRING
```

Notice how simple the code is. Another method is shown below. This is used if the number of names is not known to start with:

```
count = 1
INPUT names(count)
WHILE names(count) <> "END"
    count = count + 1
    INPUT names(count)
ENDWHILE
```

This is more efficient and easier to program than using name1, name2, name3, However, it is more complex for the compiler. Here is the Visual Basic.NET code:

```
Dim count As Integer

Dim names(100) As String

count = 1

names(count) = InputBox("Enter the next name.(END to finish)", _
    "Enter names")

While names(count) <> "END"

    count = count + 1

    names(count) = InputBox("Enter the next name.(END to finish)", _
        "Enter names")

End While
```

Now suppose the name consists of a surname followed by a first name, which need to be dealt with separately. Clearly the most efficient method is to use arrays; again there are two methods.

The first is to define two arrays called surnames() and firstNames(). Dimension these the same as before and the algorithm is then:

```
count = 1
INPUT surnames(count)
WHILE surnames(count) <> "END"
    INPUT firstNames(count)
    count = count + 1
    INPUT surnames(count)
ENDWHILE
```

The code is:

```
Dim count As Integer

Dim surnames(100), firstNames(100) As String

count = 1

surnames(count) = InputBox("Enter the next surname.(END to finish", _
    "Enter Surnames")

While surnames(count) <> "END"

    firstNames(count) = InputBox("Enter the firstName.", "First Names")

    count = count + 1

    surnames(count) = InputBox("Enter the next name.(END to finish)", _
        "Enter Surnames")

End While
```


Now write an algorithm and a program for the case where the number of names is known and is the first item of data.

This requires two arrays and needs the programmer to keep track of which element in the arrays pair off. An alternative is to use a two-dimensional array (called names here) dimensioned as String. The first column holds the surname and the second column holds the first name. Here's the algorithm.

```
count = 1
INPUT names(count, 1)
WHILE names(count, 1) <> "END"
    INPUT names(count, 2)
    count = count + 1
    INPUT names(count, 1)
ENWHILE
```

and the code is:

```
Dim count As Integer

Dim names(100, 100) As String

count = 1

names(count, 1) = InputBox( _
    "Enter the next surname. (END to finish)", "Enter Surnames")

While names(count, 1) <> "END"

    names(count, 2) = InputBox("Enter the First Name.", "First Names")

    count = count + 1

    names(count, 1) = InputBox("Enter the next surname.(END to finish)", _
        "Enter Surnames")

End While
```

Write the algorithm and program to solve the same problem when the number of names is known and is the first item of data.

The use of a two-dimensional array makes it easier to keep the two names together as they are always on the same row of a single array. Incidentally, to output the names in the order first name followed by surname simply output

```
names(count, 2), names(count,1)
```

Now suppose you need to hold the first name, surname, date of birth, gender, height and weight of the pupils in a class. These are all different data types and, therefore, we cannot use a single array because an array can only be used with a single data type. This is overcome by using a record and is discussed in the next Section.

2.3.d *Design and implement a record format*

Data stored in computers is normally connected in some way. For example, the data about the 20 students in the set that has been the example over the last three sections has a connection because it all refers to the same set of people. Each person will have their own information stored, but it seems sensible that each person will have the same information stored about them, for instance their name, address, telephone number, exam grades...

All the information stored has an identity because it is all about the set of students, this large quantity of data is called a FILE.

Each student has their own information stored. This information refers to a particular student, it is called their RECORD of information. A number of records make up a file.

Each record of information contains the same type of information, name, address and so on. Each type of information is called a FIELD. A number of fields make up a record and all records from the same file must contain the same fields.

The data that goes into each field, for example "Jane Smith", "3 Canal St." will be different in most of the records. The data that goes in a field is called an ITEM of data.

Note: Some fields may contain the same items of data in more than one record. For example, there may be two people in the set who happen to be called Jane Smith. If Jane Smith's brother Tom is in this set he will presumably have the same address as Jane. It is important that the computer can identify individual records, and it can only do this if it can be sure that one of the fields will always contain different data in all the records. Because of this quality, that particular field in the record is different from all the others and is known as the KEY FIELD. The key field is unique and is used to identify the record. In our example the records would contain a field called school number which would be different for each student.

Note: Jane Smith is 10 characters (1 for the space), Christopher Patterson is 21 characters. It makes it easier for the computer to store things if the same amount of space is allocated to the name field in each record. It might waste some space, but the searching for information can be done more quickly. When each of the records is assigned a certain amount of space the records are said to be FIXED LENGTH. Sometimes a lot of space is wasted and sometimes data has to be abbreviated to make it fit. The alternative is to be able to change the field size in every record, this comes later in the course

To design a record format, the first thing to do is to decide what information would be sensible to be stored in that situation.

e.g. A teacher is taking 50 students on a rock-climbing trip. The students are being charged £20 each and, because of the nature of the exercise, their parents may need to be contacted if there is an accident. The teacher decides to store the information as a file on a computer. Design the record format for the file.

Answer:

The fields necessary will be Student name, Amount paid, Emergency telephone number, Form (so that contact can be made in school). There are other fields that could be included but we will add just one more, the school number (to act as the key field).

For each one of these fields it is necessary to decide what type of data they will be and also to decide how many characters will be allowed for the data in that field, remember these are fixed length records.

The easiest way is to write them in a table:

School Number	Integer	2 bytes
Student number	Integer	1 byte
Student name	String	20 bytes
Amount paid	Integer	1 byte
Emergency number	String	12 bytes
Form(e.g.3RJ)	String	3 bytes

Notes: It would be perfectly reasonable to say that the school number was not a proper number so it should be stored as characters (probably 4 bytes) or in BCD (2 bytes).

The student name is quite arbitrary. 15 bytes would be perfectly reasonable, as would 25 bytes, but 5 bytes would not. In other words there is no single right answer, but there are wrong ones.

The amount paid is listed as an integer as the teacher will store the number of whole pounds so far paid. This is not the best data type for an amount of money but it is the only one that fits so far. When you have learned about other data types use them if they are more suitable.

Many students expect that the emergency number should be an integer, but phone numbers start with a 0, and integers are not allowed to. As most numbers do start with 0, the computer can be programmed to put them in at the start of the rest of the number, but you would have to say this in your answer. As we are not going to do any arithmetic with these numbers, why make life more complicated than necessary.

3 characters were allowed for form. If in doubt, give an example of what you mean by the data, it can't hurt and it may save you a mark in the question.

How is this implemented in a high-level programming language? Different languages use slightly different terms, but Visual Basic.NET uses the term Structure. A Structure is very similar to a record and here it is defined by:

```
Structure Student

    Dim schoolNumber As Short

    Dim studentNumber As Byte

    Dim studentName As String

    Dim amountPaid As Short

    Dim emergencyNumber As String

    Dim form As String

End Structure
```

The Structure is called Student and contains 6 fields. Each field is given a name and a data type. It is now possible to define a variable to be of this new data type by using

```
Dim students(20) As Student
```

which is an array of up to 21 students of data type Student.

The rest of the program to read the data from a file called Student.txt is:

```
Private Sub btnStart_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnStart.Click

    Dim count, i As Integer

    Dim fileStudent As IO.StreamReader

    Dim answer As String

    Dim students(20) As Student

    Dim schoolNum As Integer

    fileStudent = IO.File.OpenText("Student.txt")

    count = 1

    While fileStudent.Peek <> -1

        students(count).schoolNumber = CInt(fileStudent.ReadLine)

        students(count).studentNumber = CInt(fileStudent.ReadLine)

        students(count).studentName = fileStudent.ReadLine

        students(count).amountPaid = CInt(fileStudent.ReadLine)

        students(count).emergencyNumber = fileStudent.ReadLine

        students(count).form = fileStudent.ReadLine

        count = count + 1

    End While

    fileStudent.Close()
```

2.3.e Define different modes of file access: serial, sequential, indexed sequential and random; and justify a suitable mode of file access for a given example

Computers can store large volumes of data. The difficulty is to be able to get it back. In order to be able to retrieve data it must be stored in some sort of order. Imagine the phone book. It contains large volumes of data which can be used, fairly easily, to look up a particular telephone number because they are stored in alphabetic order of the subscriber's name. Imagine how difficult it would be to find a number if they had just been placed in the book at random. The value of the book is not just that it contains all the data that may be needed, but that it has a structure that makes it accessible. Similarly, the structure of the data in a computer file is just as important as the data that it contains. There are a number of ways of arranging the data that will aid access under different circumstances.

Serial access

Data is stored in the computer in the order in which it arrives. This is the simplest form of storage, but the data is effectively unstructured, so finding it again can be very difficult. This sort of data storage is only used when it is unlikely that the data will be needed again, or when the order of the data should be determined by when it is input. A good example of a serial file is what you are reading now. The characters were all typed in, in order, and that is how they should be read. Reading this book would be impossible if all the words were in alphabetic order. Another example of the use of a serial file will be seen in section 1.4.15.

Sequential access

In previous sections we used the example of a set of students whose data was stored in a computer. The data can be stored in alphabetic order of their name. It could have been stored in the order that they came in a Computing examination, or by age with the oldest first. However it is done the data has been arranged so that it is easier to find a particular record. If the data is in alphabetic order of name and the computer is asked for Zaid's record it won't start looking at the beginning of the file, but at the end, and consequently it should find the data faster.

A file of data that is held in sequence like this is known as a sequential file.

Indexed sequential

Imagine a large amount of data, like the names and numbers in a phone book. To look up a particular name will still take a long time even though it is being held in sequence. Perhaps it would be more sensible to have a table at the front of the file listing the first letters of peoples' names and giving a page reference to where those letters start. So to look up Jones, a J is found in the table which gives the page number 232; the search is then started at page 232 (where all the Js will be stored). This method of access involves looking up the first piece of information in an index which narrows the search to a smaller area, having done this, the data is then searched alphabetically in sequence. This type of data storage is called Index Sequential.

Random access

A file that stores data in no order is very useful because it makes adding new data or taking data away very simple. In any form of sequential file an individual item of data is very dependent on other items of data. Jones cannot be placed after Monks because that is the wrong 'order'. However, it is necessary to have some form of order because otherwise the file cannot be read easily. What would be wonderful is if, by looking at the data that is to be retrieved, the computer can work out where that data is stored. In other words, the user asks for Jones' record and the computer can go straight to it because the word Jones tells it where it is being stored.

To access a random file, the data itself is used to give the address of where it is stored. This is done by carrying out some arithmetic (known as pseudo arithmetic because it doesn't make much sense) on the data that is being searched for.

For example, imagine that you are searching for Jones' data.

The rules that we shall use are that the alphabetic position of the first and last letters in the name should be multiplied together; this will give the address of the student's data.

So Jones = $10 * 19 = 190$. Therefore Jones' data is being held at address 190 in memory.

This algorithm is particularly simplistic, and does not give good results, as we shall soon see, but it illustrates the principle. Any algorithm can be used as long as it remains the same for all the data.

This type of algorithm is known as a HASHING algorithm.

The problem with this example can be seen if we try to find James's data.

James = $10 * 19 = 190$. The data for James cannot be here because Jones's data is here. This is called a CLASH. When a clash occurs, the simple solution is to work down sequentially until there is a free space. So the computer would inspect address 191, and if that was being used, 192, and so on until a blank space. The algorithm suggested here will result in a lot of clashes which will slow access to the data. A simple change in the algorithm will eliminate all clashes. If the algorithm is to write down the alphabetic position of all the letters in the name as 2 digit numbers and then join them together there could be no clashes unless two people had the same name.

e.g. Jones = 10, 15, 14, 05, 19 giving an address 1015140519

James = 10, 01, 13, 05, 19 giving an address 1001130519

The problem of clashes has been solved, but at the expense of using up vast amounts of memory (in fact more memory than the computer will have at its disposal). This is known as REDUNDANCY. Having so much redundancy in the algorithm is obviously not acceptable. The trick in producing a sensible hashing algorithm is to come up with a compromise that minimises redundancy without producing too many clashes.

2.3.f *Store, retrieve and search for data in files*

To store data in a file, you must first open the file and give it a name. The data must be structured so that it can be recalled in the same order that it was stored. When all the data has been written to the file, the file must be closed otherwise it cannot be used by the program. Each item of data can be stored on a separate line which makes retrieving it very easy. An alternative is to separate each item of data by a comma; this is known as Comma Separated Variables (CSV) format. This format requires slightly more programming to retrieve it.

There are different forms of storing data. First is to simply open a new file and write the data to it. Second is to open the file for appending data, this means that new data will be added to the end of the file. The third method is the most complex as it inserts data into an already existing file.

Insertion involves finding the correct position into which the new data is to be placed. All these methods will be dealt with in Section 3(h).

Retrieving data involves opening a file for reading then reading each item in turn. Again, the file must be closed when reading is complete.

The simplest form of search is the sequential search. This involves looking at each item in turn until the item required is found or the end of the file is reached when the program should report that the item is not in the list. The method is similar to searching an array as described in Section 3(b). The only difference is that the data is read from a file instead of a one-dimensional array.

Further details of these file operations, including programming them, are given in Section 3(h).

2.3.g Estimate the size of a file from its structure and the number of records

In Section 3(d), we designed this record format:

School Number	Integer	2 bytes
Student number	Integer	1 byte
Student name	String	20 bytes
Amount paid	Integer	1 byte
Emergency number	String	12 bytes
Form(e.g.3RJ)	String	3 bytes

We now need to calculate the size of the file.

Having decided on the size of each field, it is a simple matter of adding up the individual field sizes to get the size of a record, in this case 39 bytes.

There are 50 students going on the trip, each of them having their own record, so the size of the data in the file will be $50 * 39 = 1,950$ bytes.

All files need a few extra pieces of information that the user may not see such as information at the start of the file saying when it was last updated, which file it is, is it protected in any way? These sort of extra pieces of information are known as overheads, and it is usual to add 10% to the size of a file because of the need for overheads. Therefore the size of the file is $1,950 \text{ bytes} + (10\% \text{ of } 1,950 \text{ bytes}) = 2,145 \text{ bytes}$.

The final stage is to ensure that the units are sensible for the size of the file.

There are 1024 bytes in 1 Kbyte, so the size of this file is $2,145/1024 =$

2.09Kbytes.

Note: Don't worry about dividing by 1024, because, after all, this is only an approximation anyway. If you gave the final answer as 2 (approx) then that is just as acceptable. Just make sure that you write down somewhere that you know that there are 1024 bytes in a Kbyte, otherwise you can't be given the mark for knowing that.

2.3.h *Use the facilities of a procedural language to perform file operations (opening, reading, writing, updating, inserting, appending and closing)*

The methods used by procedural languages differ slightly so you must consult a text book for the language you are using. However, the methods are very similar and here we shall use Visual Basic.NET.

To open a file you must first dimension a variable as of type StreamReader then assign the name of file to this variable and open the file for reading, writing or appending.

So, to open a file for reading you will need the two statements:

```
Dim inputData As IO.StreamReader  
  
inputData = IO.File.OpenText(<full name of the file>)
```

where `inputData` is the name of the variable to be used when reading data by a statement of the form:

```
<variable name> = inputData.ReadLine
```

and `<full name of the file>` includes its path.

For example:

```
"C:\myFiles\vbFiles\names of students.txt"
```

Note the use of speech marks because the file name must be a string.

This assigns the next value in the file to the variable `<variable name>`.

To write to a file you must first open the file for writing:

```
Dim outputData As IO.StreamWriter  
  
outputData = IO.File.CreateText(<full name of the file>)
```

Data can then be written, one line at a time, by using

```
outputData.WriteLine(<data to be written>)
```

The data that is to be written to the file must be string or numeric.

Note, if a file already exists of the same name as the one that is being opened, it will be erased and a new empty file will be opened.

To update a file, two files are required. The first consists of the data to be updated; the second is a new file into which the updated data must be put. The original file can then be renamed as the backup file and the new file is given the original file's name. Both files must be closed before these actions can take place.

Here is an algorithm and a program to update a record in a file. The program will be referred to again and improved in Section 5 which refers to making code more understandable.

```
OPEN old file for reading
Open a new file for writing to
INPUT oldName (to be changed)
INPUT newName (name to change to)
found = FALSE
WHILE NOT EOF AND NOT found
    READ name from old file
    IF name <> oldName THEN
        WRITE oldName to new file
    ELSE
        Write newName to new file
    ENDIF
ENDWHILE
WHILE NOT EOF
    READ name from old file
    WRITE name to new file
ENDWHILE
IF NOT found THEN
    OUTPUT message that name was not found
ENDIF

DELETE any back-up file
RENAME old file to a back-up file
RENAME new file to old file file

END
```

```

Dim name As String
Dim oldName, newName As String
Dim found As Boolean
Dim oldFile As IO.StreamReader
Dim newFile As IO.StreamWriter

oldFile = IO.File.OpenText("ListOfNames.txt")
newFile = IO.File.CreateText("newListOfNames.txt")

oldName = InputBox("Enter the name to be changed.", _
    "Name to Change")
newName = InputBox("Enter the new name.", "New Name")

found = False

Do While oldFile.Peek <> -1 And Not found
    name = oldFile.ReadLine
    If name <> oldName Then
        newFile.WriteLine(name)
    Else
        newFile.WriteLine(newName)
        found = True
    End If
Loop

Do While oldFile.Peek <> -1
    name = oldFile.ReadLine
    newFile.WriteLine(name)
Loop

If Not found Then
    MsgBox("The name was not in the file", , "Error")
End If

oldFile.Close()
newFile.Close()

IO.File.Delete("listOfNames.bak")
IO.File.Move("listOfNames.txt", "listOfNames.bak")
IO.File.Move("newListOfNames.txt", "ListOfNames.txt")

End Sub

```

Two files are also required to insert a new item into a file. The algorithm and program are similar to those above. The main difference is that the file into which data is to be placed must be in some order. Before continuing, try to write an insertion algorithm and program to insert a new item into a file of names that are in alphabetic order.

```

OPEN old file for reading
Open a new file for writing to
INPUT newName
inserted = FALSE
WHILE NOT EOF AND NOT inserted
    READ name from old file
    IF name > newName THEN
        WRITE newName to new file
        WRITE name to new file
        inserted = TRUE
    ELSE
        WRITE name to new file
    ENDIF
ENDWHILE
IF NOT inserted THEN
    WRITE newName to new file
ELSE
    WHILE NOT EOF
        READ name from old file
        WRITE name to new file
    ENDWHILE
ENDIF

DELETE any back-up file
RENAME old file to a back-up file
RENAME new file to old file file

END

```

```

Dim oldFile As IO.StreamReader
Dim newFile As IO.StreamWriter
Dim inserted As Boolean
Dim name, newName As String
Dim nextName As String
oldFile = IO.File.OpenText("List Of Names.txt")
newFile = IO.File.CreateText("New List Of Names.txt")
newName = InputBox("Enter name to be inserted.", "New Name")
inserted = False
Do While oldFile.Peek <> -1 And Not inserted
    name = oldFile.ReadLine
    If name > newName Then
        newFile.WriteLine(newName)
        newFile.WriteLine(name)
        inserted = True
    Else
        newFile.WriteLine(name)
    End If
Loop
If Not inserted Then
    newFile.WriteLine(newName)
Else
    Do While oldFile.Peek <> -1
        name = oldFile.ReadLine
        newFile.WriteLine(name)
    Loop
End If
oldFile.Close()
newFile.Close()
IO.File.Delete("List Of Names.bak")
IO.File.Move("List Of Names.txt", "List Of Names.bak")
IO.File.Move("New List Of Names.txt", "List Of Names.txt")

```

End

New records can be appended to an old file. This means that they are added to the end of the old file.

This can be done by simply opening a file in append mode using statements such as

```
Dim appendFile As IO.StreamWriter  
appendFile = IO.File.AppendText("dataFile.txt")
```

where `dataFile.txt` is the name of the file which is to have data appended to it.

All files must be closed before using them again. This was shown in the above examples which used the statements

```
oldFile.Close()  
newFile.Close()
```

2.3 Example Questions



- 1 A program is required that generates the numbers stored when a standard 6-sided die is thrown. The die is to be thrown 1000 times and the total number of 1s, 2s, 3s, 4s, 5s and 6s is to be calculated. The average of all 1000 throws is also to be output.

[5]

Define the variables that will be needed, clearly stating their data types.

- A
- | | |
|--------------|---|
| scores(1, 6) | integers to hold the total number of each throw |
| count | integer to count the total number of throws |
| numThrown | integer, random number from 1 to 6 to represent the throw |
| average | real to store the average throw |
| total | integer to hold the total of all throws |
| i | integer loop counter |

[5]

- 2 Write an algorithm for the above problem. You may assume the existence of a function RND() which generates a random number from 0 to 1, excluding 1.

[7]

- A
- ```
FOR i = 1 TO 6
 scores(i) = 0
NEXT i
total = 0
FOR count = 1 to 1000
 numThrown = INT(RND() * 6) + 1
 scores(numThrown) = scores(numThrown) + 1
NEXT count
FOR i = 1 TO 6
 OUTPUT "Number of ", i, "s is ", score(i)
 total = total + scores(i)
NEXT i
average = total / 1000
OUTPUT "Total is ", total
OUTPUT "Average is ", average
```

Marks are for

- Initialising array to zeros
- Initialising total to zero
- Generating 1000 random numbers from 1 to 6
- Adding 1 to appropriate running total
- Finding total of all throws
- Finding average
- Outputting results

There are a number of alternative solutions, but all must have the above components.

[7]

- 3 The details of all the stock in a shop are held in a file. When an item is sold, its barcode is read and the details of the item are looked up in the file. Describe a suitable type of file access and justify your choice. [4]

A Random access

To look through each record in the file until the one that is needed takes too long.  
On average this would involve looking at  $n/2$  records, where  $n$  is the total number of records.  
The barcode could be used to generate the position of the record ...  
which could then be accessed directly. [4]

- 4 A file is to be kept which holds details of university modules. Each module has a unique 4-digit code from 1000 to 4999, a module name, the number of credits it is worth (30 or 60) and the number of the room in which it is held (two uppercase letters followed by 3 digits).

- (a) Design a record to hold these details, specifying the length of each field and its data type. [5]
- (b) Estimate the size of the file if there are about 500 different modules.
- (c) The records are to be held in order of the module code. Write an algorithm to find the module with code 3542. [5]

A (a) The best way to do this is to draw a table.

| Field       | Data Type | Length (bytes)    |
|-------------|-----------|-------------------|
| Module Code | Integer   | 2 (allow 4)       |
| Name        | String    | 25 (allow 15 -35) |
| Credits     | Integer   | 2 (allow 1 or 4)  |
| Room        | String    | 5                 |

[5]

- (b) Bytes per record =  $2 + 25 + 2 + 5 = 34$   
500 records =  $34 \times 500 = 17000$   
Add 10% for overheads =  $17000 + 1700 = 18700$   
In kilobytes size =  $18700 / 1024$   
About 19kb

[5]

- (c) IF file is empty THEN OUTPUT error message "No records"  
count = 1  
found = FALSE  
WHILE NOT found AND NOT EOF  
    IF code of record(count) = 3542 THEN  
        OUTPUT details of this record  
        found = TRUE  
    ELSE  
        count = count + 1  
    END IF  
ENDWHILE

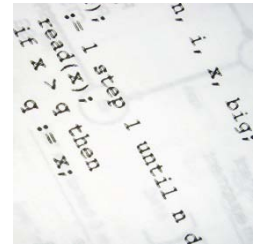
Marks are for

- Checking for empty file
- Initialising count and found (or similar variables)
- Setting up a loop
- ... with suitable terminator
- Checking to see if record is found
- Moving to next record

[5]

Note: This is not the most efficient algorithm, but would get full marks. See if you can improve it.  
Hint: What should happen if you meet a code that is greater than 3542?

## 2.4 Common Facilities of Procedural Languages



### 2.4.a *Understand and use assignment statements.*

We have already used assignment statements in some of the programs in earlier Sections. An assignment statement takes the form

```
variable = expression
```

For example:

```
found = True
surnames(i) = name
count = count + 1
```

The right hand side is first evaluated and its value is assigned to the variable on the left hand side.



## 2.4.b Understand arithmetic operators including operators for integer division (+, -, \*, /, ^, MOD and DIV) and use these to construct expressions

The symbols +, -, \*, / and ^ mean add, subtract, multiply, divide and exponentiation.

Examples of their use are the assignment statements:

```
total = 4.6 + 7.8
difference = 3.6 - 8.78
product = 4.54 * 6.43
dividend = 23.87 / 5.86
cube = 5 ^ 3
```

and:

```
total = num1 + num2
difference = firstNumber - secondNumber
product = number1 * number2
dividend = numerator / denominator
areaOfSquare = length ^ 2
```

MOD and DIV are special operators that are available in some high level languages.

a MOD b returns the remainder when a is divided by b where a and b have integer values. For example,

```
7 MOD 3 = 1
15 MOD 4 = 3
```

a DIV b means divide a by b and ignore any remainder where a and b have integer values. For example,

```
7 DIV 3 = 2
19 DIV 5 = 3
```

Thus, we can write:

```
remainder = num1 MOD num2
```

and

```
wholeNumber = num1 DIV num2
```

You will need to check whether or not these operators exist in the language you are using. Visual Basic.NET does not have these operators. The equivalent code is:

```
(a/b - Int(a / b)) * b for MOD
```

And:

```
Int(a/b) for DIV
```

### 2.4.c Understand a range of operational operators, eg =, <, <=, >, >=, and <> and use these to construct expressions

A couple of these relational operators have been used in programs in previous Sections. For example:

```
name <> oldName
```

and

```
name > newName
```

These expressions can take the values TRUE or FALSE. That is if `name` has the value "DAVIS" and `oldName` also has the value "DAVIS", then the value of the first expression is FALSE.

In the second expression, if `name` has the value "JONES" and `newName` has the value "CRADDOCK", then the value of the second expression is TRUE.

A summary of the relational operators is given below:

| Operator | Meaning                  |
|----------|--------------------------|
| =        | equals                   |
| <        | less than                |
| <=       | less than or equal to    |
| >        | greater than             |
| >=       | greater than or equal to |
| <>       | not equal to             |

## 2.4.d Understand the Boolean operators AND, OR and NOT and use these to construct expressions

The Boolean operators AND and OR need two Boolean values to produce an output. That is, if the values are value1 and value2, which can each be TRUE or FALSE, the following tables define these two operators:

| value1 | value2 | value1 AND value2 |
|--------|--------|-------------------|
| TRUE   | TRUE   | TRUE              |
| TRUE   | FALSE  | FALSE             |
| FALSE  | TRUE   | FALSE             |
| FALSE  | FALSE  | FALSE             |

| value1 | value2 | value1 OR value2 |
|--------|--------|------------------|
| TRUE   | TRUE   | TRUE             |
| TRUE   | FALSE  | TRUE             |
| FALSE  | TRUE   | TRUE             |
| FALSE  | FALSE  | FALSE            |

NOT requires a single input as shown in this table:

| value | NOT value |
|-------|-----------|
| TRUE  | FALSE     |
| FALSE | TRUE      |

AND and NOT have already been used in programs.

For example:

```
count <= n And Not found
```

This is only TRUE if the value of `count` is less than or equal to the value of `n` AND `found` is FALSE (which makes `Not found` TRUE).

Further examples may be found in many of the programs in this text.

## 2.4.e Understand the effects of the precedence of standard operators and the use of parentheses to alter the order of precedence

All operators have an order of precedence. The arithmetic operators have the following order of precedence:

| Order | Operator | Notes                          |
|-------|----------|--------------------------------|
| 1     | $\wedge$ | Left to right in an expression |
| 2     | $*$ /    | Left to right in an expression |
| 3     | $+$ -    | Left to right in an expression |

Thus

$$2 \wedge 3 \wedge 4 = 8 \wedge 4 = 4096$$

$$4 * 5 / 8 = 20 / 8 = 2.5$$

$$5 + 6 - 3 = 11 - 3 = 8$$

This order of precedence can be overridden by using left and right parentheses in pairs. The innermost pair is evaluated first

For example

$$2 * 7 - 3 = 14 - 3 = 11$$

but

$$2 * (7 - 3) = 2 * 4 = 8$$

and

$$14 - 3 \wedge 2 = 14 - 9 = 5$$

but

$$(14 - 3) \wedge 2 = 11 \wedge 2 = 121$$

If there are more than one pair of parentheses, then they are evaluated from the innermost pair outwards and from left to right. Here is an example.

$$(5 - 2) * ((16 - 1) / (2 + 3)) = 3 * (15 / 5) = 3 * 3 = 9$$

Be careful with expressions like  $\frac{a}{bc}$ . Mathematically this means divide  $a$  by the product of  $b$  and  $c$ .

However, in programming,  $a/bc$  contains **two** errors.  $bc$  is not allowed as it counts as a single variable; it should be  $b * c$ . However,

$a/b * c$  is not acceptable because it means  $a$  divided by  $b$  and then multiply the result by  $c$ . Using  $a = 20$ ,  $b = 2$  and  $c = 5$  should give  $\frac{20}{2 * 5}$  which is 2, but  $20/2 * 5$  gives  $10 * 5 = 50$ . In programming, it should be written  $20/(2 * 5) = 20 / 10 = 2$ .

The logical operators also have an order of precedence; this is

| Order | Operator | Notes                          |
|-------|----------|--------------------------------|
| 1     | NOT      | Left to right in an expression |
| 2     | AND      | Left to right in an expression |
| 3     | OR       | Left to right in an expression |

As with the arithmetic operators, parentheses can be used to change the order of precedence.

For example:

|     |                              |
|-----|------------------------------|
| is  | TRUE AND NOT FALSE OR TRUE   |
|     | TRUE AND TRUE OR TRUE        |
|     | = TRUE OR TRUE               |
|     | = TRUE                       |
| but | TRUE AND NOT (FALSE OR TRUE) |
|     | = TRUE AND NOT TRUE          |
|     | = TRUE AND FALSE             |
|     | = FALSE                      |

In order to be certain that the correct order is used, use parentheses. You may use as many as you like to be clear that you are using the right order.

## 2.4.f. Evaluate expressions containing arithmetic, relational and Boolean expressions

You have already used many of the relational operators which are given in the following table:

| Mathematical Notation | Programming Notation | Numeric Meaning          |
|-----------------------|----------------------|--------------------------|
| <                     | <                    | Less than                |
| ≤                     | <=                   | Less than or equal to    |
| =                     | =                    | Equal to                 |
| ≠                     | <>                   | Not equal to             |
| ≥                     | >=                   | Greater than or equal to |
| >                     | >                    | Greater than             |

### Example 1

Evaluate the following arithmetic expressions where  $a = 3$ ,  $b = 4$  and  $c = 5$ . Show your working.

- |                       |                       |                                   |                 |                   |
|-----------------------|-----------------------|-----------------------------------|-----------------|-------------------|
| (a) $a * b + c$       | (b) $a * (b + c)$     | (c) $a ^ b$                       | (d) $b * a ^ c$ | (e) $(c - a) ^ b$ |
| (f) $8 * (a + c) / b$ | (g) $(b + c) / a * b$ | (h) $(a + b) * (b + c) / (c - a)$ |                 |                   |

### Answers 1

- (a)  $a * b + c = 3 * 4 + 5 = 12 + 5 = 17$
- (b)  $a * (b + c) = 3 * (4 + 5) = 3 * 9 = 27$
- (c)  $a ^ b = 3 ^ 4 = 81$
- (d)  $b * a ^ c = 4 * 3 ^ 5 = 4 * 243 = 972$
- (e)  $(c - a) ^ b = (5 - 3) ^ 4 = 2 ^ 4 = 16$
- (f)  $8 * (a + c) / b = 8 * (3 + 5) / 4 = 8 * 8 / 4 = 64 / 4 = 16$
- (g)  $(b + c) / a * b = (4 + 5) / 3 * 4 = 9 / 3 * 4 = 3 / 4 = 0.75$
- (h)  $(a + b) * (b + c) / (c - a) = (3 + 4) * (4 + 5) / (5 - 3) = 7 * 9 / 2 = 63 / 2 = 31.5$

### Example 2

Evaluate the following logical expressions where  $a = 3$  and  $b = 4$ . Show your working.

- (a)  $4 * a = 3 * b$       (b)  $b <= a$       (c)  $(a + b) < (b - a)$       (d)  $3 ^ a = b ^ 2$
- (e)  $(5 - a) * (b + 1) > (a * b - 6)$       (f)  $\text{NOT } (a < b)$
- (g)  $((a + b) > 3) \text{ AND } ((2 * a - b) < 2)$
- (h)  $(ab < 10) \text{ OR } (a < 5) \text{ AND NOT } (b > a)$

## Answers 2

- (a)  $4 * a = 3 * b$   
 $4 * 3 = 3 * 4$   
 $12 = 12$   
TRUE
- (b)  $b \leq a$   
 $4 \leq 3$   
FASE
- (c)  $(a + b) < (b - a)$   
 $(3 + 4) < (4 - 3)$   
 $7 < 1$   
FALSE
- (d)  $3^a = b^2$   
 $3^3 = 4^2$   
 $27 = 16$   
FALSE
- (e)  $(5 - a) * (b + 1) > (a * b - 6)$   
 $(5 - 3) * (4 + 1) > (3 * 4 - 6)$   
 $2 * 5 > 12 - 6$   
 $10 > 6$   
TRUE
- (f) NOT  $(a < b)$   
NOT  $(3 < 4)$   
NOT TRUE  
FALSE
- (g)  $((a + b) > 3) \text{ AND } ((2 * a - b) < 2)$   
 $((3 + 4) > 3) \text{ AND } ((2 * 3 - 4) < 2)$   
 $(7 > 3) \text{ AND } ((6 - 4) < 2)$   
TRUE AND  $(2 < 2)$   
TRUE AND FALSE  
FALSE
- (h)  $(ab < 10) \text{ OR } (a < 5) \text{ AND NOT } (b > a)$   
 $(3 * 4 < 10) \text{ OR } (3 < 5) \text{ AND NOT } (4 < 3)$   
 $(12 < 10) \text{ OR TRUE AND NOT FALSE}$   
FALSE OR TRUE AND TRUE  
FALSE OR TRUE  
TRUE

## 2.4.g Understand and use a range of operators and built in functions for string manipulation, including location (LOCATE), extraction (LEFT, MID, RIGHT), comparison, concatenation, determining the length of a string (LENGTH) and converting between characters and their ASCII code (ASCII and CHAR)

The above functions are used in algorithms. Programming languages have similar functions but are not all given the same names. For example, older versions of Visual Basic use LEFT(), RIGHT() and MID() but Visual Basic.NET uses methods because it is an Object Oriented Programming (OOP) language. You will be studying OOP in Unit 3.

The formats of these functions are given in the following table, together with their meanings. In the table, str, str1 and str2 are strings and n and m are positive integers.

Also, each character of a string has a position relative to the start of the string. The first character is numbered 0 (some languages start numbering at 1). Thus, the position of p in the string "computer" is 3.

| Function                  | Meaning                                                                                      |
|---------------------------|----------------------------------------------------------------------------------------------|
| <b>LOCATE(str1, str2)</b> | Return the start position of str2 in str1, -1 if str2 is not in str1                         |
| <b>LEFT(str, n)</b>       | Return the first n characters of str as a string                                             |
| <b>MID(str, m, n)</b>     | Return, as a string, the next n characters of str, starting at the m <sup>th</sup> character |
| <b>RIGHT(str, n)</b>      | Return the last n characters of str                                                          |
| <b>LENGTH(str)</b>        | Return the number of characters in str                                                       |

Examples of these functions are:

```
LOCATE("Technology", "chno") = 2
LOCATE("Technology", "Chno") = -1 (Note: C ≠ c)
LEFT("Technology", 6) = "Techno"
MID("Technology", 2, 4) = "chno"
RIGHT("Technology", 5) = "ology"
LENGTH("Technology") = 10
```

Every character has to be held, in a computer, as a pattern of binary digits. The first method used a standard set of values for the characters. This was known as ASCII (American Standard Code for Information Interchange). Using this code, each character is represented by 7 binary digits (the 8<sup>th</sup> digit of a byte is used for parity checking). In this system, A has the code 10000001 or 65 in denary, B is 1000010 or 66, C is 67 and so on. The digits 0 to 9 are represented by the denary numbers 48 to 57. (Note: Most modern computer languages use ANSI code which uses 8 bits and can, therefore, represent more characters. However, the ANSI values start off with the same values as ASCII.

The function ASCII(ch) returns the ASCII value of ch and CHAR(n) returns the character with ASCII value n.

For example:

```
ASCII("A") = 65 and ASCII("a") = 97
CHAR(67) = "C" and CHAR(56) = 8
```

(Note: In Visual Basic.NET the corresponding functions are Asc(ch) and Chr(n).)



Two strings can be joined together by using + or &. This is known as *concatenation*.

For example:

```
"Tech" + "nology" = "Technoogy"
or
"Tech" & "nology" = "Technoogy"
```

(Visual Basic.NET uses both the & and the + signs.)

Comparison is dealt with in the next Section.

The functions for LOCATE(), LEFT(), MID() and RIGHT in Visual Basic.NET are:

| Algorithm                 | Visual Basic.NET                 |
|---------------------------|----------------------------------|
| <b>LOCATE(str1, str2)</b> | str1.IndexOf(str2)               |
| <b>LEFT(str, n)</b>       | str.Substring(0, n)              |
| <b>MID(str, m, n)</b>     | str.Substring(m, n)              |
| <b>RIGHT(str, n)</b>      | str.Substring(str.Length – n, n) |

The reason that Visual Basic.NET uses different methods is that it is an Object Oriented language as are most modern languages. More in Unit 3.

## 2.4.h *Understand that relational operations on alphanumeric strings depend on character codes of the characters and explain the results of this effect (eg why 'XYZ' < 'abc', '2' > '17' and '3' <> '3.0')*

Because each character has a value, characters can be compared in the same way as numbers. This means that:

```
"A" < "C"
```

```
"X" > "x"
```

Also, because characters are compared one by one:

```
"a" < "ab"
```

```
"3" <> "3.0"
```

Example Write an algorithm that inputs a name in the form:

```
<surname><space><first name>
```

and outputs it in the form:

```
<first name><space><surname>
```

```
INPUT name
lengthOfName = LENGTH(name)
positionOfSpace = LOCATE(name, "space")
firstName = LEFT(name, positionOfName)
surname = RIGHT(name, lengthOfName – positionOfName – 1)
OUTPUT surname & space & firstName
```

In Visual Basic.NET this is:

```
Dim name As String
Dim firstName, surname As String
Dim lengthOfName As Integer
Dim posnOfSpace As Integer

name = txtInputName.Text

lengthOfName = name.Length
posnOfSpace = name.IndexOf(" ")
firstName = name.Substring(0, posnOfSpace)
surname = name.Substring(posnOfSpace + 1, lengthOfName - posnOfSpace - 1)
txtReversedName.Text = surname & " " & firstName
```

### 2.4.i. *Input and validate data*

This has been dealt with in Unit 1 but here are some algorithms that demonstrate what you can do. Remember, all data should be validated, if possible, on input. This will not ensure the correctness of data only that it is reasonable.

Suppose you are entering the gender of a person, you could use the following algorithm to check it:

```
REPEAT
 INPUT gender
UNTIL gender = "F" OR gender = "M"
```

This ensures that the input is only F or M, no other values are allowed. Why doesn't this ensure that, when data is entered, it may be wrong, even if you have entered F or M?

Suppose a program requires the entry of the number of computers ordered by a school. How can you validate the data? It seems reasonable to argue that the order must be for at least one computer and no more than 250. We can use the following algorithm to validate the data input:

```
REPEAT
 INPUT numberOfComputers
UNTIL numberOfComputers > 0 AND numberOfComputers <= 250
```

What type of validation check is being used? Is this sufficient? What happens if someone enters 15.6 instead of 156?

The Visual Basic.NET versions of these algorithms are:

```
Dim gender As Char

Do
 gender = InputBox("Enter the gender (F or M)", "Get gender")
Loop Until gender = "F" Or gender = "M"
```

And:

```
Dim numberOfComputers As Integer

Do
 numberOfComputers = InputBox(_
 "Enter the number of computers required", "Number Required")
Loop Until numberOfComputers > 0 And numberOfComputers <= 250
```

## 2.4.j. Output data onto screen/file/printer, formatting the data for output as necessary.

The normal output from a program is on screen. Output to a file was described in Unit 2 Section 3(h). Output to printer depends on the language but usually contains formatting information. In fact, formatting information can also be used on screen output.

Formatting information tells the printer (or screen) how to lay out the output.

One of the simplest methods that can be used is to simply explain the output. Suppose two numbers, num1 and num2 are input to a program and their sum is to be output. One print statement could be:

```
PRINT answer
```

and the output might be:

```
6.1
```

This does not tell you much.

Now consider this:

```
PRINT num1, num2, answer
```

which might produce:

```
2.6 3.5 6.1
```

it's still not very clear. Now try this:

```
PRINT "The sum of ", num1, "and ", num2, " is ", answer.
```

which produces:

```
The sum of 2.6 and 3.5 is 6.1
```

This is much clearer. This is one method of formatting the output. An alternative layout is obtained by using:

```
PRINT "The first number is ", num1
PRINT "The second number is ", num2
PRINT "and their sum is ", answer
```

This produces:

```
The first number is 2.6
The second number is 3.5
and their sum is 6.1
```

These examples show the simplest ways of formatting output.

All programming languages have build in formatting methods and you should familiarise yourself with these methods and use them, particularly in your project work in A2.

Very complex formatting can be used in all languages but these are language dependant. You need to understand the need for formatting output in order to make it easy to read and understand. Consider these two outputs:

```
RogerBlackfordM04/09/3777.4165
DianaWardF15/05/4150.3150
BillWorthyM18/06/4681.3180
```

And:

| Name           | Gender | DOB      | Weight(kg) | Height(cm) |
|----------------|--------|----------|------------|------------|
| RogerBlackford | M      | 04/09/37 | 77.4       | 165        |
| DianaWard      | F      | 15/05/41 | 50.3       | 150        |
| BillWorthy     | M      | 18/06/46 | 81.3       | 180        |

Which is the clearer?

These methods can be used to output data to both printer and screen. In fact, many languages allow you to use these methods to output to a text file. For further file output see Section 3(h).

## 2.4 Example Questions



- 1 Use the functions MOD and DIV to convert a sum of money in pence to £s and pence. You should use meaningful names for any constants and variables you use.[5]

A INPUT sumOfMoney  
penceInPound = 100  
pounds = sumOfMoney DIV penceInPound  
pence = sumOfMoney MOD penceInPound  
OUTPUT sumOfMoney, "pence is £", pounds, " and ", pence, "pence"

Marks are for

- Meaningful name for the constant
- Meaningful name for the variables
- Correct use of DIV
- Correct use of MOD
- Formatted output

[5]

- 2 mark is a variable that lies between 0 and 100 inclusive. Create a statement to input the value of mark until it is valid. [3]

A REPEAT  
INPUT mark  
UNTIL mark >= 0 AND mark <= 100

Marks are for

- Correct loop used
- INPUT is inside the loop
- The two Boolean expression

[3]

*Note: This is a single iterative statement that has a single output statement inside it.*

- 3 Evaluate the following Boolean expressions when a = 8, showing your working.

(a)  $a < 5 \text{ AND } a > 0 \text{ OR } a = 10$  [3]

(b)  $a < 5 \text{ AND } (a > 0 \text{ OR } a = 10)$  [3]

- A (a)  $a < 5$  AND  $a > 0$  OR  $a = 10$   
 FALSE AND TRUE OR FALSE  
 FALSE OR FALSE  
 FALSE [3]
- (b)  $a < 5$  AND ( $a > 0$  OR  $a = 10$ )  
 FALSE AND (TRUE OR FALSE)  
 FALSE AND TRUE  
 FALSE [3]

4 Evaluate the following expressions when  $p = 4$ ,  $q = 6$  and  $r = 10$ . Show your working.

- (a)  $p + q * r$  [3]
- (b)  $(p + q) * r$  [3]
- (c)  $(q - p) ^ (p + q) / r$  [4]

- A (a)  $p + q * r$   
 $4 + 6 * 10$   
 $4 + 60$   
 $64$  [3]
- (b)  $(p + q) * r$   
 $(4 + 6) * 10$   
 $10 * 10$   
 $100$  [3]
- (c)  $(q - p) ^ (p + q) / r$   
 $(6 - 4) ^ (4 + 6) / 10$   
 $2 ^ 10 / 10$   
 $1024 / 10$   
 $102.4$  [4]

5 The variable aString holds a sentence.

- (a) Describe the string functions LOCATE(str1, str2), RIGHT(str, n) and LENGTH(str). [3]
- (b) Write an algorithm, that uses the string functions LOCATE(str) and RIGHT(str, n), to count the number of words in the sentence. [6]

- A (a) LOCATE(str1, str2) returns the start position of str2 in str1, -1 if str2 is not in str1. The first character is in position 0.  
 RIGHT(str, n) returns the last n characters of str  
 LENGTH(str) returns the number of characters in str. [3]
- (b) This is one possible algorithm.

```
count = 0
REPEAT
 positionOfSpace = LOCATE(aString, space character)
 count = count + 1
 IF positionOfSpace <> -1
 lengthOfString = LENGTH(aString)
 aString = RIGHT(aString, LENGTH - positionOfSpace - 1)
 ENDIF
UNTIL positionOfSpace = -1
OUTPUT count [6]
```

*Note that there are many possible solutions, this is only one. The examiner will accept any algorithm that uses the functions LOCATE(str1, str2), RIGHT(str, n) and LENGTH(str) provided it works.*

## 2.5 Writing Maintainable Programs



### 2.5.a *Define, understand and use the following terms correctly as they apply to programming: variable, constant, identifier, reserved word/keyword*

A variable is a name associated with a location in memory which may hold different values. The value of the variable may change during the running of the program. It enables a program to be written before the values are known.

A constant is a value that, once set, cannot be changed while the program is running. It may be used literally such as 3.151493 or "Leadbetter ". It may be given a name such as `pi = 3.141593`. Pi is the name of the constant and may be set up by means of statements like

```
Const pi = 3.141593
```

or

```
Const surname = "Leadbetter"
```

Once set, these values cannot be changed while the program is running. This is useful because the value may be used many times during the running of a program. Consider the statement

```
Const VAT = 0.175
```

and suppose this value is used many times in a program. If VAT is changed to 0.2, then it will only need to be changed once (in the above statement) and not everywhere it occurs in the program.

An identifier is a name that is used to represent a variable, constant, function, procedure, data type, object or other element in a program.

A reserved word, or keyword, is one that is defined in a programming language. It can only be used for the purpose specified by the language. Typical reserved words are FOR, NEXT, LOOP, integer and so on.



## 2.5.b Explain the need for good program-writing techniques to facilitate the ongoing maintenance of programs

It is important that standard techniques are used in writing programs. This helps to find errors in programs as well as helping to modify programs when new facilities are required. Programs are often modified by a different programmer to the one who originally wrote the program. If standard techniques are used, all programmers will understand the code. Many techniques will be described in the following Sections but one that has been used many times in this text is indentation as demonstrated by this piece of code:

```
Dim num As Integer
Dim num1, num2, num3 As Integer
Dim count As Integer = 2

num = InputBox("Enter a positive integer.", "Input")
If num = 1 Or num = 2 Then
 MsgBox("The " & num & " th number is 1", , "Result")
Else
 num1 = 1
 num2 = 1
 While count <> num
 num3 = num1 + num2
 num1 = num2
 num2 = num3
 count = count + 1
 End While
 MsgBox("The " & num & " th number is " & num3, , "Result")
End If
```

Notice how the code is indented to show which statements are inside the IF ... THEN ... ELSE ... ENDIF statement and which statements are inside the WHILE ... ENDWHILE loop.

Several other techniques are described in the following Sections.

### 2.5.c Declare variables and constants, understanding the effect of scope and issues concerning the choice of identifier (including the need to avoid reserved words/keywords).

Variables and constants need to be declared so that the compiler or interpreter can reserve the correct amount of space for the values. Also, in the case of an array, the compiler/interpreter has to save the number of subscripts and the range of values these subscripts can take.

For example, consider the declaration:

```
Dim anArray(10, 20) As Integer
```

and assume that the subscripts start at 1.

This must reserve enough space for  $10 \times 20 = 200$  integer values. The compiler must also store the number of subscripts (2) and the upper bounds (10 and 20) for the subscripts.

When a variable is declared, it may only be used in certain parts of the program. This is known as its scope. Consider the following program:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnStart.Click

 Dim a, b, c, d As Integer

 a = 1
 b = 2

 lstResults.Items.Add("Now in main program")
 lstResults.Items.Add("a = " & a)
 lstResults.Items.Add("b = " & b)
 LocalVariables()

 lstResults.Items.Add("")
 lstResults.Items.Add("Now back in main program")
 lstResults.Items.Add("a = " & a)
 lstResults.Items.Add("b = " & b)

End Sub

Sub LocalVariables()

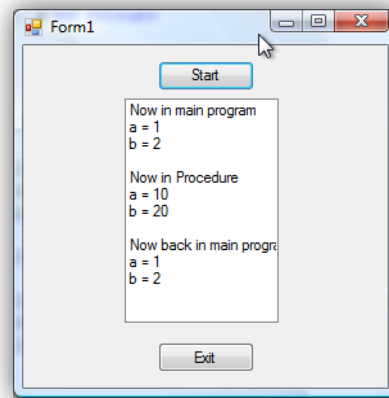
 Dim a, b As Integer

 a = 10
 b = 20

 lstResults.Items.Add("")
 lstResults.Items.Add("Now in Procedure")
 lstResults.Items.Add("a = " & a)
 lstResults.Items.Add("b = " & b)

End Sub
```

This produced the following result:



Notice how the variables *a* and *b* have different values when they are in the procedure to those they have in the main program. We say that the scope of *a* = 1 and *b* = 2 is the main program but in the procedure the values are *a* = 10 and *b* = 20. This is because these variables are in scope in the procedure. The main program cannot see *a* = 10 and *b* = 20, similarly the procedure cannot see *a* = 1 and *b* = 2.

This property of variables is very useful because many programmers may work on a project, writing many procedures, but they do not need to worry that another programmer may use the same name. This is because the scope of each variable will be different.

Other variables can be used throughout a program. These are called global variables and are declared at the start of a program. Compare the output of the following program with the one above:

```
Dim c, d As Integer

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnStart.Click

 Dim a, b As Integer
 a = 1
 b = 2
 c = 3
 d = 4

 lstResults.Items.Add("Now in main program")
 lstResults.Items.Add("a = " & a)
 lstResults.Items.Add("b = " & b)
 lstResults.Items.Add("c = " & c)
 lstResults.Items.Add("d = " & d)
 LocalVariables()
 lstResults.Items.Add("")
 lstResults.Items.Add("Now back in main program")
 lstResults.Items.Add("a = " & a)
 lstResults.Items.Add("b = " & b)
 lstResults.Items.Add("c = " & c)
 lstResults.Items.Add("d = " & d)

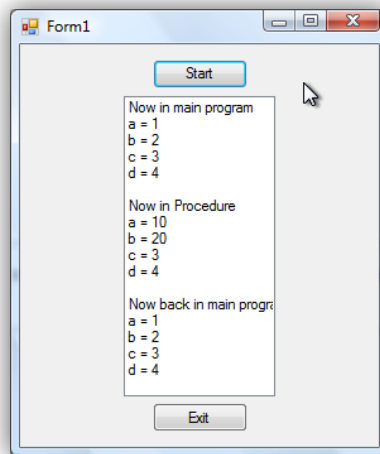
End Sub

Sub LocalVariables()

 Dim a, b As Integer
 a = 10
 b = 20
 lstResults.Items.Add("")
 lstResults.Items.Add("Now in Procedure")
 lstResults.Items.Add("a = " & a)
 lstResults.Items.Add("b = " & b)
 lstResults.Items.Add("c = " & c)
 lstResults.Items.Add("d = " & d)

End Sub
```

The results are shown below:



Notice that the values of c and d are the same in the procedure as in the main program.

What happens if c and d are given new values in the procedure? The new values only apply to c and d in the procedure; outside the procedure they remain the same as before entering the procedure. Also notice that c and d are not declared in the procedure, this is because they are global variables.

What happens if c and d are not given values in the procedure? Here is the new procedure and the results. The procedure has been modified to assign values to c and d. This is the new procedure.

```
Sub LocalVariables()

 Dim a, b As Integer

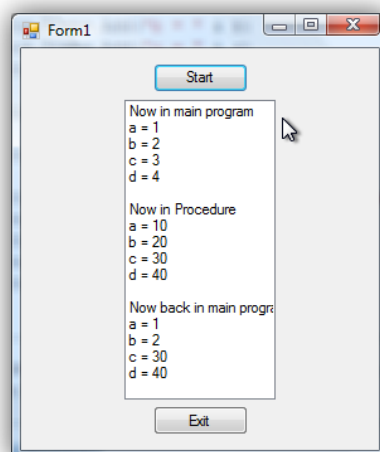
 a = 10
 b = 20
 c = 30
 d = 40

 lstResults.Items.Add(" ")

 lstResults.Items.Add("Now in Procedure")
 lstResults.Items.Add("a = " & a)
 lstResults.Items.Add("b = " & b)
 lstResults.Items.Add("c = " & c)
 lstResults.Items.Add("d = " & d)

End Sub
```

and the result is:



Notice how the change in values of c and d are reflected in the main program.

## 2.5.d *Select and use meaningful identifier names and use standard conventions to show the data types and enhance readability*

The programs used so far have used meaningful names for the variables and constants.

```
Private Sub btnCalculate_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnCalculate.Click
 Dim num1, num2 As Double
 Dim product As Double

 ExplainPurposeOfProgram()

 num1 = 5.5
 num2 = 4.0

 product = calcProduct(num1, num2)

 OutputResult(num1, num2, product)

End Sub

Sub ExplainPurposeOfProgram()

 MsgBox("The purpose of this program is to display two numbers and their
product", , "Purpose")

End Sub

Sub OutputResult(ByVal firstNumber, ByVal secondNumber, ByVal result)

 MsgBox("The product of " & firstNumber & " and " & secondNumber & " is " &
result, , "Result")

End Sub

Function calcProduct(ByVal firstNumber, ByVal secondNumber)

 Return firstNumber * secondNumber

End Function
```

Not only are the variables given meaningful names but the procedures and buttons have also. Each procedure name explains what the procedure does.

