Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/43**

Paper 4 Practical **May/June 2025**

2 hours 30 minutes

0885458016

You will need: Candidate source files (listed on page 2)
evidence.doc

INSTRUCTIONS

- Carry out every instruction in each task.
- Save your work using the file names given in the task as and when instructed.
- You must **not** have access to either the internet or any email system during this examination.
- You must save your work in the evidence document as stated in the tasks. If work is **not** saved in the evidence document, you will **not** receive marks for that task.
- You must use a high-level programming language from this list:
  - Java (console mode)
  - Python (console mode)
  - Visual Basic (console mode)
- A mark of **zero** will be awarded if a programming language other than those listed here is used.

INFORMATION

- The total mark for this paper is 75.
- The number of marks for each question or part question is shown in brackets [ ].

DC (WW) 342932/1

You have been supplied with the following source files:

`QueueData.txt`

Open the evidence document, **evidence.doc**

Make sure that your name, centre number and candidate number will appear on every page of this document. This document must contain your answers to each question.

Save this evidence document in your work area as:
evidence_ followed by your centre number_candidate number, for example: **evidence_zz999_9999**

A class declaration can be used to declare a record. If the programming language used does not support arrays, a list can be used instead.

One source file is used to answer **Question 1**. The file is called **`QueueData.txt`**

1 A program reads data from a text file and stores it in a queue. The linear queue `Queue` is stored as a 1D array of up to 50 elements. The queue has the following pointers:

- `HeadPointer` – this stores the index of the first element in the queue, initialised to `-1`
- `TailPointer` – this stores the index of the last element in the queue, initialised to `-1`

(a) `Queue` is a global array of 50 integers with all elements initialised to `-1` in the main program.

The **two** pointers are declared as global variables and initialised to `-1`

Write program code to declare and initialise `Queue`, `HeadPointer` and `TailPointer`

Save your program as **Question1_J25**. Copy and paste the program code into part **1(a)** in the evidence document.

[3]

(b) The function `Enqueue()`:

- takes an integer parameter to store in the next position in the queue
- returns `FALSE` if the queue is full and the parameter cannot be stored in the queue
- returns `TRUE` if the parameter is stored in the queue
- updates the pointers where appropriate.

Write program code for `Enqueue()`

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

[6]

(c) The function `Dequeue()`:

- returns the next item in the queue, if the queue is **not** empty
- returns `-1` if the queue is empty
- updates the pointers where appropriate.

Write program code for `Dequeue()`

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

[5]

(d) The text file `QueueData.txt` stores positive integers. Each integer is on a new line in the file.

The procedure `CreateQueue()`:

- opens the file `QueueData.txt`
- reads in each line from the text file and uses `Enqueue()` to insert each line into the queue
- outputs `"Queue full"` if any item cannot be inserted into the queue
- uses exception handling when opening and reading from the text file.

The procedure needs to work for a file that contains an unknown number of lines.

Write program code for `CreateQueue()`

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

[6]

(e) (i) The main program needs extending to call `CreateQueue()`. The main program then adds together all of the integers stored in the queue, using `Dequeue()` to access each integer. The total is then output.

Write program code to extend the main program.

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

[5]

(ii) Test your program.

Take a screenshot of the output(s).

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

[1]

2 A program sorts the data in the 1D array `DataArray` and searches `DataArray` for specific values.

`DataArray` stores 14 integer values and is declared local to the main program.

(a) Write program code to create `DataArray` and initialise it with the following data values in the order they are written:

```
0   3   4   56   67   44   43   32   31   345   45   6   54   1
```

Save your program as **Question2_J25**.

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

[1]

(b) The function `InsertionSort()` takes an array of integers as a parameter. The function sorts the data in the array into ascending numerical order using an insertion sort. The function returns the sorted array.

Write program code for `InsertionSort()`

You must **not** use any inbuilt sorting functions for your programming language.

Save your program.

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

[5]

(c) The procedure `OutputArray()` takes an array of integers as a parameter. The procedure outputs each element in the array from the first element to the last element. The output is on one line with a space between each number.

An example output is:

```
"0 3 4 56 67 44 43 32 31 345 45 6 54 1"
```

Write program code for `OutputArray()`

Save your program.

