COMPUTER SCIENCE

Specifications booklet
September 2007

This booklet supplements the Computer Science course and will be reviewed as required.

It is to be used in conjunction with the:
- course syllabus
- scope and sequence document.

180479_9.DOC

Introduction

The elaborations in this booklet are to be used in conjunction with the syllabus.

Content weightings

The following content weighting ranges will guide the development of the stage 2 and stage 3 examinations.

Components	15–20%
Design, development and management	15–20%
Tools	60–70%

Unit 2A—number systems and encoding

(Refer to syllabus content on p. 14)

Number systems

Students should recognise decimal, binary and hexadecimal numbers and explain their purpose and use in computing. Calculators can be used for all conversions, however students should be able to demonstrate the conversion from decimal to binary using one of the methods shown below. A calculator can be used to verify the answer.

Decimal to binary calculation

Division method

Divide Technique		
2	197	
	98	1
	49	0
	24	1
	12	0
	6	0
	3	0
	1	1
	0	1

Subtraction method

	197	Remainder	
128	197-128	69	1
64	=69-64	5	1
32			0
16			0
8			0
4	=5-4	1	1
2			0
1	=1-1	0	1

Alternative subtraction method

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
197-128	69-64				5-4		1-1
1	1	0	0	0	1	0	1

Encoding

Students should be able to explain the limitations of ASCII (American Standard Code for Information Interchange) as an 8 bit coding scheme and the benefits of Unicode as a 16 bit coding scheme. Students should be able to use an ASCII lookup table. Any required tables will be provided in an exam question.

Unit 2A/3A—Systems Development Life Cycle (SDLC)

(Refer to syllabus content on p. 14 and p. 18)

Stages of the systems development life cycle (SDLC)

- Preliminary analysis
  - problem definition
  - feasibility study
- Analysis
  - model of current system
  - requirements of new system
- Design
  - logical and physical design
- Development
  - hardware and software acquisition
  - construction and testing
- Implementation
  - change-over methods: direct cut, phased, pilot, parallel
- Evaluation and maintenance
  - performance evaluation
  - fault finding and correction

Unit 2B—TCP/IP model

(Refer to syllabus content on p. 16)

Unit 2B provides an overview of the 4 layer TCP/IP model, also known as the DoD (Department of Defense) model, to provide an understanding of the level at which each network device operates.

Unit 3B—network interconnectivity (OSI model, TCP/IP model)

(Refer to syllabus content on p. 20)

Unit 3B builds on the overview of the 4 layer TCP/IP model, also known as the DoD model, and provides an overview of the 7 layer OSI model. Comparisons will be made between the 4 layer TCP/IP model and the 7 layer OSI model.

Tools introduction

The following pages elaborate the use of tools at the different stages and provide standard representations for the design tools.

Units 1A, 2A, 3A programming

Overview

Units 1A, 2A and 3A progressively develop knowledge about computer languages and skills in designing, creating, modifying, testing, evaluating and documenting **programs**.

Unit 1A (Refer to syllabus content on p. 10)
Program components require students to be able to identify inputs, processing and outputs. IPO charts will be used to organise this and interface designs will be planned.

When using a **simple programming language**, students will not be required to write code. They will create programs by recording macros or using interactive drag and drop languages. They will then identify the components that have been created and inspect and edit the code.

Suggested programming languages for Unit 1A are Word macros–VBA, Scratch, Alice

Unit 2A (Refer to syllabus content on p. 14)
Program components and constructs focuses on **simple algorithms** using sequence, selection and repetition. These algorithms will be developed using flow charts (a graphical method) and pseudocode (structured English).

Students will write, compile, test and debug code using procedural type programming. It is recommended that the language chosen includes a visual interface.

Unit 3A (Refer to syllabus content on p. 18)
Programing constructs and **structured programming** extend to more complex algorithms **using modularisation and parameter passing,** and **one-dimensional arrays**. These algorithms will be developed using pseudocode. Students may continue to use flow charts, but external exams questions will represent algorithms in pseudocode.

Students will write, compile, test and debug code using procedural type programming. It is recommended that the language chosen includes a visual interface.

Unit 1A—simple programming language

(Refer to syllabus content on p. 10)

Simple programming languages may involve:

1. creating programs through the use of either:
   - recording macros in application programs such as Word or Excel **OR**
   - interactive drag and drop programs such as Scratch.
2. identifying components of the program by either:
   - inspecting code syntax for macros and making changes such as the size or font **OR**
   - recognising and using drag and drop components.

Program components may include:

Objects	Items that a user can manipulate as a single unit to perform a task
Properties	Attributes of an object
Methods	Behaviours that a particular object can have
Controls	Sequence, selection, repetition

Program design using Input, Processing, Output (IPO) charts

There are a number of ways that IPO charts can be set out, but the simple format below will be adopted that requires the student to identify any inputs, the processing that will take place and the outputs required.

Input	Processing	Output
• number of hours worked • hourly rate • tax rate	• calculate gross pay • calculate tax payable • calculate nett pay	• gross pay • tax payable • nett pay

Word Macros

Tutorial on creating and editing Word Macros–http://www.officeletter.com/favtips/wordmacros.html

Scratch programming language

Freeware, downloadable from http://scratch.mit.edu/

Getting Started Guide, Project Ideas and online help available.

Simple to use drag and drop programming that introduces programming constructs and components.

Tutorials and information on using Scratch available from kidsprogramming.pbwiki.com

Alice programming language

Freeware, downloadable from http://www.alice.org/

Demonstration videos and tutorials available.

Simple to use drag and drop programming that introduces programming constructs and components.

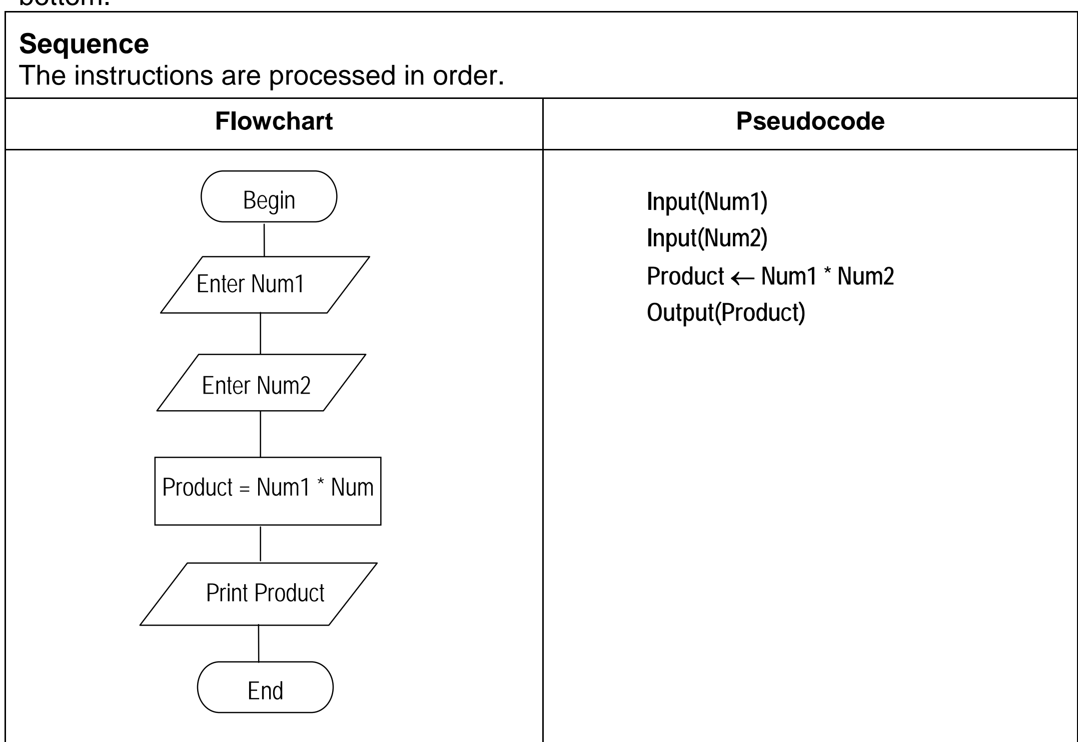
Unit 2A—program components and simple algorithms