Some programmers also start variable identifiers with three letters that indicate their type. Thus an integer variable identifier would start with int such as intCount. Doubles start with dbl such as dblNum1. It is important that every programmer sticks to the standard set down by the company for which they work. Objects, such as buttons and text boxes in Visual Basic.NET, need to have meaningful names such as btnStart or txtAnswer. Leaving the default names (Button1 or Text1) unchanged can make a program very difficult to debug.

### **2.5.e** *Use declared constants to improve maintainability*

This has been explained in Section 5(a). Remember, it is much easier to search for pi or VAT than 3.141593 or 0.175 (which may be used as 17.5/100). Naming constants also ensures that the same value is used throughout the program. Sometimes using 3.14 and other times 3.141593 is not satisfactory.

## 2.5.f Initialise variables appropriately, before using them

Most programs give variables a value when they are declared. However, this value varies from one language to another and some languages do not initialise any variables. Not initialising variables is very bad practice. The initialisation may take the form of an assignment statement or the user may be asked to enter a value.

Look at this program:

```
Private Sub btnCalculate_Click(ByVal sender As System.Object, ByVal e
 As System.EventArgs) Handles btnCalculate.Click
 Dim num As Integer
 Dim num1, num2, num3 As Integer
 Dim count As Integer = 2

 num = InputBox("Enter a positive integer.", "Input")
 If num = 1 Or num = 2 Then
 MsgBox("The " & num & " th number is 1", , "Result")
 Else
 num1 = 1
 num2 = 1
 While count <> num
 num3 = num1 + num2
 num1 = num2
 num2 = num3
 count = count + 1
 End While
 MsgBox("The " & num & " th number is " & num3, , "Result")
 End If
End Sub
```

Notice that count is initialised when it is dimensioned. The initial value of num is entered by the user and num1 and num2 are initialised before they are used.

The following code initialises the elements of a 2-dimensional array to 0 before they are used:

```
Dim anArray(10, 20) As Integer
Dim row, column As Integer

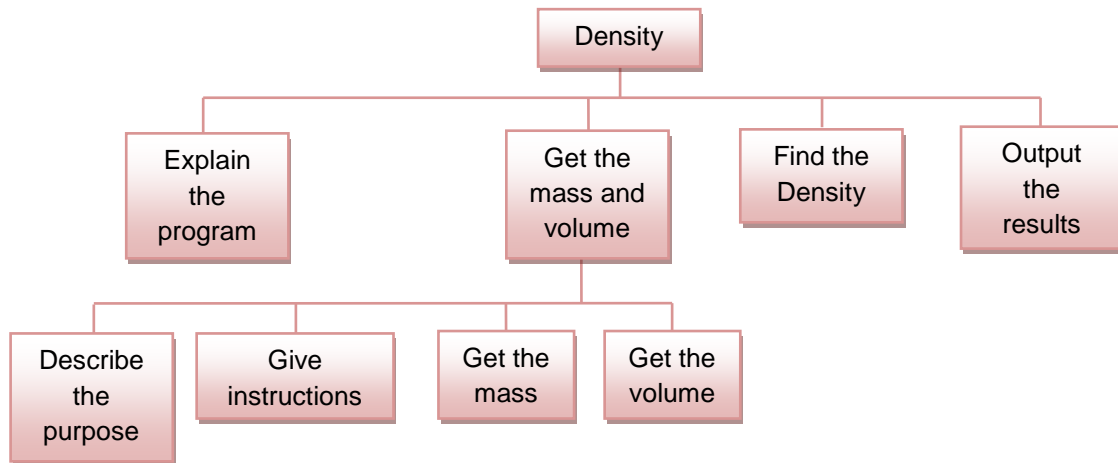
For row = 1 To 10
 For column = 1 To 20
 anArray(row, column) = 0
 Next
Next
```

**2.5.g** *Create appropriately modularised programs (following a top-down/modular design as produced in 3.2.1: Designing solutions to problems) making effective use of subroutines to improve maintainability*

This is explained in Section 2(f) but here is another example.

Write a program to calculate the density of an object given its mass and volume.

Design:



This design suggests using subroutines for

- Describe the purpose
- Give instructions
- Get the mass
- Get the volume
- Find the Density
- Output the results



Here is the code. Notice that the main program simply calls the procedures and functions:

```
Private Sub btnCalcDensity_Click(ByVal sender As System.Object, ByVal e
 As System.EventArgs) Handles btnCalcDensity.Click

 Dim mass As Double
 Dim volume As Double
 Dim density As Double

 Purpose()

 Instructions()

 mass = GetMass()
 volume = GetVolume()
 density = CalcDensity(mass, volume)

 OutputResults(mass, volume, density)

End Sub

Sub Purpose()

 Dim message As String
 message = "The program calculates the density "
 message = message & "of a substance given its mass and volume"

 MsgBox(message, , "Purpose")
End Sub

Sub Instructions()

 Dim message As String
 message = "When asked enter the mass of the object "
 message = message & "and then its volume"
 MsgBox(message, , "Instructions")

End Sub

Function GetMass()

 Return InputBox("Enter mass of object")

End Function

Function GetVolume()

 Return InputBox("Enter volume of object")

End Function

Function CalcDensity(ByVal mass, ByVal volume)

 Return mass / volume

End Function

Sub OutputResults(ByVal mass, ByVal volume, ByVal density)

 MsgBox("Mass = " & mass, , "Mass")
 MsgBox("Volume =" & volume, , "Volume")
 MsgBox("The density is " & density, , "Density")
End Sub
```

## 2.5.h *Annotate the code with comments so that the logic of the solution can be followed*

To make clear what a program is doing, the code should be annotated. The annotation should normally be written before the code so that the reader knows what is coming next. Also, the annotation should be such that if the code were removed, it would be possible to recreate it.

Here is one of the programs we saw earlier:

```
Private Sub btnStart_Click(ByVal sender As System.Object, ByVal e As
 System.EventArgs) Handles btnStart.Click

 Dim oldFile As IO.StreamReader
 Dim newFile As IO.StreamWriter
 Dim inserted As Boolean
 Dim name, newName As String
 Dim nextName As String

 oldFile = IO.File.OpenText("List Of Names.txt")
 newFile = IO.File.CreateText("New List Of Names.txt")

 newName = InputBox("Enter name to be inserted.", "New Name")
 inserted = False

 Do While oldFile.Peek <> -1 And Not inserted
 name = oldFile.ReadLine
 If name > newName Then
 newFile.WriteLine(newName)
 newFile.WriteLine(name)
 inserted = True
 Else
 newFile.WriteLine(name)
 End If
 Loop

 If Not inserted Then
 newFile.WriteLine(newName)
 Else
 Do While oldFile.Peek <> -1
 name = oldFile.ReadLine
 newFile.WriteLine(name)
 Loop
 End If

 oldFile.Close()
 newFile.Close()

 IO.File.Delete("List Of Names.bak")
 IO.File.Move("List Of Names.txt", "List Of Names.bak")
 IO.File.Move("New List Of Names.txt", "List Of Names.txt")

End Sub
```

Here is the same program suitably annotated:

```
Private Sub btnStart_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnStart.Click

 'Declare variables to be used
 Dim oldFile As IO.StreamReader 'name to be used for the
 'original file
 Dim newFile As IO.StreamWriter 'name to be used for the file
 'with new name inserted
 Dim inserted As Boolean 'inserted is true when new name
 'has been inserted into file
 Dim name, newName As String 'current name and name to be
 'inserted
 Dim nextName As String 'the next name in the list

 'Open files for reading from & writing to
 oldFile = IO.File.OpenText("List Of Names.txt")
 newFile = IO.File.CreateText("New List Of Names.txt")

 'Ask the user for the name to be inserted
 newName = InputBox("Enter name to be inserted.", "New Name")
 'Set inserted to FALSE to show name has not been inserted into file
 inserted = False

 'While there are still records to be searched and new name
 'has not been inserted, look at the next name in the file
 Do While oldFile.Peek <> -1 And Not inserted
 'Read the next name in the file
 name = oldFile.ReadLine

 If name > newName Then
 'insert the new name and then the old one into new file
 'and note that the name has been inserted
 newFile.WriteLine(newName)
 newFile.WriteLine(name)
 inserted = True
 Else
 'write old name to file
 newFile.WriteLine(name)
 End If
 Loop
 If Not inserted Then
 'insert the new name at the end of the list
 newFile.WriteLine(newName)
 Else
 'write the rest of the original file to new file
 Do While oldFile.Peek <> -1
 name = oldFile.ReadLine
 newFile.WriteLine(name)
 Loop
 End If

 'Close the files
 oldFile.Close()
 newFile.Close()

 'Delete the original file backup file
 IO.File.Delete("List Of Names.bak")

 'rename the original file as the backup
 IO.File.Move("List Of Names.txt", "List Of Names.bak")

 'rename the new file so that it can be used again by this program
 IO.File.Move("New List Of Names.txt", "List Of Names.txt")

End Sub
```

### **2.5.i** *Use indentation and formatting to show clearly the control structures within the code*

This has been shown in all the examples in this text, particularly in the program in the previous Section. It is good programming practice to insert blank lines between different control structures.

## 2.5 Example Questions



1 Define the following terms.

- (a) Variable, [1]
- (b) Constant, [1]
- (c) Reserved word [1]

2 Discuss the need for good programming techniques to facilitate the ongoing maintenance of programs [8]

A This is a new type of question and will be marked on both the content and the accuracy of grammar and the use of technical terms.

The examiner will be expecting you to include references to

- the use of meaningful names for variables, constants, subprograms and other objects
- The naming of constants rather than using explicit values
- The use of prefixes on identifier names to indicate their data type or object type
- The initialisation of variables
- Modularisation of programs
- Annotation of code
- Indentation of control structures

A list like the one above will not be enough. You will be expected to use prose because it is a discussion question; this means that you should explain the advantages of these techniques. It would also help to include some simple examples.

The marks will be banded into three categories; these are

- High level response (6-8 marks)
- Medium level response (3-5 marks)
- Low level response (0-2 marks)

*Note: These marks are only examples of how this type of question will be marked. Not every question of this type will have the same number of marks.*

## 2.6 Testing and Running a Solution



### 2.6.a *Describe types of errors in programs (Syntax, logic and run-time errors) and understand how and when these may be detected*

Syntax errors are errors in the grammar of the program language. That is, the rules of the language have been broken. Common errors are typing errors such as PINT for PRINT. However, other types of error are and IF statement without an ENDIF, a FOR statement without a NEXT statement, an opening parenthesis without a closing parenthesis for example ( a + b.

Syntax errors are picked up during syntax analysis. In the case of an interpreter the error is reported as soon as it is found but a compiler will list all the errors in the program together. Candidates should be aware that some editors can give syntax errors while the code is being entered. It is still the syntax analyser that spots the error. If asked, in an examination, when a syntax error is found, the answer is during syntax analysis.

A logic error occurs when the programmer makes a mistake in the logic of the program. Suppose a programmer wants the first 10 positive integers to be output. The programmer creates the following algorithm:

```
FOR count = 0 TO 10
 OUTPUT count
NEXT count
```

This algorithm has no grammatical errors but does not deliver the correct result. It produces the integers 0 to 10 not 1 to 10. This type of error can only be found by thorough testing. Another common error is to use the wrong arithmetic symbol,  $a + b$  instead of  $a - b$  or to put parentheses in the wrong place such as  $(a + b - c) * d$  instead of  $(a + b) - c * d$ . Again found during testing.

Run-time errors occur during the execution (running) of the program. That is, they are only detected during the execution of a program. A typical error is to have a formula, such as  $(a + b) / (c - d)$  in the program and at some point  $c$  becomes equal to  $d$  resulting in division by zero. The problem is that this error may be undetected for a considerable time because the equality of  $c$  and  $d$  rarely happens. However, it can be disastrous if it happens. It is a good idea to test the denominator before division to ensure that this error doesn't crash the program but an error message is produced.

## 2.6.b Identify why/where an error may occur in an algorithm and state how the algorithm may be corrected

Look at the following algorithm which is supposed to read n numbers into an array:

```
INPUT n
count = 1
REPEAT
 INPUT anArray(count)
 count = count + 1
UNTIL count = n
```

Suppose n = 2 and the two numbers are 5 and 10. A dry run produces the following values for the variables:

| n | count | anArray(1) | anArray(2) |
|---|-------|------------|------------|
| 2 | 1     | 5          |            |
|   | 2     |            |            |

count is now equal to n so the loop terminates so only one number is read, not two.

To correct this, initialise count to zero not 1:

```
INPUT n
count = 0
REPEAT
 INPUT anArray(count)
 count = count + 1
UNTIL count = n
```

The dry run now produced this table:

| n | count | anArray(1) | anArray(2) |
|---|-------|------------|------------|
| 2 | 0     |            |            |

Now, the algorithm tries to put the first number into anArray(0). Swap the two statements that are inside the loop:

```
INPUT n
count = 0
REPEAT
 count = count + 1
 INPUT anArray(count)
UNTIL count = n
```

Now we have:

| n | count | anArray(1) | anArray(2) |
|---|-------|------------|------------|
| 2 | 0     |            |            |
|   | 1     | 5          |            |
|   | 2     | 10         |            |

count is now equal to n, so exit the loop.

What happens if the user enters a zero value for n? The loop will be entered but there are no numbers to input. Change the REPEAT loop to a WHILE loop so that the loop will only be entered if n is positive:

```

INPUT n
count = 0
WHILE n > 0
 count = count + 1
 INPUT anArray(count)
ENDWHILE

```

This is an improvement because the loop will only be entered if a positive value for n is entered. However, this does not work either because there is no condition to terminate the loop when all the numbers have been input. Try it with the data used in the earlier versions. The next version is version is:

```

INPUT n
count = 0
WHILE n > 0 AND count <= n
 count = count + 1
 INPUT anArray(count)
ENDWHILE

```

and the dry run gives:

| n | count | anArray(1) | anArray(2) |
|---|-------|------------|------------|
| 2 | 0     |            |            |
|   | 1     | 5          |            |
|   | 2     | 10         |            |

Now the algorithm tries to enter a third number. One solution is to initialise count to 1 again and swap the two statements inside the loop:

```

INPUT n
count = 1
WHILE n > 0 AND count <= n
 INPUT anArray(count)
 count = count + 1
ENDWHILE

```

There is an alternative solution which is left to the reader to find.

These are all typical errors; Wrong initialisation, statements in the wrong order, wrong terminating values to the loop and the use of the wrong loop. The reader may think that using a FOR ... NEXT loop would be better. Before using a FOR ... NEXT loop, check when the loop is checked for termination because some compilers check it at the end of the loop. This means that the loop will always be executed once as in the case of the REPEAT loop. Modern compilers check for an end to the loop at the start like a WHILE statement, but just check.



### **2.6.c** *Describe testing strategies including white box testing, black box testing, alpha testing, beta testing and acceptance testing*

All software has to be tested. Only the simplest of programs work first time, most contain errors (or bugs) that have to be found. If a program has been written in modules, it is much easier to debug because you are only looking at a small piece of code at a time. There are several ways of testing a program.

White box testing is where testers examine each line of code for logic and accuracy. This may be done manually by drawing up a table to record the values of the variables after each instruction is executed. Alternatively, debugging software can be used to run the program step by step and then display the values of the variables. You will find more about this in Section 2.6.f.

Black box testing is one of the earliest forms of test. Here the programmer uses test data, for which the results have already been calculated and compares the results from the program with those that are expected. The programmer does not look at the individual code to see what is happening.

Alpha testing is done in the developer's own company. A number of employees who have not been involved in the production of the software are given the task of testing it. The software may not be complete at this stage and may have errors that the programmer has not found.

Beta testing is the next stage. This is done by privileged customers, including magazine reviewers, authors of independent books that tell users how to use the software and to other developers of software and hardware. At this stage the software is virtually complete but there may still be bugs in the system. These testers are expected to provide constructive criticism.

Finally, acceptance testing has to be carried out. At this stage, the software is complete and the developer has to prove to the customer that it does everything specified in the design and is in the original requirements specification.

### 2.6.d *Select suitable test data for a given problem, including normal, borderline and invalid data*

It is impossible to test every possible value that may be used. Suppose you wish to input a date of birth. You can make sure it is in a reasonable range by using a range test, but you cannot test every value inside that range. You must examine the problem and then choose data that can be used to test the program. Notice that you do not look at the program to choose the test data. This is because the program may appear to work without any bugs, but it does not prove that it correctly solves the problem.

Consider the problem of inputting a value that lies between 5.5 and 9 inclusive.

Look at this solution.

```
INPUT n
WHILE n <= 5.5 OR n >= 9
 INPUT n
ENDWHILE
```

Looking at the code we might choose as normal data  $n = 6$  and find that this is not less than 5.5 or greater than 9 so it will be accepted.

Now let us try a borderline value such as 5.5. This will be rejected because  $n \leq 5.5$  is TRUE. This means that this algorithm works for  $n = 5.5$ . However, that is not correct for the problem which says that the borderline values should be accepted. This means that the test data should be chosen by examining the problem not the solution. You are trying to prove that the solution solves the problem correctly.

In this case, any value between 5.5 and 9 (excluding 5.5 and 9) are normal data, 5.5 and 9 are called borderline data and invalid data could be 4, 10 or even 2k.

## 2.6.e Perform a dry run on a given algorithm using a trace table

Consider the following algorithm:

```
1 n = 0
2 WHILE n <= 0
3 INPUT n
4 ENDWHILE
5 Total = 0
6 Count = 0
7 REPEAT
8 INPUT Number
9 Total = Total + Number
10 Count = Count + 1
11 UNTIL Count = n
12 OUTPUT Total/Count
```

Show the results of a dry run using the following data:

3, 2, 5, 8

| Instruction | n | Total | Count | Number | Boolean | Output |
|-------------|---|-------|-------|--------|---------|--------|
| 1           | 0 |       |       |        |         |        |
| 2           |   |       |       |        | TRUE    |        |
| 3           | 3 |       |       |        |         |        |
| 4           |   |       |       |        |         |        |
| 5           |   | 0     |       |        |         |        |
| 6           |   |       | 0     |        |         |        |
| 8           |   |       |       | 2      |         |        |
| 9           |   | 2     |       |        |         |        |
| 10          |   |       | 1     |        |         |        |
| 11          |   |       |       |        | FALSE   |        |
| 8           |   |       |       | 5      |         |        |
| 9           |   | 7     |       |        |         |        |
| 10          |   |       | 2     |        |         |        |
| 11          |   |       |       |        | FALSE   |        |
| 8           |   |       |       | 8      |         |        |
| 9           |   | 15    |       |        |         |        |
| 10          |   |       | 3     |        |         |        |
| 11          |   |       |       |        | TRUE    |        |
| 12          |   |       |       |        |         | 5      |

Notice that you only need to record a variable's value if it changes.

Try doing a dry run using the data given in Section 2.1.h.

### **2.6.f** *Describe the use of a range of debugging tools and facilities available in procedural programming languages including translator diagnostics, break points, stepping, and variable checks*

During translation of a high-level language to machine code (or intermediate code) syntax errors will be spotted and reported. Some editors will report a syntax error when the code is typed in. For example, if you write the instruction:

```
mass = 56
```

and mass has not been declared, some editors will report this immediately. In any case, it will be reported during translation by either an interpreter or a compiler.

Many languages have an Integrated Development Environment (IDE). This enables the programmer to use some very sophisticated techniques. One such technique is to arrange for a program to stop at a given instruction and then to display the values of the variables at this point. The program can then be continued or stopped by the programmer.

A programmer may step through a program either by setting breakpoints or by stopping after the execution of each instruction. When a programmer sets a breakpoint, the program can then be run to this point. Next the programmer can step through the next instructions one at a time or run the program to the next breakpoint or to the end of the program.

A Watch Window can be used to display the values of variables and expressions. The window tells the programmer the value and data type of each variable and expression. To obtain these values, the programmer simply types the name of the variable or the expression into the window and the value and data type is displayed. This enables the programmer to check not only a variable's value but also its type.

You should experiment with the facilities that come with your programming language.

### **2.6.g** *Describe the purpose of an installation routine in the delivered version of the program*

When a program is delivered to the customer it is usually in encrypted form, it may also be compressed. The installation routine copies the program to the customers computer, decompresses it (if necessary), decrypts it and installs the various components onto the customer's system. The customer may be asked to name a folder into which the software is to be installed; otherwise the routine installs it into a default folder. Any links between the different modules can then be set up. The routine may also allow the customer to install the software a limited number of times and protect the original software from being copied.

## 2.6 Example Questions



1 (a) With the aid of an example, describe the meaning of the following terms and state when they are detected.

(i) syntax error,

(ii) run-time error.

[6]

A (i) Syntax error:

- Eg FOR without NEXT
- An error in the grammar of the language ...
- detected during syntax analysis

[3]

(ii) Run-time error:

- Eg Division by zero
- A statement that cannot be executed ...
- when the program is running
- 

[3]

2 The following algorithm is supposed to output the sum of n numbers where the value of n is to be input followed by the n numbers.

State the error that it contains and correct this error.

```
INPUT n
FOR count = 1 TO n
 INPUT nextNumber
 sum = sum + nextNumber
NEXT count
OUTPUT sum
```

[2]

A sum has not been initialised  
Insert sum = 0 before the FOR loop starts

[2]

3 Explain the terms

(i) alpha testing,

[3]

(ii) black box testing,

[3]

(iii) acceptance testing.

[3]

A (i) Alpha testing

- Done by a team of testers
- ... who work for the company developing the s/w
- The software may not be complete [3]

(ii) Black box testing

- Done by the programmer ...
- ... using data for which the output is known ...
- ... and has been chosen from the program
- The individual instruction of the program are not known [3]

(iii) Acceptance testing

Tests the final version of the program ...

... before it is accepted by the client

Compares the results with the original specification ...

... that was agreed with the client [3]

- 4 A program is required that inputs a mark out of 100 for an examination. All marks are integers. State five values that can be used during alpha testing, giving a reason for your choice.

A These are examples and other similar values are acceptable.

| Test Data | Reason                                     |
|-----------|--------------------------------------------|
| 0         | Borderline value                           |
| -5        | Negative values not allowed (invalid data) |
| 60        | Valid data                                 |
| 45.8      | Not an integer                             |
| 140       | Too large (invalid data)                   |

- 5 Use a trace table to do a dry run on the following algorithm. The input data is:

2, 5, -6, 0

```
1 negCount = 0
2 posCount = 0
3 INPUT number
4 WHILE number <> 0
5 IF number < 0 THEN
6 negCount = negCount + 1
7 ELSE
8 posCount = posCount + 1
9 ENDIF
10 INPUT number
11 ENDWHILE
12 OUTPUT negCount, posCount [5]
```

A *Note: In an examination you would usually be given a blank table to complete.*

| Instr    | negCount | posCount | number | Boolean | Output |
|----------|----------|----------|--------|---------|--------|
| 1        | 0        |          |        |         |        |
| 2        |          | 0        |        |         |        |
| 3        |          |          | 2      |         |        |
| 4        |          |          |        | TRUE    |        |
| 5        |          |          |        | FALSE   |        |
| 7, 8     |          | 1        |        |         |        |
| 9, 10    |          |          | 5      |         |        |
| 11, 4    |          |          |        | TRUE    |        |
| 5        |          |          |        | FALSE   |        |
| 7, 8     |          | 2        |        |         |        |
| 9, 10    |          |          | -6     |         |        |
| 11, 4    |          |          |        | TRUE    |        |
| 5        |          |          |        | TRUE    |        |
| 6        | 1        |          |        |         |        |
| 7, 9, 10 |          |          | 0      |         |        |
| 11, 4    |          |          |        | FALSE   |        |
| 12       |          |          |        |         | 1, 2   |

[5]

6 Describe three debugging tools that a programmer may use to debug a program.

[9]

A Any three of the tools that are described in Section 6.f may be described. There would normally be 1 mark for naming the tool and 2 marks for the description of the tool. You will give a better description if you have used these tools.

[9]



## 3.1 The Function of Operating Systems



### 3.1.a Describe the main features of operating systems – for example, memory management, scheduling algorithms

The operating system (OS) must:

- provide and manage hardware resources
- provide an interface between the user and the machine
- provide an interface between applications software and the machine
- provide security for the data on the system
- provide utility software to allow maintenance to be done

Originally, if an application program needed input, the program would have to contain the code to do this, similarly if output were required. This was true for every program and often multiple occurrences within a program. This led to duplication of code so the idea of an OS was born. The OS contained the necessary input and output functions that could be called by an application. Similarly, disk input and output routines were incorporated into the OS. Short pieces of program code were developed (called subroutines) to do these simple tasks such as read a character from a keyboard or send a character to a printer. The joining together of all these basic input and output routines led to the input-output control system (IOCS). Originally, (in the 1950's) the IOCS could only read a punched card or send data to a card punch. However, as new input and output media, such as magnetic tape and disk, were developed the IOCS became more complex.

Another complication was added when assembly and high-level languages were developed as the machine did not use these languages. Machines use binary codes for very simple instructions. With the development of these new types of programming language a computer would have to

- have a translator program to turn human friendly code into computer friendly code
  - an assembler to handle translation from assembly language (low level language) which is a simple language based on mnemonics for instructions and labels for memory addresses
  - a compiler, which translates the high level language code into object code which is in a form that the computer can understand
  - an interpreter, which translates a line of the high level code and runs the result before translating the next line
- load the assembly or high-level program,
- do the translation,
- store the results somewhere in memory,
- execute the program,
- read input data,
- output the results.

For a user to organise all this had now become too complex. Also, as the processor could work much faster than the manual operator and the input and output devices, much time was wasted if a human being tried to do it.

Further, to make full use of the processor, more than one program should be stored in memory and the processor should give time to each of the programs. Suppose two programs are stored in memory and, if one is using an input or output device (both very slow compared to the processor), it makes sense for the other program to be able to use the processor. In fact this can be extended to more than two programs as shown in Fig. 3.1.1

The OS must now manage the memory so that all three programs shown in Fig. 3.1.1 are kept separate as well as any data that they use. It must also schedule the jobs in the sequence that makes best use of the processor.

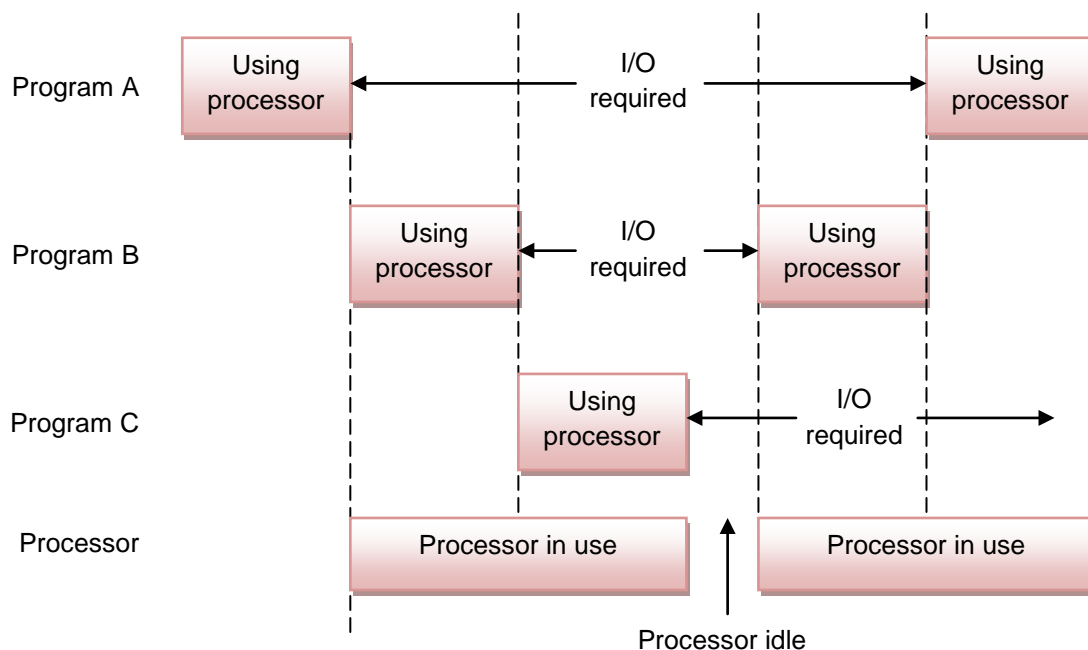


Fig. 3.1.1

The I/O phase should not hold up the processor too much which can easily happen if the I/O devices are very slow, like a keyboard or printer. This can be overcome by using Simultaneous Peripheral Operations On-Line (DON'T try to remember this, everyone calls it spooling). The idea of spooling is to store all input and output on a high-speed device such as a disk. Fig. 3.1.2 shows how this may be achieved,

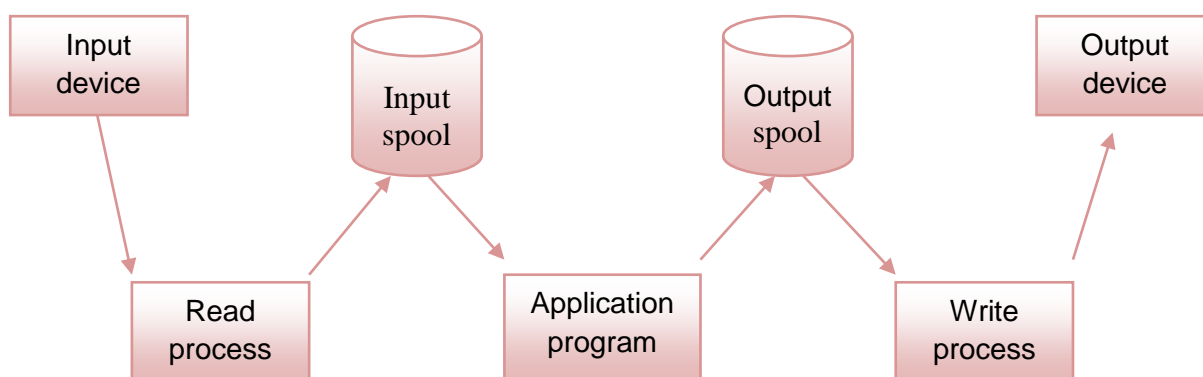


Fig. 3.1.2

Another problem is that programs may not be loaded into the same memory locations each time they are loaded. For example, suppose that three programs are loaded in the order A, B, C on one occasion and in the order C, A, B on another occasion. The results are shown in Fig. 3.1.3.

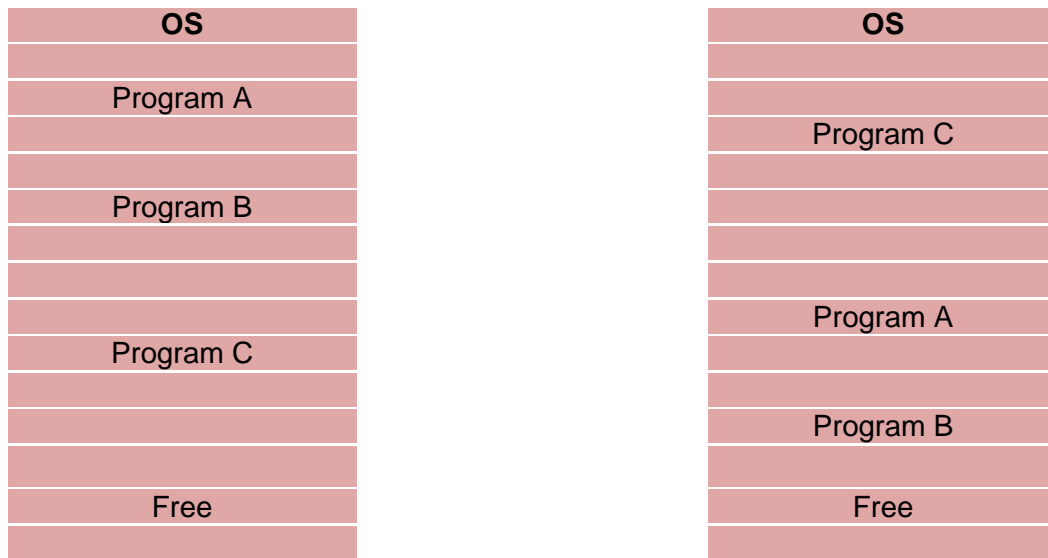


Fig. 3.1.3

The processor needs to know whereabouts everything is otherwise it cannot find an important piece of data, or the next instruction. This movement of things around inside memory provides a major problem for the operating system which is solved by a special little program called the loader which can remember exactly where everything is every time it is loaded and can also make sure that bits of program loaded into different places can stay in touch with each other.

A further problem occurs if two or more users wish to use the same program at the same time. For example, suppose user X and user Y both wish to use a compiler for at the same time. Clearly it is a waste of memory if two copies of the compiler have to be loaded into main memory. It would make much more sense if user X's program and user Y's program are stored in main memory along with a single copy of the compiler as shown in Fig. 3.1.4.

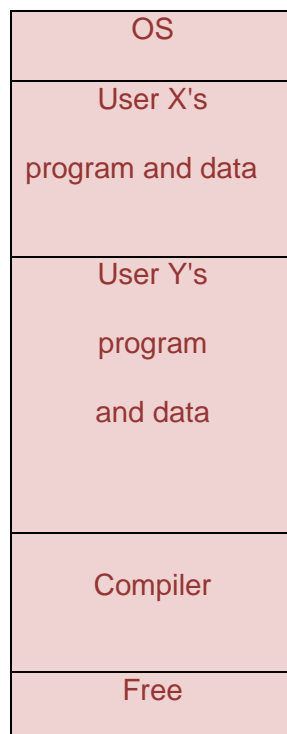


Fig. 3.1.4

Now, the two users can use the compiler in turns and will want to use different parts of the compiler. Also note that there are two different sets of data for the compiler, user X's program and user Y's program. These two sets of data and the outputs from the compiler for the two programs must be kept separate. Programs such as this compiler, working in the way described, are called re-entrant.

The process of the operating system being able to control the use of the computer's memory, rather than the user doing it, is known as memory management.

When there is more than one thing which could be done at a time something has to decide what will be done first, this process of deciding on order of jobs is known as scheduling and the program that makes the decisions is known as a scheduling algorithm.

Memory management, scheduling and spooling are described in more detail in the following Sections.

**3.1.b** *Explain how interrupts are used to obtain processor time and how processing of interrupted jobs may later be resumed, (typical sources of interrupts should be identified and any algorithms and data structures should be described)*

The simplest way of obeying instructions is shown in Fig. 3.1.5

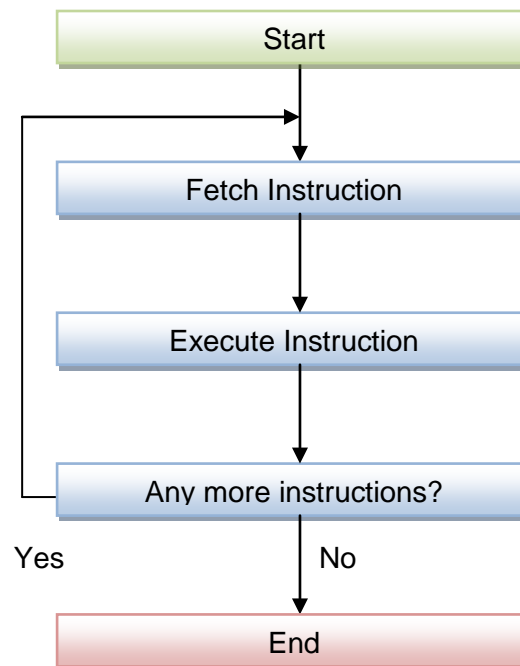


Fig. 3.1.5

This is satisfactory so long as nothing goes wrong and the program being run maintains the control of the processor. Unfortunately things do go wrong and sometimes the normal order of operation needs to be changed. For example, if a user has used up all the time allocated to his use of the processor it will be necessary to stop that program running in order to be fair to all users (Note that this is a scheduling algorithm. The operating system is distributing processor time according to what is 'fair'). This change in order is instigated by interrupts, which are messages sent to the processor by some external entity which basically say 'Can you stop what you are doing and give me some time please.' There are a number of different types of interrupt:

- I/O interrupt
  - Generated by an I/O device to signal a job is complete or an error has occurred. E.g. printer is out of paper or is not connected.
- Timer interrupt
  - Generated by an internal clock indicating that the processor must attend to time critical activities (see 3.1.c).
- Hardware error
  - For example, power failure which indicates that the OS must close down as safely as possible.
- Program interrupt
  - Generated due to an error in a program such as violation of memory use (trying to use part of the memory reserved by the OS for other use) or an attempt to execute an invalid instruction (such as division by zero).

If the OS is to manage interrupts, the sequence in Fig. 3.1.5 needs to be modified as shown in Fig. 3.1.6.

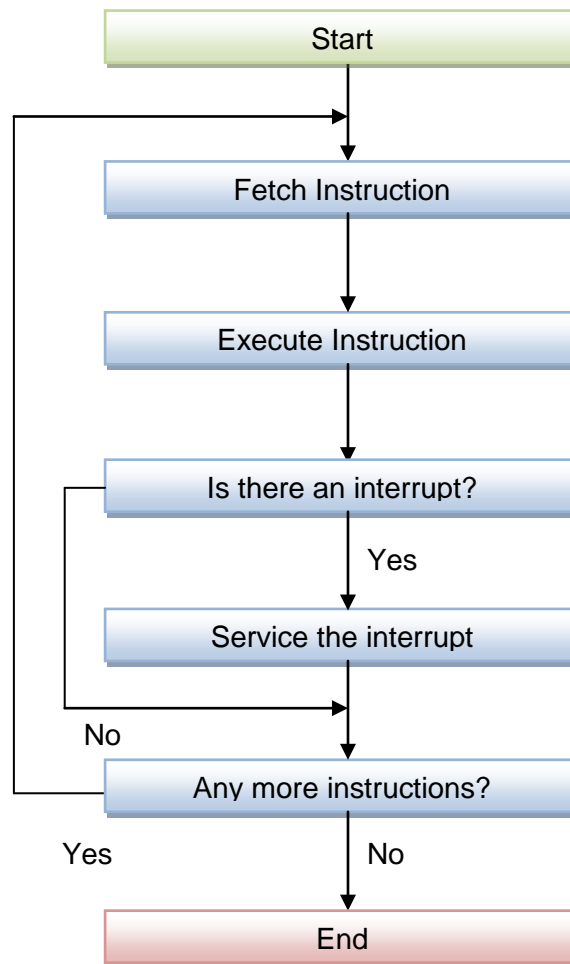


Fig. 3.1.6

This diagram shows that, after the execution of an instruction, the OS must see if an interrupt has occurred. If one has occurred, the OS must service the interrupt if it is more important than the task already being carried out (see 3.1.c). This involves obeying a new set of instructions. There is no problem in doing this, after all the computer can follow sets of instructions very easily, but a problem arises from the fact that the original program has not been finished. If we are not careful and interrupts are used too often we will end up with no programs actually being finished. The real problem here is 'how can the OS arrange for the interrupted program to resume from exactly where it left off?' In order to do this the contents of all the registers in the processor (see 1.4.b) must be saved so that the OS can use the registers to carry out whatever the interrupt wants it to do. Section 3.3 describes the registers that have to have their contents stored as well as explaining the fetch/execute cycle in detail.

Another problem the OS has to deal with happens if an interrupt occurs while another interrupt is being dealt with. There are several ways of dealing with this but the simplest is to place the interrupts in a queue and only allow return to the originally interrupted program when the queue is empty. Alternative systems are explained in Section 3.1.c. Taking the simplest case, the order of processing is shown in Fig. 3.1.7.

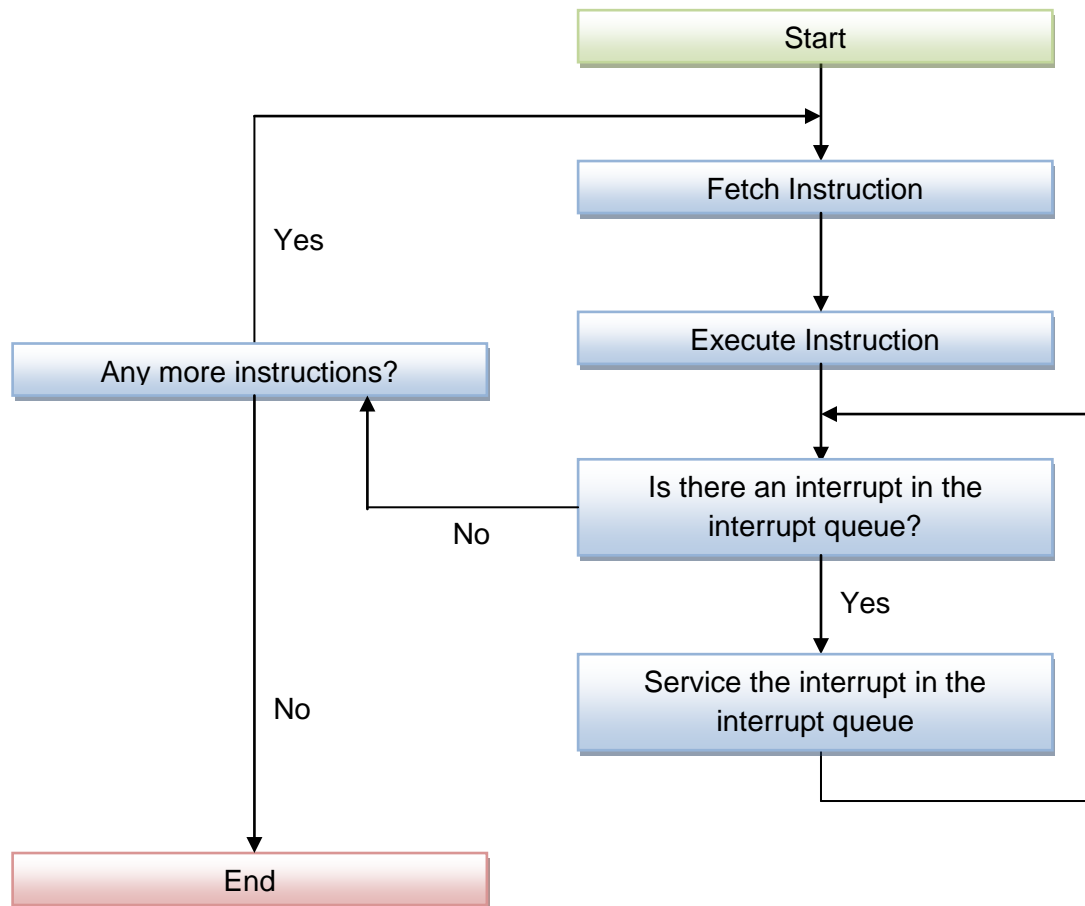


Fig. 3.1.7

The interrupts are held in a queue very like the queue of customers waiting at the checkout in a supermarket. The position of individual interrupts in the queue is determined by their importance to the user, to the system and to the job itself. Each interrupt, or job, is given a 'priority', the more important the job, the higher the priority and the further along the queue it is stored. This idea of priorities is covered in detail in 3.1.c and 3.5

**3.1.c** *Define and explain the purpose of scheduling, job queues, priorities and how they are used to manage job throughput*

One of the tasks of the OS is to arrange the jobs that need to be done into an appropriate order. The order may be chosen to ensure that maximum use is made of the processor; another order may make one job more important than another. In the latter case the OS makes use of priorities.

Suppose the processor is required by program A, which is printing wage slips for the employees of a large company, and by program B, which is analysing the annual, world-wide sales of the company which has a turnover of many millions of pounds.

Program A makes little use of the processor because the calculations are very minor whereas it needs a lot of printing to be done and it also has to get lots of data from the disk drive. Because most of the things that this job does are concerned with Input and Output jobs like this are called I/O bound jobs. Program B makes a great deal of use of the processor, only needing to collect one set of data from the drive and producing one lot of printout at the end. Jobs like this are said to be processor bound.

If program B has priority over program A for use of the processor, it could be a long time before program A can print any wage slips because all program B's processing will need to be done before A gets a look in! It seems much fairer to let A do a bit of processing and then get on with using the printer (which will take a lot of time because of the slow speed of the printer) while B is allowed to use the processor.

Fig 3.1.8 shows what happens if program B has priority over A. Notice that the printer does not get used very much.

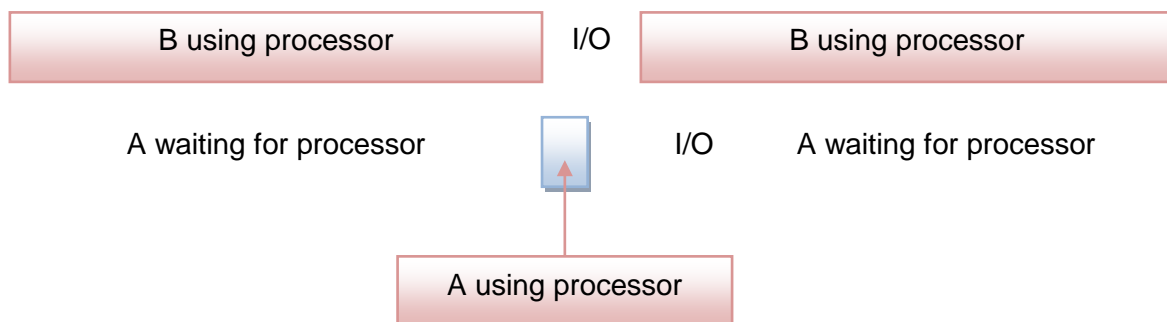


Fig. 3.1.8



Fig. 3.1.9 shows what happens if A is given priority over B. This shows that the I/O bound program can still run in a reasonable time and the printer is used far more.

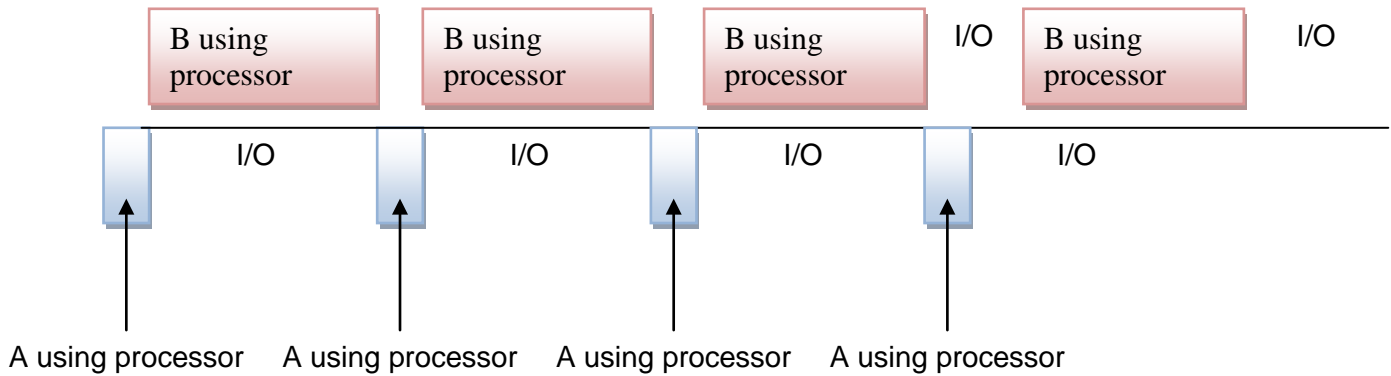


Fig. 3.1.9

The objectives of scheduling are to:

- maximise the use of the whole of the computer system;
- be fair to all users;
- provide a reasonable response time to all users, whether they are on-line users (Note: be careful, remember 'on-line' has nothing to do with the Internet) or a batch processing user;
- prevent the system failing if it is becoming overloaded;
- ensure that the system is consistent by always giving similar response times to similar activities from day to day.

To achieve these objectives some criteria are needed in order to determine the order in which jobs are executed. The following is a list of criteria, each of which may be used to determine a schedule which will achieve the above objectives. The OS can use one, or a mixture of two or more, of the following rules for determining the way the jobs will be scheduled in a system.

- Priority. Give some jobs a greater priority than others when deciding which job should be given access to the processor.
- I/O or processor bound. If a processor bound job is given the main access to the processor it could prevent the I/O devices being serviced efficiently, so I/O bound jobs should be done first.
- Type of job. Batch processing and real-time jobs require different response times.
- Resource requirements. The amount of time needed to complete the job, the memory required, I/O and processor time.
- Resources used so far. The amount of processor time used so far, how much I/O used so far.
- Waiting time. The time the job has been waiting to use the system.

In order to understand how scheduling is accomplished it is important to realise that any job may be in one, and only one, of three states. A job may be *ready* to start, *running* on the system or *blocked* because, for example, it is waiting for a peripheral. Fig. 3.1.10 shows how jobs may be moved from one state to another. Note that a job can only enter the *running* state from the *ready* state. The *ready* and *blocked* states are queues that may hold several jobs. On a standard, single processor, computer only **one** job can be in the *running* state at a time. Also, all jobs entering the system normally enter via the *ready* state and (normally) only leave the system from the *running* state.

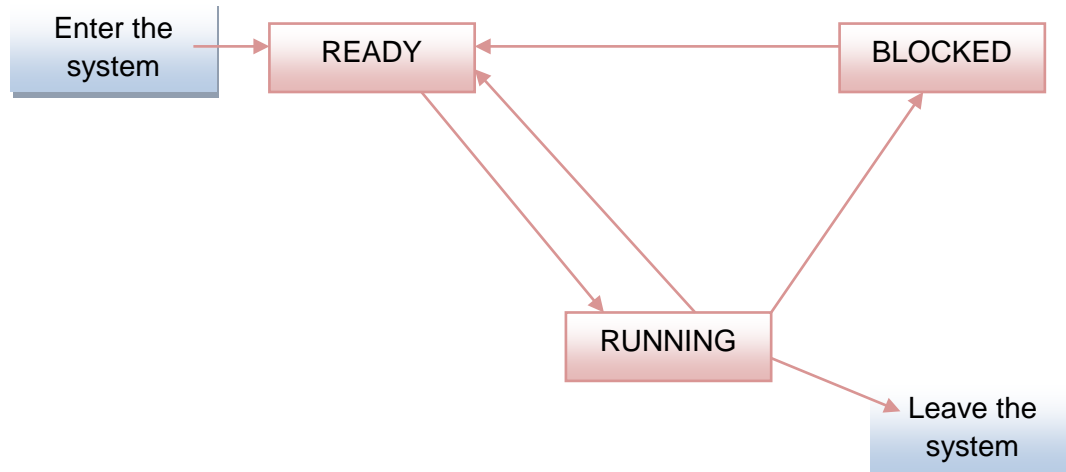


Fig. 3.1.10

When entering the system a job is placed in the ready queue by the High Level Scheduler (HLS). The HLS makes sure that the system is not over loaded.

Sometimes it is necessary to swap jobs between the main memory and backing store (see Memory Management in Section 3.1.d). This is done by the Medium Level Scheduler (MLS).

Moving jobs in and out of the ready state is done by the Low Level Scheduler (LLS). The LLS decides the order in which jobs are to be placed in the running state.

Do not be put off by all these technical terms. The scheduler is a program, a set of instructions, used by the OS to decide where each of the jobs which are in its control should be and in what order to manipulate them. All this talk of 'high level', 'medium level' and 'low level' schedulers just refer to different sections of the same program.

There are many ways of deciding the rules that may be used to do scheduling, but they can all be placed in one of two classes. One type of scheduler puts a job in the running state and then has to leave it there until the job does not need to run any more but needs to go to one of the other states, for example it may wait for an input or may have finished executing altogether. Schedulers that do this are called non-pre-emptive schedulers.

A scheduler that has the power to decide that the job that is running should be made to stop to make way for another job is called a pre-emptive scheduler.

Some common ways of deciding how jobs should be scheduled, known as scheduling policies, are:

- First Come First Served (FCFS)
- Shortest Job First (SJF)
- Round Robin (RR)
- Shortest Remaining Time (SRT)
- Multi-level Feedback Queues (MFQ)

and there are many more.

- **FCFS**
  - simply means that the first job to enter the ready queue is the first to enter the running state. This favours long jobs because once in the running state there is nothing to stop them carrying on running.
- **SJF**
  - simply means sort jobs in the ready queue in ascending order of times expected to be needed by each job. New jobs are added to the queue in such a way as to preserve this order.
- **RR**
  - this gives each job a maximum length of processor time (called a time slice) after which the job is put at the back of the ready queue and the job at the front of the queue is given use of the processor. If a job is completed before the maximum time is up it leaves the system.
- **SRT**
  - the ready queue is sorted on the amount of expected time still required by a job. This scheme favours short jobs even more than SJF. Also there is a danger of long jobs being prevented from running because they never manage to get to the front of the queue.
- **MFQ**
  - involves several queues of different priorities with jobs migrating downwards.

There are other ways of allocating priorities. Safety critical jobs will be given very high priority, on-line and real time applications will also have to have high priorities. For example, a computer monitoring the temperature and pressure in a chemical process whilst analysing results of readings taken over a period of time must give the high priority to the control program. If the temperature or pressure goes out of a pre-defined range, the control program must take over immediately. Similarly, if a bank's computer is printing bank statements over night and someone wishes to use a cash point, the cash point job must take priority. This scheme is shown in Fig. 3.1.11; this shows that queues are needed for jobs with the same priority.



Fig. 3.1.11

In this scheme, any job can only be given use of the processor if all the jobs at higher levels have been completed. Also, if a job enters a queue that has a higher priority than the queue from which the running program has come, the running program is placed back in the queue from which it came and the job that has entered the higher priority queue is placed in the running state.

