Computer Science AS & A Level | AQA | 7516 & 7517

Course Companion

for A Level AQA Computer Science

Includes AS and A Level

Update v1.1 – 29th September 2016

zigzageducation.co.uk

POD 6061

Contents

Teacher's Introductio

This resource covers all the theory needed for the A Level AQA Computer Sci for first teaching in September 2015 – with the first exams in June 2017.

Each main topic of the specification is given its own section in the resource.

1. *Programming*
2. *Data Structures*
3. *Algorithms*
4. *Theory of Computation*
5. *Data Representation*
6. *Computer Systems*
7. *Computer Organisation and Architecture*
8. *Consequen*
9. *Communic*
10. *Databases*
11. *Big Data*
12. *Functional*
13. *Systematic*

Within each section there are student notes covering the specification content a descriptions of theory supported with examples, fact boxes, diagrams, images a

In addition to more generic pseudocode, code snippets are included for the f

- Visual Basic .NET
- C#
- Python
- Pascal/Delphi
- Haskell (*Functional Programming* topic only)

Questions and tasks are interspersed throughout the guide to test and devel There is also a separate set of high-level and assembly programming tasks a combine different programming concepts to test their skills as a whole.

Answers/solutions are included at the back of this resource to save the teach comprehensive set of definite answers. In some cases, there are equally vali have been given.

As this companion also includes all the content needed for the separate AS c September 2015, with the first exams in June 2016), content which is *only* re indicated using the dotted border and **A LEVEL** stamp, as shown here.

This is designed to assist co-teaching between the levels.

Update v1.1 – September 2016

A number of improvements, including (but not limited to):

- High-level and pseudo code fixes
- Haskell code added to topic 12 (Functional Programming)

Full Contents Listing

1. Programming

2. Data Structures

3. Algorithms

4. Theory of Computation

5. Data Representation

6. Computer Systems

1. Programming

It is important to develop both practical skills and understanding of the theory behind the will be assessed in this course. Although there are many different languages – all with di share many of the same fundamental concepts, even if they do work in different ways.

This section covers:

1.1 PROGRAMMING

DATA TYPES

In order to run efficiently, computers need to be able to handle all forms of data.

When you define a *variable (see p.4)*, you must also declare a data type. This gives the computer an understanding of how much memory needs to be allocated as well as what operations can be applied to an item of data. For example, you cannot store an integer in a variable designated for storing text and vice versa.

Before you can start programming you must start with a blank canvas and start with the basics.

Language-defined data types

Integer	Any whole number (inclusive of negatives and zero). For efficie will offer a varying size. In increasing order these are 'short', 'ir when accuracy isn't of high priority.
Real	Often referred to as '*float*', this is any number within a *range* a are user defined. These can be whole numbers and contain a d *mantissa-exponent* form. As with integers there are two sizes: 'f used where accuracy is of high priority (e.g. when dealing with
Boolean	Stores whether a condition is TRUE or FALSE. Default is set to Programming languages that do not support Boolean variables FALSE and 1 is TRUE).
Character	This can contain any keyboard character inclusive of special ch
String	A set of characters; used to store a representation of text.
Date/Time	A representation of a moment in time. Can be used to return it
Pointer	Sometimes referred to as '*reference*'. Often used in linked lists. well as the memory location of the next item in the list. Not al the pointer construct.

User-defined data types

You can also declare your own type. These are called *user-defined* types. The *language-defined* types to make coding more efficient.

Enumerated

An *enumerated* data type is one that is in the form of a list. This could be a li months in a year. Often these data types will be used for comparison; for ins given football player to see whether they are in the list of players in a team. straightforward when using an enumerated data structure. A problem with th the source code of the program and as a result cannot be changed once the

Sub-range

A *sub-range* data type defines a subset of elements from an enumerated data hierarchy of structure exists. For example, in the football team example you goalkeepers, defenders, midfielders and strikers.

Sets

A *set* is a structural data type and is the same as the mathematical idea of se as being a member of a set. For example, given the numbers from 1 to 100 y numbers. Once this set had been defined you could use it to find the odd nu

```
Numbers1to100 = 1, 2… 100
evenNumbers = 2, 4... 100
oddNumbers = Numbers1to100 - evenNumbers
```

Arrays

An array is a data structure that can be used to hold elements of data of the retrieved later in the program's execution. The simplest type of array is a one have a given length, but no depth. You need to be able to search the array us the index. Arrays are *indexed* from the integer 0 to the user-defined length. A array can be seen below.

It is also possible to have an array of multiple dimensions, most commonly t dimension is the number of directions you can index information from, so in go *across* or *up and down*. For example, you could have a two-dimensional ar access them using their indexes to retrieve the information.

An example of use of a simple one-dimensional array containing integers wo be sorted.

Pseudo	.NET	C#

Numbers[100]
Numbers[56] ← 72
	Dim Numbers[99] As Integer
	Numbers[56] = 72
		Int[ ] Numbers
		Int[99];
		Numbers[56] = 7

Interpretation of an array called Numbers – note index 56 has been assigned the

0	1	2	3	4	…	55
55	102	11	87	65	…	61

Records

A record is a structural data type and is one that can provide a structure wit were required to save the name, house number and postcode of 100 custom used in order to store these records for each customer as a single element i

This way each element of the array would contain a single set of details, i.e.:

Customer[72] =	Name: Fred Bloggs
	Housenumber: 3
	Postcode: BS10 5BY

Records or structured data types are defined by the user by building a declar declaring a variable of that structure type. The program can then reference t notation.

Pseudocode

```
Declare recStructure as record
name: string
houseNum: integer
postcode: string

customers[100] as recStructure
```

```
Public Structure r
Public name As St
Public houseNum A
Public postcode A
End Structure

Dim Customers[100

Customers[72].nam
Customers[72].hou
Customers[72].pos
```

C#

```
Struct recStructure
{
Public string name
Public int housenum
Public string postcode
}

recStructure[] customers = new recStructure[100];

Customers[72].name = "Fred Bloggs"
Customers[72].houseNum = 3
Customers[72].postcode = "BS10 5BY"
```

```
type
  recStructure = r
    name:string;
    housenum:inte
    postcode:stri
  end;

var customers: ar

Customers[72].nam
Customers[72].hou
Customers[72].pos
```

Questions: Data Types

1 What data types would best fit the following?

a) Welcome to [illegible] (1 mark)
b) FALSE (1 mark)
c) [illegible]9 (1 mark)
d) 9001 (1 mark)
e) 4 (1 mark)
f) 17 Oct 1992, 3:44AM (

2 A bank is creating a new system that deals with the accounts of their cu Identify suitable data types for the following variables and give a reason

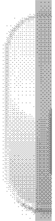
a) firstName (1 mark)
b) accountBalance (1 mark)
c) gender (1 mark)
d) hasOverdraft (1 mark)
e) dateOfBirth (1 mark)
f) sortCode (1 mark)

Variables and constants – explanation, declaration and use

Variables are parts of the program that allow it to change and perform *comp* output from a program would never change, as the values would be fixed as fixed-value variables; where variables can change at run-time, constants are source code. Both are *mnemonics* for a location in memory that need to be de

Language type and declaration

How you declare a variable (or constant) depends largely on what language y govern how the language performs *'type checking'* – the act of preventing ty action is performed on a construct, i.e. attempting to divide a string. In the e of declaring variables in two *strongly typed* languages. In dynamically typed la not declare a variable then use it; what actions can be performed on the vari compilation.

	Variable 	
VB.NET	`Dim variableName As dataType`	`Const consta`
C#	`dataType variableName;`	`Const dataTy`

Programming conventions and standards

As well as the rules of a programming language, there are also conventions a throughout your programming career. While there are universal naming conv (meaningful identifiers, no spaces, no special characters), there are also som

Indenting your code increases readability and allows the human eye to trace good practice to try to indent your code, even when writing pseudo language *languages, i.e. Python, use indentation to define how the program is run by defin than using keywords.*

Naming conventions are another technique for improving readability of your c techniques; these are called *'camelCase'* and *'PascalCase'*. PascalCase is where your identifier is upper-case while the rest of the word is lower-case; this is such as classes, subroutines and protocols. camelCase is where the first lette and the subsequent first letters are upper-case; this is used for all other stru

Did you know?!

Although these programming concepts are not enforced by any programming lang bad practice as even these simple ideas vastly improve readability. In fact, without object-oriented programs can become very hard to read and understand.

Use of assignment

Assignment is one of the most fundamental operators as it allows you to cha time. The way in which it is carried out is given the form *construct = expressic expression to construct.*

Pseudo

```
petrolCost ← 65.0
carHire ← 125.0

totalCost ← carHire +
petrolCost
```

VB.NET

```
Dim petrolCost As Double = 65.0
Dim carHire As Double
Dim totalCost As Double

carHire = 125.0
totalCost = carHire + petrolCost
```

Python

```
petrolCost = 65.0
carHire = 125.0

totalCost = petrolCost + carHire
```

Pa

```
var petrolCost: rea
    carHire: real;
    totalCost: real

totalCost:= petrolC
```

Note: in the VB.NET, Pascal and C# examples it would be equally accurate to ass declared it, as with petrolCost.

Iteration

It is quite common in programming to want to perform a certain task a fixed number of times or until a condition is met. Although it is possible to write out the code that many times, it is a bit cumbersome and rather impractical. To combat this, programmers can use the *loop* constructs.

FOR loops

The first loop you will learn about is the *FOR* loop. The FOR loop will run a s values it is given upon initialisation. The loop will use a variable to count the a limit. For example, to check through a list of test scores to see how many s 90% the loop would look at each student's score, calculate their percentage

Pseudo

```
FOR i ← 1 TO 10
    OUTPUT "i "
ENDFOR
```

VB.NET

```
For i = 1 To 10
   Console.Write(i & " ")
Next
End For
```

Python

```
For i in xrange (1, 10) # using 'xrange' to
generate a dynamic for loop
    Print i
```

```
For i:= 1 to 10 do
Begin
  Writeln(i);
End;
```

It is worth noting in the above example that in C# the FOR loop will automatical i = 10) so it doesn't require an 'End For'. In the Python code, 'xrange' is use of outputs dynamically so that, should the loop exit early, it uses much less mem

WHILE loops

The WHILE loop is a very important basic structure. The syntax begins with a co become the limit of the loop's run-cycle; once the condition is met the loop will is already met when the loop is called the code won't be run at all; it is only whi will run. An example of this can be seen when reading text from a file (*see 2.1*) v WHILE loop to condition the reader to continue reading while the end of the file

Pseudo

```
Var ← 0

WHILE var <= 5
   OUTPUT var
   Var ← var + 1
ENDWHILE
```

VB.NET

```
Dim i As Integer = 0

While I <= 5
   Console.Writeline(i)
    i = i + 1
End While
```

Python

```
i = 0 # assign the count before it can be used

While i <= 5
   Print i
i = i + 1 # if i has not been assigned it will
not compile correctly.
```

```
Var i:integer;

i:=0;
While i<= 10 do
Begin
  Writeln(i);
  i:=i+1;
End;
```

In some instances you might be unsure as to whether you should use a FOR
Sometimes it is clear that the loop should execute a finite, known amount of
should be used (*definite iteration*), but occasionally a loop might only need to
condition has been met, such as a variable changing to a specific value, in w
more appropriate (*indefinite iteration*) – the only risk is that the condition wil
exit the loop!

DO UNTIL loops

Also known as the *REPEAT UNTIL* loop, this iterative technique uses the same
except that the condition is evaluated at the end of the block. This means th
be executed at least once before it is evaluated against the control conditio
condition is true; in this case it will continue to loop until the variable i has

Pseudo

```
var ← 0

REPEAT
   OUTPUT var
   var ← var + 1
UNTIL var >= 5
```

VB.NET

```
Dim i As Integer

Do
   Console.WriteLine(i)
   i = i + 1
Loop Until i >= 5
```

C#

```
int i = 0;

Do
   Console.Write
   i = i + 1;
While (i >= 5)
```

Although there isn't a built-in *DO UNTIL* loop for Python, there is a way you c
requires a little bit of ingenuity. By combining a *WHILE* loop with an *IF* state
command, you can create the iteration yourself.

Python

Syntax

```
while True:
    do_something()
    if condition():
       break
```

```
while true:
         Print i
         i = i + 1
         if i >= 5
```

Breaking out of a loop

If you are using a FOR loop to iterate through a process and you are using a *for example*) then it may also be useful to be able to 'break' the loop if the co computer will continue to iterate through the loop until it reaches the end. Y command and it is placed after your condition variable is met. However, whe to ensure the readability of code. Multiple breaks out of loops may be down ask the question of whether a flag, better logic or a different kind of loop is

GoTo

It is possible to use *GO TO* to create loops and in other situations, such as br would do well to avoid using them. GO TO loops produce 'sloppy' and unreli wherever possible; in fact, the only time GO TO should be used is to cater fo the error reporting code) or when *patching* code in post-release updates.

Selection

Selection is a construct that is used as a control mechanism. A control state set of values and determines the outcome. Examples of selection are the IF s the CASE select statement. A summary example can be found below.

IF selection

The common control flow statement is the *IF* statement. It is carried out by a the condition is TRUE then one portion is code is run. If the condition is FAL

ELSE IF selection

Similarly to the IF statement, a condition is assessed for its value. However, i statement is carried out by the program. Look at the following example of th *the break command used to exit the loop once a condition has been met. The sa different kind of loop and altered conditions to test whether the number has rea 'popty ping'.*

Pseudo

```
For i ← 1 to 15
    a ← i MOD 2
    b ← i MOD 3
    If a AND b ← 0 Then      # if mod division of both is zero
        OUTPUT "Popty Ping!"
    Exit FOR                 #exit after first found
    Else if a ← 0 Then      # else if only a's mod division is zero
        OUTPUT "Pop!"
    Else if b ← 0 Then      # else if only b's mod division is zero
        OUTPUT "Ping!"
    Else
        OUTPUT i            # otherwise print i
    End IF
End For
```

VB.NET

```
Dim a As Integer
Dim b As Integer

For i = 1 to 10
    a = i mod 2
    b = i mod 3

    If (a = 0) And (b = 0) Then
        Console.WriteLine("Popty Ping!")
```

```
For (int i = 1; i <=1
{
    int a = i % 2;
    int b = i % 3;

    if ((a == 0) && (
    {   Console.Writ
        Break;   } //
        Exit For
    ElseIf a = 0 Then
        Console.WriteLine("Pop!")
    ElseIf b = 0 Then
        Console.WriteLine("Ping!")
    End If
Next
```

```
    else if (a == 0)
    {       Console.
    else if (b == 0)
    {       Console.
    else
    {       Console.
}
```

Python

```
i = 0

for i in range(16):
    a = i % 2
    b = i % 3

    if a == 0 and b == 0:
        print('popty ping!')
    elif a == 0:
        print('pop!')
    elif b == 0:
        print('ping!')
```

```
Var a:integer;
    B:integer;

begin
for i:=1 to 15 do
  begin
    a:= i mod 2;
    b:= i mod 3;
    if (a=0) and (
       begin
         writeln('
         break;
       end
    else if a = 0
         writeln('
    else if b = 0
         writeln('
  end;
end.
```

CASE selection

Sometimes there are multiple options to be considered, each one with a diffe letter out of the bag. If it is a *C* you will go to the cinema; if it is a *D* will go o out for a run.

CASE selection has the option of an ELSE in the same way as IF selection. Fo bag. If it is a *C* you will go to the cinema; if it is a *D* you will go out for dinne run; otherwise you will stay in and watch television.

The following example shows how IF, ELSE and CASE statements are used.

William is sitting at home and his mother says: 'Can you answer the telephone?

If it is Janice tell her I will call back later.'

```
If caller = "Janice" then
    Message ("Mother will
End if
```

William is sitting at home and his mother says: 'Can you answer the telephone?

If it is Janice pass me the phone, otherwise tell them I will phone back later.'

```
If caller = "Janice" then
    Action ("Pass phone to
Else
    Message ("Mother will
End if
```

William is sitting at home and his mother says: 'Can you answer the telephone?

If it is Janice pass me the phone, if it is Edith tell her I will be ready at 12, if it is Alfred tell him the time his cake is ready to collect, otherwise tell them I will phone back later.'

```
Select Case caller
    Case caller = "Janice"
        Action ("Pass phon
    Case caller = "Edith"
        Message ("Mother w
    Case caller = "Alfred"
        Message ("Your cak
    Case else
        Message ("Mother w
End Select
```

Subroutines – procedures and functions

Subroutines are either *functions*, which return a value, or *procedures*, which m
Functions *must* be part of an expression but subroutines can also act as state
a statement. Each is given an identifier and a list of parameters which are us
square-root calculator.

Pseudo

```
"Enter an integer: "
a ← INPUT

OUTPUT ← squareroot (a)
```

```
Dim a As Integer

Console.Write("Er
a = Console.ReadL
Console.WriteLine
```

C#

```
Console.Write ("Enter an integer: ")

Int a = Console.ReadLine( ) ;
Console.WriteLine (Math.sqrt(a)) ;
```

```
a = input('Enter
# input is the ke
keyboard

Print Math.sqrt(a
```

Pascal/Delphi

```
Var a:integer;

Begin
  Writeln('enter an integer');
  Readln(a);
  Writeln(sqrt(a):0:3);
End.
```

Nested statements

Nested statements are when you have one set of statements *inside* another s
elaborate further on the example above: pick a letter out of the bag. If it is a
a *D* you will go out for dinner (if it is raining you will drive, otherwise you wi
for a run; otherwise you will stay in and watch television.

Pseudo

```
Select Case letter
   Case letter = "C"
Action ("Go to cinema")
   Case letter = "D"
   If raining then
      Action ("Drive to restaurant")
   Else
      Action ("Walk to restaurant")
   End if
   Case letter = "R"
Action ("Go for run")
   Case else
Action ("Stay in watch tv")
End Select
```

```
Dim letter As Char
Dim raining As Char
Letter.ToUpper( )
Raining.ToUpper( )

Console.Write("Enter a le
Select Case letter
   Case letter = 'C'
      Console.WriteLine("C
   Case letter = 'D'
      Console.WriteLine("I
      raining = Console.Re
      IF raining = 'Y' The
         Console.WriteLine
      Else if raining = 'N
         Console.WriteLine
      Else
         Console.WriteLine
      End If
   Case letter = 'R'
      Console.WriteLine("C
   Case else
      Console.WriteLine("S
End Case
```

C#

```
Console.Write("Enter a letter: ");
Char letter = Console.ReadKey( );
Letter.ToUpper( );

Switch (letter)
{
   Case 'C':
      Console.WriteLine("Go to cinema");

   Case 'D':
      Console.WriteLine("Is it raining? (Y/N) ");
      Char raining = Console.ReadKey( );
      Raining.ToUpper( );
      If (raining == 'Y')
         { Console.WriteLine("Drive to restaurant"); }
}
      Else if (raining == 'N')
         { Console.WriteLine("Walk to restaurant"); }
      Else
         { Console.WriteLine("Incorrect input"); }

   Case 'R':
      Console.WriteLine("Go for run");

   Case else:
      Console.WriteLine("Stay and watch tv");
}
```

```
Writeln('Enter
readln(letter)

case uppercase
   'C':writeln
   'D': begin
      writeln(
      readln(r
      If upper
         Write
      Else if
         Write
      Else
         Write
   end;
   'R': writel
   else
   writeln('St
end;
```

Note the use of the '.ToUpper' command. In the Unicode and ASCII character sets upper-case and lower-case form. The command converts the input to upper case t CASE select. Python does not have a native 'switch-case' function built in, Pascal/

Identifiers

Identifiers are the unique names given to elements such as variables and routines so that they can be identified. For this reason it is important that th are meaningful and relevant to the program, so that the program can be understood (potentially by other people than the original programmer).

Good use of identifiers is particularly important in complex programs which use a large number of variables and routines.

Questions: Programming Concepts

1 **Study the following pseudocode. It takes an array of results for a singl total and stores it in the variable score. For each line (using the line let of programming statement it is. (8 marks)**

```
    Procedure totalScore
a)  Score = New Integer
b)  Result = New Array
c)  Pass = New Boolean
d)  Score = 0
e)  Results = [3,7,5,7,3,6,8,4,2]
f)  PassBoundary = 30
g)  While (currentElement > maxElement)
h)      Score = Score + CurrentElement
    End While
```

2 **Complete the code by writing a nested CASE select in an IF statement. whether the score is greater than the pass boundary; if TRUE begin the to pass. The CASE select should calculate the student's grade by deduc score and output their grade. The grade boundaries are A=30+, B=20-**

ARITHMETIC OPERATIONS

Arithmetic operators

Arithmetic operators are basic functions you use when doing mathematics. The following table shows these arithmetic operators.

Operator	Meaning
=	Assignment
+	Addition
-	Subtraction
*	Multiplication
/	Division
^ or ↑ (command based)	Power

Note: you should never divide by zero, unless you want your program to give an

Modular arithmetic (MOD and DIV)

You know that 9 divided by 2 is **4.5**. You also know that 9 divided by 2 is **4** r

You can work out these three numbers using /, MOD and DIV, i.e.:

```
a = 9 / 2;
b = 9 DIV 2;
c = 9 MOD 2;
```

When these three lines of code above are run, *a* is set to **4.5**, *b* is set to **4** an

You use modular arithmetic in everyday life without even thinking about it; f how many days 50 hours is, you would work it out to be 2 days and 2 hours. work this out, it would look like this:

```
wholedays = 50 DIV 24;
hoursleft = 50 MOD 24;
```

Brackets

Does 3 + 2 × 5 equal 13 or 25? Most languages follow BODMAS (brackets, or addition, subtraction) as on a calculator which means it will do 3 + (2 × 5) = 1 brackets anyway to ensure that the calculation is carried out in the order yo not slow down your program, but it will make it easier to understand the code.

Rounding

When using float/decimal numbers you can round a value to a certain numbe 'round' function. The examples below illustrate the syntax of the round state four decimal places (~3.1416).

Pseudo

```
Var ← 3.14159
Var2 ← round(var, 4)
OUPUT Var2
```

```
Dim myPi As Doubl
Dim a As Integer

a = Math.Round(pi
Console.WriteLine
```

C#

```
Double myPi = 3.14159;
Int a = Math.Round(pi, 4);

Console.WriteLine(a);
```

```
myPi = 3.14159

a = round(pi, 4)
print a
```

Pascal/Delphi

```
const myPi:real = 3.14159;

begin
   writeln(myPi:0:4);
end.
```

Note: in the Pascal being displayed to f decimal places the

Truncation

Like with rounding, truncation works on float/double numbers to remove valu is used mainly in formatting when high precision is needed for operations but value. For example, the number pi is an irrational number which has no end di even been recited from memory to over 40,000 digits by a man in the UK. To u truncate it to a manageable value without losing too much accuracy. In the fol

Pseudo

```
Var ← 3.14159
Var2 ← Truncate(var, 3)
OUPUT Var2
```

```
Dim myPi As Doubl
Dim a As Integer

a = Math.Truncate
Console.WriteLine
```

C#

```
Double myPi = 3.14159;
Int a = Math.Truncate(pi, 3);

Console.WriteLine(a);
```

```
myPi 3.14159
a = Math.trunc(pi

print a
```

Pascal/Delphi

```
const myPi:real = 3.14159;

begin
   writeln(trunc(myPi*1000)/1000:0:3);

// There is no native truncate down command in
Pascal so the above finds the number multiplied
by 1000 to gain 3 decimal places truncated.
```

Note: 'myPi' has bee because pi is a reser on other keywords c p.4.

Questions: Arithmetic Operations

1 Answer the following:

a) 17 DIV 8 (1 mark)

b) 90 MOD 16 (1 mark)

c) ((16 DIV 2) * (6 MOD 4)) (1 mark)

d) 26 MOD 2 (1 mark)

2 Write the pseudocode that can take two integer values and outputs w divisible by each other without any remainder. (2 marks)

RELATIONAL OPERATIONS

Relational operations are the basis of making choices in mathematics. They a to make the decision based on the situation. The following table contains th

Operator	Meaning
= or ==	Equal to
<> or !=	Not equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

Questions: Relational Operations

1 Answer the following (true or false):

a) 7 < 4 (1 mark)

b) 4 > 1 (1 mark)

c) 3.14159 != 3 (1 mark)

2 Write the pseudocode that can take two integer values and outputs wh divisible by each other without any remainder. (2 marks)

BOOLEAN OPERATIONS

In statements involving relational operations and conditions the following B

Operator	Result
Expression AND Expression	AND only returns TRUE if *both* expressions are true.
Expression OR Expression	*OR* returns TRUE if either expression is true, and FALSE if neither is true.
NOT Expression	*NOT* returns the opposite of the expression, i.e. TRUE if it is false and FALSE if it is true.
Expression XOR Expression	XOR returns TRUE when the expressions are different and FALSE if they are the same.

Questions: Boolean Operations

1 What would be the output for the following?

a) 'a XOR b' where *a* and *c* are *true* (1 mark)

b) 'c NOT d' where *c* and *d* are *false* (1 mark)

c) 'e AND g' where *e* is true and *g* is *false* (1 mark)

CONSTANTS AND VARIABLES

As stated earlier, all declarations are actually a shorthand representation of t memory; this allocation is then given an identifier; this is the variable/consta that require the variable name and data type to be declared are called *strong* the computer trying to perform inoperable actions to the value.

Variables and constants are very similar. A constant is effectively the same a be changed at run-time. This means variables can be assigned a value in the overwritten by a routine, but every time the program is restarted the original the value cannot be changed from the value in the source code.

An example can be seen every time you pay for something in a store or perfo total price and the Value Added Tax (VAT). The total price for the transaction be changed at run-time as more items are added to a transaction, whereas th of the transaction cannot be changed by anyone, other than a manager or ad for all intents and purposes a constant fixed rate. The pseudocode for the e

```
varNet ← 0
constVATRate ← 1.2
… # run-time code for adding price of item to net total
Total ← varNet * constVATRate
```

Did you know?!

The use of mnemonics (naming variables) hasn't always been a feature of program computers often required programmers to use the literal memory locations for com memory addresses in RAM *(before the introduction of offsetting!) every time they w*

STRING-HANDLING OPERATIONS

Strings are a series of characters. For example, a word or even an essay can be *Casting* is the process of converting between data types. Often you have inform you want to manipulate and will *cast* to another data type to perform an operat

Some of the most common string-handling operations are shown below:

Function	Description	
Length	Returns the length of *a*	`Length("Com`
Position(a, b)	Returns the position *a* in *b*, inclusive of special characters	`Position(""`
Substring(a, b)	Looks for string *a* within string *b* and returns TRUE if is found	`substring("` `substring("`
Concatenate(a, b)	Joins string *b* on to the end of string *a*	`Concatenate`

Alternatively to concatenation, you can also use string addition which functi
concatenation and can be seen below.

Pseudo

```
exampleString ← 'Ex' + 'Ample'
exampleString ← var1 + var2
```

```
exampleString = '
exampleString = va
```

C#

```
exampleString = 'ex' + 'ample';
exampleString = var1 + var2;
```

```
exampleString = '
exampleString = va
```

Pascal/Delphi

```
EXAMPLESTRING := 'EX' +'AMPLE';
EXAMPLESTRING := VAR1 + VAR2;
```

Character and character code conversions

To go from a character to a character code you can use the following:

Pseudo

```
varAscChar ← 'a'
varCharCode ← ConvertToAscCode(varAscChar)
```

```
Dim ascChar As Ch
Dim charCode As I
```

C#

```
char ascChar = 'a';
int charCode = (int) ascChar;
```

```
ascChar = 'a'
charCode = Ord(as
```

Pascal/Delphi

```
ascChar:='a';
charcode:=ord(ascChar);
```

To go from character code to the character representation you can use the f

Pseudo

```
varCharCode ← 97
varAscChar ← ConvertToChar(varCharCode)
```

```
Dim charCode As I
Dim ascChar = chr
```

C#

```
int charCode = 97;
char ascChar = (char) charCode
```

```
charCode = 97
ascChar = chr(cha
```

Pascal/Delphi

```
charCode:=97;
ascChar:= char(charCode);
```

Note: in pseudocode there is no defined method of approaching this problem. If a
try to make it perfectly clear what your code is doing. Also, you are using a <u>cast</u> i

String conversion operations

You can also convert between data types; this is especially useful when read

Conversion	VB.NET	C#	Pyt
String to integer	`CInt(string)`	`Convert.ToInt32(string)`	`Int (st`
String to float	`CDbl(string)`	`Convert.ToDouble(string)`	`Float (`
Integer to string	`CStr(integer)`	`Convert.ToString(integer)`	`Str (in`
Float to string	`CStr(double)`	`Convert.ToString(double)`	`Str (fl`
Date/time to string	`CStr(date/time)`	`Convert.ToString(Date/Time)`	–
String to date/time	`CDate(string)`	`Convert.ToDateTime(String)`	–

Note: '--' has been used because DateTime is not a native data type in Python. If yo the 'datetime' class. Remember to convert before you attempt to perform operations programming languages all input is read as a string and must be converted be

Questions: String Handling

1 What would be the results of the following built-in functions? (4 mark

a) `Length("Almost Impossible To Guess")`

b) `Round(656.3357, 2)`

c) `Length(Concat`

d) `Position('l',`

2 Write code that asks the user for a number and prints the square of th

RANDOM NUMBER GENERATION

Another built-in function is the *random number generator*. Given a minimum a a new random number when the function is called.

VB.NET	C#	Python

`Dim newRand as New Random`
`Dim x As Integer`
 	
`x = newRand.Next(1,10)`
	`Random newRand = new Random();`
	`Int x = newRand.Next(1,10);`
		`From random im randint`
		
		`x = Randint(1,`

In each case the syntax is very similar; note that in C# you call on the Random cl

Are the numbers actually *random*?

Often computers use a seed value to generate a sequence of what appears to be a random sequence. Programmers can set this seed value in many languages to be able to replicate the same sequence. This can be useful for testing situations. What you must remember about *randomness* is that it is hard for a logical computer to produce something that is truly random. This is because no matter how the random numbers are produced, the computer must rely on source code produced by a human to generate the numbers and it is impossible for humans not to introduce a portion of bias into a system. *Randomly generated* numbers are therefore given the title '*pseudo random* numbers'. These are numbers that appear random but have an underlying level of bias in how they've been produced.

EXCEPTION HANDLING

Exception handling can be used to help your program recover from run-time or divisions by zero. If an error occurs in the *Try-Catch* block it allows for the exception without the program crashing or losing any data.

Consider you'd like to check whether the user has input an integer:

Pseudocode

```
"Enter an integer"
Var ← Input
Try #attempt conversion
     Var ← Convert to Integer
Catch #output error
     "Input incorrect"
End Try #end try
```

```
Dim newInput As String
Dim convertedInt As Integer

Console.Write("Enter an integer")
newInput = Console.ReadLine( )

Try
    convertedInt = CInt(newInput)
Catch (ex as Exception)
    Console.WriteLine("Input incorrect")
End Try
```

```
Console.Write("Ente
String newInput = C

Try
{
   Int convertedInt
}
   Catch (Exception
{
   Console.WriteLin
}
```

Python

```
newInput = input('Enter an integer')

try:
    int(newInput)
except: # catches all errors
    print('Input incorrect')
```

P

```
writeln('Enter a nu
readln(newInput);

try
  numEntered:=StrTo
except
  writeln('Input in
end;
```

Learning to use these error handling methods now will save you a lot of stre own software during your project. The error handling methods you've been i help prevent almost any error that could occur in a system.

When your programming skills begin to develop beyond the basics you will l technologies; for example, you will be able to produce code that can interac renowned for being a minefield for amateur programmers, but with the corre find it more comprehensible if something does go wrong.

Task: Exception handling

Write a program that asks the user for two integers that will be divided; build against divisions by zero. If an exception occurs it prints the result as zero an

SUBROUTINES

You have already looked at built-in functions such as *square root*, but now yo these structures. Subroutines are blocks of code which are independent of ar have their own variables, and they can be passed data using parameters to p value. If a subroutine is called as part of an expression and returns a value it *procedure* is a routine that is called as a statement which executes a section amount of results including none.

For example, you can use a procedure to open a file and a function to read th An example of a function can be seen below; this function returns the larges

Pseudo

```
x = 17.0
y = 29.0

OUTPUT "The largest number is " & max(x,y)

# the returned value of the function is added to
the end of the output

FUNCTION max (a, b)
IF a > b THEN
    max = a
ELSE
    max = b
END IF
Return Max
END FUNCTION
```

```
Dim x As Double =
Dim y As Double =

Console.WriteLine(
max(x, y)

Function max (a As
IF a > b THEN
    max = a
ELSE
    max = b
End IF
Return max
END FUNCTION
```

C#

```
Double x = 17.0;
Double y = 29.0;

Console.WriteLine("The largest number is " +
max(x, y);

Static Real max (Real a, Real b)
{
    Int Max;
 {
    IF (a > b)
    {
        Max = a;
    }
    ELSE
    {
        Max = b;
    }
    Return Max;
}
```

```
var x:real;
    y:real;

function max(a,b:r
begin
    If a > b then
        max := a
    Else
        max := b;
end;

begin
x := 17.0;
y := 29.0;

writeln('The larges
readln;
end.
```

Python

```
def max(a,b):
    max = 0
    if a > b:
        max = a
    else:
        max = b

    return max

x = 17.0
y = 29.0

print('The largest number is ' + str(max(x, y)))
```

For an example of a read from / write to *2.1)*; for more on fu functions, *see p.20.*

For more informatio subroutines *also see*

Did you know?!

To call a function or a procedure you simply type the identifier and pass any param

Notice in the VB.NET code that a and b are both defined as real numbers and that real; this has been done to show that you can cast within functions.

Look at the declaration for the function in the C# code. The function return type is means that any variable you want to return must be returned as an integer. In this string and the function will still work properly.

In the Pascal/Delphi version, as soon as the function name is declared as a value it In simple code as above the multi-exit method is fine. In more complex routines it is (see p.22) and then allocate at the end of the routine. Just like the 'break' comman which method to use is based upon ease of readability.

Questions on subroutines

1 Study the following code and describe in words how the function perf

```
Function newSubroutine (Integer x)
Answer ← x
FOR var ← 1 to x
    Answer ← Answer * (x-var)
END FOR
RETURN Answer
```

2 What would be the output if the subroutine was passed the value '3'?

Procedures and functions as building blocks

Procedures and functions divide a program into building blocks. These basic to produce very complex programs and potentially reused in other projects. easily readable and more comprehensible, but also more space-efficient. Eff any large-scale programming project.

Advantages of procedures and functions

There are a number of advantages of using procedures and functions:

1. Reduced amount of repeated code. For instance, if you know that du required to perform the operation (a+b)*c very regularly then it make accepts the variables a, b and c as inputs and returns the value (a+b)
2. Once a function has been written and is known to be correct, you kn you can concentrate on the rest of the program.
3. More than one programmer can work on the project, each on differer
4. Once a function is finished the variables are deleted from memory, s needed to run the program.
5. Some quicker methods of sorting data use recursive functions.

PARAMETERS OF SUBROUTINES

Many functions require the calling program to pass information to them, whi information, passed in the form of a variable, is called a *parameter* (or *argum* number of parameters. Parameters can also be a method of returning data. P inside the brackets after a function name:

Subroutine call	

`myBase ← 6`
`myPower ← 2`
 	
`OUTPUT (myExp(myBase, myPower)`
	`Function myExp (te`
	`Counter ← tempPow`
	`Answer ← 1`
	
	`While counter > 0`
	`    Answer ← ansv`
	`    Counter ← cou`
	`End While`
	`Return Answer`

The words shown in bold above are the parameters for the function called myExp a specified number of parameters (two, in this case), and most computer language parameters to be given (unless they are specified as optional).

Procedures and functions with interfaces

When creating large programs it is important to try to minimise the number the code. Using modules and subroutines allows you to declare local variabl those code blocks. The single input and output interface ensures that the co and ends at the same place, making the code more intuitive. This restriction it much easier to debug than an unstructured approach. Consequently this m be responsible for a single task.

RETURNING A VALUE FROM A SUBROUTINE

We've already explored how you saw that you can pass data to a subroutine to be used within that block. The data you pass is given a new declaration under a new temporary identifier that can be used within the block. So what if you want to access the data within a subroutine?

Some languages allow the transfer of variables by *reference* rather than by *value*. The function that worked out the maximum of two numbers u the result passed by the name of the function. However, it is also possible to reference which will automatically alter the value in the routine that called i global variables or you may have to create a function to return it to the mai

Consider a function that is used to calculate the area of a square. In the mai assign the returned value to, this *must* be of the same data type as the retur

Pseudo

```
"How long are the sides of your square?"
a ← INPUT
answer = squareCalc(a)

FUNCTION squareCalc (sideLength)
tempArea ← sideLength * sideLength
Return var
END FUNCTION
```

```
def squareCalc (s
    tempArea = 0
    tempArea = si
    return tempAr

a = input('How lo
square?')

answer = str(squa

// note that Pyth
to be considered 
```

VB.NET (passing by Value)

```
Function squareCalc (sideLength As Double)
    Dim tempArea As Double
    tempArea = sideLength * sideLength
    Return tempArea
End Function

Dim a As Double
Dim answer As Double

Console.WriteLine("How long are the sides of your
square?")

a = Convert.ToDouble(Console.ReadLine( ))
Answer = squareCalc(a)
```

VB.NET

```
Sub squareCalc (by
    Dim tempArea A
    Answer = answ
End Sub

Dim answer As Dou

Console.WriteLine
square?")

answer = Convert.
squareCalc(byRef 

'the variable a a
answer is changed
```

C# (passing by Value)

```
Console.WriteLine("How long are the sides of your
square?");
Double a = Convert.ToDouble(Console.ReadLine( ));
Double answer = squareCalc(a);
Static Double squareCalc(Double sideLength)
{
    Double tempArea;
    tempArea = sideLength * sideLength;
    Return tempArea;
}

// here a separate variable is used (answer) to
store the result of using a passed as a value
```

C# (p

```
Console.WriteLine
square?");
Double answer =
Convert.ToDouble(
squareCalc(ref an

Static Double squ
{
    answer = answ
}

// the variable a
answer is changed
```

Pascal/Delphi (passing by value)

```
var x:real;

function squareCalc(a:real):real;
begin
   squareCalc:= a*a;
end;

begin
write('How long are the sides of your square?');
readln(x);
writeln('The area is ', squareCalc(x):0:2);
readln;
end.

// in the above function the variable x remains
as the length and the function returns the area
```

Pascal/Del

```
var x:real;

Procedure squareC
begin
   a:= a*a;
end;

begin
write('How long a
readln(x);
squareCalc(x);
writeln('The area
readln;
end.
// in this routin
through the proce
procedure automat
when it changes t
to x
```

LOCAL VARIABLES

When you begin to use subroutines you'll discover it becomes more efficient increases *modularity* and decreases the volume of memory used. Modularity i has been decomposed into individual problems; the aim is to have a subrout process in a solution. Local variables cannot be called by anything outside of need to be passed as a parameter to the subroutine and any value from a loc the main body. These variables only exist while the subroutine is being exec the main body of code and all memory is reallocated.

There is also the logistics of programming to consider. For example, on large a single team of programmers to complete the entire task so it is broken dov on specific sections. This is to stop different teams trying to use the same ide otherwise cause a problem once the modules are combined.

GLOBAL VARIABLES

When you first start learning to program in a language, most variables will b These are variables that can be called and are operable by all blocks of code simple programs this may seem easier, but consider the following:

- Global variables are assigned memory at run-time; this memory is or closes. If you're building a very large and complex program with mar assigning a lot of system resources to variables that the user may or may choose to create a new item instead of reading from a file – me from a file are made redundant. The end result is your program runni
- As these variables are callable from all blocks they may be called an variable names are similar.

Questions: Procedures, Functions and Variables

1 What is the main difference between a procedure and a function? (1 m

2 What happens if you pass variable values into a subroutine in a differer

3. In terms of memory and modularity, why is it considered bad practice to c

ROLE OF STACK FRAMES IN SUBROUTINE CALLS

Whenever a routine is called, the computer allocates memory in a specialise
'stack'. When you use a function you'll allocate the returned value to a variab
is stored to the stack frame. This is the area where all the routine's paramete
stored until the routine ends, at which point the value is stored to the return
passed back to the main function and the stack frame is removed from the st

Stacks are a very useful data structure and they have many applications in co
calling procedures in programs. For example if a procedure main() calls a pr
procedure getchar() then main() cannot continue until getstring() has finishe
getchar() has finished; i.e.:

getChar()
getString()
main()
empty

Every time a procedure is called by another procedure it is pushed onto the s
which the procedures need to be executed in an efficient way. If you have a
itself indefinitely it will run out of stack space – try it!

RECURSIVE TECHNIQUES

Recursion is the ability that a subroutine has to call on itself to complete its
Recursive solutions can be harder to produce but can often lead to very eleg
recursive solution has two parts: the *recursive* and the *limiter*. The recursive i
another iteration and passes new variable values, whereas the limiter is wha
infinite loop.

Recursive methods act as a loop that calls on itself and runs every line of co
call in the current, and will pass the current result into the next call until the

One of the best examples of recursion is factorials. Factorials are given by th

```
n! = n × (n-1)!
```

This means a factorial is the product of a number times the factorial of the p

```
4! = 4 × 3! = 3 × 2! = 2 × 1!
4! = 4 × 3 × 2 × 1 = 24
```

P[illegible]o

```
Answer [illegible]ct[illegible](4)

FUNCTI[illegible]torial (var)
IF var < 1 THEN # 1! = 1
    Return 1
Else
    Factorial  var * (var – 1)
    # recall the function with new values
END IF
END FUNCTION
```

```
Dim answer As Int
Dim base As Integ

Answer = factoria

Function factoria
    If n <= 1 The
        Return 1
    Else
        Return n
    End If
End Function
```

C#

```
Int base = 4;
Int answer = factorial (base);

Static Int factorial (Int n)
{    If (n <= 1)
     { Return 1 }
Else
     { Return n * factorial (n - 1) }
}
```

```
Base = 4
Answer factorial
 
Def factorial (n)
    If n == 0:
        Return 1
    Else:
        Return n
```

Pascal/Delphi

```
function factorial(base:integer):integer;
begin
  if base<1 then
     factorial:= 1
  else
     factorial:=base*factorial(base-1);
end;

begin
  writeln(factorial(4));
end.
```

In this example the
the ELSE statement
The limiter is the IF
variable is less than
producing an infinit

Task: Recursive Techniques

The Fibonacci sequence is a set of numbers derived from the rule: $F_n = F_{n-1} + F$
number in the sequence is the sum of the previous two numbers of the seque
pattern of numbers that is found frequently throughout nature and even has s
in the design of computer components. For example, 9 times out of 10 the nu
petals found on a newly blossomed flower will be a Fibonacci number, and in
science it is said that any number can be written as the sum of unique Fibona

Using recursion, create a function that produces a list containing a Fibonacc

Questions: Recursive Techniques

Study the following pseudocode and answer the questions below.

```
FUNCTION MyFunction (Sum)
OUTPUT "enter an integer value: "
i ← READ VALUE
WHILE True
IF i * 0 Then
    sum ← sum + i
    IF sum > 100 Then
        RETURN sum
        Exit While
    Else
        MyFunction(sum)
    END IF
END IF
END WHILE
```

1 **What task does this function carry out? (1 mark)**

2 **Does this function enter an infinite loop? Explain your answer. (2 ma**

3 **Describe the use of the stack in the above code. (1 mark)**

1.2 PROGRAMMING PARADIGMS

STRUCTURED PROGRAMMING

In the early days of computing, computers were programmed by writing mac advantage of allowing the programmer to directly control the computer by d addresses and computer operations. To aid programmability the instructions rather than binary. However, machine code is very difficult to read and so ev would struggle to understand a large piece of code without a significant am

This led to the development of assembly code, which replaced the hexadeci with mnemonics which were easier to read and understand. Most assemblers (or variables) to signify memory addresses which could be referenced to a m Assembly language, however, still operated at a very low level and became r programs became larger.

One of the big problems with assembly language is the use of *GO TO* statemen are very difficult for people to follow. Structured programming developed as providing a level of abstraction away from the operations of the computer to for humans to understand.

The structured programming paradigm encompasses procedural programmin programming, and implies the ability to use structures such as IF statements

PROCEDURAL-ORIENTED PROGRAMMING

Procedural programming is a step forward in programming and provides a st program. Programs written in procedural languages are executed line by line designed with a top-down view. From the top-down view, the program is sep written with procedures where each procedure performs a specific task. Prog variables are used to store data which is local to each procedure. Bad progra paradigms, such as using GO TO statements, become unnecessary and are re loops and procedures. Statements are grouped together and form procedures perform a specific task. Having structure also automatically introduces recur

Many solutions can be broken down into a series of operations which can be logical order. However, more complex tasks and data structures lend themse programming methods. Often modern-day programming languages allow the object-based methodology.

OBJECT-ORIENTED PROGRAMMING

As procedural programs became more widespread, people started to notice tha procedures/functions associated with them tended to be grouped together; thi *object-oriented programming (OOP)*. At their most basic level, object-oriented pr concerned with the data for the *objects* you are trying to manipulate rather tha them. However, before you start to use the object-oriented approach there are

Classes

A class is an object definition. For example, a game might have a class 'gobli the goblin, such as name, health, weapon and colour and also the actions the defend, etc. Creating an object from a class invokes the constructor for that c

1. Allocates and initialises the necessary memory
2. Assigns a label to that memory
3. Assigns values to (initialises) various properties as required (e.g. the

Objects

An *object* is an *instantiation* (or an *instance*) of a class. Each object will have the local to that object. For example, a computer can have states (on, off) and beha Objects are created using a *constructor* and a *reference* that has been assigned t

Encapsulation

Encapsulation is where attributes and methods are 'wrapped' together into ob together but their implementation details are *hidden* from one another. The a of processes from other objects and classes is achieved by using the keyword

Encapsulation is applied:

1. If the internal complexity is not needed by other objects and doesn't
2. If you need to prevent changes to an object from external objects

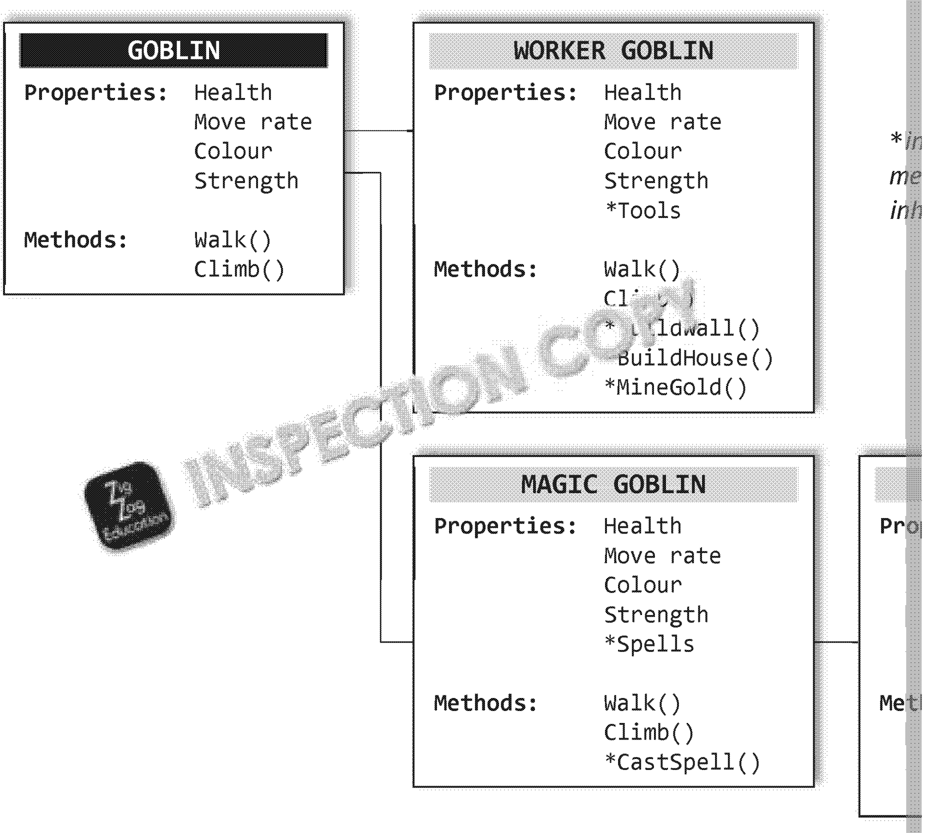
Inheritance

Inheritance is a relationship among classes wherein one class shares the stru another. This is similar to how children *inherit* the attributes of their parents.

Single Inheritance

Once a behaviour or characteristic is defined, all the categories beneath that or characteristic. For example, the class *goblin* might define a goblin as havi movement rate, etc. with methods for walking and climbing. You could then *goblin* or *worker goblin*, where both then have inherited the properties (healt methods (walking and climbing) from their *parent* class (goblin), however eac properties, for example these two subclasses could have methods for casting respectively.

Extending this idea further, subclasses can be extended with further subclas be defined, with extra methods and properties on top of those defined in the

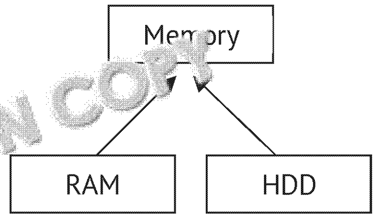


Multiple Inheritance

Multiple inheritance occurs when a class inherits from more than one parent object-oriented language because many base classes can be set up from whi

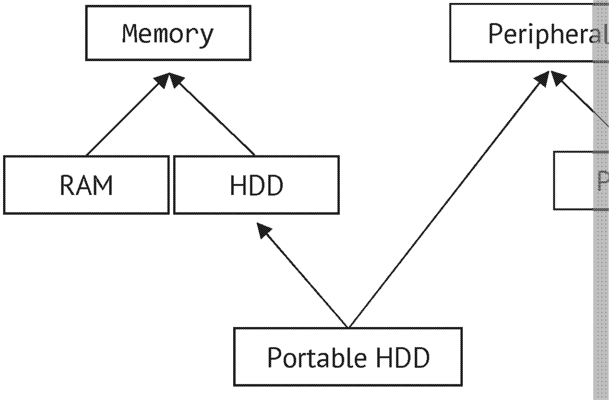
The following example of single inheritance is a very simple one. If an obje single inheritance then this limits the classes we could have; for example we under: memory, output devices, input devices and processing devices.

Hardware – Single Inheritance



However, we have a problem if we want to bring in a new class called, for ex subclass *Portable HDD* will acquire properties and methods from both the cla

Hardware – Multiple Inheritance



When applied to object-oriented programming, the process is about building together with the methods that accompany the data structures.

Aggregation

Aggregation in its everyday sense is similar to how programmers use it; it def share a relationship. In programming, there are two kinds:

1. *Association* is how objects are related without there being an owner t student can be taught by a single tutor and a tutor could teach many association because if either is removed/deleted the other remains; j moves on from a school doesn't mean the teacher is fired.
2. *Composition* is where the whole is defined by the relationship betweer parent object/class then all children objects are removed. Consider a h house is the parent class because it is the container in which all rooms house it means you've destroyed the rooms, but you can change the rc

Polymorphism

Polymorphism refers to a programming language's ability to process objects d type or class. This means that the code itself must be able to redefine metho derived objects. For example, polymorphism would allow a programmer to de methods for any number of derived shape classes.

Method overriding

Method overriding is when you change the base characteristics of a class wit code; this acts as an extension to the class. Take a look at the following exa

OOP: An example

The following code is written in pseudocode but the concepts are still the sa example you will be shown two classes that will be used together (*aggregati* You will generate a very basic object-oriented design to create an object of a and how you can use *constructors* to handle any variations in inputs.

Person.class

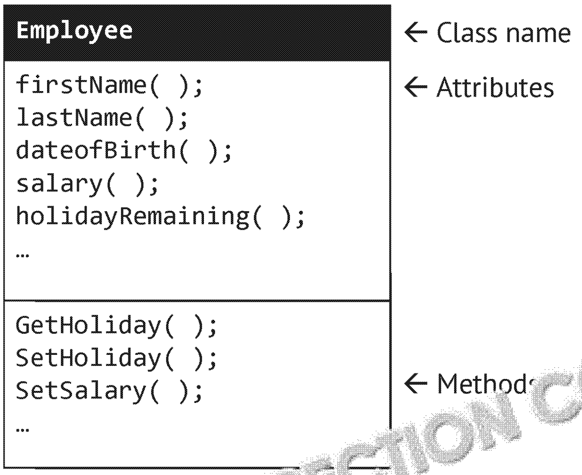
```
CLASS STRUCTURE person
    private age
    private firstName
    private surname

    # Default Constructor
    STRUCTURE Person ( )
        age ← 0
        firstName ← "No name set"
        surname ← ""
    END STRUCTURE

    # Partial Instanced Constructor
    STRUCTURE Person (initialAge, personName)
        age ← initialAge
        firstName ← personName
    END STRUCTURE

    #Fully Instanced Constructor
    STRUCTURE Person (initialAge, personName,
    personSurname)
        age ← initialAge
        firstName ← personName
        surname ← personSurname
    END STRUCTURE

    PROCEDURE GrowOlder ( )
        age++
    END PROCEDURE
END CLASS STRUCTURE
```

P

```
CLASS STRUCTURE Pe
    #Declare new p
    Person somePe
    Person someOt
    Person someSt

    # Using the de
    somePerson ←
    # using the pa
    someOtherPers
    # using the fu
    someStrange ←
    "Smith")

    #Calling a met
    someStrange.G
END CLASS STRUCTU
```

You can use these classes without having to look at all the code behind the *diagrams*: they convey all the information needed to understand how the form the final solution. They are a structural modelling technique used durin the system life cycle (*see Section 13*).

Below is the view of a single entity.

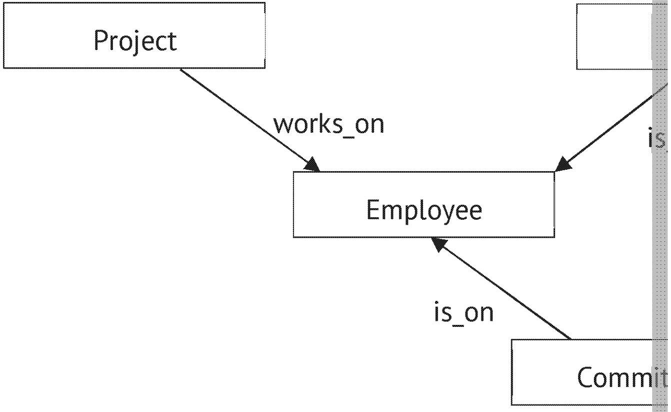


```
Employee
firstName( );
lastName( );
dateofBirth( );
salary( );
holidayRemaining( );
…
GetHoliday( );
SetHoliday( );
SetSalary( );
…
```

If you imagine that t stores the informatio would be fine; howe and so must encomp not all members of s many would be on a cafeteria workers.

You could add anoth ut then not all mem hourly rate. The solu 'SetSalary' to include

The full diagram might look something like the one below. Note the us composition or aggregation. For example, *works_on* is composition.



Advantages and disadvantages of object-oriented design

Advantages	Disadv
▪ Improved software maintainability	▪ Harder to produce efficientl
▪ Improved software stability	▪ Approach is not suited for a
▪ Lower cost of development	▪ Larger program size
▪ Higher-quality software	▪ Slower program execution c
▪ Class code is reusable	▪ Can be difficult to apply ove

Task: Object-oriented Programming

1 Using the PersonGenerator class example, produce similar code that c that contains the first name, surname and bank balance of three bank a

2 Improve on your code and write a subroutine that deposits £10 into an
Hint: *Remember you can't have negative deposits.*

2. Data Structures

Data structures play an important role in computer science. A good understanding of the
particular, their relationship with certain programming techniques, is key to be able to so
explores some structures available and how they are used.

This section covers:

2.1 DATA STRUCTURES AND ABSTRACT DATA TYPES

DATA STRUCTURES

All data types, from the most basic integer to the most complex tree, can be
data types. Each category uses memory in a different way, and compilers mu
to manage the memory accordingly.

- *Strong* types are the standard types, such as integer and character. Th
types, and a fixed amount of memory is defined.
- *Static* types are those which require a fixed amount of memory, such
not included in the pre-defined types. For example, an array in C# is
- *Dynamic* types are those that may be expanded given the limitation c
in the case of files); for example, files and pointers (thus linked lists,

SINGLE- AND MULTI-DIMENSIONAL ARRAYS

The array is one of the most useful and fundamental data structures there ar
matrix of a single data type; a matrix is where you can store data into elemen
retrieve the data from the element by using a unique identifier. Ensure you h
programs using arrays, as you will find that you can simplify many aspects of
implementation (and it will help you if an arrays question comes up in the ex

One-dimensional arrays

Storing data in several different variables becomes tedious and impractical w
amounts of similar data. The solution to this is an array. An *array* is a set of da
consecutively in memory. If you wanted a set of 10 integers called *X* you migh

Pseudo	VB.NET	C#	Pyth

`X [10]`	`int x [9]`	`int [] x = new int [9];`	`x = []`
			`# This create`
			`list of size`

*Note: in pseudocode you reference each term by X[0], X[1], X[2], ... X[9]. The num
known as array subscripts. Some languages would define the array as going from
others define it as '0 to n' (e.g. pseudocode, Pascal/Delphi). Although the latter is
method often makes array manipulation much simpler and is closer to the actual*

Suppose you stored the marks of 10 tests in an array. A common way to visu

81	75	90	64	68	72	69
X[0]	X[1]	X[2]	X[3]	X[4]	X[5]	X[6]

Most programming languages use loops to iterate through a data construct t
you can access the memory location of a result directly if you knew what res

However, in most cases you will need to iterate through an array to find the
look at the following example code which iterates through the array x to fin
the result.

Pseudo

```
OUTPUT "student with top marks: "
For i 1 to 10
    If x[i] > topScore then
        topScore = x[i]#¹
    end if
next

OUTPUT topScore

#¹ sets topScore to value of array index [i]
```

```
Dim topScore As Inte
Dim i As Integer

For I = 0 to 9
    If x[i] > topSco
        topScore = x
    End If
Next

Console.WriteLine(to
```

C#

```
Int topScore = 0;

For (Int I = 0; i = 9; i++)
{
    If (x[i] > topScore)
    {
        topScore = x[i];
    }
}
Console.WriteLine(topScore);
```

```
topScore = 0

For i in range(len(x
    if x[i] > topSco
            topScore

print(topScore)
```

Pascal/Delphi

```
var x:array [1..10] of integer;
  i:integer;
  topscore:integer;

begin
  for i:= 1 to 10 do
  begin
    if x[i]>topscore then topscore:=x[i];
  end;
  writeln(topscore);
end.
```

*Note: Python does not u
and instead uses a list. L
multiple data types; they
flexible in the actions th
the data they can contai
makes them slightly har*

Two-dimensional arrays

In most languages arrays can have more than one *dimension*, as many as 32,
dimension is simply a direction in which you can vary the specification of ele
with two-dimensional arrays you can go along the columns and across the ro

Declaration and use

The declaration is almost the same except we define the length of the array

Pseudo	VB.NET	C#	Python
`X [3, 3]`	`int x [3, 3]`	`Int x[,] = new int [3, 3];`	`x = [[]]*3`

You can then assign values as you would with a single array. This is often us
same data type that can be compared to each other. *Note: Python does not ha
although a 2D array can be emulated through a list of lists.*

For example, in this table you can see the distances between varying cities. *context and do not appear in the array*. See if you can replicate the array belo

	Plymouth	London	Edinbu
Plymouth	0	237	487
London	237	0	413
Edinburgh	487	413	0

Three-dimensional arrays

Take what you already know about arrays and think about how you would m would be able to index data across the rows, down the columns and have a t This can be hard to show on paper, but it is a natural way of storing data that convey. For example, you could store the coordinates of a vector in a single- limited to having no control over direction. You could store the vector coordi but be limited to having no control over depth.

Three-dimensional arrays allow you to store real special dimensional data, a software and modelling software that relies heavily on vectors instead of bit

Questions: Arrays

1 Consider the following single-dimension array and answer the questio

Array RawMarks

(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
31	18	27	32	9	28	11	17	21

a) What value is retrieved by 'RawMarks[6]'? (1 mark)

b) What is the output of the following algorithm? (2 marks)

```
FOR i ← 0 To Length(RawMarks)
    n ← 0
    IF RawMarks[i] > n Then
        n ← RawMarks[i]
    END IF
    OUTPUT n
END FOR
```

2 Study the following two-dimensional array about the distances betwe Solar System and answer the questions below. Planet names have bee

Array PlanetaryDistance	0 (*Mercury*)	1 (*Venus*)	2 (*Ear*
0 (*Mercury*)	0	0.33	0.61
1 (*Venus*)	0.33	0	0.28
2 (*Earth*)	0.61	0.28	0
3 (*Mars*)	1.13	0.80	0

a) How would you access the values for the distance between Merc

b) How would you access the values for the distance between Venus

c) Notice that the indexes PlanetaryDistance[2, 3] and [3, 3] are the How would you change their values so that they are correct? (1 m

FIELDS, RECORDS AND FILES

Very few programs do not utilise files at some part of their execution. Witho effectively starting from the beginning and would require the data input eve programs that are run as new each time tend to have user settings or prefere Therefore the use and understanding of files is a vital part of programming. text/ASCII files and binary files.

Text/ASCII files store the information in text or ASCII character format. If the another program (such as Notepad or Excel) then the file is stored as an ASCI file contains coded data which, without the context of the program, would no a text file might contain the data 'AB123' which has no meaning unless appli advantage of storing information in text files is that they can be created, alte the program. This can be useful in the development of a system to allow the

Binary files are stored as binary code (often in hexadecimal format). These a but are directly compatible with the computer. Binary files tend to be more s definition, which is difficult to convert the data into meaningful informat

Here is [illegible] a text/ASCII *file* of CSV (comma-separated variable) data called *students.csv*. The data is in the format *surname, ID, course, college.*

Note that you call each line of the file a *record*, and you call each part of the record a *field*.

Before reading the file to be used in a program, you may wish to create a data structure to contain the data that is shown below. Once the structure is created you can then apply the I/O from the file to the structure to contain the data.

Pseudo

```
NEW STRUCTURE ← Record # declare the structure

# Declare the variable names and types
    Surname ← string
    ID ← integer
    Course ← Integer
    College ← String
END STRUCTURE
```

```
Structure Record
         Public _
         Public _
         Public _
         Public _
End Structure
```

C#

```
struct Record
{
    public string surname;
    public int ID;
    public int Course;
    public string College;
}
```

```
Class recordStruct
def __init__(self,
         self.sur
         self.id =
         self.cou
         self.col
```

Pascal/Delphi

```
Type
  TStudent = Record
    Surname: string;
    ID : Integer;
    Course : Integer;
    College:string;
  End;
```

Reading from a text file

Using input and output is very different depending on the programming lang to be able to recognise and operate this function as it widely broadens what

For the C-family, in order to read from a text file you need to invoke the help *StreamReader*. Stream readers are a way of interfacing between the contents building, and use a string variable to read the text file line by line, whereas c as Python have a class called *open* that defines what is being used and what

When reading from a file you must remember that all values returned from a includes numbers. In order to use the values being read you may need to cas

Pseudo

```
CONST ← 'C:\MyDocuments\ReadingTestFile.txt'

Line ← Length of one line
filename ← file na

USING StreamReader ← streamreader

WHILE NOT EOF(filename) DO
    Readline(filename, OneLine)
    OUTPUT (OneLine)
End while

Close(myFile)
```

```
Dim FileRea
Const filen
Dim oneLine

fileName =
"C:\MyDocum
FileReader

Do Until Fi
        on
        Co
Loop
FileReader.
Console.Rea
```

C#

```
string lineFromFile;
streamReader reader;
string fileName =
@"C:\MyDocuments\ReadingFileTest.txt"
// '@' is added as an escape char.

reader = new StreamReader(fileName);

While (!reader.EndOFStream)
{
    lineFromFile = reader.ReadLine( );
    Console.WriteLine(lineFromFile);
}
Reader.Close( );
```

```
fileName =
'C:\MyDocum

file = open
for line in
    print(l
file.close(
```

Pascal/Delphi

```
var
 fileIn:textfile;
 lineFromFile:String;

begin

assignfile(fileIn 'C:\MyDocuments\ReadingFileTest.txt');
  reset(fileIn) // sets the program ready to read
from the file and moves the pointer to the top
  while not eof(filein) do
  begin
    readln(fileIn,lineFromFile);
    writeln(lineFromFile);
  end;
  closefile(fileIn);

end.
```

Writing to a file

Likewise, with writing to a file there is a special built-in class in most progra an interface for you. In the C-family this is called *StreamWriter* and in Python change the I/O type to *Write*. Take a look at the code extracts below.

