Abstract Datatype

Papers Dock

COMPUTER SCIENCE 9618 PAPER 4

Question 1: Unordered Linked List (9618/41/MJ/21)

1 An unordered linked list uses a 1D array to store the data.

Each item in the linked list is of a record type, `node`, with a field `data` and a field `nextNode`.

The current contents of the linked list are:

startPointer	0
emptyList	5

Index	data	nextNode
0	1	1
1	5	4
2	6	7
3	7	-1
4	2	2
5	0	6
6	0	8
7	56	3
8	0	9
9	0	-1

(a) The following is pseudocode for the record type `node`.

```
TYPE node
    DECLARE data : INTEGER
    DECLARE nextNode : INTEGER
ENDTYPE
```

Write program code to declare the record type `node`.

Save your program as **question1**.

Copy and paste the program code into **part 1(a)** in the evidence document.

[2]

(b) Write program code for the main program.

Declare a 1D array of type `node` with the identifier `linkedList`, and initialise it with the data shown in the table on page 2. Declare the pointers.

Save your program.

Copy and paste the program code into **part 1(b)** in the evidence document.

[4]

(c) The procedure `outputNodes()` takes the array and `startPointer` as parameters. The procedure outputs the data from the linked list by following the `nextNode` values.

(i) Write program code for the procedure `outputNodes()`.

Save your program.

Copy and paste the program code into **part 1(c)(i)** in the evidence document.

[6]

(ii) Edit the main program to call the procedure `outputNodes()`.

Take a screenshot to show the output of the procedure `outputNodes()`.

Save your program.

Copy and paste the screenshot into **part 1(c)(ii)** in the evidence document.

[1]

(d) The function, `addNode()`, takes the linked list and pointers as parameters, then takes as input the data to be added to the end of the `linkedList`.

The function adds the node in the next available space, updates the pointers and returns `True`. If there are no empty nodes, it returns `False`.

(i) Write program code for the function `addNode()`.

Save your program.

Copy and paste the program code into **part 1(d)(i)** in the evidence document.

[7]

(ii) Edit the main program to:

- call `addNode()`
- output an appropriate message depending on the result returned from `addNode()`
- call `outputNodes()` twice; once before calling `addNode()` and once after calling `addNode()`.

Save your program.

Copy and paste the program code into **part 1(d)(ii)** in the evidence document.

[3]

(iii) Test your program by inputting the data value `5` and take a screenshot to show the output.

Save your program.

Copy and paste the screenshot into **part 1(d)(iii)** in the evidence document.

[1]

Question 2: Binary Tree Add Node (9618/41/ON/21)

3 An ordered binary tree stores integer data in ascending numerical order.

The data for the binary tree is stored in a 2D array with the following structure:

	LeftPointer	Data	RightPointer
Index	[0]	[1]	[2]
[0]	1	10	2
[1]	-1	5	-1
[2]	-1	16	-1

Each row in the table represents one node on the tree.
The number `-1` represents a null pointer.

(a) The 2D array, `ArrayNodes`, is declared with space for 20 nodes.

Each node has a left pointer, data and right pointer.

The program also initialises the:

- `RootPointer` to `-1` (null); this points to the first node in the binary tree
- `FreeNode` to `0`; this points to the first empty node in the array.

Write program code to declare `ArrayNodes`, `RootPointer` and `FreeNode` in the main program.

If you are writing in Python programming language, include attribute declarations using comments.

Save your program as **question 3**.

Copy and paste the program code into **part 3(a)** in the evidence document.

[4]

(b) The procedure `AddNode()` adds a new node to the array `ArrayNodes`.

The procedure needs to:

- take the array, root pointer and free node pointer as parameters
- ask the user to enter the data and read this in
- add the node to the root pointer if the tree is empty
- otherwise, follow the pointers to find the position for the data item to be added
- store the data in the location and update all pointers.

There are **six** incomplete statements in the following pseudocode for the procedure `AddNode()`.

```
PROCEDURE AddNode(BYREF ArrayNodes[] : ARRAY OF INTEGER,
                  BYREF RootPointer : INTEGER, BYREF FreeNode : INTEGER)
  OUTPUT "Enter the data"
  INPUT NodeData
  IF FreeNode <= 19 THEN
    ArrayNodes[FreeNode, 0] ← -1
    ArrayNodes[FreeNode, 1] ← ……
    ArrayNodes[FreeNode, 2] ← -1
    IF RootPointer = …… THEN
      RootPointer ← 0
    ELSE
      Placed ← FALSE
      CurrentNode ← RootPointer
      WHILE Placed = FALSE
        IF NodeData < ArrayNodes[CurrentNode, 1] THEN
          IF ArrayNodes[CurrentNode, 0] = -1 THEN
            ArrayNodes[CurrentNode, 0] ← ……
            Placed ← TRUE
          ELSE
            …… ← ArrayNodes[CurrentNode, 0]
          ENDIF
        ELSE
          IF ArrayNodes[CurrentNode, 2] = -1 THEN
            ArrayNodes[CurrentNode, 2] ← FreeNode
            Placed ← ……
          ELSE
            CurrentNode ← ArrayNodes[CurrentNode, 2]
          ENDIF
        ENDIF
      ENDWHILE
    ENDIF
    FreeNode ← …… + 1
  ELSE
    OUTPUT("Tree is full")
  ENDIF
ENDPROCEDURE
```


Write **program code** for the procedure `AddNode()`.

Save your program. Copy and paste the program code into **part 3(b)** in the evidence document.

[8]

(c) The procedure `PrintAll()` outputs the data in each element in `ArrayNodes`, in the order they are stored in the array.

Each element is printed in a row in the order:

```
LeftPointer  Data  RightPointer
```

For example:

```
 1        20       -1
-1        10       -1
```

Write program code for the procedure `PrintAll()`.

Save your program. Copy and paste the program code into **part 3(c)** in the evidence document.

[4]

(d) The main program should loop 10 times, each time calling the procedure `AddNode()`. It should then call the procedure `PrintAll()`.

(i) Edit the main program to perform the actions described.

Save your program. Copy and paste the program code into **part 3(d)(i)** in the evidence document.

[3]

(ii) Test the program by entering the data:

10
5
15
8
12
6
20
11
9
4

Take a screenshot to show the output after the given data are entered.

Copy and paste the screenshot into **part 3(d)(ii)** in the evidence document.

[1]

(e) An in-order tree traversal visits the left node, then the root (and outputs this), then visits the right node.

(i) Write a recursive procedure, `InOrder()`, to perform an in-order traversal on the tree held in `ArrayNodes`.

Save your program. Copy and paste the program code into **part 3(e)(i)** in the evidence document.

[7]

(ii) Test the procedure `InOrder()` with the same data entered in **part (d)(ii)**.

Take a screenshot to show the output after entering the data.

Copy and paste the screenshot into **part 3(e)(ii)** in the evidence document.

[1]

(e) An in-order tree traversal visits the left node, then the root (and outputs this), then visits the right node.

(i) Write a recursive procedure, `InOrder()`, to perform an in-order traversal on the tree held in `ArrayNodes`.

Save your program. Copy and paste the program code into **part 3(e)(i)** in the evidence document.

[7]

(ii) Test the procedure `InOrder()` with the same data entered in **part (d)(ii)**.

Take a screenshot to show the output after entering the data.

Copy and paste the screenshot into **part 3(e)(ii)** in the evidence document.

[1]

Question 3: Queue (9618/41/MJ/22)

3 A program uses a circular queue to store strings. The queue is created as a 1D array, `QueueArray`, with 10 string items.

The following data is stored about the queue:

- the head pointer initialised to 0
- the tail pointer initialised to 0
- the number of items in the queue initialised to 0.

(a) Declare the array, head pointer, tail pointer and number of items.

If you are writing in Python, include attribute declarations using comments.

Save your program as **Question3_J2022**. Copy and paste the program code into **part 3(a)** in the evidence document.

[2]

(b) The function `Enqueue` is written in pseudocode. The function adds `DataToAdd` to the queue. It returns `FALSE` if the queue is full and returns `TRUE` if the item is added.

The function is incomplete, there are **five** incomplete statements.

```
FUNCTION Enqueue(BYREF QueueArray[] : STRING, BYREF HeadPointer : INTEGER,
                 BYREF TailPointer : INTEGER, NumberItems : INTEGER,
                 DataToAdd : STRING) RETURNS BOOLEAN

  IF NumberItems = ..... THEN

    RETURN .....

  ENDIF

  QueueArray[.....] ← DataToAdd

  IF TailPointer >= 9 THEN

    TailPointer ← .....

  ELSE

    TailPointer ← TailPointer + 1

  ENDIF

  NumberItems ← NumberItems .....

  RETURN TRUE

ENDFUNCTION
```

Write program code for the function `Enqueue()`.

Save your program. Copy and paste the program code into **part 3(b)** in the evidence document.

[7]

(c) The function `Dequeue()` returns `"FALSE"` if the queue is empty, or it returns the next data item in the queue.

Write program code for the function `Dequeue()`.

Save your program. Copy and paste the program code into **part 3(c)** in the evidence document.

[6]

(d) **(i)** Amend the main program to:

- take as input 11 string values from the user
- use the `Enqueue()` function to add each element to the queue
- output an appropriate message to state whether each addition was successful, or not
- call `Dequeue()` function twice and output the return value each time.

Save your program. Copy and paste the program code into **part 3(d)(i)** in the evidence document.

[5]

(ii) Test your program with the input data:

```
"A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K"
```

Take a screenshot to show the output.

Copy and paste the screenshot into **part 3(d)(ii)** in the evidence document.

[1]

Question 4: Stack (9618/42/MJ/22)

1 A program needs to use a stack data structure. The stack can store up to 10 integer elements.

A 1D array `StackData` is used to store the stack globally. The global variable `StackPointer` points to the next available space in the stack and is initialised to 0.

(a) Write program code to declare the array and pointer as global data structures. Initialise the pointer to 0.

Save your program as **Question1_J22**. Copy and paste the program code into **part 1(a)** in the evidence document.

[3]

(b) Write a procedure to output all 10 elements in the stack **and** the value of `StackPointer`.

Save your program. Copy and paste the program code into **part 1(b)** in the evidence document.

[3]

(c) The function `Push()` takes an integer parameter and returns `FALSE` if the stack is full. If the stack is not full, it puts the parameter value on the stack, updates the relevant pointer and returns `TRUE`.

Write program code for the function `Push()`.

Save your program. Copy and paste the program code into **part 1(c)** in the evidence document.

[6]

(d) (i) Edit the main program to test the `Push()` function. The main program needs to:

- allow the user to enter 11 numbers and attempt to add these to the stack
- output an appropriate message when a number is added to the stack
- output an appropriate message when a number is not added to the stack if it is full
- output the contents of the stack after attempting to add all 11 numbers.

Save your program. Copy and paste the program code into **part 1(d)(i)** in the evidence document.

[5]

(ii) Test your program from **part 1(d)(i)** with the following 11 inputs:

11 12 13 14 15 16 17 18 19 20 21

Take a screenshot to show the output.

Copy and paste the screenshot into **part 1(d)(ii)** in the evidence document.

[1]

(e) The function `Pop()` returns `-1` if the stack is empty. If the stack is not empty, it returns the element at the top of the stack and updates the relevant pointer.

(i) Write program code for the function `Pop()`.

Save your program. Copy and paste the program code into **part 1(e)(i)** in the evidence document.

[5]

(ii) After the code you wrote in the main program for **part 1(d)(i)**, add program code to:

- remove two elements from the stack using `Pop()`
- output the updated contents of the stack.

Test your program and take a screenshot to show the output.

Copy and paste the screenshot into **part 1(e)(ii)** in the evidence document.

[2]

Question 5: Binary Tree (9618/41/ON/22)

3 A binary tree consists of nodes. Each node has 3 integer values: a left pointer, data and a right pointer.

The binary tree is stored using a global 2D array.

The pseudocode declaration for the array is:

```
DECLARE ArrayNodes : ARRAY[0:19, 0:2] OF INTEGER
```

For example:

- `ArrayNodes[0, 0]` stores the left pointer for the first node.
- `ArrayNodes[0, 1]` stores the data for the first node.
- `ArrayNodes[0, 2]` stores the right pointer for the first node.

`-1` indicates a null pointer, or null data.

(a) Write program code to:

- declare the global 2D array `ArrayNodes`
- initialise all 3 integer values to `-1` for each node.

Save your program as **Question3_N22**. Copy and paste the program code into **part 3(a)** in the evidence document.

[3]

(b) The binary tree stores the following values:

Index	Left pointer	Data	Right pointer
0	1	20	5
1	2	15	–1
2	–1	3	3
3	–1	9	4
4	–1	10	–1
5	–1	58	–1
6	–1	–1	–1

`FreeNode` stores the index of the first free element in the array, initialised to 6.

`RootPointer` stores the index of the first node in the tree, initialised to 0.

Amend your program by writing program code to store the given data in `ArrayNodes` **and** initialise the free node and root node pointers.

Save your program. Copy and paste the program code into **part 3(b)** in the evidence document.

[2]

(c) The following recursive pseudocode function searches the binary tree for a given value. If the value is found, the function must return the index of the value. If the value is not found, the function must return –1.

The function is incomplete. There are **four** incomplete statements.

```
FUNCTION SearchValue(Root : INTEGER,
                     ValueToFind : INTEGER) RETURNS INTEGER

   IF Root = -1 THEN

      RETURN -1

   ELSE

     IF ArrayNodes[Root, 1] = ValueToFind THEN

        RETURN .....

     ELSE

        IF ArrayNodes[Root, 1] = -1 THEN

           RETURN -1

        ENDIF

     ENDIF

   ENDIF

   IF ArrayNodes[Root, 1] ..... ValueToFind THEN

      RETURN SearchValue(ArrayNodes[....., 0], ValueToFind)

   ENDIF

   IF ArrayNodes[Root, .....] < ValueToFind THEN

      RETURN SearchValue(ArrayNodes[Root, 2], ValueToFind)

   ENDIF

ENDFUNCTION
```


Write program code for the function `SearchValue()`.

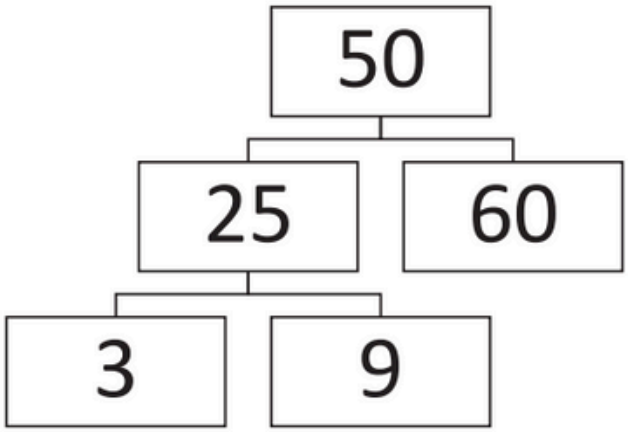
Save your program. Copy and paste the program code into **part 3(c)** in the evidence document.

[5]

(d) A post order traversal performs the following operation:

- visit the left node
- visit the right node
- output the root.

For example, in the following tree, the output would be: **3 9 25 60 50**



An outline of the `PostOrder()` procedure is:

- If left node is not empty, make a recursive call with the left node as the root.
- If right node is not empty, make a recursive call with the right node as the root.
- Output the current root node.

The procedure `PostOrder()` takes the root node as a parameter.

Write program code for the procedure `PostOrder()`.

Save your program. Copy and paste the program code into **part 3(d)** in the evidence document.

[7]

(e) **(i)** Amend the main program by writing program code to:

- call the function `SearchValue()` to find the position of the number 15 in the tree
- use the result from `SearchValue()` to output either the index of the value if found, or an appropriate message to state that the value was not found
- call the procedure `PostOrder()`.

Save your program. Copy and paste the program code into **part 3(e)(i)** in the evidence document.

[3]

(ii) Test your program.

Take a screenshot to show the output.

Copy and paste the screenshot into **part 3(e)(ii)** in the evidence document.

[1]

Question 6: Queue (9618/42/ON/22)

3 A program uses a linear queue to store up to 100 integers.

(a) A 1D array, `Queue`, is used to store the data. The head pointer points to the first number stored in the queue and the tail pointer points to the next free space in the queue.

Write program code to:

- declare the global array `Queue`
- declare the global variable head pointer and assign an appropriate initial value
- declare the global variable tail pointer and assign an appropriate initial value.

Save your program as **Question3_N22**. Copy and paste the program code into **part 3(a)** in the evidence document.

[3]

(b) The function `Enqueue()` takes an integer value as a parameter and stores it in the queue. It returns `TRUE` if the value was successfully stored and `FALSE` otherwise.

Write program code for the function `Enqueue()`.

Save your program. Copy and paste the program code into **part 3(b)** in the evidence document.

[6]

(c) The main program uses the `Enqueue()` function to store the numbers 1 to 20 (inclusive) in the queue, in ascending numerical order. The program should output 'Successful' if all numbers are successfully enqueued, and 'Unsuccessful' otherwise.

Amend the main program by writing program code to perform this task.

Save your program. Copy and paste the program code into **part 3(c)** in the evidence document.

[4]

(d) The following iterative pseudocode function calculates the total of all the values stored in the queue.

```
FUNCTION IterativeOutput(Start: INTEGER) RETURNS INTEGER

   DECLARE Total : INTEGER

   Total ← 0

   FOR Count ← Start - 1 TO HeadPointer STEP -1

      Total ← Total + Queue[Count]

   NEXT Count

   RETURN Total

ENDFUNCTION
```

Rewrite the function as a recursive function using program code.

Save your program. Copy and paste the program code into **part 3(d)** in the evidence document.

[6]

(e) The main program calls the recursive function from **part 3(d)** and outputs the value returned.

(i) Amend the main program by writing program code to perform this task.

Save your program. Copy and paste the program code into **part 3(e)(i)** in the evidence document.

[1]

(ii) Test your program.

Take a screenshot to show the output.

Copy and paste the screenshot into **part 3(e)(ii)** in the evidence document.

[1]

Question 7: Stacks (9618/41/MJ/23)

3 A program implements two stacks using 1D arrays. One stack stores the names of colours. One stack stores the names of animals.

(a) The program contains the following global arrays and variables:

- 1D array `Animal` to store the names of up to 20 animals.
- 1D array `Colour` to store the names of up to 10 colours.
- `AnimalTopPointer` to point to the next free space in the array `Animal`, initialised to 0.
- `ColourTopPointer` to point to the next free space in the array `Colour`, initialised to 0.

Write program code to declare the global arrays and variables.

Save your program as **Question3_J2023**. Copy and paste the program code into **part 3(a)** in the evidence document.

[3]

(b) **(i)** Study the pseudocode function `PushAnimal()`:

```
FUNCTION PushAnimal(DataToPush : STRING) RETURNS BOOLEAN
   IF AnimalTopPointer = 20 THEN
      RETURN FALSE
   ELSE
      Animal[AnimalTopPointer] ← DataToPush
      AnimalTopPointer ← AnimalTopPointer + 1
      RETURN TRUE
   ENDIF
ENDFUNCTION
```


Write program code for the function `PushAnimal()`

Save your program. Copy and paste the program code into **part 3(b)(i)** in the evidence document.

[3]

(ii) Study the pseudocode function `PopAnimal()`:

```
FUNCTION PopAnimal() RETURNS STRING
   DECLARE ReturnData : STRING
   IF AnimalTopPointer = 0 THEN
      RETURN ""
   ELSE
      ReturnData ← Animal[AnimalTopPointer - 1]
      AnimalTopPointer ← AnimalTopPointer - 1
      RETURN ReturnData
   ENDIF
ENDFUNCTION
```

Write program code to declare the function `PopAnimal()`

Save your program. Copy and paste the program code into **part 3(b)(ii)** in the evidence document.

[3]

(iii) The procedure `ReadData()`:

- reads the animal names from the file `AnimalData.txt`
- uses `PushAnimal()` to insert each name onto the stack
- uses appropriate exception handling if the file does not exist.

Write program code for the procedure `ReadData()`

Save your program. Copy and paste the program code into **part 3(b)(iii)** in the evidence document.

[5]

(iv) The function `PushColour()` performs the same actions as `PushAnimal()` but inserts an item into `Colour`.

The function `PopColour()` performs the same actions as `PopAnimal()` but removes the next item from `Colour`.

Write program code for the functions `PushColour()` and `PopColour()`

Save your program. Copy and paste the program code into **part 3(b)(iv)** in the evidence document.

[2]

(v) Amend the procedure `ReadData()` so that it also:

- reads the colours from the text file `ColourData.txt`
- uses `PushColour()` to insert each colour onto the stack
- uses appropriate exception handling if the file does not exist.

Save your program. Copy and paste the program code into **part 3(b)(v)** in the evidence document.

[2]

(c) The procedure `OutputItem()`:

- pops the next item from both `Animal` and `Colour`
- outputs the colour and animal on one line, for example `"black horse"`

If there is no data in `Colour`:

- the animal is pushed back onto `Animal`
- `"No colour"` is output.

If there is no data in `Animal`:

- the colour is pushed back onto `Colour`
- `"No animal"` is output.

Write program code for the procedure `OutputItem()`

Save your program. Copy and paste the program code into **part 3(c)** in the evidence document.

[5]

(d) The main program:

- calls the procedure `ReadData()`
- calls `OutputItem()` **four** times.

(i) Write program code for the main program.

Save your program. Copy and paste the program code into **part 3(d)(i)** in the evidence document.

[1]

(ii) Test your program.

Take a screenshot of the output.

Save your program. Copy and paste the screenshot into **part 3(d)(ii)** in the evidence document.

[1]

Question 8: Queue (9618/42/MJ/23)

2 A business sells a single product. Customers can purchase one or more of this product.

Each sale has an ID and a quantity, for example "ABC" and 2

The business needs a program to store the data about the sales in a circular queue data structure.

(a) Write program code to declare a record structure, `SaleData`, to store the data about each sale.

Save your program as **Question2_J2023**. Copy and paste the program code into **part 2(a)** in the evidence document.

[2]

(b) Write program code to:

- declare a global array, `CircularQueue`, of 5 items to store the sale records
- declare the global pointers `Head` and `Tail`
- declare the global variable `NumberOfItems`
- initialise all elements of the array `CircularQueue` to an empty record, where the ID is null (`""`) and quantity is `-1`
- initialise `Head`, `Tail` and `NumberOfItems` to `0`

Save your program. Copy and paste the program code into **part 2(b)** in the evidence document.

[4]

(c) The function `Enqueue()`:

- takes a new record as a parameter
- inserts the record in the circular queue at the element pointed to by `Tail`
- updates pointers and other variables as required
- returns `-1` if the circular queue is full
- returns `1` if the record is stored successfully.

Write program code for the function `Enqueue()`.

Save your program. Copy and paste the program code into **part 2(c)** in the evidence document.

[6]

(d) The function `Dequeue()`:

- returns a null or empty record if the circular queue is empty
- returns the first record in the queue if the circular queue is not empty
- updates pointers and other variables as required.

Write program code for the function `Dequeue()`.

Save your program. Copy and paste the program code into **part 2(d)** in the evidence document.