Multi-level feedback queues work in a similar way except that each job is given a maximum length of processor time. When this time is up, and the job is not completely finished, the job is placed in the queue which has the next lower priority level. At the lowest level, instead of a first in first out queue a round robin system is used. This is shown in Fig. 3.1.12.

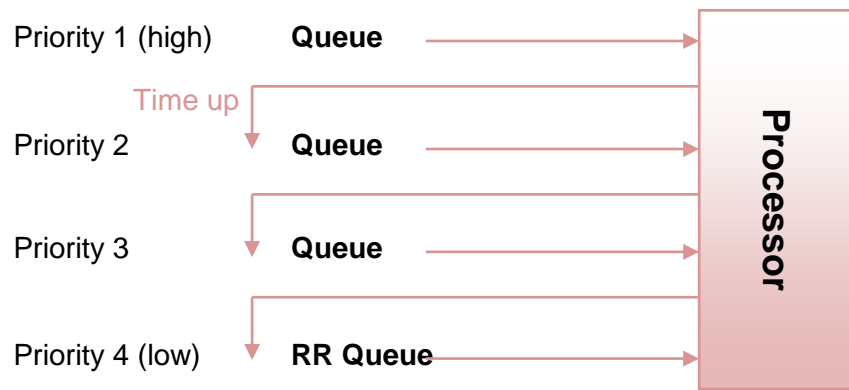


Fig. 3.1.12

**3.1.d** *Explain how memory is managed in a typical modern computer system, (virtual memory, paging and segmentation should be described along with some of the problems which could occur such as disk thrashing)*

In order for a job to be able to use the processor the job must be stored in the computer's main memory. If there are several jobs to be stored, they, and their data, must be protected from the actions of other jobs.

Suppose jobs A, B, C and D require 50k, 20k, 10k and 30k of memory respectively and the computer has a total of 130k available for jobs. (Remember the OS will require some memory.) Fig. 3.1.13 shows one possible arrangement of the jobs.

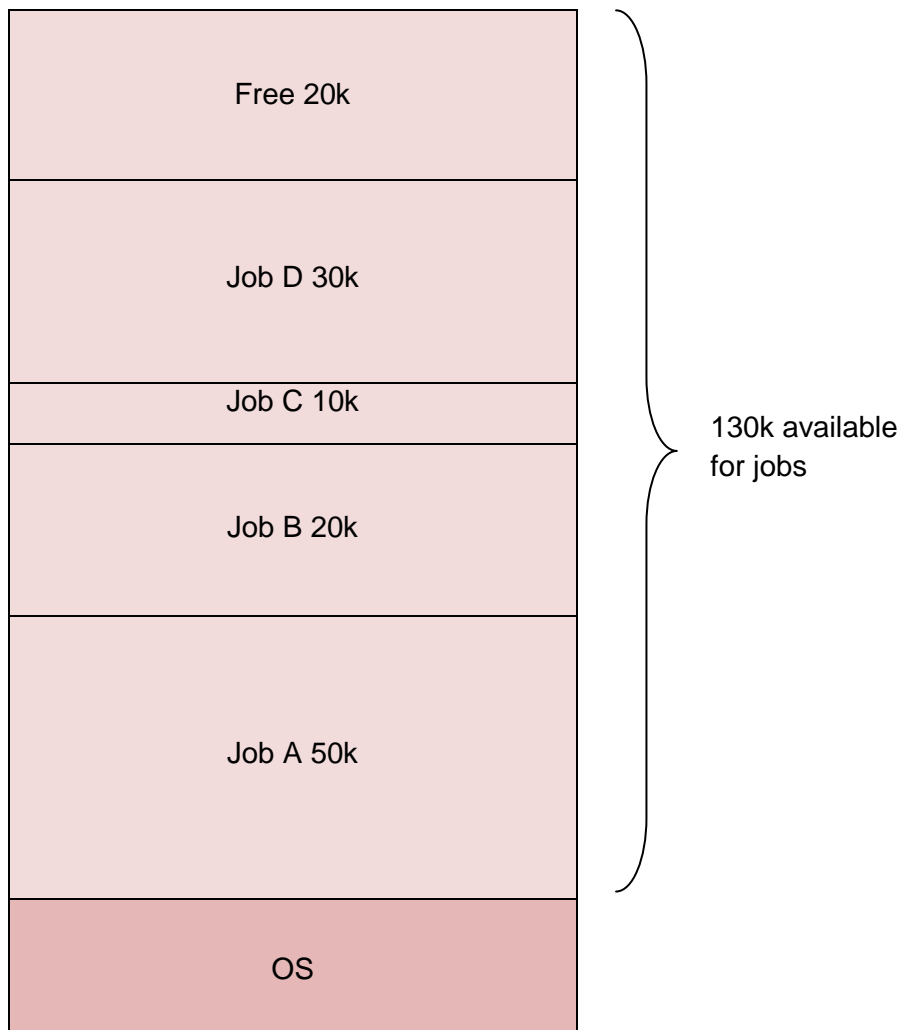


Fig 3.1.13

Now suppose job C terminates and job E, requiring 25k of memory, is next in the ready queue. Clearly job E cannot be loaded into the space that job C has relinquished. However, there is  $20k + 10k = 30k$  of memory free in total. So the OS must find some way of using it. One solution to the problem would be to move job D up to job B. This would make heavy use of the processor as not only must all the instructions be moved but all addresses used in the instructions would have to be recalculated. Another way would be to leave A, B and D where they are and to split job E into two parts, storing one part where job C has been and the other part in the free space area.

When jobs are loaded into memory, they may not always occupy the same locations. Supposing, instead of jobs A, B, C and D being needed and loaded in that order, it is required to load jobs A, B, D and E in that order. Now job D occupies different locations in memory to those shown above. So again there is a problem of using different addresses.

### Linkers and Loaders

The OS has the task of both loading the jobs and adjusting the addresses. The loader is a small program that does both these tasks. The calculation of addresses can be done by recalculating each address used in the instructions once the address of the first instruction is known. Alternatively, relative addressing can be used, that is, addresses are specified relative to the first instruction. This is beginning to get very complicated which is why it really should have a section in its own right called Memory Management. The loader is a program that helps the OS manage the use of memory while another program copes with difficulties like linking parts of programs together and is unsurprisingly called a linker. Linkers and loaders are integral to the management of the memory space.

### Paging and Segmentation

Notice in Fig 3.1.13 that the memory has been divided into equal sized sections. When this is done the sections are called pages. Pages are all of equal size and the different jobs are allocated a number of different pages to store themselves in. These pages may be in order as they are in 3.1.133 or may be scattered about inside the memory as those allocated to job E will be because it needs to be in different areas of the memory in order to have enough space. Notice that the sizes of these pages are fixed beforehand and may not be totally convenient for the jobs being stored. Perhaps in the case of job E the program instructions need 21k of memory while the data needs 4k. If the work has to be split for storage the logical way to do it would be 21k and 4k, but this does not fit the predetermined page size of 10k. A characteristic of paging is that the divisions are predetermined and the data/programs have to be squeezed in as best they can.

Another way of splitting the job up for storage is to see what the logical requirements of the job are. Job E may have 8k of instructions, 4k of data and 13k of help files for the user. The logical thing to do then is to divide the memory space required into three sections with sizes 4k, 8k and 13k. Now, these sizes required cannot be predicted in advance so it relies on the OS (or to be precise the linker and loader) to store the pieces in memory. These 'pieces' are called segments and the method of use of memory is called segmentation.

Segmentation is far more complex to control than paging because of the different and unpredictable nature of the sizes of the segments.

### Virtual memory and disk thrashing

Paging and segmentation lead to another important technique called virtual memory. We have seen that jobs can be loaded into memory when they are needed using a paging technique. When a program is running, only those pages that contain code that is needed need be loaded. For example, suppose a word processor has been written with page 1 containing the instructions to allow users to enter text and to alter the text. Also suppose that page 2 contains the code for formatting characters, page 3 the code for formatting paragraphs and page 4 contains the code for cutting, copying and pasting. To run this word processor only page 1 needs to be loaded initially. If the user then wants to format some characters so that they are in bold, then page 2 will have to be loaded. Similarly, if the user wishes to copy and paste a piece of text, page 4 will have to be loaded. When other facilities are needed, the appropriate page can be loaded. If the user now wishes to format some more characters, the OS does not need to load page 2 again as it is already loaded.

The loading of pages of information from the hard drive takes a disproportionate amount of time. This has led to small amounts of fast access storage between the drive and the memory. This storage is known as virtual memory because it 'is virtually as fast to access as the memory'. The OS now has part of the program in memory and part in the hard drive. What it needs to do is to predict which pages are most likely to be accessed next and store those in the virtual memory.

Now, what happens if there is insufficient space for the new page to be loaded? As only the page containing active instructions needs to be loaded, the new page can overwrite a page that is not currently being used. For example, suppose the user wishes to use paragraph formatting; then the OS can load page 3 into the memory currently occupied by page 2. Clearly, this means that programs can be written and used that are larger than the available memory.

The OS now has to both load and save pages. If the memory is very full, this loading and saving can use up a great deal of time and can mean that most of the processor's time is involved in swapping pages. This situation is called disk thrashing.

Systems can use both multi-programming (many programs in the memory simultaneously in order to use the power of the processor efficiently) and virtual memory. Also, virtual memory can use segmentation as well as paging although this can become very complex.

### 3.1.e *Describe spooling and how it is used*

Spooling was mentioned in 3.1.a. It is used to place input and output on a fast access storage device, such as a disk, so that slow peripheral devices do not hold up the processor. In a multi-programming, multi-access or network system, several jobs may wish to use the peripheral devices at the same time. It is essential that the input and output for different jobs do not become mixed up. This can be achieved by using spooling.

If two or more jobs, for instance, are sent to a printer at similar times (a perfect example would be in a classroom at the end of a lesson when everyone wants a printout) the printer can only cope with one at a time so the jobs are sent to a spool queue where they are queued up ready for the printer to deal with them in order.

Be careful. The last paragraph is the usual explanation of spooling but it gives the wrong impression. What actually happens is that the jobs for the printer are stored as files on the hard drive (or somewhere similar). The only thing that is stored in the spool queue is the reference to where the job is. When the reference to that job gets to the top of the queue the file is retrieved from storage and sent to the printer (or whatever peripheral device we are dealing with). Another problem is that a spool queue is not really a queue. One of the facts about a queue is that new jobs (or references to them) go at the end, otherwise you would have jobs pushing in and that would be unfair. Well, that is exactly what can happen in a spool queue, the more important jobs can be inserted into the queue, in other words, they jump the queue. This is because some jobs are more important than others.

So, when talking about a spool queue of jobs remember that it isn't really a queue of the jobs, only references to them and it is something of a strange queue because it allows pushing in.

It should be noted that spooling not only keeps output from different jobs separate, it also saves the user having to wait for the processor until the output is actually printed by a printer (a relatively slow device) and it lets the processor get on with something else while the jobs are queued up. Spooling is used on personal computers as well as large computer systems capable of multi-programming.



### **3.1.f** *Describe the main components of a typical desk top PC operating system, including the file allocation (FAT) and how it is used and the purpose of the boot file*

There are basically two types of OS used on PC's which are distinguished by their user interface. These are command driven and those that use a graphical user interface (GUI). Probably the best known of these are MS-DOS (command driven) and Windows (GUI). These differ in the way the user uses them and in the tasks that can be carried out.

All OS's for PC's allow the user to copy, delete and move files as well as letting the user create a hierarchical structure for storing files. They also allow the user to check the disk and tidy up the files on the disk.

However, Windows allows the user to use much more memory than MS-DOS and it allows multi-tasking. This is when the user opens more than one program at a time and can move from one to another. Try opening a word processor and the clipboard in Windows at the same time. Adjust the sizes of the windows so that you can see both at the same time. Now mark a piece of text and copy it to the clipboard. You will see the text appear in the clipboard window although it is not the active window. This is because the OS can handle both tasks apparently at the same time. In fact the OS is swapping between the tasks so fast that the user is not aware of the swapping.

Another good example of multi-tasking is to run the clock program while using another program. You will see that the clock is keeping time although you are using another piece of software. Try playing a CD while writing a report!

The OS not only offers the user certain facilities, it also provides application software with I/O facilities. In this Section you will see how an OS is loaded and how it controls the PC.

Some of the text in this section is not required by the OCR Computing Specification, but may be interesting and useful for understanding how the system works.

When a PC is switched on, it contains only a very few instructions. The first step the computer does is to run the power-on-self-test (POST) routine that resides in permanent memory. The POST routine clears the registers in the CPU and loads the address of the first instruction in the boot program into the program counter. This boot program is stored in read-only memory (ROM) and contains the skeleton of the basic input/output system (BIOS). If this is given as the answer to a question asking for something that would normally be stored in ROM make sure that your answer specifies it is only the BIOS structure which is stored in ROM. The whole point about BIOS is that it is user defined and consequently must be alterable by the user and hence cannot be stored in ROM because ROM is read only. The user defined parts of the BIOS would normally be stored in a type of memory called CMOS RAM, this is not on the specification but students interested in this area may find it interesting to study this and other types of memory used in PC's. Control is now passed to the boot program which first checks itself and the POST program. The CPU then sends signals to check that all the hardware is working properly. This includes checking the buses, systems clock, RAM, disk drives and keyboard. If any of these devices, such as the hard disk, contain their own BIOS, this is incorporated with the system's BIOS.

The PC is now ready to load the OS. The boot program first checks drive A to see if a disk is present. If one is present, it looks for an OS on the disk. If no OS is found, an error message is produced. If there is no disk in drive A, the boot program looks for an OS on disk C. Once found, the boot program looks, in the case of Windows systems, for the files IO.SYS and MSDOS.SYS. Once the files are found, the boot program loads the boot record, about 512 bytes, which then loads IO.SYS. IO.SYS holds extensions to the ROM BIOS and contains a routine called SYSINIT. SYSINIT controls the rest of the boot procedure. SYSINIT now takes control and loads MSDOS.SYS which works with the BIOS to manage files and execute programs.

The OS searches the root directory for a boot file such as CONFIG.SYS which tells the OS how many files may be opened at the same time. It may also contain instructions to load various device drivers. The OS tells MSDOS.SYS to load a file called COMMAND.COM. This OS file is in three parts. The first part is a further extension to the I/O functions and it joins the BIOS to become part of the OS. The second part contains resident OS commands, such as DIR and COPY.

The files CONFIG.SYS and AUTOEXEC.BAT are created by the user so that the PC starts up in the same configuration each time it is switched on.

The OS supplies the user, and applications programs, with facilities to handle input and output, copy and move files, handle memory allocation and any other basic tasks.

In the case of Windows, the operating system loads into different parts of memory. The OS then guarantees the use of a block of memory to an application program and protects this memory from being accessed by another application program. If an application program needs to use a particular piece of hardware, Windows will load the appropriate device driver. Windows also uses virtual memory if an application has not been allocated sufficient main memory.

As mentioned above, Windows allows multi-tasking; that is, the running of several applications at the same time. To do this, Windows uses the memory management techniques described in Section 3.1.d. In order to multi-task, Windows gives each application a very short period of time, called a time-slice. When a time-slice is up, an interrupt occurs and Windows passes control to the next application. In order to do this, the OS has to save the contents of the CPU registers at the end of a time-slice and load the registers with the values needed by the next application. Control is then passed to the next application. This is continued so that all the applications have use of the processor in turn. If an application needs to use a hardware device, Windows checks to see if that device is available. If it is, the application is given the use of that device. If not, the request is placed in a queue. In the case of a slow peripheral such as a printer, Windows saves the output to the hard disk first and then does the printing in the background so that the user can continue to use the application. If further printing is needed before other printing is completed, then spooling is used as described in Section 3.1.e.

#### File Allocation Tables

Any OS has to be able to find files on a disk and to be able to store user's files. To do this, the OS uses the File Allocation Table (FAT). This table uses a linked list (see 3.5.b) to point to the blocks on the disk that contain files. In order to do this the OS has a utility routine that will format a disk. This simply means dividing the disk, radially, into sectors and into concentric circles called tracks. Two or more sectors on a single track make up a cluster. This is shown in Fig. 3.1.14.

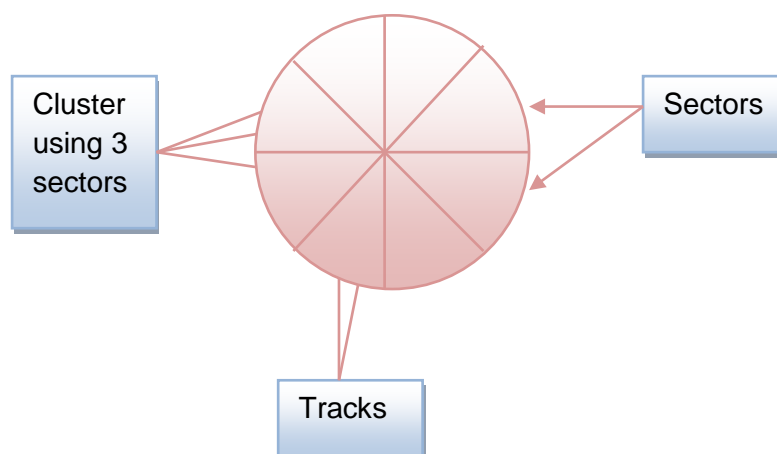


Fig 3.1.14

A typical FAT table is shown in Fig 3.1.15. The first column gives the cluster number and the second column is a pointer to the next cluster used to store a file. The last cluster used has a null pointer (usually FFFF<sub>H</sub>) to indicate the end of the linking. The directory entry for a file has a pointer to the first cluster in the FAT table. The diagram shows details of two files stored on a disk:

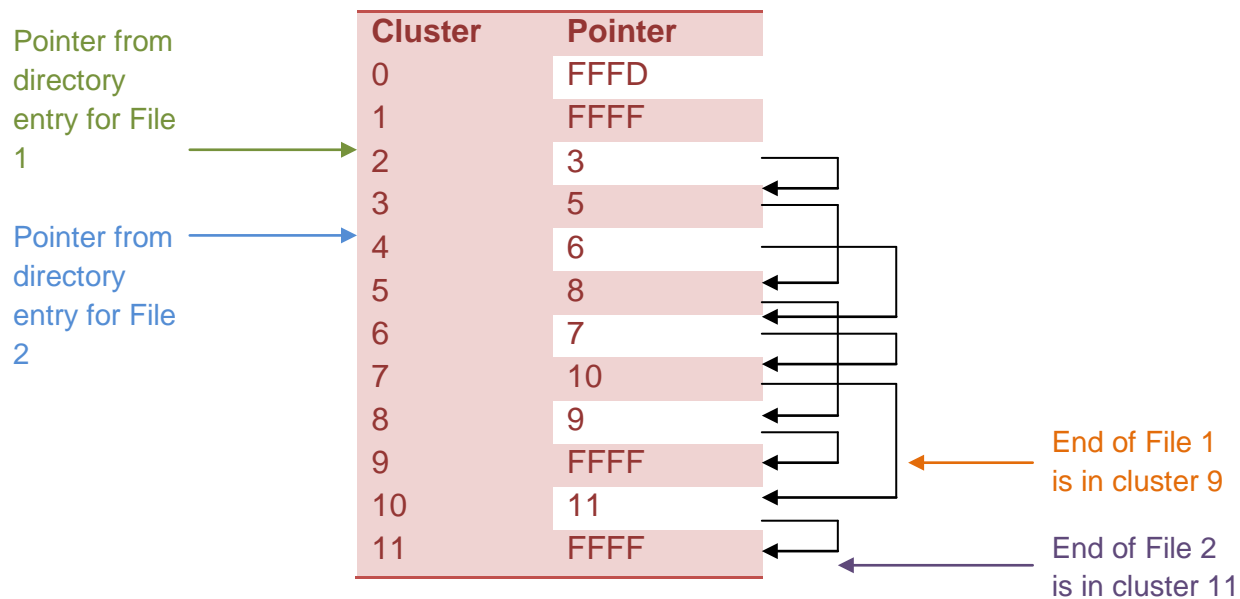


Fig. 3.1.15

In order to find a file, the OS looks in the directory for the filename and, if it finds it, the OS gets the cluster number for the start of the file. The OS can then follow the pointers in the FAT to find the rest of the file.

In this table any unused clusters have a zero entry. Thus, when a file is deleted, the clusters that were used to save the file can be set to zero. In order to store a new file, all the OS has to do is to find the first cluster with a zero entry and to enter the cluster number in the directory. Now the OS only has to linearly search for clusters with zero entries to set up the linked list.

It may appear that using linear searches will take a long time. However, the FAT table is normally loaded into RAM so that continual disk accesses can be avoided. This will speed up the search of the FAT.

Note that Windows 95/98 used virtual FAT (VFAT) which allows files to be saved 32 bits at a time (FAT uses 16 bits). It also allows file names of up to 255 characters. Windows 98 uses FAT 32 which allows hard drives greater than 2 Gbytes to be formatted.

### 3.1 Example Questions



1. Explain how an interrupt is handled by a processor which is working on another task.

(4)

- A.
- At some point in the cycle/at the end of an instruction
  - the processor will check to see if there are any outstanding interrupts
  - If there are, the priority of the present task is compared with the highest priority interrupt
  - If there is a higher priority interrupt, the current job is suspended and...
  - the contents of the special registers are stored so that it can be restarted later
  - interrupts are then serviced until all have been dealt with...
  - Control is returned to the original job.

(4)

*Notice: The handling of interrupts is rather more complex than described here and the manipulation of the special registers has to be explained in more detail, but the above answer covers all the points that arise in this section. Notice, also that the business of priorities is essential to this work. A good hint for a candidate is that they should always try to include mention of priorities when answering a question about handling interrupts.*

Further work may be interesting to some students. Areas for discussion might include:

- think about how an interrupt queue differs from an ordinary queue, and why
- why must there always be a job in the interrupt queue and what will it be like?
- what will happen if the interrupt queue is full? Under what conditions can this happen?
- what happens if two interrupts have the same priority?
- how can the system ensure that even the lowest priority jobs/interrupts will be dealt with eventually?

2. State **three** different types of interrupt that may occur and say in what order the three interrupts would be handled if they all arrived at the processor together, giving a reason for your answer.

(5)

- A.
- I/O interrupt like the printer running out of data to print and wanting the buffer refilling.
  - Timer interrupt where the processor is being forced to move onto some other task.
  - Program interrupt where the processor is being stopped from carrying out an illegal operation that has been specified in the program code.
  - Hardware interrupt the most serious of which is power failure.
  - The order is from the bottom up. The most serious is power failure because no other tasks can be carried out if there is no power, so the safe powering down of the system must be paramount.
  - Contrast that with the printer wanting more data before it can print any more out. Does it really matter if the printer has to wait a few more seconds?

*Notice: There are far more points to take into account about interrupts. Further work could involve a discussion about the following problem: An interrupt is simply a signal, it is not a piece of program code, so how does the processor know what to do when it tries to respond to an interrupt? Some students may find an investigation into vectored interrupts interesting.*

3. Explain the difference between

- (i) Input/output bound jobs and
- (ii) Processor bound jobs.

Explain how the computer should prioritise the two types of job if it is going to make maximum use of the system. (4)

A. -I/O jobs are those that require relatively little processing but do need to use the peripheral devices substantially

-Processor bound jobs require a large amount of processor time and very little use of the various peripheral devices.

-I/O jobs are given the highest priority...

-in order to keep the peripherals working as much as possible...

-and because they would be blocked and 'never see' the processor if the others had priority.

4. Describe **two** types of scheduling that are used by computer processing systems. (4)

A. The answers are to be found straight out of the notes on this chapter, page 10.

*Notice: It is an interesting exercise to discuss whether any other scheduling strategies can be considered. Try using the analogy of three home works needing to be done, how does a student decide what order to do them in. There are no right answers here, just ideas.*

5. Explain how interrupts are used in a round robin scheduling operating system. (3)

A. -Each job is given a set amount of processor time.

-At the end of the time available for a job, it is interrupted.

-The operating system inspects the queue of jobs still to be processed and

-if it is not empty allocates the next amount of processor time to the job first in the queue.

-The previous job goes to the end of the queue.

-Use of priorities to determine place in the queue.

-Need for priorities to change according to amount of recent processing time they have had.

6. a) Explain the difference between paging and segmenting when referring to memory management techniques. (2)
- b) Describe how virtual memory can allow a word processor and a spreadsheet to run simultaneously in the memory of a computer even though both pieces of software are too large to fit into the computer's memory. (3)

A. a) -They are both methods of dividing up the available memory space into more manageable pieces.

-Paging involves memory being divided into equal size areas.

-Segmenting involves areas of different size dependent upon the contents needing to be stored. (2)

*Notice: The first mark point does not really answer the question because it does not provide a difference, however it does give a context to the rest of the answer so is worth credit.*

b) -The software code is divided up into parts that have some level of equivalence.

-Those most commonly used routines are kept together

-These are loaded into memory and other routines are

-only loaded when the user calls upon them.

-Thus they give the impression of being permanently available. (3)

*Notice: A nice question. It is not only a standard bookwork question but it also relates specifically to an occurrence seen, if not understood, by computer users every day. A question with a little more detail about the software being used might expect a more detailed answer about the sort of things that would be in each of the pages, but it would also have more marks available.*

7. Describe how a PC operating system uses a file allocation table to find files when necessary.

A. -The disk surface must be divided up into small areas

-Files are stored in these small areas

-Each file will normally use more than one area

-The table of files has an entry pointing to the first area on the disk surface used by that file and

-a pointer to the next area.

-Each subsequent area has a pointer to the next area, as in a linked list with...

-a null value to signify the end of the file.

## 3.2 The Function and Purpose of Translators



### 3.2.a *Describe the need for and use of translators to convert source code to object code*

When electronic computers were first used, the programs had to be written in machine code. This code was comprised of simple instructions each of which was represented by a binary pattern in the computer. To produce these programs, programmers had to write the instructions in binary. This not only took a long time, it was also prone to errors. When the first major computing establishment was set up during the war in order to try to use computing power to break the German Enigma codes, the Mathematics departments of Oxford and Cambridge universities were used to provide the workers to program the computers. Half of the workforce was employed calculating the necessary algorithms while the other half were fully employed setting the switches and rewiring the processors on the computer (known as Colossus). The process of setting up the programs that had been written was so complex that only undergraduate mathematicians were capable of doing it. If this had not changed then we would not have been using computers the way we are today. The problems were caused by the fact that everything that the computer had to do was done in a language the computer could understand. In order that computers could be used anywhere other than research institutes it had to be made possible for human beings to use a method of communication which was closer to human communication than to computers. As computers cannot understand anything but binary, this implies the need for something to translate the human based commands into the binary of the computer. Hence the need for translators.

### 3.2.b and c *Understand the relationship between assembly language and machine code.*

#### *Describe the use of an assembler in producing machine code*

In section 3.2.a it was said that computers use their own form of language. This form of language is based on binary numbers and uses combinations of binary digits to stand for different things.

These codes are specific to a particular machine or type of machine and the set of instructions that can be created are therefore referred to as the machine code.

Imagine a computer that can handles the following operations:

|                                         |     |
|-----------------------------------------|-----|
| Add two numbers together represented by | 001 |
| Subtract one number from another        | 010 |
| Input a number from the outside         | 011 |
| Output a value                          | 100 |
| Store a number in a memory location     | 101 |
| Get a value from a memory location      | 110 |

This may be a pretty restricted set of possible operations but it will serve to illustrate the workings of a processor.

If we imagine that our processor works in 8 bit bytes then we have 5 bits left over which can be used as locations in memory, it may be a pretty small memory, but again it will serve as an illustration.

So if our computer is given the command (instruction) 00100001 it will split the instruction into the two parts 001 and 00001.

The 001 means that something has to be added.

The 00001 is the address in memory of the thing that has to be added.

If the number 4, or the length 2.3metres or the amount £2.31 or any other value is being stored in location 00001 in memory, it will be added to the accumulator (see section 1.4.b)

The instruction 00100001 is therefore understandable by humans and is necessary for the computer if it is to be able to follow it. However, imagine a program with thousands of instructions that look like this. It really would be almost impossible to follow. Human beings need a bit of help. They need a language other than binary if they are going to be able to understand a large program.

The first attempt at making programs more understandable was to change some of the combinations of binary numbers for letters. In our language:

|           |                                               |     |
|-----------|-----------------------------------------------|-----|
| 001 means | add and could be                              | ADD |
| 010 means | subtract and could be                         | SUB |
| 011 means | to put a value into the computer and could be | IN  |
| 100 means | to output a value and could be                | OUT |
| 101 means | to store a value in a memory location         | STO |
| 110 means | to get a value from a memory location         | GET |



Imagine a program in binary and the equivalent with these words instead:

|          |          |
|----------|----------|
| 01100000 | IN       |
| 10100100 | STO00100 |
| 01100000 | IN       |
| 10100101 | STO00101 |
| 11000100 | GET00100 |
| 00100101 | ADD00101 |
| 10000000 | OUT      |

Try to work out what this program is designed to do. In binary it is almost impossible, but in the second version it is just about possible because the letters are so much easier to understand. These groups of letters that stand for the binary codes for the operations are known as mnemonics. Most mnemonics are restricted to 3 letters and are chosen to clearly represent what the operation does. It is important to remember from module 1 that all the IN and OUT is carried out through the accumulator and the same is true for operations like GET and ADD. In fact the normal mnemonic used is 'LOAD' rather than GET but there are no hard and fast rules, so be prepared to see mnemonics that may vary slightly.

There are still problems because of the references to locations in memory being in binary. These are simply where values are being stored so we could give them names, or labels. The program above is simply adding two values together so instead of referring to these values by the places they are stored (00100 and 00101) why not just call them X and Y (or any other name that we will be able to remember). This makes the program:

```
IN
STO X
IN
STO Y
GET X
ADD Y
OUT
```

This ability to use mnemonics to stand for the operations and labels to stand for the locations makes the programming very much easier for human beings to understand. However, it is now impossible for a computer to run it because it is not in binary. As we saw in 3.2.a it is now necessary for the computer to have a translator which will turn the program into binary. This translator program is called an assembler and languages written in this simple way are said to be written in assembler language. The assembler is relatively simple because assembly language has a one to one relationship with machine code. Basically the assembler must translate the mnemonics into binary which is done by having a simple look up table with the mnemonics on one side and the binary equivalents on the other. The labels are a bit more complicated. The assembler does the translation in the same way, but the table is not fixed at the start, it has to be created by the assembler. As the program is input the assembler will recognise the first label as X and then will allocate a location to X. In this way it builds its own look up table for every program that it runs.

### 3.2.d *Describe the difference between interpretation and compilation*

In sections 3.2.b and c we saw how assembly languages were created in order to make program writing easier. Assembly languages allowed the use of mnemonics and names for the values held in locations in memory. Writing programs in assembly language, although easier than using machine code, was still tedious and took a long time.

After assembly languages came high-level languages, which used the type of language used by the person writing the program. Thus FORTRAN (FORmula TRANslation) was developed for science and engineering programs and it used formulae in the same way as would scientists and engineers. COBOL (Common Business Oriented Language) was developed for business applications. Assembly languages are called low level languages because they are close to the language used by the computer, whereas high level languages are closer to languages that are used by human beings. Programs written in these high level languages also needed to be translated into machine code, but as the languages became more complicated and further away from the machine code, so the translators had to become more complicated, this led to the birth of compilers.

A compiler takes a program written in a high-level language and translates it into an equivalent program in machine code. Once this is done, the machine code version can be loaded into the machine and run without any further help as it is complete in itself. The high-level language version of the program is usually called the source code and the resulting machine code program is called the object code. The relationship between them is shown in Fig. 3.2.1.

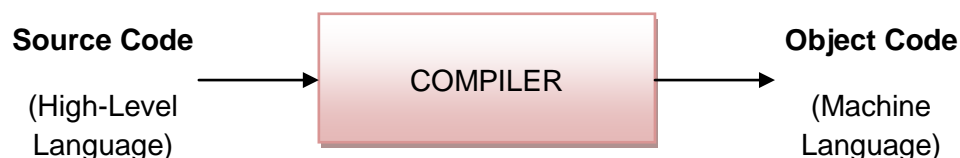


Fig. 3.2.1

The problem with using a compiler is that it uses a lot of computer resources. It has to be loaded in the computer's memory at the same time as the source code and there has to be sufficient memory to hold the object code. Further, there has to be sufficient memory for working storage while the translation is taking place. Another disadvantage is that when an error in a program occurs it is difficult to pin-point its source in the original program.

An alternative system is to use interpretation. In this system each instruction is taken in turn and translated to machine code. The instruction is then executed before the next instruction is translated. This system was developed because early personal computers lacked the power and memory needed for compilation. This method also has the advantage of producing error messages as soon as an error is encountered. This means that the instruction causing the problem can be easily identified. Against interpretation is the fact that execution of a program is slow compared to that of a compiled program. This is because the original program has to be translated every time it is executed. Also, instructions inside a loop will have to be translated each time the loop is entered.

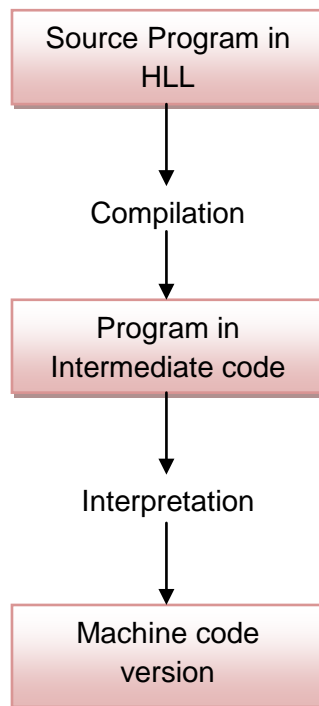
However, interpretation is very useful during program development as errors can be found and corrected as soon as they are encountered. In fact many languages, such as Visual Basic, use both an interpreter and a compiler. This enables the programmer to use the interpreter during program development, making the finding of errors much easier and, when the program is fully working, it can be translated by the compiler into machine code in order to speed up program execution. This machine code version can then be distributed to users who do not have access to the original code.

### 3.2.e *Describe the purpose of intermediate code in a virtual machine*

We have seen that a program written in any language other than the computer's own machine code needs to be translated before it can be run. We have also seen that there are two ways of translating programs written in a high level language: interpretation and compilation. There is a third way!

Different designs of computer have different versions of machine code, so 001 in one computer may stand for ADD while in another it stands for SUBTRACT. This would mean that every computer would need a different compiler for each high level language. An alternative is to use a compiler to do most of the translation and end up with a version of the program which is close to all the different machine codes. This version of the program is in 'intermediate code', so called because it is half way between the high level language version and the machine code version necessary to run the program. This version is not machine specific but it is now translated into the particular machine code needed by an interpreter specific to that machine.

This means that a program written in a high level language which is translated this way can be used on many different machines. This is called 'portability' and the program is said to be portable when it is in intermediate form. Unfortunately, full use of this method is not properly exploited, think about trying to run a game for whatever play centre you use on another system, it simply won't run. One of the few languages to use intermediate code is Java.



### **3.2.f, g, and h**     *The stages of translation*

The problem with translation, once past the simple assembler standard, is that it is not simply a matter of looking things up in a table like using a French to English dictionary. The languages now have command words in them that need many simple machine code instructions to carry them out. This implies that the translator must have an 'understanding' of what the commands will do. Also the meaning of commands can change dependent on whereabouts in the code they come, what comes before them and what is included in the instruction itself. This means that there is an element of a need to 'understand' what the code is doing. The word understand has been put in inverted commas because it does not really understand the program in the sense that we do, but there is certainly an understanding that goes beyond simple look up tables.

In order to translate a high level language program using a compiler, there are three stages: lexical analysis; syntax analysis and code generation. At each of these stages the compiler goes through the code with a specific aim in mind. There is a name for this 'going through the code', it is called parsing.

The next three sections explain what the compiler does in each of the three stages.

### 3.2.f *Describe what happens during lexical analysis*

The lexical analyser uses the source program as input and turns the high level language code into a stream of tokens for the next stage of the compilation.

Tokens are normally groups of 16-bits, and each group of characters in the code is replaced by a token. So, a command word like PRINT would be replaced by a token which will represent it.

Single characters, which have a meaning in their own right, are replaced by their ASCII values.

Variable names will need to have extra information stored about them. This is done by creating a symbol table. This table is used throughout compilation to build up information about names used in the program. So when a variable name is met for the first time, for instance 'X' it is placed in the table. During lexical analysis only the variable's name will be noted. It is during syntax analysis that details such as the variable's type and scope are added.

The lexical analyser may output some error messages and diagnostics. For example, it will report errors such as a keyword which is not entered properly. If PINT has been entered the error will probably be found here because there will be no token which can be used to replace it. Similarly, it will have rules for things like variable names which must be followed and it will spot errors when it tries to load them into the symbol table.

At various stages during compilation it will be necessary to look up details about, and add further details to, the names in the symbol table. This must be done efficiently so a linear search is not sufficiently fast. In fact, it is usual to use random access to the values in the table. This is done by creating a hash table and to calculate the position of a name by hashing it. (Section 2.2.e). When two names are hashed to the same address, a linked list can be used to avoid the symbol table filling up.

The lexical analyser also removes redundant characters such as white space (spaces, tabs, etc.) and comments. Often the lexical analysis takes longer than the other stages of compilation. This is because it has to handle the original source code, which can have many formats. For example, the following two pieces of code are equivalent although their format is considerably different.

|                         |                          |
|-------------------------|--------------------------|
| IF X = Y THEN 'square X | IF X = Y THEN Z := X * X |
| Z := X * X              | ELSE Z := Y * Y          |
| ELSE 'square Y          | PRINT Z                  |
| Z := Y * Y              |                          |
| ENDIF                   |                          |
| PRINT Z                 |                          |

When the lexical analyser has completed its task, the code will be in a standard format which means that the syntax analyser can always expect the format of its input to be the same.

### 3.2.g Describe what happens during syntax analysis

This Section should be read in conjunction with Section 3.7.d which discusses Backus-Naur Form (BNF) and syntax diagrams.

During this stage of compilation the code generated by the lexical analyser is checked to see that it is grammatically correct. All languages have rules of grammar and computer languages are no exception. The grammar of programming languages is defined by means of BNF notation or syntax diagrams. It is against these rules that the code has to be checked.

For example, taking a very elementary language, an assignment statement may be defined to be of the form:

```
<variable> <assignment_operator> <expression>
```

and an expression is:

```
<variable> <arithmetic_operator> <variable>
```

The parser must take the output from the lexical analyser and check that it is of this form.

If the statement is:

```
sum := sum + number
```

The parser will receive:

```
<variable> <assignment_operator> <variable> <arithmetic_operator> <variable>
```

which becomes:

```
<variable> <assignment_operator> <expression>
```

and then:

```
<assignment statement>
```

which is valid.

If the original statement is:

```
sum := sum + + number
```

This will be input as:

```
<variable> <assignment_operator> <variable> <arithmetic_operator> <arithmetic_operator> <variable>
```

and this does not represent a valid statement hence an error message will be returned.

It is at this stage that invalid names can be found if they were not spotted earlier. PINT may have been read as a variable under lexical analysis. This will mean that the statement containing PINT will not be a valid statement because it will not contain a key word. Note that in languages that require variables to be declared before being used, the lexical analyser may pick up this error because PINT will not have been declared as a variable and so will not be in the symbol table.

Most compilers will report errors found during syntax analysis as soon as they are found and will attempt to show where the error has occurred. However, they may not be very accurate in their conclusions nor may the error message be very clear.

During syntax analysis certain semantic checks are carried out. These include label checks, flow of control checks and declaration checks.

Some languages allow GOTO statements (not recommended by the authors) which allow control to be passed, unconditionally, to another statement which has a label. The GOTO statement specifies where control must be passed to by using the label in the program code to which the control must pass. The compiler must check that such a label exists.

Certain control constructs can only be placed in certain parts of a program. For example in C (and C++) the CONTINUE statement can only be placed inside a loop. The compiler must ensure that statements like this are used in the correct place.

Many languages insist on the programmer declaring variables and their types. It is at this stage that the compiler verifies that all variables have been properly declared and that they are used correctly. The data types and other details, like the parts of the code where it is valid (see section 3.8.b), are added to the entries in the symbol table.

### 3.2.h Explain the code generation phase and understand the need for optimisation

It is at this stage, when all the errors due to incorrect use of the language have been removed and the source code has been transformed into a form that the compiler can understand, that the program is translated into code suitable for use by the computer's processor.

During lexical and syntax analysis a table of variables has been built up which includes details of the variable name, its type and the block in which it is valid. The address of the variable is now calculated and stored in the symbol table. This is done as soon as the variable is encountered during code generation.

Before the final code can be generated, an intermediate code is produced (section 3.2.e). This intermediate code can then be interpreted or translated into machine code. In the latter case, the code can be saved and distributed as an executable program.

The compiler can now produce the machine code that is required by the computer by directly translating from the tokens to the binary code, using look up tables. A more detailed look at the process follows which is not part of the specification.

Two methods can be used to represent the high-level language in machine code. The one uses a tree structure and the other a three address code (TAC). TAC allows no more than three operands and instructions take the form:

$\text{Operand}_1 := \text{Operand}_2 \text{ Operator } \text{Operand}_3$

For example, using three address code, the assignment statement:

$A := (B + C) * (D - E) / F$

can be evaluated in the following order, where  $R_i$  represents an intermediate result:

$R_1 := B + C$

$R_2 := D - E$

$R_3 := R_1 * R_2$

$A := R_3 / F$

This can also be represented by the tree shown in Fig. 3.2.1.

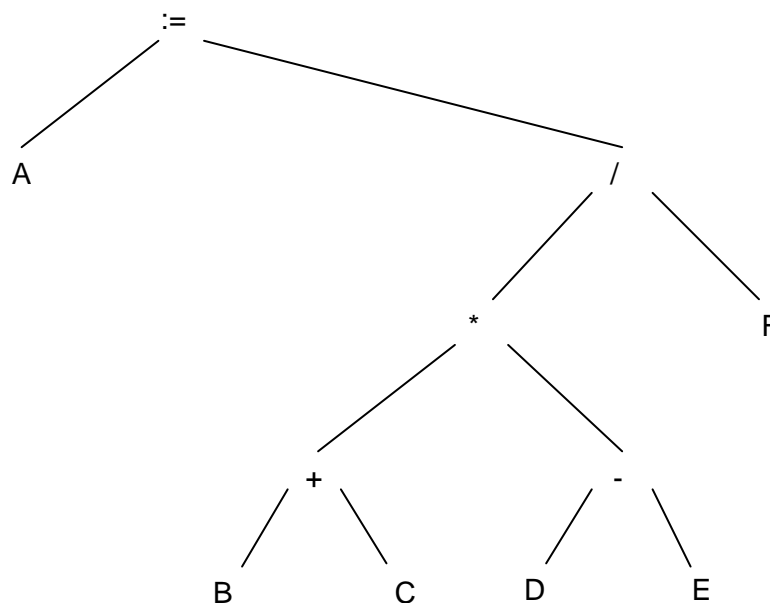


Fig. 3.2.1



Other statements can be represented in similar ways and the final stage of compilation can then take place.

The compiler has to consider, at this point, the type of code that is required. Code can be produced which executes as quickly as possible or the shortest code possible can be produced. These two ideals are not always compatible and the result is normally that a compromise between the two is used.

Consider the example code produced in section 3.2.b and c.

```
IN
 STO X
IN
 STO Y
 GET X
 ADD Y
OUT
```

It has been reproduced here in assembly language form rather than the binary which the compiler would be working with so that we can understand what is happening.

This code works perfectly well and it adds the two numbers together and produces the answer as output. However, the two values of X and Y have been stored when there seems to be no reason for doing so.

Consider the following set of instructions:

```
IN
 STO X
IN
 ADD X
OUT
```

This will produce the same result as the original but is two instructions shorter, requires less accessing of memory (always a time consuming action) and uses up less valuable memory space. The two versions carry out the same task but one is far better in computing terms than the other. This process of trying to improve on the form of the code (rather than what it produces) is called 'code optimisation'.

Another example of code optimisation is shown in Fig. 3.2.2 where the code on the left has been changed to that on the right so that  $r1 * b$  is only evaluated once:

$a := 5 + 3$	$a := 5 + 3$
$b := 5 * 3$	$b := 5 * 3$
$r1 := a + b$	$r1 := a + b$
$r2 := r1 * b$	$r2 := r1 * b$
$r3 := r2 / a$	$r3 := r2 / a$
$r4 := r1 - b$	$r4 := r2$
$r5 := r4 + 6$	$r5 := r4 + 6$
$c := r3 - r5$	$c := r3 - r5$

Fig. 3.2.2

However, care must be taken when doing this as it does not work in the case shown in Fig. 3.2.3 because the value of  $b$  changes between the two evaluations of  $r1 * b$ . Hence the code on the right is not equivalent to that on the left:

$a := 5 + 3$	$a := 5 + 3$
$b := 5 * 3$	$b := 5 * 3$
$r1 := a + b$	$r1 := a + b$
$r2 := r1 * b$	$r2 := r1 * b$
$r3 := r2 / a$	$r3 := r2 / a$
$b := b + 2$	$b := b + 2$
$r4 := r1 - b$	$r4 := r2$
$r5 := r4 + 6$	$r5 := r4 + 6$
$c := r3 - r5$	$c := r3 - r5$

Fig. 3.2.3

In the first example we have, in the revised code, the copy statement  $r4 := r2$ . This means that  $r4$  and  $r2$  represent the same value and hence one of them can be removed. This leads to the code in Fig.3.2.4:

```
a := 5 + 3
b := 5 * 3
r1 := a + b
r2 := r1 - b
r3 := r2 / a
b := b + 2
r5 := r2 + 6
c := r3 - r5
```

Fig. 3.2.4

Another way of optimising code is to use algebraic transformations. For example, the statement:

```
a := b * 1
```

can be replaced by the statement:

```
a := b
```

and:

```
a := b + 0
```

by:

```
a := b
```

There are many other ways of optimising code but they are beyond that expected at this level.

### **3.2.i**    *Describe the use of library routines*

Programs are usually built up in small, self contained, sections of code called sub programs or procedures or modules. Each of these is separately compiled so there is not only a high level language version of the code but also the compiled version. It seems sensible, then, that if a module for carrying out a particular task has already been written and compiled and shown to work, to save it so that it can be used again. These modules are stored in an area known as the library and the modules made available are library routines.

A problem immediately arises because variable names and memory addresses will be different from one use of the library routine to the next. These problems are solved by programs called loaders, whose job it is to load all the modules into the memory, and linkers whose job it is to link the different modules together. No other knowledge of these techniques will be expected here.

## 3.2 Example Questions



1. Explain why translation is important to the use of software on computer systems.

(3)

- A. -Computers can only use binary numbers/understand machine code  
-Machine code is too confusing for reliable programming by humans  
-Something is needed to bridge the gap so that both needs are satisfied

*Notice: This is an 'explain' question. The answer has been approached from two viewpoints and the final mark has been given for the conclusion drawn. Also note that apart from questions about source code and object code there is little else that can be asked from this bullet point.*

2. a Describe an assembly language and the advantages of using one for the programmer.

(4)

- b Explain why the relationship between machine code and assembly language can be described as one to one.

(1)

- A a -Use of mnemonics for operations

- Use of labels for variables/addresses  
-Human beings can understand meaningful strings better than binary  
-Gives greater control over the processor, particularly the use of memory

- b -One assembly language instruction is translated directly into one machine code instruction.

*Notice: The last mark point in (a) is one which is intended to be for higher level abilities, it is certainly not obvious. Despite being an 'Explain' question, part (b) is only worth one mark.*

3. Using the set of operations:

Divide one number by the new number	000
Add two numbers together represented by	001
Subtract one number from another	010
Input a number from the outside	011
Output a value	100
Store a number in a memory location	101
Get a value from a memory location	110

and an 8 bit byte for each instruction

a By using the test data 4,6,2 describe what this program does

01100000

10101000

01100000

10101001

01100000

10101100

11001001

00101000

00001100

10000000 (7)

b Write this program in an assembly language. (4)

c The program given will produce a single result, the values are not needed again.

(i) State what is meant by optimisation of code. (2)

(ii) Produce an optimised version of the above code. (3)

A a -The value 4 is stored in location 1000

-The value 6 is stored in location 1001

-The value 2 is stored in location 1100

-6 is fetched and is stored in the accumulator

-2 is added to the accumulator

-The value in the accumulator is divided by 2

-The answer 4 is output

-The code finds the mean of two numbers

(1 per -, max 7)

b

```
IN
STO X
IN
STO Y
IN
STO Z
GET Y
ADD X
DIV Z
OUT
```

Marks:

- Use of mnemonics and labels
- Correct use of mnemonics...
- in the correct positions
- Sensible labels

c(i) -Making code as short as possible/as fast to execute as possible

- Removing unnecessary lines of code

(ii)

```
IN
STO Z
IN
STO X
IN
ADD X
DIV Z
OUT
```

Note that the first input must be the 2, followed by the two numbers.

- Fewer than 10 commands
- works
- has a note of the change in order if necessary

*Notice: Difficult question. The way to attempt a question like this is to start by trying to understand what the algorithm is trying to do and you can only do this if you create the assembler version first. When talking about optimisation remember that the intention is to reduce the number of instructions.*

4. a Using the same set of operations consider what extra operations will be necessary to write code that will print out the first x multiples of 10. (4)

b Outline a solution to the problem. (6)

A a -Multiply in order to allow multiples to be produced

-Increment in order to produce a counter to determine how many times the multiply has been used

-Comparison in order to compare the counter with the set value x

-Repetition construct to repeat a set of code if the comparison is not true.

b -Input 10 and x

-Input 1 into a counter variable

-Multiply the 10 by the counter/or add 10 to the previous result, if this is done then the previous result must also be stored

-Output the result

-Compare counter with x

-If False then

-increment counter

-repeat from 'Multiply the 10...'

-End

*Notice; This is a very hard question and unlikely to appear in the main body of an exam paper but could be included under the heading of 'stretching' material. There are many ways of solving the problem, none is better than the others in terms of the way the question is set. The creation of the finished product might be a challenge! Note also the need for far more operations, 8 (our maximum) is simply not enough. The problem is that the more operations that are required, the smaller the number of memory locations that can be accessed. There must be a compromise between the two, but 8 bits for the instruction is obviously unrealistic in reality.*

5. Explain why the size of the memory available is particularly relevant to the process of compilation. (4)

A. The computer must be able to simultaneously hold in memory:

-The compiler software/without which the process cannot be carried out.

-The source code/the code that needs to be compiled

-The object code/because the compilation process produces the program in machine code form.

-Working space/processing area to be used by the compiler during the process.

(4)

*Notice: This question is trying to make the candidate think about the implications of the creation of a machine code version of the high-level language program. The significance of the size of the memory is not as significant as it used to be because the memory in a micro is now large enough for the*



*problem not to arise in the main, but in the past the compilation of programs could cause trouble on a micro leading to the standard translation technique of interpretation.*

6. a) Explain the difference between the two translation techniques of interpretation and compilation. (2)

- b) Give **one** advantage of the use of **each** of the two translation techniques. (2)

A.a) -Interpretation involves the translation of an instruction and the execution of that instruction before the translation of the next.

-The compilation of a program involves creating the machine code version of the program before allowing any of it to be run.

- b) -Interpretation provides the programmer with better error diagnostics because the source code is always present and hence can be used to provide a reference whenever the error occurs.

-When a program is compiled no further translation is necessary no matter how many times the program is run, consequently there is nothing to slow down the execution of the program.

*Notice: The question is probably not in its best form as there are many responses that could justifiably be given to the difference between the two processes. A perfectly acceptable response here would be that interpretation does not create an object code while compilation does.*

7. State the **three** stages of compilation and describe, briefly, the purpose of each. (6)

A. -Lexical analysis

-puts each statement into the form best suited to the syntax analyser.

-Syntax analysis

-Language statements are checked against the rules of the language.

-Code generation

-The machine code (object code) is produced.

*Notice: The number of marks for the question plays a big part in this answer. There are only six marks, three of which must be for stating the three stages. This means that there is only one mark each for saying what the purpose of each is. Do not launch into a long essay, you don't have time in the constraints of the examination room, the examiner is simply looking for an outline description of what the stage does. Be careful about writing down all you know about each stage. There is a danger that the first thing you write down may be wrong. There is only one mark available and, if the answer is very long, the mark will be lost immediately. Also, don't think that the marks can be carried across from another part. You may not know anything about code generation, but you do know a lot about lexical analysis – sorry, the marks cannot be transferred over in a question like this.*

8. Explain, in detail, the stage of compilation known as lexical analysis. (6)

A. -Source program is used as the input

-Tokens are created from the individual characters and from...

-the special reserved words in the program.

-A token is a string of binary digits.

-Variable names are loaded into a look up table known as the symbol table

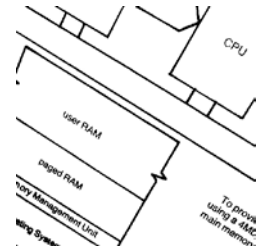
-Redundant characters (e.g. spaces) are removed

-Comments are removed

-Error diagnostics are issued. (6)

*Notice: Compare this question with the last one. This one is asking for the details, so it becomes important to say as much as possible. The examiner may be a little more lenient about something in a student's list that is wrong, but only a little! After all, the main thing about the stages of compilation is to know when each of them is appropriate, so don't make too many errors.*

### 3.3 Computer Architectures



#### 3.3.a Describe classic Von Neumann architecture, identifying the need for, and the uses of, special registers in the functioning of a processor

John Von Neumann introduced the idea of the stored program. Previously data and programs were stored in separate memories. Von Neumann realised that data and programs are indistinguishable and can, therefore, use the same memory.

