SCHOLAR Study Guide

Higher (CfE) Computing Science Unit 1

Authored by:

Ian King

Jennifer Wilson

Mark J Tennant

Previously authored by:

David Bethune

Andy Cochrane

Tom Kelly

Ian King

Richard Scott

Heriot-Watt University

Edinburgh EH14 4AS, United Kingdom.

First published 2001 by Heriot-Watt University.

This edition published in 2014 by Heriot-Watt University SCHOLAR.

Distributed by Heriot-Watt University.

SCHOLAR Study Guide Unit 1: Higher (CfE) Computing Science

1. Higher (CfE) Computing Science

ISBN 978-1-909633-24-7

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

Acknowledgements

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

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

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

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

The Scottish Government for financial support.

Contents

Topic 1

Languages and environments

Contents

Prerequisite knowledge

You should already know that:

- *natural (human) languages are much more complex than programming languages;*
- *computer programming languages, are used to write programs which are used to give a sequence of commands to a computer;*
- *all high level languages must be translated into machine code before they can be understood by the computer;*
- *there are a large number of different programming languages and that many have been created in order to solve specific types of problem.*

Learning Objectives

By the end of this topic you will be able to:

- *understand the difference between high level and low level programming languages;*

- *know that all programming languages can be reduced to three basic control structures:* ***sequence, selection*** *and* ***iteration****;*
- *appreciate that programming languages can be classified in a number of different ways;*
- *explain the differences between* ***procedural, declarative, object-oriented*** *and* ***domain-specific languages****;*
- *understand the difference between two types of translation software: compilers and interpreters;*
- *describe the tools a programmer would expect to be available in a modern programming environment.*

1.1 Revision

Quiz: Revision

Q1: Programs written in a High Level language need to be translated because?

a) Computers only understand machine code
b) Machine code is easier for the programmer to understand
c) There are many different programming languages
d) A computer can only understand a few programming languages

....

Q2: There are many different High level programming languages because?

a) There are many different human languages
b) High level languages are often written to solve particular types of problem
c) There are many types of translation software
d) There are many different types of computer

....

Q3: What are the main differences between natural languages and programming languages?

....

1.2 Low level and high level languages

Digital computers are made up of from electronic switches which only have two states, on and off. Because of this they have to be given instructions using a language called machine code which contains only two corresponding characters: 1 and 0. For this reason is the language used to directly control a computer it is referred to as a **low level language**. Although we could give commands to computers using this code; it is not easy for humans to read or understand. For this reason why we use programming languages which are more like the natural languages we use to communicate between ourselves. The programming languages we use are called **high level languages** because they always have to be translated into machine code before a computer can use them.

Natural language
High level language
Low level language

It would be nice if we could use a natural language like English to communicate with computers, but unfortunately **natural languages** are too complex and too ambiguous to be translatable directly into machine code, although there is much research going on into achieving this goal. All languages whether human or machine, have rules which can be used to decide whether a statement is grammatically correct or not. The grammatical rules for programming languages are much simpler than those for natural languages, and it is these rules, usually referred to as their **syntax**, which are used by the translation software which converts programs we have written in a high level language into machine code.

There are approximately 7000 current human languages spoken in the world. There are over 2000 computer programming languages but luckily for native English speakers, the majority of computer programming languages use English words as **keywords** in their commands.

1.3 Control structures

Programming languages are used to control the activities of computers. For this reason all programming languages use **control structures**. Strictly speaking we only need 3 basic control structures to control a computer:

- **Sequence:** This is the rule that unless otherwise directed, any set of commands will be executed one after the other.
- **Selection:** This is when a decision about which command to execute depends on whether a condition is true or not.
- **Iteration:** This is where a set of commands is repeated for a set number of times, often called a loop.

What distinguishes different programming languages is how these three basic control structures have been combined into **computational constructs**, and how these constructs are used.

Activity: Control structures

Identify the following pseudocode segments as Sequence, Selection, Iteration, or a combination of these control structures.

Q4: Identify the following pseudocode segments as A) Sequence, B) Selection, C) Iteration, or a combination of these control structures:

```
SET total TO 0
RECEIVE userValue FROM KEYBOARD
SET total TO total + userValue
```

a) A
b) B
c) C
d) A and B
e) A and C
f) B and C
g) A, B, and C

....

Q5: Identify the following pseudocode segments as A) Sequence, B) Selection, C) Iteration, or a combination of these control structures:

```
RECEIVE userValue FROM KEYBOARD
IF userValue > 100 THEN
             SEND ‘‘Number too high’’ TO DISPLAY
END IF
```

a) A
b) B
c) C
d) A and B
e) A and C
f) B and C
g) A, B, and C

....

Q6: Identify the following pseudocode segments as A) Sequence, B) Selection, C) Iteration, or a combination of these control structures:

```
RECEIVE userValue FROM KEYBOARD
WHILE userValue < 0 OR userValue > 100 DO
      SEND ‘‘Number must be between 1 and 100’’ TO DISPLAY
END WHILE

END IF
```

a) A
b) B
c) C
d) A and B
e) A and C
f) B and C
g) A, B, and C

....

Once you have complete the above questions, explain what the pseudocode does for each of the three segments.

....

1.4 Why so many programming languages?

Because programming languages have a much simpler and smaller vocabulary and set of grammar rules than natural languages, they are much easier to create. Many programming languages have been created in order to solve a particular sort of computing task, or were written with a particular set of people in mind such as those involved in Commercial Data Processing (COBOL), scientists (FORTRAN) or artificial intelligence researchers (PROLOG). This is often described as programming languages being **problem oriented**.

Most programming languages are very similar to each other which means that once you have learned how to use one programming language, learning the grammatical rules of another one is often quite easy.

Activity: Programming languages

30 min

Q7: Research the following programming languages. Once you have done this, match each to one of the brief descriptions that follow.

- HTML
- Smalltalk
- Java
- PHP
- BASIC
- Python
- Pascal
- FORTRAN
- PROLOG
- COBOL

Language	Description
	A high level procedural language designed to be easy to learn which became the popular language when computers became cheaper and popular in the 1970s.
	One of the earliest procedural programming languages designed for business and finance users.
	A general purpose procedural language developed by Niklaus Wirth and designed to encourage good programming practice.
	A highly portable object-oriented general purpose programming language designed to be platform independent and used for client-server web applications.
	A domain specific page description language used to instruct web browsers how to display web pages.
	A general-purpose, procedural programming language designed for numeric and scientific computing.
	An early object-oriented programming language originally designed for educational use.
	A declarative logic language designed for artificial intelligence applications.
	a server-side scripting language designed for web development, often in combination with the MySQL database application.
	A general purpose procedural programming language designed by Guido van Rossum with an emphasis on readability of code.

. . . .

1.5 Classifying programming languages

Although they have many similarities, programming languages are different enough from each other to be classified in a number of ways. No method of classification is perfect and no matter how they are classified, there will be languages which fall into more than one category.

The simplest classification is by how they are used: In this section we are going to look at procedural, declarative, object oriented and domain specific languages.

1.5.1 Procedural languages

Procedural languages are the commonest type of programming language. Programs written in an procedural language will consist of a sequence of commands which will be executed in a predictable order from beginning to end. A low level language like machine code is the simplest example of an procedural language, but it is not easy to write programs using it, so high level programming languages were developed to make programming easier and more productive. Many procedural programming languages were designed originally as languages to teach programming, and as a result are often called general purpose languages. BASIC (Beginners All-Purpose Symbolic Instruction Code) and Pascal are two examples of languages which were developed in this way.

Procedural languages use standard arithmetic operations (+, -, *, /, etc), and a wide variety of control structures and sub-programs.

A programmer using an procedural programming language creates a set of instructions which are executed in order to provide a solution to a problem. The set of instructions will have a definite beginning and end and will describe exactly what the computer must do at each step. These instructions are executed in written order by the computer.

The first programming languages were all procedural languages as this matched the way in which digital computers behave - they follow one instruction after the other. At machine code level it could be said that all programming languages are procedural, since they all must be translated into machine code before their instructions can be executed.

Examples of procedural programming languages are: BASIC, C, Pascal and Python.

Code example: Visual Basic 6

Example of a program to read data into an array and output results

```
'Program to fill an array with team names and print them out

Private Sub FillArray_Click()
Dim Team(5) As String
Dim Count As Integer

'Assign array values

Team(0) = "Leith Learners"
Team(1) = "Bonaly Boys"
Team(2) = "Royston Rovers"
Team(3) = "Calder Colts"
Team(4) = "Juniper Juniors"
Team(5) = "Wardie Wanderers"

For Count = 0 To 5
  Print "Teams"; Tab(8); Count+1; Tab(14); Team(Count)
Next Count
Print

End Sub
```

Notice that:

- the program is expressing at each step precisely how each statement is executed;
- the program has beginning and end points, which is a feature of procedural language programs.

Q8: From the above example, write out the predicted output.

....

1.5.2 Declarative languages

Declarative languages were an attempt to create a programing language which more closely matches how humans think than how a digital computer behaves. For this reason, they are most closely associated with artificial intelligence programming. In contrast to a program written in an procedural language, a program written in a declarative language will consist of a set of facts and rules (the knowledge base). When a program is run by entering a query, the solution is found by finding a match between the query and the facts and rules in the knowledge base. This is a quite different approach to how a procedural language is used, where a program would follow a set of commands in sequence (although that sequence may very from one run to another depending on the user choices while it is running) A program written in a declarative language will not necessarily follow a specific sequence of operations.

- Declarative languages tend to have fewer variable types and less control structures than procedural languages.
- They make more use of programming techniques like recursion (where a sub program repeatedly calls itself until a simple base fact is identified).
- They use self modifying code (where the program modifies its set of facts and rules).
- Being able to modify the facts and rules depending on circumstances while a declarative program is running makes such languages useful where an **expert system** has to build up knowledge and "learn" from experience).

Examples of Declarative programming languages are: PROLOG, Lisp

PROLOG facts and rules are stored as a database (or knowledge base) which can then be queried to provide solutions.

Code example: PROLOG:

Problem: Suppose we want to find out whether a person drives a fast car. We start by building a set of facts and rules for our knowledge base.

Solution:

```
person(judy).
person(james).

drives_car(james, ford_escort).
drives_car(judy, porsche).

drives_fast_car(X):-
drives_car(X, porsche).
```

In this example we could ask the program to tell us whether Judy drives a fast car by typing the query:

`?drives_fast_car(judy)`. The result would be YES since the goal is satisfied.

If we asked:

```
?drives_fast_car(james)
```

then the result would be NO as `drives_car(james,Y)` would evaluate Y="ford escort".

This would then cause the rule `drives_fast_car(james)` to fail as Y does not equal "porsche" and the goal is not satisfied.

You can see from the code that there is no description of the type of data or its internal representation. There are simply statements of facts and a rule.

Contrast this with an procedural language where the programmer would need to set up a structure to hold the knowledge and predefine its type (string, number etc). Then they would need to describe the steps taken to search the structure in order to answer the query. A declarative/logical language is simplistically described as telling the computer what to do and not how to do it.

Activity: PROLOG

This PROLOG program consists of a set of facts about a set of simple individuals. The facts describe who is warm, who is fed and who is cold. There are two simple rules to determine whether someone is happy or unhappy.

```
warm(fred).
warm(joe).
warm(jim).
warm(jack).

fed(fred).
fed(francis).
fed(justin).

cold(jim).
cold(justin).

happy(Person):-
  warm(Person),
  fed(Person).

unhappy(Person):-
  cold(Person).
```

Match the answers to the following queries:

Q9: ?warm(fred)

a) Yes
b) No

. . . .

Q10: ?happy(jack)

a) Yes
b) No

. . . .

Q11: ?happy(fred)

a) Yes
b) No

. . . .

Q12: ?unhappy(X)

a) X = jim
b) X = fred
c) X = justin
d) X = jim, X = justin

. . . .

1.5.3 Object-oriented languages

Object-oriented languages were originally developed from procedural languages as an attempt to make writing software more efficient. Graphical user interfaces (GUIs) are now almost universal and they are event-driven environments where user interaction is required in the form of a mouse click or a key press to represent the processing. Procedural languages do not possess the necessary constructs to deal with this style of programming and are generally unsuitable for building GUI applications. Programs were becoming increasingly complex and the use of **global variables** meant that it became more and more difficult to keep errors from occurring where data was changed accidentally.

Object-oriented programming languages address these problems by creating re-usable blocks of code (called objects) which define the data used within that block and how that data can be changed.

- The data associated with an object is referred to as its **attributes**, and the methods which can change that data are called its **operations**.
- These **objects** are defined as being of a particular type (called a **class**) which can be used to create other objects with the same characteristics.

- Objects are closed systems which cannot be altered from outside which makes errors caused by the use of global variables less likely.
- This feature is called **encapsulation**. The benefit of using this technique is that once a class has been defined and tested, sub-classes can then be created which share the main characteristics of the parent class, thus saving testing and development time.
- This ability to create a sub-class from a pre defined class is called **inheritance**.
- Because objects in an object-oriented program are closed systems, they interact with each other by sending **messages** to other objects. Messages are much easier to manage and keep track of than the changing values of global variables.
- A programmer using an object-oriented language would create a set of classes and sub-classes which define the objects to be used in the program.
- Object-oriented languages depend greatly on the concept of **class libraries**. These are sets of classes that can be used by developers as common building blocks for complex applications, rather akin to module libraries that are used in procedural programming.
- Whereas a procedural language program will be built from a number of procedures called from a main procedure, an object oriented program will have a number of **methods** which are specifically linked to the class within which they are defined, in keeping with the idea of keeping classes and objects self contained.

Object-oriented table

Object	**Attributes**	**Operations**
Button	Name, Size, Position, Colour	Click, Mouse-over
Window	Name, Size, Position, Focus, Border	Maximise, Minimise, Resize, Open, Close
Dialogue box	Contents, Priority	Open, Close, OK, Cancel

Examples of object-oriented languages are: Java, C++, Smalltalk, BYOB

Activity: Object-oriented table

Q13: Extend the above *object-orientated table* to include the attributes and operations for:

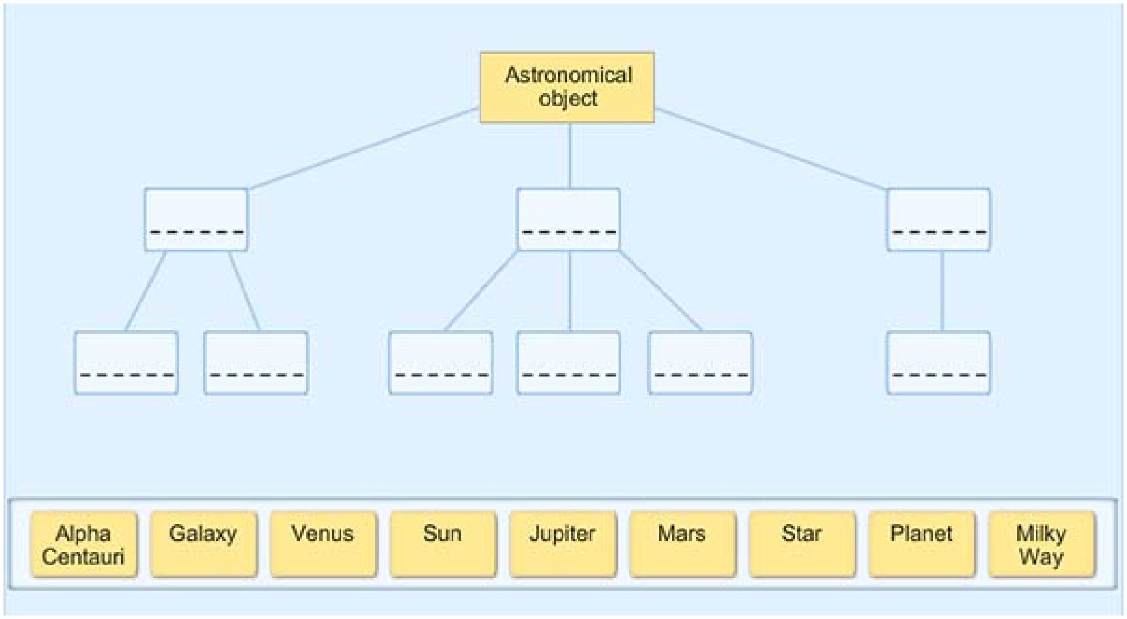
- Text box
- Radio button
- Pull-down menu

Object	**Attributes**	**Operations**
Button	Name, Size, Position, Colour	Click, Mouse-over
Window	Name, Size, Position, Focus, Border	Maximise, Minimise, Resize, Open, Close
Dialog box	Contents, Priority	Open, Close, OK, Cancel
Text box		
Radio button		
Pull-down menu		

. . . .

Activity: Inheritance diagram

Q14: Match the words to the correct nodes to complete the inheritance diagram:

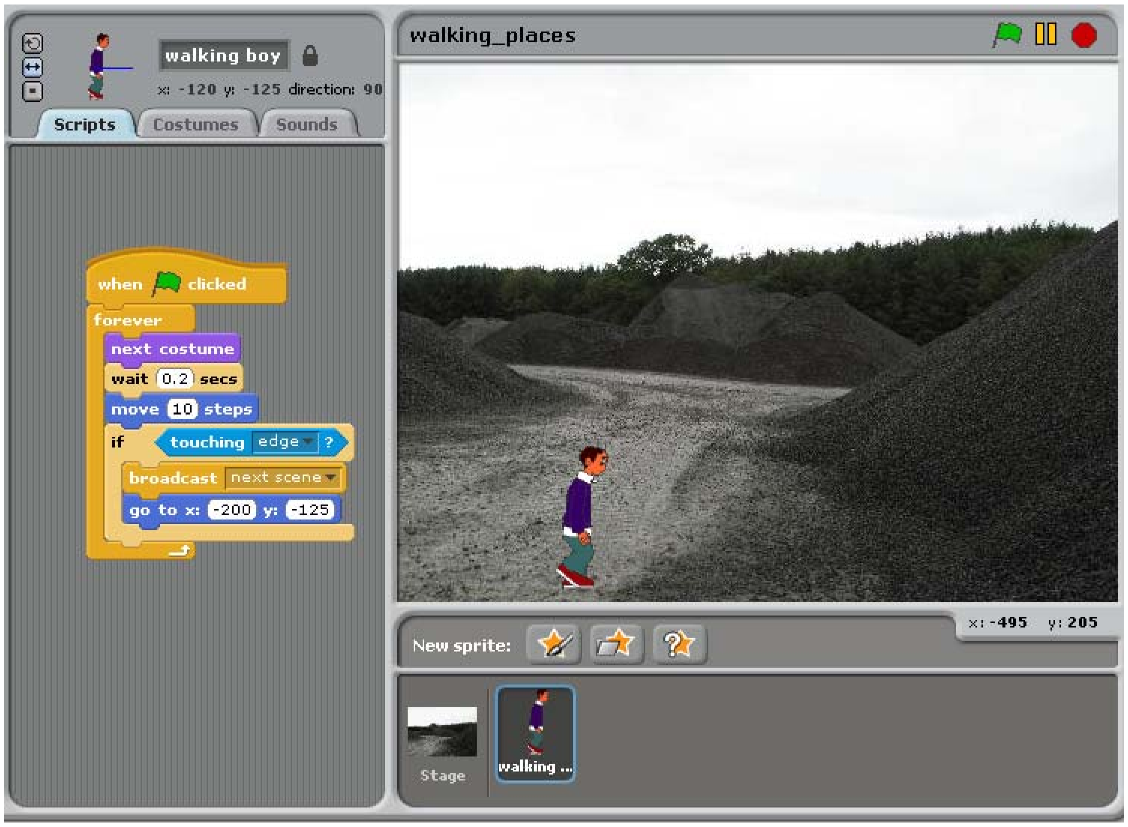


. . . .

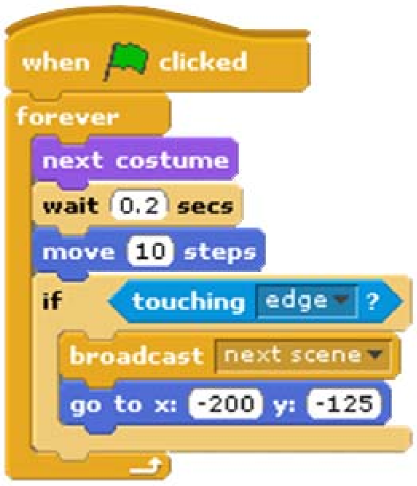
Code example: BYOB

BYOB is an example of an object-oriented language used to teach programming. In this example, the boy sprite has inherited the attributes of the general sprite class, but has the additional specific attributes of several costumes to give a walking animation effect. It communicates with the stage when it is touching the edge by sending a message. When the stage receives the message it changes its background attribute to display the next image.

Figure 1.1: Code example: BYOB



```
when [flag] clicked
forever
    next costume
    wait 0.2 secs
    move 10 steps
    if touching edge ?
        broadcast next scene
        go to x: -200 y: -125
```



```
when [flag] clicked
forever
    next costume
    wait 0.2 secs
    move 10 steps
    if touching edge ?
        broadcast next scene
        go to x: -200 y: -125
```

. . . .

In this example of the Java programming language, there are three methods defined within the average class: fillarray() which returns an integer array, average() which returns an integer value and the main method which calls fillarray() and average() and displays the average of the values in the array.

```
import java.util.Random;

class Average{

// Class to find average of values in an array

// set up constants

   static final int arraylength = 20;
   static int  arrayin []= new int [arraylength];

// method to initialise array

static void fillarray(int arrayin[]){

   Random arraycontents = new Random ();

   for(int counter=0;counter <= arrayin.length-1;counter++){
     arrayin[counter] = Math.abs(arraycontents.nextInt() % 1000);
     }
    System.out.println("Array now initialised");
 }


// method to find average value

 static int average_value (int [] arrayin){

   int total = 0;

   for(int counter=0;counter <= arrayin.length-1;counter++){
        total = total + arrayin[counter];
        }
   return total/arrayin.length;
   }

//main method

public static void main (String [ ] args){

  fillarray(arrayin);
  System.out.println("Average is " + average_value(arrayin));
}

}
```

Quiz: Object-oriented languages

Q15: Give **two** reasons for the development of object-oriented languages.

. . . .

Q16: Give **three** characteristics of object-oriented programming languages.

. . . .

Q17: Explain the concept of inheritance and why it is an important feature of object-oriented programming?

. . . .

1.5.4 Domain-specific languages

Domain-specific languages (sometimes called mini languages) are a subset of procedural languages, but what distinguishes them is that they have been designed with one specific type of task in mind. A page description language like HTML is a good example as it is used specifically to describe the layout and design of a web page. Another example is SQL (Structured Query Language) which is the language used to formulate database queries. A programmer using a domain-specific language will typically have a particular type of application in mind and wants a language which has specific commands and control structures appropriate to that application. This makes the programmer's job easier and shortens development time as a result.

Code example: HTML

Here is an example of HTML which instructs a web browser to display text in a variety of sizes, horizontal lines, an image and a link to the Google website.

```
<html>
<head>
   <title> First Page </title>
</head>
<body>
  <h1> My first Hyper-text page </h1>
  <hr>
  <h3> This section is in font size 3 </h3>
  <p> you can make text <b> bold </b></p>
  <p> or <i> italic </i></p>
  <p> or <u> underlined </u></p>
  <p></p>
  or <b> <i> <u> all three </b> </i> </u>
  <hr>
  <p></p>
  <img src= "picture.jpg">
  <p></p>
  <a href="http://www.google.co.uk"> Google </a>
</body>
</html>
```

1.5.5 Scripting languages

Scripting languages are usually designed to add extra functionality to, or automate an application program or an operating system. Scripting languages include those macro languages which are part of applications like Word and Excel and languages like Javascript and VBscript which can be embedded in HTML documents to add interactivity to web pages.

A **macro** is a sequence of operations that can be invoked as a single task. It therefore lets you automate a frequently-performed task and can be simple, such as entering text and formatting it, or complex, like automating tasks that would take several minutes to do manually.

Many programs (like Microsoft Word and Microsoft Excel) let users create macros easily by "recording" a set of actions as you perform them. For example, you could record opening a new document using a specific template, inserting a header and inserting a name and address and greeting. Each time you "replayed" the macro, it would perform those tasks. The macro is stored as a script using the application's scripting language, VBScript and can be placed on the application toolbar as a button or in a menu as a command.

Code example: VBScript

This is a example VBScript listing for a macro to create a table in Microsoft Word:

```
Sub Macro1()
'
'
' Macro to create a table in Word
'
' Macro1 Macro
'
ActiveDocument.Tables.Add Range:=Selection.Range, NumRows:=7,
NumColumns:= _
  4, DefaultTableBehavior:=wdWord9TableBehavior,
AutoFitBehavior:= _
  wdAutoFitWindow
With Selection.Tables(1)
  If .Style <> "Table Grid" Then
    .Style = "Table Grid"
  End If
  .ApplyStyleHeadingRows = True
  .ApplyStyleLastRow = True
  .ApplyStyleFirstColumn = True
  .ApplyStyleLastColumn = True
 End With
End Sub
```

Code example: Javascript

Here is a simple example of an HTML page which uses the Javascript scripting language:

```
<html>
<head>
<title>Adder</title>

<SCRIPT language = JavaScript>

function add() {
  A = document.form1.number1.value
  B = document.form1.number2.value
  A = Number(A)
  B = Number(B)
  C = (A + B)
  document.form1.answer.value = C
}

</SCRIPT>

</head>

<body>
<form name= form1>
  <p>
    Number one:
      <input type="text" name="number1" value = "">
  </p>
  <p>Number two:
     <input type="text" name="number2" value = "">
  </p>
  <p>
   Answer:
   <input type="text" name="answer" VALUE = "">
  </p>
  <p>
    <input type=Button name="Add" value="Add" onClick = add()>
  </p>

</form>
</body>
</html>
```

This program inputs two integers and displays their sum.

Scripting languages also have a simple syntax which, for the programmer:

- makes them easy to learn and use;

- assumes minimum programming knowledge or experience;
- allows complex tasks to be performed in relatively few steps;
- allows the addition of dynamic and interactive activities to web pages.

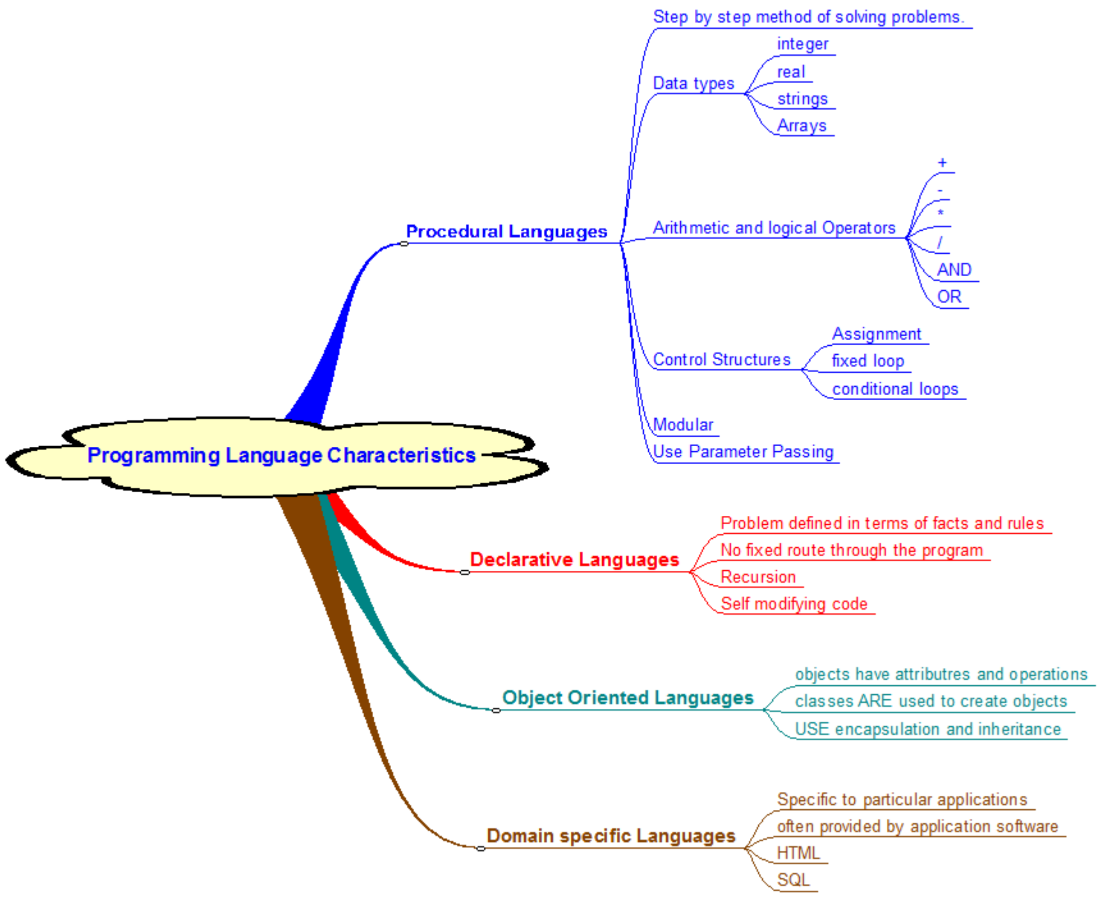
1.5.6 New programming languages

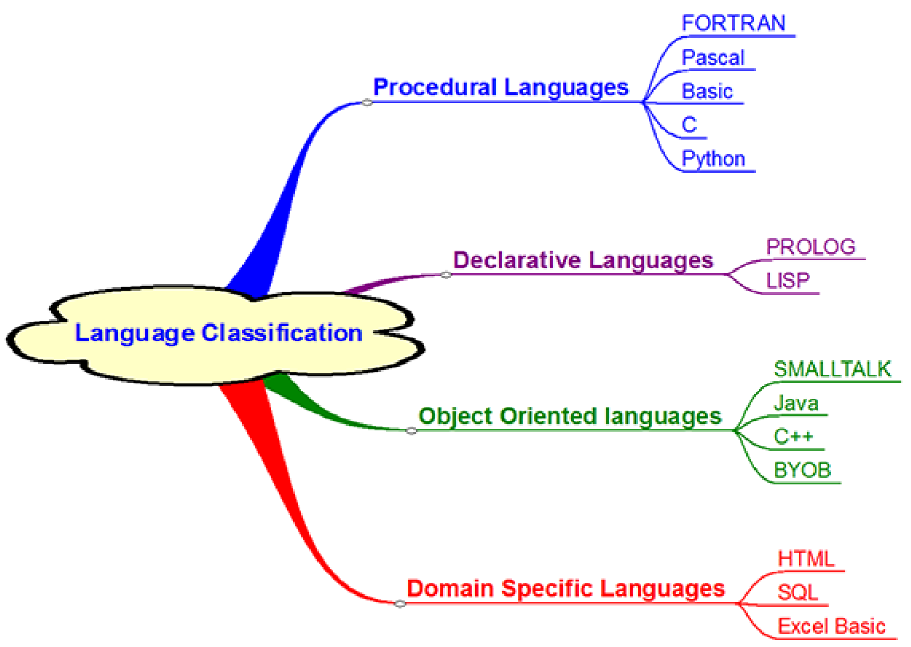
New programming languages are being developed all the time, as developments such as distributed computing, on-line and mobile applications become popular.

Recent surveys on the popularity of programming languages among both programmers and employers have come up with relatively consistent results.

Currently these are among the most popular languages:

- Java
- Python
- C#
- PHP
- Ruby
- Javascript


Programming Language Characteristics
Procedural Languages
Step by step method of solving problems.
Data types
integer
real
strings
Arrays
Arithmetic and logical Operators
+
-
*
/
AND
OR
Control Structures
Assignment
fixed loop
conditional loops
Modular
Use Parameter Passing
Declarative Languages
Problem defined in terms of facts and rules
No fixed route through the program
Recursion
Self modifying code
Object Oriented Languages
objects have attributres and operations
classes ARE used to create objects
USE encapsulation and inheritance
Domain specific Languages
Specific to particular applications
often provided by application software
HTML
SQL



1.6 Programming environments

At the end of the implementation stage of software development, if all is going well, a structured **program listing** will be produced, complete with internal (commentary) documentation. This will be thoroughly checked against the design and against the original specification.

The high-level code written at this stage is called **source code** which must be translated into machine code, called **object code** that the computer understands ready for execution by the computer.

A programming environment will normally include a program editor with text editing tools such as search and replace, predictive typing, automatic indentation, and colour coding of comments, **keywords** and variables. The programming environment may also include **debugging tools** which allow the programmer to step through a program line by line or to stop a program at a particular place by setting a **breakpoint** to investigate the value of variables at that point. Another facility provided by many programming environments is the ability to link modules from a **class library**. This is particularly common when using object-oriented programming environments.

Many modern programming environments allow programmers to create GUI objects such as windows, buttons and other graphical objects by "drawing" them on the program interface. Code still has to be created to draw these objects - in fact they *could* be programmed from scratch - but the environment does this for the programmer.

Translating high level languages into machine code

High level programming languages always need to be translated into machine code before they can be run on the machine they are being written on. There are two approaches to the translation process.

Interpreters: An interpreter translates the program (or source code) into machine code line by line. This means that the interpreter can flag up errors in code as it is being typed. Unfortunately an interpreter will be needed every time the program is run and the translation process will slow down the execution of the program.

Compilers: Compilers translate the program (or source code) into machine code in one operation. Once translated the program can be run independently of the compiler and therefore it runs faster than an interpreted program, however if the programmer wants to change code as a result of testing, it will need to be re-compiled. Compiled programs are more efficient because no extra memory or processor resources are needed for the translation software.

Some programming languages like Java are purely interpreted languages, other programming languages will only provide a compiler. Many modern environments provide the best of both worlds - an interpreter to write, test and debug the program, and a compiler to translate the source code into an executable machine code program once it is ready to distribute.

Although machine code is often described as the only language that computers "understand", we have to remember that each type of processor will have its own set of machine code instructions which refer to that processor's internal components. This means that machine code is processor-specific, so when a new programming language is created, the translation software for currently available processors will also have to be developed.

Efficiency

The main difference between an interpreter and a compiler is that compilation "requires the analysis and generation of machine code only once, whereas an interpreter may need to analyse and interpret the same program statements each time it meets them e.g. instructions appearing within a loop.

For example, consider the following code:

```
FOR counter FROM 1 TO  200  DO
  sum = sum + counter
  SEND sum TO DISPLAY
END FOR
```

Using a compiler, the source code would be analysed and compiled into machine code once only.

Using an interpreter, the source code would be converted into machine code 200 times (once each time round the loop).

Errors

This has implications for error reporting. For instance, when the interpreter encounters an error it reports this to the user immediately and halts further execution of the program. Such instant feedback, pinpointing the exact location of the error, helps the programmer to find and remove errors.

Compilers, on the other hand, analyse the entire program, taking note of where errors have occurred, and place these in an error/diagnostic file. If errors have occurred then the program cannot run. Programmers must then use the error messages to identify and remove the errors in the source code.

Some compilers assist by adding line numbers to the source listing to help pinpoint errors and all compilers will describe the nature of the error e.g. missing semi-colon, expected keyword, etc. - although interpreting some compiler diagnostics is a skill in itself.

Ease of use

Interpreters are more suitable for beginners to programming since errors are immediately displayed and can be corrected by the user, until the program is able to be executed.

On the whole compilers tend to be more difficult to use because they only give the programmer error messages once the code has been translated.

Portability

A compiler produces a complete machine code program which can be saved, copied, distributed and run on any computer which has the appropriate processor type. A programming language is said to be **portable** if recompiling it on different platforms is relatively easy.

An interpreter does not do this. The machine code has to be generated each time the program is run. This means that the interpreter must always be present, and program execution is slower.

Compiler and interpreter

A compiler, which is a complex program in itself, translates source code into object code that is then loaded into main memory and executed.

Another form of translation that converts source code into object code is an **interpreter**.

Unlike a compiler, an interpreter checks syntax and generates object code one source line at a time. Think of this as very similar to a group of translators at a United Nations' Conference, who each have to convert sentences spoken by delegates into the native language of their representative.

When an error is encountered, the interpreter immediately feeds back information on the type of error and stops interpreting the code. This allows the programmer to see instantly the nature of the error and where it has occurred. He or she can then make the necessary changes to the source code and have it re-interpreted.

As the interpreter executes each line of code at a time the programmer is able to see the results of their programs immediately which can also help with debugging.

Q18: Match up the advantages and disadvantages to the two methods of translating source code into machine code:

1. Compiler advantage
2. Compiler disadvantage
3. Interpreter advantage
4. Interpreter disadvantage

Description	**1, 2, 3, or 4?**
Can test code while it is being written	
Creates fast executable machine code	
Does not provide clear error messages	
Must be re-translated if changes have to be made to the source code	
Source code needs to be translated every time it is run	
Can partially translate source code	
Cannot translate code which contains errors	
Provides helpful error messages	
No need for the translation software once the source coded has been converted to machine cod	
Machine code cannot be converted back into source code	
The translation software is needed along with the source code every time it is run	
Programs run more slowly because they are being translated while they are running	

. . . .

Quiz: Programming environments

Q19: One of the main differences between a compiler and interpreter is?

a) An interpreter is faster then a compiler
b) A compiler is better for beginners
c) A compiler creates an independent machine code program
d) An interpreter is much harder to use

. . . .

Q20: One disadvantage of using an interpreter is?

a) Looping structures have to be interpreted each time they are executed
b) It stops execution when an error is encountered
c) It helps the user to debug programs
d) An interpreter is ideal for beginners

. . . .

Q21: High level languages have to be translated because?

a) Computers can only understand machine code
b) Source code is faster to run than object code
c) Programs run faster when converted to binary
d) All of the above

. . . .

Q22: Which one of these is NOT true?
When distributing a completed piece of software the advantage of using a compiler is:

a) The compiled program does not need an interpreter to run
b) The compiled program runs faster than it would if running in an interpreter
c) A compiled program cannot be editied without access to the source code
d) A compiled program takes up less memory than an interpreted one

. . . .

. . . .

1.7 Learning points

Summary

- all programming languages can be reduced to three basic control structures: sequence, selection and iteration;
- what distinguishes different programming languages is how these three basic control structures and methods of data representation have been combined into language specific computational constructs;
- there are many types of high level programming languages in use today, including procedural, declarative, object-oriented and domain-specific languages;
- high level languages can be general purpose or problem-specific;
- high level languages have to be translated to machine code by compiler or interpreter before execution;
- modern programming environments will include a text editor and debugging tools.