[6]

(e) The procedure `EnterRecord()`:

- takes an ID and quantity as input and creates a sale record
- uses `Enqueue()` to insert the record in the circular queue
- outputs `"Full"` if the record was not inserted in the circular queue
- outputs `"Stored"` if the record was inserted in the circular queue.

Write program code for the procedure `EnterRecord()`.

Save your program. Copy and paste the program code into **part 2(e)** in the evidence document.

[5]

(f) The following sale records need to be entered into the program:

ID	Quantity
ADF	10
OOP	1
BXW	5
XXZ	22
HQR	6
LLP	3

(i) Amend the main program to:

- use `EnterRecord()` to input the six records in the table
- use `Dequeue()` to remove one record
- output either the ID and quantity of the removed record, or an error message if the circular queue is empty
- use `EnterRecord()` to input the record with the ID "LLP" for a second time
- output the ID and quantity for all the records currently stored in the array `CircularQueue`.

Write program code to perform these tasks.

Save your program. Copy and paste the program code into **part 2(f)(i)** in the evidence document.

[4]

(ii) Test your program.

Take a screenshot of the output.

Save your program. Copy and paste the screenshot into **part 2(f)(ii)** in the evidence document.

[1]

Question 9: Queue (9618/41/ON/23)

2 A linear queue is implemented using the 1D array, `Queue`. The index of the first element in the array is 0.

(a) (i) Write program code to declare:

- `Queue` — a global array with space to store 50 IDs of type string
- `HeadPointer` — a global variable to point to the first element in the queue, initialised to −1
- `TailPointer` — a global variable to point to the next available space in the queue, initialised to 0.

Save your program as **Question2_N23**. Copy and paste the program code into part **2(a)(i)** in the evidence document.

[2]

(ii) The procedure `Enqueue()` takes a string parameter.

If the queue is full, the procedure outputs a suitable message. If the queue is not full, the procedure inserts the parameter into the queue and updates the relevant pointer(s).

Write program code for `Enqueue()`.

Save your program. Copy and paste the program code into part **2(a)(ii)** in the evidence document.

[4]

(iii) The function `Dequeue()` checks if the queue is empty.

If the queue is empty, the function outputs a suitable message and returns the string `"Empty"`.
If the queue is not empty, the function returns the first element in the queue and updates the relevant pointer(s).

Write program code for `Dequeue()`.

Save your program. Copy and paste the program code into part **2(a)(iii)** in the evidence document.

[4]

(b) A shop sells computer games. Each game has a unique identifier (ID) of string data type.

The text file `QueueData.txt` contains a list of game IDs.

The procedure `ReadData()` reads the data from the text file and inserts each item of data into the array `Queue`.

Write program code for the procedure `ReadData()`.

Save your program. Copy and paste the program code into part **2(b)** in the evidence document.

[6]

(c) Some game IDs appear in the text file more than once.

The program needs to total the number of times each game ID appears in the text file.

The record structure `RecordData` has the following fields:

- `ID` — a string to store the game ID
- `Total` — an integer to store the total number of times that game ID appears in the text file.

(i) Write program code to declare the record structure `RecordData`.

If you are writing in Python, include attribute declarations as comments.

Save your program. Copy and paste the program code into part **2(c)(i)** in the evidence document.

[2]

(ii) The global 1D array `Records` stores up to 50 items of type `RecordData`.

The global variable `NumberRecords` stores the number of records currently in the array `Records` and is initialised to 0.

Write program code to declare `Records` and `NumberRecords`.

If you are writing in Python, include attribute declarations as comments.

Save your program. Copy and paste the program code into part **2(c)(ii)** in the evidence document.

[2]

(iii) The pseudocode algorithm for the procedure `TotalData()`:

- uses `Dequeue()` to remove an ID from the queue
- checks whether a `RecordData` with the returned ID exists in `Records`
- increments the total for that ID in the record if the ID already exists
- creates a new record and stores it in `Records` if the ID does not exist.

```
PROCEDURE TotalData()
    DECLARE DataAccessed : STRING
    DECLARE Flag : BOOLEAN
    DataAccessed ← Dequeue()
    Flag ← FALSE
    IF NumberRecords = 0 THEN
        Records[NumberRecords].ID ← DataAccessed
        Records[NumberRecords].Total ← 1
        Flag ← TRUE
        NumberRecords ← NumberRecords + 1
     ELSE
        FOR X ← 0 TO NumberRecords - 1
            IF Records[X].ID = DataAccessed THEN
                Records[X].Total ← Records[X].Total + 1
                Flag ← TRUE
            ENDIF
        NEXT X
    ENDIF
    IF Flag = FALSE THEN
        Records[NumberRecords].ID ← DataAccessed
        Records[NumberRecords].Total ← 1
        NumberRecords ← NumberRecords + 1
    ENDIF
ENDPROCEDURE
```


Write program code for the procedure `TotalData()`.

Save your program. Copy and paste the program code into part **2(c)(iii)** in the evidence document.

[5]

(d) The procedure `OutputRecords()` outputs the ID and total of each record in `Records` in the format:

```
ID  1234   Total   4
```

Write program code for `OutputRecords()`.

Save your program. Copy and paste the program code into part **2(d)** in the evidence document.

[1]

(e) The main program needs to:

- call `ReadData()`
- call `TotalData()` for each element in the queue
- call `OutputRecords()`.

(i) Write program code for the main program.

Save your program. Copy and paste the program code into part **2(e)(i)** in the evidence document.

[2]

(ii) Test your program.

Take a screenshot of the output.

Save your program. Copy and paste the screenshot into part **2(e)(ii)** in the evidence document.

[1]

Question 10: Stack (9618/42/ON/23)

1 A program stores lower-case letters in two stacks.

One stack stores vowels (a, e, i, o, u) and one stack stores consonants (letters that are not vowels).

Each stack is implemented as a 1D array.

(a) (i) Write program code to declare two 1D global arrays: `StackVowel` and `StackConsonant`.

Each array needs to store up to 100 letters. The index of the first element in each array is 0.

If you are writing in Python, include declarations using comments.

Save your program as **Question1_N23**. Copy and paste the program code into part **1(a)(i)** in the evidence document.

[2]

(ii) The global variable `VowelTop` is a pointer that stores the index of the next free space in `StackVowel`.

The global variable `ConsonantTop` is a pointer that stores the index of the next free space in `StackConsonant`.

`VowelTop` and `ConsonantTop` are both initialised to 0.

Write program code to declare and initialise the two variables.

If you are writing in Python, include declarations using comments.

Save your program. Copy and paste the program code into part **1(a)(ii)** in the evidence document.

[1]

(b) (i) The procedure `PushData()` takes one letter as a parameter.

If the parameter is a vowel, it is pushed onto `StackVowel` and the relevant pointer updated.

If the stack is full, a suitable message is output.

If the parameter is a consonant, it is pushed onto `StackConsonant` and the relevant pointer updated.

If the stack is full, a suitable message is output.

You do **not** need to validate that the parameter is a letter.

Write program code for `PushData()`.

Save your program. Copy and paste the program code into part **1(b)(i)** in the evidence document.

[6]

(ii) The file `StackData.txt` stores 100 lower-case letters.

The procedure `ReadData()` reads each letter from the file and uses `PushData()` to push each letter onto its appropriate stack.

Use appropriate exception handling if the file does not exist.

Write program code for `ReadData()`.

Save your program. Copy and paste the program code into part **1(b)(ii)** in the evidence document.

[6]

(c) The function `PopVowel()` removes and returns the data at the top of `StackVowel` and updates the relevant pointer(s).

The function `PopConsonant()` removes and returns the data from the top of `StackConsonant` and updates the relevant pointer(s).

If either stack is empty, the string `"No data"` must be returned.

Write program code to declare `PopVowel()` and `PopConsonant()`.

Save your program. Copy and paste the program code into part **1(c)** in the evidence document.

[5]

(d) The program first needs to call `ReadData()` and then:

1. prompt the user to input their choice of vowel or consonant
2. take, as input, the user's choice
3. depending on the user's choice, call `PopVowel()` or `PopConsonant()` and store the return value.

The three steps are repeated until 5 letters have been successfully returned and stored.

If either stack is empty at any stage, an appropriate message must be output.

Once 5 letters have been successfully returned and stored, they are output on one line, for example:

```
abyti
```

(i) Write program code for the main program.

Save your program. Copy and paste the program code into part **1(d)(i)** in the evidence document.

[6]

(ii) Test your program with the following inputs:

```
vowel

consonant

consonant

vowel

vowel
```

Take a screenshot of the output.

Save your program. Copy and paste the screenshot into part **1(d)(ii)** in the evidence document.

[1]

Question 11: Queue (9618/41/MJ/24)

3 A program reads data from the user and stores the data that is valid in a linear queue.

The queue is stored as a global 1D array, `QueueData`, of string values. The array needs space for 20 elements.

The global variable `QueueHead` stores the index of the first element in the queue.
The global variable `QueueTail` stores the index of the last element in the queue.

(a) The main program initialises all the elements in `QueueData` to a suitable null value, `QueueHead` to −1 and `QueueTail` to −1.

Write program code for the main program.

Save your program as **Question3_J24**. Copy and paste the program code into part **3(a)** in the evidence document.

[1]

(b) The function `Enqueue()` takes the data to insert into the queue as a parameter.

If the queue is **not** full, it inserts the parameter in the queue, updates the appropriate pointer(s) and returns `TRUE`. If the queue is full, it returns `FALSE`.

Write program code for `Enqueue()`.

Save your program. Copy and paste the program code into part **3(b)** in the evidence document.

[4]

(c) The function `Dequeue()` returns `"false"` if the queue is empty. If the queue is **not** empty, it returns the next item in the queue and updates the appropriate pointer(s).

Write program code for `Dequeue()`.

Save your program. Copy and paste the program code into part **3(c)** in the evidence document.

[3]

(d) The string values to be stored in the queue are 7 characters long. The first 6 characters are digits and the 7th character is a check digit. The check digit is calculated from the first 6 digits using this algorithm:

- multiply the digits in position 0, position 2 and position 4 by 1
- multiply the digits in position 1, position 3 and position 5 by 3
- calculate the sum of the products (add together the results from all of the multiplications)
- divide the sum of the products by 10 and round the result down to the nearest integer to get the check digit
- if the check digit equals 10 then it is replaced with 'X'.

Example:

Data is 954123

Character position	**0**	**1**	**2**	**3**	**4**	**5**
Digit	9	5	4	1	2	3
Multiplier	1	3	1	3	1	3
Product	9	15	4	3	2	9

Sum of products = 9 + 15 + 4 + 3 + 2 + 9 = 42

Divide sum of products by 10: 42 / 10 = 4 (rounded down)

The check digit = 4. This is inserted into character position 6.

The data including the check digit is: **9541234**

A 7-character string is valid if the 7th character matches the check digit for that data. For example, the data 9541235 is invalid because the 7th character (5) does **not** match the check digit for 954123.

(i) The subroutine `StoreItems()` takes ten 7-character strings as input from the user and uses the check digit to validate each input.

Each valid input has the check digit removed and is stored in the queue using `Enqueue()`.
An appropriate message is output if the item is inserted. An appropriate message is output if the queue is already full.

Invalid inputs are **not** stored in the queue.

The subroutine counts and outputs the number of invalid items that were entered.

`StoreItems()` can be a procedure or a function as appropriate.

Write program code for `StoreItems()`.

Save your program.

Copy and paste the program code into part **3(d)(i)** in the evidence document.

[6]

(ii) Write program code to amend the main program to:

- call `StoreItems()`
- call `Dequeue()`
- output a suitable message if the queue was empty
- output the returned value if the queue was **not** empty.

Save your program. Copy and paste the program code into part **3(d)(ii)** in the evidence document.

[1]

(iii) Test the program with the following inputs in the order given:

999999X

1251484

5500212

0033585

9845788

6666666

3258746

8111022

7568557

0012353

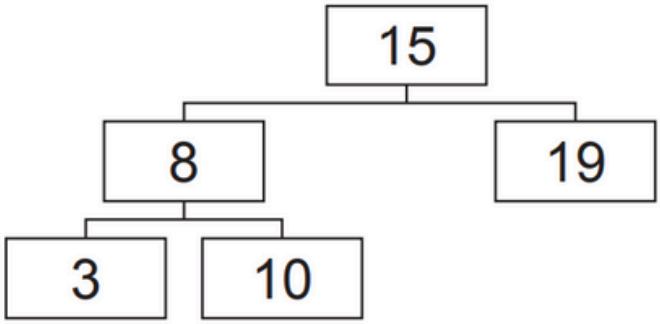
Take a screenshot of the output(s).

Save your program. Copy and paste the screenshot into part **3(d)(iii)** in the evidence document.

[2]

Question 12: Binary Tree (9618/42/MJ/24)

2 A binary tree stores data in ascending order. For example:



A computer program stores integers in a binary tree in ascending order. The program uses Object-Oriented Programming (OOP).

The binary tree is stored as a 1D array of nodes. Each node contains a left pointer, a data value and a right pointer.

The class `Node` stores the data about a node.

Node	
`LeftPointer : INTEGER`	stores the index of the node to the left in the binary tree
`Data : INTEGER`	stores the node's data
`RightPointer : INTEGER`	stores the index of the node to the right in the binary tree
`Constructor()`	initialises `Data` to its parameter value initialises `LeftPointer` and `RightPointer` to −1
`GetLeft()`	returns the left pointer
`GetRight()`	returns the right pointer
`GetData()`	returns the data value
`SetLeft()`	assigns the parameter to the left pointer
`SetRight()`	assigns the parameter to the right pointer
`SetData()`	assigns the parameter to the data

(a) (i) Write program code to declare the class `Node` and its constructor.

Do **not** declare the other methods.

Use the appropriate constructor for your programming language.

If you are writing in Python, include attribute declarations using comments.

Save your program as **Question2_J24**.

Copy and paste the program code into part **2(a)(i)** in the evidence document.

[4]

(ii) The get methods `GetLeft()`, `GetRight()` and `GetData()` each return the relevant attribute.

Write program code for the **three** get methods.

Save your program. Copy and paste the program code into part **2(a)(ii)** in the evidence document.

[3]

(iii) The set methods `SetLeft()`, `SetRight()` and `SetData()` each take a parameter and then store this in the relevant attribute.

Write program code for the **three** set methods.

Save your program. Copy and paste the program code into part **2(a)(iii)** in the evidence document.

[3]

(b) The class `TreeClass` stores the data about the binary tree.

TreeClass	
`Tree[0:19] : Node`	an array of 20 elements of type `Node`
`FirstNode : INTEGER`	stores the index of the first node in the tree
`NumberNodes : INTEGER`	stores the quantity of nodes in the tree
`Constructor()`	initialises `FirstNode` to −1 and `NumberNodes` to 0 initialises each element in `Tree` to a `Node` object with the data value of −1
`InsertNode()`	takes a `Node` object as a parameter, inserts it in the array and updates the pointer for its parent node
`OutputTree()`	outputs the left pointer, data and right pointer of each node in `Tree`

Nodes cannot be deleted from this tree.

(i) Write program code to declare the class `TreeClass` and its constructor.

Do **not** declare the other methods.

Use the appropriate constructor for your programming language.

If you are writing in Python, include attribute declarations using comments.

Save your program. Copy and paste the program code into part **2(b)(i)** in the evidence document.

[4]

(ii) The method `InsertNode()` takes a `Node` object, `NewNode`, as a parameter and inserts it into the array `Tree`.

`InsertNode()` first checks if the tree is empty.

If the tree is empty, `InsertNode()`:

- stores `NewNode` in the array `Tree` at index `NumberNodes`
- increments `NumberNodes`
- stores 0 in `FirstNode`.

If the tree is **not** empty, `InsertNode()`:

- stores `NewNode` in the array `Tree` at index `NumberNodes`
- accesses the data in the array `Tree` at index `FirstNode` and compares it to the data in `NewNode`
- repeatedly follows the pointers until the correct position for `NewNode` is found
- once the position is found, `InsertNode()` sets the left or right pointer of its parent node
- increments `NumberNodes`.

Write program code for `InsertNode()`.

Save your program.

Copy and paste the program code into part **2(b)(ii)** in the evidence document.

[6]

(iii) The method `OutputTree()` outputs the left pointer, the data and the right pointer for each node that has been inserted into the tree. The outputs are in the order they are saved in the array.

If there are no nodes in the array, the procedure outputs 'No nodes'.

Write program code for `OutputTree()`.

Save your program.

Copy and paste the program code into part **2(b)(iii)** in the evidence document.

[4]

(c) (i) The main program declares an instance of `TreeClass` with the identifier `TheTree`.

Write program code for the main program.

Save your program.

Copy and paste the program code into part **2(c)(i)** in the evidence document.

[1]

(ii) The main program inserts the following integers into the binary tree in the order given:

10

11

5

1

20

7

15

The main program then calls the method `OutputTree()`.

Write program code to amend the main program.

Save your program. Copy and paste the program code into part **2(c)(ii)** in the evidence document.

[4]

(iii) Test your program.

Take a screenshot of the output(s).

Save your program. Copy and paste the screenshot into part **2(c)(iii)** in the evidence document.

[1]

Question 13: Linked List (9618/41/ON/24)

3 A linked list stores positive integer data in a 2D array. The first dimension of the array stores the integer data. The second dimension of the array stores the pointer to the next node in the linked list.

A linked list node with no data is initialised with the integer `-1`. These nodes are linked together as an empty list. A pointer of `-1` identifies that node as the last node.

The linked list can store 20 nodes.

The global 2D array `LinkedList` stores the linked list.

`LinkedList` is initialised as an empty list. The data in each node is initialised to `-1`. Each node's pointer stores the index of the next node. The last node stores the pointer value `-1`, which indicates it is the last node.

The global variable `FirstEmpty` stores the index of the first element in the empty list. This is the first node in the empty linked list when it is initialised, which is index `0`.

The global variable `FirstNode` stores the index of the first element in the linked list. There is no data in the linked list when it is initialised, so `FirstNode` is initialised to `-1`.

This diagram shows the content of the initialised array.

```
FirstEmpty = 0

FirstNode = -1
```

Index	Data	Pointer
0	-1	1
1	-1	2
2	-1	3
3	-1	4
4	-1	5
⌒	⌒	⌒
19	-1	-1

(a) Write program code for the main program to declare and initialise `LinkedList`, `FirstNode` and `FirstEmpty`.

Save your program as **Question3_N24**.

Copy and paste the program code into part **3(a)** in the evidence document.

[2]

(b) The procedure `InsertData()` takes **five** positive integers as input from the user and inserts these into the linked list.

Each data item is inserted at the front of the linked list.

The table shows the steps to follow depending on the state of the linked list:

Linked list state	Steps
not full	insert the data in the index pointed to by `FirstEmpty` change the pointer to the index pointed to by `FirstNode` change the values of `FirstNode` and `FirstEmpty`
full	end the procedure

Any node that is at the end of the linked list has a pointer of `-1`.

Write program code for `InsertData()`.

Save your program. Copy and paste the program code into part **3(b)** in the evidence document.

[6]

(c) The procedure `OutputLinkedList()` outputs the data in the linked list in order by following the pointers from `FirstNode`.

(i) Write program code for `OutputLinkedList()`.

Save your program. Copy and paste the program code into part **3(c)(i)** in the evidence document.

[2]

(ii) Amend the main program to call `InsertData()` and then `OutputLinkedList()`.

Save your program. Copy and paste the program code into part **3(c)(ii)** in the evidence document.

[1]

(iii) Test your program with the test data:

5 1 2 3 8

Take a screenshot of the output.

Save your program. Copy and paste the screenshot into part **3(c)(iii)** in the evidence document.

[1]

(d) The procedure `RemoveData()` removes a node from the linked list.

The procedure takes the data item to be removed from the linked list as a parameter.

The procedure checks each node in the linked list, starting with the node `FirstNode`, until it finds the node to be removed. This node is added to the empty list, and pointers are changed as appropriate. The procedure only removes the first occurrence of the parameter.

Assume that the data item being removed is in the linked list.

(i) Write program code for `RemoveData()`.

Save your program. Copy and paste the program code into part **3(d)(i)** in the evidence document.

[5]

(ii) Amend the main program to:

- call `RemoveData()` with the parameter 5
- output the word `"After"`
- call `OutputLinkedList()`.

Save your program. Copy and paste the program code into part **3(d)(ii)** in the evidence document.

[1]

(iii) Test your program with both sets of given test data:

Test data set 1: 5 6 8 9 5

Test data set 2: 10 7 8 5 6

Take a screenshot of each output.

Save your program. Copy and paste the screenshot(s) into part **3(d)(iii)** in the evidence document.

[1]

Question 14: Queue (9618/42/ON/24)

2 A linear queue data structure is designed using a record structure.

The record structure `Queue` has the following fields:

- `QueueArray`, a 1D array of up to 100 integer values
- `Headpointer`, a variable that stores the index of the first data item in `QueueArray`
- `Tailpointer`, a variable that stores the index of the next free location in `QueueArray`.

(a) Write program code to declare the record structure `Queue` and its fields.

If your programming language does **not** support record structures, a class can be declared instead.

If you are writing in Python, use comments to declare the appropriate data types.

Save your program as **Question2_N24**. Copy and paste the program code into part **2(a)** in the evidence document.

[3]

(b) The main program creates a new `Queue` record with the identifier `TheQueue`. The head pointer is initialised to –1. The tail pointer is initialised to 0. Each element in the array is initialised with –1.

Write program code for the main program.

Save your program. Copy and paste the program code into part **2(b)** in the evidence document.

[3]

(c) The pseudocode function `Enqueue()` inserts an integer value into the queue.

The function is incomplete. There are **three** incomplete statements.

```
FUNCTION Enqueue(BYREF AQueue : Queue, BYVAL TheData : INTEGER)
                 RETURNS INTEGER

   IF AQueue.Headpointer = -1 THEN

      AQueue.QueueArray[AQueue.Tailpointer] ← .....

      AQueue.Headpointer ← 0

      AQueue.Tailpointer ← AQueue.Tailpointer + 1

      RETURN 1

   ELSE

      IF AQueue.Tailpointer > ..... THEN

         RETURN -1

      ELSE

         AQueue.QueueArray[AQueue.Tailpointer] ← TheData

         AQueue.Tailpointer ← AQueue.Tailpointer .....

         RETURN 1

      ENDIF

   ENDIF

ENDFUNCTION
```


Write program code for `Enqueue()`.

Save your program. Copy and paste the program code into part **2(c)** in the evidence document.

[5]

(d) The function `ReturnAllData()` accesses `TheQueue`. It concatenates all the integer values that have been inserted into the queue's array, starting from the value stored at `HeadPointer`, with a space between each integer value. The string of concatenated values is returned.

None of the integer values are removed from the queue.

Write program code for `ReturnAllData()`.

Save your program. Copy and paste the program code into part **2(d)** in the evidence document.

[3]

(e) **(i)** The main program asks the user to enter 10 integers with values of 0 or greater. It reads each input repeatedly until a valid number is entered.

All 10 valid inputs are added to the queue, using `Enqueue()`.

