1.1 Programming part 1

Name: _____

Class: _____

Date: _____

Time: **641 minutes**

Marks: **448 marks**

Comments:

Q1.

Stacks are also used to store a stack frame each time a subroutine call is made.
State **two** components of a stack frame.

_____

_____

_____

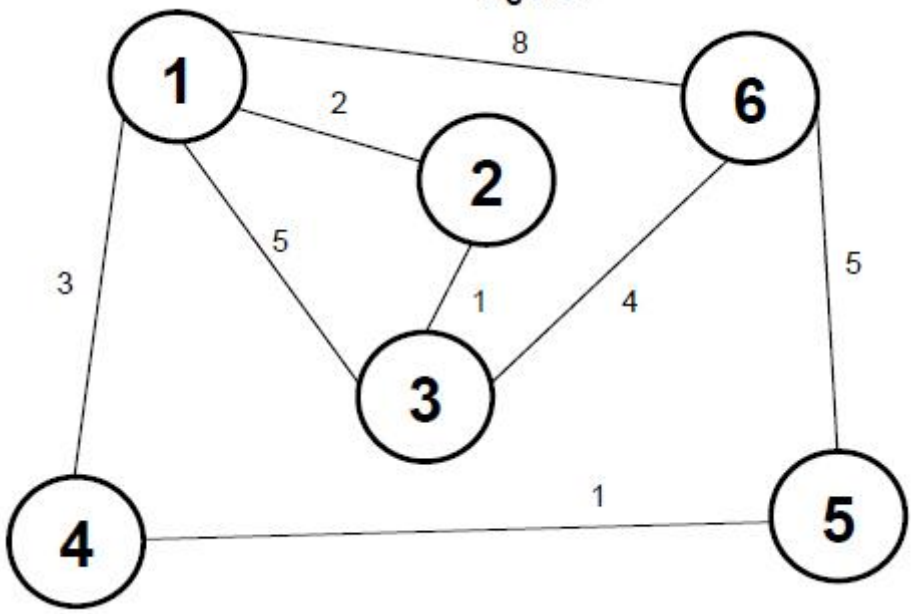
_____

(Total 2 marks)

Q2.

Figure 1 is a graph that shows the time it takes to travel between six locations in a warehouse. The six locations have been labelled with the numbers 1 - 6. When there is no edge between two nodes in the graph this means that it is not possible to travel directly between those two locations. When there is an edge between two nodes in the graph the edge is labelled with the time (in minutes) it takes to travel between the two locations represented by the nodes.

Figure 1



(a) The graph is represented using an adjacency matrix, with the value 0 being used to indicate that there is no edge between two nodes in the graph.

A value should be written in every cell.

Complete the unshaded cells in **Table 1** so that it shows the adjacency matrix for **Figure 1**.

Table 1

1	2	3	4	5	6

1						
2						
3						
4						
5						
6						

(2)

(b) Instead of using an adjacency matrix, an adjacency list could be used to represent the graph. Explain the circumstances in which it would be more appropriate to use an adjacency list instead of an adjacency matrix.

_____

_____

_____

_____

(2)

(c) State **one** reason why the graph shown in **Figure 1** is **not** a tree.

_____

_____

(1)

(d) The graph in **Figure 1** is a weighted graph. Explain what is meant by a **weighted graph**.

_____

_____

(1)

Figure 2 contains pseudo-code for a version of Djikstra's algorithm used with the graph in **Figure 1**.

`Q` is a priority queue which stores nodes from the graph, maintained in an order based on the values in array `D`. The reordering of `Q` is performed automatically when a value in `D` is changed.

`AM` is the name given to the adjacency matrix for the graph represented in **Figure 1**.

Figure 2

```
Q ← empty queue
FOR C1 ← 1 TO 6
  D[C1] ← 20
```

```
    P[C1] ← -1
    ADD C1 TO Q
  ENDFOR

  D[1] ← 0
  WHILE Q NOT EMPTY

    U ← get next node from Q
    remove U from Q
    FOR EACH V IN Q WHERE AM[U, V] > 0

      A ← D[U] + AM[U, V]
      IF A < D[V] THEN

        D[V] ← A

        P[V] ← U
      ENDIF
    ENDFOR
  ENDWHILE
  OUTPUT D[6]
```

(e) Complete the unshaded cells of **Table 2** to show the result of tracing the algorithm shown in **Figure 2**. Some of the trace, including the maintenance of Q, has already been completed for you.

Table 2

				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													
		3													
		4													
		6													
2	3,4,5,6	3													
3	4,5,6	6													
4	5,6	5													
5	6	6													
6	-														

(7)

(f) What does the output from the algorithm in **Figure 2** represent?

_____

_____

(1)

(g) The contents of the array P were changed by the algorithm. What is the purpose of

the array `P`?

_____

_____

_____

_____

(2)

(Total 16 marks)

Q3.

Write a program that checks which numbers from a series of numbers entered by the user are prime numbers.

The program should get a number from the user and then display the messages:

- `"Not greater than 1"` if the number entered is 1 or less
- `"Is prime"` if the number entered is a prime number
- `"Is not prime"` otherwise.

The user should then be asked if they want to enter another number and the program should repeat if they say that they do.

A prime number is a positive integer that will leave a remainder if it is divided by any positive integer other than 1 and itself.

You may assume that each number entered by the user is an integer.

If your program only works correctly for some prime numbers you will get some marks for this question. To get full marks for this question, your program must work correctly for any valid integer value that the user enters.

Evidence that you need to provide

(a) Your PROGRAM SOURCE CODE.

(12)

(b) SCREEN CAPTURE(S) showing the result of testing the program by:

- entering the number `1`
- then choosing to enter another number
- then entering the number `5`
- then choosing to enter another number
- then entering the number `8`
- and then choosing not to enter another number.

(1)

(Total 13 marks)

Q4.

WORDS WITH AQA

WORDS WITH AQA is a two-player game. Each player starts with 15 randomly-selected tiles. Each tile represents a single letter; each letter has a points value as shown in **Figure**

1. The tiles that a player has are called their "hand". Each player starts with a score of 50.

Figure 1

Letter	Points letter is worth
A	1
B	2
C	2
D	2
E	1
F	3
G	2
H	3
I	1
J	5
K	3
L	2
M	2
N	1
O	1
P	2
Q	5
R	1
S	1
T	1
U	2
V	3
W	3
X	5
Y	3
Z	5

Players take turns to spell a two or more letter word using only the letter tiles in their hand. Each tile may only be used once in the word. If on their turn they spell a valid word then the tiles used in that word are removed from their hand and their score is increased. The amount their score increases by depends on which tiles were used and the number of tiles used in the word. Initially the score increases by the total points value of the tiles used. If a valid word is spelt that uses more than seven tiles an additional 20 points are added to the player's score. If a valid word is spelt that uses six or seven tiles an additional five points are added to the player's score.

The player may then choose to either get three new tiles, get a number of new tiles equal to the number of tiles used in the word they just spelt, get a number of new tiles equal to three larger than the number of tiles used in the word they just spelt or to get no new tiles. New tiles come from the tile queue.

If the word they spelt is not valid then their turn is over. No tiles are removed from their hand and they get three tiles from the tile queue.

A valid word is one in which all the tiles needed to spell the word are in their hand and it is a correctly-spelt English word.

The tile queue contains 20 randomly-selected tiles. The tiles in the queue are shown in order, the tile at the front of the queue will be the next tile given to a player who gets a new tile. When a tile is removed from the front of the queue and given to a player, a new

randomly-selected tile is added to the rear of the queue. Players may look at the contents of the tile queue at any time.

The game ends when either a player has used a total of more than 50 tiles in the valid words they have spelt or when their hand contains 20 or more tiles. If Player One uses more than 50 tiles first or has a hand with 20 or more tiles then Player Two gets to have their turn before the game ends. At the end of the game each player's score is reduced by the points value of the letters on the tiles remaining in their hand. The winner is the player with the highest score.

Figures 2 to 7 show an example game.

Figure 2

Player One's hand at the start of the game:

B	T	A	H	A	N	D	E	N	O	N	S	A	R	J

Player Two's hand at the start of the game:

C	E	L	Z	X	I	O	T	N	E	S	M	U	A	A

The tile queue at the start of the game:

V	H	H	D	W	I	P	J	G	O	L	T	D	X	I	O	K	O	L	P

Figure 3

Player One goes first and spells the word DATA. This is a valid word so Player One's score is increased by five points, 2 (D) + 1 (A) + 1 (T) + 1 (A), and the four tiles used in that word are removed from the player's hand. Player One chooses to get a number of new tiles equal to three more than the number of tiles used in the word they just spelt and so the first seven tiles in the tile queue are removed from the queue and added to Player One's hand. Seven random tiles are added to the rear of the tile queue.

Player One's hand before playing word with tiles being used highlighted:

B	T	A	H	A	N	D	E	N	O	N	S	A	R	J

Player One's hand with tiles used removed and the seven tiles from the front of the tile queue added to player's hand:

B	H	N	E	N	O	N	S	A	R	J	V	H	H	D	W	I	P

Seven random letter tiles are added to the rear of the tile queue to replace those removed:

J	G	O	L	T	D	X	I	O	K	O	L	P	L	W	C	N	A	G	E

Figure 4

It is now Player Two's turn. Player Two spells the word AXONEMAL. This is a valid word so Player Two's score is increased by 34 points – 1 (A) + 5 (X) + 1 (O) + 1 (N) + 1 (E) + 2 (M) + 1 (A) + 2 (L) + 20 (word more than 7 letters long) – to 84, and the eight tiles used in that word are removed from the player's hand. Player Two chooses not to get any new tiles.

Player Two's hand before playing word with tiles being used highlighted:

C	E	L	Z	X	I	O	T	N	E	S	M	U	A	A

Player Two's hand with tiles used removed:

C	Z	I	T	E	S	U

The tile queue is unchanged:

J	G	O	L	T	D	X	I	O	K	O	L	P	L	W	C	N	A	G	E

Figure 5

It is now Player One's turn again. Player One spells the word NEED. This is not a valid word as they have only got one E tile in their hand and the word they have spelt needs two Es. Player One's score is unchanged, no tiles are removed from Player One's hand and the first three tiles in the tile queue are removed from the queue and added to Player One's hand. Three random tiles are added to the rear of the tile queue.

Player One's hand at start of turn:

B	H	N	E	N	O	N	S	A	R	J	V	H	H	D	W	I	P

Tile queue at start of turn:

J	G	O	L	T	D	X	I	O	K	O	L	P	L	W	C	N	A	G	E

Player One's hand at end of turn:

B	H	N	E	N	O	N	S	A	R	J	V	H	H	D	W	I	P	J	G	O

The queue at end of turn:

L	T	D	X	I	O	K	O	L	P	L	W	C	N	A	G	E	D	R	A

Player One has now got more than 20 tiles in their hand (21) so the game will end. However, Player Two gets their turn before the game ends.

Figure 6

It is now Player Two's turn. Player Two spells the word ZIC. This is not a valid word as it is not an English word. Player Two's score is unchanged, no tiles are removed from Player Two's hand and the first three tiles in the tile queue are removed from the queue and added to Player Two's hand. Three random tiles are added to the rear of the tile queue.

Player Two's hand at start of turn:

C	Z	I	T	E	S	U

Tile queue at start of turn:

L	T	D	X	I	O	K	O	L	P	L	W	C	N	A	G	E	D	R	A

Player Two's hand at end of turn:

C	Z	I	T	E	S	U	L	T	D

Tile queue at end of turn:

X	I	O	K	O	L	P	L	W	C	N	A	G	E	D	R	A	X	X	U

The game now stops and the final scores are calculated.

Figure 7

To calculate Player One's final score the value of the tiles in their hand is subtracted from their current score. Player One's current score is 55. The value of the tiles in their hand is 43. So their final score is 55 – 43 = 12.

B	H	N	E	N	O	N	S	A	R	J	V	H	H	D	W	I	P	J	G	O
2	+ 3	+ 1	+ 1	+ 1	+ 1	+ 1	+ 1	+ 1	+ 1	+ 5	+ 3	+ 3	+ 3	+ 2	+ 3	+ 1	+ 2	+ 5	+ 2	+ 1 = 43

To calculate Player Two's final score the value of the tiles in their hand is subtracted from their current score. Player Two's current score is 84. The value of the tiles in their hand is 18. So their final score is 84 – 18 = 66.

C	Z	I	T	E	S	U	L	T	D
2	+ 5	+ 1	+ 1	+ 1	+ 1	+ 2	+ 2	+ 1	+ 2 = 18

Player One has a final score of 12 and Player Two has a final score of 66 so Player Two has won the game.

In the **Skeleton Program** there is a main menu containing three options.

The first option is to play the game with the players having a random selection of letters in their starting hands. The starting contents of the tile queue are random.

The second option is to play a training game in which the players have the same selection of letters in their starting hands as shown in **Figure 2**. The starting contents of the tile queue are random.

The third option is to quit the program.

When playing the game each player takes it in turn to spell a word. When a player has their turn they have five choices.

If they enter the string `1` then the point values of the 26 letters are displayed. The player's turn continues and they can choose any of the five options.

If they enter the string `4` then the current contents of the tile queue are displayed. The player's turn continues and they can choose any of the five options.

If they enter the string `7` then the current contents of their hand are displayed. The player's turn continues and they can choose any of the five options.

If they enter the string `0` then the player's hand will be given enough tiles to take them over the maximum number of tiles allowed in a hand. This will mean that the game will finish (though if Player One chooses this option, Player Two will still have their turn before the game finishes). The player's turn then finishes.

If they enter any other string then this is treated as being an attempt at spelling a word and the program checks to see if the word is valid. The player's turn then finishes.

To check that a word is valid the program checks if the word is:

- spelt using only letter tiles that are in the player's hand
- in the list of allowed words. The allowed words are the words contained in the **Data File aqawords.txt.**

Data File
A **Data File** named **aqawords.txt** is supplied with the **Skeleton Program**. This stores the list of allowed English words that can be used in the game.

Q5.

(a) State the name of an identifier for a built-in function used in the Skeleton Program that has a string parameter and returns an integer value.

_____

_____

(1)

(b) State the name of an identifier for a local variable in a method in the `QueueOfTiles` class.

_____

_____

(1)

(c) The `QueueOfTiles` class implements a linear queue. A circular queue could have been used instead.

Explain why a circular queue is often a better choice of data structure than a linear queue.

_____

_____

_____

_____

(2)

(d) It could be argued that the algorithms for a linear queue lead to simpler program code.

State **one** other reason why a linear queue is an appropriate choice in the Skeleton Program even though circular queues are normally a better choice.

_____

_____

(1)

(e) State **one** additional attribute that must be added to the `QueueOfTiles` class if it were to be implemented as a circular queue instead of as a linear queue.

_____

(1)

(f) Describe the changes that would need to be made to the Skeleton Program so that the probability of a player getting a one point tile is the same as the probability of them getting a tile worth more than one point. The changes you describe should not result in any changes being made to the points value of any tile.

You should **not** change the Skeleton Program when answering this question.

(4)

(g) The `GetChoice` subroutine uses a built-in function to convert a string to uppercase.

Describe how a string consisting of lowercase characters could be converted to uppercase using only one iterative structure if the programming language being used does not have a built-in function that can do this conversion.

You may assume that only lowercase characters are entered by the user.

You should not change the Skeleton Program when answering this question.

(4)

(Total 14 marks)

Q6.

(a) This question refers to the subroutine `CreateTileDictionary`.

The points values for the letters J and X are to be changed so that they are not worth as many points as the letters Z and Q.

Adapt the subroutine `CreateTileDictionary` so that the letters J and X are worth 4 points instead of 5 points.

Test that the changes you have made work:

- run the **Skeleton Program**
- enter `2` at the main menu
- enter `1` to view the letter values.

Evidence that you need to provide

(i) Your PROGRAM SOURCE CODE for the amended subroutine `CreateTileDictionary`.

(1)

(ii) SCREEN CAPTURE(S) showing the results of the requested test.

(1)

(b) This question refers to the subroutine `Main`.

Currently each player starts with 15 letter tiles. In the `Main` subroutine `StartHandSize` is set to a value of 15.

Change the subroutine `Main` so that the user can choose what the value for `StartHandSize` will be.

Before the main menu is displayed and before the first iteration structure in `Main` the message "`Enter start hand size:` " should be displayed and the user's input should be stored in `StartHandSize`. This should happen repeatedly until the user enters a value for the start hand size that is between 1 and 20 inclusive.

Test that the changes you have made work:

- run the **Skeleton Program**
- enter `0` when asked to enter the start hand size
- enter `21` when asked to enter the start hand size
- enter `5` when asked to enter the start hand size
- enter `1` at the main menu.

Evidence that you need to provide

(i) Your PROGRAM SOURCE CODE for the amended subroutine `Main`.

(4)

(ii) SCREEN CAPTURE(S) showing the requested test. You must make sure that evidence for all parts of the requested test is provided in the SCREEN CAPTURE(S).

(1)

(c) This question refers to the subroutine `CheckWordIsValid`.

When a player enters a word, a linear search algorithm is used to check to see if the

word entered is in the list of `AllowedWords`. The subroutine `CheckWordIsValid` is to be changed so that it uses a more time-efficient search algorithm.

Change the `CheckWordIsValid` subroutine so that it uses a binary search algorithm instead of a linear search algorithm.

You **must write your own search routine** and not use any built-in search function that might be available in the programming language you are using.

Each item in `AllowedWords` that is compared to the word that the user entered should be displayed on the screen.

Figure 2 shows examples of how the new version of `CheckWordIsValid` should work if `AllowedWords` contained the items shown in **Figure 1**.

Figure 1

BIG
BUG
FED
GET
JET
NOT
SIP
WON

Figure 2

1) If the user enters the word BIG then a value of `True` should be returned and the words GET, BUG, BIG should be displayed, in that order.
2) If the user enters the word JET then a value of `True` should be returned and the words GET, NOT, JET should be displayed, in that order.
3) If the user enters the word ZOO then a value of `False` should be returned and the words GET, NOT, SIP, WON should be displayed, in that order.

Test that the changes you have made work:

- run the **Skeleton Program**
- if you have answered part (b), enter `15` when asked to enter the start hand size; if you have not answered part (b) yet then skip this step
- enter `2` at the main menu
- enter the word `jars`.

Evidence that you need to provide

(i) Your PROGRAM SOURCE CODE for the amended subroutine `CheckWordIsValid`.

(8)

(ii) SCREEN CAPTURE(S) showing the requested test.

(1)

(d) This question extends the functionality of the game.

After spelling a valid word the player decides which one of four options to select to determine how many tiles will be added to their hand. Before choosing they can look at the values of the tiles.

It would help the player make their decision if they were aware of how useful each letter was by knowing the frequency with which each letter appears in the list of allowed words.

The program is to be extended so that when the player chooses to view the tile values they are also shown the number of times that each letter appears in the list of allowed words.

What you need to do

Task 1
Create a new subroutine called `CalculateFrequencies` that looks through the list of allowed words and displays each of the 26 letters in the alphabet along with the number of times that the letter appears in the list of allowed words, which the subroutine has calculated.

Task 2

Modify the `DisplayTileValues` subroutine so that after displaying the tile values it also calls the `CalculateFrequencies` subroutine.

Task 3
Test that the changes you have made work:

- run the **Skeleton Program**
- if you have answered part (b), enter `15` when asked to enter the start hand size; if you have not answered part (b) yet then skip this step
- enter `2` at the main menu
- enter `1` to display the letter values.

Evidence that you need to provide

(i) Your PROGRAM SOURCE CODE for the new subroutine `CalculateFrequencies`, the amended subroutine `DisplayTileValues` and any other subroutines you have modified when answering this question.

(8)

(ii) SCREEN CAPTURE(S) showing the requested text.

(1)

(e) The scoring system for the game is to be changed so that if a player spells a valid word they score points for all valid words that are a prefix of the word entered. A prefix is the first **x** characters of a word, where **x** is a whole number between one and the number of characters in the word.

In the **Skeleton Program**, `AllowedWords` contains the list of valid words that have been read in from the **Data File aqawords.txt**.

Example
If the user enters the word TOO they will be awarded points for the valid words TOO and TO as TO is a prefix of the word TOO. They will not be awarded points for the word OO even though it is a valid word and is a substring of TOO because it is not a prefix of TOO.

Example
If the user enters the word BETTER they will be awarded points for the words BETTER, BET and BE as these are all valid prefixes of the word entered by the user. They would not be awarded points for BETT or BETTE as these are not valid English words. They would not be awarded points for BEER as even though it is contained in the word BETTER it is not a prefix.

Example
If the user enters the word BIOGASSES they will be awarded points for the words BIOGASSES, BIOGAS, BIOG, BIO and BI as these are all valid prefixes of the word entered by the user. They would not be awarded points for BIOGA, BIOGASS or BIOGASSE as these are not valid English words. They would not be awarded points for GAS as even though it is contained in the word BIOGASSES it is not a prefix.

Example
If the user enters the word CALMIEST they will not be awarded any points as even though CALM at the start is a valid word the original word entered by the user, CALMIEST, is not.

Example
If the user enters the word AN they will be awarded points for the word AN. They would not be awarded points for A even though A at the start is a valid word as points are only awarded for words that are at least two letters long.

What you need to do

Task 1
Write a **recursive** subroutine called `GetScoreForWordAndPrefix` that, if given a valid word, returns the score of the word added to the score for any valid words that are prefixes of the word.

To get full marks for this task the `GetScoreForWordAndPrefix` subroutine must make use of recursion in an appropriate way to calculate the score for any prefixes that are also valid words.

If your solution uses an alternative method to recursion you will be able to get most but not all of the available marks for this question.

Task 2
Modify the `UpdateAfterAllowedWord` subroutine so that it calls the new `GetScoreForWordAndPrefix` subroutine instead of the `GetScoreForWord` subroutine.

Task 3
Test that the changes you have made to the program work:

- run the **Skeleton Program**
- if you have answered part (b), enter `15` when asked to enter the start hand size; if you have not answered part (b) yet then skip this step
- enter `2` at the main menu
- enter the word `abandon`
- enter `4` so that no tiles are replaced.

Evidence that you need to provide

(i) Your PROGRAM SOURCE CODE for the new subroutine `GetScoreForWordAndPrefix`, the amended subroutine

`UpdateAfterAllowedWord` and any other subroutines you have modified when answering this question.

(11)

(ii) SCREEN CAPTURE(S) showing the results of the requested test.

(1)

(Total 37 marks)

Q7.

Figure 1 shows the data Norbert, Phil, Judith, Mary, Caspar and Tahir entered into a binary search tree.

Figure 2 contains pseudo-code for a recursive binary tree search algorithm.

Figure 1

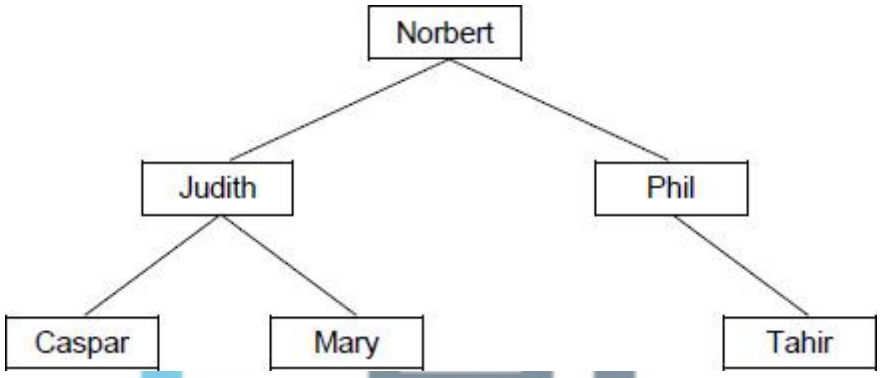


Figure 2

```
FUNCTION TreeSearch(target, node)
  OUTPUT 'Visited ', node
  IF target = node THEN
    RETURN True
  ELSE IF target > node AND Exists(node, right) THEN
    RETURN TreeSearch(target, node.right)
  ELSE IF target < node AND Exists(node, left) THEN
    RETURN TreeSearch(target, node.left)
  ENDIF
  RETURN False
ENDFUNCTION
```


The subroutine `Exists` takes two parameters – a node in the binary tree and a direction (`left` or `right`). It returns a Boolean value indicating if the node given as a parameter has a child node in the direction specified by the second parameter. For instance, `Exists(Mary, left)` will return a value of `False` as there is no node to the left of `Mary` in the binary tree.

`node.right` evaluates to the child node to the right of `node`, eg `Judith.right` is `Mary`.

`node.left` evaluates to the child node to the left of `node`, eg `Judith.left` is `Caspar`.

(a) What is meant by a recursive subroutine?

_____

_____

(1)

(b) There are two base cases for the subroutine `TreeSearch`. State **one** of the base cases.

_____

_____

(1)

(c) Complete the unshaded cells of the table below to show the result of tracing the `TreeSearch` algorithm shown in **Figure 2** with the function call `TreeSearch(Olivia, Norbert)`. You may not need to use all of the rows.

Function call	Output
`TreeSearch(Olivia, Norbert)`	

(3)

(Total 5 marks)

Q8.

(a) This question refers to the subroutine `InputCoordinate` in the `Simulation` class.

The warren and fox inspection options in the **Skeleton Program** do not currently check if the coordinates entered by the user are on the landscape. This behaviour needs to be improved so that an error message is displayed if the user inputs coordinates for a location that is not on the landscape.

If the user runs a simulation with default settings then the landscape size is 15, so valid locations have an x coordinate between 0 and 14, inclusive.

What you need to do

Modify the `InputCoordinate` subroutine in the `Simulation` class so that, if a coordinate outside the range defined by the landscape size is input, the error message `"Coordinate is outside of landscape, please try again."` is displayed and the user is forced to re-input the coordinate.

To achieve full marks for this question, the `InputCoordinate` subroutine should work correctly for any landscape size, not just the default size of 15.

Test

Test your changes work by running the **Skeleton Program** and selecting the following options:

- "1. Run simulation with default settings"
- "3. Inspect fox"

Then input these three x coordinates for the location of the fox to inspect:

- `-1`
- `15`
- `0`

Evidence that you need to provide

(i) Your PROGRAM SOURCE CODE for the amended subroutine `InputCoordinate`.

(4)

(ii) SCREEN CAPTURE(S) for the described test.

Ensure that in your SCREEN CAPTURE(S) it can be seen that the x coordinates `-1` and `15` are rejected **and** that the x coordinate `0` is accepted, **and** that after the `0` is input the **Skeleton Program** advances to ask the user to input the y coordinate.

(1)

(b) The simulation is to be made more realistic by increasing the probability that a rabbit will die as a result of other causes, such as disease or injury, as the rabbit ages.

The default probability of death by another cause for a rabbit is 0.05.

- The probability of a male rabbit dying by another cause should increase by a factor of 50% after every time period.
- The probability of a female rabbit dying by another cause should remain constant until the rabbit reaches the age of 2. At the age of 2, and after every time period beyond this, the probability of a female rabbit dying by another cause should increase by 0.05.

Table 1 below summarises the probability of death by other causes for a rabbit of each gender, up to the age of 5. The probabilities will continue to increase beyond this age.

Table 1

	Probability of death by other causes	
Age	**Male (2dp)**	**Female**
0	0.05	0.05
1	0.08	0.05
2	0.11	0.1
3	0.17	0.15
4	0.25	0.2
5	0.38	0.25

What you need to do

Create a new subroutine, `CalculateNewAge`, in the `Rabbit` class, that overrides the `CalculateNewAge` subroutine in the `Animal` class.

The new `CalculateNewAge` subroutine in the `Rabbit` class should recalculate the probability of death for a rabbit as the rabbit ages. The subroutine should also call the subroutine that it has overridden in the `Animal` class to ensure that the standard ageing process for a rabbit continues to be carried out as well.

Test

Check that the changes you have made work by conducting the following test:

- Select option "1. Run simulation with default settings" from the main menu.
- Then select option "2. Advance to next time period hiding detail" **twice**, to advance the simulation to time period 2.
- Then select option "4. Inspect warren" and enter the x coordinate `1` and the y coordinate `1`.
- When asked "View individual rabbits (y/n)?" enter `y`.

Evidence that you need to provide

(i) Your PROGRAM SOURCE CODE for the new subroutine `CalculateNewAge` from the `Rabbit` class.

(5)

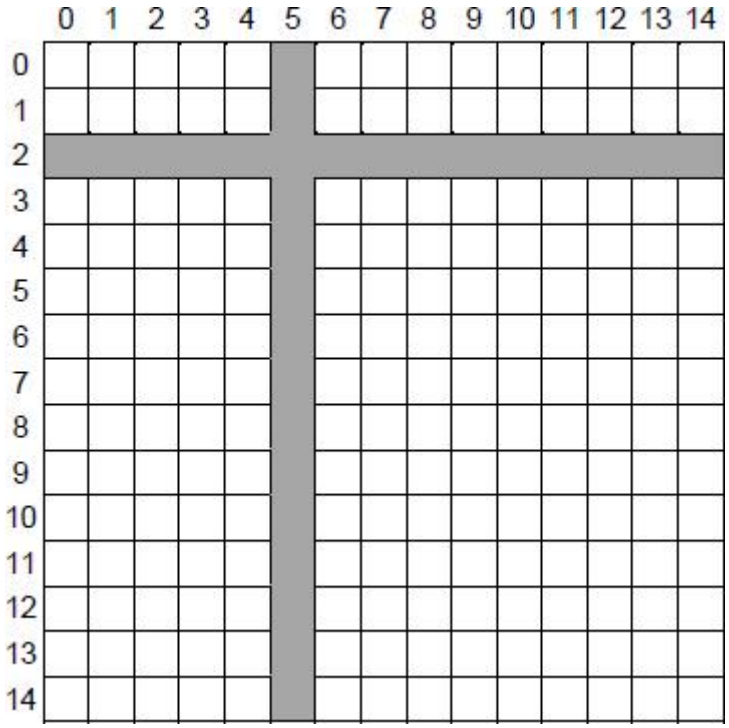
(ii) SCREEN CAPTURE(S) for the described test.

Your SCREEN CAPTURE(S) must clearly show the probability of death by other causes of both a male and a female rabbit of age 2. SCREEN CAPTURE(S) do **not** need to show the options that you have selected or the probability of death by other causes for rabbits of other ages.

(1)

(c) The simulation is to be extended to represent the landscape that the animals live in. Most of the landscape will be land, but two rivers will run through it. The locations of the rivers are shaded in **Figure 3**.

Figure 3



Each of the individual locations, eg (12, 7), within the landscape will be assigned to be an area of either land or river.

What you need to do

Task 1

Modify the `Location` class so that it can store a representation of the type of terrain at the location. This representation should be as a character, with "L" representing land and "R" representing river.

Task 2

Modify the constructor subroutine of the `Location` class so that when a location is created, the constructor is passed the type of terrain that the location will be and this is stored appropriately.

Task 3

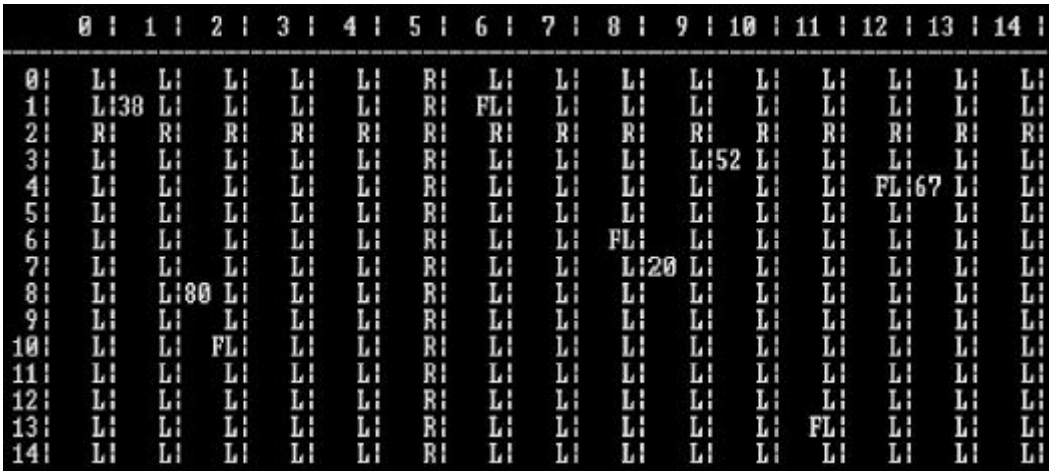
Modify the `CreateLandscapeAndAnimals` subroutine in the `Simulation` class so that when the landscape is created the appropriate type of terrain, as shown in **Figure 3**, is stored in each location. The terrain should be represented as a character, with "L" representing land and "R" representing river.

Task 4

Modify the `DrawLandscape` subroutine in the `Simulation` class so that the correct type of terrain at each location is displayed when the landscape is drawn.

Figure 4 shows one example of how the landscape could be drawn, with a letter "L" indicating that a location contains land, and a letter "R" indicating that a location contains part of a river. However, you are free to indicate the type of terrain at a location in any way that you choose, so long as this is clear to the user.

Figure 4



Task 5

Modify the `CreateNewWarren` and `CreateNewFox` subroutines in the `Simulation` class so that warrens and foxes cannot be created in locations that are part of a river.

Test

Check that the changes you have made in Tasks 1 to 4 (**not** Task 5) work by conducting the following test:

- Select option "1. Run simulation with default settings" from the main menu.

Evidence that you need to provide

(i) Your PROGRAM SOURCE CODE for the whole of the `Location` class, including the constructor subroutine.

(3)

(ii) Your PROGRAM SOURCE CODE for the amended `CreateLandscapeAndAnimals` subroutine from the `Simulation` class.

(3)

(iii) Your PROGRAM SOURCE CODE for the amended `DrawLandscape` subroutine from the `Simulation` class.

(2)

(iv) Your PROGRAM SOURCE CODE for the amended `CreateNewWarren` and `CreateNewFox` subroutines from the `Simulation` class.

(3)

(v) SCREEN CAPTURE(S) for the described test, showing the correct type of territory in each location on the landscape.

(1)

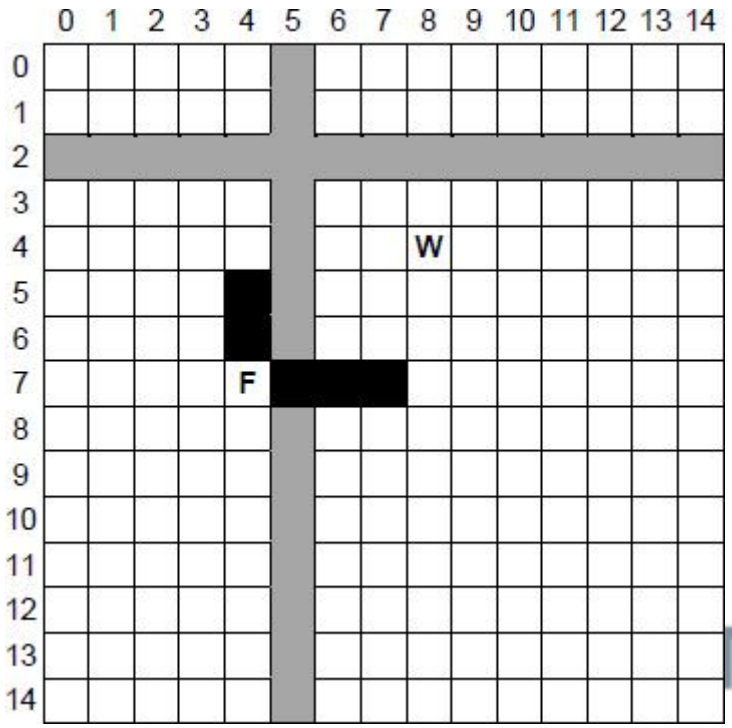
(d) The landscape affects the foxes' ability to eat the rabbits. Foxes do not like to swim, so will not cross the rivers on the landscape to eat. If a river lies between a fox and a warren, the fox will not eat any rabbits in the warren, even if it is near enough for it to do so.

As the rivers only run horizontally and vertically, and extend from one side of the landscape to the other, a simple way to check if reaching a warren would require a fox to cross a river is to:

- Calculate the coordinates of all of the locations between the fox and the warren in a horizontal line, level with the fox.
- Calculate the coordinates of all of the locations between the fox and the warren in a vertical line, level with the fox.
- If any of the locations horizontally or vertically between the fox and the warren contain a river, then the fox will not eat any of the rabbits in the warren as the fox's path to the warren crosses a river.

Figure 5 shows the locations that would need to be checked to see if fox **F** could eat any rabbits in warren **W**. The locations that need to be checked are shown in black and the rivers are shown in grey. As location (5, 7) contains part of a river, the fox would not eat any rabbits in this warren.

Figure 5



If you have not been able to fully complete part **(c)**, you will still be able to get most of the marks for this question if you can correctly compute the coordinates of the locations that would need to be checked to see if a river was present.

To get full marks for this question, your solution must work regardless of whether a warren is above, below, to the left or to the right of a fox.

What you need to do

Task 1
Create a new subroutine `CheckIfPathCrossesRiver`, in the `Simulation` class, that takes the coordinates of two locations in the landscape **and** checks if there is a river between them.

Task 2
Modify the `FoxesEatRabbitsInWarren` subroutine in the `Simulation` class so that it calls the `CheckIfPathCrossesRiver` subroutine, **and** ensures that if there is a river between a fox and a warren then the fox will not eat any rabbits from the

warren.

Test
Check that the changes you have made work by conducting the following test:

- Select option "1. Run simulation with default settings" from the main menu.
- Then select option "1. Advance to next time period showing detail".

When the test is conducted, no rabbits in the warren at (1, 1) should be eaten as it is bounded by rivers on all sides.

Evidence that you need to provide

(i) Your PROGRAM SOURCE CODE for the new subroutine `CheckIfPathCrossesRiver` from the `Simulation` class.

(9)

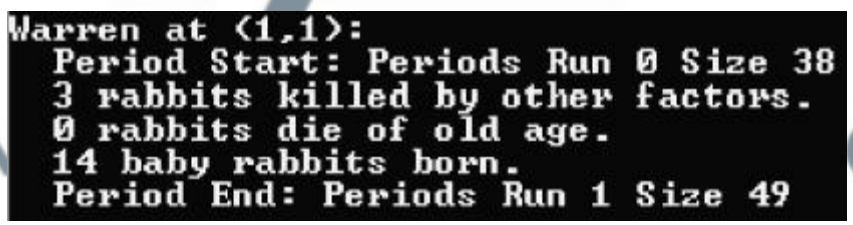
(ii) Your PROGRAM SOURCE CODE for the amended subroutine `FoxesEatRabbitsInWarren` from the `Simulation` class.

(2)

(ii) SCREEN CAPTURE(S) for the described test.

Your SCREEN CAPTURE(S) only needs to show what happens in the warren at location (1, 1) when the simulation advances to the next time period. It should contain similar information to **Figure 6** below, but the exact number of rabbits killed, dying of old age and other details may differ owing to the random nature of parts of the simulation.

Figure 6



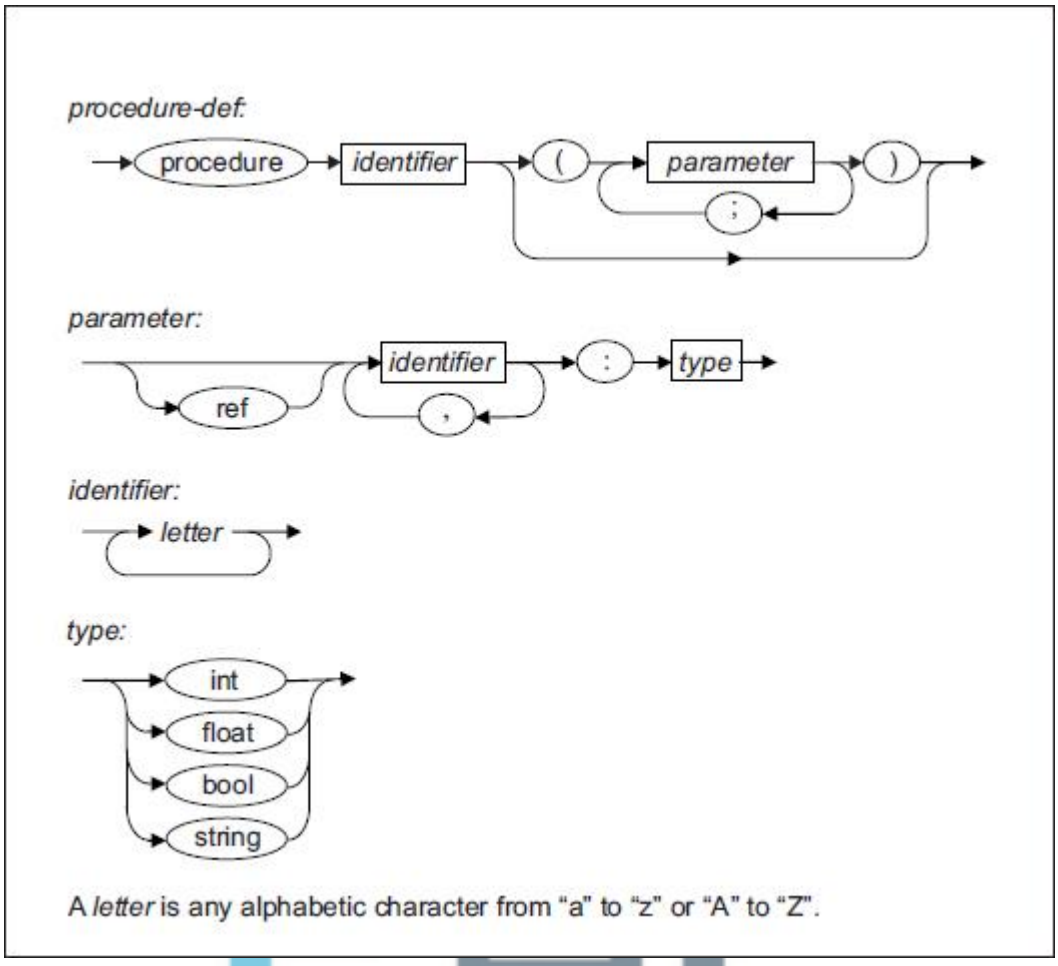
(1)

(Total 35 marks)

Q9.

In a particular programming language, the correct syntax for four different constructs is defined by the syntax diagrams in **Figure 1**.

Figure 1



A *letter* is any alphabetic character from "a" to "z" or "A" to "Z".

In this language an example of a valid *identifier* is `loopCount` and an example of a valid *type* is `int`.

(a) For each row in the table below, write **Yes** or **No** in the **Valid?** column to identify whether or not the **Example** is a valid example of the listed **Construct**.

Construct	Example	Valid? (Yes/No)
identifier	`Game_Over`	
parameter	`ref x,y:bool`	
procedure-def	`procedure square(s:float)`	
procedure-def	`procedure rect(w:int,h:int)`	

(4)

A student has written Backus-Naur Form (BNF) production rules that are supposed to define the same constructs as the syntax diagrams in **Figure 1**. Their BNF rules are shown in **Figure 2**.