The Von Neumann architecture uses a single processor which follows a linear sequence of *fetch-decode-execute*. In order to do this, the processor has to use some special registers. A register is simply an area that can store data. In a way the individual locations in memory can be thought of as registers, although they have no special purpose. Also the registers outside the immediate access store allow faster access to data stored. We are interested in a group of registers which the computer uses to control the fetch decode execute cycle.

Register	Meaning
PC	Program Counter
CIR	Current Instruction Register
MAR	Memory Address Register
MDR	Memory Data Register
Accumulator	Holds results

#### The 'Fetch' part of the cycle

The Program Counter stores the address of the next instruction that needs to be carried out. As instructions are held sequentially in memory, it is normal for the value in the PC to be incremented every time an instruction is accessed so that it always points to the *next* instruction.

When the next instruction is needed its address is copied from the PC and placed in the Memory Address Register.

This location/address is found in memory and the contents are placed in the Memory Data Register.

The contents of the MDR are then copied to the Current Instruction Register.

The current instruction register holds the instruction that is to be executed.

#### The 'Decode' part of the cycle

The instruction which is now in the CIR is split into its individual parts.

We know of two parts of the instruction already from the work in section 3.2, the operation code and the address in memory of the data to be used with that operation.

The address part is placed in the MAR and the data is fetched and placed in the MDR.

The operation is decoded by referring to a look up table which will give the control unit the instructions about what needs to be done.

#### The 'Execute' phase of the cycle

The contents of the MDR are sent to the required place together with instructions about what to do with it. Note that if the operation is to store something away in memory the data will go the other way and will flow from the MDR to the address in the MAR.

Note that if the operation is a jump instruction the contents of the MAR will be used to change the PC because the next instruction will be dealt with out of order.

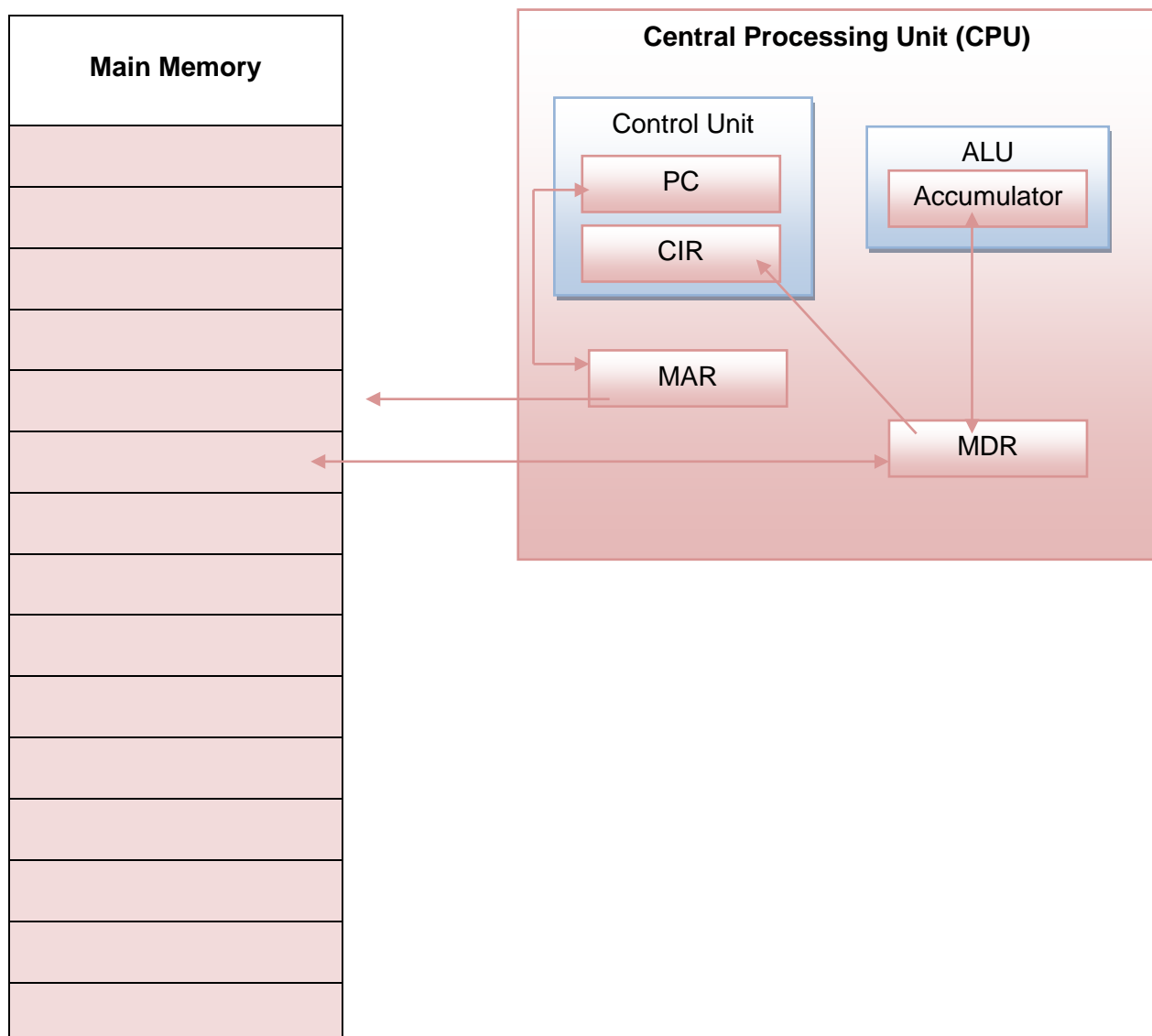
(The PC is sometimes called the Sequence Control Register and the MDR is sometimes called the Memory Buffer Register because it acts like a buffer, storing the data until whatever needs it is ready for it. These names should not turn up in papers, but it is worth noting them because you may see them in text books.)

The central processor contains the arithmetic/logic unit (also known as the ALU and as the arithmetic unit) and the control unit. The arithmetic/logic unit is where data is processed. This involves arithmetic and logical operations. Arithmetic operations are those that add and subtract numbers, and so on. Logical operations involve comparing binary patterns and making decisions.

The control unit fetches instructions from memory, decodes them and synchronises the operations before sending signals to other parts of the computer.

The accumulator is in the arithmetic unit, the program counter and the instruction registers are in the control unit and the memory data register and memory address register are in the processor.

A typical layout is shown in Fig. 4.3.1.1 which also shows the data paths:



### 3.3.b *Describe, in simple terms, the fetch/decode/execute cycle , and the effects of stages of the cycle on specific registers*

The following is an algorithm that shows the steps in the cycle. At the end the cycle is reset and the algorithm repeated.

1. Load the address that is in the program counter (PC) into the memory address register (MAR).
2. Increment the PC by 1.
3. Load the instruction that is in the memory address given by the MAR into the memory data register (MDR).
4. Load the instruction that is now in the MDR into the current instruction register (CIR).
5. Decode the instruction that is in the CIR.
6. If the instruction is a jump instruction then
  - a. Load the address part of the instruction into the PC
  - b. Reset by going to step 1.
7. Execute the instruction.
8. Reset by going to step 1.

Steps 1 to 4 are the *fetch* part of the cycle. Steps 5, 6a and 7 are the execute part of the cycle and steps 6b and 8 are the reset part.

Step 1 simply places the address of the next instruction into the memory address register so that the control unit can fetch the instruction from the right part of the memory. The program counter is then incremented by 1 so that it contains the address of the next instruction, assuming that the instructions are in consecutive locations.

The memory data register is used whenever anything is to go from the central processing unit to main memory, or vice versa. Thus the next instruction is copied from memory into the MDR and is then copied into the current instruction register.

Now that the instruction has been fetched the control unit can decode it and decide what has to be done. This is the execute part of the cycle. If it is an arithmetic instruction, this can be executed and the cycle restarted as the PC contains the address of the next instruction in order. However, if the instruction involves jumping to an instruction that is not the next one in order, the PC has to be loaded with the address of the instruction that is to be executed next. This address is in the address part of the current instruction, hence the address part is loaded into the PC before the cycle is reset and starts all over again.

Two interesting questions:

1. Why might it be necessary to add 2 to the program counter rather than 1?
2. How does the system handle a conditional jump?

### 3.3.c Discuss co-processor, parallel processor and array processor systems, their uses, advantages and disadvantages

Co-processors: (Sometimes called floating point units) Floating point representation (section 3.4) is a data representation which requires long strings of bits. A maths co-processor is a device which has a number of registers that are long enough to handle the length of floating point numbers in one go, thereby speeding up processing.

Parallel processors: Using the standard fetch/decode/execute cycle it is apparent that an instruction can be in one of three phases. It could be being fetched (from memory), decoded (by the control unit) or executed (by the control unit). An alternative would be to split the processor up into three parts, each of which handles one of the three stages. This would result in the situation shown in Fig. 3.3.1, which shows how this process, known as pipelining, works:

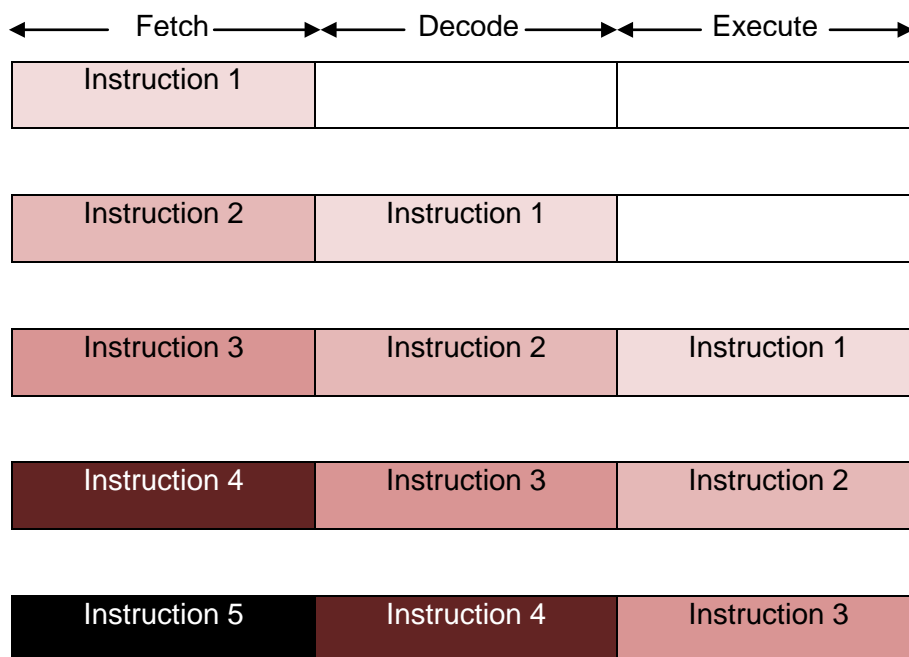


Fig. 3.3.1

This helps with the speed of throughput unless the next instruction in the pipe is not the next one that is needed. Suppose Instruction 2 is a jump to Instruction 10. Then Instructions 3, 4 and 5 need to be removed from the pipe and Instruction 10 needs to be loaded into the fetch part of the pipe. Thus, the pipe will have to be cleared and the cycle restarted in this case. The result is shown in Fig. 3.3.2

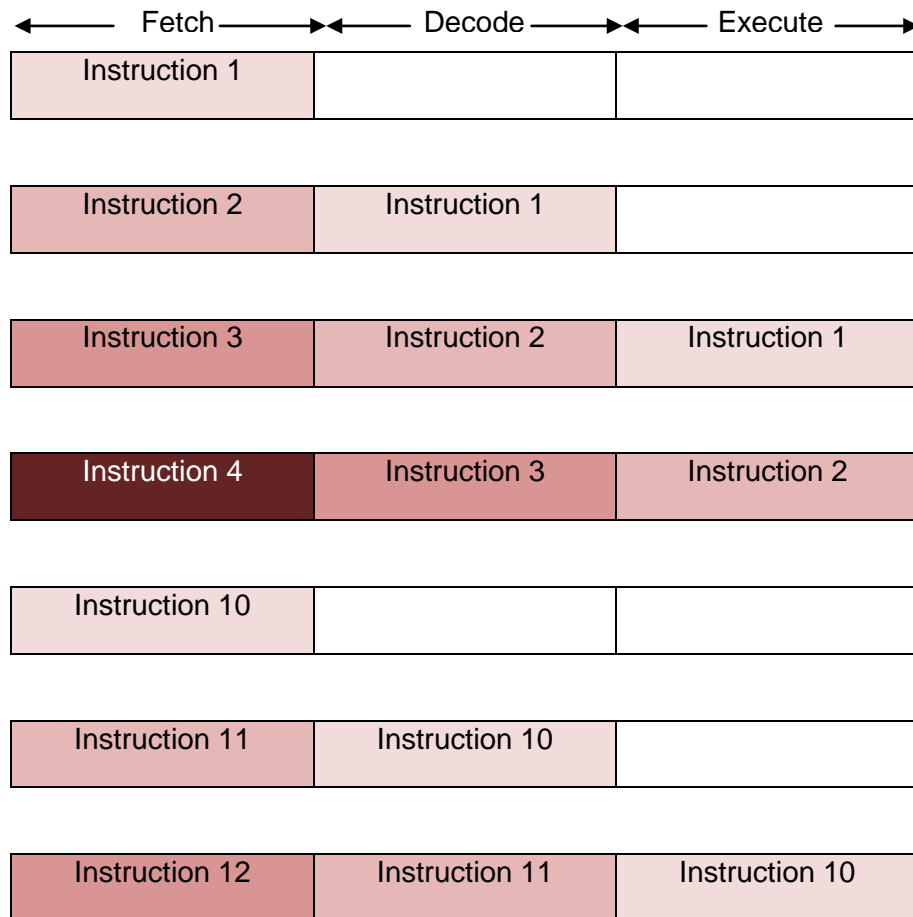


Fig. 3.3.2

**Array processors:** This involves more than one arithmetic-logic unit but still only one processor. This is particularly useful when processing data held in a one-dimensional array when the same operation is to be applied to every element of the array. An example is where all the values represent the costs of different items in stock and it is required to find the selling prices which are calculated by adding the same percentage to each value. An array processor is able to do the same calculation to each of the input values simultaneously.

**Parallel Processors:** An extension of array processors is to use parallel processors. This system uses many independent processors working in parallel on the same program. One of the difficulties with this is that the programs running on these systems need to have been written specially for them. If the programs have been written for standard architectures, then some instructions cannot be completed until others have been completed. Thus, checks have to be made to ensure that all prerequisites have been completed. However, these systems are in use particularly when systems are receiving many inputs from sensors and the data need to be processed in parallel. A simple example that shows how the use of parallel processors can speed up a solution is the summing of a series of numbers. Consider finding the sum of  $n$  numbers such as:

$$2 + 4 + 23 + 21 + \dots + 75 + 54 + 3$$

Using a single processor would involve  $(n - 1)$  additions one after the other. Using  $n/2$  processors we could simultaneously add  $n/2$  pairs of numbers in the same time it would take a single processor to add one pair of numbers. This would leave only  $n/2$  numbers to be added and this could be done using  $n/4$  processors. Continuing in this way the time to add the series would be considerably reduced.

### **3.3.d** *Describe and distinguish between Reduced Instruction Set Computer (RISC) and Complex Instruction Set Computer(CISC) architectures*

In sections 3.2.b and 3.2.c, the idea of an assembly language was introduced. The example used 6 (ultimately 7) operations to perform simple tasks like adding two numbers together. These 6 operations require 3 bits to distinguish between them. If we want more than 8 operations then we have to use more than 3 bits. Basically, the more operations that the processor can recognise the more complex it is and the more bits are needed to store all the operation codes.

Some processors are designed to have operations, and hence operation codes, for all the things they have to do. These are called complex instruction set computers (CISC).

Some processors are designed to reduce the number of operations recognised, hence saving space on the representation of the operation codes. These are called restricted instruction set computers (RISC). Just because they have fewer operations in their assembly language doesn't mean that they can do fewer things, simply that any 'missing' instructions have to be made up from the ones that are there by clever programming.



### 3.3 Example Questions



The questions in this section are meant to mirror the type and form of questions that a candidate would expect to see in an exam paper. As before, the individual questions are each followed up with comments from an examiner.

1. The Program Counter (Sequence Control Register) is a special register in the processor of a computer.
  - a) Describe the function of the *program counter*. (2)
  - b) Describe **two** ways in which the program counter can change during the normal execution of a program, explaining, in each case, how this change is initiated. (4)
  - c) Describe the initial state of the program counter before the running of the program. (2)

A. a) -The program counter stores the address...

-of the next instruction to be carried out in the sequence of the program.

b) -P.C. is incremented...

-as part of the fetch execute cycle.

-P.C. is altered to the value being held in the address part of the instruction...

-When the instruction is one that alters the normal sequence of instructions in the program.

-This second type of command involves the P.C. being reset twice in the same cycle.

c) -The P.C. will contain the address of the first instruction in the sequence to be run...

-this must have been placed in the register by some external agent, the program loader.

*Notice: Part (a) is often poorly understood by students. The majority believing that the program counter is used to keep track of the number of programs running, or the order in which programs have been called. There is obviously confusion with the idea of a stack storing return addresses of modules when they have been called.*

*Part (b) illustrates a characteristic of true examination questions. Most genuine questions will have more mark points available than there are marks for the question. This is not true of these sample questions. It should also be remembered that these sample questions have not been through the rigorous testing process that a genuine paper would have undergone, so any problems with the content should not be repeated in the examination. Candidates find difficulty in making the distinction between different types of instruction, it may be of value to spend some time talking about arithmetic/logic/jump/command type instructions as they all affect the cycle in different ways.*

2. Explain what is meant by the term Von Neumann Architecture. (2)

- A. -A way of looking at the relationships between the various pieces of hardware in a computer processor.
- A single memory used to store program instructions and the data for use with those instructions.
- A single processor is used which follows a linear sequence of instructions.

*Notice: Many students will be content with the correct answer that VN architecture is the ability to store the instructions and data in the same memory. However, a look at the mark allocation shows that something else is required or only one mark would have been available. Always look at the mark allocation and think of the examiner, is there enough in the answer given to be able to award the full number of marks?*

3. Describe the fetch/decode part of the fetch/decode/execute cycle, explaining the purpose of any special registers that you have mentioned.

(7)

- A. -Contents of PC loaded into MAR
- PC is incremented
- Contents of address stored in MAR loaded into MDR
- Contents of MDR loaded into CIR
- Instruction in CIR is decoded.
- PC (program counter) stores the address of the next instruction to be executed.
- MAR (memory address register) holds the address in memory that is currently being used
- MDR (memory data register) holds the data (or instruction) that is being stored in the address accessed by the MAR.
- CIR (current instruction register) holds the instruction which is currently being executed.

*Notice: The whole cycle may be asked for in some questions but it is more likely that it would be split up in some way in order to make the question shorter and more accessible. This is a difficult question because there is no splitting up of the points asked for. The student must rely on their own interpretation of the requirements of the question. There is a hint in the question because it asks for two parts of the cycle specifically, but students should be aware that that becomes a part of the question, in other words the answer must not contain any further information because it has been specifically ruled out in the question. A candidate who describes the execution of particular types of instruction has demonstrated that they cannot differentiate between the parts of the cycle and would probably be penalised.*

4. a) Describe how pipelining normally speeds up the processing done by a computer. (2)
- b) State one type of instruction that would cause the pipeline system to be reset, explaining why such a reset is necessary. (3)
- A. a) -All instructions have three phases...
- which are treated separately, by different parts of the processor...
  - so that more than one instruction can be being dealt with simultaneously.
- b) -Jump instruction
- The instructions in the pipeline are no longer the ones to be dealt with next...
  - so the pipeline has to be reset. (3)
5. Explain what is meant by a Reduced Instruction Set Computer (RISC) architecture and how it differs from a Complex Instruction Set Computer (CISC). (4)
- A
- The number of operations recognised by the computer is limited...
  - in order to reduce the number of bits needed to represent an instruction code
  - Complex instructions can be created by using the available operations in combination
  - This makes processing slower than in a CISC...
  - which has more standard operations.

## 3.4 Data Representation



### 3.4.a Demonstrate an understanding of floating point representation of a real floating point binary number

In section 1.3.b we learned how to represent both positive and negative integers in two's complement form. It is important that you understand this form of representing integers before you learn how to represent fractional numbers.

In decimal notation the number 23.456 can be written as  $0.23456 \times 10^2$ . This means that if we always stick to the same arrangement we need only store, in decimal notation, the numbers 0.23456 and 2 to be able to uniquely identify numbers. The decimal part of the number 0.23456 is called the *mantissa* and the number 2 is called the *exponent*. Why do we sometimes want to express numbers like this? If we have very large numbers which cannot be expressed in the normal way or if there are too many zeroes between the digits that describe the number and the point then it becomes necessary to store the number in a different form.

We can apply the same rules to binary numbers, which is useful because we only use 0 and 1 in binary so the length of the representation is going to be long anyway and it will become necessary to have an alternative representation sooner with binary numbers than with the decimal numbers we are used to.

For example, consider the binary number 10111. This could be represented by  $0.10111 \times 2^5$  or  $0.10111 \times 2^{101}$ . Here 0.10111 is the mantissa and 101 is the exponent. This could be represented as 010111101 provided we know where the gap between the mantissa and the exponent is! (Strange choice of number of bits: 9, we will have to do something about that later)

When we deal with decimal values (numbers less than 1) in decimal, the method is the same, move the point until you get a number between 0 and 1. The only difference is that the point is being moved the other way so we have to make the exponent negative to show this. So, in decimal, 0.0000246 can be written  $0.246 \times 10^{-4}$ . Now the mantissa is 0.246 and the exponent is  $-4$ .

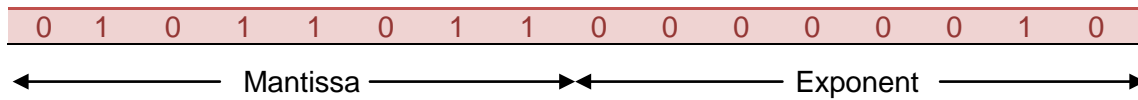
Thus, in binary, if we have 0.00010101 it can be written as  $0.10101 \times 2^{-11}$  and 0.10101 is the mantissa and  $-11$  is the exponent.

It is now clear that we need to be able to store two numbers, the mantissa and the exponent. This form of representation is called *floating point form*. Numbers that involve a fractional part, like  $2.467_{10}$  and  $101.0101_2$  are called *real numbers*.

### 3.4.b Normalise a real binary number

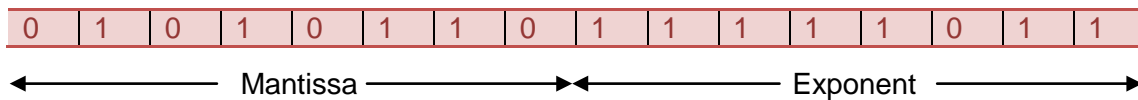
In the above examples, the point in the mantissa was always placed immediately before the first non-zero digit. This is always done like this with positive numbers because it allows us to use the maximum number of digits.

Suppose we use 8 bits to hold the mantissa and 8 bits to hold the exponent. The binary number 10.11011 becomes  $0.1011011 \times 2^{10}$  and can be held as:



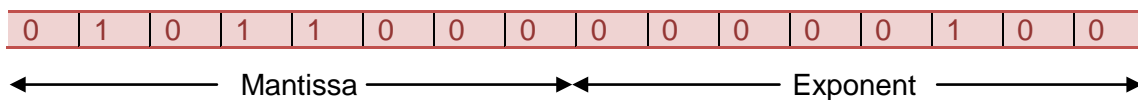
Notice that the first digit of the mantissa is zero and the second is one. **The mantissa is said to be normalised if the first two digits are different.** Thus, for a positive number, the first digit is always zero and the second is always one. The exponent is always an integer and is held in two's complement form.

Now consider the binary number 0.00000101011 which is  $0.101011 \times 2^{-101}$ . Thus the mantissa is 0.101011 and the exponent is  $-101$ . Again, using 8 bits for the mantissa and 8 bits for the exponent, we have:

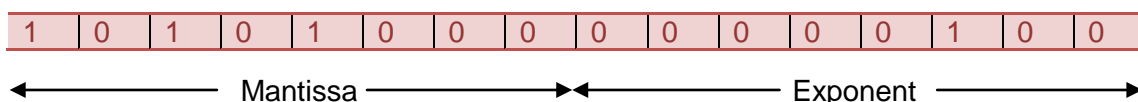


because the two's complement of  $-101$ , using 8 bits, is 11111011. (See section 1.3.b)

The reason for normalising the mantissa is in order to hold numbers to as high an accuracy as possible. Care needs to be taken when normalising negative numbers. The easiest way to normalise negative numbers is to first normalise the positive version of the number. Consider the binary number  $-1011$ . The positive version is  $1011 = 0.1011 \times 2^{100}$  and can be represented by:



Now find the two's complement of the mantissa and the result is:



Notice that the first two digits are different.

As another example, change the decimal fraction  $-11/32$  into a normalised floating point binary number:

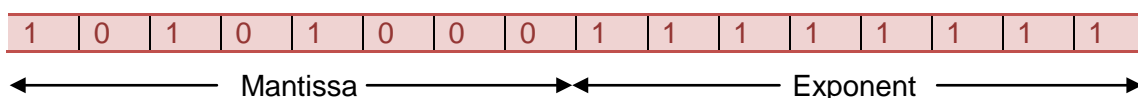
$$11/32 = 1/4 + 1/16 + 1/32 = 0.01 + 0.0001 + 0.00001 = 0.01011000$$

$$= 0.1011000 \times 2^{-1}$$

$$\text{Therefore } -11/32 = -0.1011000 \times 2^{-1}$$

Using two's complement  $-0.1011000$  is 1.0101000 and  $-1$  is 11111111

and we have:



The fact that the first two digits are always different can be used to check for invalid answers when doing calculations.

### 3.4.c Discuss the trade off between accuracy and range when representing numbers

There is always a finite number of bits that can be used to represent numbers in a computer. This means that if we use more bits for the mantissa we will have to use fewer bits for the exponent.

Let us start off by using 8 bits for the mantissa and 8 bits for the exponent. The largest positive value we can have for the mantissa is 0.1111111 and the largest positive number we can have for the exponent is 01111111. This means that we have:

$$0.1111111 \times 2^{1111111} = 0.1111111 \times 2^{127}$$

This is almost  $1 \times 2^{127}$ .

The smallest positive mantissa is 0.1000000 and the smallest exponent is 10000000. This represents:

$$0.1000000 \times 2^{10000000} = 0.1000000 \times 2^{-128}$$

which is very close to zero; in fact it is  $2^{-129}$ .

The largest negative number (i.e. the negative number closest to zero) is:

$$1.0111111 \times 2^{10000000} = -0.1000001 \times 2^{-128}$$

Note that we cannot use 1.1111111 for the mantissa because it is not normalised. The first two digits must be different.

The smallest negative number (i.e. the negative number furthest from zero) is:

$$1.0000000 \times 2^{01111111} = -1.0000000 \times 2^{127} = -2^{127}$$

Have you noticed that zero cannot be represented in normalised form? This is because 0.0000000 is not normalised because the first two digits are the same. Usually, the computer uses the smallest positive number to represent zero.

Thus, we have, for a positive number  $n$ :

$$2^{-129} \leq n < 2^{127}$$

and for a negative number  $n$ :

$$-2^{127} \leq n < -2^{-129}$$

Now suppose we use 12 of the bits for the mantissa and the other four for the exponent. This will increase the number of bits in the mantissa that can be used to represent the number. That is, we shall have more binary places in the mantissa and hence greater accuracy because we will be able to keep more digits in the fraction part before rounding off. However, the range of values in the exponent is much smaller, in this example it is from  $-8$  to  $+7$  and this produces a very small range of numbers. We have increased the accuracy but at the expense of the range of numbers that can be represented.

Similarly, reducing the size of the mantissa reduces the accuracy, but we have a much greater range of values as the exponent can now take larger values.

### 3.4 Example Questions



1. Describe a floating point representation for real numbers using two, 8 bit, bytes.

(4)

- A. -Available bits divided into fraction part (mantissa) and...

-exponent part (power of 2).

-Both are stored as Two's complement values...

-the mantissa is stored as a normalised fraction...

-the exponent as an integer.

-Example. An example like the one shown clearly explains many of the points made above and would be awarded all the earned marks simply for the diagram.

-1	1/2	1/4	1/8	1/16	1/32	1/64	1/128	128	64	32	16	8	4	2	1

*Notice: There are plenty of marks here. This is not an easy question. The question used the word describe and there are 4 marks for the question so a little more is expected on top of the idea of two parts to the representation. The example could be a worked example of a real number, but it would be acceptable, in this question, to provide a description of the column headings because the question asks that the representation be described rather than the method. The concept of normalisation is a complicated one and not really relevant to the question beyond the mention.*

2. a) Explain how a binary mantissa can be normalised. (2)

- b) State the benefit obtained by storing real numbers using normalised form. (1)

- A. a) -Digits are moved so that the first two digits are different and there is only one digit before the point.

-A positive number starts 01... and a negative number starts 10...

- b) -The accuracy of the representation is maximised.

3. a) A floating point number is represented in a certain computer system in a single 8 bit byte. 5 bits are used for the mantissa and 3 bits for the exponent. Both are stored in two's complement form and the mantissa is normalised.

(i) State the smallest positive value that can be stored.

(ii) State the most negative value that can be stored.

Give each answer as an 8 bit binary value and as a decimal equivalent. (4)

- b) Explain the relationship between accuracy and range when storing floating point representations of real numbers. (4)

A. a) (i) 01000/100 or  $\frac{1}{2} * 2^{-4} = 1/32$

(ii) 10000/011 or  $-1 * 2^3 = -8$

b)-The larger the number of bits used for the mantissa, the better the accuracy

-The larger the number of bits used for the exponent, the greater the range

-There is always a finite number of bits.

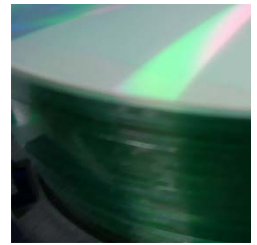
-Therefore the more bits used for one part of the representation, the fewer bits can be used for the other.

*Notice: This is a difficult question. The answers to part (a) have been written with / between the two parts of the representation for clarity, this would not be expected in a student answer. The idea of using a single byte would be ridiculous in reality, but it keeps the arithmetic possible in an examination.*

*In part (b) the idea is not difficult, but the explanation is very difficult to put into words. Students who are happy with the arithmetic may find it simpler to use the examples in part (a) and move the partition between the two parts to show the effect.*



## 3.5 Data Structures and Data Manipulation



### 3.5.a Explain how static data structures may be used to implement dynamic data structures

Static data structures are those structures that do not change in size while the program is running. A typical static data structure is an array because once you declare its size, it cannot be changed. (In fact, there are some languages that do allow the size of arrays to be changed, in which case they become dynamic data structures!)

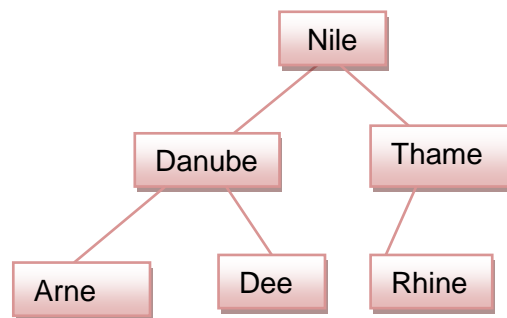
Dynamic data structures can increase and decrease in size while a program is running. A typical example is a linked list.

The following table gives advantages and disadvantages of the two types of data structure:

	Advantages	Disadvantages
<b>Static structures</b>	Compiler can allocate space during compilation.  Easy to program.  Easy to check for overflow.  An array allows random access.	Programmer has to estimate the maximum amount of space that is going to be needed.   Can waste a lot of space.
<b>Dynamic structures</b>	Only uses the space needed at any time.  Makes efficient use of memory.  Storage no longer required can be returned to the system for other uses.	Difficult to program.  Can be slow to implement searches.  A linked list only allows serial access.

A static structure like an array can hold a dynamic structure. The only real difficulty is ensuring that the array is large enough. Consider the tree structures in section 3.5.b (a little out of order, but never mind). A tree needs spaces to store the data found at the nodes, a left pointer and a right pointer for each node. This tells us that one of the dimensions of the array needs three values. The tree also needs one entry for each node that it is ever likely to have (remember that the array is a static data structure and hence we can't go changing our minds if the tree suddenly gets big).

For the sake of this illustration let us imagine that this is the biggest tree we will ever have:



This tree could be represented by the following array:

Position	Right pointer	Name	Left pointer
1	5	Danube	2
2	X	Arne	X
3	X	Thames	6
4	3	Nile	1
5	X	Dee	X
6	X	Rhine	X

If there was a possibility of 3 more rivers being discovered then the array could be created with 9 rows instead of 6.

Notice the use of a null value to signify that there is no pointer from some of the nodes.

Also, it is going to be important to know where to start the tree, so there is a need for a head of tree pointer (or a root node pointer) to point to 4.

### 3.5.b Describe algorithms for the insertion, retrieval and deletion of data items stored in stack queue and tree structures

#### Stacks – Insertion

Fig. 3.5.1 shows a stack and its head pointer. Remember, a stack is a last-in-first-out (LIFO) data structure. If we are to insert an item into a stack we must first check that the stack is not full. Having done this we shall increment the pointer and then insert the new data item into the cell pointed to by the stack pointer. This method assumes that the cells are numbered from 1 upwards and that, when the stack is empty, the pointer is zero.



Fig. 3.5.1

The algorithm for insertion is:

1. Check to see if stack is full.
2. If the stack is full report an error and stop.
3. Increment the stack pointer.
4. Insert new data item into cell pointed to by the stack pointer and stop.

#### Stacks – Deletion/Reading

When an item is deleted from a stack, the item's value is copied and the stack pointer is moved down one cell. The data itself is not deleted. This time, we must check that the stack is not empty before trying to delete an item.

The algorithm for deletion is:

1. Check to see if the stack is empty.
2. If the stack is empty report an error and stop.
3. Copy data item in cell pointed to by the stack pointer.
4. Decrement the stack pointer and stop.

These are the only two operations you can perform on a stack.

### Queues - Insertion

Fig. 3.5.2 shows a queue and its head and tail pointers. Remember, a queue is a first-in-first-out (FIFO) data structure. If we are to insert an item into a queue we must first check that the stack is not full. Having done this we shall increment the pointer and then insert the new data item into the cell pointed to by the head pointer. This method assumes that the cells are numbered from 1 upwards and that, when the queue is empty, the two pointers point to the same cell:

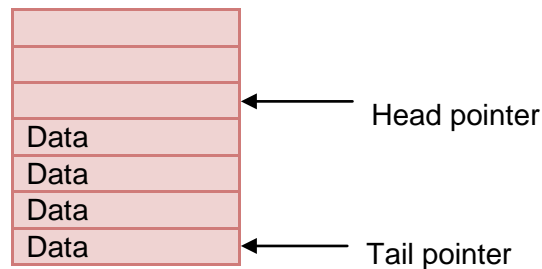


Fig. 3.5.2

The algorithm for insertion is:

1. Check to see if queue is full.
2. If the queue is full report an error and stop.
3. Insert new data item into cell pointed to by the head pointer.
4. Increment the head pointer and stop.

### Queues – Deletion/Reading

Before trying to delete an item, we must check to see that the queue is not empty. Using the representation above, this will occur when the head and tail pointers point to the same cell.

The algorithm for deletion is:

1. Check to see if the queue is empty.
2. If the queue is empty report error and stop.
3. Copy data item in cell pointed to by the tail pointer.
4. Increment tail pointer and stop.

These are the only two operations that can be performed on a queue.

### Trees - Insertion

When inserting an item into a binary tree, it is usual to preserve some sort of order. Consider the tree shown in Fig. 3.5.3 showing a tree containing data that is stored according to its alphabetic order.

To add a new value, we look at each node starting at the root. If the new value is less than the value at the node move left, otherwise move right. Repeat this for each node arrived at until there is no node. Insert a new node at this point and enter the data. If the tree is being stored in an array then follow the same logic but insert the item in a free space in the array and insert the pointer to it when the route through the array results in a X. The two pointers for the new item are also X's.

Now let's try putting "*Jack Spratt could eat no fat*" into a tree. *Jack* must be the root of the tree. *Spratt* comes after *Jack* so go right and enter *Spratt*. *could* comes before *Jack*, so go left and enter *could*. *eat* is before *Jack* so go left, it's after *could* so go right. This is continued to produce the tree in Fig. 3.5.3.

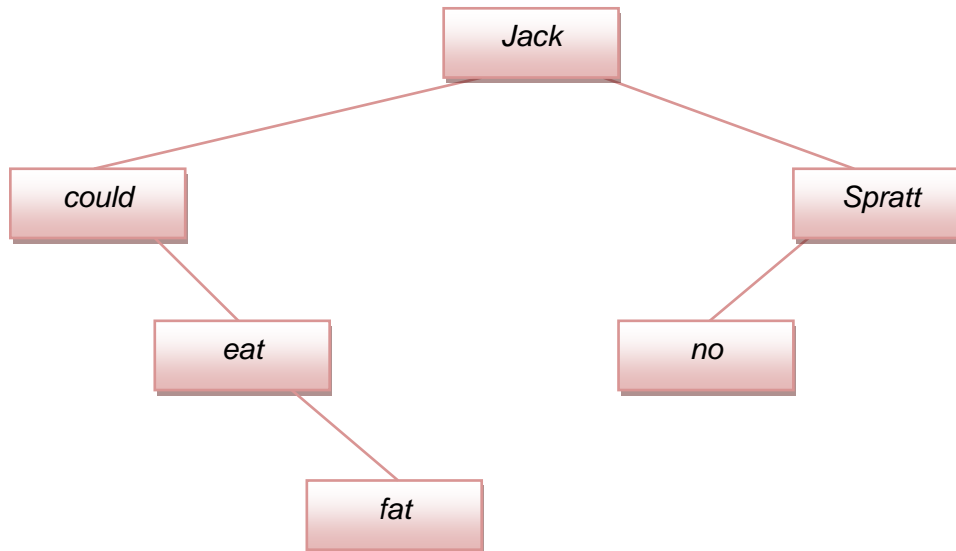


Fig. 3.5.3

The algorithm for this is:

1. If tree is empty enter data item at root and stop.
2. Current node = root.
3. Repeat steps 4 and 5 until current node is null.
4. If new data item is less than value at current node go left else go right.
5. Current node = node reached (null if no node).
6. Create new node and enter data.

Using this algorithm and adding the word *and* we follow these steps:

1. The tree is not empty so go to the next step.
2. Current node contains *Jack*.
3. Node is not null.
4. *and* is less than *Jack* so go left.
5. Current node contains *could*.
3. Node is not null.
4. *and* is less than *could* so go left.
5. Current node is null.
3. Current node is null so exit loop.
6. Create new node and insert *and* and stop.

We now have the tree shown in Fig. 3.5.4

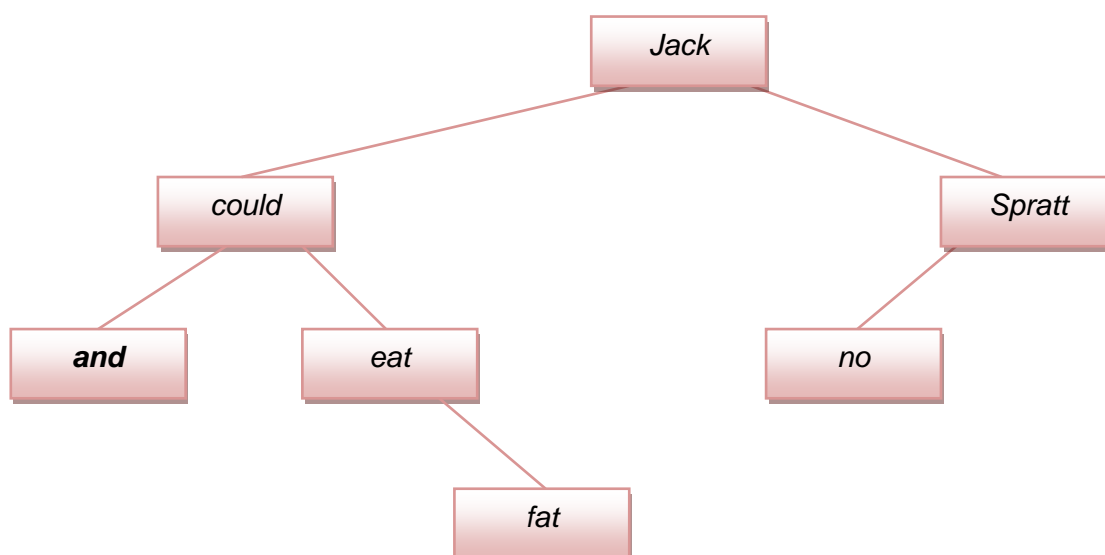


Fig. 3.5.4

### Trees - Deletion

Deleting a node from a tree is a fairly complex task. Suppose that, in Fig. 3.5.4 we wish to delete the node containing the word *could*. What do we do with the words *and*, *eat* and *fat*.

The difficulty is that each of the nodes is not only a data item but is also a part of the structure of the tree. If the data item is simply deleted it is like sawing a branch off the tree and everything else on that branch will also disappear.

The simplest way is to leave the structure of the tree unchanged by maintaining the word 'could' but labelling it as 'deleted' so that the structure stays as it was but the value is not read when the tree is searched.

The complicated way of doing it is to traverse the tree with root *could* and store all the values in a different data structure. (We could use an array, a stack, or a queue.) We must then insert these values back in the tree after removing the 'could'. Suppose we stored the values in such a way that they will be re-entered into the tree in the order *and*, *eat*, *fat*.

The result is shown in Fig.3.5.5

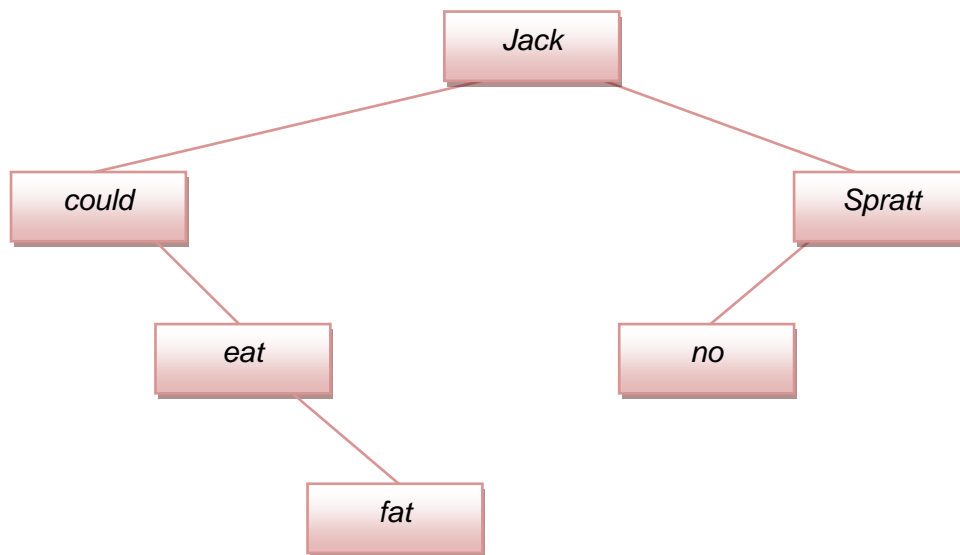


Fig. 3.5.5

Amendments are not normally carried out on a tree as they would change the order of the nodes even if we took extra special care with making sure that the data was saved, for that reason the first method would normally be used until the tree became over burdened with deleted data items.

### 3.5.c *Explain the difference between binary searching and serial searching, highlighting the advantages and disadvantages of each*

There are many different methods of searching but only two will be considered here. These methods are the serial search and the binary search.

The serial search expects the data to be in consecutive locations such as in an array. It does not expect the data to be in any particular order. To find the position of a particular value involves looking at each value in turn and comparing it with the value you are looking for. When the value is found you need to note its position. You must also be able to report that a value has not been found in some circumstances. This will happen when the item that is being searched for is not present.

This method can be very slow, particularly if there are a large number of values. The least number of comparisons is one; this occurs if the first item in the list is the one you want. However, if there are  $n$  values in the list, you will need to make  $n$  comparisons if the value you want is the last value or if the item being searched for is not in the list. This means that, on average, you will look at  $n/2$  values for each search. Clearly, if  $n$  is large, this can be a very large number of comparisons and can make the process of searching very slow.

Now suppose the list is sorted into ascending order as shown below:

Anne
Brian
Colin
Diane
Edward
Frank
Gloria
Hazel

Suppose we wish to find the position of Colin. Now compare Colin with the value that is in the middle of the table. In this case Diane and Edward are both near the middle; if there are two values in the middle the rule is to take the smaller value. As Colin is before Diane, we now look only at this list:

Anne
Brian
Colin

Now we only have a list of four entries. That is, we have halved the length of the list. We now compare Colin with Brian and find that Colin is greater than Brian so we use the list:

Colin
-------

which is less than half the length of the previous list. Comparing Colin with Colin we have found the position we want. This has only taken three comparisons.

Clearly, this is more efficient than the serial search method. However, the data must first be sorted into order and this can take some time. We need to compare these two methods using lists of different lengths. To do this we need to know the number of comparisons that need to be made for lists of different lengths. As we halve the lists each time we do the comparison, the number of comparisons is given by  $m$  where  $2^m$  is the smallest value greater than or equal to  $n$ , the number of values in the list.



Suppose there are 16 values in the list to start with. The successive lists will contain 16, 8, 4 and 2 values. This involves four comparisons. Now  $2^4$  is 16. Similarly, we only need 10 comparisons if the list contains 1000 values because  $2^{10}$  is equal to 1024, which is close to and greater than 1000.

Fig. 3.5.6 shows a comparison of the search times for the two methods allowing for the calculation of the mid points in the lists used by the binary search and for the times taken to make comparisons. It does not include sorting times for the binary search.

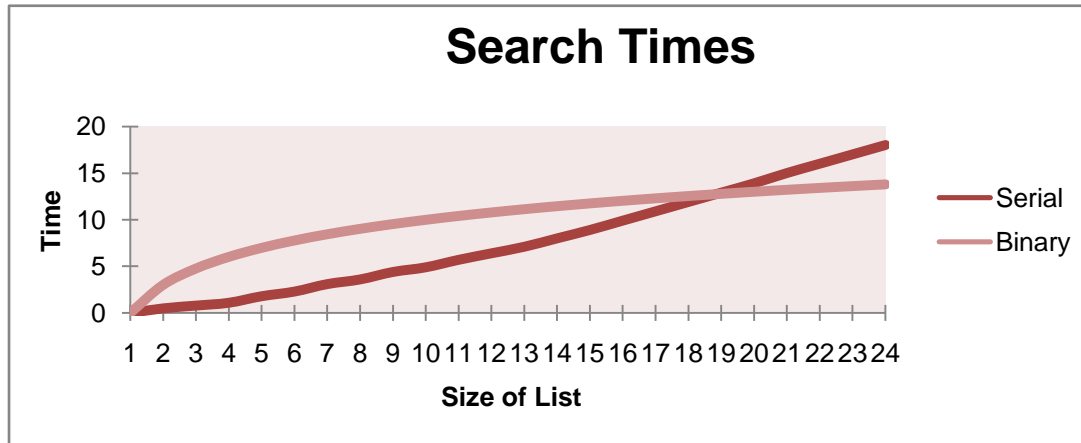


Fig. 3.5.6

### Serial Search

Let the data consist of  $n$  values held in an array called `DataArray` which has subscripts numbered from 1 upwards. Let  $X$  be the value we are trying to find. We must check that the array is not empty before starting the search.

The algorithm to find the position of  $X$  is:

1. If  $n < 1$  then report error, array is empty.
2. For  $i = 1$  to  $n$  do
  - a. If `DataArray[i] = X` then return  $i$  and stop.
3. Report error,  $X$  is not in the array and stop.

Note that the for loop only acts on the indented line. If there is more than one operation to perform inside a loop, make sure that all the lines are indented.

### Binary Search

**Be prepared! The idea of a binary search, that we keep looking at the middle value in smaller and smaller lists, is a simple one, but the algorithms needed to explain how it is done are very complicated.**

Assume that the data is held in an array as described above but that the data is in ascending order in the array. We must split the lists in two. If there is an even number of values, dividing by two will give a whole number and this will tell us where to split the list. However, if the list consists of an odd number of values we will need to find the integer part of it, as an array subscript must be an integer.

We must also make sure that, when we split a list in two, we use the correct one for the next search. Suppose we have a list of eight values. Splitting this gives a list of the four values in cells 1 to 4 and four values in cells 5 to 8. When we started we needed to consider the list in cells 1 to 8. That is the first cell was 1 and the last cell was 8. Now, if we move into the first list (cells 1 to 4), the first cell stays at 1 but the last cell becomes 4. Similarly, if we use the second list (cells 5 to 8), the first cell becomes 5 and the last is still 8. This means that if we use the first list, the first cell in the new list is unchanged but the last is changed. However, if we use the second list, the first cell is changed but the last is not changed. This gives us the clue of how to do the sort.

Here is the binary search algorithm:

1. While the list is not empty do
  - a. Find the mid-point cell in the current list.
  - b. If the value in this cell is the required value, return the cell position and stop.
  - c. If the value to be found is less than the value in the mid-point cell then
    - Make the search list the first half of the current search list
    - Else make the search list the second half of the current search list.
2. Report error, item not in the list.

A more detailed algorithm is given below that is useful if you wish to program the binary search. You would not be expected to be able to reproduce this during an examination.

### Binary Search Algorithm

In this algorithm, Vector is a one-dimensional array that holds the data in ascending order. X is the value we are trying to find and n is the number of values in the array.

```
{Initialisation}
First = 1
Last = n
Found = FALSE
{Perform the search}
WHILE First <= Last AND NOT Found DO
 {Obtain index of mid-point of interval}
 Mid = INT(First + Last) / 2
 {Compare the values}
 IF X < Vector[Mid] THEN
 Last = Mid - 1
 ELSE
 IF X > Vector[Mid] THEN
 First = Mid + 1
 ELSE
 Output 'Value is at ', Mid
 Found = TRUE
 ENDIF
 ENDIF
ENDIF
ENDWHILE
IF NOT Found THEN
 {Unsuccessful search}
 Output 'Value not in the list'
ENDIF
END
```

### 3.5.d Explain how to merge data files

Consider the two sorted lists:

2 4 7 10 15    and    3 5 12 14 18 26

In order to merge these two lists, we first compare the first values in each list, that is 2 and 3. 2 is less than 3 so we put it in the new list:

New = 2

Since 2 came from the first list we now use the next value in the first list and compare it with the number from the second list (as we have not yet used it). 3 is less than 4 so 3 is placed in the new list:

New = 2 3

As 3 came from the second list we use the next number in the second list and compare it with 4. This is continued until one of the lists is exhausted. We then copy the rest of the other list into the new list. The full merge is shown in Fig. 3.5.6.

First List	Second List	New List
2 4 7 10 15	3 5 12 14 18 26	2
2 4 7 10 15	3 5 12 14 18 26	2 3
2 4 7 10 15	3 5 12 14 18 26	2 3 4
2 4 7 10 15	3 5 12 14 18 26	2 3 4 5
2 4 7 10 15	3 5 12 14 18 26	2 3 4 5 7
2 4 7 10 15	3 5 12 14 18 26	2 3 4 5 7 10
2 4 7 10 15	3 5 12 14 18 26	2 3 4 5 7 10 12
2 4 7 10 15	3 5 12 14 18 26	2 3 4 5 7 10 12 14
2 4 7 10 15	3 5 12 14 18 26	2 3 4 5 7 10 12 14 15
2 4 7 10 15	3 5 12 14 18 26	2 3 4 5 7 10 12 14 15 18
2 4 7 10 15	3 5 12 14 18 26	2 3 4 5 7 10 12 14 15 18 26

Fig. 3.5.6

#### Algorithm for Merging two sorted files

The following algorithm merges two sets of data that are held in a two one-dimensional arrays called VectorA[1 to M] and VectorB[1 to N] into a third one-dimensional array called VectorC. VectorA and VectorB have been sorted into ascending order and if there are any duplicates they are only copied once into VectorC.

```
Merge(VectorA, VectorB, VectorC)
```

```
I = J = K = 1
```

```
WHILE I ≤ M AND J ≤ N DO
```

```
 IF VectorA(I) < VectorB(J) THEN
```

```
 VectorC(K) = A(I)
```

```
 I = I + 1
```

```
 K = K + 1
```

```
 ELSE
```

```
 IF VectorA(I) > B(J) THEN
```

```
 VectorC(K) = VectorB(J)
```

```
 J = J + 1
```

```
 K = K + 1
```

```
 ELSE
```

```
 VectorC(K) = VectorA(I)
```

```
 I = I + 1
```

```
 J = J + 1
```

```
 K = K + 1
```

```
 ENDIF
```

```
 ENDIF
```

```
ENDWHILE
```

```
IF I > M THEN
```

```
 FOR R = J TO N STEP 1 DO
```

```
 VectorC(K) = VectorB(R)
```

```
 K = K + 1
```

```
 NEXT R
```

```
ELSE
```

```
 FOR R = I TO M STEP 1 DO
```

```
 VectorC(K) = VectorA(R)
```

```
 K = K + 1
```

```
 NEXT R
```

```
ENDIF
```

```
END
```

### 3.5.e *Explain the differences between the insertion and quick sort methods, highlighting the characteristics, advantages and disadvantages of each*

Sorting is placing values in an order such as numeric order or alphabetic order. As we have seen in the last two sections, sorting files of data into some sort of order is often very important because if a file is unsorted it is not possible to merge it with another or to perform a binary sort on it.

The file may be in ascending or descending order. For example, the values:

3 5 6 8 12 16 25

are in ascending numeric order and:

William Rose Mary James Hazel Donald Anne

are in descending alphabetic order.