1.8 End of topic test

End of topic test

Q23: Which one of the following types of language would be most suitable for programming a knowledge base in an expert system?

a) Procedural
b) Declarative
c) Functional
d) Object-oriented

. . . .

Q24: Which of these is **not** a feature of a high level language?

a) They are problem oriented
b) They need to be translated into machine code before they can be understood by the compiler
c) They have specific grammatical rules
d) They can contain statements which are ambiguous

. . . .

Q25: Which type of programming language is used to create a set of steps which are executed in order to provide a solution to a problem?

a) Procedural
b) Declarative
c) Object-oriented
d) Domain-specific

. . . .

Q26: What type of programming language is used to create a knowledge base and a query which specifies the problem which are then matched to provide a solution?

a) Procedural
b) Declarative
c) Object-oriented
d) Domain-specific

. . . .

Q27: Which of these is not a feature of an object-oriented programming language?

a) Knowledge base
b) Inheritance
c) Encapsulation
d) Message passing

. . . .

Q28: Which of these is a domain-specific programming language?

a) BASIC
b) Java
c) SQL
d) Pascal

. . . .

Q29: Which of these is not true of machine code?

a) Processor specific
b) Created by a compiler from source code
c) Easy for humans to understand
d) A low level language

. . . .

Q30: Which of these features would you expect to be provided by a high level language programming environment?

a) Keyword highlighting
b) Automatic indentation
c) Search and replace
d) Debugging tools
e) Spell check

. . . .

Q31: Which one of the following statements about compilers is true?

a) Different platforms require different compilers
b) Compilers translate the program creating a source code file
c) A compiler will run part of a program even if an error is found
d) Compilers translate and execute each line of the code in turn

. . . .

Q32: Which of the statements below describes a class library?

a) A list of all of the classes used in a program
b) A file containing all the actual code used in a program
c) A set of pre-written classes that tests part of a program
d) A set of pre-tested classes which can be used in a program

. . . .

Q33: Which of the following statements about interpreters is false?

a) An interpreter will run a program until an error is found
b) Interpreters translate source code to create an object code file
c) Interpreters are platform specific
d) Interpreters translate and execute a program line by line

. . . .

Q34: Which of the following statements best describes an object-oriented programming language?

a) An object-orientated language contains special routines for handling documents
b) The programmer can define customised data types
c) The programmer defines both the data and the operations that can be carried out on it
d) An object-orientated language has built-in routines for drawing graphics objects

. . . .

Q35: Which of these statements best describes portability of software?

a) The software can be run on more than one computer
b) The software can be easily re-translated for different platforms
c) The software is small enough to be easily transferred over a network
d) The software only needs to be compiled once

. . . .

. . . .

Topic 2

Low level operations: Storing data

Contents

Prerequisite knowledge

You should already know that:

- *computers use binary code made up of ones and zeros to store integers;*
- *computers store text using ASCII code;*
- *computer graphics are stored as a series of bits corresponding to the number of pixels in the image.*

Learning Objectives

By the end of this topic you will be able to:

- *understand why computers store numbers as binary code;*
- *convert between binary and decimal;*
- *understand how computers store integers using two's complement notation;*
- *convert between a decimal integer and two's complement notation;*
- *explain how computers store real numbers using floating point notation;*
- *understand that text can be stored as ASCII code or as Unicode;*
- *understand how computers store graphics as bitmaps and as vector graphics;*
- *calculate graphic image file sizes explain the need for data compression;*
- *explain how computers store sound and video data.*

2.1 Revision

Quiz: Revision

The following exercise tests the prerequisites for this topic. Ensure that you are happy with your responses before progressing.

Q1: The decimal number 73 is expressed in binary as

a) 0100 1001
b) 0111 1100
c) 1 1111
d) 0111 0011

. . . .

Q2: Use the ASCII character set to translate the following bytes:

0100 0011 0110 1111 0110 0100 0110 0101

. . . .

Q3: A 1 bit/pixel monochrome display has 200 vertical lines. Each vertical line has 640 horizontal dots. Calculate the memory required to store a single screen shot. Express your answer in appropriate units.

. . . .

2.2 Using binary code to represent and store numbers

Learning Objective

By the end of this section you will be able to:

- understand why computers store numbers as binary code.

Digital computers are made from millions (and often billions) of tiny switches called transistors. Switches can be either on or off. For this reason, all information handled by computers must be coded into/represented by patterns of 1s and 0s. Humans are much more familiar with the decimal system and think of numbers in terms of base 10, so in order to understand how the computer processes numeric information; we must be comfortable with both binary and decimal numbers.

Decimal System:

10^4	10^3	10^2	10^1	10^0	*	10^{-1}	10^{-2}
Ten Thousands	Thousands	Hundreds	Tens	Units	*	Tenths	Hundredths
0	1	3	5	6	*	0	5

We would read this as: 1 x 1000 + 3 x 100 + 5 x 10 + 6 x 1 + 5 x 1/100= 1356.05

Like us, the computer must also be able to process positive and negative numbers that can also be very large or very small. This section looks at all such numbers and how they can be represented.

Quiz: Using binary code to represent and store numbers

Q4: Why do you think we use a base 10 number system instead of 8 or 12 for instance?

. . . .

2.3 Storing integers

Learning Objective

By the end of this section you will be able to:

- convert between binary and decimal.

An integer is a whole number without a decimal fraction. An integer can be positive or negative including the number zero.

If we consider positive integers, the binary system works in the same way as the decimal system, but using powers of two instead of powers of ten.

2.3.1 Converting binary to decimal

2^7	**2^6**	**2^5**	**2^4**	**2^3**	**2^2**	**2^1**	**2^0**
128s	64s	32s	16s	8s	4s	2s	Units
0	0	0	0	0	0	1	1

We would read this as: 1 x 2 + 1 = 3

2^7	**2^6**	**2^5**	**2^4**	**2^3**	**2^2**	**2^1**	**2^0**
128s	64s	32s	16s	8s	4s	2s	Units
0	0	0	0	1	0	1	0

We would read this as: 1 x 8 + 1 x 2 = 10

2^7	**2^6**	**2^5**	**2^4**	**2^3**	**2^2**	**2^1**	**2^0**
128s	64s	32s	16s	8s	4s	2s	Units
0	0	1	0	0	0	1	1

We would read this as: 1 x 32 + 1 x 2 + 1 = 35

2^7	**2^6**	**2^5**	**2^4**	**2^3**	**2^2**	**2^1**	**2^0**
128s	64s	32s	16s	8s	4s	2s	Units
1	1	1	1	1	1	1	1

We would read this as: 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1= 255 (2^8 = 256)

Since binary code can only consist of two values, 0 and 1, transmitting binary code is much easier and less prone to error than using a decimal system where the numbers 0 to 9 would have to be represented by 10 different voltage levels. Although this may seem counter-intuitive to us who are used to the decimal system, binary arithmetic rules are actually much simpler than decimal ones, and therefore make processor design easier.

Example of binary to decimal conversion

Look at the method used to convert binary to decimal in the interaction below. When you have finished this task, see if you can answer the questions which follow by using the method you have learned.

Convert 110 1101_2 to base 10:

Step 1:

2^6	2^5	2^4	2^3	2^2	2^1	2^0

Step 2:

2^6	2^5	2^4	2^3	2^2	2^1	2^0
64	32	16	8	4	2	1

Step 3:

2^6	2^5	2^4	2^3	2^2	2^1	2^0
64	32	16	8	4	2	1
1	1	0	1	1	0	1

Step 4:

(1 x 1) + (0 x 2) + (1 x 4) + (1 x 8) + (0 x 16) + (1 x 32) + (1 x 64)

= 1 + 0 + 4 + 8 + 0 + 32 + 64

=109

110 1101_2 = 109_{10}

Q5: Convert 111 0110_2 to decimal

....

Q6: Convert 011 0110_2 to decimal

....

Q7: Convert 11 1101 1010_2 to decimal

....

2.3.2 Converting decimal to binary

Learning Objective

By the end of this section you will be able to:

- convert between decimal and binary.

Method 1:

Find the highest column value which is less than the number to convert from decimal to binary. Subtract it and then find the next largest power of two less than the remainder. Keep repeating the same thing with the remainder until you end up with a 1 or a zero in the units position.

Convert 113 to binary:

$2^6 = 64$

113 - 64 = 49

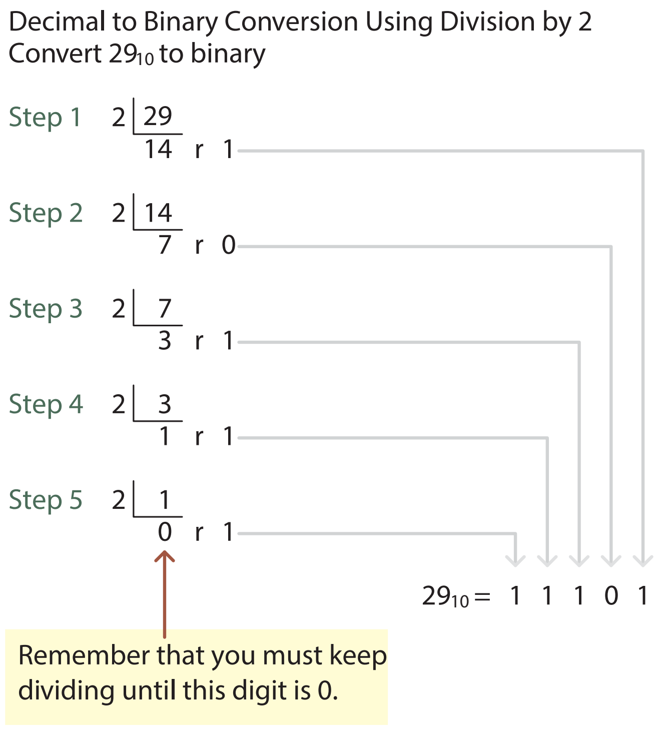
$2^5 = 32$

49 - 32 = 17

$2^4 = 16$

17 - 16 = 1

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

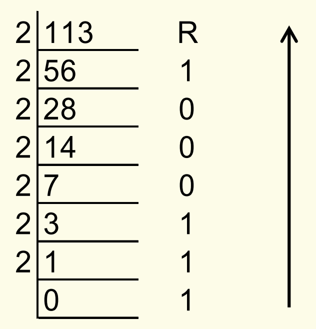

Decimal to Binary Conversion Using Division by 2
Convert 29_{10} to binary
Step 1 2 | 29 14 r 1
Step 2 2 | 14 7 r 0
Step 3 2 | 7 3 r 1
Step 4 2 | 3 1 r 1
Step 5 2 | 1 0 r 1
29_{10} = 1 1 1 0 1
Remember that you must keep dividing until this digit is 0.

To convert 29 into binary:

Method 2:

Keep dividing by 2 and writing down the remainder until you are left with 0 then read the answer from the bottom up.

Convert 113 to binary
= 1110001



Example of decimal to binary conversion

Look at the method used to convert binary to decimal in the interaction below. When you have finished this task, see if you can answer the questions which follow by using the method you have learned.

1. Write down the column values at the top of the page.

2^4	2^3	2^2	2^1	2^0
16	8	4	2	1

2. Select the biggest column value that is not greater than your chosen number (in this case 16 is the largest value $<$ 29).

3. Put a 1 in the 16s column.

2^4	2^3	2^2	2^1	2^0
16	8	4	2	1
1				

4. Subtract 16 from 29, $29 - 16 = 13$.

5. Start again with the number 13.

6. Select the biggest column value that is not greater than 13 (in this case 8).

7. Put a 1 in the 8s column.

2^4	2^3	2^2	2^1	2^0
16	8	4	2	1
1	**1**			

8. Subtract 8 from 13, $13 - 8 = 5$.

9. Start again with the number 5.

10. Select the biggest number that is not greater than 5 (in this case 4).

11. Put a 1 in the 4s column.

2^4	2^3	2^2	2^1	2^0
16	8	4	2	1
1	**1**	**1**		

12. Subtract 4 from 5, $5 - 4 = 1$.

13. Start again with 1.

14. Select the biggest position value that is not greater than 1 (in this case 1).

15. Put a 1 in the 1s column.

2^4	2^3	2^2	2^1	2^0
16	8	4	2	1
1	**1**	**1**		**1**

16. Subtract 1 from 1, $1 - 1 = 0$.

17. Put a 0 in the empty columns.

2^4	2^3	2^2	2^1	2^0
16	8	4	2	1
1	**1**	**1**	**0**	**1**

$29_{10} = 11101_2$

Q8: Convert 134_{10} to binary

. . . .

Q9: Convert 148_{10} to binary

. . . .

Q10: Convert 394_{10} to binary

. . . .

Decimal to binary conversion

On the Web is a simulation of the repeated division method used to convert a decimal number to binary. Using this interactivity you can enter your own numbers and test the conversions. You should now look at how this method works and then use it to convert from decimal to binary the list of numbers which follow.

. . . .

Using both methods to convert decimal to binary

Convert the following decimal numbers to binary.

Q11: 29_{10}

....

Q12: 18_{10}

....

Q13: 79^{10}

....

Q14: 273_{10}

....

Q15: 127_{10}

....

Q16: 742_{10}

....

Q17: 4023_{10}

....

Q18: 9755_{10}

....

2.3.3 Hexadecimal (optional)

It is very easy to make mistakes when reading long binary numbers, so you will often see them represented in hexadecimal. Hexadecimal code is just a number system which uses base 16 instead of base 10 or base 2. Because hexadecimal uses 16 digits instead of 10, we use the letters A to F to represent the decimal values 10, 11, 12, 13, 14 and 15.

16^2	16^1	16^0
256s	16s	1s
0	4	5

Would be read as: 4 x 16 + 5 x 1 = 69

It is very easy to convert base 2 numbers into base 16 numbers - just group the binary value into groups of 4 digits and then write them in base 16. This means that hex is a shorthand way of writing binary. Early computing scientists who had to work in binary, would use hex, as it's easier to remember than a string of 1s and 0s.

Examples

1.

Problem:

1111 0111 (247 in base 10)

Solution:

1111 = 15 in base ten and 0111 = 7 in base 10, so 1111 0111 = F7 in base 16

. . . .

2.

Problem:

1011 0011 (179 in base 10)

Solution:

1011 = 11 in base ten and 0011 = 3 in base 10 so 1011 0011 = B3 in base 16

. . . .

You may see memory locations or colour codes represented as hexadecimal numbers rather than binary because large binary numbers are so difficult to read and remember.

Exercise: Converting numbers

Load up the calculator on your computer and experiment with converting numbers between binary, decimal and hexadecimal.

. . . .

2.3.4 Storing negative integers

Learning Objective

By the end of this section you will be able to:

- understand how computers store integers using two's complement notation.

Remember: all information handled by computers is represented as patterns of 1s and 0s. We've now seen how any whole number can be represented by binary - but how about integers, which include a sign (negative and positive). How do we represent that in binary? Because all information handled by computers is represented as patterns of 1s and 0s.

One possibility might be to use an additional bit to represent whether the integer is negative or positive, a zero signifying a positive number and a 1 signifying a negative one. for example using 4 bits the numbers -1 to -7 could be stored like this:

+7	+6	+5	+4	+3	+2	+1	0	-1	-2	-3	-4	-5	-6	-7
0111	0110	0101	0100	0011	0010	0001	0000	1001	1010	1011	1100	1101	1110	1111

There are two problems with this method: we end up with two values, 1000 and 0000 which both mean zero, and using this system to add a positive number to a negative number gives the wrong answer as you can see from this example.

```
1011   -2
0100   +4   +
-----------
1111   -7
```

The solution is to make the value of most significant bit negative instead of just representing a + or a - sign. In this example the most significant bit has a value of -4.

-4 2 U	
010	+2
001	+1
000	0
111	-1
110	-2
101	-3

Addition now works when adding a positive and a negative number together and there is only one code for zero.

```
-4 2 U
 011      3
 100     -4   +
---------------
 111     -1
```

This system of representing integers is called **two's complement**. In two's complement notation, the most significant bit always has a negative value. This means that positive integers always start with a zero and negative integers start with a one. For this reason it is important to know how many bits are being used to represent a number in two’s complement.

Quiz: Two's complement

Q19: What two things can you tell about this two's complement number at a glance? 1000 1011

. . . .

Range of numbers represented by two's complement:

Since one of the bits is now representing a negative value, the range of numbers you can store using two's complement representation is going to be placed on either side of zero rather than from zero upwards.

+7	+6	+5	+4	+3	+2	+1	0	-1	-2	-3	-4	-5	-6	-7	-8
0111	0110	0101	0100	0011	0010	0001	0000	1001	1010	1011	1100	1101	1110	1111	1000

Storing positive integers only using an 8 bit binary code would give you the range of 0 to 2^8 ie. the range 0 to 255.

In binary this would be 0000 0000 to 1111 1111

However storing positive and negative integers using an 8 bit two's complement notation gives you a range -2^7 to 2^7 -1 ie. the range -128 to 127.

In two's complement notation this would be: 1000 0000 to 0111 1111

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
-128s	64s	32s	16s	8s	4s	2s	Units
1	**0**	**0**	**0**	**0**	**0**	**0**	**0**

= -128

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
-128s	64s	32s	16s	8s	4s	2s	Units
0	**1**	**1**	**1**	**1**	**1**	**1**	**1**

64 + 32 + 16 + 8 + 4 + 2 + 1 = 127

2.3.5 Converting two's complement numbers into decimal

Learning Objective

By the end of this section you will be able to:

- convert between two's complement notation and decimal integers.

$\mathbf{2^7}$	$\mathbf{2^6}$	$\mathbf{2^5}$	$\mathbf{2^4}$	$\mathbf{2^3}$	$\mathbf{2^2}$	$\mathbf{2^1}$	$\mathbf{2^0}$
-128s	64s	32s	16s	8s	4s	2s	Units
0	**0**	**0**	**0**	**0**	**0**	**1**	**1**

Would be read as: 2 + 1 = 3

$\mathbf{2^7}$	$\mathbf{2^6}$	$\mathbf{2^5}$	$\mathbf{2^4}$	$\mathbf{2^3}$	$\mathbf{2^2}$	$\mathbf{2^1}$	$\mathbf{2^0}$
-128s	64s	32s	16s	8s	4s	2s	Units
1	**0**	**0**	**0**	**0**	**0**	**1**	**1**

Would be read as: -128 + 2 + 1 = -125

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
-128s	64s	32s	16s	8s	4s	2s	Units
1	**1**	**0**	**1**	**0**	**0**	**1**	**0**

Would be read as: -128 + 94 + 16 + 2 = -46

2.3.6 Converting negative decimal Integer into two's complement

Learning Objective

By the end of this section you will be able to:

- convert between a decimal integer and two's complement notation.

There are 4 steps needed to convert a negative decimal integer to two's complement:

1. Establish the bit length required
2. Convert the positive version of the number to binary
3. Complement the binary number (ie. convert all 0s to 1s and vice versa)
4. Add 1

Examples

1. Convert -3 to 8 bit two's complement:

Problem:

1.	+3 =	0000 0011
2.	Complement	1111 1100
3.	Add 1	+1
4.	**Result**	**1111 1101**

Solution:

We can check this result: -128 + 64 + 32 + 16 + 8 + 4 + 1 = -3

. . . .

2. Convert -15 to 8 bit two's complement:

Problem:

1.	+15 =	0000 1111
2.	Complement	1111 0000
3.	Add 1	+1
4.	**Result**	**1111 0001**

Solution:

We can check this result: -128 + 64 + 32 + 16 + 1 = -15

. . . .

3. Convert -35 to 8 bit two's complement:

Problem:

1.	+35 =	0010 0011
2.	Complement	1101 1100
3.	Add 1	+1
4.	**Result**	**1101 1101**

Solution:

We can check this result: -128 + 64 + 16 + 8 + 4 + 1 = -35

. . . .

Simulation of two's complement representation

On the web is a simulation of the conversion of a 4-bit binary number to two's complement. You should now look at this simulation.

. . . .

2.4 Storing real numbers

Learning Objective

By the end of this section you will be able to:

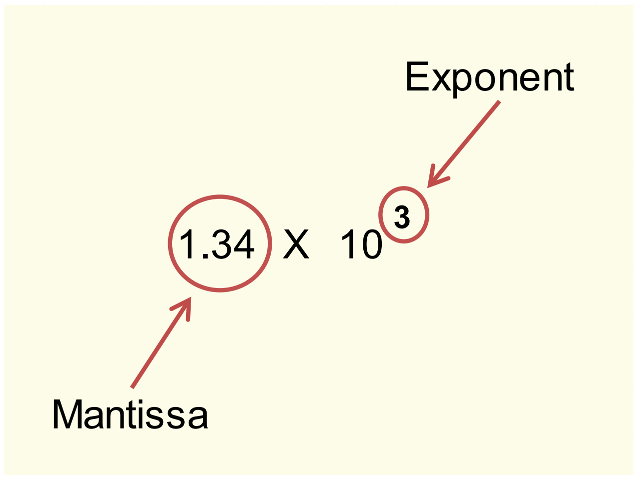
- understand how computers store real numbers using floating point notation.

A real number is a number with a decimal point. A real number can be positive or negative.

Decimal representation of real numbers uses a system known as scientific notation also known as standard form. Scientific notation allows us to represent very large or very small numbers using a short-hand where the number is represented by a value multiplied by a power of 10.

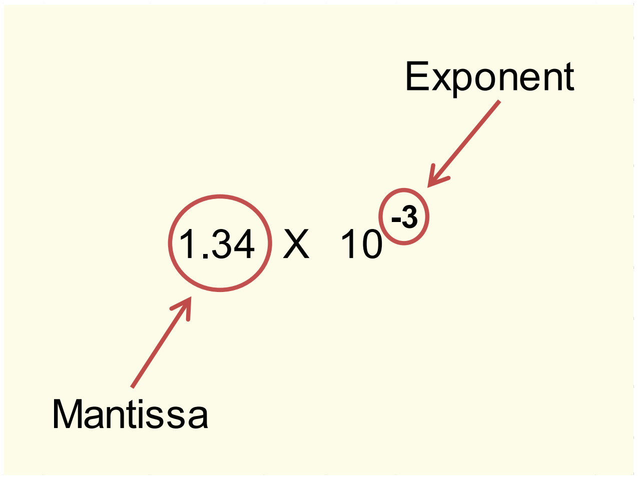
Scientific notation consists of three parts, the **mantissa** - a decimal number between 1 and 10 and the **exponent** - an integer representing a power of ten.

1340 can be represented as 1.34 X 10_3



The exponent can be positive or negative

0.00134 can be represented as 1.34 X 10^{-3}



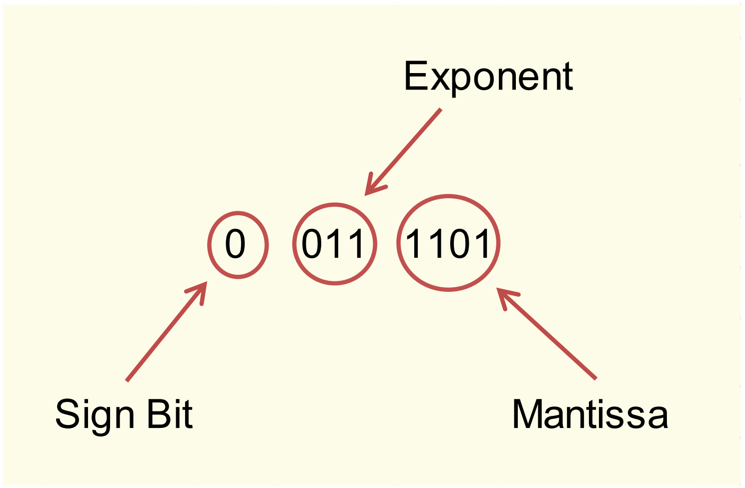
When very large or very small numbers are being represented then this system stores them with a limited degree of accuracy. This is an acceptable compromise because very large or very small measurements will not always be accurate to the same number of significant figures as would be needed to represent them as normal base 10 values.

Just as we have a decimal point, the binary point works in the same way:

2^3	2^2	2^1	2^0	*	2^{-1}	2^{-2}	2^{-3}
8s	4s	2s	Units	*	1/2s	1/4s	1/8ths
0	0	0	1	*	0	1	1

Would be read as: 1 + 0.5 + 0.25+ 0.125 = 1.875

Computers use a similar system to scientific notation to store real numbers. This is known as **floating point** representation.



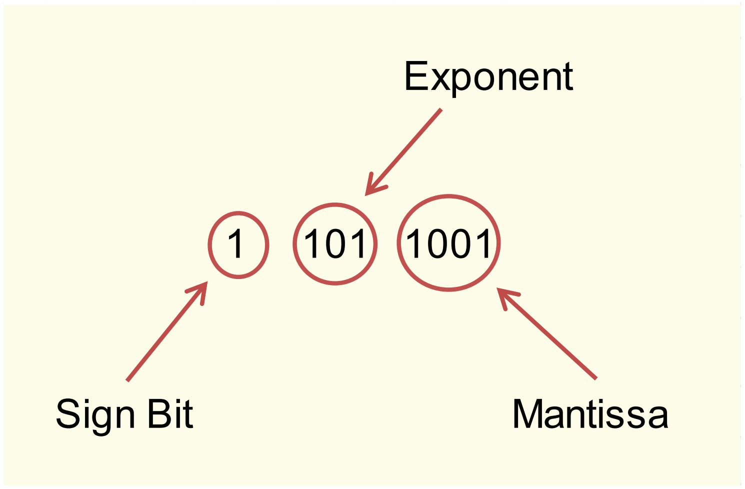
- The **sign bit** is a single bit, zero for positive and 1 for negative.
- The **Exponent** is a power of two stored in a system similar to **two's complement** notation because it can be positive <u>or</u> negative.
- The **Mantissa** is a positive binary fraction.

In this example the number represented is:

+2^3 x .1101

= 110.1

= 6.5



In this example the number represented is:

-2^{-3} x .1001

= -0001001

= -0.0351562

A **floating point** number takes up more memory and requires more processing power to calculate than a two's complement number and may also be less accurate.

Range of numbers represented by floating point:

The number of bits allocated to the **exponent** determines the **range** of numbers you can store. The exponent is an integer stored as a floating point number. If there were 8 bits allocated to the exponent this would mean that it could represent a range of numbers between 2^{127} and 2^{-128}

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

= 127

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

= -128

When a set number of bits is allocated to storing floating point numbers, there will always be a trade off between the range of numbers that can be stored (the **exponent**) and the accuracy with which they are stored (the **mantissa**). Floating Point Notation will always be a compromise between how accurately you can store a number and how large a range of numbers you wish to store.

You do not need to be able to calculate floating point numbers in Higher Computing Science, but you do need to know how real numbers are represented, and how the number of bits allocated to the exponent and mantissa affect this.

2.5 Storing text

Learning Objective

By the end of this section you will be able to:

- understand that text can be stored as ASCII code or as Unicode.

With any system of storing data it is important to have a common standard. Common standards for storing data are important because it makes exchanging information between computer systems easier. It also makes the transfer of data between applications easier. Modern computer systems use two common standards for representing text: **ASCII** and **Unicode**. The original standard for storing text was one of the earliest to be developed and actually began with a system of electric typewriters called tele-printers which could be connected together using a telephone line so that text entered on one machine would be duplicated on the other.

Please see the following for an example of an early tele-printer: British Pathe Teleprinter 1932

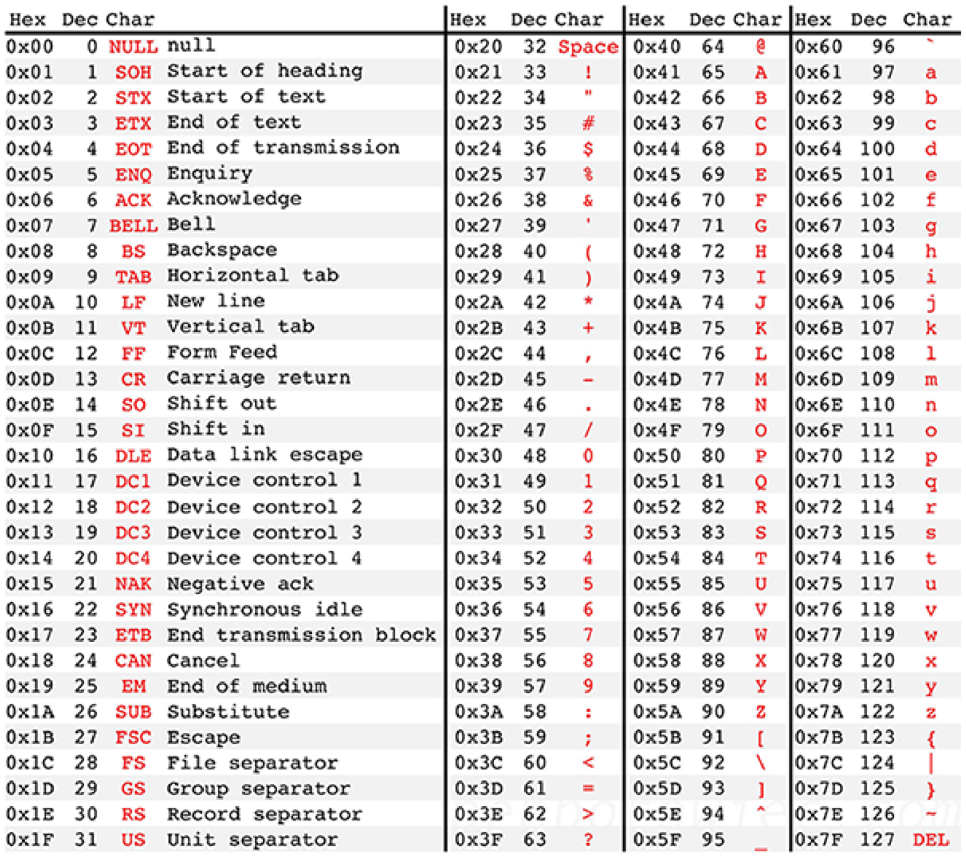
The code used in the days of tele-printers was the American Standard Code for Information Interchange (ASCII) and used a 7 bit code. Since every bit can only be either 1 or 0 then the total possible number of codes using 7 bits is 2^7 which makes 128 possible characters. There is still a legacy of the early tele-printer codes within the set of ASCII characters. Codes 0 to 31 are control characters which were originally used to set tabs, take a new line, move the print head back to overwrite a character etc. These **control characters** are still used today, for example the ASCII character produced by the enter key is still called a carriage return because it moved the print-head back to the left hand side and rolled the paper up one line.

Parity: simple error detection

128 codes were enough to represent all the characters on the keyboard, and an additional bit was added as a form of error detection called the **parity bit** making each character 8 bits or 1 byte in size. The parity bit worked by adding a 1 or a zero to ensure that the ASCII code always had an odd number of 1s in it. If a code arrived at the receiving computer with an even number of 1s then it meant that an error had occurred and the machine would ask for that character to be re-transmitted. Nowadays transmitting data is much more reliable so often the parity bit is used instead to extend the number of characters which can be represented giving 256 (2^8) possible combinations.

Binary ASCII codes are not easy to read, so for this reason most programming languages will have built-in functions which will convert a character to the decimal equivalent of the ASCII code and vice versa.

Examples of ASCII codes for alphanumeric characters



Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

Now that computers and computer programs are used internationally, there is a need for more than 256 characters to account for foreign alphabets and scripts. Unicode uses 16 bits to code text, giving 2^{16} = 65536 different possible characters. Unicode allows for all languages/character sets to be encoded.

- One disadvantage of Unicode is that a character takes up more memory than an ASCII code character.
- One disadvantage of ASCII code characters is that they are limited to 256 possible symbols (224 if control characters are excluded).

2.6 Storing graphics

Learning Objective

By the end of this section you will be able to:

- understand how computers store graphics as bitmaps and as vector graphics.

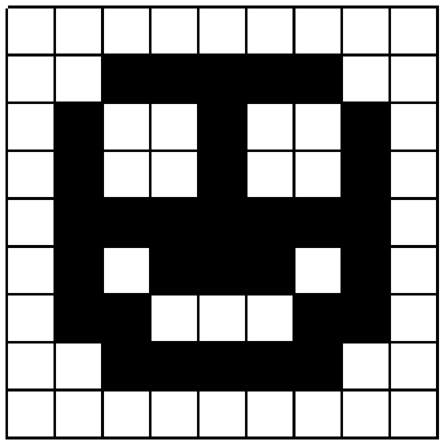
Just as with text, it is important that computers use a standard format for storing graphics because images are commonly used in web pages and other documents which users need to share. We are going to look at two methods of storing graphics:

- bitmapped graphics
- vector graphics.

2.6.1 Bitmapped graphics

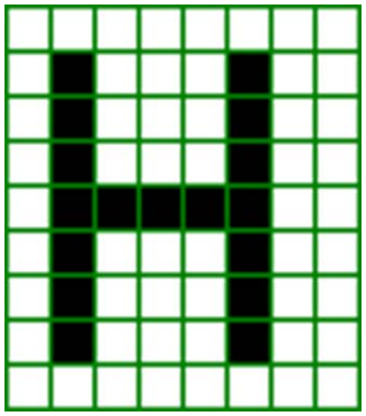
A **bitmap** is a representation of a graphic using a grid of bits to store the information about the colour of each pixel in an image. Bitmapped images appear as pixels. Each pixel corresponds (or maps) to one or more bits in memory - hence the terms bet-mapped. A bitmap for a black and white image would be a simple grid with each bit representing whether the pixel was black (off) or white (on). For example this shows how this image would be stored using a 9 X 9 bitmap:

0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	0	0
0	1	0	0	1	0	0	1	0
0	1	0	0	1	0	0	1	0
0	1	1	1	1	1	1	1	0
0	1	0	1	1	1	0	1	0
0	1	1	0	0	0	1	1	0
0	0	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0



Calculating memory requirements (black and white)

This diagram shows a small section of a black and white screen. Each pixel can be represented by a single bit (0 or 1).



The number of pixels possessed by a device can be found by multiplying the number of pixels across the screen by the number of pixels down, so for a display of 80 pixels across by 36 pixels down, the number of pixels will be $36 * 80 = 2880$ pixels.

2880 pixels, with one byte representing 8 pixels will thus require $2880/8 = 360$ Bytes of memory.

Pixels across	**Pixels down**	**Total**	**Bytes of memory needed**
80	36	$80 * 36 = 2880$	$2880/8 = 360$ Bytes

Calculating memory requirements

How much memory will the following monochrome screens require? Use the headings in the table below to help you perform the calculation.

Q20: 640 * 200 pixels

....

Q21: 800 * 600 pixels

....

Q22: 1024 * 768 pixels

....

Pixels across	Pixels down	Total (pixels across * pixels down)	Bytes of memory needed (Total/8)

....

The bit-depth of an image refers to the number of bits used to store the colour information for each pixel.

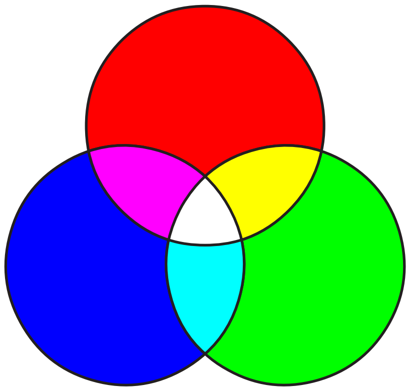
The colour information for a black and white image can be stored using one bit per pixel.

Allocating two bits per pixel would allow 4 possible colours to be stored. 3 bits allow 8 colours etc.

Experts agree that the human eye can distinguish up to 10 million colours (although an article in New Scientist in 2004 suggested that humans and other apes can distinguish around 2.3 million colours). If the true figure is indeed 10 million, then 24 bits per pixel would be required to represent this number (and significantly beyond, in fact). A bit depth of 24 bits (3 Bytes) per pixel would be required to store this amount of colour information. (2^{24} = 16777216)

2.6.2 RGB colour

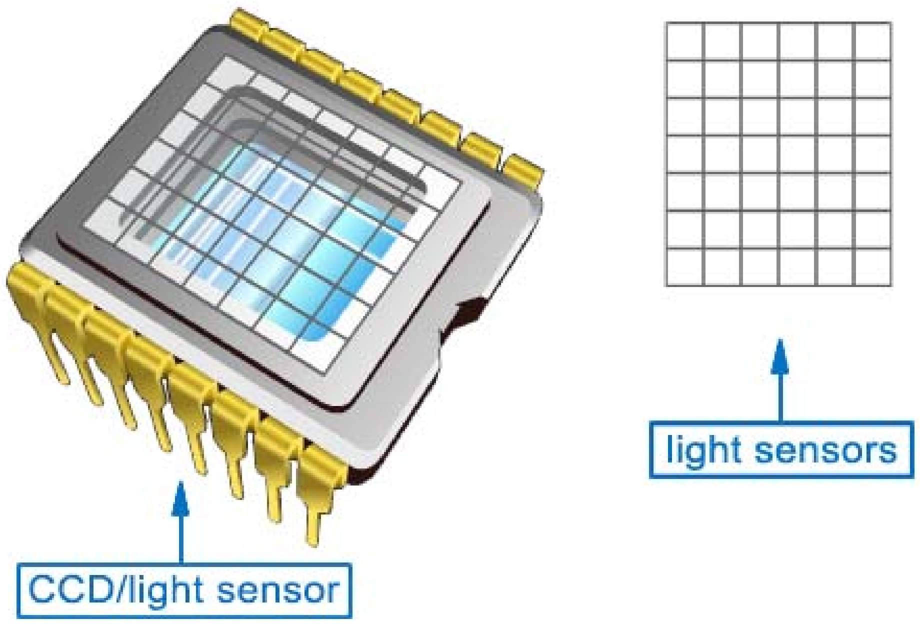
Colours on a screen can be displayed by mixing red, green and blue light. Where they overlap they create cyan, magenta, yellow and white.