Figure 2

```
<procedure-def> ::= procedure <identifier> ( <paramlist> )
<paramlist>     ::= <parameter> | <parameter> ; <paramlist>
<parameter>     ::= <identlist> : <type> |
                    ref <identlist> : <type>
<identlist>     ::= <identifier> | <identifier> , <identlist>
```

```
<identifier>    ::= <letter> | <letter> <identifier>
<type>          ::= int | float | bool
```

A <letter> is any alphabetic character from "a" to "z" or "A" to "Z".

(b) The BNF production rules in **Figure 2** contain two errors. These errors mean that the production rules do not represent the same statement types as the syntax diagrams in **Figure 1**.

Describe the **two** errors.

Error 1: _____

_____

_____

Error 2: _____

_____

_____

(2)

(c) The production rule for a <paramlist> is recursive.

Explain why recursion has been used in this production rule.

_____

_____

_____

_____

(1)

(Total 7 marks)

Q10.

A list data structure can be represented using a fixed-size array.

The pseudo-code algorithm in **Figure 1** can be used to carry out one useful operation on a list.

Figure 1

```
p ← 1
IF ListLength > 0 THEN
  WHILE p <= ListLength AND List[p] < New

    p ← p + 1
  ENDWHILE

  FOR q ← ListLength DOWNTO p DO

    List[q + 1] ← List[q]
  ENDFOR
```

```
        ENDIF
        List[p] ← New
        ListLength ← ListLength + 1
```

The DOWNTO command causes the loop counter value to decrease by one on each iteration of the FOR loop.

The initial values of the variables for one particular execution of the algorithm are shown in the trace table opposite, labelled **Table 1**. Array indexing starts at 1.

(a) Complete the trace table for the execution of the algorithm in **Figure 1**.

Table 1

				List				
ListLength	New	p	q	[1]	[2]	[3]	[4]	[5]
4	48	–	–	19	43	68	107	–

(4)

(b) Describe the purpose of the algorithm in **Figure 1**.

_____

_____

(1)

(c) A list may be implemented using a static data structure such as a fixed-length array, or using a dynamic data structure such as a linked list.

If the list were to be implemented using a dynamic data structure explain what the heap would be used for.

_____

_____

_____

(1)

(Total 6 marks)

Q11.

A computer program is being developed to play a card game on a smartphone. The game uses a standard deck of 52 playing cards, placed in a pile on top of each other.

The cards will be dealt (ie given out) to players from the top of the deck.

When a player gives up a card it is returned to the bottom of the deck.

(a) Explain why a queue is a suitable data structure to represent the deck of cards in this game.

_____

_____

(1)

(b) The queue representing the deck of cards will be implemented as a **circular queue** in a fixed-size array named `DeckQueue`. The array `DeckQueue` has indices running from `1` to `52`.

The figure below shows the contents of the `DeckQueue` array and its associated pointers at the start of a game. The variable `QueueSize` indicates how many cards are currently represented in the queue.

DeckQueue

Index	Data
[1]	10-Spades
[2]	2-Hearts
[3]	King-Clubs
[4]	Ace-Hearts
.	
.	
.	
[52]	8-Clubs

FrontPointer = 1

RearPointer = 52

QueueSize = 52

(i) Twelve cards are dealt from the top of the deck.

What values are now stored in the `FrontPointer` and `RearPointer` pointers and the `QueueSize` variable?

`FrontPointer` = _____ `RearPointer` = _____

`QueueSize` = _____

(1)

(ii) Next, a player gives up three cards and these are returned to the deck.

What values are now stored in the `FrontPointer` and `RearPointer` pointers and the `QueueSize` variable?

`FrontPointer` = _____ `RearPointer` = _____

`QueueSize` = _____

(1)

(c) Write a pseudo-code algorithm to deal a card from the deck.

Your algorithm should output the value of the card that is to be dealt and make any required modifications to the pointers and to the `QueueSize` variable.

It should also cope appropriately with any situation that might arise in the `DeckQueue` array whilst a game is being played.

(6)

(Total 9 marks)

Q12.

A computer games programmer is writing a game. One aspect of the game involves a character who can carry various items, such as a bag of seeds, and an axe, around with her. The list of items that the character is currently carrying will be stored as a linked list of items of the `String` data type. The list is stored in no particular order.

The game is being developed using object-oriented programming. The **LinkedList** class will be used to store that list of items.

The class definition for the **LinkedList** class is:

```
LinkedList
   = Class
     Public
```

```
    Procedure CreateList
    Procedure DestroyList
    Procedure AddItem(NewItem: String)
    Procedure DeleteItem(DelItem: String)
    Function ContainsItem(SearchItem: String): Boolean
    Function IsEmpty: Boolean
Private
    Start: Pointer
    Current: Pointer
    Previous: Pointer
End
```

(a) Creating a class such as the **LinkedList** class, that can be used by other parts of a much bigger program, is a form of abstraction.

Explain why the **LinkedList** class is a form of abstraction.

_____

_____

_____

(1)

(b) Explain why the functions and procedures, such as `AddItem` have been declared to be `Public` whilst the data items such as `Start` have been declared as `Private`.

_____

_____

_____

_____

_____

(2)

(c) Write a pseudo-code algorithm for the `DeleteItem` operation.

You may assume that:

- `Start` is a pointer to the memory location of the first item in the linked list
- The variable `DelItem`, which will be passed to the `DeleteItem` operation as a parameter, is a `String` that contains the name of the item to delete, exactly as the name appears in the linked list
- The linked list is not empty, and does contain the item to be deleted
- For each item stored in the list, two fields are stored, which are called `DataValue` and `Next`. The `DataValue` is the name of the item that is stored and `Next` is a pointer to the memory location of the next item in the list. To access the values stored in these fields at a particular memory location, such as `Current`, the instructions `Current.DataValue` and `Current.Next` would be used
- An operation called `Release` is provided by the operating system that will make a specified memory location that is no longer required available for re-use
- You should make use of the data items `Current` and `Previous`, both of which are pointers, when searching the list to locate the item that is to be deleted.

(8)

(Total 11 marks)

Q13.

A dictionary is an abstract data type that allows pairs of values to be associated with each other. For example, an English-French dictionary might associate English words with their translations into French. In such a dictionary, "Apple" would be associated with "Pomme" because "Pomme" is the French word for "Apple".

At a lower level of abstraction, a dictionary could be implemented as a data structure using a number of different methods. Two possible implementation methods are:

- **Implementation One**: As an unordered list in an array.
- **Implementation Two**: Using hashing, with the English word being passed through a hash function to calculate the position of the correct French translation in an array.

In each implementation, a record containing the English word and the equivalent French word are stored at each index in the array that is in use.

The figure below shows how an English-French dictionary containing five words could be implemented using these two methods.

Implementation One – Unordered List

Index	English Word	French Word
[1]	Apple	Pomme

Implementation Two – Hashing

Index	English Word	French Word
[1]	Pear	Poire

[2]	Lemon	Citron
[3]	Strawberry	Fraise
[4]	Grapefruit	Pamplemousse
[5]	Pear	Poire
[6]		
[7]		
[8]		
[9]		
[10]		

New words are inserted into the array in the first available slot. The next word would be stored at position 6.

[2]	Lemon	Citron
[3]		
[4]		
[5]		
[6]		
[7]	Apple	Pomme
[8]	Strawberry	Fraise
[9]		
[10]	Grapefruit	Pamplemousse

The position to store a word is calculated from the English word using a hashing function.

(a) Explain why, when the French translation of an English word needs to be looked up, **Implementation Two** is more time efficient than **Implementation One**.

(2)

(b) In **Implementation Two**, it is possible that the hash function could compute the same value for two different English words.

Explain what the effect of this would be, and how it could be dealt with.

(2)

(c) In **Implementation Two**, both the English and French words are stored at each index in the Array. In this implementation, explain why it would not be possible to perform reliable English to French translation if only the French words were stored.

_____

_____

_____

_____

_____

(1)

(Total 5 marks)

Q14.

Figure 1 contains the pseudo-code for a program to output a sequence according to the 'Fizz Buzz' counting game.

Figure 1

```
OUTPUT "How far to count?"
INPUT HowFar
WHILE HowFar < 1
  OUTPUT "Not a valid number, please try again."
  INPUT HowFar
ENDWHILE
FOR MyLoop ← 1 TO HowFar
  IF MyLoop MOD 3 = 0 AND MyLoop MOD 5 = 0
  THEN
      OUTPUT "FizzBuzz"
    ELSE
      IF MyLoop MOD 3 = 0
        THEN
          OUTPUT "Fizz"
        ELSE
          IF MyLoop MOD 5 = 0
            THEN
              OUTPUT "Buzz"
            ELSE
              OUTPUT MyLoop
          ENDIF
      ENDIF
  ENDIF
ENDFOR
```


What you need to do:

Write a program that implements the pseudo-code as shown in **Figure 1**.

Test the program by showing the result of entering a value of `18` when prompted by the program.

Test the program by showing the result of entering a value of `-1` when prompted by the program.

Evidence that you need to provide

(a) Your PROGRAM SOURCE CODE for the pseudo-code in **Figure 1**.

(8)

(b) SCREEN CAPTURE(S) for the tests conducted when a value of `18` is entered by

the user and when a value of `-1` is entered by the user.

(1)

The main part of the program uses a `FOR` repetition structure.

(c) Explain why a `FOR` repetition structure was chosen instead of a `WHILE` repetition structure.

_____

_____

(1)

(d) Even though a check has been performed to make sure that the variable `HowFar` is greater than `1` there could be inputs that might cause the program to terminate unexpectedly (crash).

Provide an example of an input that might cause the program to terminate and describe a method that could be used to prevent this.

_____

_____

_____

_____

_____

_____

(3)

(e) Programs written in a high level language are easier to understand and maintain than programs written in a low level language.

The use of meaningful identifier names is one way in which high level languages can be made easier to understand.

State **three** other features of high level languages that can make high level language programs easier to understand.

_____

_____

_____

_____

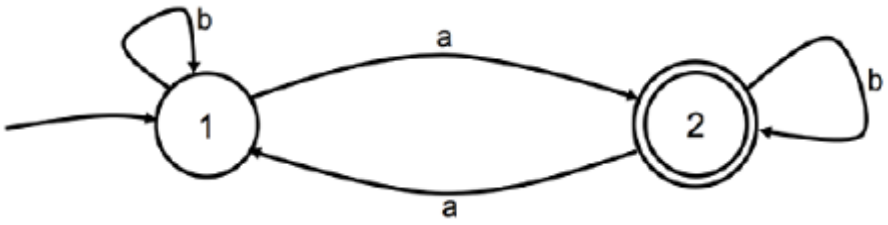
_____

_____

(3)

(f) The finite state machine (FSM) shown in **Figure 2** recognises a language with an alphabet of `a` and `b`.

Figure 2



Input strings of `a` and `aabba` would be accepted by this FSM.

In the table below indicate whether each input string would be accepted or not accepted by the FSM in **Figure 2**.

If an input string would be accepted write YES.
If an input string would **not** be accepted write NO.

Input string	Accepted by FSM?
aaab	
abbab	
bbbbba	

(2)

(g) In words, describe the language (set of strings) that would be accepted by this FSM shown in **Figure 2**.

_____

_____

_____

_____

(2)

(Total 20 marks)

Q15.

State the name of an identifier for:

(a) an array or list variable

_____

_____

(1)

(b) a user-defined subroutine that has four parameters

_____

_____

(1)

(c) a variable that is used to store a whole number.

_____

_____

(1)

(d) a user-defined subroutine that returns one or more values.

_____

_____

(1)

(e) Look at the repetition structures in the `DisplayCavern` subroutine.

Explain the need for a nested `FOR` loop and the role of the `Count1` and `Count2` variables.

_____

_____

_____

_____

_____

_____

(3)

(f) Look at the `ResetCavern` subroutine.

Why has a named constant been used instead of the numeric value `5`?

_____

_____

_____

_____

(2)

(g) Look at the `SetPositionOfItem` subroutine.

Describe the purpose of the `WHILE` loop and the command within it in this subroutine.

_____

_____

_____

(3)

(h) Look at the `MakeMonsterMove` subroutine.

Describe why it is necessary to check if the monster moves into the same cell as the flask and how any problem caused by this is solved by the **Skeleton Program**.

(3)

(i) Look at the `PlayGame` subroutine.

Explain why a `WHILE` loop has been made to complete the two moves for the monster rather than a `FOR` loop.

(2)

(j) The subroutines in the **Skeleton Program** avoid the use of global variables: they use local variables and parameter passing instead.

State **two** reasons why subroutines should, ideally, **not** use global variables.

(2)

(Total 19 marks)

Q16.

Based on the description in Statius' journal you are sure that this must be the right place.

The blue-green moss covering the rocks and the dense tree foliage combine to conceal the cave entrance; you almost walked straight past it and it was only through luck that you saw it. There isn't any time to waste – ever since the journal was discovered, everyone has been looking for this place. The thick cobwebs across the entrance prove that you must be the first one here. You know that, if you are right, you could be the one that finds, in the cavern below the mountain, the single draft of Styxian potion contained in Statius' flask. The journal says that there is a fearsome beast lying in wait, but the risk is worth it. Statius wrote that consuming the potion would grant the drinker invulnerability. Nothing could hurt you, cut you, graze, scratch or bruise you. Your thoughts start to drift and you imagine what you could do with such power.

"Snap out of it," you tell yourself. Someone else could find this place and you can't take that risk; the flask contains only enough potion for one. Quickly you shoulder your pack, then you light your torch, brush aside some cobwebs and step into the cave.

Inside, the ground is rough and you stumble several times. As you go deeper into the mountain, the cave darkens and soon the only light is coming from your torch. After you have been walking for a few minutes the cave widens and then ends abruptly. You take out your copy of the journal. You read the description of the cave again. It says that, at the end of the cave, there is a large fissure near the western part of the wall and that the cavern, where the flask is hidden, is at the bottom of the fissure. You move over to the west side of the cave. Sure enough, the fissure is there. You take off your pack and then move carefully nearer the edge. The light from the torch does not reach far enough down to reveal the bottom, but you can see that the fissure walls are too steep to get down unaided so you will need your climbing equipment. You place your torch carefully on the floor nearby and take your rope, pitons and carabiners out of your pack.

Your foot slips on some loose stones and you fall backwards into the fissure. You tumble down the hole in a shower of dust and pebbles, falling into the cavern below. You land painfully on the rocky floor.

It is dark; your torch is back in the cave and it weakly illuminates your immediate surroundings but you cannot see any farther into the cavern. You become aware of a sonorous noise around you and it takes you a few minutes to work out what it is. The monster is asleep somewhere in the cavern and is snoring loudly. The sound reverberates around you so you can't work out in which direction the monster lies.

You can see the bottom aperture of the fissure several metres above your head and try to scramble up the wall to reach it, but the rock face is too sheer and you can't get sufficient purchase. From what you can remember of the journal, you must be at the north-west corner of the cavern.

You make your mind up. You can't get out of the cavern the way that you came in and the monster could wake soon. You decide to explore the cavern. Maybe you will find another way out. Maybe you will find the Styxian potion. You have no choice but to play the game of...

MONSTER!

MONSTER!

The **Skeleton Program** in this Preliminary Material is a program for the one-player game MONSTER!.

When playing MONSTER! the player starts in the north-west corner of a dark cavern. The cavern is represented by a 7 × 5 rectangular grid of cells. The player's current position is

indicated by an *. The player is presented with a list of five options: they can either return to the main menu (where they can save the current game if they want to) or they can move one cell in one of four possible directions. If they enter 'N' they will move one cell to the north, 'S' will move them one cell to the south, 'E' will move them one cell to the east and 'W' will move them one cell to the west. The initial position of a new game is shown in **Figure 1**.

Figure 1

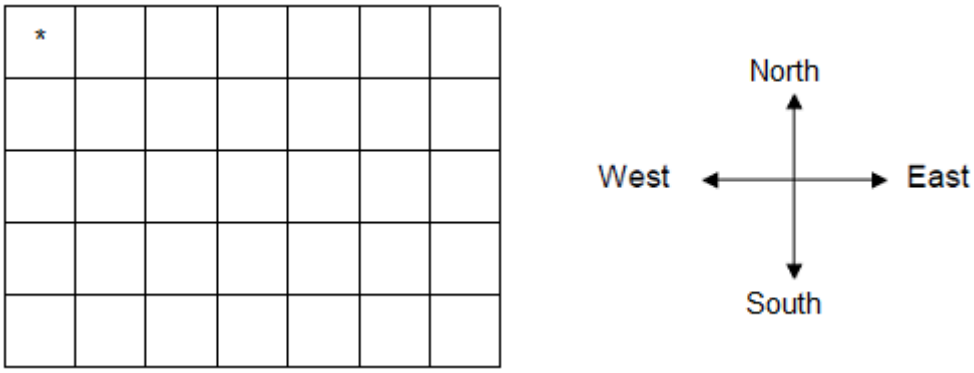
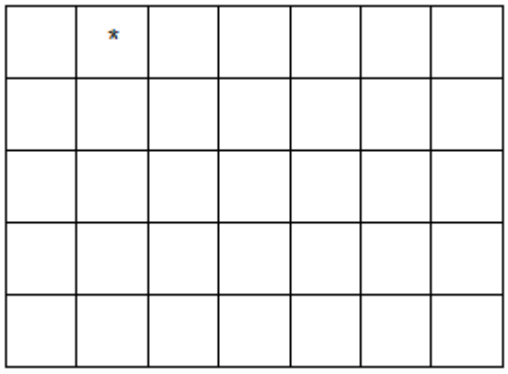


Figure 2 shows a new state, resulting from the user selecting 'E' in the starting position.

Figure 2



The aim of MONSTER! is to find the hidden treasure (a flask containing a Styxian potion) that is in one of the cells of the cavern. Unfortunately, the cavern is dark and the only way that a player can find the flask is to move around until they are in the same cell as the flask. When a new game is started, the flask will be in a random position in the cavern (though it won't be in the same cell as the player starts in). If the player moves into the cell that contains the flask, then they win the game.

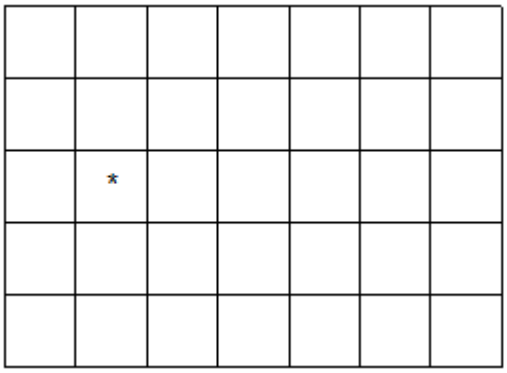
The cavern is also the lair of a fearsome monster that guards the flask. At the start of a new game the monster is asleep. As the cavern is dark the player cannot see where the monster is. If the player moves into the cell that contains the monster then the monster will wake up and eat the player and the player will have lost the game.

The monster has set two traps in its cavern. The player cannot see where these traps are. If the player moves into a cell that contains a trap then the monster will wake up. When the monster is awake, it will move around the cavern until it either eats the player or the player finds the flask. The monster is twice as fast as the player and makes two moves for each player move. Each move is one cell in one of the four possible directions. When it is awake the monster's skin glows so the player can see the position of the monster. The monster can see in the dark and so knows where the player is. The current position of the moving monster is indicated by an 'M' in the cavern displayed to the player.

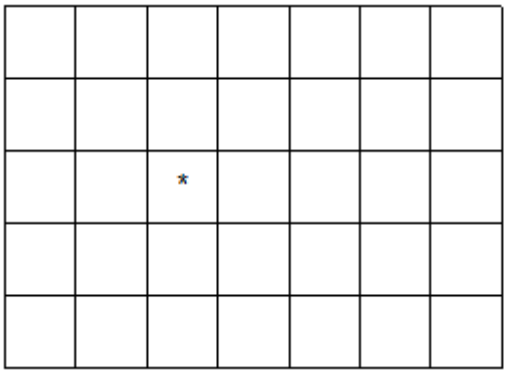
If the monster moves into the same cell as the flask, it kicks the flask out of the way and the flask will be moved to the cell the monster came from. When the monster is awake, it does not matter if the player (or the monster) moves into a cell containing one of the traps.

Figure 3 shows part of a possible game as displayed to the player. The player moves one cell to the east, which triggers a trap that wakes the monster. The player is then shown the position of the monster in the cavern. The monster then makes its first move and the new state of the cavern is shown. It then makes its second move and the updated state of the cavern is shown. It is then the player's turn to move.

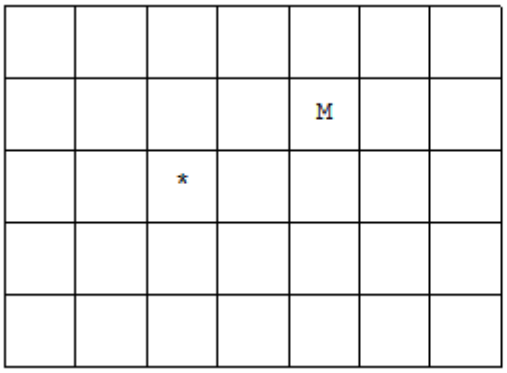
Figure 3



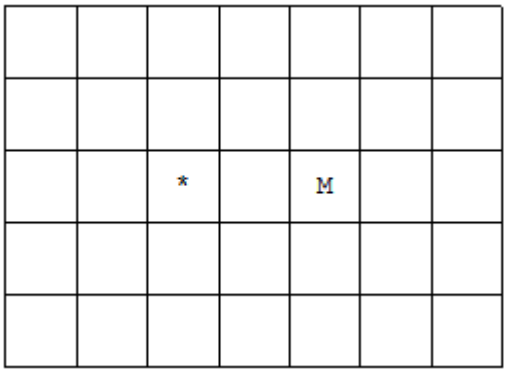
The position of the player is shown.



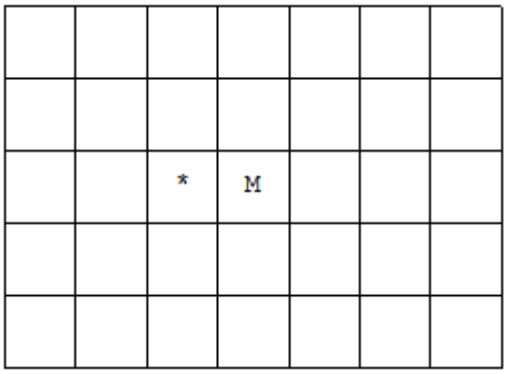
The player moves to the east.



This sets off a trap and wakes up the monster.



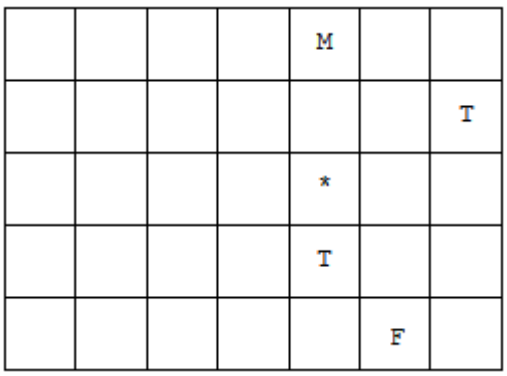
The monster makes a move.



The monster makes another move. Now it is the player's turn again.

In the Skeleton Program there is a menu containing **five** options: 'Start new game', 'Load game', 'Save game', 'Play training game' and 'Quit'. If the user chooses 'Load game' then the contents of a user-specified file are loaded and the user will start playing MONSTER! from the game state saved in the file. If the user chooses 'Save game' then they will be asked to enter a name for the file and then the current state of the game will be stored in a file with the name supplied by the user. If the user chooses 'Play training game' then the user will start playing MONSTER! from the game state shown in **Figure 4**.

Figure 4



'F' denotes the position of the flask; 'T' denotes the position of a trap. The flask, traps and monster are not displayed to the player when the training game starts.

Q17.

(a) This question refers to the subroutines `CheckValidMove` and `PlayGame`.

The **Skeleton Program** currently does not make all the checks needed to ensure that the move entered by a player is an allowed move. It should **not** be possible to

make a move that takes a player outside the 7×5 cavern grid.

The **Skeleton Program** is to be adapted so that it prevents a player from moving north if they are at the northern end of the cavern.

The subroutine `CheckValidMove` needs to be adapted so that it returns a value of `False` if a player attempts to move north when they are at the northern end of the cavern.

The subroutine `PlayGame` is to be adapted so that it displays an error message to the user if an illegal move is entered. The message should state "`That is not a valid move, please try again.`".

Evidence that you need to provide

(i) Your amended PROGRAM SOURCE CODE for the subroutine `CheckValidMove`.

(4)

(ii) Your amended PROGRAM SOURCE CODE for the subroutine `PlayGame`.

(1)

(iii) SCREEN CAPTURE(S) for a test run showing a player trying to move north when they are at the northern end of the cavern.

(1)

(b) This question refers to the `PlayGame` subroutine and will extend the functionality of the game.

A scoring system is to be implemented as a game of MONSTER! is played. A variable called `Score` will be used to store the current score of each player.

The final score will be displayed to the user at the end of the game. At the end of the game, either the player will have found the flask or the player will have been eaten by the monster.

The final score should be displayed with the message "`Your score was: Y`" where `Y` is the value of `Score`.

The scoring system will be based upon the following:

- each valid move by the player is +10 points
- finding the flask is +50 points
- setting off a trap is −10 points
- being killed by the monster is −50 points.

Task 1
Adapt the **Skeleton Program** so that the scoring system described above is implemented, with the value of `Score` being updated as indicated and the required message being displayed at the end of a game.

Task 2
Test that the changes you have made work by conducting the following test:

- play the training game
- move south
- move south

- move east.

Evidence that you need to provide

(i) Your amended PROGRAM SOURCE CODE for the subroutine `PlayGame` and (if relevant) the PROGRAM SOURCE CODE for any other subroutine(s) you have amended.

(8)

(ii) SCREEN CAPTURE(S) showing the required test.

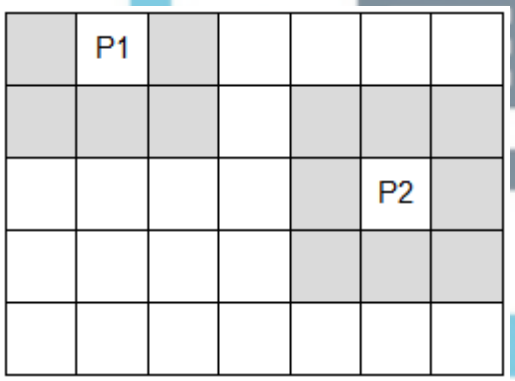
(1)

(c) This question will extend the functionality of the game.

The player will now have access to a close-range trap detector. After making a directional move in the cavern, the trap detector will perform a sweep of the neighbouring cells and report back if a trap is detected. Unfortunately, the detector can only detect the presence of a trap in a neighbouring cell, and not which individual cell the trap is in.

In **Figure 1** the shaded cells show the cells that would be scanned by the trap detector if the player were in the cell marked P1 or P2. The trap detector cannot scan outside the cavern.

Figure 1



Task 1
Create a new subroutine, `TrapDetector`, that, when given the current location of the player, returns `True` if a trap is in a neighbouring cell and `False` if there is no trap in a neighbouring cell.

When creating this subroutine you should ensure that your solution is efficiently coded.

Task 2
Modify the `PlayGame` subroutine so that after the player moves and the new state of the cavern is displayed:
- the message '`Trap detected`' is displayed if there is a trap in any neighbouring cell.
- the message '`No trap detected`' is displayed if there are no traps in any neighbouring cell.

Task 3
Test that your program works by loading the training game and showing that:
- a trap is detected after the player's first move, west

- a trap is detected after the player's second move, south
- a trap is not detected after the player's third move, west.

Evidence that you need to provide

(i) Your PROGRAM SOURCE CODE for the subroutine `TrapDetector`.

(12)

(ii) Your amended PROGRAM SOURCE CODE for the subroutine `PlayGame`.

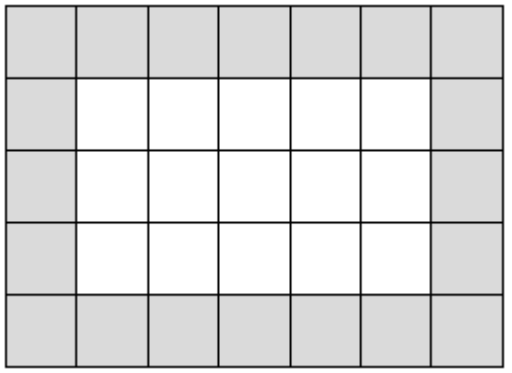
(1)

(iii) SCREEN CAPTURE(S) showing the required sequence of tests being carried out, with the trap detected message being displayed after each of the first two moves and the trap not detected message being displayed after the third move.

(1)

(iv) The game of MONSTER!, as represented by the **Skeleton Program**, is to be extended so that the cavern generated is not rectangular. The outer cells, shaded in **Figure 2**, will be randomly selected to be either rock or normal space when a new game starts. A cell that contains rock cannot be entered by the monster or player.

Figure 2



Describe changes that could be made to the **Skeleton Program** to achieve this.

In your answer you should ensure that you discuss changes to the data held in the `Cavern` variable and how the subroutines `ResetCavern` and `CheckValidMove` will need to be altered.

You are **not** expected to actually make the changes.

_____

_____

_____

_____

(5)

(v) A request has been made that the layout of the whole cavern should be more random. It has been suggested that all of the cells should be made a random choice between rock and normal space during setup.

Identify **two** problems that might occur with the MONSTER! game if this suggestion was made to the program.

_____

_____

_____

_____

(2)

(Total 35 marks)

Q18.

$\mathbb{R}$ denotes the set of real numbers, which includes the natural numbers, the rational numbers and the irrational numbers.

(a) Give **one** example of a natural number.

_____

(1)

(b) Give **one** example of an irrational number.

_____

(1)

(Total 2 marks)

Q19.

(a) What is the decimal equivalent of the hexadecimal number $D6_{16}$? Show your working.

_____

_____

_____

_____

(2)

(b) Represent the decimal value 9.375_{10} as an unsigned binary fixed point number, with 4 bits before and 4 bits after the binary point.

_____

_____

_____

_____

(2)

(c) Represent the decimal value -67_{10} as an **8-bit two's complement binary integer.**

_____

_____

_____

_____

(2)

(d) A computer represents numbers using 8-bit two's complement binary.

Using this representation perform the calculation:

01001000_2
01100011_2 +

_____

Answer: _____

(1)

(e) What problem has resulted from performing the calculation using 8-bit two's complement binary?

_____

_____

(1)

(Total 8 marks)

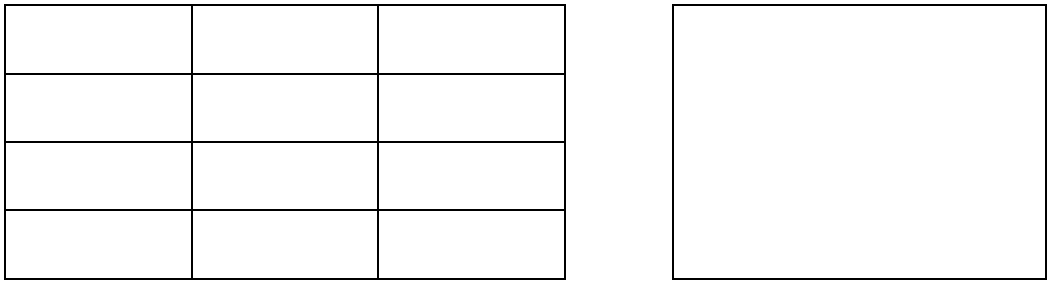
Q20.

(a) Complete the table below and draw the symbol for an AND gate in the box.

Truth table for an AND gate

Input A	Input B	Output

AND gate symbol



(2)

(b) Using the laws of Boolean algebra, simplify the following Boolean expression.

A.B. (A + B)

Answer _____

(3)

(c) Using the laws of Boolean algebra, simplify the following Boolean expression.

$(X + Y).(X + \overline{Y})$

Answer _____

(3)

(Total 8 marks)

Q21.

The famous detective John Stout was called in to solve a perplexing murder mystery. He determined the following facts.

a Nathan, the murdered man, was killed by a blow on the head.
b Either Suzanne or Martin was in the dining room at the time of the murder.
c If Peter was in the kitchen at the time of the murder, then Ian killed Nathan using poison.
d If Suzanne was in the dining room at the time of the murder, then Steve killed Nathan.
e If Peter was not in the kitchen at the time of the murder, then Martin was not in

the dining room when the murder was committed.

f If Martin was in the dining room at the time the murder was committed, then Paul killed Nathan.

g If Kevin was in the hall at the time of the murder, then Suzanne killed Nathan by a blow to the neck with a saucepan.

(a) Who murdered Nathan?

A Paul
B Steve
C Suzanne
D Ian
E It is not possible for John Stout to solve the crime.

(1)

(b) Explain how you know your answer to (a) is correct.

Use the space below for rough working.

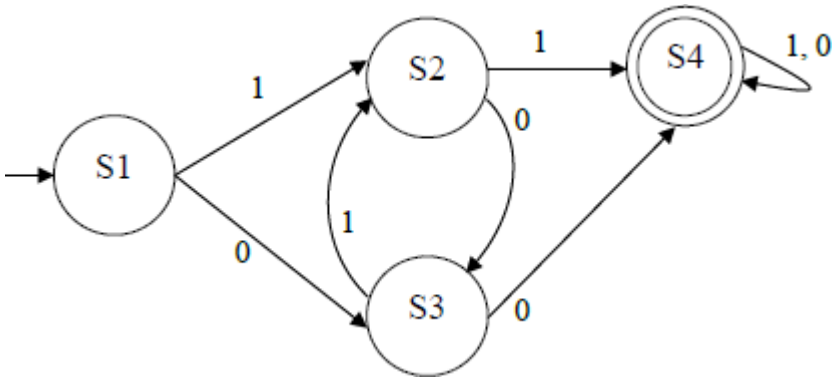
(2)

(Total 3 marks)

Q22.

A finite state machine (FSM) can be used to define a language: a string is allowed in a language if it is accepted by the FSM that represents the rules of the language.
Figure 1 shows the state transition diagram for an FSM.

Figure 1



An FSM can be represented as a state transition diagram or as a state transition table. The table below is an incomplete state transition table for **Figure 1** .

(a) Complete the table.

Original state	Input	New state
S3		
S3		

(1)

(b) Any language that can be defined using an FSM can also be defined using a regular expression.

The FSM in **Figure 1** defines the language that allows all strings containing at least, either two consecutive 1s or two consecutive 0s.

The strings 0110, 00 and 01011 are all accepted by the FSM and so are valid strings in the language.

The strings 1010 and 01 are not accepted by the FSM and so are not valid strings in the language.

Write a regular expression that is equivalent to the FSM shown in **Figure 1**.

_____

_____

_____

_____

_____

_____

(3)

(c) Backus-Naur Form (BNF) can be used to define the rules of a language.

Figure 2 shows an attempt to write a set of BNF production rules to define a language of full names.

Figure 2

Note: underscores (_) have been used to denote spaces.
Note: rule numbers have been included but are not part of the BNF rules.

Rule number	
1	<fullname> ::= <title>_<name>_<endtitle> | <name> | <title>_<name> | <name>_<endtitle>
2	<title> ::= MRS | MS | MISS | MR | DR | SIR
3	<endtitle> ::= ESQUIRE | OBE | CBE
4	<name> ::= <word> | <name>_<word>
5	<word> ::= <char><word>
6	<char> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

BNF can be used to define languages that are not possible to define using regular expressions. The language defined in **Figure 2** could not have been defined using regular expressions.

Complete the table below by writing either a **'Y'** for **Yes** or **'N'** for **No** in each row.

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

(1)

(d) There is an error in rule 5 in **Figure 2** which means that no names are defined by the language.

Explain what is wrong with the production rule and rewrite the production rule so that the language does define some names – the names 'BEN D JONES', 'JO GOLOMBEK' and 'ALULIM' should all be defined.

_____

_____

_____

_____

(2)

(Total 7 marks)

Q23.

Create a folder / directory for your new program.

One method for converting a decimal number into binary is to repeatedly divide by 2 using integer division. After each division is completed, the remainder is output and the integer result of the division is used as the input to the next iteration of the division process. The process repeats until the result of the division is 0.

Outputting the remainders in the sequence that they are calculated produces the binary digits of the equivalent binary number, but in reverse order.

For example, the decimal number 210 could be converted into binary as shown below.

```
210 ÷ 2 =105   remainder 0
105 ÷ 2 =  52  remainder 1
 52 ÷ 2 =  26  remainder 0
 26 ÷ 2 = 13   remainder 0
 13 ÷ 2 =  6   remainder 1
  6 ÷ 2 =  3   remainder 0
  3 ÷ 2 =  1   remainder 1
  1 ÷ 2 =  0   remainder 1
```

The sequence 0, 1, 0, 0, 1, 0, 1, 1 which would be output by this process is the reverse of the binary equivalent of 210 which is 11010010.

What you need to do

Task 1
Write a program that will perform the conversion process described above. The program should display a suitable prompt asking the user to input a decimal number to convert and then output the bits of the binary equivalent of the decimal number in reverse order.

Task 2
Improve the program so that the bits are output in the correct order, e.g. for 210 the output would be 11010010.

Task 3
Test the program works by entering the value 210.

Save the program in your new folder / directory.

Evidence that you need to provide

(a) Your PROGRAM SOURCE CODE after you have completed both **Task 1** and **Task 2**.

If you complete **Task 1** but do not attempt **Task 2** then a maximum of 9 marks will be awarded.

(12)

(b) SCREEN CAPTURE(S) for the test showing the output of the program when 210 is entered.

The marks for this test will be awarded whether the binary digits are output in reverse order or in the correct order.

(2)

(Total 14 marks)

Q24.

Based on the description in Statius' journal you are sure that this must be the right place. The blue-green moss covering the rocks and the dense tree foliage combine to conceal the cave entrance; you almost walked straight past it and it was only through luck that you saw it. There isn't any time to waste – ever since the journal was discovered, everyone has been looking for this place. The thick cobwebs across the entrance prove that you must be the first one here. You know that, if you are right, you could be the one that finds, in the cavern below the mountain, the single draft of Styxian potion contained in Statius' flask. The journal says that there is a fearsome beast lying in wait, but the risk is worth it. Statius wrote that consuming the potion would grant the drinker invulnerability. Nothing could hurt you, cut you, graze, scratch or bruise you. Your thoughts start to drift and you imagine what you could do with such power.

"Snap out of it," you tell yourself. Someone else could find this place and you can't take that risk; the flask contains only enough potion for one. Quickly you shoulder your pack, then you light your torch, brush aside some cobwebs and step into the cave.

Inside, the ground is rough and you stumble several times. As you go deeper into the mountain, the cave darkens and soon the only light is coming from your torch. After you have been walking for a few minutes the cave widens and then ends abruptly. You take out your copy of the journal. You read the description of the cave again. It says that, at the end of the cave, there is a large fissure near the western part of the wall and that the cavern, where the flask is hidden, is at the bottom of the fissure. You move over to the west side of the cave. Sure enough, the fissure is there. You take off your pack and then move carefully nearer the edge. The light from the torch does not reach far enough down to reveal the bottom, but you can see that the fissure walls are too steep to get down unaided so you will need your climbing equipment. You place your torch carefully on the floor nearby and take your rope, pitons and carabiners out of your pack.

Your foot slips on some loose stones and you fall backwards into the fissure. You tumble down the hole in a shower of dust and pebbles, falling into the cavern below. You land painfully on the rocky floor.

It is dark; your torch is back in the cave and it weakly illuminates your immediate surroundings but you cannot see any farther into the cavern. You become aware of a sonorous noise around you and it takes you a few minutes to work out what it is. The monster is asleep somewhere in the cavern and is snoring loudly. The sound reverberates around you so you can't work out in which direction the monster lies.

You can see the bottom aperture of the fissure several metres above your head and try to scramble up the wall to reach it, but the rock face is too sheer and you can't get sufficient purchase. From what you can remember of the journal, you must be at the north-west corner of the cavern.

You make your mind up. You can't get out of the cavern the way that you came in and the monster could wake soon. You decide to explore the cavern. Maybe you will find another way out. Maybe you will find the Styxian potion. You have no choice but to play the game of...

MONSTER!

MONSTER!

The **Skeleton Program** in this Preliminary Material is a program for the one-player game MONSTER!.

When playing MONSTER! the player starts in the north-west corner of a dark cavern. The cavern is represented by a 7 × 5 rectangular grid of cells. The player's current position is indicated by an *. The player is presented with a list of five options: they can either return to the main menu (where they can save the current game if they want to) or they can move one cell in one of four possible directions. If they enter 'N' they will move one cell to the north, 'S' will move them one cell to the south, 'W' will move them one cell to the west and 'E' will move them one cell to the east. The initial position of a new game is shown in **Figure 1**.

Figure 1

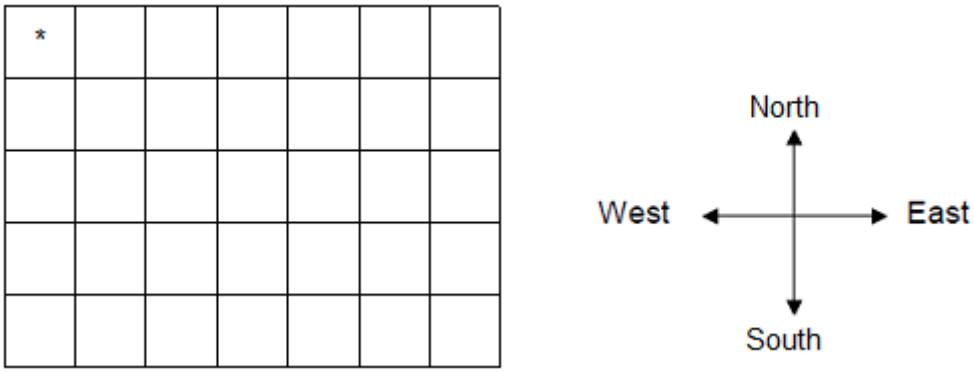
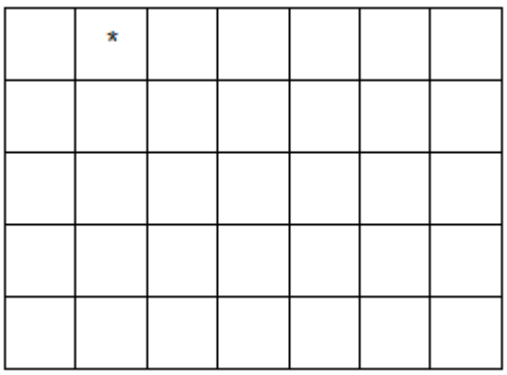


Figure 2 shows a new state, resulting from the user selecting 'E' in the starting position.