(Refer to syllabus content on p. 14)

Flow chart symbols and pseudocode

Symbol	Meaning
	Terminal: begin and end
	Input or output
	Process: the description of an action or process
	Decision: one line comes in at the top and two lines leave it
	Sub-program or module: a portion of code that performs a particular task

Flow lines do not need an arrow if the direction of flow is from top to bottom or from left to right. In the sequence example below there are no arrows on the flow lines as the flow of control is from top to bottom.



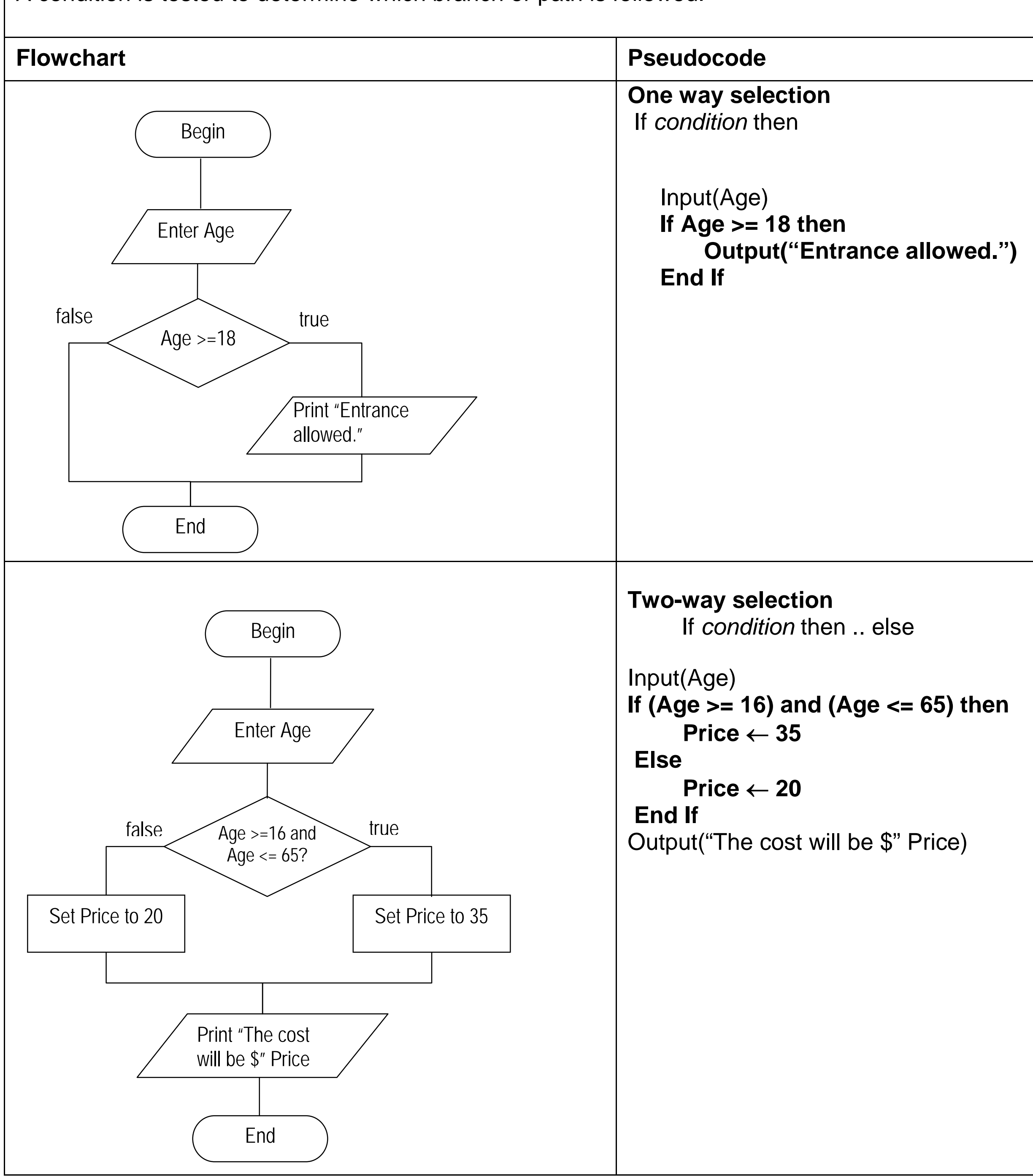
Sequence
The instructions are processed in order.

Flowchart	Pseudocode
Begin Enter Num1 Enter Num2 Product = Num1 * Num Print Product End	Input(Num1) Input(Num2) Product ← Num1 * Num2 Output(Product)

Read and **Write** can be used in place of **Input** and **Output**
Example:

```
Read(Num1)
Read(Num2)
Product ← Num1 * Num2
```

Write(Product)



Selection

A condition is tested to determine which branch or path is followed.