If the value returned from `Enqueue()` is –1, a message is output to state that the queue is full, otherwise a message is output to state that the item has been added to the queue.

The function `ReturnAllData()` is called once all 10 integers have been entered and the return value from the function call is output.

Amend the main program to perform these actions.

Save your program. Copy and paste the program code into part **2(e)(i)** in the evidence document.

[5]

(ii) Test your program with the following inputs in the order given:

10 9 –1 8 7 6 5 4 3 2 1

Take a screenshot of the output.

Save your program. Copy and paste the screenshot into part **2(e)(ii)** in the evidence document.

[2]

(f) The function `Dequeue()` accesses `TheQueue`. The function returns –1 if the queue is empty. If the queue is **not** empty, the function returns the next item in the queue and updates the relevant pointer(s).

The data is **not** replaced or deleted from the queue.

Write program code for `Dequeue()`.

Save your program. Copy and paste the program code into part **2(f)** in the evidence document.

[4]

(g) **(i)** The main program calls `Dequeue()` twice, and each time it either outputs 'Queue empty' if there is no data in the queue or outputs the return value.

The main program then calls `ReturnAllData()` a second time.

Amend the main program.

Save your program. Copy and paste the program code into part **2(g)(i)** in the evidence document.

[3]

(ii) Test your program with the following inputs in the order given:

10 9 8 7 6 5 4 3 2 1

Take a screenshot of the output.

Save your program. Copy and paste the screenshot into part **2(g)(ii)** in the evidence document.

[1]

Answer I

1(a) — **2**

1 mark per bullet point
- Declaring record/class with name node…
- …declaring data and next node (both as Integers)

Example code:

Visual Basic
```
Structure node
    Dim Data As Integer
    Dim nextNode As Integer
End Structure
```

Python
```
class node:
    def __init__(self, theData, nextNodeNumber):
        self. Data = theData
        self.nextNode = nextNodeNumber
```

Java
```
class node{
  private Integer Data;
  private Integer nextNode;
  public node(Integer dataP, Integer nextNodeP){
    this.Data = dataP;
    this.nextNode = nextNodeP;
  }
}
```

1(b) — **4**

1 mark per bullet point
- Declaring array named `linkedList` with data type `node`
- Assigning all nodes correctly as record/object nodes …
- …with correct values stored
- declaring `startPointer` as 0, `emptyList` as 5

Example code:

Visual Basic
```
Dim linkedList(9) As node
linkedList(0).data = 1
linkedList(0).nextNode = 1
linkedList(1).data = 5
linkedList(1).nextNode = 4
linkedList(2).data = 6
linkedList(2).nextNode = 7
linkedList(3).data = 7
linkedList(3).nextNode = -1
linkedList(4).data = 2
linkedList(4).nextNode = 2
linkedList(5).data = 0
linkedList(5).nextNode = 6
linkedList(6).data = 0
linkedList(6).nextNode = 8
linkedList(7).data = 56
linkedList(7).nextNode = 3
linkedList(8).data = 0
linkedList(8).nextNode = 9
linkedList(9).data = 0
linkedList(9).nextNode = -1
Dim startPointer As Integer = 0
Dim emptyList As Integer = 5
```

1(b)

Python
```
linkedList = [node(1,1),node(5,4),node(6,7),node(7,-1),node(2,2),node(0,6),
              node(0,8),node(56,3),node(0,9),node(0,-1)]
startPointer = 0
emptyList = 5
```

Java
```
public static void main(String[] args){
      node[] linkedList = new node[10];
      linkedList[0] = new node(1,1);
      linkedList[1] = new node(5, 4);
      linkedList[2] = new node(6, 7);
      linkedList[3] = new node(7,-1);
      linkedList[4] = new node(2,2);
      linkedList[5] = new node(0,6);
      linkedList[6] = new node(0,8);
      linkedList[7] = new node(56, 3);
      linkedList[8] = new node(0,9);
      linkedList[9] = new node(0,-1);
      Integer startPointer = 0;
      Integer emptyList = 5;
}
```

1(c)(i)	1 mark per bullet point
	• Procedure `outputNodes` ...
	• ...taking linked list and start pointer as parameters
	• Looping until `nextNode/pointer` is –1
	• Outputting the node data in the correct order, i.e. following pointers
	• Updating pointer to current node's `nextNode`
	• Using the correct record/class field/properties throughout
	
	Example code:
	
	Visual Basic
	`Sub outputNodes(ByRef linkedList, ByVal currentPointer)`
	`  While (currentPointer <> -1)`
	`  Console.WriteLine(linkedList(currentPointer).data)`
	`  currentPointer = linkedList(currentPointer).nextNode`
	`  End While`
	`End Sub`
	
	Python
	`def outputNodes(linkedList, currentPointer):`
	`  while(currentPointer != -1):`
	`  print(str(linkedList[currentPointer].data))`
	`  currentPointer = linkedList[currentPointer].nextNode`
	
	Java
	`public static void outputNodes(node[] linkedList, Integer currentPointer){`
	`  while(currentPointer != -1){`
	`  System.out.println(linkedList[currentPointer].data);`
	`  currentPointer = linkedList[currentPointer].nextNode;`
	`  }`
	`}`
		6

1(c)(ii)	Screenshot showing: 1 5 2 6 56 7	**1**

1(d)(i)	1 mark per bullet point to **max 7**
	• Function taking list and both pointers as parameters
	• Taking (integer) data as input
	• Checking if list is full ...
	• ... and returning False
	• Insert the input data to the empty list node's data
	• Following pointers to find last node in Linked List ...
	• ...and updating last node's pointer to empty list/location where new node is added
	• Updating empty list to it's first elements pointer
	• Returning true when added successfully
	
	Example code:
	
	Visual Basic
	`Function addNode(ByRef linkedList() As node, ByVal currentPointer As Integer, ByRef emptyList As Integer)`
	`  Console.WriteLine("Enter the data to add")`
	`  Dim dataToAdd As Integer = Console.ReadLine()`
	`  Dim previousPointer As Integer = 0`
	`  Dim newNode As node`
	`  If emptyList < 0 Or emptyList > 9 Then`
	`  Return False`
	`  Else`
	`  newNode.data = dataToAdd`
	`  newNode.nextNode = -1`
		7

1(d)(i)

```
            linkedList(emptyList) = newNode
            previousPointer = 0
            While (currentPointer <> -1)
                previousPointer = currentPointer
                currentPointer = linkedList(currentPointer).nextNode
            End While
            Dim valueToWrite As Integer = emptyList
            linkedList(previousPointer).nextNode = valueToWrite
            emptyList = linkedList(emptyList).nextNode

            Return True

        End If

    End Function
```

Python

```
def addNode(linkedList, currentPointer, emptyList):
    dataToAdd = input("Enter the data to add")

    if emptyList <0 or emptyList > 9:
        return False
    else:
        newNode = node(int(dataToAdd), -1)
        linkedList[emptyList] = (newNode)

        previousPointer = 0
        while(currentPointer != -1):
            previousPointer = currentPointer
            currentPointer = linkedList[currentPointer].nextNode
        linkedList[previousPointer].nextNode = emptyList
        emptyList = linkedList[emptyList].nextNode

        return True
```

1(d)(i)

Java

```
public static Boolean addNode(node[] linkedList, Integer currentPointer,
                              Integer emptyList){
    Integer dataToAdd;
    Integer previousPointer;
    node newNode;
    Scanner in = new Scanner(System.in);
    System.out.println("Enter the data to add");
    dataToAdd = in.nextInt();
    if(emptyList < 0 || emptyList > 9){
        return false;
    }else{
        newNode = new node(dataToAdd, -1);
        linkedList[emptyList] = newNode;
        previousPointer = 0;
        while(currentPointer != -1){
            previousPointer = currentPointer;
            currentPointer = linkedList[currentPointer].nextNode;

        }
        linkedList[previousPointer].nextNode = emptyList;
        emptyList = linkedList[emptyList].nextNode;
        return true;
    }
}
```

1(d)(ii)	1 mark per bullet point
	• Call `addNode()` with list, start and empty pointers and store/check return value …
	• …output appropriate message if True returned and if False returned
	• Calling `outputNodes()` with list and start pointer before and after `addNode()`
	
	Example code:
	
	Visual Basic
	`Sub Main()`
	`  Dim linkedList(10) As node`
	`  linkedList(0).data = 1`
	`  linkedList(0).nextNode = 1`
	`  linkedList(1).data = 5`
	`  linkedList(1).nextNode = 4`
	`  linkedList(2).data = 6`
	`  linkedList(2).nextNode = 7`
	`  linkedList(3).data = 7`
	`  linkedList(3).nextNode = -1`
	`  linkedList(4).data = 2`
	`  linkedList(4).nextNode = 2`
	`  linkedList(5).data = -1`
	`  linkedList(5).nextNode = 6`
	`  linkedList(6).data = -1`
	`  linkedList(6).nextNode = 7`
	`  linkedList(7).data = 56`
	`  linkedList(7).nextNode = 3`
	`  linkedList(8).data = -1`
	`  linkedList(8).nextNode = 9`
	`  linkedList(9).data = -1`
	`  linkedList(9).nextNode = -1`
	`  Dim startPointer As Integer = 0`
	`  Dim emptyList As Integer = 5`
	`  outputNodes(linkedList, startPointer)`
	`  Dim returnValue As Boolean`
	`  returnValue = addNode(linkedList, startPointer,`
	`  emptyList)`
		3

1(d)(ii)	`  If returnValue = True Then`
	`    Console.WriteLine("Item successfully added")`
	`  Else`
	`    Console.WriteLine("Item not added, list full")`
	`  End If`
	`  outputNodes(linkedList, startPointer)`
	`  Console.ReadLine()`
	`End Sub`
	
	Python
	`linkedList = [node(1,1),node(5,4),node(6,7),node(7,-1),node(2,2),node(-1,6),`
	`  node(-1,7),node(56,3),node(-1,9),node(-1,-1)]`
	`startPointer = 0`
	`emptyList = 5`
	`outputNodes(linkedList, startPointer)`
	`returnValue = addNode(linkedList, startPointer, emptyList)`
	`if returnValue == True:`
	`  print("Item successfully added")`
	`else:`
	`  print("Item not added, list full")`
	`outputNodes(linkedList, startPointer)`
	
	Java
	`public static void main(String[] args){`
	
	`  node[] linkedList = new node[10];`
	`  linkedList[0] = new node(1,1);`
	`  linkedList[1] = new node(5, 4);`
	`  linkedList[2] = new node(6, 7);`
	`  linkedList[3] = new node(7,-1);`
	`  linkedList[4] = new node(2,2);`
	`  linkedList[5] = new node(-1,6);`
	`  linkedList[6] = new node(-1,7);`
	`  linkedList[7] = new node(56, 3);`
	`  linkedList[8] = new node(-1,9);`	

1(d)(ii)	`linkedList[9] = new node(-1,-1);`
	`Integer startPointer = 0;`
	`Integer emptyList = 5;`
	`outputNodes(linkedList, startPointer);`
	`Boolean returnValue;`
	`returnValue = addNode(linkedList, startPointer, emptyList);`
	`if (returnValue == true){`
	`System.out.println("Item successfully added");`
	`}else{`
	`System.out.println("Item not added, list full");`
	`}`
	`outputNodes(linkedList, startPointer);`
	`}`	

1(d)(iii)	1 mark for screenshot showing : • Linked list output • 5 input • Message saying Successfully added or equivalent • Linked list output with 5 at the end. Example: 1 5 2 6 56 7 5 (being input) 1 5 2 6 56 7 5	**1**

Answer 2

3(a) — **4**

1 mark per bullet point

- Declaring array named `ArrayNodes` of type integer
- …with 20 by 3 elements
- `RootPointer` declared as integer and assigned -1
- `FreeNode` declared as integer and assigned 0

Example program code:

Python

```
ArrayNodes=[[0 for X in range(3)] for Y in range(20)]
RootPointer = -1
FreeNode = 0
```

VB.NET

```
Sub Main()
  Dim ArrayNodes(19, 2) As Integer
  Dim RootPointer As Integer = -1
  Dim FreeNode As Integer = 0
End Sub
```

Java

```
public static Integer[][] ArrayNodes = new Integer[20][3];
public static Integer RootPointer = -1;
public static Integer FreeNode = 0;
```

3(b) — **8**

1 mark for each completed statement to Max 6
1 mark per bullet point

- Function/procedure declaration either :
  taking parameters by reference
  returning the three amended values (Python)
  using global instead
- remainder of function/procedure matches the pseudocode

Example program code:

Python

```
def AddNode(ArrayNodes, RootPointer, FreeNode):
   NodeData = int(input("Enter the Data"))
   if FreeNode <= 19:
      ArrayNodes[FreeNode][0] = -1
      ArrayNodes[FreeNode][1] = NodeData
      ArrayNodes[FreeNode][2] = -1

      if RootPointer == -1:  # Add to start
          RootPointer = 0
      else:
          Placed = False
          CurrentNode = RootPointer
          while Placed == False:

              if NodeData < ArrayNodes[CurrentNode][1]:
                  if ArrayNodes[CurrentNode][0] == -1:
                    ArrayNodes[CurrentNode][0] = FreeNode
                    Placed = True
                  else:
                    CurrentNode = ArrayNodes[CurrentNode][0]
              else:
                  if ArrayNodes[CurrentNode][2] == -1:
                     ArrayNodes[CurrentNode][2] = FreeNode
                     Placed = True
                  else:
                    CurrentNode = ArrayNodes[CurrentNode][2]
     FreeNode = FreeNode + 1
  else:
      print("Tree is full")
  return ArrayNodes, RootPointer, FreeNode
```

3(b)

VB.NET

```
Sub AddNode(ByRef ArrayNodes, ByRef RootPointer,
            ByRef FreeNode)
  Console.WriteLine("Enter the Data")
  Dim NodeData As Integer = Console.ReadLine
  If FreeNode <= 19 Then
    ArrayNodes(FreeNode, 0) = -1
    ArrayNodes(FreeNode, 1) = NodeData
    ArrayNodes(FreeNode, 2) = -1

    If RootPointer = -1 Then
      RootPointer = 0
    Else
      Dim Placed As Boolean = False
      Dim CurrentNode As Integer = RootPointer
      While Placed = False
      If NodeData < ArrayNodes(CurrentNode, 1) Then
        If ArrayNodes(CurrentNode, 0) = -1 Then
          ArrayNodes(CurrentNode, 0) = FreeNode
          Placed = True
        Else
          CurrentNode = ArrayNodes(CurrentNode, 0)
        End If
      Else
        If ArrayNodes(CurrentNode, 2) = -1 Then
          ArrayNodes(CurrentNode, 2) = FreeNode
          Placed = True
        Else
          CurrentNode = ArrayNodes(CurrentNode, 2)
        End If
      End If
      End While
    Endif
    FreeNode = FreeNode + 1
  Else
    Console.WriteLine("Tree is full")
  End If
End Sub
```

3(b)	**Java**
	`public static void AddNode(){`
	`  System.out.println("Enter the Data");`
	`  Integer NodeData;`
	`  Scanner in = new Scanner(System.in);`
	`  NodeData = in.nextInt();`
	`  if(FreeNode <= 19){`
	`  ArrayNodes[FreeNode][0] = -1;`
	`  ArrayNodes[FreeNode][1] = NodeData;`
	`  ArrayNodes[FreeNode][2] = -1;`
	
	`  if (RootPointer == -1){`
	`  RootPointer = 0;`
	`  }else{`
	`  Boolean Placed = false;`
	`  Integer CurrentNode = RootPointer;`
	`  while(Placed == false){`
	`  if (NodeData < ArrayNodes[CurrentNode][1]){`
	`  if (ArrayNodes[CurrentNode][0] == -1){`
	`  ArrayNodes[CurrentNode][0] = FreeNode;`
	`  Placed = true;`
	`  }else{`
	`  CurrentNode = ArrayNodes[CurrentNode][0];`
	`  }`
	`  }else{`
	`  if (ArrayNodes[CurrentNode][2] == -1){`
	`  ArrayNodes[CurrentNode][2] = FreeNode;`
	`  Placed = true;`
	`  }else{`
	`  CurrentNode = ArrayNodes[CurrentNode][2];`
	`  }`
	`  }`
	`  }`
	`  }`
	
	`  FreeNode = FreeNode + 1;`
	`  }else{`
	`  System.out.println("Tree is full");`
	`  }`
	`  }`	

3(c)	1 mark per bullet point
	• procedure header (and end, take array as parameter)
	• Loops through all array elements // loops 20 times
	• Prints data in index 0, 1, 2 in each array element…
	• … in the correct order and format (spaces between)
	
	Example program code:
	Python
	`def PrintAll(ArrayNodes):`
	`  for X in range(0, 20):`
	`  print(str(ArrayNodes[X][0]), " ", str(ArrayNodes[X][1]),`
	`  " ", str(ArrayNodes[X][2]))`
	
	VB.NET
	`Sub PrintAll(ByRef ArrayNodes)`
	`  For X = 0 To 19`
	`  Console.WriteLine(ArrayNodes(X, 0) & " " & ArrayNodes(X,`
	`  1) & " " & ArrayNodes(X, 2))`
	`  Next`
	`End Sub`
	
	Java
	`public static void PrintAll(){`
	`  for(int X = 0; X < 20; X++){`
	`  System.out.println(ArrayNodes[X][0] + " " +`
	`  ArrayNodes[X][1] + " " + ArrayNodes[X][2]);`
	`  }`
	`}`
		4

3(d)(i)	1 mark per bullet point
	• looping 10 times
	• calling `AddNode` 10 times (check parameters in 3b)
	• calling `PrintAll` outside of loop (check parameters in 3c)
	
	Example program code:
	Python
	`for X in range(0,10):`
	`  ArrayNodes, RootPointer, FreeNode =`
	`  AddNode(ArrayNodes,RootPointer,FreeNode)`
	
	`PrintAll(ArrayNodes)`
	
	VB.NET
	`For X = 0 To 9`
	`  AddNode(ArrayNodes, RootPointer, FreeNode)`
	`Next`
	`printall(ArrayNodes)`
	
	Java
	`for (int X = 0; X < 10; X++){`
	`  AddNode();`
	`  }`
	`PrintAll();`
		3

3(d)(ii)	1 mark for screenshot showing the following output: 1 10 2 9 5 3 4 15 6 5 8 8 7 12 −1 −1 6 −1 −1 20 −1 −1 11 −1 −1 9 −1 −1 4 −1	**1**

3(e)(i)	1 mark per bullet point
	• procedure name `InOrder` taking a parameter (for current node being accessed)
	• Checking if left Node is empty (−1)
	• ….(if not) calling procedure recursively with [Current Node][0] as parameter
	• outputting the [Current Node][1]
	• checking if right Node is empty (−1)
	• …(if not) calling procedure recursively with [Current Node][2] as a parameter
	• Order is correct, left, root, right
	
	Example program code:
	
	Python
	`def InOrder(ArrayNodes, RootNode):`
	`  if ArrayNodes[RootNode][0] != -1:`
	`  InOrder(ArrayNodes, ArrayNodes[RootNode][0])`
	`  print(str(ArrayNodes[RootNode][1]))`
	`  if ArrayNodes[RootNode][2] != -1:`
	`  InOrder(ArrayNodes, ArrayNodes[RootNode][2])`
	
	VB.NET
	`Sub InOrder(ArrayNodes, RootNode)`
	`  If ArrayNodes(RootNode, 0) <> -1 Then`
	`  InOrder(ArrayNodes, ArrayNodes(RootNode, 0))`
	`  End If`
	`  Console.WriteLine(ArrayNodes(RootNode, 1))`
	`  If ArrayNodes(RootNode, 2) <> -1 Then`
	`  InOrder(ArrayNodes, ArrayNodes(RootNode, 2))`
	`  End If`
	`End Sub`
	
	Java
	`public static void InOrder(Integer Root){`
	`  if (ArrayNodes[Root][0] != -1){`
	`  InOrder(ArrayNodes[Root][0]);`
	`  }`
	`  System.out.println(ArrayNodes[Root][1]);`
	`  if(ArrayNodes[Root][2] != -1){`
	`  InOrder(ArrayNodes[Root][2]);`
	`  }`
	`}`
		7

3(e)(ii)	1 mark showing output: 4 5 6 8 9 10 11 12 15 20	**1**

Answer 3

3(a) — **2**

1 mark per mark point
- Declaring variables: head pointer, tail pointer and number of items all initialised as 0 (integer)
- `QueueArray` declared as 1D array as string with 10 elements

Example program code:

Java
```
public static void main(String[] args){
  String[] QueueArray = new String[10];
  Integer QueueHeadPointer = 0;
  Integer QueueTailPointer = 0;
  Integer NumberOfItems = 0;
}
```

Python
```
QueueArray = ['','','','','','','','','',''] #string
QueueHeadPointer = 0 #integer
QueueTailPointer = 0 #integer
NumberOfItems = 0 #integer
```

VB.NET
```
Sub Main()
 Dim QueueArray(0 To 9) As String
 Dim QueueHeadPointer As Integer = 0
 Dim QueueTailPointer As Integer = 0
 Dim NumberOfItems As Integer = 0
End Sub
```

3(b) — **7**

1 mark per complete statement (5)
1 mark for function heading and end, dealing with ByRef
1 mark for remainder of function correct and following the logic

```
FUNCTION Enqueue(BYREF QueueArray[] : STRING, BYREF HeadPointer : Integer, BYREF
                 TailPointer : Integer, NumberItems : INTEGER, DataToAdd : STRING) RETURNS
                 BOOLEAN
   IF NumberItems = 10 THEN
      RETURN FALSE
   ENDIF
   QueueArray[TailPointer] ← DataToAdd
   IF TailPointer >= 9 THEN
      TailPointer ← 0
   ELSE
      TailPointer ← TailPointer + 1
   ENDIF
   NumberItems ← NumberItems + 1
   RETURN TRUE
ENDFUNCTION
```

Example program code:

Java
```
public static Boolean Enqueue(String DataToAdd){
  if(NumberOfItems == 10){
    return false;
  }
  QueueArray[QueueTailPointer] = DataToAdd;
  if(QueueTailPointer >= 9){
    QueueTailPointer = 0;
  }else{
    QueueTailPointer = QueueTailPointer + 1;
  }
  NumberOfItems = NumberOfItems + 1;
  return true;
}
```

Question	Answer	Marks
3(b)		

Python
```
def Enqueue(Queue, Head, Tail, NumItems, InputData):
    if NumItems >= 10:
        return (False, Queue, Head, Tail, NumItems)
    Queue[Tail] = InputData
    if Tail >= 9:
        Tail = 0
    else:
        Tail = Tail + 1
    NumItems = NumItems + 1
    return (True, Queue, Head, Tail, NumItems)
```

VB.NET
```
Function Enqueue(ByRef Queue() As String, ByRef Head As Integer, ByRef Tail As Integer,
        ByRef NumItems As Integer, ByRef InputData As String)
  If NumItems = 10 Then
    Return False
  End If
  Queue(Tail) = InputData
  If Tail >= 9 Then
    Tail = 0
  Else
    Tail = Tail + 1
```