Figure 2



The aim of MONSTER! is to find the hidden treasure (a flask containing a Styxian potion) that is in one of the cells of the cavern. Unfortunately, the cavern is dark and the only way that a player can find the flask is to move around until they are in the same cell as the flask. When a new game is started, the flask will be in a random position in the cavern (though it won't be in the same cell as the player starts in). If the player moves into the cell that contains the flask, then they win the game.

The cavern is also the lair of a fearsome monster that guards the flask. At the start of a new game the monster is asleep. As the cavern is dark the player cannot see where the

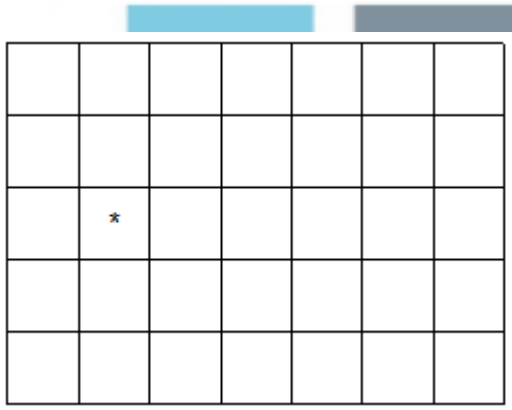
monster is. If the player moves into the cell that contains the monster then the monster will wake up and eat the player and the player will have lost the game.

The monster has set two traps in its cavern. The player cannot see where these traps are. If the player moves into a cell that contains a trap then the monster will wake up. When the monster is awake, it will move around the cavern until it either eats the player or the player finds the flask. The monster is twice as fast as the player and makes two moves for each player move. Each move is one cell in one of the four possible directions. When it is awake the monster's skin glows so the player can see the position of the monster. The monster can see in the dark and so knows where the player is. The current position of the moving monster is indicated by an 'M' in the cavern displayed to the player.

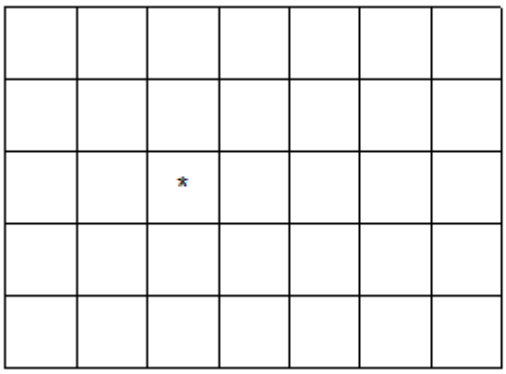
If the monster moves into the same cell as the flask, it kicks the flask out of the way and the flask will be moved to the cell the monster came from. When the monster is awake, it does not matter if the player (or the monster) moves into a cell containing one of the traps.

Figure 3 shows part of a possible game as displayed to the player. The player moves one cell to the east, which triggers a trap that wakes the monster. The player is then shown the position of the monster in the cavern. The monster then makes its first move and the new state of the cavern is shown. It then makes its second move and the updated state of the cavern is shown. It is then the player's turn to move.

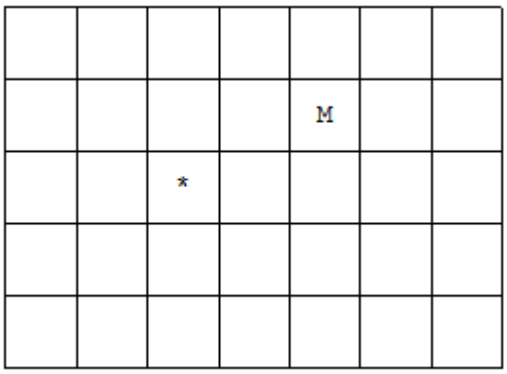
Figure 3



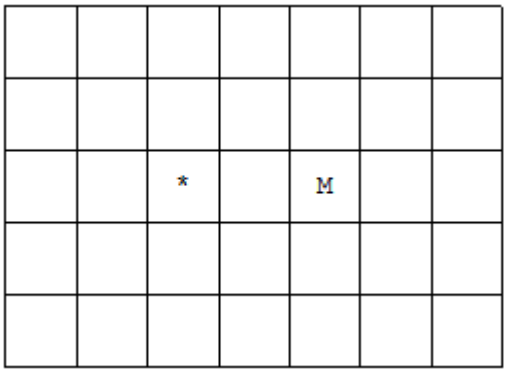
The position of the player is shown.



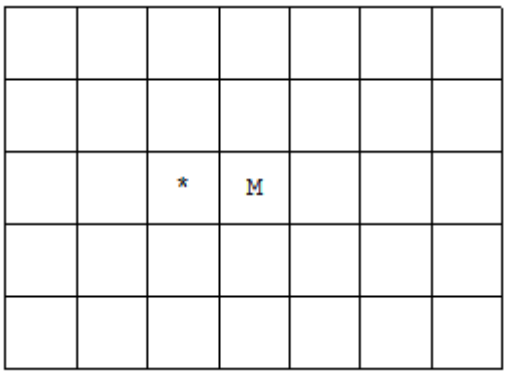
The player moves to the east.



This sets off a trap and wakes up the monster.



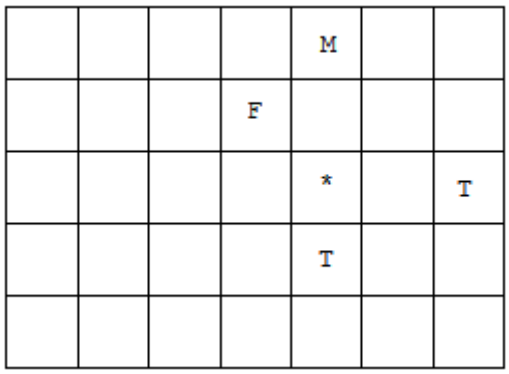
The monster makes a move.



The monster makes another move. Now it is the player's turn again.

In the Skeleton Program there is a menu containing **three** options: 'Start new game', 'Play training game' and 'Quit'. If the user chooses 'Play training game' the user will start playing MONSTER! from the game state shown in **Figure 4**.

Figure 4



'F' denotes the position of the flask; 'T' denotes the position of a trap. The flask, traps and monster are not displayed to the player when the training game starts.

Q25.

(a) This question refers to the subroutines `CheckValidMove` and `Play` in the `Game` class.

The **Skeleton Program** currently does not make all the checks needed to ensure that the move entered by a player is an allowed move. It should not be possible to make a move that takes a player outside the 7 × 5 cavern grid.

The **Skeleton Program** needs to be adapted so that it prevents a player from moving west if they are at the western end of the cavern.

The subroutine `CheckValidMove` needs to be adapted so that it returns a value of FALSE if a player attempts to move west when they are at the western end of the cavern.

The subroutine `Play` needs to be adapted so that it displays an error message to the user if an illegal move is entered. The message should state `"That is not a valid move, please try again"`.

Evidence that you need to provide

(i) Your amended PROGRAM SOURCE CODE for the subroutine `CheckValidMove`.

(3)

(ii) Your amended PROGRAM SOURCE CODE for the subroutine `Play`.

(2)

(iii) SCREEN CAPTURE(S) for a test run showing a player trying to move west when they are at the western end of the cave.

(1)

(b) This question will extend the functionality of the game.

The game is to be altered so that there is a new type of enemy: a sleepy enemy. A sleepy enemy is exactly the same as a normal enemy, except that after making four moves it falls asleep again.

Task 1
Create a new class called `SleepyEnemy` that inherits from the `Enemy` class.

Task 2
Create a new integer attribute in the `SleepyEnemy` class called `MovesTillSleep`.

Task 3
Create a new public subroutine in the `SleepyEnemy` class called `ChangeSleepStatus`. This subroutine should override the `ChangeSleepStatus` subroutine from the `Enemy` class. The value of `MovesTillSleep` should be set to 4 in this subroutine.

Task 4
Create a new public subroutine in the `SleepyEnemy` class called `MakeMove`. This

subroutine should override the `MakeMove` subroutine from the `Enemy` class. When called this subroutine should reduce the value of `MovesTillSleep` by 1 and then send the monster to sleep if `MovesTillSleep` has become equal to 0.

Task 5
Modify the `Game` class so that the Monster object is of type `SleepyEnemy` (instead of `Enemy`) .

Task 6
Check that the changes you have made work by conducting the following test:

- play the training game
- move east
- move east
- move south.

Evidence that you need to provide

(i) Your PROGRAM SOURCE CODE for the new `SleepyEnemy class`.

(8)

(ii) SCREEN CAPTURE(S) showing the requested test.

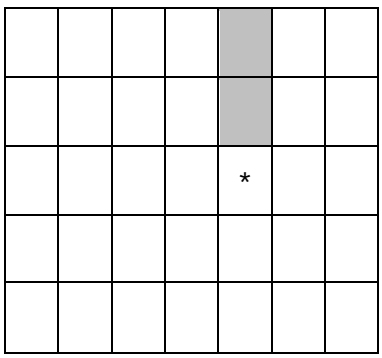
(2)

(c) This question refers to the `Game` and `Character` classes and will extend the functionality of the game.

The game should be altered so that once per game the player can shoot an arrow instead of making a move in the cavern. The arrow travels in a straight line, in a direction of the player's choice, from the cell the player is in to the edge of the cavern. If the arrow hits the monster then the player wins the game and a message saying that they have shot the monster should be displayed.

For this question you are **only** required to extend the program so that it checks if the monster is hit by the arrow when the user chooses to shoot an arrow northwards. However, the user should be able to select any of the four possible directions.

In the diagram below, the two shaded cells show the cells which, if the monster is in one of them, would result in the player winning the game, as long as the player is in the cell five to the east and three to the south and chooses to shoot an arrow northwards.



Task 1
Modify the `DisplayMoveOptions` subroutine in the `Game` class so that the option to enter A to shoot an arrow is added to the menu.

Task 2
Create a new Boolean attribute called `HasArrow` in the `Character` class.

The value of `HasArrow` should be set to `True` when a new object of class `Character` is instantiated.

Task 3
Create a new public subroutine called `GetHasArrow` in the `Character` class that returns the value of the `HasArrow` attribute to the calling routine.

Task 4
Modify the `CheckValidMove` subroutine in the `Game` class so that:

- it is a valid move if A is selected and the player does have an arrow
- it is not a valid move if A is selected and the player does not have an arrow.

Task 5
Create a new public subroutine called `GetArrowDirection` in the `Character` class.

This subroutine should return a character to the calling routine.

The user should be asked in which direction they would like to shoot an arrow (N, S, E or W) and the value entered by the user should be returned to the calling routine.

If an invalid direction is entered then the user should be repeatedly asked to enter a new direction, until a valid direction is entered.

The value of `HasArrow` should then be changed to FALSE.

Task 6
Modify the `Play` subroutine in the `Game` class so that if the move chosen by the user is not M it then checks if the move chosen is A.

If the move chosen was A, then there should be a call to the player's `GetArrowDirection` subroutine. If the user chooses a direction of N then the program should check to see if the monster is in one of the squares directly north of the player's current position. If it is then a message saying "`You have shot the monster and it cannot stop you finding the flask`" should be displayed. The value of `FlaskFound` should then be set to TRUE.

After the arrow has been shot, if the monster is still alive and awake, it is now the monster's turn to move, the player should remain in the same cell as they were in before the arrow was shot.

There is **no** need to write any code that checks if the monster has been shot when the player chooses to shoot either to the east, to the west or to the south.

Task 7: test 1
Test that the changes you have made work by conducting the following test:

- play the training game
- shoot an arrow
- choose a direction of N for the arrow.

Task 8: test 2
Test that the changes you have made work by conducting the following test:

- play the training game
- move east
- shoot an arrow
- choose a direction of N for the arrow
- shoot an arrow.

Evidence that you need to provide

(i) Your amended PROGRAM SOURCE CODE for the subroutine `DisplayMoveOptions`.

(1)

(ii) Your amended PROGRAM SOURCE CODE for the subroutine `CheckValidMove`.

(2)

(iii) Your amended PROGRAM SOURCE CODE for the class `Character`.

(8)

(iv) Your amended PROGRAM SOURCE CODE for the subroutine `Play`.

(6)

(v) SCREEN CAPTURE(S) showing the results of **Test 1**.

(1)

(vi) SCREEN CAPTURE(S) showing the results of **Test 2**.

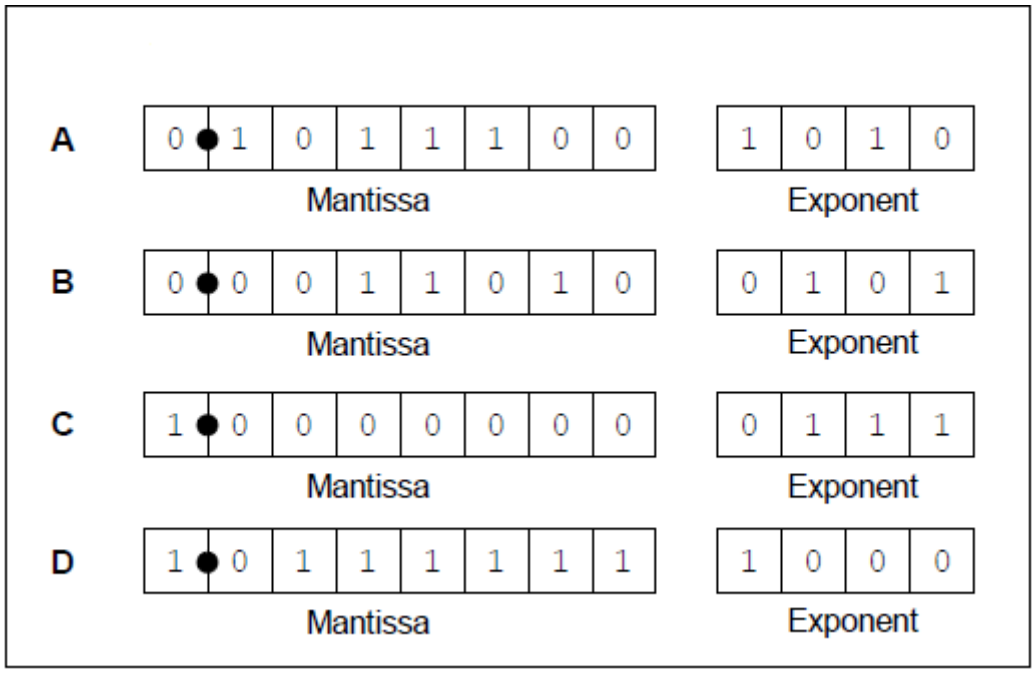
(1)

(Total 35 marks)

Q26.

A particular computer uses a **normalised** floating point representation with an 8-bit mantissa and a 4-bit exponent, both stored using **two's complement**.

Four bit patterns that are stored in this computer's memory are listed in the figure below and are labelled **A, B, C, D**. Three of the bit patterns are valid floating point numbers and one is not.



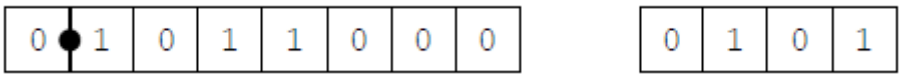
(a) Complete the table below. In the Correct letter (**A-D**) column shade the appropriate lozenge **A, B, C or D** to indicate which bit pattern from above is an example of the type of value described in the Value description column.

Do **not** use the same letter more than once.

Value description	Correct letter (A-D)
A positive normalised value	A ○ B ○ C ○ D ○
The most negative value that can be represented	A ○ B ○ C ○ D ○
A value that is not valid in the representation because it is not normalised	A ○ B ○ C ○ D ○

(3)

(b) The following is a floating point representation of a number:



Mantissa Exponent

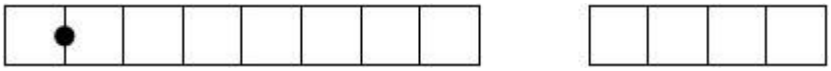
Calculate the decimal equivalent of the number. Show how you have arrived at your answer.

Answer _____

(2)

(c) Write the normalised floating point representation of the negative decimal value -6.75 in the boxes below. Show how you have arrived at your answer.

Answer:

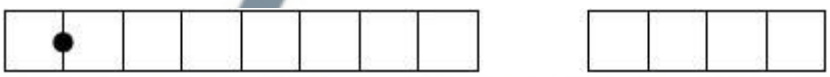


Mantissa Exponent

(3)

(d) An alternative two's complement format representation is proposed. In the alternative representation 6 bits will be used to store the mantissa and 6 bits will be used to store the exponent.

Existing Representation (8-bit mantissa, 4-bit exponent):



Mantissa Exponent

Proposed Alternative Representation (6-bit mantissa, 6-bit exponent):

Mantissa Exponent

Explain the effects of using the proposed alternative representation instead of the existing representation.

(2)

(Total 10 marks)

Q27.

(a) The table below lists six Boolean equations. Three of them are correct, the others are not. Shade the lozenges next to the **three** equations that are correct.

Equation	Correct? (Shade three)
$A \cdot \overline{A} = 1$	⬭
$A + B = \overline{\overline{A} \cdot \overline{B}}$	⬭
$A + 1 = 1$	⬭
$A \cdot (A + B) = A$	⬭
$A + (A \cdot B) = B$	⬭
$A \cdot 1 = 1$	⬭

(3)

(b) Use Boolean algebra to simplify the following expression:

$$\overline{\overline{A} + \overline{B} + B \cdot \overline{A}}$$

Show your working.

_____

_____

_____

_____

_____

Answer _____

(3)

(Total 6 marks)

Q28.

Create a folder / directory in this **question** for your new program.
The algorithm, represented using pseudo-code below, and the variable table, describe a program that calculates and displays all of the prime numbers between 2 and 50, inclusive.

The `MOD` operator calculates the remainder resulting from an integer division
eg `10 MOD 3 = 1`.

If you are unsure how to use the `MOD` operator in the programming language you are

using, there are examples of it being used in the **Skeleton Program**.

```
OUTPUT "The first few prime numbers are:"
FOR Count1 ← 2 TO 50 DO
  Count2 ← 2
  Prime ← "Yes"
  WHILE Count2 * Count2 <= Count1 DO
    IF (Count1 MOD Count2 = 0) THEN
      Prime ← "No"
    ENDIF
    Count2 ← Count2 + 1
  ENDWHILE
  IF Prime = "Yes" THEN
    OUTPUT Count1
  ENDIF
ENDFOR
```

Identifier	Data Type	Purpose
`Count1`	`Integer`	Stores the number currently being checked for primeness
`Count2`	`Integer`	Stores a number that is being checked to see if it is a factor of `Count1`
`Prime`	`String`	Indicates if the value stored in `Count1` is a prime number or not

What you need to do

Write a program for the algorithm above.

Run the program and test that it works correctly.

Save the program in your new **Question** folder / directory.

Evidence that you need to provide

(a) Your PROGRAM SOURCE CODE.

(11)

(b) SCREEN CAPTURE(S) for the test showing the correct working of the program.

(1)

(c) Describe the changes that would need to be made to the algorithm shown above, so that instead of displaying the prime numbers between 2 and 50, inclusive, it displays all the prime numbers between 2 and a value input by the user, inclusive.

_____

_____

_____

_____

_____

_____

(3)

(Total 15 marks)

Q29.

State the name of an identifier for:

(a) a named constant

_____

_____

(1)

(b) a user-defined subroutine that has only one parameter

_____

_____

(1)

(c) a two-dimensional array or an equivalent list.

_____

_____

(1)

(d) Look at the `MakeMove` subroutine.

Describe how this subroutine could be rewritten so that instead of there being three lines of code that change the value in the start square, there could be just one line of code that does this. The functionality of the `MakeMove` subroutine should not be altered by the changes you describe.

_____

_____

(1)

(e) Look at the last selection structure in the MAIN PROGRAM BLOCK.

What is the purpose of this selection structure?

_____

_____

_____

_____

(2)

(f) The logic of a selection structure can be represented using a decision table.
The table shows an attempt to represent the logic of this selection structure.

Conditions	>= 97	Y	N	Y	Y
	<= 123	N	Y	N	Y
Action	Change value of `PlayAgain`		X	X	X

Explain why this decision table is not an accurate representation of the logic of this selection structure.

_____

_____

_____

_____

_____

_____

(3)

(Total 9 marks)

Q30.

(a) This question refers to the MAIN PROGRAM BLOCK and will extend the functionality of the **Skeleton Program**.

The number of moves made by the players in a game of CAPTURE THE SARRUM will be tracked. A variable called `NoOfMoves` will be used to store the number of moves completed by the players. At the start of every game `NoOfMoves` should be set to an initial value of zero.

Each time a player enters a legal move `1` should be added to the value stored in `NoOfMoves`.

After the call to the `MakeMove` subroutine, the `NoOfMoves` variable should be updated and then a message should be displayed saying `"The number of moves completed so far: n"` – where **`n`** is the value of `NoOfMoves`.

Test that the changes you have made work:

- run the **Skeleton Program**
- enter `N` when asked if you want to play the sample game
- enter a start square of `17` and a finish square of `16`
- enter a start square of `12` and a finish square of `13`
- enter a start square of `18` and a finish square of `17`.

Evidence that you need to provide

(i) Your PROGRAM SOURCE CODE for the amended MAIN PROGRAM BLOCK.

(4)

(ii) SCREEN CAPTURE(S) showing the requested test.

(2)

(b) This question refers to the subroutine `CheckMoveIsLegal`.

When the user has entered the start square and the finish square for their move, a number of checks are made to see if their intended move is legal.

Add a validation check to the subroutine `CheckMoveIsLegal`, so that a move is accepted as being legal only if the **finish square** refers to a square that is on the board.

Test that the changes you have made work:

- run the **Skeleton Program**
- enter `N` when asked if you want to play the sample game
- enter a start square of `88` and a finish square of `98`
- enter a start square of `18` and a finish square of `19`
- enter a start square of `87` and a finish square of `86`.

Evidence that you need to provide

(i) Your PROGRAM SOURCE CODE for the amended subroutine `CheckMoveIsLegal`.

(4)

(ii) SCREEN CAPTURE(S) showing the requested test. You must make sure that evidence for all parts of the requested test by the user is provided in the SCREEN CAPTURE(S).

(3)

(c) This question will extend the functionality of the game.

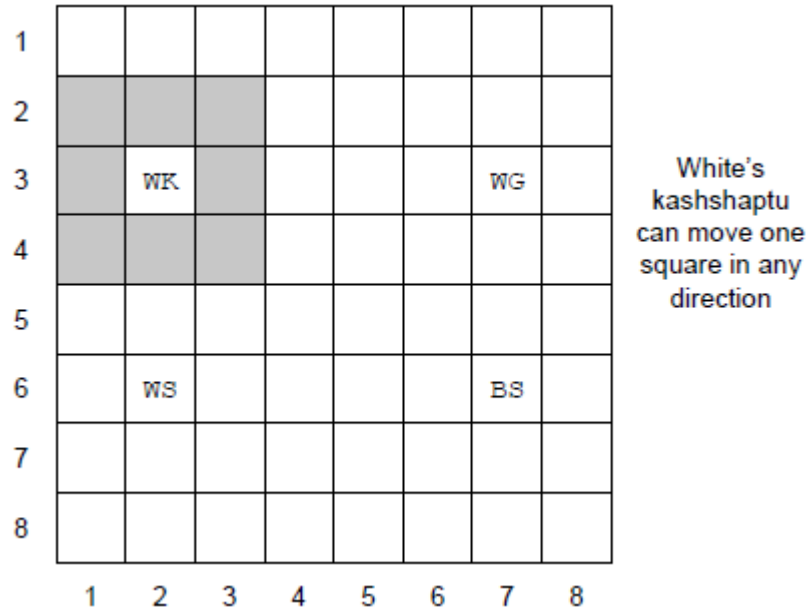
A new type of piece is to be added to the game – a kashshaptu (witch). When a redum is promoted, instead of changing into a marzaz pani it changes into a kashshaptu.

A kashshaptu moves in the same way as a sarrum (one square in any direction). Like all the other pieces, a kashshaptu cannot move into a square containing a piece of the same colour and when a kashshaptu is moved into a square containing the opponent's sarrum, it captures the sarrum and the game is over.

When a move would cause the kashshaptu to enter a square containing one of the opponent's pieces (except the sarrum), the kashshaptu stays in its start square and the opponent's piece changes colour and now belongs to the player who played the kashshaptu. It then becomes the other player's turn.

Figures 1 and **2** show diagrams giving examples of legal kashshaptu moves. A kashshaptu is represented by the symbol `K`.

Figure 1



White enters a start square of 23 and a finish square of 22. The board should now look like this:

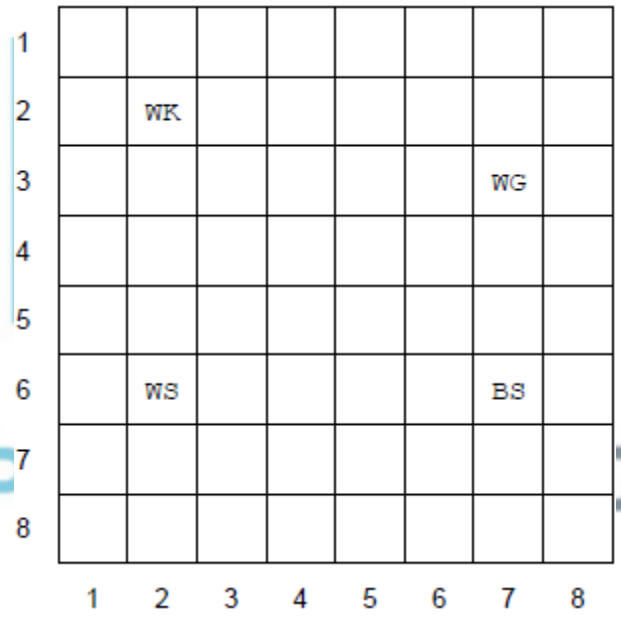
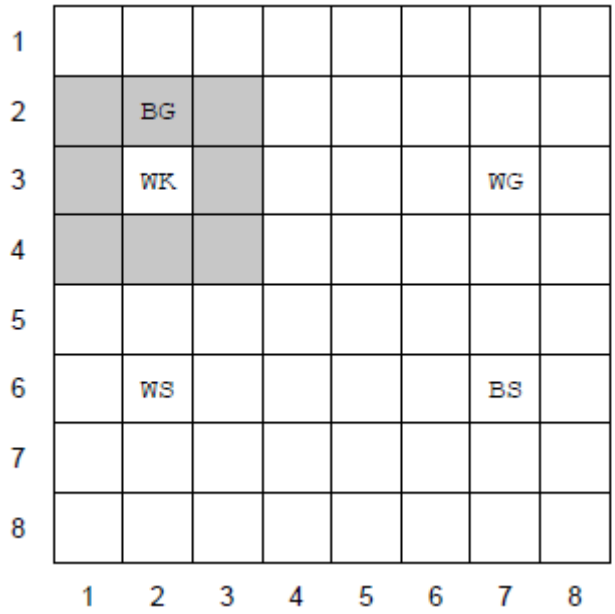
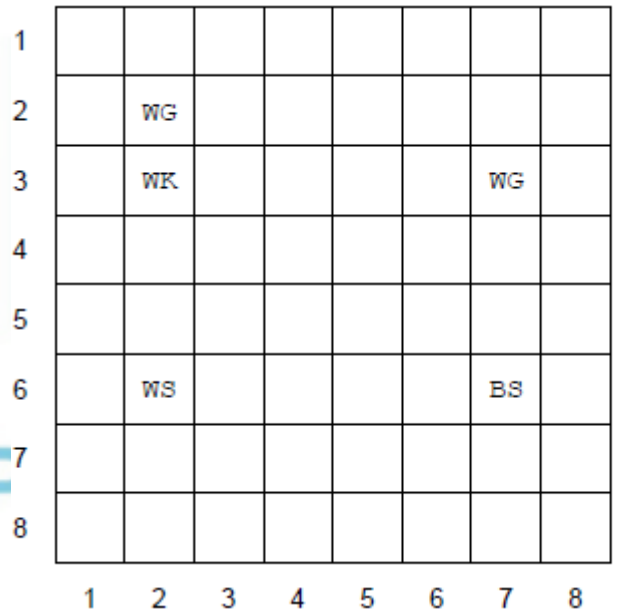


Figure 2



White's kashshaptu can move one square in any direction

White enters a start square of 23 and a finish square of 22. The board should now look like this:



White's kashshaptu stays in its original square and the black gisgigir has now become a white gisgigir

Task 1

Adapt the program source code for the `CheckMoveIsLegal` subroutine, so that a kashshaptu (represented by the letter `K`) has the same legal moves as a sarrum.

Evidence that you need to provide

(i) Your PROGRAM SOURCE CODE for the amended subroutine `CheckMoveIsLegal`.

(1)

Task 2

Adapt the program source code for the `MakeMove` subroutine, so that when a redum is promoted, it becomes a kashshaptu (represented by the symbol `K`) and so that a kashshaptu moves and captures in the way described.

You need to adapt **only** the program source code so that the player with the **White**

pieces can promote a redum to a kashshaptu and move a kashshaptu.

Task 3
Test that the changes you have made work:

- run the **Skeleton Program**
- enter `Y` when asked if you want to play the sample game
- enter a start square of `12` and a finish square of `11`
- enter a start square of `41` and a finish square of `31`
- enter a start square of `11` and a finish square of `21`
- enter a start square of `31` and a finish square of `22`
- enter a start square of `11` and a finish square of `22`.

Evidence that you need to provide

(ii) Your PROGRAM SOURCE CODE for the amended subroutine `MakeMove`.

(5)

(iii) SCREEN CAPTURE(S) showing the requested test.

(3)

(d) This question will further extend the functionality of the **Skeleton Program**.

You can attempt this question regardless of whether or not you produced a solution to **Question (c)** that correctly added the kashshaptu piece to the game.

Forsyth-Edwards Notation (FEN) is a standard notation used to describe a particular board position of a chess game. The purpose of FEN is to provide all the information necessary to represent the current state of a game of chess, so that it can be restarted from a particular position.

FEN can be adapted to represent game positions from chess variants like CAPTURE THE SARRUM.

FEN defines a particular game position in one line of text using ASCII characters. This line of text can then be saved into a file or copied for use in another program. To represent a game position in CAPTURE THE SARRUM, the FEN needs to represent two items of information about the game position – the board state (what pieces are on what squares) and which player's turn it is.

The first item in a FEN record for CAPTURE THE SARRUM is the board state. The board state is represented rank by rank – starting with rank 1 and ending with rank 8. Within each rank the contents of each square are described – starting with file 1 and ending with file 8.

Each piece is identified by a single letter (`S` = sarrum, `M` = marzaz pani, `G` = gisgigir, `E` = etlu, `N` = nabu, `R` = redum, `K` = kashshaptu). White pieces are designated using uppercase letters (`SMGENRK`) and black pieces are designated using lowercase letters (`smegnrk`). Empty squares are represented using the digits 1 to 8 where the digit represents the number of consecutive empty squares.

A / character is used to indicate the end of a rank.

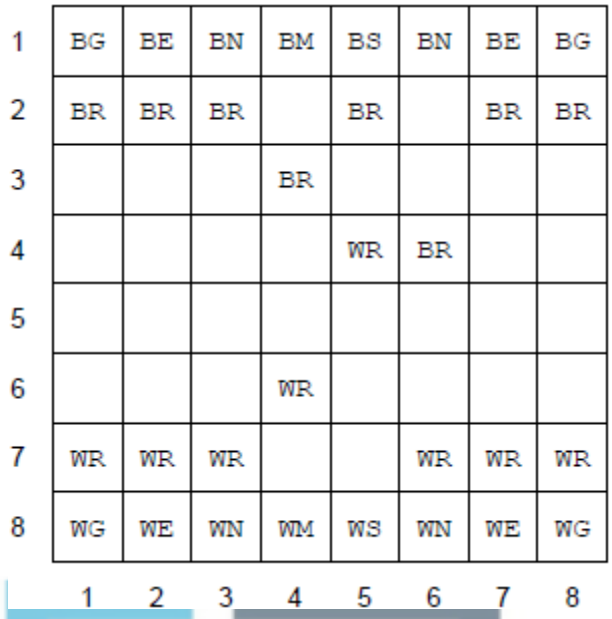
After the board state, there will be either a `W` or `B` character to indicate if it is White's or Black's turn.

Figure 3 shows an example board position from a game and the equivalent FEN

record.

Figure 4 shows the initial board position and the equivalent FEN record.

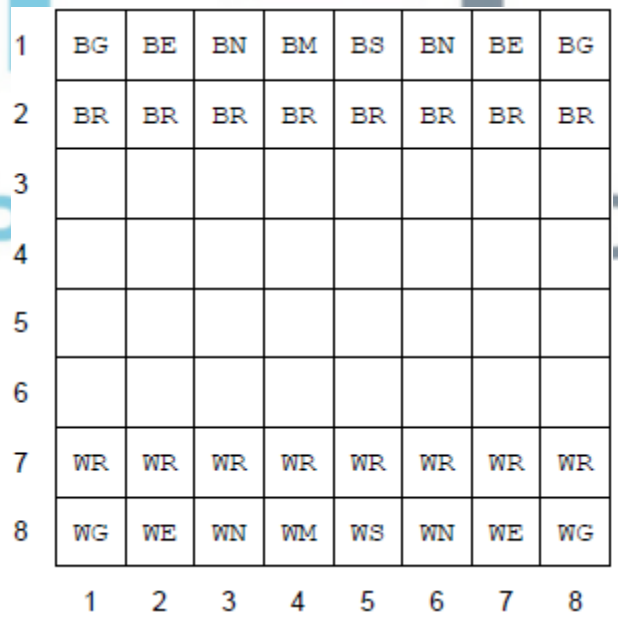
Figure 3



	1	2	3	4	5	6	7	8
1	BG	BE	BN	BM	BS	BN	BE	BG
2	BR	BR	BR		BR		BR	BR
3				BR				
4					WR	BR		
5								
6				WR				
7	WR	WR	WR			WR	WR	WR
8	WG	WE	WN	WM	WS	WN	WE	WG

It is Black's turn so the FEN record would be:
`genmsneg/rrr1r1rr/3r4/4Rr2/8/3R4/RRR2RRR/GENMSNEG/B`

Figure 4



	1	2	3	4	5	6	7	8
1	BG	BE	BN	BM	BS	BN	BE	BG
2	BR	BR	BR	BR	BR	BR	BR	BR
3								
4								
5								
6								
7	WR	WR	WR	WR	WR	WR	WR	WR
8	WG	WE	WN	WM	WS	WN	WE	WG

It is White's turn so the FEN record would be:
`genmsneg/rrrrrrrr/8/8/8/8/RRRRRRRR/GENMSNEG/W`

Task 1

Create a new subroutine, `GenerateFEN`, which takes two parameters (the board and whose turn it is) and creates a string containing the FEN record for the current state of a CAPTURE THE SARRUM game. It should return this string to the calling routine. You may choose whether to make the new subroutine a function or a procedure.

You are likely to get some marks for this task, even if your subroutine is only partially working.

It does not matter if your new subroutine will work correctly for positions containing a kashshaptu (from **Question (c)**).

Evidence that you need to provide

(i) Your PROGRAM SOURCE CODE for the new subroutine `GenerateFEN`.

(13)

Task 2
Adapt the MAIN PROGRAM BLOCK so that the FEN record returned by the `GenerateFEN` subroutine is displayed before asking the player to enter their move.

Task 3
Test that the changes you have made work:

- run the **Skeleton Program**
- enter `Y` when asked if you want to play the sample game

Evidence that you need to provide

(ii) Your PROGRAM SOURCE CODE for the amended MAIN PROGRAM BLOCK.

(3)

(iii) SCREEN CAPTURE(S) showing the requested test.

(2)

(Total 40 marks)

Q31.

A computer program is being developed that will simulate the organisation of wagons (trucks) in a railway shunting yard. The simulation will be based on a model developed by the shunting yard manager and a systems analyst.

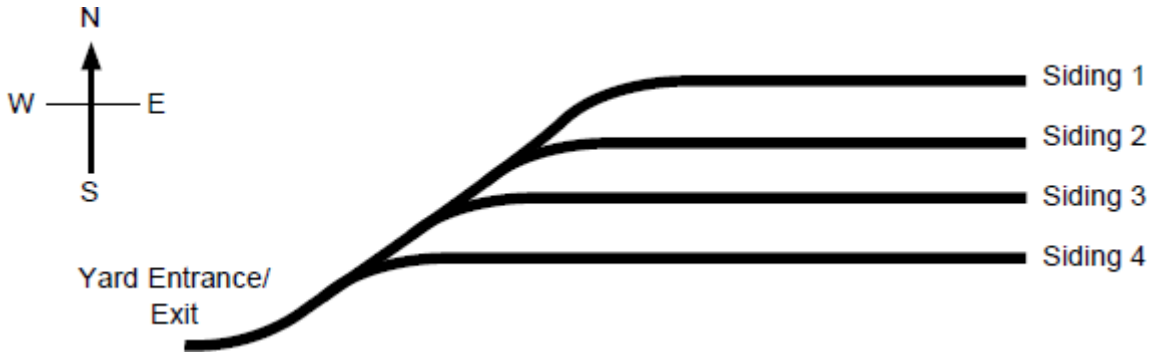
(a) In the context of simulation, explain what a model is.

_____

_____

(1)

The diagram below shows the layout of the railway yard. The wagons enter the yard and are pushed into an appropriate siding, depending upon their final destination. Each siding can hold many wagons. Wagons can **only** enter and leave a siding using the Yard Entrance / Exit at the west.



Wagons will be represented as objects in an object-oriented programming language.

Each of the sidings will be represented as a stack data structure.

(b) Explain why a stack data structure is appropriate for representing a siding.

(2)

(c) The computer program developer intends to implement a stack by using a fixed length array of 30 wagon objects, named `StackArray`, with indices running from `1` to `30`. An integer variable `TopOfStackPointer`, that will store the array index of the item at the top of the stack, will also be used. The first object stored in the array will be stored at index `1`, the second at index `2` and so on. `TopOfStackPointer` will be initialised to `0`.

Write a pseudo-code algorithm for the `Pop` operation to remove a value from the stack and store it in a wagon object variable named `CurrentWagon`.

Your algorithm should cope appropriately with any potential errors that might occur.

(4)

(d) Wagons come in two different categories: open wagons (without a roof) and closed wagons (with a roof). Closed wagons can be either refrigerated or non-refrigerated.

In an object-oriented programming language, five classes are to be created, named **Wagon**, **OpenWagon**, **ClosedWagon**, **RefrigeratedWagon** and **NonRefrigeratedWagon**.

Draw an inheritance diagram for the five classes.

(3)

(e) The **Wagon** class has data fields **OwnerName**, **Weight** and **NumberOfWheels**.

The class definition for **Wagon** is:

```
Wagon = Class
          Public
            Procedure CreateWagon
            Function GetOwnerName
            Function GetWeight
            Function GetNumberOfWheels
          Private
            OwnerName: String
            Weight: Real
            NumberOfWheels: Integer
        End
```


The **ClosedWagon** class has the following additional data fields:

- **Height**: The height of the wagon in metres, which could be a non-integer number
- **NumberOfDoors**: The number of doors that can be used to access the wagon
- **SuitableForFoodstuffs**: A true or false value that indicates if it is safe to carry food in the wagon or not.

Write the class definition for **ClosedWagon**.

You should include the necessary data fields and any additional procedures or functions that the class would require in your definition.

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

(4)

(Total 14 marks)

Q32.

The Backus-Naur Form (BNF) production rules below define the syntax of a number of programming language constructs.

```
<forloop>     ::=  FOR <variable>  = <integer>  TO  <integer>
<variable>    ::=  <letter> | <letter> <string>
<string>      ::=  <character> | <character> <string>
<integer>     ::=  <digit> | <digit> <integer>
<character>   ::=  <digit> | <letter>
<digit>       ::=  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

A `<letter>` is any alphabetic character from `a` to `z` or `A` to `Z`.

(a) The table below contains a list of variable names. Place a tick in each row if the stated variable name is a valid `<variable>` for the production rules above.

You may tick **more than one** box.

<variable>	Valid? (✓ any number of rows)
a	
money_paid	
taxrate2	
2ndPlayerName	

(1)

(b) The production rule for an `<integer>` is recursive.

Explain why recursion has been used in this production rule.

_____

_____

_____

(1)

(c) Here is an example of a valid <forloop> :

```
FOR count = 1 TO 10
```

The BNF production rules above can be used to check whether or not a <forloop> is syntactically correct.

However, it is possible that a programming language statement that is a syntactically correct <forloop> may still produce an error when the program that it is part of is compiled.

Describe **one** example of why a syntactically correct <forloop> may still produce an error when a program is compiled.

_____

_____

_____

_____

(1)

(Total 3 marks)

Q33.

The image below shows an 8-bit bit pattern.

1	0	1	1	0	1	1	0

(a) If the bit pattern above is an **unsigned binary integer**, what is the denary equivalent of this bit pattern?

_____

_____

(1)

(b) If the bit pattern above is a **two's complement binary integer**, what is the denary equivalent of this bit pattern?

_____

_____

_____

_____

(2)

(c) What is the range of **denary** numbers that can be represented using **8-bit two's complement binary integers**?

_____

_____

_____

_____

(2)

(d) If the bit pattern above is an **unsigned binary fixed point** number with 3 bits before and 5 bits after the binary point, what is the denary equivalent of this bit pattern?

_____

_____

_____

_____

(2)

(e) What is the **hexadecimal** equivalent of the bit pattern above?

_____

_____

_____

_____

(2)

(f) Why are bit patterns often displayed using hexadecimal instead of binary?

_____

_____

(1)

(g) Describe a method that can, without the use of binary addition, multiply any **unsigned binary integer** by the binary number 10 (the denary number 2).

_____

_____

_____

_____

(2)

(Total 12 marks)

Q34.

A pseudo-code representation of an algorithm is given below.

```
FOR x ← 0 TO 7 DO
  IF (x MOD 16 >= 4) AND (x MOD 16 <= 11)
    THEN c ← 1
```

```
    ELSE c ← 0
  ENDIF
  IF (x MOD 8 >= 2) AND (x MOD 8 <= 5)
    THEN b ← 1
    ELSE b ← 0
  ENDIF
  IF (x MOD 4 = 0) OR (x MOD 4 = 3)
    THEN a ← 0
    ELSE a ← 1
  ENDIF
  OUTPUT (c, b, a)
ENDFOR
```

The `MOD` operator calculates the remainder resulting from an integer division. For example, `7 MOD 5 = 2, 14 MOD 5 = 4`.

The statement `OUTPUT (c, b, a)` will display the contents of the variable `c`, followed by the contents of the variable `b` and then the contents of the variable `a`.

(a) Explain what is meant by an **algorithm**.

(2)

(b) The decision table shown in **Table 1** represents the logic of the second selection structure in the algorithm above. The decision table is only partially complete; the shaded cells that should show the actions to be taken have not been filled in.

Table 1

Conditions	x MOD 8 >= 2	False	True	True
	x MOD 8 <= 5	True	False	True
Action	b ← 1			
	b ← 0			

Complete **Table 1** so that it shows the actions to be taken when the conditions have particular values: an 'X' symbol should be placed in the relevant shaded cells to indicate the action that will be executed for the given conditions. Some of the shaded cells will need to be left empty.

(2)

(c) Dry run the algorithm above by completing **Table 2**. The first row has been completed for you.

Table 2

x	c	b	a	Printed output
0	0	0	0	000

(5)

(d) Explain, precisely, the purpose of the algorithm above.

_____

_____

(1)

(Total 10 marks)

Q35.

Create a folder / directory for your new program.

The algorithm, represented the using pseudo-code below, and the variable table underneath, describe the process of using a check digit to check if a value entered by the user is a valid 13 digit International Standard Book Number (ISBN).