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

[2]

(d) (i) The main program needs extending to:

- output the content of the unsorted array using `OutputArray()`
- sort the array using `InsertionSort()`
- output the content of the sorted array using `OutputArray()`

Write program code to extend the main program.

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

[2]

(ii) Test your program.

Take a screenshot of the output(s).

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

[1]

(e) The function `Search()` performs a binary search to find `ItemToFind` in `DataArray`

The function takes **two** parameters:

- `DataArray`, an array of integers
- `ItemToFind`, an integer to find in `DataArray`

The function returns:

- the index of `ItemToFind` if it is in `DataArray`
- `-1` if `ItemToFind` is **not** in `DataArray`

Write program code for `Search()`

You must **not** use any inbuilt searching functions for your programming language.

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

[6]

(f) **(i)** The main program needs extending to call `Search()` with the sorted array **four** times:

- the first time to find the index of the integer 0
- the second time to find the index of the integer 345
- the third time to find the index of the integer 67
- the fourth time to find the index of the integer 2

If the integer is found in the array, output an appropriate message that includes the index. If the integer is **not** found, output that it was **not** found.

Write program code to extend the main program.

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

[2]

(ii) Test your program.

Take a screenshot of the output(s).

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

[1]

3 A program stores data in a linked list that is designed using Object-Oriented Programming (OOP).

The class `Node` stores data about the nodes.

Node	
`TheData : Integer`	stores the integer data
`NextNode : Node`	stores the next node in the linked list
`Constructor()`	initialises `TheData` to its parameter value, initialises `NextNode` to a null value
`GetData()`	returns `TheData`
`GetNextNode()`	returns `NextNode`
`SetNextNode()`	takes an object of type `Node` as a parameter and stores it in `NextNode`

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

Do **not** declare the other methods.

Use your programming language appropriate constructor.

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

Save your program as **Question3_J25**.

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

[4]

(ii) Write program code for the **two** get methods.

Save your program.

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

[3]

(iii) The method `SetNextNode()` takes an object of type `Node` as a parameter. The method stores the parameter in the attribute `NextNode`

Write program code for `SetNextNode()`

Save your program.

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

[2]

(b) The class `LinkedList` stores the linked list.

LinkedList	
`HeadNode : Node`	stores the first node in the linked list
`Constructor()`	initialises `HeadNode` to a null value
`InsertNode()`	creates a new node using its integer parameter. Sets this node as the new head node and updates `NextNode`
`RemoveNode()`	finds the first node that contains its integer parameter and removes this node from the linked list
`Traverse()`	concatenates and returns the integer data in each node in the linked list

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

Do **not** declare the other methods.

Use your programming language appropriate constructor.

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

Save your program.

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

[2]

(ii) The method `InsertNode()`:

- takes an integer as a parameter
- creates a new node with the parameter as the integer data
- uses the new node's method `SetNextNode()` to store the current `HeadNode` as the next node
- replaces `HeadNode` with the current node.

Write program code for `InsertNode()`

Save your program.

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

[4]

(iii) The method `Traverse()` concatenates the integer data from the nodes in the linked list, starting with the node stored in `HeadNode`. The method returns the final string with each integer data separated with a space.

Write program code for `Traverse()`

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

[3]

(iv) The method `RemoveNode()` takes an integer parameter to search for and remove from the linked list.

The method first checks if the linked list is empty. If the linked list is empty the method returns `FALSE`

If the linked list is **not** empty, the integer data in `HeadNode` is compared to the parameter. If it matches the parameter, `HeadNode` is changed to store the next node.

If the parameter does **not** match, the nodes are followed until either:

- the node with matching integer data is found. This node is removed, the appropriate nodes updated and `TRUE` returned
  or
- none of the nodes contain matching integer data to the parameter. No nodes are removed and `FALSE` is returned.

Write program code for `RemoveNode()`

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

[6]

(c) (i) The main program creates a new `LinkedList` object and uses the appropriate method to insert **five** nodes with the following integer data values in the order given:

```
10      20      30      40      50
```

The main program then:

- calls `Traverse()`
- removes the node containing the integer data `30` using `RemoveNode()`
- calls `Traverse()`

Write program code for the main program.

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

[3]

(ii) Test your program.

Take a screenshot of the output(s).

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

[2]

BLANK PAGE

BLANK PAGE