Pseudo

```
fileName ← directory address of file

FileWriter ← new stream writer [fileName]

For i ← 1 TO 5
    Write "input line: "
    inputString ← Read written line
    FileWriter Writes inputString to file
Next

Close filewriter
```

```
Dim fileN
Dim testS
Dim FileW
fileName
C:\MyDocu

FileWrit

For i = 1
    Cons
i)
    inpu
    File
Next
FileRead
```

C#

```
Static Streamwriter FileWriter
Static void Main (…)
{
String fileName =
@"C:\MyDocuments\WritingToFileTest.txt";
fileWriter = new Streamwriter(fileName);
for (int i = 1; i<= 5; i++)
{
    Console.Write("input Line Number {0}: ", i);
    String inputString = Console.ReadLine( );
    FileWriter.WriteLine(inputString);
}
FileWriter.Close( );
}
```

```
fileName
'C:\MyDo

file = op
For x in
    inpu
    file
file.clos
```

Pascal/Delphi

```
var
 fileOut:textfile;
 inputString:String;
 count:integer;

begin

assignfile(fileOut,'C:\MyDocuments\WritingFileTest.txt');
  reset(fileOut);  // sets the program ready to write
  for count:=1 to 5 do
  begin
    readln(inputString);
    writeln(fileOut,inputString);
  end;
  closefile(fileOut);
end.
```

Binary files

Binary files are stored as binary encoded data. The content of the file itself i but can be written in binary *(see Section 5.2)*, so the content seems irrelevant directly by components without the need for translating or conversions. Bina record types and read back into that type.

Writing binary files using record structure

We can declare a record, input the appropriate data and write it to the file i than simple text which has to be converted.

Pseudo

```
Declare recStructure as record
name: string
houseNum : integer
postcode: string

CurrentRec ← recStructure
pathName ← #file pathway

open binaryfile using pathname
Set BinaryFile for Write
Loop
  Input currentRec
  BinaryWriter (currentRec)
Until finished entering data

Close binaryfile
```

```
Public Structure rec
Public name As Strin
Public houseNum As I
Public postcode As S
End Structure

Dim CurrentRec As re
Dim CurrentFileReade
Dim CurrentFile As F

Dim Filename as Stri
Filename = # file pa

CurrentFile = New
FileStream(Filename,
CurrentFileWriter =

Do
  CurrentRec.name =
  CurrentRec.houseNu
  CurrentRec.postcod
  currentFileWriter.
  currentFileWriter.
  currentFileWriter.
  console.Write("do y
  answer = Console.R
Loop until (answer="

CurrentFileWriter.Cl
CurrentFile.Close()
```

C#

```
Struct recStructure
{
Public string name
Public int housenum
Public string postcode
}

Static currentRec recStructure;
Static BinaryWriter currentFileWriter;
Static FileStream currentFile;

String filename = # file pathway;
currentFile = new
FileStream(filename,FileMode.Create);
currentFileWriter = new
BinaryWriter(currentFile);
do
{
  CurrentRec.houseNum = Console.ReadLine();
  CurrentRec.postcode = Console.ReadLine();
  currentFileWriter.Write(CurrentRec.name);

currentFileWriter.Write(CurrentRec.houseNum);

currentFileWriter.Write(CurrentRec.postcode);
  console.Write("do you want to add another
  record");
  answer = Console.ReadLine();
}

While (answer ==  "Y");

currentFileWriter.Close();
currentFile.Close();
```

P

```
type
  recStructure = rec
    name:string[15];
    housenum:integer
    postcode:string[
  end;

 var
   count:integer;
   fileOut:file of r
   currentRec:recStr
   answer:string;

begin
  assignfile(fileOut
  rewrite(fileOut);
  repeat
    readln(currentRe
    readln(currentRe
    readln(currentRe
    write(fileOut,cu
    writeln('Do you
    readln(answer);
  until answer = 'N'
  closefile(fileOut)
end.

// note that in Pasc
declared in the reco
```

Reading binary files using record types

By using record types and binary files the records are written to the file in th structure. However, when reading the files it is essential the same structure i

The examples below show a simple method with no error detection.

Pseudo

```
Declare recStructure as record
name: string
houseNum : integer
postcode: string

CurrentRec ← recStructure
pathName ← #file pathway

open binary file using pathName
binaryReader ← BinaryReader(read record)
current ← binaryReader
OUTPUT current
```

```
Public Structure recStr
Public name As String
Public houseNum As Inte
Public postcode As Stri
End Structure

Dim CurrentRec As recSt
Dim CurrentFileReader A
Dim CurrentFile As File

Dim Filename as String
Filename = # file pathw

CurrentFile = New FileS
CurrentFileReader = New

Do While CurrentFile.Po
  CurrentRec.name = Cur
  CurrentRec.houseNum =
  CurrentRec.postcode =
CurrentFileReader.ReadS
  Console.WriteLine(Cur
  Console.WriteLine(Cur
  Console.WriteLine(Cur
Loop

CurrentFileReader.Close
CurrentFile.Close()
```

C#

```
Struct recStructure
{
Public string name
Public int housenum
Public string postcode
}

Static currentRec recStructure;
Static BinaryReader currentFileReader;
Static FileStream currentFile;

String filename = # file pathway;
currentFile = new FileStream(filename,FileMode.Open);
currentFileReader = new BinaryReader(currentFile);
do
{
  CurrentRec.name = CurrentFileReader.ReadString();
  CurrentRec.houseNum = CurrentFileReader.ReadInt32();
  CurrentRec.postcode = CurrentFileReader.ReadString();
  Console.WriteLine(CurrentRec.name);
  Console.WriteLine(CurrentRec.houseNum);
  Console.WriteLine(CurrentRec.postcode);
}
While (currentFile.Position < currentFile.Length);

currentFileReader.Close();
currentFile.Close();
```

```
type
  recStr
    name
    hous
    post
  end;

 var
   count
   fileI
     curr

begin

  assign
  reset(
  while
  begin
    read
    writ
    writ
    writ
  end;
  closef

end.
```

Note: # file pathway is the full path filename, for example 'C:\mydocuments\fileN

Another method of reading binary files is to use streaming to read/write dat are binary files whose contents can be used directly by hardware processors the central processor if the processor knows how the data is formatted.

This processing of binary files is called streaming as the data is 'streamed' o program. The method is shown below.

Reading binary files with no record structure (streaming)

Here is the code that can be used for reading binary files.

Pseudo

```
Current ← Null
pathName ← #file pathway

readStream ← FileStream (pathName, OpenFileMode)
binaryReader ← BinaryReader (readStream)
current ← binaryReader
OUTPUT current
```

```
FileStream r
BinaryReader
String curre
String pathN

readStream =
FileMode.Ope
binaryreader
current =rea
Console.Writ
```

C#

```
FileStream readStream;
String current = null;
String pathName = # file path way

readStream = new FileStream (pathName,
FileMode.Open);
binaryreader = new BinaryReader (readStream);
current =binaryReader.ReadString( );
Console.WriteLine(current);
```

```
file = open(
try:
    byte = f
    while by
        # Co
          # s
        byte
finally:
    file.clo
```

Pascal/Delphi

```
uses classes,sysutils;

var
  fsOut    : TFileStream;
  fsIn     : TFileStream;
  source: array[0..4] of integer = (2, 1, 8, 6, 244);

begin
    fsIn := TFileStream.Create('binaryfile.bin',
    fmOpenRead);
    fsIn.Read(source, sizeof(source));
    fsIn.Free;

 // the array Source now contains the data

end.
```

Writing binary files with no record structure (streaming)

Writing binary files is similar to writing text files and you will notice the simi
family languages. In this example you will see that an array containing some
being written to the file.

Pseudo

```
Source ← #values for array

Using writer ← #open file, filename, file mode

For i ← 1 to source.length
    Print
```

```
Dim source[5]

Using writer A
BinaryWriter (
FileMode.Creat

For value = 0
    Writer.wr
Next
End Using
```

C#

```
int [illegible][5] [illegible]ew source [2, 1, 8, 6, 244];

Using BinaryWriter = New BinaryWriter
(file.Open("binaryFile.bin", FileMode.Create))
{
    For (int value = 0; i >= source.Length; i++)
    {
        writer.write(value);
    }
}
```

```
with open('bin
    for value
        write
```

Pascal/Delphi

```
uses classes,sysutils;

var
  fsOut    : TFileStream;
  source: array[0..4] of integer = (2, 1, 8, 6,
244);

begin
    fsOut := TFileStream.Create('binaryfile.bin',
    fmCreate);
    fsOut.Write(source, sizeof(source));
    fsOut.Free; // this prevents memory leaks
end.
```

Task:

Write th
continu
and writ

When th
all the n
be displ

ABSTRACT DATA TYPES / DATA STRUCTURES

Abstract data types are defined as data types which are not defined by their i programming language. Instead they are defined by the operations that can l

The abstract types and structures that you need to be familiar with are as fol

- Queues
- Stacks
- Lists
- Graphs
- Trees
- Hash tables
- Dictionaries
- Vectors

Each of these is covered in detail over the following pages.

2.2 QUEUES

The data structure known as a queue has the same characteristics as the que you encounter in everyday life. For instance, a queue at the checkout counte supermarket increases at its rear as customers join the queue to have their purchases totalled, and only reduces in size when a customer is served at the of the queue, the checkout counter. A queue of cars at traffic lights behaves similar manner, with cars exiting the queue only at its front and joining the only at its rear. This is a FIFO data structure (First In, First Out).

A queue requires two pointers, one of which points to the front and the othe needs to be set up in such a way that the following operations can be carrie

1. Check whether the queue is empty.
2. Return the value of the first element (front).
3. Return the value of the first element and remove it.
4. Place new element onto the rear of the queue.

There are three different types of queue which you need to know about: circ priority queues. Circular queues are particularly suited to implementation as hand, lend themselves to being implemented with lists.

CIRCULAR QUEUES

A circular queue is a queue which has a fixed amount of space, but where th the front, much like a circle. This structure lends itself easily to buffering da

1. The *front* points to the element of the array which should be remove
2. The *rear* points to the last element added. As data is added these pos array and loop back to the start of the array.

Suppose a queue is formed in the following order: Ashley, George, Julie, Jacq an array as follows (assuming a 1 by 10 array):

FRONT → REAR

Ashley	Julie	George	Jacques	Liz		

Suppose that two names (Ashley and Julie) leave the queue and two new na The queue would now look like this:

FRONT → REAR

		George	Jacques	Liz	Justin	Mary

As more names are added the array would eventually loop round and if the
Kumar were added (in that order), the queue would look like this:

REAR		FRONT				
Kumar		George	Jacques	Liz	Justin	Mary

If another name were to be added to this queue then the queue would be fu
if the front and tail are next to each other, with the front on the right of the
to the first element in the array and the tail points to the last element in the
check that the queue is empty by looking at the tail and checking if it is equ

Implementation of a circular queue

We are assuming the array is of size 10 (i.e. data [10]). This has the limitatio
maximum of 10 items. It has two pointers that point to the front and the

To test for an empty queue:

```
PROCEDURE see_empty( )
   IF front ← rear THEN
      PRINT("Queue is empty!")
   ELSE
      PRINT("Queue is not empty")
   END IF
END PROC
```

To look at th

```
PROCEDURE see_fro
   IF front ← rea
      PRINT("Queue
   ELSE
      PRINT("Front i
   END IF
END PROC
```

To add (push) an item to a queue:
(assumes the pointers and array are global variables)

```
PROCEDURE push(new_item)
   IF (rear + 1← front) OR (rear ← 10
   AND front ← 1) THEN
      PRINT("Queue is full!")
   ELSE
      IF rear ← 10 THEN
         rear ← 1
      ELSE
         rear ← rear + 1
      END IF
      data(rear) ← new_item
   END IF
END PROC
```

To take an i
The function wi

```
FUNCTION   pop( )
   IF front ← rea
      PRINT("Queue
   ELSE
      pop ← data(f
      IF front = 1
         front ← 1
      ELSE
         front ← f
      END IF
   END IF
END FUNCTION
```

To print out all the items in a queu

```
PROCEDURE print_queue( )
   DIM i As Integer          // loop counter
   DIM queue_string As STRING    //builds up the queue before
   i ← front
   WHILE i <> rear
      queue_string ← queue_string & data(i)
      IF i ← 10 THEN i ← 1 ELSE i ←
      i + 1
   End WHILE
   PRINT(queue_string)
END PROC
```

LINEAR QUEUES

Since a queue usually holds a bunch of items with the same type, it makes se an array. With linear queues, elements are always added at one end and rem diagram demonstrates the concept using a fixed-size queue of 8 elements (fr

FRONT ——————▶ REAR

0	1	2	3	4	5
Apple	Banana	Grapes	Melon	Orange	

Items are always removed from the front of the queue. To do this, the front p at the next item in the list. The queue below shows the element (Apple) bei

FRONT ——————▶ REAR

0	1	2	3	4	5
	Banana	Grapes	Melon	Orange	

Items are added to the list at the rear end. The example below shows two ne added to the list. As with the front pointer while adding elements, the rear p so that it points at the newest item:

FRONT ——————

0	1	2	3	4	5
	Banana	Grapes	Melon	Orange	Pear

PRIORITY QUEUES

Priority queues are queues where items are removed in order of their priority the items are added. Each item is assigned a priority as it is added to the que on the basis of origin, data type, time of day – anything really.

Priority queues find uses in a number of areas of computing. Network buffers queues. By using priority queues in networks, it is possible to move importan communications, to the front of the queue, while at the same time moving le that generated by file-sharing applications, to the back of the queue.

Adding an item

```
IF queue is full Then
    OUTPUT Error
ELSE
    rearPointer++
    IF rearPointer > maxIndex Then
        rearPointer ← 1
    queue[rearpointer] ← datum

END IF
```

Remov

```
IF queue is empty The
    OUTPUT error
ELSE
    Return queue[fron
    frontpoint++
    IF frontPointer >
        frontPointer ←
    END IF
END IF
```

Testing if empty

```
IF queue [frontPointer] ← 1 Then
    Return true
ELSE
    Return false
END IF
```

Test

```
IF queue[rearPointer]
Then
    Return True
ELSE
    Return False
END IF
```

Questions: Queues

1 You are tasked with writing a video buffer for a video player that can Which would be more suitable, a stack or a queue? Explain your ansv

2 A circular queue is usually implemented as an array.

a) What variables are required to keep track of such a queue? (1 ma

b) How can you check whether a circular queue is empty without u (1 mark)

c) How can you check whether a circular queue is full without using

d) Write a procedure to add an item to the queue in pseudocode. an error message if the item cannot be added. (1 mark)

3 Consider the following linear queue. Elements are added at the rear

FRONT			REAR		
0	1	2	3	4	5
SF	A2	EE	72		

a) Draw the queue after two elements have been removed. (1 mark

b) Devise a method for representing an empty queue. What steps push/pop procedures? (1 mark)

c) Write a procedure, using pseudocode, to add items to the queue queue is empty. (1 mark)

2.3 STACKS

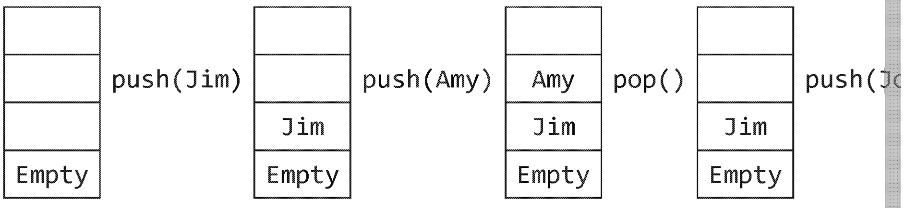
A stack is a data structure characterised by the expression Last In, First Out (recent item added to the stack is the first one which can be removed from th keep track of the last item added to the stack – that is, the current top of the

A real-life visualisation of a stack is the stack of trays at the entrance to a ca you need to take that tray off the top in order to get to the next one down. If take that tray and put the dirty tray back on the top of the stack. However, yo of the stack without first removing all the other trays!

A stack needs to be set up so that the following operations can be carried ou

- Check whether the stack is empty (NULL or not)
- Check whether the stack is full or not
- Look at the top value and remove it (pop)
- Look at the top value, without removing it (peek)
- Add new value on top of the existing stack (push)

Here is an example of a stack used to hold names, and how pushing items or from the stack works:



Implementing a stack

A stack can be implemented using an *array* or a *linked list*. To implement a s
variable is needed to keep track of the top of the stack.

For example, an array declared like the following in Java could be used as a s
variable when a string is pushed onto the stack and decrementing it when a

```
String[] myStack;
myStack ← new String[10];
int top ← -1;                // -1 means the stack is e
```

Alternatively a linked list can be used instead of an array, with only the addr
needed.

Example – Procedures to implement a stack using arrays

What follows are some examples of procedures, written in pseudocode, whic
implement a stack as an array.

We are making the following assumptions:

1. There is an integer variable, `top`, which points to the top of the stack
   stack is empty.
2. The size of the array `myStack` is given by the integer `max`.
3. Elements are accessed in the array by reference, with 0 being the first e
   - myStack[0] is the first element
   - myStack[1] is the second element
   - myStack[299] is the three-hundredth element
   - myStack[max-1] is the last element

Testing an empty stack is simple as all we need to do is check whether the t

```
PROCEDURE test_empty( )
  IF top = -1 THEN
    PRINT("Stack is empty")
  ELSE
    PRINT("Stack is not empty")
  END IF
END PROCEDURE
```

To look at the top item on a stack without removing it we can say:

```
PROCEDURE print_top( )
  IF top = -1 then
    PRINT("Stack is empty")
  ELSE
    PRINT("Top item on stack is " & myStack[top])
  END IF
END PROCEDURE
```

To add an item (push) to a stack we can use the following procedure (assumi variables):

```
PROCEDURE push(newItem)
  IF top = max - 1 THEN
    PRINT("Stack is full!")
  ELSE
    top ← top + 1
    myStack[top] ← newItem
  END IF
END PROCEDURE
```

To take off an item (pop) from a stack we can use the following function (wh

```
FUNCTION pop( ) AS INTEGER
  IF top = -1 THEN
    PRINT("Stack is empty!")
  ELSE
    top ← top - 1
    RETURN myStack[top]
  END IF
END FUNCTION
```

Questions: Stacks

1 What rule is said to govern stack data? (1 mark)

2 Imagine a stack containing the following numbers: 89, 45, 22, 90
90 is the top of the stack.

a) Rewrite the stack after the following operations have been perf

```
pop()
pop()
push(77)
push(56)
```

b) Write a function in pseudocode that will add up all the items in

3 Here is an example of a stack implemented as an array:

`Top of stack: 5`

0	John
1	Lara
2	Mike
3	Steve
4	Frank
5	Billy
6	Georgina

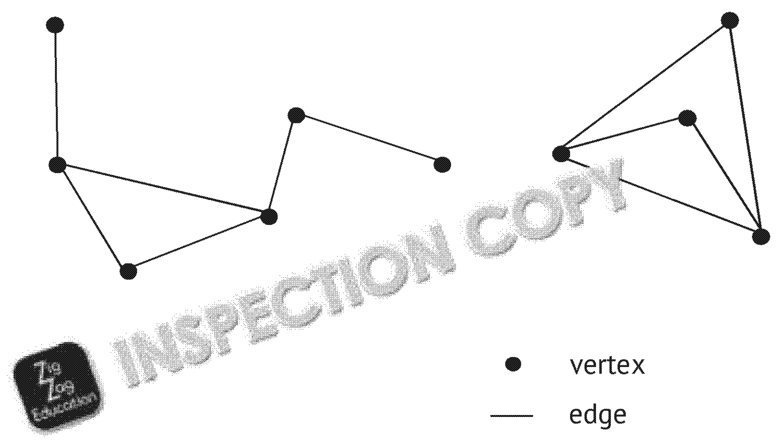
a) Which name would be removed from the stack first when the pop

b) What steps are needed to push an item onto the stack? (1 mark)

c) What would be a suitable value for the top variable when the st

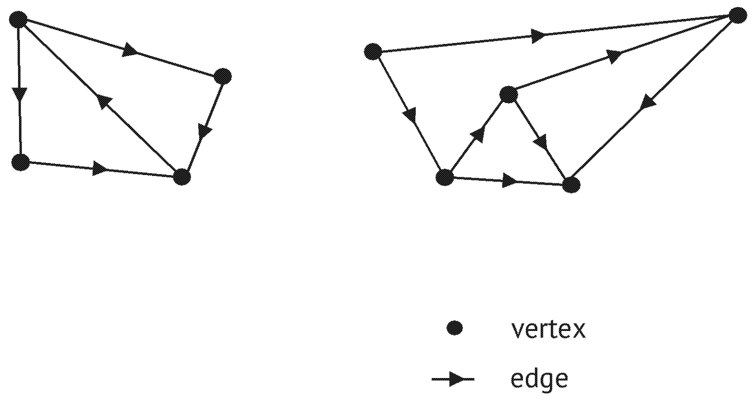
2.4 GRAPHS

A graph has a set of *vertices* (often also referred to as *nodes*) and *edges* wher vertex is a point and edges are the lines that join the points together. A labe vertices associated with a label.



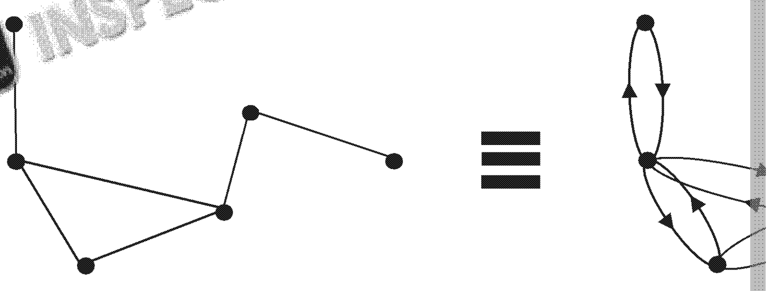
Directed graph

A directed graph (also known as digraph) consists of arcs and vertices where and has a direction. A labelled digraph is one that has its vertices associated



Undirected graph

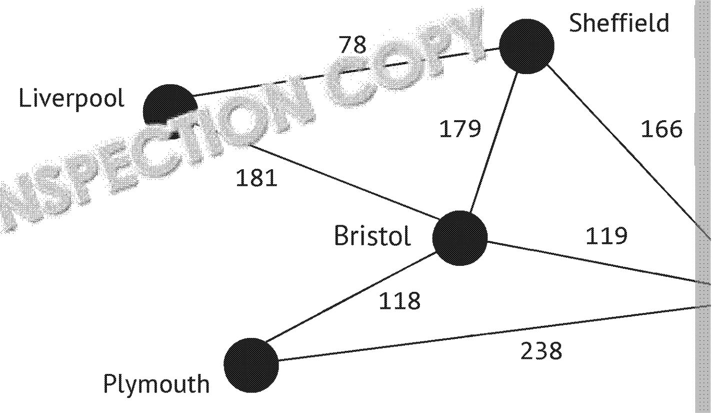
Undirected graphs can be represented using directed graphs very easily by si with two edges, each pointing in opposite directions.



Weighted graphs

Weighted graphs are graphs where every edge is given a weight. The weight represents some quantity. A very common use of weighted graphs is to repre locations. The travelling salesman problem is an example of a problem wher However, the weights could represent anything; for example, they could repr when analysing network performance or they could represent the number of another web page in a search engine.

Below is an example of a weighted undirected graph representing the distan *travelling salesman problem*:

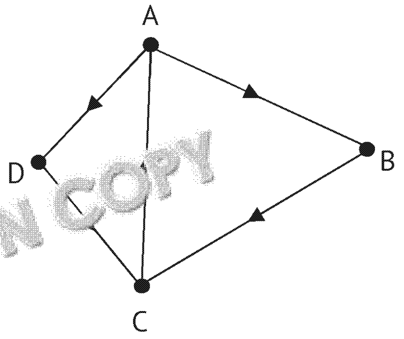


The uses of graphs

Graphs can be used in many applications, for purposes such as finding routes designing computer networks, traffic control and finding the best route to ta in solving games such as mazes and Sokoban puzzles. The trick is being able by using a graph.

Adjacency list

An adjacency list is used to represent either edges or arcs. It is a linked list w the current node, and what nodes the current node is connected to. Consider



The adjacency list would be as follows:

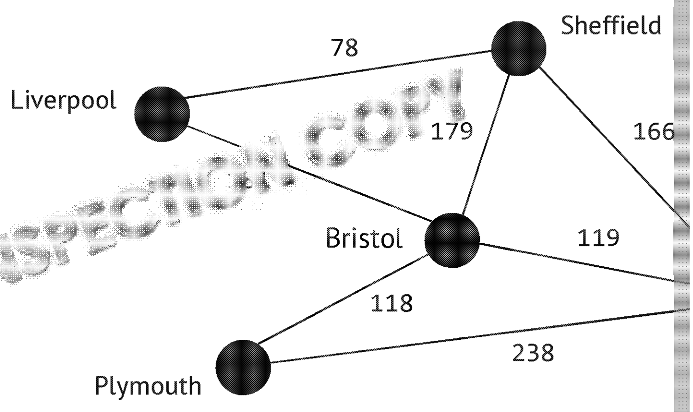
Vertex	Connected to
A	B, D
B	C
C	A
D	C

Adjacency matrix

An adjacency matrix is another method to represent edges in a graph. An *n*-by-*n* is used to represent all the edges in the graph where *n* is the number of vertices

The adjacency matrix shows how many edges are used to connect each verte together. For example, vertex A is connected with vertex B by one edge.

To come back to the travelling salesman problem a weighted adjacency mat can be used.

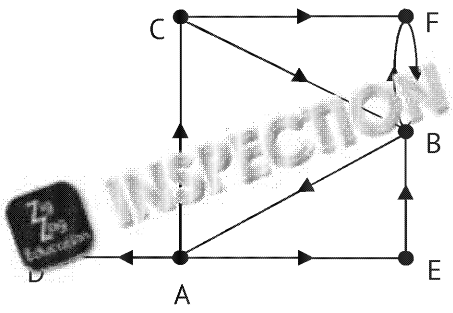


	Bristol	Liverpool	London	Plymouth
Bristol	0	181	119	118
Liverpool	181	0	0	0
London	119	0	0	238
Plymouth	118	0	238	0
Sheffield	179	78	166	0

In the above the weighting between Bristol to Liverpool is 181 however if it was from Bristol → Liverpool, and not from Liverpool → Bristol), then the Liverpool

Questions: Stacks

1 Consider the following directed graph:



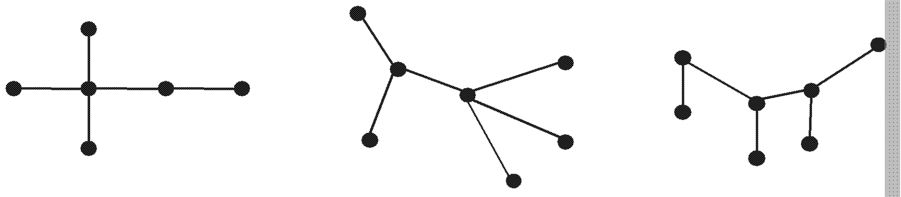
a) Write an adjacency list to represent this digraph. (1 mark)

b) Write an adjacency matrix to represent this digraph. (1 mark)

c) Assuming space is the main limiting factor, which would the why? (2 marks)

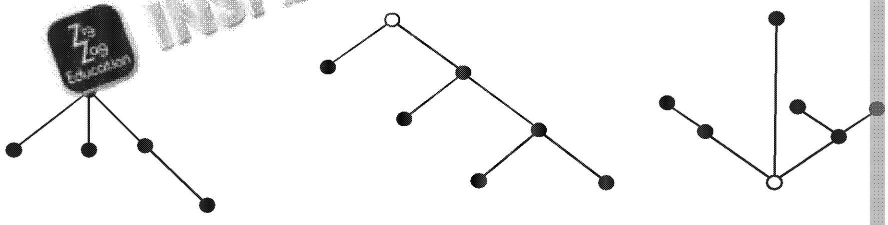
2.5 TREES

A tree is a simple undirected graph that contains no cycles in it, i.e. each ver
one path. A tree with n vertices always has n − 1 edges.



ROOTED TREE

A rooted tree is usually used to show some type of hierarchy. It contains a ro
vertices stem.

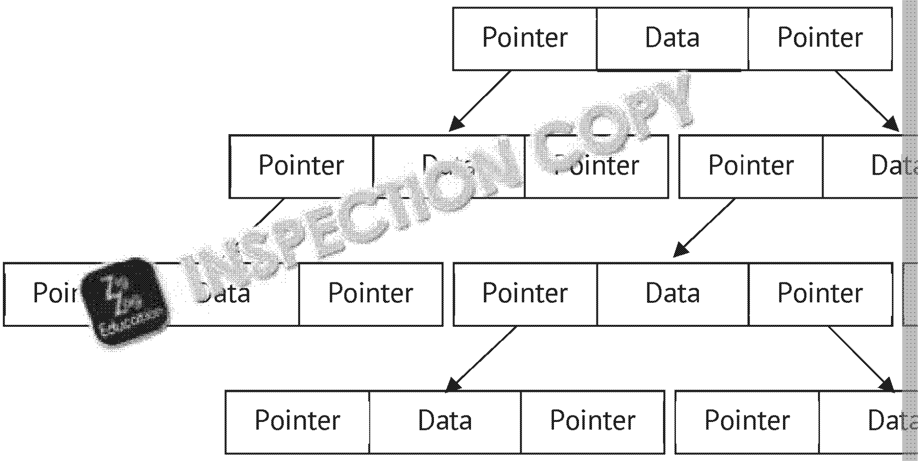


Generally rooted trees are drawn with the root at either the top or bottom of
diagram easier to understand and makes the structure clearer. Any vertex co
needs to be designated as such.

BINARY TREES

A binary tree is a type of rooted tree which is often used in computing as an
used because they are more structured and therefore quicker to search than
hashing, they can be used as a basis for constructing a database.

Binary trees consist of *nodes* (vertices) and *branches* (edges), which are the lin
Nodes which have no children are called *leaf* nodes. A binary tree must alway
and deleting elements or else it would be pointless to use one since searchin
node consists of a field and two pointer values, one for the left subtree and th



Binary trees are a form of abstract data structure and therefore, like linked li
number of ways. Binary trees can be implemented as arrays with one column
two pointers (left and right). They can also be implemented dynamically usin

Constructing a binary tree

By convention, items added to a binary tree are compared to the root node; i
node they are placed on the right-hand side and if less than the root node th
This is then repeated for every node they come across until they reach a posi

A procedure to produce a binary tree would therefore need to contain the fol

1. Create a root node.
2. Insert data into the root node.
3. If the root node exists then compare the new item to be added with t current item as the root node.
4. If item is > root node follow right pointer, if not follow left pointer.
5. Compare until the left or right pointer is Null. That is the appropriate
6. Create node, insert item into node and set both left and right pointer
7. Connect node to the tree.

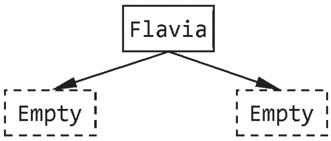
Adding an item to a tree

To add an item to a tree we can use the following procedure (assuming *array* variables and the first element is always in row 1):

```
PROCEDURE add_leaf(new_item)
  row ← 1
  current ← 1
  finished ← false

  WHILE(row <= arraysize and tree(row, 1) != "")    // l
    row ← row + 1
  END WHILE

  IF row > arraysize             // check that array isn
    PRINT("Tree is full!")
  ELSE
    tree(row,1) ← new_item           // puts new item in
    tree(row,2) ← -1             // sets pointers to nul
    tree(row,3) ← -1             // sets pointers to nul
  END IF

  WHILE finished = false
    IF new_item < tree(current,1,1 )// adds pointer to n
      IF tree(current,1,2) ← -1
        tree(current,1,2) ← row
        finished ← true
      ELSE
        current ← tree(current,1,2)
      END IF
    ELSE
      IF tree(current,1,3) ← -1 THEN // adds pointer to
        tree(current,1,3) ← row
        finished ← true
      ELSE
        current ← tree(current,1,3)
      END IF
    END IF
  END WHILE
END PROCEDURE
```

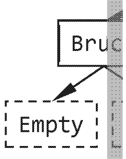
The following diagram shows how the first six steps are used to construct th

Step 1 – Add Flavia



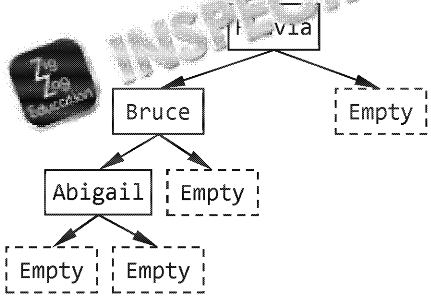
Flavia is the first node added and so becomes the root of the tree.

Step 2 – Add Bruce



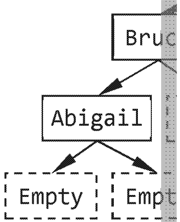
Bruce is before Flav
Empty position foun

Step 3 – Add Abigail



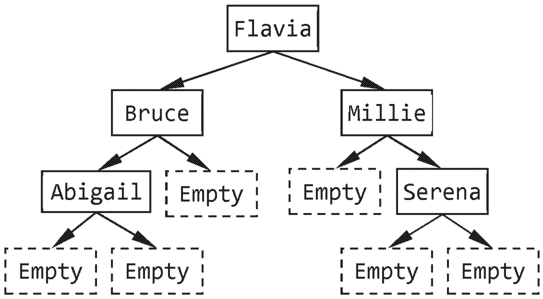
Abigail is before Flavia so go left.
Abigail is before Bruce so go left.
Empty position found so Abigail is added here.

Step 4 – Add Millie



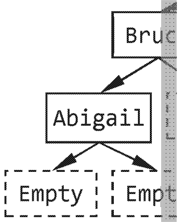
Millie is greater tha
Empty position foun

Step 5 – Add Serena



Serena is greater than Flavia so go right.
Serena is greater than Millie so go right.
Empty position found so Serena is added here.

Step 6 – Add Rache



Rachel is greater th
Rachel is greater th
Rachel is less than S
Empty position foun

Questions: Trees

1 Consider the set of integers a = {7, 2, 6, 11, 5, 9, 4, 8}. If the items, bef a binary tree, were sorted into descending order what would the tree

2 The search time for a binary tree is usually O(log n). If the binary tree search time become? (1 mark)

3. Describe a procedure to delete a node in a binary tree implemented usi

2.6 HASH TABLES

So far you have looked at algorithms which can be used to search and sort a

One problem with all of these structures is that as data volume increases in becomes more difficult and harder to manage. All of these structures require structure like a tree, which allows a binary search operation, increases in tim *For more detail, refer to the notes on Big O Notation (Section 4.4)*. A binary sea

The hashing algorithm is a mathematical calculation performed on search cr were to use the surname we could perform a calculation to find the data loc the data to be found). If well devised it can significantly cut down the search worst case; often it is O(1), i.e. one iteration).

Hashing algorithms are often tailored to the specific application that they wi hashing algorithm which is used at university. One example of a hashing algo operates on the first two letters of a name, where the letters a–z are given t index is then produced by multiplying the first number by 26 and then addin

Here is an example of a hash table showing keys that the algorithm will pro

Surname	Jones	Zheng	Patel	Ba
Calculation	9×26+14	25×26+7	15×26+0	1×
Index	248	657	390	

As can be seen, if we wanted the information about the person whose surna record location 390. So we could jump directly to memory or index 390 and

Like all hashing algorithms, however, there is a problem, and this is that it p different surnames that begin with the same two letters. For example, the s have the same index. This is known as a collision. The aim of a good hashin collisions while also optimising space needed.

Collisions

One of the main problems that hashing functions have is that they will probabl index, i.e. two different items could produce the same index. This is called a *co collision resolution strategy*. One collision resolution strategy is to use the next This seems a good idea in principle but leads to problems later. For example, if not in the position you expect, you can't assume it has been deleted because it else due to a collision. Likewise, if you delete a record you have to indicate tha know there may have been a collision.

The solution to this is to use overflow lists. There are two possibilities for an overflow list.

Firstly, that all records with duplicate indexes are simply put into another list. If an item cannot be found in the main table then the program will need to perform a linear search on that list. Only once the overflow list has been searched can the program be sure that the record is not in the table.

The other possibility is to have linked lists attached to every index. All the records with the same index are stored in the linked list, and the program simply performs a linear search on the linked list for the index it has calculated. However, this solution is slightly more complex to implement.

Questions: Has

1 Why is hashing i

2 Hashing is often stored in binary t use the hashing a

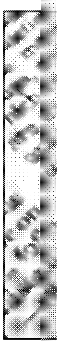
a) Work out th *Hull, Brighto Manchester,*

b) Place the pl based on the

c) What proble add Bristol t

d) Propose a s without cha

2.7 DICTIONARIES



Dictionaries are another example of an abstract data type which a programmer can use to store data in an ordered manner, although it is worth noting that it is also defined as a *generic class* in C programming.

If you take the analogy of a real-world dictionary, the data is stored in a two-part key: the *word* (the key) and the *definition* (the data). To get to the data you se arch for a specific word and read the definition; likewise in programming you search the key-field and retrieve the data associated with the key.

The functions that can be applied to a dictionary construct are dependent o are using, but all have the same basic functionality. For example, in the *C fa generic class* and implemented using a list, therefore the class has all the fu included in functions in the dictionary that have been built into the languag

Creating and adding to a dictionary

Pseudo

```
STRUCTURE dictionary (key, data)
    Dictionary[2] ← "Computer"
    Dictionary[1] ← "I"
    Dictionary[0] ← "Love"
    Dictionary[-1] ← Science"
END STRUCTURE
```

```
Dim dictionary As
Integer)
         Dictiona
         Dictiona
         Dictiona
         Dictiona
```

C#

```
Dictionary<string, int> dictionary = new
Dictionary<string, int>();
            dictionary.Add("Computer", 2);
            dictionary.Add("I", 1);
            dictionary.Add("Love", 0);
            dictionary.Add("Science", -1);
```

```
Dictionary = { }

Dictionary["Compu
Dictionary["I"] =
Dictionary["Love"
Dictionary["Scien
```

Pascal/Delphi

```
No native dictionary system is available in
Pascal/Delphi, although there is a method
utilising a library called Generics; the example
below is from free Pascal using the library fgl:

uses fgl, sysutils;  // using the generics
library

type
  TDictionary = specialize TFPGMap<string,
integer>;

var
  i: integer;
  dictionary: TDictionary;

begin
  dictionary := TDictionary.Create;
  dictionary.Sorted := True; // for fast lookup
of keys
  // assign values
  dictionary['Computer']:= 2;
  dictionary['I']:= 1;
  dictionary['Love']:= 0;
  dictionary['Science']:= -1;
```

Using a dictionary

Dictionaries are useful whenever you need to store or retrieve unique inform initialised dictionary there are a few functions you're going to want to know

Checking whether an entry exists

If you want to check whether there is an entry in the dictionary using a key, output the value by using the following code:

Pseudo

```
If dictionary Contains ["science"] Then
    Value ← dictionary "science"
    PRINT value
ELSE
    PRINT "Value not found"
END IF
```

```
If dictionary.Con
    Dim int value
    Console.Write
Else
    Console.Write
End If
```

C#

```
If (dictionary.ContainsKey("science"))
{
    Int value = dictionary["science"];
    Console.WriteLine(value);
}
Else
{
    Console.WriteLine("value not found");
}
```

```
# method one:
Print (dictionary
found"))

# method two:
If "science" in d
    Print (dictio
Else:
    Print ("value
```

Pascal/Delphi

```
if dictionary.find('Science',i) then
   writeln(dictionary['Science'])
else
  writeln('value not found');
```

Note: when using a member of the C family, you can also use the extension '.Con a better method you could research is method called '.TryGetValue'.

Removing an entry from a dictionary

Just as easily as adding an item to a dictionary, you can remove items too. Th items by removing redundant information, such as when you save over a file

C# and VB.NET	Python	P
`Dictionary.Remove("Science");`	`Del dictionary["Science"]`	d

Implementation without libraries

A dictionary may also be implemented using an array and utilising search ro of doing this. One method would be a linked list array-type structure. Anothe structure, which is illustrated in pseudocode below.

Initialising the dictionary:

```
NEW STRUCTURE ← TDictionary  # declare the structure
    Item ← string
    Index ← integer
END STRUCTURE

Dictionary ← array of TDictionary
```

Using the dictionary (finding)

This method utilises a method of knowing how many items are in the array (

```
pointer ← 0
While pointer<=numOfItems or found=false
  If dictionary[pointer].Item = itemToBeFound then found
  Pointer ← pointer + 1
End while

if found  then
  Output dictionary[pointer-1].index
Else
  Output "item not found"
Return pointer
```

Removing an item

This is [illegible] a shuffling-up method as we are using a numOfItems. Other sol constru[illegible] would move the pointers.

```
Find(itemToBeFound)
IF itemFound then
  Shuffle array up from pointer+1 to numOfItems
  numOfItems ← numOfItems -1
```

2.8 VECTORS

In mathematical terms, a vector is a geometric quantity for a position in spac direction but no location. Vectors can be expressed in a number of ways:

- As a list of numbers written in square brackets, e.g.:

  [3.14159, −0.3, 8.0211127, 1.0]

- As a power of a set where the power is the number of entries from th

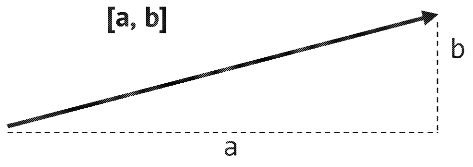
  $\mathbb{R}^4$

- As a function using a dictionary to map values to a set (‘↦’ means ‘n

```
0 ↦ 3.14159
1 ↦ -0.3
2 ↦ 8.0211127
3 ↦ 1.0
```

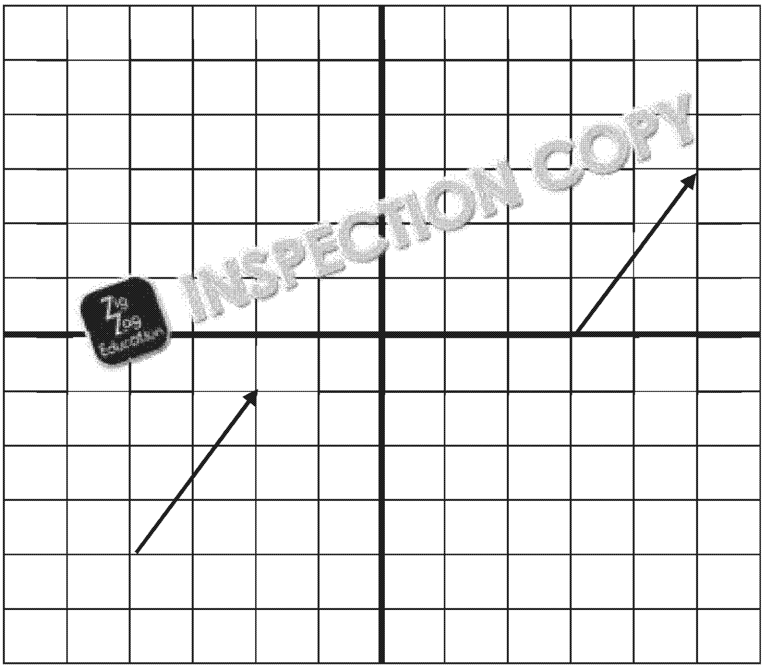
A vector can also be depicted using ‘arrow’ notation. This is the easiest way t allows you to visualise the vector as a movement; the length of the arrow is You can think of a vector by its horizontal (x) and vertical (y) displacements.

For example, the vector below would be [a, b]:



If you’re finding vectors hard you might be trying to imagine them in terms c Try to remember that although vectors have magnitude and direction, they d what makes them so useful.

Take a look at the example below.



Both have

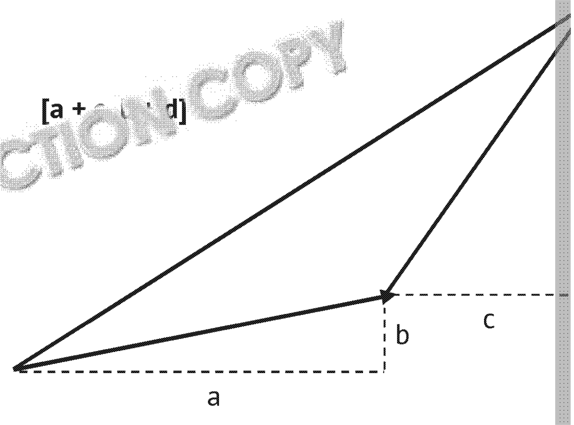
This have

Vector addition and subtraction

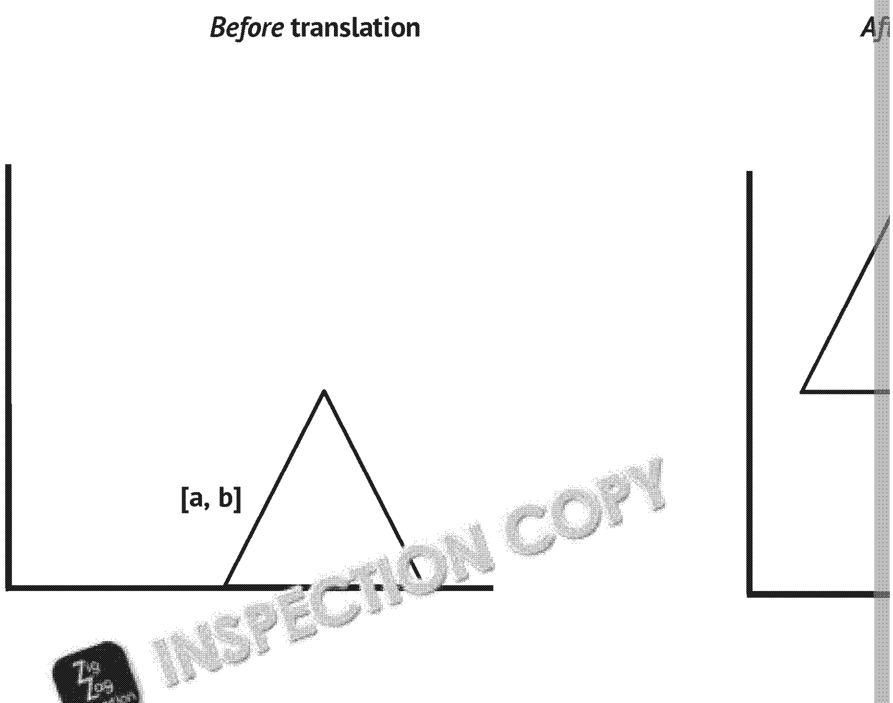
Look at the following example; note how the two vectors when used togeth
hypotenuse is given by the form [a + b, c + d] – this transformation is called
using a vector to contain the magnitude and direction of the movement.

The translation can be given by the equation: [a, b] + [c, d] = [a + c, b + d] wh

The equation becomes *[a, c]* + *[b, d]* because you group similar transformatio
transformations in the same direction and their magnitudes are added togeth



It is easier to visualise this with an example of moving a shape in two-dimen
translating the triangle given by the vector [a, b] by [j, k].


Before translation

Aft

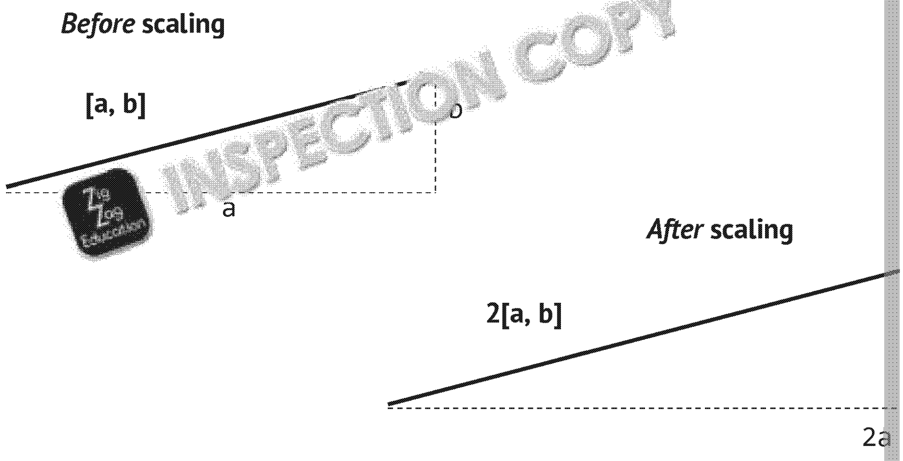
The values of these points are arbitrary but you can clearly see how the trans

The distance and the direction of the transformation are given by the values
addition to map each point: [a, b] + [j, k] → [a+j, b+k].

Vector multiplication

This transformation is nice and easy. You will see later in *Vector Graphics (se* perfect for representing objects where high precision is needed because the ways; they can be moved, inverted and scaled and they always look the sam multiplication comes in – it achieves *scaling*.

Consider the example in the introduction where you represented a vector w displacements and labelled them 'a' and 'b'. This will highlight how simple t using its scalar product. If you multiply each value of the vector by a scalar result in a vector of twice the scale. 2[a, b] → [2a + 2b]



As you can see, the direction of the vector is maintained and the magnitude

Dot products of vectors (vector scalar product)

Consider multiplying two vectors together that have the form:

$$u = [u_1, u_2 \ldots, u_n], v = [v_1, v_2, \ldots, v_n]$$

The result can be written as:

$$u \cdot v = u_1v_1 + u_2v_2 \ldots u_nv_n$$

This can be seen when cross-multiplying matrices, such as in the following

$$\begin{bmatrix} k & l \\ m & n \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} ka + lc & kb + ld \\ ma + nc & mb + nd \end{bmatrix}$$

You can see the scalar product has been used: $ka + lc = [k, l] \cdot [a, c]$

Questions: Vectors

1 $B = \begin{bmatrix} K \\ 0 \end{bmatrix}$

Based on the above, what is the name of the transformation represe

2 $A = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$

Based on the above, what is the vector notation of the transformati

3 $A = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$ and $C = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

Based on the above, consider the transformation required for the ad What is the vector notation for this transformation? (2 marks)

3. Algorithms

This section covers the fundamentals of algorithms, illustrated by common examples. Algori
in examinations, therefore it is important to be able to reproduce, trace and have a strong g

This section covers:

3.1 GRAPH TRAVERSAL

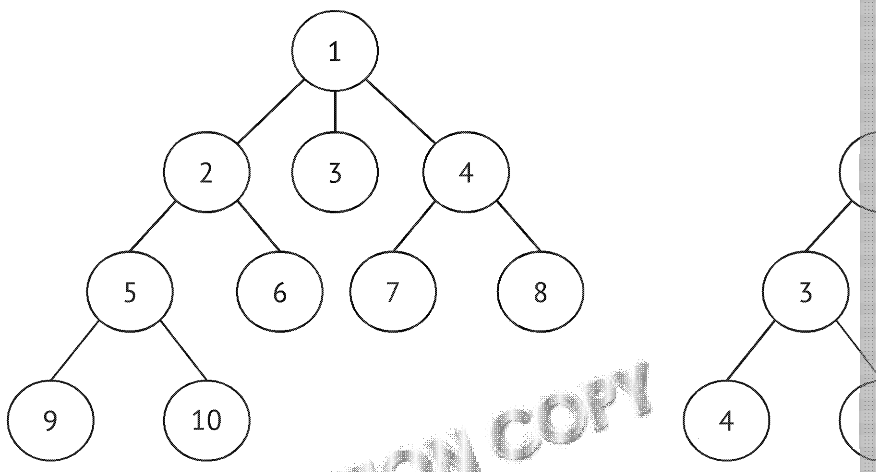
Graph traversal algorithms are algorithms which visit all of the nodes in a gr
operation. This is necessary when, for example, searching for a particular no
more complex operations, such as graph colouring, which are not covered he

Breadth-first and *depth-first* are two possible ways of traversing a graph.

- *Breadth-first* means going through the graph starting from a particula
connected nodes, then looking at the nodes connected to those node
- *Depth-first* means going through the graph, again starting from a parti
connected to the first node connected to the start node are completel
second node connected to the start node and completely exploring al
and so on.

The diagram below shows the order in which a graph would be traversed usi
depth-first search (assuming nodes on the left are visited before nodes on th

Breadth-first Traversal D



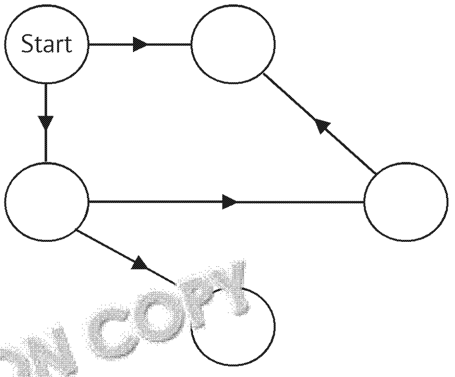
BREADTH-FIRST SEARCH

A breadth-first search first searches all the nodes which are adjacent to the c
the nodes connected to them. The informal algorithm is as follows:

1. Choose a vertex to start from.
2. Colour the start vertex (cross-hatch) to indicate that it has been visit
3. Create a list of neighbouring nodes and visit one of them.
4. Colour the visited node (cross-hatch), then visit the other nodes in th
5. Repeat this process until no more nodes are left in the list.
6. Colour the start vertex black to indicate that it has been completely
7. Go to the next vertex and repeat the process as above.

Performing a breadth-first search on a graph

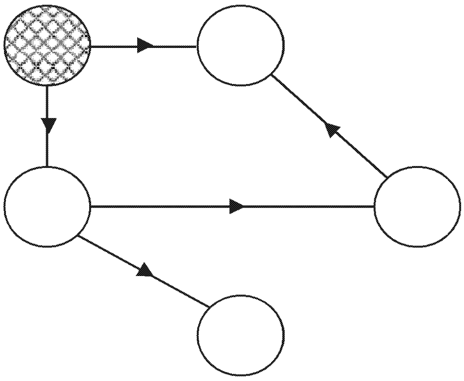
Here is an example of the steps taken during a breadth-first traversal of the



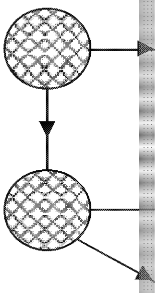
▨ indicates a node which has been **visited**

■ indicates the node has been **completely** ex

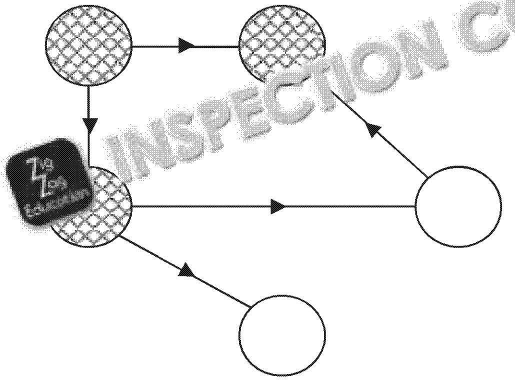
①



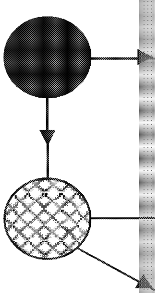
②



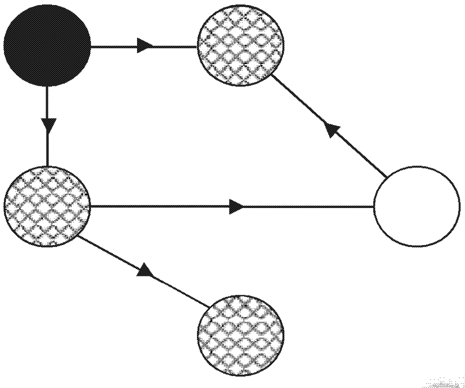
③



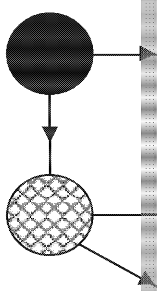
④



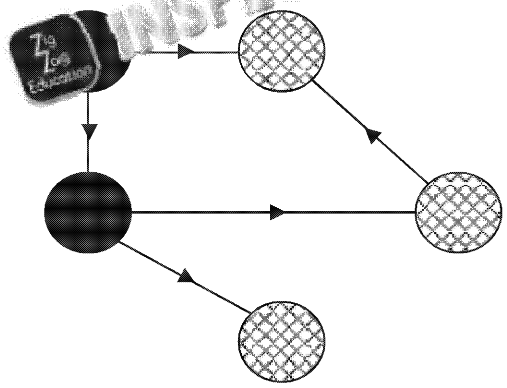
⑤



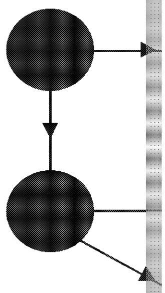
⑥



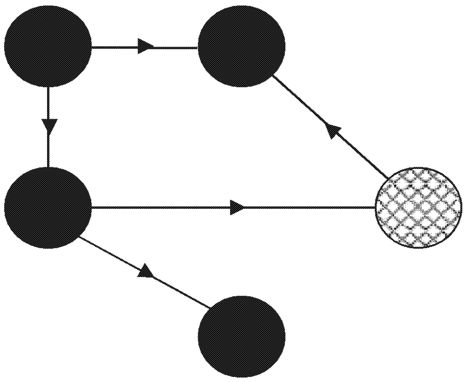
⑦



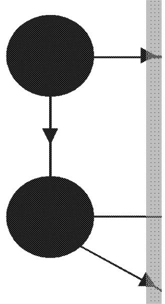
⑧



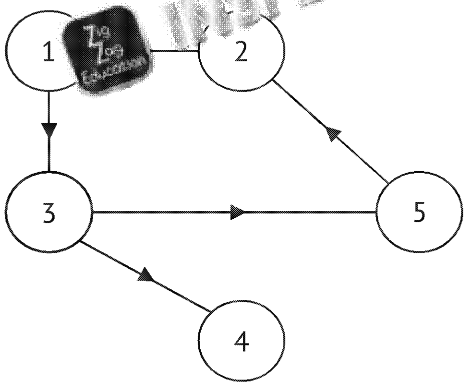
⑨



⑩



The order that the nodes have been traversed in is:



DEPTH-FIRST SEARCH

Depth-first search is another simple graph traversal search, which instead of
depth-wise. The algorithm is as follows:

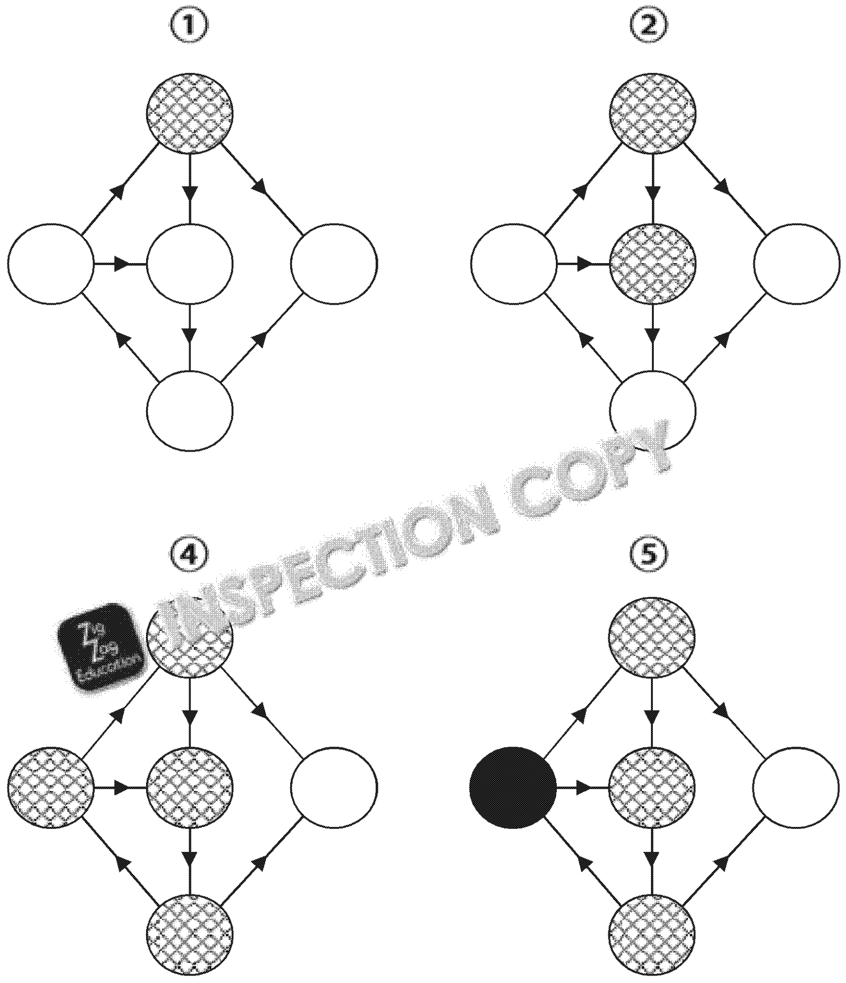
1. Select a start vertex and colour it (cross-hatch) to indicate that it has
2. Visit a new vertex adjacent to the current vertex that has not been vi
3. Colour the visited vertex (cross-hatch) and visit a new vertex that is a
4. Repeat these steps until there are no more adjacent vertices to visit.
5. Colour the vertex black to show that it has been completely discovere
6. Visit an adjacent vertex that hasn't been visited and colour it grey.
7. Repeat these steps until all the vertices are coloured black.
8. If there are any remaining vertices that haven't been coloured due to unreachable, set the start vertex to be one of the unreachable verte until all vertices are black.