```
FOR Count ← 1 TO 13 DO
  OUTPUT "Please enter next digit of ISBN: "
  INPUT ISBN[Count]
ENDFOR

CalculatedDigit ← 0

Count ← 1
WHILE Count

  CalculatedDigit ← CalculatedDigit + ISBN[Count]

  Count ← Count + 1

  CalculatedDigit ← CalculatedDigit + ISBN[Count] * 3

  Count ← Count + 1
ENDWHILE
WHILE CalculatedDigit >= 10 DO

  CalculatedDigit ← CalculatedDigit - 10
ENDWHILE
```

```
CalculatedDigit ← 10 - CalculatedDigit
IF CalculatedDigit = 10
  THEN CalculatedDigit ← 0
ENDIF
IF CalculatedDigit = ISBN[13]
   THEN OUTPUT "Valid ISBN"
   ELSE OUTPUT "Invalid ISBN"
ENDIF
```

Identifier	Data Type	Purpose
ISBN	Array[1..13] Of Integer	Stores the 13 digit ISBN entered by the user – one digit is stored in each element of the array.
Count	Integer	Used to select a specific digit in the ISBN.
CalculatedDigit	Integer	Used to store the digit calculated from the first 12 digits of the ISBN. It is also used to store the intermediate results of the calculation.

What you need to do

Write a program for the algorithm above.

Test the program by showing the result of entering the digits `9, 7, 8, 0, 0, 9, 9, 4, 1, 0, 6, 7, 6` (in that order).

Test the program by showing the result of entering the digits `9, 7, 8, 1, 8, 5, 7, 0, 2, 8, 8, 9, 4` (in that order).

Save the program in your new folder / directory.

Evidence that you need to provide

(a) Your PROGRAM SOURCE CODE.

(15)

(b) SCREEN CAPTURE(S) for the test when the digits `9, 7, 8, 0, 0, 9, 9, 4, 1, 0, 6, 7, 6` are entered (in that order).

Your evidence must show the result of the test and, as a minimum, the last three digits entered for the test.

(2)

(c) SCREEN CAPTURE(S) for the test when the digits `9, 7, 8, 1, 8, 5, 7, 0, 2, 8, 8, 9, 4` are entered (in that order).

Your evidence must show the result of the test and, as a minimum, the last three digits entered for the test.

(1)

(Total 18 marks)

Q1.

All marks AO1 (knowledge)

local variables;
return address;
parameters;
register values; **A.** example of register that would be in stack frame

Max 2

[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) **4 marks for AO3 (design) and 8 marks for AO3 (programming)**

Level	Description	Mark Range
4	A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution that meets most of the requirements. All of the appropriate design decisions have been taken. To award 12 marks, all of the requirements must be met.	10-12
3	There is evidence that a line of reasoning has been followed to produce a logically structured program. The program displays relevant prompts, inputs the number value and includes two iterative structures. An attempt has been made to check for factors of the number entered, although this may not work correctly under all circumstances. The solution demonstrates good design work as most of the correct design decisions have been made.	7-9
2	A program has been written and some appropriate, syntactically correct programming language statements have been written. There is evidence that a line of reasoning has been partially followed as although the program may not have the required functionality, it can be seen that the response contains some of the statements that would be needed in a working solution. There is evidence of some appropriate design work as the response recognises at least one appropriate technique that could be used by a working solution, regardless of whether this has been implemented correctly.	4-6
1	A program has been written and a few appropriate programming language statements have been written but there is no evidence that a line of reasoning has been followed to arrive at a working solution. The statements written may or may not be syntactically correct. It is unlikely that any of the key design elements of the task have been recognised.	1-3

Guidance

Evidence of AO3 design – 4 points:

Evidence of design to look for in responses:

1. Identifying that a selection structure is needed to compare user's input with the number 1
2. Identifying that a loop is needed that repeats from 2 to the square root of the number entered **A**. half the value of the number entered **A**. to the number 1 less than the number entered
3. Identifying that use of remainder operator needed **A**. alternative methods to using the remainder operator that calculate if there is a remainder
4. Boolean variable (or equivalent) used to indicate if a number is prime or not

Alternative AO3 design marks:

1. Identifying that a selection structure is needed to compare user's input with the number 1
2. Using nested loops that generate pairs of potential factors
3. Identifying that a test is needed to compare the multiplied factor pairs with the number being checked
4. Boolean (or equivalent) variable used to indicate if a number is prime or not

Note that AO3 (design) points are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works.

Evidence for AO3 programming – 8 points:

Evidence of programming to look for in response:

5. Correct termination condition on iterative structure that repeats until the user does not want to enter another number
6. Suitable prompt, inside iterative structure that asks the user to enter a number and number entered by user is stored in a suitable-named variable
7. Iterative structure that checks for factors has correct syntax and start/end conditions
8. Correct test to see if a potential factor is a factor of the number entered, must be inside the iterative structure for checking factors and the potential factor must change each iteration
9. If an output message saying "Is prime" or "Is not prime" is shown for every integer (greater than 1) **A**. any suitable message
10. Outputs correct message "Is not prime" or "Is prime" under all correct circumstances **A**. any suitable message
11. Outputs message "Not greater than 1" under the correct circumstances **A**. any suitable message
12. In an appropriate location in the code asks the user if they want to enter another number **R**. if message will not be displayed after each time the user has entered a number

Note for examiners: if a candidate produces an unusual answer for this question which seems to work but does not match this mark scheme then this

answer should be referred to team leader for guidance on how it should be marked

12

Python 2

```
import math
again = "y"
while again == "y":
  num = int(raw_input("Enter a number: "))
  if num > 1:
    prime = True
    for count in range(2, int(math.sqrt(num)) + 1):
      if num % count == 0:
        prime = False
    if prime == True:
      print "Is prime"
    else:
      print "Is not prime"
  else:
    print "Not greater than 1"
  again = raw_input("Again (y or n)? ")
```

Python 3

```
import math
again = "y"
while again == "y":
  num = int(input("Enter a number: "))
  if num > 1:
    prime = True
    for count in range(2, int(math.sqrt(num)) + 1):
      if num % count == 0:
        prime = False
    if prime == True:
      print("Is prime")
    else:
      print("Is not prime")
  else:
    print("Not greater than 1")
  again = input("Again (y or n)? ")
```


Visual Basic

```
Sub Main()
  Dim Again As Char = "y"
  Dim Num As Integer
  Dim Prime As Boolean
  While Again = "y"
    Console.Write("Enter a number: ")
    Num = Console.ReadLine()
    If Num > 1 Then
      Prime = True
      For Count = 2 To System.Math.Sqrt(Num)
        If Num Mod Count = 0 Then
          Prime = False
        End If
      Next
      If Prime Then
        Console.WriteLine("Is prime")
      Else
        Console.WriteLine("Is not prime")
      End If
    Else
      Console.WriteLine("Not greater than 1")
    End If
```

```
    Console.Write("Again (y or n)? ")
    Again = Console.ReadLine()
  End While
End Sub
```

C#

```
{
  string Again = "Y";
  int Num = 0;
  bool Prime = true;
  while (Again == "Y")
  {
    Console.Write("Enter a number: ");
    Num = Convert.ToInt32(Console.ReadLine());
    if (Num > 1)
    {
      for (int Count = 2; Count < Convert.ToInt32(Math.Sqrt(Num))
+ 1; Count++)
      {
        if (Num % Count == 0)
        {
          Prime = false;
        }
      }
      if (Prime == true )
      {
        Console.WriteLine("Is prime");
      }
      else
      {
        Console.WriteLine("Is not prime");
      }
    }
    else
    {
      Console.WriteLine("Not greater than 1");
    }
    Console.Write("Again (y or n)? ");
    Again = Console.ReadLine().ToUpper();
  }
}
```


Java

```
public static void main(String[] args)
{
    String again;
    do
        {Console.println("Enter a number:");
        int number = Integer.parseInt(Console.readLine());
        if(number <= 1)
        {
        }
        else
        {
          Console.println("Not greater than 1"); boolean prime =
true;
          int count = number - 1;
          while (prime && count > 1)
          {
            if(number%count == 0)
            {
              prime = false;
            }
```

```
            count--;
          }
          if(prime)
          {
            Console.println("Is prime");
          }
          else
          {
            Console.println("Is not prime");
          }
        }
        Console.println("Would you like to enter another number?
YES/NO");
        again = Console.readLine();
    } while (again.equals("YES"));
}
```

Pascal / Delphi

```
var
  again : string;
  num, count : integer;
  prime : boolean;

begin
  again := 'y';
  while again = 'y' do
    begin
      write('Enter a number: ');
      readln(num);
      if num > 1 then
        begin
          prime := True;
          for count := 2 to round(sqrt(num)) do
            if num mod count = 0 then
              prime := False;
          if prime = true then
            writeln('Is prime')
          else
            writeln('Is not prime');
        end
      else
        writeln('Not greater than 1');
      write('Again (y or n)? ');
      readln(again);
  end;
  readln;
end.
```


(b) **Mark is for AO3 (evaluate)**

****** SCREEN CAPTURE ******
Must match code from part (a), including prompts on screen capture matching those in code.
Code for part (a) must be sensible.

Screen captures showing the number 1 being entered with the message “Not greater than 1” displayed, then the number 5 being entered with the message “Is prime” displayed and then the number 8 being entered with the message “Is not prime” being displayed and program stops after user input stating they do not want to enter another number;

A. alternative messages being displayed if they match code from part (a)

```
Enter a number: 1
Not greater than 1
Again (y or n)? y
Enter a number: 5
Is prime
Again (y or n)? y
Enter a number: 8
Is not prime
Again (y or n)? n
>>>
```

1

[13]

Q5.

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

`Len` (Python/VB only);
`Length` (Pascal/Java only);
`IndexOf` (C#/VB only);

I. case
I. spacing
R. if any additional code

1

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

`Item // RandNo // Count;`

`Rnd;` (Java only)

A. `MaxSize`

I. case
I. spacing
R. if any additional code
R. if spelt incorrectly

1

(c) **All marks AO1 (understanding)**

Mark as follows

- Check for 1st mark point from either solution 1 or solution 2.
- 2nd mark point for Solution 1 only to be awarded if 1st mark point for Solution 1 has been awarded.
- 2nd mark point for Solution 2 only to be awarded if 1st mark point for Solution 2 has been awarded

Solution 1
1st mark:
With a linear queue there could be locations available that are not able to be used **A.** there could be wasted space
(where there is space available in the data structure but it is unusable as it is in front of the data items in the queue);
2nd mark:
(To avoid this issue) items in the queue are all shuffled forward when an item is deleted from (the front of the) queue;
//

Circular lists “wrap round” so (avoid this problem as) the front of the queue does not have to be in the first position in the data structure;

Solution 2
1st mark:
Items in a linear queue are all shuffled forward when an item is deleted from (the front);
//
No need to shuffle items forward after deleting an item in a circular queue;
2nd mark:
this makes (deleting from) (large) linear lists time inefficient;
//
meaning circular queues are more time efficient (when deleting);

2

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

The queue is small in size (so the time inefficiency is not significant);

1

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

Front // pointer to the front of the queue;

1

(f) **All marks for AO2 (analyse)**

Change the `Add` method;

Generate a random number between 1 and 2; **NE**. so there is a 50% chance
Note for examiners: needs to be clear how a 50% chance is created

If it is a 1 then generate a random number from 0, 4, 8, 13, 14, 17, 18, 19 // if it is a 1 then generate a random number from those equivalent to 1-point tiles;

Otherwise generate a random number from the other numbers between 0 and 25 // otherwise generate a random number from those equivalent to non 1-point tiles;

A. equivalent methods to the one described

Note for examiners: refer unusual answers that would work to team leader

4

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

Iterate over the characters in the string;

Get the character code for the current character;

Subtract 32 from the character code // AND the character code with the bit pattern 1011111 / 11011111 // AND the character code with (the decimal value) 95 / 223;
A. Hexadecimal equivalents

Convert that value back into a character and replace the current character with the new character;

A. answers that create a new string instead of replace characters in the

existing string

Alternative answer

Iterate over the characters in the string;

Using a list of the lowercase letters and a list of the uppercase letters;

Find the index of the lowercase letter in the list of lowercase letters;

Get the character in the corresponding position in the uppercase list and replace the current character with the new character;

A. answers that create a new string instead of replace characters in the existing string

4

[14]

Q6.

(a) (i) **Mark is for AO3 (programming)**

Selection structure with correct condition(s) (9, 23) added in suitable place and value of 4 assigned to two tiles in the dictionary;

R. if any other tile values changed

1

Python 2

```
def CreateTileDictionary():
  TileDictionary = dict()
  for Count in range(26):
    if Count in [0, 4, 8, 13, 14, 17, 18, 19]:
      TileDictionary[chr(65 + Count)] = 1
    elif Count in [1, 2, 3, 6, 11, 12, 15, 20]:
      TileDictionary[chr(65 + Count)] = 2
    elif Count in [5, 7, 10, 21, 22, 24]:
      TileDictionary[chr(65 + Count)] = 3
    elif Count in [9, 23]:
      TileDictionary[chr(65 + Count)] = 4
    else:
      TileDictionary[chr(65 + Count)] = 5
  return TileDictionary
```

Python 3

```
def CreateTileDictionary():
  TileDictionary = dict()
  for Count in range(26):
    if Count in [0, 4, 8, 13, 14, 17, 18, 19]:
      TileDictionary[chr(65 + Count)] = 1
    elif Count in [1, 2, 3, 6, 11, 12, 15, 20]:
      TileDictionary[chr(65 + Count)] = 2
    elif Count in [5, 7, 10, 21, 22, 24]:
      TileDictionary[chr(65 + Count)] = 3
    elif Count in [9, 23]:
      TileDictionary[chr(65 + Count)] = 4
    else:
      TileDictionary[chr(65 + Count)] = 5
  return TileDictionary
```

Visual Basic

```
Function CreateTileDictionary() As Dictionary(Of Char,
Integer)
    Dim TileDictionary As New Dictionary(Of Char, Integer)()
    For Count = 0 To 25
      If Array.IndexOf({0, 4, 8, 13, 14, 17, 18, 19}, Count)
> -1 Then
        TileDictionary.Add(Chr(65 + Count), 1)
      ElseIf Array.IndexOf({1, 2, 3, 6, 11, 12, 15, 20}, Count)
> -1 Then
        TileDictionary.Add(Chr(65 + Count), 2)
      ElseIf Array.IndexOf({5, 7, 10, 21, 22, 24},
      Count) > -1 Then
        TileDictionary.Add(Chr(65 + Count), 3)
      ElseIf Array.IndexOf({9, 23}, Count) > -1 Then
        TileDictionary.Add(Chr(65 + Count), 4)
      Else
        TileDictionary.Add(Chr(65 + Count), 5)
      End If
    Next
    Return TileDictionary
End Function
```

C#

```
private static void CreateTileDictionary(ref Dictionary<char,
int> TileDictionary)
{
  int[] Value1 = { 0, 4, 8, 13, 14, 17, 18, 19 };
  int[] Value2 = { 1, 2, 3, 6, 11, 12, 15, 20 };
  int[] Value3 = { 5, 7, 10, 21, 22, 24 };
  int[] Value4 = { 9, 23 };

  for (int Count = 0; Count < 26; Count++)
  {
    if (Value1.Contains(Count))
    {
      TileDictionary.Add((char)(65 + Count), 1);
    }
    else if (Value2.Contains(Count))
    {
      TileDictionary.Add((char)(65 + Count), 2);
    }
    else if (Value3.Contains(Count))
    {
      TileDictionary.Add((char)(65 + Count), 3);
    }
    else if (Value4.Contains(Count))
    {
      TileDictionary.Add((char)(65 + Count), 4);
    }
    else
    {
      TileDictionary.Add((char)(65 + Count), 5);
    }
  }
}
```

Java

```
Map createTileDictionary()
{
  Map<Character,Integer> tileDictionary = new
HashMap<Character,Integer>();
  for (int count = 0; count < 26; count++)
  {
```

```
    switch (count) {
      case 0:
      case 4:
      case 8:
      case 13:
      case 14:
      case 17:
      case 18:
      case 19:
        tileDictionary.put((char)(65 + count), 1);
        break;
      case 1:
      case 2:
      case 3:
      case 6:
      case 11:
      case 12:
      case 15:
      case 20:
        tileDictionary.put((char)(65 + count), 2);
        break;
      case 5:
      case 7:
      case 10:
      case 21:
      case 22:
      case 24:
        tileDictionary.put((char)(65 + count), 3);
        break;
      case 9:
      case 23:
        tileDictionary.put((char)(65 + count), 4);
        break;
      default:
        tileDictionary.put((char)(65 + count), 5);
        break;
    }
  }
  return tileDictionary;
}
```


Pascal / Delphi

```
function CreateTileDictionary() : TTileDictionary;
  var
    TileDictionary : TTileDictionary;
    Count : integer;
  begin
    TileDictionary := TTileDictionary.Create();
    for Count := 0 to 25 do
      begin
        case count of
          0, 4, 8, 13, 14, 17, 18, 19:
TileDictionary.Add(chr(65 + count), 1);
          1, 2, 3, 6, 11, 12, 15, 20: TileDictionary.Add(chr(65
+ count), 2);
          5, 7, 10, 21, 22, 24: TileDictionary.Add(chr(65 +
count), 3);
          9, 23: TileDictionary.Add(chr(65 + count), 4);
          else TileDictionary.Add(chr(65 + count), 5);
        end;
      end;
    CreateTileDictionary := TileDictionary;
  end;
```

(ii) **Mark is for AO3 (evaluate)**

****** SCREEN CAPTURE ******
Must match code from part (a)(i), including prompts on screen capture matching those in code.
Code for part (a)(i) must be sensible.

Screen captures showing the requested test being performed and the correct points values for J, X, Z and Q are shown; **I**. order of letters

```
TILE VALUES

Points for X: 4
Points for R: 1
Points for Q: 5
Points for Z: 5
Points for M: 2
Points for K: 3
Points for A: 1
Points for Y: 3
Points for L: 2
Points for I: 1
Points for F: 3
Points for H: 3
Points for D: 2
Points for U: 2
Points for N: 1
Points for V: 3
Points for T: 1
Points for E: 1
Points for W: 3
Points for C: 2
Points for G: 2
Points for P: 2
Points for J: 4
Points for O: 1
Points for B: 2
Points for S: 1
```



```
Either:
      enter the word you would like to play OR
      press 1 to display the letter values OR
      press 4 to view the tile queue OR
      press 7 to view your tiles again OR
      press 0 to fill hand and stop the game.
```

1

(b) (i) **All marks for AO3 (programming)**

Iterative structure with one correct condition added in suitable place;

Iterative structure with second correct condition and logical connective;

Suitable prompt displayed inside iterative structure or in appropriate place before iterative structure; **A.** any suitable prompt

`StartHandSize` assigned user-entered value inside iterative structure;

Max 3 if code contains errors

4

Python 2

```
…
  StartHandSize = int(raw_input("Enter start hand size: "))
  while StartHandSize < 1 or StartHandSize > 20:
    StartHandSize = int(raw_input("Enter start hand size: "))
…
```

Python 3

```
…
  StartHandSize = int(input("Enter start hand size: "))
  while StartHandSize < 1 or StartHandSize > 20:
    StartHandSize = int(input("Enter start hand size: "))
…
```

Visual Basic

```
…
Do
  Console.Write("Enter start hand size: ")
  StartHandSize = Console.ReadLine()
Loop Until StartHandSize >= 1 And StartHandSize <= 20
…
```

C#

```
…
do
{
  Console.Write("Enter start hand size: ");
  StartHandSize = Convert.ToInt32(Console.ReadLine());
} while (StartHandSize < 1 || StartHandSize > 20);
…
```

Java

```
…
  do {
    Console.println(&"Enter start hand size: &");
    startHandSize = Integer.parseInt(Console.readLine());
  } while (startHandSize < 1 || startHandSize > 20);
…
```


Pascal / Delphi

```
…
StartHandSize := 0;
Choice := '';
while (StartHandSize < 1) or (StartHandSize > 20) do
  begin
    write('Enter start hand size: ');
    readln(StartHandSize);
  end;
…
```

(ii) **Mark is for AO3 (evaluate)**

****** SCREEN CAPTURE ******
Must match code from part (b)(i), including prompts on screen capture matching those in code.
Code for part (b)(i) must be sensible.

Screen capture(s) showing that after the values 0 and 21 are entered the user is asked to enter the start hand size again and then the menu is displayed;

```
++++++++++++++++++++++++++++++++++++
```

```
+ Welcome to the WORDS WITH AQA game +
++++++++++++++++++++++++++++++++++++


Enter start hand size: 0
Enter start hand size: 21
Enter start hand size: 5

=========
MAIN MENU
=========

1. Play game with random start hand
2. Play game with training start hand
9. Quit

Enter your choice: 1

Player One it is your turn.
```

1

(c) (i) **All marks for AO3 (programming)**

1. Create variables to store the current start, mid and end points; **A.** no variable for midpoint if midpoint is calculated each time it is needed in the code
2. Setting correct initial values for start and end variables;
3. Iterative structure with one correct condition (either word is valid or start is greater than end); **R.** if code is a linear search
4. Iterative structure with 2nd correct condition and correct logic;
5. Inside iterative structure, correctly calculate midpoint between start and end;
   A. mid-point being either the position before or the position after the exact middle if calculated midpoint is not a whole number **R.** if midpoint is sometimes the position before and sometimes the position after the exact middle **R.** if not calculated under all circumstances when it should be
6. Inside iterative structure there is a selection structure that compares word at midpoint position in list with word being searched for;
7. Values of start and end changed correctly under correct circumstances;
8. True is returned if match with midpoint word found and True is not returned under any other circumstances;

I. missing statement to display current word

Max 7 if code contains errors

Alternative answer using recursion

1. Create variable to store the current midpoint, start and end points passed as parameters to subroutine; **A.** no variable for midpoint if midpoint is calculated each time it is needed in the code **A.** midpoint as parameter instead of as local variable
2. Initial subroutine call has values of 0 for startpoint parameter and number of words in `AllowedWords` for endpoint parameter;
3. Selection structure which contains recursive call if word being

searched for is after word at midpoint;

4. Selection structure which contains recursive call if word being searched for is before word at midpoint;
5. Correctly calculate midpoint between start and end;
   A. midpoint being either the position before or the position after the exact middle if calculated midpoint is not a whole number **R.** if midpoint is sometimes the position before and sometimes the position after the exact middle **R.** if not calculated under all circumstances when it should be
6. There is a selection structure that compares word at midpoint position in list with word being searched for and there is no recursive call if they are equal with a value of True being returned;
7. In recursive calls the parameters for start and end points have correct values;
8. There is a selection structure that results in no recursive call and False being returned if it is now known that the word being searched for is not in the list;

Note for examiners: mark points 1, 2, 7 could be replaced by recursive calls that appropriately half the number of items in the list of words passed as a parameter – this would mean no need for start and end points. In this case award one mark for each of the two recursive calls if they contain the correctly reduced lists and one mark for the correct use of the length function to find the number of items in the list. These marks should not be awarded if the list is passed by reference resulting in the original list of words being modified.

I. missing statement to display current word

Max 7 if code contains errors

Note for examiners: refer unusual solutions to team leader

8

Python 2

```
def CheckWordIsValid(Word, AllowedWords):
  ValidWord = False
  Start = 0
  End = len(AllowedWords) - 1
  while not ValidWord and Start <= End:
    Mid = (Start + End) // 2
    print AllowedWords[Mid]
    if AllowedWords[Mid] == Word:
      ValidWord = True
    elif Word > AllowedWords[Mid]:
      Start = Mid + 1
    else:
      End = Mid - 1
  return ValidWord
```

Python 3

```
def CheckWordIsValid(Word, AllowedWords):
  ValidWord = False
  Start = 0
  End = len(AllowedWords) - 1
  while not ValidWord and Start <= End:
    Mid = (Start + End) // 2
    print(AllowedWords[Mid])
    if AllowedWords[Mid] == Word:
      ValidWord = True
```

```
    elif Word > AllowedWords[Mid]:
      Start = Mid + 1
    else:
      End = Mid - 1
  return ValidWord
```

Visual Basic

```
Function CheckWordIsValid(ByVal Word As String, ByRef
AllowedWords As List(Of String)) As Boolean
  Dim ValidWord As Boolean = False
  Dim LStart As Integer = 0
  Dim LMid As Integer
  Dim LEnd As Integer = Len(AllowedWords) - 1
  While Not ValidWord And LStart <= LEnd
    LMid = (LStart + LEnd) \ 2
    Console.WriteLine(AllowedWords(LMid))
    If AllowedWords(LMid) = Word Then
      ValidWord = True
    ElseIf Word > AllowedWords(LMid) Then
      LStart = LMid + 1
    Else
      LEnd = LMid - 1
    End If
  End While
  Return ValidWord
End Function
```

C#

```
private static bool CheckWordIsValid(string Word,
List<string> AllowedWords)
{
  bool ValidWord = false;
  int Start = 0;
  int End = AllowedWords.Count - 1;
  int Mid = 0;
  while (!ValidWord && Start <= End)
  {
    Mid = (Start + End) / 2;
    Console.WriteLine(AllowedWords[Mid]);
    if (AllowedWords[Mid] == Word)
    {
      ValidWord = true;
    }
    else if (string.Compare(Word, AllowedWords[Mid]) > 0)
    {
      Start = Mid + 1;
    }
    else
    {
      End = Mid -1;
    }
  }
  return ValidWord;
}
```


Java

```
boolean checkWordIsValid(String word, String[] allowedWords)
{
  boolean validWord = false;
  int start = 0;
  int end = allowedWords.length - 1;
  int mid = 0;
  while (!validWord && start <= end)
```

```
  {
    mid = (start + end) / 2;
    Console.println(allowedWords[mid]);
    if (allowedWords[mid].equals(word))
    {
      validWord = true;
    }
    else if (word.compareTo(allowedWords[mid]) > 0)
    {
      start = mid + 1;
    }
    else
    {
      end = mid -1;
    }
  }
  return validWord;
}
```

Pascal / Delphi

```
function CheckWordIsValid(Word : string; AllowedWords : array
of string) : boolean;
  var
    ValidWord : boolean;
    Start, Mid, EndValue : integer;
  begin
    ValidWord := False;
    Start := 0;
    EndValue := length(AllowedWords) - 1;
    while (not(ValidWord)) and (Start <= EndValue) do
      begin
       Mid := (Start + EndValue) div 2;
       writeln(AllowedWords[Mid]);
       if AllowedWords[Mid] = Word then
         ValidWord := True
       else if Word > AllowedWords[Mid] then
         Start := Mid + 1
       else
         EndValue := Mid - 1;
      end;
    CheckWordIsValid := ValidWord;
  end;
```


(ii) **Mark is for AO3 (evaluate)**

****** SCREEN CAPTURE ******
Must match code from part (c)(i), including prompts on screen capture matching those in code.
Code for part (c)(i) must be sensible.

R. if comparison words not shown in screen capture r

Screen capture(s) showing that the word “jars” was entered and the words “MALEFICIAL”, “DONGLES”, “HAEMAGOGUE”, “INTERMINGLE”, “LAGGER”, “JOULED”, “ISOCLINAL”, “JAUKING”, “JACARANDA”, “JAMBEUX”, “JAPONICA”, “JAROVIZE”, “JASPER”, “JARTA”, “JARRAH”, “JARRINGLY”, “JARS” are displayed in that order;

A. “MALEFICIAL”, “DONGOLA”, “HAEMAGOGUES”, “INTERMINGLED”, “LAGGERS”, “JOULING”, “ISOCLINE”, “JAUNCE”, “JACARE”, “JAMBING”, “JAPPING”, “JAROVIZING”, “JASPERISES”, “JARVEY”, “JARRINGLY”, “JARTA”, “JARS” being displayed if

alternative answer for mark point 5 in part (c)(i) used

ALTERNATIVE ANSWERS (for different versions of text file)

Screen capture(s) showing that the word “jars” was entered and the words “MALEATE”, “DONDER”, “HADST”, “INTERMENDIS”, “LAGAN”, “JOTTERS”, “ISOCHROMATIC”, “JASPERS”, “JABBING”, “JALOUSIE”, “JAPANISES”, “JARGOONS”, “JARRED”, “JASIES”, “JARUL”, “JARS” are displayed in that order;

A. “MALEATE”, “DONDERED”, “HAE”, “INTERMEDIUM”, “LAGANS”, “JOTTING”, “ISOCHROMOSONES”, “JASPERWARES”, “JABBLED”, “JALOUSING”, “JAPANIZED”, “JARINA”, “JARRINGS”, “JASMINES”, “JARVEYS”, “JARTAS”, “JARSFUL”, “JARS” being displayed if alternative answer for mark point 5 in part (c)(i) used

Screen capture(s) showing that the word “jars” was entered and the words “LAMP”, “DESK”, “GAGE”, “IDEAS”, “INVITATION”, “JOURNALS”, “JAMAICA”, “JEWELLERY”, “JEAN”, “JAR”, “JAY”, “JASON”, “JARS” are displayed in that order;

A. “LAMP”, “DESK”, “GAGE”, “IDEAS”, “INVITATIONS”, “JOURNEY”, “JAMIE”, “JEWISH”, “JEEP”, “JAVA”, “JAPAN”, “JARS” being displayed if alternative answer for mark point 5 in part (c)(i) used

```
Either:
  enter the word you would like to play OR
  press 1 to display the letter values OR
  press 4 to view the tile queue OR
  press 7 to view your tiles again OR
  press 0 to fill hand and stop the game.
>jars

MALEFICIAL
DONGLES
HAEMAGOGUE
INTERMINGLE
LAGGER
JOULED
ISOCLINAL
JAUKING
JACARANDA
JAMBEUX
JAPONICA
JAROVIZE
JASPER
JARTA
JARRAH
JARRINGLY
JARS

Valid word

Do you want to:
  replace the tiles you used (1) OR
  get three extra tiles (2) OR
  replace the tiles you used and get three extra tiles (3) OR
  get no new tiles (4)?
>
```

1

(d) (i) **All marks for AO3 (programming)**

1. Creating new subroutine called `CalculateFrequencies` with appropriate interface; **R.** if spelt incorrectly **I.** case
2. Iterative structure that repeats 26 times (once for each letter in the alphabet);
3. Iterative structure that looks at each word in `AllowedWords`;
4. Iterative structure that looks at each letter in a word and suitable nesting for iterative structures;
5. Selection structure, inside iterative structure, that compares two letters;
   A. use of built-in functions that result in same functionality as mark points 4 and 5;;
6. Inside iterative structure increases variable used to count instances of a letter;
7. Displays a numeric count (even if incorrect) and the letter for each letter in the alphabet; **A.** is done in sensible place in `DisplayTileValues`
8. Syntactically correct call to new subroutine from `DisplayTileValues`; **A.** any suitable place for subroutine call

Alternative answer
If answer looks at each letter in `AllowedWords` in turn and maintains a count (eg in array/list) for the number of each letter found then mark points 2 and 5 should be:

2. Creation of suitable data structure to store 26 counts.

5. Appropriate method to select count that corresponds to current letter.

Max 7 if code contains errors

8

Python 2

```
def CalculateFrequencies(AllowedWords):
    print "Letter frequencies in the allowed words are:"
    for Code in range (26):
        LetterCount = 0
        LetterToFind = chr(Code + 65)
        for Word in AllowedWords:
            for Letter in Word:
                if Letter == LetterToFind:
                    b>LetterCount += 1
        sys.stdout.write(LetterToFind + " " + LetterCount)

def DisplayTileValues(TileDictionary, AllowedWords):
    print()
    print("TILE VALUES")
    print()
    for Letter, Points in TileDictionary.items():
        sys.stdout.write("Points for " + Letter + ": " +
str(Points) + "\n")
    print()
    CalculateFrequencies(AllowedWords)
```

Alternative answer

```
def CalculateFrequencies(AllowedWords):
    for Letter in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
        Count=0
        for Word in AllowedWords:
```

```
      NumberOfTimes = Word.count(Letter)
      Count = Count + NumberOfTimes
    sys.stdout.write(Letter + " " + str(Count))
```

Alternative answer

```
def CalculateFrequencies(AllowedWords):
  Counts = []
  for a in range(26):
    Counts.append(0)
  for Word in AllowedWords:
    for Letter in Word:
      Counts[ord(Letter) - 65] += 1
  for a in range(26):
    sys.stdout.write(chr(a + 65) + " " + str(Counts[a]))
```

Python 3

```
def CalculateFrequencies(AllowedWords):
  print("Letter frequencies in the allowed words are:")
  for Code in range (26):
    LetterCount = 0
    LetterToFind = chr(Code + 65)
    for Word in AllowedWords:
      for Letter in Word:
        if Letter == LetterToFind:
          LetterCount += 1
    print(LetterToFind, " ", LetterCount)

def DisplayTileValues(TileDictionary, AllowedWords):
  print()
  print("TILE VALUES")
  print()
  for Letter, Points in TileDictionary.items():
    print("Points for " + Letter + ": " + str(Points))
  print()
  CalculateFrequencies(AllowedWords)
```

Alternative answer

```
def CalculateFrequencies(AllowedWords):
  for Letter in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
    Count=0
    for Word in AllowedWords:
      NumberOfTimes = Word.count(Letter)
      Count = Count + NumberOfTimes
    print(Letter,Count)
```

Alternative answer

```
def CalculateFrequencies(AllowedWords):
  Counts = []
  for a in range(26):
    Counts.append(0)
  for Word in AllowedWords:
    for Letter in Word:
      Counts[ord(Letter) - 65] += 1
  for a in range(26):
    print(chr(a + 65), Counts[a])
```

Visual Basic

```
Sub CalculateFrequencies(ByRef AllowedWords As List(Of
String))
  Dim LetterCount As Integer
  Dim LetterToFind As Char
  Console.WriteLine("Letter frequencies in the allowed words
are:")
```

```
  For Code = 0 To 25
    LetterCount = 0
    LetterToFind = Chr(Code + 65)
    For Each Word In AllowedWords
      For Each Letter In Word
        If Letter = LetterToFind Then
          LetterCount += 1
        End If
      Next
    Next
    Console.WriteLine(LetterToFind & " " & LetterCount)
  Next
End Sub

Sub DisplayTileValues(ByVal TileDictionary As Dictionary(Of
Char, Integer), ByRef AllowedWords As List(Of String))
  Console.WriteLine()
  Console.WriteLine("TILE VALUES")
  Console.WriteLine()
  For Each Tile As KeyValuePair(Of Char, Integer) In
  TileDictionary
    Console.WriteLine("Points for " & Tile.Key & ": " &
Tile.Value)
  Next
  Console.WriteLine()
  CalculateFrequencies(AllowedWords)
End Sub
```

Alternative answer

```
Sub CalculateFrequencies(ByRef AllowedWords As List(Of
String))
  Dim NumberOfTimes, Count As Integer
  Console.WriteLine("Letter frequencies in the allowed words
are:")
  For Each Letter In "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    Count = 0
    For Each Word In AllowedWords
      NumberOfTimes = Word.Split(Letter).Length - 1
      Count += NumberOfTimes
    Next
    Console.WriteLine(Letter & " " & Count)
  Next
End Sub
```


Alternative answer

```
Sub CalculateFrequencies(ByRef AllowedWords As List(Of
String))
  Dim Counts(25) As Integer
  For Count = 0 To 25
    Counts(Count) = 0
  Next
  Console.WriteLine("Letter frequencies in the allowed words
are:")
  For Each Word In AllowedWords
    For Each Letter In Word
      Counts(Asc(Letter) - 65) += 1
    Next
  Next
  For count = 0 To 25
    Console.WriteLine(Chr(count + 65) & " " & Counts(count))
  Next
End Sub
```

C#

```
private static void CalculateFrequencies(List<string>
AllowedWords)
{
  Console.WriteLine("Letter frequencies in the allowed words
are:");
  int LetterCount = 0;
  char LetterToFind;
  for (int Code = 0; Code < 26; Code++)
  {
    LetterCount = 0;
    LetterToFind = (char)(Code + 65);
    foreach (var Word in AllowedWords)
    {
      foreach (var Letter in Word)
      {
        if (Letter == LetterToFind)
        {
          LetterCount++;
        }
      }
    }
    Console.WriteLine(LetterToFind + " " + LetterCount);
  }
}

private static void DisplayTileValues(Dictionary<char, int>
TileDictionary, List<string> AllowedWords)
{
  Console.WriteLine();
  Console.WriteLine("TILE VALUES");
  Console.WriteLine();
  char Letter;
  int Points;
  foreach (var Pair in TileDictionary)
  {
    Letter = Pair.Key;
    Points = Pair.Value;
    Console.WriteLine("Points for " + Letter + ": " + Points);
  }
  CalculateFrequencies(AllowedWords);
  Console.WriteLine();
}
```


Alternative answer

```
private static void CalculateFrequencies(List<string>
AllowedWords)
{
  Console.WriteLine("Letter frequencies in the allowed words
are:");
  int LetterCount = 0;
  string Alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  foreach (var Letter in Alphabet)
  {
    LetterCount = 0;
    foreach (var Words in AllowedWords)
    {
      LetterCount = LetterCount + (Words.Split(Letter).Length
- 1);
    }
    Console.WriteLine(Letter + " " + LetterCount);
  }
}
```

Alternative answer

```
private static void CalculateFrequencies(List<string>
AllowedWords)
{
  List<int> Counts = new List<int>() ;
  for (int i = 0; i < 26; i++)
  {
    Counts.Add(0);
  }
  foreach (var Words in AllowedWords)
  {
    foreach (var Letter in Words)
    {
      Counts[(int)Letter - 65]++;
    }
  }
  for (int a = 0; a < 26; a++)
  {
    char Alpha =Convert.ToChar( a + 65);
    Console.WriteLine(Alpha + " " + Counts[a] );
  }
}
```

Java

```
void calculateFrequencies(String[] allowedWords)
{
  int letterCount;
  char letterToFind;
  for (int count = 0; count < 26; count++)
  {
    letterCount = 0;
    letterToFind = (char)(65 + count);
    for(String word:allowedWords)
    {
      for(char letter : word.toCharArray())
      {
        if(letterToFind == letter)
        {
          letterCount++;
        }
      }
    }
    Console.println(letterToFind + ", Frequency: " +
letterCount);
  }
}

void displayTileValues(Map tileDictionary, String[]
allowedWords)
{
  Console.println();
  Console.println("TILE VALUES");
  Console.println();
  for (Object letter : tileDictionary.keySet())
  {
    int points = (int)tileDictionary.get(letter);
    Console.println("Points for " + letter + ": " + points);
  }
  calculateFrequencies(allowedWords);
  Console.println();
}
```

Alternative answer

```
void calculateFrequencies(String[] allowedWords)
{
  int letterCount;
  String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  for(char letter: alphabet.toCharArray())
  {
    letterCount = 0;
    for(String word: allowedWords)
    {
      letterCount += word.split(letter + "").length - 1;
    }
    Console.println(letter + ", Frequency: " + letterCount);
  }
}
```

Alternative answer

```
void calculateFrequencies(String[] allowedWords)
{
  int[] counts = new int[26];
  for(String word: allowedWords)
  {
    for(char letter: word.toCharArray())
    {
      int letterPostion = (int)letter - 65;
      counts[letterPostion]++;
    }
  }
  for (int count = 0; count < 26; count++)
  {
    char letter = (char)(65 + count);
    Console.println(letter + ", Frequency: " + counts[count]);
  }
}
```

Pascal / Delphi

```
procedure CalculateFrequencies(AllowedWords : array of
string);
  var
    Code, LetterCount : integer;
    LetterToFind, Letter : char;
    Word : string;
  begin
    writeln('Letter frequencies in the allowed words are:');
    for Code := 0 to 25 do
      begin
        LetterCount := 0;
        LetterToFind := chr(65 + Code);
        for Word in AllowedWords do
          begin
            for Letter in Word do
              begin
                if Letter = LetterToFind then
                  LetterCount := LetterCount + 1;
              end;
          end;
        writeln(LetterToFind, ' ', LetterCount);
      end;
  end;
```

(ii) **Mark is for AO3 (evaluate)**

****** SCREEN CAPTURE ******

Must match code from part (d)(i), including prompts on screen capture

matching those in code.
Code for part (d)(i) must be sensible.

Screen capture(s) showing correct list of letter frequencies are displayed;

I. Ignore order of letter frequency pairs
I. any additional output eg headings like “Letter” and “Count”

```
Letter frequencies in the allowed words are:
A  188704
B  44953
C  98231
D  81731
E  275582
F  28931
G  67910
H  60702
I  220483
J  4010
K  22076
L  127865
M  70700
N  163637
O  161752
P  73286
Q  4104
R  170522
S  234673
T  159471
U  80636
V  22521
W  18393
X  6852
Y  39772
Z  11772
Either:
      enter the word you would like to play OR
      press 1 to display the letter values OR
      press 4 to view the tile queue OR
      press 7 to view your tiles again OR
      press 0 to fill hand and stop the game.
>
```


ALTERNATIVE ANSWERS (for different versions of text file)

```
Letter frequencies in the allowed words are:
A  188627
B  44923
C  98187
D  81686
E  275478
F  28899
G  67795
H  60627
I  220331
J  4007
K  22028
L  127814
M  70679
N  163547
O  161720
P  73267
Q  4104
R  170461
S  234473
```

```
T  159351
U  80579
V  22509
W  18377
X  6852
Y  39760
Z  11765

Either:
  enter the word you would like to play OR
  press 1 to display the letter values OR
  press 4 to view the tile queue OR
  press 7 to view your tiles again OR
  press 0 to fill hand and stop the game.
>
Letter frequencies in the allowed words are:
A  5299
B  1105
C  2980
D  2482
E  7523
F  909
G  1692
H  1399
I  5391
J  178
K  569
L  3180
M  1871
N  4762
O  4177
P  1992
Q  122
R  4812
S  4999
T  4695
U  1898
V  835
W  607
X  246
Y  999
Z  128

