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]