4.1 Abstraction and automation part 1
Mark Scheme

Q1.

All marks AO2 (analyse)

Take a vegetable from the box labelled "onions and carrots";

If it is an onion then the box labelled "onions" contains carrots and the box labelled "carrots" contains onions and carrots. If it is a carrot then the box labelled "carrots" contains onions and the box labelled "onions" contains carrots and onions;

[2]

Q2.

(a) **All marks AO2 (analyse)**

	1	**2**	**3**	**4**	**5**	**6**
1	0	2	5	3	0	8
2	2	0	1	0	0	0
3	5	1	0	0	0	4
4	3	0	0	0	1	0
5	0	0	0	1	0	5
6	8	0	4	0	5	0

Alternative answer

	1	**2**	**3**	**4**	**5**	**6**
1	0	2	5	3	0	8
2		0	1	0	0	0
3			0	0	0	4
4				0	1	0
5					0	5
6						0

Alternative answer

	1	**2**	**3**	**4**	**5**	**6**
1	0					
2	2	0				
3	5	1	0			

4	3	0	0	0		
5	0	0	0	1	0	
6	8	0	4	0	5	0

Mark as follows:

1 mark 0s in correct places

1 mark all other values correct

I. non-zero symbols used to denote no edge but only for showing no edge going from a node to itself

2

(b) **All marks for AO1 (understanding)**

Adjacency list appropriate when there are few edges between vertices // when graph/matrix is sparse; **NE**. few edges

Adjacency list appropriate when edges rarely changed;

Adjacency list appropriate when presence/absence of specific edges does not need to be tested (frequently);

A. Alternative words which describe edge, eg connection, line, arc

Max 2

2

(c) **Mark is for AO2 (apply)**

It contains a cycle / cycles;

1

(d) **Mark for AO1 (knowledge)**

A graph where each edge has a weight/value associated with it;

1

(e) **All marks AO2 (apply)**

Mark as follows:

I. output column
1 mark first value of `A` is 2
1 mark second value of `A` is 5 and third value is 3
1 mark fourth and subsequent values of `A` are 8, 3, 7, 4, 9 with no more values after this
1 mark `D[2]` is set to 2 and then does not change

				D						P					
U	Q	V	A	1	2	3	4	5	6	1	2	3	4	5	6
-	1,2,3,4,5,6	-	-	20	20	20	20	20	20	-1	-1	-1	-1	-1	-1
				0											
1	2,3,4,5,6	2	2		2						1				
		3	5			5						1			
		4	3				3						1		
		6	8						8						1
2	3,4,5,6	3	3			3						2			
3	4,5,6	6	7						7						3
4	5,6	5	4					4						4	
5	6	6	9												
6	-														

1 mark `D[3]` is set to 5 and then changes to 3 and does not change again
1 mark correct final values for each position of array `P`

				D						P					
U	Q	V	A	1	2	3	4	5	6	1	2	3	4	5	6
-	1,2,3,4,5,6	-	-	20	20	20	20	20	20	-1	-1	-1	-1	-1	-1
				0											
1	2,3,4,5,6	2	2		2						1				
		3	5			5						1			
		4	3				3						1		
		6	8						8						1
2	3,4,5,6	3	3			3						2			
3	4,5,6	6	7						7						3
4	5,6	5	4					4						4	
5	6	6	9												
6	-														

1 mark correct final values for `D[1]`, `D[4]`, `D[5]`, `D[6]`

				D						P					
U	Q	V	A	1	2	3	4	5	6	1	2	3	4	5	6
-	1,2,3,4,5,6	-	-	20	20	20	20	20	20	-1	-1	-1	-1	-1	-1
				0											
1	2,3,4,5,6	2	2		2						1				
		3	5			5						1			
		4	3				3						1		
		6	8						8						1
2	3,4,5,6	3	3			3						2			
3	4,5,6	6	7						7						3
4	5,6	5	4					4						4	
5	6	6	9												
6	-														

Max 6 marks if any errors

7

(f) **Mark is for AO2 (analyse)**

The shortest distance / time between locations/nodes 1 and 6;

NE distance / time between locations/nodes 1 and 6

R. shortest route / path

1

(g) **All marks AO2 (analyse)**

Used to store the previous node/location in the path (to this node);

Allows the path (from node/location 1 to any other node/location) to be recreated // stores the path (from node/location 1 to any other node/location);

Max 1 if not clear that the values represent the shortest path

Alternative answer

Used to store the nodes that should be traversed;

And the order that they should be traversed;

Max 1 if not clear that the values represent the shortest path

2

[16]

Q3.

(a) **Mark is for AO1 (knowledge)**

A subroutine that calls itself;

1

(b) **Mark is for AO1 (understanding)**

When target equals node // (When target does not equal node and) node is a leaf // `node = target;`

1

(c) **Marks are for AO2 (apply)**

Function Call	Output
`TreeSearch(Olivia, Norbert)`	(Visited) Norbert;
`TreeSearch(Olivia, Phil);`	(Visited) Phil;

MAX 2 if any errors eg additional outputs / function calls after output of Phil

I. minor spelling and punctuation errors

3

[5]

Q4.

(a) **Mark is for AO1 (understanding)**

`Cavern / / TrapPositions ;`

1

(b) **Mark is for AO1 (understanding)**

`SetPositionOfItem / / MakeMonsterMove;`

1

(c) **Mark is for AO1 (understanding)**

`Count1 / / Count2 / / Choice;`
`(Python Only) NO_OF_TRAPS / / N_S_DISTANCE / / W_E_DISTANCE;`

1

(d) **Mark is for AO1 (understanding)**

`GetMainMenuChoice / / GetNewRandomPosition / / SetPositionOfItem / / SetUpGame / / SetUpTrainingGame / / GetMove / / CheckValidMove / / CheckIfSameCell / / MoveFlask`

1

(e) **All marks AO2 (analysis)**
a nested loop is used as we need to repeat something inside a section that is also repeating;
so that for each row we can loop through each column;
to work our way through the 2 dimensions of the cavern;
`Count1` controls the rows of the display;
`Count2` controls the columns of the display;
Max 3: Any 3 from above

3

(f) **All marks AO1 (understanding)**
1 mark: Only need to change its value once/ /at the top of the program / / from one place only (for the new value to apply wherever it is used);
1 mark: Makes program code easier to understand;

2

(g) **All marks AO2 (analyse)**
1 mark: (Command inside loop) randomly chooses coordinates to place item at;
1 mark: The condition checks that no other item has already been placed at the selected coordinates / / the location is empty;
1 mark: The `while` loop is required to repeat the coordinate selection until an empty location is found / / to keep choosing coordinates if the location found is not empty;

3

(h) **All marks AO2 (analyse)**
1 mark: If the monster moves into the flask cell and the flask is not moved elsewhere it will not be there when the monster moves away from this cell;
1 mark: You can't have two items in any cell;
1 mark: By swapping the monster and the flask cells we can ensure that the flask stays in the cavern;

3

(i) **All marks AO2 (analyse)**
1 mark: Even though the monster usually makes 2 moves the player might be eaten on the first of the two moves;
1 mark: A `while` loop allows us to complete 2 moves when necessary but exit on the first move if the player is eaten;

2

(j) **All marks AO1 (understanding)**
Easier reuse of routines in other programs;
Routine can be included in a library;
Helps to make the program code more understandable;
Ensures that the routine is self-contained / / routine is independent of the rest of the program;
(global variables use memory while a program is running) but local variables use memory for only part of the time a program is running;
reduces possibility of undesirable side effects;
using global variables makes a program harder to debug;
Max 2: Any 2 from above

2

[19]

Q5.

(a) **Mark is for AO2 (apply)**

1 mark: B;

1

(b) **All marks AO2 (analyse)**
Nathan was not killed with poison (rule a);
therefore Peter was not in the kitchen (rule c);
therefore Martin was not in the dining room (rule e);
therefore Suzanne was in the dining room (rule b);
therefore Steve murdered Nathan (rule d).

Mark as follows:
1 mark: Any correct point from the list above;
1 mark: Any two further correct points from the list above;

2

[3]

Q6.

(a) **Mark is for AO1 (understanding)**

Original state	Input	New state
S3	0	S4
S3	1	S2

1 mark: Table completed as above
I order of rows

1

(b) **All marks AO2 (analyse)**

(0|1)*((00)|(11))(0|1)*

Mark as follows:
1 mark: (0|1)* at start;
1 mark: (00)|(11);
1 mark: (0|1)* at end;

Or

Alternative answer
(0|1)*(11(0|1)*)|(00(0|1)*)

Mark as follows:
1 mark: (0|1)* at start;
1 mark: (11(0|1)*);
1 mark: |(00(0|1)*) at end;

Maximum 2 marks: If final answer not correct.

A any regular expression that correctly defines the language.

3

(c) **Mark is for AO2 (apply)**

Rule number (given in Figure 2)	Could be defined using a regular expression
1	Y
2	Y
3	Y
4	N
5	N
6	Y

1 mark: All values in the table have been completed correctly.

1

(d) **1 mark for AO2 (analyse) and 1 mark for AO3 (design)**

1 mark for AO2 (analyse): There is no non-recursive / base case;
1 mark for AO3 (design): `<word> ::= <char><word> | <char>;`

2

[7]

Q7.

(a) Values / cards need to be taken out of the data structure from the opposite end that they are put in / / cards removed from top / front and added at end / bottom / rear;
Values / cards need to be removed in the same order that they are added;
A It is First in First Out / / It is FIFO;
A It is Last in Last Out / / It is LILO;
MAX 1

1

(b) (i)
```
FrontPointer = 11
RearPointer = 52
QueueSize = 42
```
1 mark for all three values correct

1

(ii)
```
FrontPointer = 11
RearPointer = 2
QueueSize = 44
```

1 mark for all three values correct
A incorrect value for `FrontPointer` if it matches the value given in part (i) and incorrect value for `QueueSize` if it is equal to the value given for `QueueSize` in part (i) incremented by two (follow through of errors previously made)

1

(iii)
```
If DeckQueue is empty then
  Report error
Else
```

```
  Output DeckQueue[FrontPointer]
  Decrement QueueSize
  Increment FrontPointer
  If FrontPointer>52 Then
     FrontPointer = 1
EndIf
```

1 mark for If statement to check if queue is empty – alternative for test is `QueueSize = 0`.
1 mark for reporting an error message if the queue is empty / / dealing with the error in another sensible way – this mark can still be awarded if there is an error in the logic of the If statement, as long as there is an If statement with a clear purpose.
1 mark for only completing the rest of the algorithm if the queue is not empty – this mark can still be awarded if there is an error in the logic of the If statement, as long as there is an If statement with a clear purpose.
1 mark for outputting the card at the correct position
1 mark for incrementing FrontPointer and decrementing QueueSize
1 mark for If statement testing if the end of the queue has been reached
1 mark for setting FrontPointer back to 1 if this is the case – this mark can still be awarded if minor error in logic of If statement, eg >= instead of =
A `FrontPointer = (FrontPointer` MOD 52) + 1 for **3 marks** or FrontPointer = (FrontPointer MOD 52) for **2 marks**, both as alternatives to incrementing and using and the second If statement - **deduct 1 mark** from either of the above if
QueueSize has not been decremented
A any type of brackets for array indexing
I Additional reasonable EndIf Statements
MAX 5 unless all of the steps listed above are carried out

6

(c) Flow of program / execution sequence determined by events / / program executes relevant code-handling block / procedure / sub-routine in response to events;
Example of event such as clicking a button;
Message sent to program when event occurs;
System / message loop executes until application closes; this receives and processes messages / / use of event-listener / handler;
If several events occur they are queued;
MAX 2

2

(d) The smartphone operating system:

- Will not have to support as wide a range of hardware devices / peripherals / / may not support external storage devices;
- Will not / less likely to need to support the addition of new hardware devices to the system;
- Will have minimising power consumption as a higher priority; **A** will have more sophisticated power management features
- Will run application software in a sandbox / / will (more tightly) restrict access to resources by application software;
- Must be capable of running on a device with less processing power / / less RAM / memory / / smaller memory footprint;
- Needs to work with specialised hardware devices eg GPS receiver, accelerometer (**Note:** A relevant example must be given);

A Points made in reverse, but do not give **2 marks** for one point and its reverse.
MAX 2

2

[13]

Q8.