Either:
      enter the word you would like to play OR
      press 1 to display the letter values OR
      press 4 to view the tile queue OR
      press 7 to view your tiles again OR
      press 0 to fill hand and stop the game.
>
```


1

(e) (i) **All marks for AO3 (programming)**

Modifying subroutine `UpdateAfterAllowedWord`:

1. Correct subroutine call to `GetScoreForWordAndPrefix` added in `UpdateAfterAllowedWord`;
2. Result returned by `GetScoreForWordAndPrefix` added to `PlayerScore;`

A. alternative names for subroutine `GetScoreForWordAndPrefix` if

match name of subroutine created

Creating new subroutine:

3. Subroutine `GetScoreForWordAndPrefix` created; **R.** if spelt incorrectly **I.** case
4. All data needed (`Word`, `TileDictionary`, `AllowedWords`) is passed into subroutine via interface;
5. Integer value always returned by subroutine;

Base case in subroutine:

6. Selection structure for differentiating base case and recursive case with suitable condition (word length of 0 // 1 // 2); **R.** if base case will result in recursion
7. If base case word length is 0 then value of 0 is returned by subroutine and there is no recursive call // if base case word length is 1 then value of 0 is returned by subroutine and there is no recursive call // if base case word length is 2 the subroutine returns 0 if the two-letter word is not a valid word and returns the score for the two-letter word if it is a valid word;

Recursive case in subroutine:

8. Selection structure that contains code that adds value returned by call to `GetScoreForWord` to score if word is valid; **A.** no call to subroutine `GetScoreForWord` if correct code to calculate score included in sensible place in `GetScoreForWordAndPrefix` subroutine R. if no check for word being valid
9. Call to `GetScoreForWordAndPrefix`;
10. Result from recursive call added to score;
11. Recursion will eventually reach base case as recursive call has a parameter that is word with last letter removed;

How to mark question if no attempt to use recursion:

Mark points 1-5 same as for recursive attempt. No marks awarded for mark points 6-11, instead award marks as appropriate for mark points 12-14.

12. Adds the score for the original word to the score once // sets the initial score to be the score for the original word; **A.** no call to subroutine `GetScoreForWord` if correct code to calculate score included in sensible place in `GetScoreForWordAndPrefix` subroutine. **Note for examiners:** there is no need for the answer to check if the original word is valid
13. Iterative structure that will repeat $n - 1$ times where n is the length of the word; **A.** $n - 2$ **A.** n
14. Inside iterative structure adds score for current prefix word, if it is a valid word, to score once; **A.** no call to `GetScoreForWord` if own code to calculate score is correct

Max 10 if code contains errors

Max 8 if recursion not used in an appropriate way

11

Python 2

```
def UpdateAfterAllowedWord(Word, PlayerTiles, PlayerScore,
```

```
PlayerTilesPlayed, TileDictionary, AllowedWords):
  PlayerTilesPlayed += len(Word)
  for Letter in Word:
    PlayerTiles = PlayerTiles.replace(Letter, "", 1)
  PlayerScore += GetScoreForWordAndPrefix(Word,
TileDictionary, AllowedWords)
  return PlayerTiles, PlayerScore, PlayerTilesPlayed

def GetScoreForWordAndPrefix(Word, TileDictionary,
AllowedWords):
  if len(Word) <= 1:
    return 0
  else:
    Score = 0
    if CheckWordIsValid(Word, AllowedWords):
      Score += GetScoreForWord(Word, TileDictionary)
    Score += GetScoreForWordAndPrefix(Word[0:len(Word) - 1],
TileDictionary, AllowedWords)
    return Score
```

Alternative answer

```
def GetScoreForWordAndPrefix(Word,TileDictionary,
AllowedWords):
  Score = 0
  if CheckWordIsValid(Word,AllowedWords):
    Score += GetScoreForWord(Word, TileDictionary)
  if len(Word[:-1]) > 0:
    Score +=GetScoreForWordAndPrefix(Word[:-1],
TileDictionary,AllowedWords)
  return Score
```

Python 3

```
def UpdateAfterAllowedWord(Word, PlayerTiles, PlayerScore,
PlayerTilesPlayed, TileDictionary, AllowedWords):
  PlayerTilesPlayed += len(Word)
  for Letter in Word:
    PlayerTiles = PlayerTiles.replace(Letter, "", 1)
  PlayerScore += GetScoreForWordAndPrefix(Word,
TileDictionary, AllowedWords)
  return PlayerTiles, PlayerScore, PlayerTilesPlayed

def GetScoreForWordAndPrefix(Word, TileDictionary,
AllowedWords):
  if len(Word) <= 1:
    return 0
  else:
    Score = 0
    if CheckWordIsValid(Word, AllowedWords):
      Score += GetScoreForWord(Word, TileDictionary)
    Score += GetScoreForWordAndPrefix(Word[0:len(Word) - 1],
TileDictionary, AllowedWords)
    return Score
```

Alternative answer

```
def GetScoreForWordAndPrefix(Word,TileDictionary,
AllowedWords):
    Score = 0
    if CheckWordIsValid(Word,AllowedWords):
    Score += GetScoreForWord(Word, TileDictionary)
  if len(Word[:-1]) > 0:
    Score +=GetScoreForWordAndPrefix(Word[:-1],
TileDictionary,AllowedWords)
```

```
  return Score
```

Visual Basic

```
Sub UpdateAfterAllowedWord(ByVal Word As String, ByRef
PlayerTiles As String, ByRef PlayerScore As Integer, ByRef
PlayerTilesPlayed As Integer, ByVal TileDictionary As
Dictionary(Of Char, Integer), ByRef AllowedWords As List(Of
String))
  PlayerTilesPlayed += Len(Word)
  For Each Letter In Word
    PlayerTiles = Replace(PlayerTiles, Letter, "", , 1)
  Next
  PlayerScore += GetScoreForWordAndPrefix(Word,
TileDictionary, AllowedWords)
End Sub

Function GetScoreForWordAndPrefix(ByVal Word As String, ByVal
TileDictionary As Dictionary(Of Char, Integer), ByRef
AllowedWords As List(Of String)) As Integer
  Dim Score As Integer
  If Len(Word) <= 1 Then
    Return 0
  Else
    Score = 0
    If CheckWordIsValid(Word, AllowedWords) Then
      Score += GetScoreForWord(Word, TileDictionary)
    End If
    Score += GetScoreForWordAndPrefix(Mid(Word, 1, Len(Word)
- 1), TileDictionary, AllowedWords)
  End If
  Return Score
End Function
```

Alternative answer

```
Function GetScoreForWordAndPrefix(ByVal Word As String, ByVal
TileDictionary As Dictionary(Of Char, Integer), ByRef
AllowedWords As List(Of String)) As Integer
  Dim Score As Integer = 0
  If CheckWordIsValid(Word, AllowedWords) Then
    Score += GetScoreForWord(Word, TileDictionary)
  End If
  If Len(Word) - 1 > 0 Then
    Score += GetScoreForWordAndPrefix(Mid(Word, 1, Len(Word)
- 1), TileDictionary, AllowedWords)
  End If
  Return Score
End Function
```

C#

```
private static void UpdateAfterAllowedWord(string Word, ref
string PlayerTiles, ref int PlayerScore, ref int
PlayerTilesPlayed, Dictionary<char, int> TileDictionary,
List<string> AllowedWords)
{
  PlayerTilesPlayed = PlayerTilesPlayed + Word.Length;
  foreach (var Letter in Word)
  {
    PlayerTiles =
PlayerTiles.Remove(PlayerTiles.IndexOf(Letter), 1);
  }
  PlayerScore = PlayerScore + GetScoreForWordAndPrefix(Word,
TileDictionary, AllowedWords);
}
```

```
private static int GetScoreForWordAndPrefix(string Word,
Dictionary<char, int> TileDictionary, List<string>
AllowedWords)
{
  int Score = 0;
  if (Word.Length <= 1)
  {
    return 0;
  }
  else
  {
    Score = 0;
    if (CheckWordIsValid(Word, AllowedWords))
    {
      Score = Score + GetScoreForWord(Word, TileDictionary);
    }
    Score = Score +
GetScoreForWordAndPrefix(Word.Remove(Word.Length - 1),
TileDictionary, AllowedWords);
    return Score;
  }
}
```

Alternative answer

```
private static int GetScoreForWordAndPrefix(string Word,
Dictionary<char, int> TileDictionary, List<string>
AllowedWords)
{
  int Score = 0;
  if (CheckWordIsValid(Word, AllowedWords))
  {
    Score = Score + GetScoreForWord(Word, TileDictionary);
  }
  if (Word.Remove(Word.Length - 1).Length > 0)
  {
    Score = Score +
GetScoreForWordAndPrefix(Word.Remove(Word.Length - 1),
TileDictionary, AllowedWords);
  }
  return Score;
}
```


Java

```
int getScoreForWordAndPrefix(String word, Map tileDictionary,
String[] allowedWords)
{
  int score = 0;
  if(word.length() < 2)
  {
    return 0;
  }
  else
  {
    if(checkWordIsValid(word, allowedWords))
    {
      score = getScoreForWord(word, tileDictionary);
    }
  word = word.substring(0, word.length()-1);
    return score + getScoreForWordAndPrefix(word,
tileDictionary, allowedWords);
  }
}
```

```
void updateAfterAllowedWord(String word, Tiles
playerTiles,
    Score playerScore, TileCount playerTilesPlayed, Map
tileDictionary,
    String[] allowedWords)
{
  playerTilesPlayed.numberOfTiles += word.length();
  for(char letter : word.toCharArray())
  {
    playerTiles.playerTiles =
playerTiles.playerTiles.replaceFirst(letter+"", "");
  }
  playerScore.score += getScoreForWordAndPrefix(word,
tileDictionary, allowedWords);
}
```

Alternative answer

```
int getScoreForWordAndPrefix(String word, Map tileDictionary,
String[] allowedWords)
{
  int score = 0;
  if(checkWordIsValid(word, allowedWords))
  {
    score += getScoreForWord(word, tileDictionary);
  }
  word = word.substring(0, word.length()-1);
  if(word.length()>1)
  {
    score += getScoreForWordAndPrefix(word, tileDictionary,
allowedWords);
  }
  return score;
}
```

Pascal / Delphi

```
function GetScoreForWordAndPrefix(Word : string;
TileDictionary : TileDictionary; AllowedWords : array of
string) : integer;
  var
    Score : integer;
  begin
    if length(word) <= 1 then
      Score := 0
    else
      begin
        Score := 0;
        if CheckWordIsValid(Word, AllowedWords) then
          Score := Score + GetScoreForWord(Word,
TileDictionary);
        Delete(Word,length(Word),1);
        Score := Score + GetScoreForWordAndPrefix(Word,
TileDictionary, AllowedWords);
      end;
    GetScoreForWordAndPrefix := Score;
  end;

procedure UpdateAfterAllowedWord(Word : string; var
PlayerTiles : string; var PlayerScore : integer; var
PlayerTilesPlayed : integer; TileDictionary : TileDictionary;
var AllowedWords : array of string);
  var
    Letter : Char;
  begin
```

```
      PlayerTilesPlayed := PlayerTilesPlayed + length(Word);
      for Letter in Word do
        Delete(PlayerTiles,pos(letter, PlayerTiles),1);
      PlayerScore := PlayerScore +
GetScoreForWordAndPrefix(Word, TileDictionary,
AllowedWords);
    end;
```

(ii) **Mark is for AO3 (evaluate)**

****** SCREEN CAPTURE ******
Must match code from part (e)(i), including prompts on screen capture matching those in code.
Code for part (e)(i) must be sensible.

Screen capture(s) showing that the word abandon was entered and the new score of 78 is displayed;

```
Do you want to:
      replace the tiles you used (1) OR
      get three extra tiles (2) OR replace the tiles you used
and get three extra tiles (3) OR
      get no new tiles (4)?
>4

Your word was: ABANDON
Your new score is: 78
You have played 7 tiles so far in this game.

Press Enter to continue
```

1

[37]

Q7.

(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]

Q8.

(a) (i) **Marks are for AO3 (programming)**

1 mark: 1. tests for lower bound and displays error message if below
1 mark: 2. tests for upper bound and displays error message if above
1 mark: 3. Upper bound test uses `LandscapeSize` instead of data value of 14/15 **A**. in use of incorrect condition
1 mark: 4. 1-3 happen repeatedly until valid input (for the upper and lower bounds used in the code provided) and forces re-entry of data each time

A. use of pre or post-conditioned loop

MAX 3 if error message is not `Coordinate is outside of landscape, please try again` **A**. minor typos in error message **I**. case **I**. spacing **I**. minor punctuation differences

MAX 2 if new code has been added to `Simulation` constructor instead of `InputCoordinate` method

4

VB.NET

```
Do
  Console.Write(" Input " & CoordinateName & " coordinate: ")
  Coordinate = CInt(Console.ReadLine())
  If Coordinate < 0 Or Coordinate >= LandscapeSize Then
    Console.WriteLine("Coordinate is outside of landscape,
please try again.")
  End If
Loop While Coordinate < 0 Or Coordinate >= LandscapeSize
```

Alternative answer

```
Do
  Console.Write(" Input " & CoordinateName & " coordinate: ")
  Coordinate = CInt(Console.ReadLine())
  If Coordinate < 0 Or Coordinate >= LandscapeSize Then
    Console.WriteLine("Coordinate is outside of landscape,
please try again.")
  End If
Loop Until Coordinate >= 0 And Coordinate < LandscapeSize
```

PYTHON 2

```
def __InputCoordinate(self, CoordinateName):
  Coordinate = int(raw_input(" Input " + CoordinateName + "
coordinate:"))
  while Coordinate < 0 or Coordinate >= self.__LandscapeSize:
    Coordinate = int(raw_input("Coordinate is outside of
landscape, please try again."))
  return Coordinate
```

PYTHON 3

```
def __InputCoordinate(self, CoordinateName):
  Coordinate = int(input(" Input " + CoordinateName + "
coordinate:"))
  while Coordinate < 0 or Coordinate >= self.__LandscapeSize:
    Coordinate = int(input("Coordinate is outside of
```

```
landscape, please try again."))
  return Coordinate
```

C#

```
do
{
  Console.Write(" Input " + Coordinatename + " coordinate: ");
  Coordinate = Convert.ToInt32(Console.ReadLine());
  if ((Coordinate < 0) || (Coordinate >= LandscapeSize))
  {
    Console.WriteLine("Coordinate is outside of landscape,
please try again.");
  }
} while ((Coordinate < 0) || (Coordinate >= LandscapeSize));
```

PASCAL

```
repeat
  write(' Input ' , CoordinateName, ' coordinate: ');
  readln(Coordinate);
  if (Coordinate < 0) or (Coordinate >= LandscapeSize) then
    writeln('Coordinate is outside of landscape, please try
again.');
until (Coordinate >= 0) and (Coordinate < LandscapeSize);
```

JAVA

```
private int InputCoordinate(char CoordinateName)
{
  int Coordinate;
  do
  {
    Coordinate = Console.readInteger(" Input " +
CoordinateName + " coordinate: ");
    if (Coordinate >= LandscapeSize || Coordinate < 0)
    {
      Console.println("Coordinate is outside of landscape,
please try again.");
    }
  }while (Coordinate >= LandscapeSize || Coordinate < 0);
  return Coordinate;
}
```


(ii) **Mark is for AO3 (evaluate)**

****SCREEN CAPTURE(S)****
Must match code from part (a)(i), including error message. Code for part (a)(i) must be sensible.

1 mark: Screen capture(s) showing the required sequence of inputs `(-1, 15, 0)`, the correct error message being displayed for `-1` and `15`, and that `0` has been accepted as the program has displayed the prompt for the y coordinate to be input.

```
Select option: 3
  Input x coordinate: -1
Coordinate is outside of landscape, please try again.
  Input x coordinate: 15
Coordinate is outside of landscape, please try again.
  Input x coordinate: 0
  Input y coordinate:
```

A. alternative error messages if match code for part (a)(i)

1

(b) (i) **Marks are for AO3 (programming)**

1 mark: New subroutine created, with correct name, that overrides the subroutine in the `Animal` class
I. private, protected, public modifiers

1 mark: 2. `CalculateNewAge` subroutine in `Animal` class is always called
1 mark: 3. Check made on gender of rabbit, and calculations done differently for each gender
I. incorrect calculations

1 mark: 4. Probability of death by other causes calculated correctly for male rabbits
1 mark: 5. Probability of death by other causes calculated correctly for female rabbits

5

VB.NET

```
Public Overrides Sub CalculateNewAge()
  MyBase.CalculateNewAge()
  If Gender = Genders.Male Then
    ProbabilityOfDeathOtherCauses =
ProbabilityOfDeathOtherCauses * 1.5
  Else
    If Age >= 2 Then
      ProbabilityOfDeathOtherCauses =
ProbabilityOfDeathOtherCauses + 0.05
    End If
  End If
End Sub
```

A. `If Age > 1 Then` **instead of** `If Age >= 2 Then`

PYTHON 2

```
def CalculateNewAge(self):
  super(Rabbit, self).CalculateNewAge()
  if self.__Gender == Genders.Male:
    self._ProbabilityOfDeathOtherCauses =
self._ProbabilityOfDeathOtherCauses * 1.5
  else:
    if self._Age >= 2:
      self._ProbabilityOfDeathOtherCauses =
self._ProbabilityOfDeathOtherCauses + 0.05
```

PYTHON 3

```
def CalculateNewAge(self):
  super(Rabbit, self).CalculateNewAge()
  if self.__Gender == Genders.Male:
    self._ProbabilityOfDeathOtherCauses =
self._ProbabilityOfDeathOtherCauses * 1.5
  else:
    if self._Age >= 2:
      self._ProbabilityOfDeathOtherCauses =
self._ProbabilityOfDeathOtherCauses + 0.05
```

C#

```
public override void CalculateNewAge()
{
  base.CalculateNewAge();
  if (Gender == Genders.Male)
```

```
  {
    ProbabilityOfDeathOtherCauses =
ProbabilityOfDeathOtherCauses * 1.5;
  }
  else
  {
    if (Age >= 2)
    {
      ProbabilityOfDeathOtherCauses =
ProbabilityOfDeathOtherCauses + 0.5;
    }
  }
}
```

PASCAL

```
Procedure Rabbit.CalculateNewAge();
  begin
    inherited;
    if Gender = Male then
      ProbabilityOfDeathOtherCauses :=
ProbabilityOfDeathOtherCauses * 1.5
    else
      if Age >= 2 then
        ProbabilityOfDeathOtherCauses :=
ProbabilityOfDeathOtherCauses + 0.05;
  end;
```

JAVA

```
@Override
public void CalculateNewAge()
{
  super.CalculateNewAge();
  if (Gender == Genders.Male)
  {
    ProbabilityOfDeathOtherCauses *= 1.5;
  }
  else if(Age >= 2)
  {
    ProbabilityOfDeathOtherCauses += 0.05;
  }
}
```


(ii) **Mark is for AO3 (evaluate)**

****SCREEN CAPTURE(S)****
Must match code from part (b)(i). Code for part (b)(i) must be sensible.

1 mark: Any screen capture(s) showing the correct probability of death by other causes for a male rabbit (0.11 to 2dp) and a female rabbit (0.1);

Example:

```
ID 3 Age 2 LS 4 Pr dth 0.1 Rep rate 1.2 Gender Female
ID 4 Age 2 LS 4 Pr dth 0.11 Rep rate 1.2 Gender Male
```

1

(c) (i) **Marks are for AO3 (programming)**

1 mark: Structure set-up to store the representation of terrain for a location
1 mark: Type of terrain is passed to constructor as parameter
1 mark: Type of terrain stored into attribute by constructor **A**. default

value, that makes type of terrain for location clear, instead of value from a parameter

3

VB.NET
```
Class Location
  Public Fox As Fox
  Public Warren As Warren
  Public Terrain As Char

  Public Sub New(ByVal TerrainType As Char)
    Fox = Nothing
    Warren = Nothing
    Terrain = TerrainType

  End Sub
End Class
```

PYTHON 2
```
class Location:
  def __init__(self, TerrainType):
  self.Fox = None
  self.Warren = None
  self.Terrain = TerrainType
```

PYTHON 3
```
class Location:
  def __init__(self, TerrainType):
  self.Fox = None
  self.Warren = None
  self.Terrain = TerrainType
```

C#
```
class Location
{
  public Fox Fox;
  public Warren Warren;
  public char Terrain;

  public Location(char Terraintype)
  {
    Fox = null;
    Warren = null;
    Terrain = Terraintype;
  }
}
```

PASCAL
```
type
  Location = class
    Fox : Fox;
    Warren : Warren;
    Terrain : char;
    constructor New(TerrainType : char);
  end;

constructor Location.New(TerrainType : char);
  begin
    Fox := nil;
    Warren := nil;
    Terrain := TerrainType;
  end;
```

JAVA

```
class Location
{
  public Fox Fox;
  public Warren Warren;
  public char Terrain;

  public Location(char Terrain)
  {
    Fox = null;
    Warren = null;
    this.Terrain = Terrain;
  }
}
```

(ii) **Marks are for AO3 (programming)**

1 mark: 1. An indicator for type of terrain will be stored for every location
I. wrong type of terrain in a location
R. if indicators other than R or L used
I. case of indicators

1 mark: 2. Vertical river created in column 5
1 mark: 3. Horizontal river created in row 2
MAX 1 FOR 2 & 3 if only creates a river when foxes & warrens are in default locations
MAX 2 if creates any rivers in incorrect locations

3

VB.NET

```
For x = 0 To LandscapeSize - 1
  For y = 0 To LandscapeSize - 1
    If x = 5 Or y = 2 Then
      Landscape(x, y) = New Location("R")
    Else
      Landscape(x, y) = New Location("L")
    End If
  Next
Next
```


PYTHON 2

```
def __CreateLandscapeAndAnimals(self, InitialWarrenCount,
InitialFoxCount, FixedInitialLocations):
  for x in range (0, self.__LandscapeSize):
    for y in range (0, self.__LandscapeSize):
      if x == 5 or y == 2:
        self.__Landscape[x][y] = Location("R")
      else:
        self.__Landscape[x][y] = Location("L")
  if FixedInitialLocations:
...
```

PYTHON 3

```
def __CreateLandscapeAndAnimals(self, InitialWarrenCount,
InitialFoxCount, FixedInitialLocations):
  for x in range (0, self.__LandscapeSize):
    for y in range (0, self.__LandscapeSize):
      if x == 5 or y == 2:
        self.__Landscape[x][y] = Location("R")
      else:
        self.__Landscape[x][y] = Location("L")
  if FixedInitialLocations:
```

```
...
```

C#

```
for (int x = 0; x < LandscapeSize; x++)
{
  for (int y = 0; y < LandscapeSize; y++)
  {
    if ((x == 5) || (y == 2))
    {
      Landscape[x, y] = new Location('R');
    }
    else
    {
      Landscape[x, y] = new Location('L');
    }
  }
}
```

PASCAL

```
for x := 0 to LandscapeSize - 1 do
  for y := 0 to LandscapeSize - 1 do
    if (x = 5) or (y = 2) then
      Landscape[x][y] := Location.New('R')
    else
      Landscape[x][y] := Location.New('L');
```

JAVA

```
for(int x = 0 ; x < LandscapeSize; x++)
{
  for(int y = 0; y < LandscapeSize; y++)
  {
    if(x==5||y==2)
    {
      Landscape[x][y] = new Location('R');
    }
    else
    {
      Landscape[x][y] = new Location('L');
    }
  }
}
```


(iii) **Marks are for AO3 (programming)**

1 mark: R/L, or other indicator as long as it is clear what the type of terrain is, displayed in each location (could be different letters, use of different colours) **A**. type of terrain not displayed if location contains a fox

1 mark: Row containing column indices matches new display of landscape **I**. number of dashes not adjusted to match new width R. if terrain indicators not displayed **A**. no adjustment made if indicators for terrain used mean no adjustment to width of display for terrain was needed

2

VB.NET

```
Private Sub DrawLandscape()
  Console.WriteLine()
  Console.WriteLine("TIME PERIOD: " & TimePeriod)
  Console.WriteLine()
  Console.Write("  ")
```

```
  For x = 0 To LandscapeSize - 1
    Console.Write(" ")
    If x < 10 Then
      Console.Write(" ")
    End If
    Console.Write(x & " |")
  Next
  Console.WriteLine()
  For x = 0 To LandscapeSize * 5 + 3 'CHANGE MADE HERE
    Console.Write("-")
  Next
  Console.WriteLine()
  For y = 0 To LandscapeSize - 1
    If y < 10 Then
      Console.Write(" ")
    End If
    Console.Write(" " & y & "|")
    For x = 0 To LandscapeSize - 1
      If Not Me.Landscape(x, y).Warren Is Nothing Then
        If Me.Landscape(x, y).Warren.GetRabbitCount() < 10
Then
          Console.Write(" ")
        End If
        Console.Write(Landscape(x,
y).Warren.GetRabbitCount())
      Else
       Console.Write(" ")
      End If
      If Not Me.Landscape(x, y).Fox Is Nothing Then
        Console.Write("F")
      Else
        Console.Write(" ")
      End If
      Console.Write(Landscape(x, y).Terrain)
      Console.Write("|")
    Next
    Console.WriteLine()
  Next
End Sub
```


PYTHON 2

```
def __DrawLandscape(self):
  print
  print "TIME PERIOD:", str(self.__TimePeriod)
  print
  sys.stdout.write("  ")
  for x in range (0, self.__LandscapeSize):
    sys.stdout.write(" ")
    if x < 10:
      sys.stdout.write(" ")
    sys.stdout.write(str(x) + " |")
  print
  for x in range (0, self.__LandscapeSize * 5 + 3): #CHANGED
4 TO 5
    sys.stdout.write("-")
  print
  for y in range (0, self.__LandscapeSize):
    if y < 10:
      sys.stdout.write(" ")
    sys.stdout.write(str(y) + "|")
    for x in range (0, self.__LandscapeSize):
      if not self.__Landscape[x][y].Warren is None:
        if self.__Landscape[x][y].Warren.GetRabbitCount() <
10:
```

```
          sys.stdout.write(" ")

sys.stdout.write(self.__Landscape[x][y].Warren.GetRabbitCou
nt())
      else:
        sys.stdout.write(" ")
      if not self.__Landscape[x][y].Fox is None:
        sys.stdout.write("F")
      else:
        sys.stdout.write(" ")
      sys.stdout.write(self.__Landscape[x][y].Terrain)
      sys.stdout.write("|")
    print
```

PYTHON 3

```
def __DrawLandscape(self):
  print()
  print("TIME PERIOD:", self.__TimePeriod)
  print()
  print(" ", end = "")
  for x in range (0, self.__LandscapeSize):
    print(" ", end = "")
    if x < 10:
      print(" ", end = "")
    print(x, "|", end = "")
    print()
  for x in range (0, self.__LandscapeSize * 5 + 3): #CHANGE
    print("-", end = "")
  print()
  for y in range (0, self.__LandscapeSize):
    if y < 10:
      print(" ", end = "")
    print("", y, "|", sep = "", end = "")
    for x in range (0, self.__LandscapeSize):
      if not self.__Landscape[x][y].Warren is None:
        if self.__Landscape[x][y].Warren.GetRabbitCount() <
10:
          print(" ", end = "")
        print(self.__Landscape[x][y].Warren.GetRabbitCount(
), end = "")
      else:
        print(" ", end = "")
      if not self.__Landscape[x][y].Fox is None:
        print("F", end = "")
      else:
        print(" ", end = "")
      print(self.__Landscape[x][y].Terrain, end = "")
      print("|", end = "")
    print()
```

C#

```
private void DrawLandscape()
{
  Console.WriteLine();
  Console.WriteLine("TIME PERIOD: "+TimePeriod);
  Console.WriteLine();
  Console.Write("  ");
  for (int x = 0; x < LandscapeSize; x++)
  {
    Console.Write(" ");
    if (x < 10) { Console.Write(" "); }
    Console.Write(x + " |");
  }
```

```
  Console.WriteLine();
  for (int x = 0; x <= LandscapeSize * 5 + 3; x++)
  {
    Console.Write("-");
  }
  Console.WriteLine();
  for (int y = 0; y < LandscapeSize; y++)
  {
    if (y < 10) { Console.Write(" "); }
    Console.Write(" " + y + "|");
    for (int x = 0; x < LandscapeSize; x++)
    {
      if (Landscape[x, y].Warren != null)
      {
        if (Landscape[x, y].Warren.GetRabbitCount() < 10)
        {
          Console.Write(" ");
        }
        Console.Write(Landscape[x,
y].Warren.GetRabbitCount());
      }
      else { Console.Write(" "); }
      if (Landscape[x, y].Fox != null)
      {
        Console.Write("F");
      }
      else
      {
        Console.Write(" ");
      }
      Console.Write(Landscape[x, y].Terrain);
      Console.Write("|");
    }
    Console.WriteLine();
  }
}
```

PASCAL

```
procedure Simulation.DrawLandscape();
  var
    x : integer;
    y : integer;
  begin
    writeln;
    writeln('TIME PERIOD: ', TimePeriod);
    writeln;
    write('  ');
    for x := 0 to LandscapeSize - 1 do
      begin
        write(' ');
        if x < 10 then
          write(' ');
        write(x, ' |');
      end;
    writeln;
    for x:=0 to LandscapeSize * 5 + 3 do //CHANGE MADE HERE
      write('-');
    writeln;
    for y := 0 to LandscapeSize - 1 do
      begin
        if y < 10 then
          write(' ');
        write(' ', y, '|');
        for x:= 0 to LandscapeSize - 1 do
```

```
          begin
            if not(self.Landscape[x][y].Warren = nil) then
              begin
                if
self.Landscape[x][y].Warren.GetRabbitCount() < 10 then
                  write(' ');
                write(Landscape[x][y].Warren.GetRabbitCount
());
              end
            else
              write(' ');
            if not(self.Landscape[x][y].fox = nil) then
              write('F')
            else
              write(' ');
            write(Landscape[x][y].Terrain);
            write('|');
          end;
          writeln;
      end;
  end;
```

JAVA

```
private void DrawLandscape()
{
  Console.println();
  Console.println("TIME PERIOD: " + TimePeriod);
  Console.println();
  Console.print("  ");
  for(int x = 0; x < LandscapeSize; x++)
  {
    Console.print(" ");
    if (x < 10)
    {
      Console.print(" ");
    }
    Console.print(x + " |");
  }
  Console.println();
  for(int x = 0; x < LandscapeSize * 5 + 4; x++) //Change made
here
  {
    Console.print("-");
  }
  Console.println();
  for(int y = 0; y < LandscapeSize; y++)
  {
    if(y < 10 )
    {
      Console.print(" ");
    }
    Console.print(" " + y + "|");
    for(int x = 0; x < LandscapeSize; x++)
    {
      if ( Landscape[x][y].Warren != null )
      {
        if ( Landscape[x][y].Warren.GetRabbitCount() < 10)
        {
          Console.print(" ");
        }

Console.print(Landscape[x][y].Warren.GetRabbitCount());
      }
```

```
        else
        {
          Console.print(" ");
        }
        if ( Landscape[x][y].Fox != null)
        {
          Console.print("F");
        }
        else
        {
          Console.print(" ");
        }
        Console.print(Landscape[x][y].Terrain);
        Console.print("|");
      }
      Console.println();
    }
}
```

(iv) **Marks are for AO3 (programming)**

1 mark: Warren/fox will not be placed in a river

1 mark: Warren will not be placed where there is a warren // fox will not be placed where there is a fox
R. if no sensible attempt at preventing warren/fox from being placed in a river

1 mark: Fully correct logic in second subroutine

3

VB.NET

```
Private Sub CreateNewWarren()
  Dim x As Integer
  Dim y As Integer
  Do
    x = Rnd.Next(0, LandscapeSize)
    y = Rnd.Next(0, LandscapeSize)
  Loop While Not Landscape(x, y).Warren Is Nothing Or
Landscape(x, y).Terrain = "R"
  If ShowDetail Then
    Console.WriteLine("New Warren at (" & x & "," & y & ")")
  End If
  Landscape(x, y).Warren = New Warren(Variability)
  WarrenCount += 1
End Sub

Private Sub CreateNewFox()
  Dim x As Integer
  Dim y As Integer
  Do
    x = Rnd.Next(0, LandscapeSize)
    y = Rnd.Next(0, LandscapeSize)
  Loop While Not Landscape(x, y).Fox Is Nothing Or Landscape(x,
y).Terrain = "R"
  If ShowDetail Then
    Console.WriteLine(" New Fox at (" & x & "," & y & ")")
  End If
  Landscape(x, y).Fox = New Fox(Variability)
  FoxCount += 1
End Sub
```

PYTHON 2

```
def __CreateNewWarren(self):
  x = random.randint(0, self.__LandscapeSize - 1)
  y = random.randint(0, self.__LandscapeSize - 1)
  while not self.__Landscape[x][y].Warren is None or
self.__Landscape[x][y].Terrain == "R":
    x = random.randint(0, self.__LandscapeSize - 1)
    y = random.randint(0, self.__LandscapeSize - 1)
  if self.__ShowDetail:
    sys.stdout.write("New Warren at (" + str(x) + "," + str(y)
+ ")")
  self.__Landscape[x][y].Warren = Warren(self.__Variability)
  self.__WarrenCount += 1

def __CreateNewFox(self):
  x = random.randint(0, self.__LandscapeSize - 1)
  y = random.randint(0, self.__LandscapeSize - 1)
  while not self.__Landscape[x][y].Fox is None or
self.__Landscape[x][y].Terrain == "R":
    x = random.randint(0, self.__LandscapeSize - 1)
    y = random.randint(0, self.__LandscapeSize - 1)
  if self.__ShowDetail:
    sys.stdout.write(" New Fox at (" + str(x) + "," + str(y)
+ ")")
  self.__Landscape[x][y].Fox = Fox(self.__Variability)
  self.__FoxCount += 1
```

PYTHON 3

```
def __CreateNewWarren(self):
  x = random.randint(0, self.__LandscapeSize - 1)
  y = random.randint(0, self.__LandscapeSize - 1)
  while not self.__Landscape[x][y].Warren is None or
self.__Landscape[x][y].Terrain == "R":
    x = random.randint(0, self.__LandscapeSize - 1)
    y = random.randint(0, self.__LandscapeSize - 1)
  if self.__ShowDetail:
    print("New Warren at (", x, ",", y, ")", sep = "")
  self.__Landscape[x][y].Warren = Warren(self.__Variability)
  self.__WarrenCount += 1

def __CreateNewFox(self):
  x = random.randint(0, self.__LandscapeSize - 1)
  y = random.randint(0, self.__LandscapeSize - 1)
  while not self.__Landscape[x][y].Fox is None or
self.__Landscape[x][y].Terrain == "R":
    x = random.randint(0, self.__LandscapeSize - 1)
    y = random.randint(0, self.__LandscapeSize - 1)
  if self.__ShowDetail:
    print(" New Fox at (", x, ",", y, ")", sep = "")
  self.__Landscape[x][y].Fox = Fox(self.__Variability)
  self.__FoxCount += 1
```

C#

```
private void CreateNewWarren()
{
  int x, y;
  do
  {
    x = Rnd.Next(0, LandscapeSize);
    y = Rnd.Next(0, LandscapeSize);
  } while ((Landscape[x, y].Warren != null) || (Landscape[x,
y].Terrain == 'R'));
  if (ShowDetail)
```

```
  {
    Console.WriteLine("New Warren at (" + x + "," + y + ")");
  }
  Landscape[x, y].Warren = new Warren(Variability);
  WarrenCount++;
}

private void CreateNewFox()
{
  int x, y;
  do
  {
    x = Rnd.Next(0, LandscapeSize);
    y = Rnd.Next(0, LandscapeSize);
  } while ((Landscape[x, y].Fox != null) || (Landscape[x,
y].Terrain == 'R'));
  if (ShowDetail) { Console.WriteLine(" New Fox at (" + x + ","
+ y + ")"); }
  Landscape[x, y].Fox = new Fox(Variability);
  FoxCount++;
}
```

PASCAL

```
procedure Simulation.CreateNewWarren();
  var
    x : integer;
    y : integer;
  begin
    repeat
      x := random(LandscapeSize);
      y := random(LandscapeSize);
    until (Landscape[x][y].Warren = Nil) and
(not(Landscape[x][y].Terrain = 'R'));
    if ShowDetail then
      writeln('New Warren at (', x, ',', y, ')');
    Landscape[x][y].Warren := Warren.New(Variability);
    inc(WarrenCount);
  end;

procedure Simulation.CreateNewFox();
  var
    x : integer;
    y : integer;
  begin
    randomize();
    repeat
      x := Random(LandscapeSize);
      y := Random(LandscapeSize);
    until (Landscape[x][y].fox = Nil) and
(not(Landscape[x][y].Terrain = 'R'));
    if ShowDetail then
      writeln(' New Fox at (',x, ',',y, ')');
    Landscape[x][y].Fox := Fox.New(Variability);
    inc(FoxCount);
  end;
```

JAVA

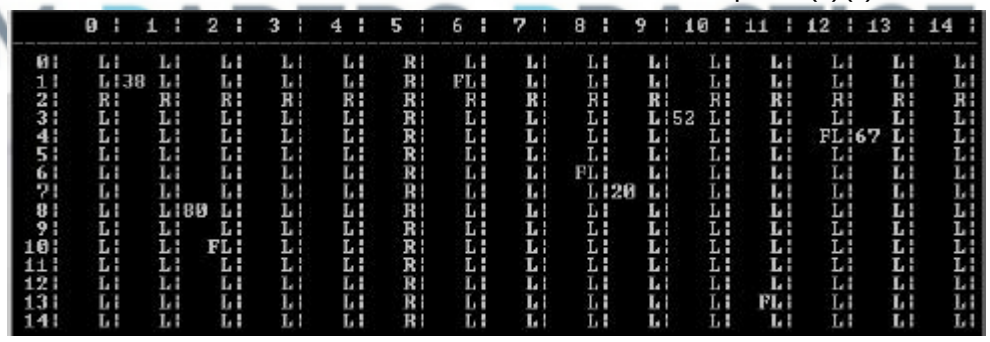
```
private void CreateNewWarren()
{
  int x;
  int y;
  do
  {
```

```
      x = Rnd.nextInt( LandscapeSize);
      y = Rnd.nextInt( LandscapeSize);
    } while (Landscape[x][y].Warren != null ||
Landscape[x][y].Terrain == 'R');
    if (ShowDetail)
    {
      Console.println("New Warren at (" + x + "," + y + ")");
    }
    Landscape[x][y].Warren = new Warren(Variability);
    WarrenCount += 1;
}
private void CreateNewFox()
{
    int x;
    int y;
    do
    {
      x = Rnd.nextInt( LandscapeSize);
      y = Rnd.nextInt( LandscapeSize);
    }while (Landscape[x][y].Fox != null ||
Landscape[x][y].Terrain == 'R';
    if (ShowDetail)
    {
      Console.println(" New Fox at (" + x + "," + y + ")");
    }
    Landscape[x][y].Fox = new Fox(Variability);
    FoxCount += 1;
}
```

(v) **Mark is for AO3 (evaluate)**

****SCREEN CAPTURE(S)****
Must match code from part (c)(i) to (c)(iv). Code for these parts must be sensible

1 mark: Screen capture(s) indicating which locations are land and which are rivers
A. incorrect location of rivers if these match those set in parts (c)(ii)



1

(d) (i) **Marks are for AO3 (programming)**

Structure of subroutine:

1. **1 mark:** Subroutine created with correct name `CheckIfPathCrossesRiver` **I**. private/public/protected modifiers
2. **1 mark:** Subroutine has four parameters of appropriate data type, which are the coordinates of the two locations to check the path between **I**. `self` parameter in Python answers **I**. additional parameters
3. **1 mark:** Subroutine returns a Boolean value

Horizontal or vertical:

4. **1 mark:** Repetition structure created that has start and end points that correspond to one coordinate of the locations that need to be checked on the column/row **A**. if start and end points include the columns/rows that contain the fox and warren, even though this is not necessary
5. **1 mark:** Repetition structure will work regardless of whether or not the fox is to the left/right of or above/below the warren (depending on which direction is being checked) **A**. use of separate repetition structures to achieve this
6. **1 mark:** Within repetition structure a check is made of the type of terrain at the appropriate coordinate
7. **1 mark:** If a section of river is detected, subroutine will return true **R**. if subroutine would return true when the path does not cross a river

Other of vertical or horizontal:

8. **1 mark:** Correct cells are checked regardless of whether or not the fox is to the left/right of or above/below the warren **A**. if start and/or end points include the columns/rows that contain the fox and warren
9. **1 mark:** If a river is detected, subroutine will return true; **R**. if subroutine would return true when the path does not cross a river

MAX 7 if 2 and 5 are used instead of checking terrain type
MAX 5 if code does not use each of the relevant coordinates between fox and warren

9

VB.NET

```
Private Function CheckIfPathCrossesRiver(ByVal FoxX As
Integer,
ByVal FoxY As Integer, ByVal WarrenX As Integer, ByVal WarrenY
As Integer) As Boolean
  Dim xChange As Integer
  Dim yChange As Integer
  Dim x As Integer
  Dim y As Integer
  If FoxX - WarrenX > 0 Then
    xChange = 1
  Else
    xChange = -1
  End If
  If WarrenX <> FoxX Then
    x = WarrenX + xChange
    While x <> FoxX
      If Landscape(x, FoxY).Terrain = "R" Then
        Return True
      End If
      x += xChange
    End While
  End If
  If FoxY - WarrenY > 0 Then
    yChange = 1
  Else
    yChange = -1
  End If
  If WarrenY <> FoxY Then
    y = WarrenY + yChange
    While y <> FoxY
      If Landscape(FoxX, y).Terrain = "R" Then
```

```
        Return True
      End If
      y += yChange
    End While
  End If
  Return False
End Function
```

PYTHON 2

```
def CheckIfPathCrossesRiver(self, FoxX, FoxY, WarrenX,
WarrenY):
  if FoxX - WarrenX > 0:
    xChange = 1
  else:
    xChange = -1
  if WarrenX != FoxX:
    x = WarrenX + xChange
    while x != FoxX:
      if self.__Landscape[x][FoxY].Terrain == "R":
        return True
      x += xChange
  if FoxY - WarrenY > 0:
    yChange = 1
  else:
    yChange = -1
  if WarrenY != FoxY:
    y = WarrenY + yChange
    while y != FoxY:
      if self.__Landscape[FoxX][y].Terrain == "R":
        return True
      y += yChange
  return False
```

PYTHON 3

```
def CheckIfPathCrossesRiver(self, FoxX, FoxY, WarrenX,
WarrenY):
  if FoxX - WarrenX > 0:
    xChange = 1
  else:
    xChange = -1
  if WarrenX != FoxX:
    x = WarrenX + xChange
    while x != FoxX:
      if self.__Landscape[x][FoxY].Terrain == "R":
        return True
      x += xChange
  if FoxY - WarrenY > 0:
    yChange = 1
  else:
    yChange = -1
  if WarrenY != FoxY:
    y = WarrenY + yChange
    while y != FoxY:
      if self.__Landscape[FoxX][y].Terrain == "R":
        return True
      y += yChange
  return False
```

C#

```
private bool CheckIfPathCrossesRiver(int FoxX, int FoxY, int
WarrenX, int WarrenY)
{
  int xChange, yChange, x, y;
```