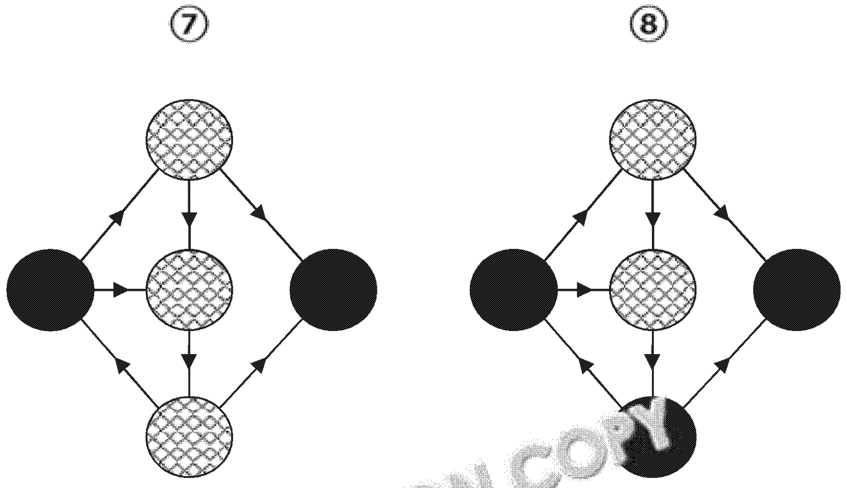
Performing a depth-first search on a graph

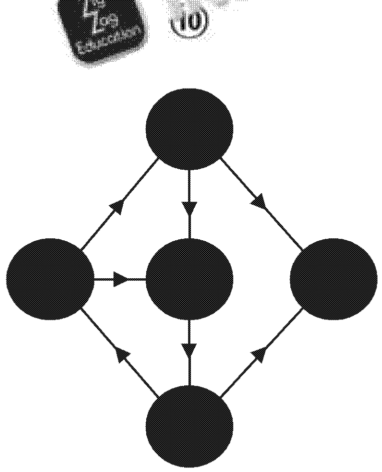
Here is an example of the steps taken during a depth-first traversal of the following graph on the right.

▨ indicates a node which has been **visited**

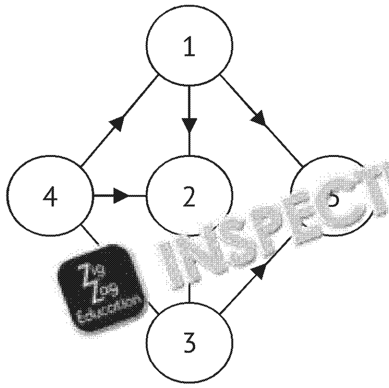
■ indicates the node has been **completely explored**.







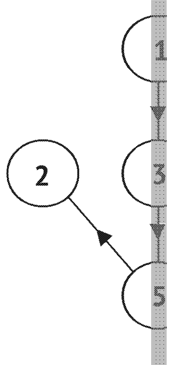
The order that the nodes have been traversed in is:



Note: depending on which node you pick, you may get a different output! How would the result change if you picked the right node instead of the left in Step 4?

Questions: G

1 Consider the



What order w
starting from

a) A breadt

b) A depth-

3.2 TREE TRAVERSAL

Searching a binary tree (O(log *n*))

Since everything is organised in a hierarchical manner, searching for an item complicated. It is the same fundamental process as inserting an element. Th are as follows:

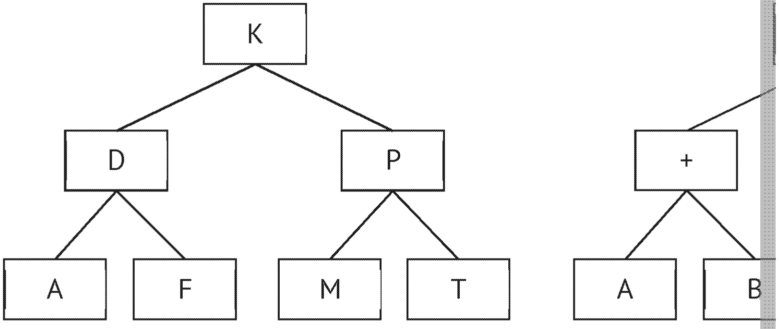
1. Start from root node.
2. If item is equal to the item in the current node then item is found, th
3. If item > the current node, follow the right pointer.
4. If item < the current node, follow the left pointer.
5. Repeat the process for the next node until Null pointer is found.
6. If Null pointer found, then the item is not in the tree (but could be ad

Other tree traversal algorithms

An algorithm that *traverses* a tree will perform an action on all nodes in the t node is [illegible]), even it if is only to compare the node to another to find the l *pre-order*, *in-order* or *post-order*. Note the use of recursion in the procedures algorithms much more elegant. It is extremely hard to do these operations w

Suppose the following letters have been added to the first tree where letters stored to the left, and letters larger than the root node are stored to the righ the algebraic expression A + B * C – D has been stored in the second tree wi the operators to the left and right of the root node.

The trees would now look like this:



Suppose that each tree is stored in three arrays called data, leftPointer and r like this:

	Data	leftPointer	rightPointer
1	K	[illegible]	2
2	[illegible]	5	3
3	T	−1	−1
4	D	6	7
5	M	−1	−1
6	A	−1	−1
7	F	−1	−1

	Data	leftPoin
1	*	2
2	+	4
3	-	6
4	A	−1
5	B	−1
6	C	−1
7	D	−1

Note: −1 means that the specific element has no children tc

Pre-order traversal

Pre-order is when the algorithm looks at the root node, then makes its way c
and then the right-hand side. The first tree above would be listed in pre-ord
tree above would be listed as *+AB-CD.

An algorithm for a pre-order traversal follows, where p is initially the array s

```
PROCEDURE preOrder (p)
    Print (data[p])
    IF leftPointer[p] > 0 THEN
        preOrder(leftPointer[p])
    END IF
    IF rightPointer[p] > 0 THEN
        preOrder(rightPointer[p])
    END IF
END PROCEDURE
```

In-order traversal

In-order is when the algorithm first goes down the left branch, then looks at
right branch. The first tree above would be listed in in-order as A, D, F, K, M,
be listed as A+B*C-D.

An algorithm for an in-order traversal follows:

```
PROCEDURE inOrder(p)
    IF leftPointer[p] > 0 THEN
        inorder(leftPointer[p])
    END IF
    PRINT(data[p])
    IF rightPointer[p] > 0 THEN
        inOrder(rightPointer[p])
    END IF
END PROC
```

Post-order traversal

Post-order is when the algorithm first goes down the left branch, then goes
looks at the root node. The first tree above would be listed in post-order as
above would be listed as AB+CD-*.

An algorithm for a post-order traversal follows:

```
PROCEDURE postOrder(p)
    IF leftPointer[p] > 0 THEN
        postOrder(leftPointer[p])
    END IF
    IF rightPointer[p] > 0 THEN
        postOrder(rightPointer[p])
    END IF
    PRINT(data[p])
END PROCEDURE
```

As you can see, the algorithms are remarkably similar. It may help you to remember which is which by considering the root node value before the left and right nodes (pre-order), after the left but before the right node (in-order) or after the left and right nodes (post-order).

Questions: Tre

1 Construct a bin
following numb
the order they a

`0 -1 1 2 -2`

Write out the o
was traversed i

a) Pre-order (
b) Post-order
c) In-order (1

3.3 REVERSE POLISH NOTATION (RPN)

An arithmetic expression is a form of numbers and operators which represen
write down an arithmetic expression; these include the infix and postfix (Re

Infix notation is what you all use every day. This is when operators and symb
operator is surrounded by two symbols, one on the left and the other on the
expressions. Examples of infix expressions include 5 + 5, 2 * 10 + 2, 16 / 4 +
general use, it requires that brackets be used to determine the order of opera
different to 5+5/2.

In the early days of computing, it was very complicated to program compute
this problem, Reverse Polish notation was invented. This places the operator
on. As you will see, this eliminates the requirement for brackets.

In Reverse Polish notation operators are evaluated from left to right. So the ex
be written without brackets in infix without increasing the number of operatio
written as 2 (3 7 + [illegible] notice that writing it with brackets does not change

Here are some more examples of infix expressions converted to postfix expressions. Note that in every case there are a variety of different orders the expression could be written in:

Reverse Polish notation has another advantage on top of allowing expressions to be written unambiguously without brackets.

Evaluating RPN using a stack

The stack can be used to evaluate Reverse Polish notation by the algorithm giv
is a LIFO data structure; this means that the first thing that is put into the stack

- Go through the input and read one character at a time until there is r
- Whenever a symbol is read from input then push (put) symbol on sta
- Whenever an operator is read from input then pop (get) the last two operation on those two symbols read and store the answer back onto

Example: 1 3 3 2 * + –

Step	Stack Contents
Symbol read: 1	
Push 1 onto stack	1
Symbol read: 3	1
Push 3 onto stack	3 1
Symbol read: 3	3 1
Push 3 onto stack	3 3 1
Symbol read: 2	3 3 1
Push 2 onto stack	2 3 3 1
Symbol read: *	2 3 3 1
Pop last two items from stack (2,3)	3 1

S
Multiply 'poppe
Push answer ont
Symbol read: +
Pop last two iter
Add 'popped' nu
Push answer ont
Symbol read: –
Pop last two iter
Subtract second from first (8)
Store result

Using the stack like this is very beneficial since it allows programs to have s
and lower complexity. It also allows expressions to be effectively infinitely l
list is traversed expressions can be pushed onto a stack until they are neede
to be stored at any time. As expressions are calculated they replace the last

Converting Reverse Polish (postfix) to infix by hand

The easiest way to convert from RPN to infix is to go through each symbol st operator is found, place brackets round the operator and the two arguments would become 5 (4 6 +) – and then (5 (4 6 +) –). Notice how anything in bra

Once this has been done, move the operator in each set of brackets into the n For example: (5 (4 6 +) –) would become (5 – (4 + 6)). You can then remove a

Example – Convert 10 4 2 * + 1 3 2 * + / to infix

Step 1 – Bracket triplets starting from left

```
10 (4 2 *) + 1 3 2 * + /
(10 (4 2 *) +) 1 3 2 * + /
(10 (4 2 *) +) 1 (3 2 *) + /
(10 (4 2 *) +) (1 (3 2 *) +) /
(((10 (4 2 *) +) (1 (3 2 *) +) /)
```

Step 2 – Move operators to middle of brackets

```
(((10 (4 2 *) +) / (1 (3 2 *) +))
(((10 + (4 2 *)) / (1 + (3 2 *)))
(((10 + (4 * 2)) / (1 + (3 * 2)))
```

Step 3 – Remove unnecessary brackets (optional)

```
 (10 + 4 * 2) / (1 + 3 * 2)
```

Example – Convert

Step 1 – Bracket tri

```
(((2 5 *) 9
```

Step 2 – Move oper

```
(((2 * 5) +
```

Step 3 – Remove u

```
(2 * 5 + 9)
```

Converting from infix to RPN by hand

To convert from infix to RPN a stack and pointer can be utilised. The first sta operators.

High	( )	Brackets
	^ or ↑	To the power
	× ÷	Multiply, divide (in computing mu
Low	+ -	Add, subtract

Steps

1. Move pointer to the next part of the infix expression. If expression is fin
2. If it is a number then write it down and go back to step 1.
3. If it is an operator follow the following:

   3a. If the operator is an open bracket (then push onto stack, reset th to step 1.

   3b. If the operator is a closed bracket, pop all the operators in the s and then discard the (from the stack.

   3c. Compare the current operator with the top of the stack. If it is hi the stack and go back to step 1.

   3d. Compare the current operator with the top of the stack. If it is lo the stack and go back to step 3c.

Examples

Infix expression	Step	RPN	Stack	
3+2*5	1	3		Number
+2*5	3c	3	+	+ is higher than nothi
2*5	1	3 2	+	Number
*5	3c	3 2	* +	* is higher than +
5	1	3 2 5	* +	Number
empty	1	325*+		Pop all from stack

3+2*5 is 325*+ in RPN. When evaluated it can be seen we would work out 2*
3+10 = 1

Infix expression	Step	RPN	Stack	
(2+3)*5-(2+4)	3a		(	Push (onto st
2+3)*5-(2+4)	1	2	(	Number
+3)*5-(2+4)	3c	2	+ (	+ is higher tha
3)*5-(2+4)	1	2 3	+ (	Number
)*5-(2+4)	3b	2 3 +	empty	Pop until (the
*5-(2+4)	3c	2 3 +	*	* is higher tha
5-(2+4)	1	2 3 + 5	*	Number
-(2+4)	3d	2 3 + 5 *	empty	Pop from stac
	3c	2 3 + 5 *	-	- is higher tha
(2+4)	3a	2 3 + 5 *	(-	Push (onto st
2+4)	1	2 3 + 5 * 2	(-	number
+4)	3c	2 3 + 5 * 2	+ (-	+ is higher as
4)	1	2 3 + 5 * 2 4	+) -	Number
)	3b	2 3 + 5 * 2 4 +	-	Pop until (the stack
	1	2 3 + 5 * 2 4 + -		Pop until stac

So (2+3)*5-(2+4) is converted to 2 3 + 5 * 2 4 + - in RPN. When evaluating it w expression would become 5 5 * 2 4 + -. The next stage would be 5*5 so the e next we work out 2 + 4 so the expression becomes 25 6 – so 25 – 6 = 19. Ch calculation is correct.

Task: Infix to RPN Converstion

Using the step method demonstrated on the previous page it is possible to
program. Try to write a program that takes an infix expression as a string ar

For simplicity assume that all numbers are integers from 0–9 as per the exa

Questions: Reverse Polish Notation

1 Why is Reverse Polish notation used and why is this beneficial? (1 ma

2 Convert the following into their Reverse Polish notation form. (4 ma

a) `(7 / 6) + 5`
b) `7 * 7 + (6 + 2 + 5)`
c) `45 / 7 + (0 - 6)`
d) `5 + ((1 + 2) * 4) - 3`

3 Convert the following to infix. (3 marks)

a) `a b -`
b) `g h b + -`
c) `b m / g h / +`

3.4 SEARCHING ALGORITHMS

LINEAR SEARCH

The linear search algorithm is used to find a given element in a list by iterati comparing it to the condition element. Linear searches always start at the be condition is met or the program reaches the end of the list. We say that linea of O(n), which means that the time it takes to run is linearly proportional to t *covered in more detail in Section 4.*

Pseudo

```
itemPosition ← 0
itemFound ← FALSE

While itemPosition < ArrayLength AND itemFound = FALSE
     If arraySearch[itemPosition] = itemWanted then
          itemFound ← True
     Else
          itemPosition++
     End if
End While

If itemFound = TRUE then
     OUTPUT "Item found at " & itemPosition
Else
     OUTPUT "Item not found"
End if
```

```
Dim itemPosition As
Dim itemFound As Bo

While (itemPosition
FALSE)
     If array[itemP
          itemFound
     Else
          itemPosit
     End if
End While

If itemFound = TRUE
     Console.WriteL
itemPosition
Else
     Console.WriteL
End If
```

C#

```
While ((Int itemPosition = 0 > arrayLength) && (NOT
isFound))
{
     If (arraySearch[itemPosition] = itemWanted)
     {
          itemFound = TRUE;
     }
     Else
     {
          itemPosition++;
     }
}
If (itemFound = TRUE)
{
     Console.WriteLine("Item found at {0}",
     itemPosition);
}
Else
{
     Console.WriteLine("Item not found.");
}
```

```
Var
   itemFound:boolea
   itemPosition:int
   itemWanted:integ

begin
   itemPosition:=0;
   itemFound:=false
   while (itemPosit
   (itemFound=false
   begin
      if arraySearch
         itemFound
      else
         itemPosit
   end;
   if itemFound the
         writeln('
      else
         writeln('
end.
```

Python

```
itemFound = False
itemPosition = 0

while itemPosition < len(arrayLength) and not found:
     if array[itemPosition] == itemWanted:
          found = true
          break
     itemPosition = itemPosition + 1
     if itemFound == true:
          print('item found at %', itemPosition)
```

BINARY SEARCH

A binary search is a method for searching a ***sorted*** set of elements. The conc

- If the element being searched for is greater than the middle elemen middle element is discarded.
- If it is less than the middle element, then everything greater than th
- If it is the same, then the element has been found!

This process is repeated until either the element has been found or the set c be further divided. Binary searches have a run time complexity O(log *n*).

For example, say you wanted to search for the letter S in the following set, sorted into alphabetical order:
You would start by checking against the middle item, which is F. S is to the right of F in the alphabet, so every letter to the left of F can be discarded. The letter F itself can also be discarded. This leaves the following set:
The middle element is now Y. S is to the left of Y in the alphabet, so every letter to the right of the Y can be discarded, as can the Y. The set now only consists of one letter:
The middle of the set is, now, the only item in the set and it is the letter S. Th searched for, so it has only taken three steps to complete the search. A linear

When searching, the index of the middle of the set is calculated by adding t together and dividing by 2. Any remainder is discarded. For example, if the l rightmost was 19, then adding the two together would be 19. Dividing 19 b 1. Therefore the index of the middle would be 9. Another way of saying this should be rounded down to the nearest integer.

The major disadvantage of the binary search algorithm is its requirement th been sorted. This is a problem because sorting the array may take more tim search! In addition, binary searches are not very efficient when performed o requires going through the whole linked list), although for sets of items tha be searched you would generally use a binary tree instead of a linked list.

Pseudo

```
PROCEDURE BinarySearch (Array, ItemWanted)

Left ← 1
Right ← ArraySize
Middle ← 0
ItemFound ← FALSE
WHILE (Left <= Right) AND (NOT ItemFound)
     Mid ← round_down((Left + Right)/2)
     IF Array[Mid] = ItemWanted THEN
        ItemFound ← TRUE
     END IF
 IF Array[Mid] > ItemWanted THEN
        Right ← Mid - 1
     END IF
     IF Search_Array[Mid] < ItemWanted THEN
        Low ← Mid + 1
     END IF
 END WHILE
END PROC
```

```
Procedure BinarySe
Dim bot As Intege
Dim mid As Intege
Dim top As Intege
Dim isFound As Boo

While (bot <= top)
    Mid = rnd((bo
    If arrayName[
        isFound =
        Break
    Else if array
        Top = arr
    Else if array
        Bot = arr
    End If
End while
End Procedure
```

C#

```
Static Void BinarySearch(arrayName, itemWanted)
{
    Int bot = 1;
    Int mid = 0;
    Int top = arrayName.Length ();
    Bool isFound = False;

    While(bot <= top) && (Not isFound)
    {
        Mid = rnd((bot + top)/2);
        If (arrayName[mid] = itemWanted)
        {
            isFound = TRUE;
            Break;
        }
        Else If (arrayName[mid] > itemWanted)
        {
            Top = Mid - 1;
        }
        Else If (arrayName[mid] < itemWanted)
        {
            Bottom = Mid + 1
        }
    }
}
```

```
Procedure BinarySe
integer; itemWante
var
  bot:Integer = 1;
  mid:Integer = 0;
  top:Integer;
  isFound:Boolean
begin
     top:= length(
     While (bot <=
     begin
    Mid := trunc((
    If arrayName[m
           begin
        isFound :=
        Break;
           end
    Else if arrayN
        Top := arr
    Else
            Bot :
      end;
end;
```

Python

```
def BinarySearch(array, itemWanted):
    bottom = 0
    top = len(array)-1
    isFound = False
    while bottom <= top and (not isFound):
        mid = (bottom + top) // 2
        if array[mid] == itemWanted:
             isFound = True
             return "Found at " + str(mid)
        elif array[mid] > itemWanted:
            top = mid - 1
        elif array[mid] < itemWanted:
            bottom = mid + 1
        return "Not found"
```

Binary search using recursion

A binary search also lends itself nicely to a recursive technique as it effective of the list. This is show in the pseudocode below

```
PROCEDURE Bsearch(min,max,itemRequired);
    mid = (min + max) DIV 2
    IF max < min THEN
        PRINT "not found"
    ELSEIF list[mid] == itemRequired THEN
        PRINT "FOUND IN SLOT " + mid
    ELSEIF list[mid] > itemRequired THEN
        Bsearch(min,mid-1,itemRequired)
    ELSE
        Bsearch(mid+1,max,itemRequired)
```

'max' and 'min' are index values that let us keep a track of what part of the li is the value that we are looking for in the list.

This works by looking at the element in the middle of the list, and seeing wh looking for. If not, it compares the value found against the value that needs again on a smaller part of the list – left of the midpoint if the actual value is +1s and -1s are used to disregard the midpoint, as if we have got this far, it the value that we want!

The terminators are the lines checking whether max is less than min, as if th list (can you see why?), and the lines checking whether the element on the m this evaluates to true, we have found our value!

Time complexity of the binary search algorithm

The binary search algorithm is a big improvement over the linear search algo comparisons a binary search requires is the smallest integer value *x* that sati

$$x \geq \log_2 n$$

This is because the worst-case time complexity of the binary search algorithm algorithm especially efficient when used on large amounts of data because

The table below shows the number of comparisons required for the linear se search algorithm:

Number of elements	Worst case for *linear* search	Worst c
100	100	
1,000	1,000	
1,000,000	1,000,000	

Questions: Searching Algorithms

1 Consider this array:

4	3	1	6	8	9	2

a) Why can't a binary search be performed on this array in its curr

b) Fix the array so that a binary search can be performed. How ma the number 2 (2 marks)

2 Search the following array for the letter 'R' using the binary search m the left, middle and right at each step. The first values are given. (2

Index	1	2	3	4	5	6	7
[illegible]	A	C	E	J	L	O	Q

Left: 1

Middle: 5

Right: 10

3.5 SORTING ALGORITHMS

BUBBLE SORT

This sort runs up the list comparing each pair of data items. If they are the w swapped over. In this way the largest item *bubbles* to the top. Next time rou second from the top item. At the same time the smallest item is moved dow run time complexity of O(n^2). How this is calculated will be described in mo put, if you have n elements to sort, in the worst case you will have to swap t

Suppose this is the original data.

9	17	6	88	28	91	12	3	95

The following table shows the data changing places through <u>one</u> pass.

No change	9	**17**	6	88	28	91	12	3	95
Swap	9	**6**	**17**	88	28	91	12	3	95
No change	9	6	**17**	**88**	28	91	12	3	95
Swap	9	6	17	**28**	**88**	91	12	3	95
No change	9	6	17	28	**88**	**91**	12	3	95
Swap	9	6	17	28	88	**12**	**91**	3	95
Swap	9	6	17	28	88	12	**3**	**91**	95
No change	9	6	17	28	88	12	3	**91**	**95**

When to stop?

The code continues to make passes through the table. It can either do it eig carry on sorting, even if the data is already sorted), or it can make a note of v swapped over at all during the last pass, and stop if it wasn't. To do this, a va time a swap is made, but is set to false before each pass through the data. Th the end of a pass through the data then no swaps were made, which means t sort can finish.

Pseudo	

```
Swapped ← True
Iterations ← 0
i ← 0
temp ← 0

iterations ← 1
While iterations <= array[Length] - 1 AND
swapped
    Swapped ← False
    For i = 0 to array[Length] - 1 - iterations
        If array(i) > array(I + 1)
            Temp = Array (i)
            Array(i) = array(i + 1)
            Array(i + 1) = temp
            Swapped = true
        End If
    Next
Iterations++
End While
```

```
Private Sub Bubble
Dim swapped As Boo
Dim iterations As
Dim i As Integer
Dim temp As Intege

swapped = True
iterations = 1
While ((iterations
swapped)
    swapped = Fals
    For i = 0 To a
        If array(i
            temp
            array
            array
            swapp
        End If
        Next
    Iterations = i
End While
End Sub
```

C#

```
Static Void BubbleSort ( int [] array  )
{
     Bool swapped;
     int iterations;
     int temp;

     swapped = true;
     iterations - 1;
     while ((iterations <= array.length( h ) -
     1) && swapped)
     {
          Swapped = false
          For (int I = 0; array.Length() - 1 -
          iterations)
          {
               If (array(i) > array(I + 1)
               {
                    Temp = array(i);
                    Array(i) = array(I + 1);
                    Array(I + 1) = temp;
               }
          }
     Iterations++
     }
}
```

```
Repeat
   swapped:=false;
   pointer:=1;
   while pointer<l
   begin
     if lst[pointe
     begin
        temp:=lst[
        lst[pointe
        lst[pointe
        swapped:=t
     end;

     pointer:=poin
   end;

 until swapped=fal
```

Python

```
def bubbleSort(myList):
    for passnum in range(len(myList)):
        for i in range(passnum):
            if myList[i]>myList[i+1]:
                temp = myList[i]
                myList[i] = myList[i+1]
                myList[i+1] = temp

myList = [81,21,16,5,44,87,50,43,20]
bubbleSort(myList)
print(myList)
```

MERGE SORT

The merge sort is a recursive technique which follows the basic 'divide and c
to break down the inputs into smaller lists until each contains just a single e
each of them individually, then merge the elements back together so that th
accomplished in two steps: the merge and the *sort*.

The following pseudocode breaks down the two steps further:

```
Function MergeSort(listInput)
    Var left, right, result
    If length(listInput) ≤ 1 Then     # single element is ator
        Return listInput
    Else
        Middle ← length (listInput) / 2  # find middle by divi
        For I ← 1 to middle - 1
            Add I to left # first resulting group becomes 'left'
        End for
        For I ← middle to length(listInput)
            Add I to right     # second resulting group becomes
        Left ← MergeSort (left)       # recursive call; input
        Right ← MergeSort (right)     # Recursive call; input
        ...
        ...
        If last (left) ≤ first(right)     # binary operator c
            Append right to left
            Return left
        End if
        Result ← MyMerge(left, right)    # merge function in
        Return result

Function MyMerge (left, right)

    Var result
    While length(left) > 0 and length(right) > 0    # whi
        If first(left) ≤ first (right)        # comparison st
            Append first(left) to result
            Remove(first(left))
        Else
            Append first(right) to result
            Remove(first(right))
        End If     # end comparison step
    If length(left) > 0
        Append left to result
    Else if length(right) > 0
        Append right to result
    End If
    Return result
```

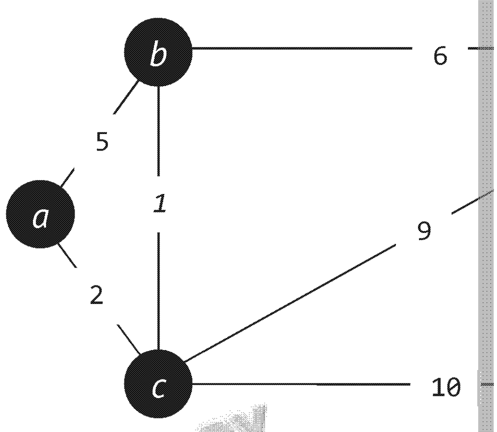
The merge sort has the average complexity of O(*n*log *n*) and the best case co are pre-sorted. It is O(nlogn) because we split at most log *n* times, and each

3.6 DIJKSTRA'S SHORTEST PATH ALGORITHM

In 1956, computer scientist Edsger Dijkstra published his research findings in algorithm which had applications for finding the shortest path between all p outputting the results by removing them from a queue of possible traversals.

```
FUNCTION DijkstraAlg (Graph, Source)

# initialisation steps
Dist (Source) ← 0     # set the start node distance to 0
FOR EACH v in Graph      # for all vertices in the graph
    IF v != source Then      # if the vertex isn't the source no
        Dist(v) ← ∞      # set the distances of that node to 'i
        Prev[v] ← NULL       # set previous node to NULL (or -1
    END IF
    Sum ← V + Q        # Sum the value of all nodes currently in t
END FOR              # results in all vertices that aren't the sou

# the main loop body
While Q != empty                # while the queue isn't empty
    U ← v in Q (min(dist))      # u is the vertex with the min
    Remove u from Q               # remove that node from the
    FOR EACH v neighbouring u       # for all neighbouring nod
        altNode ← dist(u) + length(u,v)    # altNode compare
        IF altNode < dist(v) Then       # if altNode is less t
            dist(v) ← altNode       # then altNode because the
            prev[v] ← u       # previous v is set to u
        END IF
    END FOR
END WHILE

Return dist[ ], prev [ ]    # returns the distance and list of p
END FUNCTION
```

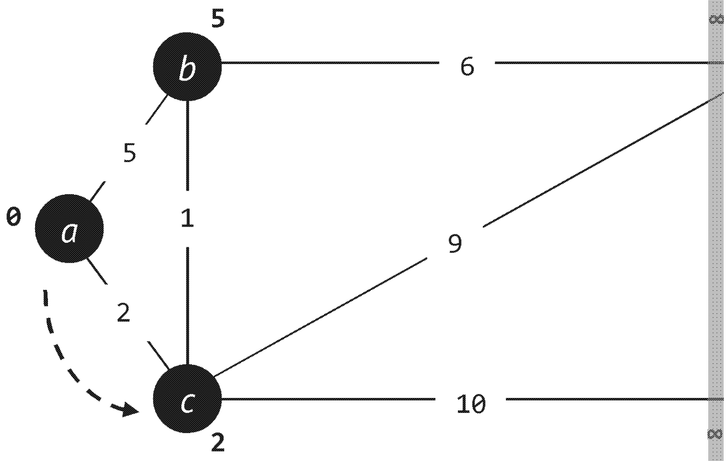
Take a look at the following worked example which will find the shortest path between points *a* and *z* by searching all vertices.



If we want to traverse between the nodes a and z we can use the pseudocode out initialisation stage our source is set to node a and has the value of 0 (that i would equal 0) and all other nodes are said to have the value of 'infinity'. We c

To find the first point, we add the current distance (0) to the distances to the

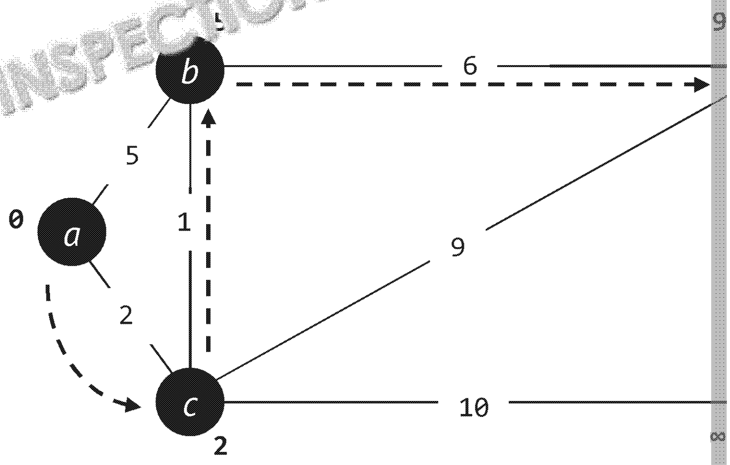
$\vec{ab} = 5$ | 5 is smaller than infinity so *b* is weighted as 5
$\vec{ac} = 2$ | 2 is smaller than infinity so *c* is weighted as 2
$\vec{ac} < \vec{ab}$ so *c* becomes our next node and a is removed from the queue



Now you can repeat the step again with node *c* as your source. For this step you

$\vec{cb} = 1$ | 1 is smaller than 5 so *b* is reweighted as (2 + 1 =) 3.
$\vec{cd} = 9$ | 9 is smaller than infinity so *d* is weighted as (2 + 9 =) 11.
$\vec{ce} = 10$ | 10 is smaller than infinity so *e* is weighted as (2 + 10 =) 12.
$\vec{cb} < \vec{cd} < \vec{ce}$ so *b* becomes our next node and c is removed from the qu

The following node has to be *d* as we have no other choices and becomes w *b* is removed from the queue.



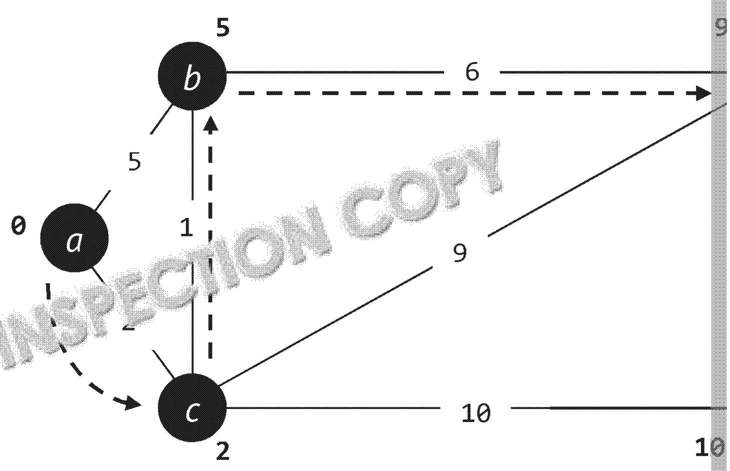
From d you could find z and end the search but you would have missed out

$\vec{de} = 1$ | 1 is smaller than infinity so e is weighted as (9 + 1 =) 10.

$\vec{dz} = 6$ | 6 is smaller than infinity so z is weighted as (9 + 6 =) 15.

$\vec{de} < \vec{dz}$ so e becomes your next node and d is removed from the queue

The final step has to be to node z which has a length of 4 so it is weighted a



So, as you can see, our search algorithm would return:

a, c, b, d, e, z | distance: 14

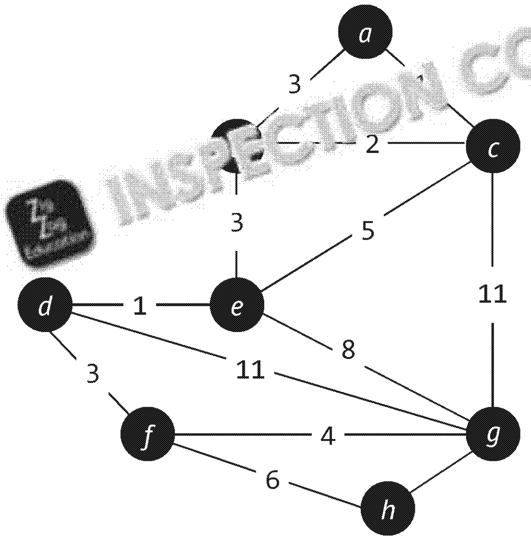
Applications

The most obvious application for the shortest path algorithm is in use with G devices that will find the shortest path between any two physical points on a 3D surface. The algorithm is applied automatically on the website for ease o for the user. More specialised uses include creating a state machine to solve shortest route to achieve a given state, or to determine the shortest time req to achieve said given state.

For example, you could solve a Rubik's cube by making each vertex a state f then solve the cube by searching for the current state of the cube and then t completed puzzle. It can also be used in networking to find the shortest path

Question: Shortest Path Algorithm

1 **Using the shortest path algorithm, trace the result for the graph bel**



ou answer should inclu
shortest path and the to

4. Theory of Computation

The concept of computational thinking is relevant not only in the field of computer scienc
to take a problem, break it down into its components and abstract the information to its s
desirable. This section also shows the comparison of algorithms and Turing machines to s
ability to justify solutions and methods.

This section covers:

4.1 ABSTRACTION AND AUTOMATION

PROBLEM SOLVING

Problem solving is something the human brain is naturally adapted to do; even subconsciously while reading these words you're solving a problem of what the words mean, interpreting what you think they mean and trying to retain the information. How you apply the concept of problem solving to software development is another matter, as the scope for problems is limitless and you can't subconsciously know how to perform every action you will need to carry o

Problem definition

The first step of problem solving is to understand the problem fully; this is c
problem definition, and it give you a stable foundation on which you can buil
understanding of what is required. A well-defined problem will give you an i
understanding of the given, the resources and the end target; however, this r
of the problem domain.

Boundary definition

The next step is *boundary definition* which states what can and cannot be do
problem; these act as constraints and there are a few that apply to almost al
constraints are things such as time, software constraints and equipment avai
all boundaries imposed on a project but do make assumptions that will impo
more facts by asking questions that will give you a specific answer to produc
facts to propose other constraints to the client.

Planning the solution

Stage three is *planning the solution*; in this stage you will ask yourself:

- What resources do I need?
- Are the existing resources adequate for the task?
- How will I use the resources?
- What strategies will I apply?

These all need to be addressed before you can start development. In this chap
and how it can be used to aid in planning; *decomposition* is particularly useful

Automation

Step four is *automation* of the plans that you have generated to complete the
out in a careful manner so as not to produce mistakes that may be costly to c

For more on software design and problem solving see Section 13.

FOLLOWING AND WRITING ALGORITHMS

What *is* an algorithm?

'An algorithm is simply a set of well-defined, step-by-step instructions that s
independent of any specific programming language that can be represented

This can be compared to dialling a telephone number. Knowing someone's t
ring them; however, if you change the input telephone number you change t

You need to be aware that when an algorithm is used in a program the compu
restrictions are on the *syntax* and the *spelling* of the algorithm and are in place
anything that isn't a built-in instruction needs to be explained to the compute
carried out.

Hand trace simple algorithms

A *trace table* is usually used in order to investigate the flow of a simple algor
with a column for each variable, a column for any notes, and a column for th
table may be drawn up for any separate procedure or function that is called.
errors can be found which by simply reading the code could be missed.

Here is an example (very simple program) with the trace table for it:

```
PROCEDURE hello()
INTEGER i
FOR i = 1 To 5
   PRINT ("Hi")
NEXT
END PROC
```

Comments	i	Out
for	1	H
	2	H
	3	H
	4	H
	5	H

Here is another example (a slightly longer program) with the trace table for i
18, 9, 12, 6, 0.

```
PROCEDURE average()
INTEGER sum, howmany, next

sum = 0
howmany = 0
READ next
WHILE next <> 0
   sum = sum + next
   howmany = howmany + 1
   READ next
WEND
PRINT "Average is " & (sum /
howmany)
END PROC
```

Comments	sum
	0
read	
while	18
read	
(while)	27
read	
(while)	39
read	
(while)	45
read	
(wend)	
print	

Pseudocode

Pseudocode literally means 'false code' and is used to allow programmers to
actually programming it fully.

<table>
<tr><td><b>Sequence</b></td><td>Commands are arranged and run sequentially.
<pre>
OUTPUT "Enter an integer: "
a ← READ INPUT
OUTPUT a
</pre></td></tr>
<tr><td><b>Assignment</b></td><td>Operator that assigns a value given to a variable. Unlike
is replaced with '←'.
<pre>
x ← 5
OUTPUT x
</pre></td></tr>
<tr><td><b>Selection</b></td><td>Commands that are executed only if certain criteria are
the IF-THEN-ELSE construct.

<b><i>If-Else</i></b>
<pre>
IF x ← true Then
    OUTPUT "Yes, it's true"
ELSE
    OUTPUT "No, it's not true"
END IF
</pre>
<b><i>Case select</i></b>
<pre>
Select Case letter
    Case letter = "C"
Action ("do something")
    Break
    Case letter = "D"
Action ("do another thing")
    break
    Case else
Action ("do something else")
End Select
</pre></td></tr>
<tr><td><b>Iteration</b></td><td>These commands are repeated in a loop until the exit c
WHILE, Repeat and FOR loops.
<pre>
X ← 7
Y ← 5
While y > 0
    Answer ← answer + (x * y)
    Y ← Y - 1
END WHILE
</pre></td></tr>
</table>

More detail of these constructs can be found in Section 1.1.

ABSTRACTION

Abstraction is a key component in software engineering and is something th generalisation of what something is, how it carries out a task and what the r system that includes only the fundamental characteristics of the problem be represents a complex system in a way that makes it clearer and easier to unc of information.

For example, a satellite navigation system may use the Dijkstra's shortest pa path. However, the route is an abstraction of the real-life problem. The node the vertices are roads. In the model the roads are weighted and shown as str roads contain bends. The actual bends in the road are irrelevant to the soluti important. Therefore the generalisation or abstraction is to 'ignore' the bend weight and a straight line.

These principles are implemented by designers of computer systems in both information is stored. In object-oriented programming languages, abstra structures the classes and objects atomic and efficient. Objects are used bo way of information to simplify the development of complex software.

Another example is a principle in mathematics called *pigeonholing* which is a used as a proof for sorting using sets. For example, if you had 10 pigeons an pigeons would be in each hole? Mathematically, the formula states that if th pigeonhole, then there is going to be either a pigeon without a hole or more

However, with generalisation you can create the following statements:

- If x pigeons are put into y pigeonholes, and $x \leqslant y$, there is always an there's a hole with more than one pigeon.
- If $x < y$ then there are going to be some pigeonholes with no pigeons
- If $x > y$ then there are going to be some pigeonholes with more than

At its heart, generalisation and abstraction allow you to create factual staten There are entire systems that are dedicated to using just these facts in their *knowledge-based* systems where the knowledge is the culmination of the fact

INFORMATION HIDING

Information hiding, as the name suggests, is the process of hiding informatio that is frequently observed in everyday life but is seldom noticed. When you see what files the computer is reading or caching, and this is the first type of information overload. The user of the computer isn't concerned with what fil as the computer turns on as expected; in fact most users have no clue how a notice if the process is slow or fails. Instead of being given a list of boot pro 'splash screen' which indicates that there is a process being undertaken and relevant to the user than a long list of filenames that they didn't know existe

The second type of information hiding is to improve security. There is an und that modularisation is better; it allows the software to be worked on by sepa parts of the program. As long as the teams understand how the modules are the overall task, they are not concerned with *how* the other teams are perfor need access to their data. Variables in modules, known as *local* variables, are that block of code because it isn't needed to interface with anything. This in contained in certain blocks of code is effectively invisible and can't be acces

Did you know?!

A good example of information hiding is the Manhattan Project during World War were, most famously, used on Hiroshima. The lead physicist knew how to build the weaponise the atomic material. The Major General at the time didn't know how to how to weaponise the atomic material for the weapons, whereas the teams that we kept separate, knew what components they were building did but didn't know why the components fitted together.

PROCEDURAL ABSTRACTION

Procedural abstraction is the act of visualising methods by abstracting actual a solution to a problem it is much simpler to focus on what each task/sub-ta actual values from subroutines you're left with a computational pattern – th programmers to focus on how the data should be handled rather than what t procedural abstraction is the use of *dummy* routines that alter the state of a program to continue to work towards the final solution.

For example, when creating a recursive subroutine to calculate the Fibonacc worry about what the values are; you abstract them away and use variables t change and the subroutine will function. The only value you would take into

FUNCTIONAL ABSTRACTION

Functional abstraction is the abstraction of particular computational methods the solution. You've read above that programmers will abstract actual values order to reach function abstraction another stage is needed. Using functiona computations a function undergoes allows the programmer to disregard pro blocks to prevent errors. Just like procedural abstraction, programmers can u an arbitrary value of the correct data type.

Put simply, the purpose of functional abstraction is to describe what method computation while hiding the details of how the computation is performed. F program you can have a method that will perform an update on a class or da in state of health, but will not show how this action is performed.

DATA ABSTRACTION

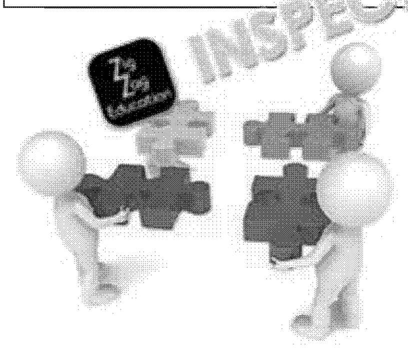
Data abstraction is the next level of abstraction and is where the data type is how the action is performed. For example, if you had a class in a program tha customer you would need a routine that would retrieve the information from An example of a getter method could be the user-defined method GetData( ) was being implemented in the class (binary tree, linked list, an array, etc.) as information in the customer class is returned the function will continue to w advantage is that the method can be altered without changing the code at the instances where the code is implemented. For example, in the above exampl as a linked list, but in an update the class code may be changed to a hash ta abstraction it means that the client code wouldn't be changed and the only t the GetData method.

PROBLEM ABSTRACTION

Problem abstraction relies on the premise that if you continue to remove co be represented in a way that is easier to solve because at some level the pr solved previously.

For example, consider the idea of producing code that works out the first 30 everything other than the idea that you only need to work out at least two f can use recursion to calculate all factorials. Another example that illustrates The system can be broken down into three main parts: the management of t and the loan system. If you look even closer at the abstraction you can see t particular functions such as Add, Edit and Delete. This shows that books are can be saved into a file for ease of storage.

DECOMPOSITION



Decomposition is the act of breaking dow task into a set of easier identifiable subt further subtasks until the problem becor you learn from a very young age; in Matl larger numbers you're taught the *divide* problem, multiplying the two halves and

In computer science, subtasks become *a* further and when all tasks are catered to

COMPOSITION

Composition is the start of removing abstraction by beginning to form compo similar abstraction processes; for example, combining abstract procedures fo to form a more complex compound procedure. This has the advantage of red that form similar tasks and results in a better-structured solution.

AUTOMATION

Automation is the final step – putting all abstractions of phenomena into action to produce the final solution. This is achieved by using the abstractions you've made to design and create the algorithms, which is usually done in pseudocode to begin with.

After the planning is complete programmers will choose a suitable programming language, or it will be in the specification of the problem, to implement the pseudocode into instructions. This step includes planning what data structures are needed to fulfil the specification.

Finally, the code will need to be executed and tested thoroughly.

4.2 REGULAR LANGUAGES

FINITE-STATE MACHINES (FSM)

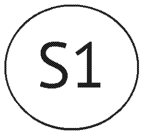
Finite state machines are a simple, intuitive way of capturing real-life events programmers to simplify and formalise the operation of programs. A lot of d designed by using finite state machines. They can also be used as an abstra operation of Turing machines.

A finite state machine is described by:

1. A set of states
2. A start state
3. Possibly a set of final states
4. Transition function/table (dictating which inputs cause which moves
5. Input alphabet (all the possible input events)

State transition diagrams

State transition diagrams are a way of representing finite-state machines gra *transitions*.



States – Represented by circles, states may be labelled anything else which is appropriate as long as it is clea

Transitions – Represented by arrows, transitions may c or may loop back to the same state.

i | o or **i / o**

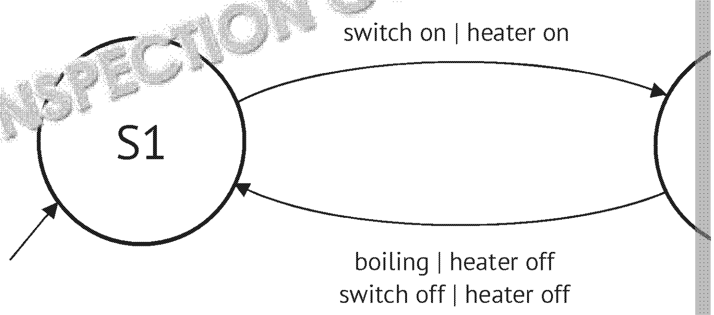
Transition Label with Output – Placed next to a transitio before the line and the output after the line. So in this c

i

Transition Label without Output – Placed next to a transitio with a transition the line (i.e. | or /) is not needed and the i

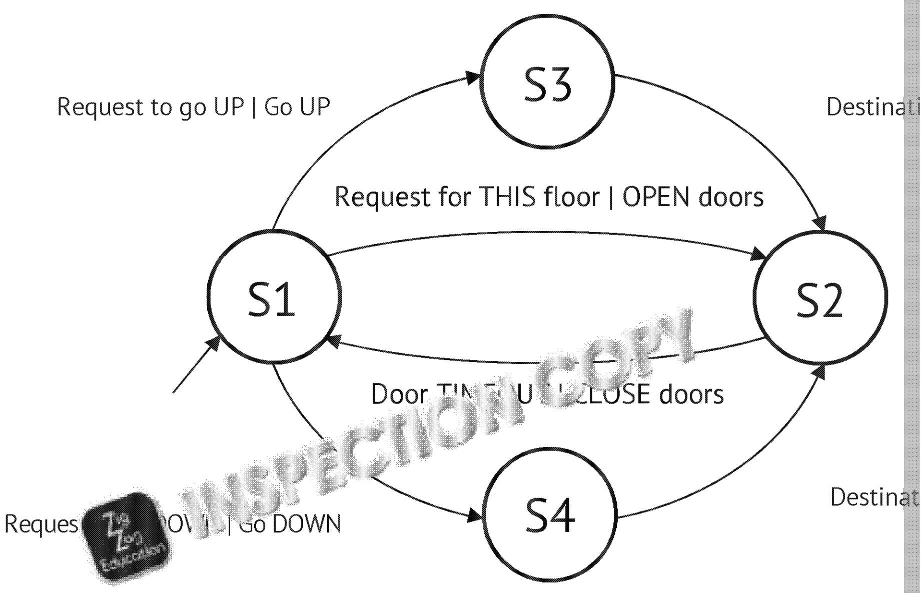
Here is an example of a simple state transition diagram. This finite-state machi kettle has an input alphabet of: {boiling, switch on, switch off}. It has an output

> *Note that at AS level you would be expected to be able to draw a FSM which requi expected to draw machines with both input and output.*



Notice that a transition can have more than one possible input/output comb the initial state has an arrow pointing towards it.

Here is a slightly more complicated finite-state machine. This time it is desig simple lift controller.



State transition tables

A state transition table simply maps input and state combinations to outputs example of a transition table for the kettle:

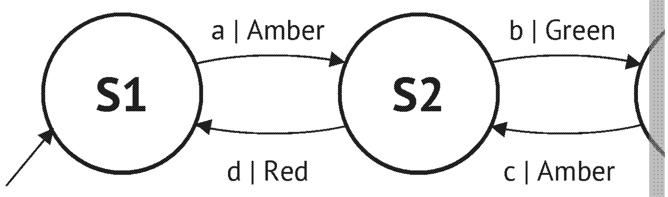
Current State	S1	S2	S2
Input Symbol	Switch on	Switch off	Boiling
Output Symbol	Heater on	Heater off	Heater off
Next State	S2	S1	S1

Here is the transition table for the lift (note that layout isn't too important as l

Current State	Input Symbol	Output Symbol(s)	Next St
S1	Request to go UP	Go UP	S3
S1	Request to go DOWN	Go DOWN	S4
S1	Request for THIS floor	OPEN doors	S2
S2	Door TIMEOUT	CLOSE doors	S1
S3	Destination REACHED	STOP and OPEN doors	S2
S4	Destination REACHED	STOP and OPEN doors	S2

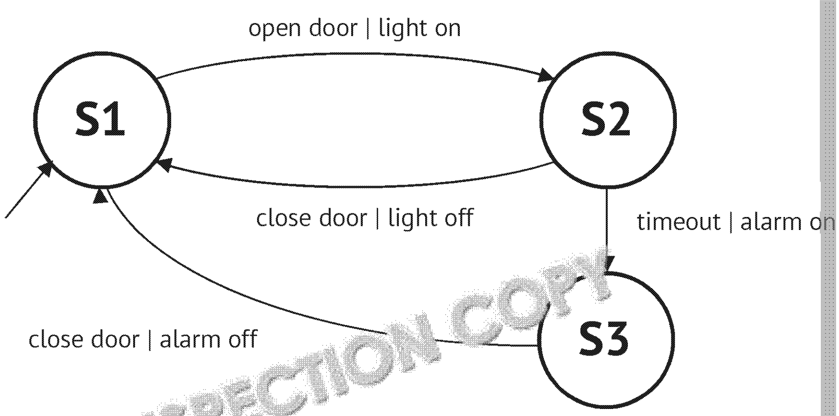
Mealy machines

All of the FSMs shown so far have been Mealy machines. In Mealy machines, outpu (i.e. they are associated with the combination of state AND input). For a traff machine it would look like this:



Questions: Finite-state Machines

1 This is a Mealy machine which turns on a fridge door light when the fr
opened and sounds an alarm if it has been open too long:



a) Give the input alphabet for this machine.
b) Give the output alphabet for this machine.
c) Complete a state transition table for this machine.

2) You are tasked with designing an automatic door system. The doors n
been triggered. The doors need to close if the sensor has not been tri

Design a Mealy machine which will do this task. The input and output

```
Input alphabet = { sensor triggered, timeout }
Output alphabet = { open doors, close doors, reset t
```

MATHS FOR REGULAR EXPRESSIONS

Before you can fully understand the concept and workings of regular express
concepts that you must come to terms with. This includes everything from se
compact representations of sets, the different types of sets and the operatio

Set creation and declaration

Several programming languages support the creation of sets as built-in data t
declaration, the default set value of any type is given the value of Ø meaning
numbers you must use curly brackets to denote the values of the set. For exa
natural numbers then…

```
A = {1, 2, 3}
```

However, sets can also be written using *set comprehension* rules as:

A = {n | n ∈ ℕ ∧ n ≥ 1}

In the expression above the pipe symbol '|' means 'such that', '∈' means '*in*' a
reads 'A is the set of numbers that are in the set of natural numbers and are

As well as the objects that make the set, there are some other concepts that

- *Finite* sets are those sets where the values can be counted using *natu*
value, i.e. a set of 30 objects would be counted from 1 to 30.
- *Infinite* sets are those where there is no end value if the range is not d
natural numbers and the *real* numbers are both examples of infinite se

- *Countable infinite* sets can be counted using natural numbers in a one you can count off all the elements in the infinite set, which, although time, you can index a number using its natural number match, i.e. if y −7...} it is clear that it will continue forever, but you can still count th
- The *cardinality* of a finite set is simply its size and is denoted using the s symbols, i.e. the declaration is the set above, A = {1, 2, 3} has the cardina

You can also make new sets out of a set, where you can make bigger sets by and you can make smallers sets by removing elements from the original set. original set. An example of a subset of the natural numbers is {1, 2, 3}.

We write this as:

$$\{1, 2, 3\} \subset \mathbb{N}$$

... to indicate that it is a proper subset, which means that the two sets are no

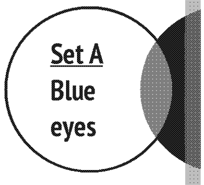
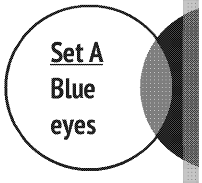
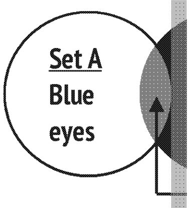
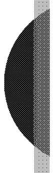
The Cartesian product is found by 'joining' two sets together. If we take the s then the Cartesian product, denoted A × B, is the set

```
A x B = {(a,b) | a ∈ A, b ∈ B}
= {(0, 4), (0, 5), (0, 6), (1, 4), (1, 5), (1, 6), (2,
```

Set comprehension is fairly straightforward with Python – the above exampl

```
A = [0,1,2]
B = [4,5,6]
AxB = [(a,b) for a in A for b in B]
print(AxB)
```

Set operators

Operator	Res
Union All of the people who have *either* blue eyes or brown hair A ∪ B = {x | x ∈ A V x ∈ B} Union (Brown hair, Blue eyes)	
Difference All the people that have brown hair but do not have blue eyes A − B = {x | x ∈ A ^ x ∉ B} Difference (Brown hair)	
Intersection All of the people that have *both* blue eyes and brown hair A ∩ B = {x | x ∈ A ^ x ∈ B} Intersection (Brown hair, Blue eyes)	
Membership Is a given individual who is a member of a certain set Membership (Fred Bloggs, Brown hair) = true	

REGULAR EXPRESSIONS

Over the course of your programming career, you'll notice that one of the mo text manipulation and pattern recognition. In *Section 1.1* we've already seen programming, but there are limitations to where this can be used and it does functions *(see Section 1.2)*. This is where you can use *regular expressions*; this flexibility for pattern recognition.

You can use regular expression to concisely find whether a string is formatte contains a value that you'd like to find; it can be used while reading files in t possibilities of applications are nearly endless. However, regular expression become complicated to read, understand and implement. *For information on r expression statements, see Section 4.2*. It is important that you know how to cr adapted to the need by combining notations.

REGULAR LANGUAGE

A regular language is one that will be accepted by finite-state machines. Thi notation to produce a set of rules to which the language will adhere to. This where the language is comprised of its two components: its alphabet and its alphabet is the finite set of symbols that are used and the language's syntax ordered. *A regular language has no rules governing semantics – the meaning of*

Regular expression notation

As regular expressions describe a set of infinite strings it's not possible to ide all valid strings using a rule, and this is where regex notation is used.

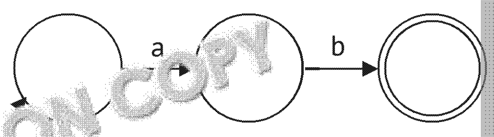
Regex notation	Meaning
a	This regular expression matches a string comprised of ju
b	This regular expression matches a string comprised of ju
ab	This regular expression matches a string comprised of the symbol 'b'.
a*	This regular expression matches a string comprised *zero*
a+	This regular expression matches a string comprised *one*
abb?	This regular expression matches a string comprised of 'a there are zero or one of the symbol it follows.
a | b	This regular expression matches a string comprised of th
[a-z]	This regular expression shows a range. It includes any le lower-case and
^[a-z]	This regular expression shows negation by using '^' – NO
[a-z&&[^q]]	This regular expression shows union and negation. The & add a condition. It reads a to z and not q.

Regular expression to finite-state representation

It is important that you know how to use these notations and are able to bui representation of the rules the statement is built for. These are a way of mak notation more comprehensive and understandable.

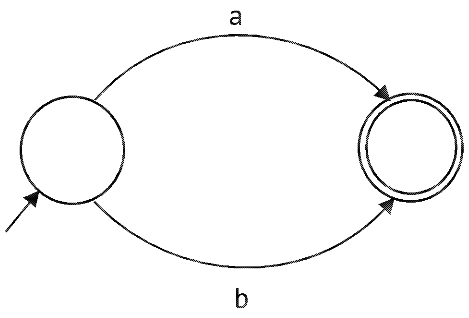
Simple statements

Simple statements where there is no function used are the most simple to re one symbol or string is followed immediately by another without an AND, OF represented with three states and the transitions are labelled accordingly. In example for the expression *ab*.



OR function (|)

When you have an OR expression it is represented by two states with two ar the [illegible] you've seen previously. The transitions are labelled accordingl expression *a* | *B*. As you can see the beginning state can either transit throu



The multiple function (*)

When you have a statement that states that a string will contain one or more symbol you use the multiple function (*). To represent this function you use a with a transition that loops back to the state and label it accordingly. Here y example for the statement *a**.

Piecing it all together

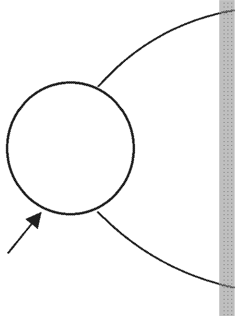
You can now start creating the representations for regular expressions by lo the diagrams above to contribute to the statements diagram. You can also u is valid with the regular expression rules.

*Consider the expression (a|c)d**

We can create the finite-state representation of the expression by breaking i

We can see by looking at it that there is a single decision (OR statement) wh show that the bracket happens first (i.e. a|c), directly followed by the symbol

Therefore this could be represented by the diagram shown below.



You can use this diagram to verify whether strings would be valid or invalid.

Expression	Valid?
ad	Valid
accd	Invalid
cddd	Valid
cad	Invalid

Questions: Regular Languages and Expression Notation

Consider the following sets:

```
A = {1, 4, 6, 8, [illegible]        B = {2, 3, 5, 6, 8, 10}
```

1 [illegible] set operator needs to be applied to the two sets to check what [illegible] numbers are they? (2 marks)

2 What values are retrieved by the expression 'B-A'? (1 mark)

3 a) Write a regular expression that retrieves all characters apart from

b) What would the expression return for the following string? (1 ma

```
the quick brown fox jumps over the lazy dog
```

4.3 CONTEXT-FREE LANGUAGES

BACKUS–NAUR FORM (BNF)

Backus–Naur form (BNF) is a notation which is used to create context-free g natural language; these rules are what govern the syntax of a language. A na naturally and is used as an everyday language with syntax rules that govern construct expressions. These rules are based on declarations and definitions are created; these phrases are in turn are made of definitions of the constitu that can be used to construct or check whether strings/expressions are valid this would be basic English grammar for creating a list of items using ',' betw final item in the list. 'Egg, milk & butter' would adhere to the rules of the lar milk and butter' would not.

Declarations and Definitions

The basic structure of a BNF statement consists of a *meta-component* (the thi The meta-component is enclosed in angle brackets ('<>'), and comes first in a symbol, which indicates that the following statements are the definition of t as follows:

```
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Similar to t the pipe sy expression reads that

Recursion can also be used to define an entity in terms of a previously defin

```
<integer> ::= <digit> | <integer> <digit>
```

This state a single c a digit. Th classed a

By utilising the recursion the integer can be any length of digits so we could

```
    <real> ::= <integer>.<integer>| <integer>
```

Even though the <real> definition contains no recursion, the recursion in <int any number of digits before the decimal point and any number after.

BNF is also used to define operators, and indeed also entire expressions, fun operator and an arithmetic statement in a simple language may look like thi

```
<operator> ::= + | - | * | / | %
<statement> ::= <numericvariable> = <integer> <operator> <in
```

Limitations of regular expressions

We say a string has well-formed brackets if they match up correctly; for insta "( )) (" is not. We want to know whether a string of brackets is well formed o cannot be solved with just regular expressions, and this is an example where problem cannot be solved with regex, because it does not easily deal with re

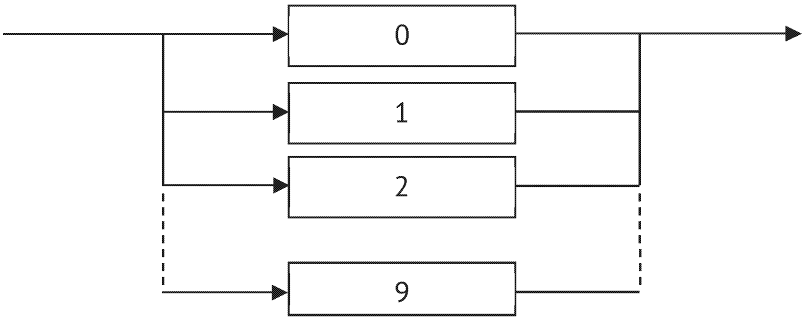
With BNF this is much easier, as the language of well-formed brackets can b

```
<lbracket> ::= (
<rbracket> ::= )
<string> ::= <string><string>|<lbracket><string><rbracke
```

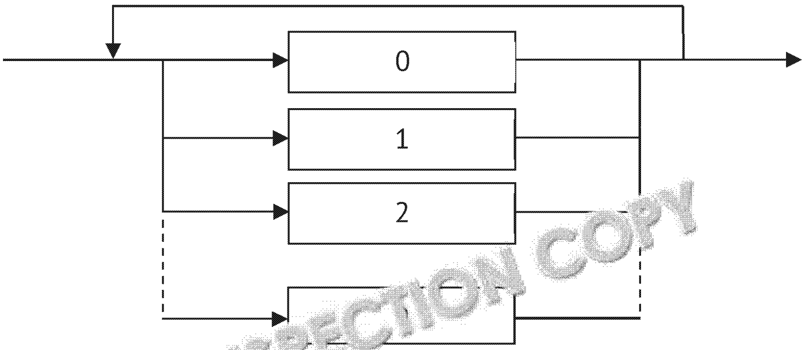
So we can simply check whether the string can be made using these rules.

SYNTAX DIAGRAMS

The syntax of a language can also be represented through syntax diagrams. language is defined by using entities and arrows, and any path through the c could be other components (the equivalent of something enclosed in angle b The arrows direct a path through the entities.



Here is definiti page. T the ope of recur



This is a synta on the instanc they ca

Syntax diagrams can be very useful when trying to visualise complex BNF st define every aspect of a programming language but the result is an intensel become lost. They aren't without their drawbacks; they take up a lot of room are not suited to being input into a computer.

Questions: Context-free Languages

1 **Study the declarations below.**

```
Sentence ::= Noun Phase Verb Phrase ;
S ::= NP VP ;
NP ::= DET Noun | Name ;
DET ::= ‘The’ | ‘A’ ;
Noun ::= ‘Hippo’ | ‘Chair’ | ‘Animal’ ;
Name ::= ‘Keith’ | ‘Chris’ | ‘Mark’ ;
VP ::= ‘Sits’ | ‘Shouts’ | ‘is’ ADJEC | ‘is’ NP, ‘has’
ADJEC ::= ‘long’ | ‘blue’ | ‘funny’ ;
```

Are these sentences valid in regard to the declarations above?

a) The Pen is an animal (1 mark)
b) Josh is funny (1 mark)
c) Mark is a long animal (1 mark)
d) An animal has a hippo (1 mark)

4.4 CLASSIFICATION OF ALGORITHMS

COMPARING ALGORITHMS

It is important to be able to compare algorithms as there may be multiple al a problem and they may vary considerably in their speed and use of memory is referred to as its *time complexity*. The relative amount of memory an algori *complexity*. Combining these two forms of complexity gives us the algorithm

The time comparison is interlinked with the space comparison because, in mos advantages of both. We will often choose to improve one and as a result the ot we were finding the factors of the numbers 1 to 100, we could do this by using integers. It is at this point that you would have to make the decision to optimis

For space, you could make the computer forget all the previous calculations it factor is required the computer will have to calculate the factor again which ta optimise it for time you could make the computer store all previous factors fo 64 has factors 1, 2, 4, 8, 16, 32 and 64; if we had saved previous factors w all the factors of 32, so they would not need to be calculated again.

When algorithms are very simple and contain very few instructions it is easy instructions to work out the time complexity. For more complex algorithms, time-consuming and sometimes impossible when there are a number of min cases it is sufficient to calculate the complexity of the algorithm based solel to the run-time / memory use. This operation is referred to as the *basic opera*

MATHS FOR UNDERSTANDING BIG O NOTATION

Not all algorithms run with the same speed for all inputs. Some algorithms r more slowly with other inputs. Big O notation is the analysis of an algorithm considering the *worst-case scenario* and provides a notation for the *upper-bou* captures the speed of an algorithm for an input that gives the worst speed o an algorithm expressed in big O notation is often called the algorithm's *orde*

To convert time functions into big O notation, take off the term from the fun number. For instance, if you had the terms n and n^2, the larger of the two is r result obtained by n^2 is much larger than n.

Big O rules:

```
O(k) = O(1)
```

Constant times are expressed as O(1).