Flowchart	Pseudocode

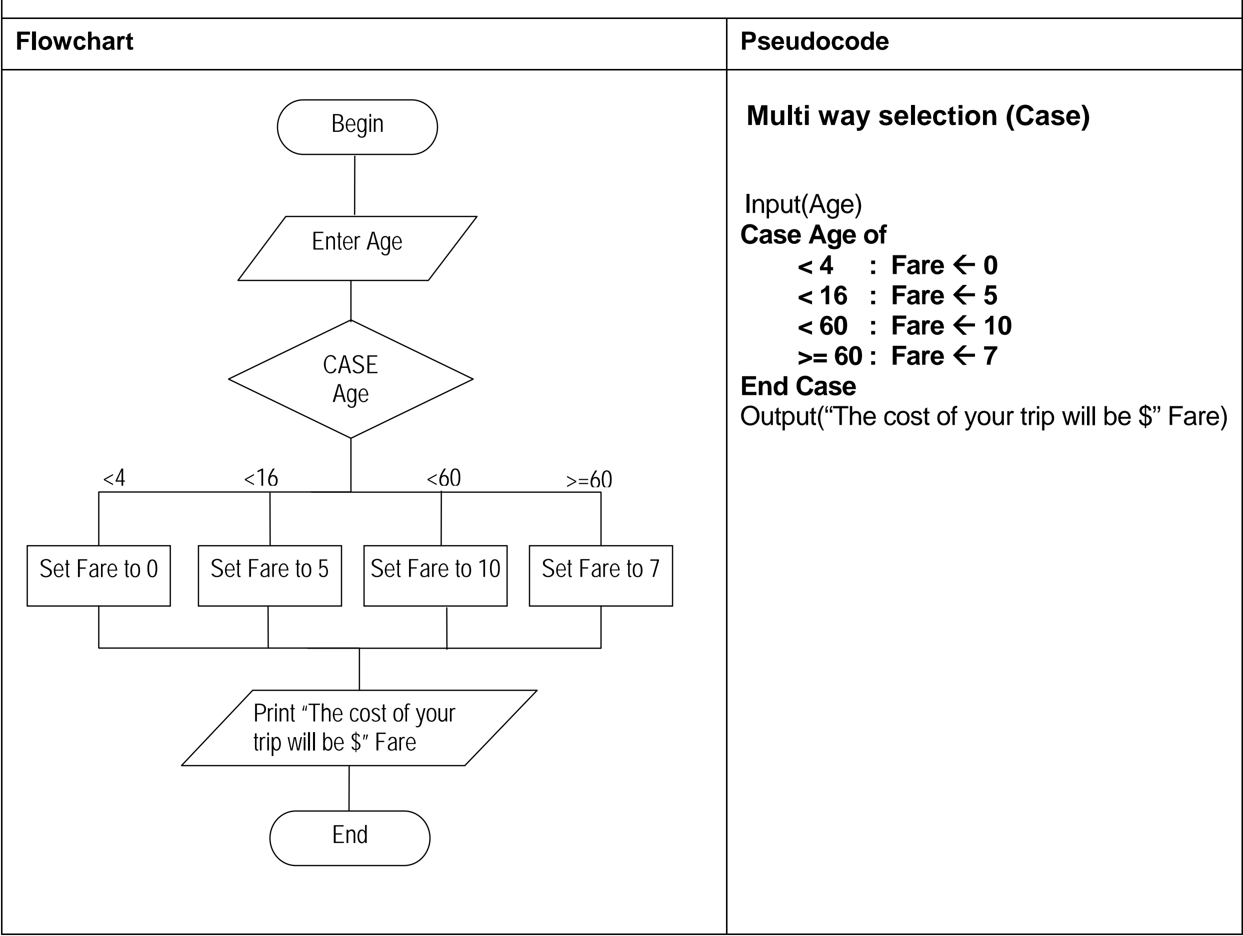
	One way selection
	If *condition* then
	
	Input(Age)
	If Age >= 18 then
	Output("Entrance allowed.")
	End If

	Two-way selection
	If *condition* then .. else
	
	Input(Age)
	If (Age >= 16) and (Age <= 65) then
	Price ← 35
	Else
	Price ← 20
	End If
	Output("The cost will be $" Price)

Selection (continued)

A condition is tested to determine which branch or path is followed.

Flowchart



Pseudocode

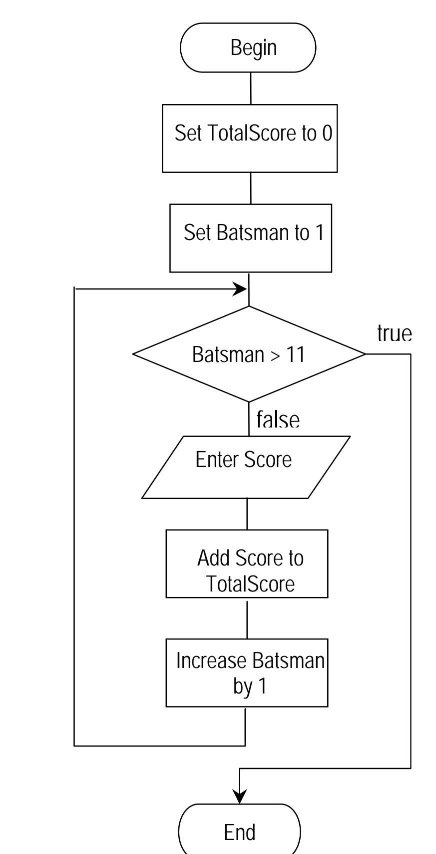
Multi way selection (Case)

```
Input(Age)
Case Age of
      < 4    :  Fare ← 0
      < 16   :  Fare ← 5
      < 60   :  Fare ← 10
      >= 60 :  Fare ← 7
End Case
Output("The cost of your trip will be $" Fare)
```

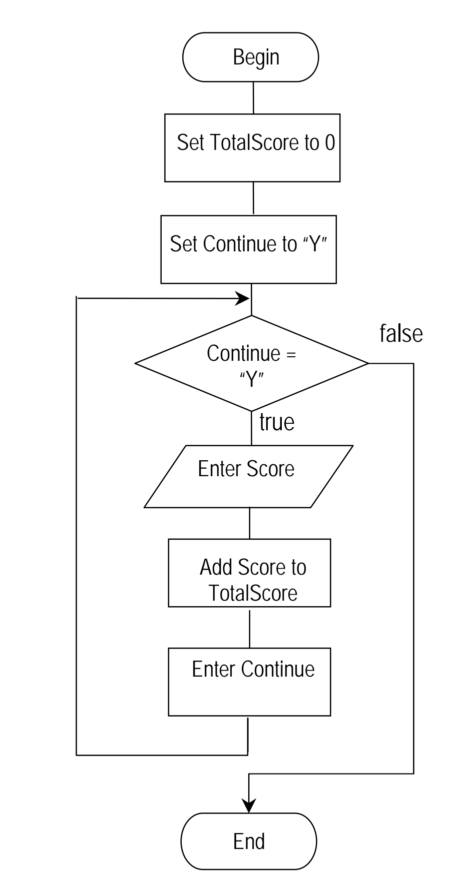
Repetition also commonly called iteration or looping

Repeating an action or series of actions a number of times.

Flow chart	Pseudocode
FOR: Fixed or counted loop This loops or repeats a counted or fixed number of times. The number of repetitions is known when the loop begins.	