(a) Omitting unnecessary details (from a representation) / / Storing only those details which are necessary (in the context of the problem being solved);
R responses that do not refer to abstraction in the context of data or modelling

1

(b) **SUBJECT MARKING POINTS:**

Representation as a graph:

Vertex / node represents a station; **A** junction between railway lines

Edge / arc / line represents a (direct) connection / railway line between two stations; **R** vector
Graph must be weighted / / edges have weights;
Distance between two stations must be written on edge / / stored with edge / / weights will be distances;
Could be more than one direct route between two stations; in which case shortest of the distances would be stored as the weight;

Graph would be undirected as trains can travel in both directions between each station;
OR
Graph would be directed as some lines may only be traversable in one direction;
Note: Only accept one of the above two points about whether the graph would be directed or undirected. Must have reason.

Implementation as array:

Each station assigned a (unique) number (to be used as array index); - **This mark available regardless of how the rest of the implementation is done**

Using an adjacency matrix:

The (adjacency) matrix would be a two-dimensional array (of numbers);
Array contains one row and one column for each station / / An n x n array is required to represent n stations; **A** rows and columns labelled with stations for BOD mark
If there is a (direct) connection between the two stations, store the distance between the two stations at the intersection of the row / column for the stations;
If there is no (direct) connection between the two stations, store a value to indicate this at the intersection of the row / column for the stations; **A** examples of values eg 0, ∞, NULL that could not be valid distances including any alphanumeric indicator

Using an adjacency list as a 2d array of numbers:

Adjacency list could be stored in a two-dimensional array (of records or similar);

In one dimension there would have to be n rows / columns for n stations / / one row / column per station;
In the other dimension the number of columns / rows would be determined by the highest degree of any vertex / / the maximum number of neighbours a vertex has / / the maximum number of (direct) connections that any station has;
If a station is (directly) connected to another station then in the row / column for the first station, a new entry (record) would be made consisting of the number of / / an identifier for the second station and the distance to it; **NE** just to state identifier, must have distance as well
Note: Also accept implementation in two two-dimensional arrays, with one storing the stations and the other the distances, as long as made clear station identifiers and distances stored in corresponding positions.

Using an adjacency list as a 1d array of strings:

Adjacency list could be stored in a (one-dimensional) array of strings;
One row per station;
The string for a station contains, for each station that it is connected to, the station identifier / name and distance;
Use of a delimiter between values;

REFER ANY OTHER WORKABLE SOLUTION TO A TEAM LEADER

If comparison made between adjacency matrix and adjacency list (not asked for):

Adjacency list might be more efficient (in terms of storage space) as graph is likely to be sparse / / as few edges between vertices / / as most stations only (directly) connected to a small number of other stations;
Adjacency matrix might be more efficient (in terms of speed) as shortest route finding algorithm is likely to need to lookup many distances when computing a route;

Note on use of diagrams: Candidates may choose to use diagrams to help clarify their responses. When marking, use may be made of such diagrams to help clarify understanding of the written description, however as this question assesses quality of written communication, marks should be awarded for the written description, not directly for the diagrams themselves.

8

HOW TO AWARD MARKS:

Mark Bands and Description

To achieve a mark in this band, candidates must meet the subject criterion (SUB) and all 5 of the quality of written communication criteria (QWCx).

SUB	Candidate has covered the graph representation and array implementation in detail and all or almost all of the required detail for an implementation is present. The candidate has made at least seven subject-related points.
QWC1	Text is legible.
QWC2	There are few, if any, errors of spelling, punctuation and grammar. Meaning is clear.
QWC3	The candidate has selected and used a form and style of writing appropriate to the purpose and has expressed ideas clearly and fluently.
QWC4	Sentences (and paragraphs) follow on from one another clearly and coherently.

QWC5 Appropriate specialist vocabulary has been used.

7-8

To achieve a mark in this band, candidates must meet the subject criterion (SUB) and 4 of the 5 quality of written communication criteria (QWCx).

SUB Candidate has covered both the graph representation and array implementation, making some valid points for each, but the level of detail may not be sufficient to implement. The candidate has made at least five subject-related points.
QWC1 Text is legible.
QWC2 There may be occasional errors of spelling, punctuation and grammar. Meaning is clear.
QWC3 The candidate has, in the main, used a form and style of writing appropriate to the purpose, with occasional lapses. The candidate has expressed ideas clearly and reasonably fluently.
QWC4 The candidate has used well-linked sentences (and paragraphs).
QWC5 Appropriate specialist vocabulary has been used.

5-6

To achieve a mark in this band, candidates must meet the subject criterion (SUB) and 4 of the 5 quality of written communication criteria (QWCx).

SUB Candidate has made some relevant points but the description is either lacking in detail or only covers one of the graph representation or array implementation.
QWC1 Most of the text is legible.
QWC2 There may be some errors of spelling, punctuation and grammar but it should still be possible to understand most of the response.
QWC3 The candidate has used a form and style of writing which has many deficiencies. Ideas are not always clearly expressed.
QWC4 Sentences (and paragraphs) may not always be well-connected.
QWC5 Specialist vocabulary has been used inappropriately or not at all.

1-4

Candidate has made no relevant points.

0

Note: Even if English is perfect, candidates can only get marks for the points made at the top of the mark scheme for this question.

If a candidate meets the subject criterion in a band but does not meet the quality of written communication criteria then drop mark by one band, providing that at least 4 of the quality of language criteria are met in the lower band. If 4 criteria are not met then drop by two bands.

[9]

Q9.

(a) Static structures have fixed (maximum) size whereas size of dynamic structures can change // Size of static structure fixed at compile-time whereas size of dynamic structure can change at run-time;
Static structures can waste storage space / memory if the number of data items stored is small relative to the size of the structure whereas dynamic structures only take up the amount of storage space required for the actual data;
Dynamic data structures (typically) require memory to store pointer(s) to the next item(s) which static structures (typically) do not need // Static structures (typically) store data in consecutive memory locations, which dynamic data structures

(typically) do not;
Max 2
A just one side of points, other side is by implication
NE Dynamic data structures use pointers

2

(b) Not possible to simply insert item into middle of list;
Must move all items that should come after the new process down in the array;
NE move all data
Moving items is time consuming;
In a dynamic implementation, insertion achieved by adjusting pointers;
Max 2

2

(c) Priority (queue);

1

(d) (i) Memory allocated / deallocated at run-time / for new items (to dynamic data structure);
(Provides a) pool of free / unused / available memory;
NE to store new items
Max 1

1

(ii) (Memory) address // memory location // position in memory;
NE position or location without reference to memory
R index

1

(iii) **OVERALL GUIDANCE:**

Solutions should be marked on this basis:

- Up to **4 marks** for correctly locating the position to insert the new process at.

- Up to **4 marks** for creating a new node and storing the correct data into it and the associated pointers.
  Some marks can be awarded for this even if the locating process is incorrect / missing.

The full 7 marks should only be awarded for a complete fully working solution. If any steps are missed out, then award a Maximum of six marks (Max 6).

The addition of any unnecessary steps that do not stop the algorithm working should not result in a reduction in marks.

Responses should be accepted in pseudo-code or structured English.
If you are unsure about the correctness of a solution please refer it to a team leader. Also, responses in prose should be referred to team leaders.

SPECIFIC MARKING POINTS

Correctly locating insertion point (Max 4):

1 Initialising current node pointer to start pointer;
2 Use of loop to attempt to move through list (regardless of correct

terminating condition);
3 Advancing current node pointer within loop;
4 Correctly maintaining a pointer to the node before the position that the new node should be inserted at;
5 Sensible condition to identify place to insert (suitable terminating condition for loop or condition in selection statement);

Correctly inserting new process (Max 4):

6 Create a new node / obtain new node from heap;
7 Store new process name and priority (in new node);
8 Update NextNodePointer of node before newly inserted one to point to new node;
9 Set NextNodePointer of new node to point to node after it;

Mark point 2 can only be awarded if, within the loop, current node pointer is being changed (even if not correctly changed).

Mark point 4 can only be awarded if mark point 3 had been awarded.

Mark point 5 can be awarded if there is a sensible condition, even if current node pointer is not correctly updated.

Mark points 8 and 9 can only be awarded if the correct insertion point has been found.

For any solution:

A use of either while or repeat loops, as long as logic is correct.
A storage of values into new node in any order, and regardless of whether the node has been created or not.
A use of ^ symbol to indicate the value stored at an address referenced by a pointer, for example CurrentNodePointer^. Priority indicates the value stored in the Priority field of the node pointed to by the pointer CurrentNodePointer.
A use of alternative variable names so long as the meaning is clear.

EXAMPLE SOLUTIONS AND MARKS:

These four examples show where marks should be awarded in some possible solutions (subject to a maximum mark of 7):

Example 1:

```
CurrentNodePointer = StartPointer;
Repeat
  PreviousNodePointer =
   CurrentNodePointer;
  CurrentNodePointer =
   NextNodePointer of current node;
Until priority of process in current node < priority
of process to add
//
 priority = "Low";;
Create new node;
Store new process name (and priority)
 in new node;
```

```
New node's NextNodePointer = Next
 NodePointer of item at position
 PreviousNodePointer;
NextNodePointer of item at position
 PreviousNodePointer = Address of new
 node;
```

Example 2:

This is an alternative way of expressing Example 1:

1 Load the Start Pointer value into the Current Node Pointer;
2 Copy value from Current Node Pointer into Previous Node Pointer;
3 Set Current Node Pointer to Next Node Pointer of current node;
4 If the priority of the data item at the current node is higher than or the same as the priority of the process to be added; then go back to step 2;
5 Create a new node;
6 Store the name of the new process (and its priority) in the new node;
7 Copy value from Next Node Pointer of list entry at position stored in Previous Node Pointer into the Next Node Pointer of the new node;
8 Set the Next Node Pointer of the list entry at position stored in the Previous Node Pointer to point to the new node;

Example 3:

```
CurrentNodePointer = StartPointer;
Inserted = False
While; Inserted = False Do
  If Current Node's priority < new item
    priority // = "Low"; Then
      Create new node;
      Store new process name (and priority)
        in new node;
      New node's NextNodePointer =
        CurrentNodePointer;
      NextNodePointer of item at position
        PreviousNodePointer = Address of new
        node;
      Inserted=True
  End If
  PreviousNodePointer = CurrentNodePointer;
  CurrentNodePointer =
    NextNodePointer of current node;
End While
```

Example 4:

```
CurrentNodePointer = StartPointer;
While; not at end of list // While
 CurrentNodePointer <> Nil // While
 priority of process at CurrentNodePointer
 >= priority of process to add Do
```

```
    If Current Node's priority is the required
     priority // = "Normal"; Then
      LastNodeOfCurrentPriorityPointer =
       CurrentNodePointer;
    End If
    CurrentNodePointer =
      NextNodePointer of current node;
  End While
  Create new node;
  Store new process name (and priority)
    in new node;
  New node 's NextNodePointer = Next
    NodePointer of item at position
    LastNodeOfCurrentPriorityPointer;
  NextNodePointer of item at position
    LastNodeOfCurrentPriorityPointer = Address
    of new node;
```

7

[14]

Q10.