3(c)	1 mark per mark point to max 6
	• Function header and end
	• checking if queue is empty …
	• … returning False
	• If not empty accessing **and** returning item at head pointer
	• … incrementing head pointer …
	• … changing head pointer to 0 if it's more than 9 after incrementing
	• … decrement number of items
	
	Example program code:
	Java
	`public static String Dequeue(){`
	`  if(NumberOfItems == 0){`
	`  return "FALSE";`
	`  }else{`
	`  String ReturnValue = QueueArray[QueueHeadPointer];`
	`  QueueHeadPointer = QueueHeadPointer + 1;`
	`  if(QueueHeadPointer >= 9){`
	`  QueueHeadPointer = 0;`
	`  }`
	`  NumberOfItems = NumberOfItems - 1;`
	`  return ReturnValue;`
	`  }`
	`}`
	
	Python
	`def Dequeue(Queue, Head, Tail, NumItems):`
	`  if NumItems == 0:`
	`  return (false, Queue, Head, Tail, NumItems)`
	`  else:`
	`  ReturnValue = Queue(Head)`
	`  Head = Head + 1`
	`  if Head >= 9:`
	`  Head = 0`
	`  NumItems = NumItems - 1`
	`  return(ReturnValue, Queue, Head, Tail, NumItems)`
		6

3(c)	**VB.NET**
	`Function Dequeue(ByRef QueueArray() As String, ByRef QueueHeadPointer As Integer, ByRef QueueTailpointer As Integer, ByRef NumberOfItems As Integer)`
	
	`  If NumberOfItems = 0 Then`
	`  Return "False"`
	`  Else`
	`  Dim ReturnValue = QueueArray(QueueHeadPointer)`
	`  QueueHeadPointer = QueueHeadPointer + 1`
	`  If QueueHeadPointer >= 9 Then`
	`  QueueHeadPointer = 0`
	`  End If`
	`  NumberOfItems = NumberOfItems - 1`
	`  Return ReturnValue`
	`  End If`
	
	`End Function`	

3(d)(i)	1 mark per mark point
	
	• Taking 11 inputs…
	• … calling Enqueue with each of the 11 inputs …
	• … outputting an appropriate message if added or not added
	• Calling Dequeue twice …
	• … outputting return value each time
	
	Example program code:
	Java
	`public static void main(String args[]){`
	`  String InputString;`
	`  for(Integer x = 0; x < 11; x++){`
	`  System.out.println("Enter a string");`
	`  Scanner scanner = new Scanner(System.in);`
	`  InputString = scanner.nextLine();`
	`  if(Enqueue(InputString)){`
	`  System.out.println("Successful");`
	`  }else{`
	`  System.out.println("Unsuccessful");`
	`  }`
	`  }`
	`  System.out.println(Dequeue());`
	`  System.out.println(Dequeue());`
	`}`
		5

3(d)(i)	**Python**
	`for x in range(0, 11):`
	`    InputString = input("Enter a string")`
	`    ReturnValue, QueueArray, QueueHeadPointer, QueueTailPointer, NumberOfItems =`
	`     Enqueue(QueueArray, QueueHeadPointer, QueueTailPointer, NumberOfItems, InputString)`
	`    if ReturnValue == True:`
	`        print("Successful")`
	`    else:`
	`        print("Unsuccessful")`
	`ReturnValue, QueueArray, QueueHeadPointer, QueueTailPointer, NumberOfItems =`
	`  Dequeue(QueueArray, QueueHeadPointer, QueueTailPointer, NumberOfItems)`
	`print(ReturnValue)`
	`ReturnValue, QueueArray, QueueHeadPointer, QueueTailPointer, NumberOfItems =`
	`  Dequeue(QueueArray, QueueHeadPointer, QueueTailPointer, NumberOfItems)`
	`print(ReturnValue)`
	VB.NET
	`For x = 0 To 10`
	` Console.WriteLine("Enter a string")`
	` InputString = Console.ReadLine`
	` If(Enqueue(QueueArray, QueueHeadPointer, QueueTailPointer, NumberOfItems, InputString))`
	`Then`
	`   Console.WriteLine("Successful")`
	` Else`
	`   Console.WriteLine("Unsuccessful")`
	` End If`
	`Next`
	`Console.WriteLine(Dequeue)`
	`Console.WriteLine(Dequeue)`	

3(d)(ii)	1 mark for showing inputs and outputs:
	A – J input and successful.
	K input and unsuccessful.
	Output: A, B
	
	e.g.
	`Enter a string`
	`A`
	`Sucessful`
	`Enter a string`
	`B`
	`Sucessful`
	`Enter a string`
	`C`
	`Sucessful`
	`Enter a string`
	`D`
	`Sucessful`
	`Enter a string`
	`E`
	`Sucessful`
	`Enter a string`
	`F`
	`Sucessful`
	`Enter a string`
	`G`
	`Sucessful`
	`Enter a string`
	`H`
	`Sucessful`
	`Enter a string`
	`I`
	`Sucessful`
	`Enter a string`
	`J`
	`Sucessful`
	`Enter a string`
	`K`
	`Unsuccessful`
	`First value A`
	`Second value B`
		1

Answer 4

1(a)	1 mark per mark point
	• declaring array `StackData` and pointer `StackPointer` as (global data structures)
	• `StackData` has 10 integer elements
	• `StackPointer` initialised to 0
	
	Example program code:
	VB.NET
	`Dim StackData(9) As Integer`
	`Dim StackPointer As Integer`
	`Sub Main()`
	`  StackPointer = 0`
	`end Sub`
	
	Python
	`global StackData #integer`
	`global StackPointer`
	`StackData = [0,0,0,0,0,0,0,0,0,0] #integer`
	`StackPointer = 0`
	
	Java
	`import java.util.Scanner;`
	`class Question1{`
	`  public static Integer[] StackData;`
	`  public static Integer StackPointer;`
	`  public static void main(String args[]){`
	`  StackData = new Integer[10];`
	`  StackPointer = 0;`
	`  }`
	`}`
		3

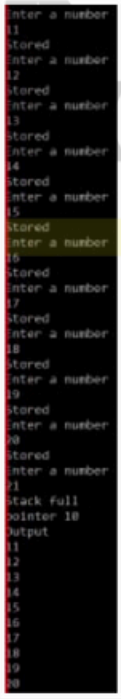
1(b)	1 mark per mark point
	• procedure header with sensible identifier (and end where appropriate)
	• outputting `StackPointer`
	• outputting all **10** elements in array
	
	Example program code:
	VB.NET
	`Sub PrintArray()`
	`  Console.WriteLine(StackPointer)`
	`  For x = 0 To 9`
	`  Console.WriteLine(StackData(x))`
	`  Next`
	`End Sub`
	
	Python
	`def PrintArray():`
	`  global StackData`
	`  global StackPointer`
	`  print(StackPointer)`
	`  for x in range (0, 10):`
	`  print(StackData[x])`
	
	Java
	`public static void PrintArray(){`
	`  System.out.println(StackPointer);`
	`  for(int x = 0; x < 10 ;x++){`
	`  System.out.println(StackData[x]);`
	`  }`
	`}`
		3

1(c)	1 mark per mark point
	• function `Push()` taking an integer parameter
	
	• checking if stack is full …
	• …and returning `FALSE`
	
	• (if not full) **storing parameter** to stack at `StackPointer` …
	• …incrementing `StackPointer`
	• …returning `TRUE`
	
	Example program code:
	VB.Net
	`Function Push(DataToPush)`
	`  If StackPointer = 10 Then`
	`  Return False`
	`  Else`
	`  StackData(StackPointer) = DataToPush`
	`  StackPointer = StackPointer + 1`
	`  Return True`
	`  End If`
	`End Function`
	
	Python
	`def Push(DataToPush):`
	`  global StackData`
	`  global StackPointer`
	`  if StackPointer == 10:`
	`  return False`
	`  else:`
	`  StackData[StackPointer] = DataToPush`
	`  StackPointer = StackPointer + 1`
	`  return True`
		6

1(c)	**Java**
	`public static Boolean Push(Integer DataToPush){`
	`  if(StackPointer == 10){`
	`  return false;`
	`  }else{`
	`  StackData[StackPointer] = DataToPush;`
	`  StackPointer = StackPointer + 1;`
	`  return true;`
	`  }`
	`}`	

1(d)(i)	1 mark per mark point
	• Inputting 11 numbers …
	• …calling `Push()` with each number input as a parameter …
	• …outputting appropriate message if `TRUE` returned
	• …outputting appropriate message if `FALSE` returned
	
	• Calling their output procedure **after** all **11** additions
	
	Example program code:
	VB.NET
	`Sub Main()`
	`  StackPointer = 0`
	`  Dim TempNumber As Integer`
	`  For x = 0 To 10`
	`  Console.WriteLine("Enter a number")`
	`  TempNumber = Console.ReadLine()`
	
	`  If Push(TempNumber) Then`
	`  Console.WriteLine("Stored")`
	`  Else`
	`  Console.WriteLine("Stack full")`
	`  End If`
	`  Next`
	`  PrintArray()`
	`  Console.ReadLine()`
	`End Sub`
		5

1(d)(i)	**Python**
	`#main`
	`StackPointer = 0`
	`StackData = [0,0,0,0,0,0,0,0,0,0]`
	`for x in range(0, 11):`
	`  TempNumber = int(input("Enter a number"))`
	
	`  if Push(TempNumber) == True:`
	`  print("Stored")`
	`  else:`
	`  print("Stack full")`
	`PrintArray()`
	
	Java
	`public static void main(String[] args){`
	`  StackData = new Integer[10];`
	`  StackPointer = 0;`
	`  Integer TempNumber = 0;`
	`  for(int x = 0; x < 10; x++){`
	`  System.out.println("Enter a number");`
	`  Scanner scanner = new Scanner(System.in);`
	`  TempNumber = Integer.parseInt(scanner.nextLine());`
	`  if(Push(TempNumber)){`
	`  System.out.println("Stored");`
	`  }else{`
	`  System.out.println("Stack full");`
	`  }`
	`  }`
	`  PrintArray();`
	`}`	

1(d)(ii)	1 mark for inputting all 11 numbers, message for first 10 saying added (11 to 20), message stating 11th number stating stack full. Full array contents output (11 12 13 14 15 16 17 18 19 20). e.g.  	1

1(e)(i)	1 mark per mark point
	• `Pop()` function header (and close where appropriate) **and** returning a number in all possible situations
	• checking if stack is empty (`StackPointer` is `0`) **and** returning `-1`
	• (otherwise) **accessing** the item at the top of the stack …
	• …decrementing the stack pointer
	• …returning the item removed
	
	Example Program code:
	VB.NET
	`Function Pop()`
	`  Dim ReturnData As Integer`
	`  If StackPointer = 0 Then`
	`  Return -1`
	`  Else`
	`  ReturnData = StackData(StackPointer - 1)`
	`  StackPointer = StackPointer - 1`
	`  Return ReturnData`
	`  End If`
	`End Function`
	
	Python
	`def Pop():`
	`  global StackData`
	`  global StackPointer`
	`  if StackPointer == 0:`
	`  return -1`
	`  else:`
	`  ReturnData = StackData[StackPointer - 1]`
	`  StackPointer = StackPointer - 1`
	`  return ReturnData`
		5

1(e)(i)	**Java**
	`public static Integer Pop(){`
	`  Integer ReturnData = 0;`
	`  if(StackPointer == 0){`
	`  return -1;`
	`  }else{`
	`  ReturnData = StackData[StackPointer - 1];`
	`  StackPointer = StackPointer - 1;`
	`  return ReturnData;`
	`  }`
	`}`	

1(e)(ii)	1 mark per mark point
	• output of before removed with 11 inputs
	• output of stack (after, this could be 11–20, 11–18, or 11–18 then 'null' 'null's)
	e.g.
	`Enter a number11`
	`Stored`
	`Enter a number12`
	`Stored`
	`Enter a number13`
	`Stored`
	`Enter a number14`
	`Stored`
	`Enter a number15`
	`Stored`
	`Enter a number16`
	`Stored`
	`Enter a number17`
	`Stored`
	`Enter a number18`
	`Stored`
	`Enter a number19`
	`Stored`
	`Enter a number20`
	`Stored`
	`Enter a number21`
	`Stack full`
	`8`
	`Output`
	`11` `12` `13` `14` `15` `16` `17` `18` `19` `20`
	`11` `12` `13` `14` `15` `16` `17` `18` `19` `20`
		2

Answer 5

3(a)	1 mark per point:
	
	• Declaring (global) 2D array `ArrayNodes`
	• looping through all 20 × 3 elements of array …
	• …. storing −1 in each element
	
	Example program code:
	
	Java
	
	`public static Integer[][] ArrayNodes = new Integer[20][3];`
	`for(Integer X = 0; X<20; X++){`
	`  for(Integer Y = 0; Y<3; Y++){`
	`  ArrayNodes[X][Y] = -1`
	`}}`
	
	Python
	
	`ArrayNodes = []`
	`for x in range(0, 20):`
	`  ArrayNodes.append([-1, -1, -1])`
	
	VB.NET
	
	`Dim ArrayNodes(19, 2) As Integer`
	`Sub main()`
	`  For X = 0 To 19`
	`  For Y = 0 To 2`
	`  ArrayNodes(X, Y) = -1`
	`  Next`
	`  Next`
	`End Sub`
		3

3(b)	1 mark per point:
	
	• initialising each of the first 6 array elements correctly
	• declaring and initialising `FreeNode` to 6 and `RootPointer` to 0
	
	Example program code:
	
	Python
	
	`ArrayNodes = [[1,20,5],[2,15,-1],[-1,3,3],[-1,9,4],[-1,10,-1],[-1,58,-1]]`
	`FreeNodes = 6`
	`RootPointer = 0`
	
	Java
	
	`ArrayNodes[0][0] = 1;`
	`ArrayNodes[0][1] = 20;`
	`ArrayNodes[0][2] = 5;`
	
	`ArrayNodes[1][0] = 2;`
	`ArrayNodes[1][1] = 15;`
	`ArrayNodes[1][2] = -1;`
	
	`ArrayNodes[2][0] = -1;`
	`ArrayNodes[2][1] = 3;`
	`ArrayNodes[2][2] = 3;`
	
	`ArrayNodes[3][0] = -1;`
	`ArrayNodes[3][1] = 9;`
	`ArrayNodes[3][2] = 4;`
	
	`ArrayNodes[4][0] = -1;`
	`ArrayNodes[4][1] = 10;`
	`ArrayNodes[4][2] = -1;`
	`ArrayNodes[5][0] = -1;`
	`ArrayNodes[5][1] = 58;`
	`ArrayNodes[5][2] = -1;`
	
	`Integer FreeNode = 6;`
	`Integer RootPointer = 0;`
		2

3(b)

VB.NET

```
ArrayNodes(0, 0) = 1
ArrayNodes(0, 1) = 20
ArrayNodes(0, 2) = 5

ArrayNodes(1, 0) = 2
ArrayNodes(1, 1) = 15
ArrayNodes(1, 2) = -1

ArrayNodes(2, 0) = -1
ArrayNodes(2, 1) = 3
ArrayNodes(2, 2) = 3

ArrayNodes(3, 0) = -1
ArrayNodes(3, 1) = 9
ArrayNodes(3, 2) = 4

ArrayNodes(4, 0) = -1
ArrayNodes(4, 1) = 10
ArrayNodes(4, 2) = -1

ArrayNodes(5, 0) = -1
ArrayNodes(5, 1) = 58
ArrayNodes(5, 2) = -1

Dim FreeNode As Integer = 6
Dim RootPointer As Integer = 0
```


3(c)

1 mark for each completed statement (4)
1 mark for remainder of function correct

5

Pseudocode:

```
FUNCTION SearchValue(BYVAL Root : INTEGER, ValueToFind :
INTEGER)

  IF Root = -1 THEN
    RETURN -1
  ELSE
    IF ArrayNodes[Root,1] = ValueToFind THEN
       RETURN Root
    ELSE
       IF ArrayNodes[Root, 1] = -1 THEN
         RETURN -1
       ENDIF
    ENDIF
  ENDIF
  IF ArrayNodes[Root,1] > ValueToFind THEN
    RETURN SearchValue(ArrayNodes[Root,0], ValueToFind)
  ENDIF

  IF ArrayNodes[Root,1] < ValueToFind THEN
    RETURN SearchValue(ArrayNodes[Root,2], ValueToFind)
  ENDIF

ENDFUNCTION
```

Example program code:

Python

```
def SearchValue(Root, ValueToFind):
  global ArrayNodes
  if Root == -1:
   return -1
  elif ArrayNodes[Root][1] == ValueToFind:
    return Root
  elif ArrayNodes[Root][1] == -1:
    return -1
  if(ArrayNodes[Root][1] > ValueToFind):
    return SearchValue(ArrayNodes[Root][0], ValueToFind)
  if(ArrayNodes[Root][1] < ValueToFind):
    return SearchValue(ArrayNodes[Root][2], ValueToFind)
```

3(c)

Java

```
public static Integer SearchValue(Integer Root, Integer
ValueToFind){
  if(Root == -1){
      return -1;
  }else if(ArrayNodes[Root][1] == ValueToFind){;
      return Root;
  }else if(ArrayNodes[Root][1] == -1){
      return -1;
  }
  if(ArrayNodes[Root][1] > ValueToFind){
      return(SearchValue(ArrayNodes[Root][0],
ValueToFind));
  }
  if(ArrayNodes[Root][1] < ValueToFind){
      return(SearchValue(ArrayNodes[Root][2],
ValueToFind));
  }
  return -1;
}
```

VB.NET

```
Function SearchValue(ByVal Root, ByVal ValueToFind)

  If ArrayNodes(Root, 1) = ValueToFind Then
    Return Root
  ElseIf ArrayNodes(Root, 1) = -1 Then
    Return -1
  End If
  If ArrayNodes(Root, 1) > ValueToFind Then
    Return SearchValue(ArrayNodes(Root, 0), ValueToFind)
  End If
  If ArrayNodes(Root, 1) < ValueToFind Then
    Return SearchValue(ArrayNodes(Root, 2), ValueToFind)
  End If
  Return -1
End Function
```

3(d)	1 mark per point (**Max 7**):
	
	• (procedure) header (and end where appropriate) with one parameter (root node or index of root node) and at least one recursive call
	• checking if left node is −1 …
	• … if not recursive call with parameter as `ArrayNodes[RootNode[0]]`
	• checking if right node is −1 …
	• …if not recursive call with parameter as `ArrayNodes[RootNode[2]]`
	• outputting the element at the parameter `RootNode[]`
	• all 3 in the correct order
	
	Example program code:
	
	Python
	`def PostOrder(RootNode):`
	`  if RootNode[0] != -1:`
	`  PostOrder(ArrayNodes[RootNode[0]])`
	`  if RootNode[2] != -1:`
	`  PostOrder(ArrayNodes[RootNode[2]])`
	`  print(str(RootNode[1]))`
	
	Java
	`public static void PostOrder(Integer[] RootNode){`
	`  if(RootNode[0] != -1){`
	`  PostOrder(ArrayNodes[RootNode[0]]);`
	`  }`
	`  if(RootNode[2] != -1){`
	`  PostOrder(ArrayNodes[RootNode[2]]);`
	`  }`
	`  System.out.println(RootNode[1]);`
	`  }`
	
	VB.NET
	`Sub PostOrder(RootNode() As Integer)`
	`  Dim TempArray(2) As Integer`
	`  If RootNode(0) <> -1 Then`
	`  TempArray(0) = ArrayNodes(RootNode(0), 0)`
	`  TempArray(1) = ArrayNodes(RootNode(0), 1)`
	`  TempArray(2) = ArrayNodes(RootNode(0), 2)`
	`  PostOrder(TempArray)`
	`  End If`
	`  If RootNode(2) <> -1 Then`
	`  TempArray(0) = ArrayNodes(RootNode(2), 0)`
	`  TempArray(1) = ArrayNodes(RootNode(2), 1)`
	`  TempArray(2) = ArrayNodes(RootNode(2), 2)`
	`  PostOrder(TempArray)`
	`  End If`
	`  Console.WriteLine(RootNode(1))`
	`End Sub`
		7

3(e)(i)	1 mark per point:
	
	• calling `SearchValue()` with 15 and `rootPointer` as a parameter …
	• … if **return value** > -1 output returned index **and** if **return value** = -1 output not found
	Both as appropriate messages
	• Calling `PostOrder()` with `ArrayNodes[RootPointer]` as a parameter
	
	Example program code:
	
	Python
	`ReturnValue = SearchValue(RootPointer, 15)`
	`if ReturnValue == -1:`
	`  print("Not found")`
	`else:`
	`  print("Found at " + str(ReturnValue))`
	`PostOrder(ArrayNodes[RootPointer])`
	
	Java
	`Integer ReturnValue = SearchValue(RootPointer, 15);`
	`if(ReturnValue == -1){`
	`  System.out.println("Not found");`
	`} else {`
	`  System.out.println("Found at " + ReturnValue);`
	`}`
	`PostOrder(ArrayNodes[RootPointer]);`
	
	VB.NET
	`Dim returnvalue As Integer = SearchValue(RootPointer, 15)`
	`If returnvalue = -1 Then`
	`  Console.WriteLine("Not found")`
	`Else`
	`  Console.WriteLine("Found at " & returnvalue)`
	`End If`
	`Console.WriteLine("Post order")`
	`Dim TempArray(2) As Integer`
	`TempArray(0) = ArrayNodes(RootPointer, 0)`
	`TempArray(1) = ArrayNodes(RootPointer, 1)`
	`TempArray(2) = ArrayNodes(RootPointer, 2)`
	`PostOrder(TempArray)`
		3

3(e)(ii)	Screenshot with result as shown, for example:
	
	`Found at 1`
	`10`
	`9`
	`3`
	`15`
	`58`
	`20`
		1

Answer 6

3(a)	1 mark per point:
	
	• 1D array with 100 (Integer) spaces
	• head pointer declared initialised to appropriate value e.g. −1
	• tail pointer declared initialised to 0
	
	Example program code:
	
	Java
	`public Integer[] queue = new Integer[100];`
	`public Integer HeadPointer = -1;`
	`public Integer TailPointer = 0;`
	
	Python
	`Queue = [-1 for I in range(100)] #Integer`
	`HeadPointer = -1`
	`TailPointer = 0`
	
	VB.NET
	`Dim Queue(0 To 99) As Integer`
	`Dim HeadPointer As Integer = -1`
	`Dim TailPointer As Integer = 0`
		3

3(b)	1 mark per point:
	
	• Function header (and close where appropriate) with integer parameter
	• Checking if queue full **and** returning false
	• If not full adding **parameter** to queue at **tail pointer** ...
	• ... incrementing tail pointer (after adding to queue)
	• ... and returning true
	• Changing head pointer to **0** if this is the first element in array
	
	Example program code:
	
	Java
	
	`public Boolean Enqueue(Integer Data){`
	`  if(TailPointer < 100){`
	`    if(HeadPointer == -1){`
	`      HeadPointer = 0;`
	`    }`
	`    Queue[TailPointer] = Data;`
	`    TailPointer = TailPointer + 1;`
	`    return true;`
	`   }`
	`  return false;`
	`}`
	Python
	`def Enqueue(Data):`
	`    global Queue`
	`    global TailPointer`
	`    if(TailPointer < 100):`
	`      if HeadPointer == -1:`
	`        HeadPointer = 0`
	`      Queue[TailPointer] = Data`
	`      TailPointer = TailPointer + 1`
	`      return True`
	`    return False`
		6

VB.NET

```
Function Enqueue(Data)
  If TailPointer < 100 Then
    If HeadPointer = -1 Then
      HeadPointer = 0
    End If
    Queue(TailPointer) = data
    TailPointer = TailPointer + 1
    Return True
  End If
  Return False
End Function
```

3(c)	1 mark per point: • Looping 20 times • … using `Enqueue()` with each number 1 to 20 in ascending numerical order… • … and storing/using the return value • … based on return value, outputting "Successful" and "Unsuccessful" if **all** numbers are added Example program code:	**4**

Java

```
public static void main(String[] args){
  Boolean success = false;
  for(Integer count = 1; count <= 20; count++){
    success = enqueue(count);
  }
  if(success == false){
    System.Out.Println("Unsuccessful ")
  else{
    System.Out.Println("Successful ")
  }
}
```

Python

```
Success = False
for Count in range(1, 21):
    Success = Enqueue(Count)
if(Success == False):
  print("Unsuccessful")
else:
  print("Successful")
```

VB.NET

```
Dim Success As Boolean
For Count = 1 To 20
  Success = Enqueue(Count)
Next
If Success = False THEN
  Console.WriteLine("Unsuccessful")
ELSE
  Console.WriteLine("Successful")
ENDIF
```

3(d)	1 mark per point:
	
	• function call (and end where appropriate) taking a parameter
	• checking if at start of queue//20 …
	• …returning the last value in the queue
	• (otherwise) adding return value to a total // adding value in queue before recursive call **and** using this in the recursive call …
	• recursive call with Start/pointer −1
	• returning the final total
	
	Example program code:
	
	Java
	`public static Integer RecursiveOutput(Integer Start){`
	`  if(Start == 0){`
	`  return Queue[Start];`
	`  }else{`
	`  return Queue[Start] + RecursiveOutput(Start -1);`
	`  }}`
	
	Python
	`def RecursiveOutput(Start):`
	`  if(Start == 0):`
	`  return Queue[Start]`
	`  else:`
	`  return Queue[Start] + RecursiveOutput(Start - 1)`
	
	VB.NET
	`Function RecursiveOutput(ByVal Start)`
	`  If (Start = 0) Then`
	`  Return Queue(Start)`
	`  Else`
	`  Return Queue(Start) + RecursiveOutput(Start - 1)`
	`  End If`
	`End Function`
		6

3(e)(i)	1 mark for calling function **and** outputting return value.
	