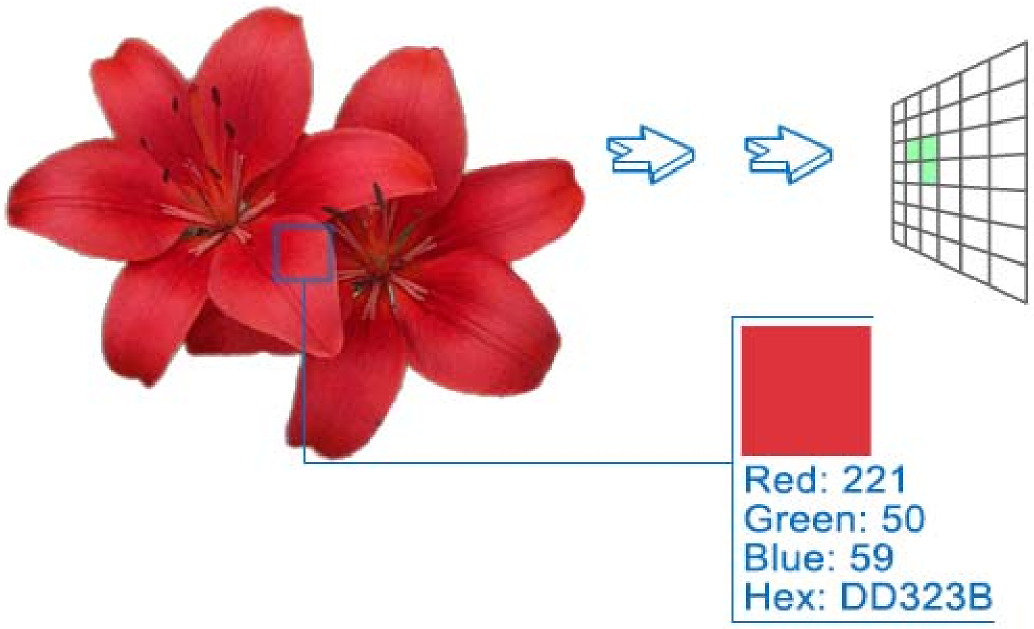
RGB colour uses 24 bits to represent the colour of each pixel. 8 bits for red, 8 bits for green and 8 bits for blue. These codes are often expressed in **hexadecimal** when they are used in programming languages or web authoring applications.

Red	**Green**	**Blue**	**Colour**	**Binary value**
00	00	00	black	0000 0000 0000 0000 0000 0000
FF	FF	FF	white	1111 1111 1111 1111 1111 1111
00	FF	00	green	0000 0000 1111 1111 0000 0000
00	80	00	half green	0000 0000 1000 0000 0000 0000

RGB colour codes



Light falls onto the sensors in the CCD, resulting in an electric signal.



This signal is then digitised (turned into a number).

The numbers for Red, Green and Blue are then combined to make a colour code for each pixel which is stored as a bitmap.

. . . .

2.6.3 Calculating bitmapped graphic file sizes

Learning Objective

By the end of this section you will be able to:

- calculate graphic image file sizes.

The **resolution** of an image refers to the total number of pixels which are used to store the image. The larger the number of pixels in an image the larger the number of bits needed to store that image therefore the larger the file size. The resolution of an image determines the amount of detail which it can display.

Bitmapped graphics are easy to edit because individual pixels can be changed from one colour to another, but when a bitmapped image is enlarged, it may become grainy because the resolution cannot be increased by just changing the size of the graphic.

To calculate the file size of an uncompressed bitmapped graphic, you need to know three things:

- the resolution (dots per inch);
- the area of graphic (square inches);
- the colour depth (bits per pixel).

Number of colours	**Bit Depth**
2 colours	1 bit
4 colours	2 bit
8 colours	3 bit
16 colours	4 bit
256 colours	8 bit
65536 colours	16 bit
16777216 colours	24 bit

Example

Problem:

Resolution: 600 dots per inch

Area of graphic: 2 x 3 inches

Colour depth: 256 colours = 8 bits

Solution:

600 x 600 x 2 x 3 = 2160000 pixels

2160000 x 8 = 17280000 bits

= 2160000 Bytes = 2109 KB = 2.05 MB

(8 bits = 1 Byte, 1024 Bytes = 1 KB, 1024 KB = 1 MB)

8 bits =	1 Byte
1024 Bytes =	1 KiloByte (KB)
1024 KiloBytes =	1 MegaByte (MB)
1024 MegaByte =	1 GigaByte (GB)

. . . .

2.6.4 Compression

Because graphic files can be very large, particularly if they are high resolution and have a large bit depth. Images are integral to modern web sites, and although download speeds have improved, bandwidth is still at a premium and web developers want their pages to load as quickly as possible. For this reason graphic images are often stored in compressed file formats such as Graphics Interchange Format (**GIF**), **JPEG** (JPG) or **Portable Network Graphics** (PNG). GIF and PNG are lossless compression formats, but GIF is limited to 8 bit colour and PNG to 24 bit colour. JPEG is a lossy compression format used primarily for photographic images.

Compression techniques will be covered in more detail in Unit 2.

Q23: Calculate the size in bytes of a bit map image of dimension 4 x 3 inch square using 72 dots per inch resolution and a pixel depth of 1 bit.

. . . .

Q24: How many colours can be represented using a pixel depth of 16 bits?

. . . .

Q25: A 3 inch by 2 inch monochrome image has a resolution of 300 dots per inch. How big will the file be (in appropriate units)?

. . . .

Q26: A graphic 3 inches by 2 inches at 600 dots per inch uses 256 colours. How much storage space is needed?

. . . .

Q27: A digital photo is 2592 x 1944 pixels and uses RGB colour: 8 bits for red, 8 bits for green and 8 bits for blue i.e. 24 bits per pixel. How big is the file?

. . . .

Example : Calculating the file size of a video

Problem:

An animation contains 1 minute of film. It is displayed in a frame size of 5 X 4 inches at a rate of 25 frames per second using a resolution of 300 dpi and colour depth of 24 bits. Assuming no compression, calculate the total storage requirements for this animation in MegaBytes.

Solution:

Each dot (or pixel) requires 24 bits or 3 Bytes.

There are 300 x 5 x 300 x 4 dots on each frame.

Each frame requires 300 x 5 x 300 x 4 x 3 = 5 400 000 Bytes storage.

There are 25 x 60 frames in the 1 minute clip giving 1500 frames in total.

Total storage is 1500 x 5 400 000 requiring 8 100 000 000 Bytes of storage.

Or 8 100 000 000/ 1024 x 1024 MB.

Equals 7724.76 MB Or 7.54 GB.

Hence the need for compression!

. . . .

2.6.5 Vector graphics

Vector graphics is an alternative method of storing graphics which stores them as a description of the shapes in the image to enable them to be re-drawn for recreating that image rather than storing them as individual pixels. There are two advantages of this system: vector graphics tend to take up much less disk space, and they do not lose resolution when enlarged, since their resolution depends on the device they are being displayed on rather than the device which created them. Because they are stored as a description, the disk space taken up by a vector graphics image will depend on the complexity of the image.

For example a rectangle could be stored as a set of attributes: start point coordinates, width, height, line colour, line thickness etc. A circle could be stored as a centre point coordinates, a radius, line colour, line thickness, fill pattern etc.

Because images stored as vector graphics use descriptions to recreate the image, they are not editable at pixel level in the same way as bitmapped images. Objects in a vector graphics application can be moved and resized independently of each other, whereas objects in an image created in a bitmapped graphics application will be stored as a number of pixels and cannot be manipulated as separate entities. Many bitmapped graphics applications use the concept of layers to get around this restriction. Each layer is a separate bitmapped image and each layer can be edited individually. Once the image is saved as a single bitmap however it can only be edited at pixel level.

Comparing the file size of bitmapped and vector graphics

20 min

1. Create a simple image such as a circle intersecting a rectangle in a bitmapped painting package. Note the size of the file when saved.
2. Create the same image as far as possible using a vector graphics package and compare the size of the saved file with that created by the bitmapped package.
3. Edit the bitmapped image to add more detail and save it. Does the file size change?
4. Add some more detail to the vector graphics file and see if the file size changes when you save it.

5. Explain your findings.

. . . .

Type	**Advantages**	**Disadvantages**
Vector	Can be scaled to large sizes, keeping original quality.	Difficult to create realistic images.
Suitable for graphic, unrealistic images and designs.	Individual objects can be edited, allowing an object to be altered without affecting the rest of the image.	Only individual objects can be edited (it is sometimes impossible to edit only part of the object).
	Are easily converted to bitmap formats.	Dependent on output hardware or software for appearance & quality.
	File sizes are relatively small.	
	Size of image can be increased keeping quality and file size the same.	
Bitmap	Images can be very realistic (e.g. digital photograph).	Scaling causes pixellation.
Suitable for natural, hand-drawn looking, realistic images.	Pixel level editing is allowed - allowing effects such as spray paint, blur, effects and so on.	Only the image as a whole can be edited.
	Same appearance in all systems, regardless of hardware or software.	Are very difficult to convert to vector formats, with unpredictable results
		File sizes can be large.
		Increasing the image size needs re-sampling and increases the file size.

2.7 Storing sound

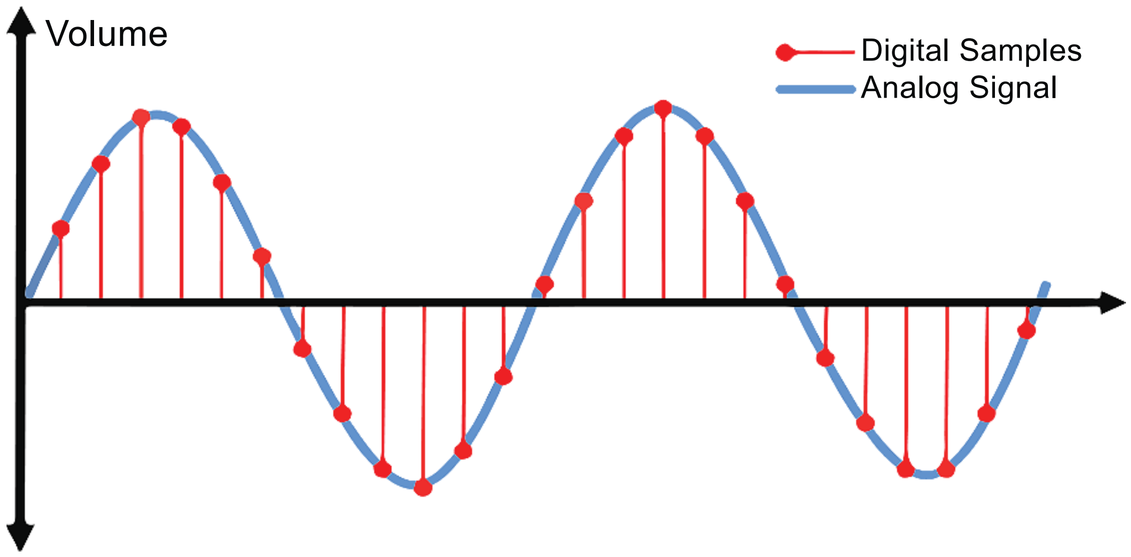
Learning Objective

By the end of this section you will be able to:

- explain how computers store sound data;
- explain the need for data compression.

Just as bitmapped graphics files have a resolution and a bit depth, the quality of a digital sound file depends on the sample rate (the number of times the value of the signal is recorded) and its bit depth (the number of bits used to store the sample). Just like graphics files, digital sound files can be very large indeed, so compression is often used to reduce the file size.

When an analogue signal is converted into a digital one, a digital sample is taken often enough to accurately reproduce the sound on a digital device such as a MP3 player or computer. The sample must be taken often enough and the sample quality must be high enough so that the human ear cannot distinguish between the analogue and the digital version. The benefit of storing sound digitally is that reproduction becomes error-free because every bit is either on or off and error detection and error correction can be built into the transmission process. Analogue recordings lose quality every time they are copied. Digital recordings are identical every time they are copied. The fact that digital media can be copied and transmitted easily without degradation in quality has resulted in enormous changes in the way music and other digital data is sold and distributed.



As a sound is played, digital signals have to be converted to analogue in order for us to hear it. The bit rate is the number of bits each second that have to be processed in order for a digital sound to be played.

If the sound is high quality, then there will be a greater number of bits as there will be a greater number of samples each second to be converted back to analogue plus each sample itself will be a larger number of bits as the sampling depth will be greater.

If a digital sound is being streamed over a computer network then the number of bits each second is important. If the number of bits each second is very high then this will also place demands on the computer hardware, as it all needs to be processed.

The bit rate for sounds can be calculated as follows:

Bit Rate (bits per second) = sampling depth (bits) * sampling frequency (Hz)

Consider the following examples:

Examples

1.

Problem:

A CD quality sound recorded using a sampling frequency of 44KHz and a sampling depth of 16 bits.

Solution:

Bit rate = 44000 * 16

Bit rate = 704000 bits per second

Bit rate = 704K bits per second

. . . .

2.

Problem:

A low quality sound recording using a sampling frequency of 2KHz and a sampling depth of 8 bits.

Solution:

Bit rate = 8 * 2000

Bit rate = 16000 bits per second

Bit rate = 16K bits per second

. . . .

This allows you to select a quality setting based on the speed that the data could be transferred, rather than the end size of the entire file - which is much better for time-dependent data like sound or videos.

2.7.1 MIDI format

Sound can also be stored using the audio equivalent of vector graphics. The **MIDI** file format stores music as a description of the components necessary for re-creating music rather than as a direct digital representation. MIDI files will contain details of instruments used, timings, volume and frequency of notes, etc. Like vector graphics, the quality of the sound output is determined by the quality of the device it's being played on. A high-end keyboard will sound much more realistic than a cheap sound card.

2.8 Storing video

Learning Objective

By the end of this section you will be able to:

- explain how computers store video data.

Storing video requires both images and sound to be stored together. Video files will be very large as a result so compression is always used unless the video is being recorded for editing later.

2.8.1 MPEG

The Moving Pictures Experts Group (MPEG) have defined a series of standards for compressing video and audio using compression based on DCT (Discrete Cosine Transform). Each frame in an MPEG video is compressed as a JPEG. The data that stays the same in successive frames is then removed.

There have been a series of standards based on this:

MPEG-1 (**VHS** video quality with 353 x 240 pixels and 30 fps frame rate support)

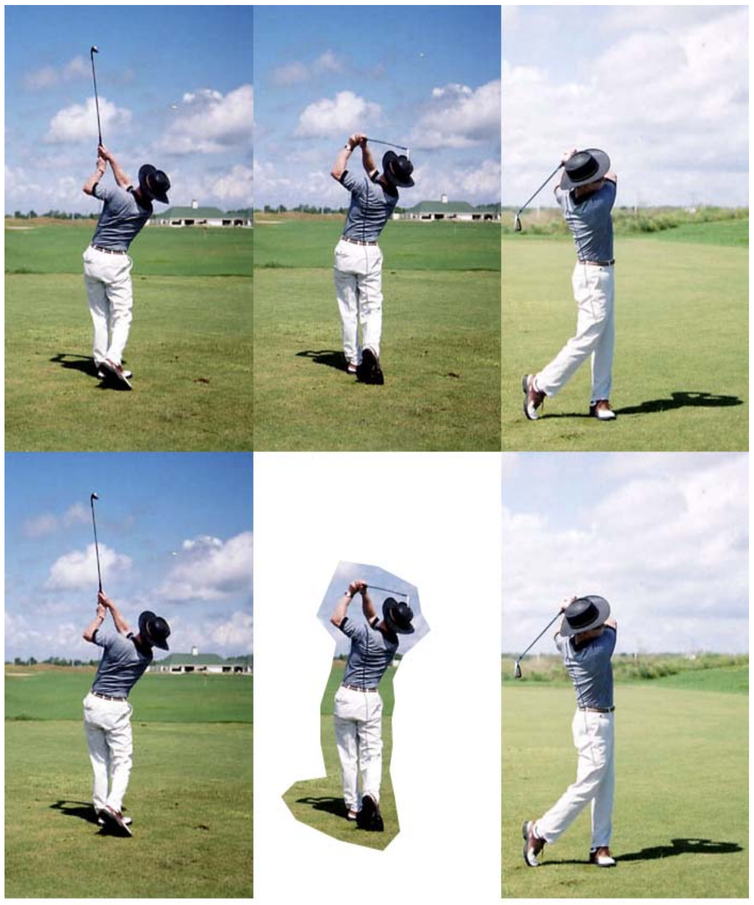
MPEG-2 (The standard for DVD-Video and Digital Television -to name two. Widely used)

MPEG-3 (Intended for HDTV but these revisions were incorporated into MPEG-2) (Not the same as MPEG-Layer 3, or MP3 used for audio - this is actually the audio subset (layer) of the MPEG-1 and MEG-2 standards)

MPEG-4 (Designed for low-bandwidth networks - e.g. video phones) (Part used by DivX)

MPEG-7 (Builds on the interactive and extra data capabilities of MPEG-4 and is a full multimedia description format) - named "Multimedia Content Description Interface".

Not all frames are stored - just a few key frames called 'i-frames'. These are JPEGs. The next set of frames does not store images, they just store data on what has changed since the last i-frame.


First (i) frame which is the base for the following (b) frames
A scene which can use most of the information from the first
A new scene which requires a whole new (i) frame

MPEG does not store each image separately, only key frames are stored as JPEG images, the rest of the data consists of predictions or actual changes since the last (or next!) key frame.

MPEG is a lossy compression codec and, as with JPEG images and MP3, has adjustable compression depending on the desired quality, file size or bit-rate.

There are different implementations of the MPEG-4 codec (for example). The playback compatibility and compression/quality gained depends on the actual version of the codec that is being used.

2.8.2 Calculating the size of video files

Video file sizes depend on:

- Frame size: (for example. PAL = 720 x 576 pixels; HDMI = 1920$\times$120 pixels)
- Frame rate: measured in Frame per second rate (for example: PAL = 25fps HDMI = 60fps)
- Bit depth (PAL = 24 bit HDMI = 36 bit)

- Audio sample rate and bit depth
- Compression method (Codec)
- As video files are simply a collection of bitmap images, all we need to do is calculate the size of one image (i.e. one frame) and multiply that by the number of frames in the entire video.
- File Size (Bytes) = Frame Size (Bytes) x Frame Rate (frames per second [fps]) x Video Time (seconds).

Quiz: Video

Q28: UK PAL television is approximately: 720 x 576 pixels at 25fps in 24bit colour. Which of these bit-rates would be needed to broadcast this, uncompressed, over a digital network?

a) 52Kbps
b) 4Mbps
c) 32Mbps
d) 237Mbps

. . . .

Q29: A video is captured using 24fps and 600 x 400 pixels, colour depth 24bits. Which of these settings would half the file size of the uncompressed video?

a) 24fps, 600 x 400 pixels, 12 bits
b) 6fps, 300 x 200 pixels, 12 bits
c) 12fps, 600 x 200 pixels, 24 bits
d) 12fps, 300 x 200 pixels, 24 bits

. . . .

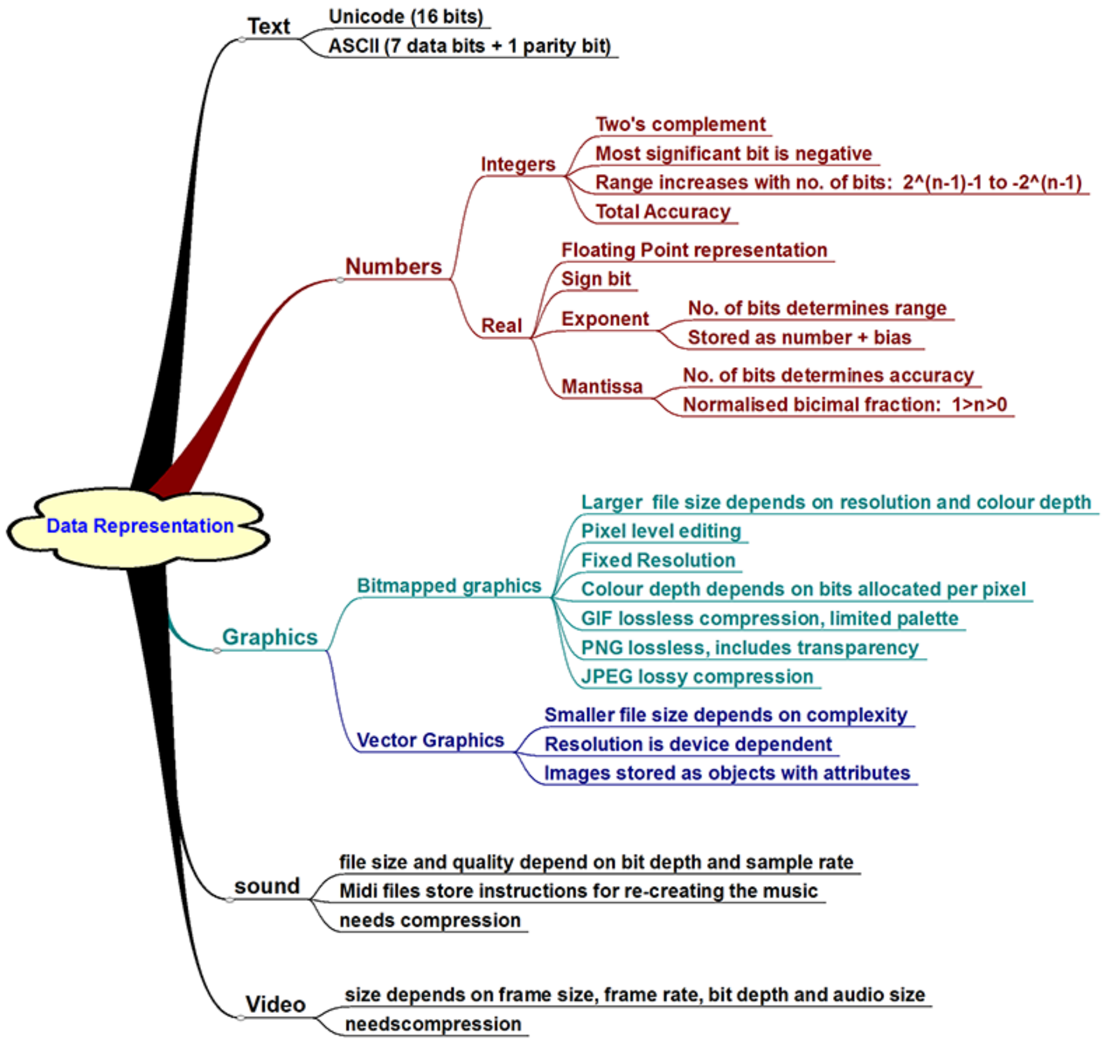
. . . .

2.9 Learning points

Summary

You should now know:

- integers and real numbers:
  - computers store all data as binary numbers (base 2, using 0s and 1s);
  - integers are stored using two's complement notation;
  - to convert binary into decimal, write the binary number under its place values, then add;
  - to convert decimal to binary, use repeated division by 2, and use the remainders read upwards;
  - to convert a negative integer into two's complement, write it as a positive binary number using the full bit length, invert it then add 1 real numbers are stored using floating point representation;
  - floating point representation uses a mantissa and exponent. The mantissa determines the accuracy of the number, the exponent determines the range of numbers which can be stored;
  - in a given number of Bytes, increasing the number of bits for the mantissa increases the precision of the number, but decreases the range;
- characters:
  - unicode is a 16-bit code for storing characters;
  - unicode uses more storage than ASCII, but allows a wider range of characters to be represented;
- graphics:
  - graphics can be stored using a bit-map, where each pixel is represented by one (or more) bits;
  - increasing the bit depth (number of bits per pixel) increases the number of colours which can be represented, but also increases the storage requirements;
  - vector graphic representation stores a graphic as a series of objects with attributes;
- sound and video:
  - sound files are stored digitally by sampling the analogue sound wave;
  - the size of a sound file depends on the frequency of the sample rate and the bit depth of each sample;
  - MIDI files have a parallel with vector graphics in that they contain instructions for reproducing the sound rather than the sound data itself;
  - uncompressed video file size depends upon the frame size, frame rate, bit depth and audio sample rate and bit depth;
  - audio and video files are often compressed as they can be very large indeed if recorded for high quality reproduction.


Data Representation
Text
Unicode (16 bits)
ASCII (7 data bits + 1 parity bit)
Numbers
Integers
Two's complement
Most significant bit is negative
Range increases with no. of bits: 2^(n-1)-1 to -2^(n-1)
Total Accuracy
Real
Floating Point representation
Sign bit
Exponent
No. of bits determines range
Stored as number + bias
Mantissa
No. of bits determines accuracy
Normalised bicimal fraction: 1>n>0
Graphics
Bitmapped graphics
Larger file size depends on resolution and colour depth
Pixel level editing
Fixed Resolution
Colour depth depends on bits allocated per pixel
GIF lossless compression, limited palette
PNG lossless, includes transparency
JPEG lossy compression
Vector Graphics
Smaller file size depends on complexity
Resolution is device dependent
Images stored as objects with attributes
sound
file size and quality depend on bit depth and sample rate
Midi files store instructions for re-creating the music
needs compression
Video
size depends on frame size, frame rate, bit depth and audio size
needscompression

2.10 End of topic test

End of topic test

Q30: The decimal number -73 (negative 73) can be represented in binary using two's complement by:

a) 1000 1101
b) 1011 0110
c) 1100 1001
d) 1011 0111

. . . .

Q31: Approximately, how much storage is required to store a scanned A4 image (8.25 x 11.25 inches) at 300dpi using a bit depth of 24 bits without data compression?

a) 200 MB
b) 85 MB
c) 23.9 MB
d) 0.7 MB

. . . .

Q32: How many colours can be stored per pixel if the bit depth is 24 bits?

a) 65536
b) 16777216
c) 4294967296
d) 1099511627776

. . . .

Q33: How many bits are used to store an ASCII code character?

a) 7 bits
b) 8 bits
c) 16 bits
d) 24 bits

. . . .

Q34: How many characters can Unicode represent if 16 bits are used per character?

a) 128
b) 256
c) 1024
d) 65536

. . . .

Q35: When giving more bits to mantissa and fewer to exponent, which is true?

a) Accuracy increases, range increases
b) Accuracy decreases, range decreases
c) Accuracy increases, range decreases
d) Accuracy decreases, range increases

. . . .

Q36: Increasing the complexity of a vector graphic image:

a) increases the file size
b) decreases the file size
c) alters the object's attributes
d) displays the object more clearly

. . . .

. . . .

Topic 3

Data types and structures

Contents

Prerequisite knowledge

From your studies at National 5 you should already know:

- *programming languages use a variety of simple data types including string, integer, real and boolean;*
- *a collection of values of the same data type can be stored using an array.*

Learning Objectives

By the end of this topic you should be able to:

- *understand that all programming languages store and manipulate data;*
- *explain the difference between a simple and a structured data type;*
- *understand the connection between simple data types and how computers store numbers and text;*
- *identify the different data types used by your chosen programming language;*

- *describe how your programming language handles sequential files;*
- *describe how your programming language handles records.*

Learning Objective

By the end of this topic you should be able to:

- understand that all programming languages store and manipulate data;
- explain the difference between a simple and a structured data type;
- understand the connection between simple data types and how computers store numbers and text;
- identify the different data types used by your chosen programming language;
- describe how your programming language handles sequential files;
- describe how your programming language handles records.

3.1 Revision

Revision

Q1: An integer is:

a) a number greater than zero
b) a negative or positive number including zero with no decimal point
c) a negative or positive number including zero with a decimal point
d) a single digit number

. . . .

Q2: A real number is:

a) a number greater than zero
b) a negative or positive number including zero with no decimal point
c) a negative or positive number including zero with a decimal point
d) a single digit number

. . . .

Q3: A Boolean is:

a) a value which can be either true or false
b) a very large number
c) a variable which can only have two possible values
d) a complex data type

. . . .

Q4: An array is:

a) a collection of values
b) a set of variables of the same type
c) a structured data type storing values of the same type
d) a list of values

. . . .

3.2 Data types and pseudocode

Learning Objective

By the end of this topic you should be able to:

- understand that all programming languages store and manipulate data.

All programming languages manipulate data. In this topic we are going to look at how general purpose imperative programming languages store and manipulate data.

Throughout this unit we will be using SQA standard **pseudocode** to describe data types and control structures. This means that you should be able to convert the examples into whatever programming language your school or college is using.

Data types can be divided into two categories: **simple** and **structured** types.

The simple data types are:

- INTEGER
- REAL
- CHARACTER
- BOOLEAN

The **structured data types** are:

- ARRAY
- STRING
- RECORD

3.3 Simple data types

Learning Objective

By the end of this topic you should be able to:

- understand the connection between simple data types and how computers store numbers and text.

We will be using the following **simple data types**:

- An INTEGER is a numerical value which has no decimal point. An INTEGER can be positive or negative including zero;
- A REAL is a numerical value which includes a decimal point;
- A CHARACTER is a single character from the keyboard or other input device;
- A BOOLEAN can have two values only: **true** or **false**.

These data types correspond to the various ways which computers store information at machine code level.

- INTEGERs are stored using **two's complement** notation;
- REAL numbers are stored using floating point notation which uses an **exponent** and a **mantissa**;
- CHARACTERs are stored as **ASCII** codes or if more than 128 are needed then they are stored using **Unicode**;
- A BOOLEAN can be stored using a single **bit** which is on or off.

Storing values using floating point notation is more memory and processor intensive than **two's complement**, and since there is always a trade-off between accuracy and range when storing values using floating point notation, it makes sense to store integer values as INTEGER rather than REAL.

All programming languages will have a have a maximum limit (positive and negative) for storing integers, and this limit is determined by the number of bits allocated to storing them.

If 32 bits were being used to store integers, then the range of possible values would be -2^{31} to 2^{31} - 1.

If 64 bits were being used to store integers, then the range of possible values would be -2^{63} to 2^{63} - 1. For this reason very large numbers are stored as REAL rather than INTEGER.

In maths and science, scientific notation is used to represent very large decimal numbers anyway. This is similar to the system of floating point used by computers.

5 min

Activity: Simple data types

Q5: Decide what simple data type you would use to store the following values. Choose from:

- INTEGER
- REAL
- CHARACTER
- BOOLEAN

No.	Value	Simple data type
1	304	
2	45.78	
3	@	
4	-4	
5	5989.4	
6	-56.3	
7	!	
8	true	

. . . .

3.4 Identifying simple data types

Learning Objective

By the end of this topic you should be able to:

- identify the different simple data types used by your chosen programming language.

30 min

Practical task: Simple data types

Look in the manuals, on the Internet or in the help documents for two of the programming languages in use in your school. List the simple data types available in your chosen languages.

. . . .

3.5 Structured data types

Learning Objective

By the end of this section you will be able to:

- explain the difference between a simple and a structured data type.

Most software applications require large amounts of data to be stored. If every single item of data had to be given a unique name then not only would this be very inconvenient, but accessing and manipulating these separate variables would be very complex.

For instance a set of numeric values could be stored by creating a set of variables:

```
SET number1 TO 23
SET number2 TO 16
SET number3 TO 3
SET number4 TO 77
SET number5 TO 23
```

They could be printed using this set of commands:

```
SEND  number1 TO DISPLAY
SEND  number2 TO DISPLAY
SEND  number3 TO DISPLAY
SEND  number4 TO DISPLAY
SEND  number5 TO DISPLAY
```

This system becomes very cumbersome indeed if we are manipulating large quantities of data. If a number of items of the same type have to be stored, it makes sense to store them in a structure which can be referred to by a single identifier, and to be able to access these items using a control structure like a loop to process them sequentially. A structure like this is called an **array**.

Storing data such as numbers in a single structure makes printing and searching through the list much easier because the **index** can be used to identify each one in turn.

Array	23	16	3	77	23	1	11	9	45	39
Index	0	1	2	3	4	5	6	7	8	9

In this example we could set up an integer array storing 10 items to store these numbers

VAR numbers [9]

So to print the contents of an array we could use the following code:

```
FOR counter FROM 0 TO 9 DO
   SEND  numbers[counter] TO DISPLAY
END FOR
```

We will be using the following structured data types:

ARRAY, STRING and RECORD.

3.5.1 Arrays

An ARRAY is an ordered sequence of simple data types, all of the same type.

[5, 9, 13] is an ARRAY storing three INTEGERs

["Fred","Sue","Jo","Anne"] is an ARRAY storing four STRINGS

Eg. [true, false, true, true] is an ARRAY storing four BOOLEANs

3.5.2 Strings

A STRING is a special sort of ARRAY containing CHARACTERs

Eg. "This is a message" is an example of a STRING

STRINGs can be joined together or embolden using the & symbol

So the command:

```
SET newString TO "Hello " & "This is a message"
```

creates a new STRING with the value: `"Hello This is a message"`

Although strictly speaking a string is a complex data type, some programming languages treat a string as a simple data type rather than as an array of characters.

The arrays we will be using are one dimensional, ie. they are equivalent to an ordered list of items, identified by a single **index**. The index of an ARRAY or STRING starts at zero.

For example, the following two commands:

```
SET myNames TO ["Fred","Jim","Betty"]
SEND myNames[2] TO DISPLAY
```

Would create an ARRAY of three STRINGS and then print out the item: Betty

```
SET myNumber  TO [56, 7, 23, -12]
SEND myNumbers[0] TO DISPLAY
```

Would create an ARRAY of three INTEGERS and then print out the value: 56

3.5.3 Records

A RECORD can contain variables of different types, just as a **record** in a database can be made up of fields of different types. So a single record would be equivalent to a list of items of different list of different types of item.

```
TYPE person IS RECORD
{STRING forename, STRING surname, STRING address, INTEGER score}
```

For example, a record for a single individual could be:

```
SET person1 TO
{Forename = "Fred", Surname = "Flintstone", Address = "Bedrock", score = 10}
```

A table in a database is equivalent to an ARRAY of RECORDS

Activity: Procedural language

Q6: Are the following Control Structures or Data Structures?

1. Arrays
2. Selection
3. Records
4. Iteration

. . . .

3.6 Handling records

Learning Objective

By the end of this section you will be able to:

- describe how your programming language handles records.

Practical task: Handling records

60 min

Find out the syntax your programming language uses to:

- Define a record structure
- Create an array containing 3 or more records
- Print out the contents of the array.

. . . .

3.7 Parallel arrays and records

If we want to store information about the real world, we often need to store information of different types.

For example if we wanted to store the results of a race, we would need to store both the participants' names as a STRING and their times as a REAL number. We might additionally want to store whether they qualified for the final or not as a BOOLEAN value.

We could store this information in 3 parallel arrays:

```
SET participants TO ["Fred", "Greg", "Ian", "Peter", "Zaphod", "Andy"]
SET times TO [4.56, 4.32, 5.01, 4.61, 4.55, 5.12]
SET qualified TO [false, true, false, false, true, false]
```

A more intuitive way of storing this information would be to store it as a set of records. In this case a single record would be the information about one participant - their name, time and qualifying status.

So we can store the race information as an ARRAY of RECORDs:

```
TYPE runner IS RECORD {STRING participant, REAL time, BOOLEAN qualified}
SET race[0] TO {participant = "Fred", time = 4.56, qualified = false}
SET race[1] TO {participant = "Greg", time = 4.32, qualified = true}
SET race[2] TO {participant = "Ian", time = 5.01, qualified = false}
SET race[3] TO {participant = "Peter", time = 4.61, qualified = false}
SET race[4] TO {participant = "Zaphod", time = 4.55, qualified = true}
SET race[5] TO {participant = "Andy", time = 5.12, qualified = false}
```

This loop will print the qualifiers:

```
FOR counter FROM 0 TO 5 DO

IF race[counter].qualified = true THEN
   SEND race[counter].participant & " has qualified for the final"
END IF

END FOR
```

Practical task: Parallel arrays and records

Use your chosen programming language to create an array of records to store the race information and print out the qualifiers.

. . . .

Another example could be if we wanted to store the characters in a kind of game. We would need to create a RECORD structure, then store the characters that used to be popular with "Dungeon and Dragon" enthusiasts as an ARRAY of RECORDS.

TYPE character IS RECORD{ string NAME, STRING weapon, INTEGER danger}

```
SET enemy[0]  TO {name = "troll", weapon = " axe", danger = 3}
SET enemy[1]  TO {name = "dwarf", weapon = " spell", danger = 3}
SET enemy[2]  TO {name = "wizard", weapon = " staff", danger = 9}
SET enemy[3]  TO {name = "ghost", weapon = " ectoplasm", danger = 2}
```

This loop will print out all the items in the array:

```
FOR counter FROM 0 TO 5 DO
   SEND "Name: ", enemy[counter].name, "Weapon: ",
enemy[counter].weapon, "Danger level: " &  enemy[counter].danger  TO DISPLAY

END FOR
```

Not all programming languages have a separate record structure, and use parallel arrays instead to store information which consists of a set of different data types.

Activity: Data types 1

5 min

Q7: Decide what data type you would use to store the following values. Choose from

- INTEGER
- REAL
- CHARACTER
- BOOLEAN
- STRING

No.	Value	Data type
1	678	
2	Open Sesame!	
3	0	
4	-5.7	
5	4000	
6	TD5 7EG	
7	joe@companymail.com	

. . . .

5 min

Activity: Data types 2

Q8: Decide what data type you would use to store the following values. Choose from

- INTEGER
- REAL
- CHARACTER
- BOOLEAN
- STRING

No.	**Value**	**Data type**
1	A UK telephone number	
2	The price of a pair of trainers	
3	Whether a character in a game has found a weapon or not	
4	The colour of a sprite	
5	The counter in a loop	
6	A URL	
7	A key-press	

. . . .

5 min

Activity: Structured data types

Q9: Decide what structured data type you would use to store the following. Choose from

- ARRAY of INTEGER
- ARRAY of REAL
- ARRAY of CHARACTER
- ARRAY of BOOLEAN
- ARRAY of STRING

No.	**Value**	**Data type**
1	A list of names	
2	A set of test scores out of 50	
3	The characters in a sentence	
4	The average temperatures during last month	
5	The last five Google searches you made	
6	Whether or not a class of pupils have passed an exam	

. . . .

Activity: Multiple data types

5 min

Q10: Decide what data types would be needed to create the following records, choose from:

- STRING, STRING, INTEGER
- STRING, INTEGER, BOOLEAN
- STRING, STRING, STRING,

No.	Records	Data types
1	Name, address and Scottish Candidate Number (SCN) for a list of pupils.	
2	Pupil ID, test score and pass/fail for a class	
3	Weapon name, ammunition type and damage value in a First Person Shooter game	

. . . .

. . . .

Quiz: Pseudocode

15 min

Use your chosen programming language to work out what the result would be from the following pseudocode examples.

Q11:
```
SET myNames  TO ["Fred","Jim","Betty", "Justin","Greg"]
SEND myNames[4] TO DISPLAY
```

a) Fred
b) Jim
c) Betty
d) Justin
e) Greg

. . . .