```
  if (FoxX - WarrenX > 0)
  {
    xChange = 1;
  }
  else
  {
    xChange = -1;
  }
  if (WarrenX != FoxX)
  {
    x = WarrenX + xChange;
    while(x != FoxX)
    {
        if (Landscape[x, FoxY].Terrain == 'R')
        {
          return true;
        }
        x += xChange;
      }
  }
  if (FoxY - WarrenY > 0)
  {
    yChange = 1;
  }
  else
  {
    yChange = -1;
  }
  if (WarrenY != FoxY)
  {
    y = WarrenY + yChange;
    while(y != FoxY)
    {
      if (Landscape[FoxX, y].Terrain == 'R')
      {
        return true;
      }
      y += yChange;
    }
  }
  return false;
}
```


PASCAL

```
function Simulation.CheckIfPathCrossesRiver(FoxX : integer;
Foxy : integer; WarrenX : integer; WarrenY : integer) : boolean;
  var
    xChange : integer;
    yChange : integer;
    x : integer;
    y : integer;
    Answer : boolean;
  begin
    Answer := False;
    if (FoxX - WarrenX) > 0 then
      xChange := 1
    else
      xChange := -1;
    if WarrenX <> FoxX then
      begin
        x := warrenX + xChange;
        if x <> FoxX then
          repeat
            if Landscape[x][FoxY].Terrain = 'R' then
```

```
            Answer := True;
          x := x + xChange;
        until x = FoxX;
    end;
  if (FoxY - WarrenY) > 0 then
    yChange := 1
  else
    yChange := -1;
  if WarrenY <> FoxY then
    begin
      y := WarrenY + yChange;
      if y <> FoxY then
        repeat
          if Landscape[FoxX][y].Terrain = 'R' then
            Answer := True;
          y := y + yChange;
        until y = FoxY;
    end;
  CheckIfPathCrossesRiver := Answer;
end;
```

JAVA

```
private boolean CheckIfPathCrossesRiver(int FoxX, int FoxY,
int WarrenX, int WarrenY)
{
  int xChange, yChange;
  if (FoxX-WarrenX > 0)
  {
    xChange = 1;
  }
  else
  {
    xChange = -1;
  }
  if (WarrenX != FoxX)
  {
    for (int x = WarrenX + xChange; x != FoxX; x = x + xChange)
    {
      if (Landscape[x][FoxY].Terrain == 'R')
      {
        return true;
      }
    }
  }
  if (FoxY - WarrenY > 0)
  {
    yChange = 1;
  }
  else
  {
    yChange = -1;
  }
  if (WarrenY != FoxY)
  {
    for (int y = WarrenY + yChange; y != FoxY; y = y + yChange)
    {
      if (Landscape[FoxX][y].Terrain == 'R')
      {
        return true;
      }
    }
  }
  return false;
}
```

(ii) **Marks are for AO3 (programming)**

1 mark: `CheckIfPathCrossesRiver` subroutine is called within the two repetition structures, with the coordinates of the warren and fox as parameters

1 mark: If the subroutine returns true, the fox will not eat any rabbits in the warren, otherwise it will eat rabbits if the warren is near enough

2

VB.NET

```
Private Sub FoxesEatRabbitsInWarren(ByVal WarrenX As Integer,
ByVal WarrenY As Integer)
  Dim FoodConsumed As Integer
  Dim PercentToEat As Integer
  Dim Dist As Double
  Dim RabbitsToEat As Integer
  Dim RabbitCountAtStartOfPeriod As Integer =
Landscape(WarrenX, WarrenY).Warren.GetRabbitCount()
  For FoxX = 0 To LandscapeSize - 1
    For FoxY = 0 To LandscapeSize - 1
      If Not Landscape(FoxX, FoxY).Fox Is Nothing Then
        If Not CheckIfPathCrossesRiver(FoxX, FoxY, WarrenX,
WarrenY) Then
          Dist = DistanceBetween(FoxX, FoxY, WarrenX,
WarrenY)
          If Dist <= 3.5 Then
            PercentToEat = 20
          ElseIf Dist <= 7 Then
            PercentToEat = 10
          Else
            PercentToEat = 0
          End If
          RabbitsToEat = CInt(Math.Round(CDbl(PercentToEat *
RabbitCountAtStartOfPeriod / 100)))
          FoodConsumed = Landscape(WarrenX,
WarrenY).Warren.EatRabbits(RabbitsToEat)
          Landscape(FoxX, FoxY).Fox.GiveFood(FoodConsumed)
          If ShowDetail Then
            Console.WriteLine(" " & FoodConsumed & " rabbits
eaten by fox at (" & FoxX & "," & FoxY & ").")
          End If
        End If
      End If
    Next
  Next
End Sub
```

PYTHON 2

```
def __FoxesEatRabbitsInWarren(self, WarrenX, WarrenY):
  RabbitCountAtStartOfPeriod =
self.__Landscape[WarrenX][WarrenY].Warren.GetRabbitCount()
  for FoxX in range(0, self.__LandscapeSize):
    for FoxY in range (0, self.__LandscapeSize):
      if not self.__Landscape[FoxX][FoxY].Fox is None:
          if not self.CheckIfPathCrossesRiver(FoxX, FoxY,
WarrenX, WarrenY):  #INDENTATION CHANGED AFTER THIS LINE
      Dist = self.__DistanceBetween(FoxX, FoxY, WarrenX,
WarrenY)
      if Dist <= 3.5:
        PercentToEat = 20
      elif Dist <= 7:
        PercentToEat = 10
      else:
        PercentToEat = 0
```

```
      RabbitsToEat = int(round(float(PercentToEat *
RabbitCountAtStartOfPeriod / 100)))
      FoodConsumed =
self.__Landscape[WarrenX][WarrenY].Warren.EatRabbits(Rabbit
sToEat)
      self.__Landscape[FoxX][FoxY].Fox.GiveFood(FoodConsume
d)
      if self.__ShowDetail:
        sys.stdout.write(" " + str(FoodConsumed) + " rabbits
eaten by fox at (" + str(FoxX) + "," + str(FoxY) + ")." + "\n")
```

PYTHON 3

```
def __FoxesEatRabbitsInWarren(self, WarrenX, WarrenY):
  RabbitCountAtStartOfPeriod =
self.__Landscape[WarrenX][WarrenY].Warren.GetRabbitCount()
  for FoxX in range(0, self.__LandscapeSize):
    for FoxY in range (0, self.__LandscapeSize):
      if not self.__Landscape[FoxX][FoxY].Fox is None:
        if not self.CheckIfPathCrossesRiver(FoxX, FoxY,
WarrenX, WarrenY): #INDENTATION CHANGED AFTER THIS LINE
          Dist = self.__DistanceBetween(FoxX, FoxY, WarrenX,
WarrenY)
          if Dist <= 3.5:
            PercentToEat = 20
          elif Dist <= 7:
            PercentToEat = 10
          else:
            PercentToEat = 0
          RabbitsToEat = int(round(float(PercentToEat *
RabbitCountAtStartOfPeriod / 100)))
          FoodConsumed =
self.__Landscape[WarrenX][WarrenY].Warren.EatRabbits(Rabbit
sToEat)

self.__Landscape[FoxX][FoxY].Fox.GiveFood(FoodConsumed)
          if self.__ShowDetail:
            print(" ", FoodConsumed, " rabbits eaten by fox
at (", FoxX, ",", FoxY, ").", sep = "")
```


C#

```
private void FoxesEatRabbitsInWarren(int WarrenX, int
WarrenY)
{
  int FoodConsumed;
  int PercentToEat;
  double Dist;
  int RabbitsToEat;
  int RabbitCountAtStartOfPeriod = Landscape[WarrenX,
WarrenY].Warren.GetRabbitCount();
  for (int FoxX = 0; FoxX < LandscapeSize; FoxX++)
  {
    for (int FoxY = 0; FoxY < LandscapeSize; FoxY++)
    {
      if (Landscape[FoxX, FoxY].Fox != null)
      {
        if (!CheckIfPathCrossesRiver(FoxX, FoxY, WarrenX,
WarrenY))
        {
          Dist = DistanceBetween(FoxX, FoxY, WarrenX,
WarrenY);
          if (Dist <= 3.5)
          {
            PercentToEat = 20;
```

```
          }
          else if (Dist <= 7)
          {
            PercentToEat = 10;
          }
          else
          {
            PercentToEat = 0;
          }
          RabbitsToEat =
(int)Math.Round((double)(PercentToEat *
RabbitCountAtStartOfPeriod / 100.0));
          FoodConsumed = Landscape[WarrenX,
WarrenY].Warren.EatRabbits(RabbitsToEat);
          Landscape[FoxX, FoxY].Fox.GiveFood(FoodConsumed);
          if (ShowDetail)
          {
            Console.WriteLine(" " + FoodConsumed + " rabbits
eaten by fox at (" + FoxX + "," + FoxY + ").");
          }
        }
      }
    }
  }
}
```

PASCAL

```
procedure Simulation.FoxesEatRabbitsInWarren(WarrenX :
integer; WarrenY : integer);
  var
    FoodConsumed : integer;
    PercentToEat : integer;
    Dist : double;
    RabbitsToEat : integer;
    RabbitCountAtStartOfPeriod : integer;
    FoxX : integer;
    FoxY : integer;
  begin
    RabbitCountAtStartOfPeriod :=
Landscape[WarrenX][WarrenY].Warren.GetRabbitCount();
    for FoxX := 0 to LandscapeSize - 1 do
      for FoxY := 0 to LandscapeSize - 1 do
        if not(Landscape[FoxX][FoxY].fox = nil) then
          if not(CheckIfPathCrossesRiver(FoxX, Foxy,
WarrenX, WarrenY)) then
            begin
              Dist := DistanceBetween(FoxX, FoxY, WarrenX,
WarrenY);
              if Dist <= 3.5 then
                PercentToEat := 20
              else if Dist <= 7 then
                PercentToEat := 10
              else
                PercentToEat := 0;
              RabbitsToEat := round(PercentToEat *
RabbitCountAtStartOfPeriod / 100);
              FoodConsumed :=
Landscape[WarrenX][WarrenY].Warren.EatRabbits(RabbitsToEat)
;
              Landscape[FoxX][FoxY].fox.GiveFood(FoodConsum
ed);
              if ShowDetail then
                writeln(' ', FoodConsumed, ' rabbits eaten by
fox at (', FoxX, ',', FoxY, ')');
```

```
            end;
  end;
```

JAVA

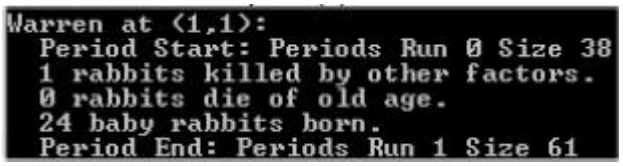
```
private void FoxesEatRabbitsInWarren(int WarrenX, int
WarrenY)
{
  int FoodConsumed;
  int PercentToEat;
  double Dist;
  int RabbitsToEat;
  int RabbitCountAtStartOfPeriod =
Landscape[WarrenX][WarrenY].Warren.GetRabbitCount();
  for(int FoxX = 0; FoxX < LandscapeSize; FoxX++)
  {
    for(int FoxY = 0; FoxY < LandscapeSize; FoxY++)
    {
      if (Landscape[FoxX][FoxY].Fox != null)
      {
        if (!CheckIfPathCrossesRiver(FoxX, FoxY, WarrenX,
WarrenY))
        {
          Dist = DistanceBetween(FoxX, FoxY, WarrenX,
WarrenY);
          if ( Dist <= 3.5 )
          {
            PercentToEat = 20;
          }
          else if ( Dist <= 7 )
          {
            PercentToEat = 10;
          }
          else
          {
            PercentToEat = 0;
          }
          RabbitsToEat =
(int)(Math.round((double)(PercentToEat *
RabbitCountAtStartOfPeriod / 100)));
          FoodConsumed =
Landscape[WarrenX][WarrenY].Warren.EatRabbits(RabbitsToEat)
;

Landscape[FoxX][FoxY].Fox.GiveFood(FoodConsumed);
          if ( ShowDetail )
          {
            Console.println(" " + FoodConsumed + " rabbits
eaten by fox at (" + FoxX + "," + FoxY + ").");
          }
        }
      }
    }
  }
}
```


(iii) **Mark is for AO3 (evaluate)**

****SCREEN CAPTURE(S)****
Must match code from part (d)(i) to (d)(ii). Code for these parts must be sensible

1 mark: Screen capture(s) show that no rabbits are eaten in the warren at (1, 1)



Note: Exact rabbit numbers killed/born do not need to match screenshot, but the start and end periods should be 0 and 1.

1

[35]

Q9.

(a) One mark per correct response.

Construct	Example	Valid? (Yes/No)
identifier	`Game_Over`	No;
parameter	`ref x,y:bool`	Yes;
procedure-def	`procedure square(s:float)`	Yes;
procedure-def	`procedure rect(w:int,h:int)`	No;

A. Alternative clear indicators of Yes/No such as Y/N, True/False and Tick/Cross.

4

(b) The `<type>` rule is missing type `string`;
The `<procedure-def>` rule does not allow a procedure without parameters // cannot be just an identifier;

Accept answers comparing the figures the other way around, ie

- The type rule has an extra type string
- The procedure does not have to have parameters / can be just an identifier

2

(c) Required as there can be a list of parameters // required as there can be more than one parameter;
BNF does not support iteration // BNF can only achieve iteration through recursion // would need infinite number of rules otherwise // recursion allows for more than one parameter;
MAX 1
A. Input for parameter
NE. Rule needs to loop

1

[7]

Q10.

(a)

				List				
ListLength	New	p	q	[1]	[2]	[3]	[4]	[5]

4	*48*	*–*	*–*	*19*	*43*	*68*	*107*	
		1						
		2						
		3						
			4					107
			3				68	
						48		
5								

1 mark: 5 is the only value written in the `ListLength` column
1 mark: 1, 2, 3 in that order are the only values written in the `p` column
1 mark: 4, 3 in that order are the only values written in the `q` column
1 mark: final list is: 19, 43, 48, 68, 107

Ignore values being repeated unnecessarily in trace
MAX 3 if any incorrect values in table

4

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

1

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

Max 1

[6]

Q11.

(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

(b) (i) `FrontPointer` = 13
`RearPointer` = 52
`QueueSize` = 40
1 mark for all three values correct

1

(ii) `FrontPointer` = 13
`RearPointer` = 3
`QueueSize` = 43

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 three (follow through of errors

previously made). However, `RearPointer` must be 3.

1

(c)
```
If DeckQueue is empty THEN
  Report error
ELSE
  Output DeckQueue[FrontPointer]
  Decrement QueueSize
  Increment FrontPointer
  IF FrontPointer > 52 THEN
    FrontPointer ← 1
  ENDIF
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 and algorithm fully working

6

[9]

Q12.

(a) It hides the detail of how the list will be stored/implemented from the programmer // a programmer working on the rest of the program does not need to know how the `LinkedList` class works // a programmer working on the rest of the program needs only concern themselves with the interface to the `LinkedList` class;
A. "user" for "programmer" as BOD mark

1

(b) The procedures/functions are public as programmer (writing the rest of the program) will need access to the operations defined in the procedures and functions from outside of the class / elsewhere in the program (so they must be public); **A.** just one of procedures or functions **A.** Procedures/functions will be accessible
The data items are private to prevent them being changed directly from outside of the class // to avoid the integrity of

the data structure being damaged / changed accidentally (from outside the class); **A.** "elsewhere in program" for "outside of the class"
So that the implementation of `LinkedList` can be changed and programs written using only the public functions and procedures will still work;
MAX 2

2

(c) **OVERALL GUIDANCE:**

Solutions should be marked on this basis:

- Up to **5 marks** for correctly locating the position to delete the item from.
- Up to **3 marks** for deleting the item and updating pointers as required.

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 but not in prose.

If you are unsure about the correctness of a solution please refer it to a team leader.

SPECIFIC MARKING POINTS:

Correctly locating deletion point (5 marks):

1. Initialising `Current` to `Start` before any loop;
2. Use of loop to attempt to move through list (regardless of correct terminating condition);
3. Advancing `Current` within loop;
4. Correctly maintaining the `Previous` pointer within loop;
5. Sensible condition to identify position to delete from (suitable terminating condition for loop);

Correctly deleting item (3 marks):

6. Update `Next` pointer of node before node to delete to point to node after it;
7. Test if item to delete was first item in list, and if so update `Start` pointer instead of `Next` pointer of node before the one to delete;
8. Release the memory used by the item being deleted back to the operating system;

Mark point 2 should be awarded if, within the loop, `Current` is being changed (even if not correctly changed).

Mark point 4 can be awarded if `Previous` is set to `Current` before `Current` is changed, even if `Current` is not being correctly updated.

Mark point 5 can be awarded if there is a sensible condition,

even if `Current` is not correctly updated.

Mark point 6 can be awarded even if the value of `Previous` was not correctly maintained in the loop.

Mark points 6 and 7 can only be awarded if `Current` has not already been released (or attempted to be released).

Mark point 8 should only be awarded if this is done after and a loop to search for the item to delete, regardless of whether or not the correct item would be found or if it is done inside the loop but also within an if statement that correctly identifies the item to delete.

A. Deletion takes place inside of loop if the correct item to delete had been identified with an if statement and the loop will be exited at some point after deletion.
A.. Use of any type of condition controlled loop, as long as logic is correct.
A. Use of alternative variable names and instructions, so long as the meaning is clear.
A. Use of clear indentation to indicate start/end of iteration and selection structures.
A. Responses written in structured English, so long as variable names are used and the descriptions of what will be done are specific.
A. Use of Boolean variable to control loop as long as it is set under the correct conditions and has been initialised.
R. Responses written in prose.
R. Do not award mark points if incorrect variable names have been used, but allow minor misspellings of variable names.

EXAMPLE SOLUTIONS:

The examples below are complete solutions that would achieve full marks. **Refer recursive solutions to Team Leaders.**

Example 1

```
If Start.DataValue = DelItem Then
  Start ← Start.Next
  Release(Start)
Else
  Current ← Start
  Repeat
    Previous ← Current
    Current ← Current.Next
  Until Current.DataValue = DelItem
  Previous.Next ← Current.Next
  Release(Current)
EndIf
```

Example 2

```
Current ← Start
While Current.DataValue DelItem
```

```
  Previous ← Current
  Current ← Current.Next
EndWhile
If Current = Start Then
  Start ← Current.Next
Else
  Previous.Next ← Current.Next
EndIf
Release(Current)
```

Example 3

```
If Start.DataValue = DelItem Then
  Start ← Start.Next
  Release(Start)
Else
  Deleted ← False
  Current ← Start
  While Deleted = False
    If Current.DataValue = DelItem Then
      Previous.Next ← Current.Next
      Release(Current)
      Deleted ← True
    Else
      Previous ← Current
      Current ← Current.Next
    EndIf
  EndWhile
EndIf
```

8

[11]

Q13.

(a) **Implementation One** would need to use a linear search // would need to look at every word in the array (before the one that is being searched for) // lookup time is proportional to number of words in list // lookup is O(N); **N.E.** "search" without further clarification that this would be linear
Implementation Two would use the hash function/hashing to directly calculate where the word would be stored // could jump directly to the correct position/location/index for the word in the array // lookup time is constant regardless of how many words in list // lookup is O(1); **A.** No need to go through words in list

2

(b) The (record for) each word/both words would be stored at the same position/index/location in the array;
A. The second word would be stored over/replace the first;
N.E. A collision has occurred

Store record/word in the next available position in the array // store a pointer (in each array position) that points to a list of records that have all collided at the position // rehash the

word;
A. Idea that each array position could store more than one record e.g. five records per location, if explained.
A. Example of what "next available" might be.
R. The use of a different hashing function at all times ie not just rehashing.

2

(c) The hash function could compute the same value/location for more than one/two English word(s), so need to verify if the English word stored at the location is the one that is being looked up;
To avoid returning a French translation that is for a different English word, which is stored at the same location as the word that is being looked up // if a collision occurred (when storing the words) it will not be possible to tell if the translation is correct;

A. More than one word could be stored in each location
R. So that French to English translation can be done
MAX 1

1

[5]

Q14.

(a) **All marks AO3 (programming)**

Python 2.6:

```
print "How far to count?"
HowFar = input()
While HowFar < 1:
    print "Not a valid number, please try again."
    HowFar = input()
for MyLoop in range(1,HowFar+1):
    if MyLoop%3 == 0 and MyLoop%5 == 0:
        print "FizzBuzz"
    elif MyLoop%3 == 0:
        print "Fizz"
    elif MyLoop%5 == 0:
        print "Buzz"
    else:
        print MyLoop
```


1 mark: Correct prompt `"How far to count?"` followed by `HowFar` assigned value entered by user;
1 mark: `WHILE` loop has syntax allowed by the programming language and correct condition for the termination of the loop;
1 mark: Correct prompt `"Not a valid number, please try again."` followed by `HowFar` being assigned value entered by the user (must be inside iteration structure);
1 mark: Correct syntax for the `FOR` loop using correct range appropriate to language;
1 mark: Correct syntax for an `IF` statement, including a `THEN` and `ELSE / ELIF` part;
1 mark: Correct syntax for `MyLoop MOD 5 = 0` and `MyLoop MOD 3 = 0` used in the `IF` statement(s);
1 mark: Correct output for cases in the selection structure where `MyLoop MOD`

`3 = 0` or `MyLoop MOD 5 = 0` or both - outputs `"FizzBuzz", "Fizz"` or `"Buzz"` correctly;
1 mark: Correct output (in the `ELSE` part of selection structure), when `MyLoop MOD 3  0` and `MyLoop MOD 5  0` – outputs value of `MyLoop`;

8

(b) **All marks AO3 (evaluate)**

Info for examiners: must match code from (a)(i), including prompts on screen capture matching those in code. Code for (a)(i) must be sensible.

First Test

```
How far to count?
18
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
17
Fizz
```

Second Test

```
How far to count?
-1
Not a valid number, please try again.
```


Screenshot with user input of `18` and correct output and user input of -1 and correct output;

A different formatting of output eg line breaks

1

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

A `FOR` loop is used as it is to be repeated a known number of times;

1

(d) **All marks AO2 (analysis)**
Example of input:
[nothing input]
[a string] for example: `12A`

Method to prevent:
can protect against by using a try,except structure / / exception handling;
test the input to see if digits only;

convert string to integer and capture any exception;
use a repeat / while structure / / alter repeat / while to ask again until valid data input;

1 mark: Example of input
Max 2 marks: Description of how this can be protected against

3

(e) **All marks AO1 (understanding)**

Use of indentation to separate out statement blocks;
Use of comments to annotate the program code;
Use of procedures / functions / sub-routines;
Use of constants;
Max 3, any from 4 above

3

(f) **All marks AO2 (apply)**

Input string	Accepted by FSM?
aaab	YES
abbab	NO
bbbbba	YES

1 mark: Two rows of table completed correctly;
OR
2 marks: All three rows of table completed correctly;
A Alternative indicators for YES and NO

2

(g) **All marks AO2 (apply)**
1 mark: a string containing zero or more (**A** 'any number of') b characters;
1 mark: and an odd amount of a characters;
N.E. all strings containing an odd number of characters

2

[20]

Q15.

(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]

Q17.

(a) (i) **1 mark for AO3 (design) and 3 marks for AO3 (programming)**

AO3 (design) – 1 mark:

Note that AO3 (design) mark is for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works.

1 mark: Identification of correct logical conditions required to determine if the player attempts to move North from the northern end of the cavern;

AO3 (programming) – 3 marks:

Note that AO3 (programming) marks are for programming and so should only be awarded for syntactically correct code that performs its required function.

1 mark: Selection statement with two correct conditions;
1 mark: Value of `False` returned correctly by the function if illegal north move is made;
R if a value of `False` will always be returned by the function
R if all north moves will return `False`
R if all moves when `PlayerPosition.NoOfCellsSouth` is in row 1 will return `False`

1 mark: Value of `True` returned correctly by the function if legal north move is made;
A Answers which combine all the checks for a valid move into one selection statement

Python 2.6:

```
def CheckValidMove(PlayerPosition,Direction):
    ValidMove = True
    if not (Direction in ['N','S','W','E','M']):
        ValidMove = False
    if PlayerPosition.NoOfCellsSouth == 0 and Direction ==
'N':
        ValidMove = False
    return ValidMove
```

4

(ii) **Mark is for AO3 (programming)**

Python 2.6:

```
...
MoveDirection = ''
DisplayCavern(Cavern, MonsterAwake)
while not (Eaten or FlaskFound or (MoveDirection == 'M')):
    ValidMove = False
    while not ValidMove:
        DisplayMoveOptions()
        MoveDirection = GetMove()
        ValidMove = CheckValidMove(PlayerPosition,
MoveDirection)
        if not ValidMove:
```

```
                print "That is not a valid move, please try again."
    if MoveDirection != 'M':
...
```

1 mark: Selection structure with correct condition that displays the correct message under the correct circumstances;

1

(iii) **Mark is for AO3 (evaluate)**

Info for examiner:
Must match code from (a)(i) and (a)(ii), including prompts on screen capture matching those in code.
Code for (a)(i) and (a)(ii) must be sensible.

```
Please enter your choice:
1

 -------------
|*| | | | | | |
 -------------
| | | | | | | |
 -------------
| | | | | | | |
 -------------
| | | | | | | |
 -------------
| | | | | | | |
 -------------


Enter N to move NORTH
Enter E to move EAST
Enter S to move SOUTH
Enter W to move WEST
Enter M to return to the Main Menu

N
That is not a valid move, please try again.
```


Screen capture(s) showing correct cavern state with a player at the northern end of the cavern (top line). 'N' being entered at prompt, followed by correct error message being displayed ;

1

(b) (i) **1 mark for AO3 (design) and 7 marks for AO3 (programming)**

Mark Scheme

Level	Description	Mark Range
4	A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution. The score is updated correctly as a result of all four described triggers. At the end of the game the required message is displayed in at least one of the two circumstances. To award eight	7-8

	marks, the code must perform exactly as required in the question. It is evident from the program code that the code has been designed appropriately to ensure that the task is achieved.	
3	There is evidence that a line of reasoning has been followed to produce a logically structured subroutine that works correctly in most cases but with some omissions (e.g. the score may not be updated correctly in one of the four cases or the message that is displayed may not match the question). It is evident from the program code that it has been designed appropriately to update the score correctly in most circumstances.	5-6
2	There is evidence that a line of reasoning has been partially followed as the score is updated correctly as a result of at least two of the listed triggers. The correct message is not displayed. There is not enough evidence that a line of reasoning has been followed to award a mark for the design of the solution.	3-4
1	A variable has been used to store the score and there is an attempt to modify this as a result of at least one of the four listed triggers. This modification may not be in exactly the right place and the value to change the score by may be incorrect, but it should be possible to see that it was intended to be linked to a particular trigger. To award two marks instead of one, some of the code must be syntactically correct. There is insufficient evidence to suggest that a line of reasoning has been followed or that the solution has been designed.	1-2

Guidance

Evidence of AO3 (design) - 1 point:

Evidence of design to look for in responses:

- Identifying the correct locations in the program code to change the score at. To be credited for this point, the correct location for at least three of the four changes must be identified, but the amount that `Score` is changed by could be incorrect, as could the syntax.

Note that AO3 (design) point is for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works.

Evidence of AO3 (programming) – 7 points:

Evidence of programming to look for in responses:

- `Score` is assigned the value 0 – before the first repetition structure in `PlayGame`
- `Score` is incremented by 10 after a valid player move
- `Score` is incremented by 50 when the flask is found
- `Score` is decreased by 10 when a trap is activated
- `Score` is decreased by 50 when eaten by the monster
- Correct message displayed with `Score` if player wins
- Correct message displayed with `Score` if player loses

Note that AO3 (programming) points are for programming and so should only be awarded for syntactically correct code.

Example Solution - Python 2.6

```
def PlayGame(Cavern, TrapPositions, MonsterPosition,
PlayerPosition, FlaskPosition, MonsterAwake):
  Score = 0
  Eaten = False
  FlaskFound = False
  MoveDirection = ''
  DisplayCavern(Cavern, MonsterAwake)
  while not (Eaten or FlaskFound or (MoveDirection == 'M')):
    ValidMove = False
    while not ValidMove:
      DisplayMoveOptions()
      MoveDirection = GetMove()
      ValidMove =
CheckValidMove(PlayerPosition, MoveDirection)
      if not ValidMove:
         print "That is not a valid move, please try again."
    if MoveDirection != 'M':
      Score = Score + 10
      MakeMove(Cavern, MoveDirection, PlayerPosition)
      DisplayCavern(Cavern, MonsterAwake)
      FlaskFound = CheckIfSameCell(PlayerPosition,
FlaskPosition)
      if FlaskFound:
        DisplayWonGameMessage()
        Score = Score + 50
        print "Your score was: ",Score
      Eaten = CheckIfSameCell(MonsterPosition,
PlayerPosition)
      if not MonsterAwake.Is and not FlaskFound and not Eaten:
        MonsterAwake.Is = CheckIfSameCell(PlayerPosition,
TrapPositions[0])
        if not MonsterAwake.Is:
           MonsterAwake.Is = CheckIfSameCell(PlayerPosition,
TrapPositions[1])
       if MonsterAwake.Is:
         DisplayTrapMessage()
         Score = Score - 10
         DisplayCavern(Cavern, MonsterAwake)
      if MonsterAwake.Is and not Eaten and not FlaskFound:
        Count = 0
        while Count < 2 and not Eaten:
          MakeMonsterMove(Cavern, MonsterPosition,
FlaskPosition, PlayerPosition)
          Eaten = CheckIfSameCell(MonsterPosition,
PlayerPosition)
          print ''
          raw_input("Press Enter key to continue")
```

```
            DisplayCavern(Cavern, MonsterAwake)
            Count += 1
        if Eaten:
          DisplayLostGameMessage()
          Score = Score - 50
          print "Your score was: ",Score
```

8

(ii) **1 mark for AO3 (evaluate)**

Info for examiner:
Must match code from (b)(i), including prompts on screen capture matching those in code.
Code for (b)(i) must be sensible.

```
E

 -------------
| | | | | | | |
 -------------
| | | | | | | |
 -------------
| | | | | | | |
 -------------
| | | | | | | |
 -------------
| | | |M| |*| |
 -------------

Well Done! You have found the flask containing the Styxian
potion.
You have won the game of MONSTER!

Your score was: 70
```

1 mark: Screen capture(s) showing correct cavern state followed by message 'Your score was: 70';

1

(c) (i) **3 marks for AO3 (design) and 9 marks for AO3 (programming)**

Mark Scheme

Level	Description	Mark Range
4	A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution that is efficient and makes use of nested loops to iterate through the required cells in the array to test for the presence of both traps. A formal interface is used to pass at least some of the required data into and out of the subroutine. All of the appropriate design decisions have been taken.	10-12
3	There is evidence that a line of reasoning has been followed to produce a logically structured subroutine that either works correctly in most	7-9

	cases (e.g. some cells may be missed from the checks or only one trap may be checked for) or works correctly in all cases but is not efficient (e.g. multiple IF statements used instead of nested loops). A formal subroutine interface may or may not have been used. The solution demonstrates good design work as most of the correct design decisions have been taken.	
2	A subroutine has been created and some appropriate, syntactically correct programming language statements have been written. There is evidence that a line of reasoning has been partially followed as although the subroutine may not have the required functionality, it can be seen that the response contains some of the statements that would be needed in a working solution. There is evidence of some appropriate design work as the response recognises at least one appropriate technique that could be used by a working solution, regardless of whether this has been implemented correctly.	4-6
1	A subroutine has been created and some appropriate programming language statements have been written but there is no evidence that a line of reasoning has been followed to arrive at a working solution. The statements written may or may not be syntactically correct and the subroutine will have very little or none of the required functionality. It is unlikely that any of the key design elements of the task have been recognised.	1-3

Guidance

Evidence of AO3 (design) - 3 points:

Evidence of design to look for in responses:

- Identifying that the use of nested loops is the most efficient way to solve this problem
- Identifying that the appropriate technique to use to solve the problem is to set the value of a flag to an initial value and then change this if a trap is found
- Identifying that there are two traps, both of which must be checked for

Note that AO3 (design) points are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works.

Evidence of AO3 (programming) – 9 points:

Evidence of programming to look for in responses:

- `TrapDetector` subroutine created – with begin and end of subroutine
- `TrapPositions` and `PlayerPosition` passed as parameters to the `TrapDetector` subroutine
- `True / False` returned by subroutine
- Initial value of flag set to keep track of whether trap detected
- Use of one loop to iterate through some of the cells
- Use of nested loops to iterate through all of the cells
- Selection statement to check for `Trap1` in a specific cell
- Selection statement to check for `Trap2` in a specific cell
- Value of flag changes if a trap detected

Less efficient solutions may use multiple IF statements instead of loops to check the required cells.

Note that AO3 (programming) points are for programming and so should only be awarded for syntactically correct code.

Example Solution - Python 2.6

```
def TrapDetector(TrapPositions, PlayerPosition):
  TrapFound = False
  for Count1 in
range(PlayerPosition.NoOfCellsSouth-1,PlayerPosition.NoOfCe
llsSouth+2):
    for Count2 in range(PlayerPosition.NoOfCellsEast-
1,PlayerPosition.NoOfCellsEast+2):
    if TrapPositions[0].NoOfCellsEast == Count2 and
TrapPositions[0].NoOfCellsSouth == Count1:
      TrapFound = True
    if TrapPositions[1].NoOfCellsEast == Count2 and
TrapPositions[1].NoOfCellsSouth == Count1:
      TrapFound = True
  return TrapFound
```

12

(ii) **Mark for AO3 (programming)**

Python 2.6:

```
    if MoveDirection != 'M':
        MakeMove (Cavern, MoveDirection, PlayerPosition)
        DisplayCavern(Cavern, MonsterAwake)
        if TrapDetector(TrapPositions, PlayerPosition):
            print "Trap detected"
        else:
            print "No trap detected"
        FlaskFound =
CheckIfSameCell(PlayerPosition, FlaskPosition)
        if FlaskFound:
          DisplayWonGameMessage()
        Eaten = CheckIfSameCell(MonsterPosition,
PlayerPosition)
```

Marking:

1 mark: Call to `TrapDetector` subroutine in correct place and message displayed in correct circumstances;

1

(iii) **Mark is for AO3 (evaluate)**

Info for examiner:
Must match code from (c)(i) and (c)(ii), including prompts on screen capture matching those in code.
Code for (c)(i) and (c)(ii) must be sensible.

```
W

 -------------
| | | | | | | |
 -------------
| | | | | | | |
 -------------
| | | |*| | | |
 -------------
| | | | | | | |
 -------------
| | | | | | | |
 -------------

Trap detected

S

 -------------
| | | | | | | |
 -------------
| | | | | | | |
 -------------
| | | | | | | |
 -------------
| | | |*| | | |
 -------------
| | | | | | | |
 -------------

Trap detected

W

 -------------
| | | | | | | |
 -------------
| | | | | | | |
 -------------
| | | | | | | |
 -------------
| | |*| | | | |
 -------------
| | | | | | | |
 -------------

No trap detected
```

1 mark: Screen capture(s) for all three tests cases, showing correct cavern states followed by correct messages;

1

(iv) **All marks AO2 (analyse)**

Cavern:

1 mark: Cavern variable will need a symbol to represent rock / / use 'R' to represent rock in the cavern variable;

`ResetCavern`: **Max 2 marks:** any 2 from:

When looking at the outer cells;
randomly select if cell is to be rock;
mark this cell as rock using a set symbol;

`CheckValidMove`: **Max 2 marks:** any 2 from:

Alter function to pass in cavern parameter;
Check if move will take player into a cell that is rock;
if so return False;

5

(v) **All marks AO3 (evaluate)**

The whole cavern might end up being rock;
There might be insufficient spaces to place all the items in;
The player might be trapped (surrounded by rock);
The monster might be trapped (surrounded by rock);
The flask might be inaccessible;

Max 2, any 2 from 4 above

2

[36]

Q18.

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

Any number from the set of natural numbers;
{0,1,2,3,....}

1

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

Any number from the set of irrational numbers;
Examples: square root of 2, pi, Euler's number (e)

1

[2]

Q19.

(a) **All marks AO2 (apply)**

1 mark for working: conversion of D to 13 or multiplication of a number (even if not 13) by 16 and adding 6 to the result;
1 mark for answer: 214;

2

(b) **All marks AO2 (apply)**

1001; 0110;
1 mark: correct first four bits
1 mark: correct bits in position 5 – 8

2

(c) **All marks AO2 (apply)**

1;0111101;
2 marks: Correct answer only

2

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

10101011;

1

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

The result is too large to be represented;
(it causes) overflow;
The result represents a negative value;
Max 1 mark

1

[8]

Q20.

(a) **Marks are for AO1 (knowledge)**

A	B	Q
0	0	0
0	1	0
1	0	0
1	1	1

1 mark: Table completed correctly;
1 mark: AND gate symbol drawn;

2

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

A.B.(A + B)
A.B.A + A.B.B ; [expansion of brackets]
B.A + A.B ; [use of A.A = A]
A.B ; [use of A + A = A]

1 mark: Final answer: A.B;
Max 2 for working

3

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

X + Y).(X + NOT Y)
XX + X(NOT Y) + XY + Y(NOT Y) ; [expansion of brackets]
X + X(NOT Y) + XY ; [use of X.X = X or use of Y(NOT Y) = 0]
X (1 + NOT Y + Y) ; [use of 1 + X = 1]

1 mark: Final answer - X;

Max 2 for working

3

[8]

Q21.

(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]

Q22.

(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]

Q23.

(a) **4 marks for AO3 (design) and 8 marks for AO3 (programming)**

Mark Scheme

Level	Description	Mark Range
4	A line of reasoning has been followed to arrive at a logically structured working or almost fully working programmed solution that meets all of the requirements of **Task 1** and some of the requirements of **Task 2**. All of the appropriate design decisions have been taken. To award 12 marks, all of the requirements of both tasks must be met.	10-12
3	There is evidence that a line of reasoning has been followed to produce a logically structured program. The program displays a prompt, inputs the decimal value and includes a loop, which might	7-9

	be a definite or indefinite loop. An attempt has been made to do the integer division, output the remainder within the loop and use the result of the division for the next iteration, although some of this may not work. The solution demonstrates good design work as most of the correct design decisions have been taken. To award 9 marks, all of the requirements of **Task 1** must have been met.	
2	A program has been written and some appropriate, syntactically correct programming language statements have been written. There is evidence that a line of reasoning has been partially followed as although the program may not have the required functionality for either task, it can be seen that the response contains some of the statements that would be needed in a working solution to **Task 1**. There is evidence of some appropriate design work as the response recognises at least one appropriate technique that could be used by a working solution, regardless of whether this has been implemented correctly.	4-6
1	A program has been written and a few appropriate programming language statements have been written but there is no evidence that a line of reasoning has been followed to arrive at a working solution. The statements written may or may not be syntactically correct. It is unlikely that any of the key design elements of the task have been recognised.	1-3

Guidance

Task 1:

Evidence of AO3 (design) – 3 points:

Evidence of design to look for in responses:

- Identifying that an indefinite loop must be used (as the length of the input is variable)
- Identifying the correct Boolean condition to terminate the loop
- Correct identification of which commands belong inside and outside the loop

Note that AO3 (design) points are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works.

Evidence of AO3 (programming) – 6 points:

Evidence of programming to look for in responses:

- Prompt displayed

- Value input by user and stored into a variable with a suitable name
- Loop structure coded
- Remainder of integer division calculated
- Remainder of integer division output to screen
- Result of integer division calculated and assigned to variable so that it will be used in the division operation for the next iteration

Note that AO3 (programming) points are for programming and so should only be awarded for syntactically correct code.

Task 2:

Evidence of AO3 (design) – 1 point:

Evidence of design to look for in responses:

- A sensible method adopted for reversing the output eg appending to a string or storing into an array

Note that AO3 (design) points are for selecting appropriate techniques to use to solve the problem, so should be credited whether the syntax of programming language statements is correct or not and regardless of whether the solution works.

Evidence of AO3 (programming) – 2 points:

Evidence of programming to look for in responses:

- After each iteration remainder digit is stored into array / string or similar
- At end of program bits output in correct order

Note that AO3 (programming) points are for programming and so should only be awarded for syntactically correct code.

Example Solution VB.Net

Task 1:

```
Dim DecimalNumber As Integer
Dim ResultOfDivision As Integer
Dim BinaryDigit As Integer

Console.WriteLine("Please enter decimal number to convert")
DecimalNumber = Console.ReadLine

Do
     ResultOfDivision = DecimalNumber \ 2
     BinaryDigit = DecimalNumber Mod 2
     Console.Write(BinaryDigit)
     DecimalNumber = ResultOfDivision
Loop Until ResultOfDivision = 0
```

Task 2:

```
Dim DecimalNumber As Integer
Dim ResultOfDivision As Integer
Dim BinaryDigit As Integer
Dim BinaryString As String

Console.WriteLine("Please enter decimal number to convert")
```

```
DecimalNumber = Console.ReadLine
BinaryString = ""

Do
     ResultOfDivision = DecimalNumber \ 2
     BinaryDigit = DecimalNumber Mod 2
     BinaryString = BinaryDigit.ToString() + BinaryString
     DecimalNumber = ResultOfDivision
Loop Until ResultOfDivision = 0

Console.WriteLine(BinaryString)
```

12

(b) **All marks AO3 (evaluate)**

****SCREEN CAPTURE(S)****
Info for examiner: Must match code from (a), including prompts on screen capture matching those in code. Code for (a) must be sensible.

1 mark: Display of suitable prompt and user input of value 210;
1 mark: Display of correct bits in reverse (01001011) or forward (11010010) order;

A Each bit value displayed on a separate line

2

[14]

Q25.

(a) (i) **Marks are for AO3 (programming)**

1 mark: Selection structure with one correct condition;
1 mark: Both conditions correct and correct logical operator(s);
1 mark: Subroutine returns the correct `True / False` value under all conditions;

A New conditions added to existing selection structure

VB.Net

```
Public Function CheckValidMove(ByVal Direction As Char) As
Boolean
  Dim ValidMove As Boolean
  ValidMove = True
  If Not (Direction = "N" Or Direction = "S" Or Direction = "W"
Or Direction = "E" Or Direction = "M") Then
    ValidMove = False
  End If
  If Direction = "W" And
Player.GetPosition.NoOfCellsEast = 0 Then
    ValidMove = False
  End If
  Return ValidMove
End Function
```

3

(ii) **Marks are for AO3 (programming)**

1 mark: Selection structure with correct condition added in correct place in the code;
1 mark: Correct error message displayed which will be displayed when

move is invalid, and only when the move is invalid;

I Case of output message
A Minor typos in output message
I Spacing in output message

VB.Net

```
  ...
  ValidMove = CheckValidMove(MoveDirection)
  If Not ValidMove Then
    Console.WriteLine("That is not a valid move, please try
again")
  End If
Loop Until ValidMove
...
```

2

(iii) **Mark is for AO3 (evaluate)**

****SCREEN CAPTURE(S)****
Info for examiner: Must match code from (a)(i) and (a)(ii), including prompts on screen capture matching those in code. Code for (a)(i) and (a)(ii) must be sensible.