```
O(kT) = O(T)
```

Constants inside a function are ignored.

```
O(T) + O(J) = O(T + J) = max(O(T), O(J))
```

When adding two functions together, the bigger of the two functions is chos

```
O(T)O(J) = O(TJ)
```

The product of two separate functions gives the product of functions inside

Expressing complexity

Measuring the complexity of an algorithm is not as straightforward as it may
measuring it is for the algorithm to be written in a programming language a
timed. However, the timings generated by this method are dependent on the
efficiency of the programming language. Therefore this is a crude way to me
complexity. Instead it is better to measure the speed of the algorithm based
requires to be carried out.

Example 1 – Two algorithms with different complexities

Consider the following problem. You are given a 3×3 grid. There are three X
in the same row or column as another X. For example:

	1	2	3
1	X		
2			X
3		X	

The task is to locate the positions of all the Xs in the grid. So in this case th
right (1,1) (2,3) and (3,2).

There are at least two ways to solve this problem. The first way is simply to
grid to find the X. Here is some pseudocode for this algorithm:

```
FOR n = 1 to n = 3
found ← false
i ← 1
WHILE found = false
    IF grid[n,i] = 'X'
        ans[n] ← i
        found ← true
    ELSE
        i ← i + 1
END
END
PRINTLINE "The answers are:"
PRINTLINE "(1," + ans[1] + ")"
PRINTLINE "(2," + ans[2] + ")"
PRINTLINE "(3," + ans[3] + ")"
```

For the grid above this will output:

The answers are:
(1,1)
(2,3)
(3,2)

Assume that the instructions involving comparisons with the grid take by far t
are therefore the key measure of the speed of the algorithm. This might well
example, held on a hard disk. This instruction, which is the key factor in dete
is called the *basic operation*. Using this algorithm, six comparisons will always
solution, no matter how the grid is laid out. Is there an algorithm which will
The answer is yes. This is because not only do we know that there is only one
is only one X per row. This means we can disregard the rows where we have

One way to implement this is to use a FIFO stack to store the rows. In pseud
look like this:

```
STACK.PUSH(1)
STACK.PUSH(2)
STACK.PUSH(3)
FOR n = 1 to n = 3
found ← false
WHILE found = false
    i ← STACK.POP()
    IF grid[n,i] = 'X'
        ans[n] ← i
        found ← true
    ELSE
        STACK.PUSH(i)
END
END
PRINTLINE "The answers are: "
PRINTLINE "(1," + ans[1] + ")"
PRINTLINE "(2," + ans[2] + ")"
PRINTLINE "(3," + ans[3] + ")"
```

The number of comparisons in the above grid will now only be four! Howev
which uses a different number of comparisons for different grids. Below are
comparisons now required:

	1	2	3
1	X		
2		X	
3			X

3 comparisons

	1	2
1		
2		X
3	X	

6 compar

These two situations are called the *best-case* and the *worst-case* complexities
algorithm has a best-case complexity of three comparisons and a worst-case

The second algorithm has a better *time* complexity than the first on average.
complexity? Well, the second algorithm uses a stack, which the first doesn't
that the second algorithm has a worse *space* complexity than the first as it u

Of course, in this case the difference is minor, but what if the algorithm was
100,000×100,000 grids? Also, for this example we have assumed that the st
amount of time compared to the comparisons; what if in reality they don't?
average run a lot more slowly than the first if this were the case. Utilising m
necessarily speed up an algorithm.

Calculating the execution time of algorithms

When calculating the time complexity of an algorithm it is rarely sufficient si
operations. Usually algorithms contain conditional statements and, more im
times these loops repeat is often related to the *size of the input*, denoted by *r*
may loop through every element of an array; obviously, the larger the array,
through it will take to execute.

When statements are simply evaluated sequentially by the computer, we say
constant amount of time to execute, usually denoted by c or k. Example 2 sh
sequence of simple statements expressed in this way.

Example 2 – Simple statements (i.e. the input size cannot vary)

Operation	Time
A ← 3	k
B ← 2	k
C ← A	k

Overall time taken is 3k

In comparison to simple statements, each statement inside a FOR loop is exe
condition is executed *n + 1* times since it requires one extra check to see wh
satisfied. To elaborate on this point, look at example 3 and assume that *n* wa
iteration was taking place; *j* must be checked once more to see whether it is

Example 3 – Single FOR loop

Operation	Time
For j = 1 to n	n + 1
A ← j + 1	n
End for loop	

Overall time taken is T (n) = n + 1 + n = 2n + 1

Nested FOR loops are slightly more complex to work out. Generally the inne
number of times dictated in the outer FOR loop. Thus the execution times w
multiplied by those of the outer FOR loop. Example 4 gives an example of a
help to make this idea clearer.

Example 4 – Nested FOR loop

Operation	Time
For j = 1 to n	n + 1 times
For I = 1 to m	(m + 1) * (n + 1) times
[illegible] ← [illegible] n	m * n times
End for loop	
End for loop	

Overall time taken is

Note that the constants such as 1 are not important in the long run.

For example, assume that n = 5000 and m = 5,000. There is not much differe
expressions, 5000 + 1 + (5001) * (5001) + 5001^2 and 5000 + $(5000)^2$ + 5000^2.

Due to the small difference the constants can be ignored since they do not a

ORDER OF COMPLEXITY

Complexity is often split up into general classes which make it easier to com
complexity of different algorithms. Four such classes listed from most compl

Complexity	General Form	Examples		
Exponential	$O(a^n)$	$O(2^n)$	$O(3^n)$	$O(4^n)$
Polynomial	$O(n^a)$	$O(n^2)$	$O(n^3)$	$O(n^4)$
Linear	$O(n)$	$O(n)$ $O(2n)$ $O(2n$ –		
Logarithmic	$O(\log n)$	$O(\log n)$		

- *Exponential time* algorithms are algorithms that take a very long time
Sometimes even the universe might end and computers would still b
example of an exponential time algorithm is the travelling salesman
[illegible] increases exponentially when the number of cities is increased
- *Polynomial time* often arises when nested loops are used, for exampl
such as bubble sort.
- *Linear time* usually arises from algorithms which go through data ste
structure with *n* elements and you go through it step by step, you en
- *Logarithmic time* arises when an algorithm is designed in such a way
increase as fast as the input size. In an ideal world all algorithms wo
it is not generally possible to achieve. For example, could you sort ar
every element in the array at least once?

LIMITS OF COMPUTATION

While the development of computer hardware and software has meant that r
by computer have become reality, there is a limit to what is actually able to b
problems are where there is no algorithmic solution. For example, artificial i
computation; we may model behaviour using a series of algorithms but is th

Another example is paradox-type problems in which there is no decidable so
only window cleaner in a village. All the windows are cleaned by people in t
their own windows or let the window cleaner do it. The window cleaner is th
people's windows. The window cleaner cleans all and only the windows that
themselves. Does the window cleaner clean his own windows?'

It is useful to classify problems into algorithmic or non-algorithmic to deterr
solved using a computer.

CLASSIFICATION OF ALGORITHMIC PROBLEMS

An algorithmic problem with a finite set of inputs will always be solvable. Th
comput[illegible] (see below) but a solution that can be mapped from a set of inpu
yes/no can be solved by algorithm. The simplest method of understanding th
to the appropriate answer.

A problem that has an infinite set of valid inputs causes more problems as so
others may not.

One classification of algorithmic problems can be determined by its time co
solved with *an* polynomial time complexity or less, i.e. $O(n^a)$ or less, is known

Any problem that has no polynomial time complexity or less is called *intract*

COMPUTABLE AND NON-COMPUTABLE PROBLEMS

Correct solutions can always be found for a solvable problem using an algori solvable, however, does not mean that they are computable in a reasonable solvable in less than infinite time. Unsolvable problems are those that cann will produce the right answer all the time, or problems that might take an i

A problem which can't be solved by an algorithm

Legendre's conjecture is a seemingly simple problem, which so far has not b there exists at least one prime number, *p*, between every n^2 and $(n+1)^2$. In oth exists a prime *p* where $n^2 < p < (n+1)^2$.

One approach would be simply to loop through all the values of *n* and see w and $(n+1)^2$. However, there is an infinite number of values of *n*! This means t infinite, so a solution would never be found. Of course, it would be possible number of values of n. However, this would simply mean that the conjecture be proven that it was true.

Be wary of unsolvable problems; they are often far better disguised than thi waste a lot of time trying to come up with an answer when a workaround w

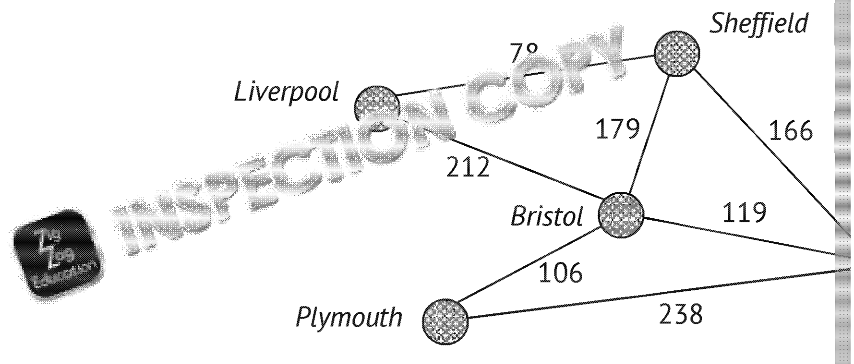
Untraceable problems

Untraceable problems are problems which can be solved by a computer, but w reasonable amount of time for large inputs. A reasonable amount of time is g or less (e.g. O(n), O(log n), O(n^4)), so any algorithm that takes more than this i algorithms therefore have an exponential order of complexity.

An heuristic approach may be taken to help solve some untraceable problems. has been 'guessed' can be checked in polynomial time. In other words, given a the problem becomes tractable. When this is the case the problem is referred t

The travelling salesman problem

The travelling salesman problem is a well-known problem that is difficult t is this: a salesman is trying to get through many different towns across the c he can take that will pass through every town and go through each town on graphical representation of the travelling salesman problem.



Example paths:

London ⇨ Plymouth ⇨ Bristol ⇨ Sheffield ⇨ Liverpo
London ⇨ Sheffield ⇨ Liverpool ⇨ Bristol ⇨ Plymout
Liverpool ⇨ Bristol ⇨ Sheffield ⇨ London ⇨ Plymout

The most obvious solution is to try every possible path, compare all the tota simply the shortest path which meets the criteria above. In computer scienc to as a *brute force* method. The issue with this approach to the problem is th solve. As already explained, this means that the algorithm takes unfeasible a inputs (i.e. large numbers of cities). It is, however, a perfectly reasonable app a small number of cities (as in the diagram above), the advantage being that shortest path possible.

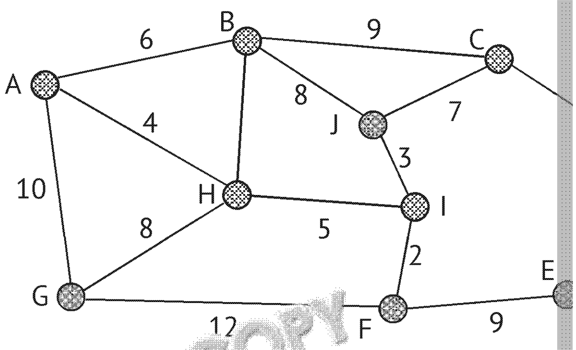
Exponential growth of complexity

One of the natural consequences of the growing complexity is described by e routes have to be taken into consideration and calculated. Taking a look at t there are five cities between which all possible routes must be calculated, us

Number of cities	Number of permutations
2	2
3	8
4	24
5	3,628,800
…	…
10	2,432,902,008,176,640,000

Nearest neighbour heuristic approach

A simple method of solving the travelling salesman problem for larger numb 'nearest neighbour'.



Select a node to start from, such as A, and choose the nearest neighbour whi the nearest neighbour until all the nodes have been visited. If not all nodes a starting node and start again.

This is simplistic and does not guarantee to finish or give a good solution bu nearest neighbours from A would result in travelling the following nodes: A,

HALTING PROBLEM

By now you'll have tried to produce some code on your own; you might even more advanced functions in your chosen language. Even if you haven't, there at some point, crashed your program. Unfortunately, it isn't possible for the going to loop infinitely. It is impossible to write a program that will detect w because you would need to know the state of the first program, which would the state of the second program.

Alan Turing *(more on on p.25)* used this as proof saying that this could go on would only halt if it halts itself, which as you know is impossible if it has ent

Another example is shown below.

```
Input x
While x > 1
    y ← [illegible]       # modulo-division
    If y = 0 Then      # if y = 0 then x is even
        x ← x / 2
    Else
        x ← 5x + 1
End While
```

By tracing the algorithm you can find the limits of the algorithm and see tha algorithm will halt, if the input *x* is 5 the machine will continue to loop infin higher the machine will crash because it has exceeded the maximum space. have to develop a program to test each number and the program wouldn't kn infinite loop, which is why a second program would be required.

Another way of looking at it is to assume that you have a debugging program program along with its input and is meant to halt the program if it enters inf which checks whether or not the other program has stopped is called the *che* program and its input and waits for the program to return true if the program program failed. Suppose that the program goes in an infinite loop and never for the response keeps on waiting until it obtains one. This will never happe never terminated and so the *checker* is also never terminated since it is waiti

Questions: Classification of Algorithms

1 What is the run time complexity of the followinc algorithm? (count e

```
FOR i=1 to n
    s = s + n
END FOR
FOR i=1 to n
    s = s * n
END FOR
```

2 What is the name given to each of the following orders of complexit

a) $O(n^{10})$
b) $O(2_n)$
c) $O(_n)$

3 If an algorithm has a best- and worst-case scenario, how is the order o

4 What is the key difference that an algorithm with a worse space com an algorithm with a better space complexity when run as a compute

4.5 A MODEL OF COMPUTATION

TURING MACHINES

The Turing machine was the brainchild of Alan Turing during the Second Wo worked on his research into computing and *computability*, a term he formally development of the *Turing machine*. The Turing machine is an abstract conce out any computable algorithm. Turing suggested that…

> ***A number, sequence or algorithm is computable if, and only if, a Turing m capable of computing it.***

Although Turing machines seem outdated in concept, it is an undeniable fac of computing anything that is computable. This means that anything which a machine can do as well, albeit over a greater period of time. It is also useful modern-day computers without having to deal with the complexity of device

A Turing machine is comprised of:

- A tape divided into squares for reading and writing symbols to; the ta the tape is infinitely long
- A head which can move left and right to read and write from the tape
- A transition table which depicts the operations performed by a Turing

A Turing machine is an example of a finite-state machine and therefore has state and a 'next' state (defined within the transition table) which Turing cal will also have an output and an input alphabet made of the symbols to be w

The tape might look something like this, with the leftmost being the beginni

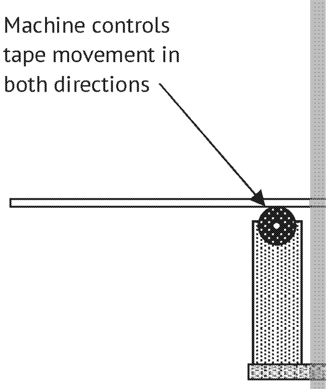
1	#	1	1	0	1	1	□	0	#	1	□	1	#	0

Each square contains one symbol. For example, in the tape shown above '0' a There is also a set of standard symbols which are used by convention to den is a delimiting character, i.e. it is used to separate symbols. The □ character i

Since Turing machines are a type of finite-state machine, the transition funct as it does for any other finite-state machine. The input in this case is the sym head. The output is the symbol to be written to the tape and the direction in function is described in four parts (current state, symbol read, next state, tap machine is in a current state and reads a specific symbol it proceeds to the n operation. These rules are written down and should be followed; if no rule is the Turing machine does not halt, it simply doesn't know what to do!

How does a Turing machine operate?

The input tape contains the input to the Turing machine. The head of the Turing machine starts from the leftmost area of the input and can never go off the tape. A blank symbol is used to the right of the tape to indicate the end of the input stream. The head operates in accordance to the transition table where when an input is read, the transition table depicts what the next operation should be.



How can a Turing machine with a limited number of symbols rep

Turing machines can use strings of symbols to encode other symbols. This is electronic computer where strings of binary numbers are used to represent d example of such an encoding scheme.

Turing machines are often represented using unary, which is the simplest for

```
1, 11, 111, 1111, 11111
```

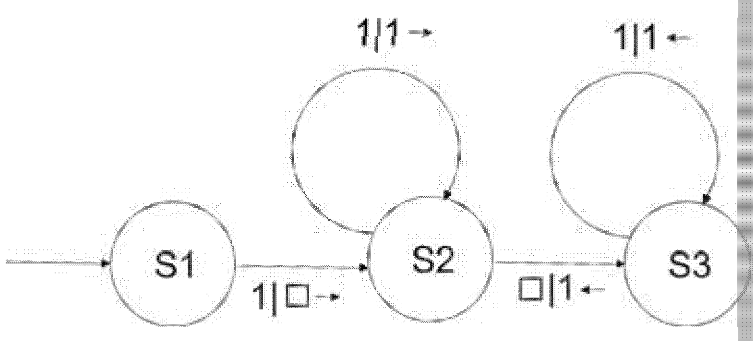
So to have two numbers of the tape, such as 5 and 8, we have:

```
11111 11111111
```

As spaces are hard to see, we represent them using □ so our numbers look li

□11111□11111111□

When programming a Turing machine we assume the pointer starts and finis build a finite-state machine to perform the operations.



This Turing machine adds two unary numbers by removing the first 1 and mo moving the pointer back to the start of the number.

Written in transition rules this would be:

```
δ(S1,1) = (S2,□,→)
δ(S2,1) = (S2,1,→)
δ(S2,□) = (S3,1,←)
δ(S3,1) = (S3,1,←)
δ(S3,□) = (Stop,□,→)
```

Where the rule δ(S1,1) = (S2,□,→) *means if we are on state 1, and re move the read head right finally moving to state 2.*

Common way of using Turing machines

A common example of the use of a Turing machine is to validate an input stri to describe a Turing machine that would recognise the language x#x#....x#x# input string could contain a zero or one followed by a # followed by a zero or

How would this Turing machine work?

Let the Turing machine start from the left-hand side and read the first symb machine would enter a reject state; if the symbol is a 0 or a 1 it would conti next symbol read is #, then it would continue reading; if it is something diffe state. The machine would continue reading the input by moving the head to 0 or a = 1 is preceded by a # symbol.

Universal Turing machine

Alan Turing realised that a Turing machine could be extended so that it co first part of the tape, carrying out each one on the rest of the tape as require could simulate the actions of any other Turing machine. He called this the u universal Turing machine concept is important because modern computers o the stored program, or Von Neumann architecture) in that they read in a pro that program as required. If it weren't for this revelation, hardware would ha running on it and so personal computers would not exist.

Questions: A Model of Computation

1 The Turing machine is an abstract concept invented by Alan Turing i

a) Why is the concept still used today? (1 mark)

b) Why is the universal Turing machine concept an important deve

c) Does a Turing machine calculate its next move purely based on Explain your answer. (2 marks)

2 Consider a Turing machine with the following transitions (transitions input symbol, next state, output symbol/move):

```
(S0, 1, S0, >>)
(S0, •, S1, 1)
(S1, •, S2, >>)
(S2, 1, S2, >>)
(S2, •, S3, <<)
(S3, 1, S3, •)
(S3, •, S4, <<)
(S4, 1, S4, •)
(S4, •, S4, •)
```

a) The Turing machine described above is in S0, with the read/writ to by the arrow.

1	1	1	1	1	1	1	□	1	1	1

(i) Which direction does the head first move in? (1 mark)

(ii) How many moves does the head make before it writes a sy

(ii) Does the machine ever enter S4? (1 mark)

b) Describe what happens if the machine is in S4 and reads in a □.

c) Assuming numbers are coded in such a way that 1 = 0, 11 = 1, 11 what is the purpose of this Turing machine? (1 mark)

5. Data Representation

This section explores how computer systems are able to store a variety of different forms of da
area has led to people misunderstanding the complexity with which computers seem to under
solve seemingly 'complex' tasks. How information is stored accurately and securely is a key as

This section covers:

5.1 NUMBER SYSTEMS

System	Description
Natural	Natural numbers are the very first numbers you're taught positive-only integers and belong to set ℕ. *Example:* ℕ = {0, 1, 2, 3...}
Integer	Integers, often called 'whole' numbers, are numbers that decimal component. These are inclusive of negative num the number set ℤ. *Example:* ℤ = {... −2, −1, 0, 1, 2...}
Rational	Rational numbers are those that can be expressed as a fr means that all integers are rational numbers. Rational nu *Example:* ℚ = {... 0.5, 1, 1.5...}
Irrational	Irrational numbers are those that cannot be written as a *Example:* π = 3.14159...
Real	Real numbers encompass all of the numbers sets as a se quantities. Real numbers belong to the set ℝ.
Ordinal	Ordinal numbers are numerical values that hold the posi in an order. Consider a sorted array; the index of each el

Uses of number systems

It is important in a system to use the correct numbering system when perfor
using the correct data type for a variable. The general rule is:

- Natural numbers for counting
- Real numbers for measurements

As real numbers use up considerably more memory due to their accuracy, it
for counting. For this same reason you wouldn't use integers when you need
levels of accuracy (i.e. tracking bank balances).

Question: Number Systems

1 **Why is it considered bad practice to use real numbers for counting an**
measuring? (2 marks)

5.2 NUMBER BASES

Data is inherently difficult to represent, store and display in a computer syst words and symbols, or dots for images, or frequencies for sound, which can b turn can be converted into binary which can then, finally, be stored to memo representation of numbers in various forms. *For more on data representation s*

RADIX AND RADICES

Any value can be represented exactly using any base (*radix*). When writing va in *subscript* to avoid confusion. You need to be aware of, be able to use, and *denary, binary and hexadecimal.*

Denary (n_{10})

The decimal value that we use every day (also known as *denary*) has a radix omitted. It uses *positional numbering*; it uses powers of 10 for each position o

For example, the number 947_{10} can be represented as:

$(9 \times 10^2) + (4 \times 10^1) + (7 \times 10^0)$

The number 1747.62_{10} can be represented as:

$(1 \times 10^3) + (7 \times 10^2) + (4 \times 10^1) + (7 \times 10^0) + (6 \times 10^{-1})$

Binary (n_2)

Binary representations are made up of groups of bits *(see p.5)* to convey a va Similarly to denary, binary uses *positional numbering* but instead of powers o writing numbers in binary it is often useful to write down the values of each

For example, 86_{10} can be written in binary as:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
0	1	0	1	0	1	1	0

Another example: 200_{10} can be written in binary as:

2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
256	128	64	32	16	8	4	2	1
0	1	1	0	0	1	0	0	0

Hexadecimal (n_{16})

Digital systems store everything using binary values, but humans find it hard such a[illegible]til requires 8 bits to represent in binary. In order to make it eas content of memory or make changes to a file, binary numbers are often grou displayed using a hexadecimal value. The hexadecimal number system is bas and the letters A to F to represent the numbers 10 to 15.

Base 10	0	1	2	3	4	5	6	7	8	9
Base 16	0	1	2	3	4	5	6	7	8	9

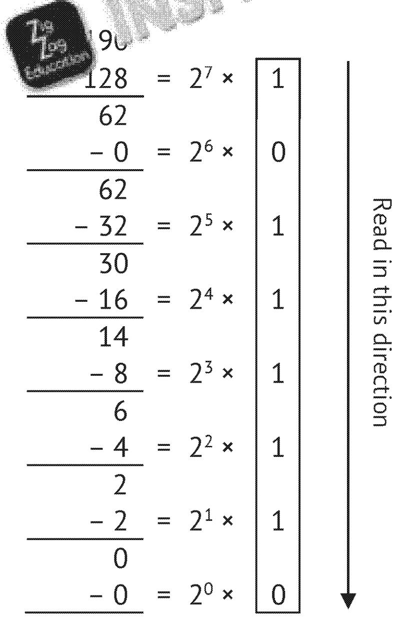
The hexadecimal representations of the binary values would be 01010110_2 =

CONVERSIONS BETWEEN RADICES

At some point you will need to convert integer numbers between radices an eventually become second nature. Although it may seem strange at first, the both simple and straightforward. There are two methods that can be used to These are the repeated *subtraction* and *division* methods. These methods wo most easily applied between decimal and other radii.

Repeated subtraction

The repeated subtraction method uses the powers of a radix to reduce the n powers work, you're left with the number 1 or 0. Take a look at the example number 190_{10} into base 2. You start off by finding the highest power that you number, then begin reducing the power by one after each subtraction. If the simply use a 0. The highest number you can multiply the power by is given b converting into.

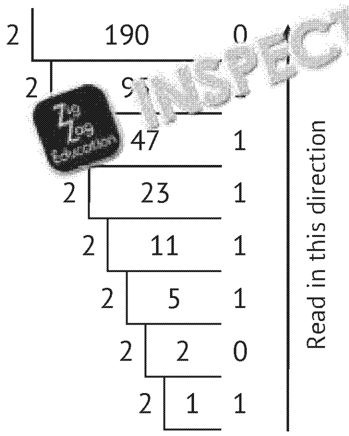


As you can see, the continuous with the result of the radix yo finish the conversion you read using the numbers you've mult example, 190_{10} is 10111110_2.

Repeated division

Another method of converting integers between radices uses division instea easy, mechanical and more intuitive than the subtraction. It uses the idea th the same as successive subtraction by powers of the base.

In the following example we'll use 190_{10} to base 2 again.



As you can see in this example from the division that leaves y conversion between the two ra

HEXADECIMAL CONVERSIONS

Denary to hexadecimal

As explained in *5.2.1*, the binary representation is divided into groups of 4 bi then labelled 8, 4, 2 and 1, respectively. You sum the values of each group u number system convention (0 to 9 and A to F). This is shown in the followin

Example – Convert 213_{10} to hexadecimal

Step 1 – convert 213_{10} to binary

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

Step 2 – sum the values using the new labels

8	4	2	1	8	4	2	1
1	1	0	1	0	1	0	1
8 + 4 + 1 = 13_{10} = $\mathbf{D_{16}}$				4 + 1 = $\mathbf{5_{16}}$			

Hexadecimal to denary

To convert between these two radices, simply use the hexadecimal place val 16. For example, convert A3 to denary.

$A3_{16} = (10 \times 16) + 3 = 163_{10}$

Question: Number Bases

1 Convert the following:

a) $26_{10} \rightarrow ?_2$ (1 mark)

b) $100_{10} \rightarrow ?_2 \rightarrow ?_{16}$ (1 mark)

c) $7A_{16} \rightarrow ?_2 \rightarrow ?_{10}$ (1 mark)

d) $01001001_2 \rightarrow ?_{10}$ (1 mark)

e) $188_{10} \rightarrow ?_2 \rightarrow ?_{16}$ (1 mark)

f) $?_{16} \rightarrow 11010011_2 \rightarrow 201_{10}$ (1 mar

2 What is the highest value that can be stored stored using a single byt

5.3 UNITS OF INFORMATION

BITS AND BYTES

These are the two simplest units of data:

- A *bit* is the most basic unit of the data representation used in compu of 'on' or 'off' (1 or 0) on a digital circuit.
- A *byte* is a group of 8 bits with increasing value from right to left. A b *addressable* memory – meaning a specific byte can be retrieved acco

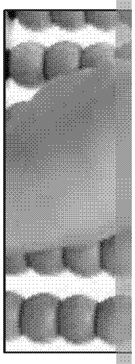
There are also two other frequently used representations – these might not know! They are:

- *Words* are groups of bytes in a sequence. Frequently found word size
- *Nibbles* (yes, really) are groups of 4 bits. Therefore, a byte is form so called high-order nibble and the last 4 are the low-order nib

You can work out how many values a bit pattern can represent by using the 2 bits. For example, how many values can be represented by 3 bits? The answ

000 001 010 011 100 1

UNITS



While talking about small volumes of data, bits and bytes are fine. However, you're probably more than aware of the volumes of data used in real-world applications.

So how can you represent bigger volumes?

Denary		Binary	
Kilo, k	10^3	*Kibi, Ki*	2^{10}
Mega, M	10^6	*Mibi, Mi*	2^{20}
Giga, G	10^9	*Gibi, Gi*	2^{30}
Tera, T	10^{12}	*Tebi, Ti*	2^{40}

Historically the two naming conventions have been confused and the denar name is meant because it is easier to count in terms of denary.

For example, 1 kB = 1,000 bytes whereas 1 KiB = 1,024 bytes.

Questions: Units of Information

1 How many bits are there in 64 bytes? (1 mark)

2 How many bit patterns can be represented using 32 unsigned bits? (1

3 How many bytes are contained in a single tebibyte? (1 mark)

5.4 BINARY NUMBER SYSTEMS

UNSIGNED VS SIGNED BINARY

All the binary you've been shown in this course companion so far has been u
has a very large limitation; it cannot represent negative numbers, no matter
round this problem is to assign the highest-order bit to represent whether th
this is called signed binary.

```
+45      0 | 0101101
-45      1 | 0101101
```

This is often called sign magnitude and it too isn't a perfect way of represent
allows for two values of zero: a 'positive' zero and a 'negative' zero. Also, by a
the range of magnitude: an unsigned byte can represent the numbers 0 to 25
represent -127 to 127.

The minimum and maximum unsigned binary values for a given number of bits,

UNSIGNED BINARY ARITHMETIC

Addition

Adding binary numbers is very similar to adding denary numbers except you'
1). Similarly to when you carry digits when you reach the number 10, when y
the 1. Here is a worked example.

Example: Using unsigned binary arithmetic, calculate the sum of the number

Step 1 **– Convert both values to unsigned binary**

51_{10} = 00110011
143_{10} = 10001110

Step 2 **– calculate each bit, carrying any values**

```
 0 0 1 1 0 0 1 1   +
 1 0 0 0 1 1 1 0
-----------------
 1 1 0 0 0 0 0 1
   1 1 1  1 1 1
```

Step 3 **– Convert the answer to denary**

$11000001_2 = 193_{10}$

Multiplication

If you want to multiply 5 by 4 you could say that this is the same as adding larger numbers this becomes slow. Multiplying two numbers is also very sim multiplication remembering that 0 * 0 = 0, 0 * 1 = 0, 1 * 0 = 0 and 1 * 1 = 10.

Consider the following example:

Binary	Denary
00110011	51 (1+2+16+32)
x 10001110	x 142 (2+4+8+128)
= 00000000	
001100110	
0011001100	
00110011000	
000000000000	
0000000000000	
00000000000000	
+ 001100110000000	
001110001001010	7242 (2+8+64+1024+2048+4096)

If this multiplication were to have taken place in a computer with only an 8-bit caused an overflow error (there wouldn't have been enough bits to represent the binary arithmetic by hand it is possible to simply add the additional bits required

SIGNED BINARY USING TWO'S COMPLEMENT

Subtraction

As shown earlier, you can represent a negative value by assigning a sign bit. representing a negative number without losing magnitude. If you invert all t resulting binary form behaves like a negative number. This is called the num inherently difficult to use because it results in an offset of −1. Adding 1 to a in the number's two's complement, which is much easier to use. Follow the

Example: *Using two's complement, calculate the sum of 24 − 18.*

Step 1 **– Calculate the two's complement of 18 to get −18**

0 0 0 1 0 0 1 0	18
1 1 1 0 1 1 0 1	One's Complement
0 0 0 0 0 0 0 1	+ 1
1 1 1 0 1 1 1 0	Two's Complement

Step 2 **– Calculate the sum using the two's complement of 18**

```
0 0 0 1 1 0 0 0   +
1 1 1 0 1 1 1 0
0 0 0 0 0 1 1 0
 1 1  1  1 1
```

Step 3 **– Convert the answer into denary**

$00000110_2 = 6_{10}$

The main advantage of two's complement, however, is in calculations. This c the method.

Imagine that a number is like a counter with so many digits. If you imagine a backwards, as you reach 0 and go back it would revert to 9999 for −1, 9998 to 'subtract by adding'.

```
9997 = -3
9998 = -2
9999 = -1
0000 = 0
0001 = 1
0002 = 2
```

Etc.

So in base 10 we can work out 7 – 3 by adding: `0007 + 9997 = 10004`

The additional 1 at the front cannot exist in a four-digit way and is called *ov* to give the answer 0004. Two's complement does the same as the denary so equivalent of '9999' on our counter.

7 in binary is `00000111` (8 bit)

To work out−3 in binary:

+3 is `000000011`

One's complement: `11111100`

+ 1 becomes: `11111101`

We can now add the two numbers together:

```
   00000111
+  11111101
  100000100
```

First digit is overflow and ignored answer is `00000100 = 4`

Questions: Binary Number Systems

1 Using unsigned binary, complete the following:

a) $16_{10} + 44_{10} = ?_2$ (1 mark)

b) $7_{10} * 8_{10} = ?_2$ (1 mark)

c) $74_{10} + 63_{10} = ?_2$ (1 mark)

d) $9_{10} * 10_{10} = ?_2$ (1 mark)

2 Convert the following into their two's complement form and complete t

a) [illegible] – 5[illegible] = $?_2$ (1 mark)

b) [illegible] - $48_{10} = ?_2$ (1 mark)

3 Can –166 be represented using a single byte? (1 mark)

NUMBERS WITH A FRACTIONAL PART

An integer is a whole number: 1, 2, 3, 4, 5, 6, etc. Decimals are numbers wit
or 4.2039. You call the second part of these numbers the fractional part; 0.2
computer terms this creates a problem with representation because our bin
whole numbers. In order to get round this problem a fixed- or floating-point
the parts in front of and after the decimal place are distinguishable.

There are two ways you can solve this problem:

1. *Fixed-point decimals:* allocate one set of bits for the integer part and
   fractional part (e.g. 2.25 would be represented as 0010.0100)
2. *Floating-point numbers:* put the number into standard form and then
   and another for the power (e.g. 2.4 = [illegible] 1[illegible] which could be repres

Fixed-point binary

A deci[illegible]g. [illegible], is made up of the integer part (52) and the fractional
decima[illegible] allocate a certain number of bits for the integer part, and the r

```
integer part . fractional part
```

The first digit of the fractional part represents $^1/_2$, the second digit represent
etc. So for example the binary number 1011.1011 can be displayed as:

8	4	2	1	$^1/_2$	$^1/_4$	$^1/_8$	$^1/_{16}$
1	0	1	1 ●	1	0	1	1

$= 8 + 2 + 1 + ^1/_2 + ^1/_8 + ^1/_{16}$
$= 11\ ^{11}/_{16}$
$= 11.6875$ to 4 decimal places

If more bits in a memory word are assigned to the fractional part, greater pr
hand, fewer bits are then available for the integer part and this reduces the r
increasing the proportion of a word given to the integer part increases the m
possible level of precision.

To convert the fractional part from decimal to binary you use a similar proce
integers, but multiplying by 2 rather than dividing:

```
0.671875
≡ 0.671875 × 2 = 1.34375, 0.34375 × 2 = 0.6875, 0.6875 ×
  0.375 × 2 = 0.75, 0.75 × 2 = 1.5, 0.5 × 2 = 1
```

Writing out the 1s [illegible]s [illegible] order, this gives 101011. In this case there is an
happe[illegible]qu[illegible] $^{43}/_{64}$.

Howeve[illegible]uppose you took a random six-digit decimal: 0.328774. Using fou
is 0.3125. Even using eight binary digits you get 01010100 which is 0.32812

Negative floating-point numbers can be represented using the two's comple the least significant bit, NOT actually 1. So for example:

8	4	2	1	$^1/_2$	$^1/_4$	$^1/_8$	$^1/_{16}$
0	0	1	1 ●	1	0	1	1

represents +3.6875. To convert to −3.6875 we would perform the operation not there. So:

+3.6875 = 0011.1011

One's complement = 1100.0100

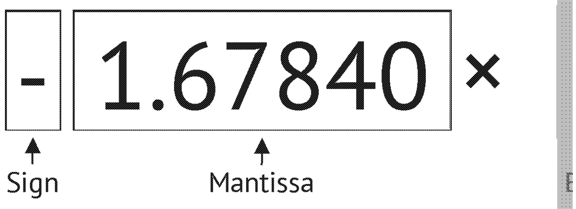
+1 to the least significant bit: 1100.0101

So −3.6875 = 1100.0101

Notice that if two's complement is being used, the maximum positive numbe very quickly run out of range unless more bits are used.

Floating-point binary

Floating-point numbers are a representation of rational numbers in a binary floating-point numbers rather than fixed-point numbers is that the range of with a set number of bits is far larger. Floating-point numbers achieve this similar to that used in scientific notation. Rather than have an exponent bas however, most floating-point standards have an exponent base of 2. Floatin two parts: a *mantissa* and an *exponent*. They also need to have a way to repr be done with a single bit to represent the sign (a sign bit) or using the two's example of a real decimal number with the sign, mantissa and exponent ide



As the exponent base is defined by the standard being used, it is not necess itself. The same is true of the binary point.

In general the mantissa is specified as a fixed-point binary number. The bin most significant bit and the second most significant bit.

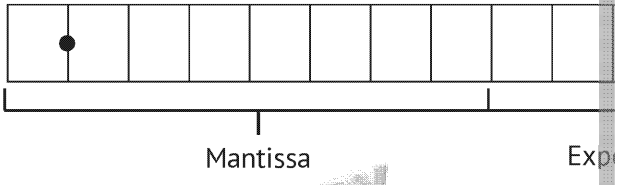
Converting a decimal real number to a floating-point representation

No matter which standard is used, the steps involved in converting from a de point number are very similar:

1. Convert the number from decimal to binary
2. Change binary format to the mantissa and exponent format
3. Perform two's complement conversion if numbers are to be negative
4. Normalise the mantissa and adjust the exponent so that the number

Two's complement floating-point numbers

One way of representing real numbers is to use the two's complement stand
exponent. This allows the representation of both negative numbers (by usin
numbers smaller than 1 (by using a negative exponent). Here is an example
might be represented using 12 bits, 8 for the mantissa and 4 for the exponer



The number −1.25 represented in this way would be:

1	1	0	1	1	0	0	0	0	0

The sign bit is relatively straightforward; a 0 means the number is positive, a
The exponent is in a format called excess −127; this means that the number
way as a normal binary number, but then 127 should be subtracted from it.
127, 00000001 would be −126 and so on. This is a simple way to represent

Unless all the bits in the mantissa are 0, the mantissa is assumed to begin w
of 0101101... is equivalent to 1.0101101... In effect the mantissa therefore h
than the 23 bits that are actually used.

The standard also includes ways to represent +0, −0, +∞ and −∞. This mea
that is larger than can be stored, it can be stored as ∞.

ROUNDING ERRORS

Examples of rounding errors are shown in the next two sections.

ABSOLUTE AND RELATIVE ERRORS

There are two ways of classifying errors: absolute errors and relative errors.

Absolute errors are simply the difference between the number wanted (desir
representation of that number:

```
Absolute error = desired value − computer representation
```

For example:

```
0.492 = 1710.1992 − 1710.15
```

If the computer couldn't represent 0.1992 but could represent 0.15 the comp
of 0.0492.

The relative error is the ratio of the absolute error over the number wanted (

```
Relative error = absolute error / desired value
```

For example:

2.87×10^{-5} = 0.0492 / 1710.1992 = 0.00002876992

RANGE AND PRECISION

The limited size of memory within a computer used for floating-point numbe of numbers which can be represented. Precision is a measure of how close t within the computer is to the actual value. Say, for example, a calculation re and this had to be stored as 0.123458 due to a combination of rounding erro floating-point system. The loss of precision would be 0.000001211.

Storing irrational numbers

Not all fractions can be represented by a finite number of bits in base 2 form cannot be represented exactly.

1/10 is represented by 0.1 in decimal. However, in binary it becomes an infin 0.0001100110011001100110011... In the same way, 1/3 is represented by 0. converted to 0.010101... in binary. Computers represent numbers using a fini cannot represent these fractions exactly. Fractions such as 1/3 and 1/10 will *significant bits* depending on the size of storage. The precision of the repres increases the more bits that are used.

CANCELLATION ERRORS

Cancellation errors occur when two floating-point numbers are subtracted to result is unchanged from the larger of the two numbers.

One way in which cancellation errors occur is when two numbers of complet subtracted. An example of this type of cancellation is subtracting 0.0000000 obtained would be 1 instead of a theoretical 0. 9999999999996.

One more way in which cancellation can occur is when you subtract two nu example, if you subtract 1.000 from 1.000 the result would be a very small n instead of 0 due to the approximated representation of 1.000. This can cause division, since 1 / (1.000 – 1.000) would result in infinity.

Cancellation errors are usually avoided by rearranging the equation in such a subtracted.

NORMALISATION OF FLOATING-POINT FORM

Normalisation is all about maximising the precision of the number within the is that leading digits in the mantissa should be removed, and the exponent i possible. Normalisation is extremely important; take the table below as an e

Mantissa	Exponent	Normal
0.0000000000011001	010000	No
0.11001	000100	Ye

Both of these examples represent the same two's complement number, 12.5 requires a much larger mantissa than the second to keep the same precision. limit on the size of the mantissa, say 8 bits as above, the normalised number rounding the number that has not been normalised would mean that it woul

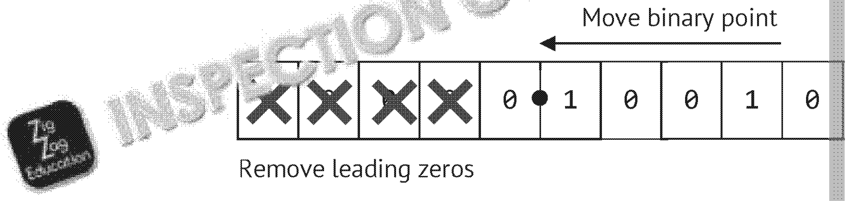
Normalising binary numbers which are not using the two's complement stan All leading zeros in the mantissa should be removed and the exponent adjus same. As a guide, the number of places the binary point moves left by is the should be increased by and vice versa.

Positive two's complement numbers can be treated in the same way as other important exception – there should always remain a single leading zero. Ot to a completely different negative one! To normalise negative values, all lea the exception of one.

So, in summary, to normalise positive two's complement numbers all the lea the exception of a single zero. For example, take the following number in tw

0	0	0	0	0	1	0	0	1 ●	0

To normalise this number, remove all the leading zeros bar one and move th immediately after the new most significant bit:



The exponent is the number of places the binary point has been moved to th it has been moved to the right. In this case the binary point was moved four

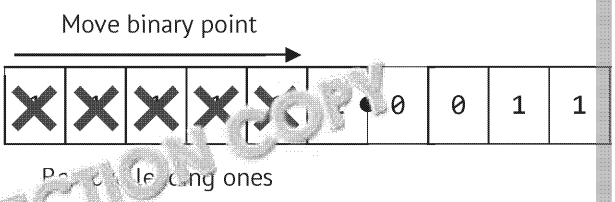
0 ●	1	0	0	1	0	1	0	0

Mantissa

The process for normalising negative two's complement numbers is very sim zeros, the leading ones should be removed. Take the following number as ar

1 ●	1	1	1	1	1	0	0	1	1

Again, the leading ones should be removed, except one, and the binary poin immediately after the most significant bit:



Since the binary point has been moved five places right, the exponent will b normalised floating-point number:

1 ●	0	0	1	1	1	1	0	1

Mantissa

UNDERFLOW AND OVERFLOW

Underflow Underflow occurs when you are using very small numbers an the computer can store. For example, if the smallest number and you attempted the following sum:

$^1/_{128} \times {}^1/_{128}$

The computer cannot store the resulting number and so ar computer flagging that there has been a loss in precision.

Overflow Overflow is when the result of a numeric calculation becom space reserved for numbers. Usually some indication will b overflow – some machines will have a variable called an c after a calculation if overflow has occurred. An example of trying to compute the factorial of 100 on your calculator. A overflow was shown in the binary subtraction example on p

5.5 INFORMATION CODING SYSTEMS

Communication between computer systems used to pose a large problem in the institutes used different standards so computer systems would almost be talking coding schemes allow the written characters you use every day to be converted manipulate, display and transmit via a computer system. The main two that are

While representing symbols and characters in a computer system is fairly stra transferring the data can represent a bit of a problem as one computer might another computer. This is where standardised character sets are introduced i uniquely identifying a character from a decimal value to allow communicatic data without the need to define an entire character set for each file.

ASCII AND UNICODE

ASCII character set

ASCII (The American Standard Code for Information Interchange) is a 7-bit ch a standard in 1963 and was widely used to represent symbols and characters *Remember that the numbers are the denary values; 65 in the ASCII character set*

0	*NUL*	16	*DLE*	32	SPC	48	0	64	@	80	
1	*SOH*	17	*DC1*	33	!	49	1	65	A	81	
2	*STX*	18	*DC2*	34	"	50	2	66	B	82	
3	*ETX*	19	*DC3*	35	#	51	3	67	C	83	
4	*EOT*	20	*DC4*	36	$	52	4	68	D	84	
5	*ENQ*	21	*NAK*	37	%	53	5	69	E	85	
6	*ACK*	22	*SYN*	38	&	54	6	70	F	86	
7	*BEL*	23	*ETB*	39	'	55	7	71	G	87	
8	*BS*	24	*CAN*	40	(	56	8	72	H	88	
9	*HT*	25	*EM*	41	)	57	9	73	I	89	
10	*LF*	26	*SUB*	42	*	58	:	74	J	90	
11	*VT*	27	*ESC*	43	+	59	;	75	K	91	
12	*FF*	28	*FS*	44	,	60	<	76	L	92	
13	*CR*	29	*GS*	45	-	61	=	77	M	93	
14	*SO*	30	*RS*	46	.	62	>	78	N	94	
15	*SI*	31	*US*	47	/	63	?	79	O	95	

Values 8, 9, 10, and 13 convert to backspace, tab, linefeed and carriage return characte representation but may affect the visual display of text. The remaining of the first 31 ch that are used within the computer system itself, i.e. pointe

Unicode character set

Unicode has now become the industry standard, made solely to allow the us
ASCII except that it is an industry standard that can be used to represent ove
result, the majority of the world's writing systems. Like ASCII there is a set o
characters. This allows for the codes to be broadcast as numbers and then c
reading. Without the Unicode character set you wouldn't be able to use emo

ERROR CHECKING AND CORRECTION

While communicating over a network it is possible for bits (*packets*) of data t
receiving device has misinterpreted a packet of data; this might result in an e
mathematics, to drastically reduce this chance in a technique known as error

There are four techniques you're expected to describe and know the use of: p
and *checksums*.

Did you know?

Handshaking is a procedure that all devices undergo before transmitting data. The handshake allows both devices to check whether the other device is ready for data transmission. During the handshake, the devices agree on a method of error checking, the rate of transfer and other criteria such as method of transmission.

How this is done is defined by the particular protocol they are using, although this largely out of the scope of your course.

Parity checking

Parity checking is a simple technique of detecting transmission errors by usi
as a *parity bit*. This parity bit is added to the end of a transmission and is the
received in reverse order. There are two types of parity: *odd* and *even*.

Odd parity	If using odd parity, the number of 1s in the transmission is su the parity bit is set to 0 to ensure that there is still an odd nu even, the parity bit is set to 1 to ensure that there is now an o
Even parity	Even parity is the same as odd parity except that, after counti number of 1s is even then a 0 is added to the end of the trans of 1s stays even. Likewise, if the number of 1s is odd then a 1 transmission, as that will mean there is now an even number

Check digits

Check digits are used to ensure that a range of numbers has been entered c
looking at the *International Standard Book Number* (*ISBN*) found on books. The
book using between 10 and 12 digits and a check digit created by multiplying
multiples using a modulo-11 division of the total to check whether the i

Look at the following worked example:

Example: Consider the ISBN 1 74157 103 x.

Step 1 – Write out the ISBN and calculate the positional multiples (ignoring

1	**7**	**4**	**1**	**5**	**7**
10	9	8	7	6	5
10	63	32	7	30	45

Step 2 – Sum the total

```
10 + 63 + 32 + 7 + 30 + 45 + 4 + 0 + 6 = 197
```

Step 3 **– Use modulo-11 to calculate the remainder**

```
197 MOD 11 = 10
```

Remember: if the remainder is '0' then the check digit is 0; if the remainder is 1C On these two occasions you skip step 4.

Step 4 – Deduct remainder from 11 to produce check digit
In this worked example the result is one of the exceptions. However, if the r exceptions then you have to deduct the remainder from 11 – this produces t

Majority voting

Majority voting uses *repetitive transmission* to deduce where the error has oc number of times to the receiving device, and applies majority voting to dete allowing for correction to be made easily. Take a look at the following exan

If one device wants to send the letter 'K' to another it can use majority votin The receiving device knows that the binary representation of the ASCII chara *ASCII uses 7 bits*).

The sending device would then send each bit three times, so the receiving d

```
111, 000, 000, 111, 000, 111, 111
```
(commas added for cl

If the receiver gets the bit pattern '000' or '111' then it assumes that the bit error and is either a 0 or a 1, respectively. However, if 1 bit in each triplet is deduce that there has been an error. If, for the third bit, the receiver gets '00 can deduce that there has been an error and applies majority voting to corre than 1s, the original bit was a 0. Otherwise, if there were more 1s than 0s th

Checksum

A checksum is possibly one of the oldest validation methods in data transmiss authentication because if the checksum is invalid it means that packets have b with. The checksum for a transmission is determined in one of two ways. If yo of data containing a single byte of data, the byte contains 8 bits which can be total of 256 possible combinations. Ignoring the representation for 0 leaves yo

Steps of checksum

1. Divide the total number of bytes per packet by 256.
2. Rounding the result down if needed, multiply this number by 256 (th
3. Deducting this number from the total number of bytes being sent leaves you with the checksum value.

The following example shows how a checksum is generated for a packet that is sending 1152 bytes of data.

1. `1152 / 256 = 4.5 (≈ 4)`
2. `4 * 256 = 1,024`
3. `1152 - 1024 = 128`

The checksum for the transmission would be 128.

Questions: Informatio

1 **If you were using odd pari '010011010110', what woul**

2 **What would the check dig (2 marks)**

3 **Using the ASCII character what would the transmissi using the majority voting (2 marks)**

5.6 REPRESENTING IMAGES, SOUND AND OTHER DATA

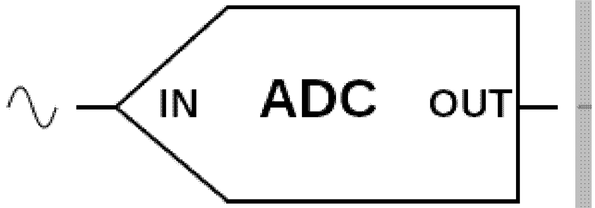
BIT PATTERNS, IMAGES, SOUND AND OTHER DATA

You already know that a computer can only store data in binary, but it can al In binary, data can only be in one of two states – on or off, positive or negati systems as it removes the ambiguity that computers cannot handle and make predictable and consistent. This means you can use bit patterns to do a pleth images, transforming waveforms into audio and calculating the size of files.

ANALOG AND DIGITAL

The difference between digital and analog is in the continuity of the data. An transmission, whereas digital is discrete and a set of fixed values.

Most analog signals are from physical dev ices which are constantly changin usually as a voltage change but a computer would be unable to read these discrete set of values. The way the signal is converted is through sampling a number representing its amplitude. This is called analog-to-digital conversi



One such example of analog-to-digital conversion could be a light sensor wh the amount of light that is falling onto it at any particular time. The voltage range (for example, 0 means no light, 1 full light). The conversion to digital represent the value of the sensor. If only 1 bit was used then the signal woul light); 2 bits would represent four different signals (00 no light, 01 1/3 light,

Equally, a device may be controlled by a computer in a similar way; for exam at different brightnesses. The actual light is an analog device which can rece provide the light range. However, the computer could only provide a discrete depending on the number of bits used would increase the variance in signal. to-analog conversion (DAC).

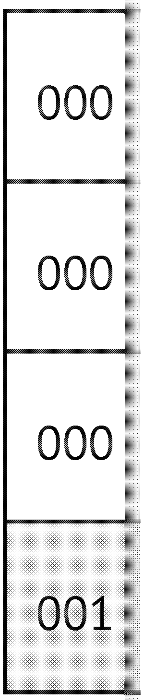
The most common use for ADC and DAC is in the sampling of sound – *for mo*

More digits means larger numbers

Unfortunately, there is a finite number of digits that a computer can store. Th what a computer can represent in terms of what it can store. However, the nu of bits n can represent is equal to 2^n.

n	Bit Patterns
1	$2^1 = 2$
2	$2^2 = 4$
3	$2^3 = 8$
4	$2^4 = 16$

BITMAPPED GRAPHICS



3-bit bit

Bitmapped graphics use streams of bits to store information about each individual pixel in an image. These bits encode what colour the pixel should be, and as a result bitmapped images result in large file sizes. In its simplest form, 0 encodes for white and 1 encodes for black, but the more bits used, the more colours can be used. However, that's not all that the bits code for.

The *resolution* of an image is the number of pixels that appear in each inch of the image. A higher resolution means that the quality of the image is higher; this means that the image can be zoomed and scaled without visual artefacts or distortions.

The *colour depth* of an image depends on the number of bits used to store the value of the pixel. If the bit number increases then the colour depth will increase because there are more colours available to represent what the computer is trying to display, as each combination of bits will correspond to a colour in a colour chart.

The *size* of an image determines the number of rows and columns of pixels that create the image. Size and resolution are inversely proportional to each other in the sense that when you zoom in to double the size you view the image at half the resolution.

Simple bitmap file calculations

Suppose a bitmap file was 3 inches by 3 inches, had a resolution of 72 pixels
colours. How big would the file be? We have all the information we need to

Size = 3 × 3 inches^2 = 9 inches^2

Number of pixels per inch^2 = 72 × 72 = 5,184

Total number of pixels = 5,184 × 9 = 46,656

256 colours require 8 bits

File size = 46,656 × 8 bits = 373,248 bits = 46,656 bytes
= 45.56 kilobytes (2dp)
= 0.04 megabytes (2dp)

This is only a rough estimate at best; in real-world applications there will al
as file headers. These headers contain metadata for the image. Metadata is t
colour depth, etc.; the storage requirements are given by the equation:

```
Storage requirements = resolution × colour depth
Resolution = width (pixels) × height (pixels)
```

Question: Bitmapped Graphics

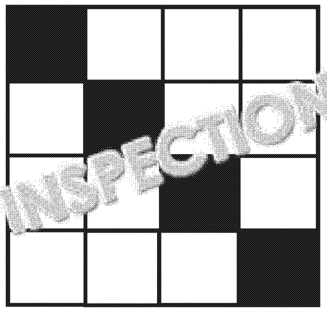
1 **Estimate the size of a bitmapped file that is 7 inches by 7 inches, conta square inch and has a colour depth of 8. Give your answer in kilobytes.**

VECTOR GRAPHICS

Vector graphics are slightly different to bitmap images. Whereas a bitmap imag
pixels in an image, a vector graphic holds the information or instructions of th
create the image.

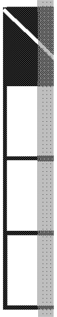
Take the following example of a line. Instead of holding information about the
information on the start and end point of a line; any pixels that intersect that lin

Bitmap Graphic



Black-and-white bitmap:

```
1000 0100
0010 0001
```



Bl

```
Line(
```

VECTOR GRAPHICS VERSUS BITMAPPED GRAPHICS

The table below explains the relative pros and cons of the two image types.

Bitmap graphics	
Takes up more resource memory	Takes up considera
Takes up more storage space	Takes up considera
Images are less precise	Graphics are more
Images aren't scalable without visual artefacts	Graphics are scalab
Images use less processing power	Vectors require mor
Made of pixels that can't be grouped	Made of elements t

Bitmap images can be easier to edit, because you can work with individual p
manipulate entire planes. This makes bitmaps the only real option for photo
contrasting sections of an image that you may wish to edit separately to the
bitmap images with a small number of colours or continuous areas of colour
vector graphics.

Vector graphics are useful in situations such as architecture and design whe
more important than editing small sections of an image. Also, scaling up the
quality of the image and some manipulation of vector images is easier, e.g. c

DIGITAL REPRESENTATION OF SOUND

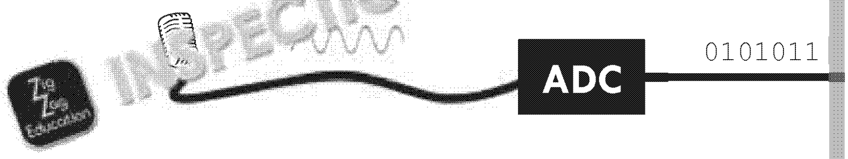
Sound is an air pressure wave that causes vibrations in our eardrums which c

In order for computers to capture sound waves they need a device known as energy from one form into another.

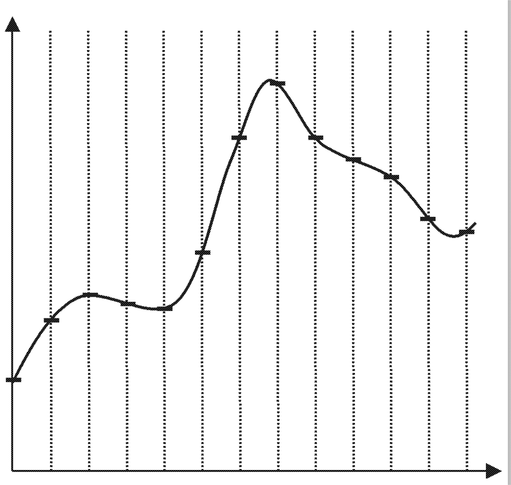
- The original sound wave is known as *analog* data, which is a set of co
- In contrast, *digital* data such as the wave the analog data is converte discontinuous quantities.

Analog-to-digital conversions

Analog-to-digital conversion (ADC) takes place by sampling the height of an an transducer, i.e. a microphone, and produces a digital signal representation that

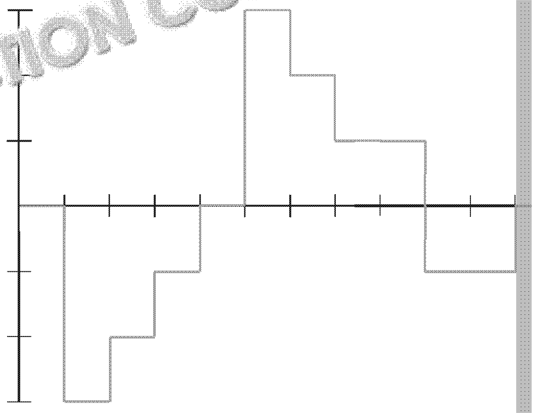


Below is a sample of a sound wave captured by the microphone with a samp



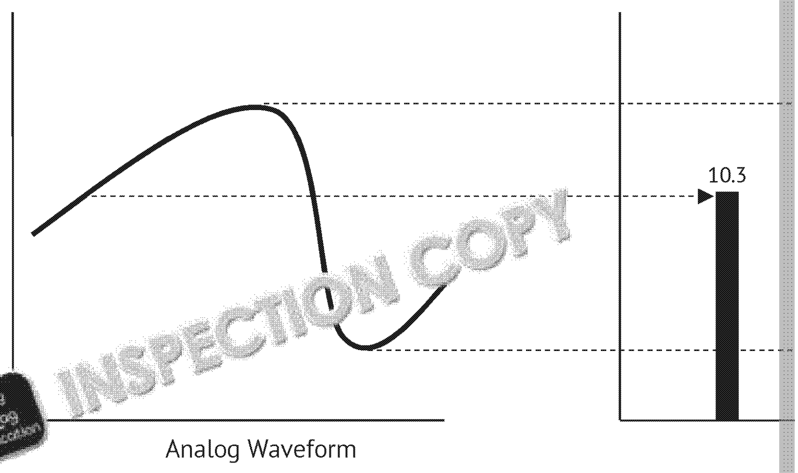
Each of the dotted lines represents a sampling point; at each of these samplin recorded. The distance between these sampling points is known as the *sampli*

Below is an example of how the analog wave could look after being convert



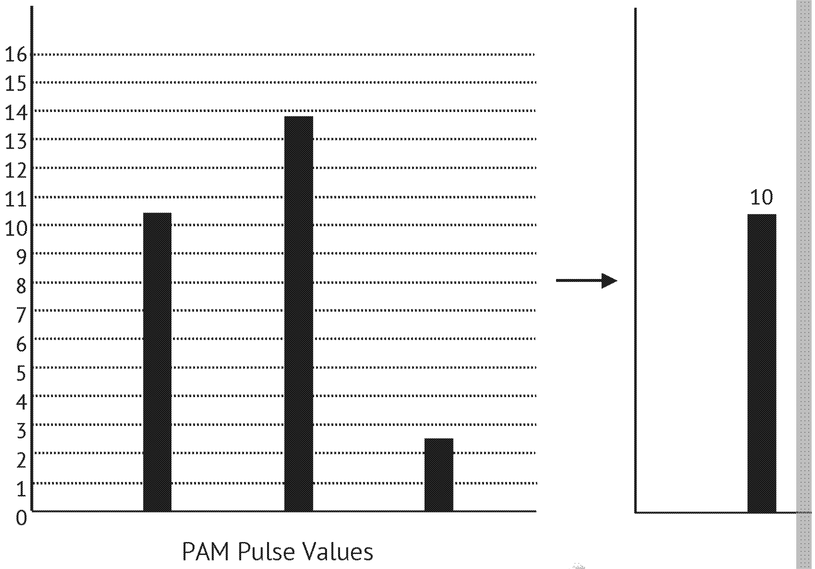
There are actually a number of steps needed to complete a conversion from computer relies heavily on a technique known as *pulse code modulation* (PC

1. Samples are taken from the analog at a set value of hertz but must, be twice the highest frequency in the analog signal. There are repres proportional to the original signal's value in a process known as *puls*



2. The pulse amplification modulation values are approximated using integer where n is the lowest number of bits that can represent the

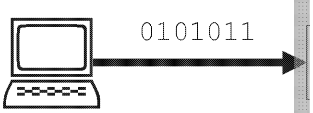
   For example, if $n = 4$ then 16 levels can be used to approximate the



3. The final step is to encode the height of each of the pulses into bina

   In the example you have $10_{10} = 1010_2$, $14_{10} = 1110_2$ and $2_{10} = 0010_2$. sequence of fixed-height pulses which can be stored, manipulated o digital signal back into an analog waveform.

Digital-to-analog conversion (DAC)



The reverse of the analog-to-digital conversion is what you hear when listening to any sound generated from a computer *or* storage device such as an MP3 player.

However, during the conversion from analog to digital it is possible that dur you are left with the *staircase effect*, which is where there is a subtle loss of analog sound and the digital representation which can be removed by using

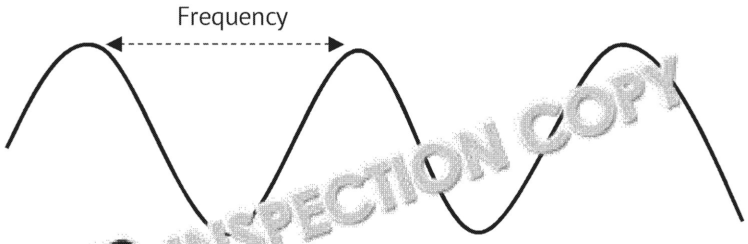
The Nyquist theorem

Then Nyquist theorem states that…

> **The sampling frequency must be at least twice as high as the highest frequ**
> **because you must have at least one data point for each half-circle of the au**

What this means is that if you want a full 20 kHz audio bandwidth, your sam
fast, i.e. over 40 kHz. This means you need a higher sampling rate to record h

Consider the following wave; the frequency of it is constant and hence it pro



The sa
order t
importa
sound c
Sounds
high in
frequen

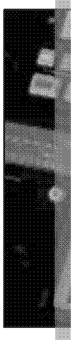
Sample resolution and sample file sizes

As stated earlier, to record a sound you must have an input in the form of sa
of samples the reproduced sound is closer to the original. Similarly, if you ta
period you can increase the 'quality' of the sound. This is the sound's *sample*
taken in one second of recording. If you record in the MP3 format there is a s
whereas the sample rate needed to capture the human voice is much lower,
you try playing music down a telephone line the sound quality is poor.

This leads on to *sample resolution* – the number of bits assigned to each sam
depth of an image file; if you increase the number of bits used to store each
range of frequencies increases, meaning that you can reproduce a sound tha

```
    File size = sample rate * sample resolution * length of
```

MUSICAL INSTRUMENT DIGITAL INTERFACE (MIDI)



MIDI is a technical standard which enables a wide range of electronic musical instruments, computers and devices to connect and communicate with one another. MIDI files take up a considerably smaller volume that other sound files because only information about the notes is stored rather than the sound itself. These data items are what instrument needs to be played, what note needs to be played, and how loudly and for how long the notes are played. These data items are conveyed in what are known as *event messages* that are used to generate the sound that is required.

The major drawback of the MIDI format is that there is very little sound qual
different sound file. This is because all the sounds are generated by the sour
sound cards only use frequency modulation (FM) synthesis or simple wavetab
instructions into sound. However, MIDI is not designed for storing high-quali
file size. Rather, it is designed to allow easy composition and editing for mu

Questions: Representing Sound

1 **Estimate the size of the sound file of a 30-second recording with a sam**
and a sample resolution of 16 bits. Give your answer in kilobytes. (2 ma

2 **Calculate the sample rate of a file 1100,000 bits in size that is 10 second**
of 10 bits. (2 marks)

3 **In audio conversion, why can you never recreate the original analog si**

DATA COMPRESSION

As our computer systems have become larger and more powerful so have th what about storing and transporting all that data? The limitation on capacity data compression. Almost all files can be compressed so that they require le to transmit and are easier to work with.

Lossy compression

Lossy compression attempts to identify seemingly redundant data and remov common example of this is in the way MP3 files are generated. The file type e that are outside of the human hearing range. This can also apply to images in may be reduced to a certain level without a visible loss of quality.

Look at the example below – the same image of the apple was saved three ti 260×260 pixels. The only difference between them is the quality percentage lower the percentage, the more compression is applied.

Quality @ 100%
73 KB

Quality @ 66%
14 KB

As you can see, the file has been compressed from 73 KB (100% quality) to just quality has been degraded significantly. In this example, the most suitable sett which has still reduced the file size significantly (from 73 KB to 14 KB), without

Lossless compression

As the name implies, this is a method of compressing the file without loss of text, data and programs where all the information is required. The disadvant unable to reduce file sizes as significantly as lossy compression, it often requ compress and decompress the files. A common example used on the Interne

In sound, lossless compression uses a pair of algorithms. The first is to conde waveform when the sound is not in use. The second algorithm uses the inver algorithm to regenerate the sound when it is used for playback.

Lossless compression can use *run length encoding* (RLE) which identifies repe patterns and stores one copy of the pattern and how many times it repeats in This can drastically reduce the size of the stored file, especially in text/sourc and other formatting parts.

For example, suppose we want to compress the string: "I LOOOOOOOOOV would store each repeating O, V and E once, making our compressed string: is over 10 characters shorter than the uncompressed version.

Another approach is to use a *dictionary-based* method which uses a type of su the file to those stored in a data structure called *a library*. If a match is found substitutes the string with a reference pointer to the item in the dictionary. I is added to the dictionary and the reference is substituted.

ENCRYPTION

Encryption is the act of protecting your personal data by making it unreadabl
Ciphers are special algorithms designed to convert the readable data into a j
rendering it unreadable. This is called *cipher text* and is completely unreadab
which was used to generate the cipher text in the first place.

Cipher keys are an important constituent of the ciphers working and will either
Symmetric keys can be used to encrypt (go from plain text to cipher text) and
text) the data, whereas asymmetric keys will be used for one or the other but
key it is largely impossible to decrypt the cipher text, meaning if the key is los

Did you know?!

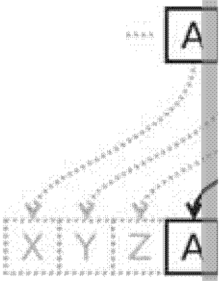
When passwords are stored on a web server they are not stored in plain text form
hashing table and the hash equivalent is stored. The long-lived and widely used sta
hash algorithm, which was retired in 2012 after the algorithm was 'cracked' when
leaked 6.4 million passwords. It was cracked in a single day by operators across th
standard algorithms used and the security associated with them.

Cryptanalysis

This is the act of trying to determine the plain-text representation from a cip
knowing the decryption key. It takes a lot of theory and is largely based arou
cipher keys use to secure data. In practice it is often done using what is know
cipher key has been narrowed down to a selection of possibilities. Although t
course, it can provide a greater understanding of encryption, as well as being
technology industry.

Caesar ciphers

Ciphers are not a technological breakthrough. Julius Caesar was known for writing any confidential military messages in a cipher with an offset of three letters to the left. It sounds like an easy system to break, but all syntax and semantics to the words are lost; only the spaces remained. Look at the following example.