Q12:
```
SET myNames  TO ["Fred","Jim","Betty", "Justin","Greg"]
SEND myNames[1] TO DISPLAY
```

a) Fred
b) Jim
c) Betty
d) Justin
e) Greg

. . . .

Q13:
```
SET myVals TO [5, 12, 21, 35]
SEND myVals[1] + myVals[3] TO DISPLAY
```

a) 5
b) 12
c) 21
d) 26
e) 33
f) 35
g) 47
h) 56

. . . .

Q14:
```
SET mySentence TO "Hello World"
SEND mySentence[6] TO DISPLAY
```

a) H
b) e
c) l
d) W
e) o

. . . .

3.8 Handling records

Learning Objective

By the end of this section you will be able to:

- describe how your programming language handles records.

60 min

Practical task: Handling records

Find out the syntax your programming language uses to:

- Define a record structure
- Create an array containing 3 or more records
- Print out the contents of the array and the record.

. . . .

3.9 Identifying structured data types

Learning Objective

By the end of this section you will be able to:

- identify the different structured data types used by your chosen programming language.

Practical task: Structured data types

30 min

Look in the manuals, on the Internet or in the help documents for two of the programming languages in use in your school. List the structured data types available in your chosen languages:

....

Quiz: Identifying structured data types

5 min

Use your chosen programming language to work out what the result would be from the following pseudocode examples:

Q15:

```
TYPE monster IS RECORD {STRING name, BOOLEAN exists}
SET myMonster TO {name = "hydra", exists = false}
SEND myMonster.name to DISPLAY
```

a) hydra
b) exists
c) myMonster
d) false
e) monster

....

Q16:

```
TYPE monster IS RECORD {STRING name, BOOLEAN exists}
SET yourMonster TO {name = "crocodile", exists = true}
IF yourMonster.exists THEN send "Run!" TO DISPLAY
```

a) crocodile
b) Run!
c) yourMonster
d) monster
e) exists

....

3.10 Sequential files

Learning Objective

By the end of this section you will be able to:

- describe how your programming language handles sequential files.

Just as a computer program can receive data from an input device such as a keyboard, and send data to an output device such as the display, it can also receive data from and send data to a sequential file held on backing storage. A sequential file is identified by a path and filename which is treated as a STRING. For example **"n:\myfiles\testfile.txt"** would identify the file **testfile.txt** in the folder **n:\myfiles**.

The most common sequential file format is one which uses ASCII code. A sequential data file can be thought of as a 1-dimensional array with each array location storing a single ASCII code.

Sequential files can contain non printable ASCII codes such as an end of line character LF (ASCII code 10) or a carriage return CR (ASCII code 13).

All programming languages will have a syntax for the following file operations:

- creating and deleting
- opening and closing
- reading and writing

Sequential files are read from beginning to end and so a file cannot be read and written to simultaneously, although several different sequential files can be open for reading or writing at the same time.

When a file is opened, the operating system will lock it so that the program using it has exclusive access to it. When a file is closed, the file can then be accessed by other programs in the usual way. A file needs to be opened for the program to read from it or write to it.

Practical task: Programming language syntax

60 min

Find out the syntax your programming language uses to:

- open (or create if it does not exist) an existing file;
- write a character to the file;
- close the file;
- open an existing file;
- read a character from the file;
- close the file.

. . . .

3.11 Learning points

Summary

- All programming languages work with data, and that data can be held in a variety of ways depending on what type of data it is.
- Data types can be divided into two sorts: **simple** and **structured**.
- Simple data types are: INTEGER, REAL, CHARACTER and BOOLEAN.
- Simple data types correspond to the various ways which computers store information at machine code level: two's complement notation, floating point notation, ASCII code and as a single bit: 0 or 1.
- Structured data types are ARRAY and STRING (an ARRAY of CHARACTERS) and RECORD.
- Arrays, strings and records use an index to identify their contents. Indexes start at zero.
- Almost all programming languages are able to read and write to sequential files.
- The syntax for reading and writing sequential files is often similar to that to input and output for external devices such as keyboard and display.

3.12 End of topic test

End of topic test

Q17: From the data types listed above which would you use to store the following:

- INTEGER
- REAL
- CHARACTER
- BOOLEAN
- STRING
- ARRAY of INTEGER
- ARRAY of REAL
- ARRAY of CHARACTER
- ARRAY of BOOLEAN
- ARRAY of STRING
- RECORD

a) The average of 5 INTEGERs?
b) The visibility of a sprite in a game?
c) The room descriptions in an adventure game.
d) Time spent per day in seconds on the Internet over a month.
e) Stock levels of products in a supermarket.
f) The last 20 key-presses made while editing a document.
g) A list of Email addresses.
h) Whether a set of emails has been read or not.
i) A set of room descriptions and contents in an adventure game.
j) A set of pupil names and test scores.

. . . .

Topic 4

Development methodologies

Contents

Prerequisite knowledge

From your studies at National 5 you should already know:

- *software development is a complex process which needs careful planning and management;*
- *mistakes at the early stages of the planning process can mean problems and delays at later stages;*
- *using meaningful identifiers (variable and procedure names) and internal documentation makes readable source code.*

Learning Objectives

By the end of this topic you will be able to:

- *understand the iterative nature of the software development process;*
- *describe the seven stages in the traditional software development process, analysis, design, implementation, testing, documentation, evaluation, and maintenance;*

- *understand when Rapid Application Development would be an appropriate development methodology;*
- *describe the Top-Down Design / Stepwise Refinement process;*
- *understand when Agile Development is an appropriate development methodology.*

4.1 Revision

Revision

Q1: A programmer is storing 20 customer names in an array. Which one of these would be the best name for the data structure?

a) customerNames
b) names
c) mynames
d) strings

. . . .

Q2: Which one of these is in the correct order?

a) Code, design, test
b) Design, code, test
c) Test, code, design
d) Design, test, code

. . . .

Q3: Why is readability of source code important?

. . . .

Q4: When an application is built, who is responsible for reading and checking the source code?

a) The client
b) The user
c) The developer
d) The tester

. . . .

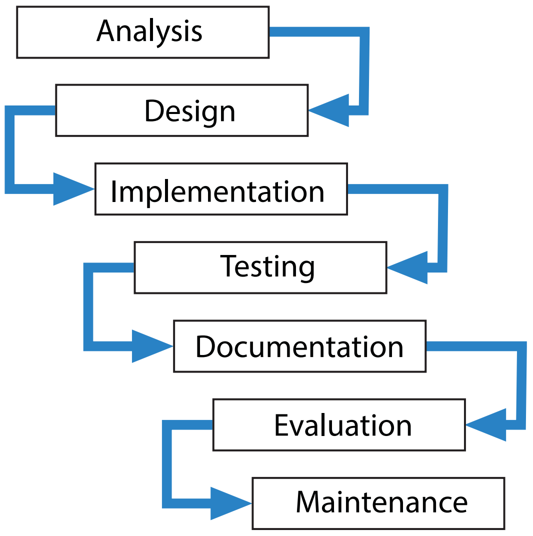
4.2 The traditional software development process

Learning Objective

By the end of this section you will be able to:

- understand the iterative nature of the software development process;
- describe the seven stages in the traditional software development process, analysis, design, implementation, testing, documentation, evaluation, and maintenance;
- describe the Top-down Design / stepwise refinement process.

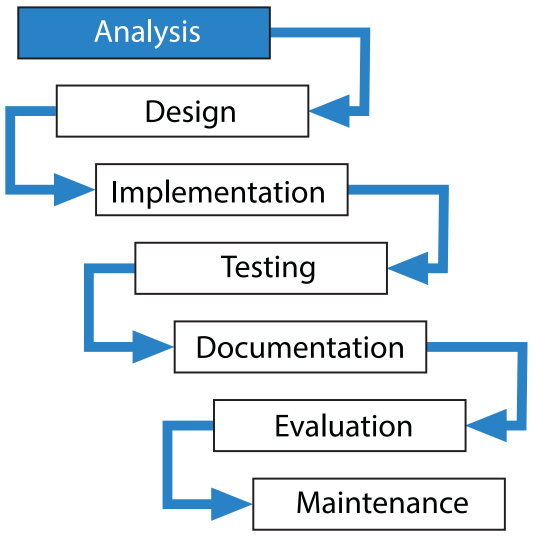
In this topic we are going to look at the seven stages in the software development process. These stages are:



The traditional software development process is often called the **waterfall model** because the development is seen as a series of developments flowing down from the analysis stage to the final evaluation stage.

This model has evolved as software developers have attempted to reduce the time and money spent creating and maintaining the increasingly complex applications they are being asked to create. Commercial software development is a process undertaken by a team of people who need to be able to work together in a structured and efficient way. The waterfall model involves constant revision and evaluation at every phase which makes it an **iterative** process. This ensures quality and efficiency in the final product. Large scale commercial software projects have traditionally followed this development process as far as possible in order to create successful product.

4.2.1 Analysis



The **analysis** stage of software development is the stage where an initial description of the problem to be solved is examined and turned into a precise description of exactly what the software will be able to do. Analysis of a problem is usually carried out by a Systems Analyst whose job is to take the description of the problem provided by the client and turn it into a **software specification** which can be used by the development team to create the completed application.

The **Systems Analyst** must be able to communicate effectively with the client in order to discover exactly what problem is that they want to solve, and also be able to communicate with the development team in order to accurately describe what needs to be produced. This is often more difficult than you might think for a number of reasons.

- The client may not be able to describe the problem they want solved accurately enough to convert directly into a clear description of a piece of software.
- The client may have unrealistic ideas of what is possible, or not be aware of what might be possible.
- In a large organisation there is often no one single person who knows exactly how every part of the system operates, or understands exactly what information needs to flow from one part of the organisation to another.

The Systems Analyst will collect as much information as they can about the organisation and the problem they want solved, because they need to know how the existing system works in order to design a solution which will work with the new one. As a result of their research they will create a **software specification** which accurately describes exactly what the software will be able to do, and will often also describe how long it will take to build and how much it will cost. This software specification is a **legally binding document** which can be used by either party to resolve possible disputes in the future. For this reason it is very important indeed that the document accurately describes what the client requires, and that they fully understand its contents.

Note that the software specification describes *what the software will be able to do*, not *how it does it*. That is the responsibility of the people who undertake the design stage.

Errors made or shortcuts taken at this stage in the software development process can have disastrous effects on subsequent stages.

Quiz: Analysis

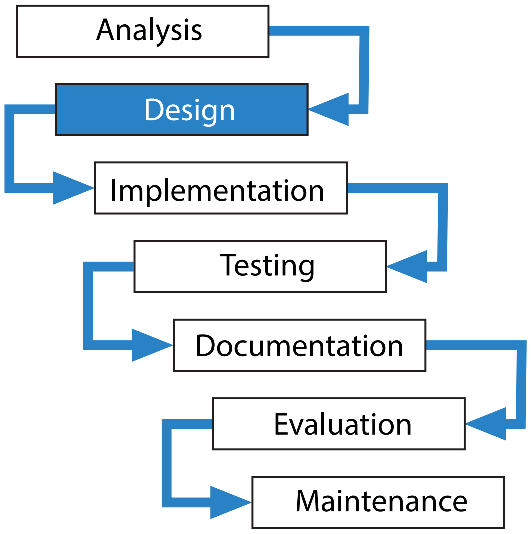
Q5: Why is the analysis stage of software development important?

. . . .

Q6: The Systems Analyst will create a software specification at the end of the Analysis stage of software development. What is a software specification?

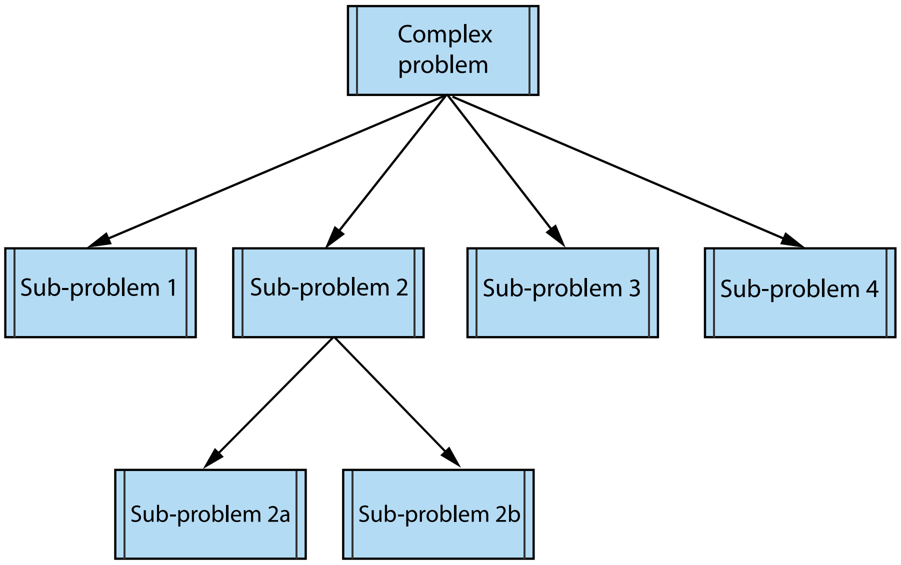
. . . .

4.2.2 Design



The **design** stage of software development is when the **software specification** created by the systems analyst is turned into a design which can be used by the team of programmers to write the code. The more time spent on this stage, the less time will be needed to be spent on the next one.

In theory any software problem can be broken down into smaller more easily managed problems. These problems can subsequently be further broken down into a set of simple steps. This process is often referred to as **Top Down Design and Stepwise Refinement**. Unfortunately things are not always as simple as this; as knowing how to break a problem down into smaller sub-problems takes practice. If it can be done successfully, a **structure diagram** will usually be created showing how the different sub-problems relate to each other.



When a problem is broken down into smaller sub-problems, the task becomes more manageable because each part can be worked on separately. This is called **modular design**. There are several advantages to this system:

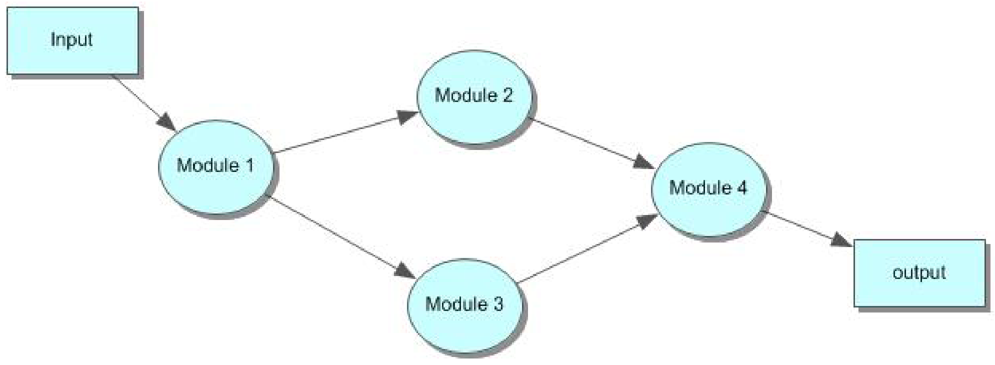
- Different modules can be worked on simultaneously by separate programming teams.
- Modules can be tested independently.
- A well written module may be able to be re-used in another application.
- Modules can mirror the structure of the data being processed by the application. For instance a school management system may have a timetable module, a registration module, and an assessment module.

Eventually all the modules need to be able to communicate with each other. This means that the **flow of data** around the application needs to have been made clear when the initial structure diagram was created.

The programming language to be used will have a set of simple and structured data types which are used to define variables.

The design of the data structures the software will use is very important indeed and the choice of data structure will affect the entire project. As well as a **structure diagram**, the design stage will also result in a **data dictionary** which details all the data structures which the software will use and how they are related. Even small-scale programming problems benefit from an initial consideration of what arrays are going to be needed, what size they should be and what data types they will use. In larger projects this is an essential part of the design process.

Once the data structures have been decided upon the flow of data around a system may be represented in a **data flow diagram**.



Once the structure of the program and its sub modules has been determined, the detailed logic of each component will be designed, using **pseudocode**.

If the pseudocode is written clearly and is thoroughly tested by working through the logic manually, then creating source code from it should be a relatively simple process.

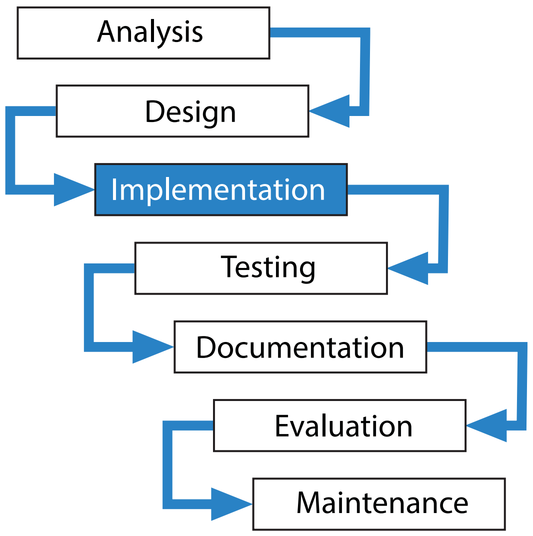
Stepwise Refinement

The process of designing the logic of each module is known as **stepwise refinement**. This is a process of breaking the module down into successively smaller steps, eventually resulting in a set of pseudocode instructions which can be converted into the chosen programming language.

```
PROCEDURE Module1()
  <menu>
  <step_2>
  <step_3>
  <step_4>
END PROCEDURE

PROCEDURE menu()

SEND "Menu: press H for help or C to continue"
RECEIVE{userInput FROM  (STRING) KEYBOARD)
WHILE userInput ≠ ["H"] AND userInput ≠ ["C"] DO
SEND "Please press H or C " TO DISPLAY
RECEIVE userInput FROM (STRING) KEYBOARD
END WHILE

END PROCEDURE
```

Each step may be further broken down into sub steps until the logic of the program is complete.

4.2.3 Implementation



Just as the design stage depends upon how well the analysis stage has been done, the **implementation** stage depends very much upon how clear the design is. If problems are spotted at the implementation stage, then this may well mean that the original design needs to be re-examined. This process of looking back to a previous stage to solve problems encountered in a subsequent one is what makes the software development process an **iterative** one.

If the design has been written in enough detail, the implementation stage should just be a matter of using the design to create the source code for the application.

The choice of high level programming language will depend on a number of factors:

- The type of problem the software is being created to solve, as many programming languages are domain specific.
- The language and the programming environment the development team are familiar with.
- The operating systems the software will be running on, since some programming languages are more easily adapted to run on different operating systems than others.

The design stage will have identified the sub-problems which the programming team need to complete, and because there are several people working on the same project, communication between them is very important. This means that the code they produce must be as **readable** as possible. Readable code means that if staff turnover is high, it will be easier for a new programmer to take over work from someone who has moved jobs or from a colleague who has to take time off due to illness.

Readability of code is achieved by:

- using meaningful identifiers (variable and procedure names),
- inserting internal documentation,
- white space and indentation to make the logic of the code clear,

- using local variables and creating modular code - this makes the modules re-usable in other parts of the program and on subsequent projects.

Code readability is so important that a software development company will often impose a **house style** on their employees. A house style will stipulate how internal documentation should be written, conventions for variable names, where and when variables should be declared, use of indentation and white space etc. This means that code written by one programmer should be easily read and understood by others in the team, and of course if someone is ill or leaves the company, their code can be taken over by another member of the team without a problem.

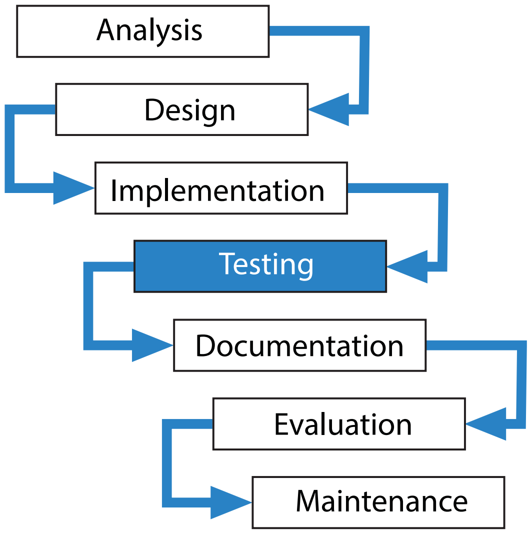
Debugging

Debugging is the process of finding and correcting errors in code. For obvious reasons, readable code makes debugging easier.

Some errors in code will be discovered during the implementation stage but some will only be identified at the testing stage which means that the implementation stage needs to be re-visited to correct them. For example, it might turn out that an implemented algorithm runs too slowly to be useful and that the designer will have to develop a faster algorithm. Or it might be that the slowness of operation is due not to a poor algorithm but to slow hardware. In such a case, the development team might have to return to the analysis stage and reconsider the hardware specification.

At the end of the implementation stage a **structured listing** is produced, complete with internal documentation. This will be checked against the design and against the original specification, to ensure that the project remains on target.

4.2.4 Testing



Testing a piece of software has several purposes. It should check that:

- the software meets the specification;
- it is **robust**;
- it is **reliable**.

At the design stage, a set of test data will have been created which can be used to determine whether the software meets the specification or not, so that it is possible to see if the program does what it is supposed to do.

Modern programs are so complex that testing can never show that a program is correct in all circumstances. Even with extensive testing, it is almost certain that undetected errors exist.

Testing can only demonstrate the presence of errors, it cannot demonstrate their absence.

As far as possible, testing should be both systematic, which means testing in a logical order, and comprehensive which means testing every possible scenario.

Test data :

When you are testing software, it should be tested with three types of data.

- **Normal data:** data that the program is expected to deal with.
- **Extreme data:** data that represents the values at the boundaries of the range of data it should accept. For instance if a program should only accept numbers between 1 and 100 then it should be tested with 1 and 100.
- **Exceptional data:** data that lie beyond the extremes of the program's normal operation; these should include a selection of what might be entered by accident or misunderstanding, so if a program should only accept whole numbers it should be tested with text input and decimal values. Exceptional data should also include data which is just **outside** the boundaries of the range of data it should accept.

Activity: Testing

Q7:

Match **three** of the following phrases to their correct descriptions:

1. data that is invalid;
2. data to test the extremes of a program's operations;
3. Abnormal data;
4. data that the program has been built to process;
5. data which lies beyond the extremes of normal operations.

Descriptions:

- Normal
- Extreme
- Exceptional

. . . .

Q8: Give examples of normal, extreme and exceptional test data for a program that should accept a numerical value for the months of the year.

. . . .

Testing should be **systematic** and the results recorded so that time is not wasted repeating work done already, and the developers have a clear list of what has and what hasn't been tested. Test results should be documented in a clear list matching test data with results.

This kind of testing of a program within the organisation is called **alpha testing**.

Alpha testing will hopefully detect any logical errors in the code and if there are any discovered this will result in that part of the program being looked at again by the programming team and corrected. It will then have to be re-tested.

Alpha testing does not have to wait until an application is complete. It may be done on modules or on parts of an application while other parts are still being developed.

Once the software has been fully tested and corrected by the software developers, the next stage is to test it in the environment which it has been designed for. This is called **beta testing**.

Beta testing is important for a number of reasons:

- it is essential that the client is able to test the software and make sure that it meets the specification they agreed to at the analysis stage;
- although the programming team will have tested the software with appropriate test data, it is can be difficult for the programming team to test their work as a user who might make unpredictable mistakes rather than as a developer who is very familiar with the application they have been building;
- it is important that the people who are actually going to use the software are able to test it.

If a program has been developed for use by particular clients, it is installed on their site. The clients use the program for a given period and then report back to the development team. The process might be **iterative**, with the development team making adjustments to the software. When the clients regard the program's operation as acceptable, the testing stage is complete.

If a program is being developed by a software house for sale to a market rather than an individual client, the developers will provide an alpha-tested version to a group of expert users such as computer journalists and authors. This is of benefit to both parties; the software house gets its product tested by people who are good at noticing faults, and the writers get to know about products in advance.

People involved in beta testing will send back error reports to the development team. An error report is about a malfunction of the program and should contain details of the circumstances which lead to the malfunction. These error reports are used by the development team to find and correct the problem.

Quiz: Testing

5 min

Q9: Which of the following statements are true of **alpha** testing of an application?

a) Testing is done by the users
b) Testing is done by the programmers responsible for the application
c) Testing is done by specialist companies
d) Testing is done by the client
e) Testing may be done on parts of the application

. . . .

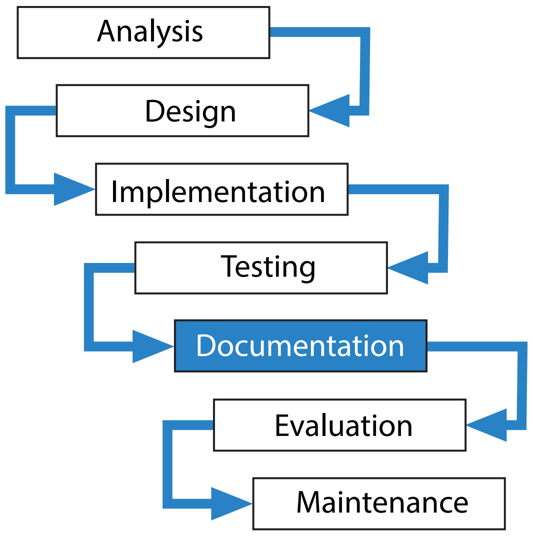
Q10: Which one of the following statements describe **beta** testing?

a) The testing is performed by the clients
b) The testing is more rigorous than alpha
c) The testing is for market research
d) The testing is performed by specialist companies

. . . .

. . . .

4.2.5 Documentation



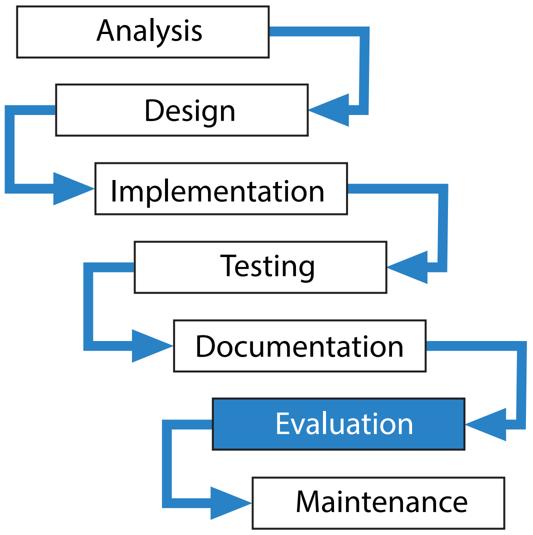
The **documentation** stage is when the guides to using and installing the software are produced. The documents created during the previous stages such as the software specification and test history are also collected together so that they can be referred to in case of changes or problems discovered at a later date.

The **user guide** for the software should include a comprehensive guide to the menu options, and how each one functions. On-line user guides are convenient for both the user and the developer. The user can make use of a search facility and can access the guide whenever they are using the software. The developer saves on distribution costs and can update it whenever changes are made to the software.

The program should include a help facility. It is common for on-line help to be presented in three tagged pages: Contents, Index, and Search. The contents present the help chapter by chapter; the index refers to certain key words in the chapters; and search offers the facility to locate key terms within the guide.

The **technical guide** for the software should include details of the minimum specification of hardware required such as available RAM, processor clock speed, hard disk capacity and graphics card specification. It will also specify the platform and operating system versions which it is compatible with and any other software requirements or incompatibilities. The technical guide will give details of how to install the program, and if it is to function on a network, where to install it and how to licence it.

4.2.6 Evaluation



The evaluation stage is where the software is critically examined by both the client and the developer. The single most important criterion for evaluating software is whether it is fit for purpose i.e. does it match the software specification written at the analysis stage. It will also be evaluated against the following criteria:

- Robustness
- Reliability
- Maintainability
- Efficiency
- User interface

The evaluation is useful to the client because once it is complete, they can be sure that they have the software they need, and useful to the developer because if any problems are found at this stage it can save work later on. An evaluation can also help the developer improve their performance for future projects.

Robustness

Software is robust if it is able to cope with mistakes that users might make or unexpected conditions that might occur. These should not lead to wrong results or cause the program to hang. For examples an unexpected condition, could be something going wrong with a printer, (it jams, or it runs out of paper) a disc drive not being available for writing, because it simply isn't there, or the user entering a number when asked for a letter. Put simply, a robust program is one which should never crash.

Reliability

Software is reliable if it always produces the expected result when given the expected input. It should not stop due to design faults.

Maintainability

Software is **maintainable** if it can be easily changed and adapted. This is why **readability** and **modularity** are so important in software design. The person maintaining it may not be the same person as the one who wrote it. Even the original author may find their code difficult to understand at a later date if it has not been written clearly. Modularity makes a program easier to maintain because the separate functions can be tested and changed without causing unexpected consequences with other parts of the program. It is also easier to locate (and therefore correct) errors in a modular program.

Efficiency

Software is considered to be efficient if it avoids using resources unnecessarily. Resources may be processor time, RAM, hard disk space, or Internet bandwidth. There may be a trade-off between programmer time and efficiency. The increased processor speed and memory capacity of modern machines can encourage saving valuable programmer time but at the expense of creating less efficient software. This can be seen with newer versions of operating systems, which often perform more slowly than their predecessor on the same hardware.

User Interface

The user interface of a software product is the part which has most influence on the reaction of its users. A user interface should be:

- Customisable
- Appropriate
- Consistent
- Provide protection from error
- Accessible

A user interface should be **customisable** so that users can alter the way they use the software to their own preferences.

A user interface must be **appropriate** to the expertise of the user expected to be using the software. Ideally it will provide a number of different levels of interface depending on the expertise of the user. A word processor for instance will provide a number of different ways of performing the same function, menu options, shortcut keys, and toolbar icons.

A user interface should be **consistent** so that users find similar functions grouped together under menus, dialog boxes with commonly used options like OK and Cancel in the same place.

A user interface should **provide protection from error** so that critical events such as deleting data give sufficient warning to the user before they are completed.

A user interface should be **accessible** so that its design does not impede those users with disabilities. This might mean making the interface compatible with a text reader, providing customisable colour settings for a high contrast display or the provision of optional large size icons or toolbars.

Activity: Evaluation terminology

Q11: Match each term to the correct description:

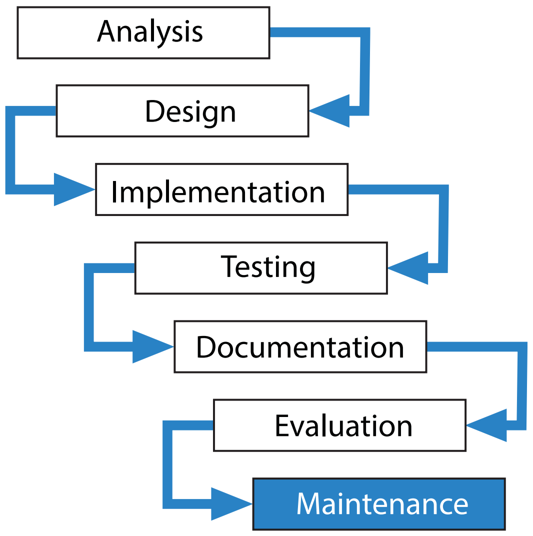
- Reliable
- Maintainable
- Readable
- Portable
- Efficient
- Fit for purpose
- Robust

Term	Description
	Ability of a program to keep running even when external errors occur.
	The program always produces the expected result when given the expected input.
	Whether or not the program can easily be used on a variety of hardware and/or operating systems.
	Whether the program wastes memory or processor time.
	Has the program been designed to easily altered by another programmer.
	Is the coding easy to understand, because it uses meaningful variable names and is well-structured.
	Does the program fulfil all the requirements of the specification.

. . . .

. . . .

4.2.7 Maintenance



There are **three** types of software **maintenance**:

- **Corrective**
- **Adaptive**
- **Perfective**

Corrective maintenance is concerned with errors that were not detected during testing but which occur during actual use of the program. Users are encouraged to complete an error report, stating the inputs that seemed to cause the problem and any messages that the program might have displayed. These error reports are invaluable to the development team, who will attempt to replicate the errors and provide a solution. The developer will be responsible for any costs incurred by this type of maintenance.

Adaptive maintenance is necessary when the program's environment changes. For example, a change of operating system could require changes in the program, or a new printer might call for a new printer driver to be added. A change of computer system will require the program to have its code adapted to run on to the new system. The cost of adaptive maintenance is usually borne by the client but there may be negotiation depending upon how predictable the change in circumstances was.

Perfective maintenance occurs when the client requests changes or improvements to the program which were not in the original software specification. This may be due to changes in the requirements or new legislation. Such maintenance can involve revision of the entire system and can be expensive. The cost of perfective maintenance is likely to be borne by the client.

Activity: Maintenance

Q12:

Match **three** of the following phrases to their correct descriptions:

1. only done under extreme circumstances;
2. errors removed that were initially undetected;
3. requirements incorrect;
4. occurs in response to requests to add new features;
5. needed when environment changes.

Descriptions:

- Corrective
- Adaptive
- Perfective

....

Activity: Waterfall model

Q13:

Place these terms in the correct order and match them to their correct definition:

1. Checking to see how well the software meets its specification
2. Writing the source code
3. Looking at the problem and collecting information
4. Fixing problems and adapting the software to new circumstances
5. Creating a structure diagram and pseudocode
6. Trying to find ways in which the program will fail
7. Creating a user guide and technical guide

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

....

4.3 Rapid Application Development (RAD)

Learning Objective

By the end of this section you will be able to:

- understand when Rapid Application Development would be an appropriate development methodology.

Although the waterfall model has traditionally been the one used for large scale software development projects, it has often been criticised as being too rigid and too slow a process, resulting in projects where the software specification had to be changed substantially during the lifetime of the project, or software became out of date before it was even complete. In theory the analysis stage should result in a software specification which can then be used throughout the rest of the project, but in practice this is often unrealistic.

Rapid Application Development means that the users should be involved at all stages of the development process and that changes to the design can be made at any time throughout the life of the project. It also means that the development process is faster and more flexible.

There are **four** stages to the Rapid Application Development model:

1. **Requirements Planning phase**: This is similar to the analysis stage but where users, managers and IT staff discuss and agree on what they need, the project scope, its constraints and the system requirements. They do not necessarily create a legally binding software specification.
2. **User Design phase**: During this phase the developers work with users to translate their requirements into prototypes of the different systems required. This is a continuous interactive process that allows users to understand, modify, and eventually approve a working model of the system that meets their needs.
3. **Construction phase**: This phase is similar to the implementation and testing stages of the waterfall model, but users continue to be involved and can still suggest changes or improvements as the software is built.
4. **Cutover phase**: In this phase, the software is installed and tested on the new system and user training and maintenance starts.

As you might guess from the name, the emphasis of Rapid Application Development is on creating software quickly and efficiently. It is however, still considered to be more appropriate to smaller software projects.

4.4 Agile software development

Learning Objective

By the end of this section you will be able to:

- understand when Agile Development is an appropriate Development Methodology.

Agile software development is a product of the Rapid Application Development idea, with an emphasis on small scale developments, and with teams of people who have a flexible approach to change in requirements. Agile software development is seen as a more adaptable and responsive process compared to the rigid waterfall model. Agile development has been widely seen as being more suitable for certain types of environment using small teams of experts such as web development.

The benefits of the Agile method are:

- reduced development time;
- increased responsiveness to changing circumstances;
- more reduced costs due to the efficiency of using small groups of developers.

Agile development will make use of **prototyping** where working models of the proposed system are tried out and tested throughout the development process so that client feedback can be taken into account as early as possible. Developers and clients will use tools such as **version management software** and online ticket systems to keep track of issues and bugs and give feedback on progress.

A common feature of agile development is the frequent appearance of updates to the software, often given sequential version numbers.

There has been some criticism of the agile software development process as being too extreme a contrast to the tried and trusted **waterfall model**, or as just a management fad that simply describes existing good practices under new jargon, and wrongly emphasizes method over results. Agile development can also mean that it encourages the client to make changes to the specification throughout the development process rather than thinking clearly about what they require at the beginning.

4.5 Learning points

Summary

You should now know:

- The traditional waterfall model of software development consists of seven stages: analysis, design, implementation, testing, documentation, evaluation, and maintenance.
- Software development is an iterative process.
- In the analysis stage the client's project description is carefully examined and after discussion, a legally binding software specification is written.
- In the design stage, top down design and stepwise refinement is used to turn the software specification into structure diagrams, pseudocode and a data dictionary which the programming team can use.
- In the implementation stage the programming team use the design to create and debug the program code.
- In the testing stage the code is alpha tested using normal, extreme and exceptional test data, and then beta tested using clients or individuals outside the development team organization.
- In the documentation stage the user guide and technical guide are produced.
- In the evaluation stage the software is examined to see if it is reliable, robust, maintainable, efficient and user friendly.
- The maintenance stage is where problems are fixed, the software may be adapted to new circumstances and additional features may be added.
- Rapid Application Development (RAD) is an attempt to streamline the waterfall model by using prototyping and involving the client at more stages in the development process.
- Agile programming is a type of RAD suited to smaller projects. It is designed to be as flexible as possible where the specification may change throughout the development process resulting in reduced development time and costs.

4.6 End of topic test

End of topic test

10 min

Q14: A company wishes to add a network capability to their recently acquired computer program. In maintenance terms this would be an example of:

a) Perfective maintenance
b) Routine maintenance
c) Corrective maintenance

. . . .

Q15: Which one of these would not be found in the technical guide?

a) Operating system required
b) Hardware requirements
c) Memory requirements
d) Tutorial

. . . .

Q16: During the software development process, which one of the following is responsible for converting the design into actual program code?

a) Programmers
b) System Analyst
c) Independent test group
d) Client

. . . .

Q17: A computer program is designed to accept whole numbers between 0 and 99 as input. If the value 56.8 was entered this would be an example of:

a) Boundary data
b) Exceptional data
c) Extreme data
d) Normal data

. . . .

Q18: Which one of the following terms is **best** described by the phrase below?

"how well a program operates without stopping due to design faults"

a) Robustness
b) User friendly
c) Reliability
d) Efficiency

. . . .

Q19: Software is evaluated according to a number of different criteria. Which one of the following would **not** be included in an evaluation report?

a) Portability
b) Efficiency
c) Editability
d) Maintainability

. . . .

Q20: Which one of these is NOT an advantage of agile software development

a) Reduced development time
b) Responsiveness to changed circumstances
c) Reduced costs
d) Reduced time spent on analysis

. . . .