There are many methods that can be used to sort lists. You only need to understand two of them. These are the insertion sort and the quick sort. This Section describes the two sorts in general terms; the next section gives the algorithms.

#### Insertion Sort

In this method we compare each number in turn with the numbers before it in the list. We then insert the number into its correct position.

Consider the list:

20 47 12 53 32 84 85 96 45 18

We start with the second number, 47, and compare it with the numbers preceding it. There is only one and it is less than 47, so no change in the order is made. We now compare the third number, 12, with its predecessors. 12 is less than 20 so 12 is inserted before 20 in the list to give the list:

12 20 47 53 32 84 85 96 45 18

This is continued until the last number is inserted in its correct position. In Fig. 3.5.7 the blue numbers are the ones before the one we are trying to insert in the correct position. The red number is the one we are trying to insert:

2	4	1	5	3	8	8	9	4	1	Original list, start with second number.
0	7	2	3	2	4	5	6	5	8	
2	4	1	5	3	8	8	9	4	1	No change needed.
0	7	2	3	2	4	5	6	5	8	
			5	3	8	8	9	4	1	Now compare 12 with its predecessors.
			3	2	4	5	6	5	8	
1	2	4	5	3	8	8	9	4	1	Insert 12 before 20.
2	0	7	3	2	4	5	6	5	8	
1	2	4	5	3	8	8	9	4	1	Move to next value.
2	0	7	3	2	4	5	6	5	8	
1	2	4	5	3	8	8	9	4	1	53 is in the correct place.
2	0	7	3	2	4	5	6	5	8	
1	2	4	5	3	8	8	9	4	1	Move to the next value.
2	0	7	3	2	4	5	6	5	8	
1	2	3	4	5	8	8	9	4	1	Insert it between 20 and 47
2	0	2	7	3	4	5	6	5	8	
1	2	3	4	5	8	8	9	4	1	Move to the next value.
2	0	2	7	3	4	5	6	5	8	
1	2	3	4	5	8	8	9	4	1	84 is in the correct place.
2	0	2	7	3	4	5	6	5	8	
1	2	3	4	5	8	8	9	4	1	Move to the next value.
2	0	2	7	3	4	5	6	5	8	
1	2	3	4	5	8	8	9	4	1	85 is in the correct place.
2	0	2	7	3	4	5	6	5	8	
1	2	3	4	5	8	8	9	4	1	Move to the next value.
2	0	2	7	3	4	5	6	5	8	
1	2	3	4	5	8	8	9	4	1	96 is in the correct place.
2	0	2	7	3	4	5	6	5	8	
1	2	3	4	5	8	8	9	4	1	Move to the next value.
2	0	2	7	3	4	5	6	5	8	
1	2	3	4	4	5	8	8	9	1	Insert 45 between 32 and 47.
2	0	2	5	7	3	4	5	6	8	
1	2	3	4	4	5	8	8	9	1	Move to the next value.
2	0	2	5	7	3	4	5	6	8	
1	1	2	3	4	4	5	8	8	9	Insert 18 between 12 and 20.
2	8	0	2	5	7	3	4	5	6	

Fig. 3.5.7

### Quick Sort

At first this method appears to be very slow. This is only because of the length of time it takes to explain the method. In fact, for long lists, it is a very efficient method of sorting.

Consider the same list as we used earlier:

20 47 12 53 32 84 85 96 45 18

We use two pointers. One pointer points left and the other points right. Let the number we are trying to place in its correct position be marked in red in the following description.

Start with the left pointer in the first position and the right pointer in the last position. Let the number in the first position be the one we are trying to place in its correct position. We have:

20 47 12 53 32 84 85 96 45 18



Compare the two numbers that are being pointed to. If they are in the wrong order, swap them. We have

18 47 12 53 32 84 85 96 45 20



Now keep moving the pointer that is not attached to the red number towards the red number until either the two pointers meet or the two numbers being pointed to are in the wrong order. In this case the left pointer moves to 47 and 47 and 20 are in the wrong order, so swap them:

18 47 12 53 32 84 85 96 45 20



18 20 12 53 32 84 85 96 45 47



Now again move the pointer not at the red number towards the red number until either the two pointers meet or two numbers need to be swapped:

18 20 12 53 32 84 85 96 45 47



18 20 12 53 32 84 85 96 45 47



18 20 12 53 32 84 85 96 45 47



18 20 12 53 32 84 85 96 45 47



18 20 12 53 32 84 85 96 45 47



18 20 12 53 32 84 85 96 45 47



18 20 12 53 32 84 85 96 45 47



Swap

18 12 20 53 32 84 85 96 45 47

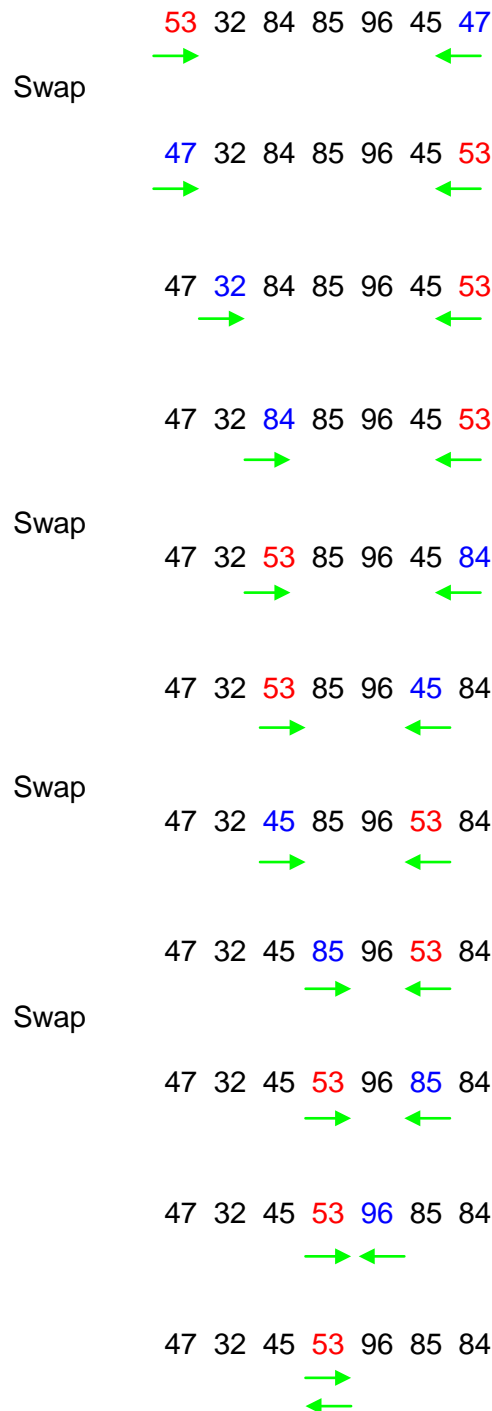


18 12 20 53 32 84 85 96 45 47



Pointers coincide. Now you will find that the number 20 is in its correct position. That is, all the numbers to the left of 20 are less than 20 and all the numbers to the right of 20 are greater than 20.

We now split the list into two; the one to the left of the 20 and the one to the right of the 20. We then quick sort each of these lists. The following shows the steps for the right hand list:



The number 53 is now in the correct position for this list. Split this list in two, as before, to give the two lists:

47 32 45 and 96 85 84

and sort these two lists using the quick sort method. When all the sublists have a single number, they can be put back together to form a sorted list.



### Algorithm for an Insertion Sort

The following is an algorithm for the insertion sort:

{Look at each card in turn, starting with the second card}

FOR Position = 2 to n DO

    CurrentValue = Vector[Position]

    {Keep value of current number}

    {and store it in position 0}

    Vector[0] = CurrentValue

    {Now look at each number to the left of current number (Position – 1) and move

    each number right if it is greater than CurrentValue. If the number is not greater

    than CurrentValue, insert CurrentValue.}

    Pointer = Position - 1

    WHILE Vector[Pointer] > CurrentValue DO

        Vector[Pointer + 1] = Vector[Pointer]

        {Move left one place}

        Pointer = Pointer – 1

    ENDWHILE

    Vector[Pointer + 1] = CurrentValue

ENDFOR

END

### Algorithm for a Quick Sort

Clearly, the quick sort algorithm is more complex. In fact, when you have applied the quick sort to the original list, you end up with two unsorted lists that have to be sorted. These are sorted using quick sort. Hence we have a recursive algorithm.

Suppose the data are held in a one-dimensional array Vector with a lower bound LB and an upper bound UB for the array subscripts. For example, if the array is from Vector[1] to Vector[10], LB is 1 and UB is 10. Similarly, if the data to be sorted are in cells Vector[4] to Vector[7], LB is 4 and UB is 7

The algorithm for this is shown below:

```
QuickSort(Vector, LB, UB)

IF LB <> UB THEN

 {There is more than one element in Vector, therefore it needs sorting}
 Set I = LB {Left pointer}
 Set J = UB {Right pointer}
 REPEAT
 WHILE I <> J AND Vector[I] < Vector[J] DO
 {Move right pointer left}
 Make J = J - 1

 ENDWHILE

 IF I <> J THEN swap Vector[I] and Vector[J]

 WHILE I <> J AND Vector[I] < Vector[J] DO

 {Move left pointer right}

 Make I = I + 1

 ENDWHILE

 IF I <> J THEN swap Vector[I] and Vector[J]

 UNTIL I = J {Value now in correct position}

 {So sort left sublist}

 QuickSort(Vector, LB, I - 1)

 {Now sort right sublist}

 QuickSort(Vector, I + 1, UB)

ENDIF

END
```

### 3.5 Example Questions



1. State the difference between dynamic and static data structures giving an example of each.

(3)

- A. -A dynamic data structure can change in size during a program run, while a static data structure maintains a fixed size.

-Dynamic List/Tree.

-Static Array.

*Notice: An array may be able to change size in certain circumstances, but it is the only example we've got, so don't labour the point!*

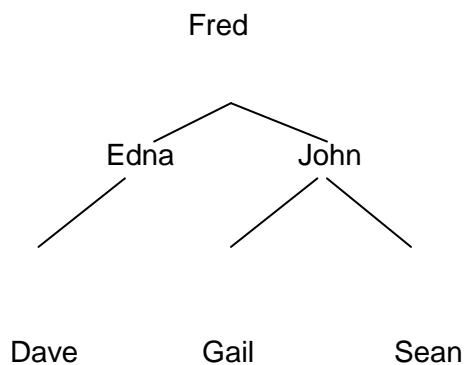
2. a) Show how a binary tree can be used to store the data items Fred, Edna, John, Sean, Dave, Gail in alphabetic order.

(4)

- b) Explain why problems may arise if John is deleted from the tree and how such problems may be overcome.

(4)

A. a)



Marks:

-Root node

-Edna and John correctly positioned in relation to Fred

-Edna's subtree

-John's subtree

- b) -Data in a tree serves two purposes (one is to be the data itself) the other is to act as a reference for the creation of the subtree below it

-If John is deleted there is no way of knowing which direction to take at that node to find the details of the data beneath it.

-Solution 1 is to store John's subtree in temporary storage and then rewrite it to the tree after John is deleted. (The effect is that one member of the subtree will take over from John as the root of that subtree).

-The data, John, can be marked as deleted so that the data no longer exists but it can maintain its action as an index for that part of the tree so that the subtree can be correctly negotiated.

*Notice: The tree can be reflected in a vertical mirror quite justifiably. For this reason it is probably worthwhile making a note on the diagram of what rule has been used, just in case it does not match the examiner's version.*

3. Describe **two** types of search routine, giving an indication of when it would be advisable to use each. (6)

A. -Serial search carried out on data list by...

-comparing each key with the desired one in turn...

-until found or error message issued.

-Useful if data keys are not in order or if the number of items of data is small.

-Binary search carried out by...

-continually comparing the middle key in the list with the required key...

-and hence repeatedly splitting the list into two equal halves until...

-middle value = required key or error if new list is empty.

-Fast method of searching if the data list is large and the data is sorted on the key.

(1 per -, max 6)

4. Describe the steps in sorting a list of numbers into order using an insertion sort. (4)

A. -Each value in turn is compared with the values already in the list and is...

- inserted in the correct location.

-Once a value has been inserted the algorithm is repeated by comparing all the values in the list, in turn...

-with the next value in the list.

-This continues until all the values have been inserted.

-Example. (4)

*Notice: A difficult topic to ask a question about because the examination is relatively short. Consequently there is little time for each of the questions. If the question does not look like this it will almost certainly consist of a number of data items, numerical or alphabetic, and the candidate will be asked to explain a sort technique by showing how the data items are affected by a particular sorting method. There is nothing in the syllabus to say that sort algorithms will not be asked for, however, with the time pressure in the examination such a question would be most unlikely. A good idea would be to practice the techniques of sorting regularly using simple, short, unsorted lists and get used to showing each stage of the sort by writing out the list of items again and again, just as it is done in the section above.*

5. Given two sorted lists describe an algorithm for merging them into one sorted list.

(6)

A. -Select one value from each list

-While neither list is empty

-Compare the values in the lists

-Copy the smaller value to the new list

-Read the next value in the list that has had a value copied.

-End while loop

-When one list is empty copy the remainder...

-of the other list to the new list.

*Notice: The solution takes no notice of what would happen if the two original lists contained duplicate values. Neither does the solution consider the case when one of the original lists is empty. Both of these points are worth consideration and would earn credit if mentioned in an exam answer.*

*The “algorithm” is not in any formal style. The syllabus states that the candidate should be able to describe the differences between the methods, this presupposes that the methods need to be known and may be asked to be described. However it does not expect the methods (algorithms) in any formal form. The mark points suggested are mark points to look for in the solution and not a rigid form of solution, a prose solution would be acceptable but not encouraged at this level and would be very difficult to make precise..*

## 3.6 High Level Language Programming Paradigms

based programming  
line programming (as in the UNIX command line)  
native programming  
reactive programming  
rule-based programming (as in ContextJS/ContextS/ContextS.0)  
rule-based programming, contrasted with imperative programming  
rule-based programming as in Snobol  
rule-based programming  
• Constraint programming  
• Constraint logic programming  
• Logic programming (as in Prolog)  
• Inductive logic programming  
• Functional programming (as in Lisp, Scheme and Haskell)  
• Declarative programming (as in spreadsheets)  
• Reactive programming  
• Event-driven programming  
• Feature-oriented programming  
• Function-level programming, contrasted with value-level programming  
Function-level programming, contrasted with declarative programming

This Chapter uses a number of different programming languages to illustrate the points

being explained. You will not be asked to write program code in the examination. However, you may well find it helpful, in answering examination questions, to include code in your answers. This is perfectly satisfactory; indeed it will often help you to clarify what you are saying. If you do write code, the accuracy of the syntax will not be marked simply the logic that you manage to portray. If your work contains evidence then examiners will use it to see if you understand the question and can explain your answer.

Thus, treat the code in this section simply as an explanation of the different facilities available in different programming paradigms. Do not think that you have to be able to program in any specific language

Much of the work here will be shaded, remember that this signifies optional work which you may find interesting.

### 3.6.a *Identify a variety of programming paradigms (low level, object-oriented, declarative and procedural)*

‘A variety of programming whats?’ is the normal question. Don’t worry about it, a programming paradigm is simply a method of programming, a way of programming, an example of programming. So why don’t we say ‘A variety of ways of programming’? Simply because there is a specific piece of jargon which we should use, but don’t worry about it.

So, programming paradigms are simply methods of programming. Initially, computers were programmed using binary. This was difficult and led to many errors that were difficult to find. Programs written in binary are said to be written in machine code, this is a very low-level programming paradigm.

As we saw in section 3.2, in order to make programming easier, assembly languages were developed. These replaced machine code functions with mnemonics and addresses with labels. Assembly language programming is also a low-level paradigm (remember it simply means a method). It was the next stage, though, after having computers that you could only communicate with in binary, so it came second after machine code. Because of this assembly languages are called second generation languages. Figure 3.6.1 shows an assembly language program that adds together two numbers and stores the result. (Notice that it has been set out a bit more formally than the ones in section 3.2.

Label	Function	Addresses	Comments
	LDA	X	Load the accumulator with the value of X
	ADD	Y	Add the value of Y to the accumulator
	STA	Z	Store the result in Z
	STOP		Stop the program
X:	20		Value of X = 20
Y:	35		Value of Y = 35
Z:			Location for result

Figure 3.6.1

Although this assembly language is an improvement over machine code, it is still prone to errors and code is difficult to debug, correct and maintain.

The next advance was the development of procedural languages. These are called third generation languages and are also known as high-level languages. These languages are problem oriented as they use terms appropriate to the type of problem being solved. For example, COBOL (Common Business Oriented Language) uses the language of business. It uses terms like file, move and copy. FORTRAN (FORmula TRANslation) and ALGOL (ALGOrithmic Language) were developed mainly for scientific and engineering problems. BASIC (Beginners All purpose Symbolic Instruction Code) was developed to enable more people to write programs. All these languages follow the procedural paradigm. That is, they describe, step by step, exactly the procedure that should be followed to solve a problem.

The problem with procedural languages is that it can be difficult to reuse code in other programs and modifying solutions when there is a small change in the problem to be solved can be very difficult, often resulting in the program needing to be rewritten. In order to address these problems, object-oriented languages (like Eiffel, Smalltalk and Java) were developed. In these languages data, and the methods of manipulating the data, are kept as a single unit called an object. The only way that a user can access the data is via the object's methods. This means that, once an object is fully working, it cannot be corrupted by the user because they cannot change any of the parameters. It also means that the internal workings of an object may be changed without affecting any code that uses the object.

A further advance was made when declarative programming paradigms were developed. In these languages the computer is told what the problem is, not how to solve the problem. Given a database the

computer searches for a solution. The computer is not given a procedure to follow as in the languages discussed so far.



### 3.6.b *Explain with examples, the terms object-oriented, declarative, and procedural as applied to high level languages, showing an understanding of typical uses*

Procedural languages specify, exactly, the steps required to solve a problem. These languages use the constructs sequence, selection and repetition. For example, to find the area of a rectangle the steps are:

1. Read the length
2. Read the breadth
3. Multiply the length by the breadth
4. Output the result

In C++ this can be coded as:

```
cout << "Enter the length: ";

cin >> Length;

cout << "Enter the breadth: ";

cin >> Breadth;

Area = Length * Breadth;

cout << "The area is "

 << Area << endl;
```

Here each line of code is executed one after the other in sequence.

See if you can understand the code here. Relate it to the initial algorithm and remember that no one is suggesting that you should learn C++, but simply try to understand the logic.

There are a number of methods (constructs) that are common to all procedural languages. The abilities to do selection and repetition are the most obvious. Students are advised to refresh their memories from section 2.2 for this work.

The point to note with these procedural languages is that the programmer has to specify exactly what the computer is to do.

Procedural languages are used to solve a wide variety of problems. Some of these languages are more robust than others. This means that the compiler will not let the programmer write statements that may lead to problems in certain circumstances. As stated earlier, there are procedural languages designed to solve scientific and engineering problems while others are more suitable for solving business problems. There are some that are particularly designed for solving problems of control that need real time solutions.

Procedural languages may use functions and procedures but they always specify the order in which instructions must be used to solve a problem. The use of functions and procedures help programmers to reuse code but there is always the danger of variables being altered inadvertently.

In the 1970s it was realised that code was not easily reused and there was little security of data in a program. Also, the real world consists of objects not individual values. My car, registration number W123ARB, is an object. Kay's car, registration number S123KAY, is another object. Both of these objects are cars and cars have similar attributes such as registration number, engine capacity, colour, and so on. That is, my car and Kay's car are instances of a type of object, a class, called cars. In order to model the real world, the object-oriented Programming (OOP) paradigm was developed. Unfortunately, OOP requires a large amount of memory and, in the 1970s, memory was expensive and CPUs still lacked power. This slowed the development of OOP. However, as memory became cheaper and CPUs more powerful, OOP became more popular. By the 1980s Smalltalk, and later Eiffel, had become well established. These were true object-oriented languages. Although OOP languages are procedural in nature, the way that they hold the data and the methods that can be used on the data together makes them different, so OOP is considered to be a new programming paradigm.

**The following gives an example, but may be omitted.**

The following is an example, using Java, of a class that specifies a rectangle and the methods that can be used to access and manipulate the data:

```
class Shapes {
 /* Shapes Version 1 by Roger Blackford July 2001

 This illustrates the basic ideas of OOP
 */
 // Declare three object variables of type Rectangle
 Rectangle small, medium, large;

 // Create a constructor where the initial work is done
 Shapes () {
 // Create the three rectangles
 small = new Rectangle(2, 5);
 medium = new Rectangle(10, 25);
 large = new Rectangle(50, 100);

 //Print out a header
 System.out.println("The areas of the rectangles are:\n");

 //Print the details of the rectangles
 small.write();
 medium.write();
 large.write();
 } //end of constructor Shapes.

 //All programs have to have a main method
 public static void main(String [] args) {
 //Start the program from its constructor
 new Shapes ();
 } //end of main method.
} //end of class Shapes.

class Rectangle {
 //Declare the variables related to a rectangle
 int length;
 int width;
 int area;

 //Create a constructor that copies the initial values into the object's variables
 Rectangle (int w, int l) {
 width = w;
 length = l;
 //Calculate the area
 area = width * length;
 } //end of constructor Rectangle

 //Create a method to output the details of the rectangle
 void write () {
 System.out.println("The area of a rectangle " + width + " by " + length + " is " + area);
 } //end of write method.
} //end of constructor
```

This example contains two classes. The first is called Shapes and is the main part of the program. It is from here that the program will run. The second class is called Rectangle and it is a template for the description of a rectangle.

The class Shapes has a constructor called Shapes, which declares two objects of type Rectangle. This is a declaration and does not assign any values to these objects. In fact, Java simply says that, at this stage, they have null values. Later, the new statement creates actual rectangles. Here small is given a width of 2 and a length of 5, medium is given a width of 10 and a length of 25 and large is given a width of 50 and a length of 100. When a new object is created from a class, the class constructor, which has the same name as the class, is called.

The class Rectangle has a constructor that assigns values to width and length and then calculates the area of the rectangle.

The class Rectangle also has a method called write( ). This method has to be used to output the details of the rectangles.

In the class Shapes, its constructor then prints a heading and the details of the rectangles. The latter is achieved by calling the write method. Remember, small, medium and large are objects of the Rectangle class. This means that, for example, small.write( ) will cause Java to look in the class called Rectangle for a write method and will then use it.

Another programming paradigm is the declarative one. Declarative languages tell the computer what is wanted but do not provide the details of how to do it. These languages are particularly useful when solving problems in artificial intelligence such as medical diagnosis, fault finding in equipment and oil exploration. The method is also used in robot control. An example of a declarative language is Prolog. The idea behind declarative languages is shown in Fig. 3.6.2.

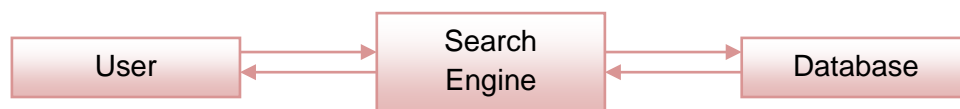


Fig. 3.6.2

Here the user inputs a query to the search engine, which then searches the database for the answers and returns them to the user. (Notice the similarity with expert systems). For example, using Prolog, suppose the database is:

```
female(jane).
female(anne).
female(sandip).
male(chnarnjit).
male(jaz).
male(tom).
```

Note that in Prolog values start with a lowercase letter and variables start with an uppercase letter. A user may want to know the names of all the males. The query:

```
male(X).
```

will return:

```
X = charnjit
X = jaz
X = tom
```

Notice that the user does not have to tell Prolog how to search for the values of X that satisfy the query. In a procedural language the database may be held in a two-dimensional array Gender as shown below:

#### Array Gender

	1	2
1	female	jane
2	female	anne
3	female	sandip
4	male	charnjit
5	male	jaz
6	male	tom

In Visual Basic we could write:

```
For count = 1 To 6
 If Gender[count, 1] = "male" Then
 picResults.Print Gender[count, 2]
 End If
```

This is fairly straightforward. However, suppose we now add to the Prolog database the following data:

```
parent(jane,mary).
parent(jane, rajinder).
parent(charnjit, mary).
parent(charnjit, rajinder).
parent(sandip, atif).
parent(jaz, atif).
```

which would be read as parent(X,Y) means X is a parent of Y.

Suppose we wish to know the name of the mother of Atif. In Prolog we use the query:

```
parent(X, atif), female(X).
```

Prolog will output:

```
X = sandip
```

because it will find that sandip and jaz satisfy the condition to be parents and it will then go and find which of these is female. That reduces the choices to one.

To get a list of all the fathers, we can simply write:

```
parent(X, Y), male(X).
```

The result is:

```
X = charnjit Y = mary
```

```
X = charnjit Y = rajinder
```

```
X = jaz Y = atif
```

If we only want a list of fathers we use the underscore and create the query:

```
parent(X, _), male(X).
```

and the result is:

```
X = charnjit
```

```
X = charnjit
```

```
X = jaz
```

**The important point is that the programmer does not have to tell the computer how to answer the query. There are no FOR ... NEXT, WHILE ... DO ... or REPEAT ... UNTIL ... loops as such. There is no IF ... THEN ... statement. The system simply consists of a search engine and a database of facts and rules.**

**3.6.c** *Discuss the concepts, and using examples, show an understanding of, data encapsulation, classes and derived classes, and inheritance when referring to object-oriented languages*

All object oriented languages have classes and derived classes and use the concepts of encapsulation, and inheritance. In this Section we consider these concepts in a general way, using diagrams rather than any particular language.

Encapsulation

Data encapsulation is the concept that data can only be accessed via the methods provided by the class. This is shown in Fig. 3.6.3 where the objects, that is, instantiations of a class, are prevented from directly accessing the data by the methods:

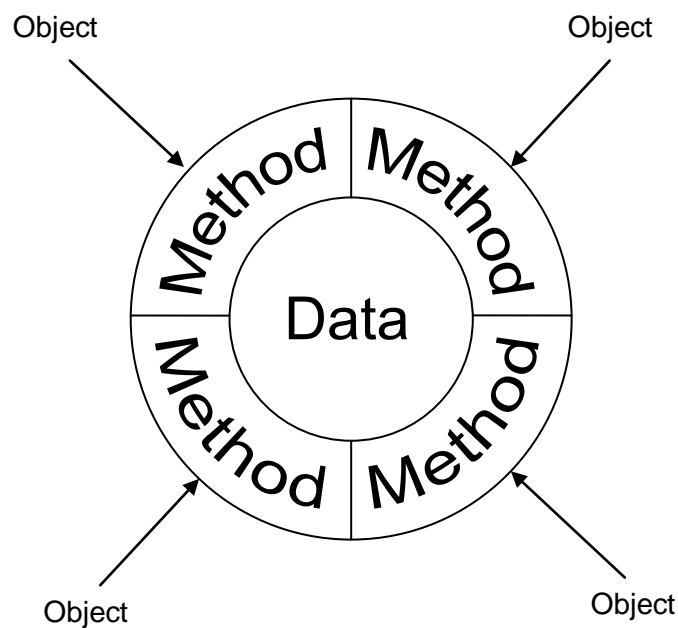


Fig. 3.6.3

Objects can only access the data by using the methods provided. An object cannot manipulate the data directly. In the case of the rectangle class, an object of this class cannot directly calculate its area. That is, we **cannot** write:

```
area := myRectangle.width * myRectangle.length;
```

where myRectangle is an object of type Rectangle.

To find the area of myRectangle, class Rectangle must provide a suitable method. The only way to find the area is to use the write( ) method which outputs the area. If a user wishes to access the width and length of a rectangle, the class must provide methods to do this. Methods to return the width and length are given below.

```
Integer getWidth() {
 getWidth := width;
}//end of getWidth method.
```

```
integer getLength() {
 getLength := length;
}//end of getLength method.
```

myRectangle can now use these methods to get at the width and length. However, it cannot change their values. To find the perimeter we can write:

```
myWidth := myRectangle.getWidth();
myLength := myRectangle.getLength();
myPerimeter := 2 * (myWidth + myLength);
```

Thus, an object consists of the data and the methods provided by the class. The concept of data being only accessible by means of the methods provided is very important as it ensures data integrity. Once a class has been written and fully tested, neither its methods nor the data can be tampered with. Also, if the original design of a method is found to be inefficient, the design can be changed, unknowingly to the user, without the user's program being affected.

### Inheritance

Another powerful concept is that of inheritance. Inheritance allows the re-use of code and the facility to extend the data and methods without affecting the original code. In the following diagrams, we shall use a rounded rectangle to represent a class. The name of the class will appear at the top of the rectangle, followed by the data followed by the methods.

Consider the class Person that has data about a person's name and address and a methods called outputData( ) that outputs the name and address, getName( ) and getAddress( ) that return the name and address respectively. This is shown in Fig.3.6.4.

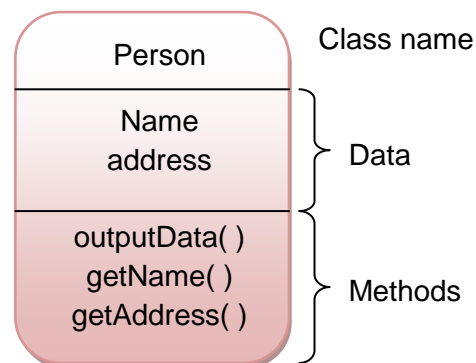


Fig. 3.6.4

Now suppose we want a class Employee that requires the same data and methods as Person and also needs to store and output an employee's National Insurance number. Clearly, we do not wish to rewrite the contents of the class person. We can do this by creating a class called Employee that inherits all the details of the class Person and adds on the extra data and methods needed. This is shown in Fig. 3.6.5 where the arrow signifies that Employee inherits the data and methods provided by the class Person. Person is called the *super-class* of Employee and Employee is the *derived class* from Person. An object of type Employee can use the methods provided by Employee and those provided by Person.



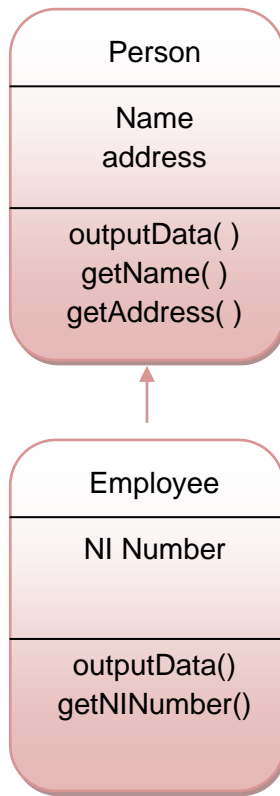


Fig. 3.6.5

Now suppose we have two types of employee; one is hourly paid and the other is paid a salary. Both of these require the data and methods of the classes Person and Employee but they also need different data to one another. This is shown in Fig. 3.6.6

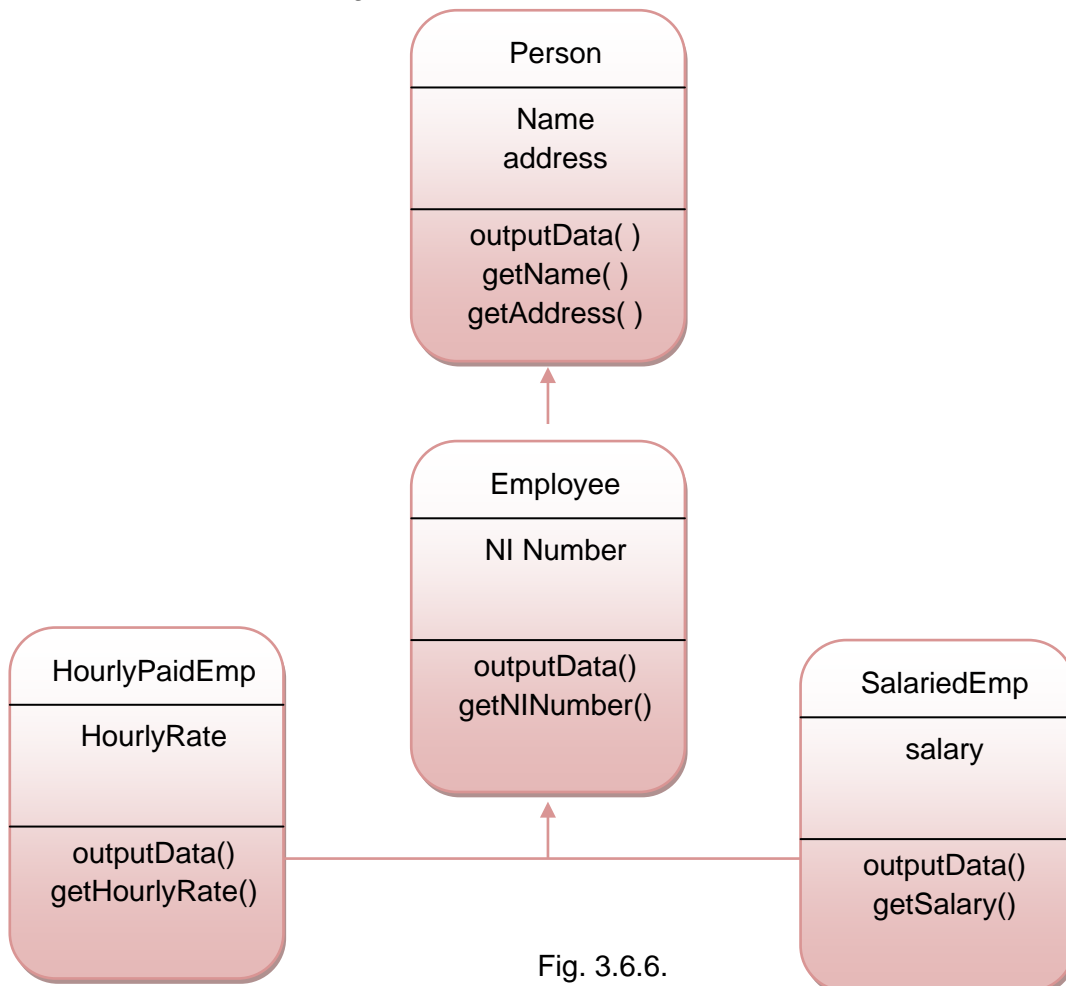


Fig. 3.6.6.

We have just explained the meanings of terms such as data encapsulation, class and inheritance. However, sometimes the examiner may ask you to simply state the meanings of these terms. In this case a simple definition is all that is required. Note also that there will only be one (or possibly two) marks for this type of question. The following definitions would be satisfactory answers to questions that say 'State the meaning of the term ... '.

#### Definitions

*Data encapsulation* is the combining together of the variables and the methods that can operate on the variables so that the methods are the only ways of using the variables..

A *class* describes the variables and methods appropriate to some real-world entity.

An *object* is an instance of a class and is an actual real-world entity.

*Inheritance* is the ability of a class to use the variables and methods of a class from which the new class is derived.

### **3.6.d** *Understand the purpose of the Unified Modelling Language (UML)*

In 3.6.b we saw that a particular type of programming is called object oriented programming where the facts about an entity and the methods to be used on those facts are stored together, creating an object which can be manipulated. Programming, using these objects, is very different to other types of programming and methods have had to be developed for doing the tasks involved. These methods for planning and explaining object oriented programming techniques are a meta language (in much the same way that BNF in section 3.7.f is a meta language for defining terms and expressions in procedural languages).

The Unified Modelling Language consists of a set of descriptive diagrammatic representations to describe the stages to produce effective object oriented programs. The number and type of diagrams is something of a movable feast, the current thinking being that 13 different types of diagram are necessary. We will be looking at 7 types in the very briefest way. Candidates should be able to follow a diagram and answer questions about it, they should also be able to create their own diagrams, given a simple scenario or set of facts. Detailed work will not be expected but the following sections will give a taste of what may be expected if further courses are studied at a higher level.

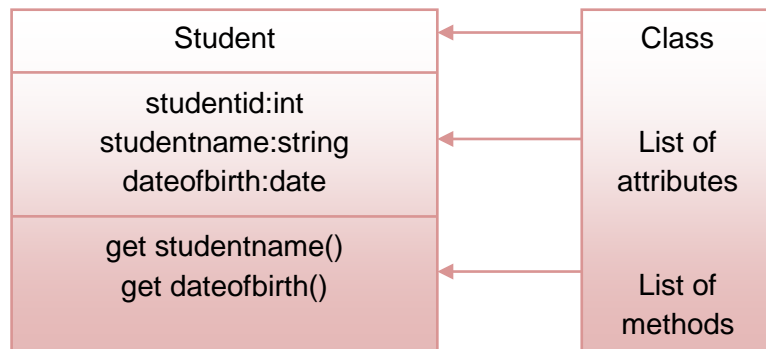
### 3.6.e & f Create and interpret class, object, use case and communication diagrams and interpret state, sequence and activity diagrams

#### Class diagrams

Object oriented programming uses classes. A class is 'an entity of a given system that provides an encapsulation of the functionality of a given entity.' Meaning: A class is something real like a set of students or forms, which have certain things that are true about it and certain ways of getting at those things that are true. Further, it will allow other classes to use some of the facts if they are allowed to and it can use some facts from other classes if it is allowed to.

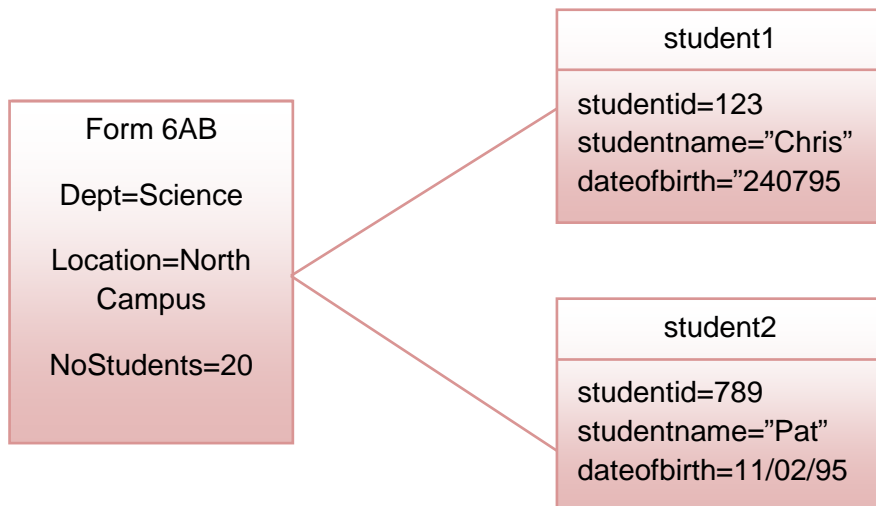
A method of defining this class is to write about it (!) but that can be imprecise and difficult to understand. Another method is to draw it and put all the facts about it into the diagram.

A class diagram is a rectangle, divided into 3 parts. The first gives the name of the class, the second gives the facts that should be known about any element belonging to that class and the third gives methods that can be used to look at the facts that are stored in the class.



#### Object Diagrams

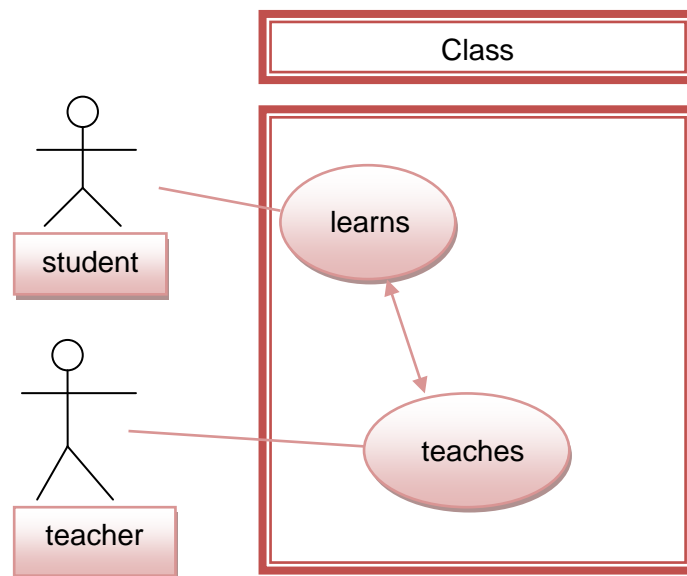
Think of classes as groups of real things. That means that each class must represent at least one entity, so if there is a class called 'Student' it will presuppose that there are some students which are represented by it. Each student will have specific data to match to each of the attributes. A specific student is an object of this class and a diagram which shows an individual student's attributes is an object diagram.



### Use case Diagrams

Use case diagrams are, at this level, very simple. They are more like business oriented diagrams than computing ones because they depict what is meant to be going on in a system rather than how it is done.

All systems have users otherwise the system is not worth producing. In our example these users are human but they do not need to be. These users are called the 'actors' and are depicted on the diagram as stick people! They have some interaction or carry out some process. These processes are in oval boxes and are drawn inside a rectangle so that all the processes are held together. There can also be some interaction shown between the processes.



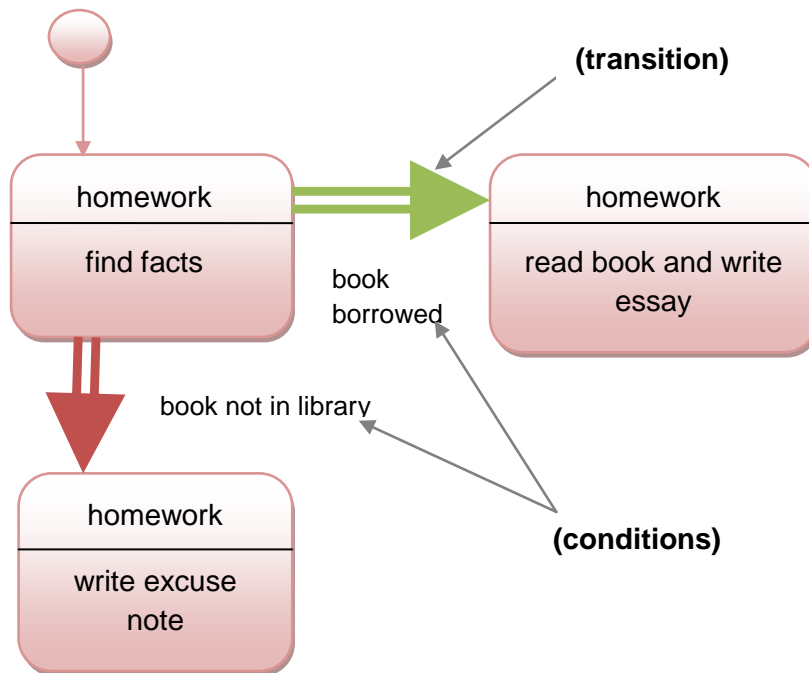
This shows that a student and a teacher are both participants in this procedure and that they have different uses of the procedure. It also shows that the uses are related by a process.

## State Diagrams

A state diagram shows how an object may behave through the various processes of a system. It is a bit like a cross between a DFD and a flow diagram.

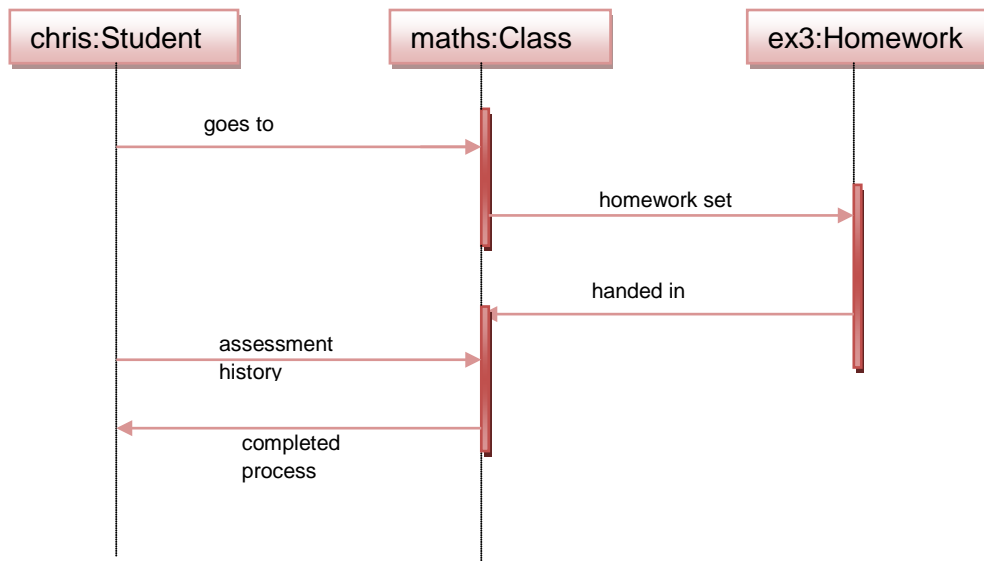
It starts with a shaded in circle to show the initial state before anything has happened. Arrows are used to show the flow of activity to different outcomes for the object as it goes through the system.

Take as an example the production of homework after the teacher has taught the lesson. The process from the point of view of the student might be:



## Sequence Diagrams

Sequence diagrams show how the objects in the classes interact with one another. The classes (can be other things as well) are along the top of the diagram. Each one has a dotted line underneath it which shows how long that object from that class can be said to exist in the process, it is called its 'lifeline'. The dotted lines sometimes turn into long thin boxes, these show where the methods of that class have been activated to do something.

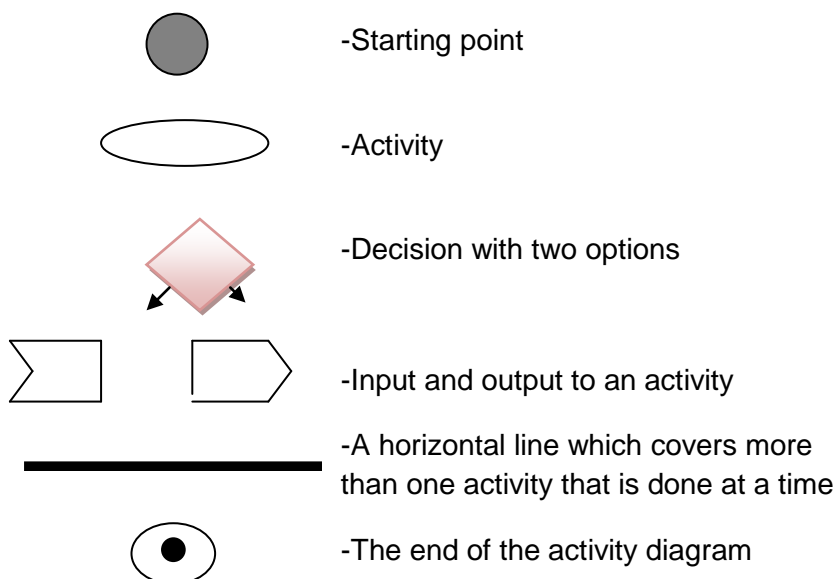


Notice that the homework has a finite life and the lifeline has stopped, whereas the student and the class continue.

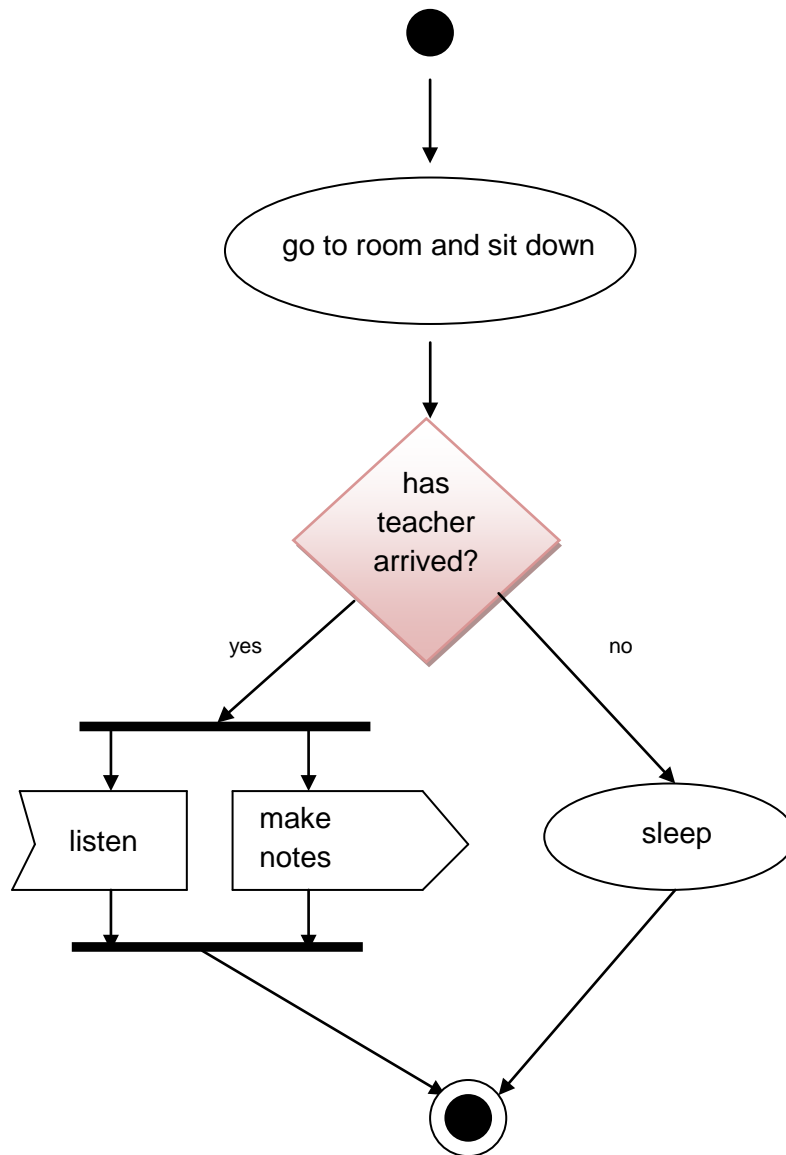
## Activity Diagram

An activity diagram is very like a flow chart in procedural programming. However, a flow chart shows the development of the logic behind the code whereas an activity diagram is a diagram which shows the activities necessary to get the object into a particular state.

Activity diagrams use a standard set of symbols:



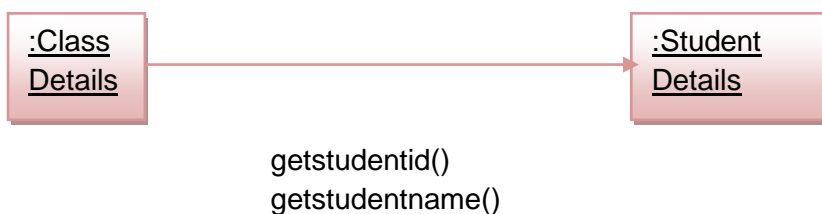
Using the symbols above an activity diagram for a lesson can be created:



### Communication Diagrams

Communication diagrams are used to show how different objects combine to pursue a common purpose. Each object is contained in a rectangle and the data that they share with one another is placed beside arrows showing in which direction the data is flowing.

Imagine a teacher writing details in a mark book for a new class:



An important point about UML is that diagrams are used to model the system and that different diagrams are used for the different things that need to be modelled. The work here has only scratched the surface of UML but if you could follow the logic of the diagrams then you have an understanding of the different representations and will be able to interpret diagrams in an exam.



### 3.6.g *Discuss the concepts and, using examples, show an understanding of backtracking, instantiation, predicate logic and satisfying goals when referring to declarative languages*

In section 3.6.a, we saw that, in declarative languages, the programmer can simply state what is wanted having declared a set of facts and rules. We now look at how this works using examples of Prolog scripts. In order to do this, we shall use the following facts:

```
female(jane).
female(anne).
female(sandip).
male(charnjit).
male(jaz).
male(tom).
parent(jane,mary).
parent(jane, rajinder).
parent(charnjit, mary).
parent(charnjit, rajinder).
parent(sandip, atif).
parent(jaz, atif).
```

Remember that variables must start with an uppercase letter; constants start with a lowercase letter.

Suppose we ask:

```
male(X).
```

Prolog starts searching the database and finds `male(charnjit)` matches `male(X)` if `X` is given the value `charnjit`. We say that `X` is *instantiated* to `charnjit`. Prolog now outputs:

```
X = charnjit
```

Prolog then goes back to the database and continues its search. It finds `male(jaz)` so outputs:

```
X = jaz
```

and again continues its search. It continues in this way until the whole database has been searched. The complete output is:

```
X = charnjit
X = jaz
X = tom
No
```

The last line means that there are no more solutions.

The query `male(X)` is known as a goal to be tested. That is, the goal is to find all `X` that satisfy `male(X)`. If Prolog finds a match, we say that the search has succeeded and the goal is true. When the goal is true, Prolog outputs the corresponding values of the variables.

Now we shall add the rule:

```
father(X, Y) :- parent(X, Y), male(X).
```

This rule states that `X` is father of `Y` if (the `:-` symbol) `X` is a parent of `Y` AND (the comma) `X` is male.