```
Plaintext:    We attack at dawn
Ciphertext:   TB XQQXZH XQ AXTK
```

Without knowing that there is a Caesar cipher in use it looks almost impossible to break. It is hard to recognise that there could be one in use; you could only infer its use by recognising that the spaces are retained and that there are some repeating characters, but this isn't reliable. If you did know the Caesar cipher was being used, you could break the code by using a technique known as *brute force*.

Shifts

Brute force attacks use a repetitive approach to crack the key value and decrypt the data; it is for this reason that the Caesar cipher is vulnerable to this type of attack. Our simple example is limited by the number of letters in the English language, meaning at most you will have to do 25 shifts. This is often done using a *planar table*.

The table on the right shows us that our previous example is n = 3.

Vernam cipher

The Vernam cipher was create by Gilbert Vernam during his research into cry a truly unbreakable cipher to protect data. The cipher text is produced by *sec* sender and receiver of the message/data to be encrypted. The keys are then combination with modulo-26, to produce a truly secret message. But what is

The keys are made of streams of randomly generated letters which have eac combined with the values of the letters in the data to produce an *intermediat* applied to this intermediate to create the cipher text. Take a look at the wor

Encrypting using Vernam cipher

Message	ATTACK					
Character values	0(A)	19(T)	19(T)	0(A)	2(C)	10(K)
Key values	2(C)	11(L)	16(Q)	14(O)	7(H)	24(Y)
Message + Key	2	30	35	14	9	34
Mod addition (26)	2	4	9	14	9	8
Cipher text	C	E	J	O	J	I

Decrypting using the Vernam cipher.

Now that you've seen how to encrypt using the Vernam cipher, you need to k essentially the same process but in reverse, except where you used modular modular subtraction to decrypt.

Message	CEJOJI					
Character values	2(C)	4(E)	9(J)	14(O)	9(J)	8(I)
Key values	2(C)	11(L)	16(Q)	14(O)	7(H)	24(Y)
Message + Key	0	-7	-7	0	2	-16
Mod addition (26)	0	19	19	0	2	10
Cipher text	A	T	T	A	C	K

It has been mathematically proven that, when used properly, the Vernam cip offers no information about the plaintext it represents and that, if the criteri data encrypted using the cipher will be secure if, and only if:

1. The keys must be equal to or greater than the length of the original
2. The keys must be discarded or unused
3. The keys are generated using algorithms that produce truly random
4. The keys are distributed in a secure and secretive manner

Given enough material (cipher text) and time, all cryptographic algorithms, e be broken. This is because all other devised encryption algorithms that aren' algorithm depend on *computation security*.

Computation security is creating security based on an exponential time algo encryption length grows, the time taken to break the encryption increases ex

Questions: Data Compression and Encryption

1 How does lossy data compression reduce the size of a file? (1 mark)

2 Using a left-shift Caesar cipher, decrypt the following message: (1 mar

```
JVTWBALY ZJPLUJL
```

3 Why is it important that the length of a key being used for a Vernam c message being encrypted? (2 marks)

4 Is the Vernam cipher immune to brute force attacks even if the keys ha

6. Computer Systems

In this section we look at the fundamental concepts of computing and how a series of ele program computer that has become part of everyday life. The concepts behind this theory processors is vital for the modern-day computer scientist.

This section covers:

6.1 HARDWARE AND SOFTWARE

RELATIONSHIP BETWEEN HARDWARE AND SOFTWARE

- The *hardware* of a system is the physical components that make up a computer, e.g. the motherboard, power supply, hard drive, peripherals (plug-in devices) such as keyboards, mice, scanners, printers, are also hardware.
- The *software* of a system is a collection of procedures and rules (lines computer code) that carry out operations on a computer's hardware. software code may be stored on the computer's hard drive, a DVD, a stick or any other storage device.

It is necessary to provide input to make the computer do something. This inpu through many different means, for example your mouse, keyboard or scanner other devices. Less obvious input is the power button, the computer's interna (which has its own battery) and possibly instructions given to the computer fr

The software reads in the input, works out what it is that you've told it to do, carry out a certain specific sequence of instructions to achieve the desired re sent to an output device such as your monitor or printer.

Both hardware and software are essential for a computer to run; without the s give the computer through the input devices would not be recognised – the c boot up! But without the hardware the software would not be able to physical

The hardware and software in your computer are not just working when you t from the moment you press the 'on' button the two are working together to p when there doesn't seem to be anything happening on the computer, they are running continual checks and maintenance to ensure that everything runs sm

CLASSIFICATION OF SOFTWARE

The purpose of a computer is to execute the software store on its hard drive, the computer will not [illegible] For example, you can have a computer comprise without [illegible] tor [illegible] boot system the computer won't start. Software can be cl system software and application software, depending on its purpose.

- *System software* is independent of any general-purpose package or any particular application area, but is designed to assist the computer in th efficient execution of application programs. An example of a piece of s software would be an operating system (such as Windows or Linux).
- *Application software* allows the user to achieve a specific task that the wishes it to perform; this could be anything from a word-processing p or web browser, to the software that helps to run a nuclear power stat

SYSTEM SOFTWARE

System software acts as the brains behind the operation as far as managing without it, the computer is just a collection of components that wouldn't fu system software, including the *operating system*, *library programs*, *translator* s

Library programs

A *library program* is the generic name given to a collection of programs which software. If you install certain modern games you will be asked if you accept required for the computer to interpret the instructions required to run the ga provide libraries required to run built-in services. The code in libraries is not edited by independent programmers to allow them to save having to create c explorer can be used to save files instead of the programmer having to create

Translator software

All software translators have a single purpose and that is to convert one pro Translators are used to convert source code created by a programmer into a understood. The three main types of translator are:

- *Assemblers* are used to convert a low-level language called '*Assembly* code. Assembly language used to be the only option other than codi of words to represent memory locations or operations; these words code is a language that the computer can use and understand, and is
- *Compilers* check that all the lines of the program are valid to the synt converting the entire source code into object code. Object code is us low-level code which is dependent on the machine it is running on. may be translated to machine code using assembler, linker, binders the compiled code is distributed as software for a particular machine programs for sale. The original source code would be similar with mi compiled into a particular machine format.
- *Interpreters* are similar to compilers except that they read the source the statement is valid and then execute that line before moving on t until the end of the source code is reached. Some programming lang operation. For example, JavaScript is an interpreted language. Each JavaScript interpreter which translates the code received into machi languages are slower in operation than a compiled program but have across platforms, i.e. to change a JavaScript code you change the pro and distribute for each platform.

Utility programs

Any programs that are operated by the user to maintain the functionality of are given the title *utility programs*. These are small and useful programs that seen in file management, diagnostic tools and system information tools that information (i.e. system resource usage) about your computer.

Most utility programs that are required for safe, maintainable operation of y computer are included with your operating system software package and co disk defragmentation, file explorers and everything down to copy-and-paste maintain or configure, monitor certain items on your system or enable trans

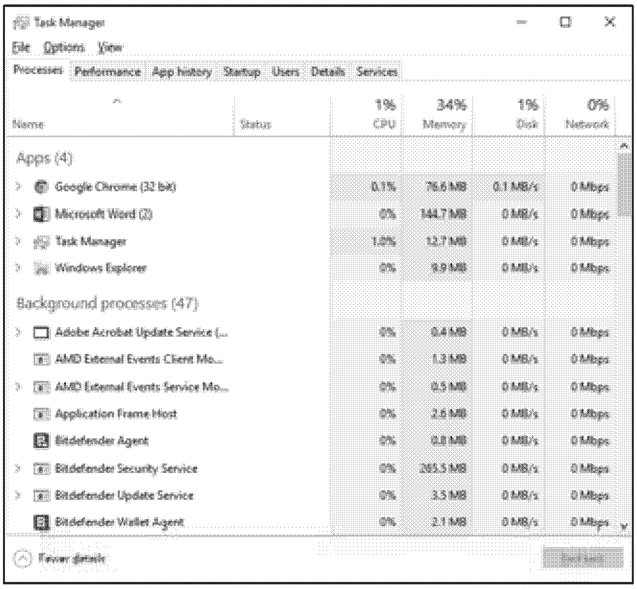
ROLE OF AN OPERATING SYSTEM

The operating system (also known as the *OS*) is the single most important a necessary piece of system software. While the computer is in use, the operat runs in the background managing the system's resources and processes. System processes include communications with input/output devices, managing memory and managing programs – meaning the user doesn't need to worry about how the system resources are being used (unless they're doing a task which is particularly resource-intensive, e.g. 3D rendering). This acts as an interface between the user and the computer, application software and components of the system.

The operating system is not a single entity but a collection of programs that work together to supply a level of abstraction for the user. For example, a user can tell the computer to delete files from a directory and the computer interprets this and issues the corresponding instr to carry out the task. The abstraction is achieved by creating a *virtual machin* the complexities of the tasks are hidden from the user.

The general role of an operating system is to provide an environment from w This results in operating systems needing to provide the following services:

- Resource management (process management, memory managemen
- User interface


Task Manager
File Options View
Processes Performance App history Startup Users Details Services
Fewer details
End task

Name	Status	1% CPU	34% Memory	1% Disk	0% Network
Apps (4)					
Google Chrome (32 bit)		0.1%	76.6 MB	0.1 MB/s	0 Mbps
Microsoft Word (2)		0%	144.7 MB	0 MB/s	0 Mbps
Task Manager		1.0%	12.7 MB	0 MB/s	0 Mbps
Windows Explorer		0%	9.9 MB	0 MB/s	0 Mbps
Background processes (47)					
Adobe Acrobat Update Service (...		0%	0.4 MB	0 MB/s	0 Mbps
AMD External Events Client Mo...		0%	1.3 MB	0 MB/s	0 Mbps
AMD External Events Service Mo...		0%	0.5 MB	0 MB/s	0 Mbps
Application Frame Host		0%	2.6 MB	0 MB/s	0 Mbps
Bitdefender Agent		0%	0.8 MB	0 MB/s	0 Mbps
Bitdefender Security Service		0%	265.5 MB	0 MB/s	0 Mbps
Bitdefender Update Service		0%	3.5 MB	0 MB/s	0 Mbps
Bitdefender Wallet Agent		0%	2.1 MB	0 MB/s	0 Mbps

Process manage

Process manageme switching of progra operating systems a to run at the same t play an MP3 file an In effect, this is wh each processor can

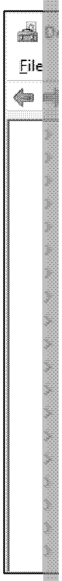
Many modern comp each of which can h means that a certai simultaneously. Ho computer systems t processes at once a need to switch betw

An operating system schedules programs and switches between them by ma

Switching between programs is so fast that it appears that multiple process Since there are many processes that want to run, they have to be organised i and priorities should be given to them. To be fair to the programs, all of the least o a limited amount of time before then being switched in an ord affect the response time for interactive applications (delay when using mous also try to finish as many processes as possible, and also balance out the sm so that everyone gets a fair chance.

Users may wish to alter the priorities of processes, and this can sometimes b application such as the Task Manager in Windows *(shown on p.3)*. For exampl edit photos, the user may want to increase the priority of the video applicati editing the photos.

Device management



Managing devices using an operating system is useful because the operating system can make each device accessible to programs. In order for devices to be handled appropriately by the operating system, device drivers are needed. A *device driver* is a special piece of software that controls the hardware and importantly provides an interface so that programs can use the device. In other words, device drivers give a layer of abstraction to the software which makes use of the devices.

By using this abstraction, programs can access and perform operations on the hardware via simple function calls. An example could be the following command: 'play sound from file'. Behind the scenes the file would be decoded and sent to the sound card which would then process the sound and output it to the speakers. From the programmer's point of view, the only thing that he or she is required to know is the function name.

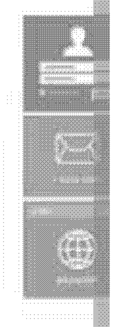
Hardware devices are usually organised in terms of priorities. When a hardware device wants to perform a duty, an interrupt is signalled. It
decide whether or not interrupts are allowed to occur. If they are, then the o
interrupt and processes it by saving the current state of the running program
executing an interrupt handler which directs the flow of execution to the ap
device causes an interrupt, then the operating system decides which device
that device to obtain the attention of the processor by executing the interru

Memory management

Memory in a computer system is finite and used by all processes, and theref
the operating system will have to divide up and keep track of the memory th
is available. Sharing a finite resource in a fair and effective way between ma
entities is not an easy job. In order to avoid processes running out of memor
and crashing, many operating systems operate using virtual memory.

This is where the operating system moves the contents of memory to and fr
intelligently – hard disks are generally much slower than RAM, and so overu
down considerably.

Provision of a virtual machine



Abstraction is the key idea behind operating systems that makes them so popular today. By abstracting all the processes behind hardware and software, in addition to providing user-friendly interfaces, all the complicated operations behind a computer system are hidden.

A machine which is complicated in the background but easy to use due to these layers of abstraction is also classed as a virtual machine.

Questions: Hardware and Software

1 Identify the software types of the following:

a) Antivirus software (1 mark)
c) C# compiler (1 mark)
b) Ubuntu (1 mark)
d) File explorer (1 mark)

2 How is abstraction created between the user and the computer syste

3 What is the main drawback of using an interpreter for translation? (1 m

6.2 CLASSIFICATION OF PROGRAMMING LANGUAGES

Programming languages haven't always been advanced as they are now – as we've taken baby steps to where we are now – but before you can learn abo languages you must first understand the concept of *language levels*.

LOW-LEVEL LANGUAGES

Low-level programming language's defining feature is the lack of human-lar no abstraction between the programmer and the instructions being program highly dependent on the design of the system and are described as *machine-*

There are two low-level languages that you need to know about; these are *m*

Machine code

Machine code is the lowest coding language and is comprised entirely of 1s a on and off electrical states in the computer. Machine code is usually written hexadecimal representation of the byte, because it is more concise to type, b frustratingly tedious and difficult. However, because of how precisely the pro programs were highly dependent on what processor they were being run on.

Machine code instructions

Although when you look at machine code it just looks like a load of 1s and 0 commands and data. Here is an example set of commands for a machine cod

```
    0000    X   Store accumulator value in memory location X
    0001    X   Load contents of memory location X into the a
    0010    X   Set accumulator value to X
    0011    X   Add contents of memory location X to accumula
    0100        Halt execution
```

The *accumulator* is a register that is used to manipulate numbers. X in this ca represents a memory location *(see Section 7 for more information).*

Machine language example 1

Using the machine instructions above, this program adds the values in memo puts the answer in 0010:

```
    0001    0000    0011    0001    0000    0010    0100
```

Machine language example 2

As mentioned above, the code is shown in the hexadecimal numbering syste space on the page. The computer commands shown here (taken from machir for an IBM mainframe computer) are operation codes instructing the comput two numbers, compare the quotient, move the result into the output area of and set up the output so it can be printed.

Assembly language

Assembly language is considered a very low-level language by today's standa big leap forwards in programming design because of one particular feature: r

Instead of the 1s and 0s used in machine language, assembly languages use abbreviations that are easy to remember. For example, an assembly languag Compare, 'MP' for Multiply, STO for storing information in memory, etc. Altho words, they still make programming a lot simpler than coding in hexadecima

Assembly language also allows for the use of rudimental variables such as C memory address. These mnemonics are said to have a one-to-one relationsh which is what allows them to be used unambiguously.

As with all languages, assembly language requires a *translator* to be read by translator software is called an *assembler program*, or simply *assembler*. It tak assembly language and converts it into machine code instructions.

Assembly language example

This example shows an assembly language extract from a programming language called A68000. Notice that the names for subroutines are given on the left, the instructions are given in the next column and memory locations (variables) and other information are given in the third column.

An exclamation mark (**!**) precedes a comment. Note that this code does exactly the same as the high-level language shown in the *high-level* section.

HIGH-LEVEL LANGUAGES

High-level programming languages, as you might have guessed, contain ele elements of natural spoken languages. These elements are what provide abs and the instructions being programmed. Instead of dealing with registers, me codes, high-level programmers deal with variables, data structures and comp the complex memory pointers are hidden from the programmer and the com

Examples of high-level languages include the group of programming langua Imperative languages are the most dominant and widespread paradigm for p Basic, C#, Pascal, etc. The main characteristic of these languages is that all s default and the flow of the program is diverted using constructs (i.e. *iteration* which statements are executed is crucial to the success of the code. Variable the 'state' of the program which can be used to control the flow of executio

Compilers

A translator is needed to translate the symbolic statements of a high-level la machine language. This translator is usually a compiler. Keep in mind the fo

- A compiler converts a high-level language into machine code. As eve a different machine code, it follows that each different computer nee radically different to take advantage of the particular machine code). different compilers for each language.
- Although each compiler is different, high-level languages are more p source program can be used on different machines, and the compiler the correct machine code.
- Different organisations may bring out different compilers for the sam improvements or bug fixing which result in more than one version o Sometimes the high-level language may have differences to account machine. Therefore standardisation is needed to keep the portability

High-level code example

Many examples of high-level code can found throughout this resource, for pr Python and VB .NET. Below is an extract of C# code that is a translation of t language example.

```
Static int sign (int i)
{
  int sign = 0;
If (i <> 0)
  {
    if (i < 0)
    {
sign = -1;
    }
    Else
    {
      sign = 1;
    }
  Else
  {
    sign = 0;
  }
Return sign;
}
```

Did you know?!

Back in the day programming was a early 1990s magazines would publisl use and even algorithm puzzles in as trace and dissect. Suddenly Sudoku d

Questions: Programming L

1 Using the high-levelled languag

a) Write comments for what t

b) State what the output wou passed into the function? (

2 What are the main differences l language and machine code? (3

3 If a high-level language runs or different system? Explain your a

6.3 TYPES OF PROGRAM TRANSLATOR

As mentioned earlier, a *translator* converts program statements written in on programming language into another programming language. The most comm translators are assemblers that convert program statements written by a programmer in an appropriate language for the job (the source code) into machine code that can be executed by a computer.

Originally, programming was carried out in machine code. Programmers wou type in commands using hexadecimal, which was converted to binary commands for the computer. A major step forward was when assembly languages were written which had commands that corresponded more close to English abbreviations (*mnemonics*).

Assemblers

An *assembler* then converted these assembly language instructions into the r

Early assemblers did little more than convert instruction mnemonics to their codes, but later assemblers did much more. These assembler tasks have to b instructions can be converted to machine code:

- If assembly language supports the use of *macro* instructions (the equ language), they are expanded by the assembler and the code inserted in assembly. Instructions are checked for syntactical correctness; errors are is constructed to link symbolic operands and labels with their correspon
- Many assemblers support the use of *pseudo-operations*, or *directives*, w translatable into machine instructions. These involve such things as re values of identifiers used in the program and defining where the progr

Interpreters

An *interpreter* takes one line of a high-level language source code program a instruction for that programming language, translates the statement to mac resulting machine language before translating the next program statement.

Advantages of an interpreter over a compiler are:

- The programmer can execute the program and sort out each problem even if there are many invalid lines of code later on in the program. interpreter while developing a program.
- If an error is found the source code can be corrected.

Compilers

A *compiler* checks that all the lines in the program are valid. Then, providing the whole program into machine language that can then be executed by a c date. Advantages of a compiler over an interpreter are:

- The resulting machine code runs faster than an interpreted program,
- The resulting machine code can be run on a computer that doesn't h
- A person given a copy of the compiled program can't see the original people stealing the code for their own uses.
- The compiler program doesn't have to be resident in memory at the larger programs can be run in memory.

When are translators used?

Assembler	An assembler takes assembly code, which is written in mne These are understandable by the computer and therefore ca
Interpreter	Translating code a line at a time and running it is very usef languages which have both an interpreter and a compiler y development for easy bug fixing, and the compiler when yo working to produce the executable program. Interpreters are also useful for compiling on the fly on web JavaScript.
Compiler	These translators are used to create an executable file of m language. This executable file can then be run at a later tim Because the compiled code runs faster than trying to compi can give a performance boost for computationally heavy pr

6.4 LOGIC GATES

All modern computers work on a binary system, so it is important to have a v formulate [illegible] circuits. In logic circuits you're generally answering a 'yes' or 'no' q formulate a final outcome. For example, a logic circuit may produce a respon the computer?' by asking 'am I bored?' and 'do I have any work to do?' – if th answer to the second is no then you can play on the computer.

We may want to accept a user if he is the system manager or technician, if th correct, and if the user is not barred from the system. This is just one use of operations. In programming we apply them to *conditions* (a condition in this or false). They are often used in conjunction with relational operators, which

TRUTH TABLES

The following tables show the outcomes when each logical operator is appli

- NOT is used to negate (or change) a condition.
- AND is used to check that both of two conditions are true.
- OR is used to see whether either of two conditions is true.
- *NAND* (Not AND) provides the inverse of the AND operator.
- *NOR* (Not OR) provides the inverse of the OR operator.
- *XOR* (exclusive OR) is used to see if only one or the other applies.

Note: XOR is not a basic logical operator as the equivalent test can be made usi here as it is useful to see whether two conditions are different.

Here you can see the completed truth tables for all logical operators.

Condition A	Condition B	A AND B
True	True	True
True	False	False
False	True	False
False	False	False

Condition A
True
True
False
False

Condition A	Condition B	A NAND B
True	True	False
True	False	True
False	True	True
False	False	True

Condition A
True
True
False
False

Condition A	Condition B	A XOR B
True	True	False
True	False	True
False	True	True
False	False	False

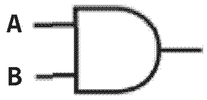
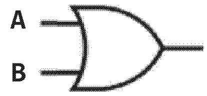
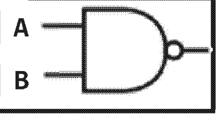
Condition A
True
False

LOGIC DIAGRAMS

There is a better way of representing logical procedures, and it allows the read of a procedure. We call these *logic diagrams* and they make understanding a pr

Each logic gate has its own symbol and one or more inputs, and produces a s together it is possible to create circuits that perform a complicated logical o

The examples of the logic gates can be explained using the truth tables abo left can take the values of the conditions A and B from the columns of the ta

Operator	Boolean	Logic Diagram
AND	$(a \cdot b)$	
OR	$(a + b)$	
NAND	$\overline{a \cdot b}$ or $(a \cdot b)'$	

Operator	B
NOR	$\overline{a+b}$
XOR	$(a \cdot \overline{b}$
NOT	$\overline{a}$

LOGIC CIRCUITS

Logic gate symbols can be combined to create circuits of logical operations t input to give an output. In a Boolean circuit the information flows from left t

Below is an example of a simple Boolean circuit which has the following inp

A → *is the weather forecast sunny?*

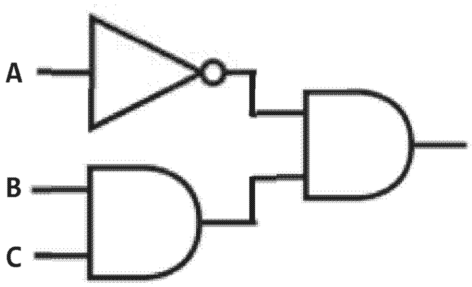
B → *is the weather forecast not windy?*

C → *is the umbrella not broken?*

This produces a response to the question: 'Shall I take an umbrella?'

One possible set of inputs where the circuit will produce a 'yes' is if weather and neither is the umbrella broken (C) nor is there wind forecast (B). Putting

A' AND (B AND C))



We can construct the truth table for the above Boolean circuit. There are thr possible input combinations each resulting in one true (T) or false (F) output would be 2^4 combinations, five inputs would mean 2^5 combinations and so o

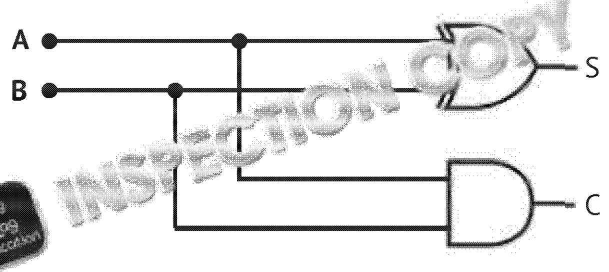
Input A	Input B	Input C	Output (A' AND
True	True	True	Fals
True	True	False	Fals
True	False	True	Fals
True	False	False	Fals
False	True	True	True
False	True	False	Fals
False	False	True	Fals
False	False	False	Fals

Half adders and full adders

In electronics, an adder is a combination of logic gates that are used to produce t
In modern computers adders are used in the processor and in system memory wh
calculating addresses for memory allocation, calculating return addresses and in

Half adders

Half adders add two single bits that we will call *A* and *B*. It uses these two in
(represented by the letter S) and a signal known as *carry* (C). The carry signal
used for producing full adders, as seen below. The example below uses an X
table can be seen below.



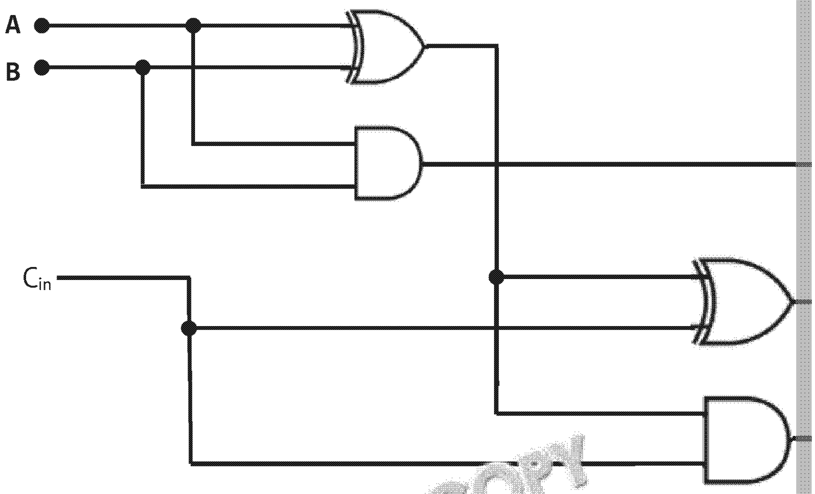
A
Fals
Fals
True
True

Full adders

Full adders are the combination of two or more half adders and where the ha
output for sum and carry, the full adder will take the carry and use it as a *car*

Consider you're adding together two 3-bit numbers. You will need to create t
them to create the full adder. The carry-out bit of one is when you're adding
carry the one 1. Therefore your truth table will have five columns.

The logic gate diagram and truth table can be seen below.



A	B	C_{in}	C_{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Edge-triggered D-type flip-flop circuits as memory units

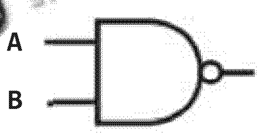
Flip-flop circuits can take a signal and, depending on previous states, alter th
sounds complicated it is the basic principal for computer memory.

Edge-triggered D-type flip-flop circuits are used in *static* RAM modules. Static R
usually used for core system operations because, although the speed is much
sizes of the modules are much smaller and they take up considerably more s

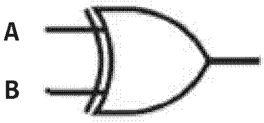
Questions: Logic Gates

1 State the outputs of the following logic gates.

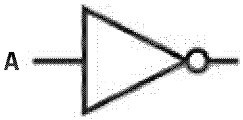
a) Where A is TRUE and B is FALSE?



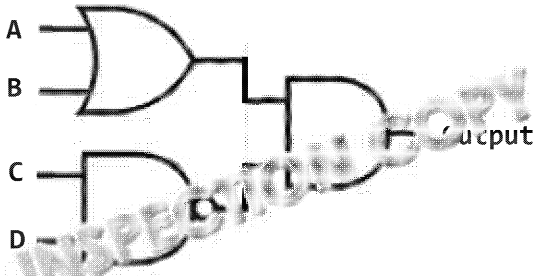
b) Where A is FALSE and B is FALSE?



c) Where A is TRUE?



2 Draw a truth table for the following Boolean circuit:



3 Construct a Boolean circuit for the following: (1 mark)

(A+B) • (C+D)

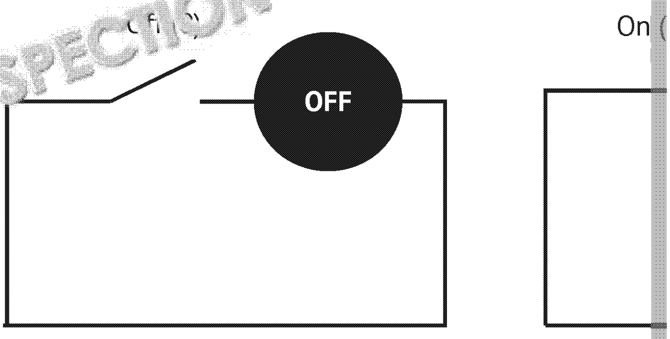
6.5 BOOLEAN ALGEBRA

In Boolean algebra there are always only two possible outcomes, true or fals represented as 0 or 1). Boolean algebra has a set of *postulates* which are basi all the possible operations. In Boolean algebra we read a + as OR and · as AN

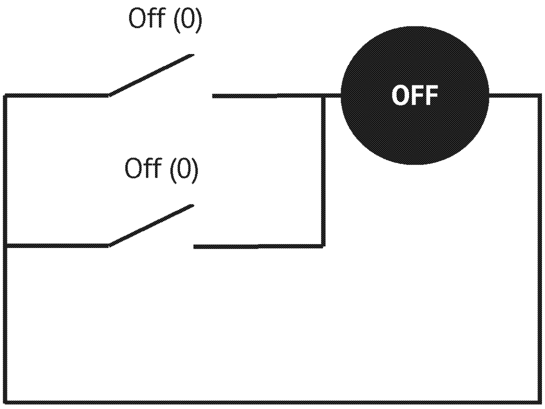
One way to visualise Boolean algebra is as simple electrical circuits with swi is on the corresponding Boolean expression is outputting a 1, and a 0 if the two switches on the same wire and an OR is equivalent to two switches on s (or closed) then it is equivalent to a 1 and a 0 if it is off (or open).

THE BASIC RULES OF BOOLEAN ALGEBRA

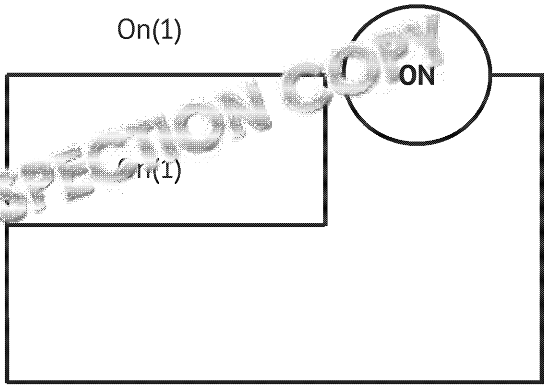
1. A variable can be either 1 or 0



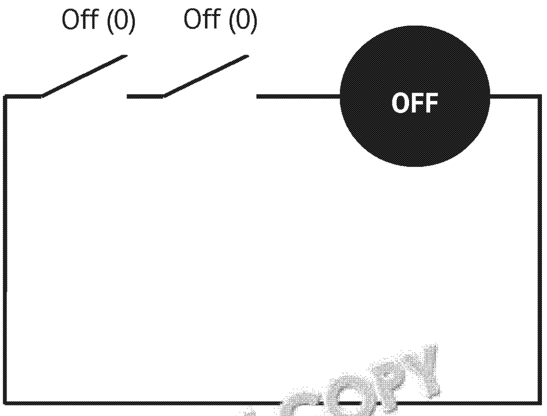
2. 0 + 0 = 0 (off OR off = off)



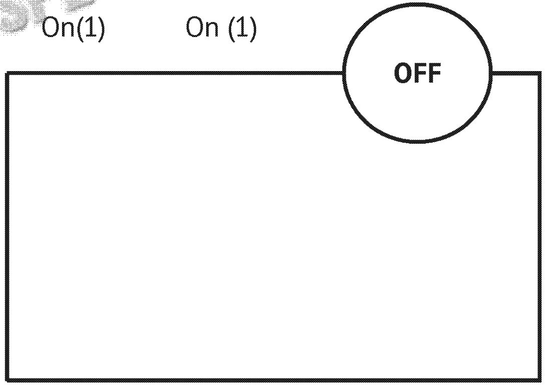
3. 1 + 1 = 1 (on OR on = on)



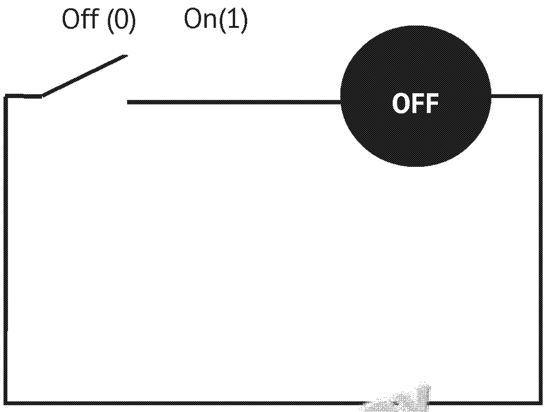
4. $0 \cdot 0 = 0$ (off AND off = off)



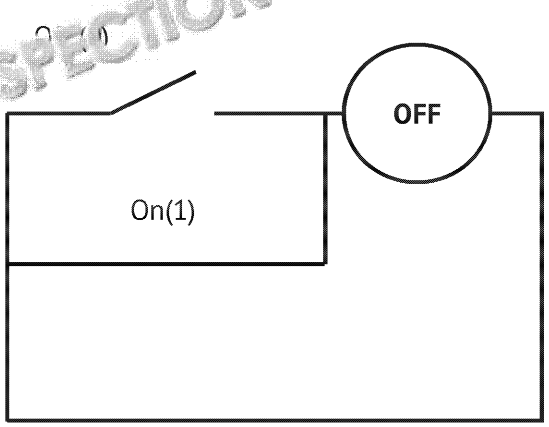
5. $1 \cdot 1 = 1$ (on AND on = on)



6. $0 \cdot 1 = 1 \cdot 0 = 0$ (off AND on = off , on AND off = off)



7. $1 + 0 = 0 + 1 = 1$ (off OR on = on, on OR off = on)



The Boolean algebra laws

There are 11 Boolean algebra laws, 10 of which are shown below. The eleve discussed in more depth in the following section. In this section *A*, *B* and *C* r remember that $\overline{A}$ means not (*A*) and *A*, *B* and *C* can be any Boolean expressi umbrella circuit example described earlier in this section *(see p.10)*. The laws expressions, normally with the intention of simplifying them.

1. $A + B = B + A$
   $A \cdot B = B \cdot A$

   This means that Boolean algebra is *commutative*; the order of the inputs

2. $(A + B) + C = A + (B + C)$
   $(A \cdot B) \cdot C = A \cdot (B \cdot C)$

   This means that Boolean algebra is *associative*; the order in which the s

3. $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$
   $A + (B \cdot C) = (A + B) \cdot (A + C)$

   This law means that Boolean algebra is *distributive* so a multiplication c of two other multiplications. This is the equivalent of 'expanding bracke

4. $A + A = A$
   $A \cdot A = A$

   This law is very different to normal algebra but is a consequence of the possible outcomes in Boolean algebra so adding (or multiplying) two tr another true. It is known as the *identity law*.

5. $A \cdot B + A \cdot \overline{B} = A$
   $(A + B) \cdot (A + \overline{B}) = A$

   Because either B or not B will be true, multiplying by each of them will having a true outcome.

6. $A + (A \cdot B) = A$
   $A \cdot (A + B) = A$

   Although this law doesn't look like it should be true, it can be proved by $A + (A \cdot B) = A \cdot (1 + B) = A \cdot 1 = A$. This law is known as the *redundancy la*

7. $0 + A = A$
   $0 \cdot A = 0$

   This law simply describes the effect of adding or multiplying by 0.

8. $1 + A = 1$
   $1 \cdot A = A$

   If there is one true input in a Boolean expression then the outcome will b or adding with a true value means that the result will always be true.

9. $A + A = 1$
   $A \cdot A = 0$

   This law follows from the previous two, remembering that when you ha always be true (1) and the other false (0).

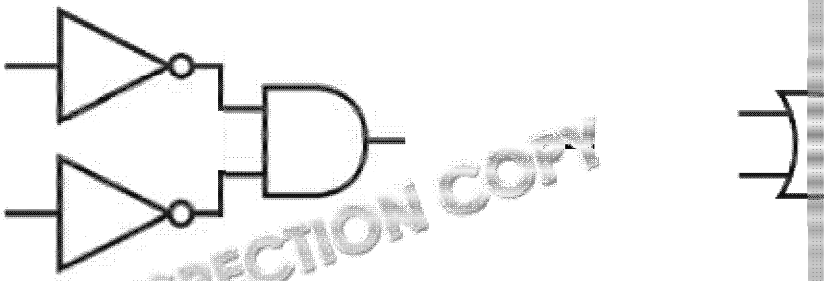
10. $A + (\overline{A} \cdot B) = A + B$
    $A \cdot (\overline{A} + B) = A \cdot B$

    Similar to rule 9, this law holds because *A* and $\overline{A}$ will always have one

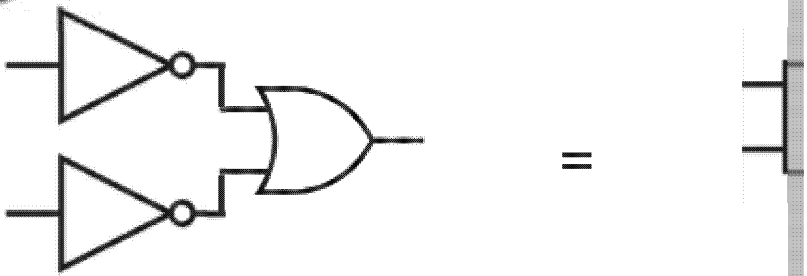
De Morgan's law

De Morgan was an English mathematician who moved to England from India discovered that in logic it was possible to show that the following two sets c allows for Boolean circuits to be simplified and hence more understandable, space if they are used in hardware devices.

De Morgan's first law is $\overline{(A+B)} = \overline{A} \cdot \overline{B}$



De Mo econd law is $\overline{(A \cdot B)} = \overline{A} + \overline{B}$

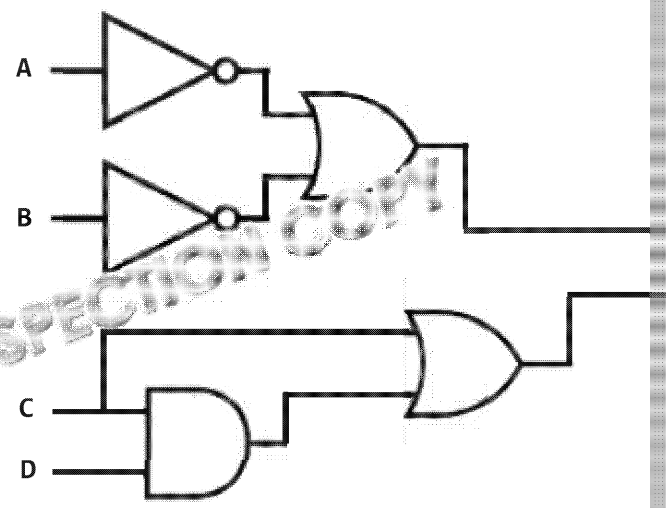


You can check these laws by constructing the truth tables for the circuits and

By using these laws it is possible to see how Boolean expressions can be sim circuit (or sub-circuit if it is part of a larger circuit) with the simpler two-gate

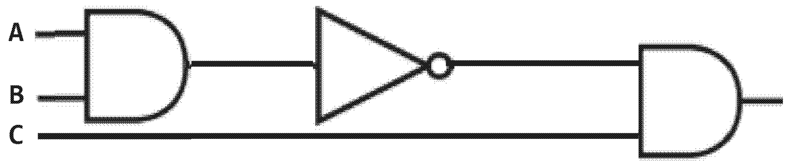
SIMPLIFYING A BOOLEAN EXPRESSION

Using the laws it is possible to take a typical Boolean circuit and simplify it. Belo



- This is the circuit for *($\overline{A}$ OR $\overline{B}$)* AND *(C OR(C AND D))*, which can be v
- This circuit can be simplified using the second De Morgan's law and
- Law 6 simplifies the *(C+(C.D)* to *C*
- De Morgan's law simplifies *(A'+B')* to *(A.B)'*

- So the expression becomes *(A.B)'.C*



- This is the circuit for $(\overline{(A \text{ AND } B)} \text{ AND } C)$.

As you can see, this is a much simpler circuit and as result will be much simp inside the computer.

Simplifying Boolean expressions can be performed like simplifying mathema

Example

Simplify (A+B)(A+C)

We can effectively 'multiply the brackets':

AA + AC + BA + BC

AA is equivalent to *A*

A + AC + AB + BC

We can see a 'factor' in the first three expressions of A:

A(1 + C + B) + BC

Using law 8 we know that 1 or anything is 1:

A.1 + BC

Using law 9 we know that 1 and A is A:

A + BC

As can be seen, the simplified expression would be less complicated to cons

Questions: Boolean Algebra

1 Use the laws of Boolean algebra to simplify the following: (5 marks)

a) $A + (A \cdot B)$

b) $A \cdot B + A \cdot \overline{B}$

c) $(A + A \cdot B) + (A + A \cdot B)$

d) $A + \overline{B}$

e) $\overline{(\overline{A}+\overline{B})}+B$

7. Computer Organisation and Archi

Computer architecture is an extension of the logic shown in the previous sections. Knowledg machine actually handles information is a vital part of understanding how a computer operat

This section covers:

7.1 INTERNAL HARDWARE COMPONENTS

INTERNAL STRUCTURE AT A GLANCE

So far you have already covered what is required in order for components of allow users to use the computer to complete a task *(see Section 6.1)*. Even th and more complex the list of components found in the average machine has

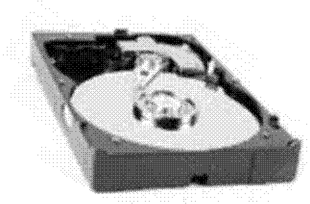
When you open up your computer, inside you will find the following:

Motherboard

Everything in the computer is in some way connected to the motherboard, b via slots, wires, connectors or readouts; everything must connect to the cent board. The motherboard acts as a central interface for all components; not a components connect to the board in the same way, though. How a device is connected depends on the bandwidth of the device, how often it is used and what the device is used for.

For example, the *Peripheral Component Interconnect Express (PCI-E)* lanes tha as graphics cards are not the same as those used to connect hard drives.

Hard disk drive (HDD) and solid-state drives (SSD)



For a long time, HDDs have been the most commo the computer to store its programs and data. They computer thinks there is more than one hard drive multiple operating systems to be installed simulta HDDs are large in volume but are not that fast com

Nowadays, SSDs are more being used more increasingly, either alongside a Despite being more expensive than HDDs, their write/speeds are much faste system to boot in just seconds.

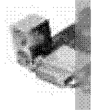
Power supply

The *power supply unit (PSU)* provides and regulates the supply of power to th all other components. Modern PSUs allow the computer to control when it t saving power and for the user to define how they want the power to be delivered to the system.

Power supplies come in varying sizes – not just in terms of their physical size, but in the number of watts they can support; the standard power supply on an average computer is roughly between 200 and 500 watts but there are high-tier PSUs available that can deliver up to 1,500 watts!

How the pieces are connected

As mentioned previously, all the components are connecting to the motherboard. The method of communicating along the wires connecting components to the motherboard may be restrictive, i.e. may limit the speed at which the computer can operate.



The hardware components on the motherboard are also connected together using printed circuitry called *buses (see p.3 for information)*. The connections are vital for different parts of the computer to work togethe for instance loading a program code from the hard disk into RAM when an a is launched.

A CLOSER LOOK AT INTERNAL STRUCTURE

Processor

The *Central Processing Unit (CPU)* is the brains of the computer and is the co the work of everyday applications. It manipulates the information sent to th information may be instructions for the processor, or data for the processor t A computer is not limited to just one processor; some machines require a *va* volume of processing power and thus have processors CPUs. In desktop co CPUs are mounted in a socket attached to the motherboard, and enclosed w *heatsink* (as shown on the right) which helps to prevent overheating.

The processor speed is given in megahertz (MHz) or gigahertz (GHz). A singl means that the computer is performing one million instructions per second a single GHz is one billion instructions per second. A processor can deal with c numbers of bits of information per instruction; with modern computers this i usually 64 bits, or 32 bits for a modern laptop.

Main memory

Main memory is the memory that is directly accessible to the processor. This n order not to slow the system down – an event known as *bottlenecking*. There a

- *Read-only memory (ROM)* is a permanent area of storage for special p installed during the process of computer manufacture. The contents because the data has been written onto a ROM chip which cannot b RAM, ROM retains its contents even when the computer is turned o being non-volatile, whereas RAM is volatile.

  The *basic input/output system (BIOS)* is a part of ROM that stores criti that starts the computer. For example, a pocket calculator contains 1 store the instructions of the calculator.
- *Random access memory (RAM)* constitutes the working area of the co used for storing programs and data currently in use. Modern RAM (often gigabytes) and frequency (often megahertz). Currently, th RAM for a new modern desktop computer is around 4 to 6 gigab modern boards can support up to a staggering 64 gigabytes of RAM. *volatile*, meaning once the power supply is shut off all data that was lost. RAM is directly written and read by the processor. This takes ti the transfer rate the better; this is where frequency comes in.
- *Cache memory* holds frequently accessed code and data. It is *extreme* processor so that it doesn't need to access main memory where the copied. This limits the size of how much data can be cached and so c just megabytes).

Did you know?!

Theoretically, the most RAM a 64-bit computer could possibly contain would be with today's technology and using modern standards of RAM capacity and main slow operations due to finding the specific RAM chip a piece of data was stored roughly 16 square kilometres!

Buses

The concept behind the term computer *bus* is similar to that of a vehicle bus one place to another inside the computer. Although it is convenient to think physically transferred from one place to another, in fact a bus line is just an wires) along which 1s and 0s are sent. This width is what limits the amount called its *word size*.

As with most components, more is better – the larger a word size or bus is, t This is because if the computer has a larger bus size, it can transfer more da faster; it can reference larger numbers allowing more memory, so the compu and variety of instructions.

Buses are often common pathways shared by electronic signals to and from computer. However, not all buses in the computer are the same and manufa buses; these are:

- *Data buses* carry the data being exchanged around a system in their likely to have the largest word size as actual information will be the be sent via the bus. The other buses will determine how and where t processed or stored.
- *Address buses* carry the information about where the data is being se the same wires go to each component in turn. The components watc that they recognise is present. When this occurs the computer either bus or places new data onto the data bus for the CPU to use. The nu addresses is equivalent to 2n where *n* is the bus size.
- *Control bus* is the bus that carries the signal that regulates data flow with timing operations such as memory writing, memory reading and bus follow strict timing sequences, some operations taking longer th
- *PCI (Peripheral Component Interconnect) bus* is the bus to which most in a modern PC are connected. It runs at 33 MHz, with a bandwidth than the old ISA standard, which only allows 8 Mbps. Finally, the PCI ISA's 16-bit width.

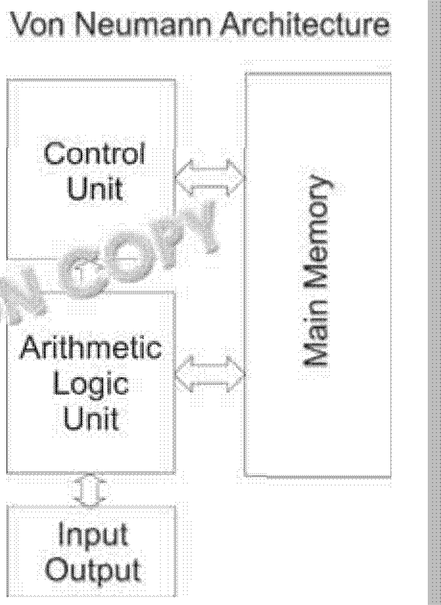
I/O controllers

Input and output controllers provide the interface between the ports and the computer. Most modern motherboards have a number of ports built in so th cards, which are available for each of the port types below in order to add a computer.

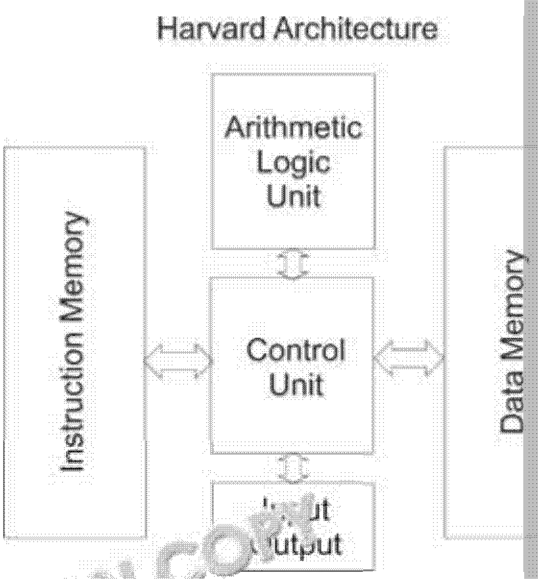
Computer architectures

There are two types of architecture you are expected to understand: the *Von Ne*

The Von Neumann was the earlier version in which memory is used to store bo processor uses the fetch–decode–execute method *(covered in 7.3)* and therefor



The Harvard architecture uses separate memory for data and programs/instructio centre of the structure. It also allows the processor to *pipeline*, often utilising a R



A simple comparison summary is shown in the table below.

Von Neumann	
Single storage system for programs and data	Separate storage sy
Each instruction takes two clock cycles (decode and execute)	Processor can comp cycle if pipelining i
Pipelining cannot be implemented	Pipelining can be i
Older than Harvard, much more robust	Modern architectur

Embedded systems such as those found in digital signal processing systems widely, whereas the Von Neumann architecture is primarily used in general-

Addressable memory

In the early days of computing, computers didn't need large volumes of RAM themselves were very simple and the majority of data processing could be d in the processor itself to complete the operation. However, as computers be rely on the computer being able to store quantities of data on a medium and currently being used.

Therefore, *addressable memory* is memory that is accessible from a compute RAM). When the processor needs to access a section of memory it conveys w the address bus and uses the control bus to state whether it is reading from The data bus is then used to transmit data to or from the address location.

Questions: Internal Hardware Components

1 How are the buses used when transferring data? (3 marks)

2 What is RAM, what is it used for and why is it called 'volatile'? (3 mark

3 Which components are the following describing?

a) A component that contains the majority of the buses and acts as must connect through. (1 mark)

b) A small volume of memory that contains frequently used data. Th very limited volume. (1 mark)

c) A component that acts as the interface which all external data bei speed is measured in hertz which measures the number of calcula

7.2 THE STORED PROGRAM CONCEPT

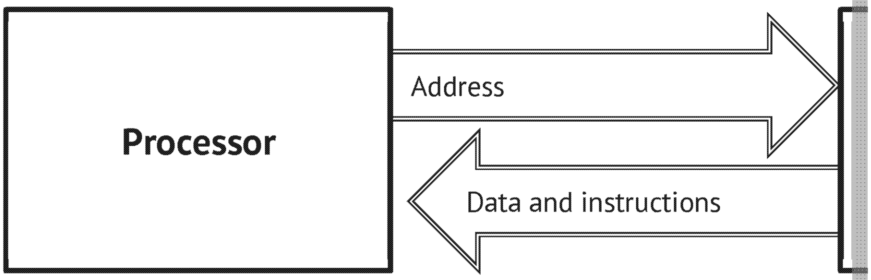
The stored program concept was born in the 1940s out of an idea of John Vo computer programs could be run in computer memory rather than externally the instructions to be run immediately and to be modified by the computer d

The concept proposed that:

- In order to execute a program it must be contained in main memory.
- Instructions are read in machine code and are fetched, decoded if ne memory and executed in the processor.
- The processor then performs the arithmetic, or logical, operations of

The instructions would be collected in a periodic manor (serially) and then th arithmetic or logical operations that each instruction defined. The results of determine changes to the stored instructions and hence the course that prog

This concept allowed for the introduction of much more complicated and us type of computing that we are used to today.



This idea is linked closely with computer organisation and how we design ar concept allowed digital computers to become more adaptable and more flex concept. It was the first presentation of a computational device where the re could change the outcome of the program that required no human interventi

The concept is still used to this day by most processor and computer manufa makes the machine. A computer can be adapted and built upon to perform a to how the hardware will interact; without the stored program model there v what hardware could be used with other hardware which would make buildi which possess moderate computation capabilities, almost impossible.

Did you know?!

The introduction of the stored program concept as well as theoretical compute innovation in the field that has allowed us to develop our technology to where w and potential that it is hard to imagine what computer architectures would be li

The most notable of the stored program implementations is the Von Neumann earlier in this chapter.

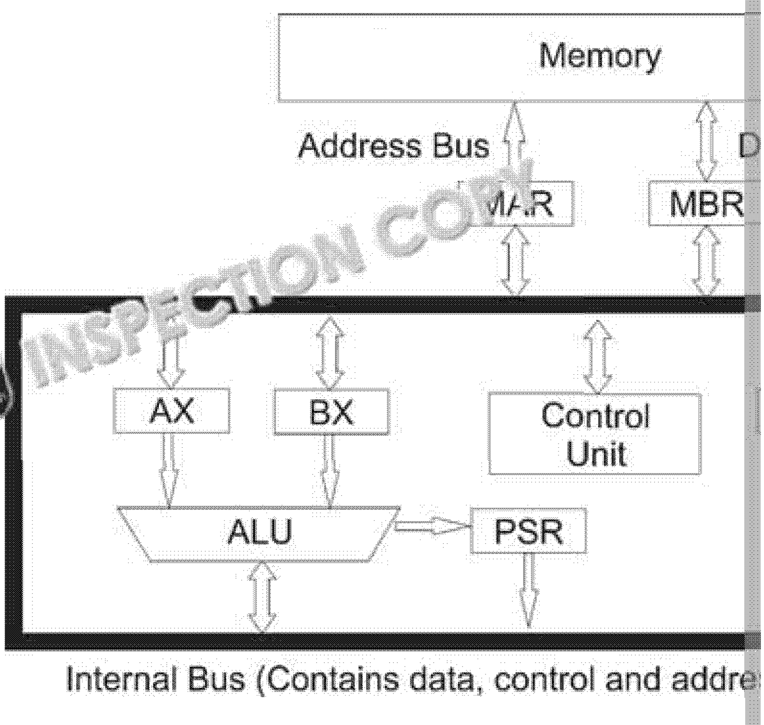
7.3 STRUCTURE AND ROLE OF THE PROCESSOR AND ITS

THE PROCESSOR AND ITS COMPONENTS

The processor is a CPU (central processing unit) on a chip and provides the ce The specification of processors changes fast because there is a constant dema

The processor can be considered to be similar to a travel system in which da location via buses. It is constantly changing the paths to the different eleme is performed by the control unit and regulated by the system clock.

Inside a processor there are certain elements which are required in order to The structure and role of the processor are covered in more detail on the fol



Arithmetic logic unit (ALU)

Arithmetic logic unit carries out arithmetic operations such as addition, multi It can also make logical comparisons between items of data; for example, it greater than another. Such logical operations can also be performed on non- first Intel processor to have two ALUs so it could manipulate two sets of nu of the ALU is held in a register called the accumulator.

Control unit (CU)

The *control unit* governs the operation of all hardware, including input and o this by fetching, interpreting and executing each instruction in turn, in an a fetch-execute cycle is described in detail below.

Clock

The *clock* is the part of the processor that regulates all of the actions that tak regular pulse of high voltage followed by low voltage (voltage is the electric Each high-low transition is known as a cycle, and each cycle implements

Program counter (PC)

The *program counter*, sometimes called the *accumulator*, holds the address of When a sequence of instructions is to be executed the PC is automatically inc instruction. Depending on the length of the current instruction, 1, 2 or 3 has the current instruction is a jump, in which case the destination address is use

General-purpose registers (AX, BX CX and DX)

The other *general-purpose registers* are used for performing general arithmeti no pre-defined role by the chip designer and as a result can be manipulated perform the role of any other registers.

Memory buffer register (MBR)

Values about to be added or subtracted can be copied, via the *memory buffer memory data register (MDR)*, into the accumulators. The arithmetic result can copied from there into a main memory location. All communications betwee place through the MBR.

Memory address register (MAR) and current instruction register

In order to fetch an instruction from memory the CPU places the address of *address register* and then carries out a memory read; the instruction is then c into the *current instruction register*. Similarly, an instruction which itself requi word causes the address of the data word to be placed into the MAR. The ex results in the copying of the addressed data word into the MBR, from where it The MBR acts as the point of transfer for both data and instructions passing, main memory and the CPU.

Status or flag register (SR or FR)

The *status or flag register*, also known as the *processor status register (PSR)*, co based on the result of an instruction. These flags determine the operation of result of the previous action. It also handles interrupts to signal to the cont

THE FETCH–EXECUTE CYCLE AND THE ROLE OF REGIS

The control unit (CU) in the processor manages the execution of instructions sequence, decodes and synchronises it before executing by sending control s computer. This is known as the *fetch–execute cycle* (or *fetch–decode–execute* in more detail below:

Fetch phase

Common to all instructions:

1. The contents of the PC are copied into the MAR. The MAR now contai instruction and a memory read is initiated to copy the instruction wor
2. The PC is incremented and now contains the address of the next inst
3. The instruction word is then copied from the MBR (MDR) into the CIR parts: the operation code (*opcode*) and *operand*. The opcode is the ins data to perform the operation on. *This is described further on the foll*

Execute phase

The action taken is unique to the instruction:

1. The opcode instruction in the CIR is decoded to a simple operation s *(see p.10)* which affects the path the data will then follow.
2. The instruction in the CIR is executed; if the result needs to be comm held in the MAR.
3. Unless the instruction is a STOP instruction, then the cycle is repeate

Example of the fetch–decode–execute cycle

Imagine a very simple computer that could have a program that consists of a look like this:

```
LDA 2 (load 2 into the ALU)
ADD 1 (add 1 to 2 in the ALU – result in the accumulator
STA result (store the result in memory)
HLT
```

In machine code these could be translated into instructions. Imagine a 2-bit 11 = STA store in memory location and 00 = HLT; the last three bits hold the

```
01 010 LDA 2
10 001 ADD 1
11 101 STA (memory location 101 or 5)
00 000 HLT
```

This program is loaded into memory and stored in the locations as shown:

Location	Contents	Comment
001	01 010	Load 2 into ALU
010	10 001	Add 1 to the ALU result in the accumulator
011	11 101	Store accumulator in location 101 (memory l
100	00 000	HLT – stop the processor
101	Empty	Where the result will be stored

The fetch–decode–execute cycle would do the following:

1. The PC is set to 001 which is sent to the MAR – this goes to the me 010) into the MBR. The PC is incremented for the next instruction; th
2. The contents of the MBR (01 010) are transferred to the CIR; the CIR (01) and the operand (010). The instruction is then executed and the the ALU. Instruction complete.
3. The next instruction from the PC (010) is transferred to the MAR – thi the instruction (10 001) into the MBR. The PC is incremented for the n
4. The contents of the MBR (10 001) are transferred to the CIR; the CIR b and the operand (001). The instruction is then executed and the opera and stored in the accumulator which now holds a value of (011) or 3, i
5. The next instruction from the PC (011) is transferred to the MAR – thi the instruction (11 101) into the MBR. The PC is incremented for the n
6. The contents of the MBR (11 101) are transferred to the CIR; the CIR [illegible] and the operand (101). The instruction is then executed. In this c [illegible]1 and the contents of the accumulator are copied into that loca memory location 101. Instruction complete.
7. The next instruction from the PC (100) is transferred to the MAR – thi the instruction (00 000) into the MBR. The PC is incremented for the n
8. The contents of the MBR (00 000) are transferred to the CIR; the CIR brea the operand (00). The instruction is then executed. In this case it is HLT c

This is an example of a very small computer with a very limited instruction s larger instruction set as described in the next section.

THE PROCESSOR INSTRUCTION SET

These are the set of instructions that a processor can apply to a flow of data All processors will have a different instruction set as there is not a standard This gives manufacturers the capability to push their hardware further and f

Modern instruction sets consist of roughly 100–250 instructions and cover e expects will be required by the hardware, therefore if an instruction is read t set then an error flag is produced and the computer halts that program tree.

There are two approaches used in instruction sets:

- *Reduced Instruction Set Computer (RISC)*
- *Complex Instruction Set Computer (CISC)*

RISC vs CISC

RISC provides a fairly [illegible] basic instruction set where in every operatio stored from/to memory, whereas in CISC the instruction set is much more va entire sequence of operations across multiple clock cycles.

After the conception of the instruction sets, RISC tended to run faster becaus simplicity but it slowly became less commonplace because of how CISC mac leader for CISC components is currently Intel and the reason why CISC becar because of how they managed to implement RISC principles into CISC archit

As the name implies, the instruction set is composed of instructions of a give two parts: the *opcodes* and the *operand*. The *opcodes* (an abbreviation of *oper* machine language instructions that tells the processor what operation is to b data that is stored in the opcodes address.

In a simple model, 4 bits might be assigned to the opcodes (3 bits for the ba be done and 1 for the addressing mode) and 4 bits will be allocated to conta size of each instruction is 2 bytes. This means that a total of 16 instructions

For example, this might be the opcodes for addition using direct addressing.

ADDRESSING MODES

Direct addressing

Direct addressing specifies the actual or effective memory address containin address field of the instruction word. This addressing mode allows for the co quickly and efficiently; it is also the most intuitive of the addressing modes b think about tasks, but it isn't always the best choice for a computer because processed and reallocated a memory address. The addressed memory locatio obtain the operand.

An example of direct addressing is that if the command was STA 5, the 5 wo data. In the example of the simple computer in the fetch–decode–execute c 11 101 (= STA 5) is using direct memory addressing.

Immediate addressing

Immediate addressing is where the data is actually part of the command; for e will load the value 2 into the ALU register, not the contents of memory locat

MACHINE-CODE/ASSEMBLY-LANGUAGE OPERATIONS

The basic machine-code operators

The syllabus states that you should be able to understand and write progra including immediate and direct addressing. There are various free versions o such as Little Man Computer which will allow you to practise these skills as

Three basic machine-code operations are LOAD, ADD and STORE. These are the memory to be moved about and added up.

LOAD and STORE are data transfer functions that make it possible to move n and then store them in a relevant memory slot. ADD is an example of an arit value from the memory and adds another number to it. Other arithmetic ope and SUBTRACT.

A typical machine-code segment might be LOAD A, [15] which would load th memory at location 15. Another would be ADD A, [16] which would add the v to the accumulator. So a simple program might consist of loading three num storing the result in the computer's memory.

This program would be as follows:

```
LOAD A, [15]
ADD A, [16]
ADD A, [17]
STORE A, [18]
Halt
```

In order for this to work, the instructions for this program (i.e. the code writt in locations other than 15, 16, 17 and 18, and then the values from locations the result saved in memory location 18. If the instructions were not saved in program would overwrite the instructions and hence destroy itself.

Operation	Example	Explana
Load	Load A, [x]	Loads the contents of a variable, or a m accumulator so that it can be used in a usually found inside square brackets. X
Add	Add A, [4]	Performs an addition operation on the co write it as a direct value, the value of a m the keyboard or device.
Subtract	Subtr A, [4]	Similar to the Add command, performs the content of the accumulator. You ca value of a memory address, or user inpu
Store	Store A, [4B]	Stores the contents of the accumulator memory location. The memory location to work correctly.
Bitwise logic	OR A,	The AND, OR, NOT *and* XOR commands *operation* A, B. These are used in the ex them in other programming languages.
Compare	Comp A, [x >1]	Compares the contents of the accumula memory location to produce a True or F used to allow branching and determinis
Shift Left	SHL A,3	Multiplies a number by powers of 2 (2, binary digits left and inserting 0 into th If we were to shift left 101 (5) by 2 to g
Shift Right	SHR A,2	Similar to Shift Left except this is *dividi* leftmost column is filled with zeros. Th signed numbers. Least significant bits a 101 (5) by 1 to get 10 (2 or 5 DIV 2)
Unconditional Branch (goto)	BRA Jump to the line with that label	Go to a particular line in the program. T often is used with an unconditional bra condition.
Conditional Branch	BRZ Jump to the line with that label IF the accumulator is currently zero BRP Jump to the line with that label IF the accumulator is currently zero or positive [illegible] Jump to the line with that label IF the accumulator is currently not zero	Go to a particular line in the program b (see compare). Most common branches ALU and the contents of the accumulat the value being zero, positive or not ze
Halt	Halt	Tells the computer to halt all processes end the program safely and properly to to using System.Exit in VB.NET/C-based

ARM assembly language

The AQA specification states that the code written in exams will be of the fo
the commands will be described in the paper.

ARM technology uses RISC technology and is used in devices such as the Raspberry Pi and mobile phones. The ARM registers and instruction sets are different from those for ×86 Intel processors as memory and instructions are stored and referenced as separate locations.

The basic differences are described below and include additional command sets. ARM technology also has more registers to be used within the processor, and it is recommended that these are used rather than linking to memory as this simplifies the process. Although registers can be copied to memory (using direct or indirect memory addresses), this can significantly slov

For most examples, utilising the registers r4 to r9 (as variables / general pu

Although commands in Little Man Computer and ARM may look similar, their
commands allow the use of a different register to store a result from the regis
example, Little Man Computer instruction for ADD is ADD num1 with the re
num1 to the contents of the accumulator and store the result in the accumula
ADD is ADD r0,r4,r5 which would add the contents of r5 to the contents

Example showing the difference between Little Man Computer a

If we were to write a program to take in two numbers and output the result, i
inside an emulator whereas the Raspberry Pi (RPi) uses its own registers, so t
complex due to setting up the messages and output but the fundamental ass
given also utilises the routines bl and printf to output which allows the outpu

LMC code to add two numbers and output the result

```
INP
STA num1
INP
STA num2
ADD num1
OUT
```

ARM code to add two numbers and output the result for a RPi ARM

(note: included statements for input)

```
@ set up data reference and output statements
.data
.align 2
scan_format:
.asciz "%d"
.align 2
out_format:
.asciz "Answer is : %d\n"
.align 2
instr1:
.asciz "Enter first integer."
instr2:
.asciz "Enter second integer."
.align  2
num1:
.word 0
num2:
.word 0
.text
.global main
main:
    push {ip, lr}           @ used with pop at end of main, allow
                            of the program
    ldr r0, =instr1         @ load the instruction into r0
    bl puts                 @ output to screen
    ldr r1, =num1           @ sets up num1 for input
    ldr r0, =scan_format
    bl scanf                @ calls routine to input num1
    ldr r0, =instr2         @ load the instruction into r0
    bl puts
    ldr r1, =num2           @ sets up num2 for input
    ldr r0, =scan_format
    bl scanf                @ calls routine to input num2
    ldr r6, =num1           @ load address of num1 into r6 needed
    ldr r4, [r6]            @ load value of num1 into r4 nb other
                            allow ldr r4,num1
    ldr r6, =num2
    ldr r5, [r6]
    add r1, r4, r5          @ adds contents of r5 to r4 and store
    ldr r0, =out_format     @ sets output for "answer is: " follow
    bl printf               @ output the solution
    pop {ip, lr}            @ used with push at start of main, al
```

Commands similar to Little Man Computer

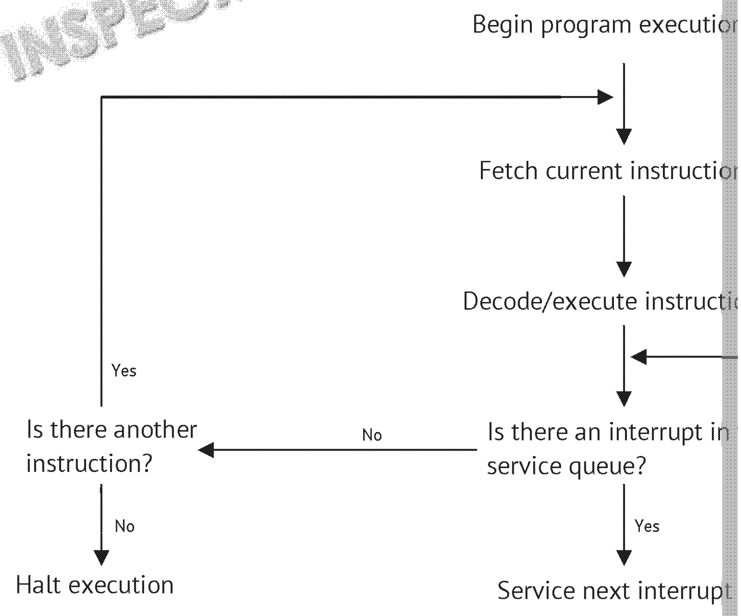
Below is a table of the ARM type commands which are similar to Little Man

Mnemonic	Example	Function
ADD	ADD r0,r4,r5	Addition adds contents to r5 to r4 and store
SUB	SUB r0,r4,r5	Subtraction subtracts r5 from r4 and stores
RSB	RSB r4,r4,#300	Reverse subtraction subtracts r4 from secon 300) and stores it in this case back to r4
MUL	MUL r0,r4,r5	Multiplication multiplies r4 by r5 and stores answer in a register used in the calculation r4,r4,r5 will cause an error
AND	AND r0,r4,r5	Bitwise AND between r4 and r5 returned in
ORR	ORR r0,r4,r5	Bitwise OR between r4 and r5 returned in r
EOR	EOR r0,r4,r5	Bitwise exclusive OR between r4 and r5 ret
MVN	MVN r0, r4	Bitwise NOT stored in r0
TST	TST r4,r5	Test (performs bitwise AND sets flags accor
CMP	CMP r4,r5	Compare r4 and r5 (actually uses SUB) but e values and sets flags on whether they are e
B	B label	Unconditional jump/branch to a label
BEQ	BEQ label r4,r5	Conditional jump/branch to a label based o
MOV	MOV r0,r4	Move contents of r4 to r0
LDR STR	ldr r3, =sum str r0, [r3]	Load and store work together, so in this exa The value in r0 is then stored in =sum using
BL	BL printf	Call a subroutine then return; in this exampl routine

INTERRUPTS

Interrupts are the computer's way of diverting away from the sequential natu Without interrupts a computer would continue to finish all the instructions f to see whether there are any interrupts. When an internal error occurs withir sent from the device to notify the processor. Interrupts are stored in a *priorit* In modern-day general-purpose computers the interrupt queue is checked af executed. If the queue is empty then the next instruction in the sequence is is something in the queue then the processor runs a program called the *inter*

Each type of interrupt has its own interrupt service routine, so the ISR for an to the ISR for a paper jam in a printer. However, while the ISR is at work serv solve the problem, the volatile environment of what was running before the stored so that when control is passed back to the main program the user car



Types of interrupt

As mentioned earlier, not all interrupts are the same; each interrupt needs it understand what needs to happen when that interrupt is present in the inter types of interrupt that you should be familiar with just by general use of a co

I/O interrupts

Input/output interrupts occur when the computer is performing a data transf been completed or there has been an error during transmission. An example of a file from main memory to a USB memory stick and during the transfer y without warning. The computer generates an interrupt that prevents the furt the destination address is no longer reachable.

Timer interrupts

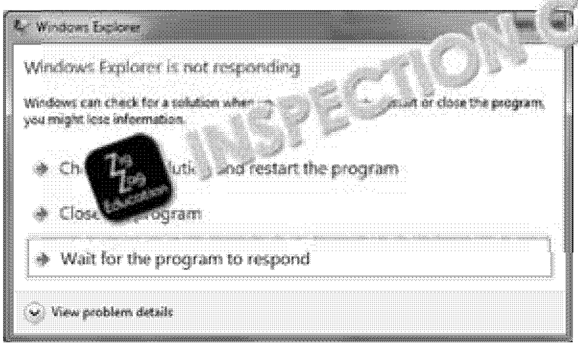
There is a concept in computing and technology called *time-critical* data. Thi critical files that take priority over the current instruction being performed. F be seen if you stream a movie over a service such as Netflix – if the data wa audio and the video would be out of sync because the video portion of the d transmit and process than the audio. Another example of this can be seen in loading a YouTube video; the audio would play first while the video would b blank or only the first frame would be displayed until the video buffer caugh

Hardware interrupts

These are the most common type of interrupt that all computer users will ha interrupts mark critical errors in the hardware of the computer itself that affe performed by the CPU. For example, if there is a sudden loss of power, the C much data as possible and close the OS down as safely as possible to preven OS is rebooted.

Program Interrupts

These interrupts are produced by the software of a computer system to tell t or an exception has been met. These frequently occur in games but will be h disrupt the gameplay experience.



Windows Explorer

Windows Explorer is not responding

Windows can check for a solution when [illegible] restart or close the program, you might lose information.

Check for a solution and restart the program
Close the program
Wait for the program to respond
View problem details

The most common cause o produced is memory access access a memory location t

When a program enters a st unresponsive, which is whe Windows operating system

FACTORS AFFECTING PROCESSOR PERFORMANCE

Multiple cores

A processor that supports multiple *cores* is essentially a processor with multiple identical processors integrated into a single chip. The increase in cores allows for a greater throughput of data. It allows the processor to divide the labour of performing a task between all the cores.

If the software is written for multiple cores, the processor can split the instructions into a number of simpler instructions in a process called 'threading'. If the software is not coded for use of multiple cores then the extra cores aren't used for that process and are assigned for performing other background tasks.

Cache memory

The *cache* memory for a processor is a small amount of *high*-performance RA It enables the CPU to access repeatedly accessed data directly from its own boar requesting it from system memory. Cache is critical to applications such as v applications but is less critical for general applications such as emails and v

Words

A *word* is a group of bits that can be addressed, transferred and manipulated processor. The size of a word, the word length, is determined by the width o the computer and is usually a multiple of a byte, thus consisting of 16, 24, 3 word size may mean that a computer can work faster than a computer with a efficient when that data does not need to use the full word length.

Clock speed

Every computer contains an internal clock; this clock is used to regulate the are carried out. The central processing unit in the computer needs to have a carry out an instruction; this means that the faster a computer's clock 'ticks', carried out per second. The speed of the clock is measured in megahertz (MH

Bus width

The width of the bus is another factor that relates to the performance of the the amount of data that the central processing unit can transmit at a time to that the memory or the input and output devices. For example, a 16-bit bus Consider the bus in terms of passengers; if each passenger was carrying som destination you would want to fit as many on the bus as possible to save doi

Did you know?!

The ultimate restriction on CPU performance comes down to how temperature CPU you know that it has a large metal hood over the chip – this is actually exchanger or heat sink designed to carry heat away from the delicate compone motherboard will reduce its clock speed, drop threads and reduce the number c called thermal throttling and is the computer's way of cooling the CPU down. If simply shut itself off.

Overclocking is the process of increasing the clock speed of the processor throu speed, the faster the instructions can perform. However, as above, the tempera critical and so 'overclockers' may spend more money on different methods of ke water cooling, to prevent throttling.

Questions: Structure and the Role of the Processor

1 Which register in the processor must all input data pass through? (1 m

2 Describe all the steps required in the fetch and execute steps of the fetc

3 Your computer is sat idle at the desktop; you are connected to the Int open and no programs are running.

- a) Why do your CPU and memory usage never reach 0%? (2 marks)
- b) You decide to load an application; are interrupts used? (1 mark)

4 Are each of the following statements true or false:

- a) The entirety of a frequently used program is stored to the CPU's c
- b) The word size is controlled by the CPU. (1 mark)
- c) All software will automatically use multiple cores. (1 mark)
- d) Generally, a larger bus width will improve performance. (1 mark)

7.4 EXTERNAL HARDWARE DEVICES

INPUT AND OUTPUT DEVICES

Barcode reader

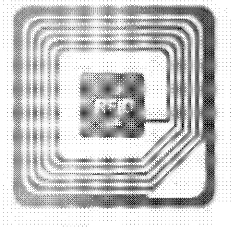
Barcodes are used everywhere nowadays. They are generally used to track items, such as products in a supermarket, or books in a library. Although there are many different specific systems for encoding the information in a barcode, they all work in a similar manner. In the EAN and UPC barcode systems, the most popular for food in the UK and USA, two bars and two spaces represent each character in the code. The actual system used is quite complex because the scanners have to cope with the barcode being folded, c odd or varying speed. The two types of reader commonly found connected t across the barcode (wands) and scanners that read the entire barcode at once.

How does a barcode reader work?

A barcode is made up of a series of bars which in combinations form the diff each other at a set distance and so some multiple bars appear as thicker line to pass across the barcode. In most cases this is a laser light which is directe The light will either be absorbed by the black bars or reflected by the white known as a photo diode) produces an electrical signal based on the amount interpreted as a 1 or 0 and fed into the computer. The combinations of 1s an bar code number. The number is then used as an identifier on a computer da about the item that has been scanned.

Radio frequency identification (RFID)

Radio frequency identification is used in a similar way to barcodes in that th by a computer database to obtain the details. However, the technology uses



that sends out an identifying signal that can be interpr passports, animal identifications and libraries among m semi-passive or active; passive and semi-passive rely o the power to send a response, whereas active devices h broadcast a signal continuously. Semi-passive devices c facilitate functions with the device (such as data storag

Digital camera

Digital cameras and digital video cameras take pictures and store them digit rather than on analogue or photographic film. The storage device is unimpor from a photography point of view, but has space (MB) implications and size (portable) implications. Most cameras these days use flash memory cards to s images on, with the top-of-the-range cameras having many gigabytes of me meaning they can store thousands of high-resolution images.

In addition, since the images are stored digitally, the camera can manipulate directly, adding effects such as sepia tone to the image, and easily and quick editing and inclusion in a word-processing document. Of course, the real adva is that you do not have to wait for the film to be developed after you take th

How do digital cameras work?

The lens on a digital camera magnifies and focuses the light in the same way than focusing the light onto a conventional film (which then creates a chemic cameras focus the light onto a device known as a charge-coupled device (CCD array of transistors which create electrical currents in proportion to the intens is used for each pixel and for each primary colour. The image is then stored o

Laser printer

These give a very high-quality print. For an example of running costs, the HP
of paper from one new cartridge (which currently costs about £30) – a cost o
Colour laser printers are relatively quick and high quality but can be expensi

How do laser printers work?

1. The computer's printer driver sends information to the printer about
2. The printer uses a special wire called a 'corona wire' which statically even across the whole area.
3. The printer's processor uses the received information to shine a laser the printer drum. The laser does not actually move; the beam is direc The drum becomes negatively charged in those places where the las positively charged elsewhere.
4. The toner (a special ink powder) is positively charged with another c
5. The drum rotates and the positively charged toner jumps onto the place charged. Equally, the toner is repelled by the positive charge, making th
6. Meanwhile, the paper is rolled through the paper train and given a stron The paper is then fed very close to the drum and the toner particles jum
7. Finally, the paper with its particles of toner are passed through two h paper. This is known as fusing.

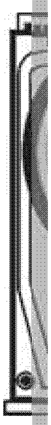
SECONDARY STORAGE DEVICES

What is secondary storage?

The most common peripheral is secondary storage, which in this context mea stores information permanently and is not immediately accessible by the pro hard disk as being the main memory of the computer, in fact the computer ta programs stored on the hard disk than those stored in RAM. The computer co from the hard disk to RAM, then writes back to the hard disk where required. very large file or lots of programs running at once) it will start using the hard computer appears to slow down.

Hard disk drives (HDDs)

A HDD is a storage medium that stores large amounts of data. They first bega 1970s as a method of quickly storing data; they were faster than the tape sys expensive. For this reason, they were typically used to store program informa how RAM is used nowadays. Over the years, however, their cost dropped and became a standard piece of computing equipment.



Nowadays, they are generally used for storing programs, documents and anything that needs to be stored. External hard drives are becoming increasingly popular as the amount of information that a typical computer uses is growing rapidly. External hard drives connect through USB and provide additional storage when the internal hard disks become full.

Information is held in blocks formed by tracks and sectors (see diagram). Each block of information contains the sane amount; this means that information is more compact closer to the centre of the disk.

Rotation speed

A hard disk consists of one or more disks which spin very rapidly in an evacu is such that the reading heads, which are very small indeed, are suspended a *effect*, typically about two millionths of a centimetre for a modern hard drive to prevent specks of dust, or other airborne detritus, from damaging the disk speeds of rotation (typically 7200 rpm).

Capacity

The early hard drives had a capacity of about 5 MB, but nowadays it is comm specifications of above 1 TB of hard drive space. At the same time, the cost h hence hard drives are more affordable than ever. By way of example, a 10 MB might have cost £1,500 – a cost per megabyte of about £150. By contrast, in purchased for as little as £50, giving a cost per megabyte of about 0.005p per

Optical disk

CDs and DVDs are metal discs embedded in a plastic protective housing. Eac the process of creating the disk, and placing the data on it.

Although you can get re-writable (RW) versions, most optical disks in use are *(Write Once, Read Many)* media; this refers to the fact that once they have bee there is no way to change the data on them. For this reason, and the fact be facto standard for the distrubition of software and other media such as musi

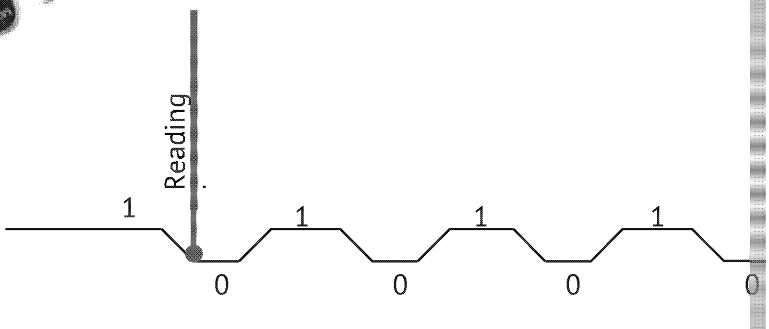
More recently, Blu-Ray discs are becoming more widespread, mainly because storage capacity (up to 50 GB) makes them a popular choice for storing data- content such as HD films and video games.

Optical spiral

Data on the disk is arranged in a very different way from that on a hard drive arranged in concentric tracks and sectors. For example on a CD-ROM, data is track from the inside of the CD to the outside. The packing density is uniform order to sustain transfer rates at the inside of the disc, with reference to the inside. Another difference between otpical disks and hard drives is in the wa hard drives use a magnetic system, whereas CD-ROMs use a purely optical s

Binary pits

Originally, during the mastering process, a very powerful laser used to burns disc. When the disc is read back, the reading laser sees these pits as one sort part of the track as the other. In this way, we have the storage of 0s and 1s th storage. When the user reads the disk, a high intensity laser is shone at the s pits and peaks produce a difference in how they scatter the light; this is pick as the binary digits. Music CDs and CD-R (for distribution of software) may st method as this makes the disk read-only.



Modern CD-RW/DVD-RW use a rewritable technology in the form of dye wh laser beam. The dye has a special property whereby if it is heated to a certa cool it goes opaque, whereas if it is heated to a hotter temperature and the transparent. This feature allows the disk to be reused. The principle of the r with the transparent layer acting like a pit.

Solid-state drive (SSD)

Flash drives are everywhere in modern technology; something that was onc now been reduced to the point where two or more modules can fit inside a drives are a type of memory known as Not-And (NAND) flash memory that u reading and writing of data. Modern operating systems find it easier to tran as *pages*; a page is a fixed block of memory that is used to facilitate the efficient transfer of data. The controller obeys two certain constraints. Firstly, when you combine many pages you form a block – but a block cannot overwrite other individual pages. Secondly, before you can write to an allocated memory location you must first erase the page that has been allocated to that location, but the underlying technology requires that the entire block is erased if the page is linked to a block. The result is a form of memory that has very low latency (response time) and transfer speeds that vastly outpace that of the magnetic or optical format.

These benefits do come at a price, however. The way in which the data is ha damaging for the memory components of the SSD and it can lead to data er have an expected lifetime that is based on the number of read/write cycles decreased the more the data on the drive is accessed. Recent studies have s longer than expected but the cost still remains high compared to magnetic that most who employ a solid-state drive only store the OS and frequently a time, while storing all other personal files to a magnetic drive and keeping

Speeds of access and suitability

The suitability of which medium you use depends entirely on your requirem change in the way that data is stored even in the last 10 years, and there ar made every day so what is right now may change in the near future. It is eas drives (HDDs) have been employed all over the world in modern computers large storage capacity and have a relatively fast read and write speed for pe instantly. Their primary role is to facilitate the storage of your personal files the OS to interact with the user.

Solid-state drives, while having fast transfer speeds, are expensive to produ just beginning to match the capacity of HDDs but with a much higher price t LSI announced it had created a 4 terabyte solid-state drive that would retail same time a 4 terabyte HDD would cost around £100.

Optical disks are slowly going the way of the floppy disk; they're becoming ob manufacturers are starting to produce computers that don't have optical bays access is slowly becoming digital; everything can just as easily be placed

There is a hierarchy of speeds for I/O devices; in descending order they are:

Device	Read/write speed
Solid-state	550 MB/s
Hard disk	300 MB/s
Optical disk	7,200 KB/s

8. Consequences of Uses of Computi

The power of modern computers and the volume of data handling has led to a whole new considerations. This area will continue to grow as technology continues to expand and in discusses moral consequences of computing, and the professional computer scientist mus computing gives with responsibility.

This section covers:

8.1 INDIVIDUAL, SOCIAL, LEGAL AND CULTURAL ISSUES

In the mid 1940s the smallest computer occupied 1,800 square feet, used ov version of a transistor) and weighed nearly 50 tonnes. It was capable of perf consumed an [illegible] about 150,000 watts of power. Modern computers, wit in technology, are now able to fit neatly under a desk and even into the palm 'computer' can now be applied to anything with an embedded microchip in it

As technological breakthroughs are made it becomes more common place to appliance found in a home or workplace. With great technological power co few people take into account when using computers. These consequences c

- *Individual* consequences cover the *moral* standings of computer use. physiological and physical effects of computer use as well as how yo
- *Social* consequences cover the *ethical* standings of computer use. The individuals, governing their principles and behaviour.
- *Legal* consequences are the *written* laws that govern computer use a fraud and the challenges facing legislators.
- *Cultural issues* can cover a large variety of meanings from traditions and b

Note: you are not required to know the specific acts and laws governing social a context it would be a good idea to look into these acts to build on your understa matters will better your awareness of how you yourself use computers.

INDIVIDUAL CONSEQUENCES

Morality

Morality is the notion of what is right and wrong or good and evil in society. larger and more widespread, there have been issues that have needed to be of individuals when using their computer. As most computer users are not as algorithms in software, the majority of the power lies with computer scientis

However, those who develop software also have the responsibility to ensure maliciously.

Did you know?!

When you download social media apps on to a smartphone there is a small ter agree to before the app will download. Many social media state the requiremen

When you agree to these you often unknowingly sign in agreement to the app messages, call history and, most worryingly, your camera and microphone. In fa Cyanogen mod now comes with a built-in data access protector to prevent apps remain private.

Health and Safety

Prolonged use of computers is a relatively new occurrence that has been stu has uncovered several health risks. Stress is common in employees where co needed due to the fast-paced nature of communication. There is a wide varie disorders that are linked to computer use. These pressures can make you mo issues, potentially making it dangerous in the long term.

When working with computers, workers are at risk from health issues that mi risk is *RSI (repetitive strain injury),* caused by long periods of typing or repetiti RSI is an umbrella term for a variety of injuries which can be traced back to i environments. Specific terms include tenosynovitis, tendonitis, epicondylitis, tunnel syndrome, and thoracic outlet syndrome. In 1993 a British judge dism concept of RSI had 'no place in medical textbooks'. However, also in 1993, sp use of VDUs in the workplace was introduced, in the form of the *Display Scre*

RSI is now a recognised problem, to the manufacturers put health warnings on th more of a problem than traditional typew diversity of movement. A traditional typi of paper, and push the carriage. These a computer, and it is perfectly possible fo more without a break. In addition, they w harder on a typewriter than on a keyboa therefore not as likely to cause RSI as ke

Other health problems include *eye strain*, caused by staring at a screen for a caused by sitting without moving in an uncomfortable or badly designed cha with computers can be damaging to the individual's health. The directive on with computers states that:

- An employee's work must be planned so that work involving the pro periodically interrupted by other activities.
- The workstations must be made of separate components (chair, keyb employees can adjust the station to suit them. Monitors must meet s glare screens. An adjustable chair should be provided.
- Employers must analyse the risks of physical and mental stress cause appropriate action to remedy the situation, including giving employe keyboard work.

You can now buy both software and hardware to help with these problems. Hardware includes ergonomic mice and keyboards, plus accessories such as foot rests and document holders. Speech recognition and other software can help to automate tasks and therefore reduce the amount of typing needed.

As individuals we have a responsibility to ensure that the information, software and devices we use are not used maliciously, and to abide by the laws. When using computers we must consider how our actions will affect others.

For example, consider the possible implication of sharing personal or financi someone be able to access this information and what could they gain from d

The Ten Commandments of Computer Ethics

These were created in 1992 by the Computer Ethics Institute and form a gui and ICT organisations. They are fairly simplistic and follow common sense o good base to consider when developing software or using computers.

1. Do not use a computer to harm others.
2. Do not interfere with other people's computer work.
3. Do not snoop around in other people's computer files.
4. Do not use a computer to steal.
5. Do not use a computer to bear false witness.
6. Do not copy or use proprietary software for which you have not paid.
7. Do not use other people's computer resources without authorisation
8. Do not appropriate other people's intellectual output.
9. Do think about the social consequences of the program you are writi
10. Do always use a computer in ways that ensure consideration and res

Malicious Software

It is illegal to write and/or deliberately distribute malicious software such a worms and viruses. Most computer professionals find the idea of producing s software morally wrong. Programs such as viruses can cause serious problem computer systems and are also 'out of your control'. The use of such program severe consequences; for example, if a hospital computer system was attack malicious software, information and treatments could be seriously affected.

SOCIAL CONSEQUENCES

The social implications of computer use are ones that few consider. There is Internet can create what is known as a 'two-tier' society, comprised of those and fast Internet, known as *information rich*, and those who cannot, who are be seen at an increasing rate in situations such as education where pupils a digital versions of their work, or work that requires research that would othe traditional methods of research.

With the widespread use of the Internet, particularly among the younger ge with social peers has become much easier. This has meant that social relatio have very different geographical locations, so now people can socialise in g strongly by interest or opinion rather than being limited by where they live.

The Internet and the reduction in the cost of electronics have also had an i experience and discover entertainment material. These days, websites such signed to a music label, as musicians are able to publish their work free of c can publish their works for free to a large audience. Recording equipment h scale recording that at one time would have only been possible in expensiv performed in bedrooms and garages.

The amount of personal information that is accessible to businesses and col increased exponentially. These days, large businesses such as supermarkets habits and preferences in order to target marketing and even shop layout to more money. Even the shops that we use today are only possible because o stock ordering and POS (point-of-sale) terminals allows such large-scale sh locally run and sourced shops out of business.

The social make-up of cities has begun to change because large industriali during the Industrial Revolution and beyond. People have been looking for a the countryside ever since. Computers have helped to make this possible, al wealthier within society. This has left a disproportionately large number of t

These aforementioned developments in digital technologies have drasticall made, and the way information is accessed and preserved has enabled the c large groups of people with very little effort. By monitoring data flows you c personal data without the person's knowledge. Once this data has been gath make money by selling it, publishing it or using it maliciously.

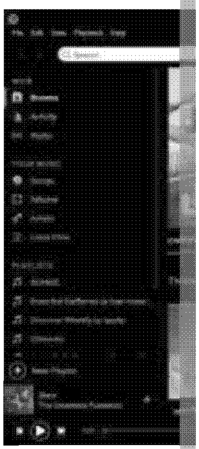
LEGAL CONSEQUENCES

Lawful computer use is no different to living by the laws of a country; there illegal. With the growth of the computer industry and integration into devices an en There are many parallels that can be drawn from laws that are found in com that are put in place to govern computer use and many of these are to prote right to privacy and protection of data.

One industry that has been seriously affected by copyright issues is the music are copyrighted; it is illegal to broadcast or receive copies of them. However, s and Napster made the illegal sharing of such files very straightforward and ca

This is because there is so much Internet traffic that monitoring is an almost i very small proportion of Internet pirates are caught. This means that in order t methods have to be employed. These methods hope to limit access, ability to once purchased. These have always been a concern, but since the widespread degrade each time they are copied and their distribution is fast and straightfo

Modern approaches include:



- Spotify (shown on the right) allows users to stream music to their devices for a monthly fee. It also offers a free service, which has limitations such as lower streaming quality and the inclusion of advertisements.
- Systems such as BBC iPlayer use a Microsoft Digital Rights Management (DRM) system that makes sure that BBC programs can be downloaded and stored for a certain length of time, but they then expire, hence making it impossible to copy and broadcast them indefinitely.
- Windows media player formats (WMA) place a copy restriction on files s they have been imported from CD; however, this technique is easily b

Digital rights management helps make sure that artists, etc. are paid for the copyright that exists on that work. However, it is argued that the technolog providers; should they change player or the company go bust, then the co it makes competition between providers difficult and stops a collaborative c since the increased use of digital media.

With the nature of data technology what it is, the legislation governing its u the biggest challenges facing legislators is the act of copyright infringemen of copyright was still difficult, but digital technology has made it significant price of copying and distributing has fallen to zero. Music and software are t change as people cannot see why they should continue to pay for these serv

Identity theft / fraud

In 2013 alone there were 13.1 million reported cases of identity theft, and it
use and will be for the foreseeable future. Techniques such as 'phishing' and
attacks seen on the Internet when someone is trying to gather information a
attempt to gather personal information by masquerading as a reliable source
other hand, pharming is a social engineering method of diverting data flows

Cybercrimes leave no *physical* evidence, and a well-performed data-gathering cyber-attack will leave little trace that an attack ever happened. To add to this, the Internet allows for near perfect anonymity for those that know how to protect themselves. Given that there is no police force dedicated to cyber-crime prevention and that an attack can be launched from any computer, it is next to impossible to intervene while an attack is happening. Once the attack is over, very little can be found out about it because of the lack of evidence; a log of an IP address can be checked but with the use of virtual private networks the IP will usually be a dead end.

CULTURAL ISSUES

When computers became more prevalent it became seen as a basic right to ha
revolutionised teaching. Traditionally, teachers were the only source of know
textbooks, but computers have slowly replaced teachers as the main source o
challenge: what about those who are in poverty or are less financially secure
been bought by most schools to provide for those students that are less fortu
to buy and maintain, and are often comprised of outdated hardware to drastic

It is the responsibility of the owner of a website to ensure that the data foun
high quality; in real-world applications (i.e. a newspaper) this is easy to gove
complain, and it is governed by various acts that reduce libel printings and s
computer/Internet use; information is largely unadulterated, but there are nu
dedicated to personal views which are promoted as fact that can be seen as
propaganda. It is for this reason that the *quality of information* needs to be ta
from a reliable source.

It can also be said that cultural issues take into consideration how data can a
consider a research group that wanted to document the location of every sm
of Africa; how will that information be used and how will that use affect the
villages? Will there be any safeguards put in to prevent people using the info

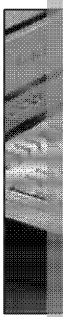
Speech synthesisers and voice recognition

Such programs as word processors, spreadsheets and databases can be used
display. They can combine voice recognition software, allowing the user to is
with a speech synthesiser to provide a response to the user. Currently under
allow blind and visually impaired people to surf the Internet.

A synthesiser must be developed with a number of factors in mind, includin

- Compatibility with the desired software
- The quality of the synthesised voice
- The ability to pronounce Russian and other foreign texts

Braille terminals and printers



A Braille terminal consists of an array of 20–80 Braille cells, connected to the computer. This displays part of the screen display in Braille, so that a visually impaired person can read it.

Such terminals are very expensive, but are preferable to the voice-operated/synthesiser system in some contexts. It is highly suitable for:

- Programming
- Dealing with formatted texts
- Non-natural-language texts

There are a number of Braille printers on the market, allowing documents to well in conjunction with the Braille interface. A good Braille printer will take

- The operation of the printer by a visually impaired person
- The facility to print simultaneously in normal type
- The facility to print in Cyrillic Braille code
- Graphic Braille printing

Computer technology can be used to improve quality of life for disabled peo movement can control equipment attached to a computer by using just a fin been developed to assist motion-impaired people. *IRVIS (Interactive Robotic V* system being developed to allow disabled users to interact with a GUI to faci control of a robot.

Information overload

There are many information systems available to executives and managers. T informed decisions, based on the most up-to-date information. Systems inclu intranets and email. However, one argument is that increased access to infor detrimental to business interest. A recent survey commissioned by Reuters fo executive managers claimed to have been made ill by stress.

One of the key reasons cited was the overwhelming volume of information b Internet named as a major source. The fact that so much information is avail where they have to keep up, or else they will be disadvantaged. Though ther subject, it is clear from the Reuters survey that in some situations the quanti

Computing can do more to help situations such as this, through more intellig solution to this problem lies not in reducing the information load, but in bett organisation of the data. Both of these tasks can be performed by computers would take a person to organise the same amount of data.

Forged documentation

Advances in computing and printing technology, combined with its increased a affordability, has made it more possible to create accurately-looking forged do has seen an increase in identity theft and illegal immigration. This kind of tren impact of society as a whole as the level of control over identity is important i maintaining a safe and secure society. Producing counterfeit money is another example of forgery that has become more widespread due to the level of accuracy that is achievable, due to technology.

In order to combat this, documents such as passports are becoming increasingly complex in terms of the security features they contain. For example, the most recent UK passport (2015) is printed using UV and infrared light, inks and watermarks and uses a single sheet of paper for the personal details page adjoined to the back cover to prevent it being tampered with. However, with the illegal trade of ID being so widespread, it will surely only be a matter of time until a method of duplication comes to li

Questions: Computing Issues

1 You work for an organisation that would like to gather information regardi which calls are handled by employees. Discuss the potential ethical issues r

2 Consider the implications of pirating software and how the act of repro those who have produced it, those who are using it and those who hav

3 What are the implications of a government flying reconnaissance drone area?

4 Research the health risk factors involved with heavy and prolonged use with common health risks found 50 years ago.

9. Communication and Networking

The need for machines to communicate with each other has become an essential part of i shows how we have developed our standards and methods to include networking. This, ir revolutionised the world and its business and social links.

This section covers:

9.1 COMMUNICATION

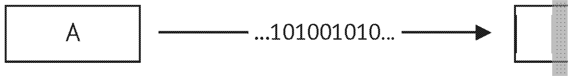
COMMUNICATION METHODS

In order for a computer system to be responsive and useful it needs to have t user, its surrounding peripheral devices and its internal devices, so there are imposed data transmission in a system that enable the computer to do ju

Serial transmission

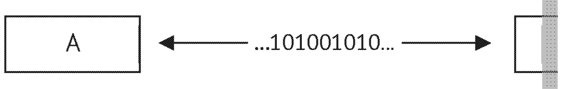
In serial transmission, the binary signals representing the data are transmitte several different classes of serial transmission:

- *Simplex* method allows communication in one direction only. This o



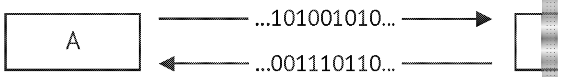
Serial Simplex Connection

- *Half-duplex* method allows communication in both directions, but no there is a single channel and the direction is switched after completi direction.



Serial Half-duplex Connection

- *Duplex method* allows communication in both directions at the same permanently available. This is a necessity in interactive systems (whe continuously required).



Serial Duplex Connection

Serial transmission can operate either in synchronous mode (with a mutual c mutual clock). USB, SATA and RS232 are all examples of common serial com serial connection standards, such as SATA which is used to connect hard driv USB (universal serial bus) is a very common connection type used to connect

Parallel transmission

With parallel transmission more than one series of data bits can be transmitt cables in place of the single cable found in serial communication systems. Pa be synchronous since there is a central clock that controls the timing of data

Parallel communication is only practical over very short distances due to the wires get longer, parallel transmission suffers from skew. As the distance is i the lines move slightly out of sync and this massively reduces reliability.

Parallel communication used to be very commonly used to connect periphera However, due to the expense and limitations of the parallel system, serial co commonly used now. One place parallel communications can still be found i PCI bus, for example, is a parallel system. However, cases where serial commu replace parallel ones. For example, PCI express is actually primarily a serial s

Asynchronous data transmission

Asynchronous data transmission is a method of transferring data that has co order to synchronise the transmission. This means the transmitter's and rece synchronised and only become synchronised at the time of the transmission

Start and stop bits are used as markers to indicate where the message starts stop bits added to them before sending. Start and stop bits are used to synch the two devices. Both devices must receive and send signals at the same rat This is the reason why start and stop bits are required.

For example, the bit string 00101111001 would have the stop bit 0 and a st 0 00101111001 1.

COMMUNICATION BASICS

Before you can understand how data flows through a system and how comp some basic principles and terms that you must learn.

Baud rate

Baud rate is the rate at which the signal changes (per second) in the communica measurement used for a baud rate is called a baud. For example, 2 kBd means 2

Baud rate is often confused with bit rate. The main misconception is that on sending one bit. However this may not be true, since one signal change may represent more than one bit. If we take a voltage of a wire to determine the voltages to signify more than one bit. For example, we could detect betwee than one bit at a time, as below.

Signal Voltage Level	Binary Signal	Signal Voltage L
0 V	0	0 V
7.5 V	1	2.5 V
		5.0 V
		7.5 V

Bit rate

Bit rate is the number of bits that can be transmitted per second. The units o kbps (kilobits per second), Mbps (megabits per second) or Gbps (gigabits per second) means that 9,600 bits of data can be transferred per second. A bit ca

Bit rate = baud rate × bits per baud

Bandwidth

Bandwidth is the useful bit rate, also known as the information rate.

Usually, when transferring data between networks, some bits are required to handle communications which are considered 'wasted' bits. Bandwidth is also measured in bits per second. A bandwidth of 2 Mbps (2 megabits per second) indicates that there is 2 megabits of information lying in the communication medium. This means that the bit rate and the bandwidth are directly proportional to each other.

Latency

Latency is the time required for one bit to be transferred from one end to an between the sender sending a message and the receiver receiving the messa milliseconds. It is usually related to the length of the connection and the spe routing/forwarding taking place.

Protocol

When data is being transmitted in a computer syste that constrain how the transfer of data will be done established before transmission can take place is w Protocols are most commonly found in web-based s be indexed.

Questions: Communication Methods

1 What is the difference between synchronous and asynchronous transn

2 Are the following true or false?

a) Baud rate is the same as bit rate. (1 mark)

b) Bandwidth is the time taken to receive packets of data. (1 mark)

c) Latency is the time taken for 1 bit to be transferred across a given

3 Using what you know of the different buses, what controls the directio transmission? (1 mark)

9.2 NETWORKING

NETWORK TOPOLOGY

Bus network topology

In this system, a single cable called the bus is used, to which the workstatio

This system is simple to run and normally uses less cable than any other sys as one cable is used for the bus, and computers simply connect into it using cables. However, if there is a break anywhere on the bus cable, it results in a the workstations being unable to communicate. Another disadvantage is tha there is a larger degradation of performance under load than with some other topologies.

Advantages	
■ Inexpensive ■ Scalable – easy to add new nodes	■ Single point of f ■ Network perfor

Star network topology

A star network has one central message-switching device through which all computers communicate. In this system, each workstation is connected direc the central device – normally a switch – by its own unique cable. A major disadvantage, of course, is dependency on the central device, as all communications will cease if it breaks down.

The main advantage of the star topology is that it is easy to track down prob because each workstation has its own separate cable. Also, this means that a with one cable or workstation does not affect the rest of the system. Perform doesn't degrade when new nodes are added so the system is very scalable. A more cable to install a star topology, and the switch can be quite expensive. instead of a switch; however, if the computer crashes, then the other comput with each other.

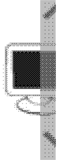
The star topology is now the most popular choice when a new network insta

Advantages	
■ No performance degradation ■ Secure communication – communication is di therefore no eavesdropping is present ■ Scalable – easy to add new nodes	■ Expensive to im involved theref

TYPES OF NETWORKING BETWEEN HOSTS

Peer-to-peer networking

Peer-to-peer networks are those where there is no central server and every computer has the same rights. There is no central management. Peer-to-peer is useful when most of the time computers do not need to interchange information with each other. Small companies benefit from peer-to-peer networks since having independent computers does not require an expensive server or hardware to manage and maintain.



Programs are installed on every computer; however, careful management is required to ensure all software and data is up to date and protected. Sometimes a peer-to-peer network will have shared directories to store information on one machine, however, on most peer-to-peer networks the data is stored locally on each machine and so therefore each machine requires regular backups to prevent data loss.

Small networks often are peer-to-peer as the cost of maintenance does not w server l[illegible] however, as the number of machines increases in an organisati networks will justify the extra expense.

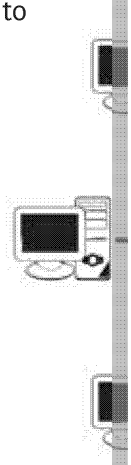
In February 2009, peer-to-peer networks have been estimated to collectively 70% of all Internet traffic (depending on geographical location). *BitTorrent* is for sharing files on the internet, and uses peer-to-peer networking to distrib Internet. This involves multiple computers (or *peers*) using a BitTorrent clien with each other. No central authority is needed to coordinate the download c segmenting the file into pieces and finding separate nodes to download thos 2013, BitTorrent was responsible for 3.35% of all worldwide bandwidth; mor bandwidth dedicated to file sharing

Server-based networking

Also known as client-server networking Server-based networking is where ce managing resources, security and other services. These services are provided these servers. The server is the only computer that holds all the information a organisation; this is advantageous where a lot of interaction is required betw organisation is able to control the access to the information better, provide b resources more efficiently (i.e. remotely) across a network rather than having

Server-based networks are used in companies where people need to access resources all the time. A server can cope with many people connecting and gaining a resource without a drop in performance.

Security is centralised and users can authenticate themselves by using a single password to access the resources they are allowed to. This allows a user to be able to access their information on any computer in the network rather than relying on an individual machine. There is also no need to worry about backing up data, since it is all done by the server itself. A lot of people can be added to the network without a drop in performance. Increasingly, home servers are also beginning to appear as the need to share data and resources around homes with multiple computers grows.



WIRELESS NETWORKING

Wi-Fi

Wi-Fi is a high-bandwidth wireless method of communication which can be used in place of, or in combination with, wired Ethernet networks. Wired Ethernet networks are, however, still capable of far higher transfer rates and are more reliable than Wi-Fi networks. Wi-Fi networks are accessed through *hotspots* which are the areas within which a device can connect to a local area network, and are used to allow a device to connect to the Internet.

All Wi-Fi networks are based on an international standard so that, given you
device can connect to any network without hassle.

Connecting to a wireless network

Wi-Fi hotspots are actually the broadcast area of what is known as the *wireles*
often connected to a router which in turn is connected to the network itself. \
the device would need to be connected to the router directly using cables.

The other device you need to connect to a wireless network is either a *wirele*
attached externally to a computer via USB), or a *wireless network interface co*
hardware component attached directly to the motherboard (in desktop comp
the PCI slots). Both of these act as an interpreter – to send and receive data
easily transmitted.

The type of network adapter needed will vary depending on the protocols it
communication medium and the topology of the network to be connected to.

How networks are secured

There are a number of ways in which you can improve the security on a wirel
developed over time to help secure a network and prevent *rogue users* being a

The methods that you are required to know are:

- *Wi-Fi Protected Access (WPA)* versions 1 and 2
- *Service Set Identification (SSID)* broadcasting
- MAC address whitelisting

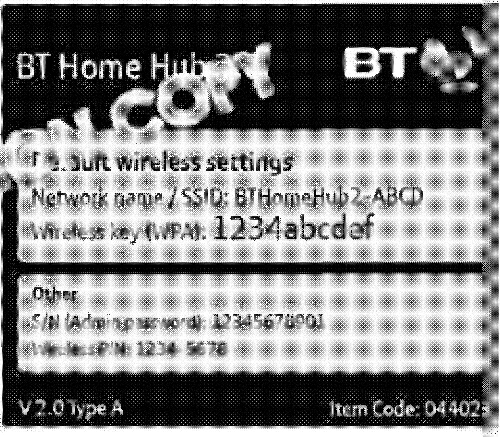
Wi-Fi Protected Access (WPA/WPA2)

WPA was introduced in 2003 to address the shortcomings of Wireless Encryption Protocol (WEP). When you connect to a network that uses WPA as a security procedure you'll need to enter in a passphrase; this acts as a lay
you cannot connect to the network. However, when you connect to the netwo
adapter communicates by creating an encryption key. The key is generated usi
the resulting key is unique for each device connected to the network. Securit
generated keys are temporal. Temporal key integrity protocol (TKIP) is a prot
integrity checks to ensure that the keys have not been tampered with; when
encryption type between devices is changed.

Service Set Identification (SSID)

An SSID is the name given to a wireless local area network by the administra connected to a network must have the same SSID employed to their connecti each other through the network, otherwise they are invisible to each other.

Think about your home network; the odds are that your router broadcasts its so that if the device knows the WEP key it can connect. A way of improving t be to switch off the automatic SSID broadcast; that way it becomes invisible that know the *exact* name of your Wi-Fi can connect to it.


BT Home Hub 2

BT

Default wireless settings

Network name / SSID: BTHomeHub2-ABCD

Wireless key (WPA): 1234abcdef

Other

S/N (Admin password): 12345678901

Wireless PIN: 1234-5678

V 2.0 Type A

Item Code: 04402

MAC address whitelisting

Many administrators of WLAN will use a MAC address whitelist to control wh network. The whitelist effectively acts as a spam filter to block every device whitelist table, which can only be accessed by logging in to the router. This i data being sent over a network simply isn't feasible.

Carrier Sense Multiple Access with Collision Avoidance (CSMA/C

CSMA/CA is transmission protocol used in networks that acts to prevent the c nodes. As soon as a node receives a packet that is to be sent on the network, is a channel clear that it can use to send the packet. If there is no available c back-off time is generated, at which point the node will check again.

If a packet of data that is larger than a predetermined size is needed to be se be a certain *handshake* that needs to happen; this is called the *Request to Se* This protocol only takes effect if the packet is larger than the threshold beca or will take more bandwidth to send than a smaller packet.

Questions: Networking

1 What are the advantages of a star network topology? (3 marks)

2 Which piece of hardware is required to access a wireless network? (1 ma

3 Explain how you could use a combination of networking methods to i network. (6 marks)

9.3 THE INTERNET

THE INTERNET AND HOW IT WORKS

The structure of the Internet

The Internet is vast, and the Internet you see on a day-to-day basis when usi is just the tip of the iceberg – but if the Internet is truly that large, what is it

The hardware that makes up the Internet is made up of *servers* which hold In servers together and other equipment *(routers)* that link computers to the Int makes up the Internet is invisible to people who use the Internet. It produce from one side of the world to another, and organises the email system.

Packets

For data to be sent across a network it is broken down into a structure know broadcast as a [illegible] across the network medium. To give you some idea, a of about [illegible] bits, so a 2 megabyte file will be broadcast as 20,000 packets. network so the most direct route to the destination will change during the c packets will arrive in a different order to that which they were sent in.

To enable packets to reach the correct destinations and then be assembled i information is stored in part of the packet known as the *packet header*.

The header consists of:

- The destination address
- The source identification
- A checksum (for transmission error detection)
- The packet sequence number

Destination
Source
Checksum
Sequence

Packet switching

Packet switches have two main functions:

1. To enable more than one device to share a (usually high-speed) data
2. To find the most direct route for information to travel

Packet switching was developed to replace circuit switching and to make th autonomous. For example, the telephone system originally used dedicated c each other. This is clearly suboptimal, as a normal conversation would tie up constant data stream, which could use far more bandwidth. A more efficient across a network as they are required, which results in potentially increased

Routers

A *router* is a device that forwards packets from one network to another. Each network will have a router which works out where to send packets which are destined for the network they are actually within. When a packet is sent acro Internet, it will go through many routers before it reaches its destination.

Routers are able to work because every piece of network hardware (with the exception) has a unique MAC address (*see 9.2.3*). Routers can therefore chan packet without changing the destination IP address. This means that routers another, changing the destination MAC address as they go, while maintainin destination. Each router on the path is therefore able to work out where nex at the destination IP address.

Suppose a user wants to send information from one computer to another co
prepares the message and sends it off to the router. The router obtains the f
table. When the route is found, the message is transferred to another router.
its network the destination node then the router transfers the message to th
the message to another router which performs the same process. All routers
each other, which makes communication universal, and are able to transfer d
architectures.

Gateways

Gateways are the entrance and exit of a network and can be considered as p
main use of a gateway is to connect multiple networks of different architectu
system. The gateway doesn't have to be a physical object, though; it can be b
and into the network software which can then be distributed throughout the
physically bound to a single location. The gateway takes in a packet from an
removing all the information apart from the raw data. The raw data is then r
network in the protocol which that network supports so that the data ca
to where it appears in a network map, usually towards the edges and access
the network security that has been employed by the network administrator.

Uniform Resource Locator (URL) and internetworking

An internetwork is a collection of independent networks connected to each other. Each network functions on its own and does not depend on other networks. An example of this is the Internet. The networks in an internetwork are joined together with routers which handle the URL requests to find certain volumes of information. The Uniform Resource Identifier can be either a Uniform Resource Locator (URL) or Uniform Resource Name (URN). A URI provides a unique reference for an Internet resource while not necessarily providing an exact location for the resource as it could a query in a database or call to some application. This is why the
URL is an identifying location for a resource providing a means of getting th

For example, the URL **http://www.google.com** is the URI that exclusively ide
namely the Google homepage. It describes the type of representation that is
implies that the Hypertext Transfer Protocol (HTTP) must be used to collect
hand, provides a way of exclusively identifying something, for instance an IS
identify a specific book, but provides no means of actually getting hold of th
equivalent to a person's phone number or postal address, while a URN is eq

Domain names and IP addresses

Domain names provide the text that is entered in the navigation bar of the w
domain name server (explained below) to return the specific IP address of t
the address '**www.google.com**', then '**www.google.com**' is the domain name w
identifies the top-level part of the domain and is often related to the type o
particular domain name. The '**google**' part of the domain, known as the seco
word identifier for a particular IP address. The '**www**' provides the name of th
deals with Internet requests; it may also be possible to find addresses such a
the host server that would deal with file transfer protocol (FTP) requests. Als
in the hierarchy between the secondary and sever part of the address; for ex
'**www.images.google.com**'.

Internet registries

If you stopped to think about IP addresses for a moment you'd soon realise t values, so an impartial system was required in order to facilitate the distribut consequently IP addresses. This is the role of Internet registries and registrar organisation called the Internet Assigned Numbers Authority (IANA) that dec used in which regions, and the local Internet Registries and their registrars th assigned to them according to local legislation, etc. An example of a UK Inte **www.nominet.org.uk**. If you had just founded a company it would be a regist approach in order to register the domain name of your new company. This of registry and you will also often have to 'rent' the web space where your web them to be accessed by others.

These companies will also often be involved with legal proceedings when it should not own specific domain names, or when they are profiting from another company other than Ford Motors had owned **www.ford.com** before it was a t that they were profiting from Ford's name, then the domain name might be v given to Ford Motors.

INTERNET SECURITY

Firewalls

Firewalls are a very common and easy method of protecting your network, and subsequently your system, from the risks of using the Internet. Traditionally, users of a network would rely on the security of the individual hosts to protect them but as the number of hosts increases it made it less manageable; there was more likely going to be a lapse in security due to the drop in uniformity of security. Personal firewalls were developed to combat this and use services to provide the protection.

Packet-filtering firewall

Packet filtering operates at the packet level and studies each packet as it arrives and passes through router interfaces depending on the filtering protocol that has been put in place, which is checked against the features of the packets themselves.

Filtering protocols are put in place by the network administrator or someone who is overseeing the protection of the network. There are obviously too many packets involved in data transfer to monitor each packet that attempts to gain access to a network; this is why the network admin will employ several policies to regulate the ports of a network.

Advantages	Di
■ Functionality provided by standard router firmware ■ Fast ■ Flexible ■ No user action required for installation	■ Filtering rules are v ■ Cannot be tested fo ■ Routers may only be when a break-in occ ■ May not be able to f

Stateful inspection firewall

These firewalls have no built-in concept of 'state' or 'context' – that is, the fi is for, where it is going or what it does. This type of firewall examines *every* prevents any packets from entering or leaving a network if they do not meet of packet examination is *very* resource-intensive as there is constantly a stre the system at any given time. It stores the *states* of connections into a state hashing of the data to be processed more quickly; the states that are stored against the states of the allowed policies. However, if the first packet of a co standard then there is no need to check the following packets associated wi

Advantages	
■ Faster than using a proxy server ■ Due to its nature it has added pseudo-security built in by performing application layer filtering of [illegible] co[illegible] ■ More secure than basic packet filtering ■ Cheap to set up ■ Flexible but strict on the rule set	■ Can be less secu ■ Slower than bas weighed down b ■ Very resource-d ■ Can be vulnerab target the proto ■ Creating the pol can be difficult

Proxy server

A proxy server is often used with a firewall to increase the security of the sys Computers inside a network use proxy servers to access external information main advantage of this is that only one computer is exposed to the outside n the rules to be 'fine-tuned' to allow control over connections.

Advantages	
■ Only allows proxy services through ■ Protocols can be filtered and manipulated ■ Hides internal structure through information *hiding* ■ Improved authentication and logging ■ Cost-effective ■ Rule set is less complex than packet filtering	■ Protocols are m authentication s ■ Editing protocol ■ Cost-effective b

SYMMETRIC AND ASYMMETRIC ENCRYPTION

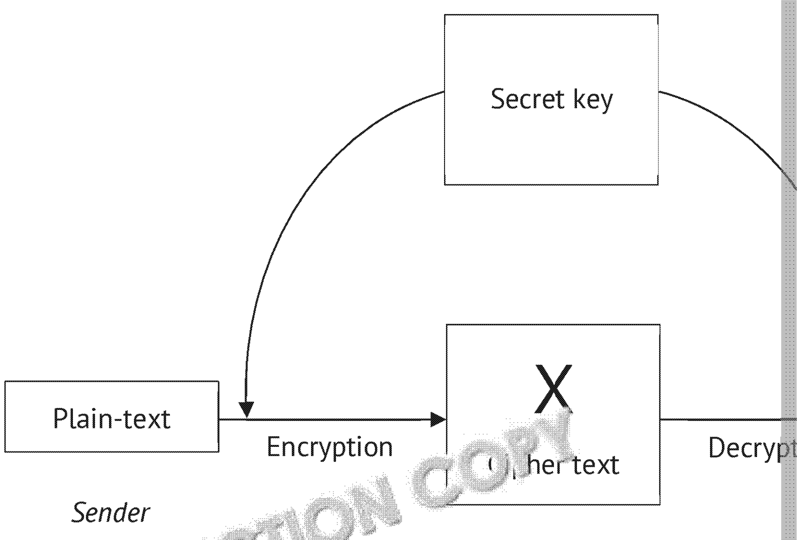
Encryption is a technique used to protect data by making it unreadable. Spec
used to convert the data, which is in plain text, to cipher text. Plain text is th
is the original data that has been transformed into a completely unreadable f
understand without the use of a key. Keys are used to decrypt messages into
messages into cipher text. Symmetric encryption uses the same key for encry
asymmetric uses different keys for encryption and decryption.

In public key cryptography, each party using a public/private key encryption s
known only to them, and a public key, which is freely available. These keys a
private key can decode messages encoded by the public key, and the public k
encrypted with the private key. Importantly, however, the public key cannot
encoded by itself and the same goes for the private key. To send an encrypte
transmitter will encode the message using the public key of the receiver. On
private key to decode the message and so only they can decode it.

Messages encrypted using a public key can only be decrypted using the corre
way, messages encrypted using the private key can only be decrypted using t
what allows the process of digital signatures.

Symmetric encryption

As mentioned above, symmetric encryption is where the key that is used to se
encrypt and decrypt a message. Technically, it is the inverse of the key that is
the data required to complete a process is derived from the secret key that is
feature of the application that is being used to transmit the data.



A good example of a symmetric encryption key that was used widely was the
which dates back to 1970. When it was first introduced the developers said t
symmetric key's 52-bit secret key, but it soon came under scrutiny because c
be secure. In 1997 there was a joint effort of 14,000 as an attempt to try and
and it took four months of continuous computation to derive the key's value.
was built for £250,000 which managed to crack the algorithm in just three d

Asymmetric keys

The opposite of the symmetric key is the asymmetric key. These are frequentl business world because they offer one of the highest forms of protection. The

- The sender doesn't need to decrypt the data once they've sent it
- Only certain people should be able to read the encrypted data
- By separating the keys people can only send data

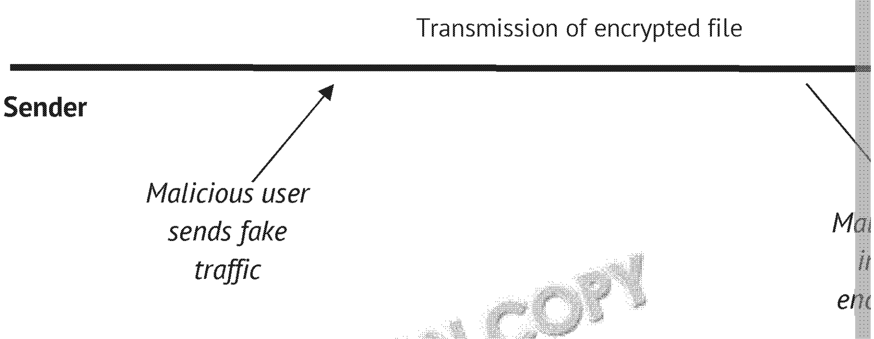
By separating the keys and making them different, it allows you to control w messages because once the message or data is encrypted it becomes increas private key to go with it. This is because with the symmetric encryption you were affected and still decrypt the message to retrieve the plain text.

Without the private key it becomes almost impossible to retrieve the plain te improves the integrity of the message being sent.

Did you know?!

There is a wide variety of security threats that have been developed by people an example, one of the most prevalent threats that has been making its way around ware. This is where a virus has infected a computer and instead of self-replicating devices on the network, it will encrypt the data on the hard drive with a key with you don't pay the ransom then they don't unlock your computer.

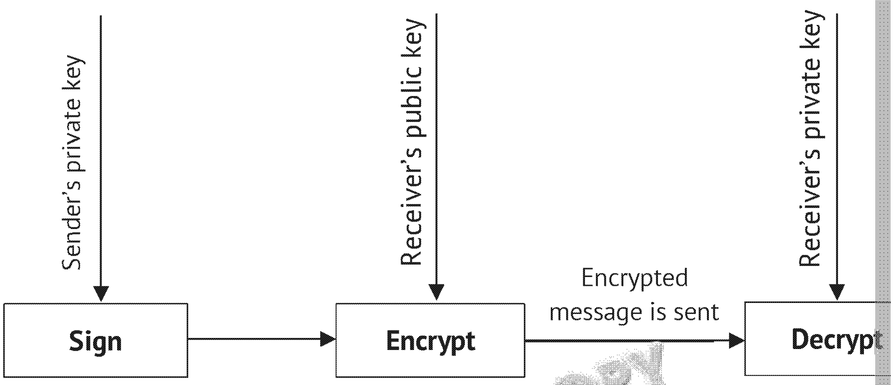
The public key is made available to everyone; it is either found on a compan given to them by other means. The sender then encrypts the file using the pu the receiver. Malicious users can try to intercept the file, but if they do it is m cipher text; other malicious users can still send traffic using the public key b by a spam filter on the receiver's end. The receiver can then decrypt the mes produce the plain-text representation.


Transmission of encrypted file
Sender
Malicious user sends fake traffic
Ma
i
enc

DIGITAL SIGNATURES

Digital signatures are used in conjunction with public key cryptography. Publi the data secure but it does not guarantee that the message being sent has n signature, the person who receives the message is able to tell who the sende message has been tampered with.

A digital signature is generated by analysing the document and formalising i using the mathematical notation, a hash function acts on the data and gener depends on the content of the message. If any of the contents change then t document and show that it has been tampered with. Digital signatures are al programmers sign their applications, which shows that a specific application



DIGITAL CERTIFICATES

A digital certificate is a type of ID card which lets you identify a specific user. name, ID such as a serial number, the user's public key and the certific that the certificate is real. Certificates are used to verify that the sender is act reason for this is that maybe someone pretends to be someone else and senc you can encrypt the message using their public key instead of the intended p situation, a certificate authority gives you a certificate which verifies your ide someone wanted to send you some piece of data, then they should include th that the certificate authority is contacted to verify that the certificate is valid.

SECURITY THREATS

Viruses

A *virus* is a computer program embedded into another apparently harmless p harm to a computer. The first step that a computer virus performs when the p itself onto disk and hide itself. After being copied onto disk, the virus can res system so that it causes problems. Viruses tend to create multiple copies of t to other computers. Antivirus programs are used to detect and remove these have built-in virus scanners which scan files that are available for download.

An infamous example of a virus would be that of the Commwarrior, the first v multimedia messages or via Bluetooth. The virus would be sent by an MMS; the message then the .SIS file would install itself onto the phone; it would th messages containing a copy of itself to other phones from the contact list. Th the virus was through Bluetooth connectivity; when this was used the virus w phones every minute and try to send a copy of itself to them.

Worms

A *worm* is a malicious program designed to replicate itself in an attempt to s such as the internet. The most significant difference between a worm and a v program [illegible] its own. Worms can disrupt entire networks, causing traffic, deny The worms spread by several means but the most common is by email; altho damage to the host device until their payload has been delivered, they can c at which traffic is handled in a service.

Consider a server that handles the call requests for a business's login details. is the possibility that every time a data call is made to the server, instead of worm could return a copy of itself. The user wouldn't log in, so they would tr they wanted – they'd be none the wiser that their computer had now been in

Did you know?!

One of the most famous worms that has been seen is that of the Morris worm. Th
a Cornell University student in the USA. The student wanted to see how big the In
lines of code to trace an IP address, return the IP address and search the host co
when files were transferred. Although the code wasn't intended to be malicious, th
that made the host service very unstable and often resulted in the machine being

Trojan

The term *Trojan* is derived from the Trojan horse used to defeat Troy by the G
replicating virus that is hidden in a downloaded file and is unleashed when t
gain access into a system. The Trojan program then acts as a back door for s
infected system.

Not to leave Mac users out from potential threats found in modern computin
found a virus circulating Apple computers in 2006. The virus was called 'Leap'
be transmitted over the Internet. Instead the virus would affect local area ne
home with multiple computers or those found in the business world. Leap w
chat program found for the Mac OS. The virus would then target other progr
access to the most secure parts of the computer. It would ask non-admin use
admin rights; the admin would then put their password in which would be lo
to access the whole of the computer and would use the instant messenger's
computers and send a copy of itself.

Did you know?!

One of the perceived advantages of Apple Mac computers is that there are no vir
However, this perception isn't entirely accurate. In practice this is true because of
an Apple machine. The problem, or solution, with transmitting viruses on a Mac is
run third-party applications without explicit permissions, so you can infect a Mac v
be slow and limited. Apple products are no harder to infect than any other machin

The first virus ever produced on a Mac was by Rich Skrenta in 1982. Skrenta, a I
cloner which was capable of infecting the Mac's boot sector, a feat that is still imp
virus predates the first IBM virus (the 'Brain' virus) by nearly five years.

Questions: The Internet

1 A packet is said to have a payload of 64 bits. What does this mean? (1

2 What is packet switching and why is it useful? (2 marks)

3 What is a packet-filtering firewall? (1 marks)

4 What is the difference between a computer virus and a Trojan? (2 ma

9.4 TRANSMISSION CONTROL PROTOCOL / INTERNET P

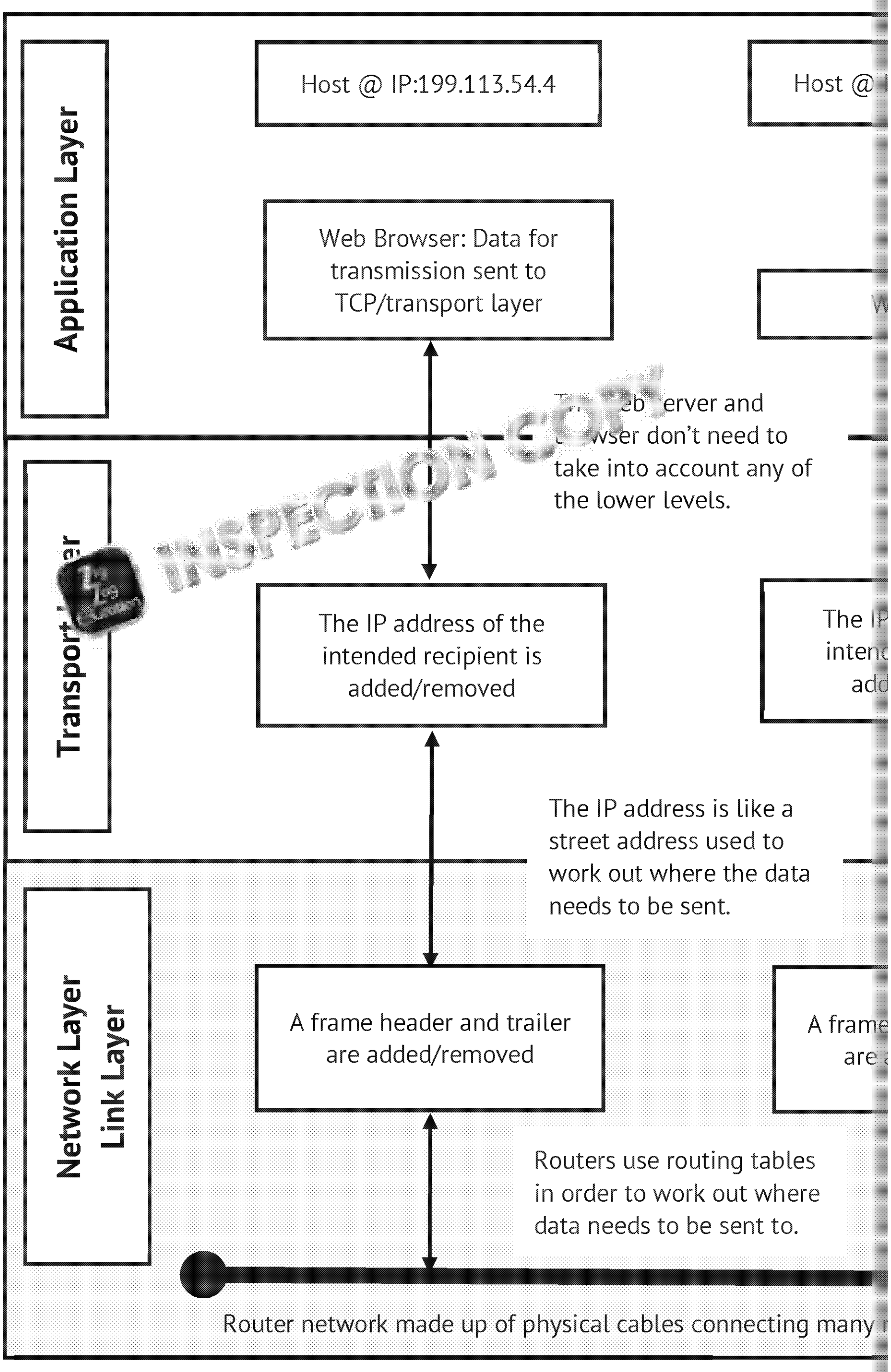
TCP/IP

TCP/IP stands for *Transmission Control Protocol / Internet Protocol*. It is a protocol that allows communications on LAN and WAN networks and is widely considered the standard for Internet communications. The protocol is arranged in a stack with four layers which are:

1. Application Layer
2. Transport Layer
3. Network Layer (IP)
4. Link Layer

This layer process means that data at the application layer, which is the actu transmission, undergoes several stages before it can actually be transmitted. the transport layer before being given an IP recipient in the Internet layer a to distinguish it from other communications on the network. Once these sta information is broadcast to the intended recipient, where the stages are reve original data. This involves removing the header and trailer and also the IP a separate packets. The recipient can then process the application data as it w packets are labelled with an order position because often they will get disru reordering on receipt.

The complexities of each of these layers are removed by the introduction of a programmer to have any involvement in the layers below the application l of Internet applications much more straightforward. They often allow socket to file paths and then used to transmit data to the intended recipient. A sock '**www.google.com:8080**' which sends the request to the IP address associat 8080, which may be waiting to receive information from a given application page in text format once a connection has been made. This will be used by t into a web page using the formatting information within the HTML tags. The render the Internet resources correctly and control the content being displa following diagram shows how the different parts of the protocol fit together


Application Layer
Transport Layer
Network Layer
Link Layer
Host @ IP:199.113.54.4
Host @ I
Web Browser: Data for transmission sent to TCP/transport layer
W
The IP address of the intended recipient is added/removed
The IP
intend
add
A frame header and trailer are added/removed
A frame
are a
Router network made up of physical cables connecting many r

The web server and browser don't need to take into account any of the lower levels.

The IP address is like a street address used to work out where the data needs to be sent.

Routers use routing tables in order to work out where data needs to be sent to.

Well-known ports

Ports are the computer software's way of exchanging data directly; they each l
relates to the kind of packet it is. Often certain protocols and applications
numbers from 0 to 1023 are the well-known ports and contain entries such a

20 – data FTP
21 – control FTP
25 – Telnet

80 – HTTP
110 – POP3
443 – HTTPS

The port numbers from 1024–49151 are the registered ports which have bee
Assigned Numbers Authority (IANA). The port numbers from 49152 are the pri
outside of these three ranges are known as the ephemeral ports.

Media access control (MAC) address

MAC addresses are a unique 6-byte (42-bit) identifier that all network interfaces have in order to communicate with a network. The MAC address is assigned to a device by the manufacturer and, unlike an IP address of network mask, cannot be changed once it is assigned. It is used by the media access controller in the link layer of the TCP/IP model.

All MAC addresses on a network are kept in a table by the router for the net
with an available subnet address so that the packets that you're trying to re
not someone else's while also helping the router to maintain what addresse

STANDARD APPLICATION LAYER PROTOCOLS

Hypertext Transfer Protocol (HTTP)

This is the protocol that defines how data from web pages is transferred fro
when t[illegible]ne is being viewed over a TCP/IP network. It is common to see t
beginning of a web address, for example **http://www.google.com**. The HTTP
the IP address and the server will usually return a web page or the like to th
eight HTTP commands that are combined to allow all the required functiona
requests a specific resource, POST which returns information, typically from
is used among other things to create an HTTPS connection (which is discuss
to make use of this protocol and to be able to retrieve web pages in text for
Java require Internet browser plugins which are initiated by these protocols

File Transfer Protocol (FTP)

The File Transfer Protocol is what is used to download and upload files between computers.

Downloading is the term used for copying files from the Internet to your own computer. You will have noticed that Internet addresses have *http* at the beginning. This tells your Internet browser that you are downloading a web page from the Internet. Sometimes you want to download a different sort of file from the Internet, for example a trial program, a collection of pictures or a word-processing document. This is carried out using *FTP*.

There are also many FTP sites around the world. These are computers that h
other files that are freely available to everyone. Most major companies on th
Microsoft, IBM and Novell). Most modern browsers have FTP built into them,
are *downloading* such files you won't need a special FTP program, although i
downloading files it may be worth your while buying one.

If large files are being distributed, then it is quicker to collect them using FT
reliable because FTP programs can often continue to download if somethin
connection halfway, rather than starting from scratch. Uploading means tra
onto the Internet. You would do this if you have your own Internet site and
written to your web space on the Internet. Again *FTP* is used, and most mod
upload functionality built in as long as you have the correct permissions for

Secure Hypertext Transfer Protocol (HTTPS)

The HTTPS scheme is very similar to HTTP except that it connects to a different standard port and there is a layer of encryption between HTTP and TCP protocols. In standard HTTP the information that is transmitted could be intercepted and decoded by anyone, which makes the information transmitted over those connections insecure. In situations such as ecommerce and e-banking this level of security is not high enough.

The HTTPS protocol encrypts the data before it is transmitted over the network and sends it to a different port to the standard HTTP protocol. This suitable for private information.

Post Office Protocol version 3 (POP3)

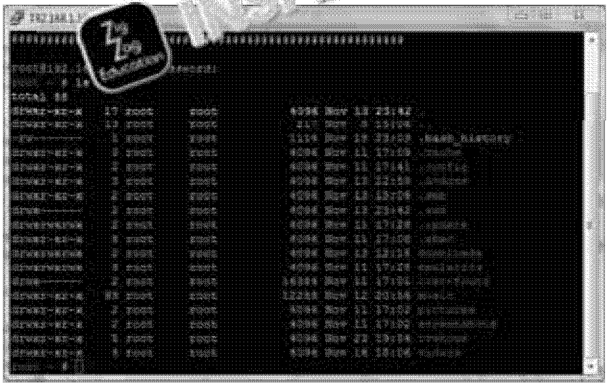
This protocol allows an email application to receive emails from an email serve the email server, download the messages, save them as new messages on the u message from the server. This means that once the messages have been access they are no longer available on the server. A newer protocol, IMAP (Internet Me suitable for email users who have two modes of email viewing, one of which is messages are downloaded and saved, but also from a web browser where mess through an online application. In this protocol the messages are only deleted w user explicitly requests that they are. The main difference between these two p is that POP3 uses the user's mail store as the primary store and IMAP uses the s store as the primary store. The email server will be located with the ISP or at th location of the email service company being used. This is the location that ema sent to and where they are received from using the POP3 and SMTP protocols.

Simple Mail Transfer Protocol (SMTP)

SMTP stands for Simple Mail Transfer Protocol and is used for the distributi message will include the intended recipients and the message as well as ot necessary. These are then sent to the sender's mail server which will use th correct mail server for the recipient, who will use a POP3 or IMAP protocol t

Secure Shell (SSH)

Secure Shell is a remote-access protocol that allows secure communication The user has to be using the SSH client while the server must be running th connection is made over a secure channel on a potentially insecure network one else but the user can see the data transferred. It is used to log in to a re commands across the server. The connection is made using TCP to a port of using other application-level protocols to perform a certain task. For exampl company's mail server the POP3 [illegible] to [illegible] can be used to retrieve email, or i command for HTML.



This image shows an access the directory Embedded Linux Ent a Windows network.

IP ADDRESS STRUCTURE

An IP address should be a value unique to a device so that packets can be co between devices, otherwise an *IP conflict* will occur as packets are distribute the same IP address. All IP addresses are made up of four quadrants separate binary address:

XXX · XXX · XXX · XXX
IP address: 192 · 168 · 10 · 5
Binary: 11000000 · 10101000 · 00001010 · 00000101

If you liken an IP address to your home address you can understand better h to identify devices. Your home address will have a county part and a city par network will have the same 'county' part and the devices will be given separ

For example, say there is a LAN with three devices on it: a server, a printer

Server: 10.30.15.1	Computer: 10.30.15.5	P

You can see that the company's IP is in the first three quadrants and is '10.3 the device itself. This is called the company's *host address.*

IP class

There are also different classes of IP which were introduced in the *classful In* The class system was developed to assist in the distribution of IP addresses at the table below.

Address class	First octets of address		Range of first octet values
Class A	**0**xxx	xxxx	1 to 126
Class B	**10**xx	xxxx	128 to 191
Class C	**110**x	xxxx	192 to 223

Class A	Class B
126 routable addresses *For example:* ***84****.42.199.5*	16,384 routable addresses *For example:* ***140****.153.82.254*

SUBNET MASKING

A subnet mask tells a computer which IP addresses it is able to reach directl switch) and which it cannot reach directly and therefore needs to access thr the same form as IP addresses; however, each number is either 255 or 0. 25 must be the same as the one on the host. A 0 means that that part of the IP can still be accessed directly. The mask itself is a 32-bit number that is appli AND comparison to split the address into the network identifier and the host result is only true if *both* the values are the same and is set to 0 if they are n

Address class	Bits for subnet mask			
Class A	11111111	00000000	00000000	00000
Class B	11111111	11111111	00000000	00000
Class C	11111111	11111111	11111111	00000

Therefore, an example of a B class network with the IP address 138.96.0.0 w network notation would be written as 138.96.0.0/16. Take a look at the follo this is used.

In this worked example you will see how to find the network address for the the subnet mask 255.255.240.0.

10000001	00111000	10111101	00101001	IP address
11111111	11111111	11110000	00000000	Subnet mask
10000001	00111000	10110000	00000000	Network add

The IP address 129.56.189.41 therefore has a network address of 129.56 wit usable host identifier.

IP STANDARDS

There are currently two standards of IP address that are in use today. These a Internet Protocol and are *IPv4* and *IPv6*. These are the two fundamental tech devices need to connect to the Internet.

IPv4

IPv4 is the 32-bit IP address that everyone is familiar with and that has been the previous page for explaining IP addresses. The 32-bit IPv4 IP address me are available (roughly *4.29 billion* addresses). However, since its inception th predicted and in 2011 it was noted that the last of the IPv4 addresses had be that we've run out, though; the majority of addresses are dormant addresses used by conglomerate companies, but it does mean that in the near future w standard (IPv6). An example of an IPv4 address would be 192.168.0.0, a *Clas*

IPv6

The move to IPv6 is inevitable; IPv4 addresses are becoming a dwindling res trade item. IPv6 is similar to its predecessor in that it assigns a unique nume connect and communicate with a network, but it has one *major* difference. In 128-bit address which will *vastly* increase the pool of available IP addresses. whereas 2^{128} grants an IP address pool of 3.40×10^{38} – three hundred and fo addresses. There are numerous advantages of IPv6 over IPv4. The large incre address means that instead of writing the IP address as you would for IPv4 y hexadecimal.

An example would be *4aae:1803:[illegible]0:b9be:de12:94cd.*

Did you know?!

Are you struggling to get your head round the size of the IP pool for IPv6? Let's pu million IP addresses per day, every day of the year it would take 9.32×10^{29} yea another way 6.7×10^{19} time the current age of the universe. Try to write these n length of time we're talking about. Compare these to the number of seconds in a

Advantages of IPv6

- Removes the need for *Network Address Translation (NAT)* as all devices a
- By removing NAT you remove the possibility of private address collis
- Easier to administrate – naturally leads to the removal of *Dynamic Host C*
- Removes the need for a system administrator to set up complex netv
- Higher data transmission speed due to direct routing of packets
- Increase in security due to packet routing and the size of the IP pool
- Network infrastructure becomes simplified

Disadvantages of IPv6

- Specifying IP addresses becomes considerably harder with the increa
- Internet service providers are granted absolute control over their net issues, denial of service or restricted service by throttling an IP range
- There has been no clear decision as to when or how the Internet pro transition and how this will affect the service they are providing to t
- The transition will be very complex for everyone involved as the mo phased transition to get the core components functional
- All machines that operate on IPv4 will become obsolete almost imm which means the transition will be costly for everyone
- Potential rise in the cost of computers – with everyone needing nev Internet service, manufacturers could charge whatever they wanted f

PUBLIC AND PRIVATE IP ADDRESSES

Routable IP addresses

Public IP addresses are assigned by IANA (Internet Assigned Numbers Author localised registrars, who then assign numbers to individual users/companies address on the Internet is unique, otherwise routers would not know where t public IP address of your router, not your computer. When you connect to th middle man for all the packet traffic going in and out of your home network and as a way of controlling the number of applied IP addresses. If every con address then we would have run out of IP addresses several years ago, but w distribution of packets is taken over by a network router.

Non-routable IP addresses

IANA has specified certain IP addresses as private. These are for local netwo to the Internet. To connect a LAN using these IP addresses to the Internet a translation (NAT) must be used. Home routers universally use NAT to work o supposed to end up [illegible] change outgoing packets so that they appear to b address of the router.

An analogy for this would be the internal post at a company. The Royal Mail office block (the public IP address) and then the internal post works out whic (the private IP address).

Private IP addresses fall within the following ranges:

10.0.0.0–10.255.255.255

172.16.0.0–172.31.255.255

192.168.0.0–192.168.255.255

DYNAMIC HOST CONFIGURATION PROTOCOL (DHCP)

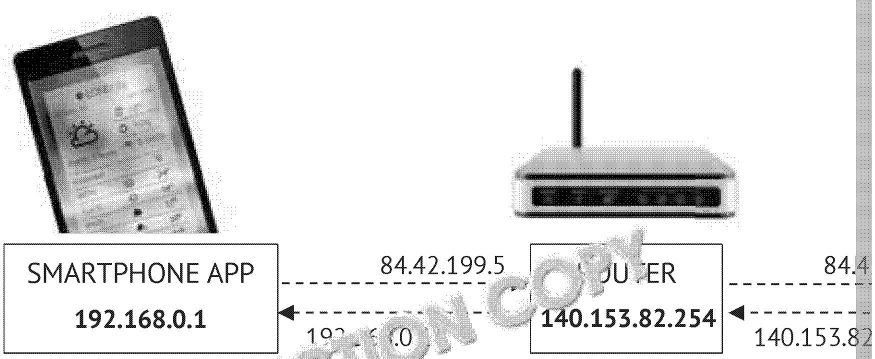
Dynamic Host Configuration is the protocol that supplies an IP host with all th data across a network. Without this protocol, if you were to move a compute the network admin would have to manually recalibrate the network so that t operate in the new subnet location, the IP address of the computer from the reclaimed manually and the process would be lengthy.

With the protocol, however, the entire process in automated. The protocol ma addresses on a network for devices and it operates on what is called a *lease* p a network it sends a request to the DHCP server which will service the request available internal IP addresses that can be assigned to the device and allocate policy that allows the device to connect to the network until it expires because t an elapsed time frame. When a lease expires, the IP address is added back into

NETWORK ADDRESS TRANSLATION (NAT)

Network Address Translation can be seen as a bit of a 'workaround' for IPv4 to number of IP addresses available. As mentioned before, IPv4 allows for 4.3 b use but there are over 7 billion people on the planet, each of which would p Internet with more than one device. To combat this, each device is given a p these addresses are private, no servers can communicate directly with a devi the router and the router responds to the device.

For example, if you were getting ready to go to college in the morning and y weather will be like later in the day, you could use your smartphone to look send a packet to the router containing the private IP address (return address server (destination address) and a message of what is being requested. Whe creates a log in the NAT forwarding table and instead of broadcasting the pr it will broadcast the network's public IP address. When the request is handle formulates a response, the reply is sent back to the router IP address and th the device by checking the NAT forwarding table for the private IP address.



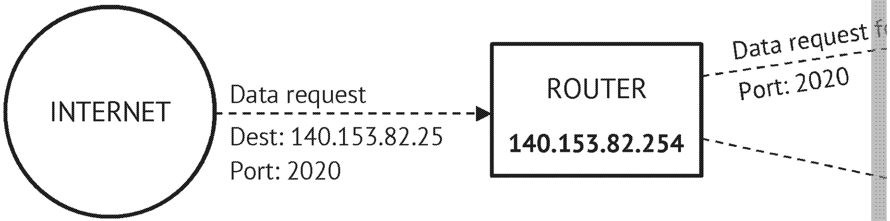
The outbound data request	The i
In the outgoing request, as described above, the smartphone sends a request message to the weather app server at 84.42.199.5 on a given port. In the request it also includes the return address for the data, but when the message reaches the router the return address is changed to the public address of the router.	If the router hadn't weather app server and would fail beca non-routable. Inste address and stores t device in a table so from the server to t

PORT FORWARDING

NAT sorts out the majority of problems when it comes to granting each com a network, but what if a computer wanted to connect to your network to use ports are used; they allow computers to exchange data directly between eac complex translation step that would increase the latency of the data connec

You have already seen some well-known ports in one of the examples on pre similar to network address translation, but it operates on the port level. Once allows you to put a port forward with a specific port number, so that all data or are directed at that port will be forwarded to a specific computer on that online games where a user has the opportunity to create a personal server f safer to enable port forwarding so that the other users cannot connect to the to allow complete access to the network.

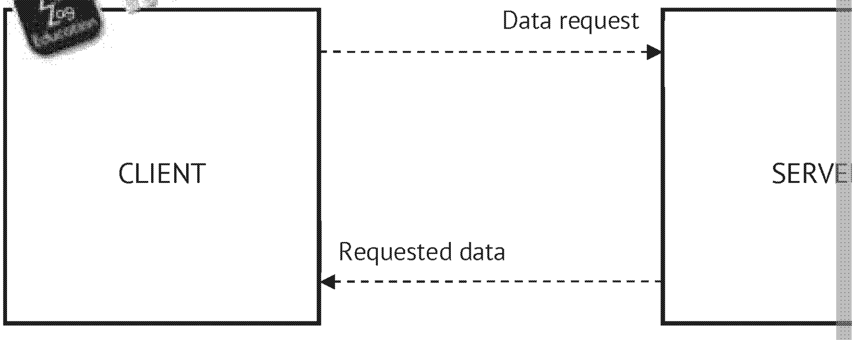
Take a look at the example below; the network has two internal connections and the other is a personal server for a game that a user has created and is o



CLIENT–SERVER MODEL

The client–server model is a network architecture that dominates network d computers in the network belong to one of two sets, or in some cases they c machine or application that is usually interacting with the user. It is respons via the network to a server, or small group of servers. A server receives these accordingly and then returns the requested data to the client over the netwo

The connection between the client and the server uses the '*WebSocket' Proto* programming interface over which a full-duplex connection is established be via TCP. This allows for a stable flow of data between both parties at any tim



The protocols used to control the data flow to and from the server use a pro State Transfer (REST) which relies on the HTTP methods. The client cannot i though. Instead the server creates a set of instructions that take into accoun functionality of any given method that needs to be available; these instructi programming interface (API). Therefore, the REST API is created and run by t enabled by the client's web browser calls the API.

The acronym 'CRUD' describes the basic functionality that needs to be imple

C – Create **R** – Read **U** – Update **D** – De

In order for the client computer to interface with a server database, the REST methods to be mapped onto the basic SQL commands as follows:

HTTP Method	SQL Method
GET	SELECT
POST	INSERT
DELETE	DELETE
PUT	UPDATE

Any data that is transmitted during communication between a client and a se either Extensible Mark-up Language (XML) or JavaScript Object Notation (JSO server admin might need to change some of the functionality; traditionally th of the aforementioned languages – there are some large differences betwee

JSON is considered to be much easier to use than XML because:

- More compact (same complexity but operates much more easily)
- Easier to create during initialisation of the server
- It is higher-level – it is easier to read and understand
- Easier for the computer to perform the operations required, therefore more reliable

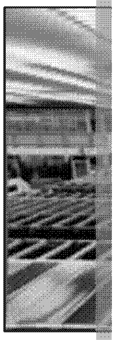
THIN- VERSUS THICK-CLIENT COMPUTING

In client–server situations a decision needs to be made as to who handles th this is a crucial decision to make because it can affect all clients connecting device or an early version of a smartphone then you can see the difference if 'web version' of a social media site; the device can't handle the volume of da because the server it has connected to is intended for computers.

Thin-client computing

Thin-client computing is a new way in which a computer is used. 'Thin' comp relatively low processors and low amounts of memory that are continually c This separates the computer into two parts where the thin computer perform containing only the minimum number of parts, whereas the server performs data. If the server has any 'downtime' then all data transfer is halted, renderi server reconnects.

A great example of thin-client computing can be found when looking at the Large Hadron Collider, the world's largest particle collider at CERN in Geneva. The collider consists of a 27-kilometre-long ring made of superconducting magnets that collides beams of particles together at 99% of the speed of light. There are so many sensors in the ring that it is impossible for a single computer to process the data that is created or to perform any operations on it, so data is stored and manipulated by a central server. Scientists can then request the results of the operations to be displayed on their monitors.



For more information see Sections 11 and Section 10.5

Thick-client computing

Perhaps unsurprisingly, the opposite of thin-client computing is known as *th* computing. This is where the computer has the capability to operate and sto stored [illegible] it can be verified instantly, but this can also run the risk of de risk of data loss. This is more expensive than deploying a thin-client server b potentially expensive hardware to cope with the flow of data during connect intermittent than thin-client servers as there isn't the need for continuous da

One of the best examples of thick-client computing is that of online gaming modern games do not require a dedicated server; instead a single player wil entire game but all of the calculations are done on the separate nodes (the c central hub which will store various data such as the scores and locations of is called for by other consoles. This can be shown when there is an element host or when there is an issue with the host's Internet connection. Everyone affected and the game doesn't function properly; this is because the host ca by the nodes and the game experiences 'lag'.

Questions: Transmission Control Protocol / Internet P

1 Explain the following concepts:

a) IP address (2 marks)

b) Public and private addresses (2 marks)

c) Subnet mask (2 marks)

2 What is a MAC address? (2 marks)

3 What advantages does IPv6 have over IPv4? (3 marks)

4 What is Network Address Translation (NAT)? (3 marks)

10. Databases

Computer systems are nothing without data and information. The very purpose of a comp
produce the desired output. The processing of data is of the utmost importance in all are
learn about the database and how it can be used to become an efficient tool for storing a

This section covers:

10.1 CONCEPTUAL DATA MODELS AND ENTITY RELATIO

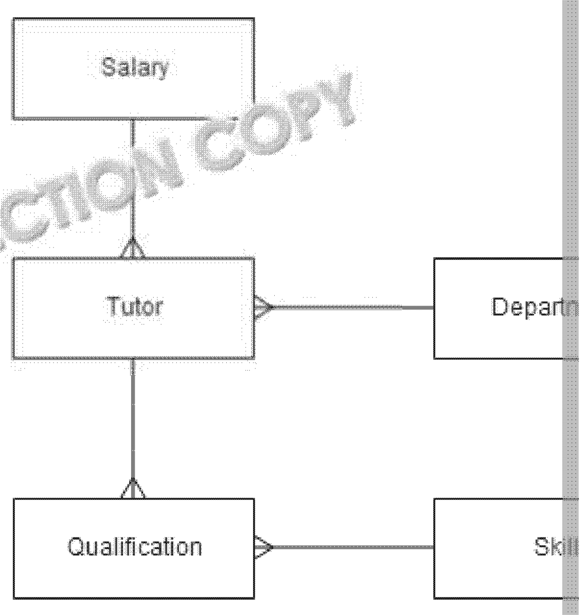
Logical data modelling is the process of using scenario information to produ model for a database. Entities are abstractions of real-world concepts (e.g. St which information needs to be kept in the system. Possible entities can be fo scenario. The relations between the entities can be found by looking for to look at *how* entities could form relationships and resolve any irregularitie

Databases are structured systems for holding data records. They are a step u tend to become less usable as the data structure becomes more complex. Th to its usefulness as databases need to be easily searchable. They also need t updated and maintained properly. Poorly designed databases can be very dif so it is important to design them well. For this reason, conceptual data mode is dependent on what other data.

Data structures can be modelled by breaking the proposed data structure do relationships:

- An *entity* is any item in the system about which data is stored, e.g. *Sa*
- An attribute is a property of an entity, equivalent to a field in a datab *Manager* might be *Name* or *Age*
- A relationship exists between entities. This dictates whether an entit entities or just one.

A simple version of an ERD for the tutor example that follows would look lik



Entity definitions

This is the act of producing an overview of the entities displaying all attribut maintenance tool so that a database admin can see how the entities relate t The attributes for a table are written in parentheses after the entity name. Th underlined) followed by all other attributes. The last attributes tend to be fo and are marked by an asterisk (*). In the tutor example, the Tutor entity defi