	Example program code:
	
	Java
	`System.out.println(RecursiveOutput(TailPointer-1));`
	
	Python
	`print(str(RecursiveOutput(TailPointer - 1)))`
	
	VB.NET
	`Console.WriteLine(RecursiveOutput(TailPointer - 1))`
		1

3(e)(ii)	1 mark for screenshot showing 210, for example:  	**1**

Answer 7

3(a)	1 mark each
	• (Global) `Animal` array (with 20 string elements)
	• (Global) `Colour` array (with 10 string elements)
	• (Global) `AnimalTopPointer` and `ColourTopPointer` initialised to 0
	
	Example program code:
	
	Java
	`public static String[] Animal = new String[20];`
	`public static String[] Colour = new String[10];`
	`public static Integer AnimalTopPointer = 0;`
	`public static Integer ColourTopPointer = 0;`
	
	VB.NET
	`Dim Animal(0 to 19) As String`
	`Dim Colour(0 to 9) As String`
	`Dim AnimalTopPointer As Integer = 0`
	`Dim ColourTopPointer As Integer = 0`
	
	Python
	`Animal = [] #20 elements`
	`Colour = [] #10 elements`
	`global AnimalTopPointer`
	`global ColourTopPointer`
	`AnimalTopPointer = 0`
	`ColourTopPointer = 0`
		3

3(b)(i)	1 mark each
	• Function header (and close where appropriate) with parameter, checking if full (AnimalTopPointer = 20) **and** returning false
	• If not full, inserting parameter value into AnimalTopPointer
	• …incrementing pointer **and** returning true
	
	Example program code:
	
	Java
	`public static Boolean PushAnimal(String DataToPush){`
	`  if(AnimalTopPointer == 20){`
	`  return false;`
	`  }else{`
	`  Animal[AnimalTopPointer] = DataToPush;`
	`  AnimalTopPointer++;`
	`  return true;`
	`  }`
	`}`
	
	VB.NET
	`Function PushAnimal(DataToPush)`
	`  If AnimalTopPointer = 20 Then`
	`  Return False`
	`  Else`
	`  Animal(AnimalTopPointer) = DataToPush`
	`  AnimalTopPointer = AnimalTopPointer + 1`
	`  Return True`
	`  End If`
	`End Function`
	
	Python
	`def PushAnimal(DataToPush):`
	`  global AnimalTopPointer`
	`  global ColourTopPointer`
	`  if AnimalTopPointer == 20:`
	`  return False`
		3

3(b)(i)	`else:`
	`  Animal.append(DataToPush)`
	`  AnimalTopPointer +=1`
	`  return True`	

3(b)(ii)	1 mark each
	• Procedure header (and end where appropriate) with no parameter, checking if empty (AnimalTopPointer = 0) **and** returning empty string
	• If not empty returning the top data item (AnimalTopPointer-1)
	• … and decrementing AnimalTopPointer
	
	Example program code:
	
	Java
	`public static String PopAnimal(){`
	`  String ReturnData;`
	`  if(AnimalTopPointer == 0){`
	`  return "";`
	`  }else{`
	`  ReturnData = Animal[AnimalTopPointer - 1];`
	`  AnimalTopPointer--;`
	`  return ReturnData;`
	`  }`
	`}`
	
	VB.NET
	`Function PopAnimal()`
	`  Dim ReturnData As String`
	`  If AnimalTopPointer = 0 Then`
	`  Return ""`
	`  Else`
	`  ReturnData = Animal(AnimalTopPointer - 1)`
	`  AnimalTopPointer = AnimalTopPointer - 1`
	`  Return ReturnData`
	`  End If`
	`End Function`
		3

3(b)(ii)	**Python**
	`def PopAnimal():`
	`  global AnimalTopPointer`
	`  global ColourTopPointer`
	`  if AnimalTopPointer == 0:`
	`  return ""`
	`  else:`
	`  ReturnData = Animal[AnimalTopPointer - 1]`
	`  AnimalTopPointer -=1`
	`  return ReturnData`	

3(b)(iii)	1 mark
	• Procedure header (and close where appropriate) **and** opening correct file for read
	• Looping until end of file // looping until all animal names read in // looping 8 times
	• Calling `PushAnimal()` with each line read from file (for all lines)
	• Closing the file
	• Exception handling with appropriate error message
	
	Example program code:
	
	Java
	`private static void ReadData(){`
	`  try{`
	`  Scanner Scanner1 = new Scanner(new File("AnimalData.txt"));`
	`  while(Scanner1.hasNextLine()){`
	`  PushAnimal(Scanner1.next());`
	`  }`
	`  Scanner1.close();`
	`  }catch(FileNotFoundException ex){`
	`  System.out.println("No Animal file found");`
	`  }`
	`}`
	
	VB.NET
	`Sub ReadData()`
	`  try`
	`  Dim AnimalFile As String = "AnimalData.txt"`
	`  Dim AnimalFileReader As New System.IO.StreamReader(AnimalFile)`
	`  Do Until AnimalFileReader.EndOfStream`
	`  PushAnimal(AnimalFileReader.ReadLine())`
	`  Loop`
	`  AnimalFileReader.Close()`
	`  Catch ex As Exception`
	`  Console.WriteLine("Invalid file")`
	`  End Try`
	
	`End Sub`
		5

3(b)(iii)	**Python**
	`def ReadData():`
	`  try:`
	`  global AnimalTopPointer`
	`  global ColourTopPointer`
	`  AnimalFile = open("AnimalData.txt", 'r')`
	`  for Line in AnimalFile:`
	`  PushAnimal(Line)`
	`  AnimalFile.close()`
	`  except IOError:`
	`  print("Could not find file")`	

3(b)(iv)	1 mark each
	• `PushColour` function
	• `PopColour` function
	
	Example program code:
	
	Java
	`public static Boolean PushColour(String DataToPush){`
	`  if(ColourTopPointer == 10){`
	`  return false;`
	`  }else{`
	`  Colour[ColourTopPointer] = DataToPush;`
	`  ColourTopPointer++;`
	`  return true;`
	`  }`
	`}`
	`public static String PopColour(){`
	`  String ReturnData;`
	`  if(ColourTopPointer == 0){`
	`  return "";`
	`  }else{`
	`  ReturnData = Colour[ColourTopPointer - 1];`
	`  ColourTopPointer--;`
	`  return ReturnData;`
	`  }`
	`}`
	
	VB.NET
	`Function PushColour(DataToPush)`
	`  If ColourTopPointer = 10 Then`
	`  Return False`
	`  Else`
	`  Colour(ColourTopPointer) = DataToPush`
	`  ColourTopPointer = ColourTopPointer + 1`
	`  Return True`
	`  End If`
		2

3(b)(iv)	`End Function`
	`Function PopColour()`
	`  Dim ReturnData As String`
	`  If ColourTopPointer = 0 Then`
	`  Return ""`
	`  Else`
	`  ReturnData = Colour(ColourTopPointer - 1)`
	`  ColourTopPointer = ColourTopPointer - 1`
	`  Return ReturnData`
	`  End If`
	`End Function`
	
	Python
	`def PushColour(DataToPush):`
	`  global AnimalTopPointer`
	`  global ColourTopPointer`
	`  if ColourTopPointer == 10:`
	`  return False`
	`  else:`
	`  Colour.append(DataToPush)`
	`  ColourTopPointer +=1`
	`  return True`
	
	`def PopColour():`
	`  global AnimalTopPointer`
	`  global ColourTopPointer`
	`  if ColourTopPointer == 0:`
	`  return ""`
	`  else:`
	`  ReturnData = Colour[ColourTopPointer - 1]`
	`  ColourTopPointer -=1`
	`  return ReturnData`	

3(b)(v)	1 mark each
	• Opening `ColourData.txt` to read, reading until EOF, closing file and exception handling
	• Using `PushColour()` to store each item read from the file **for all lines**
	
	Example program code:
	
	Java
	`private static void ReadData(){`
	`  try{`
	`  Scanner Scanner1 = new Scanner(new File("AnimalData.txt"));`
	`  while(Scanner1.hasNextLine()){`
	`  PushAnimal(Scanner1.next());`
	`  }`
	`  Scanner1.close();`
	`  }catch(FileNotFoundException ex){`
	`  System.out.println("No Animal file found");`
	`  }`
	
	`  try{`
	`  Scanner Scanner2 = new Scanner(new File("ColourData.txt"));`
	`  while(Scanner2.hasNextLine()){`
	`  PushColour(Scanner2.next());`
	`  }`
	`  Scanner2.close();`
	`  }catch(FileNotFoundException ex){`
	`  System.out.println("No Colour file found");`
	`  }`
	`}`
	
	VB.NET
	`Sub ReadData()`
	`  try`
	`  Dim AnimalFile As String = "AnimalData.txt"`
	`  Dim AnimalFileReader As New System.IO.StreamReader(AnimalFile)`
	`  Do Until AnimalFileReader.EndOfStream`
	`  PushAnimal(AnimalFileReader.ReadLine())`
		2

3(b)(v)	`  Loop`
	`  AnimalFileReader.Close()`
	`  Dim ColourFile As String = "ColourData.txt"`
	`  Dim ColourFileReader As New System.IO.StreamReader(ColourFile)`
	`  Do Until ColourFileReader.EndOfStream`
	`  PushColour(ColourFileReader.ReadLine())`
	`  Loop`
	`  ColourFileReader.Close()`
	`  Catch ex As Exception`
	`  Console.WriteLine("Invalid file")`
	`  End Try`
	
	`End Sub`
	
	Python
	`def ReadData():`
	`  try:`
	`  global AnimalTopPointer`
	`  global ColourTopPointer`
	`  AnimalFile = open("AnimalData.txt", 'r')`
	`  for Line in AnimalFile:`
	`  PushAnimal(Line)`
	`  AnimalFile.close()`
	
	`  ColourFile = open("ColourData.txt", 'r')`
	`  for Line in ColourFile:`
	`  PushColour(Line)`
	`  ColourFile.close()`
	`  except IOError:`
	`  print("Could not find file")`	

3(c)	1 mark each to max 5
	• Procedure heading (and close where appropriate) **and** outputting the colour and animal using `PopColour()` **and** `PopAnimal()` (only if both are successfully popped)
	• Checking if no colour **and** outputting "No colour" …
	• ….pushing the removed animal back onto the stack
	• Checking if no animal **and** outputting "No animal" …
	• …pushing the removed colour back onto the stack
	
	Example program code:
	
	Java
	`public static void OutputItem(){`
	`  String ColourReturned = PopColour();`
	`  String AnimalReturned = PopAnimal();`
	`  if(ColourReturned.equals("")){`
	`  System.out.println("No colour");`
	`  PushAnimal(AnimalReturned);`
	`  }else{`
	`  if(AnimalReturned.equals("")){`
	`  System.out.println("No animal");`
	`  PushColour(ColourReturned);`
	`  }else{`
	`  System.out.println("A " + ColourReturned + " " + AnimalReturned);`
	`  }`
	`  }`
	`}`
	
	VB.NET
	`Sub OutputItem()`
	`  Dim ColourReturned As String = PopColour()`
	`  Dim Animalreturned As String = PopAnimal()`
	`  If ColourReturned = "" Then`
	`  Console.WriteLine("No colour")`
	`  PushAnimal(AnimalReturned)`
		5

3(c)	`  Else`
	`  If Animalreturned = "" Then`
	`  Console.WriteLine("No animal")`
	`  PushColour(ColourReturned)`
	`  Else`
	`  Console.WriteLine("A " & ColourReturned & " " & Animalreturned)`
	`  End If`
	`  End If`
	`End Sub`
	
	Python
	`def OutputItem():`
	`  global AnimalTopPointer`
	`  global ColourTopPointer`
	`  ColourReturned = PopColour()`
	`  AnimalReturned = PopAnimal()`
	`  if ColourReturned == "":`
	`  print("No colour")`
	`  PushAnimal(AnimalReturned)`
	`  else:`
	`  if AnimalReturned == "":`
	`  print("No animal")`
	`  PushColour(ColourReturned)`
	`  else:`
	`  print(ColourReturned, AnimalReturned)`	

3(d)(i)	1 mark for
	• Calling `ReadData()` and calling `OutputItem()` 4 times
	
	Example program code:
	
	Java
	`public static void main(String args[]){`
	`  ReadData();`
	`  OutputItem();`
	`  OutputItem();`
	`  OutputItem();`
	`  OutputItem();`
	`}`
	
	VB.NET
	`Sub Main()`
	`  ReadData()`
	`  OutputItem()`
	`  OutputItem()`
	`  OutputItem()`
	`  OutputItem()`
	`End Sub`
	
	Python
	`ReadData()`
	`OutputItem()`
	`OutputItem()`
	`OutputItem()`
	`OutputItem()`
		1

3(d)(ii)	1 mark for output
	e.g.
	`pink hamster`
	`blue elephant`
	`white eagle`
	`yellow rabbit`
		1

Answer 8

<table>
<tr><th>Question</th><th>Answer</th><th>Marks</th></tr>
<tr><td>2(a)</td><td>1 mark each
• Record declaration named <code>SaleData</code> // class declaration (and end) named <code>SaleData</code>...
• ...<code>SaleID</code> declared as string, <code>Quantity</code> as integer in record
// if a class then a constructor assigning attributes <code>SaleID</code> and <code>Quantity</code>

Example Program code:

Java
<pre>class SaleData{
      private String SaleId;
      private Integer Quantity;
      public SaleData(String SaleIDP, Integer Quantityp){
            SaleId = SaleIDP;
            Quantity = Quantityp;
      }
}</pre>VB.NET
<pre>Structure SaleData
        Public SaleID As String
        Public Quantity As Integer
End Structure</pre>Python
<pre>class SaleData:
    def __init__(self, SaleIDp, Quantityp):
        self.SaleID = SaleIDp #string
        self.Quantity = Quantityp #integer</pre></td><td>2</td></tr>
<tr><td>2(b)</td><td>1 mark each
• Global array <code>CircularQueue</code> of 5 items of type <code>SaleData</code>
• Global variables <code>Head</code>, <code>Tail</code> and <code>NumberOfItems</code> all initialised to <code>0</code>
• One record declared setting ID to <code>""</code> and Quantity to <code>-1</code> ...
• ...stored in all 5 array elements

Example Program code:

Java
<pre>public static SaleData[] CircularQueue = new SaleData[5];
public static Integer NumberOfItems = 0;
public static Integer Head = 0;
public static Integer Tail = 0;

public static void main(String args[]){
      for(Integer X = 0; X < 5; X++){
            CircularQueue[X] = new SaleData("",-1);
      }}</pre>VB.NET
<pre>Dim CircularQueue(0 To 4) As SaleData
Dim NumberOfItems As Integer
Dim Head As Integer
Dim Tail As Integer
Sub Main()
   NumberOfItems = 0
   Head = 0
   Tail = 0
   For x = 0 To 4
      CircularQueue(x).SaleID = ""
      CircularQueue(x).Quantity = -1
   Next
End Sub</pre></td><td>4</td></tr>
<tr><td>2(b)</td><td>Python
<pre>CircularQueue = [] #SaleData, 5 items
global NumberOfItems #int
global Head #int
global Tail #int

#main
NumberOfItems = 0
Head = 0
Tail = 0
for x in range(0, 5):
    CircularQueue.append((SaleData("",-1)))</pre></td><td></td></tr>
</table>

2(c)	1 mark each
	• Function `Enqueue()` header (and end) taking **one** parameter (type `SaleData`)
	• Checks if queue is full …
	• … and returns -1
	• (otherwise) Inserts **parameter** to `CircularQueue[Tail]` …
	• … increments `Tail` **and** resets to 0 if 5
	• Increments number of items **and** returns 1
	
	Example Program code:
	
	Java
	`public static Integer Enqueue(SaleData RecordToAdd){`
	`  if(NumberOfItems == 5){`
	`  return -1;`
	`  }else{`
	`  CircularQueue[Tail].SetSaleID(RecordToAdd.GetSaleID());`
	`  CircularQueue[Tail].SetQuantity(RecordToAdd.GetQuantity());`
	`  if(Tail == 4){`
	`  Tail = 0;`
	`  }else{`
	`  Tail++;`
	`  }`
	`  NumberOfItems++;`
	`  return 1;`
	`  }`
	`}`
	VB.NET
	`Function Enqueue(RecordToAdd)`
	`  If (NumberOfItems = 5) Then`
	`  Return -1`
	`  Else`
	`  CircularQueue(Tail) = RecordToAdd`
	`  If (Tail = 4) Then`
	`  Tail = 0`
		6

2(c)	`  Else`
	`  Tail += 1`
	`  End If`
	`  NumberOfItems += 1`
	`  Return 1`
	`  End If`
	`End Function`
	
	Python
	`def Enqueue(RecordToAdd):`
	`  global NumberOfItems #int`
	`  global Head #int`
	`  global Tail #int`
	`  if(NumberOfItems == 5):`
	`  return -1`
	`  else:`
	`  CircularQueue[Tail] = RecordToAdd`
	`  if(Tail == 4):`
	`  Tail = 0`
	`  else:`
	`  Tail +=1`
	`  NumberOfItems +=1`
	`  return 1`	

2(d)	1 mark each
	• Function header `Dequeue()` (and end where appropriate)
	• Checking if queue is empty…
	• ….and returning appropriate empty/null **record/object/list element**
	• (Otherwise) returning the item at `Head`
	• Incrementing `Head` **and** changing value 0 if it is 4/5
	• Decrement number of items
	
	Example Program code:
	
	Java
	`public static SaleData Dequeue(){`
	`  SaleData RecordRemoved;`
	`  RecordRemoved = new SaleData("", -1);`
	`  if(!(NumberOfItems == 0)){`
	`  RecordRemoved.SetSaleID(CircularQueue[Head].GetSaleID());`
	`  RecordRemoved.SetQuantity(CircularQueue[Head].GetQuantity());`
	`  NumberOfItems--;`
	`  if(Head == 4){`
	`  Head = 0;`
	`  }else{Head++;}`
	`  }`
	`  return RecordRemoved;`
	`}`
	VB.NET
	`Function Dequeue()`
	`  Dim RecordRemoved As SaleData`
	`  RecordRemoved.SaleID = ""`
	`  RecordRemoved.Quantity = -1`
	`  If Not (NumberOfItems = 0) Then`
	`  RecordRemoved = CircularQueue(Head)`
	`  NumberOfItems -= 1`
	`  If Head = 4 Then`
	`  Head = 0`
		6

2(d)	`  Else`
	`  Head += 1`
	`  End If`
	`  End If`
	`  Return RecordRemoved`
	`End Function`
	
	Python
	`def Dequeue():`
	`  global NumberOfItems #int`
	`  global Head #int`
	`  global Tail #int`
	`  RecordRemoved = SaleData("", -1)`
	`  if not(NumberOfItems == 0):`
	
	`  RecordRemoved = CircularQueue[Head]`
	`  NumberOfItems -=1`
	`  if Head == 4:`
	`  Head = 0`
	`  else:`
	`  Head +=1`
	`  return RecordRemoved`	

2(e)	1 mark each
	• Procedure header `EnterRecord` (and end where appropriate) (ignore parameters)
	• Takes as input an ID (string) **and** quantity (integer)
	• Creates a **record/object** using **inputs**
	• Calls `Enqueue()` with **record** as parameter **and** stores/uses return value
	• Outputs `"Full"` and `"Stored"` in correct places
	
	Example Program code:
	
	Java
	`public static void EnterRecord(){`
	`  System.out.println("Enter ID");`
	`  Scanner NewScanner = new Scanner(System.in);`
	`  String ID = NewScanner.nextLine();`
	`  System.out.println("Enter quantity");`
	`  Quan = Integer.parseInt(NewScanner.nextLine());`
	`  SaleData Record;`
	`  Record = new SaleData(ID, Quan);`
	`  if(Enqueue(Record) == -1){ System.out.println("Full");}`
	`  else{System.out.println("Stored");}`
	`}`
		5

VB.NET

```
Sub EnterRecord()
        Dim Record As SaleData
        Console.WriteLine("Enter ID")
        Record.SaleID = Console.ReadLine()
        Console.WriteLine("Enter quantity")
        Record.Quantity = Console.ReadLine()
        If Enqueue(Record) = -1 Then
            Console.WriteLine("Full")
        Else
            Console.WriteLine("Stored")
        End If
    End Sub
```

2(e)