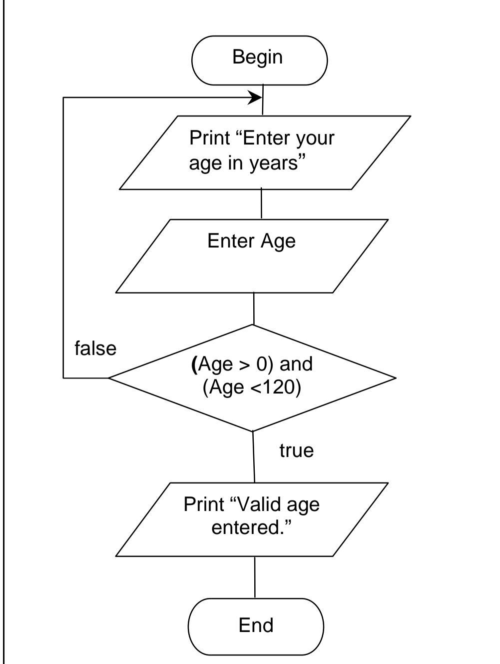

 	
	TotalScore ← 0
	For Batsman ← 1 to 11 do
	Input(Score)
	TotalScore ← TotalScore + Score
	EndFor
	
	OR
	
	TotalScore ← 0
	For Batsman ← 1 to 11
	Input(Score)
	TotalScore ← TotalScore + Score
	Next Batsman

While: test first or pre-test loop
This loops a variable number of times.
The number of repetitions is not known when the loop begins. This is tested before the loop is entered—test first—it is possible that the loop is executed zero times.



TotalScore ← 0
Continue ← "Y"
While Continue = "Y"
    Input(Score)
    TotalScore ← TotalScore + Score
    Input(Continue)
EndWhile

Repeat ... Until: test last of post-test loop This loops a variable number of times. The number of repetitions is not known when the loop begins. This is tested at the end of the loop—test last—and therefore must be executed at least once.	

	**Repeat**
	    Output(“Enter your age in years.”)
	    Input(Age)
	Until (Age > 0) and (Age <120)
	Output(“Valid age entered.”)

Modules

In Unit 2A students design and write code segments that may be modularised, but they will not be required to pass parameters between modules

Unit 2A—trace tables for desk checking testing and debugging

(Refer to syllabus content on p. 14)

The correctness of an algorithm should be checked before coding begins. Trace tables provide a formal method for tracing the logic of an algorithm.

A set of data values (test data) is chosen to test all paths within the algorithm.

All variables, constants and formal parameter values need to be represented.

A desk check of the following pseudocode using the data values [2,3,6,5,7,999].

```
Module DisplayLargestNumber
1  Largest ← 0
2  Input(Number)
3  Repeat
4     If Number > Largest then
5        Largest ← Number
6     End If
7     Input(Number)
8  Until (Number = 999)
9  Output("The largest number is ", Largest)
```

Expanded method

Line	*Largest*	*Number*	Repeat	If	output
1	0				
2		2			
4				TRUE	
5	2				
7		3			
8			FALSE		
4				TRUE	
5	3				
7		6			
8			FALSE		
4				TRUE	
5	6				
7		5			
8			FALSE		
4				FALSE	
7		7			
8			FALSE		
4				TRUE	
5	7				
7		999			
8			TRUE		
9					The largest number is 7

Lines 3 to 8 are a Repeat—Until loop, line 8 is the condition (test LAST) which will repeat the loop until this is TRUE.

Lines 4 to 6 are an If statement. If the condition in line 4 is TRUE, then line 5 is processed, otherwise line 5 is skipped.

Unit 3A—program constructs and structured programming

(Refer to syllabus content on p. 18)

Pseudocode concepts for program design from Unit 2A are also required for Unit 3A.

Structured programming using modularisation and parameter passing

In Unit 3A students design and write modularised code segments that pass parameters between the modules.

Module example

```
Module CalcPay (Rate, Hours, Pay)
    Pay ← Rate * Hours
End CalcPay
```

Calling the module

```
Module Main
    Rate ← 25.5
    Input(Hours)
    Call CalcPay (Rate, Hours, Pay)
    GrossPay ← Bonus + Pay
    Output(GrossPay)
End Main
```

The Rate and Hours parameters are sent to the module CalcPay and the calculated Pay is returned through the Pay parameter.

Rate and Hours would be pass by value parameters that receive a value, but do not return a changed value.

Pay would be a pass by reference (variable) parameter as it returns a value to the calling module.

Function example

```
Function Pay (Rate, Hours)
    Pay ← Rate * Hours
End Function
```

Using the function in a calculation and as an output

```
Module Main
    Rate ← 25.5
    Input(Hours)
    Output(Pay(Rate, Hours)
    GrossPay ← Bonus + Pay(Rate, Hours)
    Output(GrossPay)
End Main
```

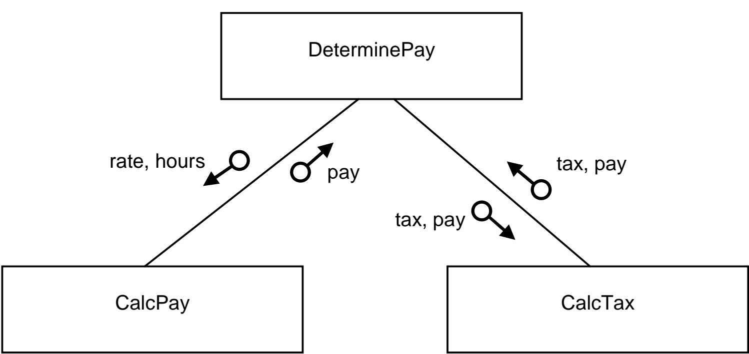
A function is a special type of module that:

- receives data through its parameters and returns a single value through the function name. In the following example values are received through rate and hours and the calculated result is returned through the function name Pay
- has no input or output statements
- is used in a calculation, assignment statement or output statement.

Unit 3A—Structured programming using structure charts

(Refer to syllabus content on p. 18)

Unit 3A structure charts represent modules graphically. The data parameters passed between modules are included.



Rate and hours are **value parameters** that send a value to the module, but do not return any value.

Two-way parameters *pay* and *tax* would be var or **variable parameters** in Pascal/Delphi or by Ref parameters in Visual Basic) show that any changes made to the values are passed back to the calling module.

Fundamentals of data structures—arrays (one-dimensional) and records

Array pseudocode examples

Initialising an array with zeros

```
For Student ← 1 to 25
    MarksList[Student] ← 0
End for
```

Reading data into an array

```
For Student ← 1 to 25
    Input(MarksList[Student])
End for
```

Displaying all the data from an array

```
For Student ← 1 to 25
    Output(MarksList[Student])
End for
```

Record pseudocode examples

Record structure

```
StudentData
    Firsname
    Surname
    DateOfBirth
    Phone
```

Reading data into the student record

```
Input(StudentData.Firstname)
Input(StudentData.Surname)
Input(StudentData.DateOfBirth)
Input(StudentData.Phone)
```

Displaying data from the student record

```
Output(StudentData.Firstname)
Output (StudentData.Surname)
Output (StudentData.DateOfBirth)
Output (StudentData.Phone)
```

Unit 3A—Testing and debugging

(Refer to syllabus content on p. 18)

The correctness of an algorithm should be checked before coding begins. Trace tables provide a formal method for tracing the logic of an algorithm. A set of data values (test data) is chosen to test all paths within the algorithm.