(a) Correct variable declarations for `Bit, Answer` and `Column;`
I additional variable declarations
`Column` initialised correctly before the start of the loop;
`Answer` initialised correctly before the start of the loop;
`While/Repeat` loop, with syntax allowed by the programming language used, after the variable initialisations; and correct condition for the termination of the loop;
R `For` loop
A any `While/Repeat` loop with logic corresponding to that in flowchart
(for a loop with a condition at the start accept >=1 or >0 but reject <>0)
Correct prompt `"Enter bit value:" ;`
followed by `Bit` assigned value entered by user;
followed by `Answer` given new value;
R if incorrect value would be calculated [followed by value of `Column` divided by 2;
A multiplying by 0.5
Correct prompt and the assignment statements altering `Bit, Answer` and `Column` are all within the loop;
After the loop – output message followed by value of `Answer`;
I Case of variable names, player names and output messages
A Minor typos in variable names and output messages
I spacing in prompts
A answers where formatting of decimal values is included e.g.
`Writeln('Decimal value is: ', Answer : 3)`
A `initialisation of variables at declaration stage`
A `no brackets around column * bit`

Pascal

```
Program Question;
    Var
         Answer : Integer;
         Column : Integer;
         Bit : Integer;
    Begin
```

```
        Answer := 0;
        Column := 8;
        Repeat
            Writeln('Enter bit value: ');
            Readln(Bit);
            Answer := Answer + (Column * Bit);
            Column := Column DIV 2;
      Until Column < 1;
      Writeln('Decimal value is: ', Answer);
      Readln;
  End.
```

VB.NET

```
Sub Main()
   Dim Answer As Integer
   Dim Column As Integer
   Dim Bit As Integer
   Answer = 0
   Column = 8
   Do
        Console.Write("Enter bit value: ")
        Bit = Console.ReadLine
        Answer = Answer + (Column * Bit)
        Column = Column / 2
  Loop Until Column < 1
  Console.Write("Decimal value is: " & Answer)
  Console.ReadLine()
End Sub
```

Alternative Answer

```
Column = Column \ 2
```

VB6

```
Private Sub Form_Load()
   Dim Answer As Integer
   Dim Column As Integer
   Dim Bit As Integer
   Answer = 0
   Column = 8
   Do
        Bit = InputBox("Enter bit value: ")
        Answer = Answer + (Column * Bit)
        Column = Column / 2
  Loop Until Column < 1
  MsgBox ("Decimal value is: " & Answer)
End Sub
```

Alternative Answer

```
Column = Column \ 2
```

Java

```
public class Question {
   AQAConsole console=new AQAConsole();
   public Question(){
        int column;
        int answer;
        int bit;
        answer=0;
        column=8;
        do{
            console.print("Enter bit value: ");
            bit=console.readInteger("");
            answer=answer+(column*bit);
```

```
            column=column/2;
         }while(column>=1);
         console.print("Decimal value is: ");
         console.println(answer);
      }
      public static void main(String[] arrays){
         new Question();
      }
}
```

Python 2.6

```
Answer = 0
Bit = 0
Column = 8
while Column >= 1:
   print "Enter bit value: "
   # Accept: Bit = int(raw_input("Enter bit value: "))
   Bit = input()
   Answer = Answer + (Column * Bit)
   Column = Column // 2
print "Decimal value is: ", Answer
# or + str(Answer)
```

Python 3.1

```
Answer = 0
Bit = 0
Column = 8
while Column >= 1:
   print("Enter bit value: ")
   # Accept: Bit = int(input("Enter bit value: "))
   Bit = int(input())
   Answer = Answer + (Column * Bit)
   Column = Column // 2
print("Decimal value is: " + str(Answer))
# or print("Decimal value is: {0}".format(Answer))
```

A. Answer and Bit not declared at start as long as they are spelt correctly and when they are given an initial value that value is of the correct data type

11

(b) ****SCREEN CAPTURE****
Must match code from 16, including prompts on screen capture matching those in code

Mark as follows:
"Enter bit value:" + first user input of 1
'Enter bit value: ' + second user input of 1
'Enter bit value: ' + third user input of 0
'Enter bit value: ' + fourth user input of 1
Value of 13 outputted;

3

(c) 15;

1

(d) 16 // twice as many // double;

1

(e) Design;
A Planning

1

(f) Implementation;

1

[18]

Q11.

(a) `ResetCavern;` (all languages)
// `GetNewRandomPosition` (Pascal only)
// `WriteWithMsg` (VB6 only)
// `WriteLineWithMsg` (VB6 only)
// `WriteLine` (VB6 only)
// `WriteNoLine` (VB6 only)
// `ReadLine` (VB6 only);
// `SetUpTrapPostions` (Python / Java only);
R if any additional code (including routine interface)
R if spelt incorrectly
I case

1

(b) `DisplayMenu // DisplayMoveOptions // DisplayWonGameMessage // DisplayTrapMessage // DisplayLostGameMessage // WriteWithMsg` (VB6 only) // `WriteLineWithMsg` (VB6 only) // `WriteLine` (VB6 only) // `WriteNoLine`(VB6 only);
A `DisplayCavern;`
R if any additional code (including routine interface)
R if spelt incorrectly
I case

1

(c) `Count1 // Count2 // Count;`
R if any additional code
R if spelt incorrectly
I case

1

(d) `Cavern // TrapPositions;`
R if any additional code (including routine interface)
R if spelt incorrectly
A `LoadedGameData.TrapPositions`
A `CurrentGameData.TrapPositions`
I case

1

(e) When the value of the cell in the `Cavern` array // When the value of the cell to place the item in;
Indicated by the `Position` variable;
Contains a space // does not contain another item;
R is empty

Max 2 if no variable names used in description

3

(f) The number of times to repeat is known;
A fixed

1

(g) Makes the program code easier to understand;
Makes it easier to update the program;
Makes it easier to change the number of traps (in the game);

Max 1

(h) In text files all data is stored as strings / ASCII values / lines/characters // Text files use only byte values that display sensibly on a VDU // stores only values that can be opened and read in a text editor;

Binary files store data using different data types; **A** by example **A** Binary files can only be correctly interpreted by application that created them

2

(i) Easier reuse of routines in other programs;
Routine can be included in a library;
Helps to make the program code more understandable;
Ensures that the routine is self-contained // routine is independent of the rest of the program;
(Global variables use memory while a program is running) but local variables use memory for only part of the time a program is running;
reduces possibility of undesirable side effects;
Using global variables makes a program harder to debug;

Max 2

(j) (If it was not then) `MonsterAwake` is set to the Boolean value returned by the second call to `CheckIfSameCell;`
this would overwrite any True value returned by the first call to `CheckIfSameCell;`
//
Otherwise the monster would never wake up when the player triggers the first trap;;
//
Otherwise the monster would only wake up when the player triggers the second trap;;

2

[15]

Q12.

(a) Space / Memory (complexity);
A amount of memory used

1

(b) (i)

N	Pos1	W1	Pos2	W2	Output	
3	1	Rope	1	Rope		
			2	Dagger		1 mark
			3	Rope	Duplicate: Rope	
	2	Dagger	1	Rope		
			2	Dagger		1 mark
			3	Rope		
	3	Rope	1	Rope	Duplicate: Rope	
			2	Dagger		1 mark
			3	Rope		

A answers which have correct values repeated in empty cells, but do not award a mark if there are any incorrect values within the block for which the mark is being awarded.
A additional rows in trace table, so long as the trace is correct.
DPT if just "Duplicate" or "Rope" are written in the Output column when it should be "Duplicate: Rope" or if the value of Pos1 is written in the output instead of W1 e.g. 1 instead of "Rope"

If candidate has not written in the value of N, only one mark should be lost (for the top rectangular area) for this mistake

3

(ii) $O(n^2)$;

1

(iii) ***Marks can only be awarded if correct answer for part (b)(ii)***
Alternative 1:
Algorithm has nested loops // two loops with one inside the other;
A reference to inner and outer loops
Each loop repeats N times;

Alternative 2:
The (basic) operation / If statement / file read / comparison is carried out N^2 times; because it is inside nested loops // because each loop executes N times;
Alternative 3:
Each of the (N) entries is compared to each of the (N) others // each entry is compared N times; so N2 (A N*N) comparisons/operations are required // $N*N=N^2$;
A uppercase or lowercase n
A answers where examples are used instead of N and N^2, e.g. 3 and 9.
A check as alternative to comparison

Max 2

[7]

Q13.

(a) **Connected** // There is a path between each pair of vertices;
Undirected // No direction is associated with each edge;
Has no cycles // No (simple) circuits // No closed chains // No closed paths in which all the edges are different and all the intermediate vertices are different // No route from a vertex back to itself that doesn't use an edge more than once or visit an intermediate vertex more than once;
A no loops
Alternative definitions:
A simple cycle is formed if any edge is added to graph;
Any two vertices can be connected by a unique simple path;

Max 1

(b) No route from entrance to exit / through maze;
Maze contains a loop/circuit ;
A more than one route through maze;
Part of the maze is inaccessible / enclosed;
R Responses that clearly relate to a graph rather than the maze

Max 1

(c)

	1	2	3	4	5	6	7
1	0	1	0	0	0	0	0
2	1	0	1	1	0	0	0
3	0	1	0	0	0	0	0
4	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1
6	0	0	0	0	1	0	0
7	0	0	0	0	1	0	0

(allow some symbol in the central diagonal to indicate unused)

or

	1	2	3	4	5	6	7
1	0	1	0	0	0	0	0
2		0	1	1	0	0	0
3			0	0	0	0	0
4				0	1	0	0
5					0	1	1
6						0	0
7							0

(with the shaded portion in either half – some indication must be made that half of the matrix is not being used. This could just be leaving it blank, unless the candidate has also represented absence of an edge by leaving cells blank)

1 mark for drawing a 7x7 matrix, labelled with indices on both axis and filled only with 0s and 1s, or some other symbol to indicate presence/absence of edge. e.g. T/F. Absence can be represented by an empty cell.
1 mark for correct values entered into matrix, as shown above;

2

(d) (i) Routine defined in terms of itself // Routine that calls itself;
A alternative names for routine e.g. procedure, algorithm
NE repeats itself

1

(ii) Stores return addresses;
Stores parameters;
Stores local variables; NE temporary variables
Stores contents of registers;
A To keep track of calls to subroutines/methods etc.

Max 1

Procedures / invocations / calls must be returned to in reverse order (of being called);
As it is a LIFO structure;
A FILO
As more than one / many return addresses / sets of values may need to be stored (at same time) // As the routine calls itself and for each call/invocation a new return address / new values must be stored;

Max 1

2

(e)

				Discovered							Completely Explored							
Call	V	U	EndV	1	2	3	4	5	6	7	1	2	3	4	5	6	7	F
	-	-	7	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
DFS(1,7)	1	2	7	**T**	F	F	F	F	F	F	F	F	F	F	F	F	F	F
DFS(2,7)	2	1	7	T	**T**	F	F	F	F	F	F	F	F	F	F	F	F	F
		3	7	T	T	F	F	F	F	F	F	F	F	F	F	F	F	F
DFS(3,7)	3	2	7	T	T	**T**	F	F	F	F	F	F	**T**	F	F	F	F	F
DFS(2,7)	2	4	7	T	T	T	F	F	F	F	F	F	T	F	F	F	F	F
DFS(4,7)	4	2	7	T	T	T	**T**	F	F	F	F	F	T	F	F	F	F	F
		5	7	T	T	T	T	F	F	F	F	F	T	F	F	F	F	F
DFS(5,7)	5	4	7	T	T	T	T	**T**	F	F	F	F	T	F	F	F	F	F
		6	7	T	T	T	T	T	F	F	F	F	T	F	F	F	F	F
DFS(6,7)	6	5	7	T	T	T	T	T	**T**	F	F	F	T	F	F	**T**	F	F
DFS(5,7)	5	7	7	T	T	T	T	T	T	F	F	F	T	F	F	T	F	F
DFS(7,7)	7	5	7	T	T	T	T	T	T	**T**	F	F	T	F	F	T	**T**	**T**
DFS(5,7)	5	-	7	T	T	T	T	T	T	T	F	F	T	F	**T**	T	T	T
DFS(4,7)	4	-	7	T	T	T	T	T	T	T	F	F	T	**T**	T	T	T	T
DFS(2,7)	2	-	7	T	T	T	T	T	T	T	F	**T**	T	T	T	T	T	T
DFS(1,7)	1	-	7	T	T	T	T	T	T	T	**T**	T	T	T	T	T	T	T

1 mark for having the correct values changes in each region highlighted by a rectangle and no incorrect changes in the region. Ignore the contents of any cells that are not changed.

A alternative indicators that clearly mean True and False.
A it is not necessary to repeat values that are already set (shown lighter in table)

5

[12]

Q14.