Python

```
def EnterRecord():
    ID = input("Enter ID")
    QuantityP = input("Enter quantity")
    Record = SaleData(ID, QuantityP)
    if Enqueue(Record) == -1:
        print("Full")
    else:
        print("Stored")
```

2(f)(i)

1 mark each to max 4 **4**

- Calling `EnterRecord()` **6 times before** dequeue
- Calling `Dequeue()` **and** storing/using return value …
- … checking if an empty record is returned **and** outputting either the ID and quantity of returned record **or** outputting the error message if empty record
- Calling `EnterRecord()` again **after** dequeue
- Output the ID and quantity for all the records currently stored in `CircularQueue`

Example Program code:

Java

```
EnterRecord();
EnterRecord();
EnterRecord();
EnterRecord();
EnterRecord();
EnterRecord();
SaleData ReturnValue = new SaleData;
ReturnValue = Dequeue();
if(ReturnValue.GetSaleID() == ""){
      System.out.println("No items");
}else{
      System.out.println(ReturnValue.GetSaleID() + " " + ReturnValue. GetQuantity());
}
EnterRecord();
for(Integer X = 0; X < 5; X++){
      System.out.println(CircularQueue[X].GetSaleID() + " " +
CircularQueue[X].GetQuantity());
}
```

VB.NET

```
EnterRecord()
EnterRecord()
EnterRecord()
EnterRecord()
```

2(f)(i)

```
EnterRecord()
EnterRecord()
Dim ReturnValue As SaleData = new SaleData
ReturnValue = Dequeue()

If (ReturnValue.SaleID = "") Then
    Console.WriteLine("No items")
Else
    Console.WriteLine(ReturnValue.SaleID & " " & ReturnValue.Quantity)
End If
EnterRecord()

For x = 0 To 4
    Console.WriteLine(CircularQueue(x).SaleID & " " & CircularQueue(x).Quantity)
Next
```

Python

```
EnterRecord()
EnterRecord()
EnterRecord()
EnterRecord()
EnterRecord()
EnterRecord()
ReturnValue = Dequeue()
if ReturnValue.SaleID == "":
    print("No items")
else:
    print(ReturnValue.SaleID, " ", ReturnValue.Quantity)
EnterRecord()

for x in range(0, 5):
    print(CircularQueue[x].SaleID, " ", CircularQueue[x].Quantity)
```

2(f)(ii)	1 mark for screenshot showing: • Data for 6 records input • 5 messages stating (e.g.) stored and 1 message stating (e.g.) full • 1 output of ADF 10 (dequeued) • Repeat successful input of LLP 3 • Output of the 5 records e.g. Enter IDADF Enter quantity10 Stored Enter IDOOP Enter quantity1 Stored Enter IDBXM Enter quantity5 Stored Enter IDXXL Enter quantity22 Stored Enter IDHQR Enter quantity6 Stored Enter IDLLP Enter quantity3 Full ADF 10 Enter IDLLP Enter quantity3 Stored LLP 3 OOP 1 BXM 5 XXL 22 HQR 6	1

Answer 9

2(a)(i)	**One** mark each
	• (Global) array with identifier `Queue` with (minimum) 50 elements (of type string)
	• `TailPointer` (integer) initialised to 0, `HeadPointer` (integer) initialised to -1
	
	Example program code:
	
	Java
	
	`public static String[] Queue = new String[50];`
	`public static Integer HeadPointer = -1;`
	`public static Integer TailPointer = 0;`
	
	VB.NET
	
	`Dim Queue(50) As String`
	`Dim HeadPointer As Integer`
	`Dim TailPointer As Integer`
	`Sub Main(args As String())`
	`  HeadPointer = -1`
	`  TailPointer = 0`
	`End Sub`
	
	Python
	
	`global Queue #string 50 elements`
	`global HeadPointer`
	`global TailPointer`
	`#main`
	`Queue = []`
	`HeadPointer = -1`
	`TailPointer = 0`
		2

2(a)(ii)	**One** mark each
	• Procedure `Enqueue()` header (and close where appropriate) with **one** (string) parameter
	• Checking if queue is full and outputting suitable message
	• … otherwise inserting parameter to next space
	• … increment `TailPointer` and set `HeadPointer` to 0 if first item (HeadPointer = -1)
	
	Example program code:
	
	Java
	`public static void Enqueue(String Value){`
	`  if(TailPointer == 50){`
	`  System.out.println("Queue full");`
	`  }else{`
	`  Queue[TailPointer] = Value;`
	`  TailPointer++;`
	`  if(HeadPointer == -1){ HeadPointer = 0;}`
	`  }`
	`}`
	
	VB.NET
	
	`Sub Enqueue(Data)`
	`  If TailPointer = 50 Then`
	`  Console.WriteLine("Queue full")`
	`  Else`
	`  Queue(TailPointer) = Data`
	`  TailPointer = TailPointer + 1`
	`  If (HeadPointer = -1) Then`
	`  HeadPointer = 0`
	`  End If`
	`  End If`
	`End Sub`
		4

2(a)(ii)	Python
	
	`def Enqueue(Data):`
	`  global TailPointer`
	`  global HeadPointer`
	`  global Queue`
	`  if TailPointer == 50:`
	`  print("Queue full")`
	`  else:`
	`  Queue.append(Data)`
	`  TailPointer +=1`
	`  if HeadPointer == -1:`
	`  HeadPointer = 0`	

2(a)(iii)	**One** mark each to **max 4**
	• Function header `Dequeue()` (and end where appropriate) with no parameter
	• Checking if empty …
	• … outputting suitable message and returning "Empty"
	• (otherwise) incrementing head pointer
	• returning next value (at head pointer before incrementing)
	
	Example program code:
	
	Java
	
	`public static String Dequeue(){`
	`  if(HeadPointer == -1 || HeadPointer == TailPointer){`
	`  System.out.println("Queue empty");`
	`  return "Empty";`
	`  }else{`
	`  HeadPointer ++;`
	`  return Queue[HeadPointer - 1];}}`
	
	VB.NET
	
	`Function Dequeue()`
	`  If HeadPointer = -1 Or HeadPointer = TailPointer Then`
	`  Console.WriteLine("Queue empty")`
	`  Return "Empty"`
	`  Else`
	`  HeadPointer = HeadPointer + 1`
	`  Return Queue(HeadPointer - 1)`
	`  End If`
	`End Function`
		4

2(a)(iii)	Python
	
	`def Dequeue():`
	`  global Queue`
	`  global HeadPointer`
	`  if HeadPointer == -1 or HeadPointer == TailPointer:`
	`  print("Queue empty")`
	`  return "Empty"`
	`  else:`
	`  HeadPointer +=1`
	`  return Queue[HeadPointer - 1]`	

2(b)	**One** mark each to **max 6**
	• Procedure header `ReadData()` with no parameters
	• Opening file …
	• … and closing file
	• Looping until EOF/set amount
	• Reading in each value
	• … calling `Enqueue()` with each value
	
	• Use of exception handling with appropriate output
	
	Example program code:
	
	Java
	
	`public static void ReadData(){`
	`  try{`
	`  Scanner Scanner1 = new Scanner(new File("QueueData.txt"));`
	`  while(Scanner1.hasNextLine()){`
	`  Enqueue(Scanner1.next());`
	`  }`
	`  Scanner1.close();`
	`  }catch(FileNotFoundException ex){`
	`  System.out.println("No file found");`
	`  }`
	`}`
	
	VB.NET
	
	`Sub ReadData()`
	`  Try`
	`  Dim DataReader As New System.IO.StreamReader("QueueData.txt")`
	`  Do Until DataReader.EndOfStream`
	`  Enqueue(DataReader.ReadLine())`
		6

2(b)	Loop
	DataReader.Close()
	Catch ex As Exception
	Console.WriteLine("No file")
	End Try
	End Sub
	
	Python
	def ReadData():
	try:
	DataFile = open("QueueData.txt")
	for Line in DataFile:
	Enqueue(Line.strip())
	DataFile.close()
	except IOError:
	print("No file")	

2(c)(i)	**One** mark each
	• Declaration of record type/class `RecordData`
	• ID as a string and total as an Integer
	
	Example program code:
	
	Java
	
	class RecordData{
	public String ID;
	public Integer Total;
	public RecordData(String IDP, Integer TotalP){
	ID = IDP;
	Total = TotalP;
	} }
	
	VB.NET
	
	Structure RecordData
	Dim ID As String
	Dim Total As Integer
	End Structure
	
	Python
	
	class RecordData:
	
	#self._ID string
	#self._Total integer
	def _init_(self, IDP, TotalP):
	self._ID = IDP
	self._Total = TotalP
		2

2(c)(i)	def SetID(self, Value):
	self._ID = Value
	
	def GetID(self):
	return self._ID
	
	def SetTotal(self, Value):
	self._Total = Value
	
	def GetTotal(self): return self._Total	

2(c)(ii)	**One** mark each
	• (global) 1D Array named `Records` of type `RecordData`
	• (global) `NumberRecords` declared as integer and initialised to 0
	
	Example program code:
	
	Java
	
	public static RecordData[] Records = new RecordData[50]; public static Integer
	NumberRecords = 0;
	
	VB.NET
	
	Dim Records(49) As RecordData Dim NumberRecords As Integer
	Sub Main(args As String())
	NumberRecords = 0
	End Sub
	
	Python
	
	#main
	Records = [] #50 elements of type RecordData NumberRecords = 0
		2

2(c)(iii)	**One** mark each to **max 5**
	• Incrementing `NumberRecords` each time (twice) a new record is added
	• Procedure header (and end) and using `Dequeue()` and storing/using return value
	`DataAccessed ← Dequeue()`
	• Checking if `NumberRecords` is 0 and creating a new record with ID and total as 1:
	`IF NumberRecords = 0 THEN`
	`Records[NumberRecords].ID ← DataAccessed`
	`Records[NumberRecords].Total ← 1`
	`Flag ← TRUE`
	• Looping through all array elements to find matching ID and incrementing total if found
	`FOR X ← 0 TO NumberRecords - 1` Check Python loop end
	`IF Records[X].ID = DataAccessed THEN`
	`Records[X].Total ← Records[X].Total + 1`
	`Flag ← TRUE`
	`ENDIF`
	`NEXT X`
	• Adding new record if record is not found, storing ID and total as 1
	`IF Flag = FALSE THEN`
	`Records[NumberRecords].ID ← DataAccessed`
	`Records[NumberRecords].Total ← 1`
	`NumberRecords ← NumberRecords + 1`
	`ENDIF`
		5

2(c)(iii)	• Example program code:
	Java
	`public static void TotalData(){`
	`String DataAccessed = Dequeue();`
	`Boolean Flag = false;`
	`if(NumberRecords == 0){`
	`Records[NumberRecords] = new RecordData(DataAccessed, 1);`
	`NumberRecords ++;`
	`Flag = true;`
	`}else{`
	`for(Integer X = 0; X < NumberRecords; X++){`
	`if(Records[X].ID.equals(DataAccessed)){`
	`Records[X].Total++;`
	`Flag = true;`
	`}`
	`}`
	`}`
	`if(Flag == false){`
	`Records[NumberRecords] = new RecordData(DataAccessed, 1);`
	`NumberRecords ++;`
	`}`
	`}`
	VB.NET
	`Sub TotalData()`
	`Dim DataAccessed As String`
	`Dim Flag As Boolean = False`
	`DataAccessed = Dequeue()`	

2(c)(iii)

```
    If NumberRecords = 0 Then
        Records(NumberRecords).ID = DataAccessed
        Records(NumberRecords).Total = Records(NumberRecords).Total + 1
        NumberRecords = NumberRecords + 1
        Flag = True
    Else
        For X = 0 To NumberRecords - 1
        If Records(X).ID = DataAccessed Then
            Records(X).Total = Records(X).Total + 1
            Flag = True

        End If
    Next
    End If
    If Flag = False Then
        Records(NumberRecords).ID = DataAccessed
        Records(NumberRecords).Total = Records(NumberRecords).Total + 1
        NumberRecords = NumberRecords + 1
    End If

End Sub
```

Python

```
def TotalData():
    global NumberRecords
    global Records
    Flag = False
    DataAccessed = Dequeue()


    if NumberRecords == 0:
        Records.append(RecordData(DataAccessed, 1))
        NumberRecords += 1
        Flag = True
    else:
        for X in range(0, NumberRecords):
            if(Records[X].GetID() == DataAccessed):
                Records[X].SetTotal(Records[X].GetTotal() + 1)
                Flag = True
        if Flag == False:
            Records.append(RecordData(DataAccessed, 1))
                NumberRecords += 1
```

2(c)(iii)

2(d)

One mark each

- Looping through all array elements and outputting ID and total in correct format

1

Example program code:

Java

```
public static void OutputRecords(){
    for(Integer X = 0; X < NumberRecords; X++){
        System.out.println("ID ", Records[X].ID + " Total " + Records[X].Total);
    }
}
```

VB.NET

```
Sub OutputRecords()
    For X = 0 To NumberRecords - 1
        Console.WriteLine("ID " & Records(X).ID & " Total " & Records(X).Total)
    Next
End Sub
```

Python

```
def OutputRecords():
    for X in range(0, NumberRecords):
        print("ID", Records[X].GetID(), " Total ", Records[X].GetTotal())
```

2(e)(i)	**One** mark each
	• Calling `ReadData()` first and `OutputRecords()` last
	• Looping through all queue elements and calling `TotalData()` for each queue element
	
	Example program code:
	
	Java
	
	`public static void main(String args[]){`
	`  ReadData();`
	`  while(HeadPointer != TailPointer){`
	`  TotalData();`
	`  }`
	`  OutputRecords();`
	`}`
	
	VB.NET
	
	`Sub Main(args As String())`
	`  HeadPointer = 0`
	`  TailPointer = 0`
	`  ReadData()`
	`  NumberRecords = 0`
	`  While HeadPointer <> TailPointer`
	`  TotalData()`
	`  End While`
	`  OutputRecords()`
	`End Sub`
		2

2(e)(i)	Python
	
	`#main Queue = []`
	`Records = []`
	`HeadPointer = 0`
	`TailPointer = 0`
	`ReadData()`
	`NumberRecords = 0`
	
	`while HeadPointer != TailPointer:`
	`  TotalData()`
	`  OutputRecords()`	

2(e)(ii)	**One** mark for screenshot e.g.
	
	`ID 1234 Total 1`
	`ID 1568 Total 1`
	`ID 9512 Total 2`
	`ID 4567 Total 4`
	`ID 8512 Total 6`
	`ID 4125 Total 3`
	`ID 9651 Total 1`
	`ID 4851 Total 1`
	`ID 2520 Total 2`
	`ID 3265 Total 1`
	`ID 8966 Total 1`
		1

Answer 10

1(a)(i)	**One** mark each
	• Two arrays with correct identifiers of type string/character
	• Each has 100 elements
	
	Example program code:
	
	Java
	
	`public static String[] StackVowel = new String[100];`
	`public static String[] StackConsonant = new String[100];`
	
	VB.NET
	
	`Dim StackVowel(0 To 99) As Char`
	`Dim StackConsonant(0 To 99) As Char`
	
	Python
	
	`StackVowel = [] #string 100`
	`StackConsonant = [] #string 100`
		2

1(a)(ii)	**One** mark for
	• Declaring both variables as type integer global and initialised to 0
	
	Example program code:
	
	Java
	
	`public static Integer VowelTop = 0;`
	`public static Integer ConsonantTop = 0;`
	
	VB.NET
	
	`Dim VowelTop As Integer = 0`
	`Dim ConsonantTop As Integer = 0`
	
	Python
	
	`global VowelTop #integer`
	`global ConsonantTop #integer`
	`#main`
	`VowelTop = 0`
	`ConsonantTop = 0`
		1

1(b)(i)	**One** mark each
	
	• Procedure `PushData()` heading (and end where appropriate) taking one parameter
	
	• Checking if parameter is a (lowercase) vowel …
	• … checking if `StackVowel` is full and outputting suitable message
	• … otherwise inserting parameter in next position
	• … incrementing `VowelTop`
	
	• Repeated for `Consonant`
	
	Example program code:
	
	Java
	
	`public static void PushData(String Letter){`
	`  if(Letter.equals("a") || Letter.equals("e") || Letter.equals("i") || Letter.equals("o") || Letter.equals("u")){`
	`  if(VowelTop == 100){`
	`  System.out.println("Vowel stack full");`
	`  }else{`
	`  StackVowel[VowelTop] = Letter;`
	`  VowelTop++;`
	`  }`
	`  }else{`
	`  if(ConsonantTop == 100){`
	`  System.out.println("Consonant stack full");`
	`  }else{`
	`  StackConsonant[ConsonantTop] = Letter;`
	`  ConsonantTop++;`
	`  }`
	`  }`
	`}`
		6

1(b)(i)

VB.NET

```
Sub PushData(Letter As Char)
    If Letter = "a" Or Letter = "e" Or Letter = "i" Or Letter = "o" Or Letter = "u" Then
        If VowelTop = 100 Then
            Console.WriteLine("Vowel stack full")
        Else
            StackVowel(VowelTop) = Letter
            VowelTop += 1
        End If
    Else
        If ConsonantTop = 100 Then
            Console.WriteLine("Consonant stack full")
        Else
            StackConsonant(ConsonantTop) = Letter
            ConsonantTop += 1
        End If
    End If
End Sub
```

Python

```
def PushData(Letter):
    global VowelTop
    global ConsonantTop
    if Letter == "a" or Letter == "e" or Letter == "i" or Letter == "o" or Letter == "u":
        if VowelTop == 100:
            print("Vowel stack full")
        else:
            StackVowel.append(Letter)
            VowelTop = VowelTop + 1
    else:
        if ConsonantTop == 100:
            print("Consonant stack full")
        else:
            StackConsonant.append(Letter)
            ConsonantTop = ConsonantTop + 1
```

1(b)(ii)

6

One mark each

- Procedure header `ReadData()` with no parameter
- Opening `StackData.txt` to read and closing file
- Looping until EOF // Looping 100 times
- Read each item from the file
- Calling `PushData()` with each value as parameter
- Appropriate exception handling with suitable output

Example program code:

Java

```
    private static void ReadData(){
        try{
            Scanner Scanner1 = new Scanner(new File("StackData.txt"));
            while(Scanner1.hasNextLine()){
                PushData(Scanner1.next());
            }
            Scanner1.close();
        }catch(FileNotFoundException ex){
            System.out.println("No file found");
        }
    }
```

VB.NET

```
    Sub ReadData()
        Try
            Dim DataReader As New System.IO.StreamReader("StackData.txt")

            Do Until DataReader.EndOfStream
                PushData(DataReader.ReadLine())
            Loop
            DataReader.Close()
```

1(b)(ii)	Catch ex As Exception
	Console.WriteLine("File not found")
	End Try
	
	End Sub
	
	Python
	
	def ReadData():
	try:
	DataFile = open("StackData.txt")
	for Line in DataFile:
	PushData(Line.strip())
	DataFile.close()
	except:
	print("File not found")	

1(c)	**One** mark each
	• One function header with no parameter
	
	• Checking if stack is empty and **returning** "No data"
	• …otherwise, decrementing correct pointer
	• Returning value at top of stack
	
	• 2nd function fully correct
	
	Example program code:
	
	Java
	
	public static String PopVowel(){
	String DataToReturn = "";
	if(VowelTop - 1 >= 0){
	VowelTop --;
	DataToReturn = StackVowel[VowelTop];
	return DataToReturn;
	}else{
	return "No data";
	}
	}
	
	public static String PopConsonant(){
	String DataToReturn = "";
	if(ConsonantTop - 1 >= 0){
	ConsonantTop--;
	DataToReturn = StackConsonant[ConsonantTop];
	return DataToReturn;
	}else{
	return "No data";
	}
	}
		5

1(c)	VB.NET
	
	Function PopVowel()
	If VowelTop - 1 >= 0 Then
	VowelTop -= 1
	Dim DataToReturn As Char = StackVowel(VowelTop)
	Return DataToReturn
	Else
	Return "No data"
	End If
	End Function
	
	Function PopConsonant()
	If ConsonantTop - 1 >= 0 Then
	ConsonantTop -= 1
	Dim DataToReturn As Char = StackConsonant(ConsonantTop)
	Return DataToReturn
	Else
	Return "No data"
	End If
	End Function
	
	Python
	
	def PopVowel():
	global VowelTop
	global ConsonantTop
	if VowelTop - 1 >= 0:
	VowelTop = VowelTop - 1
	DataToReturn = StackVowel[VowelTop]
	del StackVowel[-1]
	return DataToReturn
	else:
	return "No data"	

1(c)

```
def PopConsonant():
    global VowelTop
    global ConsonantTop
    if ConsonantTop - 1 >= 0:
        ConsonantTop = ConsonantTop - 1
        DataToReturn = StackConsonant[ConsonantTop]
        del StackConsonant[-1]
        return DataToReturn
    else:
        return "No data"
```

1(d)(i)

One mark each to **max 6**

- Calling `ReadData()`
- Looping until 5 letters **successfully accessed**

- Prompt and read in input of choice …
- … if vowel is input calling `PopVowel()` and if consonant calling `PopConsonant()` …
- … storing return values
- Outputting appropriate message if no vowels and if no consonants (stacks full) **within loop**
- Outputting the **five returned letters** on one line

Example program code:

Java

```
    public static void main(String args[]){
        VowelTop = 0;
        ConsonantTop = 0;
        ReadData();
        String Letters = "";
        Boolean Flag = false;
        String Choice = "";
        String DataAccessed = "";
        for(Integer X = 0; X < 5; X++){
            Flag = false;
            while(Flag == false){
                System.out.println("Vowel or Consonant");
                Scanner scanner = new Scanner(System.in);
                Choice = (scanner.nextLine()).toLowerCase();
                if(Choice.equals("vowel")){
                    DataAccessed = PopVowel();
                    if(DataAccessed.equals("No data") == false){
                        Letters = Letters + DataAccessed;
                        Flag = true;
                    }else{
                        System.out.println("No vowels left");
                    }
```

1(d)(i)

```
                }else if(Choice.equals("consonant")){
                    DataAccessed = PopConsonant();
                    if(DataAccessed.equals("No data") == false){
                        Letters = Letters + DataAccessed;
                        Flag = true;
                    }else{
                        System.out.println("No consonants left");
                    }
                }
            }
        }
        System.out.println(Letters);
    }
```

VB.NET

```
Sub Main(args As String())
    VowelTop = 0
    ConsonantTop = 0
    ReadData()
    Dim Letters As String = ""
    Dim Flag As Boolean = False
    Dim Choice As String
    Dim DataAccessed As String
    For x = 0 To 4
        Flag = False
        While Flag = False
            Console.WriteLine("Vowel or Consonant?")
            Choice = Console.ReadLine().ToLower()
            If Choice = "vowel" Then
                DataAccessed = PopVowel()
                If DataAccessed <> "No data" Then
                    Letters = Letters & DataAccessed
                    Flag = True
                Else
                    Console.WriteLine("No vowels left")
```

6

1(d)(i)	End If
	ElseIf Choice = "consonant" Then
	DataAccessed = PopConsonant()
	If DataAccessed <> "No data" Then
	Letters = Letters & DataAccessed
	Flag = True
	Else
	Console.WriteLine("No consonants left")
	End If
	End If
	End While
	Next
	Console.WriteLine(Letters)
	End Sub
	
	Python
	
	#main
	VowelTop = 0
	ConsonantTop = 0
	ReadData()
	Letters = ""
	Flag = False
	
	for x in range(0, 5):
	Flag = False
	while Flag == False:
	Choice = input("Vowel or Consonant").lower()
	if Choice == "vowel":
	DataAccessed = PopVowel()
	if DataAccessed != "No data":
	Letters = Letters + DataAccessed
	Flag = True
	else:
	print("No vowels left")	

1(d)(i)	elif Choice == "consonant":
	DataAccessed = PopConsonant()
	if DataAccessed != "No data":
	Letters = Letters + DataAccessed
	Flag = True
	else:
	print("No consonants left")
	
	print(Letters)	

1(d)(ii)	**One** mark showing input in order vowel, cons, cons, vowel, vowel. Output is then **utxoe** e.g. Vowel or Consonantvowel Vowel or Consonantconsonant Vowel or Consonantconsonant Vowel or Consonantvowel Vowel or Consonantvowel utxoe	**1**

Answer II

3(a) — **1**

1 mark each

- `QueueData` as 1D (string) array initialised to 20 null values
  and `QueueHead` initialised to `-1`, `QueueTail` initialised to `-1`

e.g.