The database now looks like this:

```
female(jane).
female(anne).
female(sandip).
male(chnarnjit).
male(jaz).
male(tom).
parent(jane,mary).
parent(jane, rajinder).
parent(chnarnjit, mary).
parent(chnarnjit, rajinder).
parent(sandip, atif).
parent(jaz, atif).
father(X, Y) :- parent(X, Y), male(X).
```

Suppose our goal is to find the father of `rajinder`. That is, our goal is to find all `X` that satisfy:

```
father(X, rajinder).
```

In the database and the rule the components `female`, `male`, `parent` and `father` are called predicates and the values inside the parentheses are called arguments. Prolog now looks for the predicate `father` and finds the rule:

```
father(X, Y) :- parent(X, Y), male(X).
```

In this rule `Y` is instantiated to `rajinder` and Prolog starts to search the data base for:

```
parent(X, rajinder)
```

This is the new goal. Prolog finds the match:

```
parent(jane, rajinder)
```

if `X` is instantiated to `jane`. Prolog now uses the second part of the rule:

```
male(X)
```

with  $X = \text{jane}$ . That is, Prolog's new goal is  $\text{male}(\text{jane})$  which fails. Prolog does not give up at this stage but *backtracks* to the match:

```
parent(jane, rajinder)
```

and starts again, from this point in the database, to try to match the goal:

```
parent(X, rajinder)
```

This time Prolog finds the match:

```
parent(charnjit, rajinder)
```

with  $X$  instantiated to  $\text{charnjit}$ . The next step is to try to satisfy the goal

```
male(charnjit)
```

This is successful so:

```
parent(charnjit, rajinder) and male(charnjit)
```

are true. Thus  $\text{father}(\text{charnjit}, \text{rajinder})$  is true and Prolog returns:

```
X = charnjit
```

Prolog continues to see if there are any more matches. There are no more matches so Prolog outputs :

```
No
```

We met recursion in 2.3.g. Remember that we only need to be able to trace a recursive routine, not produce one, and bearing that in mind we will take a look at the use of recursion in a declarative language. This can be used to create alternative versions for a rule. The Fig. 3.6.7 shows how ancestor is related to parent:

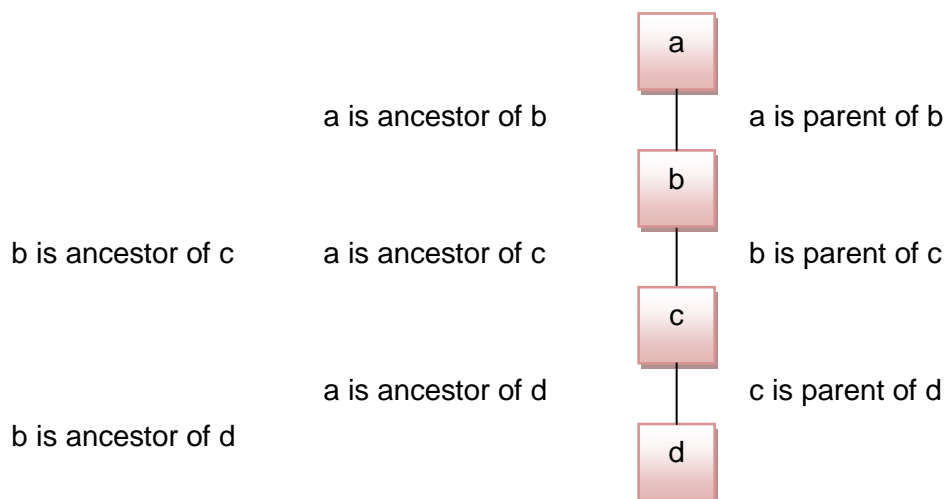


Fig. 3.6.7

This shows that  $X$  is an ancestor of  $Y$  if  $X$  is a parent of  $Y$ . But it also shows that  $X$  is an ancestor of  $Y$  if  $X$  is a parent of  $Z$  and  $Z$  is a parent of  $Y$ . It also shows that  $X$  is an ancestor of  $Y$  if  $X$  is a parent of  $Z$ , and  $Z$  is a parent of  $W$  and  $W$  is a parent of  $Y$ . This can continue forever. Thus the rule is recursive. In Prolog we require two rules that are written as:

```
ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- parent(X, Z),
 ancestor(Z, Y).
```

The first rule states that X is an ancestor of Y if X is a parent of Y. In Fig.3.6.7, this is saying that a is an ancestor of b, b is an ancestor of c and c is an ancestor of d.

The second rule is in two parts. Let us see how it works using Fig.3.6.7 which represents the database:

```
parent(a, b).
parent(b, c).
parent(c, d).
```

by setting the goal:

```
ancestor(a, c).
```

Prolog finds the first rule and tries to match parent(a, c) with each predicate in the database. Prolog fails but does not give up. It backtracks and looks for another rule for ancestor. Prolog finds the second rule and tries to match:

```
parent(a, Z).
```

It finds:

```
parent(a, b)
```

so instantiates Z to b.

This is now put into the second part of the rule to produce:

```
ancestor(b, c).
```

This means that Prolog has to look for a rule for ancestor. It finds the *first* rule:

```
ancestor(X, Y) :- parent(X, Y).
```

Thus Prolog looks for:

```
parent(b, c)
```

and succeeds.

This means that with X = a, Y = c we have Z = b and the second rule succeeds. Therefore Prolog returns Yes.

Now try to trace what happens if the goals are

```
ancestor(a,d)
```

and

```
ancestor(c, b).
```

You should find that the first goal succeeds and the second fails.

This form of programming is based on the mathematics of predicate calculus. Predicate calculus is a branch of mathematics that deals with logic. All we have done in this Section is based on the rules of predicate calculus. Prolog stands for Programming in LOGic and its notation is based on that of predicate calculus. In predicate calculus the simplest structures are atomic sentences such as:

Mary *loves* Harry

Philip *likes* Zak

The words in italics are part of the names of relationships. We also have atomic conclusions such as:

Mary *loves* Harry if Harry *loves* Mary

John *likes* dogs if Jane *likes* dogs

Frank *likes* x if x *likes* computing

In the last atomic conclusion, x is a variable. The meaning of the conclusion is that Frank likes anything (or anybody) that likes computing.

Joint conditions use the logical operators OR and AND, examples of which are:

John likes x if x is female AND x is blonde

Mary loves Bill OR Mary loves Colin if Mary loves Don

The atomic formulae that serve as conditions and conclusions may be written in a simplified form. In this form the name of the relation is written in front of the atomic formula. The names of the relations are called predicate symbols. Examples are:

loves(Mary, Harry)

likes(Philip, Zak)

We use the symbol '(' to represent if and have:

loves(Mary, Harry) ( loves(Harry, Mary)

likes(John, dogs) ( likes(Jane, dogs)

The 'and' is represented by a comma in the condition part of the atomic conclusion. For example:

likes(John, x) ( female(x), blonde(x)

The 'or' is represented by a comma in the conclusion. For example:

female(x), male(x) ( human(x)

These examples show the connection between Prolog and predicate calculus. You do not need to understand how to manipulate the examples you have seen any further than has been shown in this Section.

AS with the previous Section we include here the definitions of terms used in this Section. Remember, they can be used when a question says "State the meaning of the term ... '.

### Definitions

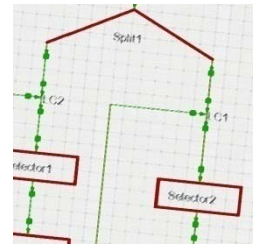
Backtracking is going back to a previously found successful match in order to continue a search.

Instantiation is giving a variable in a statement a value.

Predicate logic is a branch of mathematics that manipulates logical statements that can be either True or False.

A goal is a statement that we are trying to prove whether or not it is True or False.

## 3.7 Programming Techniques



### 3.7.a Explain how functions, procedures and their related variables may be used to develop a program in a structured way, using stepwise refinement

A complex problem needs to be broken down into smaller and smaller sub-problems until all the sub-problems can be solved easily otherwise the problem will be too complex to allow for a satisfactory solution. This process is called step-wise refinement.

Consider the problem of calculating the wages for an hourly paid worker. The worker is paid £6.50 per hour for up to 40 hours and time-and-a-half for all hours over 40. Tax and National Insurance contributions have to be deducted. This can be represented by Fig.3.7.1.

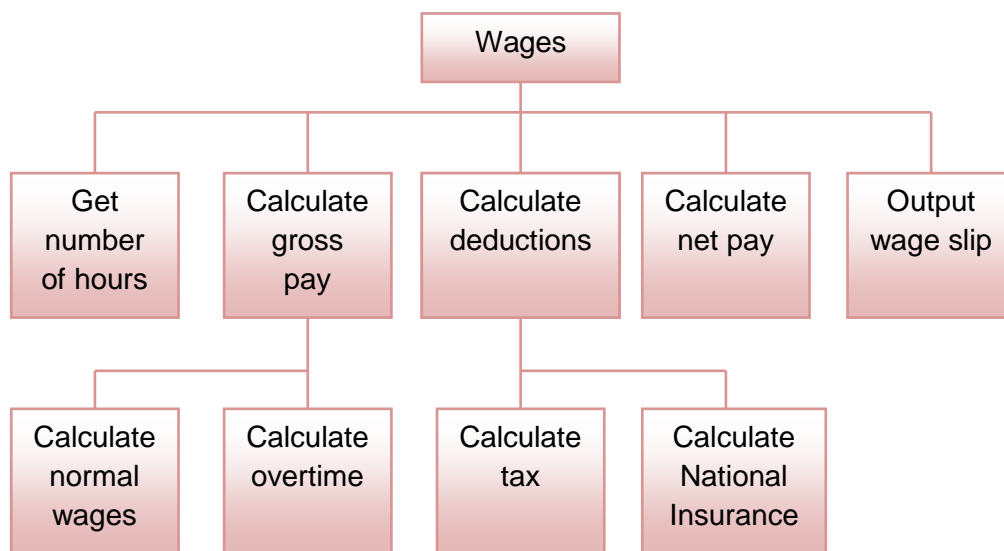


Fig. 3.7.1

An alternative way of writing this is to use numbered statements. This can be easier if there are many sub-problems to be solved:

#### 1. Wages

##### 1.1 Get number of hours

##### 1.2 Calculate gross pay

###### 1.2.1 Calculate normal wages

###### 1.2.2 Calculate overtime

##### 1.3 Calculate deductions

###### 1.3.1 Calculate tax

###### 1.3.2 Calculate National Insurance

##### 1.4 Calculate net pay

##### 1.5 Output wages slip

Either of these designs can be turned into a series of functions and procedures. The program could be called Wages and consist of the following functions and procedures:

Wages	
GetHours( )	returns an integer in range 0 to 60
CalculateWages(Hours)	returns gross wage
CalculateNormalWages(Hours)	returns wage for up to 40     hours
CalculateOvertime(Hours)	returns pay for any hours over 40
CalculateDeductions(GrossWage)	returns total deductions
CalculateTax(GrossWage)	returns tax due
CalculateNI(GrossWage)	returns N.I. due
CalculateNetWage(GrossWage, Deductions)	returns net wage after deductions
Procedure OutputResults(Hours, GrossWage, Tax, NI, Deductions, NetWage)	Procedure to print the wage slip

A procedure is a small section of code designed to carry out a definable task.

A function is a procedure which returns a single value to the program that called it.

Here we can see that if a single value is to be returned, the simplest way to do this is to use a function. If there are no values to be returned, then a procedure should be used. If more than one value is to be returned, a procedure should be used.

For example, in the above, we may have:

```
Int GetHours() or GetHours() As Integer
```

These statements state that the function will return an integer value and it does not expect any values to be fed into it:

```
Double CalculateNormalWages(Int Hours)
```

or

```
CalculateNormalWages(Hours As Integer) As Double
```

expects to be given an integer value as input and returns a value of type Double.

*Note: If you are programming in C, C++ or Java, there are no procedures. These languages only use functions. A function has to be typed. That is, the programmer must specify the type of value to be returned. This is true in all languages. In C, C++ and Java, if a function is not going to return a value, its return type is void. That is, no value is actually returned.*

Parameters will be discussed further in section 3.7.b



### 3.7.b Describe the use of parameters, local and global variables as standard programming techniques

The previous Section discussed stepwise refinement, this leads to modules that can be programmed easily. Each module is a solution to an individual problem and each module has to interface with other modules. As long as the interfaces are clearly specified, each module can be given to a different programmer to code. All that the programmers need to know is the problem and how its solution must communicate with the solutions to other modules. This means that two programmers may happen to use the same name for a variable. Also, the programmers will need to pass values to other modules and be able to accept values from other modules.

Let us first consider how data can be input to a module, be it a function or a procedure. This is done by means of parameters.

The function below, written in Visual Basic, finds the perimeter of a rectangle when given its length and breadth. This is not the only way of finding the perimeter and it probably is not the best way. However, it has been written like this in order to illustrate certain programming points.

```
Public Function PerimeterOfRectangle(X As Integer, Y As Integer) As Integer

 X = 2 * X

 Y = 2 * Y

 PerimeterOfRectangle = X + Y

End Function
```

In this function X and Y are integers the values of which must be passed to the function before it can find the area of the rectangle. These variables are called *formal parameters*. To use this function, another program will have to call it and provide the values for X and Y. This can be done by means of a statement of the form:

```
Perimeter = PerimeterOfRectangle(4, 6)
```

or we can use:

```
A = 3
B = 4

Perimeter = PerimeterOfRectangle(A, B)
```

In both of these statements the variables inside the brackets (4 and 6 in the first example and A and B in the second) are called *actual parameters*. How the values are passed to the function or procedure depends on the programming language. In the first example the values 4 and 6 are stored in the variables X and Y. This is called passing the parameters by value whereas, in the second example, the addresses of A and B are passed to the function so that X and Y have the same address as A and B respectively, this is called passing the parameter by reference because it is a reference to where the value is that is being passed over, the function then has to go and get it. Thus we have two different ways of passing parameters.

It is interesting to see the effect of passing values by reference (sometimes called passing by address). Here is the function described above and a copy of the calling function in Visual Basic:

```
Public Function PerimeterOfRectangle(X As Integer, Y As Integer) As Integer
 X = 2 * X
 Y = 2 * Y
 PerimeterOfRectangle = X + Y
End Function
```

```
Private Sub cmdShow_Click()
 Dim A As Integer
 Dim B As Integer
 Dim Perimeter As Integer
 A = 3
 B = 4
 picResults.Print "Before call to Sub A = "; A; " and B = "; B
 Perimeter = PerimeterOfRectangle(A, B)
 picResults.Print "Perimeter = "; Perimeter
 picResults.Print "After call to Sub A = "; A; " and B = "; B;
End Sub
```

Fig.3.7.1 shows the output when this program is run.

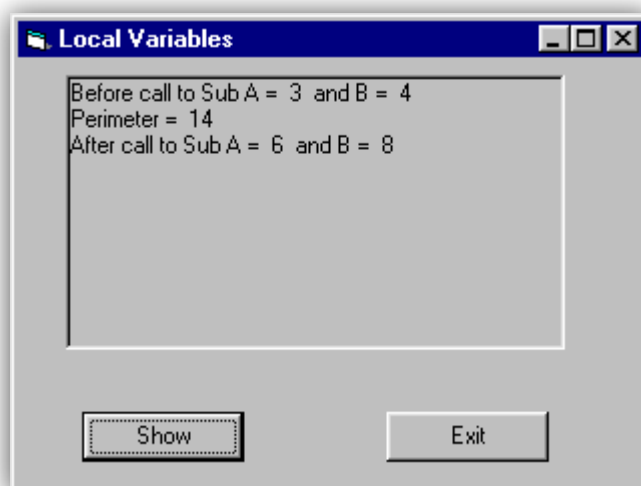


Fig.3.7.1

Notice that after the function has been run the values of A and B have changed. This is because the addresses of A and B were passed not their actual values. When the algorithm then doubled those values with a line like:

```
X = 2 * X
```

the effect is to replace the number that was in X with a different number.

Visual Basic can pass parameters by value as well as by reference but we have to use the ByVal key word if we want values to be passed by value. Here is a modified form of the Visual Basic function together with the output from running the modified program:

```
Public Function PerimeterOfRectangle(ByVal X As Integer, ByVal Y As Integer) As Integer

 X = 2 * X

 Y = 2 * Y

 PerimeterOfRectangle = X + Y

End Function
```

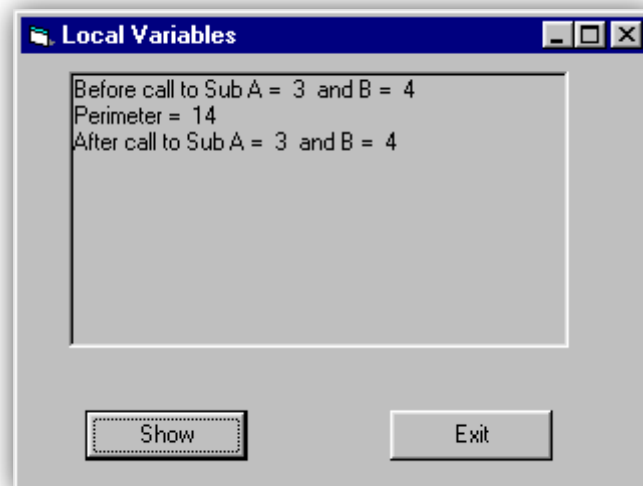


Fig. 3.7.2

Variables can have different values in different parts of the program. Look at the following Visual Basic code and its output, shown in Fig. 3.7.3.

```
Private Sub cmdShow_Click()
```

```
 Dim A As Integer
```

```
 Dim B As Integer
```

```
 Dim C As Integer
```

```
 Dim Perimeter As Integer
```

```
 A = 3
```

```
 B = 4
```

```
 C = 5
```

```
 picResults.Print "Before call to Sub A = "; A; " and B = "; B
```

```
 Perimeter = PerimeterOfRectangle(A, B)
```

```
 picResults.Print "Perimeter = "; Perimeter
```

```
 picResults.Print "After call to Sub A ="; A; " and B = "; B; " and C = "; C
```

```
End Sub
```

```
Public Function PerimeterOfRectangle(X As Integer, Y As Integer) As Integer
```

```
 Dim C As Integer
```

```
 C = 10
```

```
 picResults.Print "In Sub C = "; C
```

```
 X = 2 * X
```

```
 Y = 2 * Y
```

```
 PerimeterOfRectangle = X + Y
```

```
End Function
```

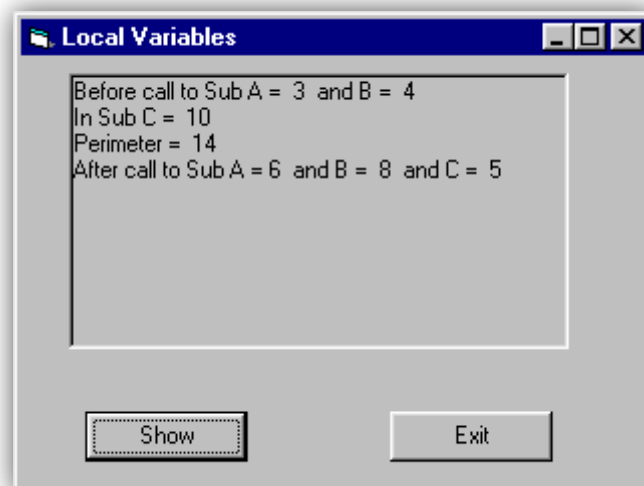


Fig. 3.7.3

This shows that C has a different value in the function PerimeterOfRectangle to in the calling function cmdShow\_Click.

C is said to be a **local** variable and the C in PerimeterOfRectangle is stored in a different address to the C in cmdShow\_Click. Local variables only exist in the block in which they are declared. So a variable C which is a local variable and is declared in two different parts of an algorithm can be thought of as being two different variables.

This is very helpful as it means that different programmers, writing different routines, do not have to worry about the names of variables used by other programmers. However, it is sometimes useful to be able to use the same variable in many parts of a program. To do this, the variable has to be declared as **global**, this means that it can be used anywhere in the program. In Visual Basic this is done by means of the statement:

```
Public C As Integer
```

which is placed in a module. If we do this with the previous example the code becomes:

```
Public C As Integer 'This is in the module

Private Sub cmdShow_Click()

 Dim A As Integer

 Dim B As Integer

 Dim Perimeter As Integer

 A = 3

 B = 4

 C = 5

 picResults.Print "Before call to Sub A = "; A; " and B = "; B; " and C = "; C

 Perimeter = PerimeterOfRectangle(A, B)

 picResults.Print "Perimeter = "; Perimeter

 picResults.Print "After call to Sub A ="; A; " and B = "; B; " and C = "; C

End Sub

Public Function PerimeterOfRectangle(X As Integer, Y As Integer) As Integer

 C = 10

 picResults.Print "In Sub C = "; C

 X = 2 * X

 Y = 2 * Y

 PerimeterOfRectangle = X + Y

End Function
```

Fig. 3.7.4 shows that the value of C, when changed in the function PerimeterOfRectangle, is changed in the calling routine also. In fact it is changed throughout the program:

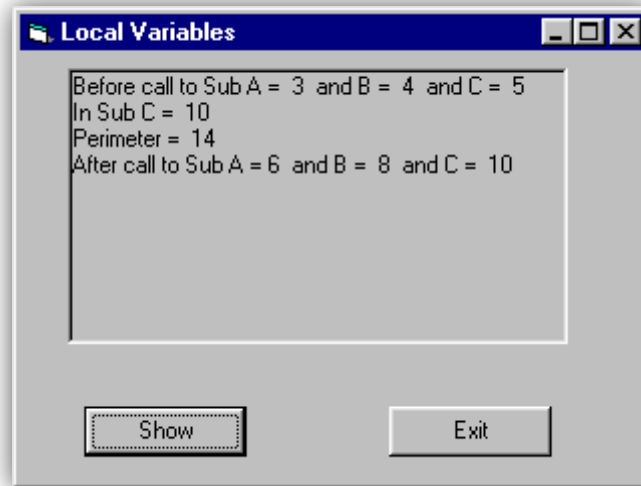


Fig. 3.7.4

What happens if a variable is declared as both global and local? The following code declares C as global and C as local in the function PerimeterOfRectangle and the result of running it is shown in Fig.3.7.5. Notice that the value of C in the function cmdShow\_Click is not changed although its value is changed in PerimeterOfRectangle. This is because C is declared as a local variable in this function and this means that the global C is not used.

```

Public C As Integer 'global declaration

Private Sub cmdShow_Click()

 Dim A As Integer

 Dim B As Integer

 Dim Perimeter As Integer

 A = 3

 B = 4

 C = 5

 picResults.Print "Before call to Sub A = "; A; " and B = "; B; " and C = "; C

 Perimeter = PerimeterOfRectangle(A, B)

 picResults.Print "Perimeter = "; Perimeter

 picResults.Print "After call to Sub A ="; A; " and B = "; B; " and C = "; C

End Sub

Public Function PerimeterOfRectangle(X As Integer, Y As Integer) As Integer

 Dim C As Integer 'local declaration

 C = 10

 picResults.Print "In Sub C = "; C

 X = 2 * X

 Y = 2 * Y

 PerimeterOfRectangle = X + Y

End Function

```

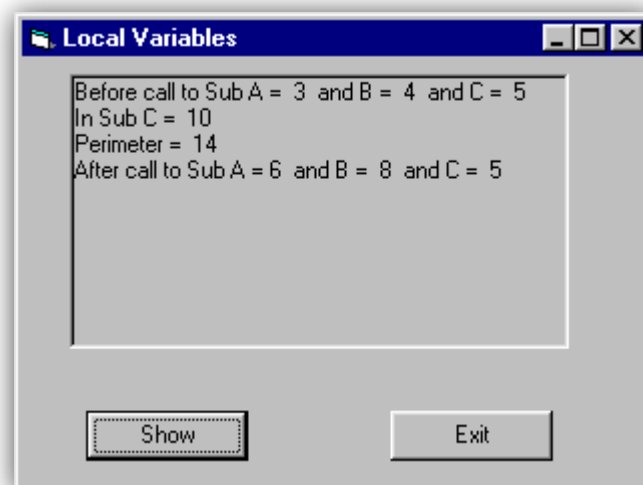


Fig. 3.7.5

### Summary

A parameter is a value that can be passed to a subroutine, be it a procedure or a function.

A parameter can be passed by value which means that the actual value is passed across and that the old value is still there when the subroutine is finished.

A parameter passed by reference (or address) is passed as the contents of an address, so if the subroutine changes it the old value is effectively rubbed out.

A local variable only exists in the subroutine in which it is declared which means that the same variable name can stand for lots of different values in different subroutines.

A global variable is a single variable throughout a program.



### 3.7.c Explain how a stack is used to handle procedure calling and parameter passing.

When a procedure or function is called, the computer needs to know where to return to when the function or procedure is completed. That is, the return address must be known. Further, functions and procedures may call other functions and procedures which means that not only must several return addresses be stored but they must be retrieved in the right order. This can be achieved by using a stack. Fig. 3.7.6 shows what happens when three functions are called, one after another. The numbers represent the addresses of the instructions following the calls to functions.

#### Main program

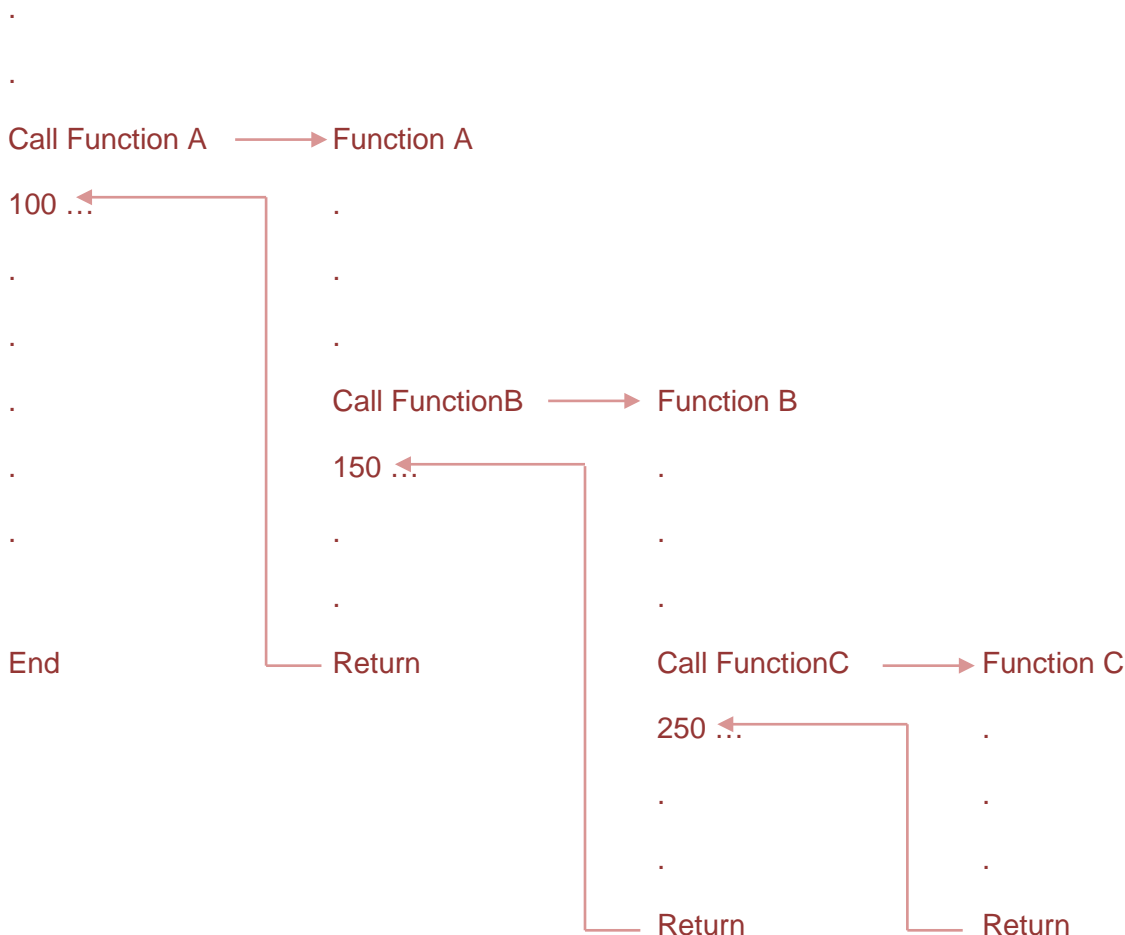


Fig. 3.7.6

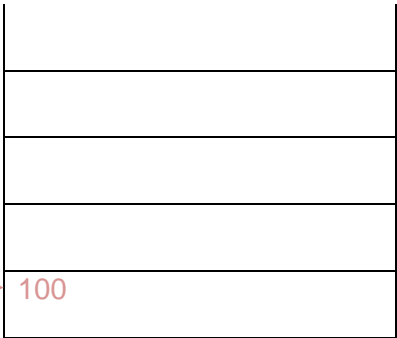
Notice that the return addresses will have to be stored somewhere, otherwise the computer will not know where to go to get back to the original code that called the functions in the first place. Also, it is necessary to remember the order in which they were done otherwise the computer will go to the wrong place (If, after doing function C it remembered the 100 to go back to, the rest of functions A and B would never be done. Because of this it is essential that the computer not only remembers the return addresses 100, 150 and 250 but that it remembers them in the right order. When the returns take place the addresses will be needed in the order 250, 150 then 100. That is, the last address stored is the first address needed on returning from a function. This means that we need a data structure that provides a last in first out facility. A stack does precisely this, so we store the return addresses in a stack. In the above example, the addresses will be stored in the stack each time a function is called and will be removed from the stack each time a return instruction is executed. This is shown in Fig. 3.7.7.

**Calls and Returns**

**Stack**

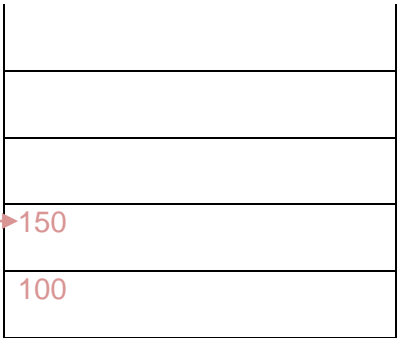
Call Function A  
Push return address onto stack

Stack pointer → 100



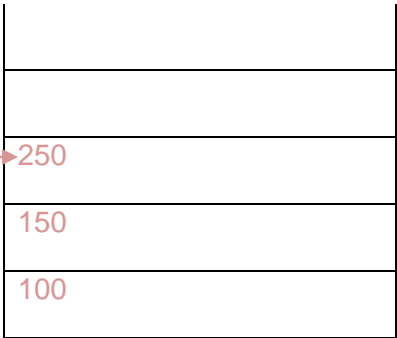
Call Function B  
Push return address onto stack

Stack pointer → 150



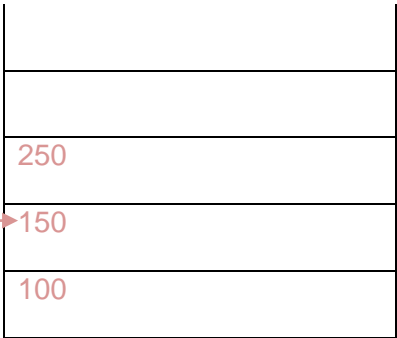
Call Function C  
Push return address onto stack

Stack pointer → 250



Return from C  
Pop return address off stack

Stack pointer → 150



Return from B

Pop return address off stack

Stack pointer

Return from A

Pop return address off stack

Stack pointer NULL

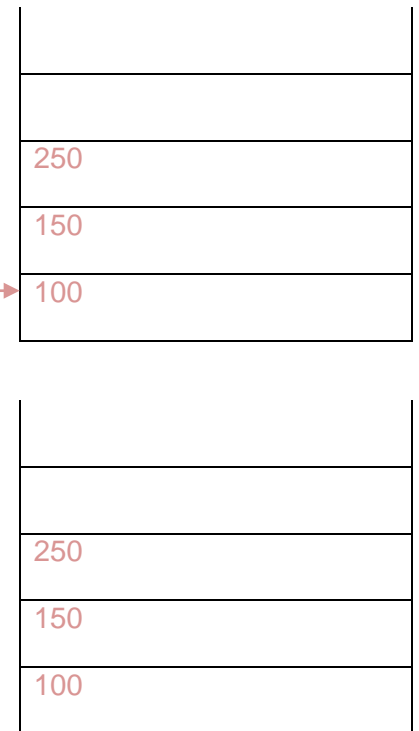


Fig. 3.7.7

Notice that when the values are read (popped off the stack) they remain in the stack, it is just that they can never be used again because the only way they can be accessed is via the stack pointer and the only way of getting the stack pointer back up there is to put new things in which will overwrite those values in there, so they might as well be thought of as being deleted after they have been read.

Now suppose that values need to be passed to, or from, a function or procedure, as we saw in section 3.7.b. Again a stack can be used. Suppose we have a main program and two procedures, Proc A(A1, A2) and Proc B(B1, B2, B3). That is, A1 and A2 are the formal parameters for Proc A and B1, B2 and B3 are the formal parameters for Proc B. Now look at Fig. 3.7.8 which shows the procedures being called and the return addresses that must be placed on the stack.

## Main program

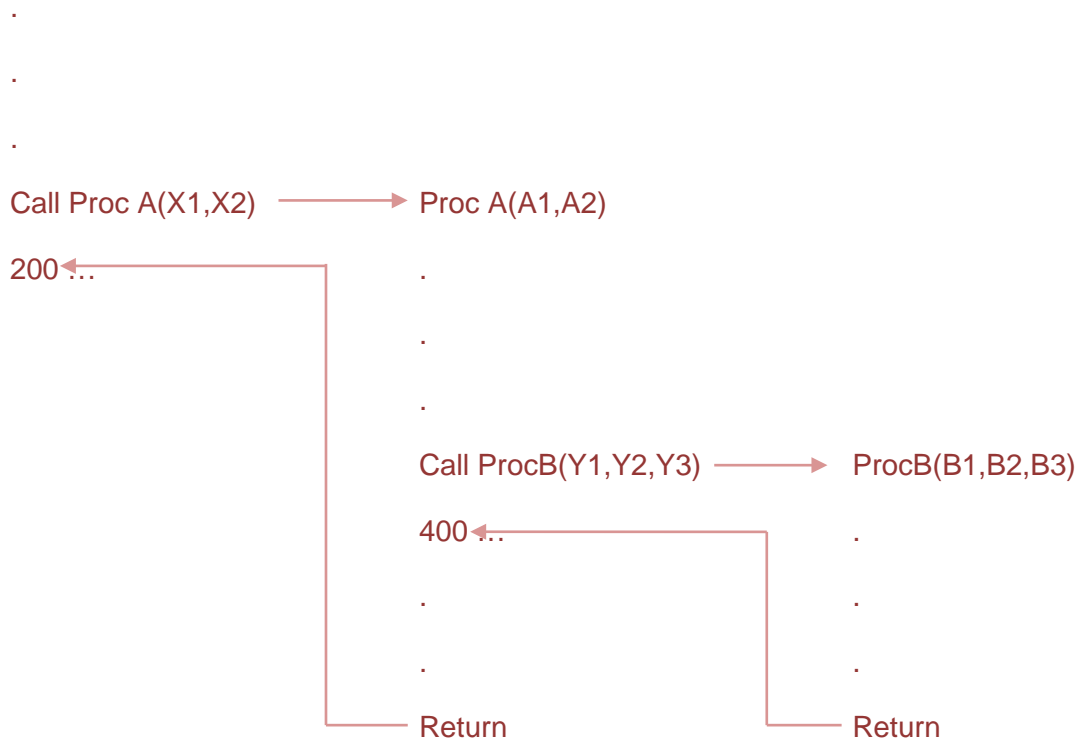


Fig. 3.7.8

Now let us suppose that all the parameters are being passed by value. Then, when the procedures are called the actual parameters must be placed on the stack and the procedures must pop the values off the stack and store the values in the formal parameters. This is shown in Fig.3.7.9; note how the stack pointer is moved each time an address or actual parameter is popped onto or popped off the stack.

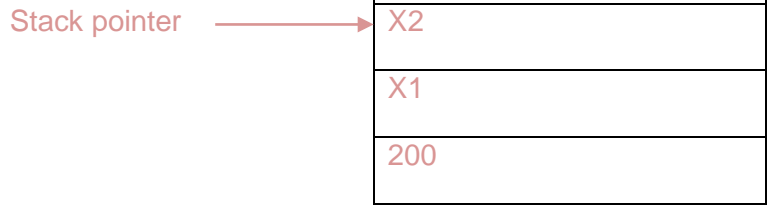
Call Proc A(X1, X2)

PUSH 200

PUSH X1

PUSH X2

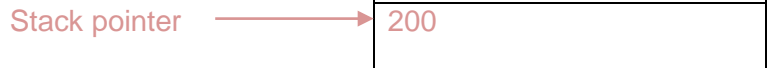
Stack pointer



A2 = X2 (POP X2)

A1 = X1 (POP X1)

Stack pointer



Call Proc B(Y1,Y2,Y3)

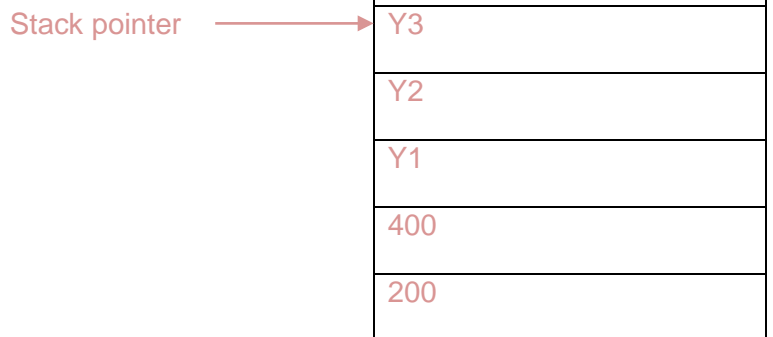
PUSH 400

PUSH Y1

PUSH Y2

PUSH Y3

Stack pointer



B3 = Y3 (POP Y3)

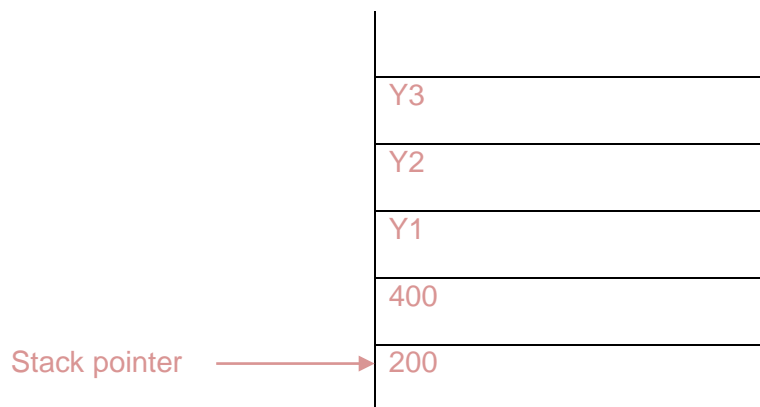
B2 = Y2 (POP Y2)

B1 = Y1 (POP Y1)

Stack pointer



Return from Proc B



Return from Proc A

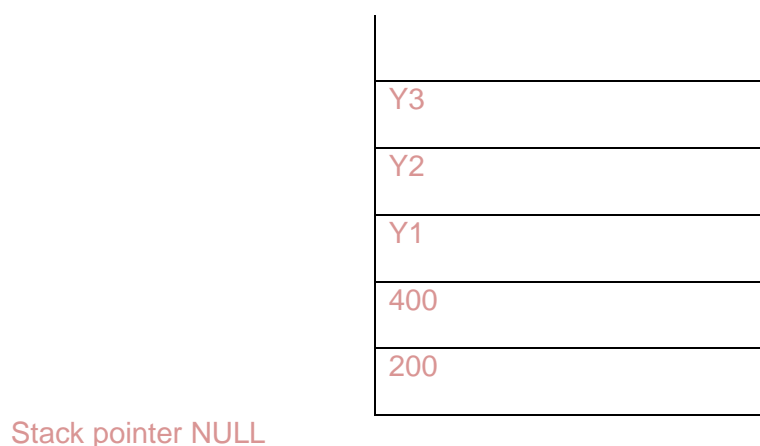


Fig 3.7.9

Notice how the values X1 and X2 are deleted, not when they are read (popped) but when something else is pushed on top off them.

Next we must consider what happens if the values are passed by reference. This works in exactly the same way as the addresses of variables are passed so there is no need to return the values via parameters. The procedures, or functions, will access the actual addresses where the variables are stored. Finally, how do functions return values? Simply push them on the stack immediately before returning. The calling program can then pop the value off the stack. Note that the return address has to be popped off the stack before pushing the return value onto the stack.

The golden rules to think about when considering how stacks are used are:

1. Where is the stack pointer? because that tells you what is next to be read and also tells you that anything above it is, to all intents and purposes, deleted.
2. Make sure that the order that things are put on is correct because you cannot change the order to read them off.

One other thing about this: Do not worry if the diagram looks slightly different sometimes. Some books will print similar diagrams to ours but will put the stack pointer pointing to the next space. This means that before reading anything the pointer needs to be brought down by one, but that anything added is added where the pointer is pointing. In our way of doing things the pointer is pointing to the top value still in the stack so if we want to add anything we have to move the pointer up one first.

### 3.7.d *Explain the need for, and be able to create and apply, BNF (Backus-Naur form) and syntax diagrams*

Since all programming languages have to be translated to machine code by means of a computer, they must be clearly defined. Each statement must be of a prescribed form. An example of the start of a FOR loop in Visual Basic is:

```
For count = 1 To 10
```

but other languages have to have the command written differently otherwise they can't understand it. For example, C++ expects:

```
for (count = 1, count <= 10, count++)
```

A Visual Basic compiler would not understand the C++ syntax and vice versa. We therefore need, for each language, a set of rules that specify precisely every part of the language. These rules are specified using Backus Naur Form (BNF) or syntax diagrams.

All languages use integers, so we shall start with the definition of an integer. An integer is a sequence of the digits 0, 1, 2, ... , 9. Now the number of digits in an integer is arbitrary. That is, it can be any number. A particular compiler will restrict the number of digits only because of the storage space set aside for an integer. But a computer language does not restrict the number of digits. Thus the following are all valid integers:

```
0
2
415
3040513002976
```

Thus, an integer can be a single digit. We can write this as:

```
<integer> ::= <digit>
```

This is read as 'an integer is defined to be (::=) a digit'.

But we must now define a digit. A digit is 0 or 1 or 2 or ... or 9 and we write this as:

```
<digit> ::= 0|1|2|3|4|5|6|7|8|9
```

where the vertical line is read as OR. Notice that all the digits have to be specified and that they are not inside angle brackets (< and >) like <integer> and <digit>. This is because integer and digit have definitions elsewhere; the digits 0, 1, 2, ... , 9 do not. There must be somewhere that we can start. This tends to be with the letters of the alphabet, the arithmetic signs, or with, as here, the digits. Anything else that we want has to be defined, and anything that needs defining is put in <> brackets.

Our full definition of a single digit integer is:

```
<integer> ::= <digit>
<digit> ::= 0|1|2|3|4|5|6|7|8|9
```

The language that this is written in is called Backus Naur Form (BNF).

But how are we going to specify integers of any length? Consider the integer:

147

This is a single digit integer ( 1 ) followed by the integer 47. But 47 is a single digit integer ( 4 ) followed by a single digit integer ( 7 ). Thus, all integers of more than one digit start with a single digit and are followed by an integer. Eventually the final integer is a single digit integer. Thus, an indefinitely long integer is defined as:

```
<integer> ::= <digit><integer>
```

(Note that  $\langle \text{integer} \rangle ::= \langle \text{integer} \rangle \langle \text{digit} \rangle$  is just as good)

This is a recursive definition as integer is defined in terms of itself. Applying this definition several times produces the sequence:

$\langle \text{integer} \rangle ::= \langle \text{digit} \rangle \langle \text{integer} \rangle$   
 $= \langle \text{digit} \rangle \langle \text{digit} \rangle \langle \text{integer} \rangle$   
 $= \langle \text{digit} \rangle \langle \text{digit} \rangle \langle \text{digit} \rangle \langle \text{integer} \rangle$

To stop this we use the fact that, eventually, `<integer>` is a single digit and write:

```
<integer> ::= <digit>|<digit><integer>
```

That is, `<integer>` is a `<digit>` OR a `<digit>` followed by an `<integer>`. This means that at any time `<integer>` can be replaced by `<digit>` and the recursion stops. Strictly speaking we have defined an unsigned integer as we have not allowed a leading plus sign ( + ) or minus sign ( - ). This will be dealt with later. We now have the full definition of an unsigned integer which, in BNF, is:

```
<unsigned integer> ::= <digit>|<digit><unsigned integer>
<digit> ::= 0|1|2|3|4|5|6|7|8|9
```

This definition of an unsigned integer can also be described by means of a syntax diagram. A syntax diagram should be read from the left hand side and then just follow the arrows. All acceptable definitions should be created by paths through the diagram from the left to the right. Any option that cannot be made by following the arrows is not allowed. S syntax diagram for an unsigned integer is shown in Fig 3.7.10.



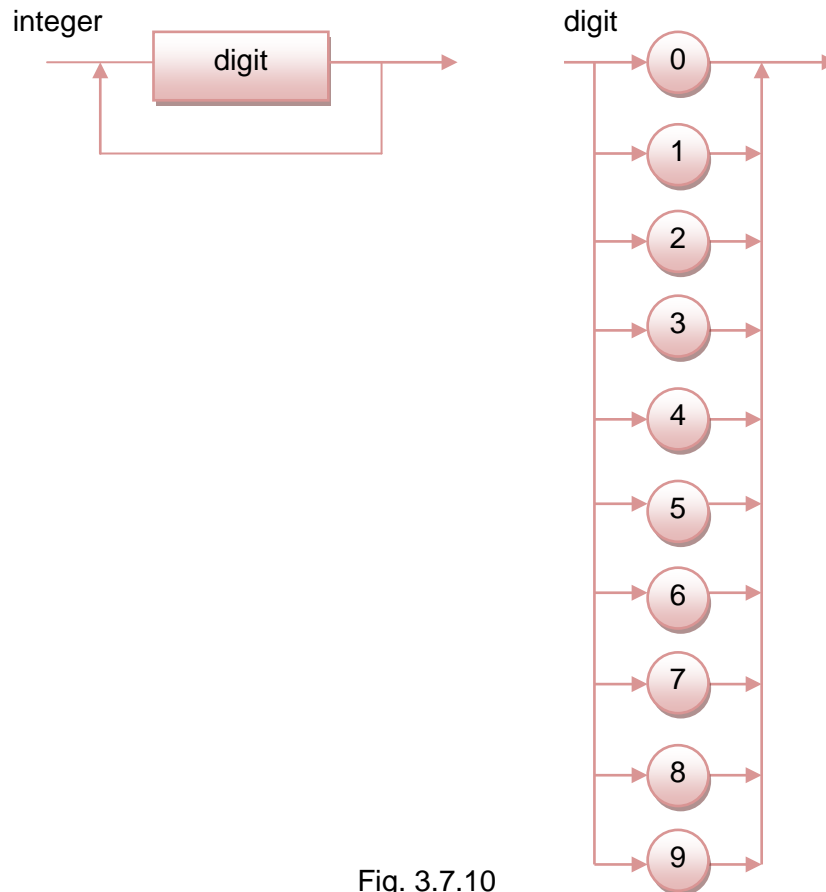


Fig. 3.7.10

Now we shall define a signed integer such as:

```
+27
-3415
```

This is simply an unsigned integer preceded by a + or – sign. Thus:

```
<signed integer> ::= + <unsigned integer> | - <unsigned integer>
```

and we can use the earlier definition of an <unsigned integer>. It is usual to say that an integer is an unsigned integer or a signed integer. If we do this we get the following definition, in BNF, of an integer:

```
<integer> ::= <unsigned integer> | <signed integer>
<signed integer> ::= + <unsigned integer> | - <unsigned integer>
<unsigned integer> ::= <digit> <digit> <unsigned integer>
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

There are other valid ways of writing these definitions. Try to rewrite the definition of an integer but using the extra line:

```
<sign> ::= + | -
```

It is better to use several definitions than try to put all the possibilities into a single definition. In other words, try to start at the top with a general definition and then try to break the definitions down into simpler and simpler ones. That is, we have used top-down design when creating these definitions. We have broken the definitions down until we have terms whose values can be easily determined.

Fig. 3.7.11 shows the corresponding syntax diagrams.

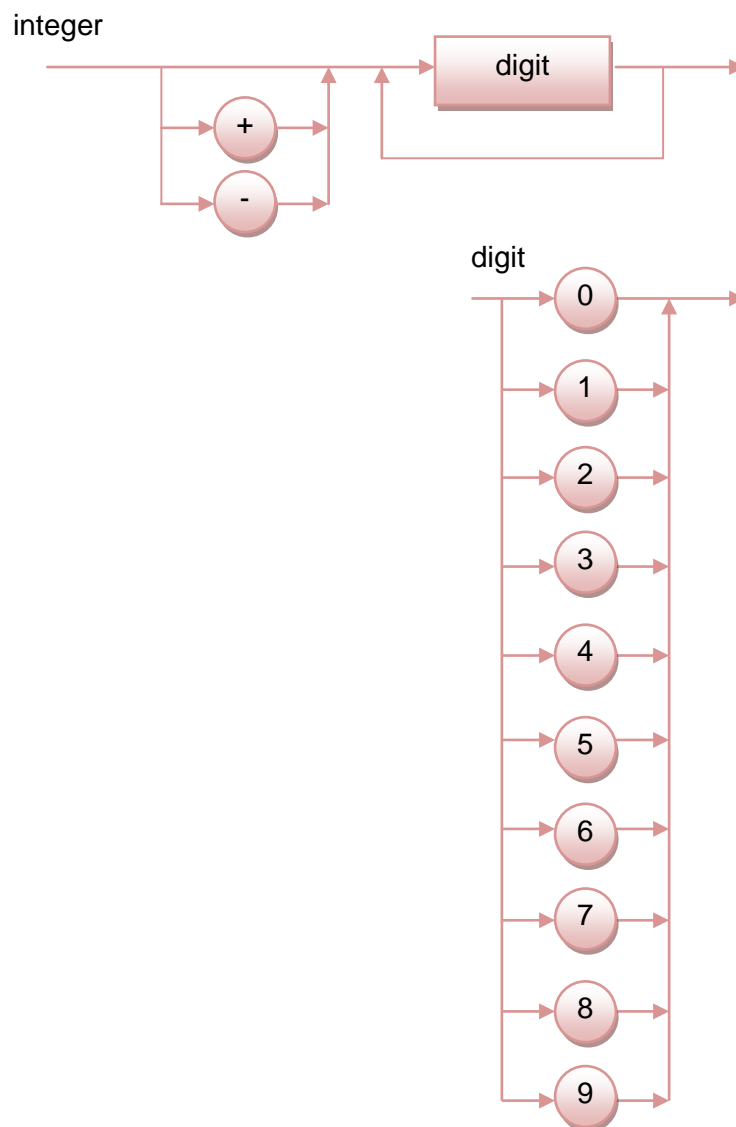


Fig. 3.7.11

Care must be taken when positioning the recursion in the definitions using BNF. Suppose we define a variable as a sequence of one or more characters starting with a letter. The characters can be any letter, digit or the underscore. Valid examples are:

```
A
x
sum
total24
mass_of_product
MyAge
```

Let us see what happens if we use a similar definition to that for an unsigned integer.