(a) To measure out one litre of water;

1

(b) 3 litre capacity jug; and 5 litre capacity jug;
A 2 jugs;
sink; with a tap/water;
drain;
Bob // Bob's knowledge and problem solving skills;
Time;

Max 3

(c) Who is responsible for solving the problem;

1

[5]

Q15.

(a) **VB.Net/VB6**

```
Const MaxSize = 4
```

I capitalisation

Pascal

```
Const MaxSize = 4
```

I missing semicolon, capitalisation
NE MaxSize
A MaxSize = 4

Java

```
final int MAX_SIZE = 4;
```

I missing semicolon, capitalisation
NE `MAX_SIZE`

Python 2.6 and 3

```
MAX_SIZE = 4
```

1

(b) Improves readability of code // Easier to update the programming code if the value changes (**A** by implication) // reduce the likelihood of errors;

1

(c) `PlayerOneName // PlayerTwoName;`
R if any additional code
R if spelt incorrectly
I case & spaces
A `Max_SIZE (Python only)`
A `Currentfile` (**R** for VB6/VB.Net)

1

(d) `LowestCurrentTopScore ;`
A `PositionOfLowestCurrentTopScore;`
R if any additional code
R if spelt incorrectly
I case & spaces

1

(e) `b;`

1

(f) `True;`

1

(g) `False;`

1

(h) `UpdateTopScores;`
R if spelt incorrectly
I case & spaces

1

(i) `VirtualDiceGame;`
R if spelt incorrectly
I case & spaces

1

(j) `AppealDieResult;`

`RollAppealDie;`
R if spelt incorrectly
R RollAppealDie (Python only)
I case & spaces

1

(k) Until PlayerOut // Until PlayerOut = True // until player is out;
A any unambiguous description of the loop termination condition

1

(l) Because the scope; of the two variables is different; //
Because they are both local variables; in different subroutines;
A Because where they are accessible is different;

2

(m) 3;

1

(n) It compares the score of the current record/position (in the `TopScores` array);
with the lowest score found so far // with LowestCurrentTopScore;
if it is less than it then it changes the lowest score found so far; **R** swaps and makes the position of the lowest top score equal to count / equal to the current position in the array;

4

[18]

Q16.

(a)

Reverse Polish Notation	Equivalent Infix Expression
45 6 +	*45 + 6* ***R*** *6 + 45*
12 19 + 8 *	*(12 + 19) * 8* ***R*** *12+19*8, (19+12)*8* ***A*** *x for **

2

(b) Simpler for a machine / computer to evaluate // simpler to code algorithm
A easier **R** to understand
Do not need brackets (to show correct order of evaluation/calculation);
Operators appear in the order required for computation;
No need for order of precedence of operators;
No need to backtrack when evaluating;
A RPN expressions cannot be ambiguous as **BOD**

1

(c)

String Pos	Token	Integer Val	Op1	Op2	Result	Stack
0	-	-	-	-	-	
1	6	6				6
2	4	4				4 6
3	+		**6**	**4**	10	10
4	3	3				3 10
5	2	2				2 3 10
6	+		**3**	**2**	5	5 10
7	*		**10**	**5**	50	50

Output : **50**

1 mark for each of rows 1–3
1 mark for rows 4 and 5 together
1 mark for rows 6 and 7 together
1 mark for correct final output
Values of Op1 and Op2 MUST be assigned in rows 3, 6 and 7 to award the marks for these rows. They cannot be inferred from incorrectly entered previous values.

I values in empty cells, even if they are incorrect.

6

(d)
```
If StackArray is full
  Then Stack Full Error
  Else
    Increment TopOfStackPointer
    StackArray [TopOfStackPointer] ←
        ANumber
EndIf
```

1 mark for appropriate `If` structure including condition (does not need both `Then` and `Else`) – Do not award this mark if `ANumber` is put into `StackArray` outside the `If`.
1 mark for reporting error in correct place
1 mark for incrementing `TopOfStackPointer`*
1 mark for storing value in `ANumber` into correct position in array*
** = if the store instruction is given before the increment instruction OR*

*the `If` structure then award **Max 1** of these two marks UNLESS the item is inserted at position `TopOfStackPointer+1` so the code would work.*

I initialisation of `TopOfStackPointer` to 0
A `TopOfStackPointer=20/>=20` for Stack is full
A Logic of `if` structure reversed i.e. If stack is not full / `TopOfStackPointer<20 / <>20/!=20` and `Then, Else` swapped
A Any type of brackets or reasonable notation for the array index
DPT If candidate has used a different name any variable then do not award first mark but award subsequent marks as if correct name used.
Refer answers where candidate has used a loop to find position to insert item into stack to team leaders.

4

[13]

Q17.

(a) *Must have the concept of problem/task for the first mark*

A (step-by-step) description of how to complete a task / a description of a process that achieves some task / a sequence of steps that solve a problem /
A sequence of unambiguous instructions for solving a problem;
R Set of instructions
Independent of any programming language;
That can be completed in finite time;

Max 2

(b)

Answer	Count	Remainder
True	-	-
	2	1
	3	1
	4	3
	5	2
	6	1

Mark as follows:
answer column;
A True instead of blank cells
R if no evidence that dry run has been attempted
count column;

1 mark per correct value in remainder column;;;;

6

(c) Works out if x is a prime number //
Checks if x is divisible (with no remainder);

1

[9]

Q18.

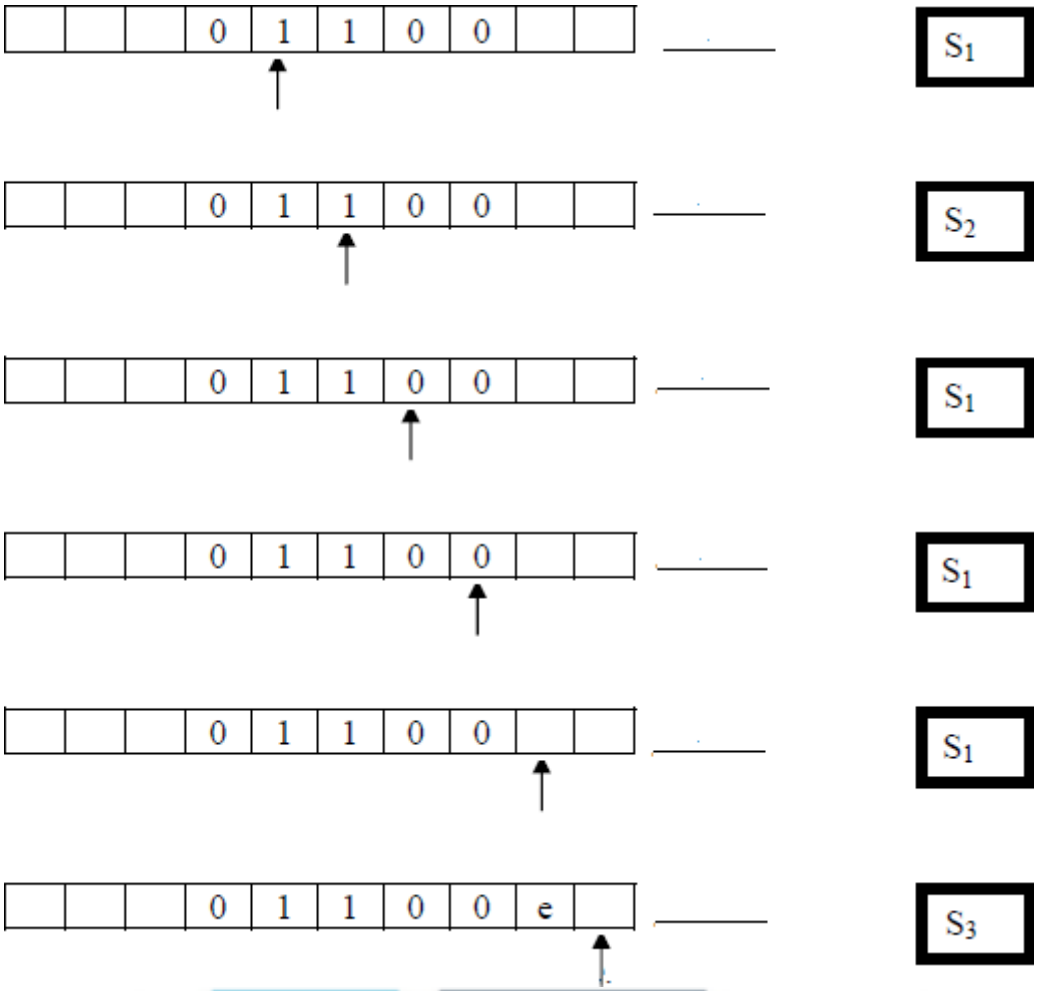
(a)

Number	Correct Label
❶	(0, 0, →)
❷	S_1
❸	(, o, →)
❹	S_3

1 mark for 1 and 3 correct – brackets not required
1 mark for 2 and 4 correct

2

(b)



Mark to end, do not stop at first mistake.
1 mark for first row correct;
1 mark for second row correct;
1 mark for both rows three and four correct;
1 mark for both rows five and six correct;
Must have correct tape contents and current state for each mark

A answers where the tape has been shifted
DPT for missing read/write head

4

(c) Check if the tape contains an even / odd number of 1s // check parity of number on tape;

1

(d) Turing machines provide a (general/formal) model of computation;
Provides a definition of what is computable // a task is computable if (and only if) it can be computed by a Turing machine;
No computing device can be more powerful than a Turing machine // any algorithm that can be computed by any computer can be computed by a Turing machine;
(The Church–Turing thesis states that) if an algorithm exists then there is an equivalent Turing machine for that algorithm // a Turing machine that can implement the algorithm;
Through the Halting Problem, can be used to prove that some functions cannot be computed;

Max 2

[9]

Q19.

(a)

				List				
ListLength	New	P	q	[1]	[2]	[3]	[4]	[5]
4	*38*	-	-	*9*	*21*	*49*	*107*	
		1						
		2						
		3						
			4					107
			3				49	
						38		
5								

4,5 in sequence for ListLength;
1,2,3 in sequence for p;
4,3 in sequence for q;
Final list in array is 9, 21, 38, 49, 107;

Do not award a mark if additional values indicated e.g. 4 for p

4

(b) Inserts an item/variable New into list at <u>correct position/preserving order//into sorted list (or equivalent)</u>;

1

(c) (i) Static structures have fixed (maximum) size whereas size of dynamic structures can change // Size of static structure fixed at compile-time

whereas size of dynamic structure can change at run–time;
Static structures can waste storage space/memory if the number of data items stored is small relative to the size of the structure whereas dynamic structures only take up the amount of storage space required for the actual data;
Dynamic data structures (typically) require memory to store pointer(s) to the next item(s) which static structures do not need;
A just one side of points, other side is by implication

Max 1

(ii) Heap is pool of free/unused/available memory;
Memory allocated/deallocated at run-time (to dynamic data structure);

Max 1

[7]

Q20.