Java

```
class Queue{
      public static String[] QueueData = new String[20];
      public static Integer QueueHead;
      public static Integer QueueTail;
      public static void main(String args[]){
            for(Integer x = 0; x < 20; x++){
                  QueueData[x] = "";
            }
            QueueHead = -1;
            QueueTail = -1;
      }
}
```

VB.NET

```
 Dim QueueData(0 To 20) As String
  Dim QueueHead As Integer = -1
  Dim QueueTail As Integer = -1
Sub Main(args As String())
    For x = 0 To 19
        QueueData(x) = ""
    Next
End Sub
```

Python

```
global QueueData
global QueueHead
global QueueTail
QueueData = []
for x in range(0, 20):
    QueueData.append("")
QueueHead = -1
QueueTail = -1
```

3(b) — **4**

1 mark each

- Function header (and end) taking **one** parameter **and** returns a Boolean value in all instances
- Checks if queue is full **and** returns `FALSE`
- (If not full) Inserts data item to `QueueTail + 1`
  and increments `QueueTail`
  and returns `TRUE`
- Assigns `QueueHead` to `0` when first element is entered (this can come from incrementing)

e.g.

Java

```
public static Boolean Enqueue(String DataToInsert){
      if(QueueTail == 19){
            return false;
      }else if(QueueHead == -1){
            QueueHead = 0;
      }
      QueueTail = QueueTail + 1;
      QueueData[QueueTail] = DataToInsert.substring(0,6);
      return true;
}
```

VB.NET

```
  Function Enqueue(ByVal DataToInsert)
      If QueueTail = 19 Then
          Return False
      ElseIf QueueHead = -1 Then
          QueueHead = 0
      End If
      QueueTail = QueueTail + 1
      QueueData(QueueTail) = DataToInsert
      Return True
 End Function
```

3(b)

Python

```
def Enqueue(DataToInsert):
    global QueueData
    global QueueHead
    global QueueTail
    if QueueTail == 19:
        return False
    elif QueueHead == -1:
        QueueHead = 0
    QueueTail = QueueTail + 1
    QueueData.append(DataToInsert)
    return True
```

3(c)	1 mark each
	• `Dequeue` function header (and end) returning a **string** in all cases
	• Check if queue is empty
	and return `"false"`
	• (otherwise) remove value at `QueueHead`
	and increment `QueueHead`
	and return value from array
	e.g.
	Java
	`public static String Dequeue(){`
	`  if(QueueHead < 0 || QueueHead > 20 || QueueHead > QueueTail){`
	`  return "false";`
	`  }`
	`  QueueHead++;`
	`  return QueueData[QueueHead-1];`
	`}`
	VB.NET
	`Function Dequeue()`
	`  If QueueHead < 0 Or QueueHead > 20 Or QueueHead > QueueTail Then`
	`  Return "false"`
	`  Else`
	`  QueueHead = QueueHead + 1`
	`  Return QueueData(QueueHead - 1)`
	`  End If`
	`End Function`
		3

3(c)	Python
	`def Dequeue():`
	`  global QueueData`
	`  global QueueHead`
	`  global QueueTail`
	`  if QueueHead < 0 or QueueHead > 20 or QueueHead > QueueTail:`
	`  return False`
	`  else:`
	`  QueueHead = QueueHead + 1`
	`  return QueueData[QueueHead-1]`	

3(d)(i)	1 mark each to max 6
	• `StoreItems` header (function/procedure and end where appropriate)
	and takes 10 inputsi
	• Input is split and first 6 characters used in calculation (as integers) …
	• … multiplication by 1 and 3 alternately, adding to total, dividing by 10, rounding down/cast int …
	• … comparing check digit to character in position 6
	• … including comparison of X for 10
	• Calling `Enqueue` with first 6 characters **when valid**
	• … outputting appropriate message on return (for both inserted and queue full)
	• Counts **and** outputs number of invalid inputs
	e.g.
	Java
	`public static void StoreItems(){`
	`  Integer Count = 0;`
	`  Integer Total = 0;`
	`  String Data;`
	`  Boolean Result;`
	`  Scanner scanner = new Scanner(System.in);`
	`  for(Integer X = 0; X < 10; X++){`
	`  System.out.println("Enter data");`
	`  Data = scanner.nextLine();`
	`  Total = Integer.parseInt(Data.substring(0,1)) +`
		6

3(d)(i)

```
Integer.parseInt(Data.substring(1,2)) * 3 + Integer.parseInt(Data.substring(2,3)) +
Integer.parseInt(Data.substring(3,4)) * 3 + Integer.parseInt(Data.substring(4,5)) +
Integer.parseInt(Data.substring(5,6)) * 3;
          Total = Total / 10;
          if((Total == 10 && Data.substring(6).compareTo("X")==0)){
               Result = Enqueue(Data);
               if(Result == true){
                    System.out.println("Inserted item");
               }else{
                    System.out.println("Queue full");
               }
          }else if(Total == Integer.parseInt(Data.substring(6,7))){
               Result = Enqueue(Data);
               if(Result == true){
                    System.out.println("Inserted item");
               }else{
                    System.out.println("Queue full");
               }
          }else{
               Count = Count + 1;
          }
      }

          System.out.println("There were " + Count + " invalid items");
}
```

VB.NET

```
 Sub StoreItems()

    Dim Count As Integer = 0
    Dim Total As Integer = 0
    Dim Data As String
    Dim Result As Boolean
    For X = 0 To 9
        Console.WriteLine("Enter data")
        Data = Console.ReadLine()
```

3(d)(i)

```
        Total = Integer.Parse(Data.Substring(0, 1)) + Integer.Parse(Data.Substring(1, 1)) *
3 + Integer.Parse(Data.Substring(2, 1)) + Integer.Parse(Data.Substring(3, 1)) * 3 +
Integer.Parse(Data.Substring(4, 1)) + Integer.Parse(Data.Substring(5, 1)) * 3
        Total = Total \ 10
        If (Total = 10 And Data.Substring(6, 1) = "X") Then
            Result = Enqueue(Data.Substring(0, 6))
            If Result = True Then
                Console.WriteLine("Inserted item")
            Else
                Console.WriteLine("Queue full")
            End If
        ElseIf Total = Integer.Parse(Data.Substring(6, 1)) Then
            Result = Enqueue(Data)
            If Result = True Then
                Console.WriteLine("Inserted item")
            Else
                Console.WriteLine("Queue full")
            End If
        Else
            Count = Count + 1
        End If
    Next
    Console.WriteLine("There were " & Count & " invalid items")

End Sub
```

3(d)(i)

Python

```
def StoreItems():
    global QueueData
    global QueueHead
    global QueueTail
    Count = 0
    for X in range(0, 10):
        Data = input("Enter data")
        Total= int(Data[0]) + int(Data[1]) * 3 + int(Data[2]) + int(Data[3]) * 3 +
int(Data[4]) + int(Data[5]) * 3
        Total = int(Total / 10)
        if((Total == 10 and Data[6] == "X") or (Total == int(Data[6]))):
            Result = Enqueue(Data[0:6])

            if(Result == True):
                print("Inserted item")
            else:
                print("Queue full")
        else:
            Count = Count + 1

    print("There were", Count,"Invalid items")
```

3(d)(ii)	• Calling `StoreItems()` **and** `Dequeue()` **once** **and** outputting a suitable message if the queue was empty **and** outputting the returned value if the queue was not empty
	e.g.
	Java
	`public static void main(String args[]){ for(Integer x = 0; x < 20; x++){ QueueData[x] = ""; } QueueHead = -1; QueueTail = -1; StoreItems(); String Value = Dequeue(); if(Value.compareTo("false") == 0){ System.out.println("No data items"); }else{ System.out.println("Item code " + Value); } }`
	VB.NET
	`Sub Main(args As String()) For x = 0 To 19 QueueData(x) = "" Next StoreItems() Dim ReturnValue As String = Dequeue() If (ReturnValue = "false") Then Console.WriteLine("No data items") Else Console.WriteLine("Item code " & ReturnValue) End If End Sub`
		1

3(d)(ii)	Python `QueueData = [] for x in range(0, 20): QueueData.append("") QueueHead = -1 QueueTail = -1 StoreItems() Value = Dequeue() if Value == False: print("No data items") else: print("Item code", Value)`	
3(d)(iii)	1 mark each • Data input of 10 values **and** output a message saying there are 4 invalid items • 999999 output e.g. `Enter data999999X Inserted item Enter data1251484 Inserted item Enter data5500212 Inserted item Enter data0033585 Enter data9845788 Inserted item Enter data6666666 Enter data3258746 Enter data8111022 Inserted item Enter data7568557 Inserted item Enter data0012353 There were 4 Invalid items Item code 999999`	2

Answer 12

2(a)(i)	1 mark each to max 4
	• Class declaration (and end where appropriate) with identifier `Node`
	• `LeftPointer`, `Data` and `RightPointer`, integer
	• Constructor taking 1 parameter (within class) …
	• … assigning parameter to `Data` initialising `LeftPointer` and `RightPointer` to –1
	
	e.g.
	
	Java
	
	`public class Node{`
	`  private Integer LeftPointer;`
	`  private Integer Data;`
	`  private Integer RightPointer;`
	
	`  public Node(Integer PData){`
	`  LeftPointer = -1;`
	`  Data = PData;`
	`  RightPointer = -1;`
	`  }`
	`}`
	
	VB.NET
	
	`Class Node`
	`  Private LeftPointer As Integer`
	`  Private Data As Integer`
	`  Private RightPointer As Integer`
	`  Sub New(PData)`
	`  LeftPointer = -1`
	`  Data = PData`
	`  RightPointer = -1`
	`  End Sub`
	`End Class`
		4

2(a)(i)	Python
	
	`class Node():`
	`  def __init__(self, PData):`
	`  self._LeftPointer = -1 #int`
	`  self._Data = PData #int`
	`  self.__RightPointer = -1 #int`	

2(a)(ii)	1 mark each
	• 1 get method with no parameter…
	• …returning correct attribute
	• Remaining 2 correct (FT minor errors)
	
	e.g.
	
	Java
	
	`public Integer GetLeft(){`
	`  return LeftPointer;`
	`  }`
	`  public Integer GetRight(){`
	`  return RightPointer;`
	`  }`
	`  public Integer GetData(){`
	`  return Data;`
	`  }`
	
	VB.NET
	
	`Function GetLeft()`
	`  Return LeftPointer`
	`End Function`
	`Function GetRight()`
	`  Return RightPointer`
	`End Function`
	`Function GetData()`
	`  Return Data`
	`End Function`
	
	Python
	
	`def GetLeft(self):`
	`  return self._LeftPointer`
		3

2(a)(ii)	`def GetRight(self):`
	`  return self._RightPointer`
	`def GetData(self):`
	`  return self.__Data`	

2(a)(iii)	1 mark each
	• 1 set method with parameter …
	• … assigning to attribute
	• Remaining 2 correct (FT minor errors)
	
	e.g.
	
	Java
	
	`public void SetLeft(Integer NewLeft){`
	`  LeftPointer = NewLeft;`
	`}`
	`public void SetRight(Integer NewRight){`
	`  RightPointer = NewRight;`
	`}`
	`public void SetData(Integer NewData){`
	`  Data = NewData;`
	`}`
	
	VB.NET
	
	`Sub SetLeft(NewLeft)`
	`  LeftPointer = NewLeft`
	`End Sub`
	`Sub SetRight(NewRight)`
	`  RightPointer = NewRight`
	`End Sub`
	`Sub SetData(NewData)`
	`  Data = NewData`
	`End Sub`
	
	Python
	
	`def SetLeft(self, NewLeft):`
	`  self._LeftPointer = NewLeft`
	`def SetRight(self, NewRight):`
		3

2(a)(iii)	`  self._RightPointer = NewRight`
	`def SetData(self, NewData):`
	`  self.__Data = NewData`	

2(b)(i)	1 mark each
	• Class header (and end)
	• Private array `Tree` of type `Node` with 20 elements, private `FirstNode` and private `NumberNodes`
	• Constructor assigns –1 to `FirstNode` and 0 to `NumberNodes`
	• … initialises all `Tree` (20) elements to `Node` object with data value –1
	
	e.g.
	
	Java
	
	`class TreeClass{`
	
	`  private static Node[] Tree = new Node[20];`
	`  private static Integer FirstNode;`
	`  private static Integer NumberNodes;`
	
	`  public TreeClass(){`
	`  FirstNode = -1;`
	`  NumberNodes = 0;`
	`  Integer MinusOne = -1;`
	`  for(Integer x = 0; x < 20; x++){`
	`  Tree[x] = new Node(MinusOne);`
	`  }`
	`  }`
	`}`
	
	VB.NET
	
	`Class TreeClass`
	`  Private Tree(20) As Node`
	`  Private FirstNode As Integer`
	`  Private NumberNodes As Integer`
	
	`  Sub New()`
	`  FirstNode = -1`
	`  NumberNodes = 0`
		4

2(b)(i)

```
        For x = 0 To 19
            Tree(x) = New Node(-1)
        Next
    End Sub
End Class
```

Python

```
class TreeClass():

    def __init__(self):
        self.__Tree = [] #type node 20 spaces
        self._FirstNode = -1 #int
        self._NumberNodes = 0 #int
        for x in range(20):
            self.__Tree.append(Node(-1))
```

2(b)(ii)

1 mark each:

- Method header and end, taking node as parameter **and** checking if empty and inserting in first position, updating FirstNode
- … otherwise inserting node in tree
- Accessing first node and comparing data …
- … checking whether to go left or right …
- … repeatedly until data found
- Updating left and right pointer for parent node

6

e.g.

Java

```
public void InsertNode(Node NewNode){
    Integer NodeAccess;
    Integer Previous = -1;
    String Direction;

    if(NumberNodes == 0){

        Tree[0] = NewNode;
        FirstNode = 0;
        NumberNodes++;
    }else{

        Tree[NumberNodes] = NewNode;
        NodeAccess = FirstNode;
        Direction = "";
        System.out.println(Tree[0].GetData());
        while(NodeAccess != -1){
            Previous = NodeAccess;

            if(NewNode.GetData() < Tree[NodeAccess].GetData()){
                NodeAccess = Tree[NodeAccess].GetLeft();
                Direction = "left";
```

2(b)(ii)

```
            }else if(NewNode.GetData() > Tree[NodeAccess].GetData()){
                NodeAccess = Tree[NodeAccess].GetRight();
                Direction = "right";
            }

        }
        if(Direction.equals("left")){
            Tree[Previous].SetLeft(NumberNodes);
        }else{
            Tree[Previous].SetRight(NumberNodes);
        }
        NumberNodes++;
    }
}
```

VB.NET

```
Sub InsertNode(NewNode)
    Dim NodeAccess As Integer
    Dim Direction As String
    Dim Previous As Integer

    If NumberNodes = 0 Then
        Tree(0) = NewNode
        FirstNode = 0
        NumberNodes += 1
    Else
        Tree(NumberNodes) = NewNode
        NodeAccess = FirstNode
        Direction = ""

        While NodeAccess <> -1
            Previous = NodeAccess
```

2(b)(ii)

```
            If NewNode.GetData() < Tree(NodeAccess).GetData() Then
               NodeAccess = Tree(NodeAccess).GetLeft()
               Direction = "left"
            ElseIf NewNode.GetData() > Tree(NodeAccess).GetData() Then
               NodeAccess = Tree(NodeAccess).GetRight()
               Direction = "right"
            End If
         End While

         If Direction = "left" Then
            Tree(Previous).SetLeft(NumberNodes)
         Else
            Tree(Previous).SetRight(NumberNodes)
         End If
         NumberNodes += 1
      End If
End Sub
```

Python

```
def InsertNode(self, NewNode):

   if(self.__NumberNodes == 0):

      self._Tree[0] = NewNode self._FirstNode = 0
      self._NumberNodes = self._NumberNodes + 1
   else:
      self.__Tree[self.__NumberNodes] = NewNode

      NodeAccess = self._FirstNode
      Direction = ""

      while(NodeAccess != -1):
         Previous = NodeAccess
         if NewNode.GetData() < self.__Tree[NodeAccess].GetData():
```

2(b)(ii)

```
            NodeAccess = self._Tree[NodeAccess].GetLeft()
            Direction = "left"
         elif NewNode.GetData() > self.__Tree[NodeAccess].GetData():

            NodeAccess = self._Tree[NodeAccess].GetRight()
            Direction = "right"

      if(Direction == "left"):

         self._Tree[Previous].SetLeft(self._NumberNodes)
      else:
         self._Tree[Previous].SetRight(self._NumberNodes)
         self._NumberNodes = self._NumberNodes + 1
```

2(b)(iii)

1 mark each

- Procedure header (and end) with no parameter and if no nodes output 'No nodes'
- (otherwise) Loop from index 0 to NumberNodes (or equivalent) …
- … Outputting LeftPointer, Data then RightPointer
- … using get methods

4

e.g.

Java

```
public void OutputTree(){
   if(NumberNodes == 0){
      System.out.println("No nodes");
   }else{
      for(Integer x = 0; x < NumberNodes; x++){
         System.out.println(Tree[x].GetLeft() + " " + Tree[x].GetData() + " " +
Tree[x].GetRight());

      }
   }
}
```

VB.NET

```
Sub OutputTree()
   If NumberNodes = 0 Then
      Console.WriteLine("No nodes")
   Else
      For x = 0 To NumberNodes - 1
         Console.WriteLine(Tree(x).GetLeft() & " " & Tree(x).GetData() & " " &
Tree(x).GetRight())
      Next
   End If
End Sub
```

2(b)(iii)	Python
	
	`def OutputTree(self):`
	`  if self._NumberNodes == 0:`
	`  print("No nodes")`
	`  else:`
	`  for x in range(0, self.__NumberNodes):`
	`  print(self.__Tree[x].GetLeft(), " ", self.__Tree[x].GetData(), " ",self.Tree[x].GetRight())`	

2(c)(i)	1 mark for
	• Instance of `TreeClass` created with identifier `TheTree`
	
	e.g.
	
	Java
	
	`public static void main(String args[]){`
	`  TreeClass TheTree = new TreeClass();`
	`}`
	
	VB.NET
	
	`Sub Main(args As String())`
	`  Dim TheTree As TreeClass = New TreeClass()`
	`End Sub`
	
	Python
	
	`TheTree = TreeClass()`
		1

2(c)(ii)	1 mark each
	• Creating one node with one correct value (e.g. 10)
	• Calling `InsertNode` for `TheTree` with each new `Node`
	• All nodes correctly assigned in order
	• Calling `OutputTree`
	
	e.g.
	