Q21: Which statement is the best description of pseudocode?

a) A list of pre-written subroutines in the program.
b) An informal list of testing instructions within the code.
c) A list of statements in a high level language.
d) A high-level description of how a computer program functions.

. . . .

Q22: Which of these has the waterfall model stages in the right order?

a) Analysis, Design, Implementation, Documentation, Testing, Evaluation, Maintenance
b) Analysis, Design, Implementation, Testing, Documentation, Evaluation, Maintenance
c) Analysis, Design, Evaluation, Implementation, Testing, Documentation, Maintenance
d) Analysis, Design, Implementation, Testing, Documentation, Maintenance, Evaluation

. . . .

Q23: Who is responsible for writing the software specification during the software development process?

a) The systems analyst
b) The programming team
c) The client
d) The beta testers

. . . .

Topic 5

Software design notations

Contents

Prerequisite knowledge

From your studies at National 5 you should already know:

- *that* ***top down design*** *and* ***stepwise refinement*** *are part of the design phase in software development process;*
- *that a* ***structure diagram*** *is a graphical representation of the logic of a program;*
- *that* ***pseudocode*** *is an informal high-level description of how a computer program functions.*

Learning Objectives

By the end of this topic you will be able to:

- *explain how design notations can help the software development process;*
- *describe a variety of graphical program design notations including structure diagrams, data flow diagrams and wireframes;*
- *describe the relationship between pseudocode and source code; interpret examples of pseudocode.*

5.1 Revision

Revision

All 3 questions refer to the following:

```
SET total TO 0
SET count TO 0
WHILE count < 10 DO
      RECEIVE nextInput FROM (INTEGER) KEYBOARD
      SET total TO total + nextInput
       SET count TO count + 1
END WHILE
SEND total / 10 TO DISPLAY
```

Q1: The above is an example of?

a) Pseudocode
b) Source code
c) Machine code
d) High level language code

. . . .

Q2: How many numbers will be input into this program?

a) 1
b) 9
c) 10
d) 11

. . . .

Q3: What is the value of the count variable when the WHILE loop ends?

a) 1
b) 9
c) 10
d) 11

. . . .

5.2 Introduction

Learning Objective

By the end of this section you will be able to:

- explain how design notations can help the software development process;
- describe a variety of graphical program design notations including **structure diagrams, data flow diagrams** and **wireframes**.

A software specification describes what a program must do. The design stage of the software development process is where a set of documents is created that describe how the program will do it. These documents might describe the **structure of the program** in terms of different modules, the **flow of data** between these modules and the **detailed logic** of the modules themselves. It makes sense to discuss the **user interface** at an early stage in the design process as well.

The more accurately these documents reflect the program specification, the easier it will be for the programmers to create the program **source code** from them.

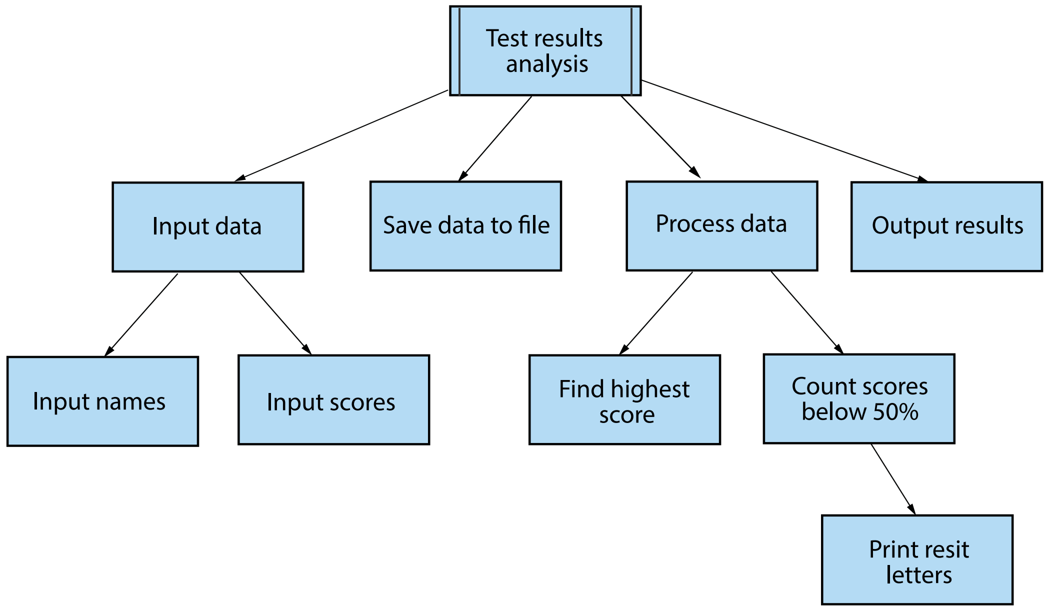
5.3 Structure diagrams

A **structure diagram** will be created as part of the top down analysis of the software specification. This allows the developers to break this complex problem description into a series of smaller sub-problem descriptions. These sub problems can be regarded as modules within the system and they themselves may be further divided into smaller (and hopefully simpler) sub problems.

Example - Structure diagram

A program is required to take in a set of test results, save the data to file, calculate the highest score and how many failures there were. It should also print re-sit letters for all those candidates who failed.

This problem can be broken down into four main modules, two of which can be further broken down into sub tasks.



A structure diagram is organised to show the level or hierarchy of each sub task. The sequence of operations in the program is read from top to bottom, going left to right.

Once the structure of the program has been decided, the next step is to work out what data each module will need and what data it will pass on to the next module. This can be shown either by either annotating the structure *diagram* or by creating a **data flow diagram**.

Quiz: Structure diagram

Q4: In the above example, what data structures would be used to store candidate names?

. . . .

Q5: In the above example, what data structures would be used to store test scores?

. . . .

Q6: What is the 7th process "visited" in this structure diagram?

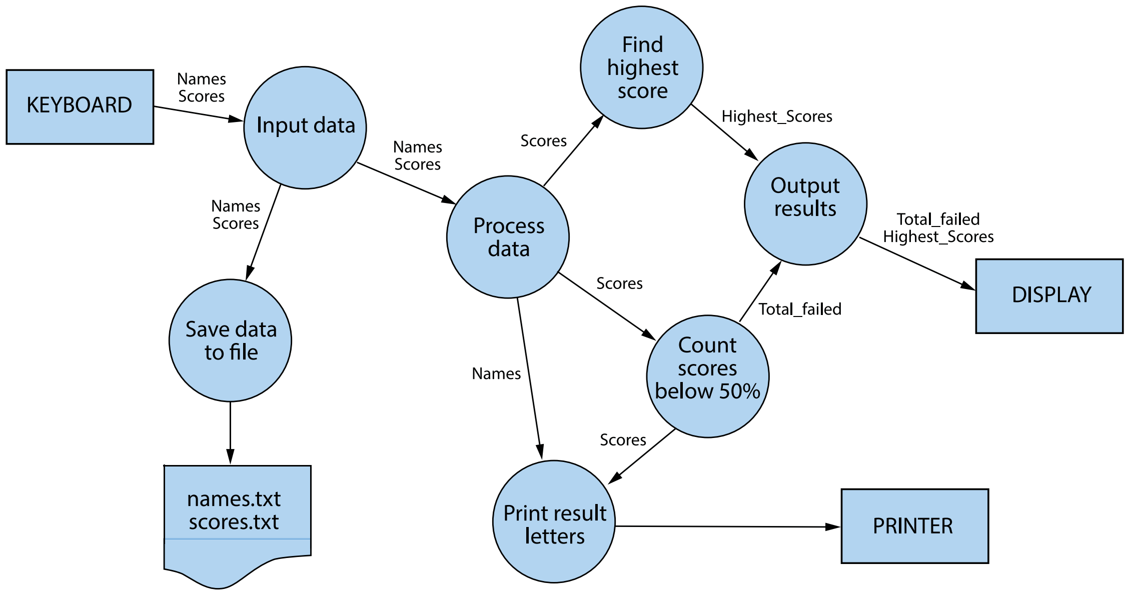
. . . .

5.4 Data flow diagrams

The data used by the different modules can be represented by a data flow diagram. Here the modules are shown as circles and the physical devices are shown as rectangles. Note that not all data is needed by every module.

The data types and structures storing the information used would be:

```
names ARRAY[20] of STRING
scores ARRAY[20] of INTEGER
highest_score INTEGER
total_failed INTEGER
```



Like a structure diagram, a data flow diagram reads from left to right. A data flow diagram can be used to decide on what data needs to be passed as parameters to each module.

Quiz: Data flow diagrams

Q7: What inputs are there to the output results module?

. . . .

Q8: Which variables have a different value at the end of the process than they did at the beginning?

. . . .

There are many diagram creation tools available. Gliffy is a free tool which lets you create structure diagrams and data flow diagrams online.

5.5 Pseudocode

Learning Objective

By the end of this section you will be able to:

- describe the relationship between pseudocode and source code;
- interpret examples of pseudocode.

The detailed logic of each module can now be written as pseudocode. Pseudocode is often referred to as a "halfway house" between an algorithm (written in English) and final code.

```
# inputdata

SET counter TO 0
SET input_ok TO true

WHILE input_ok = true DO
     RECEIVE names[counter] FROM (STRING) KEYBOARD
     RECEIVE scores[counter] FROM (STRING) KEYBOARD
     SEND  "Another result? Yes/No" TO DISPLAY
     RECEIVE response FROM (STRING) KEYBOARD
     IF response = "No" THEN
       SET input_ok TO false
     END IF
     SET counter TO  counter + 1
END WHILE



# save data to file

CREATE "names.txt"
FOREACH name FROM names DO
    SEND name TO "names.txt"
END FOREACH

CREATE "scores.txt"
FOREACH score FROM scores DO
    SEND score TO "scores.txt"
END FOREACH


# process data
    <find highest score>
    <count scores below 50%>
    <print resit letters>
```

```
# find highest score

SET highest_score TO scores[0]

FOREACH score FROM scores DO
    IF highest_score < score THEN
        SET highest_score TO score
    END IF
END FOREACH

SEND "The highest score was:" & highest_score TO DISPLAY


# count scores below 50%

SET total_failed TO  0
FOREACH score FROM scores DO
   IF score < 50 THEN
      SET total_failed TO total_failed + 1
   END IF
END FOREACH
SEND "There were " & total_failed & " failed students" TO DISPLAY


# print re-sit letters

SET counter TO 0
REPEAT
    IF (score[counter] < 50) THEN
    SEND "Dear " & names[counter] &
" Your resit is on " & now() + 14  TO <printer>
    END IF
UNTIL names[counter] = ""
END REPEAT
```

The more clearly the pseudocode can be written, the easier it will be to write the final solution in a high level language.

Practical: Creating a structure diagram and Data flow diagram

30 min

Create a structure diagram and data flow diagram for the following problem:

A custom computer company wants a piece of software which rates processors using a points system according to their clock speed, data bus width and cache size.

The program should input the data and calculate the points awarded then allocate a number of stars depending on the range of points allocated.

The program should calculate and display the number of processors at each star rating then find and display the processor with the highest number of points.

Data structures used could be:

```
processors[10]  of STRING
clockSpeed[10] of REAL
dataBusWidth[10] of INTEGER
cacheSize[10] of INTEGER
points[10] of INTEGER
starRating[10] of INTEGER
```

....

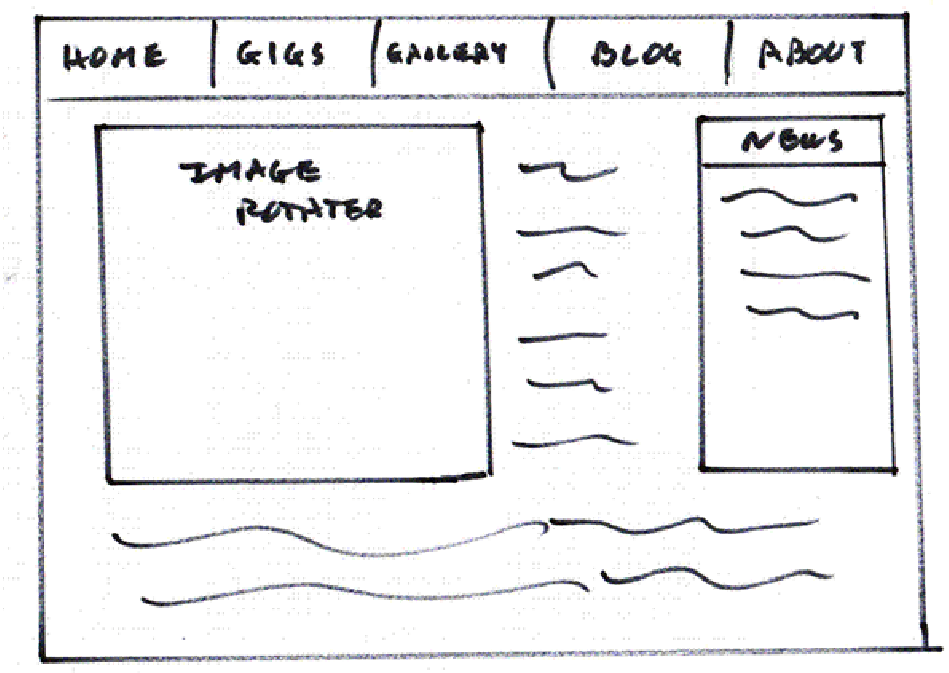
5.6 Wireframes

Wireframes are one of the techniques used for user interface design. The user interface of any software is the part which users experience and is therefore a crucial part of the design process. A **wireframe** is a visual guide that represents a website or program interface, and is normally created at an early stage in the development of an application to give the client and developers a clear idea of how the finished product will function and how users will interact with it.

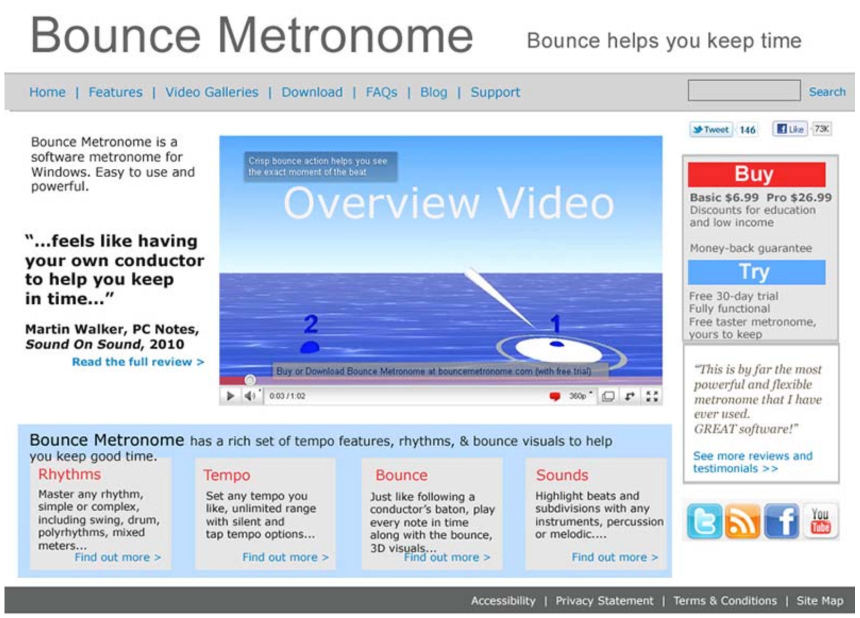
Web pages

A wireframe can show how a website will look and how its navigations structure works. In the case of a software application the wireframe will show how the menu and sub-menu options will appear and how they interact with any dialog boxes used.

A wireframe for a website can be as simple as a rough sketch or can be a detailed design showing colour combinations and images.



A wireframe can also be a detailed design that could be used to give the client an idea of how the final product will look:



Bounce Metronome Bounce helps you keep time

Home | Features | Video Galleries | Download | FAQs | Blog | Support Search

Tweet 146 Like 73K

Bounce Metronome is a software metronome for Windows. Easy to use and powerful.

"...feels like having your own conductor to help you keep in time..."

Martin Walker, PC Notes, *Sound On Sound*, 2010

Read the full review >

Crisp bounce action helps you see the exact moment of the beat

Overview Video

Buy or Download Bounce Metronome at bouncemetronome.com (with free trial)

0:03 / 1:02 360p

Buy

Basic $6.99 Pro $26.99

Discounts for education and low income

Money-back guarantee

Try

Free 30-day trial

Fully functional

Free taster metronome, yours to keep

"This is by far the most powerful and flexible metronome that I have ever used. GREAT software!"

See more reviews and testimonials >>

Bounce Metronome has a rich set of tempo features, rhythms, & bounce visuals to help you keep good time.

Rhythms

Master any rhythm, simple or complex, including swing, drum, polyrhythms, mixed meters...

Find out more >

Tempo

Set any tempo you like, unlimited range with silent and tap tempo options...

Find out more >

Bounce

Just like following a conductor's baton, play every note in time along with the bounce, 3D visuals...

Find out more >

Sounds

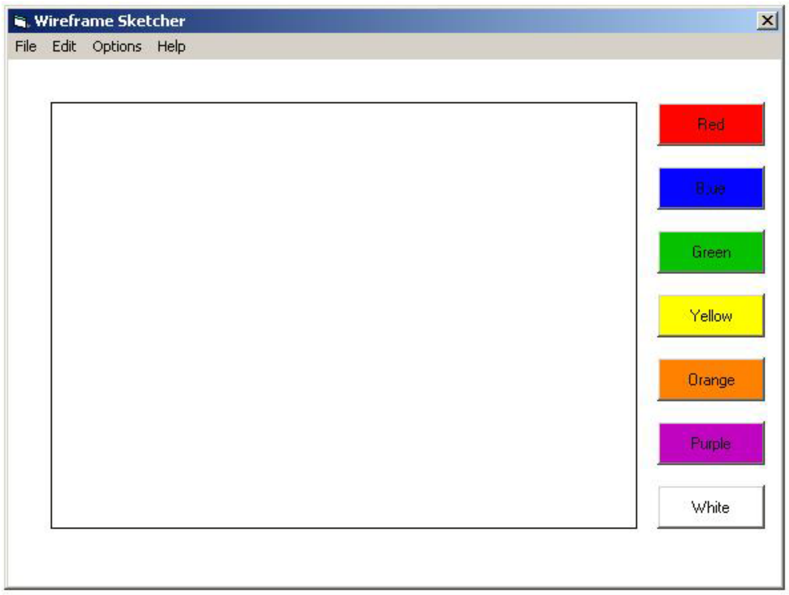
Highlight beats and subdivisions with any instruments, percussion or melodic....

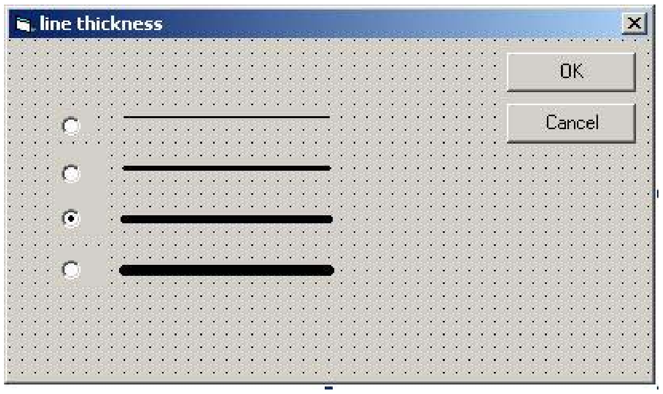
Find out more >

Accessibility | Privacy Statement | Terms & Conditions | Site Map

Applications

In many modern software development environments such as Visual Basic or LiveCode, it is possible to create the interface for an application without including any functionality other than the menu drop down or dialog box selection. This allows the client to see the proposed interface and request changes at an early stage in the development.





This means that the client will have a clear idea of what the finished application will look and feel like. The advantage for the developer is that they can be reassured that what they are going to create is what the client actually wants.

Activity: Wireframes

20 min

A client wants a game which allows them to move around a 2D maze which is stored in memory but not shown on screen. Each room in the maze will have a text description which is displayed when the player enters. The program will have a library of levels which can be loaded from file. Design a user interface and make sketches of any menu options or dialog boxes you think should be used.

. . . .

5.7 Learning points

Summary

You should now know:

- Graphical design notations are used in the design stage of the software development process to help the developers break the problem down into manageable sub problems.
- Structure diagrams are a graphical representation of the top down design process where blocks in the diagram correspond to modules in the program.
- Data flow diagrams are used to represent how data is passed between modules.
- Pseudocode is an informal description of the logic of each module which can then be used to write source code.
- Wireframes are a graphical notation which not only helps the developer plan the application but also allows the client to influence the design and navigation at an early stage.
- Wireframes are use in web development as well as program design.

5.8 End of topic test

End of topic test

10 min

Q9: The design approach of breaking a large and complex problem into smaller, more manageable sub-problems is known as:

a) Process refinement
b) Top-down refinement
c) Bottom-up design
d) Top-down design

. . . .

Q10: Which one of these statements is true?

a) It is not necessary to bother about the module names as these will change in the code.
b) The modules in a structure chart will become modules in the finished program.
c) Structure charts are not hierarchical.
d) A structure chart cannot show the data flow between modules.

. . . .

Q11: Which one of these is **not** a graphical design notation?

a) Wireframe
b) Structure diagram
c) Pseudocode
d) Data flow diagram

. . . .

Q12: Which design notation needs the data structures to be decided on before it can be created?

a) Wireframe
b) Structure diagram
c) Pseudocode
d) Data flow diagram

. . . .

Q13: Which design notation would you use to design a user interface?

a) Wireframe
b) Structure diagram
c) Pseudocode
d) Data flow diagram

. . . .

Q14: Which design notation is the easiest to create source code from?

a) Wireframe
b) Structure diagram
c) Pseudocode
d) Data flow diagram

. . . .

Q15: Stepwise refinement is:

a) Creating pseudocode from the structure diagram and data flow diagram.
b) Breaking a large and complex problem into smaller, more manageable sub-problems.
c) Writing source code.
d) Creating a wireframe and interface design.

. . . .

Topic 6

Algorithm specification

Contents

Prerequisite knowledge

From your studies at National 5 you should already know:

- *an algorithm is a detailed sequence of steps which, when followed, will accomplish a task;*
- *an array is a data structure that stores a range of values of the same type in a single indexed structure;*
- *pseudocode is a method of describing a computer program in an informal English-like language;*
- *a fixed loop repeats program code a set number of times and a conditional loop repeats program code until a condition is met.*

Learning Objectives

By the end of this topic you will be able to:

- *recognise appropriate use of the following standard algorithms:*
  - *input validation;*
  - *find minimum/maximum;*
  - *count occurrences;*
  - *linear search.*
- *describe these standard algorithms in pseudocode and implement them in a high level programming language.*

6.1 Revision

Revision

Q1: What complex data structure would you use to store a set of 20 student names?

a) 20 STRING variables
b) Array of CHARACTER
c) ARRAY of STRING
d) A file of STRING

. . . .

Q2: An index is?

a) An ARRAY of INTEGER
b) A list of numbers
c) The position in an ARRAY
d) A list of INTEGERS

. . . .

Q3: Which one of these is best used to describe the algorithm when designing software?

a) Source code
b) Pseudocode
c) Binary code
d) Machine code

. . . .

Q4: What kind of loop is this pseudocode an example of?

```
FOR counter from 0 TO 9 DO
  SEND "Hello world" TO DISPLAY
END FOR
```

a) A fixed loop
b) A conditional loop
c) A repeated loop
d) A indexed loop

. . . .

6.2 Standard algorithms

Learning Objective

By the end of this section you will be able to:

- recognise appropriate use of the following standard algorithms:
    - input validation;
    - find minimum/maximum;
    - count occurrences;
    - linear search.

There are certain **algorithms** that appear in program after program. These are called **standard algorithms**.

The first one we are going to look at is **input validation**. This is the task of making sure that the data input by the user is acceptable e.g. in a suitable format and within the upper and lower limits of the data required by the software, so that the program is both **robust** and **reliable**.

The other three algorithms we are going to examine all operate on lists of values: finding the maximum or minimum value, searching for a value, and counting the occurrences of a value. A common data structure used to store a list in a program is an **array**.

6.3 Input validation

Learning Objective

By the end of this section you will be able to:

- describe the Input validation algorithm in pseudocode and implement it in a high level programming language.

Input validation can be achieved in a number of ways: you can restrict the data the user can input by only presenting them with a limited set of possibilities such as a drop box or pull down menu, or you can accept their input but reject any which does not meet the restrictions imposed by the software and ask them to input it again. This second approach is the one we are going to study.

In this algorithm we are going to assume that the input from the user is going to come from the keyboard, and that the user will be asked for a value repeatedly until they provide one which meets the requirements of the program.

This means that we have to use a conditional loop to check to see whether the data is valid or not.

This algorithm uses a WHILE . . . DO . . . END WHILE conditional loop.

The user inputs the value then the **conditional loop** checks in case it is an invalid entry and asks for the value again.

```
PROCEDURE InputValidation()

 RECEIVE userInput FROM (INTEGER) KEYBOARD

 WHILE userInput < lowerLimit OR userInput > upperLimit DO

    SEND "Input must be between "& lowerLimit & " and " & upperLimit TO DISPLAY
    RECEIVE userInput FROM (INTEGER) KEYBOARD

 END WHILE

END PROCEDURE
```

We could use a REPEAT . . . UNTIL loop to do the same job.

The user inputs the value then the IF . . . THEN . . . END IF control structure checks to see if it is an invalid entry. Note that this algorithm is less efficient than the previous one because the input is being checked twice.

```
PROCEDURE InputValidation()

 REPEAT

   RECEIVE userInput FROM (INTEGER) KEYBOARD

    IF userInput < lowerLimit OR userInput > upperLimit THEN
       SEND "Input must be between " &  lowerLimit & " and "
       & upperLimit TO DISPLAY
    END IF

 UNTIL userInput >= lowerLimit AND userInput <= upperLimit

END PROCEDURE
```

Practical task: Algorithms 1

10 min

Implement one of these algorithms in your chosen programming language to ask the user for a number between 1 and 100.

. . . .

Input validation for strings follows the same pattern:

```
PROCEDURE InputValidation()

 RECEIVE userInput FROM (STRING) KEYBOARD

 WHILE userInput ≠ ["Y"] AND userInput ≠["N"] DO

  SEND "Input must be Y or N " TO DISPLAY
  RECEIVE userInput FROM (STRING) KEYBOARD

 END WHILE

END PROCEDURE
```

Practical task: Algorithms 2

10 min

Write an input routine in your programming language which only accepts Y or N, but accepts them in upper or lower case.

. . . .

Sometimes you may wish to limit the length of an input string.

This version of the input validation algorithm uses the length function to check the length of the *userInput* string against the *lengthLimit* value.

```
PROCEDURE InputValidation()

 RECEIVE userInput FROM (STRING) KEYBOARD

 WHILE (length(userInput) > lengthLimit) DO

   SEND "Input must be less than " & lengthLimit & " Characters" TO DISPLAY
   RECEIVE userInput FROM (STRING) KEYBOARD

 END WHILE

END PROCEDURE
```

Using a Boolean flag

In this version of the algorithm the **boolean** variable `validinput` is initialised to false, and the conditional loop only terminates when it has been set to true. This version of the algorithm is useful if you want to check a number of different conditions in the input string.

```
PROCEDURE InputValidation()

 SET validInput TO false

 REPEAT

  RECEIVE userInput FROM (STRING) KEYBOARD

  IF  (length(userInput) < lengthLimit) THEN
    SET validInput TO TRUE
  ELSE
    SEND "Input must be less than " & lengthLimit & " characters" TO DISPLAY
  END IF

 UNTIL validInput = true

END PROCEDURE
```

Practical task: Algorithms 3

30 min

You have been asked to write an input validation routine in your programming language to input a telephone number which can only contain the characters 0,1,2,3,4,5,6,7,8,9 and must be exactly 12 characters long.

(Note: if your programming language does not store a string as an array but stores them as a simple data type, you will have to use a string function to check each digit in turn.)

. . . .

6.4 Finding the minimum or the maximum value in an array

Learning Objective

By the end of this section you will be able to:

- describe the Find Max and Min algorithm in pseudocode and implement it in a high level programming language.

To understand this algorithm, you have to remember that the processor can only do one thing at a time. The contents of the array to be examined can only be looked at individually. The *Finding the Maximum* algorithm uses a variable which will store the largest value in the array and at the beginning of the algorithm and makes it equal to the first item in the array. The rest of the array is then checked through one by one using an **fixed loop** comparing the contents with this variable and updating its value whenever a larger item is discovered.

Activity: Find the maximum value in an array

Imagine a row of cards numbered from 0 to 9, each has a numeric value shown in the following table:

ID	Value
0	42
1	13
2	56
3	20
4	34
5	74
6	29
7	105
8	149
9	64

Maximum value

Each time you select a card, the value of that card is displayed as the maximum value IF it is greater than the value before. For example, card 4, followed by card 5 would display the value "74".

Q5: If you check the value of cards 0 through to 4 what would be the maximum value at this stage?

....

Q6: If you check the value of cards 0 through to 5 what would be the maximum value at this stage?

....

Q7: If you check the value of cards 0 through to 7 what would be the maximum value at this stage?

....

Q8: If you select card 5, then 0, then 6, what value would be the maximum value?

....

Q9: If you check the value of cards 0 through to 9 what would be the maximum value at this stage?

....

Assuming we have an array of 10 values: numbers[9] OF INTEGER

This algorithm sets maximumValue to the first item in the array then compares it to every item in the rest of the array. If one of these items is higher than the maximum, then it becomes the new maximum.

```
PROCEDURE  FindMax()

 SET maximumValue TO numbers[0]
  FOR counter FROM 1 TO 9 DO
   IF maximumValue < numbers[counter] THEN
    SET maximumValue TO numbers[counter]
   END IF
  END FOR
 SEND "The largest value was "& maximumValue TO DISPLAY

END PROCEDURE
```

....

10 min

Practical task: Finding the Maximum

Implement this algorithm in your programming language to find the maximum value in an array of 10 numbers.

....

The Finding the Minimum algorithm is very similar. Make the variable which will store the smallest value equal to the first item in the array then check through the rest of the array comparing each value in turn, swapping it if a smaller one is found.

```
PROCEDURE  FindMin()

 SET minimumValue TO numbers[0]
  FOR counter FROM 1 TO 9 DO
   IF minimumValue > numbers[counter] THEN
    SET minimumValue TO numbers[counter]
   END IF
  END FOR
 SEND "The smallest value was " & minimumValue TO DISPLAY

END PROCEDURE
```

If you want to know where in the array the value was found (ie. The **index** of the maximum or minimum value), as before you would use a fixed loop with a counter. The counter is then used to set the value of the foundAt variable whenever the minimumValue variable is updated.

```
PROCEDURE  FindMin()

 SET foundAt TO 0
 SET minimumValue TO numbers[0]

 FOR index FROM 1 TO 9  DO
  IF minimumValue > numbers[index] THEN
    SET minimumValue TO numbers[index]
    SET foundAt TO index
  END IF
 END FOR
 SEND "The smallest value was "& minimumValue & " at position "
       & foundAt &  " in the list" TO DISPLAY

END PROCEDURE
```

Practical task: Find winner

Implement this algorithm in your programming language to find the name of the winner of a race when the results are stored in two parallel arrays: names[9] and times[9].

20 min

. . . .

6.5 Counting Occurrences

Learning Objective

By the end of this section you will be able to:

- describe the Counting Occurrences algorithm in pseudocode and implement it in a high level programming language.

The Counting Occurrences algorithm also uses an fixed loop to process an array. A variable that stores the number of times a particular value occurs is set to zero at the beginning of the procedure. It is then incremented every time the search value is identified in the array.

Activity: Counting Occurrences

Imagine a row of cards numbered from 0 to 9, each has a value shown in the following table:

ID	Value
0	A
1	B
2	C
3	D
4	B
5	A
6	A
7	B
8	E
9	C

Item to find

Occurrences

Each card has a value between A and E. Only one card can be revealed at a time in sequential order from 0 to 9.

The **Item to find** box is where you enter the value you wish to find.

What would be the output in the **Occurences** box if the item to be found were:

Q10: A?

....

Q11: B?

....

Q12: C?

....

Q13: D?

....

Q14: E?

....

Assuming we have an array of 10 values: numbers[9] OF INTEGER

This algorithm sets maximumValue to the first item in the array then compares it to every item in the rest of the array.

```
PROCEDURE CountOccurrences()

 RECEIVE itemToFind FROM (INTEGER) KEYBOARD

 SET numberFound TO 0

 FOREACH number FROM numbers DO
  IF number = itemToFind THEN
    SET numberFound TO numberFound + 1
  END IF
 END FOREACH

 SEND "There were " & numberFound & "occurrences of " & itemToFind &
      " in the list" TO DISPLAY

END PROCEDURE
```

. . . .

Practical task: Occurrences

15 min

Implement this algorithm in your programming language to find the number of times a the character "e" occurs in a string typed at the keyboard.

(Note: if your programming language does not store a string as an array but stores them as a simple data type, you will have to use a string function to check each character in turn.)

. . . .

Practical task: Occurrences 2

10 min

Adapt your program to ask for the character you wish to search for as well as the phrase to be searched. Use input validation to make sure that the user only inputs a single character.

. . . .

6.6 Linear search

Learning Objective

By the end of this section you will be able to:

- describe the Linear Search algorithm in pseudocode and implement it in a high level programming language.

The linear search algorithm is used to find an item in a list. In this algorithm, a conditional loop is used to compare each item to the search term; the loop terminates when the search term is found.

A boolean variable is set to false at the beginning, then set to true when the item is found. A counter is used to keep track of where the item was found and the algorithm stops when either the first occurrence of the item has been found and the boolean variable has been set to true or the end of the array has been reached.

Activity: Linear Search

Imagine a row of cards numbered from 0 to 9, each has a value shown in the following table:

Index	Value
0	A
1	B
2	C
3	D
4	E
5	F
6	G
7	H
8	I
9	J

Item to find

Found

Found at

Each card has a value between A and J. Only one card can be revealed at a time in sequential order from 0 to 9.

The **Item to find** box is where you enter the value you wish to find.

The **Found** box value is either TRUE or FALSE with a initial value of FALSE.

The **Found at** box value is the index value when your item is found.

Q15: If the item to find is C, what would be the **Found at** value be?

. . . .

Q16: If the item to find is H, what would be the **Found at** value be?

. . . .

Q17: If the item to find is D, and the **Found at** value is 8, what would the **Found** value be?

. . . .

Q18: If the item to find is F, and the **Found at** value is 5, what would the **Found** value be?

. . . .

Assuming we have an array of 10 values: numbers[9] OF INTEGER

This algorithm sets maximumValue to the first item in the array then compares it to every item in the rest of the array.

```
PROCEDURE LinearSearch()

 RECEIVE itemToFind FROM (INTEGER) KEYBOARD

 SET found TO false
 SET arraySize TO highestIndex
 SET counter TO 0

 REPEAT
  SET counter TO counter + 1
  UNTIL counter > arraySize

  IF found = true THEN
    SEND itemToFind & " found at position" & (counter - 1) TO DISPLAY
  ELSE
    SEND "Item not found" TO DISPLAY
  END IF

END PROCEDURE
```

. . . .

15 min

Practical task: Linear Search

Implement this algorithm in your programming language to search for a name in a string array names[9].

. . . .

60 min

Practical task: Linear Search

Create a program in your programming language which does the following:

- Fills an array with random integers between 1 and 10.
- Prints the array contents to screen.
- Finds and displays the maximum and minimum values in the array.
- Asks the user for an integer between 1 and 10 using input validation and displays how many times it occurs in the array.
- Asks the user for an integer between 1 and 10 using input validation and displays where in the array that number first occurs and indicates when it is not present.

. . . .

6.7 Learning points

Summary

You should now know:

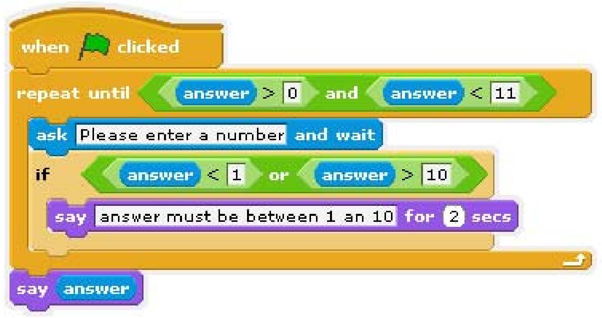
- Input validation is used to ensure that software is robust by repeatedly asking the user for input data and rejecting invalid data until the data meets the restrictions imposed by the software;
- Finding the maximum or minimum, counting occurrences and linear search all operate on arrays;
- Finding the maximum or minimum sets an initial value to the first item in the array then compares it to the remaining items;
- Counting occurrences sets a total to zero at the beginning and increments it as items are found to match the search item;
- The linear search sets a boolean variable to false initially and uses a conditional loop to set it to true when the item is found. The loop terminates when the item is found or the end of the array is reached.

6.8 End of topic test

End of topic test

10 min

Q19:



```
when [flag] clicked
repeat until < <(answer) > [0]> and <(answer) < [11]> >
    ask [Please enter a number] and wait
    if < <(answer) < [1]> or <(answer) > [10]> >
        say [answer must be between 1 an 10] for (2) secs
say (answer)
```

This is an example of?

a) Counting occurrences
b) Input validation
c) Linear search
d) Finding the maximum

. . . .

Q20:

```
Private Sub Command1_Click()
 Dim numbers(10) As Integer
 Dim counter as Integer

 For counter = 0 To 10
 numbers(Counter) = Int(Rnd * 10) + 1
 Next counter

 value = numbers(0)
 For counter = 1 To 10
 If Value < numbers(counter) Then value = numbers(Counter)
 Next counter

 Print value
End Sub
```

This is an example of?

a) Counting occurrences
b) Input validation
c) Linear search
d) Finding the maximum

. . . .

Q21: The linear search algorithm uses?

a) A fixed loop and a boolean variable
b) A conditional loop and a boolean variable
c) A maximum value and a boolean variable

d) A minimum value and a boolean variable

....

Q22: The counting occurrences algorithm uses?

a) A fixed loop
b) A conditional loop and a boolean variable
c) A conditional loop
d) A minimum value and a boolean variable

....