(a) (i) Picture element // smallest resolvable/rectangular area/unit (**A** smallest dot) which can be drawn on screen // smallest addressable part/unit of a picture;
smallest unit which is mapped to memory;

1

(ii) Pixels are stored as numbers/bit patterns (**A** values) which represent different colours;
A or by example;

1

(b) 1;

1

(c) (picture / image) width;
(picture / image) height;
A (picture / image) dimensions
R size
image resolution / colour depth / No. of bits per pixel;
colour palette / No. of colours in image;
offset to the start of image data;
compression type;

Max 2

(d) (i) loop counter / (loop) control variable // array subscript/index;
array of Byte;
A array of Integer

2

(ii) 1101;
I. any additional leading 0's

1

(e) (i) ThisWidth;
X;

2

(ii) 2-dimensional array (of Byte);

1

(f)

ThisWidth	ThisHeight	Counter	X	Y	ThisByte		Final
8	5	0	1	1	255	[0]	25
				2	255	[1]	96
				3	255	[2]	96
				4	255	[3]	24
				5	255	[4]	24
				6	255	[5]	113
				7	255	[6]	
				8	255	[7]	
			2	1	255	[8]	
				2	25	[9]	
		1		3	25	[10]	
		2		4	96	[11]	
		3		5	96	[12]	
		4		6	24	[13]	
		5		7	24	[14]	
		6		8	113	[15]	

Mark as follows:
Counter has incremented from 0 to 6 (only);
X variable has incremented 1 and 2 (only);
Y variable has incremented 1 – 8 (only) at least once;
ThisByte contains first ten correct values;
Final[0] contains 25;
Final[1] to Final[5] are correct and with no other array subscripts used;

A correct six values (only) in Final array (in consecutive but wrong positions)

Max 6

(g) (i) program / constant / module / unit / user defined type / label /object / component / control / class;

A 'control' by example e.g. text box, drop down list
A any elements which are SQL specific

1

(ii) Maximum number of characters;
No punctuation characters;
No use of reserved words;
Must not start with a digit character;

case critical e.g. must start with lower case character;

A any answer which describes 'general' programming language restrictions.
identifier names must be unique;free-format not allowed for certain constructs, e.g. statement must not spread over two lines;
restrictions on identifiers used for labels;
loop control variable must be ordinal/integer;
array index range is restricted;
all variables must be pre-declared;

Max 2

[20]

Q21.

(a)
```
CarFailed : = False
Input NextCar
For Position ← 1 to 4
     Do
     NextCategory ← SingleCharacter( A. NextCar; , B. Position; )
     If C. NextCategory = '0'/ NextCategory <> '1';
     Then CarFailed ← True
```

Etc…
Part C – **I.** omission of quotes **A** double quotes

3

(b)

Variable	Data Type	Comment
Position	D **Integer**;	E **loop counter/loop control;** **Takes the range of value 1, 2, 3 and 4;** **Indicates the current test/category or suitable description;** **Provides an index for the string // indicates the position in the string;**
NextCar	String	
NextCategory	F **Char;**	
CarFailed	Boolean	

3

[6]

Q22.

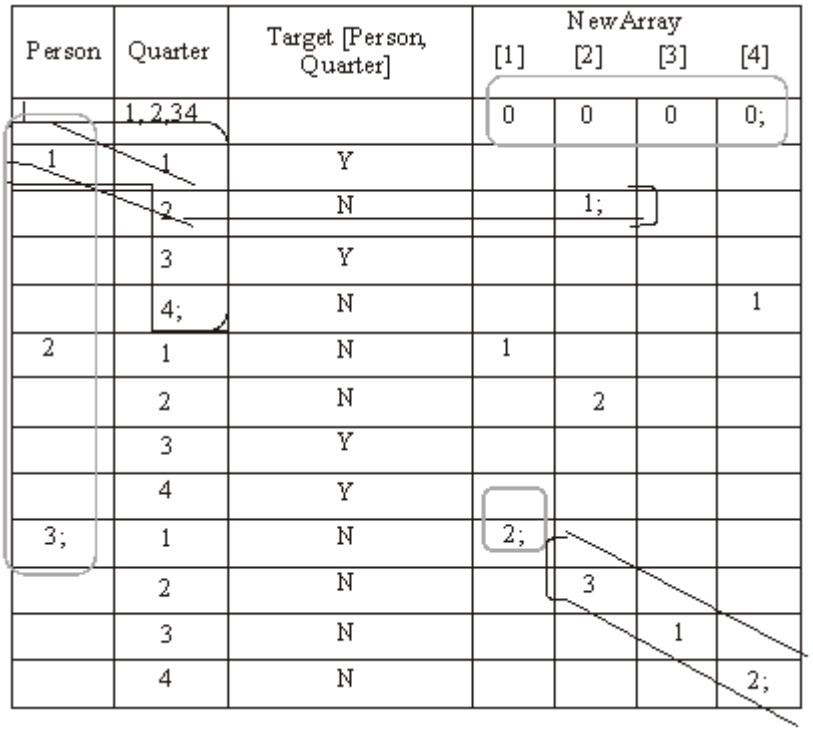
(a) 2–D array;

1

(b) Shows that sales person 2; did meet their target; for Quarter 3 /July – September;

Max 1

(c)



Person	Quarter	Target [Person, Quarter]	NewArray [1]	[2]	[3]	[4]
	1, 2,34		0	0	0	0;
1	1	Y				
	2	N		1;		
	3	Y				
	4;	N				1
2	1	N	1			
	2	N		2		
	3	Y				
	4	Y				
3;	1	N	2;			
	2	N		3		
	3	N			1	
	4	N				2;

NewArray initial values all 0;

1

Person loop counter 1 to 3;

1

Person 1 – is followed by quarters 1 to 4 in sequence;

1

NewArray[2] = 1 for person = 1 and Quarter = 2;

1

Final NewArray[1] = 2;

1

Final NewArray[2 and 3 and 4] values are correct;

1

(d) Stores the (total) number of sales staff who did not meet their target // the(total) number of sales targets not met;
for each quarter;

2

[10]

Q23.

(a) (i) String / Text / Char ;
R alpha / alpha-numeric / character

1

(ii) Integer / Date (and Time) ;
A String

1

(iii) Boolean ;
R Yes/No

1

(b) (i) Book ;

1

(ii) False / F / No // f/t from the (a) (iii) answer e.g. stated as integer - value 0/1

1

(iii) True / T / Yes // f/t from the (a) (iii) answer e.g. stated as integer - value 1/0
(Max 1 for (ii) and (iii) if no indication of meaning when integer used)

1

(c) (i) T76542 ; 1 ;

2

(ii) T ;
I. the quote marks (i) and (ii)

1

(iii)

NextAvailableCode	Book	LocationLetter
1	1	'T'
2	2	'T'
3	3	(gap not required)
4	4	'M'
(in sequence – possible repeat of 3 and/or 4	5	Penalty -1 if the first 'M' is followed by either 'T' or 'X'
	6	

Figure 2

	Location
[1]	'Torrington'
[2]	'Torrington'
[3]	
[4]	'Morristown'
[5]	

	NewCode
[1]	1
[2]	2
[3]	
[4]	3
[5]	

Figure 3 **Figure 4**

6

[15]

Q24.

(a) A procedure that is defined in terms of itself;
A A procedure that calls itself
R re-entrant

1

(b) Store return addresses;
Store parameters;
Store local variables/ return values;

Max 1

(c)

Number	Entry	Output
11	1	
11	2;	
11	3;	
11	4;	4;

4

(d) A linear search//
To find/output the position/index of Number in Items;

1

(e) Number is not an entry in Items// Stack overflows;

1

(f) Test for reaching the end of Items;

1

(g) Binary Search;
An iterative solution;

Max 1

[10]

Q25.

(a) *Any three from*
Procedures which have an interface / using parameters to pass values ;
Use of modules / use of libraries ;
Avoid global variables / use of local variables;
Meaningful identifier/variable/constant/ procedure / function / program / parameter names;
Consistent use of case for identifiers ;
Use of selection / loops / iteration ;
Avoid the use of GoTo structures ;

Effective use of white space / indentation;
R spacing/ space out the
Code
Use of named constants ;
Use of user-defined data types ;
Use of pseudo-code / top down approach / Jackson methodology / process Decomposition ;
R the use of comments/documentation
R declaration of variables

3

(b) (i)

Surname	String / Text ; **A.** String[n]
NoOfYearsService	Integer /Byte / Int / Short;
PayRate	Single / Real / Float / Currency;
BasicRate	Single/Real/Float / Currency;
AdditionalRate	Single / Real / Float / Currency;

Sensible name + correct data type for single mark

***BUT Penalise once** occurrence of names containing space/other illegal character(s) which would have scored*

Max 3

(ii) 3.1 If NoOfYearsService > 5 ;

1

A >= in the statement **R** =>
A mathematical notation
NoOfYearsService := 5 ;

1

A = or := or ←

3.2 PayRate := 7.88 + NoOfYearsService * 0.65

1

A £ symbol
R use of undefined/unassigned variable(s) in the calculation

A in words 'greater than', 'equals'

3

[9]

Q26.

(a) Calculates the total rejects for the week / calculates the total of array DailyRejects ;
Outputs the total rejects for the week ;
A Output the total only (if already mentions that calculates total rejects for the week)

2

(b) (i) RejectTotal := RejectTotal + DailyRejects[DayNo] ;
A ; may be omitted
A minor spelling errors

A omission of the subscript

1

(ii) RejectTotal: Integer //
DayNo : Integer //
DailyRejects : Array[1 ..7] of integer;
I. Dim …

Max 1

(iii) Loop counter / control the loop / Loop control variable / inference of a loop counter ;
Index/subscript for the array DailyRejects / reference the array elements ;
R days of the week

Max 1

(iv) Array of integers // array

1

(c) If RejectTotal > 7 ;
Then WriteLn ('Investigate')
Else WriteLn ('Inside weekly tolerance') ;
A reversed logic for both parts

2

(d) *Library program …*
Tried and tested routines should reduce the debugging time;
Evelopment time may be reduced ; **A** less code to write
Code can be dynamically loaded only when needed ;
Library files can be shared between different applications ;
A previously written/saved program code can be reused/
A program routines were previously saved/compiled ;
A program code is available and used from third party providers ;

Max 2

(e) (i) 3 / [3] / SupervisorTotal[3] := etc …..;

1

(ii)

WeekNo	ThisNumber	Output	SupervisorTotal [1]	SupervisorTotal [2]	SupervisorTotal [3]
			0	0	0
1	8	Investigate	1	0	
2	9	Investigate		1	
3	1				
4	8	Investigate		2	
5 ;	9 ;	Investigate ;	;	;	1 ;

6

[17]

Q27.

(a) Temp ← Front;
Front ← Temp.Next//Front ← Temp^.Next;

Dispose (Temp); **A** Free(Temp)

Alternative

Temp ← Front.Next// Temp ← Front^.Next;
Dispose (Front); **A** Free(Temp)
Front ← Temp;

3

(b) AddItem//Add;

1

(c) (i) Full/FullQueue;

1

(ii) No memory used for pointers;
I Faster
R Easier to program

1

(iii) Size is limited by array size;
memory wasted when not full;

2

[8]

Q28.

(a)

Number	Lower	Upper	Current
12	1	9	
		5	5
	3		3
	4	4	4

Value returned	4

1 mark for 1st row (12, 1, 9)
2 marks for second row (1 mark for each 5)
2 marks for 3rd row (3 and 3)
2 marks for 4th row (1 mark for Lower = 4, 1mark for upper = 4)
1 mark for correct return value

8

(b) Find the position of 12/ a number in the array// search for 12/ a number in the array;

1

[9]

Q29.

(a) Salesperson 7;
April /month 4;
The number of storecards 'taken out';

Max 2

(b) StoreCards + sensible subscripts [1..10, 1..6] / (1 to 10, 1 to 6) / [0..10, 0..6] / (0 to 10, 0 to 6) / (10,6) / [10] 6];
StoreCards + Integer / Byte;

2

(c) StoreCards (8, 1);
= 13 / := 13 / ← 13;
Must be an assignment statement

2

(d) Key in / Input the employee number; the program calculates the total number of store cards for a single person // print/outputs/displays the total for a single person; over six months;

Max 2

(e) (i) Single / real / float;
R Floating point / Double

1

(i) Boolean /Yes-No / True-False; **R** Y/N / T/F

1

(iii) Integer/ byte;

1

[11]

Q30.

(a) (i) 101 0110;

1

(ii) 1101 0110 / or follow-through from 'their 7 bit code' from (a);
A Parity bit positioned in bit position 0

1

(b) (i) 'D';
R Lower case

1

(ii) 'J';
R Lower case

1

(c) (i) FirstName // Surname;

1

(ii) Surname // FirstName; (i) and (ii) must be different

1

(iii) FullName;
I. any incorrect case

1

(d)

Position	NextNumber	NextChar	FinalString
			''

1	(65)	'A'	'A'
2	78	'N'	'AN'
3	32	<Space>	'AN '
4	69	'E'	'AN E'
5	82	'R'	'AN ER'
6	82	'R'	'AN ERR'
7	79	'O'	'AN ERRO'
8	82	'R'	'AN ERROR'

Position values incrementing to at least 4;to maximum 8;
NextNumber[2] has value 78;
Remaining NextNumber values are all correct and in correct positions;
NextChar[3] has <Space> character + NextNumber[3] is 32;
FinalString correct / f/t from their NextChar column;

6

[13]

Q31.