	Java
	
	`public static void main(String args[]){`
	
	`  TreeClass TheTree = new TreeClass();`
	
	`  TheTree.InsertNode(new Node(10));`
	`  TheTree.InsertNode(new Node(11));`
	`  TheTree.InsertNode(new Node(5));`
	`  TheTree.InsertNode(new Node(1));`
	`  TheTree.InsertNode(new Node(20));`
	`  TheTree.InsertNode(new Node(7));`
	`  TheTree.InsertNode(new Node(15));`
	
	`  TheTree.OutputTree();`
	`}`
	
	VB.NET
	
	`Sub Main(args As String())`
	`  Dim TheTree As TreeClass = New TreeClass()`
	`  TheTree.InsertNode(New Node(10))`
	`  TheTree.InsertNode(New Node(11))`
	`  TheTree.InsertNode(New Node(5))`
	`  TheTree.InsertNode(New Node(1))`
	`  TheTree.InsertNode(New Node(20))`
	`  TheTree.InsertNode(New Node(7))`
	`  TheTree.InsertNode(New Node(15))`
	`  TheTree.OutputTree()`
	`End Sub`
		4

2(c)(ii)	Python
	
	`TheTree = TreeClass()`
	`TheTree.InsertNode(Node(10))`
	`TheTree.InsertNode(Node(11))`
	`TheTree.InsertNode(Node(5))`
	`TheTree.InsertNode(Node(1))`
	`TheTree.InsertNode(Node(20))`
	`TheTree.InsertNode(Node(7))`
	`TheTree.InsertNode(Node(15))`
	`TheTree.OutputTree()`	

2(c)(iii)	1 mark for correct output e.g. 2 10 1 -1 11 4 3 5 5 -1 1 -1 6 20 -1 -1 7 -1 -1 15 -1	1

Answer 13

3(a)	1 mark each
	• `LinkedList` declared as **2D** array with (min) 20 × 2 elements (Integer) with all data initialised to `-1`, all nodes linked correctly
	• (Global) `FirstNode` (Int) initialised as `-1` and (global) `FirstEmpty` (Int) initialised as `0`
	
	VB.NET
	`Dim LinkedList(20, 2) As Integer`
	`Dim FirstNode As Integer`
	`Dim FirstEmpty As Integer`
	
	`Sub Main(args As String())`
	`  FirstNode = -1`
	`  FirstEmpty = 0`
	`  For x = 0 To 18`
	`  LinkedList(x, 0) = -1`
	`  LinkedList(x, 1) = x + 1`
	
	`  Next`
	`  LinkedList(19, 0) = -1`
	`  LinkedList(19, 1) = -1`
	`End Sub`
		2

3(a)	Python
	`LinkedList = [] #global`
	`FirstNode = -1`
	`FirstEmpty = 0`
	`for x in range(0, 19):`
	`  LinkedList.append([-1, x + 1])`
	`LinkedList[19][0] = -1`
	`LinkedList[19][1] = -1`
	
	Java
	`private static Integer[][] LinkedList = new Integer[20][2];`
	`private static Integer FirstNode;`
	`private static Integer FirstEmpty;`
	`public static void main(String args[]){`
	`  FirstNode = -1;`
	`  FirstEmpty = 0;`
	`  for(Integer X = 0; X < 19; X++){`
	`  LinkedList[X][0] = -1;`
	`  LinkedList[X][1] = X + 1;`
	`  }`
	`  LinkedList[19][0] = -1;`
	`  LinkedList[19][1] = -1;`
	`}`	

3(b)	1 mark each to max 6
	• Procedure header (and end) taking (min) 5 data items as input from the user
	• Checking if linked list is full (`FirstEmpty = -1`) …
	• …ending procedure/loop/not doing anything further
	
	• (otherwise) `LinkedList[FirstEmpty, 0]` = data input
	• `LinkedList[FirstEmpty, 1] = FirstNode`
	• `FirstNode = FirstEmpty`
	• `FirstEmpty = LinkedList[FirstEmpty, 1]` **before** any update to `FirstEmpty` 's pointer
	
	e.g.
	Python
	`def InsertData():`
	`  global LinkedList`
	`  global FirstNode`
	`  global FirstEmpty`
	`  for _ in range(5):`
	`  if FirstEmpty != -1:`
	`  nextEmpty = LinkedList[FirstEmpty][1]`
	`  LinkedList[FirstEmpty][0] = int(input("Value: "))`
	`  LinkedList[FirstEmpty][1] = FirstNode`
	`  FirstNode = FirstEmpty`
	`  FirstEmpty = nextEmpty`
		6

3(b)	VB.NET
	`Sub InsertData()`
	`    Dim NewItem As Integer`
	`    Dim NextEmpty As Integer`
	
	`    For x = 0 To 4`
	`        Console.WriteLine("Enter the next number")`
	`        NewItem = Console.ReadLine()`
	
	`        If FirstEmpty = -1 Then`
	`            x = 5`
	`        Else`
	`            NextEmpty = LinkedList(FirstEmpty, 1)`
	`            LinkedList(FirstEmpty, 0) = NewItem`
	`            LinkedList(FirstEmpty, 1) = FirstNode`
	`            FirstNode = FirstEmpty`
	`            FirstEmpty = NextEmpty`
	`        End If`
	
	`    Next x`
	`End Sub`	

3(b)	Java
	`public static void InsertData(){`
	`      Integer NewItem;`
	`      Integer CurrentPointer = 0;`
	`      Integer PreviousPointer = 0;`
	`      Scanner scanner = new Scanner(System.in);`
	`      Integer NextEmpty;`
	
	`      for(Integer X = 0; X < 5; X++){`
	`            System.out.println("Enter the next number");`
	`            NewItem = Integer.parseInt(scanner.nextLine());`
	`            if(FirstEmpty == -1){`
	`                  X = 5;`
	`            }else{`
	`                  NextEmpty = LinkedList[FirstEmpty][1];`
	`                  LinkedList[FirstEmpty][0] = NewItem;`
	`                  LinkedList[FirstEmpty][1] = FirstNode;`
	`                  FirstNode = FirstEmpty;`
	`                  FirstEmpty = NextEmpty;`
	`            }`
	`      }`
	`}`	

3(c)(i)	1 mark each
	• Procedure header (and end) starting with node at index `FirstNode` and outputting data `LinkedList[FirstNode,0]`
	• Following pointers until **end** reached and outputting data for each node
	
	Python
	`def OutputLinkedList():`
	`    global LinkedList`
	`    global FirstNode`
	`    global FirstEmpty`
	`    CurrentPointer = FirstNode`
	`    Flag = True`
	`    while Flag:`
	`        print(LinkedList[CurrentPointer][0])`
	`        CurrentPointer = LinkedList[CurrentPointer][1]`
	`        if CurrentPointer == -1:`
	`            Flag = False`
	
	VB.NET
	`Sub OutputLinkedList()`
	
	`        Dim CurrentPointer As Integer = FirstNode`
	`        Dim Flag As Boolean = True`
	`        While Flag`
	`            Console.WriteLine(LinkedList(CurrentPointer, 0))`
	`            CurrentPointer = LinkedList(CurrentPointer, 1)`
	`            If CurrentPointer = -1 Then`
	`                Flag = False`
	`            End If`
	`        End While`
		2

3(c)(i)	End Sub
	
	Java
	public static void OutputLinkedList(){
	Integer CurrentPointer = FirstNode;
	Boolean Flag = true;
	while(Flag){
	System.out.println(LinkedList[CurrentPointer][0]);
	CurrentPointer = LinkedList[CurrentPointer][1];
	if(CurrentPointer == -1){Flag = false;}
	}
	}	

3(c)(ii)	1 mark for calling `InsertData()` then `OutputLinkedList()` Python InsertData() OutputLinkedList() VB.NET InsertData() OutputLinkedList() Java InsertData(); OutputLinkedList();	1
3(c)(iii)	1 mark for inputs of 5 1 2 3 8 and output of 8 3 2 1 5	1

3(d)(i)	1 mark each to max 5
	• Procedure header (and end) with parameter
	• Checking data in `FirstNode` against parameter …
	• … (if found) updating `FirstNode` to `LinkedList[FirstNode, 1]`
	• (Otherwise) following pointers in loop/recursive call …
	• …comparing to data to remove each time
	• … storing previous pointer through each loop…
	• … when found, updating previous pointer to found node's pointer
	• Adding deleted node to end of/start of empty list (and updating `FirstEmpty` if needed)
	
	Python
	def RemoveData(ItemToRemove):
	global LinkedList
	global FirstNode
	global FirstEmpty
	
	if LinkedList[FirstNode][0] == ItemToRemove:
	NewFirst = LinkedList[FirstNode][1]
	LinkedList[FirstNode][1] = FirstEmpty
	FirstEmpty = FirstNode
	FirstNode = NewFirst
	
	else:
	if FirstNode != -1:
	CurrentPointer = FirstNode
	PreviousNode = -1
	while(ItemToRemove != LinkedList[CurrentPointer][0] and CurrentPointer != -1):
	PreviousNode = CurrentPointer
	CurrentPointer = LinkedList[CurrentPointer][1]
	
	if ItemToRemove == LinkedList[CurrentPointer][0]:
	LinkedList[PreviousNode][1] = LinkedList[CurrentPointer][1]
	LinkedList[CurrentPointer][0] = -1
	LinkedList[CurrentPointer][1] = FirstEmpty
	FirstEmpty = CurrentPointer
		5

3(d)(i)

VB.NET

```
Sub RemoveData(ItemToRemove)
    If LinkedList(FirstNode, 0) = ItemToRemove Then
        Dim NewFirst As Integer = LinkedList(FirstNode, 1)
        LinkedList(FirstNode, 1) = FirstEmpty
        FirstEmpty = FirstNode
        FirstNode = NewFirst
    Else
        If FirstNode <> -1 Then
            Dim CurrentPointer As Integer = FirstNode
            Dim PreviousNode As Integer = -1
            Dim Flag As Boolean = True
            Dim Found As Boolean = False
            While Flag And Not (Found)
                If (CurrentPointer <> -1) Then
                    If (ItemToRemove <> LinkedList(CurrentPointer, 0)) Then
                        PreviousNode = CurrentPointer
                        CurrentPointer = LinkedList(CurrentPointer, 1)
                    Else
                        Found = True
                    End If
                Else
                    Flag = False
                End If
            End While

            If Found Then
                LinkedList(PreviousNode, 1) = LinkedList(CurrentPointer, 1)
                LinkedList(CurrentPointer, 0) = -1
                LinkedList(CurrentPointer, 1) = FirstEmpty
                FirstEmpty = CurrentPointer
            End If
        End If
    End If
End Sub
```

3(d)(i)

Java

```
public static void RemoveData(Integer ItemToRemove){
     Integer CurrentPointer = 0;
     Integer PreviousNode = 0;
     Integer NewFirst = 0;

     if(LinkedList[FirstNode][0] == ItemToRemove){
          NewFirst = LinkedList[FirstNode][1];
          LinkedList[FirstNode][1] = FirstEmpty;
          FirstEmpty = FirstNode;
          FirstNode = NewFirst;

     }else{
          if (FirstNode != -1){
               CurrentPointer = FirstNode;
               PreviousNode = -1;
                  while(ItemToRemove != LinkedList[CurrentPointer][0]  && CurrentPointer
!= -1){
                              PreviousNode = CurrentPointer;
                        CurrentPointer = LinkedList[CurrentPointer][1];

               }
               if(ItemToRemove == LinkedList[CurrentPointer][0]){
                     LinkedList[PreviousNode][1] = LinkedList[CurrentPointer][1];
                     LinkedList[CurrentPointer][0] = -1;
                     LinkedList[CurrentPointer][1] = FirstEmpty;
                     FirstEmpty = CurrentPointer;

               }
          }
     }
}
```

3(d)(ii)	1 mark for calling `RemoveData(5)`, outputting `"After"`, calling `OutputLinkedList()`
	
	Python
	`LinkedList = []`
	`FirstNode = -1`
	`FirstEmpty = 0`
	`for x in range(0, 19):`
	`  LinkedList.append([-1, x + 1])`
	`InsertData()`
	`OutputLinkedList()`
	`RemoveData(5)`
	`print("After")`
	`OutputLinkedList()`
	
	VB.NET
	`Sub Main(args As String())`
	`  FirstNode = -1`
	`  FirstEmpty = 0`
	`  For x = 0 To 19`
	`  LinkedList(x, 0) = -1`
	`  LinkedList(x, 1) = x + 1`
	`  Next`
	`  InsertData()`
	`  OutputLinkedList()`
	`  RemoveData(5)`
	`  Console.WriteLine("After")`
	`  OutputLinkedList()`
	`End Sub`
		1

3(d)(ii)	Java
	`public static void main(String args[]){`
	`  FirstNode = -1;`
	`  FirstEmpty = 0;`
	`  for(Integer X = 0; X < 20; X++){`
	`  LinkedList[X][0] = -1;`
	`  LinkedList[X][1] = X + 1;`
	`  }`
	`  InsertData();`
	`  OutputLinkedList();`
	
	`  RemoveData(5);`
	`  System.out.println("After");`
	`  OutputLinkedList();`
	`}`	

3(d)(iii)	1 mark for input and output. Test data 1: Input 5 6 8 9 5 'After' Output: 9 8 6 5 Test data 2: Input 10 7 8 5 6 "After" Output: 6 8 7 10	1

Answer 14

2(a) — **3**

1 mark each to max
- Record structure or class with constructor `Queue` (and end where appropriate) …
- … containing a 1D array of (100) integers `QueueArray`
- … containing `HeadPointer` and `TailPointer` as integers

e.g.
Python
```
class Queue:
    def __init__(self):
        self.QueueArray = []
        HeadPointer = 0 #integer
        TailPointer = 0 #integer
        for x in range(0, 100):
            self.QueueArray.append(-1)
```

VB.NET
```
Structure Queue
    Dim QueueArray() As Integer
    Dim HeadPointer As Integer
    Dim TailPointer As Integer
End Structure
```

Java
```
class queue{
     private static Integer[] QueueArray = new Integer[100];
     private static Integer HeadPointer;
     private static Integer TailPointer;

     public queue(){
     }
}
```

2(b) — **3**

1 mark each
- New `Queue` record/object created/instance of class
- Queue **field/attribute** head pointer initialised to `-1`, tail pointer to `0`
- All 100 array **field/attribute** elements initialised with `-1`

e.g.
Python
```
class Queue:
    def __init__(self):
        self.QueueArray = []
        for x in range(0, 100):
            self.QueueArray.append(-1)

        self.HeadPointer = -1
        self.TailPointer = 0
TheQueue= Queue()
```

VB.NET
```
 Dim TheQueue As New Queue
        TheQueue.HeadPointer = -1
        TheQueue.TailPointer = 0
        ReDim TheQueue.QueueArray(100)
        For x = 0 To 99
            TheQueue.QueueArray(x) = -1
        Next
```

Java
```
class queue{
     private static Integer[] QueueArray = new Integer[100];
     private static Integer HeadPointer;
     private static Integer TailPointer;
```

2(b)
```
     public queue(){
          HeadPointer = -1;
          TailPointer = 0;
          for(Integer x = 0; x < 100; x++){
               QueueArray[x] = -1;
          }
     }
}
public static void main(String args[]){
     queue TheQueue = new queue();
}
```

2(c)	1 mark for each completed statement to max 3
	1 mark for correct values returned in correct places
	1 mark for function header taking (at least) one parameter and the rest of function correct and using the record/class data structure accurately.
	
	Pseudocode
	`FUNCTION Enqueue(BYREF AQueue : Queue, BYVAL TheData : INTEGER)`
	`  RETURNS INTEGER`
	`  IF AQueue.HeadPointer = -1 THEN`
	`  AQueue.QueueArray[AQueue.TailPointer] ← TheData`
	`  AQueue.HeadPointer ← 0`
	`  AQueue.TailPointer ← AQueue.TailPointer + 1`
	`  RETURN 1`
	`  ELSE`
	`  IF AQueue.TailPointer > 99 THEN`
	`  RETURN -1`
	`  ELSE`
	`  AQueue.QueueArray[AQueue.TailPointer] ← TheData`
	`  AQueue.TailPointer ← AQueue.TailPointer + 1`
	`  RETURN 1`
	`  ENDIF`
	`  ENDIF`
	`ENDFUNCTION`
		5

2(c)	e.g.
	Python
	`def Enqueue(AQueue, TheData):`
	
	`  if AQueue.HeadPointer == -1:`
	`  AQueue.HeadPointer = 0`
	`  AQueue.QueueArray[AQueue.HeadPointer] = TheData`
	`  AQueue.TailPointer +=1`
	`  return AQueue, 1`
	`  elif AQueue.TailPointer > 99:`
	`  return AQueue, -1`
	`  else:`
	`  AQueue.QueueArray[AQueue.TailPointer] = TheData`
	`  AQueue.TailPointer = AQueue.TailPointer + 1`
	`  return AQueue, 1`
	
	VB.NET
	`Function Enqueue(ByRef AQueue As Queue, ByVal TheData As Integer)`
	`  If AQueue.HeadPointer = -1 Then`
	`  AQueue.QueueArray(AQueue.TailPointer) = TheData`
	`  AQueue.HeadPointer = 0`
	`  AQueue.TailPointer += 1`
	`  Return 1`
	`  ElseIf AQueue.TailPointer > 99 Then`
	`  Return -1`
	`  Else`
	`  AQueue.QueueArray(AQueue.TailPointer) = TheData`
	`  AQueue.TailPointer += 1`
	`  Return 1`
	`  End If`
	
	`End Function`	

2(c)	Java
	`public static Integer Enqueue(Integer TheData){`
	`  if(GetHeadPointer() == -1){`
	`  SetData(TheData);`
	`  SetHeadPointer(0);`
	`  SetTailPointer(GetTailPointer() + 1);`
	`  return 1;`
	`  }else if(GetTailPointer() > 99){`
	`  return -1;`
	`  }else{`
	`  SetData(TheData);`
	`  SetTailPointer(GetTailPointer() + 1);`
	`  return 1;`
	`  }`
	`}`	

2(d)	1 mark each to max 3
	• Function header (and end) iterating through each element in the queue
	• Starting at `HeadPointer` and incrementing until `TailPointer - 1` ...
	• ... concatenating and returning all integer values with a space between
	
	e.g.
	Python
	
	`def ReturnAllData(TheQueue):`
	`  Temp = ""`
	
	`  for X in range(TheQueue.HeadPointer, TheQueue.TailPointer):`
	`  Temp = Temp + str(TheQueue.QueueArray[X]) + " "`
	`  return Temp`
		3

2(d)	VB.NET
	`Function ReturnAllData(AQueue As Queue)`
	`  Dim Temp As String = ""`
	`  For X = AQueue.HeadPointer To AQueue.TailPointer - 1`
	`  Temp = Temp & AQueue.QueueArray(X).ToString() & " "`
	`  Next X`
	`  Return Temp`
	`End Function`
	
	Java
	`public static String ReturnAllData(){`
	`  String Temp = "";`
	`  Integer Counter = 0;`
	`  for(int X = HeadPointer; X < TailPointer; X++){`
	`  Temp = Temp + Integer.toString(QueueArray[X]) + " ";`
	`  }`
	`  return Temp;`
	`}`	

2(e)(i)	1 mark each • Taking only **10** inputs in loop/one at a time • Calling `Enqueue()` with each input (min) **and** storing/using return value … • … only calling `Enqueue()` **once** when each input is an integer > = 0. Do **not** award if this validation stops 10 valid inputs being enqueued. • Outputting message if each item is inserted **and** outputting a message if queue is full • Calling `ReturnAllData()` and outputting return value at the end	**5**

2(e)(i)	e.g.
	Python
	`for x in range(0, 10):`
	`  Continue = True`
	`  while(Continue == True):`
	`  DataInput = int(input("Enter an integer that is 0 or more"))`
	`  if DataInput > -1:`
	`  Continue = False`
	
	`  TheQueue, ReturnValue = Enqueue(TheQueue, DataInput)`
	
	`  if(ReturnValue == -1):`
	`  print("Queue full")`
	`  else:`
	`  print("Item inserted")`
	
	`print(ReturnAllData(TheQueue))`
	
	VB.NET
	`Dim ContinueLoop As Boolean`
	`Dim DataInput As Integer`
	`Dim ReturnValue As Integer`
	`For x = 0 To 9`
	`  ContinueLoop = True`
	`  While ContinueLoop = True`
	`  Console.WriteLine("Enter an integer that is 0 or more")`
	`  DataInput = Console.ReadLine`
	`  If DataInput > -1 Then`
	`  ContinueLoop = False`	

2(e)(i)

```
        End If
    End While
    ReturnValue = Enqueue(TheQueue, DataInput)
    If ReturnValue = 2 Then
        Console.WriteLine("Queue full")
    Else
        Console.WriteLine("Item inserted")
    End If
Next

Console.WriteLine(ReturnAllData(TheQueue))
```

Java

```
Boolean Continue = true;
Integer DataInput = -1;
Scanner scanner = new Scanner(System.in);
Integer ReturnValue;

for(Integer x = 0; x < 10; x++){
     Continue = true;
     while(Continue == true){
          System.out.println("Enter an integer that is 0 or more");
          DataInput = Integer.parseInt(scanner.nextLine());
          if(DataInput > -1){
               Continue = false;
          }
     }
     ReturnValue = Enqueue(DataInput);
     if(ReturnValue == -1){
          System.out.println("Queue full");
     }else{
          System.out.println("Item inserted");
     }
}
System.out.println(ReturnAllData());
```

2(e)(ii) — 2

1 mark for each

- All values input and 10 messages 'Inserted' (i.e. –1 is not inserted)
- Screenshot show 10 9 8 7 6 5 4 3 2 1 on one line with a space between each number

e.g.

```
Enter an integer that is 0 or more10
Item inserted
Enter an integer that is 0 or more9
Item inserted
Enter an integer that is 0 or more-1
Enter an integer that is 0 or more8
Item inserted
Enter an integer that is 0 or more7
Item inserted
Enter an integer that is 0 or more6
Item inserted
Enter an integer that is 0 or more5
Item inserted
Enter an integer that is 0 or more4
Item inserted
Enter an integer that is 0 or more3
Item inserted
Enter an integer that is 0 or more2
Item inserted
Enter an integer that is 0 or more1
Item inserted
10 9 8 7 6 5 4 3 2 1
```

2(f) — 4

1 mark each

- Function `Dequeue()` (head and close), returning a value in all cases
- Checking if empty **and** returning –1
- Returning item at `HeadPointer` **without deleting**/changing it
- Incrementing `HeadPointer`

Example program code:

Python

```
def Dequeue(AQueue):
    if AQueue.HeadPointer = 100 or AQueue.HeadPointer == -1 or AQueue.HeadPointer ==
AQueue.TailPointer:
        return AQueue, -1
    else:
        Temp = AQueue.QueueArray[AQueue.HeadPointer]
        AQueue.HeadPointer = AQueue.HeadPointer + 1
        return AQueue, Temp
```

2(f)	VB.NET
	`Function Dequeue(ByRef AQueue As Queue)`
	`If AQueue.HeadPointer = 100 or AQueue.HeadPointer = -1 or AQueue.HeadPointer = AQueue.TailPointer Then`
	`Return -1`
	`Else`
	`Dim Temp As Integer = AQueue.QueueArray(AQueue.HeadPointer)`
	`AQueue.HeadPointer += 1`
	`Return Temp`
	`End If`
	`End Function`
	
	Java
	`public static Integer Dequeue(){`
	`if(GetHeadPointer() = 100 || GetTailpointer() == -1 || GetHeadPointer() == GetTailpointer()){`
	`return -1;`
	`}else{`
	`Integer Temp = GetData(GetHeadPointer());`
	`SetHeadPointer(GetHeadPointer() + 1);`
	`return Temp;`
	`}`
	`}`	

2(g)(i)	1 mark each
	• Calls `Dequeue()` **twice and** stores/uses return value
	• Outputs "Queue empty" when each return values is –1 and outputs return value otherwise
	• Calls `ReturnAllData()` at the end
	
	e.g.
	Python
	`TheQueue, ReturnValue = Dequeue(TheQueue)`
	`if ReturnValue == -1:`
	`print("Queue empty")`
	`else:`
	`print(ReturnValue, " is returned")`
	`TheQueue, ReturnValue = Dequeue(TheQueue)`
	`if ReturnValue == -1:`
	`print("Queue empty")`
	`else:`
	`print(ReturnValue, " is returned")`
	`print(ReturnAllData(TheQueue))`
	
	VB.NET
	`ReturnValue = Dequeue(TheQueue)`
	`If ReturnValue = -1 Then`
	`Console.WriteLine("Queue empty")`
	`Else`
	`Console.WriteLine(ReturnValue, " is returned")`
	`End If`
		3

2(g)(i)	`ReturnValue = Dequeue(TheQueue)`
	`If ReturnValue = -1 Then`
	`Console.WriteLine("Queue empty")`
	`Else`
	`Console.WriteLine(ReturnValue, " is returned")`
	`End If`
	`Console.WriteLine(ReturnAllData(TheQueue))`
	
	Java
	`ReturnValue = Dequeue();`
	`if(ReturnValue == -1){`
	`System.out.println("Queue empty");`
	`}else{`
	`System.out.println(ReturnValue + " is returned");`
	`}`
	
	`ReturnValue = Dequeue();`
	`if(ReturnValue == -1){`
	`System.out.println("Queue empty");`
	`}else{`
	`System.out.println(ReturnValue + " is returned");`
	`}`
	`ReturnAllData(TheQueue);`	

2(g)(ii)	1 mark each • Screenshot shows the input of 10 9 8 7 6 5 4 3 2 1 Output for 10 returned Output for 9 is returned e.g. Enter an integer that is 0 or more10 Item inserted Enter an integer that is 0 or more9 Item inserted Enter an integer that is 0 or more-1 Enter an integer that is 0 or more8 Item inserted Enter an integer that is 0 or more7 Item inserted Enter an integer that is 0 or more6 Item inserted Enter an integer that is 0 or more5 Item inserted Enter an integer that is 0 or more4 Item inserted Enter an integer that is 0 or more3 Item inserted Enter an integer that is 0 or more2 Item inserted Enter an integer that is 0 or more1 Item inserted 10 9 8 7 6 5 4 3 2 1 10 is returned 9 is returned 8 7 6 5 4 3 2 1	1