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]