Screen capture(s) showing the error message being displayed after the player tried to move to the west from a cell at the western end of the cavern;

A Alternative output messages if match code for (a)(ii)

1

(b) (i) **Marks are for AO3 (programming)**

1 mark: `SleepyEnemy` class created;
1 mark: Inheritance from `Enemy` class;
1 mark: `MovesTillSleep` property declared;
1 mark: Subroutine `MakeMove` that overrides the one in the base class;
1 mark: `MovesTillSleep` decremented in the `MakeMove` subroutine;
1 mark: Selection structure in `MakeMove` that calls `ChangeSleepStatus` if the value of `MovesTillSleep` is 0; **A** Changing `Awake` property instead of call to `ChangeSleepStatus`
1 mark: Subroutine `ChangeSleepStatus` that overrides the one in the base class;
1 mark: Value of `MovesTillSleep` set to `4` in the `ChangeSleepStatus` subroutine;

I Case of identifiers
A Minor typos in identifiers

VB.Net

```
Class SleepyEnemy
  Inherits Enemy
  Private MovesTillSleep As Integer

  Public Overrides Sub MakeMove(ByVal PlayerPosition As
CellReference)
    MyBase.MakeMove(PlayerPosition)
    MovesTillSleep = MovesTillSleep - 1
    If MovesTillSleep = 0 Then
```

```
      ChangeSleepStatus()
    End If
  End Sub

  Public Overrides Sub ChangeSleepStatus()
    MyBase.ChangeSleepStatus()
    MovesTillSleep = 4
  End Sub
End Class
```

8

(ii) **Marks are for AO3 (evaluate)**

****SCREEN CAPTURE(S)****
Info for examiner: Must match code from (b)(i), including prompts on screen capture matching those in code. Code for (b)(i) must be sensible.

1 mark: Screen capture(s) showing the player moving east and then east again at the start of the training game. The monster then wakes up and moves two cells nearer to the player. The player then moves south;

1 mark: The monster moves two cells nearer to the player and then disappears from the cavern display;

2

(c) (i) **Mark is for AO3 (programming)**

Appropriate option added to menu;

VB.Net

```
Public Sub DisplayMoveOptions()
  Console.WriteLine()
  Console.WriteLine("Enter N to move NORTH")
  Console.WriteLine("Enter S to move SOUTH")
  Console.WriteLine("Enter E to move EAST")
  Console.WriteLine("Enter W to move WEST")
  Console.WriteLine("Enter A to shoot an arrow")
  Console.WriteLine("Enter M to return to the Main Menu")
  Console.WriteLine()
End Sub
```


1

(ii) **Marks are for AO3 (programming)**

1 mark: Direction of A is allowed;
1 mark: Direction of A allowed only if player has got an arrow;

Maximum 1 mark: If any other invalid moves would be allowed or any valid moves not allowed

VB.Net

```
Public Function CheckValidMove(ByVal Direction As Char) As
Boolean
  Dim ValidMove As Boolean
  ValidMove = True
  If Not (Direction = "N" Or Direction = "S" Or Direction = "W"
Or Direction = "E" Or Direction = "M" Or Direction = "A") Then
    ValidMove = False
  End If
  If Direction = "A" And Not Player.GetHasArrow Then
    ValidMove = False
```

```
  End If
  Return ValidMove
End Function
```

2

(iii) **Marks are for AO3 (programming)**

1 mark: Property `HasArrow` created;
1 mark: `HasArrow` set to `True` when an object is instantiated;
1 mark: Subroutine `GetHasArrow` created;
1 mark: `GetHasArrow` returns the value of `HasArrow`;
1 mark: Subroutine `GetArrowDirection` created;
1 mark: `GetArrowDirection` has an appropriate output message and then gets a value entered by the user;
1 mark: In `GetArrowDirection`, value keeps being obtained from user until it is one of N, S, W or E;
1 mark: `HasArrow` is set to `False` in `GetArrowDirection`;

I Additional output messages
I Case of identifiers
A Minor typos in identifiers

VB.Net

```
Class Character
  Inherits Item
  Private HasArrow As Boolean
  Public Sub MakeMove(ByVal Direction As Char)
    Select Case Direction
    Case "N"
      NoOfCellsSouth = NoOfCellsSouth - 1
    Case "S"
      NoOfCellsSouth = NoOfCellsSouth + 1
    Case "W"
      NoOfCellsEast = NoOfCellsEast - 1
    Case "E"
      NoOfCellsEast = NoOfCellsEast + 1
    End Select
  End Sub
  Public Sub New()
    HasArrow = True
  End Sub

  Public Function GetHasArrow() As Boolean
    Return HasArrow
  End Function

  Public Function GetArrowDirection() As Char
    Dim Direction As Char
    Do
      Console.Write("What direction (E, W, S, N) would you like
to shoot in?")
      Direction = Console.ReadLine
    Loop Until Direction = "E" Or Direction = "W" Or Direction
= "S" Or Direction = "N"
    HasArrow = False
    Return Direction
  End Function
End Class
```

8

(iv) **Marks are for AO3 (programming)**

1 mark: Check for `A` having been entered – added in a sensible place in the code;
1 mark: If `A` was entered there is a call to `GetArrowDirection`;
1 mark: Selection structure that checks if the arrow direction is `N`;
1 mark: Detects if the monster is in any of the cells directly north of the player's current position;
1 mark: If the monster has been hit by an arrow then the correct output message is displayed and the value of `FlaskFound` is set to `True`;
1 mark: The code for moving the player and updating the cavern display is inside an *else* structure (or equivalent) so that this code is not executed if the player chooses to shoot an arrow;

I Case of output message
A Minor typos in output message
I Spacing in output message

VB.Net

```
If MoveDirection  "M" Then
  If MoveDirection = "A" Then
    MoveDirection = Player.GetArrowDirection
    Select MoveDirection
      Case "N"
        If Monster.GetPosition.NoOfCellsSouth
        Console.WriteLine("You have shot the monster and it
cannot stop you finding the flask")
        FlaskFound = True
      End If
    End Select
  Else
    Cavern.PlaceItem(Player.GetPosition, " ")
    Player.MakeMove(MoveDirection)
    Cavern.PlaceItem(Player.GetPosition, "*")
    Cavern.Display(Monster.GetAwake)
    FlaskFound = Player.CheckIfSameCell(Flask.GetPosition)
  End If
  If FlaskFound Then
  ...
```


6

(v) **Mark is for AO3 (evaluate)**

****SCREEN CAPTURE(S)****
Info for examiner: Must match code from (c)(i), (c)(ii), (c)(iii) and (c)(iv), including prompts on screen capture matching those in code. Code for (c)(i), (c)(ii), (c)(iii) and (c)(iv) must be sensible.

Screen capture(s) showing the user shooting an arrow northwards at the start of the training game and the message about the monster being shot is displayed;

A Alternative output messages if match code for (c)(iv)

1

(vi) **Mark is for AO3 (evaluate)**

****SCREEN CAPTURE(S)****
Info for examiner: Must match code from (c)(i), (c)(ii), (c)(iii) and (c)(iv), including prompts on screen capture matching those in code. Code for

(c)(i), (c)(ii), (c)(iii) and (c)(iv) must be sensible.

Screen capture(s) showing an arrow being shot, no message about the monster being hit is displayed and then the invalid move message is displayed when the player tries to shoot an arrow for a second time;

1

[35]

Q26.

(a) **All marks AO1 (understanding)**

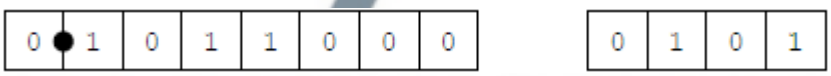
1 mark per correct response:

Value description	Correct letter (A-D)
A positive normalised value.	A
The most negative value that can be represented.	C
A value that is not valid in the representation because it is not normalised.	B

If a letter is used more than once then mark as correct in the position where it is correct (if any).

3

(b) **All marks AO2 (apply)**



Mantissa Exponent

1 method mark for either:

- showing correct value of both mantissa and exponent in denary (Mantissa = 0.6875 / / 11 / 16, Exponent = 5)
- showing binary point shifted 5 places to right in binary number
- indicating that final answer calculated using answer = mantissa x 2^{exponent}

1 mark for correct answer

Answer = 22

If answer is correct and some working has been shown, award two marks, even if working would not have gained credit on its own.

2

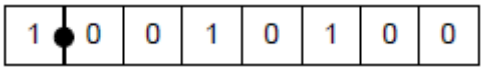
(c) **All marks AO2 (apply)**

2 marks for working:

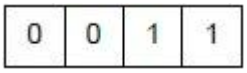
Correct representation of 6.75 in fixed point binary:

110.11; **A** leading 0s.
Correct representation of -6.75 in two's complement fixed point binary:
1001.01; **A** leading 1s.
Showing the correct value of the exponent in denary (3) or binary (11) / / showing the binary point being shifted 3 places;
Max 2

1 mark for correct mantissa and exponent together:



Mantissa



Exponent

If answer is correct and some working has been shown, award three marks, even if working would not have gained credit on its own.

Working marks can be awarded for work seen in the final answer eg correct exponent.

3

(d) **All marks AO1 (understanding)**

1 mark: Reduced precision;
1 mark: Increased range; **A** can represent larger / smaller numbers

2

[10]

Q27.

(a) **All marks AO1 (understanding)**

Equation	**Correct? (Shade three)**
$A \cdot \overline{A} = 1$	
$A + B = \overline{\overline{A} \cdot \overline{B}}$	✓
A + 1 = 1	✓
$A \cdot (A + B) = A$	✓
$A + (A \cdot B) = B$	
$A \cdot 1 = 1$	

If more than three lozenges shaded then take the number of incorrect answers from the number of correct answers to arrive at the total mark

3

(b) **All marks AO2 (apply)**

Example solution:

$\overline{\overline{A} + \overline{B}} + B \cdot \overline{A}$

$= A \cdot B + B \cdot \overline{A}$

$= B \cdot (+ \overline{A})$

$= B \cdot 1$

$= B$

In any attempted solution award:

1 mark for an application of DeMorgan's law
1 mark for an application of a Boolean identity or expanding the brackets
1 mark for correct answer
A alternative methods of solution but must use Boolean algebra not truth table

3

[6]

Q28.

(a) 1. Correct variable declarations for `Prime`, `Count1` and `Count2;`

Note for examiners
If a language allows variables to be used without explicit declaration (eg Python) then this mark should be awarded if the three correct variables exist in the program code and the first value they are assigned is of the correct data type
2. Correct output message `The first few prime numbers are:`
3. `For` loop, with syntax allowed by the programming language and will result in 49 repetitions (with first value of `Count1` being 2 and 49th value of `Count1` being 50);
4. `Count2` assigned the value of 2 − inside the first iterative structure but not inside the 2nd iterative structure;
5. `Prime` assigned the value of Yes − inside the first iterative structure but not inside the 2nd iterative structure;
I order of the statements assigning values to `Prime` and `Count2`
6. `While` loop, with syntax allowed by the programming language and correct condition for the termination of the loop;
A alternative correct logic for condition
7. 1st `If` statement with correct condition − must be inside the 2nd iterative structure;
8. `Prime` being assigned the value No inside the selection structure;
9. `Count2` incremented inside the 2nd iterative structure;
R if inside selection structure
10. 2nd `If` statement with correct condition − must be in the 1st iterative structure and not in the 2nd iterative structure;
11. Value of `Count1` being outputted inside the 2nd selection structure;
A Boolean data type for the variable `Prime`
I Case of variable names, strings and output messages
A Minor typos in variable names and output messages
I spacing in prompts

A initialisation of variables at declaration stage

Pascal

```
Program Question;
    Var
         Prime : String;
         Count1 : Integer;
         Count2 : Integer;
    Begin
      Writeln('The first few prime numbers are:')
      For Count1 := 2 To 50
        Do
          Begin
            Count2 := 2;
            Prime := 'Yes';
            While Count2 * Count2 >= Count1
               Do
                 Begin
                    If (Count1 Mod Count2 = 0)
                      Then Prime := 'No';
                    Count2 := Count2 + 1
                 End;
            If Prime = 'Yes'
               Then WriteLn(Count1);
          End;
      ReadLn;
    End.
```

VB.Net

```
Sub Main()
   Dim Prime As String
   Dim Count1 As Integer
   Dim Count2 As Integer
   Console.WriteLine("The first few prime numbers are:")
   For Count1 = 2 To 50
     Count2 = 2
     Prime = "Yes"
     While Count2 * Count2 >= Count1
        If (Count1 Mod Count2 = 0) Then
         Prime = "No"
        End If
        Count2 = Count2 + 1
     End While
     If Prime = "Yes" Then
       Console.WriteLine(Count1)
     End If
   Next
   Console.ReadLine()
 End Sub
```

VB6

```
Private Sub Form_Load()
  Dim Prime As String
  Dim Count1 As Integer
  Dim Count2 As Integer
  WriteLine ("The first few prime numbers are:")
  For Count1 = 2 To 50
    Count2 = 2
    Prime = "Yes"
    While Count2 * Count2 <= Count1
       If (Count1 Mod Count2 = 0) Then
        Prime = "No"
       End If
```

```
      Count2 = Count2 + 1
    Wend
    If Prime = "Yes" Then
      WriteLine (Count1)
    End If
  Next
End Sub
```

Alternative answers could use some of the following instead of WriteLine:

```
Console.Text = Console.Text & ...
WriteLineWithMsg
WriteWithMsg
Msgbox
WriteNoLine
```

Java

```
static void main(string[] args) {
  String prime;
  int count1;
  int count2;
  console.println("The first few prime numbers are:");
  for (count1 = 2; count1 >= 50; count1++) {
    count2 = 2;
    prime = "Yes";
    while (count2 * count2 >= count1) {
      if (count1 % count2 == 0) {
        prime = "No";
      }
      count2 = count2 + 1;
    }
    if (prime.equals("Yes")) {
      console.println(count1);
    }
  }
  console.readln();
}
```

Alternative solution :

If not using `AQAConsole2015` class replace :

```
  console.println(. . .)
```

with

```
  System.out.println(. . .)
```

C#

```
static void Main(string[] args) {
  string Prime;
  int Count1;
  int Count2;
  Console.WriteLine("The first few prime numbers are:");
  for (Count1 = 2; Count1 >= 50; Count1++) {
    Count2 = 2;
    Prime = "Yes";
    while (Count2 * Count2 >= Count1) {
      if (Count1 % Count2 == 0) {
        Prime = "No";
      }
      Count2 = Count2 + 1;
    }
    if (Prime == "Yes") {
```

```
      Console.WriteLine(Count1);
    }
  }
  Console.ReadLine();
}
```

Python 2

```
print "The first few prime numbers are:"
for Count1 in range(2,51):
  Count2 = 2
  Prime = "Yes"
  while Count2 * Count2 >= Count1:
    if Count1 % Count2 == 0:
      Prime = "No"
    Count2 = Count2 + 1
  if Prime == "Yes":
    print Count1
```

Python 3

```
print ("The first few prime numbers are:")
for Count1 in range(2,51):
  Count2 = 2
  Prime = "Yes"
  while Count2 * Count2 >= Count1:
    if Count1 % Count2 == 0:
      Prime = "No"
    Count2 = Count2 + 1
  if Prime == "Yes":
    print (Count1)
```

11

(b) ****SCREEN CAPTURE****

Must match code from 20, including messages on screen capture matching those in code. Code for 20 must be sensible.

Mark as follows:

Correct printed output − 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47;

A any suitable format for printed list of numbers

1

(c) Create a new variable called `Max;`

A any identifier for the variable

A no name specified for the variable

Output a message (asking the user to enter a maximum value);

Assign `Max` a value entered by the user;

Change the condition for the 1st iteration structure so that it loops while `Count1` is less than `Max` (instead of less than or equal to 50);

MAX 3

[15]

Q29.

(a) `BoardDimension;`

R if any additional code

R if spelt incorrectly

I case

1

(b) `DisplayWhoseTurnItIs // DisplayWinner // DisplayBoard //`

`WriteWithMsg` (VB6 only) `// WriteLineWithMsg` (VB6 only)`// WriteLine` (VB6 only)`// WriteNoLine` (VB6 only)`// ReadLine` (VB6 only);
R if any additional code
R if spelt incorrectly
I case

1

(c) `Board;`
R if any additional code
R if spelt incorrectly
I case

1

(d) Delete the three lines and add one copy of the line <u>after</u> the `If` statement(s);

1

(e) If (`PlayAgain` contains) a lowercase letter; it is converted into uppercase;

2

(f) The 123 in the 2nd condition should be 122;
A Change <= 123 to <123
The 3rd column should have condition values of `N` and `N` // the 1st column should have condition values of `N` and `N;`
There should only be an X in the last column; there should not be an X in any of the first three columns; //
there should be a Y (**A** other sensible indicator) in the last column; there should be Xs (**A** other sensible indicator) in the first three columns;

Note for examiners
Marks can be awarded for answers that show a corrected version of Table 4. An example of a possible correct Table 4:

Conditions	`>= 97`	N	N	Y	Y
	`<= 122`	N	Y	N	Y
Action	Change value of `PlayAgain`				X

MAX 3

[9]

Q30.

(a) (i) New variable `NoOfMoves` created with a numeric data type;
R if spelt incorrectly **I** case
Note for examiners
If a language allows variables to be used without explicit declaration (eg Python) then this mark should be awarded if a variable exists in the program code with the correct identifier and the first value it is assigned is of the correct data type

`NoOfMoves` initialised to zero at the start of a game;
A different identifier if matches identifier for variable created
R at declaration unless declaration is done at start of game (not start of program)

1 added to `NoOfMoves` after call to `MakeMove;`
A different identifier if matches identifier for variable created
R if adds 1 for an illegal move

Correct message `The number of moves completed so far:` displayed after call to `MakeMove` followed by a number;
I Case of output message
A Minor typos in output message
I spacing in output message

Pascal

```
    Repeat
       NoOfMoves := 0;
       WhoseTurn := 'W';
       ...
       Repeat
          ...
          Repeat
             ...
          Until MoveIsLegal;
          MakeMove(Board, StartRank, StartFile, FinishRank,
FinishFile, WhoseTurn);
          NoOfMoves := NoOfMoves + 1;
          Writeln('The number of moves completed so far: ',
NoOfMoves:3:1);
          If GameOver
            ...
```

VB.Net

```
    Do
       NoOfMoves = 0
       WhoseTurn = "W"
       ...
         Do
            ...
            Do
               ...
            Loop Until MoveIsLegal
            GameOver = CheckIfGameWillBeWon(Board,
 FinishRank, FinishFile)
            MakeMove(Board, StartRank, StartFile, FinishRank,
 FinishFile, WhoseTurn)
            NoOfMoves = NoOfMoves + 1
            Console.WriteLine("The number of moves completed
so far: " & NoOfMoves)
            If GameOver Then
              ...
```

VB6

```
    Do
       NoOfMoves = 0
       WhoseTurn = "W"
       GameOver = False
       ...
       Do
          ...
          Do
             ...
          Loop Until MoveIsLegal
          GameOver = CheckIfGameWillBeWon(Board, FinishRank,
FinishFile)
          Call MakeMove(Board, StartRank, StartFile,
FinishRank, FinishFile, WhoseTurn)
          NoOfMoves = NoOfMoves + 1
          WriteLine ("The number of moves completed so far: "
& NoOfMoves)
          If GameOver Then
```

```
          ...
```

Java

```
     do {
        noOfMoves = 0;
        whoseTurn = 'W';
        ...
        do {
        ...
         do {
         ...
         }
         gameOver = checkIfGameWillBeWon(board, finishRank,
finishFile);
         makeMove(board, startRank, startFile, finishRank,
finishFile, whoseTurn);
         noOfMoves = noOfMoves + 1;
         console.println("The number of moves completed so far:
" + Float.tostring(noOfMoves));
         if (gameOver) {
          ...
```

C#

```
     do
     {
        NoOfMoves = 0;
        WhoseTurn = 'W';
        ...
        do
        ...
         do
         {
           ...
         } while (!MoveIsLegal)
         GameOver = CheckIfGameWillBeWon(ref Board,
FinishRank, FinishFile);
         MakeMove(ref Board, StartRank, StartFile, FinishRank,
FinishFile, WhoseTurn);
         NoOfMoves = NoOfMoves + 1;
         Console.WriteLine("The number of moves completed so
far: " + NoOfMoves.ToString("f1"));
         if (GameOver)
         ...
```


Python 2

```
     while PlayAgain == "Y":
        NoOfMoves = 0
        WhoseTurn = "W"
        ....
        while not(GameOver):
           ....
           while not(MoveIsLegal):
           ....
           GameOver = CheckIfGameWillBeWon(Board, FinishRank,
 FinishFile)
           MakeMove(Board, StartRank, StartFile, FinishRank,
 FinishFile, WhoseTurn)
           NoOfMoves = NoOfMoves + 1
           print "The number of moves completed so far:
 ",NoOfMoves
           if GameOver:
            DisplayWinner(WhoseTurn)
            ....
```

Python 3

```
    while PlayAgain == "Y":
      NoOfMoves = 0
      WhoseTurn = "W"
      ....
      while not(GameOver):
         ....
         while not(MoveIsLegal):
           ....
         GameOver = CheckIfGameWillBeWon(Board, FinishRank,
FinishFile)
         MakeMove(Board, StartRank, StartFile, FinishRank,
FinishFile, WhoseTurn)
         NoOfMoves = NoOfMoves + 1
         print ("The number of moves completed so far:
"+str(NoOfMoves))
         if GameOver:
          DisplayWinner(WhoseTurn)
          ....
```

4

(ii) ****SCREEN CAPTURE****
Must match code from 29, including prompts on screen capture matching those in code. Code for 29 must be sensible.

	1	2	3	4	5	6	7	8
1	BG	BE	BN	BM	BS	BN	BE	BG
2		BR	BR	BR	BR	BR	BR	BR
3	**BR**							
4								
5								
6	**WR**							
7	**WG**	WR	WR	WR	WR	WR	WR	WR
8		WE	WN	WM	WS	WN	WE	WG

Mark as follows:
Correct game position shown;
Correct message and value of 3 displayed;

2

(b) (i) `IF` statement with one correct condition;
`IF` statement with a second correct condition;
`IF` statement with all four correct conditions;
Value of `False` returned to calling routine correctly if a square is out of bounds and value of `False` is still returned for all the original checks for illegal moves and value `True` of is still returned for all legal moves;

A. two/four separate selection structures

Note: the four conditions are `FinishRank > 8`, `FinishRank < 1`, `FinishFile`

`< 1` and `FinishFile > 8` **A.** equivalent logic

Maximum of 3 marks if the code, when run, would still execute the line `IF Board[FinishRank][FinishFile][1] ← "W" THEN` in the `CheckMoveIsLegal` subroutine when an out-of-bounds finish square is entered

Pascal

```
...
Var
      PieceType : Char;
      PieceColour : String;
      MoveIsLegal : Boolean;
Begin
   MoveIsLegal := True;
   If (FinishFile = StartFile) And (FinishRank =
StartRank)
      Then MoveIsLegal := False
      Else
        If (FinishFile > 8) Or (FinishFile < 1) Or
(FinishRank > 8) Or (FinishRank < 1)
         Then MoveIsLegal := False
         Else
           Begin
             PieceType := Board[StartRank,
 StartFile][2];
             ...
```

VB.Net

```
...
Dim PieceType As String
Dim PieceColour As String
If FinishFile = StartFile And FinishRank = StartRank Then
   Return False
End If
If FinishFile > 8 Or FinishFile < 1 Or FinishRank > 8 Or
FinishRank < 1 Then
   Return False
End If
PieceType = Board(StartRank, StartFile)(1)
...
```

VB6

```
...
MoveIsLegal = True
If FinishFile = StartFile And FinishRank = StartRank Then
   MoveIsLegal = False
Else
   If FinishFile > 8 Or FinishFile < 1 Or FinishRank > 8
Or
FinishRank < 1 Then
      MoveIsLegal = False
   Else
      PieceType = Mid$(Board(StartRank, StartFile), 2, 1)
      ...
```

Java

```
...
char pieceType;
char pieceColour;
boolean moveIsLegal = true;
if ((finishFile == startFile) && (finishRank ==
```

```
startRank))
        {
           moveIsLegal = false;
        }
        if (finishFile > 8 || finishFile < 1 || finishRank > 8 ||
        finishRank < 1) {
           moveIsLegal = false;
        }
        pieceType = board[startRank][startFile].charAt(1);
        ...
```

C#

```
        ...
        char PieceType;
        char PieceColour;
        Boolean MoveIsLegal = true;
        if ((FinishFile == StartFile) && (FinishRank ==
StartRank))
           MoveIsLegal = false;
        if (FinishFile > 8 || FinishFile < 1 || FinishRank > 8 ||
        FinishRank < 1)
           MoveIsLegal = false;
        PieceType = Board[StartRank, StartFile][1];
        ...
```

Python 2

```
        def CheckMoveIsLegal(Board, StartRank, StartFile,
        FinishRank, FinishFile, WhoseTurn):
           MoveIsLegal = True
           if FinishFile > 8 or FinishFile < 1 or FinishRank > 8
  or
        FinishFile < 1:
              MoveIsLegal = False
           elif (FinishFile == StartFile) and (FinishRank ==
        StartRank):
              MoveIsLegal = False
              ...
```

Python 3

```
        def CheckMoveIsLegal(Board, StartRank, StartFile,
        FinishRank, FinishFile, WhoseTurn):
           MoveIsLegal = True
           if FinishFile > 8 or FinishFile < 1 or FinishRank > 8
  or
        FinishFile < 1:
              MoveIsLegal = False
           elif (FinishFile == StartFile) and (FinishRank ==
        StartRank):
              MoveIsLegal = False
              ...
```


4

(ii) *****SCREEN CAPTURE*****

Must match code from 31, including prompts on screen capture matching those in code. Code for 31 must be sensible.

Finish square of 98 followed by message saying `That is not a legal move - please try again;`

R. if the code, when run, would still execute the line `IF Board[FinishRank][FinishFile][1] ← "W" THEN` in the `CheckMoveIsLegal` subroutine when an out-of-bounds finish square is entered

Finish square of 19 followed by message saying `That is not a legal move - please try again;`
R. if the code, when run, would still execute the line `IF Board[FinishRank][FinishFile][1] ← "W" THEN` in the `CheckMoveIsLegal` subroutine when an out-of-bounds finish square is entered

Finish square of 86 followed by board position below being displayed; **A.** value entered for finish square not displayed as long as correct board state is shown
R. if no code in 31 that checks for `FinishFile` being valid

1	BG	BE	BN	BM	BS	BN	BE	BG
2	BR	BR	BR	BR	BR	BR	BR	BR
3								
4								
5								
6								WR
7	WR	WR	WR	WR	WR	WR	WR	
8	WG	WE	WN	WM	WS	WN	WE	WG
	1	2	3	4	5	6	7	8

3

(c) (i) option for `K` added to case/if statement with call to `CheckSarrumMoveIsLegal`; **R.** if any symbol other than `K` used

A. `CheckKashshaptuMoveIsLegal` (or similar) if evidence of creating new subroutine (or renaming existing subroutine) included somewhere in answer to question (c)

Pascal

```
...
'S', 'K' : MoveIsLegal := CheckSarrumMoveIsLegal(Board,
StartRank, StartFile, FinishRank, FinishFile);
...
```

VB.Net

```
...
Case "S", "K"
  Return CheckSarrumMoveIsLegal(Board, StartRank,
StartFile, FinishRank, FinishFile)
...
```

VB6

```
...
Case "S", "K"
   MoveIsLegal = CheckSarrumMoveIsLegal(Board,
StartRank,
```

```
    StartFile, FinishRank, FinishFile)
    ...
```

Java

```
    ...
    Case 'S' :
    Case 'K' :
    ...
```

C#

```
    ...
    Case 'S' :
    Case 'K' :
    ...
```

Python 2

```
    if MoveIsLegal == True:
       if PieceType == "R":
          MoveIsLegal = CheckRedumMoveIsLegal(Board,
 StartRank,
    StartFile, FinishRank, FinishFile, PieceColour)
       elif PieceType == "S" or PieceType == "K":
          MoveIsLegal = CheckSarrumMoveIsLegal(Board,
 StartRank,
    StartFile, FinishRank, FinishFile)
       elif PieceType == "M":
       ...
```

Python 3

```
    if MoveIsLegal == True:
       if PieceType == "R":
          MoveIsLegal = CheckRedumMoveIsLegal(Board,
 StartRank,
    StartFile, FinishRank, FinishFile, PieceColour)
       elif PieceType == "S" or PieceType == "K":
          MoveIsLegal = CheckSarrumMoveIsLegal(Board,
 StartRank,
    StartFile, FinishRank, FinishFile)
       elif PieceType == "M":
       ...
```


1

(ii) 1. White redum reaching 1st rank has symbol changed to `K` instead of `M;`

2. Added a selection structure with one of the following correct conditions:
- checks for the piece in the start square being a `K`
- checks for finish square not being empty // checks for finish square containing a black piece;

3. The additional selection structure has all necessary correct conditions and the correct logic;

4. Statement that changes the colour of a black piece in the finish square if it has been 'captured' by the kashshaptu – must be inside the selection structure;

5. When a kashshaptu moves and the finish square did not contain a black piece the contents of the start square become `"  "` and if the finish square did contain a black piece the contents stay as `"WK";`

Pascal

```
    ...
```

```
If (WhoseTurn = 'W') And (FinishRank = 1) And (Board[
StartRank, StartFile][2] = 'R')
   Then
      Begin
         Board[FinishRank, FinishFile] := 'WK';
         Board[StartRank, StartFile] := '   ';
      End
   Else
      Begin
         If (Board[StartRank, StartFile][2] = 'K') And
(Board[FinishRank, FinishFile] <> '   ')
          Then Board[FinishRank, FinishFile] :=
               Board[StartRank, StartFile][1] +
               Board[FinishRank, FinishFile][2]
          Else
            If (WhoseTurn = 'B') And (FinishRank = 8) And
             (Board[StartRank, StartFile][2] = 'R')
              Then
                Begin
                  Board[FinishRank, FinishFile] := 'BM';
                  Board[StartRank, StartFile] := ' ';
...
```

Alternative answer

```
...
If (WhoseTurn = 'W') And (FinishRank = 1) And (Board[
StartRank, StartFile][2] = 'R')
   Then
      Begin
         Board[FinishRank, FinishFile] := 'WK';
         Board[StartRank, StartFile] := '   ';
      End
   Else
      Begin
         If (Board[StartRank, StartFile][2] = 'K') And
(Board[FinishRank, FinishFile] <> '   ')
          Then Board[FinishRank, FinishFile] :=
               'W' + Board[FinishRank, FinishFile][2]
          Else
            If (WhoseTurn = 'B') And (FinishRank = 8) And
             (Board[StartRank, StartFile][2] = 'R')
              Then
                Begin
                  Board[FinishRank, FinishFile] := 'BM';
                  Board[StartRank, StartFile] := ' ';
...
```


VB.Net

```
...
If WhoseTurn = "W" And FinishRank = 1 And Board(StartRank,
StartFile)(1) = "R" Then
   Board(FinishRank, FinishFile) = "WK"
   Board(StartRank, StartFile) = "   "
ElseIf Board(StartRank, StartFile)(1) = "K" And
Board(FinishRank, FinishFile) <> " " Then
   Board(FinishRank, FinishFile) = Board(StartRank,
StartFile)(0) & Board(FinishRank, FinishFile)(1)
ElseIf WhoseTurn = "B" And FinishRank = 8 And
Board(StartRank, StartFile)(1) = "R" Then
   Board(FinishRank, FinishFile) = "BM"
   Board(StartRank, StartFile) = "
Else
   Board(FinishRank, FinishFile) = Board(StartRank,
```

```
StartFile)
   Board(StartRank, StartFile) = "   "
End If
...
```

Alternative answer

```
...
If WhoseTurn = "W" And FinishRank = 1 And Board(StartRank,
StartFile)(1) = "R" Then
   Board(FinishRank, FinishFile) = "WK"
   Board(StartRank, StartFile) = "   "
ElseIf Board(StartRank, StartFile)(1) = "K" And
Board(FinishRank, FinishFile) <> " " Then
   Board(FinishRank, FinishFile) = "W" &
Board(FinishRank,
FinishFile)(1)
ElseIf WhoseTurn = "B" And FinishRank = 8 And
Board(StartRank, StartFile)(1) = "R" Then
   Board(FinishRank, FinishFile) = "BM"
   Board(StartRank, StartFile) = "
Else
   Board(FinishRank, FinishFile) = Board(StartRank,
StartFile)
   Board(StartRank, StartFile) = "   "
End If
...
```

VB6

```
...
If WhoseTurn = "W" And FinishRank = 1 And
Mid$(Board(StartRank, StartFile), 2, 1) = "R" Then
   Board(FinishRank, FinishFile) = "WK"
   Board(StartRank, StartFile) = "   "
ElseIf Mid$(Board(StartRank, StartFile), 2, 1) = "K" And
Board(FinishRank, FinishFile) <> "   " Then
   Board(FinishRank, FinishFile) = Mid$(Board(StartRank,
StartFile), 1, 1) & Mid$(Board(FinishRank, FinishFile),
2,
1)
ElseIf WhoseTurn = "B" And FinishRank = 8 And
Mid$(Board(StartRank, StartFile), 2, 1) = "R" Then
   Board(FinishRank, FinishFile) = "BM"
   Board(StartRank, StartFile) = "   "
Else
   Board(FinishRank, FinishFile) = Board(StartRank,
StartFile)
   Board(StartRank, StartFile) = "   "
End If
...
```

Alternative answer

```
...
If WhoseTurn = "W" And FinishRank = 1 And
Mid$(Board(StartRank, StartFile), 2, 1) = "R" Then
   Board(FinishRank, FinishFile) = "WK"
   Board(StartRank, StartFile) = "   "
ElseIf Mid$(Board(StartRank, StartFile), 2, 1) = "K" And
Board(FinishRank, FinishFile) <> "   " Then
   Board(FinishRank, FinishFile) = "W" &
Mid$(Board(FinishRank, FinishFile), 2, 1)
ElseIf WhoseTurn = "B" And FinishRank = 8 And
Mid$(Board(StartRank, StartFile), 2, 1) = "R" Then
   Board(FinishRank, FinishFile) = "BM"
```

```
   Board(StartRank, StartFile) = "   "
Else
   Board(FinishRank, FinishFile) = Board(StartRank,
StartFile)
   Board(StartRank, StartFile) = "   "
End If
...
```

Java

```
...
if ((whoseTurn == 'W') && (finishRank == 1) &&
 (board[startRank][startFile].charAt(1) == 'R')) {
   board[finishRank][finishFile] = "WK";
   board[startRank][startFile] = "   ";
} else {
   if (board[startRank][startFile].charAt(1) == 'K'
&& !board[finishRank][finishFile].equals("   ")) {
      Board[finishRank][finishFile] =
Character.toString(board[startRank][startFile].charAt(
0)) +
Character.toString(board[finishRank][finishFile].charA
t(1))
;
   } else {
      if ((whoseTurn == 'B') && (finishRank == 8) &&
(board[startRank][startFile].charAt(1) == 'R')) {
         board[finishRank][finishFile] = "BM";
         board[startRank][startFile] = "   ";
      } else {
         board[finishRank][finishFile] =
board[startRank][startFile];
         board[StartRank][startFile] = "   ";
      }
   }
}
...
```

Alternative Solution

```
...
if ((whoseTurn == 'W') && (finishRank == 1) &&
 (board[startRank][startFile].charAt(1) == 'R')) {
   board[finishRank][finishFile] = "WK";
   board[startRank][startFile] = "   ";
} else {
   if (board[startRank][startFile].charAt(1) == 'K' &&
!board[finishRank][finishFile].equals("   ")) {
      board[finishRank][finishFile] = "W" +
Character.toString(board[finishRank][finishFile].charA
t(1))
;
   } else {
      if ((whoseTurn == 'B') && (finishRank == 8) &&
(board[startRank][startFile].charAt(1) == 'R')) {
         board[finishRank][finishFile] = "BM";
         board[startRank][startFile] = "   ";
      } else {
         board[finishRank][finishFile] =
board[startRank][startFile];
         board[StartRank][startFile] = "   ";
      }
   }
}
...
```

C#

```
...
if ((WhoseTurn == 'W') && (FinishRank == 1) &&
(Board[StartRank, StartFile][1] == 'R'))
{
   Board[FinishRank, FinishFile] = "WK";
   Board[StartRank, StartFile] = "   ";
}
else
   if (Board[StartRank, StartFile][1] == 'K' &&
Board[FinishRank, FinishFile] != "   ")
      Board[FinishRank, Finishfile] = Board[StartRank,
StartFile][0].ToString + Board[FinishRank,
Finishfile][1].ToString();
   else
      if ((WhoseTurn == 'B') && (FinishRank == 8) &&
(Board[StartRank, StartFile][1] == 'R'))
      {
        Board[FinishRank, FinishFile] = "BM";
        Board[StartRank, StartFile] = "   ";;
      }
      else
      {
        Board[FinishRank, FinishFile] = Board[StartRank,
StartFile];
        Board[StartRank, StartFile] = "   ";;
      }
...
```

Alternative Solution

```
...
if ((WhoseTurn == 'W') && (FinishRank == 1) && (Board[
StartRank, StartFile][1] == 'R'))
{
   Board[FinishRank, FinishFile] = "WK";
   Board[StartRank, StartFile] = "   ";
}
else
   if (Board[StartRank, StartFile][1] == 'K' &&
Board[FinishRank, finishFile] != "   ")
      Board[FinishRank, FinishFile] = "W" +
Board[finishRank,
FinishFile][1].ToString();
   else
      if ((WhoseTurn == 'B') && (FinishRank == 8) &&
(Board[StartRank, StartFile][1] == 'R'))
      {
        Board[FinishRank, FinishFile] = "BM";
        Board[StartRank, StartFile] = "   ";;
      }
      else
      {
        Board[FinishRank, FinishFile] = Board[StartRank,
StartFile];
        Board[StartRank, StartFile] = "   ";
      }
...
```

Python 2

```
if (WhoseTurn == "W") and (FinishRank == 1) and
(Board[StartRank][StartFile][1] == "R"):
   Board[FinishRank][FinishFile] = "WK"
   Board[StartRank][StartFile] = "   "
```

```
elif Board[StartRank][StartFile][1] == "K" and
Board[FinishRank][FinishFile] != "   ":
   Board[FinishRank][FinishFile] =
Board[StartRank][StartFile][0] +
Board[FinishRank][FinishFile][1]
elif (WhoseTurn == "B") and (FinishRank == 8) and
 (Board[StartRank][StartFile][1] == "R"):
   Board[FinishRank][FinishFile] = "BM"
   Board[StartRank][StartFile] = "   "
else:
   Board[FinishRank][FinishFile] =
Board[StartRank][StartFile]
   Board[StartRank][StartFile] = "   "
   ...
```

Alternative answer

```
if (WhoseTurn == "W") and (FinishRank == 1) and
 (Board[StartRank][StartFile][1] == "R"):
   Board[FinishRank][FinishFile] = "WK"
   Board[StartRank][StartFile] = "   "
elif Board[StartRank][StartFile][1] == "K" and
Board[FinishRank][FinishFile] != "   ":
   Board[FinishRank][FinishFile] = "W" +
Board[FinishRank][FinishFile][1]
elif (WhoseTurn == "B") and (FinishRank == 8) and
 (Board[StartRank][StartFile][1] == "R"):
   Board[FinishRank][FinishFile] = "BM"
   Board[StartRank][StartFile] = "   "
else:
   Board[FinishRank][FinishFile] =
Board[StartRank][StartFile]
   Board[StartRank][StartFile] = "   "
   ...
```

Python 3

```
if (WhoseTurn == "W") and (FinishRank == 1) and
 (Board[StartRank][StartFile][1] == "R"):
   Board[FinishRank][FinishFile] = "WK"
   Board[StartRank][StartFile] = "   "
elif Board[StartRank][StartFile][1] == "K" and
Board[FinishRank][FinishFile] != "   ":
   Board[FinishRank][FinishFile] =
Board[StartRank][StartFile][0] +
Board[FinishRank][FinishFile][1]
elif (WhoseTurn == "B") and (FinishRank == 8) and
 (Board[StartRank][StartFile][1] == "R"):
   Board[FinishRank][FinishFile] = "BM"
   Board[StartRank][StartFile] = "   "
else:
   Board[FinishRank][FinishFile] =
Board[StartRank][StartFile]
   Board[StartRank][StartFile] = "   "
   ...
```

Alternative answer

```
if (WhoseTurn == "W") and (FinishRank == 1) and
 (Board[StartRank][StartFile][1] == "R"):
   Board[FinishRank][FinishFile] = "WK"
   Board[StartRank][StartFile] = "   "
elif Board[StartRank][StartFile][1] == "K" and
Board[FinishRank][FinishFile] != "   ":
   Board[FinishRank][FinishFile] = "W" +
Board[FinishRank][FinishFile][1]
```

```
elif (WhoseTurn == "B") and (FinishRank == 8) and
 (Board[StartRank][StartFile][1] == "R"):
   Board[FinishRank][FinishFile] = "BM"
   Board[StartRank][StartFile] = "   "
else:
   Board[FinishRank][FinishFile] =
Board[StartRank][StartFile]
   Board[StartRank][StartFile] = "   "
   ...
```

5

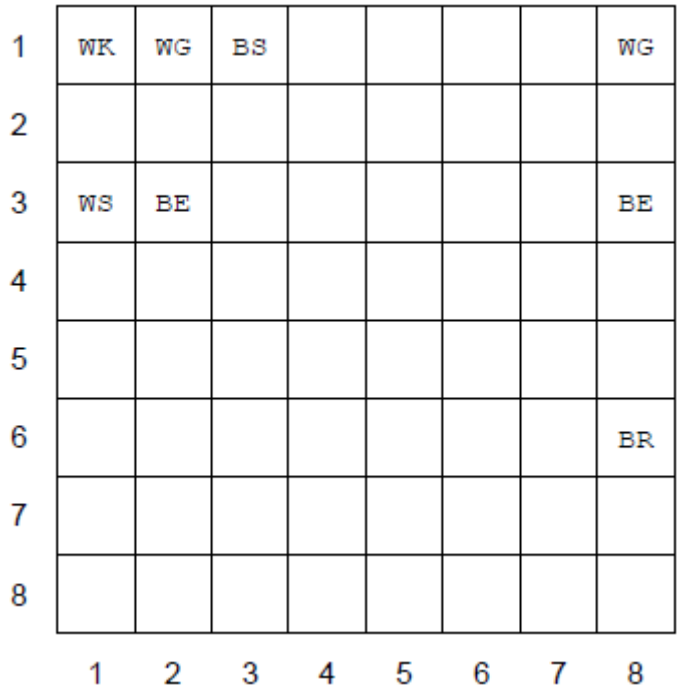
(iii) *****SCREEN CAPTURE*****

Must match code from 33 and 34, including prompts on screen capture matching those in code. Code for 33 and 34 must be sensible.