- All variables, constants and formal parameter values need to be represented.
- Any data structure (such as an array or record) should be represented separately to the table of simple data types, so that changing values can be represented more easily.

Condensed method

In Unit 3A this condensed method is more compact for longer more complex algorithms or where there is more test data, but it can be more difficult for students to use.

A desk check of the following pseudocode using the data values [2,3,6,5,7,-10,20,3,999].

```
Module DisplayLargestNumber
Largest ← 0
Input(Number)
Repeat
     If Number > Largest then
          Largest ← Number
     End if
     Input(Number)
     Output("Largest so far is ", Largest)
Until (Number = 999)
Output("The largest number of all is ", Largest)
```

Largest	***Number***	***Number > Largest***	***Number* = 999**	**output**
0	2	2 > 0 is T		
2	3	3 > 2 is T	F	Largest so far is 2
3	6	6 > 3 is T	F	Largest so far is 3
6	5	5 > 6 is F	F	Largest so far is 6
6	7	7 > 6 is T	F	Largest so far is 6
7	-10	-10 > 7 is F	F	Largest so far is 7
7	20	20 > 7 is T	F	Largest so far is 7
20	3	3 > 20 is F	F	Largest so far is 20
20	999		T	Largest so far is 20
				The largest number of all is 20

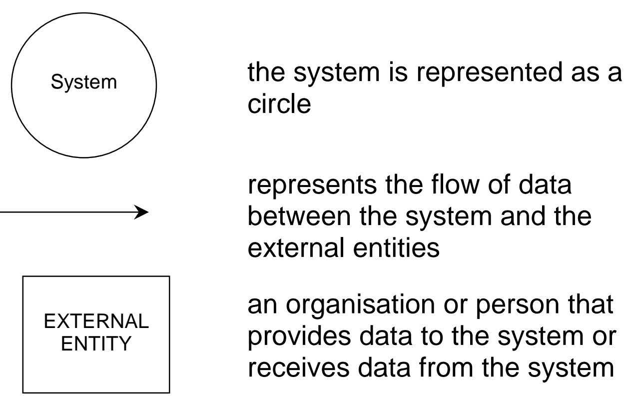
Unit 2A—Data flow diagrams

(Refer to syllabus content on p. 15)

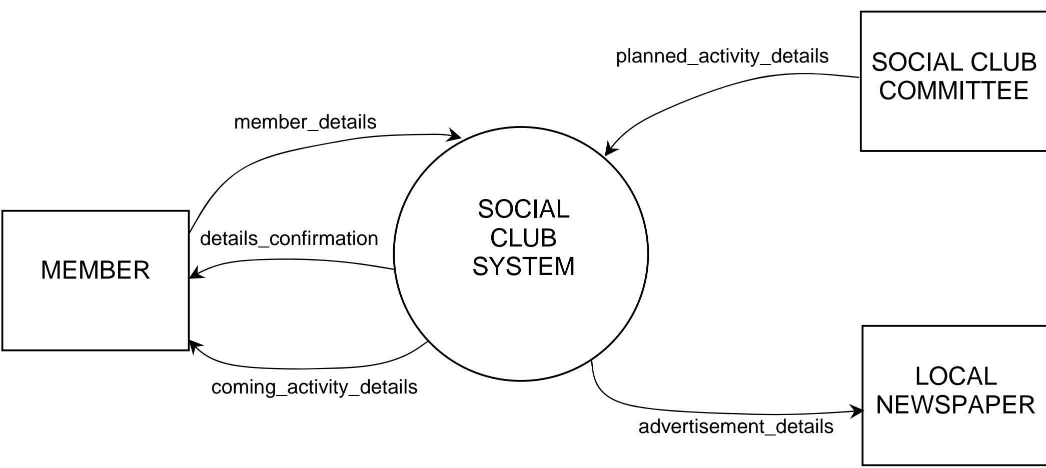
These conventions are based on the De Marko/Yourdan symbols.

Context diagram

The context diagram, also called level zero, is the top level of a set of hierarchically related diagrams that form a set that decomposes a system into successively finer detail with each move down the diagram set. This diagram represents the system being modelled as a single circle interacting with external entities. The emphasis of this diagram is to identify the boundary of the system. The name inside the single circle representing the system should describe the system being modelled. The symbols used are:



Context diagram for social club system



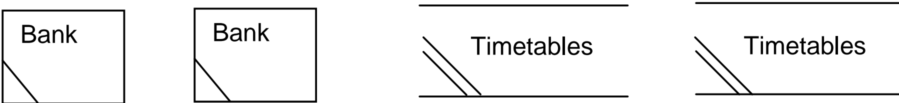
The circle is a representation of the system boundary. The system boundary defines what is inside and outside the system.

Deciding on which side objects lie is an important consideration. Is a particular object part of the system being considered, and hence invisible inside the circle, or is it really outside the system's considerations and therefore an external item supplying data, or taking information from the system?

Notice that data stores or files must never appear in a context diagram. They are part of the system and are therefore inside the circle.

<table>
<tr><td>LIBRARY DATABASE</td><td>External entities: (sources or sinks): These are any organisation or person that provides data to the system or receives data from the system.
They exist outside of the system.
An external entity can be both a source and a sink.
They should be named in the singular as a person, place or thing.</td></tr>
<tr><td>1
Update
pay details</td><td>Processes: These are actions taking place that transform inputs into outputs.
They must always have at least one inflow and one outflow.
They should be named with an active verb associated with a noun or very short phrases of that type, reflecting what transformation the process is making to the data passing through it.
The numbering of a process does not indicate timing or sequence.
The data flowing out of a process should differ from that going in. (e.g. payment_cheque_details goes in and cancelled_cheque_details comes out of an enter cheque transaction process.)</td></tr>
<tr><td>student timetables</td><td>Data stores: (files, repositories of data or temporary data stores) These store data used within a system.
They cannot transform data, and must usually contain at least one inflow and one outflow.
A data store’s identifier should be a noun reflecting the data it contains and not its physical nature. (e.g. customer details NOT sorted magnetic tape file).</td></tr>
<tr><td></td><td>Data flows: These vectors indicate the data being transferred (not physical objects), e.g. invoice_details not invoice.
They should connect at each end directly to their source and destination with only one arrowhead.</td></tr>
</table>

Sometimes in order to simplify a diagram, an entity or data store requires duplication. Each of the duplicated objects should contain a diagonal line(s) in the bottom corner as shown below:

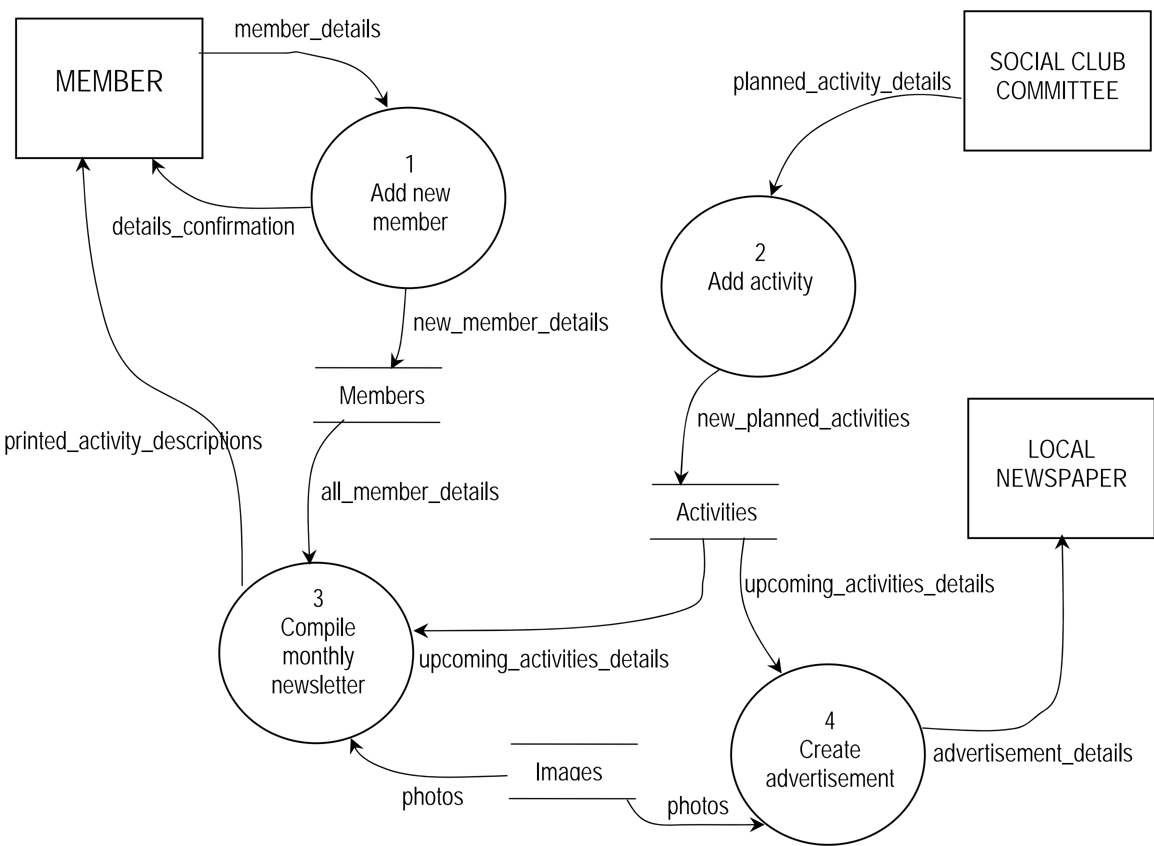


Levelled data flow diagrams

A level one data flow diagram should show all external entities. The processes are numbered, but do not indicate sequence.

In the level one data flow diagram, the same total number of inflows and outflows (and external entities) must exist as in the context diagram.

Top level or level one DFD for the social club system



Any similar information that data flows carry are resolved in the data dictionary. The number of processes that are in the level one data flow diagram depend on the number of major processes described.

Unit 3A-levelled data flow diagrams

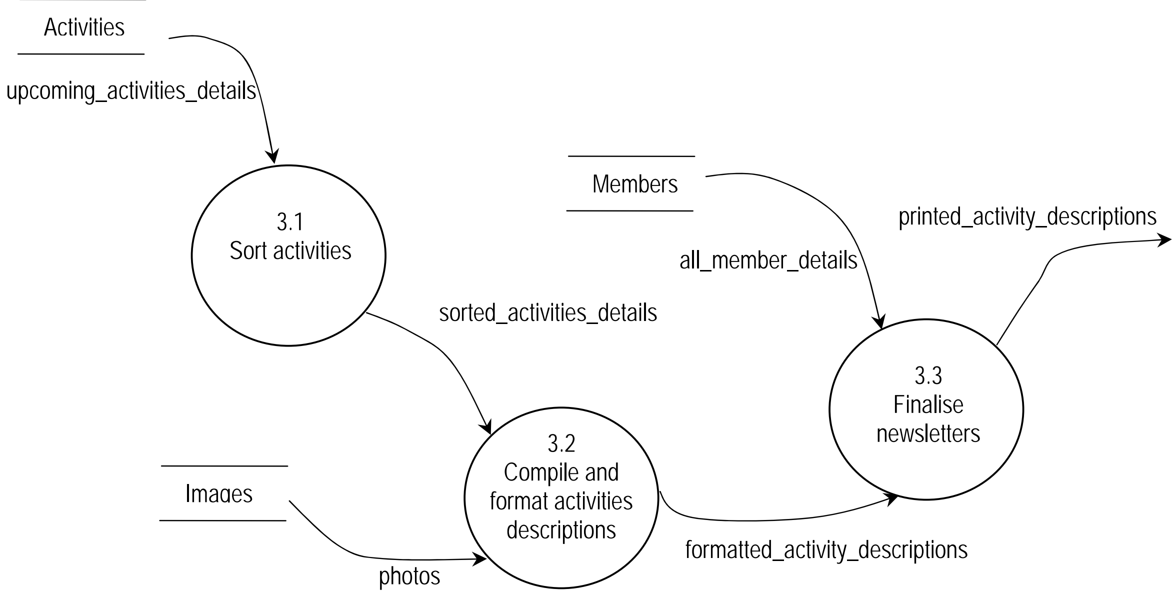
(Refer to syllabus content on p. 18)

Unit 3A build on from Unit 2A data flow diagrams.

Level 2 DFD for Process 3.0 Compile monthly newsletter

In the level 2 data flow diagram, the same total number of inflows and outflows must exist as in the level 1diagram. External entities are not shown.

Process 3 can be expanded to show more detail.



Units 1A, 2A, 3A databases

Units 1A, 2A and 3A progressively develop knowledge and skills in designing and developing **databases**.

Unit 1B (Refer to syllabus content on p. 12)
Components of a single table database—students will identify the fields and data types required to create a single table. Planning for the table structures will not require the use of a diagrammatic tool.

Students will apply skills in a **single table database application** to create a table by defining the fields with their data types and entering records of data into the table. These table records will be manipulated by sorting and filtering on various fields. Forms will be created to provide a user interface to the data. Simple queries and reports will extract and present data.

Unit 2B (Refer to syllabus content on p. 16)
Unit 2B focuses on the design and development of 2 or 3 table **relational databases** requiring one to many (1:M) relationships. **Entitiy Relationship Diagrams** will be used to represent these designs.