Q23:

```
Private Sub Command1_Click()
 Dim numbers(10) As Integer
 Dim counter as Integer

 For Counter = 0 To 10
 numbers(Counter) = Int(Rnd * 10) + 1
 Next Counter

 total = 0
 value = 5
 For counter = 0 To 10
  If value = numbers(counter) Then
   total = total + 1
  End If
 Next counter
 Print total
End Sub
```

This is an example of?

a) Counting occurrences
b) Input validation
c) Linear search
d) Finding the maximum

....

Topic 7

Computational constructs

Contents

Prerequisite knowledge

From your studies at National 5 you should already know:

- *assign a value to a variable;*
- *use arithmetic and logical operators;*
- *use fixed and conditional loops;*
- *use simple and complex conditional statements;*
- *use pre-defined functions.*

Learning Objectives

By the end of this topic you will be able to:

- *understand the difference between the scope of a global variable and a local variable;*
- *create and use sub-programs in your programming language;*
- *create your own user-defined functions;*
- *understand the concept of parameter passing and the difference between actual and formal parameters;*

- *understand the difference between passing a parameter by value and passing a parameter by reference;*
- *use your chosen programming language to transfer data to and from sequential files.*

7.1 Revision

Revision

Q1:

```
SET startPosition TO 1
```

This is an example of?

a) Assignment
b) Definition
c) Declaration
d) Optimisation

. . . .

Q2:

```
SET userInput TO validItem
```

`ValidItem` is an example of?

a) A function
b) A procedure
c) A user defined function
d) A definition

. . . .

Q3:

```
IF age <=21 THEN checkId
```

This is an example of?

a) A complex conditional
b) A simple conditional
c) A conditional loop
d) An unconditional

. . . .

Q4:

```
IF  inputNumber >= 1 AND inputNumber <= 10 THEN
  SEND "Number OK" TO DISPLAY
ELSE
  SEND "Invalid entry" TO DISPLAY
END IF
```

This is an example of?

a) A simple conditional
b) A conditional loop
c) A complex conditional
d) An unconditional

....

Q5: When the program segment in question 4 is running, what would be the result if the value of `inputNumber` was 11?

....

Q6: When the program segment in question 4 is running, what would be the result if the value of `inputNumber` was 1?

....

Q7:

```
FOR  counter FROM 0 TO 9 DO
   SET numbers[counter] TO rand(100)
END FOR
```

This is an example of?

a) A fixed loop
b) A conditional loop
c) A conditional statement
d) An unconditional

....

7.2 Introduction

A computational construct is a system of data representation and control structures used to solve problems using a computer through a programming language. What we are doing with any computer program is **storing and manipulating information**. Computational constructs are the features of a high level language which have been designed to make this task easier.

Although there are only three control structures: **sequence**, **selection** and **iteration**, to perform all programming tasks, code can be made more understandable if these control structures can be combined to make more powerful computational constructs.

This unit will look at a variety of constructs which are used in modern programming languages, and how they make the task of writing solutions to problems in a high level programming language easier.

7.3 Variables and scope

Learning Objective

By the end of this section you will be able to:

- understand the difference between the scope of a global variable and a local variable.

When a variable is created in a program this is called **variable declaration**. When a variable is declared, most languages allow it to be declared as being of a particular type and structure, depending on the kind of data it is required to hold. Once a variable has been declared, it can be given a value. The value it has can then be used or changed (varied) during the running of the program.

Modern programming environments enable the programmer to create sub-programs within their main program. These sub-programs will correspond to the tasks which have been identified in the top down development process when the initial problem has been broken down into smaller sub-problems. Making these sub-programs as self-contained as possible is a good idea because if they contain variables which can change a value elsewhere in the code, it is often difficult to predict what the effects of this will be. This also improves the **modularity** of the code, so that the sub-programs can be tested independently, and also improves the portability of the code allowing the sub-programs to be used elsewhere without alteration.

A sub-program can only be self- contained if the variables declared and used within it are **local variables**. A local variable is one which only has a value within the sub-program where it is being used. This is often referred to as the **scope** of a variable. In most programming environments, the default for a variable is to be local to the sub-programs where they have been declared. Since a local variable only has a value inside its own sub-program, the variable name can be used again elsewhere without the danger of having an unexpected effect elsewhere.

A **global variable** has a wider scope - it exists and can be altered throughout the entire program. This means that if its value is changed inside a sub-procedure, that value will remain changed and will affect its value wherever in the program it is used, possibly unintentionally. A sub-procedure which uses a global variable will not be self contained, and is not going to be able to be used in a different context where that global variable does not exist. For these reasons, global variables are best avoided whenever possible.

7.4 Sub-programs

Learning Objective

By the end of this section you will be able to:

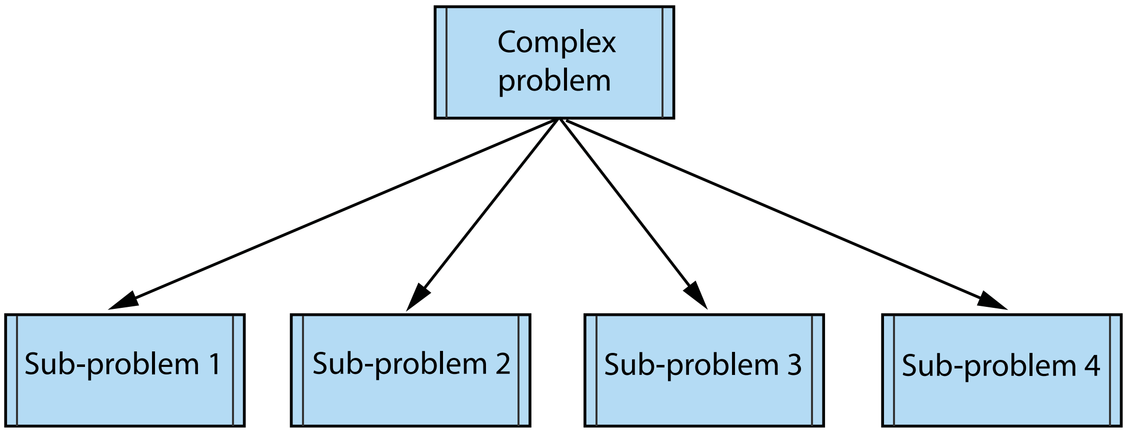
- create and use sub-programs in your programming language.

Sub-programs are named blocks of code which can be run from within another part of the program. When a sub-program is used like this we say that it is "called". Because they can be called from any part of the program they can be used over again if needed. For instance, an input validation sub-program may be called several times asking for different inputs. This makes your program easier to understand and makes writing code more efficient.

Sub-programs are often called **procedures** (which execute a set of commands), or **functions** (which return a value). In the case of object-oriented programming languages (such as Java) they are called **methods**. Breaking your program down into sub-programs is a good idea because it makes your code **modular**, readable and therefore more maintainable.

The structure of a program should follow the top down analysis which was originally used to break the problem down into smaller sub-tasks.

```
<main program>
      subProgram1()
      subProgram2()
      subProgram3()
      subProgram4()


<subProgram1>

<subProgram2>

<subProgram3>

<subProgram4>
```



For example the exercise in topic 6 might have the following structure:

Note:

- In this example we are using a global variable: numbers[9] of INTEGER.
- rand() is a function which returns a random number, so rand(100) returns a random number between 1 and 100.
- GetValidInteger() is a user defined function which returns an integer value between 1 and 100 when the function is called.

```
FillArray()
PrintArray()
FindMinimum()
FindMaximum()
CountOccurrences()
LinearSearch()

PROCEDURE FillArray()

 FOR  counter FROM 0 TO 9 DO
   SET numbers[counter] TO rand(100)
 END FOR

END PROCEDURE

PROCEDURE PrintArray()

 FOR  counter FROM 0 TO 9 DO
   SEND numbers[counter] TO DISPLAY
 END FOR

END PROCEDURE

PROCEDURE FindMaximum()

 SET maximumValue TO numbers[0]
 FOR counter FROM 1 TO 9 DO
```

```
   IF maximumValue < numbers[counter] THEN
       SET maximumValue TO numbers[counter]
   END IF
 END FOR
 SEND "The largest value was " & maximumValue TO DISPLAY

END PROCEDURE

PROCEDURE FindMinimum()

 SET minimumValue TO numbers[0]
 FOR counter FROM 1 TO 9 DO
   IF minimumValue > numbers[counter] THEN
       SET minimumValue TO numbers[counter]
   END IF
 END FOR
 SEND "The smallest value was " & minimumValue TO DISPLAY

END PROCEDURE

PROCEDURE CountOccurrences()

 SET itemToFind TO GetValidInteger()
 SET numberFound TO 0
 FOR EACH number FROM numbers DO
   IF number = itemToFind THEN
     SET numberFound TO numberFound + 1
   END IF
 END FOREACH
 SEND "There were " & numberFound & "occurrences of " & itemToFind
     & " in the list" TO DISPLAY

END PROCEDURE


FUNCTION GetValidInteger() RETURNS INTEGER

 RECEIVE userInput FROM (INTEGER) KEYBOARD
 WHILE userInput < 1  OR userInput > 100 DO
   SEND "Input must be between 1 and 100 "  TO DISPLAY
   RECEIVE userInput FROM (INTEGER) KEYBOARD
 END WHILE
 RETURN userInput

END FUNCTION
```

```
PROCEDURE LinearSearch()

  SET itemToFind TO GetValidInteger()
  SET found TO false
  SET arraySize TO highestIndex
  SET counter TO 0

   REPEAT
    SET found TO numbers[counter] = itemToFind
    SET counter TO counter + 1
   UNTIL found OR counter > arraySize

  IF found THEN
    SEND itemToFind &" found at position" & counter - 1 TO DISPLAY
  ELSE
    SEND "Item not found" TO DISPLAY
  END IF

END PROCEDURE
```

Practical task: Linear search

60 min

Edit your programming language solution to the exercise at the end of Topic 6 to reflect the pseudocode structure above.

The exercise was to write a program in your programming language which does the following:

- Fills an array with random integers between 1 and 10.
- Prints the array contents to screen.
- Finds and displays the maximum and minimum values in the array.
- Asks the user for an integer between 1 and 10 using input validation and displays how many times it occurs in the array.
- Asks the user for an integer between 1 and 10 using input validation and displays where in the array that number first occurs and indicates when it is not present.

. . . .

Methods

A method in an object-oriented language is a function that is defined inside a class.

In our example we would create the class `mainProgram` with the `GetValidInteger` function and other methods defined within it.

Object-oriented programming languages often use the syntax:

```
object.functionName()
```

So our `GetValidInteger` function would be called like this

```
mainProgram.GetValidInteger()
```

7.5 User defined functions

Learning Objective

By the end of this section you will be able to:

- create your own user-defined functions.

A sub-program performs sequence of actions and usually have names which are verbs. A function returns a value. Function names are usually nouns.

Your programming language will have a number of pre-defined functions available. Examples of common numeric functions are:

- int(n) returns the integer value of n
- abs(n) returns the absolute value of n
- sqr(n) returns the square root of n

```
SET value TO 3.4
SET newValue TO int(value)
SEND newValue TO DISPLAY
```

Result would be *3*

```
SET value TO -56
SET newValue TO abs(value)
SEND newValue TO DISPLAY
```

Result would be *56*

```
SET value TO 4
SET newValue TO sqr(value)
SEND newValue TO DISPLAY
```

Result would be *2*

In languages where strings are simple data structures, examples of common string functions are:

- left(string, n) returns the first n characters of **string**
- right(string, n) returns the last n characters of **string**
- mid(string, r, n) returns n characters of **string** starting at r

```
SET myString TO "elephant"
Set newString TO left(myString, 3)
SEND newString TO DISPLAY
```

Result would be *ele*

```
SET myString TO "elephant"
Set newString TO right(myString, 3)
SEND newString TO DISPLAY
```

Result would be *ant*

```
SET myString TO "elephant"
Set newString TO mid(myString, 3, 5)
SEND newString TO DISPLAY
```

Result would be *phant*

```
SET myString TO "elephant"
SEND length(myString) TO DISPLAY
```

Result would be *8*

In the previous example we used our own user-defined function `GetValidInteger()` to return a value between 1 and 100.

```
FUNCTION GetValidInteger() RETURNS INTEGER

 RECEIVE userInput FROM (INTEGER) KEYBOARD

 WHILE userInput < 1  OR userInput > 100 DO
   SEND "Input must be between 1 and 100 "  TO DISPLAY
   RECEIVE userInput FROM (INTEGER) KEYBOARD
 END WHILE

 RETURN userInput

END FUNCTION
```

30 min

Practical task: User-defined functions

Create your own user-defined functions to return the following.

```
FUNCTION NewRandom()RETURNS INTEGER
```

which returns a random number between 50 and 100.

```
STRING FUNCTION UserName()
```

which returns a string with a maximum length of 10 characters.

....

7.6 Parameters

Learning Objective

By the end of this section you will be able to:

- understand the concept of parameter passing and the difference between actual and formal parameters.

A more flexible solution to the input validation problem in the previous section would be to use **parameters** to set the range of numbers we wanted to return from the GetValidInteger function when we called it. This would mean that the function could be called with different values (or variables) depending on what range of number we wanted to restrict it to.

Formal parameters are the parameters within brackets in the declaration of a function or procedure. A function or procedure without formal parameters will still have a set of empty brackets after the name in its declaration. Many programming languages will require these formal parameters to be given a data type in the declaration as well as a name. In this example the `GetValidInteger` function is declared with two formal parameters, `lowerLimit` and `upperLimit`.

Sub-programs are defined with **formal** parameters and called with **actual** parameters.

```
FUNCTION GetValidInteger(lowerLimit, upperLimit) RETURNS INTEGER

  RECEIVE userInput FROM (INTEGER) KEYBOARD

  WHILE userInput < lowerLimit OR userInput > upperLimit DO
     SEND "Input must be between "& lowerLimit &" and "& upperLimit TO DISPLAY
     RECEIVE userInput FROM (INTEGER) KEYBOARD
  END WHILE

  RETURN userInput

END FUNCTION
```

This function can now be used in any program to return a valid number within the range provided.

We could call this function with actual parameters, 1 and 50 to return a number between 1 and 50:

```
numberToUse = GetValidInteger(1,50)
```

or we could call it with the actual parameters 1 and inputRange - a variable which has a value assigned elsewhere in the program:

```
RECEIVE   inputRange  FROM (INTEGER) KEYBOARD
numberToUse = GetValidInteger(1,inputRange)
```

This call would return a value between 1 and whatever was stored/held in the variable `inputRange`.

Practical task: Parameters 1

Edit the code for the previous exercise to use the user-defined `GetValidInteger` function with parameters.

15 min

. . . .

15 min

Practical task: Parameters 2

Create your own user-defined functions to return the following

```
FUNCTION Double(value) RETURNS INTEGER
```

which returns a number which is double the number passed into it.

```
FUNCTION UserName(forename, yearOfBirth) RETURNS STRING
```

which returns a string which concatenates the forename and birth year passed into it.

....

Procedures can have formal parameters as well. For example we could declare the `PrintArray` sub-procedure with the formal integer array, numbers. Again many programming languages will require that formal parameters are given a data type as well as a name.

```
PROCEDURE PrintArray(numbers)

  FOR  counter FROM 0 TO 9 DO
     SEND numbers[counter] TO DISPLAY
  END FOR

END PROCEDURE
```

This would mean that the numbers array could be declared as a local variable within the main program, and then passed as an actual parameter to each one of the sub-programs.

The code for the `CountOccurrences` sub-program would now be:

```
PROCEDURE CountOccurrences(numbers)

  SET itemToFind TO GetValidInteger( 1, 100)
  SET numberFound TO 0

  FOREACH number FROM numbers DO
    IF number = itemToFind THEN
       SET numberFound TO numberFound + 1
    END IF
  END FOREACH

  SEND "There were " & numberFound & "occurrences of " & itemToFind
& " in the list" TO DISPLAY

END PROCEDURE
```

If we now rewrite our original program using parameter passing throughout, we get:

```
PROCEDURE Main()

  fillArray(numbers)
  printArray(numbers)
  findMinimum(numbers)
  findMaximum(numbers)
  countOccurrences(numbers)
  linearSearch(numbers)

END PROCEDURE


PROCEDURE FillArray(numbers)

 FOR  counter FROM 0 TO 9 DO
   SET numbers[counter] TO rand(100)
 END FOR

END PROCEDURE


PROCEDURE PrintArray(numbers)

 FOR  counter FROM 0 TO 9 DO
   SEND numbers[counter] TO DISPLAY
 END FOR

END PROCEDURE


FUNCTION GetValidInteger(lowerlimit, upperlimit) RETURNS INTEGER

  RECEIVE userInput FROM (INTEGER) KEYBOARD
  WHILE userInput < lowerLimit OR userInput > upperLimit DO
     SEND "Input must be between "& lowerLimit &" and "& upperLimit TO DISPLAY
     RECEIVE userInput FROM (INTEGER) KEYBOARD
  END WHILE
  RETURN userInput

END FUNCTION
```

```
PROCEDURE CountOccurrences(numbers)

  SET itemToFind TO GetValidInteger(1,100)
  SET numberFound TO 0
  FOR EACH number FROM numbers DO
    IF number = itemToFind THEN
       SET numberFound TO numberFound + 1
    END IF
  END FOREACH
  SEND "There were " & numberFound & "occurrences of " &  itemToFind
& " in the list"  TO DISPLAY

END PROCEDURE


PROCEDURE LinearSearch(numbers)

 SET itemToFind TO GetValidInteger(1,100)
 SET found TO false
 SET arraySize TO highestIndex
 SET counter TO 0
 REPEAT
    SET found TO numbers[counter] = itemToFind
    SET counter TO counter + 1
 UNTIL found OR counter > arraySize
 IF found THEN
   SEND itemToFind & " found at position" & (counter-1) TO DISPLAY
 ELSE
   SEND "Item not found" TO DISPLAY
 END IF

END PROCEDURE
```

Now all the variables used in the program are local to the main sub-program and so there are no global variables. The sub-programs are modular and each one could be used again with a different array as their actual parameter - in elsewhere in this or another program. In the same way that the `GetValidInteger` function could be used again with different actual parameters to return a value within a different range.

Practical task: Parameters 3

30 min

Create a procedure which takes two parallel arrays as parameters, a set of 10 names and a set of 10 scores and prints out the highest scoring name.

. . . .

7.7 Passing parameters by value and reference

Learning Objective

By the end of this section you will be able to:

- understand the difference between passing a parameter by value and passing a parameter by reference.

Defining a sub-program (function or procedure) with formal parameters makes it possible call it with different actual values, making the program they are part of more efficient and modular. Defining sub-programs with formal parameters also means that these sub-programs can be saved as independent modules in a **module library** and re-used elsewhere.

Parameters which are passed into a procedure but not changed are said to be passed by **value**. When a variable is passed to a sub-program by value, a temporary copy of that variable is made and is used while the sub-program is running. The copy is discarded once it has finished executing.

Parameters which are passed into a procedure and may be changed by that procedure are said to be passed by **reference**. When a variable is passed to a sub-program by reference, then the formal parameter refers to the actual parameter, so any changes to the formal parameter are changes to the actual parameter.

Although simple data structures can be passed as either a value or a reference parameter, complex data structures such as arrays are almost always passed by reference.

In this example we want to swap the values of two integer variables **a** and **b**.

```
PROCEDURE Swap (REF a, REF b)
   SET  temp TO  a
   SET a TO b
   SET b TO temp
END PROCEDURE
```

We use **a** the temp variable to store the value of **a** while it is being swapped with **b**

Extension material

A common task in programming is to sort the contents of an array into ascending or descending value. A swap sub-procedure could be used to swap items in an array from one index position to another. In this example we want to ask the user for the index values of the two items to swap, and then swap their positions in the array. The formal parameters **a** and **b** are reference parameters because their values are being changed by the procedure.

```
PROCEDURE Main ()

    fillArray(numbers)
    printArray(numbers)
    SEND "Please enter the index position of first item" TO DISPLAY
    SET  index1 TO GetValidInteger(1,10)
    SEND "Please enter the index position of second item" TO DISPLAY
    SET  index2 TO GetValidInteger(1,10)
    swap(numbers[index1], numbers[index2])
    printArray(numbers)

END PROCEDURE

PROCEDURE FillArray(numbers)

 FOR  counter FROM 0 TO 9 DO
   SET numbers[counter] TO rand(100)
 END FOR

END PROCEDURE

PROCEDURE PrintArray(numbers)

 FOR  counter FROM 0 TO 9 DO
   SEND numbers[counter] TO DISPLAY
 END FOR

END PROCEDURE


PROCEDURE Swap (REF a, REF b)

   SET  temp TO  a
   SET a TO b
   SET b TO temp

END PROCEDURE


FUNCTION GetValidInteger(lowerlimit, upperlimit) RETURNS INTEGER

  RECEIVE userInput FROM (INTEGER) KEYBOARD
  WHILE userInput < lowerLimit OR userInput > upperLimit DO
     SEND "Input must be between "& lowerLimit &" and "&  upperLimit TO DISPLAY
     RECEIVE userInput FROM (INTEGER) KEYBOARD
  END WHILE

  RETURN userInput

END FUNCTION
```

Practical task: Extension material

Implement this pseudocode above in your chosen programming language.

....

7.8 Sequential files

Learning Objective

By the end of this section you will be able to:

- use your chosen programming language to transfer data to and from sequential files.

As far as the computer is concerned, data can be input from a keyboard or a file and can be output to a display or file. If a file does not already exist, it may have to be created with a specific command, or your programming language may create it as part of the OPEN command.

```
PROCEDURE GetData(numbers)
  <open file "mydata.txt">
   FOR  counter FROM 0 TO 9 DO
     RECEIVE numbers[counter] FROM (INTEGER) "mydata.txt"
   END FOR
  <close file "mydata.txt">
END PROCEDURE

PROCEDURE SaveData(numbers)
  CREATE "newdata.txt"
  <open file "newdata.txt">
   FOR  counter FROM 0 TO 9 DO
     SEND numbers[counter] TO "newdata.txt"
   END FOR
  <close file "newdata.txt">
END PROCEDURE
```

Practical task: Sequential files

Adapt the parallel arrays exercise to read the scores array from a file. Use a text editor such as Notepad to create the file.

....

7.9 Learning points

Summary

You should now know:

- A **computational construct** is a combination of control structures which can be used to make solving programming problems more intuitive.
- The scope of a variable describes where it can be accessed from.
- **Global** variables have scope throughout a program, **local** variables can only be accessed from within their own sub-procedure.
- If possible, global variables should be avoided and all variables in a program should be local.
- Breaking a problem down into smaller sub-problems enables **modular** code to be created where each sub-problem is coded as a separate procedure.
- A user-defined function is a sub-program which returns a value, and is defined as being of the data type corresponding to that value.
- A method in an object-oriented language is a function that is defined inside a class.
- Subprograms are defined with **formal** parameters and called with **actual** parameters.
- Parameters can be passed by **value** or **reference**. Parameters which are passed into a procedure but not changed are said to be passed by **value**. (They are in fact, just a copy of the actual parameter.)
- Parameters which are passed into a procedure and may be changed by that procedure are said to be passed by **reference**.
- Sequential files are treated in the same way as other input and output devices, but with specific commands for opening and closing.

7.10 End of topic test

End of topic test

10 min

Q8:

```
SET items TO 0

FOR EACH number FROM numbers DO
  IF number = item THEN
     SET total TO total + 1
  END IF
END FOREACH
```

This is an example of?

a) Find the maximum
b) Find the minimum
c) Counting occurrences
d) Linear search

....

Q9:

```
FUNCTION GetValidInteger()RETURNS STRING

 RECEIVE userInput FROM (STRING) KEYBOARD
 WHILE length(userInput > 10 DO
   SEND "Input must less than 10 characters ") TO DISPLAY
   RECEIVE userInput FROM (STRING) KEYBOARD
 END WHILE
 RETURN userInput

END FUNCTION
```

This function `GetValidInteger()` returns a?

a) Real value
b) Integer value
c) Boolean value
d) String value

....

Q10: This line of code is in a program to add the name "Fred" to an array of STRINGS:

```
addNameToList("Fred", names)
```

"Fred" and names are?

a) Formal parameters
b) Actual parameters
c) Real parameters
d) Reference parameters

....

Q11:

```
PROCEDURE addNameToList (name, listOfNames)
```

In this procedure definition, *name* and *listOfNames* are?

a) Formal parameters
b) Actual parameters
c) Real parameters
d) Reference parameters

....

Q12:

```
SET myString TO "ORANGES AND LEMONS"
Set newString TO left(9,myString)
SEND newString TO DISPLAY
```

Would display as?

a) ORANGES A
b) ND LEMONS
c) ORANGE
d) AND LEMON

....

Q13:

```
SET myString TO "ORANGES AND LEMONS"
SEND length(myString) TO DISPLAY
```

Would display as?

a) 10
b) 16
c) 18
d) 20

....

Q14: A formal parameter whose value may be changed by the procedure where it is defined is?

a) A reference parameter
b) A value parameter
c) An actual parameter
d) A real parameter

. . . .

Q15: . A formal parameter whose value can NOT be changed by the procedure where it is defined is?

a) A reference parameter
b) A value parameter
c) An actual parameter
d) A real parameter

. . . .

Topic 8

Testing and documenting solutions

Contents

Prerequisite knowledge

From your studies at National 5 you should already know:

- *why we should use normal, extreme and exceptional test data;*
- *that using internal commentary, meaningful identifiers, and indentation aids code readability.*

Learning Objectives

By the end of this topic you will be able to:

- *construct a test plan;*
- *describe comprehensive testing;*
- *describe systematic testing;*
- *explain the difference between syntax, execution and logic errors;*
- *understand how dry runs, trace tables, trace tools and breakpoints are used in the debugging process.*

8.1 Revision

Revision

Q1: Which set of test data would be the best one to use to test an input routine asking for numbers between 1 and 100?

a) 0, 1,10,50, 99,100, 101, X
b) 1 5 20 40, 50, 60, 90 100
c) 5,9,2,60,80 100, 101, A
d) 0, 1,5, 46, 67, 84, 90, 93

. . . .

Q2: Which identifier would be the best one to use for an array of integers storing test scores?

a) score
b) scores
c) s
d) values

. . . .

Q3: Which lines of code in this example should be indented to make it more readable?

```
Line 1  INTEGER FUNCTION getvalidItem()
Line 2  RECEIVE userInput FROM (INTEGER) KEYBOARD
Line 3  WHILE userInput < 1  OR userInput > 100 DO
Line 4  SEND ["Input must be between 1 and 100 "]  TO DISPLAY
Line 5  RECEIVE userInput FROM (INTEGER) KEYBOARD
Line 6  END WHILE
Line 7  RETURN userInput
Line 8  END FUNCTION
```

. . . .

Q4: What is internal commentary?

. . . .

8.2 Test plans

Learning Objective

By the end of this section you will be able to:

- construct a test plan;
- describe comprehensive testing;
- describe systematic testing.

As we have seen in the Development Methodologies topic, testing can only demonstrate the presence of errors, it cannot demonstrate their absence. For this reason, testing should be both **systematic** and **comprehensive**.

Systematic testing is where tests are done in a way which is planned, and which can be documented as a result. **Comprehensive** testing is when every aspect of the software is tested.

A **test plan** is a set of test data which has been created in order to systematically and comprehensively test the software which the client has requested in order to ensure that it meets the original specification when delivered. Much of the test plan will be created during the design stage of the software development process, because by this stage it should be known what the inputs will be and outputs should be, and also what the what the user interface looks like.

A test plan will include:

1. **The software specification against which the results of the tests will be evaluated.**

   The **software specification** is produced at the end of the analysis stage of the software development process, and is a legally binding document which protects both client and developer. The design of the test plan must take this document into account.

2. **A schedule for the testing process.**

   The testing schedule is necessary for the same reason as every other part of the software development process needs to scheduled in order to deliver the project on time.

3. **Details of what is and what is not to be tested.**

   Exhaustive testing - where every possible input and permutations of input to a program are tested - is not possible. Even a simple input validation routine could theoretically need to be tested with every possible valid number, and the possibilities run into millions once you have several different inputs which could be applied in any order. The tests selected should be ones which are practical within the time available. There will always be external circumstances which cannot be tested until the software is in the hands of the client or the user base. This is where **acceptance testing (beta testing)** is important.

4. **The test data and the expected results.**

   A test plan will include **normal**, **extreme** and **exceptional** test data; the results expected from inputting this data to the program and whether the result passes or fails the test.

This is a set of test data for a sub-program which should allow the user to input a whole number between 1 and 100.

	Data	**Expected Result**	**Actual Result**	**Pass / Fail**
Normal	46, 62, 80	accept		
Extreme	1, 100	accept		
Exceptional	0, 101, -5, 15.6, A, %	reject		

NB. It is important that values outside the input range, but on the boundary of acceptable data are included in the exceptional test data.

5. **Documentation of the testing process.**

   The testing process needs to be documented so that if problems are encountered at a later date, the test results can be checked and duplication of work avoided.

This kind of testing of a program by the developers is called **alpha testing**.

8.3 Debugging

Learning Objective

By the end of this section you will be able to:

- explain the difference between syntax, execution and logic errors.

Debugging is the process of finding and correcting errors in code.

Some errors in code will be discovered during the implementation stage. However some will only be identified at the testing stage which means that the implementation stage needs to be re-visited to correct them.

Errors likely to be spotted at the implementation stage are syntax errors and execution (runtime) errors. A **syntax error** is one which can be spotted by a translator: by a compiler when the source code is translated into machine code, or by an interpreter while the code is being entered by the programmer. A syntax error could be a misspelling of a keyword, or a mistake in the structure of a program like a missing END WHILE in a WHILE loop or a missing END IF in an IF condition.

An **execution error** is one which happens when the program is run, causing it to stop running (crash). Examples include division by zero or trying to access an array index that's beyond the range of that array. These types of error are not identified by the compiler or the interpreter, but appear when the program is run.

Logical errors, sometimes called **semantic errors**, are ones where the code is grammatically correct as far as the interpreter or compiler is concerned, but does not do what the programmer intended. These types of error may be spotted during the implementation stage, but may also be spotted during the testing stage.

Quiz: Debugging

Q5: Which of these are **not** syntax errors?

- Missing semi colon
- Division by zero
- IF without END IF
- Out of memory
- WHILE without DO

....

Q6: There is an error is this pseudocode. What kind of error is it?

```
FOR counter FROM 1 TO 20 DO
   SET total TO 0
   RECEIVE  userInput FROM (INTEGER)KEYBOARD
   SET total TO total + userInput
END FOR
SET average TO total / 20
SEND "The average of these numbers was " & average TO DISPLAY
```

....

8.4 Debugging tools

Learning Objective

By the end of this section you will be able to:

- understand how dry runs, trace tables, trace tools and breakpoints are used in the debugging process.

Debugging is made much easier if source code is well documented, and uses meaningful variable names and indentation. Modularity makes debugging easier since sub programs can be tested independently, especially if they are self contained and do not use global variables.

Syntax errors will be highlighted by the interpreter or compiler while code is being written or compiled, but logical errors can only be found by running a program and watching its operation. There are a number of techniques which can be used to monitor the values of variables at different points in the code execution to aid this process.

8.4.1 Dry runs

A dry run is simply a manual run-through the pseudocode or source code of the program, usually taking notes of the values of variables at various points in the process while doing so. In effect the person doing the dry run is taking the place of the computer in order to check that the code is doing what they expect it to do. Keeping track of the values of variables at different stages of the code execution is complicated so normally the tester would use a table, either on paper or on computer to help.

8.4.2 Trace tables

A trace table is similar to the table that would be used during a dry run, but is often used to test an algorithm for a specific sub program when the tester wants to check the result of a number of different values of a variable.

Algorithm:

```
SET total TO  0
FOR counter FROM 1 TO 5 DO
   RECEIVE userInput FROM (INTEGER)KEYBOARD
   SET total TO  total + userInput
END FOR
SET average TO  total / 5
SEND "The average of these numbers was "& average TO DISPLAY
```

Trace table:

Total	Counter	userInput	average
0	1	3	0
3	2	7	0
10	3	4	0
14	4	11	0
25	5	11	5

Output: The average of these numbers was 5.

Algorithm:

```
PROCEDURE linearSearch(numbers,)

 SET itemToFind TO 10

 SET found TO false
 SET arraySize TO 4
 SET counter TO 0

 REPEAT
   IF number[counter] = itemToFind THEN
    SET found TO true
   END IF
    SET counter TO counter + 1
 UNTIL found OR counter > arraySize

 IF found THEN
    SEND itemToFind &" found at position" & counter - 1 TO DISPLAY
 ELSE
```

```
     SEND "Item not found" TO DISPLAY
    END IF

   END PROCEDURE
```

Trace table:

itemToFind	found	arraysize	counter
10	false	4	0
10	false	4	1
10	false	4	2
10	true	4	3

Output: 10 was found at position 2 in the list.

Note: arrays are indexed from zero.

Practical task: Trace tables

10 min

Create a trace table for this algorithm:

```
SET numbers TO [3, 15, 4, 7, 8]

PROCEDURE findMaximum(numbers)

 SET maximumValue TO numbers[0]
 FOR counter FROM 1 TO 5 DO
    IF maximumValue < numbers[counter] THEN
        SET maximumValue TO numbers[counter]
    END IF
 END FOR
 SEND "The largest value was "& maximumValue TO DISPLAY

END PROCEDURE
```

....

8.4.3 Trace tools

Some programming environments have **trace** facilities as a debugging feature. Tracing tools let the programmer see which lines of code are being executed and what variables are changing their value while the program is running.

A **watch** takes a variable and displays its value as the program progresses. The programmer steps through the code, one statement at a time, and the value of the variable being traced is displayed on the watch screen which can be set to stop when reaches a particular value.

Some trace tools also allow investigation of actual memory locations and, in particular, the contents of the **stack**.

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

8.4.4 Breakpoints

Some programming environments will enable the programmer to set a **breakpoint**.

Setting a breakpoint in a program sets a point in the source code where the program will stop execution, at which point the values of variables at this point can be examined.

Breakpoints can be set to stop execution at a particular point in code, or to stop when a variable has a particular value (watch) or a particular key is pressed.

Once the program has stopped, the values of the variables in use can be examined, or written to a file for study later.

8.5 Learning points

Summary

You should now know:

- Testing can only demonstrate the presence of errors - it cannot demonstrate their absence.
- A test plan is a set of test data which should systematically and comprehensively test the software to ensure that it meets the original specification.
- A test plan will include normal, extreme and exceptional test data.
- A syntax error is where the "grammatical" rules of the language have been broken. It is normally detected spotted by a compiler or interpreter.
- A logic error is one where the code is grammatically correct but does not do what the programmer intended.
- An execution error is one which causes the program to stop (crash) when it is run.
- A dry run is a manual run through pseudocode or the source code of the program.
- A breakpoint is a set point in a program where it will stop execution so that the values of variables can be examined.

8.6 End of topic test

End of topic test

10 min

Q7: Which of these are syntax errors?

- Missing semi colon
- Division by zero
- IF without END IF
- Out of memory
- WHILE without DO

. . . .

Q8: Which of these are execution errors?

- Missing semi colon
- Division by zero
- IF without END IF
- Out of memory
- WHILE without DO

. . . .

Q9: Fill in the missing values in this trace table:

miniimumValue	counter	numbers[counter]
	1	
	2	
	3	
	4	

```
SET numbers TO [17, 15, 4, 7, 8]

PROCEDURE findMinimum(numbers)

 SET minimumValue TO numbers[0]
 FOR counter FROM 1 TO 5 DO
   IF minimumValue > numbers[counter] THEN
       SET minimumValue TO numbers[counter]
   END IF
 END FOR
 SEND "The smallest value was "& minimumValue TO DISPLAY

END PROCEDURE
```

. . . .

Q10: A computer program is designed to accept input values between 0 and 99 as whole numbers. If the value 99 was entered this would be an example of?

a) Exceptional data
b) Normal data
c) Invalid data
d) Extreme data

....

Q11: Alpha testing is carried out by end users of a program.

a) True
b) False

....

Q12: Beta testing is carried out by end users of a program.

a) True
b) False

....

Q13: Fill in the missing values in this trace table:

value	display	counter
1	no display	0
		1
		2
		3
		4
		5
		6

```
PROCEDURE sequence()
 SET value TO  1
 SET counter TO  0
 REPEAT UNTIL counter > 5
   SET value TO value + counter
   SEND value TO DISPLAY
   SET counter TO counter + 1
 END REPEAT
END PROCEDURE
```

....

Q14: Which of these source code characteristics do NOT help in debugging a program?

a) Internal documentation
b) Modular code
c) Global variables
d) Meaningful variable names

....

Topic 9

Computer architecture

Contents

Prerequisite knowledge

From your studies at National 5 you should already know:

- *the difference between RAM and ROM;*
- *that the processor and memory communicate via collections of lines called buses;*
- *that peripheral devices need an interface in order to communicate with the processor.*

Learning Objectives

By the end of this topic you will be able to:

- *describe the function of the registers, the arithmetic and logic unit and the control unit in a processor;*
- *describe the role of the control, address and data buses in the fetch execute cycle;*
- *describe how cache memory affects processor performance;*
- *describe modern trends in computer architecture;*
- *understand what an emulator is and how it is used;*
- *describe the concept of a virtual machine;*

- *understand the influence that mobile devices have on the software development process.*

9.1 Revision

Revision

Q1: Which one of these statements is false?

a) ROM is memory whose contents cannot be changed.
b) RAM is memory whose contents cannot be changed.
c) RAM stores data and instructions while a computer is running.
d) ROM can be used to store part of a computer operating system.

. . . .

Q2: Which function of an interface is responsible for informing the user that a printer is out of ink?

a) Data conversion.
b) Data storage.
c) Transferring status information.
d) Protocol conversion.

. . . .

Q3: What is a buffer?

a) A safety device inside the processor.
b) Memory used to store information being transferred by an interface.
c) Memory which is part of the processor.
d) Part of an interface used to convert data from one format to another.

. . . .

Q4: Which of these buses is used to identify a memory location for reading or writing?

a) The address bus.
b) The data bus.
c) The control bus.
d) The system bus.

. . . .

Q5: Which of these buses is used to transfer data between processor and memory?

a) The address bus.
b) The data bus.
c) The control bus.
d) The system bus.

. . . .