(a) (i) 271;

1

(ii) The required item might be the 271st one/last one/ not be present// Every item accessed;

1

(b) (i) 9;

1

(ii) Each comparison halves the number of items to be accessed//271 lies between 2^8 and 2^9;

1

(c)

Count1	Count2	Temp	A				
			[1]	[2]	[3]	[4]	[5]
-	-	-	23	45	16	12	31
1	1						
	2	45		16	45		
	3	45			12	45	
	4	45				31	45
2	1	23	16	23			
	2	23		12	23		
	3						
	4						
3	1	16	12	16			
	2						
	3						
	4						
4	1						
	2						
	3						
	4						

1 mark

1 mark

1 mark for Count1
1 mark for Count2
1 mark for Temp

5

(ii) (bubble) sort the items into ascending order;

1

(iii) Reduce the number of tests each pass// stop when no swaps occur during a pass//Add a flag No Swaps to indicate when no swaps occur// change loop control to Repeat until no swaps// sort variable sized array;

1

[11]

Q32.

(a) (i) Functions always return some value when called;
Procedures may return a value;
Functions appear in expressions;
Procedures do not appear in expressions;
Procedures name alone makes up the statement / call <name>

Max 2

(ii) Anything named which is plausible;
Examples could include: computation / formatting / string handling;
R software features / button events / DLL
A Dynamic Linked Library

2

(b) (i) True/Yes/ 1;

1

(ii) False/No/0;

1

(iii) Error;

1

(c) Program / constant / function / procedure / module / unit / user defined type / record / label / object /class;

Max 2

(d) **Advantage of an Interpreter:**

- Should allow faster/easier program development // faster/easier testing / debugging / finding errors;
- Correcting mistakes is less time consuming;

Max 1

Advantage of a compiler:

- The executable code/object code/program will run faster;
- Once the executable file has been produced no further action;
- Software distribution requires no further software to be available to the user;
- Prevents tampering of the code by users other than the developer;

Max 1

2

[11]

Q33.

(a) (i)
- poorly structured code;
- uses GoTo statements;
- the flow of control jumps out of a loop;
- nothing reported to the user when no matching name found;
- abbreviated variable for 'position' variable;
- ReadLn is better than Read;
- Program only iterates once / considers only the first array element;
- (if duplicates) only the first matching surname is found;
- (loop terminates at 20) does not allow for additional array /name entries;

A poor layout - excessive indentation used;

I. variable declaration // reference to the syntax

Max 2

(ii) All statements must have correct identifier name correct data type (String / Text // Integer / Byte / Word / Int / Shortint / Short as appropriate)

In addition, either array must have brackets to indicate an 'array' 19/20 to indicate a range;

Max 2

(b) Intialisation of counter or Boolean variable
P := 1 / P := 0 / For P := 1 to 20 // IsFound := False;

Looping
LOOP UNTIL // DO WHILE // WHILE DO // REPEAT UNTIL and used at the beginning/end of a code block as appropriate;

Some loop condition is met
(P = 20/21) OR IsFound = TRUE / P = 20/21 // IsFound = TRUE / IsFound;

IF with use of the array
IF NoOfClaims [P];

Selection condition
>4 / >=5;

Loop counter incremented
P = P+1

Final output
Correct logic followed with OUTPUT 'Yes'
A multiple times

Final output
Correct logic followed with OUTPUT 'No'
R Multiple times
R 'Prose' scores 0

5

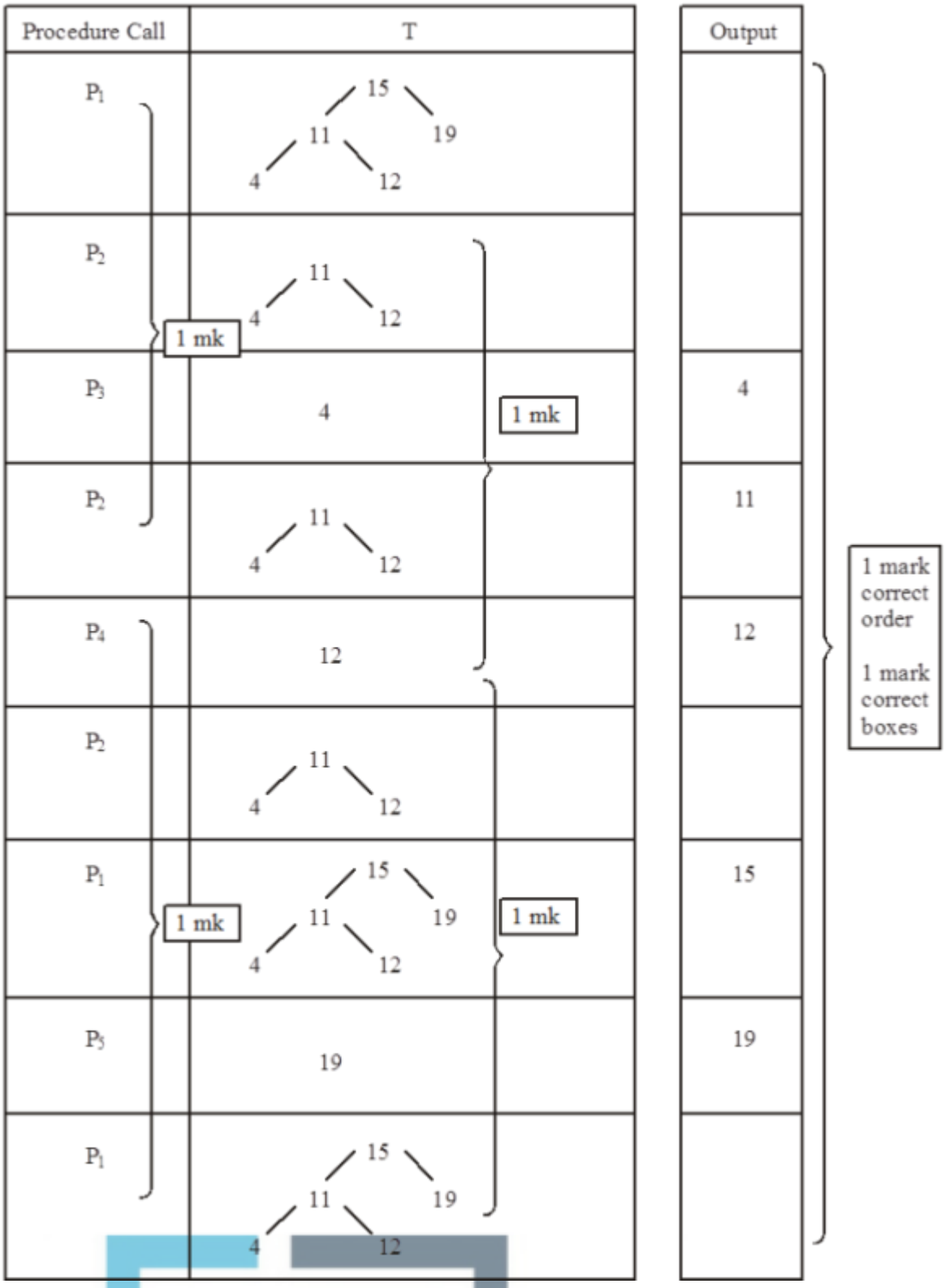
[9]

Q34.

(a) A procedure/routine that calls itself/ is defined in terms of itself;
A Function instead of procedure
R re-entrant
R program
R iteration

1

(b) (i)



6

(ii) In order; (tree) traversal

2

[9]

Q35.

(a) *1 mark for each correct entry*

New	Last	Ptr	Values				
			[1]	[2] **max**	[3] **6**	[4]	[5]
6	3	1	4	7 **max**	9 **1**		
		2					
	2					9	
	1				7		
				6			

6

(b) Insert 6/a value into the array/ in the correct position;

1

[7]

Q1.

Only about 20% of students were able to solve the logic puzzle and obtain both marks, with a significant number of students removing multiple vegetables rather than just one as specified in the question. Some students correctly stated that they would take a vegetable from the box labelled as onions and carrots but then gave an example rather than a general explanation or an explanation of both possible results, e.g. explained what it would mean if the vegetable was a carrot but not what it would mean if it was an onion.

Q2.

In previous years there have been questions asking students to complete an adjacency matrix based on a diagram of a graph and most students were able to answer question (a) this year. This was the first time that an adjacency matrix for a weighted graph had been asked for and some students had clearly not seen this type of question before and only included an indicator that there was an edge between two nodes rather than the weight of the edge between the two nodes; this meant they only got one of the two marks available for this question.

Questions (b)-(d) were about graph theory. Question (c) was well-answered with students identifying that it was not a tree because there were cycles. The most common incorrect answer was to say that it wasn't a tree because the edges have weights associated with them. Question (d) was also well-answered. Answers to (b) often showed that students were not as familiar with adjacency lists as they are with adjacency matrices.

For question (e) students had to complete a trace of Djikstra's Algorithm. This topic was not on the previous A-level specification and was often poorly answered suggesting many students had not tried to complete a trace for the algorithm before. For question (f) many students gave an answer that explained the point of Djikstra's Algorithm (find the shortest route from a node to each other node) rather than what the specific output in the algorithm given in the question would be (the distance of the shortest route from node 1 to node 6).

Q3.

Most students could explain what was meant by a recursive subroutine though some answers showed that the difference between iteration and recursion was not always understood. The trace was reasonably well done with the most common error being to include additional function calls or outputs in the table.

Q7.

(a) This part was well tackled with over two thirds of candidates recognising that a queue was an appropriate data structure to represent the deck of cards because it was a first-in-first out structure.

(b) (i) Just over half of the candidates were able to correctly update the pointers to the queue in this part

(ii) And just under half were able to do likewise in this part. In the former, the most common mistakes were to change FrontPointer to 10 instead of 11 and to leave QueueSize at 52. In the latter, the most common mistake was to change RearPointer to 54, failing to recognise that this was a circular queue and RearPointer should change to 2.

(iii) Students achieved a broad range of marks on this part which required them to write an algorithm for dealing a card. Over two thirds got some marks, but just under 10% scored full marks. The most commonly made mistakes were to try to deal more than one card, to fail to wrap FrontPointer back to the start of the array when it went past the end of it, and to deal a card when the deck was empty.

(c) This part was well answered, with most students recognising that in an event-driven program, subroutines would be associated with events, such as a button being clicked or a timer reaching 0, and that the appropriate subroutine would be called when the event occurred. Some responses were not considered creditworthy as they could have applied to any style of programming, for example “the program waits for user input before executing instructions”.

(d) Approximately two thirds of students achieved some marks for comparing a mobile phone operating system and a desktop operating system in this part of the question. A common mistake was to compare the devices rather than the operating systems, for example stating that the desktop would have more memory that the mobile phone. Many students explained that the mobile phone OS would have lower hardware requirements but it would have been nice to see some more concrete examples of these reduced requirements, and phrases such as “smaller” and “more lightweight” were used too often. Some students gave responses to a question from an earlier paper that were not appropriate for this scenario.

Q8.

(a) More than half of the candidates were able to correctly define abstraction in this part. The question was asked in the context of data representation, so an appropriate definition would have been to store only those details of a problem / model that were required in the context of the problem being solved. Common mistakes were to state that unnecessary complexity would be hidden from the user, rather than being removed from the model altogether, or to define a simulation instead of abstraction.

(b) This part was the question that assessed quality of written communication. As such, it was particularly important the candidates used technical terms accurately when writing their responses. Almost all candidates covered both aspects of the question: the representation as a graph and the representation using arrays as an adjacency matrix or list.

In almost all responses, the graph part of the answer was better than the array representation part, with most students being able to correctly explain how the underground railway network could be represented as a graph. Some students lost marks by not using the correct terminology or by drawing diagrams but not explaining points such as that a station would be represented as a node.