A **relational database application** will be used to implement:

Tables—define field names; set data types, field formats, default values, primary keys, validation rules and validation text, sort and filter on selected fields
Relationships—link tables through primary and foreign keys; enforce referential integrity
Queries—create single and multiple table queries; use relational operators (= > >= < <=); use logical operators (and, or, not); use wild cards (* ?)
Forms—create forms for displaying and entering data; create a database navigation (switchboard) form
Reports—create a report based on a table or a query.

Unit 3B (Refer to syllabus content on p. 20)
Unit 3B focuses on the conceptual planning of a larger multiple table **relational database** using **normalisation** or **Entity Relationship Diagrams**.

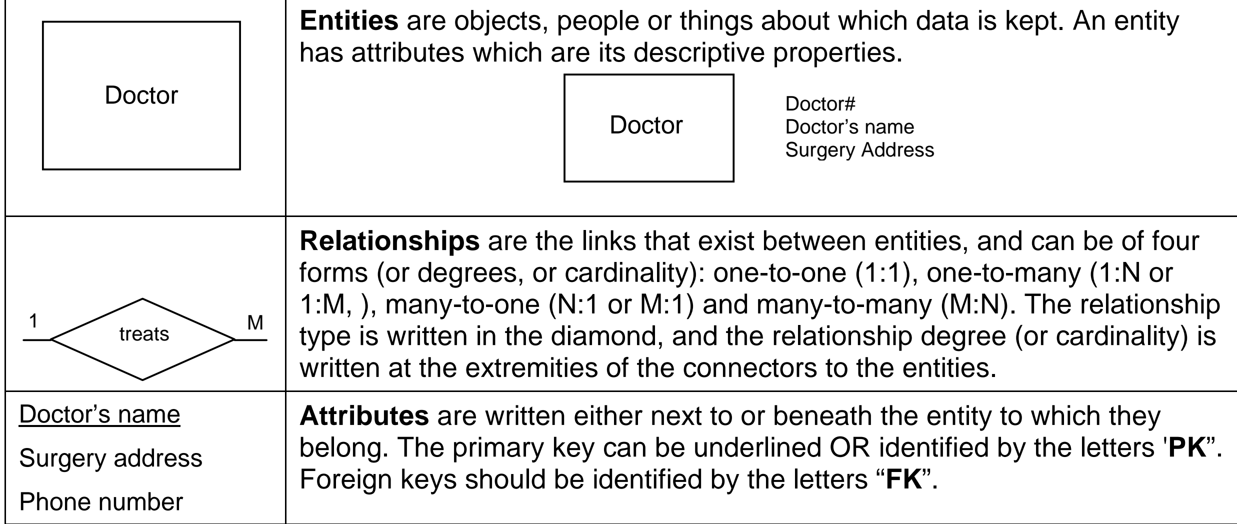
Unit 3B builds on the Unit 2B **relational database application** skills.

Relationships—set cascade updates and deletes
Queries—create parameter, calculated field concatenated field, aggregation, append, update, delete and make table queries.

Unit 2B—introduction to entity relationship diagrams

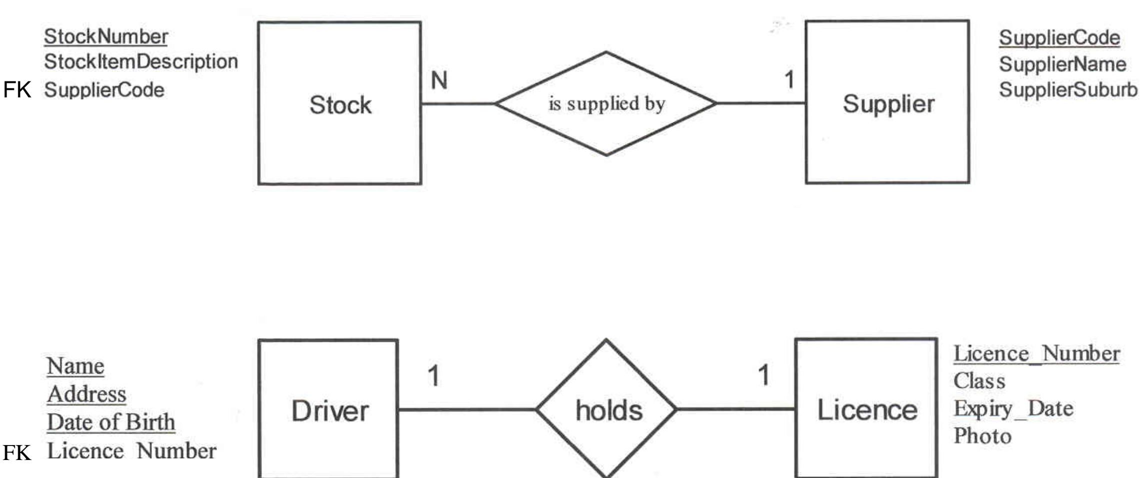
(Refer to syllabus content on p. 16)
Students represent 1:M relationships for 2 or 3 linked entities.

Symbols and characteristics



Doctor	**Entities** are objects, people or things about which data is kept. An entity has attributes which are its descriptive properties. Doctor — Doctor#, Doctor's name, Surgery Address
1 treats M	**Relationships** are the links that exist between entities, and can be of four forms (or degrees, or cardinality): one-to-one (1:1), one-to-many (1:N or 1:M,), many-to-one (N:1 or M:1) and many-to-many (M:N). The relationship type is written in the diamond, and the relationship degree (or cardinality) is written at the extremities of the connectors to the entities.
Doctor's name Surgery address Phone number	**Attributes** are written either next to or beneath the entity to which they belong. The primary key can be underlined OR identified by the letters '**PK**". Foreign keys should be identified by the letters "**FK**".

Sample ERDs



The above example shows a composite primary key for the driver.

Unit 3B—entity relationship diagrams

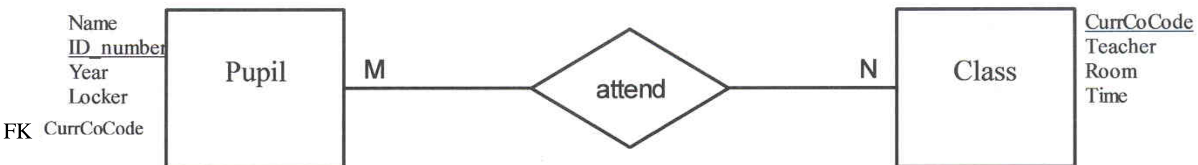
(Refer to syllabus content on p. 21)

Unit 3B builds on from Unit 2A.