9.2 The parts of the processor

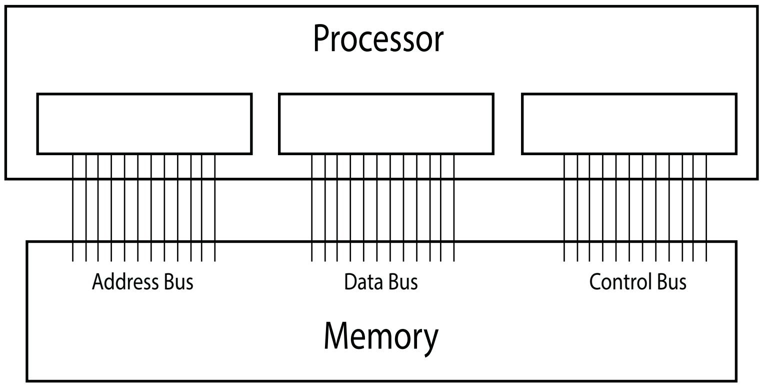
Learning Objective

By the end of this section you will be able to:

- describe the function of the registers, the arithmetic and logic unit and the control unit in a processor.

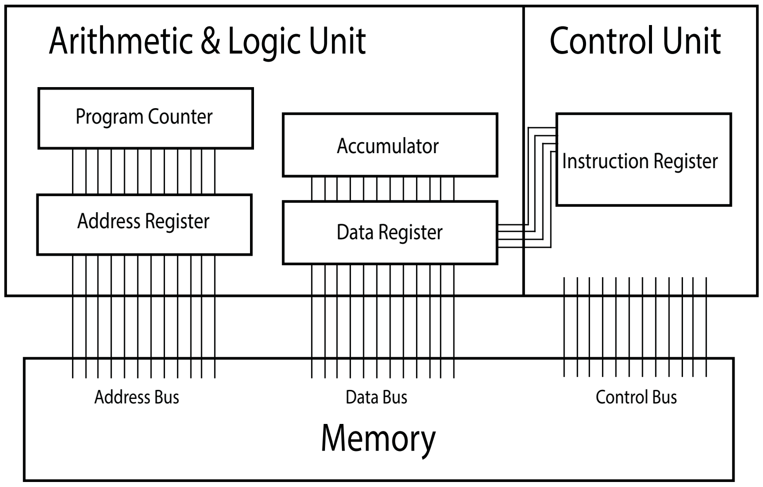
The CPU consists of several different parts: the **Arithmetic and Logic Unit** which performs calculations; the **Control Unit** which loads, decodes and executes instructions, and **Registers** which are small memory locations used by the processor.

Buses are the lines which connect the CPU to the main memory.



Registers are temporary storage areas in the processor which can be used to hold information such as:

- the address of the next instruction to be fetched (Program Counter);
- the address of the memory location where data is to be read from or written to (Memory Address Register);
- (intermediate) results of arithmetic and logic operations (Accumulator);
- data or instructions transferred between the CPU and memory (Memory Data Register);
- the current instruction being decoded and executed (Instruction Register).

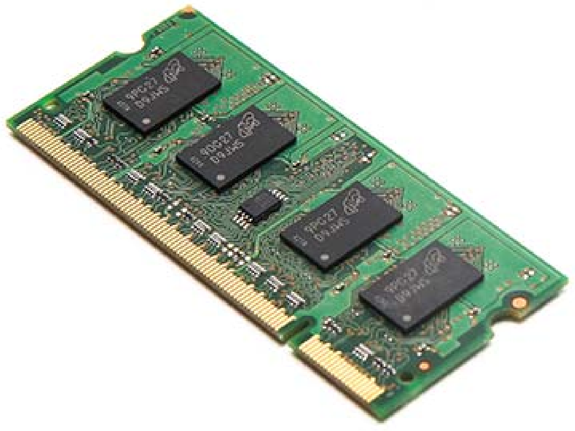


9.3 Buses and their function

Learning Objective

By the end of this section you will be able to:

- describe the role of the control, address and data buses in the fetch execute cycle.



The **address bus** is a unidirectional (1 way) bus, whilst the **data bus** is a bi-directional (2 way) bus.

When data is read from or written to memory:

- the processor sets up the address register with the address of the memory location to be accessed,
- the processor activates the read or write line on the control bus,

- and data is then transferred to or from the data register via the data bus.

Activity: Read and write operations

Q6:

Place each stage of the **read** operation in the right order:

	Read line is activated.
	Memory location is identified.
	Address of memory location to be read from is placed on Address register.
	Data is transferred to data register from memory location via data bus.

. . . .

Q7:

Place each stage of the **write** operation in the right order:

	Write line is activated.
	Address of memory location to be written to is placed on Address register.
	Data is transferred from data register to memory location via data bus.
	Memory location is identified.

. . . .

Each memory location has a unique binary address. Each line in the address bus can be on or off (1 or 0), so the total number of memory locations which can be addressed by the processor is determined by the number of lines in the address bus.

The total number of memory locations will be 2 to the power of the number of lines in the address bus:

- 16 lines = 2^16 possible memory locations;
- 32 lines = 2^32 possible memory locations.

The Address and Data buses are sets of lines which work together to perform the same sort of function, however the control bus is really just a convenient name given to a collection of control lines including:

- Read line;
- Write line;
- Clock line;
- Interrupt line;
- Non-Maskable Interrupt line;
- Reset line.

Quiz: Buses and their function

10 min

Q8: The purpose of the address bus is to: (Choose one option)

a) initiate a read from memory operation.
b) carry a memory address from which data can be read or to which data can be written.
c) store results of calculations.

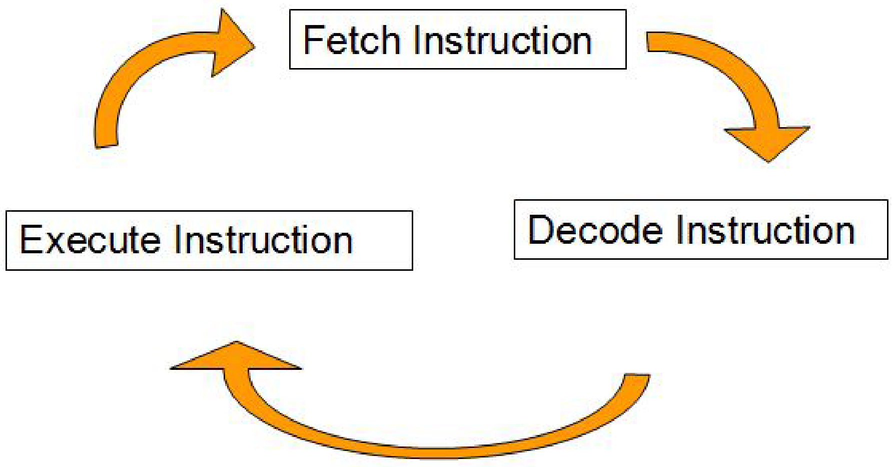
. . . .

Q9: The data bus is used:

a) to store the results of calculations.
b) to signal a read event.
c) transfer data between memory and processor.

. . . .

The fetch- execute cycle is the set of steps which the processor takes when reading and executing an instruction. Such an instruction may be to read a piece of data from a location in memory and load it into a register; increment or add data to a register; or to write a piece of data to a memory location from a register.



The detailed steps are:

1. Transfer the contents of the Program Counter to Memory Address Register
2. Increment the Program Counter
3. Activate Read line (thereby transfering instruction to the data register)
4. Transfer contents of data register to the instruction register ready for decoding
5. Decode Instruction
6. Execute Instruction

The execute step might involve carrying out a simple instruction to increment a register; an instruction to load data from a memory location and add it to the accumulator; or an instruction to place a new memory location into the program counter.

Activity: Fetch-execute cycle

Q10:

Place each stage of the Fetch-Execution cycle in the right order:

Note: of the following 9 stages, only 6 are correct.

	Transfer Program Counter to Memory Address Register
	Store instruction in accumulator
	Decode Instruction
	Increment the Program Counter
	Update control unit
	Execute Instruction
	Activate Read line
	Transfer data to memory address register
	Transfer instruction to Data Register and then to Control Unit

. . . .

Quiz: Program counter

15 min

Q11: Why does the processor need a program counter as well as an address register?

. . . .

9.4 Interfaces

As well as enabling communication between processor and memory, the data bus also enables the processor to communicate with peripheral devices. In order for this to work however, peripheral devices need an interface.

Interfaces are needed for a number of reasons:

- Because peripherals and the CPU often operate at different speeds, an interface will compensate for these differences when data needs to be transferred.
- Since the CPU can only perform one operation at a time, data needs to be stored in transit between a peripheral and the CPU while data is being transferred between the processor and a different peripheral. This data is often stored in memory called a buffer.
- Since the CPU and peripherals often deal with data in different formats and use different protocols. The interface will change the data into a format which the CPU or peripheral can understand.

The functions of an interface will include:

- **Temporary Data Storage in transit**: because of the differences in speed of operation, the processor is performing many other tasks as well as interfacing with the peripheral. For example there may be a keyboard buffer which stores key presses until the processor next checks it for input data.

- **Data Format Conversion**: data used by a peripheral may be at different voltage or use a different frequency The interface may need to convert from serial to parallel transmission, or convert an audio signal from analog to digital.
- **Transmitting Status Information**: a printer may need to inform the processor that it is low on ink or a hard disk drive may need to send data about the position of the read/write heads.
- **Transmitting control signals**: the processor may need to send control signals to a hard disk drive in order to read a file from a specific location on the disk.
- **Device Selection**: since several peripherals may be connected to the same bus the processor needs to be able to send a code to identify which interface a piece of data is intended for.

An interface will often be a combination of hardware and software. For instance a graphics card or a sound card will often require software drivers to be installed as well as the card itself.

9.5 Cache

Learning Objective

By the end of this section you will be able to:

- describe how cache memory affects processor performance.

Although accessing Random Access Memory (RAM) is much faster than retrieving data from hard disk, processor performance can be improved by using cache memory. The **cache** is a faster kind of memory which stores copies of the data from frequently used main memory locations. The processor will use various techniques to improve performance using the cache including reading ahead in a program to load the cache with instructions which are likely to be needed soon. This technique is not foolproof, however, as user menu choices or branch instructions make it difficult to predict exactly which code will be needed next.

When writing to main memory the processor uses the cache to deposit data and then resumes its operations immediately. The data is transferred to main memory by the cache controller circuitry.

When reading from memory the processor first checks whether the information is already available in the cache memory. If so then it can transfer this at high speed to the processor. If not, then this step is a waste of time; however, more often than not, the information being sought is indeed in cache and the benefits in terms of access time can be quite dramatic.

Activity: Cache

Example of processor, cache and main memory operations

Programme details:

```
Do some action
Set loop counter to 0
    Add 1 to the loop counter
    Do something using data item 1
    Do something using data item 2
    Do something using data item 3
If loop counter $<$ 3 return to step 3
END
```

Steps:

1. The processor is about to run a machine code program. The instructions and data for the program are stored in main memory. However, this system has cache memory between the processor and main memory.
2. The processor requires instruction 1. It checks to see if it is in the cache. It isn't, so a block of 4 instructions, including instruction 1, is moved from main memory to

the cache.

3. The processor can now fetch instruction 1 from the cache. This is much quicker than fetching from main memory.
4. The processor fetches instruction 2 from cache and executes it.
5. The processor fetches instruction 3 from cache and executes it.
6. The processor fetches instruction 4 from cache and executes it. This instruction requires data item 1, but this is not in the cache, so a block of data items is fetched from memory to the cache.
7. Data item 1 is moved from cache to processor, and is processed.
8. The processor needs instruction 5. It is not in the cache, so a block of 4 instructions is moved from main memory into the cache.
9. The processor fetches instruction 5 from cache and executes it. This instruction requires data item 2, so this is fetched from the cache.
10. The processor needs instruction 6. It is fetched from cache. It needs data item 3, which is also in cache and can be retrieved quickly.
11. The processor fetches instruction 7, which instructs the processor to go back to instruction 3 and continue from there.
12. The processor fetches instruction 3 from cache and executes it.
13. The processor fetches instruction 4 from cache and executes it. This instruction requires data item 1, which is also in cache, so this is fetched quickly.
14. The processor fetches instruction 5 from cache and executes it. This instruction requires data item 2, which is also in cache, so this is fetched quickly.
15. The processor needs instruction 6. It is fetched from cache. It needs data item 3, which is also in cache and can be retrieved quickly.
16. The processor fetches instruction 7, which instructs the processor to go back to instruction 3 and continue from there.
17. The processor fetches instruction 3 from cache and executes it.
18. The processor fetches instruction 4 from cache and executes it. This instruction requires data item 1, which is also in cache, so this is fetched quickly.
19. The processor fetches instruction 5 from cache and executes it. This instruction requires data item 2, which is also in cache, so this is fetched quickly.
20. The processor needs instruction 6. It is fetched from cache. It needs data item 3, which is also in cache and can be retrieved quickly.
21. The processor fetches instruction 7, but as the loop counter is now 3, the program continues to instruction 8.
22. The processor fetches instruction 8, which is an END instruction, so the program stops.

Note that this has taken less time than the same program without cache, because it was able to access instructions and data that were used repeatedly from cache more quickly than having to fetch them from main memory.

In a real processor, cache can hold much larger blocks of instructions and/or data, and typically loops are repeated many more times, so the time savings are even greater.

Please see online for related activity.

. . . .

9.6 Advances in processor design

Learning Objective

By the end of this section you will be able to:

- describe modern trends in computer architecture.

In 1965, Gordon Moore (one of the founders of Intel) observed that the number of transistors that could be fitted on a silicon chip was approximately doubling every 18-24 months. Since then this observation has held true and has become known as Moore's Law. Moore's law stated that the number of transistors on integrated circuits is doubling every two years. This means that computing performance per unit cost doubles roughly every two years. So far Moore's law has held true, but the strategies for improving processor performance are limited.

Speeding up the processor

Increasing the clock speed of the processor means that more instructions can be executed every second. The disadvantage of this approach is that the power consumption and heat generated by the processor increases, so the cooling required becomes more and more complex and itself consumes more power. Modern supercomputers use super-cooled circuitry to reduce the resistance in their processors and increase their speed as a result.

Increasing the size of instruction which can be executed in one operation

Increasing the width of the data bus means that larger and/or more instructions can be processed at a time. Improving performance this way requires that the processor decoding circuitry becomes more complex and as the size of the chip increases, more time is taken for signals to pass from one part to another. This means that transistors on the chip need to be smaller to benefit from the improved complexity. Densely packed transistors mean that processor power consumption increases and again heat dissipation becomes an issue.

Reducing the size of the transistors

Reducing the size of the transistors means that more transistors can be placed on a chip and they can be placed closer together. Several processors can also be placed on a single chip resulting in **multi-core** processors. The disadvantage of this approach is also one of heat dissipation, as well as the need for additional circuitry to coordinate the activities of the cores.

Increasing on-chip memory

Registers are memory locations which are physically part of the processor itself and they are fast to access, but can only hold a single piece of data, which is the maximum size of instruction which the processor can read in a single operation. This is known as the **word size** of the processor.

Placing cache memory on the processor itself allows an improvement in performance as this allows the processor to access frequently used instructions and data much faster than if it had to access RAM. This type of cache is known as **Level 1 Cache** (L1).

Parallel computing

The classical model of the computer is a machine which can only execute one instruction at a time. Although these instructions are simple, the fact that so many of them can be executed in a short time means that complex tasks and the illusion of simultaneous operation can be achieved. **Parallel computing** is where many processors work together to process instructions simultaneously. There are many situations where this approach can dramatically improve the time it takes to find a solution, however the software required to coordinate multiple processors gets very complicated very quickly as the number of parallel processor operations increases.

You may wish to read more about parallel computing here:

http://en.wikipedia.org/wiki/Parallel_computing

Unfortunately, computer software has not kept up with the advances in processor architecture. Niklaus Wirth created another law which is less optimistic than Moore's law which states that "software is getting slower more rapidly than hardware becomes faster." This applies particularly to parallel computing because the algorithms for solving the type of problem which are best suited to parallel computing solutions are very complex, particularly when several processors are manipulating the same variables.

Alternative processor technologies

Optical and quantum computing are two possible approaches to increasing the power of the processor. Optical computers would replace electrical signals with light which should theoretically need less power and operate at a higher speed. Quantum computing would replace the bit which can be either on or off with the Qubit which can have a value of both 1 and 0 at the same time. Both are very much at the experimental stage.

You may wish to read more about these technologies here:

http://en.wikipedia.org/wiki/Quantum_computer

http://en.wikipedia.org/wiki/Optical_computing

9.7 Emulators and virtual machines

Learning Objective

By the end of this section you will be able to:

- understand what an emulator is and how they are used;
- describe the concept of a virtual machine.

An **emulator** is software which replicates the function of one computer system on another. Emulators may be used to allow software designed for a computer system which is no longer manufactured to run on a modern machine. An example of this is when users wish to play games originally designed for arcade systems or games for older consoles.

Emulators are also used to test new processor design as the new processor can be created as an emulation and tested before the system is actually built. Emulators used in processor design also allow software to be designed and tested before the hardware it will run on exists.

A **virtual machine** can be an emulation of a complete computer system which can be used to run alternative operating systems, or several copies of a single operating system. One of the advantages of using virtual machines are improved security as several virtual machines can be running independently in one physical machine. Virtual machines, since they exist in RAM can be re-booted very quickly and can be migrated from one physical machine to another very easily.

A virtual machine can also be an emulation of a specific process such as an interpreter for a programming language. The advantage of using a virtual machine for this sort of task is that the virtual machine can be run under a variety of operating systems and means that the language is truly portable. Only one version of a piece of software is ever needed as it runs in the virtual machine on any operating system which can support the virtual machine. Java is frequently used in this way and the Java Virtual Machine (JVM) is available on a wide variety of platforms.

9.8 Mobile devices

Learning Objective

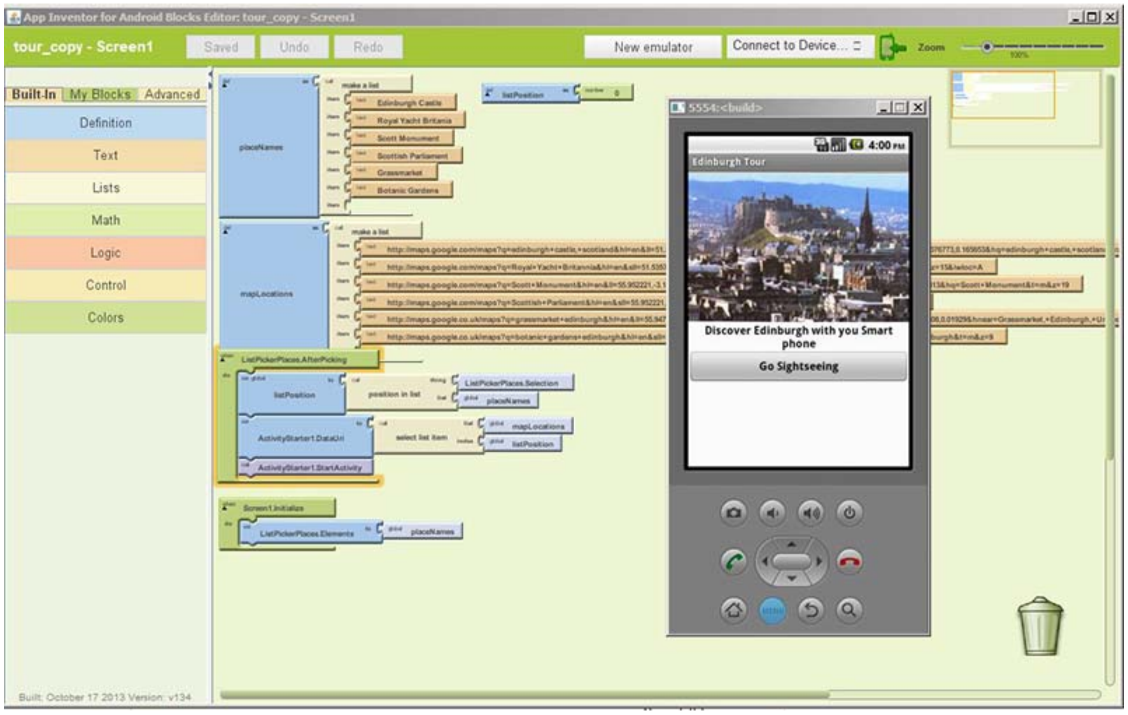
By the end of this section you will be able to:

- understand the influence that mobile devices have on the software development process.

With the improvement in wireless bandwidth and miniaturisation of components, mobile devices such as smart-phones and tablets have become increasingly popular.

Most software development for mobile devices is done on conventional desktop hardware, so the use of emulators has become a common method for testing mobile applications. Using an emulator means that applications can be tested and debugged on a variety of different mobile platforms before they need to be tested on the mobile devices themselves.

Applications for mobile devices need to use a touch-screen interface, present data on a small size display, and cope with limited bandwidth. Many applications will make use of the additional data which a mobile device provides such as geographical location (through the GPS unit), physical movement (through the built-in accelerometer), user facial characteristics and eye movement (through the screen-facing camera) and proximity of other users and mobile devices (through Bluetooth or the fact that their location can be tracked via the mobile network).



9.9 Learning points

Summary

You should now know:

- the CPU consists of several different parts: the Arithmetic and Logic Unit which performs calculations; the Control Unit which loads, decodes and executes instructions, and Registers which are small memory locations used by the processor.
- Buses are groups of lines which connect the CPU to the main memory.
- The total number of memory locations which can be addressed by the processor is determined by the number of lines in the address bus.
- The number of lines in the data bus determines the word size of the processor. The control bus is not really a bus in the strict sense, merely a collection of control lines.
- Interfaces are needed to allow the processor to communicate with peripherals.
- Processor performance can be improved by using faster cache memory.
- Speeding up the processor and reducing the size of transistors can improve performance at the expense of additional power consumption and heat dissipation.
- An emulator is software which duplicates the function of one computer system in another.
- A virtual machine is an emulation of a complete computer system which can be used to run alternative operating systems, or several copies of a single operating system.
- Mobile devices have features which require quite different types of software from conventional desktop systems.

9.10 End of topic test

End of topic test

10 min

Q12: The program counter stores the:

a) address of the next instruction to be fetched.
b) current instruction being decoded.
c) result of the last calculation.
d) address of the data to be transferred.

. . . .

Q13: The accumulator stores the:

a) address of the next instruction.
b) current instruction being decoded.
c) result of the last calculation.
d) address of the data to be transferred.

. . . .

Q14: Which one these is a unidirectional bus?

a) Address bus
b) Data bus
c) System bus
d) Control bus

. . . .

Q15: Which of these is **not** a line in the control bus?

a) Data line
b) Write like
c) Clock line
d) Interrupt line

. . . .

Q16: Place these in descending order of access speed:

1. RAM
2. Hard Disk
3. Cache
4. Registers

. . . .

Q17: The number of lines in the data bus determines:

a) the word size of the processor.
b) the maximum addressable memory.
c) the clock speed.
d) the number of instructions processed per cycle.

. . . .

Q18: The number of lines in the address bus determines:

a) the word size of the processor.
b) the maximum addressable memory.
c) the clock speed.
d) the number of instructions processed per cycle.

. . . .

Q19: A virtual machine exists:

a) on a hard disk.
b) in the cache.
c) in the registers.
d) in RAM.

. . . .

Topic 10

End of Unit 1 test

Contents

10.1 End of Unit 1 Test

End of Unit 1 Test

Q1: Which of these is a feature of a low level language?

1. The language is problem oriented.
2. The language uses key words similar to those used in human languages.
3. The language is processor specific.
4. There is a 1 to 1 relationship between language commands and machine instructions.

. . . .

Q2: A programming language which uses a knowledge base of facts and rules and matches them with a query to provide a solution to a problem is a description of :

a) an imperative language.
b) a declarative language.
c) a object-oriented language.
d) a domain-specific language.

. . . .

Q3: Which of the following statements best describes an object-oriented programming language?

a) An object-oriented language contains special routines for handling vector images.
b) The language is particularly suited to the control of physical devices.
c) The programmer defines both the data and the operations that can be carried out on it.
d) An object-oriented language has built-in routines for drawing objects.

. . . .

Q4: Which of the following statements about compilers are **true**?

1. Compilers are processor specific.
2. Compilers translate and execute source code into machine code line by line.
3. Compilers translate source code into machine code in a single operation.
4. A compiled program will run faster than an interpreted one.

. . . .

Q5: Which of the following statements about interpreters are **false**?

1. An interpreter will run a program until an error is found.
2. Interpreters translate source code to create an object code file.
3. Interpreters are not platform specific.
4. Interpreters translate and execute a program line by line.

. . . .

Q6: Which data structure would be best suited to store a set of test marks for a class of 20 pupils?

a) 20 STRING variables
b) A STRING array of 20
c) An INTEGER array of 20
d) A REAL array of 20

. . . .

Q7: What is type of variable should be used to identify the index of an array?

a) STRING
b) REAL
c) BOOLEAN
d) INTEGER

. . . .

Q8: What is the largest positive decimal number which can be stored using 8 bit two's complement notation?

a) 256
b) 257
c) 127
d) 128

. . . .

Q9: Convert -35 into 8-bit two's complement notation.

a) 1101 1101
b) 1101 1100
c) 0010 0011
d) 1101 1110

. . . .

Q10: How many colours can be represented if the bit depth of an image is 32 bits?

a) 65536
b) 16777216
c) 4294967296
d) 1099511627776

. . . .

Q11: Increasing the number of bits representing the mantissa in a floating point number:

a) increases the range of numbers which can be represented.
b) increases the accuracy of the numbers represented.
c) decreases the accuracy of the numbers represented.
d) decreases the range of numbers which can be represented.

. . . .

Q12: Increasing the number of bits representing the exponent in a floating point number:

a) increases the range of numbers which can be represented.
b) increases the accuracy of the numbers represented.
c) decreases the accuracy of the numbers represented.
d) decreases the range of numbers which can be represented.

. . . .

Q13: Decreasing the number of objects in a vector graphic image:

a) increases the file size.
b) decreases the file size.
c) increases the resolution of the image.
d) decreases the resolution of the image.

. . . .

Q14: How will the contents of an array of integers be stored in memory?

a) Using floating point notation.
b) Using two's complement notation.
c) As binary numbers.
d) As a binary file.

. . . .

Q15: What would be the output from this pseudocode example?

```
SET myVals TO [1, 12, 13, 35]
SEND myVals[0] + myVals[3] TO DISPLAY
```

a) 1
b) 14
c) 47
d) 36

. . . .

Q16: During the software development process, who is responsible for finding out the requirements of the client?

a) Programmers
b) System Analyst
c) Independent Test Group
d) Client

. . . .

Q17: Which of the following characteristics are true of Agile software development?

1. Responsiveness to changed circumstances.
2. Increased costs.
3. Reduced time spent on analysis.
4. Reduced development time.

. . . .

Q18: Which of these is **not** a design notation?

a) Structure diagram
b) Data flow diagram
c) Source code
d) Pseudocode

. . . .

Q19: Top Down Design is:

a) creating pseudocode from the structure diagram and data flow diagram.
b) breaking a large and complex problem into smaller, more manageable sub-problems.
c) writing source code.
d) creating a wireframe interface design.

. . . .

Q20: Which design notation would you use to design a user interface?

a) Wireframe
b) Structure diagram
c) Pseudocode
d) Data flow diagram

. . . .

Q21: The competitor's names and times in a race are stored in two arrays. Which data types will be used for the arrays?

a) STRING array and INTEGER array
b) REAL array and INTEGER array
c) STRING array and REAL array
d) STRING array and BOOLEAN array

. . . .

Q22: Race time data is used to calculate the number of qualifiers who have performed better than the average race time. Which of these algorithms will be used?

a) Input validation
b) Counting occurrences
c) Linear search
d) Finding the maximum

....

Q23: Which of these does **not** improve the readability of code?

a) Indentation
b) Meaningful variable names
c) Global variables
d) Modularity

....

Q24: In this procedure definition, name and times are:

```
PROCEDURE findWinner(name, times)
```

a) formal parameters
b) actual parameters
c) real parameters
d) reference parameters

....

Q25: In this function call the parameters 1 and 100 are:

```
Userinput = validNumber( 1, 100)
```

a) formal parameters
b) actual parameters
c) real parameters
d) reference parameters

....

Q26: Beta testing is done by:

a) the systems analyst.
b) the project manager.
c) the programmers.
d) the client.

....

Q27: A formal parameter whose value is changed by the procedure where it is defined is:

a) a reference parameter.
b) a value parameter.
c) an actual parameter.
d) a real parameter.

....

Q28: The number of lines in the data bus determines:

a) the word size of the processor.
b) the maximum addressable memory.
c) the maximum clock speed.
d) the minimum number of instructions processed per cycle.

. . . .

Q29: The number of lines in the address bus determines:

a) the maximum word size of the processor.
b) the maximum clock speed.
c) the number of instructions processed per cycle.
d) the maximum amount of addressable memory.

. . . .

Q30: Why does the linear search algorithm need a Boolean variable?

a) To count the number of items found
b) To terminate the loop when the item is found.
c) To store where the item is found.
d) To terminate the loop when the end of the array is reached.

. . . .

Q31: Which of these statements are **true**?

1. A function returns a value.
2. A function can be called with formal parameters.
3. A function can be user-defined.
4. A function is the same a procedure.

. . . .

Q32: Parallel arrays are:

a) two arrays containing linked data with the same index values.
b) two arrays with different index values containing different data.
c) arrays with identical information content.
d) arrays which are part of the same procedure.

. . . .

Q33: Which of these are syntax errors?

1. Missing semi colon
2. Division by zero
3. IF without END IF
4. Overflow error
5. Out of memory
6. WHILE without DO

. . . .

Q34: Registers are:

a) areas on a hard disk.
b) memory locations in RAM.
c) memory locations in the processor.
d) memory locations in the cache.

. . . .

Q35: Which standard algorithm is being used in this pseudocode segment?

```
SEND "Please enter Y or N" TO DISPLAY
RECEIVE userInput FROM (STRING) KEYBOARD
WHILE userInput ≠ ["Y"] AND userInput ≠ ["N"] DO
   SEND "Input must be Y or N" TO DISPLAY
   RECEIVE userInput FROM (STRING) KEYBOARD
END WHILE
```

a) Counting occurrences
b) Input validation
c) Finding the Maximum
d) Linear search

. . . .

Q36: Which standard algorithm is being used in this pseudocode segment?

```
RECEIVE item FROM (INTEGER) KEYBOARD
SET total TO 0
FOR EACH number FROM numbers DO
  IF number = item THEN
     SET total TO total + 1
  END IF
END FOREACH
SEND  total TO DISPLAY
```

a) Counting occurrences
b) Input validation
c) Finding the Maximum
d) Linear search

. . . .

The next three questions refers to the following information:

Q37:

Data for a cycle race stores the following information using these data structures:

Race times:	REAL array
Names:	STRING array
Nationalities:	STRING array
Qualifiers:	BOOLEAN array

Which standard algorithm would you use to find the name of the winner?

a) Counting occurrences
b) Finding the Minimum
c) Finding the Maximum
d) Linear search

. . . .

Q38: Which standard algorithm would you use to find out how many qualifiers there were?

a) Counting occurrences
b) Finding the Minimum
c) Finding the Maximum
d) Linear search

. . . .

Q39: Which standard algorithm would you use to find the time of a specific contestant?

a) Counting occurrences
b) Finding the Minimum
c) Finding the Maximum
d) Linear search

. . . .

Q40: Runners in a race have the following information stored about them: name, nationality, previous personal best time, and lane number. What is the best way of storing this data?

a) A set of 4 variables for each runner
b) 4 separate arrays
c) A single record structure
d) A single variable for each runner

. . . .

Glossary

Accessible

an interface is said to be accessible if its design does not impede those users with disabilities such as impaired vision or hearing

Actual parameter

the parameters which are used when a procedure or function is called.

Address bus

the address bus is used by the processor to identify a memory location for reading from or writing to.

Algorithm

a detailed sequence of steps which, when followed, will accomplish a task.

Alpha testing

testing of software within the development organisation which does not necessarily wait until the product is complete.

Arithmetic and Logic Unit

the part of a CPU where data is processed and manipulated.

Array

an array is a way of storing a range of values of the same type in a single indexed structure.

ASCII

American **S**tandard **C**ode for **I**nformation **I**nterchange. An 8 bit system for storing text.

Attributes

in object-oriented programming, this refers to the data associated with an object

Beta testing

testing of software outside the development organisation using clients or selected members of the public.

Bit

a single unit of binary data.

Bitmap

a representation of image data where each bit corresponds to an individual pixel on the screen.

Boolean

a value which can only be true or false.

Breakpoint

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

Class libraries

collections of classes that can be used in software development, as building blocks for further class definitions.

Compiler

a program that translates a complete high level language program (source code) into an independent machine code program.

Computational construct

a computational construct is a system of data representation and control structures used to solve problems using a computer through a programming language.

Concatenation

when two strings are joined together to make a new string.

Conditional loop

a control construct which allows a block of code to be repeated until a condition is met, often depending on user input.

Control Characters

special non-printing characters in a character set, used for special purposes, e.g. carriage return and end of file.

Control structures

a programming language structure which determines the flow of execution through the code.

Control Unit

includes timing/control logic and an instruction decoder. It sends signals to other parts of the computer to direct the fetch and execution of machine instructions.

Data bus

used to transfer data to and from the processor. The data bus can be common to devices and main memory allowing transfers to take place from and to peripherals or from and to main memory.

Data dictionary

a detailed list of data structures and relationships in a programming project.

Data flow diagram

a diagram that shows how data flows between the different modules in a piece of software.

Encapsulation

a characteristic of objects in object-oriented programming which means that objects are closed systems which cannot be altered from outside.

Execution error

an error which only manifests itself when a program is run rather than when its source code is translated.

Expert system

a system consisting of a database of facts and rules, an inference engine and a user interface used to help people make decisions.

Exponent

represents the range of a floating point number.

Fetch-Execute cycle

the repeated process of fetching instructions from main memory, decoding the instructions and executing them.

Fixed loop

a loop which repeats a set number of times.

Floating Point

a method of representing an approximation of a real number in a way that can support a wide range of values.

Formal parameter

the parameters in the definition of a procedure or function.

Function

a function is a sub-program which returns a value.

GIF

a lossless data compression format for graphics.

Global variable

a global variable is one which has scope throughout the entire program where it occurs.

Hexadecimal

a number system which uses base 16 instead of 10. Binary numbers can be very easily converted to hexadecimal because 1 hexadecimal digit is equivalent to 4 binary digits. This makes them much clearer for humans to read.

High level language

English-like programming language which has to be translated into machine code before computers can understand it.

House style

a set of rules for writing readable code imposed by a software development company on their programmers.

Increment

incrementing a variable means to increase its value by a fixed amount (usually 1).

Index

an integer value which identifies the position of an item in an ARRAY.

Inheritance

used in object-oriented programming, the sharing of characteristics between a class of objects and a newly created sub-class. This allows code re-use by extending an existing class.

Interpreter

a program that translates a high level program line by line, which it then tries to execute. No independent object code is produced.

Iterative

an iterative process is one that incorporates feedback and involves an element of repetition.

JPEG

a lossy compression format for graphics used for photographic images.

Keyword

a reserved word with a specific meaning in a high level language. Keywords will often be highlighted in colour or by some other means in the programming environment and cannot be used as variable names. Examples are IF, THEN, WHILE DO in **B**ASIC.

Level 1 Cache

the fastest form of cache memory, located on the processor itself.

Local variable

a local variable is one which only has scope within the sub-procedure it has been declared in.

Low level language

the machine code language which computers use.

Mantissa

a non-zero value used to represent the precision of a floating point number.

Messages

in object oriented programming, objects are closed systems and communicate by passing messages between them.

Method

a method in an object oriented language is a function that is defined inside a class.

MIDI

(Musical Instrument Digital Interface) is a file format that provides a standardized way of storing musical sequences.

Modular design

organising a large complex program into smaller parts coded as separate modules. Modular design is often the result of breaking a complex problem down into smaller sub-problems.

Modularity

a program is said to be modular if it is composed of sub-programs which can be tested independently.

Module library

a self contained pre-written and pre-tested blocks of code which can be re-used in other programs.

Multi-core processor

one which has several CPUs on a single chip.

Natural language

a language spoken by human populations

Object code

the machine code produced by a compiler, ready for execution by a processor.

Operations

in object-oriented programming this refers to the behaviour of an object is how it can manipulate data.

Parity Bit

an additional bit that is transmitted as part of a byte to make the total number of ones odd (odd parity) or even (even parity). Data transmitted can be checked by counting the number of ones in a character code to check that there have been no errors during transmission.

Portable

the ability of a program to run on different machine architectures with different operating systems.

Portable Network Graphics

a lossless data compression format for graphics created as an improvement on the GIF file format.

Problem oriented

used to describe a programming language where the focus is on the problem and how it is to be solved rather than on the hardware on which the program will run.

Procedural language

a programming language which follows a specific set of steps.

Procedure

a sub-program which can be called from within it's main program.

Program listing

the text version of the object code created by the programmer before translation. This should contain internal documentation, white space and indentation to make the code as readable as possible.

Prototyping

the process of creating partly working models of a system in order to test feasibility or to get customer feedback before the project is complete.

Pseudocode

an informal high-level description of how a computer program functions.

Readable

high level language code is said to be readable when it is made easy to understand using meaningful variable names and internal documentation.

Record

a structured data type which can contain values of different types in a single indexed structure.

Reference parameter

a reference or variable parameter in a sub program definition is one whose value is changed by that function or procedure.

Registers

internal memory locations which the processor uses to store data temporarily during the fetch execute cycle.

Reliable

a program is reliable if it runs well and is never brought to a halt by a design flaw.

Resolution

the total number of pixels in the width and height of an image determines the amount of detail represented in an image.

Robust

a program is robust if it can cope with problems that come from outside and are not of its own making. A robust program should not crash.

Scope

the scope of a variable is the range of sub-programs where it has a value.

Semantic error

a logical error in a program when the code is grammatically correct, but does not do what it is intended to do.

Simple data type

a programming language data type which is not made up of other data types.

Software specification

a legally binding document which describes exactly what a program will be able to do.

Source code

the high level language program code for an application before it has been translated into machine code.

Stack

a dynamic data structure much used by software applications and the computer for storing temporary data. Data can only be accessed via the top of the stack.

Structured data type

a programming language data type which is created using a collection of items. which are simple data types.

Structure diagram

a diagrammatic method of designing a solution to solve a software problem.

Structured listing

a structured listing is documented source code.

Syntax

the grammatical rules of a language.

Syntax error

an error in a program where the code is grammatically incorrect and cannot be translated into machine code.