```
<variable> ::= <letter>|<character><variable>
<character> ::= <letter>|<digit>|<under-score>
```

Now 2Sum is valid as we use:

```
<character><variable>
```

with `<character> = 2` and `<variable> = Sum`. Continuing in this way we use 2, S and u for `<character>` and then m for `<letter>`. This means that our definition simply means that we must end with a letter not start with one. We must rewrite our definition in such a way as to ensure that the first character is a letter. Moving the recursive call to the front of `<character>` can do this. This means that the last time it is called it will be a letter and this will be at the head of the variable. The correct definition is:

```
<variable> ::= <letter>|<variable><character>
<character> ::= <letter>|<digit>|<under-score>
<letter> ::= <uppercase>|<lowercase>
<uppercase> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z
<lowercase> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
<digit> ::= 0|1|2|3|4|5|6|7|8|9
<under-score> ::= _
```

A syntax diagram can also represent this. This is left as an exercise.

Let us now use our definition of an integer to define a real number such as:

```
0.347
-2.862
+14.34
00235.006
```

The result is very simple, it is

```
<real number> ::= <integer> . <unsigned integer>
```

Finally, suppose we do not want to allow leading zeros in our integers. That is:

```
00135 is not allowed
0 is allowed.
```

This means that an integer can be a:

```
zero digit
non-zero digit
non-zero digit followed by any digit.
```

This means that an integer is:

```
zero or digits
```

where digits must start with a non-zero digit. In BNF, this is:

```
<integer> ::= <zero>|<digits>
```

<digits> must be a single non-zero digit or a non-zero digit followed by any digits. This gives us:

```
<digits> ::= <non-zero digit>|<digits><digit>
```

Where:

```
<zero> ::= 0
```

```
<non-zero digit> ::= 1|2|3|4|5|6|7|8|9
```

```
<digit> ::= <zero>|<non-zero digit>
```

Fig. 3.7.12 shows the corresponding syntax diagram.

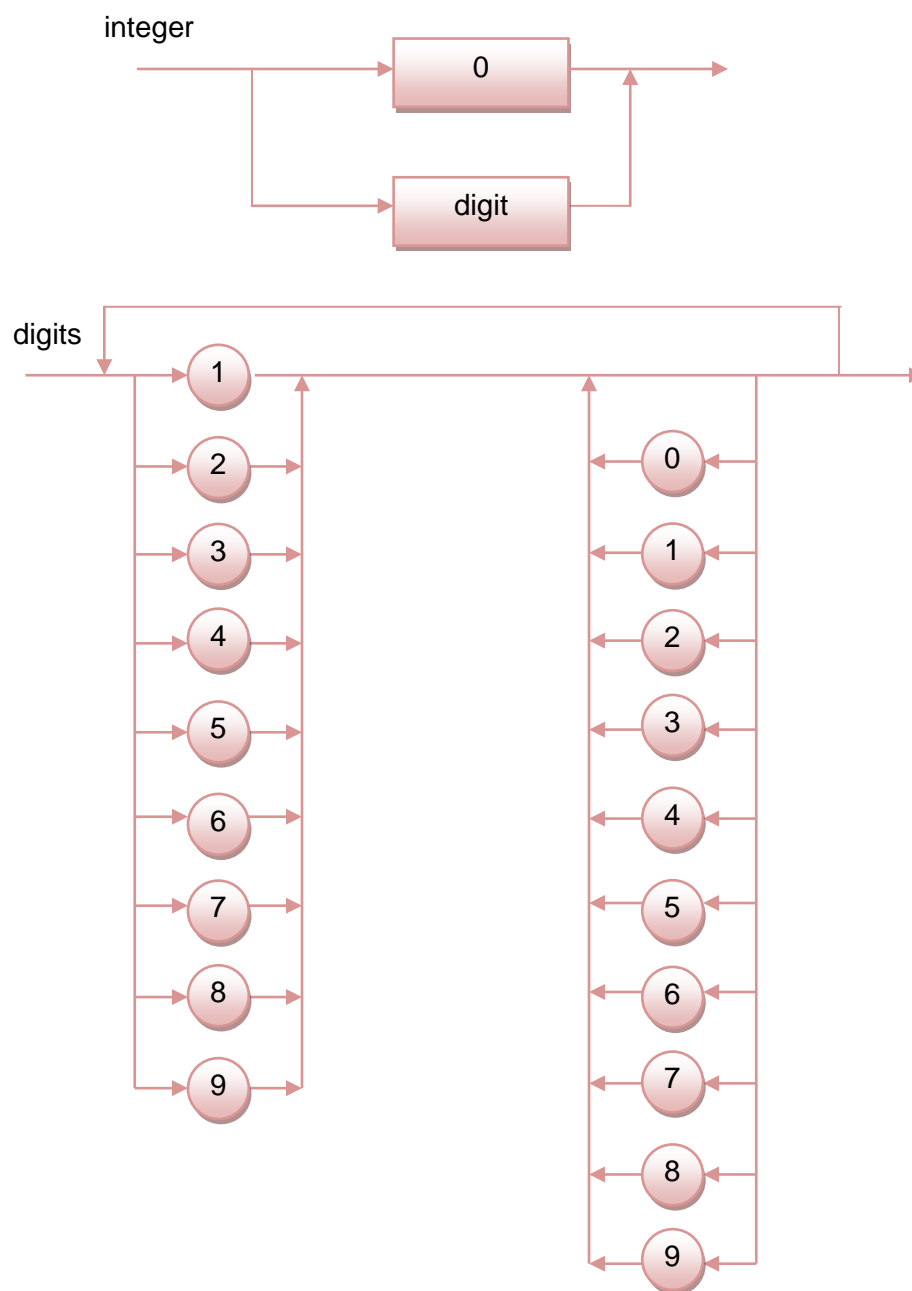


Fig. 3.7.12

### 3.7.e Explain the need for reverse Polish notation

First, what is 'reverse Polish notation'?

If we want to add two numbers, A and B together we would write the expression needed to do the addition as

$A + B$ . This is called infix notation because the operator (+) comes between the two operands (A, B). The problem with infix notation is that if we have an expression like  $2 * A + B$  we need to have brackets in order to decide whether to add the A and B first or whether to do the multiplication first. In other words infix notation without brackets can be ambiguous about the order in which things have to be done.

A Polish mathematician (hence the 'Polish' bit) came up with an alternative to infix notation by putting the operator at the front of the expression so that

$A + B$  became  $+ A B$ . This is not as silly as it looks at first. Instead of saying 'Number A is added to number B', this Polish notation simply says 'Add number A to number B'. The important thing about Polish notation is that it is unambiguous, the operator always applies to the two things (more about these in section 3.7.6) that come after it. Because there is no ambiguity, brackets are not needed which makes understanding and working out Polish expressions a lot easier than infix ones. Polish expressions are called 'prefix' expressions rather than infix, because the operator comes before the operands.

Reverse Polish is simply Polish, but the other way around, so the operator is put at the end.  $A + B$  becomes  $A B +$ . This can be interpreted as 'Take two numbers and add them together.' Reverse Polish is important in computing because not only is it unambiguous and so does not need brackets, but it can be worked out using a technique with stacks which makes the algorithms to work out expressions very simple (see section 3.7.6) Because the operator comes at the end, this is called postfix notation.

The expression  $2 * (A + B)$  means 'add the numbers A and B together and then multiply the answer by 2.'

In reverse Polish

The first thing that needs to be done is the bit in the bracket, so that becomes 'take the two numbers A and B and add them together' or  $A B +$ . After that multiply the answer and 2 together. So the whole thing becomes

$(A B +) 2 *$ . But things in reverse Polish are unambiguous so we don't need the brackets (they are only there to make it more understandable to us) so

$2 * (A + B)$  in infix becomes  $A B + 2 *$  in reverse Polish.

Reverse Polish is very important to computers because it makes the algorithms for solving expressions much easier to produce and to follow. The first calculator to do anything more than the four rules of arithmetic was called the 'Sinclair Scientific'. It was a very powerful calculator which was produced in the mid 1970's. It could do virtually all the things which we would expect a calculator to do now except it could not work out the reverse Polish form of the expressions to be calculated so the operator had to do it before feeding the values in. So if you wanted to add the numbers A and B you could not key in  $A + B$  you had to key in: A, IN, B, IN, +, =.

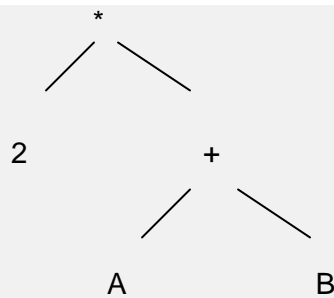
#### Summary

Reverse Polish is important in computing because expressions written in it are unambiguous, do not need brackets and they can be solved using stacks.

### 3.7.f Convert between reverse Polish notation and infix form of algebraic expressions using trees and stacks

All expressions in infix form consist of an operator and two sub-expressions or variables, one on either side. This sounds very like 'a node and its two branches', in other words a tree structure.

Take the example  $2 * (A + B)$ . What does this mean? It is 2 \* the answer to (two numbers added together). So we need to start with the \* in the tree and then one of the things (branches) is going to be its own little tree:



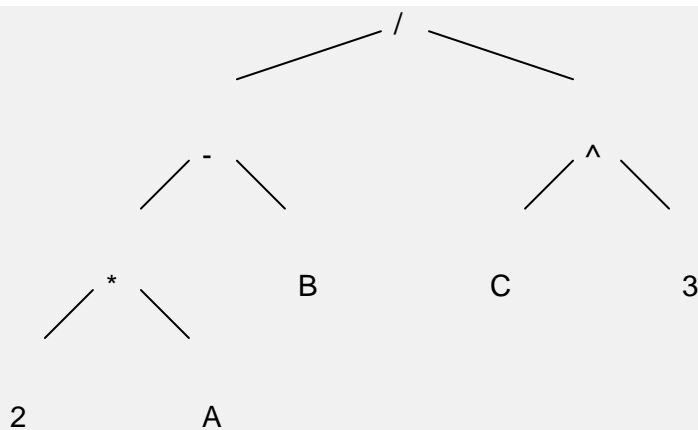
Another example:

$(2A - B)/(C ^ 3)$

This means 'Divide the answer to 'take B away from 2 times A' by the answer to C cubed'. So which operator do we start with in the tree? Divide was the first one mentioned when we managed to write it out. Another way of deciding is to underline the options and you then get something that looks a bit like a tree but upside down, e.g:

$(2 * A - B) / (C ^ 3)$   
\_\_\_\_\_

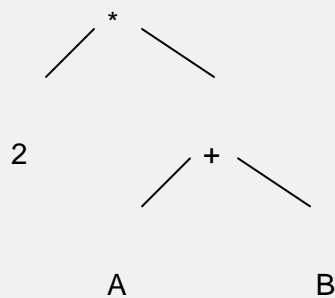
This gives the following tree:



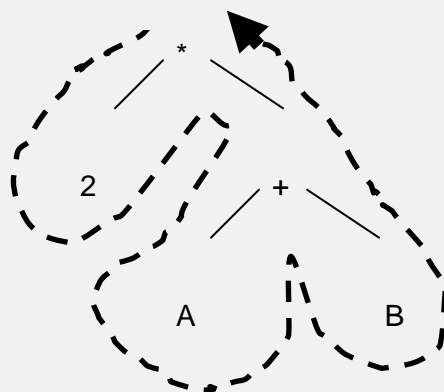
#### Why bother?

The reason why trees are important for this representation is that if the tree is read in a particular way then the result is the reverse Polish form of the expression.

Take the first tree that we had:



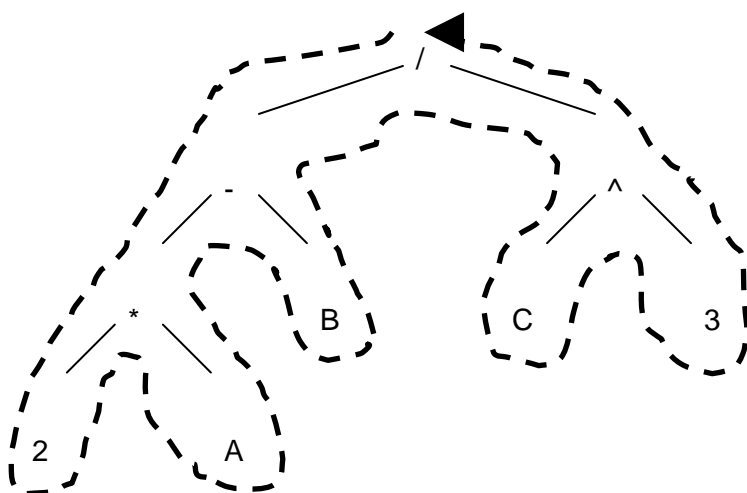
This can be read by following the dotted line:



The rule is to read the values as you pass them the first time. So, in this case we get:

$* 2 + A B$

The more complicated tree gives:



Which gives:

$/ - * 2 A B ^ C 3$

Another example:

If we start with :

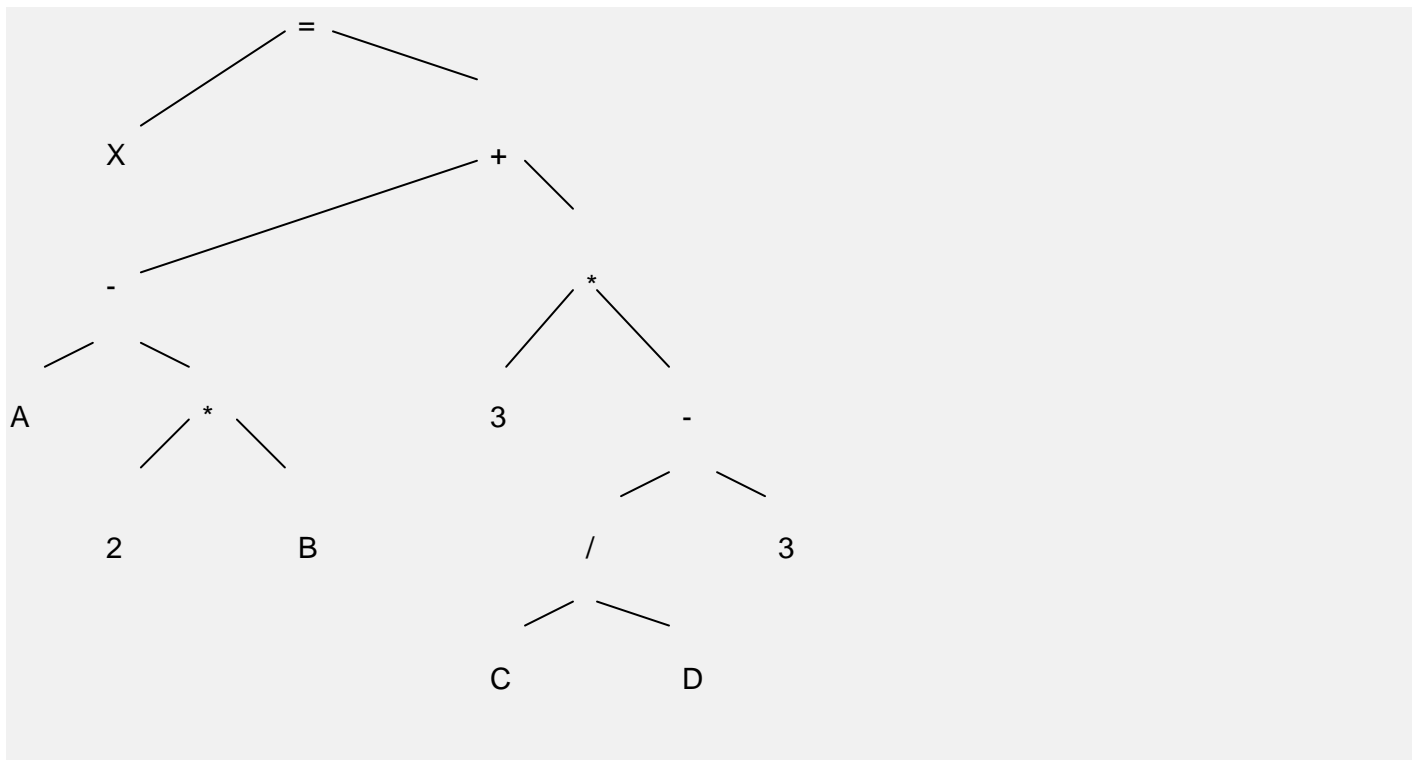
$$X = (A - 2B) + 3(C / D - 3)$$

an unlikely formula, perhaps, but it is complex enough for practice!

First stage is to write it properly, to ensure that everything is there, the most common is to miss out the \*:

$$X = (A - 2 * B) + 3 * (C / D - 3)$$

Note: This one has an = in it so this must be included. Assigning the answer to X is the last thing to be done, so the = must be the root of the tree. The only things outside brackets are the + and the \*. We can't add the two bits together until the \* has been done so + must be the next thing in the tree. Then we unravel the two brackets, one on either side of the + sign:



This may look complicated, in fact it is complicated! but only because of the volume needed, not because of the general principles, which are just the same as with the first one that we did.

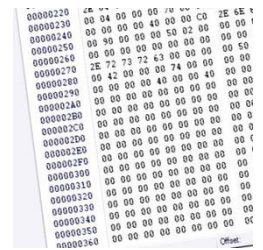
To read off the reverse Polish is now just the same as before and we get:

$$= X + - A * *$$



## 3.8 Low Level Languages

You may notice how this section is in a different order to that of the syllabus headings. (We are starting here with section c). We recommend that this is the order in which you should teach/ learn this section of the book.



### 3.8.c *Discuss the concepts and, using examples, show an understanding of mnemonics, opcode, operand and symbolic addressing in assembly language to include simple arithmetic operations, data transfer and flow control*

This section is deliberately out of order because an understanding of it will help when considering sections a and b. It is based very largely on the work done in 3.2.b and 3.2.c. Much of that work is reproduced here with additional comment.

In section 3.2.a it was said that computers use their own form of language. This form of language is based on binary numbers and uses combinations of binary digits to stand for different things.

These codes are specific to a particular machine or type of machine and the set of instructions that can be created are therefore referred to as the machine code.

We will use the same imaginary computer as we used in 3.2:

Add two numbers together is represented by	001
Subtract one number from another	010
Input a number from the outside	011
Output a value	100
Store a number in a memory location	101
Get a value from a memory location	110

#### Opcodes

These are the codes which represent the operations that the computer can understand and carry out. Because of this it is hardly surprising that they are called opcodes.

#### Mnemonics

In section 3.2 we said that the use of binary that we have in the opcodes is unsatisfactory for human beings because we can't understand binary and find it very difficult to deal with. For this reason each of the binary opcodes is given a pseudo name to make life easier for us. These pseudo names are called mnemonics. The ones we came up with for the opcodes above were:

ADD  
SUB  
IN  
OUT  
STO  
GET

## Operand

There is no point in telling the computer to ADD if you don't tell it what to add. Along with the opcode, the operand makes up the instruction which is placed in the current instruction register for decoding.

Sometimes the operand field holds the number to be added, sometimes it holds the address of where the number is that has to be added, sometimes it holds other information which we will see in 3.8.b. Whatever it is, this makes up the other part of the instruction.

## Address labels

In the same way that human beings can't handle the binary opcodes, they find the idea of binary operands difficult. Because of this, labels are used to stand for the operands that are to be used.

Put all this together and the instruction 00100010 means add whatever is in address 2, and can be written as something like ADD X.

The meaning is the same, it just looks different because we have used symbols instead of the binary, hence the expression 'symbolic addressing'.

There are different types of operation which can be done and they have different effects on the processor. Basically they fall into one of three categories. Different processors may have different commands, remember that these languages are machine specific, but they will all support these basic types: Arithmetic and logic operations, memory manipulation or 'data transfer' and jump instructions or 'flow control'.

## Arithmetic operations

When the opcode is decoded, the data is collected and placed in the memory data register. It is then manipulated in the ALU, part of which is the accumulator where the result will be temporarily stored until it is decided what to do with it. You may hear about other parts of the ALU like the arithmetic register(s) and we have mentioned array processors before in this text, but the important thing is the accumulator as far as this section is concerned.

## Data transfer

Operations like GET and STO are operations that simply move things out of and into memory. (Often the mnemonic LDA is used to get things from memory, it stands for 'Load the accumulator with...')

## Flow control

We don't have a command for flow control yet. But imagine we had a command that was 00000100, which could be written symbolically as JMP X.

This means Jump out of the sequence of instructions being carried out and go to the instruction being stored in location X instead.

This would be called an unconditional jump because the system is being forced to make the jump. Jumps like this should not be used in programming but it serves a purpose here because it is nice and simple.

The effect of a jump instruction is to change the flow. The processor was going along quite happily taking one instruction at a time, in sequence and then this instruction comes along and says 'change the order'. It is done by using the special register which controls which instruction is to be done next: The Program Counter (PC). If the instruction is JMP X, the simplest way of getting the processor to do the instruction in X next is to put the X in the PC. Simple as that.

Of course it is not that simple but the basic principle is there.

### 3.8.a Explain the concepts and, using examples, demonstrate an understanding of the use of the accumulator, registers, and program counter

Fig. 3.8.1 shows the minimum number of registers needed to execute instructions. Remember that these are used to execute machine code instructions not high-level language instructions.

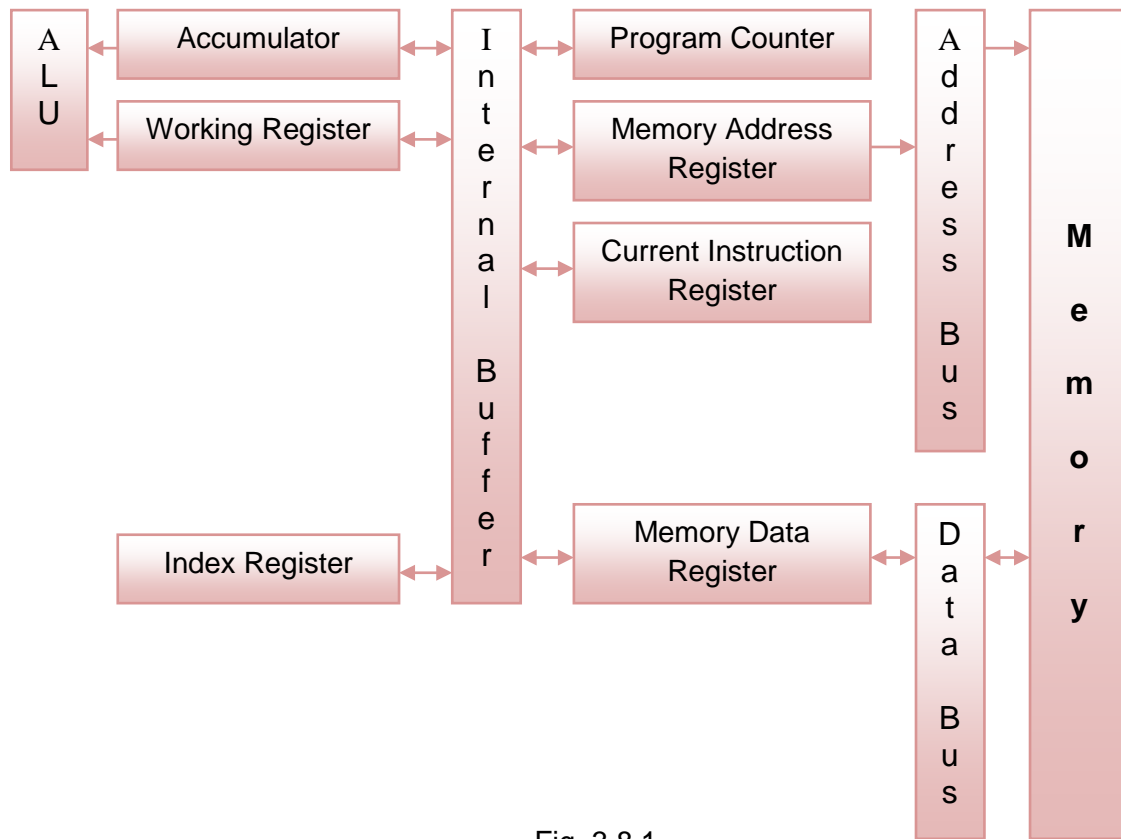


Fig. 3.8.1

Much of what follows has been seen before in this text and the purpose of this section is largely revision, but is also intended to bring things together. You will have met most of the ideas in sections 1.4.b, 1.4.c, 3.3.b.

The *program counter* (PC) is used to keep track of the location of the next instruction to be executed. (This register is also known as the Sequence Control Register (SCR)). It is used so that the processor always knows where the next instruction is. Note that the content can be changed by some instructions as well as simple incrementation.

The *memory address register* (MAR) holds the address of the instruction or data that is to be fetched from memory. This address can come directly from the PC if the data to be fetched is the next instruction, or it can come from the CIR when the address of the data to be used is found in the decoding process.

The *current instruction register* (CIR) holds the instruction that is to be executed, ready for decoding. This instruction will consist of (at least) an operation code which will be looked up in the table of codes so that the processor knows what actions are necessary and the address which will be sent to the MAR.

The *memory data register* (MDR) holds data to be transferred to memory and data that is being transferred from memory, including instructions on their way to the CIR. Remember that the computer cannot distinguish between data and instructions. Both are held as binary numbers. How these binary numbers are interpreted depends on the registers in which they end up. The MDR is the only route between the other registers and the main memory of the computer.

The *accumulator* is where results are temporarily held and is used in conjunction with a *working register* (ALU) to do calculations.

The *index register* is a special register used to adjust the address part of an instruction. This will be explained in more detail later. If this is to be used then the simple case of two parts to the instruction which was detailed above, must become three parts because there must be a part of the instruction which defines which sort of addressing must be used.

*Note that the diagram does not show the control bus and the signals needed for instructions to be correctly executed. These are not required for this module but an understanding of their importance is expected in module 1.*

## Some extra concepts will now be introduced which provide extra work beyond the narrow confines of the specification

Consider the instruction which will be held in the CIR and used to tell the processor what to do. At the moment we have two fairly obvious parts of it, the operation code itself and the address of the data to be used. When the index register was introduced it became apparent that different types of addresses were available which have to be handled differently. Some of the different methods are explained in the next section. See if you can work out what the instruction might now look like.

In reality the instruction is even more different than this. See what you can find out about 'two address format'. Why might a three address format sometimes be sensible? What other differences can you find to the standard instruction format?

Try to understand how the processor interprets the instructions. Will errors in the program be found here? or should they have been found at an earlier stage?

We shall now see how these registers are used to execute instructions. In order to do this we shall assume that a memory location can hold both the instruction code and the address part of the instruction. For example, a 32-bit memory location may use 12 bits for the instruction code and 20 bits for the address part. This will allow us to use up to  $2^{12}$  (= 4096) instruction codes and  $2^{20}$  (= 1 048 576) memory addresses.

To simplify things further, we shall use mnemonics for instructions such as:

Code	Meaning
LDA	load the accumulator
STA	store the accumulator
ADD	add the contents of memory to the accumulator
STOP	Stop

We shall also use decimal numbers rather than binary for the address part of an instruction.

Suppose four instructions are stored in locations 300, 301, 302 and 303 as shown in the following table and that the PC contains the number 300.

Address	Contents	Notes
.	.	.
300	LDA 400	Load accumulator with contents of location 400
301	ADD 401	Add contents of location 401 to accumulator
302	STA 402	Store contents of accumulator in location 402
303	STOP	Stop
304		
.	.	.
400	5	Location 400 contains the number 5
401	7	Location 401 contains the number 7
402	?	Not known what is in location 402
.	.	.

The fetch part of the instruction is:

Action	PC	MAR	CIR	MDR
Copy contents of PC to MAR	300	300	?	?
Add 1 to PC	301	300	?	?
Copy contents of location pointed to by MAR into MDR	301	300	?	LDA 400
Copy instruction in MDR to CIR	301	300	LDA 400	LDA 400

The instruction is now decoded (not shown in the table) and is interpreted as 'load the contents of the location whose address is given into the accumulator'.

We now start the execution phase. As the contents of an address are needed, the address part of the instruction is copied into the MAR, in this case 400.

Action	PC	MAR	CIR	MDR
Copy address part of instruction to MAR	301	400	LDA 400	LDA 400

Now use the MAR to find the value required and copy it into the MDR.

Action	PC	MAR	CIR	MDR
Copy contents of address given in MAR to MDR	301	400	LDA 400	5

Finally copy the contents of the MDR to the accumulator.

Now use the same steps to fetch and execute the next instruction. Note that the PC already contains the address of the next instruction.

Note that all data moves between memory and the MDR via the data bus. All addresses use the address bus.

A summary of the steps needed to fetch and execute the LDA instruction are shown in Fig. 3.8.2

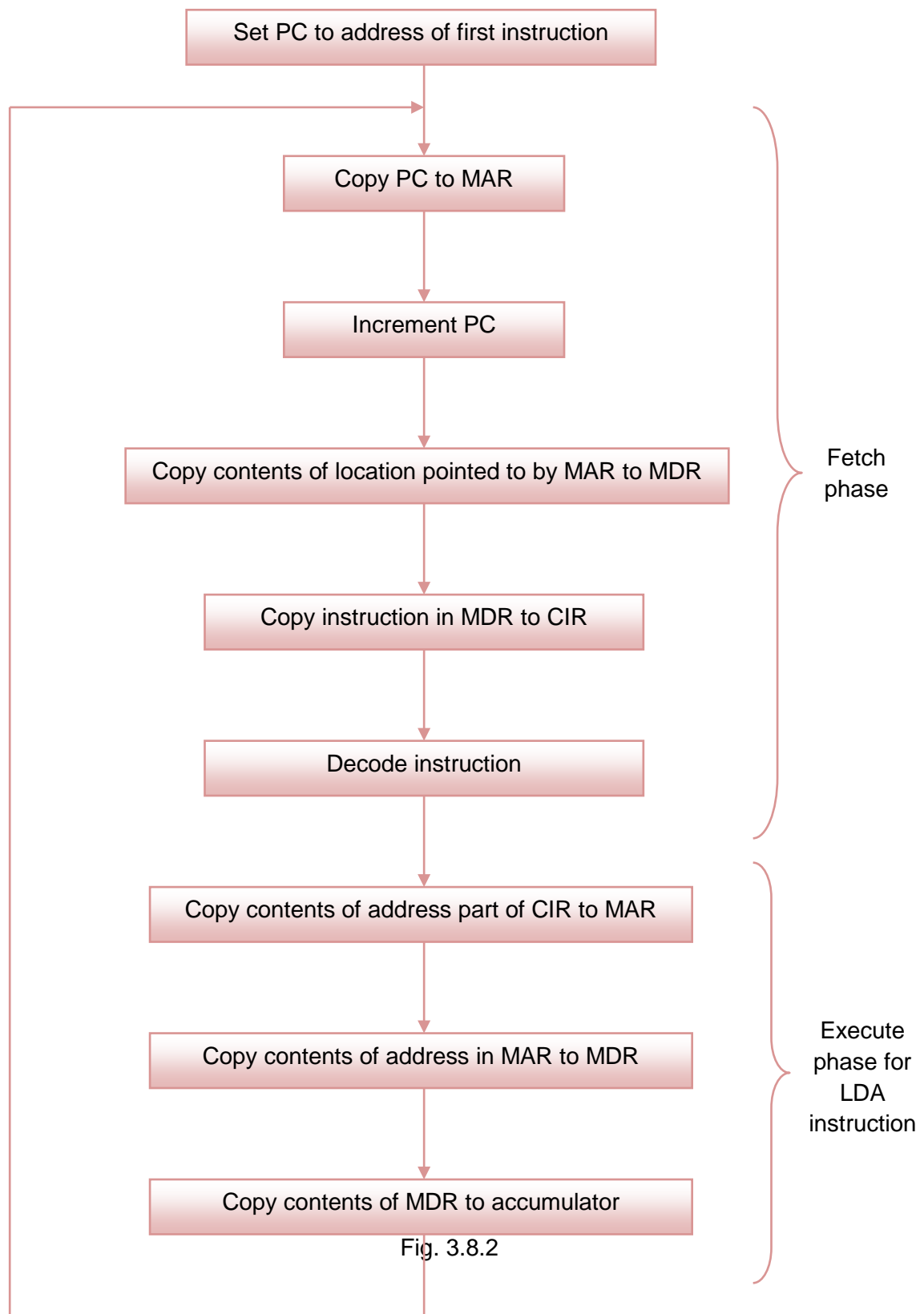


Fig. 3.8.2

In Fig. 3.8.3, what happens during the execute cycle depends on the instruction. For example, the STA n (store the contents of the accumulator in the location with address n) has the execute steps shown in Fig. 3.8.3.

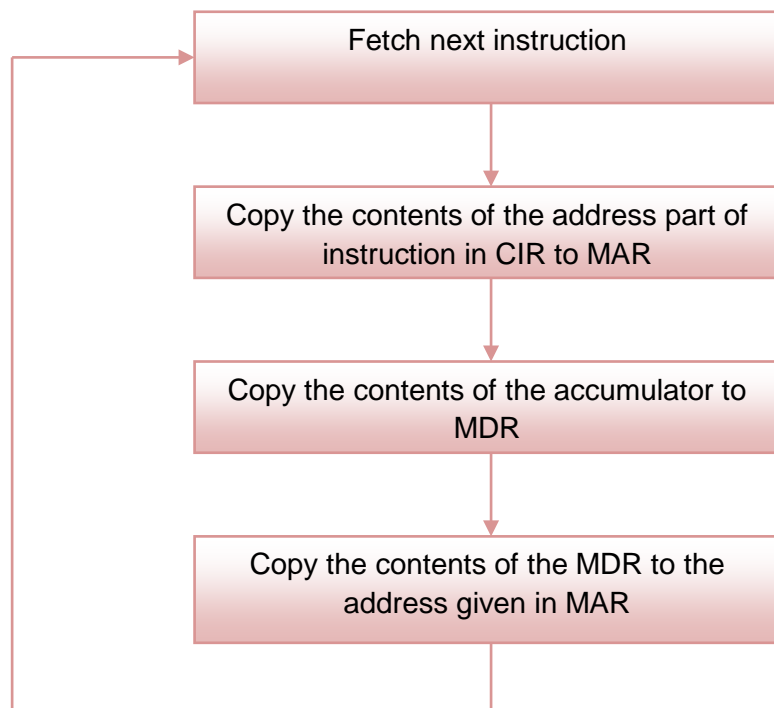


Fig. 3.8.3



This process works fine but only allows for the sequential execution of instructions. This is because the PC is only changed by successively adding 1 to it. How can we arrange to change the order in which instructions are fetched? Consider these instructions:

Address	Contents	Notes
.	.	
.	.	
.	.	
300	ADD 500	Add the contents of location 500 to the accumulator
301	JLZ 300	If accumulator < 0 go back to the instruction in location 300
.	.	
.	.	
.	.	

Suppose the PC contains the number 300, after the instruction ADD 500 has been fetched and executed the PC will hold the number 301. Now the instruction JLZ 300 will be fetched in the usual way and the PC will be incremented to 302. The next step is to execute this instruction. The steps are shown in Fig. 3.8.4.

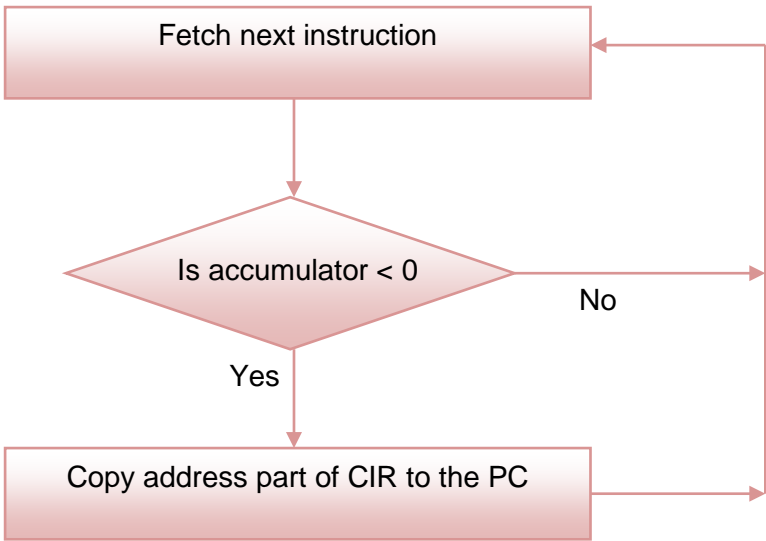


Fig 3.8.4

So far we have used two copy instructions (LDA and STA), one arithmetic instruction (ADD) and one jump instruction (JLZ). In the case of the copy and arithmetic instructions, the address part has specified where to find or put the data. This is known as *direct addressing*. We shall now go on to the next section where we shall meet some other types of addressing.

### 3.8.b *Describe immediate, direct, indirect, relative and indexed addressing of memory when referring to low level languages*

#### Immediate addressing

This is sometimes called an 'immediate operand'. It is when the value in the address part of the instruction is actually the value to be used. It means that the memory does not need to be searched for the value to be used.

In effect, the instruction ADD 10 actually does mean 'add 10', not 'add whatever is in location 10'. There are two problems here. The first is that because the value is part of the instruction it cannot be used easily by other instructions and secondly it is difficult to change the value.

A good exercise is to try to work out how the value can be used by other instructions. Write a piece of code with the instructions that we have mentioned so far so that another instructions can subtract the value.

#### Direct addressing

We have already used this. This is the simple case where the address in the memory is where the value actually is that should be used. It means that

ADD 10 simply means 'go and find whatever is in location 10 and add it to the accumulator'.

There are two problems with direct addressing:

1. It is only possible to access addresses which can be stored in the address part of the instruction. Imagine 32 bits being available for the instruction. We have already established that 12 bits is reasonable for the operation code and we have also said that some extra bits will be reserved for other things, so 16 bits is probably reasonable for the address part of the instruction. This means that 2 to the power of 16 or 65536 locations can be addressed. This will give us a 64Kbyte memory. There would not be any point in making it bigger because any more could not be accessed as it could not be addressed.
2. The second problem is that if we want to do the same instruction to a number of pieces of data we have to write the same instruction out a lot of times. This can be solved by using indexed addressing.

#### Indirect addressing

The first problem would not be so important if we could use the whole 32 bits to store the address we needed because the numbers we could store would be massive. Well, there are lots of 32 bit locations where we could store the real address, they are all in the memory. With indirect addressing the address in the instruction is the address where we will find the real address of the data to be used.

So, ADD 10 means 'go to 10 in the memory; in there you will find another address; go to this other address and read the data that you want to use.'

This is very useful because it means that the larger addresses in memory can be used to store data.

### Indexed addressing

Imagine that the contents of an array of 100 items have to be added together. If the first item in the array is in location 10 then the instructions would have to be:

ADD 10, ADD 11, ADD 12.....ADD 109.

Basically it is the same instruction 100 times. Far easier is to have a special register (called the Index Register (IR)) which is set to 0 so that the first value is taken from  $10+0$ . After this is done the IR is incremented and the same instruction is done again. This time the address used will be  $10+1$ . When the IR is incremented and used again the next address will be  $10+2$  and so on all the way up to  $10+99$ . This time we only needed 1 instruction and simply said that it should be carried out 100 times.

Many languages require arrays to be declared before programs are run. Try to formulate an argument why this should be necessary.

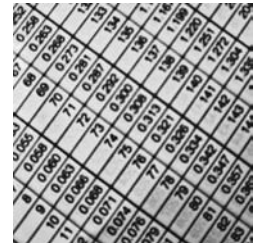
### Relative addressing

When a program segment is written or translated it is not always possible to predict where the program and its data are going to end up in the memory. So there is no point in having an instruction of ADD 10 because the data may not be in location 10. We can't insist on that (not normally anyway), but what we can insist on is that the program and data go into memory as a single block. There is no problem then if we use relative addressing which simply means that all the addresses are numbered from the start of the program. If the first instruction of the program was stored in location 100 it means that an instruction ADD 10 will have the data stored in  $100+10$  or location 110. Everything will be 100 more than it actually says. Everything is dependent on the address of the first instruction, or to say it another way, every address is relative to the first one.

Try to relate the idea of relative addressing to multi-tasking and multi-programming operating systems.

Most programs and data will be stored in many partitions or segments of memory because they will be too large to fit into one. Try to explain how the operating system can still keep track of addresses when the instructions and data are split across many partitions/segments of memory.

## 3.9 Databases



### 3.9.a Describe flat files and relational databases, explaining the difference between them

Originally all data were held in files. A typical file would consist of a large number of records each of which would consist of a number of fields. Each field would have its own data type and hold a single item of data.

Typically, a stock file would contain records describing stock. Each record may consist of the following fields:

Field Name	Data Type
Description	String
Cost Price	Currency
Selling Price	Currency
Number in Stock	Integer
Reorder Level	Integer
Supplier Name	String
Supplier Address	String

This led to very large files that were difficult to process. Suppose we want to know which items need to be reordered. This is fairly straightforward, as we only need to sequentially search the file and, if Number in Stock is less than the Reorder Level, make a note of the item and the supplier and output the details.

The problem is when we check the stock the next day, we will create a new order because the stock that has been ordered has not been delivered. To overcome this we could introduce a new field called On Order of type Boolean. This can be set to True when an order has been placed and reset to False when an order has been delivered. Unfortunately it is not that easy.

The original software is expecting the original seven fields not eight fields. This means that the software designed to manipulate the original file must be modified to read the new file layout.

Further ad hoc enquiries are virtually impossible. What happens if management ask for a list of best selling products? The file has not been set up for this and to change it so that such a request can be satisfied in the future involves modifying all existing software. Further, suppose we want to know which products are supplied by Food & Drink Ltd.. In some cases the company's name has been entered as Food & Drink Ltd., sometimes as Food and Drink Ltd. and sometimes the full stop after Ltd has been omitted. This means that a match is very difficult because the data is inconsistent. Another problem is that each time a new product is added to the database both the name and address of the supplier must be entered. This leads to redundant data or data duplication.

The following example, shown in Fig. 3.9.1, shows how data can be proliferated when each department keeps its own files:

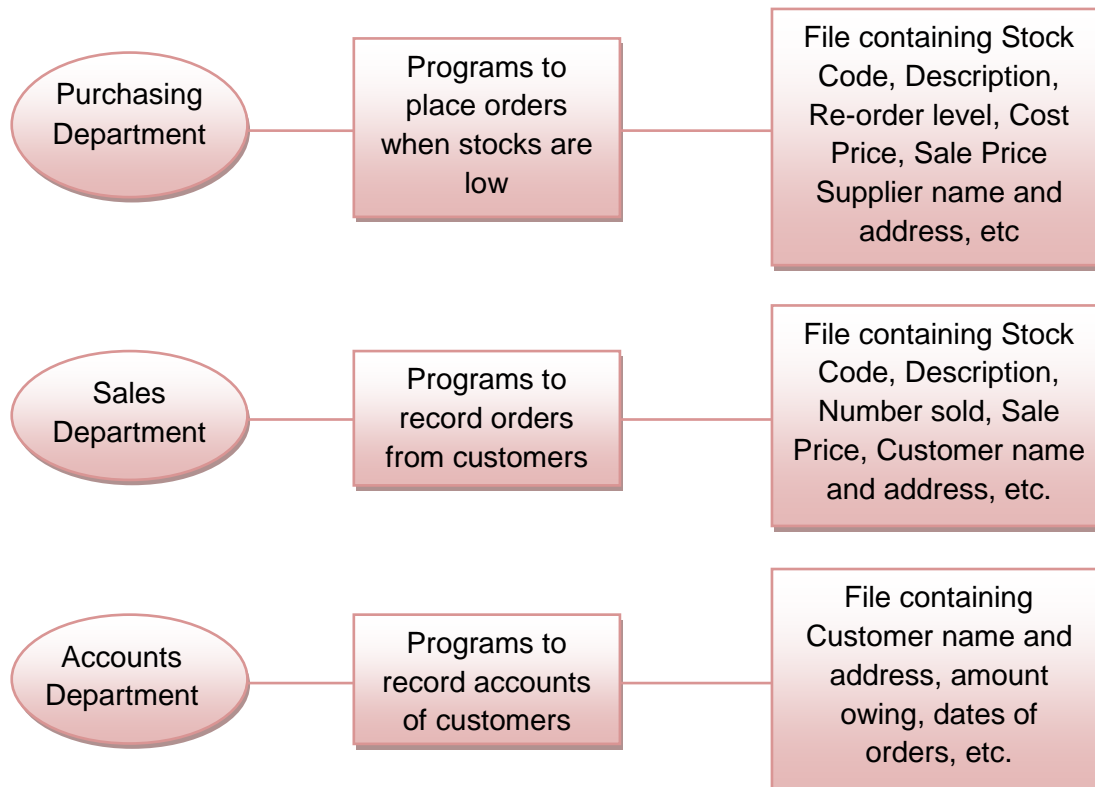


Fig. 3.9.1

This method of keeping data uses *flat files*. Flat files have the following limitations:

- Separation and isolation of data  
Suppose we wish to know which customers have bought parts produced by a particular supplier. We first need to find the parts supplied by a particular supplier from one file and then use a second file to find which customers have bought those parts. This difficulty can be compounded if data is needed from more than two files.
- Duplication of data  
Details of suppliers have to be duplicated if a supplier supplies more than one part. Details of customers are held in two different files.  
Duplication is wasteful as it costs time and money. Data has to be entered more than once, therefore it takes up time and more space.  
Duplication leads to loss of data integrity. What happens if a customer changes his address? The Sales Department may update their files but the Accounts Department may not do this at the same time. Worse still, suppose the Order Department order some parts and there is an increase in price. The Order Department increases the Cost and Sale prices but the Accounts Department do not, there is now a discrepancy.
- Data dependence  
Data formats are defined in the application programs. If there is a need to change any of these formats, whole programs have to be changed. Different applications may hold the data in different forms, again causing a problem. Suppose an extra field is needed in a file, again all applications using that file have to be changed, even if they do not use that new item of data.

- Incompatibility of files  
Suppose one department writes its applications in COBOL and another in C. Then COBOL files have a completely different structure to C files. C programs cannot read files created by a COBOL program.
- Fixed queries and the proliferation of application programs  
File processing was a huge advance on manual processing of queries. This led to end-users wanting more and more information. This means that each time a new query was asked for, a new program had to be written. Often, the data needed to answer the query were in more than one file, some of which were incompatible.

A solution to these problems was to devise relational databases where, instead of being placed in individual, unrelated files, the data is stored in tables which are related to each other. Each table has a key field, by which all the values in that table are identified. The records (or entities) in the tables can be related to entities in other tables by sharing keys as attributes within the entities. So, the problem of the supplier details being duplicated can be solved by the relevant attribute in the order table simply containing the key of the supplier entity. The system can then search the supplier table for details of the relevant supplier using the supplier key when it is necessary. In this way all but the keys need only be stored once in the database thus almost eradicating duplication. Also, if data is only stored once there can be no danger of lack of integrity of the data. The different needs of the departments will come down to the software that will be used to control the data as the data is stored somewhere in the system so all a department needs is the software that can search for it. In this way each department does not need its own set of data, simply its own view of the data that everyone is using.

### 3.9.b Design a simple relational database to the third normal form (3NF), using entity-relationship (E-R) diagrams and decomposition

Consider the following delivery note from Easy Fasteners Ltd:

Easy Fasteners Ltd	
Old Park, The Square, Berrington, Midshire BN2 5RG	
To: Bill Jones London England	No.: 005 Date:14/08/01
Product No.	Description
1	Table
2	Desk
3	Chair

Fig. 3.9.2

In this example, the delivery note has more than one product on it. This is called a repeating group. In the relational database model, each record must be of a fixed length and each field must contain only one item of data. Also, each record must be of a fixed length so a variable number of fields is not allowed. In this example, we cannot say 'let there be three fields for the products as some customers may order more products than this and others fewer products. So, repeating groups are not allowed.

At this stage we should start to use the correct vocabulary for relational databases. Instead of fields we call the columns *attributes* and the rows are called *tuples*. The files are called *tables*.

We write the details of our delivery note as:

DELNOTE(Num, CustName, City, Country, (ProdID, Description))

where DELNOTE is the name of the table and Num, CustName, City, Country, ProdID and Description are the attributes. ProdID and Description are put inside parentheses because they form a repeating group. In tabular form the data may be represented by Fig. 3.9.3:

Num	CustName	City	Country	ProdID	Description
005	Bill Jones	London	England	1	Table
				2	Desk
				3	Chair

Fig. 3.9.3

This again shows the repeating group. We say that this is in un-normalised form (UNF). To put it into 1<sup>st</sup> normal form (1NF) we complete the table and identify a key that will make each tuple unique. This is shown in Fig. 3.9.4:

Num	CustName	City	Country	ProdID	Description
005	Bill Jones	London	England	1	Table
005	Bill Jones	London	England	2	Desk
005	Bill Jones	London	England	3	Chair

Fig 3.9.4

To make each row unique we need to choose Num together with ProdID as the key. Remember, another delivery note may have the same products on it, so we need to use the combination of Num and ProdID to form the key. We can write this as:

DELNOTE(Num, CustName, City, Country, ProdID, Description)

To indicate the key, we simply underline the attributes that make up the key.

Because we have identified a key that uniquely identifies each tuple, we have removed the repeating group.

#### Definition of 1NF

A relation with repeating groups removed is said to be in First Normal Form (1NF). That is, a relation in which the intersection of each tuple and attribute (row and column) contains one and only one value.

Let us now see how to move from 1NF to 2NF and then on to 3NF.

#### Definition of 2NF

A relation that is in 1NF and every non-primary key attribute is fully dependent on the primary key is in Second Normal Form (2NF). That is, all the incomplete dependencies have been removed.

In our example, using the data supplied, CustName, City and Country depend only on Num and not on ProdID. Description only depends on ProdID, it does not depend on Num. We say that:

Num *determines* CustName, City, Country

ProdID *determines* Description

and write:

Num → CustName, City, Country

ProdID → Description

If we do this, we lose the connection that tells us which parts have been delivered to which customer. To maintain this connection we add the dependency:

Num, ProdID → 0 (Dummy functional dependency)

We now have three relations:

DELNOTE(Num, CustName, City, Country)

PRODUCT(ProdID, Description)

DEL\_PROD(Num, ProdID)



Note the keys (underlined) for each relation. DEL\_PROD needs a compound key because a delivery note may contain several parts and similar parts may be on several delivery notes. We now have the relations in 2NF.

Can you see any more data repetitions? The following table of data may help:

Num	CustName	City	Country	ProdID	Description
005	Bill Jones	London	England	1	Table
005	Bill Jones	London	England	2	Desk
005	Bill Jones	London	England	3	Chair
008	Mary Hill	Paris	France	2	Desk
008	Mary Hill	Paris	France	7	Cabinet
014	Anne Smith	New York	USA	5	Cabinet
002	Tom Allen	London	England	7	Cupboard
002	Tom Allen	London	England	1	Table
002	Tom Allen	London	England	2	Desk

Country depends on City not directly on Num. We need to move on to 3NF.

#### Definition of 3NF

A relation that is in 1NF and 2NF, and in which no non-primary key attribute is transitively dependent on the primary key is in 3NF. That is, all non-key elements are fully dependent on the primary key.

In our example we are saying:

$\text{Num} \rightarrow \text{CustName, City, Country}$

but it is City that determines Country, that is:

$\text{City} \rightarrow \text{Country}$

and we can write:

$\text{Num} \rightarrow \text{City} \rightarrow \text{Country}$

$\text{Num} \rightarrow \text{CustName}$

We say that Num transitively functionally determines Country because it is determined via another attribute (City).

Removing this transitive functional determinacy, we have:

DELNOTE(Num, CustName, City)

CITY\_COUNTRY(City, Country)

PRODUCT(ProdID, Description)

DEL\_PROD(Num, ProdID)

Let us now use the data above and see what happens to it as the relations are normalised:

1NF

DELNOTE

Num	CustName	City	Country	ProdID	Description
005	Bill Jones	London	England	1	Table
005	Bill Jones	London	England	2	Desk
005	Bill Jones	London	England	3	Chair
008	Mary Hill	Paris	France	2	Desk
008	Mary Hill	Paris	France	7	Cupboard
014	Anne Smith	New York	USA	5	Cabinet
002	Tom Allen	London	England	7	Cupboard
002	Tom Allen	London	England	1	Table
002	Tom Allen	London	England	2	Desk

Convert to  
2NF

DELNOTE

PRODUCT

Num	CustName	City	Country	ProdID	Description
005	Bill Jones	London	England	1	Table
008	Mary Hill	Paris	France	2	Desk
014	Anne Smith	New York	USA	3	Chair
002	Tom Allen	London	England	7	Cupboard
				5	Cabinet

DEL\_PROD

Num	ProdID
005	1
005	2
005	3
008	2
008	7
014	5
002	7
002	1
002	2

Convert to  
3NF

DELNOTE

DEL\_PROD

Num	CustName	City		Num	ProdID
005	Bill Jones	London		005	1
008	Mary Hill	Paris		005	2
014	Anne Smith	New York		005	3
002	Tom Allen	London		008	2
				008	7
				014	5
				002	7
				002	1
				002	2

PRODUCT

CITY\_COUNTRY

ProdID	Description		City	Country
1	Table		London	England
2	Desk		Paris	France
3	Chair		New York	USA
7	Cupboard			
5	Cabinet			

Now we can see that redundancy of data has been removed.

In tabular form we have:

UNF

DELNOTE(Num, CustName, City, Country, (ProdID, Description))

1NF

DELNOTE(Num, CustName, City, Country, ProdID, Description)

2NF

DELNOTE(Num, CustName, City, Country)PRODUCT(ProdID, Description)DEL\_PROD(Num, ProdID)

3NF

DELNOTE(Num, CustName, City)CITY\_COUNTRY(City, Country)PRODUCT(ProdID, Description)DEL\_PROD(Num, ProdID)

In this Section we have seen the data presented as tables. These tables give us a *view* of the data. The tables do NOT tell us how the data is stored in the computer, whether it be in memory or on backing store. Tables are used simply because this is how users view the data. We can create new tables from the ones that hold the data in 3NF. Remember, these tables simply define relations.

Users often require different views of data. For example, a user may wish to find out the countries to which they have sent desks. This is a simple view consisting of one column. We can create this table by using the following relations (tables):

PRODUCT	to find ProdID for Desk
DEL_PROD	to find Num for this ProdID
DELNOTE	to find City corresponding to Num
CITY_COUNTRY	to find Country from City

Here is another example of normalisation

Films are shown at many cinemas, each of which has a manager. A manager may manage more than one cinema. The takings for each film are recorded for each cinema at which the film was shown.

The following table is in UNF and uses the attribute names:

FID	Unique number identifying a film
Title	Film title
CID	Unique string identifying a cinema
Cname	Name of cinema
Loc	Location of cinema
MID	Unique 2-digit string identifying a manager
MName	Manager's name
Takings	Takings for a film

FID	Title	CID	Cname	Loc	MID	MName	Takings
15	Jaws	TF	Odeon	Croyden	01	Smith	£350
		GH	Embassy	Osney	01	Smith	£180
		JK	Palace	Lye	02	Jones	£220
23	Tomb Raider	TF	Odeon	Croyden	01	Smith	£430
		GH	Embassy	Osney	01	Smith	£200
		JK	Palace	Lye	02	Jones	£250
		FB	Classic	Sutton	03	Allen	£300
		NM	Roxy	Longden	03	Allen	£290
45	Cats & Dogs	TF	Odeon	Croyden	01	Smith	£390
		LM	Odeon	Sutton	03	Allen	£310
56	Colditz	TF	Odeon	Croyden	01	Smith	£310
		NM	Roxy	Longden	03	Allen	£250

Converting this to 1NF can be achieved by 'filling in the blanks' to give the relation:

FID	Title	CID	Cname	Loc	MID	MName	Takings
15	Jaws	TF	Odeon	Croyden	01	Smith	£350
15	Jaws	GH	Embassy	Osney	01	Smith	£180
15	Jaws	JK	Palace	Lye	02	Jones	£220
23	Tomb Raider	TF	Odeon	Croyden	01	Smith	£430
23	Tomb Raider	GH	Embassy	Osney	01	Smith	£200
23	Tomb Raider	JK	Palace	Lye	02	Jones	£250
23	Tomb Raider	FB	Classic	Sutton	03	Allen	£300
23	Tomb Raider	NM	Roxy	Longden	03	Allen	£290
45	Cats & Dogs	TF	Odeon	Croyden	01	Smith	£390
45	Cats & Dogs	LM	Odeon	Sutton	03	Allen	£310
56	Colditz	TF	Odeon	Croyden	01	Smith	£310
56	Colditz	NM	Roxy	Longden	03	Allen	£250

This is the relation:

R(FID, Title, CID, Cname, Loc, MID, MName, Takings)

Title is only dependent on FID

Cname, Loc, MID, MName are only dependent on CID

Takings is dependent on both FID and CID

Therefore 2NF is:

FILM(FID, Title)

CINEMA(CID, Cname, Loc, MID, MName)

TAKINGS(FID, CID, Takings)

In Cinema, the non-key attribute MName is dependent on MID. This means that it is transitively dependent on the primary key. So we must move this out to get the 3NF relations

FILM(FID, Title)

CINEMA(CID, Cname, Loc, MID)

TAKINGS(FID, CID, Takings)

MANAGER(MID, MName)

### Entity Relationship (E-R) diagrams

Entity-Relationship (E-R) diagrams can be used to illustrate the relationships between entities. In the earlier example we had the four relations:

DELNOTE(Num, CustName, City)

CITY\_COUNTRY(City, Country)

PRODUCT(ProdID, Description)

DEL\_PROD(Num, ProdID)

In an E-R diagram DELNOTE, CITY\_COUNTRY, PRODUCT and DEL\_PROD are called *entities*. Entities have the same names as relations but we do not usually show the attributes in E-R diagrams.

We now consider the *relationships* between the entities:

Each DELNOTE can be for only one CITY\_COUNTRY  
because a City only occurs once on DELNOTE

Each CITY\_COUNTRY may have many DELNOTE  
because a City may occur on more than one DELNOTE

Each DELNOTE will have many DEL\_PROD  
Num in DELNOTE could occur more than once in DEL\_PROD

Each DEL\_PROD will be for only one DELNOTE  
because each Num in DEL\_PROD can only occur once in DELNOTE

Each PRODUCT will be on many DEL\_PROD  
PRODUCT can occur more than once in DEL\_PROD

Each DEL\_PROD will have only one PRODUCT  
because each ProdID in DEL\_PROD can only occur once in PRODUCT

The statements show two types of relationship. There are in fact four altogether. These are:

one-to-one	represented by	_____
one-to-many	represented by	_____<
many-to-one	represented by	>_____
many-to-many	represented by	>_____<

Fig. 3.9.5 is the E-R diagram showing the relationships between DELNOTE, CITY\_COUNTRY, PRODUCT and DEL\_PROD:

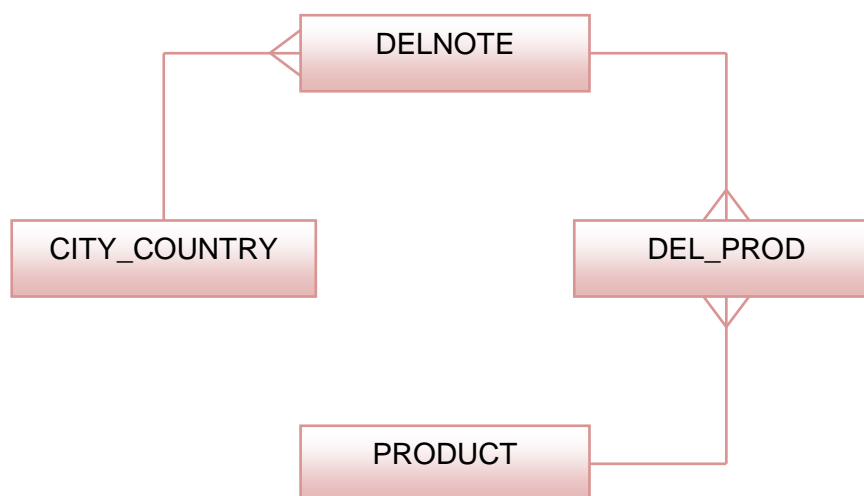


Fig. 3.9.5

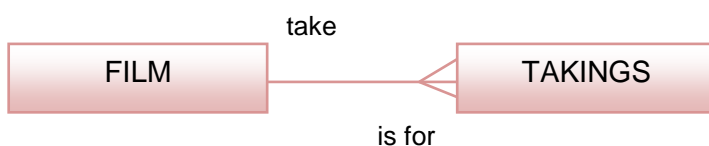
If the relations are in 3NF, the E-R diagram will not contain any many-to-many relationships. If there are any one-to-one relationships, one of the entities can be removed and its attributes added to the entity that is left.

Let us now look at our solution to the cinema problem which contained the relations:

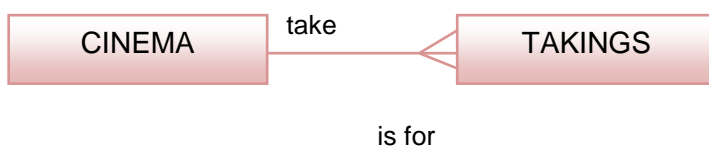
```
FILM(FID, Title)
CINEMA(CID, Cname, Loc, MID)
TAKINGS(FID, CID, Takings)
MANAGER(MID, MName)
```

in 3NF.

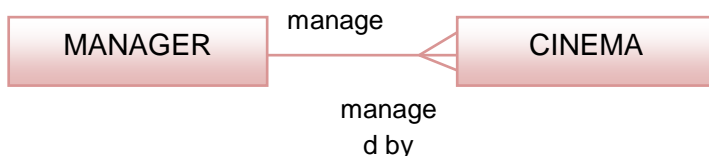
We have the following relationships:



connected by FID



connected by CID



connected by MID

These produce the ERD shown in Fig. 3.9.6:

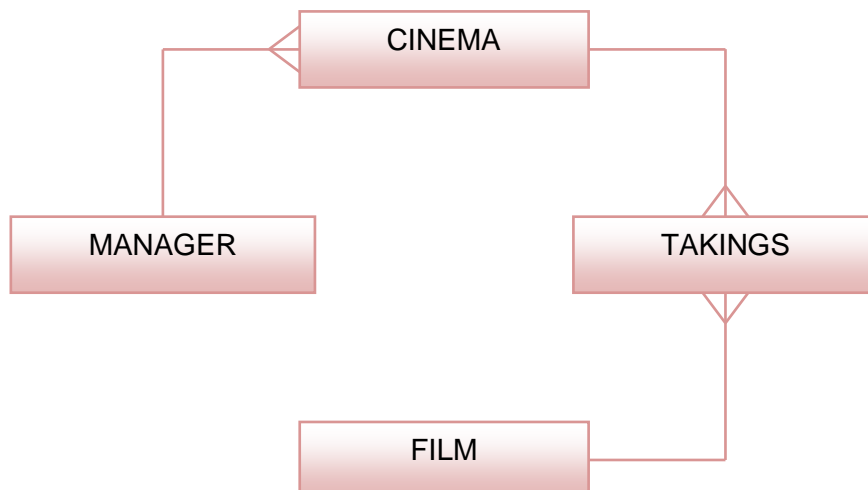


Fig. 3.9.6

In this problem we actually have the relationship:

CINEMA shows many FILMs

FILM is shown at many CINEMAs

That is:



But this cannot be normalised to 3NF because it is a many-to-many relationship. Many-to-many relationships are removed by using a *link entity* as shown here:



If you now look at Fig.3.9.6, you will see that the link entity is TAKINGS.



### 3.9.c Define and explain the purpose of, primary, secondary and foreign keys

We have used keys in all our earlier examples to uniquely identify tuples (rows) in a relation (table). A key may consist of a single attribute or many attributes, in which case it is called a *compound key*.

The key used to uniquely identify a tuple is called the *primary key*.

In some cases more than one attribute, or group of attributes, could act as the primary key. Suppose we have the relation:

```
EMP(EmpID, NINumber, Name, Address)
```

Clearly, EmpID could act as the primary key. However, NINumber could also act as the primary key as it is unique for each employee. In this case we say that EmpID and NINumber are candidate keys. If we choose EmpID as the primary key, then NINumber is called a *secondary key*.

Now look at these two relations that we saw in Section 4.6.4:

```
CINEMA(CID, Cname, Loc, MID)
```

```
MANAGER(MID, MName)
```

We see that MID occurs in CINEMA and is the primary key in MANAGER. In CINEMA we say that MID is the *foreign key*.

An attribute is a *foreign key* in a relation if it is the primary key in another relation. Foreign keys are used to link relations.

**3.9.d** *Describe the structure of a DBMS, including the function and purpose of the data dictionary, data description language (DDL) and data manipulation language (DML)*

Let us first look at the architecture of a DBMS as shown in Fig.3.9.7:

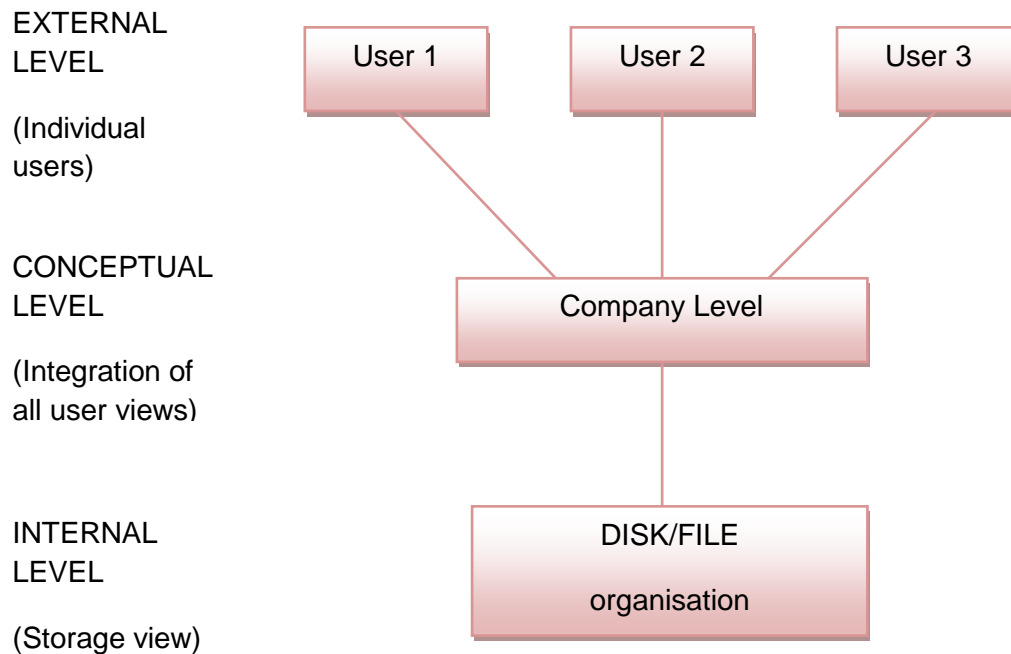


Fig. 3.9.7

At the external level there are many different views of the data. Each view consists of an abstract representation of part of the total database. Application programs will use a data manipulation language (DML) to create these views.

At the conceptual level there is one view of the data. This view is an abstract representation of the whole database.

The internal view of the data occurs at the internal level. This view represents the total database as actually stored. It is at this level that the data is organised into random access, indexed and fully indexed files. This is hidden from the user by the DBMS.

### The DBMS is a piece of software that provides the following facilities

The DBMS contains a data definition language (DDL). The DDL is used, by the database designer, to define the tables of the database. It allows the designer to specify the data types and structures and any constraints on the data. The Structured Query Language (SQL) contains facilities to do this. A DBMS such as Microsoft Access allows the user to avoid direct use of a DDL by presenting the user with a design view in which the tables are defined.

The DDL cannot be used to manipulate the data. When a set of instructions in a DDL are compiled, tables are created that hold data about the data in the database. That is, it holds information about the data types of attributes, the attributes in a relation and any validation checks that may be required. These tables are stored in the data dictionary that can be accessed by the DBMS to validate data when input. The DBMS normally accesses the data dictionary when trying to retrieve data so that it knows what to retrieve. The data dictionary contains tables that are in the same format as a relational database. This means that the data can be queried and manipulated in the same way as any other data in a database.

The other language used is the data manipulation language (DML). This language allows the user to insert, update, delete, modify and retrieve data. SQL includes this language. Again, Access allows a user to avoid directly using the DML by providing query by example (QBE).

Sometimes it must not be possible for a user to access the data in a database. For example, in a banking system, accounts must be updated with the day's transactions. While this is taking place users must not be able to access the database. Thus, at certain times of the day, users will not be able to use a cash point. Another occasion is if two people have a joint account and one of them is withdrawing cash from a cash point. In this case the one user will be able to change the contents of the database while the other will only be allowed to query the database.

Similarly, while a database system is checking stock for re-ordering purposes, the POS terminals will not be able to use the database as each sale would change the stock levels. Incidentally, there are ways in which the POS terminals could still operate. One is to only use the database for querying prices and to create a transaction file of sales which can be used later to update the database.

It is often important that users have restricted views of the database. Consider a large hospital that has a large network of computers. There are terminals in reception, on the wards and in consulting rooms. All the terminals have access to the patient database which contains details of the patients' names and addresses, drugs to be administered and details of patients' illnesses.

It is important that when a patient registers at reception the receptionist can check the patient's name and address. However, the receptionist should not have access to the drugs to be administered nor to the patient's medical history. This can be done by means of passwords. That is, the receptionists' passwords will only allow access to the information to which receptionists are entitled. When a receptionist logs onto the network the DBMS will check the password and will ensure that the receptionist can only access the appropriate data.

Now the terminals on the wards will be used by nurses who will need to see what drugs are to be administered. Therefore nurses should have access to the same data as the receptionists and to the information about the drugs to be given. However, they may not have access to the patients' medical histories. This can be achieved by giving nurses a different password to the receptionists. In this case the DBMS will recognise the different password and give a higher level of access to the nurses than to the receptionists.

Finally, the consultants will want to access all the data. This can be done by giving them another password.

All three categories of user of the database, receptionist, nurse and consultant, must only be allowed to see the data that is needed by them to do their job.

So far we have only mentioned the use of passwords to give levels of security. However, suppose two consultants are discussing a case as they walk through reception. Now suppose they want to see a patient's record. Both consultants have the right to see all the data that is in the database but the terminal is in a public place and patients and receptionists can see the screen. This means that, even if the consultants enter the correct password, the system should not allow them to access all the data.

This can be achieved by the DBMS noting the address of the terminal and, because the terminal is not in the right place, refusing to supply the data requested. This is a hardware method of preventing access. All terminals have a unique address on their network cards. This means that the DBMS can decide which data can be supplied to a terminal.

## 4.0 Computing Project -Introduction



### 4.0.1 General Description

The computing project is a major part of the A level specification (20%). It takes on an added importance because it is unique in comparison with the other modules as it is the only module that gives the student the freedom to choose the content for themselves.

The volume of work expected should not be underestimated. While the degree of difficulty is largely decided by the student and by the problem to be solved, the time required to produce a first class piece of work is considerable. Most centres will be expecting the work to be completed sometime around the Easter holiday in the second year of a two year course. The project work will normally be started in the second half of the summer term in the first year of the course. Students must realise early that, despite having almost a year to complete the work, there is no room for time wasting. Students/Centres are strongly advised to use the natural way that the guidance for marking is set out in order to set individual deadlines to break the time period down into a number of smaller, more easily manageable, ones; a project that is completed in a short period, close to the date for handing in the work, is unlikely to have been completed satisfactorily, if only because it will be difficult to involve the person whose problem is being solved, except on a perfunctory basis.

The computing techniques employed in completing the solution must include a degree of programming and the finished product must show a detailed analysis and design of the circumstances surrounding the problem and the implementation of the proposed solution, along with the testing to ensure its satisfactory performance. The student is expected to play the part of the systems analyst and the programmer (whose duties were expounded in module 1), the testing of the various computing skills necessary having been done in module 2.

The importance of the client cannot be over stated as the best solutions will demonstrate a clear collaboration between the student and a real person with a real problem needing a solution. This collaboration and the way that the student is able to work with another person is a clear requirement of the specification. Notice the reference to 'client' rather than 'end user'. The use of these two terms tends to be somewhat lazy. However, a distinction between them where necessary (many projects will have the same person as client and end user) is an important distinction.

## 4.0.2 Project Selection

This module is very different from all the others, both because the student chooses the topic for solution, and because there is a single topic which lasts for the full length of the module. The theory modules are in different sections and are examined by asking a set of stand alone questions on an exam paper. Because of this, considerable thought must be given to the selection of a suitable problem, that selection being dictated by a number of criteria:

### 1. Syllabus fit

The specification for this subject contains a section (pages 51 to 55) which specifies how the projects are going to be marked. Students and their tutors should ensure that they familiarise themselves with this section and choose a problem the solution to which will require the student to complete all the sections mentioned. The purpose of the marking section in the specification is to act as a guide to the student, indicating what is expected at each stage, it is not intended to be a limiting factor and students should realise that many types of project are possible. Attention should be particularly drawn to sections b(ii) and c(i) both of which require algorithmic/programming work which was not a requirement of the previous specification.

### 2. Self penalising

Students should be encouraged to select a problem whose solution is within their capabilities, but should not imagine that the whole solution will be possible during the time that they have available. Indeed, if the solution is complete, they will be penalised because the final mark points are awarded for identifying possible extensions to the work. Many of the marks in the scheme are awarded for the student identifying the need for a partnership between the owner of the problem and the analyst (student). If there is no real client, the student is severely handicapped when it comes to the marking.

### 3. Real client and/or end user

The intention of the project is that the problem should be real and consequently the person who has the problem should be real. As was stated in module 1, the end-user is the expert (along with other members of the organisation) in the problem and what has given rise to it. The systems analyst (the student) is the expert in computing and the limitations that are inherent in any computer solution to a problem. The interaction between the end-user and the student forms a major part of the project. The end-user should not be thought of as someone who is prepared to give time to be interviewed about the problem and then will try to use the solution 9 months later. Rather, the end-user should be involved throughout the project. At different stages the student should keep the end-user involved so that they develop an ownership of the solution and that they feel to be important to the project (after all they are very important to the student). It is a good idea, for example, if the prototypes of the different parts of the solution are shown to the end-user and then altered according to their comments.

Throughout this the term 'end user' has been used, the distinction between occasions when the end user and the client are the correct collaborators shows a higher level of understanding on the part of the student.

The student is more likely to get more from the work if the problem is a real one and if the end-user is unconnected with the institution. However, such a goal is not always possible because of individual circumstance and also problems experienced by some Centres. In such cases, the use of family members or members of staff within school is perfectly acceptable, but the student should try to maintain a professional relationship with the person being used in order to make the experience as real as possible. The use of fellow students as the end users is not to be encouraged because rarely would the problem be real and the fact that they are both students make it impossible to create the professional, working relationship so essential to this project.

#### 4. Interest

As has already been stated, the student will be working on the problem for a long time. It is essential that there is plenty of scope for maintaining the interest of the student. The typical GCSE project of the football club is unlikely to be suitable because it is about football, although it may be if the local football club is considering computerising its manual filing system because the FA states that information in the future must be in electronic form. This is a real problem, and the choice of project has changed from a choice based on football to a choice based on the solution of a problem.

Another danger in choosing the project is to choose something that the student knows something about. This makes the analysis and the information collection very tricky. The old adage that a little knowledge is a dangerous thing is particularly important here if the student is not aware of the pitfalls. The advantage of a project about which the student has no knowledge is that in order to find out enough to solve the problem, all the marks will be available, rather than if the student has the knowledge already they are likely not to include it in the documentation because they assume that such knowledge is general.

#### 5. Hardware and software

The student and the tutor should have some idea what is going to be required by the solution; otherwise there is the danger that half way through the solution it is found that the hardware or software required to complete the solution is not available. If the student sticks to the basic data handling type of project then such problems are unlikely to arise (unless the solution involves specialist hardware like barcode readers), but if the problem requires a solution which is a bit more unusual, then care should be taken to anticipate problems before they arise.

#### 6. Teacher

Computing is a subject unlike most others on the curriculum, in that the content changes on a regular basis. Software that is available now is very different from that available five years ago. Unlike Maths where a division sum is solved in the same way now as it was fifty years ago, and everyone does it the same way, computing teachers have great difficulty keeping up with some of the changes in the subject. In particular they should not be expected to be familiar with the facilities provided by different types of database programs or spreadsheet programs just because a particular student happens to run it on their machine. At the initial stages of the project it is necessary for student and tutor to have a reasonable idea where the problem solution is going to lead so that, hopefully, there are no unpleasant surprises in store later on in the year.

### **4.0.3 *Problems that may arise***

Problems that arise during the solution of the problem are not something to be worried about. If the problem is a real one, there will be problems. Parts of the solution may not work as anticipated leading to revisiting that part of the analysis. Hardware may malfunction at particularly inopportune times. Such problems should be logged and the solution should be part of the final report.

If the problem is real, one difficulty is likely to be the use of real data to test the solution. The organisation supplying the data is almost certainly covered by the data protection act and at this point it is necessary to accept that it will not be possible to use real data to test the solution. An acknowledgement of this, together with a statement that fictional data has been created to allow testing to be done, is enough to solve the problem.



#### **4.0.4 *Specification requirements***

It is very important that those of us who are used to the previous project work take time to read the 'Guidance for marking' in the specification and the notes here. While the general philosophy of the work is unchanged, the emphases on different parts of the work have changed and the students need to choose a topic for their project which will include the need to program. There is still the requirement that the work is to be a collaboration, there is still the important point that only 30 of the available marks are for the development of the solution and the testing. While this is a greater proportion of the available marks than in the previous specification, the fact still remains that the majority of the marks are available for the systems analysis part of the work.

The programming supplied by the student may range from a project where the entire solution is built on original coding, to projects where a standard information handling problem has been solved with database software and VB has been used to write some macros which can be used to manipulate the data held.

It is particularly important during the first years of the project for teachers to ensure that they are interpreting the detail of the requirements properly. OCR are very helpful in providing various services to assist teachers and readers of this text are advised to take advantage of the services available, details of which can be found in the published specification.

### **4.0.5 *Format of the Chapters***

This module is very different from the others, both in the way that it is presented in the specification and in the way that Centres and students approach it. The specification has the usual learning outcomes as part of the content, but also has the marking guidelines for the projects at the back of the booklet. Experience shows that Centres tend to use the marking guidelines to inform their work, as do the individual students. Because of this, module 4 is presented according to the marking guidelines section of the specification rather than the specification itself, although references will be made to the learning outcomes throughout.

At the end of each section there will be a list of the evidence that the examiner will be looking for in the final report. Every item in the evidence list may not be required for every project, and there may be specific project types that may require additional pieces of evidence that are specific to that problem solution, but the evidence list will provide a good background to the content expected.

The wording of the specification (pages 27 to 29) should be noted. For each of the sections the wording is 'candidates should be able to'. This does not mean that students must demonstrate skill with all of these techniques mentioned. Rather, it means that the student should have the abilities and should then demonstrate the skill of selection in deciding which of the techniques available to them is appropriate to the problem which they are trying to solve. The student who decides for good reason that it would be necessary to interview the client but that no other information collection is necessary because the client is also the single user will be considered to have satisfied the requirements of the project assessment better than a student in the same situation who manufactures questionnaires, collects forms of questionable value and observes the client at work, for no other reason but to 'jump through hoops' or 'tick boxes'!

Throughout this module the person that the student is collaborating with will be referred to as the 'end user'. In many projects it must be remembered that the client and end user may be different people and in some projects there may be a number of end users so care must be taken when seeing the term 'end user' to interpret it in a way that is suitable for the particular piece of work being done.

## 4.a Computing Project

### -Definition, Investigation & Analysis



#### 4.a.i Definition - Nature of the problem to be investigated.

The first task of the student is to select a problem to be solved as their project according to the notes in the introductory chapter of this module. There are two very important points about the problem. The first is that the person who has to assess the worth of the project when it is finished knows nothing about the problem and must have it explained to them in great detail if they are to understand the rest of the work. Secondly, the student and the owner of the problem must have the same idea about the nature of the project otherwise the student can never be sure that they are solving the correct problem.

The first of these problems gives rise to this section of the project work.

There is a need for the organisation to be explained. The difficulty here is the depth of description that needs to be given. Is it necessary to list the members of the organisation? Probably not. However it is necessary to list those members of the organisation that have direct access to the data around which the project is going to be written, and to describe what rights of access each of them should enjoy.

Students who are familiar with the area of the problem find this section more difficult than others because they find it very difficult to gauge how much detail should be given. The teacher is essential in this aspect of the work, to act as a sounding board for the student, to ensure that the student is providing the right level of information. The teacher may find this difficult if they have knowledge of the problem area. If this arises then it is sensible for the student to seek another arbiter who knows little or nothing about the problem area.

In order to analyse the situation, the student must determine how the present system handles the data, or solves some other problem. To do this, initial discussions must take place with the users and the student needs to begin to collect some information in the form of

- methods by which these data are presently stored
- types of data that are used/produced by the system
- how the data is collected/where does it come from?
- what form the data takes and in what form is it stored?

There should be some indication of possible areas that are not performing as well as they should or that are not covered by the present system, because if there are no such areas then the project does not have a problem to solve. This discussion must, by definition, be somewhat sketchy because the views are all coming from the point of view of the systems analyst, the owner of the problem has not yet become fully involved. This is what happens in the next stage of the work.

## Evidence

The report should contain:

- A description of the organisation that has the problem and the place of the problem within it. This does not have to be in any great detail, as a guide, half a side of A4 should be quite adequate.
- A description of how the chosen problem is dealt with at the moment. This can only be a sketchy description because, *until the analysis section has been completed, it is not possible to describe the area in any detail.*
- A clear description of the data that is used in the area of the problem. The exact data that will form part of the solution is not yet known because the problem has not yet been fully specified; however, it is necessary to be aware of all the data that may be required. This does not mean all the information. We do not yet want to know that John Jones hired a school locker (NO. 453) on the 18/11/08. He is in form 8JHL and that he ... What we do need to know is that the pupil name is important, that the name must be linked to a locker, that the date the hire began is important in order to calculate the cost, that the form of the pupil is important in order to give a point of contact...
- A clear indication of where the data came from. How it is collected. Something along the lines of: The school sends a letter home with a tear off portion that asks for the following facts....; the lockers are in different areas of the building with those in block 1 starting with the number 1...; Different year groups are in different blocks...

Much of this information may not be known. The analyst may have to report, at this stage, that the means by which particular locker numbers are assigned to different pupils is not known, but that it will be necessary to find out during the interview with the end-user or with the client.

## 4.a.ii *Investigation and Analysis*

This section is really the pivotal section of the whole project. It is in this section that the analyst and the end-user pool their knowledge in order that decisions can be made about the direction that the project should take.

Following the first stage, the analyst should have a number of questions to which they need the answers. Although the example used was one from within the experience of most students because something similar will happen in their own schools, this is simply for illustrative purposes. The ideal problem is one which the student genuinely has to find out about because they know little about it.

This section of the work is in four parts:

1. Finding out what the problem is in some detail.
2. Learning about the area within which the problem is situated.
3. Analysis of the present system and use of different techniques to illustrate the analysis, including an analysis of the problem itself, though not the solution yet.
4. A requirements specification which will allow the student to carry on to the next stage. This will include a hardware specification and a software specification.

1. Problem. This is going to involve the end user. The analyst (student) needs to find out, in detail, what the end user wants from the system. If this part of the project is not done carefully, then the project will not be successful because the end user and the analyst will have different ideas of what the problem is. The normal solution is to arrange an interview with the end user. This is not a matter of the student having a general chat with their end user, or even having detailed discussions where the end user has carefully considered all the questions and asks them in order. It must be planned and yet be very different from simply reading from a questionnaire.

The first point to be considered is how the interview is going to be conducted. Will the student try to copy down the end-users comments as they are being spoken or not? This is a very poor interviewing technique because it does not allow the analyst to consider what is being said before asking the next question. Two better alternatives are for the analyst to take a third person with them whose job it is to write down everything that is said, or to use a tape recorder to tape the interview. Whichever method is used, it is simple politeness to tell the end user how the interview is planned and to ask permission (particularly if the interview is to be taped). At the same time it is wise to check with the end-user whether there are any subjects that should not be brought up in the interview. In the real world there may be sensitive topics that the student will not be allowed access to. Although it is simple for the end-user to refuse to answer a question in the interview, it shows an understanding on the part of the student if they cover this problem beforehand.

The student should go into the interview with a series of well prepared questions to ask. However, they should not be asked as a simple list. If this is all that is done, the interview is a waste of time; the end-user could simply have filled in a questionnaire. The whole point about an interview is that the questioning should be flexible. Students should have 'starter' questions on each of the areas to which they want answers. They should also have prepared a series of follow up questions that can be used, dependent upon the answer to the initial question. If a student finds themselves saying something on the lines of "That is interesting, in which case what would you do about..." then they are demonstrating flexibility in their questions, they are demonstrating that they are listening to the responses, and that they are showing an interest in the end user. In short they really are playing the part of a true systems analyst. In order to do this properly the questions must be very carefully planned. The order of doing this should be: Decide on something that needs to be asked about (If you already know the answer what is the point of asking?); decide on a question to ask which will give the information; All questions must have more than one possible

answer, so think of a follow up question that can be asked for each possible answer that the end user can give. For example the student may need to know what types of information are important. If the end user states that information needs to be kept on customers the follow up question may be to ask about whether this information needs to be protected, if the answer to that is yes then a further follow up may be to ask who on the staff is allowed to access the information and who is allowed to alter the information. The important thing is to have these questions ready and to only use the ones that follow on from what the end user says in answer to the original. In this way the interview transcript looks like a conversation rather than just the answers to a questionnaire and demonstrates that the student was playing a full part in the process.

2. Background. Students should not forget that there are different people involved in this organisation. Mrs. Robinson may be responsible for the allocation of lockers, so she was the end user who was interviewed. However, Mr. Bramhall is in charge of year 8 and therefore deals with any problems that may arise, the form teachers actually distribute the keys and collect in the money, and the pupils and their parents are the actual users of the lockers. There will be important information to be gleaned from these people, which can be done by using different methods of collection. For example, a sample of parents (how are they to be chosen?) could be sent a questionnaire, while the student may be invited to go to the next Year 8 form teachers' meeting to lead a discussion about the present system. The present documentation needs to be collected and analysed. Does the present letter collect all the information which is thought necessary?

Notice that, in the example, a lot of different forms of information collection are to be used. The important thing to realize is that they are not being used to demonstrate to the moderator that they know what they are; they are being used because they are a sensible form of information collection for this particular problem. The interview is mandatory, but other types of information collection should be justified by reference to the problem.

3. Presentation of the Analysis. This is another set of evidence where it is so easy to fall into the trap of giving an example of everything you can think of. Produce a data flow diagram, a system flowchart, structure diagrams. If this is done without thought then it can be counterproductive. The student is not trying to show the moderator that they know how to draw a DFD, they are showing whether they can determine whether a DFD is sensible evidence, and then drawing one if necessary. Do not lose sight of what is being assessed – the ability to solve a problem.

4. A requirements specification. The student has now identified the problem and should be able to list the requirements that the end user has in mind. This set of requirements should be signed off by the end user. It is important to note that this process, like so many others in the work, will probably not be a linear process because the end user may not agree with the student when the requirements are produced. This is not something to be hidden away, this is good evidence that the work is 'real'. In the real world such collaboration does not mean that the student states something and the end user has to agree immediately. Evidence of disagreement which involves the student in reworking the evidence is to be applauded and credit is given for it.

Thoughts must now turn toward the hardware and the software that will be necessary to provide a solution. It is at this stage that the teacher must discuss with the student the requirements that they consider are going to be necessary in order to make sure that the requirements are going to be available and that there are not going to be any unpleasant surprises later on in the work, when it is suddenly discovered that the chosen software cannot do something that was expected of it. Hardware and software choices must be justified in relation to the problem solution and should (like all other evidence) be aimed at the correct audience, in this case the end user who is the person who has to agree with the choices.

## Evidence

The report should contain:

- Detail relating to the planning of the end-user interview.
- The original plan of the questions, showing that all areas had been planned to be covered and that sensible follow up questions had been considered, dependent on the answers which the interviewee could give.
- Transcript of the interview itself, including an element of later analysis and isolation of important facts.
- Further evidence of information collection relevant to the problem area.
- Possibly a data flow diagram and/or some other representation showing how the present system works.
- An agreed list of requirements of the solution.
- Hardware and software requirements of the system with discussion about the needs for each and any problems that may arise because some elements are not available.

## 4.b Computing Project -Design



### 4.b.i *Nature of the Solution*

Following the collection of information about the organisation and the perception of the end-user about the problem that needs to be solved, the student should now decide what the scope of the project is going to be. In other words, the student should decide what needs to be done by the time the project is finished. The student should draw up a list of required outcomes. This list should contain objective points, not subjective ones, i.e. points that can be marked at the end of the work as being obviously achieved or not.

In the example, the list might start with the following objectives: Mrs. Robinson must be able to access all the records. Mr. Bramhall must have complete access to year 8 records but not to other years. It must be possible to pass on records to the next year/form groupings at the end of a year. A form teacher must be able to output a series of letters to the parents of pupils who are behind with their payments... These objectives are all easily quantifiable. When the project is over it is simple to check to see if form teachers can produce the required letters. Objectives like 'The solution must be easy to use' or 'The solution must be presented in pleasant colours' are very difficult to judge. Indeed, two people can quite reasonably have different views on the same piece of work. This list of objectives must be agreed with the end-user. This may well mean that the original list of objectives may need to be altered when the end user sees it. The thing to be remembered is that neither the student nor the end user should dictate the objectives, they should be agreed upon. This final list should be signed by both parties and is going to provide the basis both for the evaluation of how successful the project has been, and also for the testing regime that is going to be used in section c.

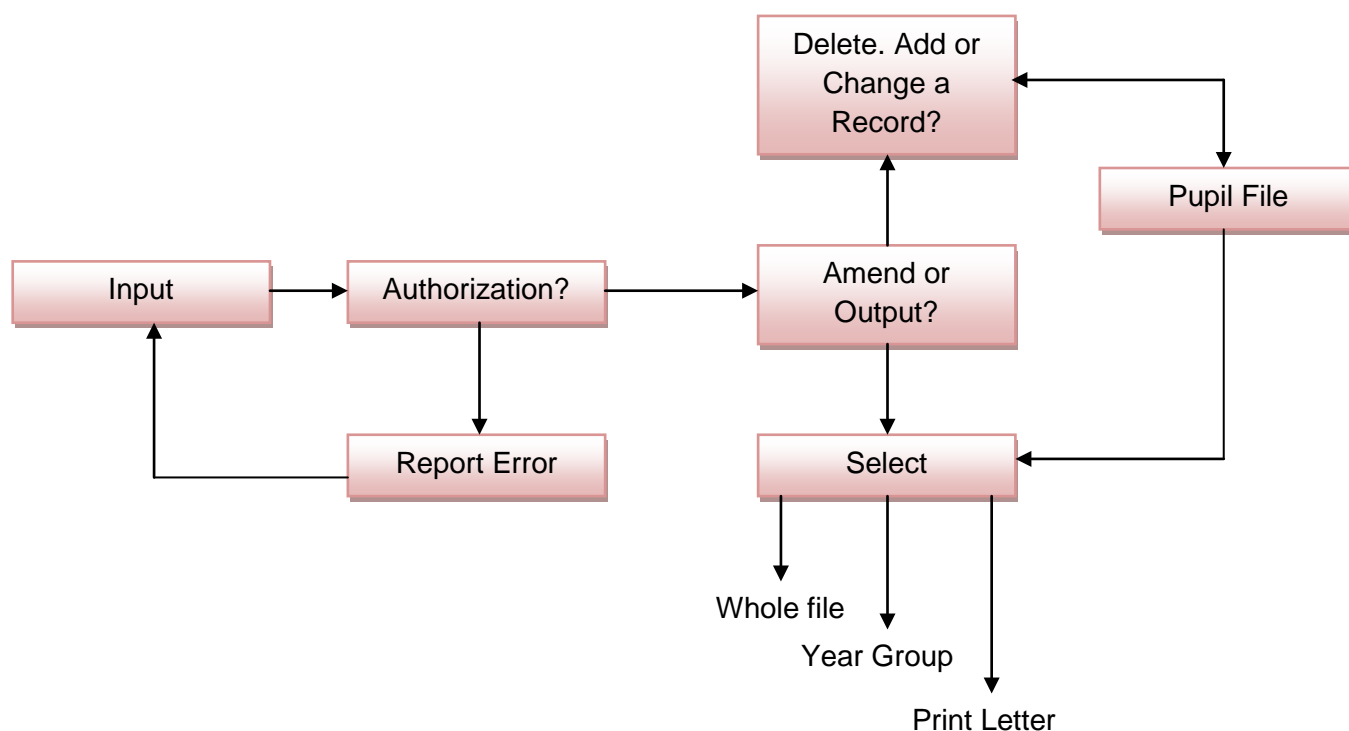
Whatever the problem which is being solved, it will involve some form of data entry, some processing and then an appropriate output. We tend to think in terms of data storage/manipulation problems but this applies to all projects. Consequently, the next stage is for the student to design the output documents and the input formats with appropriate screen designs. The end-user does not need to know anything about the processing or the data structures that are to be used because they are not in the realm of the user. However the end-user does need to know what the input and output formats are likely to look like. These, together with the list of objectives, make up the design specification and should now all be agreed with the end-user.

Students should realise that this process of agreeing the design specification should take some time. If the student has one interview with the end-user, who, subsequently, immediately agrees to all the student's suggestions without needing further modification, then the relationship is not a real one. Or to put it another way, there is a fiddle going on! Remember that the intention is to mirror the real world, in which the process would continually need fine tuning. Later in the project the student may well find that the original design specification is no longer satisfactory, this is not a point at which the student should think that their project is a disaster, but rather an opportunity for involving their end-user again in modifying the design specification. The student should be encouraged to document all such occurrences and how they were resolved.



The next stage is to determine how the data will be moved around their system and stored within it. The simplest way of describing this is to draw it. Students often find this the most difficult part of the project, but it needn't be. Often the difficulties arise because the student thinks that data flow diagrams or system flowcharts are highly structured and because they do not know the 'language' of boxes and how they should be joined together. They should not worry about this. If a student particularly wants to use 'boxes' that are universally recognised as particular types, they are going to be out of luck anyway, because there are lots of different systems. By all means choose one, but that is not the point of this part of the assessment. The important thing is to convey to the reader that there is a clear idea of how the different parts of their system fit together and where the data is stored within the system.

The example that we have has inputs being made to the system which are different dependent on the user. This means that inputs have to be shown, as does some mechanism for checking the authority of the person to make the inputs, and that files have to be altered. The files that have to exist seem to be a file of pupils and another of payments, which will be linked in some way, and the outputs will include lists of locker keys and pupils by year or form or all at once. Also letters to parents.



Although this is not the sort of diagram that you would find in a text book, what is missing? The ability to draw pretty boxes is not being tested here, simply the student's understanding of the logic of the intended solution and of how the various pieces fit together. Individual parts of the logic may need to be considered in more detail: How is the authorisation gained? How does the user select what type of operation? But the basic details are there. Note that the pupil file is shown, but that the payments file is not and would be in the full diagram.

The next stage is to consider the data that needs to be stored on the system. The data should be divided up into different groups (files) dependent on the use that will be made and the links with the other files. An attempt should be made to arrange the data into files so that there is as little duplication of the data as possible. This will inevitably mean that the files will need to be linked so that data from one file can be used in an application area designed for a different file.

The data attributes, or fields, from each file should be listed. It should be clear what field or fields will be used as the key and also the foreign keys that are going to provide links with the other files in the system. It is then necessary to explain the purpose of storing each of the fields and to decide the data type that is to be used for each field. When the files have been defined, it is then necessary to size them by giving a size to each of the fields and scaling up to give an indication of size for the whole file. Each of the items of data in the file will need to be input, consequently there will need to be validation done on the data input. These proposed validation checks should also be detailed.

The most important sentence in the specification of this section appears toward the end of the paragraph for 5-6 marks on page 52: "...sufficient for someone to pick up and develop an end result..." This states that the person who takes up the project at this stage will need to do none of the analysis or planning work for themselves, but will simply have to put the plan into action.

It is important to note that these notes seem to have veered towards a database solution. While this is probably going to be the most common type of project, it should be understood that any form of solution is appropriate, the important thing is to have a real problem to solve, not what the solution is. The exception to this is that there must be room in the solution for the student to provide some original coding, as we see in the next section.

### Evidence

The report should contain:

- A list of objectives against which the final product can be measured. This list should be signed by the end-user.
- An explanation of the way that the data is to be collected.
- An explanation of the way that the data is to be input to the system together with any initial plans of input screens, and explanation of intended means of validation.
- An explanation of the output that should be expected and any initial screen designs for the output of data
- Details of the file structures that are to be used and the links between them.
- Explanation of the way the data is manipulated through the system. This section should contain a diagrammatic representation of the data through the system.
- If the solution is to be attempted in module form there should be a description of the way the problem has been modularised and also how the modules will fit together.

The specification clearly states that the design should be sufficient for someone else to produce the solution with no further analysis or design. The other members of the group can be used to critically view each other's efforts to see if they could consider attempting a solution from the information in the design specification.

## 4.b.ii Algorithms

It is expected that the Computing project should include an element of coding using a high level language. Such coding needs to be planned and this is the section where that planning is going to be assessed. The algorithms of the solution should demonstrate how the different elements of the solution function to provide the desired solution. The intended coding should have preparatory work done which will allow the coding to be constructed simply by reference to the algorithm supplied. The algorithms must not only have the different stages of the solution explained, but must be ordered.

The form that algorithms can take can be various, but should always include two things: information/instructions about a stage of the solution and an implied order that the information/instructions should take. They can be presented in the form of pseudo code, flow diagrams, system diagrams, ordered points, written descriptions... The form of presentation is secondary to the logic offered up and the perceived usefulness by a third party. Care should be taken that whatever the form of presentation there is no ambiguity.

### Evidence

The report should contain:

- A complete set of algorithms in appropriate form which describe the method of solution
- Algorithms which describe the original coding which is intended
- Testing of the algorithms presented to provide evidence that the algorithms have been worked and that they produce the intended results

### 4.b.iii Test Strategy

In this section, the student is encouraged to decide what the necessary testing of the solution should be. This testing should be related explicitly to the objectives of the solution. The intention of the testing in this strategy is to provide evidence to the end user that the solution works. To that end each test in the test plan should not only specify what the data is that is going to be used and the expected outcome, but also the objective which is being tested. This can normally be included in the 'reason for the test' but another method is to have a separate column in the test plan which is headed 'objective' and if the objectives are numbered appropriately, this linking process is simple to do. All the objectives should be included in the test plan, in enough detail to persuade the end user that the system works.

If an objective is of the form: 'A mark should be allowed to be input between 1 and 100' then it is necessary to ensure that a mark can be entered (normal data), that marks that are too small or large are not accepted (abnormal data) and that a mark of 1 or 100 is/is not accepted dependent on where the line is drawn (extreme data).

The normal format for a test plan is tabular because it is the simplest to produce and to understand. This table form is advised to students, but any other form of strategy presentation could be used. The table will have some blank columns at the moment because they cannot be filled in because this table has been produced before the solution. It should have a column for the expected results, filled in, but the columns for the actual results and the one for any comments about the differences or otherwise between the expected and actual will have to wait until after the solution has been produced.

The assessment will be based on completeness and a satisfactory presentation.

#### Evidence

The report should contain:

- A test plan covering sufficient functional testing to persuade the end user that the solution will work
- Sufficient testing to adequately test all the stated objectives in b(i)

Normal, abnormal and extreme tests particularly of any validation routines that are planned on data input.

## 4.c Computing Project

### -Software Development and Testing



#### 4.c.i *Software Development*

Time has arrived to actually use a computer. The planning of the intended system has been done and the design has been fully documented. The decision has been made whether to use a software package with an element of original coding to manipulate data, or to write a piece of software from scratch.

The design section has been completed and the software development section should follow the design that was produced. This is particularly important when referring to the coding, the algorithms in b(iii) must match. Students should be aware that it is only when the actual production of the software is considered that some problems will inevitably arise. There is a right way and a wrong way to deal with occasions when it is necessary to depart from the original design. The wrong way is to simply carry on and do something different to the design. It may work, but it is departing from what was agreed with the end user and this path leads to a solution which ends up not doing what the problem made clear was necessary. The correct thing to do is to return to the design, change the design to match what is going to be necessary and then get renewed agreement from the end user. So often, students think that this demonstrates a failing on their part. Quite the opposite is true. This is what happens in real life. The design is seen to be inadequate in some way so it is necessary to return to it and adapt it to the new circumstances. A project where it all works perfectly first time probably points to a very simple problem or to a fiddle, as mentioned in section b. A report on the project (what the moderator sees) that shows this continual process of adapting to changing needs shows a real situation and must earn the student credit.

When the coding has been written it is necessary to ensure that all the work has been annotated. Do not be scared of 'spoiling' your beautiful printout by annotating it. The more written explanation on it the better. It is a good idea to do all the annotation in pencil because it distinguishes it from the other work and it also means that it can be altered easily, alternatively use the comment facility in the language that you have used.

How much annotation is necessary? Officially the definition of 'fully annotated' is that it should be possible to remove all the code and then a person experienced in the use of the language should be able to recreate your code from the annotation alone. Perhaps a better definition is to say that the annotation should explain the purpose of each part/line of code to the reader (in this case to the moderator). After all, the algorithms in b(iii) should be sufficient to produce the code if they have been done properly.

#### Evidence

The report should include:

- Any necessary data structures are described and their existence is proved by means of printing them out.
- Input and output screens are described and produced. They should be fully annotated to show their effectiveness.
- Files before and after the input of the data to show that data is, indeed, input to the file. Remember that it is not good enough to simply type in the different fields of a record and print out the form.
- Detailed and annotated program listing of the original code that has been produced.
- The white box testing that has been carried out during the production of the software solution to ensure that different stages have been successful before going on to the next stage.

#### 4.c.ii Testing

It is essential that the solution is tested in order to make sure that it is able to do all the things expected of it. If the testing is planned after the solution is developed there are two problems. The first is that the testing will tend to be devised in order to show what the solution can do, not whether it can do what was intended and this leads to the second problem, that the original specification may not be tested at all. For these reasons the testing needs to be planned before the development of the solution and it should have been done in section b(iii).

The elements that were detailed in the test plan, which made up the section on test strategy, before the development of the software solution should now be executed and the results tabulated. Remember that it is impossible to fully test a software solution to ensure that it always works because for most problems there will be an infinite number of possible input/process/output combinations. All that can be reasonably expected is that enough test runs have been carried out to show that the different parts of the system work as anticipated and that errors in the data are captured and don't produce any unpleasant surprises. The mark scheme does state that normal, abnormal, and extreme data should be used in the testing of the system.

##### Evidence

The report should include:

- The testing plan. This should consist of a number of different tests that will test whether different areas of the solution satisfy the original objectives agreed with the end user in b(i). If the full marks are wanted from this section, the student should include enough testing to be able to test all the different areas of the solution, though, remember, it is not possible to test every aspect of the solution with all possible values, and this should not be attempted. This testing should be considered to be the alpha testing.
- The results of beta testing, carried out by the end users and the owners of the solution. To just state that the end users have used the system and are happy with it, is not appropriate to this work, the beta testing must be part of the test plan and the hard copy evidence of the testing should be provided.
- For full credit the testing should provide enough evidence to persuade a reasonable end user that the agreed objectives have been satisfied.

#### 4.d Computing Project -Documentation



The first thing to realise is that there are two quite distinct parts to the documentation for the user. A User guide is simply that. It is a guide for the person who is intending to use the project to do something useful. Unlike a technician the user does not want to understand how the solution works (indeed, the user may well not be able to understand the logic behind the solution) but they do need to know how to make it do something useful. The two types of user guide are a guide in paper form and one which is available on screen, electronically. The first is useful because it can be used away from the computer, while the second is useful because the contents required by the user are determined by the position in the solution where help was requested. In reality the best guide will combine the two types and that is the requirement here.

Students should be wary about including details of the software that has been tailored. It is the tailoring of the software that provides the solution to the problem, it is the tailoring of the software that the user will use, and consequently, it will be the tailoring of the software that should be documented here. As a simple illustration, consider the case of a project that allowed the user to search a file of data for all the people on the file who live in a certain town and then to mail merge their details with a standard letter in order to send information to them about something happening in their town. If the solution is to be based on the use of a standard query from within the software package itself, then the software has not been tailored properly and the inclusion of the necessary techniques in the user guide would mean copying out chunks of the software guide which would not be appropriate. However, if the solution has involved the correct degree of tailoring then the user guide will contain explanation of the forms that have to be completed and the type of data that is required. Examples of valid data would also be provided and the user would be presented with example outputs so that it was known what to expect. The user would not expect to see a long explanation about the way that the software created the query and how it was dealt with.

Students should remember that there are a number of things that are important in the guide, above and beyond the obvious 'How to use it'.

There will need to be 'an initial set-up guide'. This may only ever be used once, but it is essential nevertheless. A 'getting started' section will be expected, which, again may only be used once, it may include notes on the initial setting up of a file of data for instance which may only need doing once. Also necessary will be a trouble shooting guide, the contents of which will be determined by the particular project. A good guide as to what to include is to go back to the objectives section. The student should ensure that the guide allows the end user to do all the things that were listed in the objectives as being important. There are, of course, other things which would be needed in most projects as standard contents: Back up routines, use of the hardware, error messages and actions necessary following them.

The input requirements and the choices that the software will offer the user together with an explanation of the results of those choices should be present. Details of the outputs, both hard copy and to storage for the purposes of backing up the data are important. In some circumstances it may be appropriate to go further and discuss the need for archiving some of the data, particularly if the solution is based on a regular cycle, like the start of the school year. The solution may use a number of different pieces of software. In this instance the student should ensure that they include an explanation of the different types of entry for different types of data. This will normally take the form of a menu screen offering the user one of a number of forms that can be filled in with the data. These different options should be understood. The user should be able to find their way around the user guide with ease, consequently, an index is necessary (or at least a contents page). The student should not lose sight of the fact that the final user will probably be largely computer illiterate. This means that there are likely to be a number of terms used in the project as a whole, and in the guide in particular, which will need to be explained. A glossary of terminology used (like a short

dictionary) is valuable, though do bear in mind that the appropriate use of technical terms, correct spelling and grammar, form part of the assessment in this section.

There must be an on-screen help guide to supplement the paper one. The sophistication of this guide will depend largely on the abilities of the students. It could range from simple on screen messages to a full user guide, book-marked to allow relevant access from particular screens that the user is using at the time. Evidence of the on screen guide must be produced in hard copy form in enough detail to demonstrate its use to the reader of the project report.

### Evidence

The report should contain:

- A contents page and possibly an index.
- Details of input requirements including examples of valid and invalid data with explanations, examples of the input screens so that the user knows what the screen will look like. How to move from input to input on the screen, how to show that the screen input is completed
- Methods of output. How to select them, where the output will appear, how to facilitate output, even simple things like 'ensure that the printer is switched on'. If there is a choice of hardcopy output, how is the choice made and what criteria are used to decide on which output device to use?
- Explanation of the error messages that may appear. In most projects these will tend to be related either to the printer or to data that has failed a validation test on input to the system.
- Back-up routines and data archiving strategies.
- Methods that have been used to protect the data. For example passwords on files.
- Full evidence of on-screen help facilities. This may take many forms but note should be made that the assessor can only award credit for what they can see so it is important to the student to ensure that all aspects of the help are evidenced in the hard copy material.
- Ensure that there are 'getting started', 'troubleshooting' sections.
- A section on the security of both the software and the data in use.



## 4.e Computing Project -Evaluation



### 4.e.i *Discussion of the degree of success in meeting the original objectives*

This section centres around a comparison of the hopes that were originally identified for the solution, way back in sections a(ii) and b(i), and what was achieved by the end of the work. Each of the requirements that were originally agreed with the end user should be compared with the final outcome and an assessment made of how well the project has worked. It is important to provide evidence from the testing of the finished work to support the assertions made.

#### Evidence

The report should contain:

- A clear matching up of the agreed objectives of the project, with evidence that the objectives have been met. For each objective a discussion about the degree of success that the student believes has been achieved (with reference to the evidence in the rest of the report), any short comings that the student has been able to identify and the reasons why such shortcomings occurred.
- The results that were obtained upon using 'real' data rather than manufactured test data. It is understood that any problems outlined earlier about restrictions because of the DPA or similar are still accepted. The distinction between the two types of data should be seen as: test data has been specially chosen to have a certain characteristic which can be used to test a specific area of the software solution, while real data is the sort of data that will arise in operation of the system.

#### 4.e.ii *Evaluate the User's Response to the System*

The end user needs to use the finished system.

Ideally the end user will be presented with the user guide and a machine that has been switched on with the software loaded up, and left to get on with it. This is a bit unreasonable, and the student should be on hand to answer any questions and to help in the use of the software as the end user will not have been trained yet. However, the principle that the end user should use the solution (and not just sit and watch the student make it work) is an important one. There should be evidence that the end-user has used the system, probably in the form of print outs of completed tasks and a **letter from the end-user stating how well they think the solution has met the specification**. It is often helpful to present the end user with a worksheet to follow so that they are led through the system gently and so that there are no areas of the software that they fail to use.

##### Evidence

The report should contain:

- A letter from the end-user stating that they have used the software solution and listing the things that they managed to do and those that they could not, or that still need work to improve them.
- Some evidence to support the letter. For example a printout showing an amendment to a record in the file.
- Some indication of the degree of success that the student has achieved in the opinion of the end user.
- The student describes how any inadequacies highlighted by the end user could be dealt with.

#### 4.e.iii *Desirable Extensions*

The student should be aware that no system is ever finished. However well a system works there are always going to be extra things that could be done, rough edges that could be smoothed out, new hardware that could be used to give an improved finished product... The intention of this section is to show that the student is aware that however good their project is, and however wonderful their end user said it was, there is always something that could have been better.

The student should also be prepared to give themselves a pat on the back when they get something right. This ability to recognise that a part of the work is successful is an important point about the production of a system because it is necessary to have the skill to know that there is nothing to gain from continuing this part of the system and that efforts would more profitably be spent on something else.

##### Evidence

The report should contain:

- A list of those parts of the system that the student believes are satisfactorily completed.
- A list of those areas that have not been successful. A reason why the student holds that opinion and what should be done about it if there was enough time to rectify the situation.
- Areas within the system where previously identified limitations have had a marked effect on the results.
- A clear indication that the student realises that, although the project is finished, the solution is not and that there are a number of possibilities for extending the solution with brief outlines about what would be involved in such extensions.