```
Tutor (TutorID, FirstName, LastName, D_o_B, OfficeNo, Salar
```

Questions: Conceptual Data Models and ERM

1 What are the 'top-down' models used for? (1 mark)

2 If the entity 'Salary' is said to contain two fields, what might the entit

10.2 RELATIONAL DATABASES

This model states that all data is stored in relationships formed by tables. Da *tuples*, and the records are identifiable via their primary key. These relationsh capable of handling data while rejecting erroneous data types.

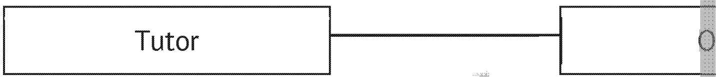
RELATIONSHIP TYPES

The relations in a database between entities are the foundations of the datal relation found between the entities where the variation comes from. Relatio or 'many-to-many'.

One-to-one

Examples of non-redundant one-to-one relationships are relatively rare. The linked to only one of another entity. Redundant one-to-one relations can be attributes, as this relation tends to only occur between attributes within the

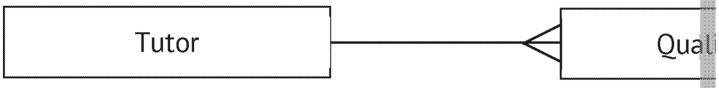
Dor example, if you look at the relationship between a tutor and their office,



One-to-many

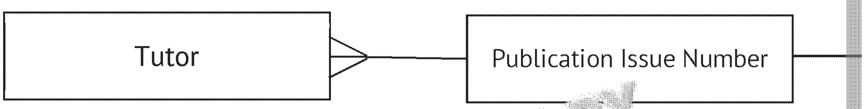
One-to-many is the most common relationship type where one field from a t from another table.

An example of a one-to-many relationship would be the relationship betwe they have. A tutor can have one or multiple qualifications, but a qualificatio doesn't make sense semantically or technically.



Many-to-many

Many-to-many relationships are quite common. For example, a tutor might h might have multiple tutors. This type of relationship can become complicate because they cannot occur in real life; data cannot be transferred in that fash sense. Therefore, many-to-many relationships are resolved by using an *assoc* can publish many papers and a paper can have many publishers so the assoc *Number*; this is because it's the simplest link.



ATTRIBUTES AND KEYS

Attributes

These are elements of data that denote properties of the parent entity – the consider a table that needs to hold personal information about the tutors in would include name and contact details but you'd need to consider items su office number. Look at whether the attributes you store can be derived by ot of birth or age, not both.

Primary keys

A primary key is an attribute that allows others to be uniquely identified. Lo key in our example would be first name. However, a primary key needs to be name. Instead, we create a new field, one that allows us to create complete field (e.g. TutorID).

Composite key

A primary key that involves more than one attribute is known as a *composite* needs a composite key would be one where the names of people are stored first name or a surname alone would not be sufficient as a primary key. For e 'Mark Jones', 'Mark Peterson' and 'Jan Peterson' which means that neither the would identify a person individually. In order to identify them uniquely you c two names.

Foreign key

A foreign key is a primary key in another table. Flat databases don't use forei reliable and robust databases relationships *must* be created. These relationsh duplicate data from the tables by referring to existing items. In the tutor e form a department, the Department Code will become a foreign key in the tu

Questions: Relational Databases

1 **What is meant by a relational database? (1 mark)**

2 **The following definition is for an employee entity within a relational**

```
Employee (EmployeeID, FirstName, LastName, D_o_B, Offi
```

a) **Identify the primary key in the Employee entity. What is a prima**

b) **Identify the foreign key in the Employee entity. What is a foreig**

10.3 DATABASE DESIGN AND NORMALISATION TECHNI

Normalisation is a rational data analysis tool useful for validating entity rela optimising database structure. In order to normalise, several (as many as pos the form of data samples. These are inputs (forms) and outputs (reports and

UNF – Un-normalised Form

1. Select an *initial key*. This acts as a 'starter' key. It needs to be unique can derive other values from it. If there is no suitable key, add one.
2. Transfer all identifiable attributes, ensuring each has their own *relev unique* name.
3. Look for repeating groups. These are a group of similar attributes tha multiple values for a single value of the initial key. Select a suitable for the repeating group, surround them with brackets, and write them from the single-valued attributes.

As you can see in the example on the right, the group in the brackets is the r group and is separated from the non-repeating group. SkillCode is promoted become a *key attribute* for the relation. Note that in un-normalised form ther propagation for the separated group.

1NF – First Normal Form

This is the most basic of the normalised forms; in order for data to be in the fi form you must:

4. Create new relations by separating all repeating groups; select a new key for the new relation and *propagate* (copy) the initial key to form a *composite key.*
5. All other single-valued attributes remain with the initial key.

As you can see below, this is the first step to providing some structure and re to the database. There is a new composite key formed when you propagate t key down to form the new relation with SkillCode.

2NF – Second Normal Form

The second stage of normalisation requires that all key attributes in a table a dependent on each other and that the table is in first normal form. Only ther move on to creating a database that is in second normal form which will pro more strength to the data structure.

In order for the data to be in second normal form you must:

6. Separate any attributes from keys formed in the previous step that ar dependent on one part of the composite key.

As you can see in our example to the right for the tutor example, SkillDescri only dependent on the SkillCode, not the TutorID. If this was left in first norm and implemented as a solution, you would quickly find that the database wa performing as optimally as it was intended to and would accumulate redund

This would slow the database's performance greatly. Logically it makes sens create a new relation to prevent this from happening. Key propagation must again but only for the original primary key for that relation (SkillCode). There will have created a new relation as can be seen on the right.

3NF – Third Normal Form

The third and final step of the standard normalisation is the transition into t
form. This is the step that most people struggle on, but if you use the pointe
look at the example that has been used throughout the steps it is clear to se
required of you. In order for data to be in the third normal form you must:

7. Separate any attributes that are dependent on other non-key attribut foreign keys are retained in the original relation.
8. Check composite keys for redundant parts. If a part of a key can be d from other attributes, demote key attribute to non-key.

Take a look at the example on the right: in the first group that had been left
unchanged since the beginning, we've moved two attributes down and creat
relation for each. This has been done because for each they are only partly d
on the TutorID and rather there would be multiples for each. There will be m
tutors in the same department faculty and there will be multiple tutors on t
pay grade. You will also need to create a new 'ID' attribute for each of the ne
relations and use these as foreign keys in the first relation. The database is n
third normal form.

For an overview of all the steps take a look at the table on the following page.
work from left to right on the table you can see how each step naturally and lo
leads to the next, and how the final format of the data is optimised for removin
redundant and duplicate data. Following this table format can be very beneficia
the information you need is available for you and you can see how entities beg

Did you know?!

*The normalisation ladder actually has six more forms before the database is fully
operations that can be performed. These push the database theories as far as the
concerned, but the final steps are aimed towards improving the performance of jo
together during a query or maintenance of the database. Seeing as joins can be v
the size of the database, these final steps are seen as the final requirement but a
applications where the database needs to operate as smoothly as possible.*

You'll be pleased to know that this is well and truly out of the scope of this course

Normalisation example – overview

Salary and department are separated and their ID fields are retained as forei

The entity names have been added to the right.

UNF	1NF	2NF	3NF
TutorID	**TutorID**	**TutorID**	**TutorID**
FirstName	FirstName	FirstName	FirstName
LastName	LastName	LastName	LastName
D_o_B	D_o_B	D_o_B	D_o_B
Grade	Grade	Grade	OfficeNo
SalaryCode	SalaryCode	SalaryCode	SalaryCode*
OfficeNo	OfficeNo	OfficeNo	DepartmentCo
DepartmentName	DepartmentName	DepartmentName	
			DepartmentCo
(SkillCode	**TutorID**	**TutorID**	DepartmentNa
SkillDescription	**SkillCode**	SkillCode*	
Qualification)	SkillDescription	Qualification	**SalaryCode**
	Qualification		Grade
		SkillCode	
		SkillDescription	**TutorID**
			SkillCode*
			Qualificati
			SkillCode
			SkillDescrip

Bold & underline indicates primary key
Asterisk (*) indicates foreign key

Normalisation is primarily used by system and data analysts. When carried o optimised database model. The resulting structure means that managing the span of the solution is increased. The relationships will eliminate redundant changing data easier. In the tutor example, if a tutor changes address or dep would need to be changed, but if the database is normalised the *database m* care of all changes.

Questions: Database Design and Normalisation

1 Describe the steps of transitioning from the following:

a) 1NF to 2NF? (1 mark)

b) 2NF to 3NF (2 marks)

2 How does normalisation affect a database's performance and reliabili

10.4 STRUCTURED QUERY LANGUAGE (SQL)

Structured query languages are the regular languages that are used to create databases in a variety of management systems. There is more than one type 'SQL' denotes the original language that was developed in the early 1970s b but the functionality still remains the same. The aim of SQL was to provide a language that was very high level so that database administrators could mar

THE DATA DEFINITION LANGUAGE AND DATA MANIPUL

There are two language sections in SQL that have been developed for perfor database; these are the *data definition language (DDL)* and the *data manipulat*

DDL

The *data definition language* is a language used to build the structure of a dat commands which allow the user to define the structure of the database by cr The DDL also allows the administrator to apply constraints to a database tab add the constraint of access rights for a table to prevent certain users from e altogether. Here is an example of a DDL statement to create a database calle customer and order:

```
CREATE DATABASE db; # create an empty database
CREATE USER dbuser IDENTIFIED BY 'password123'; # identi
GRANT ALL ON db.* to dbuser; # grant all permissions to
CREATE TABLE db.customer ( # create a new table in db ca
    customerID INTEGER,   # attribute definitions (name
    firstName  VARCHAR(20),
    surname    VARCHAR(20),
    phone      VARCHAR(14),
    PRIMARY KEY   (customerID),
    UNIQUE INDEX  (customerID)
);

CREATE TABLE db.order ( # create a new table in db calle
    customerID INTEGER, # attribute definitions (name |
    orderID    INTEGER,
    PRIMARY KEY   (orderID),
    FOREIGN KEY   (customerID) REFERENCES (customer(cust
    UNIQUE INDEX  (orderID)
);
```

Note:

The 'CREATE USER ...' and 'GRANT ...' commands simply create a user with a p to do whatever they want with the database. The 'CREATE TABLE <table_nam <column1> <datatype>, ..., PRIMARY KEY (<columnx>));' construct is probably th it is how tables are defined using this language. Another significant comman delete databases and/or tables.

The normalised database with multiple tables is linked together utilising the command shown above. This links the tables by identifying that the custome the customerID in the table 'customer'.

DML

The *data manipulation language* is the language used to populate and update t
created. DML allows the user to insert, modify, delete and query a database ta
table, for a specific record or for records that meet criteria. Here is an example

```
USE db;
INSERT INTO customer(customerID, firstName, surname, pho
VALUES (1, ‘Bob’, ‘Smith’, ‘+441234568909’);
```

The basic commands for the DML are:

SELECT	Retrieves the data of a record from a ta
INSERT	Inserts data into a table in a new record
UPDATE	Overwrites existing data in a record
DELETE	Removes a record from table

It is these commands that are will allow you to implement a database, perfo
information from a database.

QUERYING DATABASES USING STRUCTURED QUERY LA

Structured Query Language (SQL) is a language designed to search databases i

In order to gain some understanding of SQL we will look at some of the quer
we will call Customers.

Surname	Age	Children	Occupat
Harvey	21	1	Salesm
Ellison	67	2	Retir
Watson	40	6	Nurs
Clarkson	25	0	Unemplo
Butcher	18	0	Butch
Smith	30	4	Docto

Similar to programming in an imperative language, SQL uses comparison op

=	Equal to	>	More than
<> *or* !=	Not equal to	<=	Less than or
<	Less than	>=	More than o

LIKE

The LIKE operator allows you to search for wild cards that come close to ma conjunction with operators to meet the need of the query. Take a look at the

LIKE	Result
LIKE “D*”	This will search for strings that begin with the letter ‘D’ f characters.
LIKE “*son”	This will search for strings that end in the string ‘son’ pre characters.
LIKE “*l*”	This will search for strings that contain the letter ‘l’ prec of other characters.
LIKE “Sm?th”	This will search for strings that match the search string b the question mark
LIKE “*#3	Similar to searching for a string that varies by a characte numerical value as opposed to a string value.

Given what you’ve been told about the DML above, you will now see a simpl above table use the SELECT command. The SELECT command selects which the FROM keyword selects which table the results should be drawn from. Th meaning and basically means ‘all’. The statement below shows how this wou

```
SELECT Surname, Occupation FROM Customers
```

This statement would return:

Surname	Occupation
Harvey	Salesman
Ellison	Retired
Watson	Nurse
Clarkson	Unemployed
Butcher	Butcher
Smith	Doctor

WHERE

However, it is very rare that you will require all the information contained in the information that is returned by a query. You can accomplish this by using example that would retrieve the records of customers that have no children.

```
SELECT * FROM Customers WHERE Customers.Children = 0
```

This statement would return the following:

Surname	Age	Children	Occupat
Clarkson	25	0	Unemplo
Butcher	18	0	Butch

Conditionals can also be used. For example, to select everyone 40 or more y
statement:

```
SELECT * FROM Customers WHERE Customers.Age >= 40
```

Other keywords also exist which allow you to select a particular range of val
NOT among others. For example, the following statement would select every
not have any children:

```
SELECT * FROM Customers WHERE Customers.Age > 20 AND Cus
```

Another useful clause to use is ORDER BY. This means that the results are re
(using the keyword ASC) or descending order (using the keyword DESC). Take
instance:

```
SELECT Surname, Age FROM Customers WHERE Customers.Childr
ORDER BY Customers.Age ASC
```

This would return:

Surname	Age
Harvey	21
Smith	30
Watson	40
Ellison	67

Multiple tables in SQL

As has been seen with normalisation (section 10.3), the data in a database is
important to be able to retrieve, update and delete data using other referenc

For example, using the orders system it may be important to contact the pers
tell them it has arrived or similar. The order ID is in the orders table, wherea
in the customer table.

Method 1

SELECT customer.Surname, customer.FirstName, customer.phone FROM cust
orderReq AND customer.customerID = order.CustomerID

Method 2

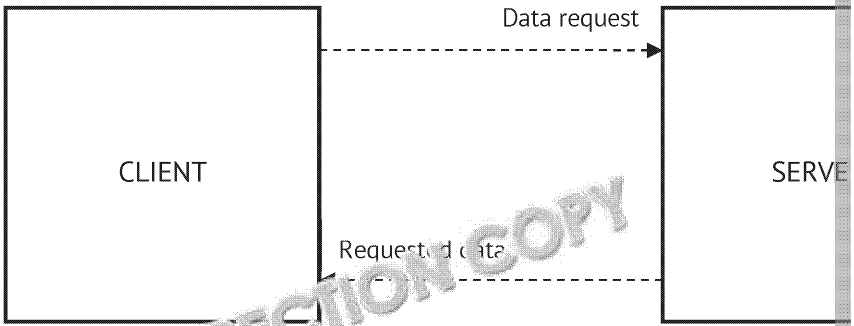
SELECT customer.Surname, customer.FirstName, customer.phone FROM cust
orderReq INNER JOIN customer ON customer.customerID = order.CustomerID

Please note the SQL here shows both tables in the FROM statement; some v
just the 'customers' table with the inner join referencing the 'orders' table.

You will be required to be able to write and interpret SQL statements in an e
practise these statements. There are several ways of doing this: you could se
some network space or local host; some DBMS software such as Microsoft Ac
query definitions.

10.5 CLIENT–SERVER DATABASES

The following diagram is the same as the one found in the notes for *client–s* the basic structure and actions of the model, and the same applies for a clie



The client can be a node on a network, an individual computer or someone s planet and the structure still works the same. The idea is that a client reque server) by a connection made over a wide area network (such as the Internet as those found in an office building. Once the data request is fulfilled, the c example of this is found on any Internet browser; the browser acts as a clien computer with a database that has the stored data of the website, and once packets to display a website the connection is terminated.

Another example can be seen in online banking. When online banking was firs questions and concerns to be answered about security, integrity and cost. Thes most are unfounded as most do not understand how the technology works. Th interfacing client for the database again and sends a request for data from the server replies with the authentication and authorisation steps that are require requesting the information. If the checks are passed then the request for the d server returns the data that was requested, and the connection for that specifi

ADVANTAGES AND DISADVANTAGES

Advantages

- Centralisation (data being stored in a single location) – This allows serve of the server's network, how the data requests are handled and how files
- Accessibility – As long as the server is up and running then the data any time and from any location as long as the user has permission t
- Adaptability – The model allows the use and adaptation of the thick *(for information see Section*
- Scalability – Due to the adaptability of the model, the database can capacity and it doesn't necessarily mean that the client's computer p accommodate the increase in volume of data; only the database's hardware
- Backup and recovery – Because all the data is centralised it means that backups of the database so that if there is a failure or loss of data the d or no consequences for the users of the service, minus server downtime
- Security – The model allows a server/database administrator to con and how data is returned to the client. Limiting what the client can p the data in the database can be preserved. This is also increased wh client and the server is dropped after the request is fulfilled.

Disadvantages

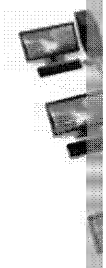
- Network overload – Due to the way the database is implemented, it can become overloaded with data resulting in slow response times o This can lead to data loss, corruption or *concurrent access*.
- Cost – These servers and their hardware are not cheap to implement increased when the server is being used over the long term or for fre need to prevent a network overload and server downtime.
- Robustness – It is the centralisation of the data in the database that robust as a peer-to-peer network. In peer-to-peer networks, if the se performance the data can be transferred to another host computer q the server in a client–server model is lost then all transitions of data terminate until the server comes back online, which may require spe
- Maintenance – There is a level of IT knowledge that is required to m this need is increased when you're dealing with a service that many can't be any reason why the database isn't available (downtime). A sp administrator, is required to ensure that the network isn't becoming perform maintenance tasks on the server's hardware.

Although the client–server architecture is costly to maintain, it is relatively ea ingrained into modern life and how we interact with data because of how it al the server. *Concurrent access* was referred to above; this is when the server allo the same piece of data at the same time. However, *concurrent access* can occur same piece of data at the same time, meaning that an update is lost between

CONCURRENT ACCESS

Concurrent access is making sure that more than one user can at least view the same data at the same time, and there can be many reasons as to why you would want this in a computer system.

However, the real issue with concurrent access is that it could lead to both users making changes to the file, in which case one change will always be lost after both users save the file. This will inevitably lead to data loss from a system and potential critical errors for the clients connecting to the server.



Protocols have been developed to help maintain the integrity of the data in the database if concurrent access occurs. Concurrent access can

Record locking

Record locking is something that your computer will implement when you ha another copy of the file. The second instance of the file will be opened in a r changes to be made. This is also implemented on a database and is the act o being accessed while someone is editing the file or data item.

Serialisation

Another way of preventing concurrent access is to create a clone of a data ite the user to apply changes to and then upload a copy of this clone to the data *serialisation*, and by creating a clone of an object before amending changes i can be lost during editing.

Timestamp ordering

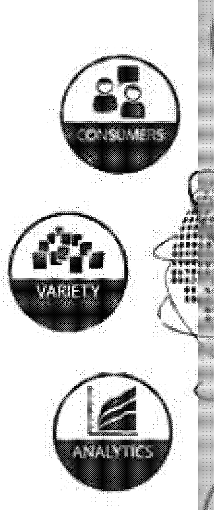
Timestamping is a non-lock method of concurrent access so that multiple pe the same time without causing ordering errors. The assumption is that all ti unique so multiple users can access the database data.

11. Big Data

Big data is seen as an 'all-encompassing' term given to data that won't fit the usual data constructs or containers.

The reference to 'big' however is potentially misleading as it implies that volume is the only factor when determining whether data should be classed as 'big data', when in fact there are factors to consider.

These factors, and some ways of addressing the issues posed by Big Data are explored in this section.



Big data can be described in terms of:

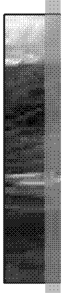
- *Volume* is the obvious reason why data would be considered 'big'. If s that it is no longer suitable to store on a single server then there is n process that data without breaking it down somehow. You may think be to store it across multiple servers on a relational database. Howe scale very well across multiple machines.
- *Variety* is possibly one of the more relevant reasons to call somethin and in different formats and data types that it becomes difficult to st hard to interpret meaningful information.
- *Velocity* is the speed at which data needs to be accessed, i.e. during s time to respond to the data.

Did you know?!

If you fully compressed all of the new data created daily, how much informational data storage do you think is required? A few hundred gigabytes? Terabytes?

Not even close, as of 2014, 2.5 exabytes of information is generated each day. That's 2.5 billion gigabytes, 2.5 × 10^{18} bytes, every single day.

The largest data capacity facility is found at the Utah Data Centre (shown here on the right) and can store a massive 12 exabytes of data.



VOLUME, VARIETY AND VELOCITY

Volume

Volumes of data are the easiest to picture when it comes to handling data because everyone can relate to how difficult it is to handle and cope with large amounts of information. When a dataset is so large that it can no longer be accommodated by a single database then it is easy to see why it is given the name *big data*. It is hard to link computers together in such a way that the memory in the co mputer can function as a single unit, so instead the dataset is broken down and spread across large arrays of computers in a database and the data can then be processed.

To help put big data into context, consider the following. At CERN in Geneva contains 150 million sensors, each with a polling rate of 40 million times ev of *raw* input data created every *second*; in a single day it would fill 86 1 tera calculation, that is 30 petabytes (31,457,280 GB) of data being recorded ann

All of that raw data needs to be stored before it can be processed and that is comes into use; this is explained *Section 13*.

Variety

There is no way for a computer to visualise a dataset and see patterns without being able to look at each individual element of the dataset. The odds of a dataset being retrieved in a logical order that is ready to be processed without being stored and processed first are minimal, so it is natural to assume that there is always going to be variety in a dataset especially when it is raw data that is being collected.

Within a dataset there can be large differences in the data that is being processed. This is why it is one of the more relevant constraints of big data – that a system can struggle to cope with large quantities of data that varies in such a way that it becomes difficult to know how to store it. On large networks where there is the need varying types, it is easy to see how a computer system can be weighed down

Velocity

Velocity takes into account the increasing rate at which data is transmitted through a network, and it has followed the same trend as the volume of data that is created each year.

In the past, our networking capabilities were quite primitive compared to the fast networks that we can develop today. These primitive networks restricted the way in which we could access data by restricting the speed at which we could access it. There have always been situations where a fast computer was needed to turn out and move data quickly and efficiently.

For example, the stock market servers must be ultra-efficient because of how frequently the stock prices and resources change.

It's not just where fast computing is needed, though; the way in which the gl us to access information from any corner of the globe has had a large impact handled. When you load a page on your smartphone, the data has to be colle network provider's mast. Imagine how many people are connecting to the m that is required – but people aren't satisfied with slow data connections and possible so the technology has had to be developed to enable this.

ADDRESSING THE ISSUES

When data banks become so large that they will no longer fit onto a single s
data must be distributed across a bank of computers. Using more than a sing
data bank requires specialist programming that is very complex and expensi
made to order for the data bank that is being processed. However, the functi
Section 12) can be used because it makes it easier to create and maintain co
be efficiently distributed, because it supports:

- *Immutable data structures* are data structures that are unchanging. Th
  used cannot be changed or altered, meaning that there are strict rule
  manipulated.
- *Statelessness* is inherent in functional programming – that is, the par
  of state; it doesn't remember the results or states of any preceding e
  instruction being used, as states can sometimes be restricting. For ex
  complicated if a thread enters an altered state because a core has m
  te the program could run over many threads.
- *High-order functions* are functions that can accept a function(s) as arg
  result. This allows the language to be highly adaptable for whatever

These features allow a programmer to produce the code to handle vast volu
mean that the code will be correct for its purpose.

Fact-based model

Generating a fact-based system is much like creating the Unified Modelling L
you've not come across UML before, it is a set of approaches for representin
functionality of a system in a way that can convey large systems in a relative
comprehensive manner. Similarly, fact-based systems are not concerned wit
dataset is structured, how the dataset is linked and how the dataset can be u
to handle vast quantities of data without needing to be concerned with how
system will only perform operations within the constraints of the facts that h

The databases that you're most likely to have experience with aren't very ef
large quantities of data due to how they are constructed. There is an efficien
and that is by using what is known as a graph database.

Graph databases

Graph databases offer the same functionality as a standard database *(see Se*
being an index for a field or an entity, the graph database uses a pointer to r
graphs are an abstraction of data that are related to each other or linked and
and properties; these three criteria are found in the database's *schema*. A sche
database is structured, how the data is stored and what constraints there are
which data can be stored.

Nodes

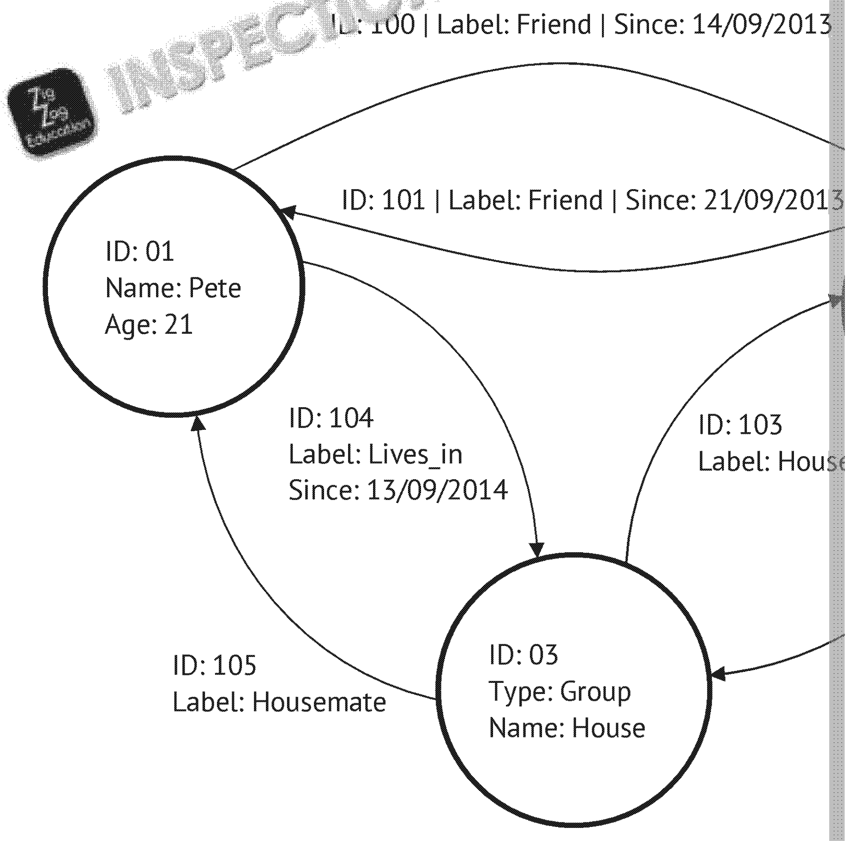
In a relational database the data of an instance of the attributes is stored in
records contain all the data values for the attributes in the data table and it i
these records and the data tables are linked to each other to form relationsh
the elements of data themselves are stored in what is known as a *node*. Each
that describe the object in its entirety.

Properties

In a relational database the properties of a record are assigned in the DDL a contribute to the database's structure. In graph databases the properties of a the *node* or the *edges* between nodes and are represented by text. The prope attached to, including the name, attributes and how the items are linked.

Edges

In a relational database the relationships between tables are shown in the *rel* how the links between the tables will provide integrity and how the data stor represented by the *arrows* between graph objects (nodes). In graph databases and instead the relationships between nodes have properties of their own an



Questions: Big Data

1 What is 'big data' and what three criteria can define it? (3 marks)

2 How does functional programming help to address issues posed by

12. Fundamentals of Functional Pro

Functional programming is called such because its primary and fundamental approach is t
In this section demonstrates how functional programming can be used to solve problems.

This section covers:

12.1 FUNCTIONAL PROGRAMMING PARADIGM

Just as with other paradigms, you can create a main body that you can pass p in functional programming the entire body of the program is a function that result which is returned. Generally speaking, the paradigm is one that encou complete computations by using functions as arguments for the function and result, without the explicit coding found in paradigms such as imperative or

There are programming languages you can learn that are only applicable to can just as easily use the programming language you've already learnt if it a most modern multi-paradigm languages do. You will be expected to show th simple programs in the paradigm to perform a simple task, which could be a in an array to writing a function that produces the average of a set of numbe this section will be written in *Haskell,* a *purely functional* programming langu

FUNCTION TYPE

Functions in all languages must have a type, often called their return type. W is how the function is mapping the input and creating a return. It is said that

f: A → B

This says that the function *f*, has a domain *A*, and a co-domain *B*. This shows argument for a function and the co-domain is your returned type. When the f type is A and the result type is B. There are some limitations, though; in orde B have to be subsets of objects of some data type.

In Haskell, a function's arguement and return type must be declared.

For example, if you wanted to write a function that added two numbers and the function *addNumbers*:

Haskell

```
addNumbers :: Double -> Double -> Double
addNumbers x y = x + y
```

This may seem slightly confusing, but we define the output as well. So in thi the answer is returned as a double!

FIRST-CLASS OBJECTS

First-class objects are actually a concept that occurs in many programming l you've probably already witnessed without noticing. First-class objects are o in an *expression*, be *assigned as an argument*, be *assigned as a variable* or be *r* programming languages first-class objects are integers, floating point, Boole

A first class function can return and take in functions. You will see an examp

FUNCTION APPLICATION

Unlike in traditional coding paradigms that you're probably used to, the way in programming is slightly different. In functional programming, to apply a functi applied to its arguments to produce the result. The process of giving specific va *application*. This is how the entire paradigm works; it is the application of functi will return a value. For example, if we had written *our addNumbers* function, bu

```
Haskell
addNumbers :: Integer -> Integer -> Integer
```

The function will continue to work as long as the data types of the input mat work for decimals any more as the function hasn't been built to handle decir argument as a pair of values: the function cannot work or be called without b addNumbers (4) would not work. To add 4 + 4 we would call the function ad this function is [illegible] pair.

PARTIAL FUNCTION APPLICATION

In functional programming, literally everything is a function, and like all func return a value. However, with the functional paradigm all functions actually while this sounds as comprehensive as the sound of one hand clapping it is a to handling data. What the function actually does is return a function that ac called *currying*. Take a look at the following example.

The function *myMult* takes two arguments: *x* and *y*.