As the question paper stated, diagrams needed to be fully explained if they were used. A common mistake was to state that stations would be represented by nodes and that railway tracks would be represented by vertices; vertex in a synonym for node but was being mistakenly used instead of the term edge or arc. A small but not insignificant number of students misinterpreted the term graph and explained how the network might be represented as a bar chart or scatter graph.

The part of the question relating to the representation using arrays as an adjacency matrix or list was poorly tackled. Many students discussed how the representation could be drawn out as a grid or list on paper instead of how it could be represented using arrays in a programming language, or described solutions that were not really

either an adjacency matrix or list.

Those who decided to represent the data as an adjacency matrix often suggested that the station names would be written into the array rather than that stations would be associated with a number that would be used as an index into the array. That said, pleasingly, most students who suggested a matrix representation recognised that the distances would be filled into the array at the appropriate location and that a null value would be used where there was no route. Some students used an array containing only 0s and 1s instead of distances.

Students who chose to present an adjacency list solution rarely explained how this would be achieved in a programming language, focussing instead on how it could be depicted diagrammatically. Another common mistake was to talk about pointers, without explaining how these would work in the context of the array.

Q9.

For (a), about three quarters of the candidates achieved one or two marks for this question part. The most common creditworthy response was that the size of a dynamic data structure could change at run time whereas a static data structure's size was fixed at compile time. Some candidates missed out on this mark by making the vaguer statement that a dynamic structure could change at run time whereas a static structure could not, without any reference to size. Some candidates lost a mark by just giving opposite sides of the same point.

Question (b) was well answered. Some candidates missed out on one mark scheme point by writing that all elements in the array would have to be moved down, without making clear that it was only the elements below the insertion point that needed to move. Many of this group of candidates nevertheless achieved full marks by making other valid points. A small but surprising number of students wrote about deleting an item when the question was about inserting one.

For part (c), just over half of the candidates correctly identified that this was a priority queue.

For part (d)(i), only about a quarter of the candidates offered a reasonable explanation of what the heap would be used for. Many stated that it would be used to store the list or to store pointers. Some asserted that it would be used to store new items, which was a better response but was not creditworthy as the suggestion was that the new items would be stored in the heap. Good responses recognised that when a new item was added, memory would be claimed from the heap to store the new item.

Part (d)(ii) was poorly answered with only about a third of the candidates recognising that the pointer value was a memory address. Answers referring to a value being an index or location were not enough for a mark without further explanation that made clear that this was to a memory location.

In part (d)(iii), this algorithm was the most complex one that candidates have been asked to write on a COMP3 paper. Answers given in pseudo-code or structured English were acceptable and all reasonable styles of syntax were marked as we are aware that candidates will have used different programming languages. Responses written in prose were acceptable if they included clear steps and showed how the variables would be updated. There were very few prose answers, but they often scored poorly due to being too general.

There were some very good responses and marks were awarded later on in the algorithms even if earlier parts were not working, although this was not always possible as

sometimes whether or not later parts of the given algorithm worked depended fundamentally on earlier parts.

The most common misconception was to treat the linked list as if it were a list stored at sequential locations in an array and to adjust the CurrentNodePointer by adding one onto it inside a loop rather than by setting its value to the NextNodePointer of the current node. It remained possible to pick up some marks for this type of implementation by, for example having a suitable loop.

Very few candidates, even those achieving almost full marks, included the step of actually claiming the memory from the heap to store the new item in.

A small number of candidates produced recursive solutions which were also accepted.

Q10.

For the first time a flowchart was used to represent an algorithm in a COMP1 exam. There was no increase in difficulty resulting from this and the standard of answers was the same as seen in the previous year.

Some students did not follow the algorithm given and instead developed their own program to convert binary to denary. This resulted in them not getting many marks as they had not answered the question.

Students using VB6 tended to get lower marks on this question than those using the other languages available for COMP1. This was partly due to not providing the correct evidence for the testing (screen captures needed to show the data entered for the test as well as the result of the test), although many students using VB6 also seemed to have weaker programming skills.

Students need to be aware that an algorithm is not the same as a program and that simply copying the algorithm into their development environment will not result in a working program in any of the COMP1 programming languages – the pseudo–code/flowchart needs to be adapted to match the syntax of the programming language they are using. As in previous years, a number of students simply copied parts of the algorithm into their program code eg trying to use a keyword of OUTPUT. These appeared to be less able students who generally struggled on the Section D programming as well. The vast majority of students were able to convert the algorithm successfully into working program code and the marks obtained on this question were virtually identical to those achieved on Section B on the 2011 COMP1 exam.

Q11.

Answers to this section were often of poor quality and very few students achieved good marks on this question.

A number of students are still including additional code when asked for the name of an identifier. This means that they are not getting the marks for these questions as they have not made it clear which entity is the identifier (sometimes there is more than one identifier in lines of code that they have copied from the Skeleton Program). To reduce the chance of errors, when asked to give the name of an identifier students should be encouraged to copy and paste the identifier from the Skeleton Program, rather than typing the identifier into the EAD.

Very few students showed any understanding of binary files, even though these were used in the Skeleton Program. Part (a) was answered better than most other parts of Section C with most students able to give at least one reason why the use of global

variables should be avoided. The majority of students were also able to state an advantage of using a named constant.

Q12.

Part (a): This question part was well answered, with most students correctly identifying that space complexity was the second measure of complexity (in addition to time complexity).

Part (b)(i): In this question part, students were required to carry out a trace of a simple algorithm for looking for duplicate values in a file. The primary purpose of this question part was to give students the opportunity to study the algorithm so that they could discuss its complexity in question parts (b)(ii) and b(iii). The majority of students achieved all three marks for fully completing the trace table. However, disappointingly, approximately a third of students scored no marks at all for what was a relatively straightforward trace to complete.

Part (b)(ii): Pleasingly, approximately two thirds of students recognised that the algorithm had order of time complexity O($n^{2)}$.

Part (b)(iii): Students justified the order of time complexity as being O(n^2) in various ways. Some did it by analysing the code and recognising that there was a for loop nested inside another for loop, with each loop repeating n times. Others did it by considering the number of comparisons made and recognising that each of the n items in the file would be compared to each other item, resulting in n^2 comparisons being made. The most commonly made mistake was to refer to there being two loops in the algorithm, without making clear that these were nested. Some responses were too superficial to be creditworthy, failing to relate the complexity to the steps involved in the algorithm.

Q13.

Part (a): Two thirds of students were able to identify one property that a graph must have to be a tree. A small number confused a tree with a rooted tree and made assertions such as that a tree must have a root, which is incorrect.

Part (b): This question part tested students' understanding of the method being used to represent a maze as a graph. The majority of students correctly identified a feature of the maze that would stop its graph being a tree. The most commonly seen correct response identified that there could be a loop in the maze. Other possibilities included that part of the maze could be inaccessible or that part of the maze might only be traversable in one direction. Some students failed to achieve the mark because they re-answered part (a), discussing a feature of a graph that would stop it being a tree, rather than a feature of a maze.

Part (c): Students were asked to represent the graph of the maze as an adjacency matrix. Three quarters of students scored both marks for this question part. Responses where symbols other than 0s and 1s were used in the matrix were accepted, as long as they could be viewed as an accurate representation of the graph.

Part (d)(i): The vast majority of students were able to identify that a recursive routine would call itself. A small number asserted that a recursive routine would repeat itself, which was not considered to be enough for a mark as this could equally have been a description of iteration.

Part (d)(ii): Most students scored some marks for this question part, but less than a fifth achieved both. The most widely understood point was that the data would need to be removed from the stack in the reverse of the order that it was put onto it so that the

recursion could be unwound. Less well understood was the types of data that would be stored, such as return addresses and local variables.

Part (e): Most students achieved some marks on this question part and around a quarter achieved all five for a fully complete trace. The most commonly made mistake was to update, incorrectly, the Completely Explored array as the recursive calls were made, as opposed to when the recursion unwound.

Q14.

This was the first COMP1 question in which candidates were asked about the theory of problem solving. Most candidates received good marks and were able to identify the goal and three resources that were available. Far fewer candidates were able to define clearly what is meant by the ownership of a problem with many stating that the owner was the person affected by the problem, rather than the person responsible for solving the problem. Another frequently seen incorrect response was to write who had ownership of the problem described in the question (Bob), rather than to define ownership.

Q15.

Most candidates were not well prepared for this section and did not do as well on these questions about the Skeleton Program as they did on the questions where they were asked to modify the Skeleton Program. In particular, little understanding of structure charts or decision tables was shown by a significant number of candidates.

It was pleasing to note that most candidates only gave the name of an identifier when asked to do so – those who copied and pasted sections of code from the Skeleton Program did not get the marks for these questions as they had not demonstrated that they understood what an identifier is (some candidates gave answers that contained multiple identifiers). Some candidates did not get the mark for giving an example of a constant declaration as they provided only the name of the constant. Candidates should ensure that when asked for the name of an identifier they provide only the identifier in their answer and when asked for an example of a type of program statement that the entire program statement is given in their answer.

For part (n) many candidates described the repetition structure rather than the selection structure inside the repetition structure.

Q16.

Part (a): This question part was very well answered with the majority of candidates getting both marks. The only common mistake was to miss out the brackets in the expression that should be (12+19)*8.

Part (b): As with part (a), this question part was also well answered. The most common correct response was that brackets are not required. It would have been nice to see some more detailed explanations of this point, rather than just a brief statement of it. A common incorrect answer was that RPN was easier for a computer to understand. The word “understand” is not appropriate in this context.

Part (c): Responses to this question part were excellent, with relatively few errors made. The majority of candidates got full marks which is unusual for a question involving a trace table. The only two recurring mistakes were to pop the numbers off the stack in the wrong order, resulting in the transposition of the values in Op1 and Op2 and forgetting to push 50 back onto the stack at the very end.

Part (d): This question was well answered, with most candidates getting some marks and

a significant number more than half marks. The most common mistake was to increment the `TopOfStackPointer` in the wrong place – either before the If construct which tested for the stack full condition or after the value in `ANumber` was stored into the `StackArray`. Some candidates implemented solutions that used a loop to find the first empty position in the array to insert the number into. These were awarded credit if they would have worked, but many failed to test properly for the stack being full. It is important that candidates use the correct variable names when they are given on the question paper.

Q17.

The definition of an algorithm was often unclear. Many candidates described an algorithm as being a set of instructions – a set implies that there is no order to the instructions – without qualifying with, "executed step-by-step". To get full marks candidates had to clearly get across that an algorithm is a sequence of instructions that describe how to solve a problem and that these instructions are independent of any programming language.

The dry run was attempted by most candidates and many got full marks. The MOD operator was not understood by some candidates even though it is explicitly stated in the specification and a reminder of what it does was included in the question. Fewer candidates were able to identify the purpose of the algorithm.

Q18.

Part (a): The vast majority of candidates scored both marks for this question. Candidates who only scored one mark usually named the states correctly.

Part (b): Most candidates gained some of the four available marks and over half gained all four.
Some chose to keep the position of the tape head fixed and move the tape, rather than vice-versa which was perfectly acceptable.

Part (c): The Turing machine outputted 'e' if the tape contained an even number of ones and ‘o’ if the number of ones was odd. Determining this was a difficult task given the limited trace that candidates were asked to complete. Nevertheless a quarter of candidates were able to do this.
Many who did not get the correct answer had managed to understand that the use of the Turing machine related to evenness but that this was whether the number itself was odd or even. A commonly made mistake was to assume that the output of the Turing machine depended only on whether the last digit read was a 0 or 1.

Part (d): It was pleasing to see that many candidates understood that a problem is computable if and only if it can be computed by a Turing machine. Some went on to make a further point, such as that no computer could be more powerful than a Turing machine, but answers scoring both marks were rare.

Q19.

Part (a): This question part was very well answered with the many candidates getting full marks.

Part (b): The algorithm inserted an item into an ordered list at the correct position. Some candidates lost marks by failing to refer to the ‘correct position’. Others stated that a sort was performed which was not true, as the order of the items in the existing list was not changed.