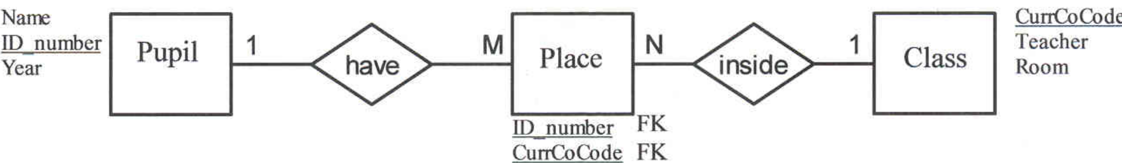
Students use 1:1, 1:M and resolve M:N relationships to design databases with up to 7 entities.

Example

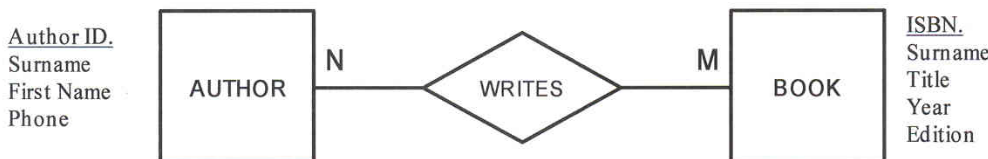
Many pupils can attend many classes. The following diagram describes this relationship.



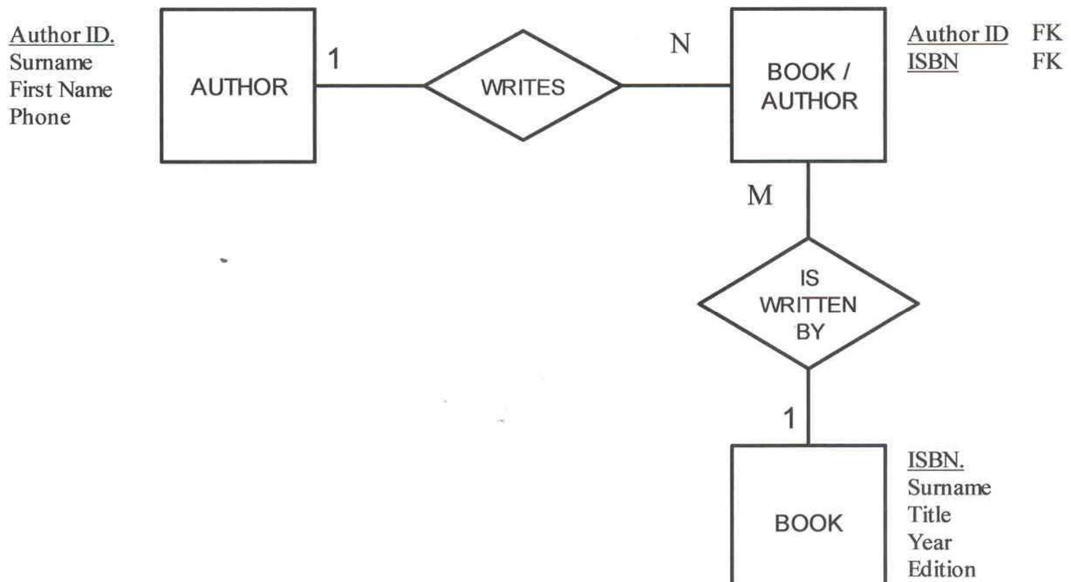
To create a model that can be implemented in a relational database the above Many to Many (M:N or M:M) relationship needs to be resolved by introducing an intersecting entity as shown below.
The intersecting Place entity has a composite primary key, but PlaceID field could have been created to create a single field primary key instead of having a composite primary key.



Many authors can write many books. The following diagram describes this relationship.



All M:N relationships resolve to 1:N M:1 format.


Author ID.
Surname
First Name
Phone
AUTHOR
1
WRITES
N
BOOK / AUTHOR
Author ID FK
ISBN FK
M
IS WRITTEN BY
1
BOOK
ISBN.
Surname
Title
Year
Edition

Unit 3B—normalisation to third normal form (3NF)

(Refer to syllabus content on p. 21)

Students normalise a set of unnormalised data to 3NF.

Normalisation is the process of splitting (decomposing) a set of data into a number of smaller sets of data suitable for implementation in a relational database.

Steps for splitting (decomposing) a relation into smaller relations

1. Identify the object fields and create a separate relation for it.
2. Create or identify a primary key for that object/relation.
3. Create a link field in the relation that you have removed the object from.

Unnormalised data (traditionally non-computerised data held in a card index file)

- Data contains repeated fields such as game 1, game 2 etc.
- Non-atomic fields such as more than one item of data in each cell/location e.g. game date, time and ground number in the same cell.
- No key fields.
- Redundant (unnecessary repetition) of data.

Example of an unnormalised Team Card

<table>
<tr><td rowspan="2">Team:
Bombers

Age Group:
Under 15</td><th>Game 1</th><th>Game 2</th><th>Game 3</th><th>Game 4</th><th>Game 5</th><th>Game 6</th></tr>
<tr><td>5th May
2:30pm,
Grnd 43</td><td>12th May
11:00am
Grnd 22</td><td>19th May
9:30am
Grnd 101</td><td>26th May
2:30pm
Grnd 43</td><td>2nd June
11:00am
Grnd 22</td><td>9th June
9:30am
Grnd 43</td></tr>
<tr><td colspan="7">Coach:
Jason Finch, 13 Green St, Wembley. 04101 23456</td></tr>
</table>

First Normal Form (1NF)—usually a flat file or single table

- Fields holds only atomic values, that is the intersection of each row and column contains only one value.
- Redundant (unnecessary repetition) of data in the records
- No key fields

Example of the fields for PlayerTeamList (in 1NF)

PlayerTeamList
PlayerSName, PlayerFName, DOB, Phone, TeamCode, TeamName, AgeGroup, HomeGroundCode, HomeGroundName, HomeGroundLocation, HomeGroundPhone

Second Normal Form (2NF)—transitive dependencies exist

All non-key fields are fully functionally dependent on the primary key, but transitive dependencies still exist.

In the example below, **Team** is still in 2NF.

- Location of the home ground is only partially dependent on the PK TeamCode as it is also dependent on the HomeGroundName or
- Location of the home ground is dependent on the HomeGroundCode as well as the TeamCode field that is, the field dependency is transitive, it has to pass through the HomeGroundName field to get to its dependency on the TeamCode.

Example of 2NF

PlayerTeamList
fPlayerID, fTeamCode, Year, CurrentPlayer
Player
PlayerID, PlayerSName, PlayerFName, DOB, Phone
Team
TeamCode, TeamName, AgeGroup, HomeGroundCode, HomeGroundName, HomeGroundLocation, HomeGroundPhone

Third Normal Form (3NF)

All transitive dependencies have been removed so that all non-key attributes are fully functionally dependent only on the primary key.

Example of 3NF

PlayerTeamList
fPlayerID, fTeamCode, Year, CurrentPlayer
Player
PlayerID, PlayerSName, PlayerFName, DOB, Phone
Team
TeamCode, TeamName, AgeGroup, fHomeGroundID
HomeGround
HomeGroundCode, HomeGroundName, HomeGroundLocation, HomeGroundPhone