Two's complement

a system for storing positive and negative integers where the most significant bit has a negative value.

Unicode

a symbol representation system for storing text which uses more bits than ASCII in order to represent character sets form around the world.

Value parameter

a value parameter in a function or procedure definition is one which does not change. It is a copy of the value being passed into the sub-program. The original outside the sub-program does not change.

Variable declaration

when a variable is defined for the first time giving it a name and a data type.

Vector Graphic

a method of storing a graphical object as a description which can be used to recreate it on an output device.

Version management software

is software which keeps track of versions of a program during development, allowing changes to be tracked and members of the programming team informed of their implications.

VHS

(Video Home System) was an analogue video recording system using magnetic tape video cassettes.

Watch

where the programmer identifies a variable whose value can be displayed in a separate window while a program is running in order to help with de-bugging.

Waterfall model

the traditional iterative model of software development using the 7 stages: Analysis, Design, Implementation, Testing, Documentation, Evaluation, and Maintenance.

Wireframe

a wireframe is a pictorial representation of the design of a website or piece of software.

Word size

the word size of a processor is the maximum length of instruction which it can read in any one operation.

Answers to questions and activities

1 Languages and environments

Quiz: Revision (page 3)

Q1: a) Computers only understand machine code

Q2: b) High level languages are often written to solve particular types of problem

Q3: Natural languages have more complex syntax rules, and can be more ambiguous.

Activity: Control structures (page 4)

Q4: a) A

Q5: d) A and B

Q6: g) A, B, and C

Activity: Programming languages (page 6)

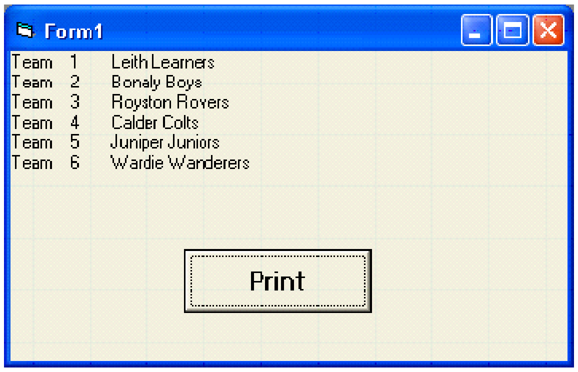
Q7:

Language	Description
BASIC	A high level procedural language designed to be easy to learn which became the popular language when computers became cheaper and popular in the 1970s.
COBOL	One of the earliest procedural programming languages designed for business and finance users.
Pascal	A general purpose procedural language developed by Niklaus Wirth and designed to encourage good programming practice.
Java	A highly portable object-oriented general purpose programming language designed to be platform independent and used for client-server web applications.
HTML	A domain specific page description language used to instruct web browsers how to display web pages.
FORTRAN	A general-purpose, procedural programming language designed for numeric and scientific computing.
Smalltalk	An early object-oriented programming language originally designed for educational use.
PROLOG	A declarative logic language designed for artificial intelligence applications.
PHP	a server-side scripting language designed for web development, often in combination with the MySQL database application.
Python	A general purpose procedural programming language designed by Guido van Rossum with an emphasis on readability of code.

Answers from page 8.

Q8:

The program output is:



```
Team  1    Leith Learners
Team  2    Bonaly Boys
Team  3    Royston Rovers
Team  4    Calder Colts
Team  5    Juniper Juniors
Team  6    Wardie Wanderers
```

Activity: PROLOG (page 10)

Q9: a) Yes

Q10: b) No

Q11: a) Yes

Q12: d) X = jim, X = justin

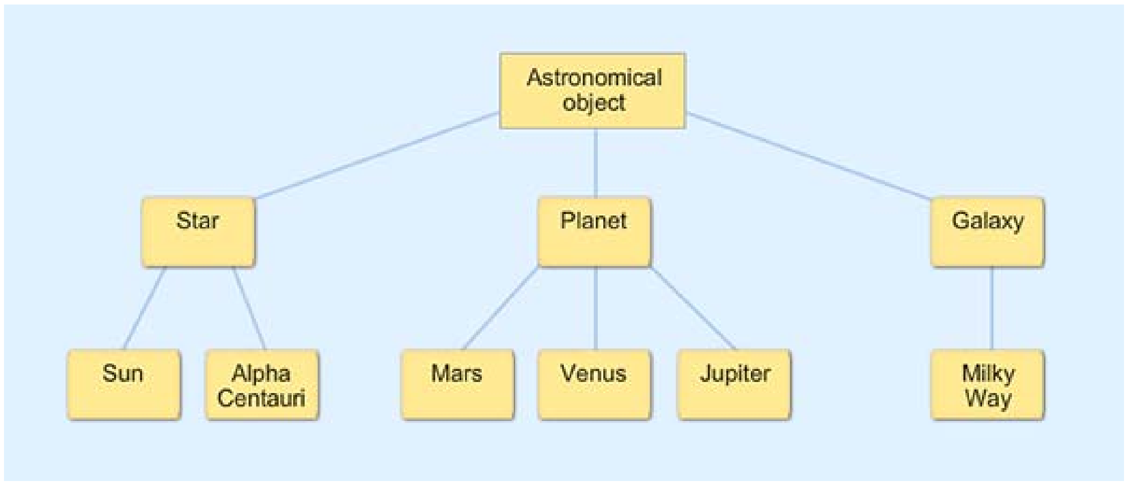
Activity: Object-oriented table (page 12)

Q13:

Object	**Attributes**	**Operations**
Button	Name, Size, Position, Colour	Click, Mouse-over
Window	Name, Size, Position, Focus, Border	Maximise, Minimise, Resize, Open, Close
Dialog box	Contents, Priority	Open, Close, OK, Cancel
Text box	Name, Size, Position	Create, Drag, Drop
Radio button	Name, Position, Initial state	Click
Pulldown menu	Name, Contents	Select, Mouse-over

Activity: Inheritance diagram (page 13)

Q14:



Quiz: Object-oriented languages (page 16)

Q15: Increasing complexity of programs produced problems in managing and maintaining them. GUI environments cannot be programmed by the constructs of conventional languages. Use of global variables meant that it became more and more difficult to keep errors from occurring where data was changed accidentally.

Q16: Encapsulation, inheritance and message passing

Q17: Inheritance means that new classes can be created by extending existing classes. The new classes will have all the properties of the existing classes. This promotes the reuse of code that can be used elsewhere in other programs.

Compiler and interpreter (page 22)

Q18:

Description	**1, 2, 3, or 4?**
Can test code while it is being written	3. Interpreter advantage
Creates fast executable machine code	1. Compiler advantage
Does not provide clear error messages	2. Compiler disadvantage
Must be re-translated if changes have to be made to the source code	2. Compiler disadvantage
Source code needs to be translated every time it is run	4. Interpreter disadvantage
Can partially translate source code	3. Interpreter advantage
Cannot translate code which contains errors	2. Compiler disadvantage
Provides helpful error messages	3. Interpreter advantage
No need for the translation software once the source coded has been converted to machine cod	1. Compiler advantage
Machine code cannot be converted back into source code	1.Compiler advantage
The translation software is needed along with the source code every time it is run	4. Interpreter disadvantage
Programs run more slowly because they are being translated while they are running	4. Interpreter disadvantage

Quiz: Programming environments (page 23)

Q19: c) A compiler creates an independent machine code program

Q20: a) Looping structures have to be interpreted each time they are executed

Q21: a) Computers can only understand machine code

Q22: d) A compiled program takes up less memory than an interpreted one

End of topic test (page 25)

Q23: b) Declarative

Q24: d) They can contain statements which are ambiguous

Q25: a) Procedural

Q26: b) Declarative

Q27: a) Knowledge base

Q28: c) SQL

Q29: c) Easy for humans to understand

Q30: a) Keyword highlighting b) Automatic indentation c) Search and replace d) Debugging tools, and e) Spell check

Q31: a) Different platforms require different compilers

Q32: d) A set of pre-tested classes which can be used in a program

Q33: b) Interpreters translate source code to create an object code file

Q34: c) The programmer defines both the data and the operations that can be carried out on it

Q35: b) The software can be easily re-translated for different platforms

2 Low level operations: Storing data

Quiz: Revision (page 31)

Q1: a) 0100 1001

Q2:

0100 0011	0110 1111	0110 0100	0110 0101
67	111	100	101
C	o	d	e

Q3: 200 x 640 pixels requires 128 000 bits of memory
= 128 000/8 = 16 000 Bytes
= 16 000/1024 = **15.6 KBytes**

Quiz: Using binary code to represent and store numbers (page 32)

Q4: Arithmetically things would have been simpler for humans if we had evolved with six digits on each hand rather than five, because this would have probably meant that we would have used a base 12 number system instead of base 10. 10 is not easily divisible by anything other than 2 and 5 making fractions like a third and two thirds an inconvenience. We are stuck with the decimal system however and are unlikely to be changing it in the near future due to the fact that we have 5 digits on each hand.

Example of binary to decimal conversion (page 33)

Q5: 118

Q6: 54

Q7: 986

Example of decimal to binary conversion (page 35)

Q8: 1000 0110

Q9: 1001 0100

Q10: 1 1000 1010

Using both methods to convert decimal to binary (page 37)

Q11: 1 1101

Q12: 1 0010

Q13: 100 1111

Q14: 1 0001 0001

Q15: 111 1111

Q16: 10 1110 0110

Q17: 1111 1011 0111

Q18: 10 0110 0001 1011

Quiz: Two's complement (page 39)

Q19: the number is negative because its most significant (leftmost) bit is a 1 and it is odd because there is a 1 at the end whose value is 1.

Calculating memory requirements (page 49)

Q20: 16000 Bytes

Q21: 60000 Bytes

Q22: 98304 Bytes

Answers from page 53.

Q23: $4x3x72x72pixels = 62208bits = 7776Bytes$

Q24: 65536

Q25: $300x300x3x2 = 540000bits540000/8 = 67500 = 65.9KByte$

Q26:

3 x 2 x 600 x 600 = 2160000 pixels

2160000 x 8 = 17280000 bits

17280000 / 8 = 2160000 Bytes

2160000 / 1024 = 2109 KB

2109 / 1024 = 2.06 MB

Q27:

2592 x 1944 = 5038848 pixels

5038848 x 24 = 120932352 bits

120932352 / 8 = 15116544 Bytes = 14.4 MB

Quiz: Video (page 60)

Q28: d) 237Mbps

Q29: a) 24fps, 600 x 400 pixels, 12 bits

End of topic test (page 63)

Q30: d) 1011 0111

Q31: c) 23.9 MB

Q32: b) 16777216

Q33: b) 8 bits

Q34: d) 65536

Q35: c) Accuracy increases, range decreases

Q36: a) increases the file size

3 Data types and structures

Revision (page 67)

Q1: b) a negative or positive number including zero with no decimal point

Q2: c) a negative or positive number including zero with a decimal point

Q3: a) a value which can be either true or false

Q4: c) a structured data type storing values of the same type

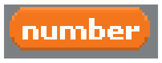
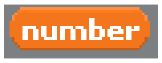
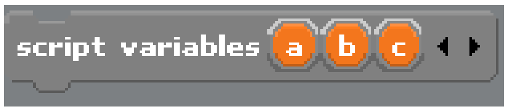
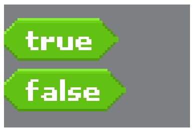
Activity: Simple data types (page 70)

Q5:

No.	Value	Simple data type
1	304	Integer
2	45.78	Real
3	@	Character
4	-4	Integer
5	5989.4	Real
6	-56.3	Real
7	!	Character
8	true	Boolean

Practical task: Simple data types (page 70)

Example answer

Data Type	Visual Basic 6	Snap! (formerly called BYOB)
Integer	Integer	
Real	Single, Double	
Character	String	
Boolean	Boolean	

Activity: Procedural language (page 73)

Q6:

Control Structures	Data Structures
Selection	Arrays
Iteration	Records

Practical task: Handling records (page 73)

Example answers

Visual Basic 6:

Module code:

```
Type game_character
  name As String
  weapon As String
  danger As Integer
End Type
```

Program code:

```
Dim enemy(3) As game_character
Private Sub fill_array()
enemy(0).name = "troll"
enemy(0).weapon = "axe"
enemy(0).danger = 3
enemy(1).name = "Dwarf"
enemy(1).weapon = "spell"
enemy(1).danger = 5
enemy(2).name = "wizard"
enemy(2).weapon = "staff"
enemy(2).danger = 9
enemy(3).name = "ghost"
enemy(3).weapon = "ectoplasm"
enemy(3).danger = 2
End Sub

Private Sub CmdDisplayRecord_Click()

fill_array
For Counter = 0 To 3
  Picture1.Print "Name: " ;enemy(Counter).name
  Picture1.Print "Weapon: ";enemy(Counter).weapon
  Picture1.Print "Danger level:";enemy(Counter).danger
Next Counter
End Sub
```

Python

```
#declare dictionary
enemy = {}

enemy[0] = ["troll","axe",3]
enemy[1] = ("dwarf","spell",5)
enemy[2] = ("wizard", "staff",9)
enemy[3] = ("ghost","ectoplasm",2)

for counter in enemy:
    print ("Name:", enemy[counter][0])
    print ("Weapon:", enemy[counter][1])
    print ("Danger level:", enemy[counter][2])
    print()
```

Java

In Java a record is just an object which has instance variables but not instance methods

```
Class enemy{
   String name;
   String weapon;
    int  danger;
  }
```

Activity: Data types 1 (page 75)

Q7:

No.	**Value**	**Data type**
1	678	INTEGER
2	Open Sesame!	STRING
3	0	CHARACTER
4	-5.7	REAL
5	4000	INTEGER
6	TD5 7EG	STRING
7	joe@companymail.com	STRING

Activity: Data types 2 (page 76)

Q8:

No.	**Value**	**Data type**
1	A UK telephone number	STRING *
2	The price of a pair of trainers	REAL
3	Whether a character in a game has found a weapon or not	BOOLEAN
4	The colour of a sprite	STRING
5	The counter in a loop	INTEGER
6	A URL	STRING
7	A key-press	CHARACTER

* Telephone numbers can start with a leading zero which would be ignored if they were stored as an integer.

Activity: Structured data types (page 76)

Q9:

No.	**Value**	**Data type**
1	A list of names	ARRAY of STRING
2	A set of test scores out of 50	ARRAY of INTEGER
3	The characters in a sentence	ARRAY of CHARACTER
4	The average temperatures during last month	ARRAY of REAL
5	The last five Google searches you made	ARRAY of STRING
6	Whether or not a class of pupils have passed an exam	ARRAY of BOOLEAN

Activity: Multiple data types (page 77)

Q10:

No.	**Records**	**Data types**
1	Name, address and Scottish Candidate Number (SCN) for a list of pupils.	STRING, STRING, STRING
2	Pupil ID, test score and pass/fail for a class	STRING, INTEGER, BOOLEAN
3	Weapon name, ammunition type and damage value in a First Person Shooter game	STRING, STRING, INTEGER

Quiz: Pseudocode (page 77)

Q11: e) Greg

Q12: b) Jim

Q13: g) 47

Q14: d) W

Practical task: Handling records (page 78)

Example answers

Visual Basic 6:

Module code:

```
Type game_character
  name As String
  weapon As String
  danger As Integer
End Type
```

Program code:

```
Dim enemy(3) As game_character
Private Sub fill_array()
enemy(0).name = "troll"
enemy(0).weapon = "axe"
enemy(0).danger = 3
enemy(1).name = "Dwarf"
enemy(1).weapon = "spell"
enemy(1).danger = 5
enemy(2).name = "wizard"
enemy(2).weapon = "staff"
enemy(2).danger = 9
enemy(3).name = "ghost"
enemy(3).weapon = "ectoplasm"
enemy(3).danger = 2
End Sub

Private Sub CmdDisplayRecord_Click()

fill_array
For Counter = 0 To 3
  Picture1.Print "Name: " ;enemy(Counter).name
  Picture1.Print " Weapon: ";enemy(Counter).weapon
  Picture1.Print " Danger level:";enemy(Counter).danger
```

```
Next Counter
End Sub
```

Python

```
#declare dictionary
enemy = {}

enemy[0] = ["troll","axe",3]
enemy[1] = ("dwarf","spell",5)
enemy[2] = ("wizard", "staff",9)
enemy[3] = ("ghost","ectoplasm",2)

for counter in enemy:
    print ("Name:", enemy[counter][0])
    print ("Weapon:", enemy[counter][1])
    print ("Danger level:", enemy[counter][2])
    print()
```

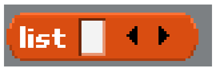
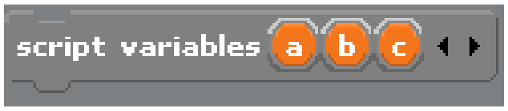
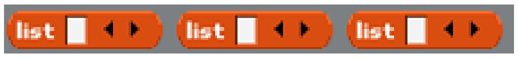
Java

In Java a record is just an object which has instance variables but not instance methods

```
Class enemy{
   String name;
   String weapon;
    int  danger;
  }
```

Practical task: Structured data types (page 79)

Example answer

Data Type	**Visual Basic 6**	**Snap! (formerly called BYOB)**	**Python**
ARRAY	Array		list / array
STRING	String		list / string
RECORD	Record		dictionary

Quiz: Identifying structured data types (page 79)

Q15: a) hydra

Q16: b) Run!

Practical task: Programming language syntax (page 80)

Example answers

Python

```
# open the file to read(r)
    inputfile = open("myfile.txt","r")
# print each item in turn to display
for line in inputfile:
    print line
# close the file
inputfile.close()

# open the file to write(w)
outputfile = open("myfile.txt","w")
    outputfile.write("This is an example of a text file")
# close the file
outputfile.close()
```

Visual Basic 6

```
Private Sub cmdReadFile_Click() Filename = "myfile.txt"
   Open Filename For Input As #1
   Textbox.Text = Input$(LOF(1), 1)
   Close #1
End Sub

Private Sub cmdWriteFile_Click()
   Filename =  "myfile.txt"
   Open Filename For Output As #1
   TextToWrite = "This is an example of a text file"
   Print #1, TextToWrite
   Close #1
End Sub
```

Java

```
import java.io.*;
public String readFile(String filename)
{
   String content = null;
   File file = new File(filename);
   try {
       FileReader reader = new FileReader(file);
       char[] chars = new char[(int) file.length()];
       reader.read(chars);
```

```
        content = new String(chars);
        reader.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return content;
}


import java.io.*;
public class WriteFile
{
  public static void main(String[] args)
  {
    try
      {
        FileWriter fw = new FileWriter(args[0]);
        fw.write("This is an example of a text file\n");
        fw.close();
      }
    catch (Exception e)
      { System.out.println(e);   }
  }
}
```

End of topic test (page 82)

Q17:

a) REAL
b) BOOLEAN
c) ARRAY of STRING
d) ARRAY of INTEGER
e) ARRAY of INTEGER
f) ARRAY of CHARACTER
g) ARRAY of STRING
h) ARRAY of BOOLEAN
i) RECORD
j) RECORD

4 Development methodologies

Revision (page 85)

Q1: a) customerNames

Q2: b) Design, code, test

Q3: Source code should be readable so that it can be understood by other programmers in case it needs changed or debugged.

Q4: c) The developer

Quiz: Analysis (page 88)

Q5: The analysis stage is important because unless the initial problem description is clearly stated and the software specification agreed upon, then subsequent stages will suffer from delays and difficulties due to the need to re analyse the task and rewrite the software specification.

Q6: The software specification describes what the software to be created must be able to do.

Activity: Testing (page 93)

Q7: 2) Extreme, 4) Normal, and 5) Exceptional

Q8:

Normal:	2, 4, 5
Extreme:	1, 7
Exceptional:	0, 8, @, 67

Quiz: Testing (page 95)

Q9:

b) Testing is done by the programmers responsible for the application

e) Testing may be done on parts of the application

Q10: a) The testing is performed by the clients

Activity: Evaluation terminology (page 98)

Q11:

Term	**Description**
Robust	Ability of a program to keep running even when external errors occur.
Reliable	The program always produces the expected result when given the expected input.
Portable	Whether or not the program can easily be used on a variety of hardware and/or operating systems.
Efficient	Whether the program wastes memory or processor time.
Maintainable	Has the program been designed to easily altered by another programmer.
Readable	Is the coding easy to understand, because it uses meaningful variable names and is well-structured.
Fit for purpose	Does the program fulfil all the requirements of the specification.

Activity: Maintenance (page 100)

Q12: 2) Corrective, 4) Perfective, and 5) Adaptive

Activity: Waterfall model (page 100)

Q13:

Analysis:	Looking at the problem and collecting information
Design:	Creating a structure diagram and pseudocode
Implementation:	Writing the source code
Testing:	Trying to find ways in which the program will fail
Documentation:	Creating a user guide and technical guide
Evaluation:	Checking to see how well the software meets its specification
Maintenance:	Fixing problems and adapting the software to new circumstances

End of topic test (page 104)

Q14: a) Perfective maintenance

Q15: d) Tutorial

Q16: a) Programmers

Q17: b) Exceptional data

Q18: c) Reliability

Q19: c) Editability

Q20: d) Reduced time spent on analysis

Q21: d) A high-level description of how a computer program functions.

Q22: b) Analysis, Design, Implementation, Testing, Documentation, Evaluation, Maintenance

Q23: a) The systems analyst

5 Software design notations

Revision (page 108)

Q1: a) Pseudocode

Q2: c) 10

Q3: c) 10

Quiz: Structure diagram (page 110)

Q4: ARRAY of STRING

Q5: ARRAY of INTEGER

Q6: Find highest score.

Quiz: Data flow diagrams (page 111)

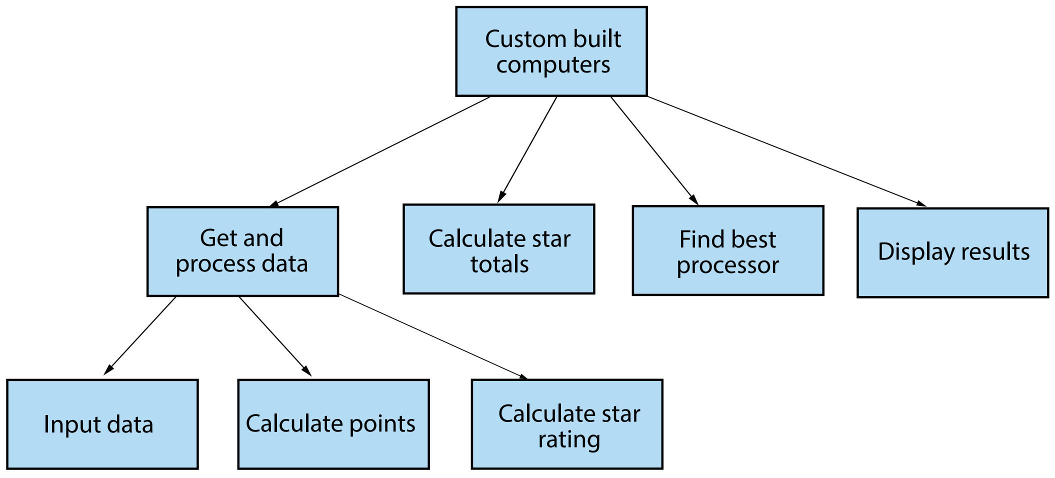
Q7: highest_score and total_failed

Q8: highest_score and total_failed

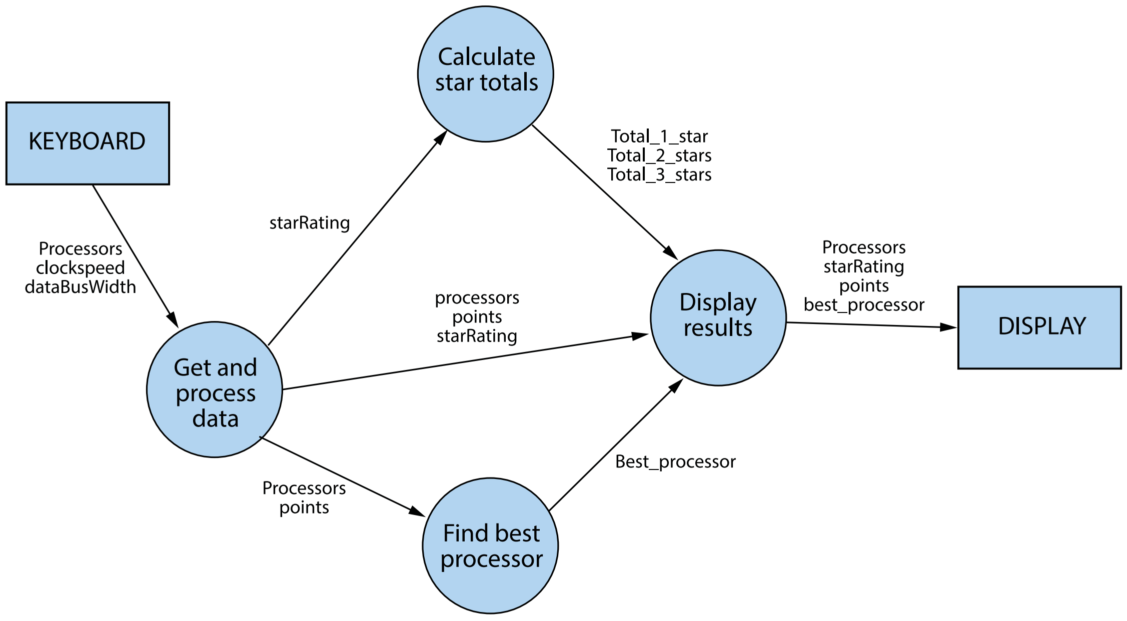
Practical: Creating a structure diagram and Data flow diagram (page 113)

Diagram solutions

Possible structure diagram solution:



Possible data flow diagram solution:


KEYBOARD
Calculate star totals
Processors clockspeed dataBusWidth
starRating
Total_1_star Total_2_stars Total_3_stars
Get and process data
processors points starRating
Display results
Processors starRating points best_processor
DISPLAY
Processors points
Best_processor
Find best processor

Note: There is more than one solution to this problem, this is only one possibility.

End of topic test (page 117)

Q9: d) Top-down design

Q10: b) The modules in a structure chart will become modules in the finished program.

Q11: c) Pseudocode

Q12: d) Data flow diagram

Q13: a) Wireframe

Q14: c) Pseudocode

Q15: a) Creating pseudocode from the structure diagram and data flow diagram.

6 Algorithm specification

Revision (page 120)

Q1: c) ARRAY of STRING

Q2: c) The position in an ARRAY

Q3: b) Pseudocode

Q4: a) A fixed loop

Practical task: Algorithms 1 (page 123)

Possible algorithm for solution

```
PROCEDURE InputValidation()

 RECEIVE userInput FROM (INTEGER) KEYBOARD

 WHILE userInput < 0 OR userInput > 100 DO

  SEND "Input must be between 1 and 100 inclusive" TO DISPLAY
  RECEIVE userInput FROM (INTEGER) KEYBOARD

 END WHILE

END PROCEDURE
```

Practical task: Algorithms 2 (page 123)

Possible solution

```
PROCEDURE InputValidation()

 RECEIVE userInput FROM (STRING) KEYBOARD

 WHILE userInput ≠["Y"] AND userInput ≠ ["N"] AND userInput ≠ ["y"]
 AND userInput ≠ ["n"] DO

   SEND "Input must be Y or y or N or n" TO DISPLAY
   RECEIVE userInput FROM (STRING) KEYBOARD

 END WHILE

END PROCEDURE
```

Practical task: Algorithms 3 (page 124)

Possible solution

```
PROCEDURE InputValidation()

 validInput = false
 validLength = 12

 REPEAT

 RECEIVE userInput FROM (STRING) KEYBOARD

  IF length(userInput) = validLength THEN
     validInput = true
  END IF

  FOR EACH letter FROM userInput DO
   IF letter < 0 OR letter > 9
     THEN validInput = false
   END IF
  END FOR EACH

  IF validInput = false THEN
    SEND "Input must contain exactly " & validLength & " digits" TO DISPLAY
  END IF

 UNTIL validInput = true

END PROCEDURE
```

Possible solution using mid$ function:

```
PROCEDURE InputValidation()

 SET validInput TO false
 SET validLength TO 12

 REPEAT

  RECEIVE userInput FROM (STRING) KEYBOARD

  IF length(userInput) = validLength THEN
   SET validInput TO true
  END IF

  FOR counter FROM 1 TO validlength DO
   IF mid$(userInput, counter, 1) < 0 OR mid$(userInput, counter, 1) > 9 THEN
```

```
            SET validInput TO false
          END IF
         END FOR

         IF validInput = false THEN
          SEND "Input must contain exactly " & validLength & " digits" TO DISPLAY
         END IF

        UNTIL validInput = true

       END PROCEDURE
```

Activity: Find the maximum value in an array (page 125)

Q5: 56

Q6: 74

Q7: 105

Q8: 74

Q9: 149

Practical task: Find winner (page 127)

Possible solution

```
PROCEDURE FindWinner()

 SET foundAt TO 0
 SET bestTime TO times[0]

 FOR index FROM 1 TO 9  DO
  IF bestTime > times[index] THEN
    SET bestTime TO times[index]
    SET foundAt TO index
  END IF
 END FOR
 SEND "The winner was "& names[foundAt] & "with a time of "& bestTime TO DISPLAY

END PROCEDURE
```

Activity: Counting Occurrences (page 128)

Q10: 3

Q11: 3

Q12: 2

Q13: 1

Q14: 1

Practical task: Occurrences (page 129)

Possible solution

```
PROCEDURE CountOccurrences()

 RECEIVE phrase FROM (STRING) KEYBOARD

 SET numberFound TO 0
 SET itemToFind TO "e"

 FOREACH letter FROM phrase DO
  IF letter = itemToFind THEN
    SET numberFound TO numberFound + 1
  END IF
 END FOREACH

 SEND "There were " & numberFound & "occurrences of the letter e
       in the phrase you typed" TO DISPLAY

END PROCEDURE
```

Possible solution using mid$ function:

```
PROCEDURE CountOccurrences()

 RECEIVE phrase FROM (STRING) KEYBOARD

 SET numberFound TO 0
 SET itemToFind TO "e"

 FOR counter FROM 1 TO length(phrase) DO
  IF mid$(phrase, counter, 1)  = itemToFind THEN
    SET numberFound TO numberFound + 1
  END IF
 END FOR

 SEND "There were " & numberFound & "occurrences of the letter e
       in the phrase you typed" TO DISPLAY

END PROCEDURE
```

Activity: Linear Search (page 130)

Q15: 2

Q16: 7

Q17: FALSE

Q18: TRUE

End of topic test (page 134)

Q19: b) Input validation

Q20: d) Finding the maximum

Q21: b) A conditional loop and a boolean variable

Q22: a) A fixed loop

Q23: a) Counting occurrences

7 Computational constructs

Revision (page 139)

Q1: a) Assignment

Q2: c) A user defined function

Q3: b) A simple conditional

Q4: c) A complex conditional

Q5: "Number OK"

Q6: "Invalid Entry"

Q7: a) A fixed loop

Practical task: User-defined functions (page 148)

Possible solutions:

```
FUNCTION NewRandom() RETURNS INTEGER

  randomNumber = rand(50) + 50

  RETURN randomNumber

END FUNCTION



STRING FUNCTION UserName() RETURNS STRING

 RECEIVE userInput FROM (STRING) KEYBOARD

 WHILE length(userInput) > 10
   SEND "Input must be Less than 10 Characters" TO DISPLAY
   RECEIVE userInput FROM (STRING) KEYBOARD
 END WHILE

 RETURN userInput

END FUNCTION
```

Practical task: Parameters 2 (page 150)

Possible solution

```
FUNCTION Double(value) RETURNS INTEGER
```

```
    value = value * 2
  RETURN value

END FUNCTION


FUNCTION UserName(foreName, yearOfBirth) RETURNS STRING

  newString = foreName & yearOfBirth
  RETURN newString

END FUNCTION
```

Practical task: Parameters 3 (page 152)

Possible solution

```
SET numbers TO [4,12,6,23,6,34,2,6,17,2]
SET names TO['Fred','Barney','Wilma','Betty','Jim','Sally','Joe',
'Zaphod','Greg','Jo']



PROCEDURE FindWinner(names, scores)

 SET maximumScore TO scores[0]
 SET winner TO names[0]
 FOR counter FROM 1 TO 9 DO
   IF maximumScore < scores[counter] THEN
       SET maximumScore TO scores[counter]
       SET winner TO names[counter]
   END IF
 END FOR
 SEND "The winner was " & winner TO DISPLAY

END PROCEDURE
```

Practical task: Sequential files (page 155)

Possible solution

```
SET names TO['Fred','Barney','Wilma','Betty','Jim','Sally',
'Joe','Zaphod','Greg','Jo']


PROCEDURE FindWinner(names, scores)
```

```
 GetData(scores)

 SET maximumScore TO scores[0]
 SET winner TO names[0]
 FOR counter FROM 1 TO 9 DO
   IF maximumScore < scores[counter] THEN
       SET maximumScore TO scores[counter]
       SET winner TO names[counter]
   END IF
 END FOR
 SEND "The winner was " & winner TO DISPLAY

END PROCEDURE

PROCEDURE GetData(scores)
  <open file "scores.txt">
  FOR  counter FROM 0 TO 9 DO
     RECEIVE scores[counter] FROM (INTEGER) "scores.txt"
  END FOR
  <close file "scores.txt">
END PROCEDURE
```

End of topic test (page 157)

Q8: c) Counting occurrences

Q9: d) String value

Q10: b) Actual parameters

Q11: a) Formal parameters

Q12: a) ORANGES A

Q13: c) 18

Q14: a) A reference parameter

Q15: b) A value parameter

8 Testing and documenting solutions

Revision (page 162)

Q1: a) 0, 1,10,50, 99,100, 101, X

Q2: b) scores

Q3:

```
Line 4  SEND ["Input must be between 1 and 100 "]  TO DISPLAY
Line 5  RECEIVE userInput FROM (INTEGER) KEYBOARD
```

Q4: Short statements embedded in the source code describing how a that part of the program functions in order to aid code readability.

Quiz: Debugging (page 165)

Q5:

- Division by zero
- Out of memory

Q6: The error is a logic (semantic) error. because the total variable is set to zero every time the loop repeats. It should be set to zero before the loop starts.

Practical task: Trace tables (page 167)

Possible solutions:

maximumValue	counter	numbers[counter]
3	1	15
15	2	4
15	3	7
15	4	8

Output: The largest value was 15.

End of topic test (page 170)

Q7:

- Missing semi colon
- IF without END IF
- WHILE without DO

Q8:

- Division by zero
- Overflow error

Q9:

miniimumValue	counter	numbers[counter]
17	1	15
15	2	4
4	3	7
4	4	8

Output: The smallest value was 4.

Q10: d) Extreme data

Q11: b) False

Q12: a) True

Q13:

value	no display	counter
1		0
1	1	1
2	2	2
4	4	3
7	7	4
11	11	5
16	16	6

Q14: c) Global variables

9 Computer architecture

Revision (page 175)

Q1: b) RAM is memory whose contents cannot be changed.

Q2: c) Transferring status information.

Q3: b) Memory used to store information being transferred by an interface.

Q4: a) The address bus.

Q5: b) The data bus.

Activity: Read and write operations (page 178)

Q6:

1. Address of memory location to be read from is placed on Address register.
2. Memory location is identified.
3. Read line is activated.
4. Data is transferred to data register from memory location via data bus.

Q7:

1. Address of memory location to be written to is placed on Address register.
2. Memory location is identified.
3. Write line is activated.
4. Data is transferred from data register to memory location via data bus.

Quiz: Buses and their function (page 179)

Q8: b) carry a memory address from which data can be read or to which data can be written.

Q9: c) transfer data between memory and processor.

Activity: Fetch-execute cycle (page 180)

Q10:

1. Transfer Program Counter to Memory Address Register
2. Increment the Program Counter
3. Activate Read line
4. Transfer instruction to Data Register and then to Control Unit
5. Decode Instruction
6. Execute Instruction

Quiz: Program counter (page 180)

Q11: An instruction in a program may be to load additional data from memory. The address of this data will need to be placed on the address bus using the address register, but the program counter is still needed to keep track of the address of the next instruction in the program.

End of topic test (page 189)

Q12: a) address of the next instruction to be fetched.

Q13: c) result of the last calculation.

Q14: a) Address bus

Q15: a) Data line

Q16:

1. Registers
2. Cache
3. RAM
4. Hard Disk

Q17: a) the word size of the processor.

Q18: b) the maximum addressable memory.

Q19: d) in RAM.

10 End of Unit 1 test

End of Unit 1 Test (page 192)

Q1:

3. The language is processor specific
4. There is a 1 to 1 relationship between language commands and machine instructions.

Q2: b) a declarative language.

Q3: c) The programmer defines both the data and the operations that can be carried out on it.

Q4: answer

1. Compilers are processor specific.
3. Compilers translate source code into machine code in a single operation.
4. A compiled program will run faster than an interpreted one.

Q5: answer

2. Interpreters translate source code to create an object code file.
3. Interpreters are not platform specific.

Q6: c) An INTEGER array of 20

Q7: d) INTEGER

Q8: c) 127

Q9: a) 1101 1101

Q10: c) 4294967296

Q11: b) increases the accuracy of the numbers represented.

Q12: a) increases the range of numbers which can be represented.

Q13: b) decreases the file size.

Q14: b) Using two's complement notation.

Q15: c) 47

Q16: b) System Analyst

Q17:

1. Responsiveness to changed circumstances.
4. Reduced development time.

Q18: c) Source code

Q19: b) breaking a large and complex problem into smaller, more manageable sub-problems.

Q20: a) Wireframe

Q21: c) STRING array and REAL array

Q22: b) Counting occurrences

Q23: c) Global variables

Q24: a) formal parameters

Q25: b) actual parameters

Q26: d) the client.

Q27: a) a reference parameter.

Q28: a) the word size of the processor.

Q29: d) the maximum amount of addressable memory.

Q30: b) To terminate the loop when the item is found.

Q31:

1. A function returns a value.
3. A function can be user-defined.

Q32: a) two arrays containing linked data with the same index values.

Q33:

1. Missing semi colon
3. IF without END IF
6. WHILE without DO

Q34: c) memory locations in the processor.

Q35: b) Input validation

Q36: a) Counting occurrences

Q37: b) Finding the Minimum

Q38: a) Counting occurrences

Q39: d) Linear search

Q40: c) A single record structure