Part (c): For part (i), around half of the candidates were able to explain that a static data structure has a fixed size which could not change at compile time, whereas the size of a dynamic data structure can change at runtime.

There were few correct answers to part (ii). The heap is a pool of available memory locationsand as the linked list grew, memory would be allocated to it from the heap and deallocated memory would be returned to it. Some candidates erroneously believed that the heap would be used as a temporary store or that the pointers in the linked list would be stored in the heap.

Q20.

(a) See earlier comment in the General section of this Report.

(b) All that was required in this question was the association between a number value and a colour, and hence that different numbers are used to represent different colours. The suspicion was that the candidates were not clear of the meaning of the word 'encoding' in the question stem. Some candidates described the idea that the picture was formed by putting together many pixels. A common misconception was that the pixel value stored its location.

(c) A common wrong answer – as seen in a previous examination – described 'data which is stored in the file directory' (not the file header).

(d) (i) Very poorly answered, despite a very similar question on a recent January series question paper.

(e) (i) Well answered.

(ii) Often candidates latched on to the term 'data structure' and then chose from the stack, queue options, failing to appreciate that an array is referred to as a data structure.

(f) On the one previous question paper on which the algorithm trace used a nested loop, the quality of answers seen was encouraging and the Report commented on this. Alas, the impetus was not maintained, and the number of candidates who were able to score 5 or 6 marks was small. The common error on the better scripts was not to make the final increment of the Counter variable value 6.

(g) (i) Most candidates came up with a valid answer from the large range deemed acceptable.

(ii) Many candidates were able to come up with two restrictions on the choice of identifier names. Some scored 1 mark only by quoting two near identical reasons e.g. cannot contain a 'comma' character followed by 'cannot contain a question mark' character. Some candidates answered their own question e.g. 'cannot store a text character in a integer data type variable'. Other candidates read the question as 'general restrictions' of the programming language and so gave answers such as 'variables must be declared before they can be used,' Answers of this nature were given credit.

Q21.

(a) The algorithm given contained a single loop, one selection statement and a function. However, the majority of candidates were unable to correctly complete the given algorithm, which begs the question - how many candidates would be able to write a similar algorithm from scratch?

(b) The candidates were given for the first time a specific list of data types from which to choose. This approach is likely to be used again. If other data types are to be used in future questions e.g. Date, then they would clearly be included in a given table.

Candidates did better for D and F, although securing a mark for the purpose of the Position identifier for E was more elusive.

Q22.

It has been commented on in previous papers how the standard of candidates' answers for this question has improved and this continued with this paper.

In part (c) a nested loop (given for the first time) was the key control structure and the majority of candidates identified this. Some confused the nesting, although were still able to secure 5 of the available 6 marks. The most commonly lost mark was at the start of the algorithm for the failure to initialise the array.

Generally part (d) was well answered. Where a mark was lost the suspicion was that it was through poor expression, not lack of understanding of the algorithm. Incorrect answers often stated the numbers for individual sales staff or started to include the concept of target setting, etc.

Q23.

In general the dry run was poorly answered and left completely blank on too many scripts.

(a) Many candidates scored the maximum three marks for identifying the data types. Some candidates lost a mark for suggesting that 'yes/no' or a 'check box' was an acceptable data type. This comes from their practical experience with database design software and a visual programming language, but candidates should appreciate they are not acceptable names for programming language data types.

(b) This was a different style of question from that previously seen. Candidates seemed to cope well with being asked to 'fill in the blanks' in the algorithm.

(c) (ii) Answers were often incorrect, but then inexplicably candidates were able to use the same function correctly in part (iii).

Q24.

Candidates generally scored well on this question. Recursively-defined was well understood although many candidates were unable to describe the use of the stack well enough. It was pleasing to see the majority of candidates obtaining most of the marks on part (c). Candidates often failed to obtain the mark for part (d) due to inadequate descriptions. Although many candidates provided a situation where the algorithm will fail, fewer were able to suggest a suitable modification. Once again this was often due to an inability to express themselves well. A wide range of answers were supplied for part (g) but a substantial number of correct responses were given.

Q25.

(a) Despite an extensive (and perhaps 'generous') mark scheme list it was rare for candidates to score more than 1 mark, and this was usually for a "selection/iteration" answer.

(b) (i) Candidates often failed to score three easy marks. The inclusion of <Space>

or other illegal characters used in the identifier names was penalised once only. The other common error was the suggestion of incorrect data types, the most common being 'Number' and 'Decimal'. However, this was answered significantly better than on previous papers.

(ii) Despite a question of this type not having been set previously, it was clear from answers seen that candidates knew what was required. The most common error was simply not to make the connection between part (b)(i) and (b)(ii); for example, by introducing new identifiers to answer (ii) which gained no credit.

Q26.

A general observation was that candidates scored significantly better with tracing the algorithm than with the first part of the question where they were asked to recognise various components of the given program.

(a) Almost all candidates got the idea that the program was calculating a weekly total. Very few stated for the second mark that it output the result.

(b) (i) A common error was to copy the first assignment statement which appeared, ignoring the rubric that it should 'perform a calculation'.

(ii) A common error was the statements that RejectTotal:=0 was a declaration statement.

(iii) Very few answers scored here. The most common (wrong) answer was that it represented the day of the week.

(c) This should have been an easy two marks. Common errors were for candidates to introduce their own output messages, or to use incorrect logic; typically where the equality condition produced both messages.

A wide variety of answers were considered acceptable including the use of two separate IF statements.

(d) This is only the second paper on which an explanation of the use of library programs was required and it is clearly still not well understood. The most common correct answers were that library programs are pre-written code which has the potential for reuse or code which is purchased from 3rd party suppliers. Such answers were however rare and there were far too many vague answers with statements such as "their use will make life easier for the programmer".

(e) An encouraging sign on this paper, continuing on from June 2006, is much improved answers seen for the trace table question, especially as this question contained a procedure which had not appeared in previous questions.

Q27.

This question discovered that most candidates did not really understand the concept of a queue as a data structure. Throughout the question many candidates referred to insert/delete operations thus showing a lack of understanding of the operation of a queue. Part (a) was quite beyond the ability of most candidates and this suggested that they had never attempted to program a queue using a linked data structure. Very few candidates recognised that the value of Front had to be stored in a temporary variable in order to free the memory used. The answers to part (c) suggested that they had not programmed a queue using an array either. Most candidates, however, managed to obtain some marks

on this question, most frequently parts (c)(i) and (c)(ii). Many marks were lost in part (c)(iii) due to inadequate descriptions.

Q28.

Very few candidates did not get some marks for the trace and many returned full marks for this part of the question. Some candidates who did achieve full marks on part (a) could not say what the algorithm does. Many candidates seemed to make a wild guess.

Q29.

This was the first question paper on which two-dimensional arrays had been set and the answers seen were encouraging.

(a) Most candidates correctly described that this was the issues figure for salesperson 7 in month 4. Some candidates described the figure as the highest sales figure for April which gained no credit.

(b) Only better candidates wrote an acceptable declaration statement which required the correct identifier StoreCards with the correct subscripts in the correct order.

(c) Few acceptable statements were seen.

(d) Encouragingly, this was well answered, with most candidates able to describe the purpose of the algorithm. Answers which did little more that re-write statement(s) from the given algorithm into a narrative form - e.g. “person total set to zero” - which was little different, did not gain credit. The common error was stating that the algorithm calculated a total for ‘each’ salesperson.

(e) Somewhat surprisingly – despite similar questions on previous papers - candidates were often unable to state a correct data type, which would suggest the fundamental concept in programming that “identifiers will have a stated or implied data type” is not understood.

For (ii) almost all gave Boolean, with every possible phonetic spelling, and some gave integer for (iii). Real/Float or other acceptable alternatives for (i) were rare.

Q30.

(a) Most candidates were able to write the correct binary pattern and successfully add a correct parity bit. Candidates need reminding that it is normal practice to use bit position 7 (the left-most bit) for the parity bit.

(b) (i) Most candidates wrote character ‘D’.

(ii) Some candidates left the answer as 74 failing to use the Chr( ) function as the final step.

(c) All candidates scored well for the structure chart appreciating that such a diagram can be used to represent the interface of a function. The mark scheme was generous in allowing a mixture of case used in the identifier labels in the diagram. This might not gain credit in future papers.

(d) This question was well answered by the majority of candidates - possibly as there had been a ‘build up’ to the algorithm in the earlier parts of the question.

This is an encouraging trend as the algorithm contained a loop, the use of functions,

the use of arrays (which had been poorly tackled on previous papers), yet candidates scored highly.

Q31.

It was pleasing to see many good answers to parts (a) and (b) although a number of candidates failed to obtain full marks through inadequate explanations. Part (c) was disappointing with few candidates completing the trace table correctly. Nevertheless it was pleasing to see a greater number of candidates able to partially complete a dry run. A surprising number of candidates were able to state that this was a bubble sort even though they failed to complete the trace table. Fewer were able to give a suitable improvement. The most common incorrect suggestion was to "make the algorithm recursive".

Q32.

(a) Many candidates were able to explain that functions always return a value but few candidates were able to distinguish this from the way a procedure behaves.

For candidates who had covered this theory in a practical context this was an easy two marks. Candidates should have been exposed to a subset of the functions available in their programming language. The final part of the question stem "…or when using a generic software package" was intended to help the weaker candidates in triggering some of the functions they would have used; unfortunately, candidates often gave answers describing features of a generic software package.

(b) This question was generally well answered, although it was noticeable that the standard of answers varied between centres. Candidates who found the question easy were undoubtedly those who had practical experience of using functions which required none, one or two parameters when used.

(c) The most popular answer was to use identifier names for constants, followed by procedures and functions.

(d) This was well answered with most candidates able to score marks. The key word in the question stem was "advantage" and so answers required more than just a description of a compiler and an interpreter.

Q33.

(a) (i) The use of GoTo statements has not previously been examined on this paper and most candidates struggled to suggest a single reason why this was poorly designed code, despite a large number of acceptable answers. The most common correct answers were that the use of GoTo statements gives rise to code which is difficult to follow and trace; there is no output produced when the SearchName value is not found; when there is more than one occurrence of SearchName in the PolicyHolder array, the program will output the number of claims value for the first occurrence of the name only.

(ii) Few marks were obtained here with most candidates failing to give the bounds of the array for PolicyHolder or NoOfClaims, or omitting a data type for the identifier.

(b) Candidates should be able to write small amounts of program code in a unit that has the word 'programming' in its title. Knowledge of loops other than a For loop was rare. It was hoped that candidates would have constructed a Repeat – Until or While loop which terminated when a NoOfClaims value of 5 or more was found.

Candidates who used a For loop were, however, still able to score the maximum 5 marks.

Examiners were not looking for the correct use of exact syntax for the language as stated by the candidate.

The use of IF statements was better understood, but this often did not extend to using an array index for the NoOfClaims as part of the IF statement. Very many candidates used the maths operator incorrectly, e.g. ≥ or more usually =>. Quite a few candidates reversed the logic testing for <5 and gave appropriate output for which they gained marks. Most popular languages seen were Pascal and Visual Basic but the candidates that used C on the whole answered the question very well indeed.

Q34.

This was another question which most candidates found difficult, if not impossible. However, some good candidates produced very good answers.

Most candidates were able to answer part (a).

The examiners only rarely awarded full marks for the trace table. A lot of candidates abandoned the trace once they realised that the numbers were being output in ascending order. This limited their reward to two or three marks at best since half of the marks depended on the trace being completed. Many candidates had difficulty logging the procedure calls even when they made a good attempt at showing the tree in the T column.

Some candidates got the two marks for part (b)(ii) without attempting the trace while others who showed the right output in (i) called the procedure a search or a bubble sort.

Q35.

(a) This question was very badly answered. Many candidates failed to attempt the question at all and, of those that did, there were very few correct answers.

(b) As so few candidates were able to answer part (a) it was not surprising that very few acceptable responses were received for part (b). What was surprising was the number of candidates who were able work through the algorithm to answer part (a) but were unable to describe its purpose.