```
myMult x y = x * y
```

The calling scheme of the function could be written as:

```
myMult :: Int -> Int -> Int
```

Partial functions can also be defined as functions whose result is only part o problems can occur and needs to be investigated. For example, the function domains of natural numbers ($\mathbb{N}$) and real ($\mathbb{R}$).

However, the function `halve x = x/2` is defined with the domain of rea

```
halve :: real -> real
```

However, it does not work with natural ($\mathbb{N}$) if the number is odd. So the don numbers would have to be defined using the domain ($2\mathbb{N}$), i.e. the group of

COMPOSITION OF FUNCTIONS

Functional composition is the act of combining two functions to produce a n aggregation of both in what appears to be a single task. This is something th patterns in mathematics and is something that you're taught to do from the b is a natural progression that you'll undergo as you learn the ins and outs of y in C# you can declare an array and then populate it, or you can do both steps more succinct. It is similar in functional programming to look for a quick way

Suppose you wanted to remove the top element from a list, and then reverse exist as funcitons, so using function composition we can do it in one go:

```
Haskell
reverse . tail [1,2,3,4,5]
output: [5,4,3,2]
```

Here you can see the dot notation functions together, and the result *reveserdSorted* which is then print

12.2 WRITING FUNCTIONAL PROGRAMS

FUNCTIONAL LANGUAGE PROGRAMS

Before you get too concerned, the level of your programming skills for functi
to be near the level of your imperative programming skills. In this section yo
knowledge of the functional programming paradigm and it will aim to give y
then build on. Make sure you read the code slowly and read the comments b

There are programming languages that were developed solely for the purpos
it is also a feature that is being introduced to many modern multi-paradigm

As mentioned before, one of the most important features of functional progr
function'. A function is *high-order* if it takes a function as an argument or retu
both. The three that you're expected to know are *map*, *filter* and *reduce* (*fold*).

Map

In its simplest form *map* is a function that accepts a list, *L*, as an input whose
A, and a function, *f*, which maps *A* to another type *B*. Map then applies *f* to ea
list of the results as type *B*. A very simple Haskell example can be seen belov

Haskell

```
square :: Integer -> Integer
square x = x^2
map square [1,3,5,7,9]
```

As you can see, first you define what the function, *square*, is doing and then i
producing more advanced programs the list would be passed in as a paramet
return of the function would be *1, 9, 25, 49, 81* as map applies the *square* fun

Filter

Filter uses a Boolean value called a *predicate,* and a list as inputs. The functi
the list and returns a list of all elements that satisfy the predicate. The predi
the element to be added to the newly created list.

Haskell

```
filter odd [1,2,3,4,5]
```

In Haskell, 'odd' is a function that returns true if a value is odd.

Reduce

Reduce, also known as fold, is a function that is a way of reducing an entire l
value, *B*, through built-in functions. It returns a single value as a list. A simpl
will reduce a list of numbers and produce the sum of the list. In Haskell ther
foldr and foldl, which act slightly differently. Foldr works by evaluating the r
then backtracking in a similar fashion to recursion. Foldl works more using it
(1+2), then adding 3, then adding four and so on!

Haskell

```
foldr (+) 1 [2,3,4,5]
-- (1+(2+(3+(4+5))))

foldl (+) 1 [2,3,4,5]
-- (((1+2)+3)+4)+5
```

12.3 LISTS IN FUNCTIONAL PROGRAMMING

LIST PROCESSING

When processing lists in a functional programming language, you can talk i written as Head: Tail. For example, the list [10, 9, 8, 7] has a head of '10' and written as 10: [9, 8, 7]. Similarly to how list structures are written in other la completely empty and is written as '[ ]'.

Functions of a list

Here are the functions which you need to know when working with lists. You instead of lists, as really they are just lists of characters!

Returning head of a list
This is similar to searching for a given index in a list. 'Head' is a keyword in H of a list is shown in the following:

Haskell

```
head [1,2,3,4,5]
output: 1
head "Hello"
output: 'H'
```

Returning the tail of a list
Almost identical to returning the head value of a list, this function returns th is shown in the following:

Haskell

```
tail [1,2,3,4,5]
output: [2,3,4,5]
tail "Hello"
output: ello
```

Test for an empty list
This is a very basic test that can be used to prevent underflow errors in a pro

Haskell

```
null [1,2,3,4,5]
output: False
null "Hello World!"
output: False
```

Return the length of a list
The length of a list can be returned using the *length* function as shown:

Haskell

```
length [1,2,3,4,5]
output: 5
length "Hello World!"
output: 12
```

Note the answer to length "Hello World!" as it includes the space between the two

Prepend an item to a list

In order to prepend (adding something to the beginning of a list) you must c then assign it to the list using colon notation. Take a look at the following e

```
Haskell
li = [1,2,3,4]
st = “Hello world!”

3:li
output: [3,1,2,3,4]

“I want to say” ++ st
output: I want to sayHello world!
```

Note that as the space was missing in the first list, the append adds directly to g uses the combining (see below) rather than prepend. For strings it is a little bit diff can only add one character at a time.

Append an item to a list

To append an item to the end of a list you use exactly the same principle as that we have used the same variable to hold the value of the new element.

```
Haskell
st = “Hello world”
st ++ “ out there!”
output: Hello world out there!
```

Combining lists

Just like text, we can add lists by using the ++ function so:

```
    [1,2,3] ++ [4,5,6]
```

Combines to

```
    [1,2,3,4,5,6]
```

Finding elements of a list

We can find an element of list using `!!`

```
    [1,2,3,4,5] !! 3
```

This will find the number 4. NB remember that list elements count from 0 so

```
    [1,2,3,4,5] !! 0
    output: 1
```

This also works with text:

```
Haskell
“Hello world!" !! 7
Output: 'w'

“Hello world!" !! 0
output: H
```

Mathematical functions on lists

You can build your own functions to work on lists, but basic calculations are

Add up a list	**Haskell**
	sum ourList Output: 15 sum ourStrList Output: Error

Multiply a list	**Haskell**
	product ourList Output: 120 product ourStrList Output: Error

Arrange smallest to largest	**Haskell**
	sort ourList Output: [1,2,3,4,5] sort ourStrList Output: " HWdellloor"

Note the space is capitals are befor

Reverse the order	**Haskell**
	reverse ourList Output: [5,4,3,2,1] reverse ourStrList Output: "dlroW olleH"

Find minimum value	**Haskell**
	minimum ourList Output: 1 minimum ourStrList Output: " "

Find maximum value	**Haskell**
	maximum ourList Output: 5 maximum ourStrList Output: "r"

Custom functions on lists involve function declarations.

Double every element	**Haskell**
	[x*2 | x <- ourList] Output: [2,4,6,4,10]

Square every element	**Haskell**
	[x*x | x <- ourList] Output: [1,4,9,16,25]

By using the above functions, it can be seen that by combining and working can be produced.

Recursion using lists

We can use the head and tail of a list to write functions that use recursion.

Suppose you wanted to double every element of a list without using the ma
could do it as above, or you could write a function that goes through each el

In Haskell we can do this using the following:

Haskell

```
double :: [Double] -> [Double]
double [] = []
double (x:xs) = 2*x : double xs
```

The recursion is done in line 3. (x:xs) splits the list into its head and its tail –
separately. What happens is that the head is doubled, and then the function
head of the first tail is the second element; the head of the second tail is the

Eventually we are left with a blank list, which in this case is the base case/te

Once the base case returns [], all of the doubled elements are added to it through

Calling the function we can see that it works correctly.

Haskell

```
double [1, 2, 3, 4]
Output: [2.0, 4.0, 6.0, 8.0]
```

13. Systematic Approach to Problem

This section is all about the skills needed to produce solutions to problems within teams systems are developed. In addition to being articulate and highly skilled, having strong p both essential requirements of all modern day computer scientists.

13.1 ASPECTS OF SOFTWARE DEVELOPMENT

Software development is not just the process of coding a piece of software to a specification; it actually encompasses the entire project. This covers *why* you are developing the software, *who* the client is and *what* it needs to do. The most commonly used model of software development is the *waterfall* model where decomposition leads to specific areas of development, where the result of one area is the input for the succeeding process. Before you can start software development you need a definition of:

- Project objectives – what the problem is
- Stakeholders – who is affected and has a say in important decisions
- Services and constraints – what the system needs to do and how to
- Expectations of 'quality' – how the system should function, i.e. ease
- Project time frame and budget – how long you have to finish and ho

Software development takes a great deal of time and the most important as intended users and client. Good communication means that a problem will b solution to meet those clearly defined objectives is more likely to be accepte

Prototyping

A prototype is a model of the new system to be developed. This model is the final production of the system is developed. It is used in industry a great dea identifying exactly what the user requires. By using prototypes the user gets can make comments before time-consuming mistakes can be made.

There are several methods of prototyping. Two such methods are:

- *Piloting* – using a prototype to test the feasibility of a design propos
- *Modelling* – built to develop a deeper understanding of the user requ

The above are *throwaway prototypes*, i.e. once they have achieved their purpo the knowledge found. However, sometimes in development *evolutionary prot* prototype is actually the system under development and is a step closer each

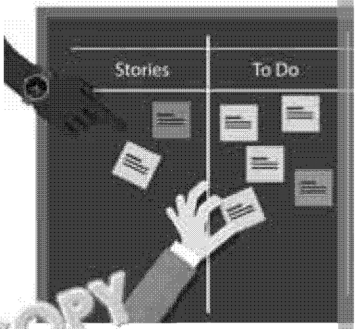
In your system development, prototyping methods can be used to an advant new idea, instead of using the whole program and trying to solve a particula problem part you need and developing a separate test prototype that just d identifying the exact solution. The advantages of this method are:

1. It allows you to concentrate on just the problem itself, rather than b
2. Other areas of the project/system do not affect the functionality of t developing. (For example, a faulty validation routine may stop the c causes the new feature to fail.) Setting up a separate system with va goes wrong it must be the feature you are developing.
3. If your new feature fails catastrophically (for example, an update fea database data) you do not have to retype/redo any more work, as th
4. When complete, introducing into the actual system should be relativ are based on the communication between the system and your new 'copy and paste' straight into your system and then link within the c

Agile software development

In larger projects the agile software development management is an approa

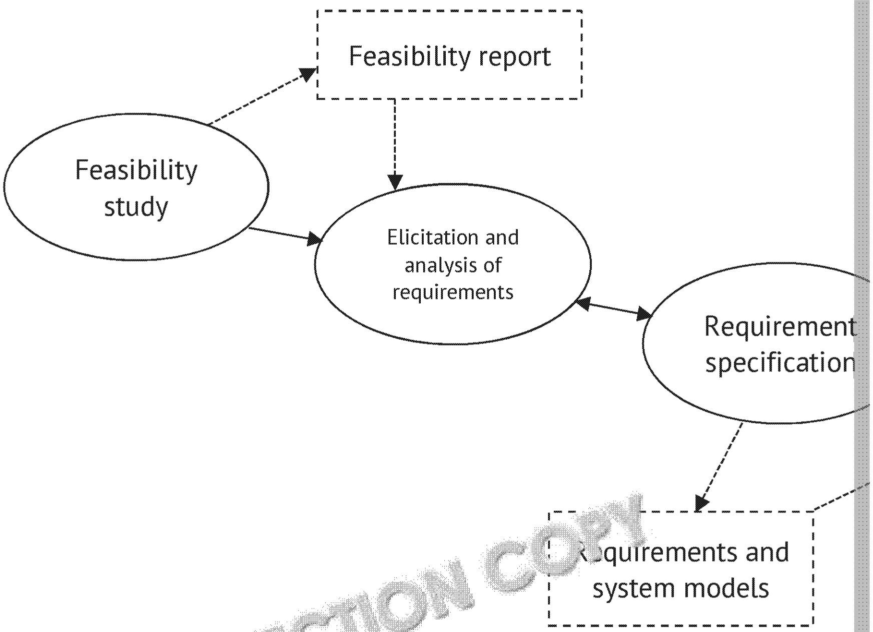
This methodology is to develop the solution through teams. These teams collaborate but focus on particular areas to evolve the project. The solutions produced are used by the client and then actively evolved into a better solution based on feedback. This means that the client sees progress sooner and is able to provide adaptive improvements to allow the project to finally meet its objectives.



ANALYSIS

Analysis of software development is also called *requirement engineering* and is th will be [illegible]ed. Before any work can be done, the requirements for the solutio model. The system requirements must be defined as well, as these will be what function fully and interact with the end user. Two of the most common causes fo

1. Poorly defined requirements – a requirement is poorly defined if it is v
2. Poorly managed requirements– if requirements are not managed co the software can keep changing or being added to by the end client;


Feasibility report
Feasibility study
Elicitation and analysis of requirements
Requirement specification
Requirements and system models

The *feasibility report* is the examination of whether the system is achievable the constraints of the requirements.

Requirement elicitation is the method you used to gather the requirements, i.e interviews with end users, observation of how the current system works and

Requirement analysis is using business scenarios or business models to derive elicited requirements into their requirement types before the requirements a *specification* to make them precise and detailed.

Requirement type	Meaning
Functional	Functional requirements are statements of services t how it should react to certain inputs.
Non-functional	Non-functional requirements are constraints on serv performance, usability and efficiency. This also cove requirements.
Quality	'Quality' can be an intangible property but it covers v adhere to standards and the expectations of the syst
Domain	Domain requirements are the requirements for the s environment, e.g. files being limited to staff with cle
Interoperability	These requirements are used for when the new syste using the pre-existing service.
C.R.U.D.	C.R.U.D. – Create, Read, Update and Delete. These re to be handled by the system.

Requirement validation is the final stage and produces the requirements repo *design*. Validating requirements is important to make sure that all requireme realistic. This can be accomplished by running a scenario walkthrough to ma covered, prototyping to find requirements that have been missed during the a technique that uses a set of conditions to test whether or not a system wil

DESIGN

Using the requirements report from the previous stage you can create what i design model is similar to an architect's blueprint – it is a model of somethi allows you to assess the eventual product without building it to test for requ

This stage is where the developers look at different aspects of the proposed need to interact for the system to function correctly. Decomposition allows t into *modules* which act as a way of segmenting the full build so that it is mo or *patches*, use modular design to correct mistakes or inefficiencies found in down database designs are created if the new system needs a database. Fina create pseudocode for the developers to produce the working code. This wil using prototyping to quickly produce the algorithms that will be robust whe

The design is validated by checking the proposed design against the require the requirements of the system, the needs of the user and the 'quality' the c final design and is the input for the next section, *Implementation*.

Design can be developed using the prototyping and agile methods as descri

IMPLEMENTATION

At this stage, the flesh is put on the bones of the design. This will involve ta stage to having it installed for the customer, and the users trained. Steps tha to be possible include:

- Installing software and hardware
- Creating the correct data files
- Properly documenting the system and providing training

This stage can be made easier by the effective use of a *CASE* tool.

CASE (computer-aided software engineering) tools

CASE tools are used to assist in the development of a design into a working s

Fourth-generation languages This often makes use of concepts such as c The actual machine code they produce ma nature of such languages is such that the s tailored to meet the user's requirements.

Interface generators Such as that contained in Visual Basic, whi form's dialogue (menus, textboxes, button write any code.

Code generation facilities To automatically create some source code. for Applications) systems are written to ru Microsoft Word or Excel. Simple operatio copying chunks of text can be coded autor 'Record', and performs the operation he or system. The operation is translated into co a keyboard shortcut, or assigned to a butto code can be created in this way; for examp does not allow you to produce iterative or

Data dictionary To record the details of the data in the sys in database systems.

Project management tools Software such as *PERT (Program Evaluation* with scheduling.

Design tools Such as *desktop publishing (DTP)* packages,

Report generator Used to automatically create documentati

Implementation in agile development methodology would be through the ite important part (critical path) solution would be deployed before the rest of t

Acceptance testing

Acceptance testing is testing to make sure that the system matches the user' whether the system actually works in practice, and on whether any changes required. In order for acceptance testing to be useful, the user should be all so that any bugs can be identified, not just given a 'walk-through' that might must also verify that the interface and 'feel' of the program is right.

Unit testing

Each module in the system is tested to make sure that it functions correctly. modules) is tested.

Integration testing

The integration test makes sure that all the units of the system work togethe system will function correctly in themselves, but the whole system must The system must be subjected to:

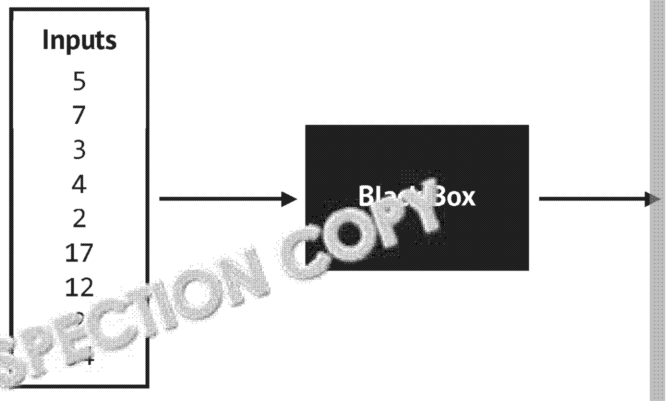
- *Functional tests* to make sure that every aspect of the system functio
- *Performance tests* to make sure that the system can fulfil its role in a
- *Recovery tests* to make sure that the system can recover from various

Black- and white-box testing

These two techniques are aimed at testing the inputs and expected outputs structure of the algorithm. If both these tests are passed, then an assumptio parts of the system are behaving as expected.

In *black-box testing* the program itself is completely ignored, hence the anal take a set of inputs with known outputs, put them into the system and comp expected results.

For example, consider the following diagram, where the black box represent doubling the output:



This system appears to be working correctly so would pass the black-box tes

White-box testing aims to consider all of the possible paths through the algori operating as expected. This is done by using the source code being tested to of movements through the algorithm. The result of this should be a graph tha during the design phase. Also, it should be obvious where possible problems are parts of the graph that you cannot get out of once you have entered and impossible to get to, then there is obviously a problem with the code. There the flow diagrams for the white-box testing to avoid having to devote numer

Dry-run testing

A *dry run* is working through a program manually, i.e. on paper. This is used (e.g. it crashes or produces a wrong answer), and so the programmer must w line trying to find out where the problem is. Incidentally, never say in an ex through the code to find problems; always say instead that they will do dry- done using trace tables which are covered in *Section 4.1*.

Choice of test data

A wide range of test data must be entered into the system, in order to test it

- Valid normal data – data of the sort that will be entered into the system
- Valid boundary data – data that will occur only rarely, but which the syste
- Standard incorrect data – this could be data that is only slightly wro but is not valid information – to observe the effects on the system o
- Standard invalid data – data that should not be entered into the system (
- Extreme invalid/incorrect data

Test plan

Draw up tables containing your test data. There are several areas that you a of your testing:

- Validation data for all input.
- Data for individual modules (functions and procedures); these may b of the system – perhaps at the coding stage.
- Sets of input data – to check each module gives a correct set of outp
- Whole-system sets of data (e.g. a day's/week's/month's/year's set of

The test plan is usually structured using a table, with the following columns:

- Test number (for later reference)
- Test title (to indicate what is being tested)
- Explanation if necessary of what is being tested
- The test data itself
- The expected result(s) of the test, including where appropriate what output from the test
- The actual result after testing took place – just a yes/no won't do

Justification of test data

You must provide full and detailed justification of why you picked all the tes did, and why that is the entire test data you used.

Retesting

Where tests fail (and some must), you need to explain what was wrong, what retest.

If none of your tests failed, there are two possible reasons:

1. There are no errors anywhere in your system (do you really believe t
2. You have failed to test your entire system thoroughly enough and ha

EVALUATION

Evaluation is carried out for several reasons. It may be carried out by the pr that were drawn up at the analysis stage to check that the program does wh by the company purchasing the system before they pay the programmers; ag criteria. Some organisations will bring in an independent evaluation team t

The evaluation/appraisal may include the following sections:

- Comparison of system against original objectives
- Feedback from actual user
- Improvements needed – how they could be incorporated
- How effective the solution is
- Other possible future developments

Evaluation criteria

The exact criteria by which a project must be assessed will vary from system general guidelines for evaluation of a system:

- Whether the system fulfils the user's needs to a satisfactory degree, i terms of function, interface, reliability, efficiency, etc.
- The incidence of system's failure, whether catastrophic or merely irr
- The amount and cost of maintenance required
- The cost of the system, compared to what was predicted
- The timescale of completion, compared to what was originally proje

It is also important to realise that it may often be necessary to restart or imp cycle before you can continue. For example, during the design it may becom understood about the current procedure, so further interviews or questionna Also, as a result of testing, the whole life cycle may be restarted if it appears However, provided that the design stage was performed well enough this sh

Software maintanence

The evaluation stage may uncover problems in the system, potential for imp not mean going back to the problem definition stage and beginning the who changes are needed, rather than a major overhaul, we call this *software main*

Corrective maintenance

This is where the system has an error that needs to be corrected.

For example, the system may have been tested fully and function correctly b reports. Equally, the system may fail in a situation that was not expected by

Corrective maintenance is often expensive to both the company and the dev are hard to trace.

Perfective maintenance

This is where the system functions satisfactorily but improvements to the sy

For example, a screen layout may not present the information in the best wa particular area where a change to the system could improve the delay.

The above maintenance methods would be performed after a short period of period there would be:

Adaptive maintenance

As the organisation expends it may be necessary to alter the system to meet

For example, a single-user system could be adapted to a multi-user system, a change to the program is necessary (for example, tax changes, or hardware

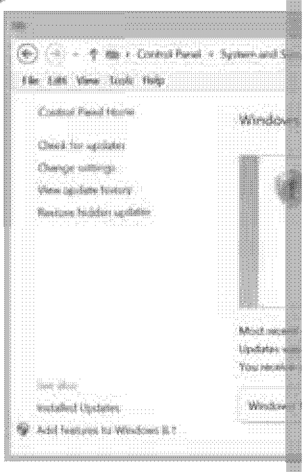
Sometimes this maintenance may lead to a whole new project and the life cy

Modifying systems and patches

The maintenance on systems that have a limited number of users/clients is d team/company.

For systems that are for a larger market it may not be possible for the company to correct all versions directly so the maintenance improvements are often released as programs called patches or service packs. *Patches* are usually for corrective maintenance and *service packs* for perfective/adaptive maintenance.

These programs are run by the user and perform the alterations through a series of instructions (often changing settings, or replacing a program or file).



Programming Challen

Now that you have covered all of the basic theory of programming, you can b
on your understanding and build your confidence. Here are six programming
have been devised for you. They start at a basic level and increase in comple
so it's expected that you may find the latter ones particularly challenging.
The ✱ symbol next to each challenge title indicates the relative difficulty of
that challenge.

There are no skills in this section that you have not covered. If you are
struggling, refer back to the programming notes in this resource, and break th
tasks down into manageable processes. Even if you feel you have a good
understanding of programming, it is recommended that you start at the begin
progress through all tasks.

You should try using [illegible] code in all of your solutions where it is s

1. TO THE POWER OF…

String manipulation, casting, arithmetic operations

Produce the code that asks the user for two inputs. Without using built-in fu
the power of the second and return the result to the screen.

2. CONTINUOUS DIVISION

Basic subroutines with parameters, selection, arithmetic operations

Using procedural programming, produce the code that will continuously hal
result. When the program reaches the number 1 the program should termina

Extension:

Modify your code so that it uses a single subroutine that does the division a
the user for an input and pass the value as a parameter.

3. GUESSING GAME

Random number generation, iteration, subroutines, reading inputs from keyboa

Produce a game where the user has 10 attempts to guess the random numb
attempt the program should say whether the guessed value was too high or

Extension:

Extend your code so that the user can set the parameters of the game to ma
should be able to set the number of guesses they're allowed and the range o
even s[illegible] feedback to give them a hint as to how close they are to the ra

4. CASE SELECTION

Selection, arithmetic operations, reading inputs from keyboard, subroutines

A hotel needs a new booking system for their rooms. The price of a single ni culmination of the room type and board basis.

Room type		Board	
Single (1):	£50	Self-catering:	£0
Double (2):	£40	Half-board:	£10
Family (4):	£30	Full board:	£20

They also offer a discount for stays over a week. The discount is 25% for ever week long. The result should be the total and should show how much has be

You should try to break these tasks down into separate subroutines!

5. WRITING TO & READING FROM FILES

Iteration, selection, casting, arithmetic operations, subroutines, arrays, Boolean file input/output

You have been tasked to write three procedures to be used for a leaderboar

- LoadLeaderBoard( ) – This *function* should load the values from a file an array which will then be returned. *You may find it easier to cast the for ease of use later.*
- PrintLeaderBoard( ) – This *subroutine* should be passed the leaderboa
- SaveLeaderBoard( ) – This *subroutine* should be passed the leader boa contents back to the file.

Extension:

Write an additional procedure called CompareScores( ) which prompts the us The program should then iterate through the array from the file to see wher the leaderboard. If the score should appear in the leaderboard all other scor the new score should be input.

6. MAGIC SQUARES

Complex iterations, 2D arrays, Boolean operations, arithmetic operations, date/t

Magic squares are a mathematical phenomenon where all the values in rows given number. Using a number random generator (between 1 and 10) and a square generator where all the rows and columns sum up to 15. Output the values of the

For this task you may find it useful to copy each row and column of the array the comparison to make it easier.

Note: True magic squares take the diagonals into consideration. For this task yo diagonals but you can if you want an extra challenge.

Extension:

Research how to produce a stopwatch timer using the system clock (*hint: Sy* you print the array values the time taken is also printed; you may be surpris

Assembly Programming Ch

Below is a series of problems to solve. The problems get more and more diffi

1. *Input two numbers and add them together; output the result.*

2. *Input two numbers and subtract them; output the result.*

3. *Write a program that counts backwards from 10.*

4. *Write a counter that counts up to 10.*

5. *Input a number and output its times table up to 10 times.*

6. *Write a counter that counts how many times I enter a non-zero numb*

7. *Write a program which adds up any list of numbers (by looping until many numbers I entered.*

8. *Take in two numbers and output them in order (smallest first).*

9. *Take in a number and divide it by 2 (clue: repeated subtraction, coun*

10. *Input two numbers and find the average (add together and divide by*

11. *Write a program that finds any number entered (a) divided by anothe (whole-number part only).*

12. *Write a program that finds the square root of a number entered.*

Question Solutions

SECTION 1

Data Types

Question 1

a) String
b) Boolean
c) Float/Double/Real
d) Integer
e) Character/
f) Date/Time

Question 2

a) String – variable must contain multiple characters
d) Boolean – the account will either have an overdraft or not
b) Float – values must be highly accurate to avoid rounding errors and increase consi
e) Date – preferably 'short' date so as not to include time
c) Character – only needs to contain a single letter (M or F)
f) Integer – sort code contains numbers only and is within the size boundary for a si

Programming Concepts

Question 1

a) Variable Declaration
b) Variable Declaration
c) Variable Declaration
d) Assignment
e) Assignme
f) Assignme
g) Relationa
h) Arithmeti

Question 2

```
If Score > PassBoundary
    OUTPUT PASS
    Select Case gradeCalc
        Score – Pass Boundary >= 30
            Pass = TRUE
            Action ("Grade is A")
        Score – Pass Boundary >= 20
            Pass = TRUE
            Action ("Grade is B")
        Score – Pass Boundary >= 10
            Pass = TRUE
            Action ("Grade is C")
        Score – Pass Boundary >= 0
            Pass = TRUE
            Action ("Grade is D")
    End Select
Else
    Pass = FALSE
End If
End Procedure
```

Pseudocode is structured

- CASE select nested in

Arithmetic operation for g

- must be score minus p
- grade boundaries mus
- correct use of equal to

Set Pass to true and outp

- allow for OUTPUT("Gr

Arithmetic Operations

Question 1

a) 17 DIV 8 = 2

c) ((16 DIV 2) * (6 MOD 4)) = 16

b) 90 MOD 16 = 10

d) 26 MOD 2 = 0

Question 2

```
Var1 ← Value
Var2 ← Value

IF Var1 MOD Var2 == 0 Then
    OUTPUT("No remainder!")
Else
    OUTPUT("There is a remainder of " & Var1 MOD Var2)
```

IF statement use

- only accept th
  from user if a

Correct use of M

Relational Operations

Question 1

a) False

b) True

c) True

Question 2

```
Var1 ← value
Var2 ← value

IF Var1 > Var2 Then
    OUTPUT(var1 & "is bigger")
Else If Var1 < Var2 Then
    OUTPUT(Var2 & "is bigger")
Else
    OUTPUT("The numbers are equal")
```

Correct use of both greate

Correct outputs for all 3 r

Boolean Operations

Question 1

a) False

b) True

c) False

String Handling

Question 1

a) [illegible]

b) 656.34

c) 11

d) 8

Question 2

```
Var1 ← INPUT
ConvertToInt(Var1)
OUTPUT (Var1 * Var1)
```

Conversion to integ

Correctly produces s

Random Number Generation

Task

```
isFound ← False
rand ← New Random (1, 10)

WHILE isFound = False
OUTPUT ("Enter a number between 1 and 10: ")
userGuess ← INPUT
IF userGuess ==rand Then
    isFound = True
    OUTPUT ("Correct the number was " & userGuess)
End While
```

Exception Handling

Task

```
Var1 ← Input
Var2 ← Input
Result ← 0

Try
Var1 ← Convert to Integer
Var2 ← Convert to Integer
Result ← Var1 / Var2
OUTPUT (Result)

Catch Exception
        IF Var2 == 0
          OUTPUT ("You cannot divide by 0")
    END IF
End Try
```

Subroutines

Question 1

Takes the passed value and iterates through the values of 1 to 4 and sums the answe variable minus the number of the loop. This produces a number called a factorial.

Question 2

3! = 3 * 2 * 1 = 6

Procedures, Functions and Variables

Question 1

The difference between a function and a subroutine is that a function returns a be used to perform arbitrary tasks that do not necessarily require an output.

Question 2

Regardless of the programming language, an error message will be produced either o which would result in the program failing to compile or crashing.

Question 3

It is considered bad practice because global variables are assigned memory at compil program has closed.

Recursive Techniques

Task

```
OUTPUT MyFib(a)

FUNCTION MyFib (x) # number up to F_x
        A ← 0
        B ← 1
        IF counter >= x then
          Return a
        ELSE
          Temp ← a
                B ← temp + b
          MyFib(a)
        END IF
END FUNC
```

Question 1

a) The code asks the user for an input and sums all inputs till sum reaches 100.
b) Yes – i*0 is always true but i*0 is always 0 so sum never reaches 100; the escape

Question 2

Returns the address for 'sum' which is continuously stored to stack after each call.

Object-oriented Programming

TASK

Account.Class

```
CLASS STRUCTURE Account
    Private fullName
    Private balance

    # Default constructor
    STRUCTURE Account ( )
        fullName ← "no name"
        balance ← 0
    END STRUCTURE

    # partially initialised constructor
    STRUCTURE Account (name)
        fullName ← Name
        balance ← 0
    END STRUCTURE

    # fully initialised constructor
    STRUCTURE Account (name, currentBalance)
        fullName ← Name
        balance ← currentBalance
    END STRUCTURE

    # Class subroutine to an account
    PROCEDURE MyDeposit ( )
        Balance ← balance + 10
        OUTPUT deposit
    END PROCEDURE
```

Generator.Class

```
# Declare 3 new
Account Account
Account Account
Account Account

# generate 3 acc
accountOne ← n
accountTwo ← n
accountThree ←

accountThree.My
```

SECTION 2

Arrays

Question 1

a) 11

b) 32

Question 2

a) [0,3] and [3,0]

b) [0,1] and [1,0]

c) By storing [2,3] to a temporary variable and setting [2,3] = [3,3] and [3,3] equal to

Reading and Writing Files

Task

```
Total ← 0
WHILE input != -1
        PRINT "Total is: " + total
        PRINT "Enter new integer: "
        inputString ← ReadInput
        total = total + inputString

        fileWriter [fileName]
        writeToFile[inputString]
END WHILE
fileWrite.Close
```

Queues

Question 1

A queue – items are added at one end and removed at the other, meaning the video

Question 2

a) Head and Tail

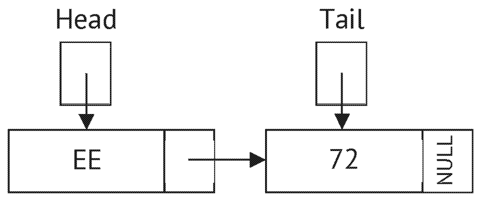
b) Head == Tail

c) Head == Tail + 1 OR Head = 0 Tail = max

d)
```
PROCEDURE push(new_item)
        IF (tail + 1 = head) OR (tail = 10 AND head = 1) THEN
                PRINT("Queue is full!")
        ELSE
                IF tail = 10 THEN
                        tail = 1
                ELSE
                        tail = tail + 1
                END IF
        data(tail) = new_item
        END IF
END PROC
```

Question 3

a)



b) Head and tail both equal NULL; both would need to be changed when an item is remove an item, only one needs to be checked for NULL.

c)
```
PROCEDURE push(newItem)
    IF head = NULL
        head = new element()
        tail = head
    ELSE
        head.next = new element()
        head = head.next
    END IF
    head.next = NULL
    head.contents = newItem
END PROCEDURE
```

Stacks

Question 1

Last In, First Out (LIFO)

Question 2

a) 89, 45, 77, 56

b)
```
FUNCTION add(stack) RETURNS INTEGER
    total = 0
    WHILE stack is not empty
        total = total + stack.pop()
    END WHILE
RETURN total
```

Question 3

a) Billy

b) Check stack is not full, increment top of stack, insert element at top of stack

c) −1 (adding to stack when empty is exactly the same as when not empty)

Graphs

Question 1

a)

Vertex	Connected to
A	C, D, E
B	A, F
C	B, F
D	
E	B
F	B

b)

	A	B	C	D	E	F
A	0	0	1	1	1	0
B	1	0	0	0	0	1
C	0	1	0	0	0	1
D	0	0	0	0	0	0
E	0	1	0	0	0	0
F	0	1	0	0	0	0

c) Adjacency list – it uses less memory resources than using a matrix.

Trees

Question 1

The tree would lo[illegible] with every node being off the left pointer of the one be

Question 2

O(n)

Question 3

The simplest way is to add all the parentless items back onto the tree, ignoring their

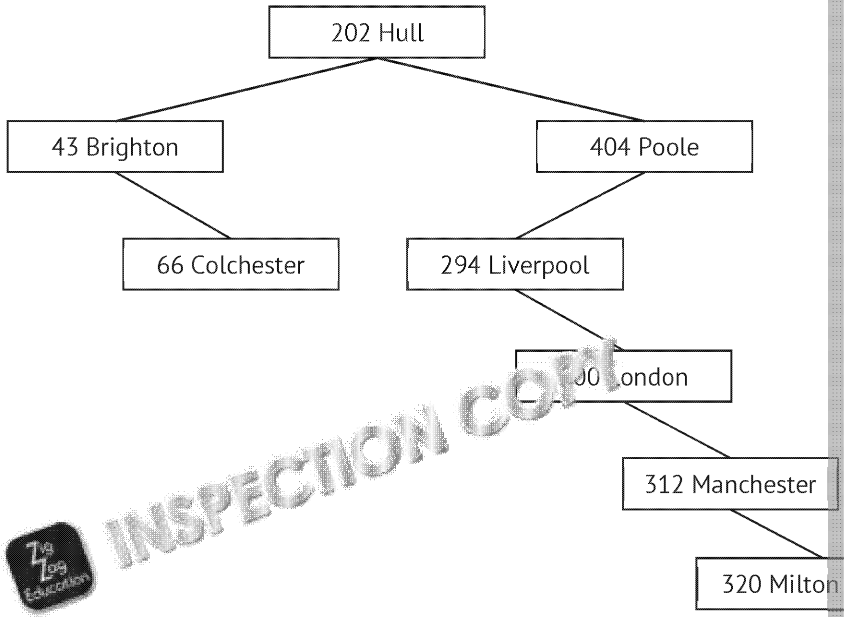
Hash Tables

Question 1

Allows records to be found through an index which can be generated from the index

Question 2

a) + b)



c) Collision

d) A collision resolution strategy – e.g. linked list from each node

Vectors

Question 1

Scaling

Question 2

$\begin{bmatrix} 3k \\ 0 \end{bmatrix}$

Question 3

$A = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$ $C = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$ Therefore, $A + C = \begin{bmatrix} 3 \\ 1 \end{bmatrix} + \begin{bmatrix} 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$

SECTION 3

Graph Traversal

Question 1

a) 1, 3, 5, 2, 4, 7, 6

b) 1, 3, 5, 2, 4, 6, 7

Tree Traversal

Question 1

a) 0, −1, −2, −5, −7, −6, −4, −3, 1, 2, 5, 3, 4, 6, 7

b) −7, −6, −5, −4, −3, −2, −1, 0, 1, 2, 3, 4, 5, 6, 7

c) −6, −7, −3, −4, −5, −2, −1, 0, 4, 3, 7, 6, 5, 2, 1

Reverse Polish Notation

Question 1

RPN removes the need for brackets as computers do not understand how to use them

Question 2

a) 56 / 5 +

b) 77 * 625++

c) 457/06-+

Question 3

a) a*b

b) g + h − b

c) b/m + g/h

Searching Algorithms

Question 1

a) A binary search cannot be performed because the data items are not sorted into a

b)

1	2	3	4	6	8	9

It would take two passes to find the number 2.

Question 2

Index	1	2	3	4	5	6	7	8	9	10
Data	A	C	E	J	L	O	Q	R	Y	Z
	L				M					R

Index	1	2	3	4	5	6	7	8	9	10
Data	A	C	E	J	L	O	Q	R	Y	Z
						L		M		R

R is found in index 8.

Shortest Path Algorithm

Question 1

Shortest path (path of least cost):

$\vec{a}c = 1$

$\vec{c}b = 2$

$\vec{b}e = 3$

$\vec{e}d = 1$

$\vec{d}f = 3$

$\vec{f}g = 4$

$\vec{g}h = 3$

$\vec{a}h$ = 17

a, c, b, e, d, f, g, h

SECTION 4

Finite-state Machines

Question 1

a) Input alphabet = {open door, close door, timeout}

b) Output alphabet = {light on, light off, alarm on, alarm off}

c)

Current State	Input	Next State	Output
S1	open door	S2	light on
S2	close door	S1	light off
S2	timeout	S3	alarm on
S3	close door	S1	alarm off, light off

Question 2

Current State	Input	Next State	Output
S1	sensor triggered	S2	open doors, reset timer
S2	sensor triggered	S1	reset timer
S2	timeout	S3	close doors

Regular Languages and Expression Notation

Question 1

Intersection can be applied to check what numbers appear in both and they are: 6 an

Question 2

2, 3, 5, 10

Question 3

a) ^[a-z

b) False – all letters of the English language appear in the sentence

Context-free Languages

Question 1

a) False

b) False

c) [illegible]

d) [illegible]

Classification of Algorithms

Question 1

(n +1) + n + (n + 1) + n = 4n + 2

Question 2

a) Polynomial

b) Exponential

c) Linear

Question 3

Its order of complexity is calculated from the worst-case scenario run-time.

Question 4

An algorithm with a worse space complexity manages system resources poorly and w

A Model of Computation

Question 1

a) Turing machines are still used to this day because they allow the study of what is modern computers.

b) Modern computers operate on a similar principle: programs are stored in the sam a universal Turing machine.

c) No – it also has a 'state of mind' which defines what it should do when given a pa

Question 2

a) [illegible]gh

iii) Yes

b) The Turing machine enters into an infinite loop.

c) It will add two numbers together.

SECTION 5

Number Systems

Question 1

Natural numbers are all integers without negatives, meaning they can all be used in t counting numbers; whereas real numbers encompass all the types of sets including d amount of memory due to their value of the accuracy.

Number Bases

Question 1

a) 26_{10} → 00011000_2
b) 100_{10} → 01100100_2 → 64_{16}
c) $7C_{16}$ → 1111010_2 → 122_{10}
d) 01001001_2 → 73
e) 188_{10} → 10111100_2 → BC_{16}
f) [illegible] → 11010011_2 → 201_{10}

Question 2

The highest value that can be stored in a single byte is 256 (2^8). In order to store anythin

Units of Information

Question 1

512 bits

Question 2

2^{32} = 4,294,967,296
(four billion, two hundred and ninety-four thousand, nine hundred and sixty-seven th

Question 3

2^{40} = 1.0995116e+12 or 1,099,511,600,000 (one trillion, ninety-nine billion, five hundre

Binary Number Systems

Question 1

a) 00111100_2
b) 00111000_2
c) 10001001_2
d) 01011010_2

Question 2

a) 00100011_2
b) 00001100_2

Question 3

N[illegible] lowest value is −128 (10000000_2)

Information Coding Systems

Question 1

1

Question 2

6

Question 3

000 111 000 111 000 000 000 111

Bitmapped Graphics

Question 1

$7 * 7 = 49$ inches^2

$80^2 = 6400$ pixels per inch

$6400 * 49 = 313{,}600$ pixels in total

4 bits required to store the 8 colours

File size = $313{,}600 * 4 = (\frac{1{,}254{,}400}{8000})$ = 156 kilobytes

Respresenting Sound

Question 1

$(8000 * 16) * 30 = (\frac{3\,840\,000}{8000})$ = 480 kilobytes

Question 2

Sample rate = $(\frac{\text{file size / bits}}{\text{length}}) = (\frac{\frac{100{,}000}{10}}{10})$ = 1,000 Hz

Question 3

Because no matter how many bits you have to represent the value of the pulse peaks be lost which leads to the staircase effect.

Data Compression and Encryption

Question 1

Lossy compression identifies seemingly redundant data and removes it from the file.

Question 2

'Computer Science'

Question 3

It needs to be at least the same length because otherwise the modulo division no lon of the message not encrypted. If the key is longer it just means that part of the key be

Question 4

Yes – the cipher effectively works as a one-time key. As the keys are never reused, it been shared it would always be impervious to all attempts to break the key values.

SECTION 6

Hardware and software

Question 1

a) Utility software

b) Operating system software

c) Translator software

d) [illegible] program

Question 2

Abstraction is created by the operating acting as a virtual machine where the comple is hidden from them.

Question 3

The main drawback is that the code is not checked while it is being programmed. It is errors are found.

Classification of Programming Languages

Question 1

a) The code reads the passed value and compares it to 0 – if the value is not 0 then If the value is less than 0, then it is set to −1; if it is greater than 0 then the value value of sign is then returned.

b) The output would be 1.

Question 2

<u>Assembly language</u>	<u>Machine language</u>
Uses mnemonics	Comprised entirely of 1s and 0s or hexade
Some abstraction	No abstraction
Uses assembler	Doesn't need translating
Easier to program in	Harder to program in
Runs more slowly due to translation	Runs faster

Question 3

Yes – although all compilers are different, the high-level languages are more portabl assignment of memory is automated by the computer. As long as the compiler can un was coded in, the program will compile on any computer.

Logic Gates

Question 1

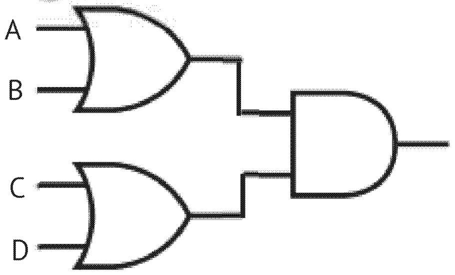
a) True b) False c) False

Question 2

Input A	Input B	Input C	Input D	Output
0	0	0	1	0
0	0	1	0	0
0	1	0	0	1
1	0	0	0	1
1	1	1	1	0
1	1	1	0	1
1	1	0	1	1
1	0	1	1	0
0	1	1	1	0
1	1	0	0	1
1	0	0	1	1
0	0	1	1	0
0	1	1	0	1
0	1	0	1	1
1	0	1	0	1
0	0	0	0	0

1 = true, 0 = false

Question 3



Boolean Algebra

Question 1

a) $A + (A \cdot B) = A$

b) $A \cdot B + A \cdot \overline{B} = A$

c) $(A + A \cdot B) + (A + A \cdot B) = A$

d) $\overline{A} + \overline{B} = \overline{(A \cdot B)}$

e) $\overline{(\overline{A} + \overline{B})} + B = B$

SECTION 7

Internal hardware components of a computer

Question 1

The address bus holds the value of the memory address being indexed. If a valid men is [illegible] then data is permitted to be conveyed via the data bus. The flow of data is c

Question 2

RAM stands for Random Access Memory. It is used by the system as a storage mediun current tasks, used by the operating system and used to store values during run-time after current is lost from the board all data that was stored within the memory modul

Question 3

a) Motherboard
b) Cache memory
c) CPU

Structure and the Role of the Processor

Question 1

Arithmetic logic unit (ALU)

Question 2

The contents of the program counter are transferred to the memory address buffer, w information into the memory buffer register (or memory data register). The informatio This is called the fetch part or phase of the cycle.

The control unit then splits the information into two parts: the operation code and th computer which instruction to perform. The control unit then switches the processor action. This is called the decode phase.

The processor then transfers the data (operand) to the appropriate part of the processor then the operand would be transferred to the MAR and used appropriately, otherwise it processor (for calculations or comparisons, for example). This final phase is called the e

Question 3

a) The computer is constantly running background tasks/services that ensure the co services that connect to time servers to update the time allowing you to connect [illegible]s OK because the machine is always running the OS which is running the ta [illegible]ntly being exchanged.
b) Yes – an interrupt is sent to the CPU; the need to load an application is of higher undergone while the computer is idle. Once the interrupt is serviced and the appl other background tasks.

Question 4

a) False
b) True
c) False
d) True

SECTION 8

Computing Issues

Question 1

A company must consider that constant monitoring of its employees may be considere
employees. The employees themselves may feel that this monitoring is intrusive to the
employer. They may also feel resentful towards this practice. Another aspect the compa
of the call is monitored it is possible that employees may give a poorer service to custo
of calls.

Question 2

Pirated software is stealing software, and just like any other product or service the m
people that have created the software. Financially, the piracy industry removes a larg
consequently the prices may have to be raised to cover their costs.

Equally, the quality of the software is reduced as pirates will often alter software to b
auth[illegible] he[illegible] piracy. These altered or 'cracked' programs often contain mista
rel[illegible] robust.

Question 3

Often government agencies will use data to track and monitor indivuduals in society. Th
national security. However with the use of drones over highly populated areas, the gove
individual rather than suspects. The 'big brother' effect is that the government would be
suspects. Many would consider this an invasion of privacy and be concerned about how s
it is used.

Question 4

People are spending more time stationary and performing repetitive actions.

RSI – Repetative strain injury occurs with people that spend a lot of time typing on c
that performed repetitive tasks in factories.

Eye strain is becoming more of a proble with computers which previously was not co
environments in which they would not focus on a fixed point for many hours (comput

Back ache has issues in common with industrial accidents in which people were requ
(accountants/book keeping). Mobile devices such as laptops make it difficult to be se

Stress has increased with the use of computers as information is required quicker an
industries 50 years ago. A computer failure in modern age can be life threatening an
information often suffer from stress.

SECTION 9

Communication

Question 1

Serial transmission can operate either in synchronous mode (with a mutual clock) or a

Question 2

a) False
b) False
c) True

Question 3

Control bus

Networking

Question 1

No performance degradation

Secure communication – communication is direct therefore no eavesdropping is pres

Scalable – easy to add new nodes

Question 2

Wireless network adapter

Question 3

Method	Application
Hiding the network SSID	Keeps network hidden and secure to all but those wh
MAC address white listing	Even if someone does manage to gain knowledge of y white list prevents all but those on the list from con
Wi-Fi Protected Acce	Prevents people connecting to the network if they do packets being sent that bluff the passphrase to gain a

The Internet

Question 1

The payload is the volume of data that is being transmitted – the useful part of the p

Question 2

Packet switching is the digital equivalent of a switchboard that allows packets to be s route the packets take is completely random and independent of any other packets b

Question 3

A firewall that operates at the packet level and studies each packet as it arrives and p router; each packet must pass the packet policy to gain access to the network.

Question 4

Computer viruses tend to copy themselves, infect the core system files and hide them any given moment. Trojans are not self-replicating and tend to be hidden among ins active when the file is executed. They are used to gain access into a system and take

Transmission Control Protocol / Internet Protocol

Question 1

a) An IP address stands for Internet Protocol address and acts as the address of the c transferred to and from the computer.

b) Also known as routable and non-routable, these are the different variations of IP will have a public IP address that is routable – that is data can be sent to and fro to it will have a private IP address which is non-routable that the router uses to c

c) A subnet mask is used to split the IP address into the host identifier and the netw

Question 2

The media access control (MAC) address is used to identify a device and to communic unique to each device.

Question 3

As it uses 128 bits the IP address pool becomes much larger; it could remove the nee many addresses available it would be near impossible to use them all in the foreseea

Question 4

NAT is the 'workaround' used by IPv4-enabled routers to increase the number of avai public address for each device, NAT allows the router to apply a private network addr traffic to that device only.

SECTION 10

Conceptual Data Models and Entity Relationship Modelling

Question 1

The models are used as a way of gaining information about how data flows in a syste

Question 2

e.g. Salary(<u>SalaryCode</u>, SalaryValue)

Relational Databases

Question 1

This is where the data is stored in different tables which are linked together using partic

Question 2

a) EmployeeID. A key is an item of data which is a unique identifier for that pa [illegible] the [illegible] EmployeeID.

b) SalaryCode. A foreign key is a primary key in another table/entity within the data entity to the other entity. In this case the employee will be linked to the salary er

Database Design and Normalisation

Question 1

a) Separate any attributes from keys formed in the previous step that are only deper

b) Separate any attributes that are dependent on other non-key attributes; foreign keys composite keys for redundant parts. If a part of a key can be derived from other attrit

Question 2

Normalisation increases speeds of data retrieval by structuring data in an easy-to-ind because of this structure as there is less likely to be anomalous data in the fields.

SECTION 11

Big Data

Question 1

Big data is a term given to any data source that is *difficult* to process due to its lack o volume, variety or velocity.

Question 2

By allowing programmers to produce code in terms of

- Immutable data structures
- Stateless [illegible]
- Higher-order functions

PROGRAMMING CHALLENGES

1. To the Power Of…

```
SUBROUTINE RunProgram ()
    Output "enter the first value: "
    baseVal <-- UserInput
    Output "enter the second value: "
    powerVal <-- UserInput
    Output baseVal^powerVal = ToThePower(baseVal, powerVal)
END SUBROUTINE

FUNCTION ToThePower (Base, Power)
    temp <-- 0
    For i <-- 1 to power
        temp += Base
    End For
    Return temp
END FUNCTION
```

2. Continuous Division

```
PROCEDURE RunProgram ( )
    OUTPUT "Enter an integer number: "
    userInput ← input # Converted to integer
    StartDivision (userInput)
END PROCEDURE

PROCEDURE StartDivision ( n )
    WHILE n > 1
        n ← n / 2
        OUTPUT n
    End While
END PROCEDURE
```

3. Guessing Game

```
PROCEDURE Initialise ( )
    OUTPUT "Enter the lowest value: "
    lowest ← Input
    OUTPUT "Enter the highest value: "
    highest ← Input
    secretNumber ← CreateRandom(lowest, highest)
    OUTPUT "How many guesses would you like to have? "
    guesses ← Input
    PlayGame(secretNumber, guesses)
END PROCEDURE

FUCNTION CreateRandom (minimum, maximum)
    tempRandom ← 0
    tempRandom ← NewRandom(minimum, maximum)
    return tempRandom
END FUNCTION
```

```
PROCEDURE PlayGame (secret, maxGuesses)
    isFound ←False
    totalGuesses ← 0

    While isFound ← False AND totalGuesses < maxGuesses
        OUTPUT "Enter your guess: "
        playerGuess ← input
        totalGuess++

        If playGuess ← secret
            OUTPUT "Congratulations! You've won in " & totalGuesses
            OUTPUT "Would you like to play again?"

            If replay = y Then
                isfound ← false
                Initialise()
            Else if replay ← n Then
                isFound ← true
                break
            End If
        Else if playerGuess - secret < 5 AND playerGuess - secret > -5 Then
            OUTPUT "So close!"
        Else if playerGuess - secret < 10 AND playerGuess - secret > 10 Then
            OUTPUT "Quite close!"
        Else
            OUTPUT "Not even close"
        End If
    End While
END PROCEDURE
```

4. Case Selection

```
PROCEDURE RunProgram ( )
    newRoom ← y
    While newRoom ← y
        NewBooking( )
        OUTPUT "Would you like to calculate a new room?"
        newRoom ← Input
    End While
END PROCEDURE

PROCEDURE NewBooking( )
    input ← 0
    OUTPUT "1 for single, 2 for twin, 3 for double, 4 for family"
    roomTotal ← room(input)
    OUTPUT "1 for self-catered, 2 for half-board, 3 for full board"
    boardTotal ← board(input)
        OUTPUT "Total: " & totalCost(roomTotal, boardTotal)
END PROCEDURE

...
```

```
...
PROCEDURE Room (input)
    tempTotal ← 0
    switch (input)
        case 1:
            tempTotal ← 50
            break
        case 2:
            tempTotal ← 75
            break
        case 3:
            tempTotal ← 90
            break
    return tempTotal
END PROCEDURE

PROCEDURE Board (input)
    tempTotal ← 0
    switch (input)
        case 1:
            tempTotal ← 0
            break
        case 2:
            tempTotal ← 5
            break
        case 3:
            tempTotal ← 10
            break
    return tempTotal
END PROCEDURE

FUNCTION TotalCost (Room, Board)
    OUTPUT "How long would you like to stay for?"
    stayLength ← input
    roomCost ← Room + Board
    tempTotal ← 0
    If stayLength > 7 Then
        tempTotal ← (roomCost * 7) - ((stayLength - 7) * (roomCost * 0.25))
    Else
        tempTotal ← roomTotal * stayLength
    End if
    Return tempTotal
END FUNCTION
```

5. Writing to and Reading from Files

```
PROCEDURE RunProgram ( )
    newArray [] ← LoadLeaderBoard ()
    PrintArray(newArray)
    OUTPUT "Enter new score: "
    newInput ← Input
    compareScore(newArray, newInput)
END PROCEDURE
...
```

```
…
PROCEDURE PrintArray (arrayName[] )
            OUTPUT "Current leaderboard is: "
            For i ← 0 to array.Length
                  OUTPUT (array[i] + " ")
            End For
END PROCEDURE

FUNCTION LoadLeaderBoard ( )
      leaderBoardArray[] ← leaderboardArray [5]
      lineFromFile ← Null
      new StreamReader ←reader
      fileName ← #newFileName
      reader ← new streamreader (fileName)
      counter ← 0
      while (!reader.EndOfStream)
            lineFromFile ← reader.ReadLine()
            leaderboardArray[counter] ← lineFromFile
            counter++
      End While
      Close Reader
      Return leaderboardArray
END PROCEDURE

PROCEDURE SaveLeaderBoard (leaderboard[])
      Streamwriter ← fileWriter
      fileName ← #newFileName
      FileWriter ← new FileWriter(fileName)
      for i ← 0 to leaderboard.Length
            inputFromArray ← leaderboard[i]
            FileWriter.Writeline(inputFromArray)
      End For
      OUTPUT "New entry added. Leaderboard saved."
      FileWriter.Close()
END PROCEDURE

PROCEDURE CompareScore (newArray[], newScore)
      temp1, temp2 ← 0
      If newScore < newArray[4]
            OUTPUT "Not on leaderboard"
      Else
            For i ← 0 to newArray.Length
                  If newScore > newArray[i]
                        temp1 ← newArray[i]
                        newArray[i] ← newScore
                        For j ← i + 1 to newArray.Length
                              temp2 ← newArray[j]
                              newArray[j] ← temp1
                              temp1 ← temp2
                        End For
                  PrintArray(newArray)
                  SaveLeaderBoard(newArray)
                  Break
            End For
      End If
END PROCEDURE
```

6. Magic Squares

```
SUBROUTINE RunProgram ()
    matrixIsMagix <-- false
    counter <-- 1
    while matrixIsMagic == false
        theMatrix[3,3]
        theMatrix <-- PopulateArray(theMatrix)
        rowsAreMagic <-- PerformRowCheck(theMatrix)
        colsAreMagic <-- PerformColCheck(theMatrix)
        OUPUT "Squares produced: " + Counter
        counter++
        if rowsAreMagic && colsAreMagix == true
            output "magic square found"
            PrintArray(theMatrix)
            break
        End if
    End While
END SUBROUTINE

FUNCTION PopulateArray (tempArray)
    rand <-- New Random
    for i <-- 1 to 3
        for j <-- 1 to 3
            tempArray[i,j] <-- rand.Next(1 to 10)
        End For
    End For
    return tempArray
END FUNCTION

SUBROUTINE PrintArray (tempArray)
    for i <-- 1 to 3
        for j <-- 1 to 3
            output tempArray[i,j]
        End For
        Output " " # Print to new line
    End For
END SUBROUTINE

FUNCTION PerformRowCheck (tempArray)
    isMagic <-- false
    row[3]
    for i <-- 0 to 3
        for j <-- 0 to 3
            row[j] <-- tempArray[i,j]
            tempTotal <-- tempTotal + row[j]

            If tempTotal != 15
                break
            End if
        End For
    End For
    return isMagic
END FUNCTION
...
```

```
...

FUNCTION PerformColCheck (tempArray)
        isMagic <-- false
        col[3]
        for i <-- 0 to 3
                for j <-- 0 to 3
                        col[j] <-- tempArray[j,i]
                        tempTotal <-- tempTotal + col[j]

                        If tempTotal != 15
                                break
                        End if
                End For
        End For
        return isMagic
END FUNCTION
```

ASSEMBLY PROGRAMMING CHALLENGES

Solutions below are for Little Man Computer (LMC).

1. Input two numbers and add them together; output the result.

```
INP
STA A
INP
ADD A
OUT
HLT
A DAT
```

2. Input two numbers and subtract them; output the result.

```
INP
STA A
INP
SUB A
OUT
HLT
A DAT
```

3. Write a program that counts backwards from 10.

```
LDA COUNT
START OUT
          SUB ONE
          BRZ END
BRA START
END     HLT
COUNT DAT 10
ONE DAT 1
```

4. Write a counter that counts up to 10.

```
START LDA COUNT
      ADD ONE
      STA COUNT
      OUT
      SUB TEN
      BRZ END
BRA START
END   HLT
COUNT DAT
ONE DAT 1
TEN DAT 10
```

5. Input a number and output its times table up to 10 times.

```
INP A
      [illegible]
START LDA TOTAL
      ADD A
      STA TOTAL
      OUT
      LDA COUNT
      ADD ONE
      STA COUNT
      SUB TEN
      BRZ END
BRA START
END   HLT
A DAT
TOTAL DAT
COUNT DAT
ONE DAT 1
TEN DAT 10
```

There are different methods to do this; however, above is probably the easiest to remember count at set-up is 0 not 1.

6. Write a counter that counts how many times I enter a non-zero number.

```
START INP
      BRZ END
      LDA COUNT
      ADD ONE
      STA COUNT
BRA START
END   LDA COUNT
OUT
HLT
COUNT DAT
ONE DAT 1
```

This introduces the idea that we can branch out at any point and then load the i

7. Write a program which adds up any list of numbers (by looping until I en
numbers I entered.

```
START INP
        BRZ END
        ADD TOTAL
        STA TOTAL
        LDA COUNT
        ADD ONE
        STA COUNT
BRA START
END     LDA TOTAL
OUT
LDA COUNT
OUT
HLT
TOTAL DAT
COUNT DAT
ONE DAT 1
```

8. Take in two numbers and output them in order (smallest first).

```
INP
STA A
INP
STA B
SUB A
BRP RESULT
LDA B
STA TEMP
LDA A
STA B
LDA TEMP
STA A
RESULT      LDA A
OUT
LDA B
OUT
HLT
A DAT
B DAT
TEMP DAT
```

This introduces the idea of using a temp location to allow the values to swap ove

9. Take in a number and divide it by 2 (clue: repeated subtraction, counting

```
INP
START SUB TWO
        BRP GOES
        BRA END
GOES    STA A
        LDA COUNT
        ADD ONE
        STA COUNT
        LDA A
BRA START
END LDA COUNT
OUT
HLT
A DAT
TWO DAT
COUNT DAT
ONE DAT 1
```

This gives the idea that you jump out to run a routine then jump back in, i.e. I tes the count and then jump back in.

10. Input two numbers and find the average (add together and divide by 2).

```
INP
STA A
INP
ADD A
START SUB TWO
        BRP GOES
        BRA END
GOES    STA A
        LDA COUNT
        ADD ONE
        STA COUNT
        LDA A
BRA START
END LDA COUNT
OUT
HLT
A DAT
TWO DAT 2
COUNT DAT
One DAT 1
```

Notice the simple bit at the top; just take in a number, take in another and ad

11. Write a program that finds any number entered (a) divided by another nu

```
INP
STA A
INP
STA B
LDA A
START SUB B
        BRP GOES
        BRA END
GOES   STA A
        LDA COUNT
        ADD ONE
        STA COUNT
        LDA A
BRA START
END LDA COUNT
[illegible]
HLT
A DAT
B DAT 4
COUNT DAT
ONE DAT 1
```

12. Write a program that finds the square root of a number entered.

```
        INP
        STA NUMBER
LOOP   SUB MINUS
        STA NUMBER
        LDA MINUS
        ADD INCREASE
        STA MINUS
        LDA COUNT
        ADD INC
        STA COUNT
        LDA NUMBER
        BRZ END
        BRP LOOP
        BRA UNABLE
END    LDA COUNT
        OUT COUNT
        HLT
UNABLE      LDA FAIL
        OUT FAIL
        HLT
NUMBER DAT
MINUS DAT 1
INCREASE DAT 2
COUNT DAT
INC DAT 1
FAIL DAT 0
```