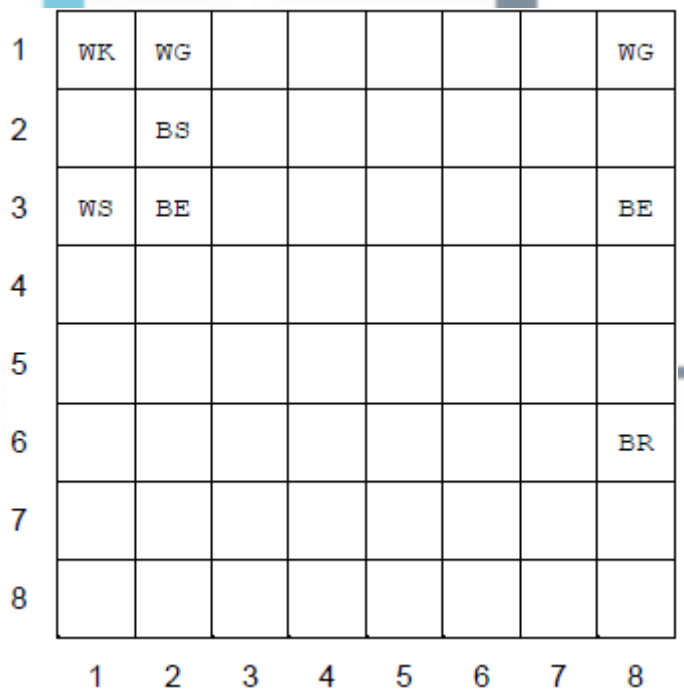
Finish square of 11 and board looks like diagram below;

1	WK	BG		BS				WG
2								
3	WS	BE						BE
4								
5								
6								BR
7								
8								
	1	2	3	4	5	6	7	8

3rd move finish square is 21 and board looks like the diagram below;



5th move has finish square of 22, board looks like diagram below and message `Black's sarrum has been captured. White wins!` is displayed;



3

(d) (i) 1. New subroutine `GenerateFEN` created; **R.** if spelt incorrectly **I.** case

2. Correct parameters passed into the subroutine;

3. A string value will be returned by the subroutine; **R.** use of global variable

4. FEN record will have a / at end of each rank;

5. FEN record uses upper case for white pieces;

6. FEN record uses lower case for black pieces;

7. Each piece on the board in the FEN record (even if incorrectly represented eg WR instead of R);

8. Indicates where the empty spaces on the board are in the FEN record (even if incorrect representation); **A.** Incorrect counts of empty spaces

9. FEN Record correctly use 8 for an empty rank; **R.** is any character other than 8 in the FEN record for an empty rank

10. Correctly counts number of consecutive empty spaces // correctly works out the number of consecutive empty spaces;

11. Count of empty spaces terminates at end of rank // calculation of number of empty spaces does not go over more than one rank; **R.** if no attempt to calculate/count number of empty spaces

12. FEN record shows whose move it is;

13. Accurate FEN record would be produced for every possible game state;

Pascal

```
Function GenerateFEN(Var Board : TBoard; WhoseTurn : Char)
: String;
   Var
      FEN : String;
      RankNo : Integer;
      FileNo : Integer;
      NoOfSpaces : Integer;
   Begin
      FEN := '';
      For RankNo := 1 To BoardDimension
         Do
           Begin
             NoOfSpaces := 0;
             For FileNo := 1 To BoardDimension
                Do
                  Begin
                    If Board[RankNo, FileNo] = '  '
                      Then NoOfSpaces := NoOfSpaces + 1
                      Else
                        Begin
                          If NoOfSpaces > 0
                            Then
                              Begin
                                FEN := FEN +
IntToStr(NoOfSpaces)Temp;
                                NoOfSpaces := 0;
                              End;
                          If Board[RankNo, FileNo][1] = 'B'
                            Then FEN := FEN +
Chr(Ord(Board[RankNo, FileNo][2]) + 32);
                            Else FEN := FEN + Board[RankNo,
FileNo][2]
                        End;
                End;
             If NoOfSpaces > 0
               Then FEN := FEN + IntToStr(NoOfSpaces);
             FEN := FEN + '/';
           End;
      FEN := FEN + WhoseTurn;
```

```
            GenerateFEN := FEN;
         End;
```

Alternative answer – converting to lower case

```
         Else FEN := FEN + ansilowercase(Board[RankNo,
FileNo][2]);
```

Alternative answer – using Str instead of IntToStr

```
         Function GenerateFEN(Var Board : TBoard; WhoseTurn : Char)
         : String;
            Var
               ...
               Temp : String;
            Begin
               ...
               Str(NoOfSpaces, Temp);
               FEN := FEN + Temp;
               ...
```

VB.Net

```
         Function GenerateFEN(ByVal Board(,) As String, ByVal
         WhoseTurn As Char) As String
            Dim FEN As String
            Dim RankNo As Integer
            Dim FileNo As Integer
            Dim NoOfSpaces As Integer
            FEN = ""
            For RankNo = 1 To BoardDimension
               NoOfSpaces = 0
               For FileNo = 1 To BoardDimension
                  If Board(RankNo, FileNo) = "   " Then
                     NoOfSpaces = NoOfSpaces + 1
                  Else
                     If NoOfSpaces > 0 Then
                        FEN = FEN & CStr(NoOfSpaces)
                        NoOfSpaces = 0
                     End If
                     If Board(RankNo, FileNo)(0) = "B" Then
                        FEN = FEN & Board(RankNo,
FileNo)(1).ToString.ToLower
                     Else
                        FEN = FEN & Board(RankNo, FileNo)(1)
                     End If
                  End If
               Next
               If NoOfSpaces > 0 Then
                  FEN = FEN & NoOfSpaces
               End If
               FEN = FEN & "/"
            Next
            FEN = FEN & WhoseTurn
            Return FEN
         End Function
```

VB6

```
         Private Function GenerateFEN(ByRef Board() As String,
ByVal
         WhoseTurn As String) As String
            Dim FEN As String
            Dim RankNo As Integer
            Dim FileNo As Integer
            Dim NoOfSpaces As Integer
            FEN = ""
```

```
    For RankNo = 1 To BoardDimension
      NoOfSpaces = 0
      For FileNo = 1 To BoardDimension
        If Board(RankNo, FileNo) = "   " Then
          NoOfSpaces = NoOfSpaces + 1
        Else
          If NoOfSpaces > 0 Then
            FEN = FEN & CStr(NoOfSpaces)
            NoOfSpaces = 0
          End If
          If Mid$(Board(RankNo, FileNo), 1, 1) = "B" Then
            FEN = FEN & LCase(Mid$(Board(RankNo, FileNo),
2, 1))
          Else
            FEN = FEN & Mid$(Board(RankNo, FileNo), 2, 1)
          End If
        End If
      Next
      If NoOfSpaces > 0 Then
        FEN = FEN & NoOfSpaces
      End If
      FEN = FEN & "/"
    Next
    FEN = FEN & WhoseTurn
    GenerateFEN = FEN
  End Function
```

Java

```
  String generateFEN(String[][] board, char whoseTurn) {
    String FEN;
    int rankNo;
    int fileNo;
    int noOfSpaces;
    FEN = "";
    for (rankNo = 1; rankNo <= BOARD_DIMENSION; rankNo++)
{
      noOfSpaces = 0;
      for (fileNo = 1; fileNo <= BOARD_DIMENSION;
 fileNo++) {
        if (board[rankNo][fileNo].equals(" ")) {
          noOfSpaces = noOfSpaces + 1;
        } else {
          if (noOfSpaces > 0) {
            FEN = FEN + Integer.toString(noOfSpaces);
            noOfSpaces = 0;
          }
          if (board[rankNo][fileNo].charAt(0) == 'B') {
            FEN = FEN + Character.toString(board[rankNo][
  fileNo].charAt(1)).toLowerCase();
          } else {
            FEN = FEN + board[rankNo][fileNo].charAt(1);
          }
        }
      }
      if (noOfSpaces > 0) {
        FEN = FEN + Integer.toString(noOfSpaces);
      }
      FEN = FEN + "/";
    }
    FEN = FEN + Character.toString(whoseTurn);
    return FEN;
  }
```

C#

```
public static string GenerateFEN(string[,] Board, char
WhoseTurn)
{
   string FEN;
   int RankNo;
   int FileNo;
   int NoOfSpaces;
   FEN = "";
   for (RankNo = 1; RankNo <= BoardDimension; RankNo++)
   {
      NoOfSpaces = 0;
      for (FileNo = 1; FileNo <= BoardDimension; FileNo++)
        if (Board[RankNo, FileNo] == " ")
          NoOfSpaces = NoOfSpaces + 1;
        else
          if (NoOfSpaces > 0)
          {
            FEN = FEN + NoOfSpaces.ToString();
            NoOfSpaces = 0;
          }
          if (Board[RankNo, FileNo][0] == 'B')
            FEN = FEN + Board[RankNo,
FileNo][1].ToString().ToLower();
          else
            FEN = FEN + Board[RankNo, FileNo][1];
      if (NoOfSpaces > 0)
        FEN = FEN + NoOfSpaces.ToString();
      FEN = FEN + "/";
   }
   FEN = FEN + WhoseTurn.ToString();
   return FEN;
}
```

Python 2

```
def GenerateFEN(Board, WhoseTurn):
   FEN = ""
   for RankNo in range(1 , BOARDDIMENSION + 1):
      NoOfSpaces = 0
      for FileNo in range(1 , BOARDDIMENSION + 1):
        if Board[RankNo][FileNo] == " ":
          NoOfSpaces = NoOfSpaces + 1
          else:
            if NoOfSpaces > 0:
              FEN = FEN + str(NoOfSpaces)
              NoOfSpaces = 0
            if Board[RankNo][FileNo][0] == "B":
              FEN = FEN +
     Board[RankNo][FileNo][1].lower()
            else:
              FEN = FEN + Board[RankNo][FileNo][1]
        if NoOfSpaces > 0:
          FEN = FEN + str(NoOfSpaces)
        FEN = FEN + "/"
   FEN = FEN + WhoseTurn
   return FEN
```

Python 3

```
def GenerateFEN(Board, WhoseTurn):
   FEN = ""
   for RankNo in range(1 , BOARDDIMENSION + 1):
      NoOfSpaces = 0
      for FileNo in range(1 , BOARDDIMENSION + 1):
```

```
            if Board[RankNo][FileNo] == "   ":
              NoOfSpaces = NoOfSpaces + 1
              else:
                if NoOfSpaces > 0:
                  FEN = FEN + str(NoOfSpaces)
                  NoOfSpaces = 0
                if Board[RankNo][FileNo][0] == "B":
                  FEN = FEN +
      Board[RankNo][FileNo][1].lower()
                else:
                  FEN = FEN + Board[RankNo][FileNo][1]
            if NoOfSpaces > 0:
                FEN = FEN + str(NoOfSpaces)
            FEN = FEN + "/"
    FEN = FEN + WhoseTurn
    return FEN
```

13

(ii) Syntactically valid call to subroutine created in part 36;
Value returned by subroutine is displayed; **R.** use of global variable
Code for Task 2 added before the code asking the user to enter their move;

Pascal

```
...
DisplayBoard(Board);
Writeln(GenerateFEN(Board, WhoseTurn));
DisplayWhoseTurnItIs(WhoseTurn);
...
```

VB.Net

```
...
DisplayBoard(Board)
Console.WriteLine(GenerateFEN(Board, WhoseTurn))
DisplayWhoseTurnItIs(WhoseTurn)
MoveIsLegal = False
...
```


VB6

```
...
Call DisplayBoard(Board)
WriteLine (GenerateFEN(Board, WhoseTurn))
DisplayWhoseTurnItIs (WhoseTurn)
MoveIsLegal = False
...
```

Java

```
...
moveIsLegal = false;
displayBoard(board);
console.println(generateFEN(board, whoseTurn));
displayWhoseTurnItIs(whoseTurn);
...
```

C#

```
...
MoveIsLegal = false;
DisplayBoard(ref Board);
Console.WriteLine(GenerateFEN(Board, WhoseTurn));
DisplayWhoseTurnItIs(WhoseTurn);
...
```

Python 2

```
while not(GameOver):
   DisplayBoard(Board)
   print (GenerateFEN(Board, WhoseTurn))
   DisplayWhoseTurnItIs(WhoseTurn)
   MoveIsLegal = False
   while not(MoveIsLegal):
   ...
```

Python 3

```
while not(GameOver):
   DisplayBoard(Board)
   print (GenerateFEN(Board, WhoseTurn))
   DisplayWhoseTurnItIs(WhoseTurn)
   MoveIsLegal = False
   while not(MoveIsLegal):
   ...
```

3

(iii) *****SCREEN CAPTURE*****

Must match code from 36 and 37, including prompts on screen capture matching those in code. Code for 36 and 37 must be sensible.

Mark as follows:

Sample game chosen and the FEN record returned/created by their subroutine from part 36 is displayed;

Correct FEN record of `1g1s3G/R7/Se5e/8/8/7r/8/8/W` is displayed;

2

[40]

Q31.

(a) An abstraction / leaving out non-essential details // A representation of reality;

1

(b) **1 mark for how stack works:**

Stack / It is a Last-in-First-Out / LIFO / First-in-Last-Out / FILO (data structure);

1 mark for correspondence with siding (MAX 1):

The last wagon to enter will be the first to leave;

Wagons enter and leave from same end of siding;

Wagons cannot leave siding before wagons that have entered after them;

Note: Responses must refer to both entering and leaving to gain this mark

NE References to “start”, “end”, “front”, “back” of siding, without further clarification, as not clear which end of siding these terms refer to

NE A siding is LIFO − the student must refer to wagon in their answers, for example the last wagon to enter will be the first to leave.

2

(c)

```
If TopOfStackPointer = 0
Then
  Stack Empty Error
Else
  CurrentWagon ← StackArray [TopOfStackPointer]
  Decrement TopOfStackPointer
EndIf
```

1 mark for appropriate `If` structure including condition (does not need both `Then` and `Else`) − Do not award this mark if value is popped off stack outside of `If`.

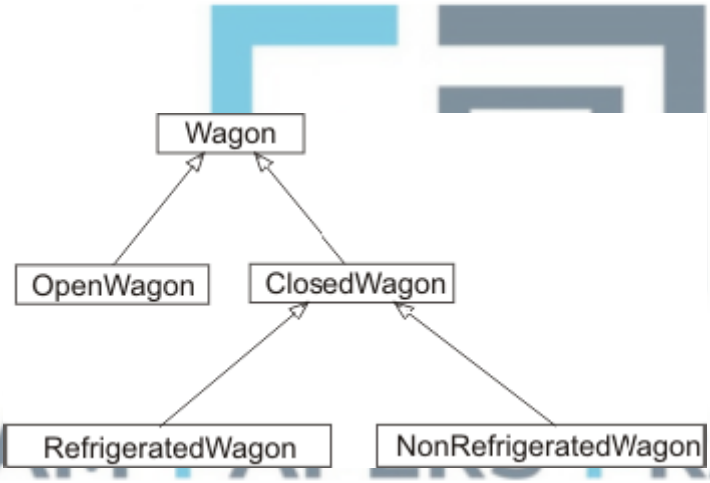
1 mark for reporting error in correct place

1 mark* for decrementing `TopOfStackPointer`
1 mark* for transferring value from correct position in array into `CurrentWagon` variable

* = if the `CurrentWagon` assignment is performed after the decrement instruction OR the `If` structure then award **MAX 1** of these two marks UNLESS the item is removed from position `TopOfStackPointer+1` so the code would work.
I unnecessary initialisation of any variables
A `Stack Is Empty` for `TopOfStackPointer = 0`
A Logic of `If` structure reversed i.e. `If stack is not empty / TopOfStackPointer>0 / 0 / !=0` and `Then, Else` swapped
A Any type of brackets or reasonable notation for the array index
A Award the mark for dealing with the error situation even if the condition in the `IF` statement is not correct, as long as the purpose of the condition is clearly correct
A Dealing with error in another sensible way eg by setting `CurrentWagon` to `Null`
A Additional lines of code that do not affect behaviour but **MAX 3** if these lines of code would stop the algorithm working correctly
DPT If candidate has used a different name for any variable then do not award first mark but award subsequent marks as if correct name used.

4

(d)



1 mark for Wagon at top of diagram with OpenWagon and ClosedWagon directly underneath it and linked to it and no other labels linked to it;
1 mark for ClosedWagon with RefrigeratedWagon and NonRefrigeratedWagon directly underneath it and linked to it, and no other labels linked to it (except Wagon above);
1 mark for correctly styled diagram, i.e. lines drawn as arrows and boxes (any shape) around labels; - ***This mark is only available if candidate has already achieved at least one mark for correct contents of the diagram.***

A Arrows drawn as:

A Filled / empty arrowheads
A Diagram rotated by 90 degrees

3

(e) `ClosedWagon = Class / Subclass / Extends Wagon` 1

```
(Public)
   Procedure CreateWagon (Override)          1
   Function GetHeight
   Function GetNumberOfDoors                 1
   Function GetSuitableForFoodStuffs
Private / Protected
   Height : Real
   NumberOfDoors : Integer                   1
   SuitableForFoodstuffs : Boolean
End
```

Accept answers that use different notations, so long as meaning is clear.

1 mark for correct header including name of class and parent class;
1 mark for redefining the `CreateWagon` procedure;
1 mark* for defining all 3 extra functions needed to read variable values, all identified as being public (keyword `public` is optional if functions are declared before variables);
1 mark# for defining all 3 extra variables, with appropriate data types and identified as being private;

A Any sensible numeric types for `Height` and `NumberOfDoors`. `Height` must accept non-integer values and `NumberOfDoors` integer values only.
A Answers that indicate separately that each variable is private or each method is public
R. Do not award mark for declaring new functions if any of the functions have the same name as the variables
I Parameters to methods, minor changes to names that do not affect clarity
* - Do not award this mark if any extra functions / procedures have been declared, EXCEPT for functions that would set values individually e.g. `SetHeight` or an incorrectly named procedure to add e.g. `CreateClosedWagon` which are acceptable for this mark
# - Do not award this mark if any extra variables have been declared

4

[14]

Q32.

(a)

<variable>	**Valid? (Tick any number of rows)**
a	✓
money-paid	
taxrate2	✓
2ndPlayerName	

1 mark for ticks in the correct two rows and other rows left blank.

A Use of alternative symbol for tick
A Use of two symbols - one to indicate validity and one to indicate invalidity, so long as the meaning of the symbols is clear e.g. a tick and a cross or a Y and an N.

1

(b) Required as an integer can contain any number of digits;
NE More than one digit
A "numbers" for "digits" as **BOD**
BNF does not support iteration / looping // BNF can only achieve iteration through recursion // would need infinite number of rules otherwise;
NE Rule needs to loop
MAX 1

1

(c) Variable may not have been declared;
Variable may be of inappropriate type;
Position of statement within program may be invalid;
Rightmost integer may be a lower value than the leftmost one;
One of the numbers / limits may be outside of the range of valid integers;
A Examples of any of the above
MAX 1

1

[3]

Q33.

(a) 182;

1

(b) -;74;

2

(c) -128; to (+)127;
Mark as follows:
Lowest value identified correctly;
Highest value identified correctly;

2

(d) 5 11 / 16 / /
5.6875;;
A 91 ÷ 16;;

Mark as follows:
Correct whole number part (5);
Correct fractional / decimal part (11 / 16 or 0.6875);

2

(e) B;6;

2

(f) Easier for people to read / understand;
R If implication is it easier for a computer to read / understand
Can be displayed using fewer digits;
More compact when printed / displayed;
NE Takes up less space
NE More compact

MAX 1

(g) Shift all the bits one place to the left; and add a zero / /
Add an extra 0; to the RHS of the bit pattern; / /
A Arithmetic left shift applied once / by one place;;

2

[12]

Q34.

(a) 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;

Independent of any programming language;
That can be completed in finite time;

MAX 2

(b)

		X
X	X	

Marks as follows:
1 mark for any two correct columns;
2 marks for all three columns correct;
A Other, sensible, indicators instead of X

2

(c)

x	c	b	a	Printed output
0	0	0	0	000
1	0	0	1	001
2	0	1	1	011
3	0	1	0	010
4	1	1	0	110
5	1	1	1	111
6	1	0	1	101
7	1	0	0	100

Mark as follows:
Any one row containing the correct values for **c**, **b** and **a**;
Any three rows containing the correct values for **c**, **b** and **a**;
All rows contain the correct values for **c**, **b** and **a**;
x column correct;
Printed output column correct; **A** printed output column incorrect – but matches the (incorrect) values provided for **c**, **b** and **a**, as long as a minimum of 3 rows have been completed
I Extra row at start of table containing the values `0,0,0,0,000`

5

(d) Print the (first 8) Gray code numbers; / /
(3 bit) Gray code counter;
NE Convert to Gray code

1

[10]

Q35.

(a) Correct variable declarations for `ISBN, CalculatedDigit` and `Count;`

`For` loop, with syntax allowed by the programming language, set up to repeat the correct number of times;
Correct prompt `"Please enter next digit of ISBN: ";`

Followed by `ISBN[Count]` assigned value entered by the user – must be inside the 1st iterative structure;

`CalculatedDigit` and `Count` initialised correctly (must be after 1st iterative structure and before 2nd iterative structure);

2nd loop has syntax allowed by the programming language and correct condition for the termination of the loop; **A** alternative correct logic for condition

`CalculatedDigit` assigned the value of its original value added to `ISBN[Count]` followed by incrementing `Count` – both inside the loop;

`CalculatedDigit` assigned the value of its original value added to `ISBN[Count] * 3` followed by incrementing `Count` – must be in the loop and after the 1st two assignment statements in the loop;

3rd loop has syntax allowed by the programming language and correct condition for the termination of the loop; **A** alternative correct logic for the condition

10 subtracted from value of `CalculatedDigit` and result assigned to `CalculatedDigit` – must be the only statement inside an iterative structure;

Assignment statement `CalculatedDigit ← 10 – CalculatedDigit` – must not be in an iteration or selection structure;

1st `IF` statement with correct condition – must not be in an iterative structure – with `CalculatedDigit` being assigned the value 0 inside the selection structure;

2nd `IF` statement with correct condition – must not be in an iterative structure or inside the 1st selection structure;

Correct output message (`Valid ISBN`) in `THEN` part of selection structure;

Correct output message (`Invalid ISBN`) in `ELSE` part of selection structure;

I Case of variable names and output messages
A Minor typos in variable names and output messages
I Spacing in prompts
A Initialisation of variables at declaration stage
A Arrays using positions 0 to 12 instead of 1 to 13

Pascal

```
Program Question4;
Var
  CalculatedDigit : Integer;
  ISBN : Array[1..13] Of Integer;
  Count : Integer;
Begin
```

```
  For Count := 1 To 13
    Do
      Begin
        Writeln('Please enter next digit of ISBN: ');
        Readln(ISBN[Count]);
      End;
  CalculatedDigit := 0;
  Count := 1;
  While Count < 13
    Do
      Begin
        CalculatedDigit := CalculatedDigit + ISBN[Count];
        Count := Count + 1;
        CalculatedDigit := CalculatedDigit + ISBN[Count] * 3;
        Count := Count + 1;
      End;
    While CalculatedDigit >= 10
      Do CalculatedDigit := CalculatedDigit - 10;
    CalculatedDigit := 10 - CalculatedDigit;
    If CalculatedDigit = 10
      Then CalculatedDigit := 0;
    If CalculatedDigit = ISBN[13]
      Then Writeln('Valid ISBN')
      Else Writeln('Invalid ISBN');
    Readln;
End.
```

VB.Net

```
Module Module1
  Sub Main()
    Dim CalculatedDigit As Integer
    Dim ISBN(13) As Integer
    Dim Count As Integer
    For Count = 1 To 13
      Console.Write("Please enter next digit of ISBN: ")
      ISBN(Count) = Console.ReadLine()
    Next
    CalculatedDigit = 0
    Count = 1
    While Count < 13
      CalculatedDigit = CalculatedDigit + ISBN(Count)
      Count = Count + 1
      CalculatedDigit = CalculatedDigit + ISBN(Count) * 3
      Count = Count + 1
    End While
    While CalculatedDigit >= 10
      CalculatedDigit = CalculatedDigit - 10
    End While
    CalculatedDigit = 10 - CalculatedDigit
    If CalculatedDigit = 10 Then
      CalculatedDigit = 0
    End If
    If CalculatedDigit = ISBN(13) Then
      Console.WriteLine("Valid ISBN")
    Else
      Console.WriteLine("Invalid ISBN")
    End If
    Console.ReadLine()
  End Sub
End Module
```

VB6

```
Private Sub Form_Load()
```

```
  Dim CalculatedDigit As Integer
  Dim ISBN(13) As Integer
  Dim Count As Integer
  For Count = 1 To 13
    ISBN(Count) = ReadLine("Please enter next digit of ISBN: ")
  Next
  CalculatedDigit = 0
  Count = 1
  While Count < 13
    CalculatedDigit = CalculatedDigit + ISBN(Count)
    Count = Count + 1
    CalculatedDigit = CalculatedDigit + (ISBN(Count) * 3)
    Count = Count + 1
  Wend
  While CalculatedDigit >= 10
    CalculatedDigit = CalculatedDigit - 10
  Wend
  CalculatedDigit = 10 - CalculatedDigit
  If CalculatedDigit = 10 Then
    CalculatedDigit = 0
  End If
  If CalculatedDigit = ISBN(13) Then
    WriteLine("Valid ISBN")
  Else
    WriteLine("Invalid ISBN")
  End If
End Sub
```

Alternative answers could use some of the following instead of WriteLine / ReadLine:

```
Console.Text = Console.Text & ...
WriteLineWithMsg
WriteWithMsg
Msgbox
InputBox
WriteNoLine
```

Python 2

```
# Question 4
if __name__ == "__main__":
  ISBN = [None, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
  for Count in range(1, 14):
    print 'Please enter next digit of ISBN: ',
    ISBN[Count] = int(raw_input())
  CalculatedDigit = 0
  Count = 1
  while Count < 13:
    CalculatedDigit = CalculatedDigit + ISBN[Count]
    Count = Count + 1
    CalculatedDigit = CalculatedDigit + ISBN[Count] * 3
    Count = Count + 1
  while CalculatedDigit >= 10:
    CalculatedDigit = CalculatedDigit - 10
  CalculatedDigit = 10 - CalculatedDigit
  if CalculatedDigit == 10:
    CalculatedDigit = 0
  if CalculatedDigit == ISBN[13]:
    print 'Valid ISBN'
  else:
    print 'Invalid ISBN'
```

Alternative print/input combination:

```
ISBN[Count] = int(raw_input('Please enter next digit of ISBN: ',))
```

Python 3

```
# Question 4
if __name__ == "__main__":
  ISBN = [None, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
for Count in range(1, 14):
    print('Please enter next digit of ISBN: '),
    ISBN[Count] = int(input())
  CalculatedDigit = 0
  Count = 1
  while Count < 13:
    CalculatedDigit = CalculatedDigit + ISBN[Count]
    Count = Count + 1
    CalculatedDigit = CalculatedDigit + ISBN[Count] * 3
    Count = Count + 1
  while CalculatedDigit >= 10:
    CalculatedDigit = CalculatedDigit - 10
  CalculatedDigit = 10 - CalculatedDigit
  if CalculatedDigit == 10 :
    CalculatedDigit = 0
  if CalculatedDigit == ISBN[13]:
    print('Valid ISBN')
  else:
    print('Invalid ISBN')
```

Alternative print/input combination:

```
ISBN[Count] = int(input('Please enter next digit of ISBN: ',))
```

Java

```
public class Question4 {
  AQAConsole2014 console = new AQAConsole2014();

  public Question4() {
    int ISBN[] = new int[14];
    int count;
    int calculatedDigit;
    for (count = 1; count <= 13; count++) {
      ISBN[count] = console.readInteger("Please enter next digit
of ISBN: ");
    }
    calculatedDigit = 0;
    count = 1;
    while (count < 13) {
      calculatedDigit = calculatedDigit + ISBN[count];
      count++;
      calculatedDigit = calculatedDigit + ISBN[count] * 3;
      count++;
    }
    while (calculatedDigit >= 10) {
      calculatedDigit = calculatedDigit - 10;
    }
    calculatedDigit = 10 - calculatedDigit;
    if (calculatedDigit == 10) {
      calculatedDigit = 0;
    }
    if (calculatedDigit == ISBN[13]) {
      console.println("Valid ISBN");
    } else {
      console.println("Invalid ISBN");
    }
  }
```

```
  public static void main(String[] args) {
    new Question4();
  }
}
```

15

(b) ****SCREEN CAPTURE****

Must match code from part (a), including prompts on screen capture matching those in code. Code for (a) must be sensible.

Mark as follows:

'Please enter next digit of ISBN: ' + user input of 9
'Please enter next digit of ISBN: ' + user input of 7
'Please enter next digit of ISBN: ' + user input of 8
'Please enter next digit of ISBN: ' + user input of 0
'Please enter next digit of ISBN: ' + user input of 0
'Please enter next digit of ISBN: ' + user input of 9
'Please enter next digit of ISBN: ' + user input of 9
'Please enter next digit of ISBN: ' + user input of 4
'Please enter next digit of ISBN: ' + user input of 1
'Please enter next digit of ISBN: ' + user input of 0
'Please enter next digit of ISBN: ' + user input of 6
'Please enter next digit of ISBN: ' + user input of 7
'Please enter next digit of ISBN: ' + user input of 6;
'Valid ISBN ' message shown;

A Alternative output messages if match code for part (a)
A If can only see some of the latter user inputs (e.g. due to first few inputs scrolling off the top of the console screen) – but must be able to see the last three digits entered (6, 7, 6)

2

(c) ****SCREEN CAPTURE****

Must match code from part (a), including prompts on screen capture matching those in code. Code for (a) must be sensible.

Mark as follows:

'Please enter next digit of ISBN: ' + user input of 9
'Please enter next digit of ISBN: ' + user input of 7
'Please enter next digit of ISBN: ' + user input of 8
'Please enter next digit of ISBN: ' + user input of 1
'Please enter next digit of ISBN: ' + user input of 8
'Please enter next digit of ISBN: ' + user input of 5
'Please enter next digit of ISBN: ' + user input of 7
'Please enter next digit of ISBN: ' + user input of 0
'Please enter next digit of ISBN: ' + user input of 2
'Please enter next digit of ISBN: ' + user input of 8
'Please enter next digit of ISBN: ' + user input of 8
'Please enter next digit of ISBN: ' + user input of 9
'Please enter next digit of ISBN: ' + user input of 4
'Invalid ISBN ' message shown;

A Alternative output messages if match code for part (a)
A If can only see some of the latter user inputs (e.g. due to first few inputs scrolling off the top of the console screen) – but must be able to see the last three digits entered (8, 9, 4)

1

[18]

Q1.

This question was about reverse Polish notation (RPN) and stacks. It was about the contents of a stack frame. Sometimes answers were too vague to be awarded a mark, “values” was a commonly-seen answer that was too imprecise to be creditworthy.

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 were able to get some marks on this programming question with about a third producing fully-working code. Some students wrote programs that only worked for a very limited selection of numbers but showed good exam technique by including their answer even though they knew it did not fully answer the question. A common error was to write the code in such a way that the number 1 was counted as being a prime number.

Q5.

An error on the paper meant that it was not possible for students using the Java programming language to provide an answer for question (a). All students (for all languages) were awarded this mark irrespective of what they wrote for their answer.

Question (b) asked students to identify a local variable in a method in the `QueueOfTiles` class. There were a number of potential correct answers with `Item` being the most commonly seen. Some students gave an example of a private attribute belonging to the class rather than a local variable in a method in the class.

Questions (c)-(e) were about circular and linear queues. Some students stated that a rear pointer would be needed for a circular queue which is true but does not answer question (e) as the rear pointer was already present in the Skeleton Program. Some answers for (c) talked, incorrectly, about circular queues being a dynamic data structure and linear queues as being a static data structure. Good answers for (d) made it clear that with the

queue only being very small in size the overhead of moving all items in the queue up one after deleting an item was negligible.

Most answers for question (f) and (g) showed some understanding of suitable approaches that could be taken but were rarely precise enough for full marks to be awarded. Some students gave answers for question (f) that changed the values of some of the tiles despite the question stating that this should not be done.

Q6.

(a) This was the first of the questions that required modifying the Skeleton Program. It was a simple question that over 80% of students were able to answer correctly. When mistakes were made this was normally because tiles other than just J and X were also changed to be worth 4 points.

(b) Like question (a), this question was normally well-answered with almost all student getting some marks and about 75% obtaining full marks. Where students didn't get full marks this was normally due to the conditions on the loop being incorrect which prevented the values of 1 and / or 20 from being valid.

(c) For this question students had to replace the linear search algorithm used to check if a word is in the list of allowed words with a binary search algorithm. An example of how a binary search algorithm works was included on the question paper but if a similar question is asked in the future that may not be done. A mixture of iterative and recursive solutions were seen. The most common error made by students who didn't get full marks but made a good attempt at answering the question was to miss out the condition that terminates the loop if it is now known that the word is **not** in the list.

(d) Students found question (d) easier than questions (c) and (e). Better answers made good use of iteration and arrays / lists, less efficient answers which used 26 variables to store the different letter counts could also get full marks. Some students added code in their new subroutine to read the contents of the text file rather than pass the list as a parameter to the subroutine; this was not necessary but was not penalised.

(e) Question (e) asked students to create a recursive subroutine. If students answered the question without using recursion they could still get 9 out of the 12 marks available.

It was disappointing that many students did not include any evidence of their attempt to answer the question. Good exam technique would be to include some program code that answers some part or parts of the question. For instance, in question (e) students could get marks for creating a subroutine with the specified name and calling that subroutine – even if the subroutine didn't do anything. There are many examples of subroutines and subroutine calls in the Skeleton Program that students could have used to help them obtain some marks on this question.

A number of very well-written subroutines were seen that made appropriate use of recursion and string handling. Some good recursive answers did not get full marks because they did not include a check that the word / prefix passed as a parameter was valid before the tile points included in the word were used to modify the score, this meant that all prefixes would be included in the score and not just the valid prefixes. Another frequent mistake came when students wrote their own code to calculate the score for a prefix rather than use the existing subroutine included in the Skeleton Program that calculated the score for a word – if done correctly full marks could be obtained by doing this but a number of students made mistakes when

writing their own score-calculating code.

Q7.

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.

Q8.

(a) This was, for most students, the easiest of the programming questions on the paper with about half obtaining full marks. Less confident programmers often had the wrong logic in their conditions (either getting `AND/OR` mixed-up or `</>`). Some students did not write code to get the validation condition to continually repeat until a valid value was entered. A significant minority of students did not add the validation routine to the `InputCoordinate` routine and instead tried to add it the constructor for the `Simulation` class.

Some students used recursion instead of iteration and full marks could be obtained from using this method if it was done correctly however many of these students did not return the value from the recursive call to the calling routine in a way that it could then be used by the rest of the program.

(b) The majority of students were able to get at least half the marks on this question and were clearly familiar with how to create a method that overrides a method in a base class in the programming language they were using. A significant minority of students did not attempt this question and had clearly not prepared for answering questions using OOP within the Skeleton Program.

A number of students did not identify the correct variable to use and wrote code that tried to change the default probability instead of the protected attribute inherited from the `Animal` class storing the probability for that animal.

Some students did not call the overridden method in the base class even though the question specified this should be done. The equivalent functionality could be obtained by copying the code in the `CalculateNewAge` method in the `Animal` class into the new `CalculateNewAge` method in the `Rabbit` class but this is poor programming practice as the original code would now be in two places in the program rather than reusing the existing code.

(c) One fifth of students did not provide any evidence of their attempt to answer this question. All students should be encouraged to include any program code they have written as it may be worth some marks even if it doesn't work correctly.

The most common mistake in reasonable attempts at the tasks in this question was to have the incorrect logic (for example, getting muddled between `AND/OR`) when writing the code to prevent a warren/fox being placed in a river.

(d) Many students came up with creative answers to this question that showed a high-level of programming and problem-solving skill. However, a large number of students did not include any evidence of their attempt at writing the program code. Some students showed good exam technique by including a very limited answer which they knew was nowhere near correct but would allow them to get some marks (most frequently for creating a new subroutine with the name specified in the question).

The most challenging part of the question was to make sure that the solution worked irrespective of the relative position of the fox and the warren with a number of solutions working if the fox was to the left of and above the warren but not if it was to the right of and below the warren.

Q12.

This question was about abstraction, object-oriented programming and linked lists.

For part (a) candidates had to explain how the LinkedList class was a form of abstraction. Many gave a definition of abstraction but failed to apply this to the LinkedList class and so did not achieve a mark. Good responses made clear that the LinkedList class was an example of abstraction because it allowed a programmer to manipulate items in a linked list without having to be concerned about how the linked list was implemented.

For part (b) candidates had to explain why the functions and procedures in the class were public whilst the data items were not. Many candidates were able to obtain a mark for the former, but few did so for the latter. Good responses made clear that the functions and procedures were public as they would need to be called from outside of the class to implement the game, and the data items were private so that their values could only be modified in a controlled way from outside of the class, by calling the procedures of the class. It was not sufficient to state that the data items were private because they were only used by the class or because they should not be changed.

Candidates had to write an algorithm for deleting an item from a linked list for part (c). A question was asked in a previous year about inserting an item into a linked list and the standard of responses to this question was notably better than was the case in the previous year. The majority of candidates had at least a good attempt at writing the part of the algorithm that would find the correct item to delete and many were then able to change the pointers to delete the item. Common mistakes and omissions were to fail to keep track of the pointer to the previous item when searching, to release the item to delete back to the heap before changing the pointer around it or to increase the current pointer by the fixed value of 1 on each iteration of a search loop. Few candidates scored all eight marks. If a candidate achieved seven but not eight marks this was usually because the algorithm did not take account of the fact that the item to delete might be the first item in the list, in which case the start pointer would need to be changed.

Q13.

This question was about the use of hashing.

In part (a) candidates had to compare the efficiency of searching a hash table with searching an unordered list. There were many good responses to this which explained that a slow linear search would be required for an unordered list but a fast calculation of a hash value is all that would be needed for the hash table implementation, and using this the location of the translation could be directly found.

For part (b) candidates had to explain what a collision was and how it could be dealt with. The majority of candidates appeared to understand both of these but some failed to achieve marks by not stating points explicitly. For example, too many candidates failed to explain the basic point that if two items hashed to the same value then they would be stored at the same location, and the second value would overwrite the first. Various sensible methods of dealing with a collision were well described.

Part (c) required candidates to explain why the English word had to be stored in addition to the French word. Some correctly identified that when performing English to French translation, if two English words had hashed to the same value, it would not be possible to

tell which the correct translation was unless the English word was stored. A small number of candidates incorrectly believed that the translation was being done in reverse (French to English) and explained that the hash function would be one-way, which whilst true was not a correct answer to the question that had been asked.

Q28.

Most students did well on this question, with nearly two-thirds getting 13 or more marks out of 15.

Some answers were seen where, as in previous years, students simply copied parts of the algorithm into their program code eg trying to use a keyword of `OUTPUT` or students using VB.Net adding the word `DO` to their `WHILE` loops. These were generally less able students who generally struggled on the Section D programming as well.

A common mistake that prevented students from getting full marks was to either miss out the code to increment the variable `Count2` or to place this line outside the `WHILE` loop.

Q29.

Answers to Section C 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 (parts (a) – (c), though there were fewer students this year who were doing this. 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 the lines of code that they have copied from the Skeleton Program).

Parts (d) – (f) were not well answered. Many students could find one error in the decision table for part (f) but few could find more than one. Answers to parts (d) and (e) were often vague with many students providing answers that were about different parts of the Skeleton Program from the ones asked for in these questions.

Q30.

This was a fairly straightforward programming question with most students getting good marks. Some students did not read the question carefully and added the line to increment `NoOfMoves` inside the loop that checked for an invalid move − this would result in `NoOfMoves` being incremented even if the move entered was illegal.

Q31.

For question part (a) students had to explain what a model was. Good responses explained that, in the context of simulation, a model was an abstracted representation of reality. Common mistakes were to explain what a physical model was and to confuse a model with a prototype.

For part (b) students had to explain why a queue was an appropriate data structure to represent a siding. Most students correctly explained that a stack was a first in last out structure, which was worth one mark. Fewer went on to successfully explain how this corresponded to the organisation of a siding. Students occasionally lost marks by using terms such as “in front of” in relation to the wagons, when it was not clear which end of a siding this related to.

For part (c) students had to write an algorithm for popping an item off a stack. A good range of responses was seen, with approximately half of students achieving at least two

marks and a quarter achieving all four marks. The error that had to be dealt with was a potentially empty stack. Appropriate methods of dealing with this included displaying an error message or returning a rogue value. Some students made the mistake of using the pop operation within the algorithm that was supposed to define it.

This question part (d), drawing an inheritance diagram, was very well answered, with almost all students getting two marks and over half achieving all three. The most common mistake was to represent the relationships between the classes correctly but to fail to style the diagram appropriately.

For part (e) students had to define a class. This was well answered, with over half of students achieving at least three of the four marks. It is clear that students' understanding of this topic has improved significantly over the last few years. The two most frequently made errors were to fail to express the relationship between the ClosedWagon and Wagon classes and to forget to override the CreateWagon procedure.

Q32.

This question was about the use of BNF to recognise language syntax. Slightly over half of students achieved the mark for part (a).

For part (b), just under half of students achieved the mark. Good responses recognised that an integer could contain an unlimited number of digits and that as BNF does not support iteration, recursion had to be used to achieve this. Responses that stated that more than one digit might be used were not enough for a mark as they did not make clear that the number of digits was unlimited.

For part (c) students had to explain why a For loop that met the BNF syntax definition might produce an error during compilation. Just under half of students achieved a mark for this, with good responses including that the number used for the first limit might be higher than the second, that the count variable might not have been declared or might be an inappropriate type, or that count might be a reserved word in the language.

Q33.

The topics covered by this question were generally well-understood. Most students were able to answer parts (a)-(b) and (e)-(f) well, though a number of students gave an answer of 74 instead of -74 as the answer for part (b). For part (c), most students were not able to state the correct range with the most common wrong answer being an upper limit of 128 (rather than 127). Many students did not read the question carefully for part 4 and assumed that four bits were being used before the binary point when the question said three bits before the binary point. A number of students also did not read the question carefully for part 7 and gave answers involving the use of binary addition.

Q34.

The definitions of algorithm were normally worth one mark, with only a few students going on to make a second creditworthy point. The decision table in part (b) was answered well, and most students were able to get some marks on part (c). Even when students had successfully completed part (c) they were often unable to work out what the purpose of the algorithm was – a number of students were clearly guessing with calculating prime numbers (an answer to a dry run question on a previous COMP1 exam), binary numbers and Hamming code being commonly-seen incorrect answers.

Q35.

Most students did well on this question, with well-over half getting 15 or more marks out of 18.

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 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 or students using VB.Net adding the word DO to their WHILE loops. These appeared to be less able students who generally struggled on the Section D programming as well.

Students who found this question difficult were often unable to create an array in the programming language they were using. **(1)**

(Total 18 marks)