Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/41**

Paper 4 Practical **May/June 2025**

2 hours 30 minutes

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 [ ].

This document has **12** pages. Any blank pages are indicated.

You have been supplied with the following source files:

TheData.txt
Blue.txt
Green.txt
Orange.txt
Pink.txt
Red.txt
Yellow.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.

Seven source files are used to answer **Question 2**. The files are called **TheData.txt**, **Blue.txt**, **Green.txt**, **Orange.txt**, **Pink.txt**, **Red.txt** and **Yellow.txt**

1 A program stores positive integers in a circular queue.

The queue is stored as a global 1D array of 20 integers with the identifier `Queue`. Each index is initialised with the data `-1`

The global variable `HeadPointer`, initialised to `-1`, points to the first element in the queue.
The global variable `TailPointer`, initialised to `-1`, points to the last element in the queue.
The global variable `NumberItems`, initialised to `0`, stores the number of items in the queue.

(a) Write program code to declare and initialise `Queue`, `HeadPointer`, `TailPointer` and `NumberItems`

Save your program as **Question1_J25**.

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

[2]

(b) The function `Enqueue()`:

- takes an integer as a parameter
- checks if the queue is full
- returns `FALSE` if the queue is full
- stores the parameter in the next position in the queue and returns `TRUE` if the queue is **not** full
- updates the appropriate pointers and `NumberItems`

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 main program:

- attempts to store each of the integers 1 to 25 (inclusive) in the queue in ascending numerical order using `Enqueue()`
- outputs the integer that was passed to `Enqueue()` and `"Successful"` if it was stored in the queue, or `"Unsuccessful"` if it was **not** stored in the queue.
  For example:
  - if the integer 5 is passed to `Enqueue()` and is stored in the queue, the output will be: `"5 Successful"`
  - if the integer 23 is passed to `Enqueue()` and is **not** stored in the queue, the output will be: `"23 Unsuccessful"`

Write program code for the main program.

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

[3]

(d) The function `Dequeue()` returns `-1` if the queue is empty. If the queue is **not** empty, the function returns the next item in the queue, updates the appropriate pointers and updates `NumberItems`

Write program code for `Dequeue()`

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

[6]

(e) (i) Write program code to extend the main program to call `Dequeue()` twice and output the return value each time.

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

[1]

2 A program reads data from a text file, splits the data depending on its content and stores the separated data into different files.

The text file `TheData.txt` contains 72 lines of data. Each line of data has an integer number and a string colour that are separated by a comma. For example, the first line in the file is:

```
10,red
```

The integer is `10` and the string is `"red"`

The file contains six different colours: red, green, blue, orange, yellow, pink.

(a) The function `ReadData()`:

- prompts the user to enter a filename and reads this filename from the user
- opens the file and reads each line of data into a 1D array
- returns the populated 1D array.

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

Write program code for `ReadData()`

Save your program as **Question2_J25**. Copy and paste the program code into part **2(a)** in the evidence document.

[7]

(b) The procedure `SplitData()` takes a 1D string array as a parameter with the identifier `DataArray`

The procedure declares six 1D arrays: one array for each colour that appears in the file (red, green, blue, orange, yellow, pink).

The procedure accesses each string in `DataArray`. The data in each string is split into the integer and the colour. The integer is stored in the array that matches the colour.

For example, the first string in `DataArray` has the integer `10` and the colour `red`, so the integer `10` is stored in the array for the colour red.

Write program code for `SplitData()`

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

[6]

(c) The procedure `StoreData()`:

- takes **two** parameters: a 1D array `DataToStore` and a filename
- opens the text file with the filename that is passed as a parameter
- appends each item of data from `DataToStore` to a new line in the text file
- uses exception handling when opening and writing data to the text file.

Write program code for `StoreData()`

Save your program.

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

[5]

(d) Each of the six colours has a blank text file where the numbers will be stored. The names of these six text files are:

- `Blue.txt`
- `Green.txt`
- `Orange.txt`
- `Pink.txt`
- `Red.txt`
- `Yellow.txt`

The procedure `SplitData()` needs amending to call `StoreData()` **six** times, with each of the **six** colour arrays and the name of the text file that corresponds to that colour.

For example, `StoreData()` will be called with the red array and the file name `"Red.txt"`

Write program code to amend `SplitData()`

Save your program.

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

[2]

(e) The main program calls `ReadData()` and then `SplitData()`

(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.

[3]

(ii) Test your program. Input the text `"TheData.txt"` when prompted.

Take a screenshot of the output(s) and a screenshot showing the content of the file that stores the red numbers.

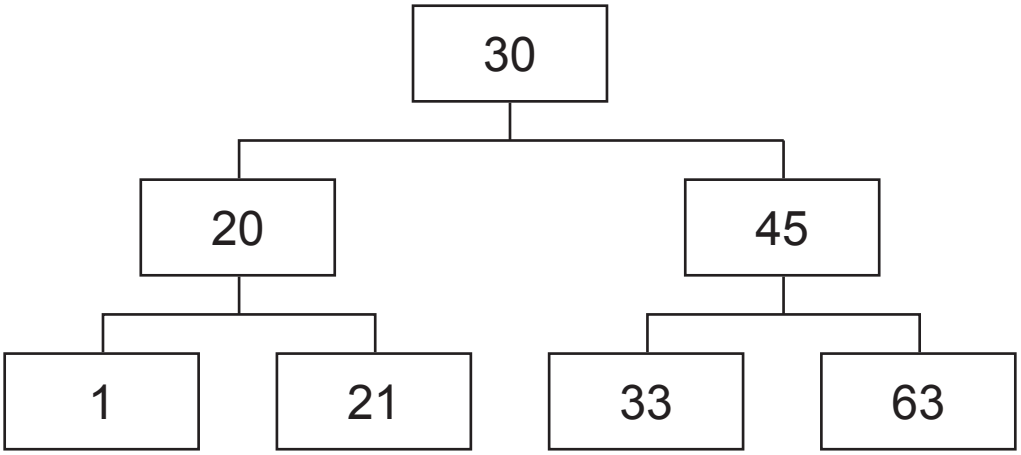
Save your program.

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

[2]

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

The tree stores data in ascending numerical order, for example:



The class `Node` stores data about the nodes.

Node	
`NodeData : Integer`	stores the node's integer data
`LeftNode : Node`	stores the node that is stored to the left of the current node, or a null value if there is no node to the left
`RightNode : Node`	stores the node that is stored to the right of the current node, or a null value if there is no node to the right
`Constructor()`	initialises `NodeData` to its parameter value; initialises `LeftNode` and `RightNode` to a null value
`GetLeft()`	returns `LeftNode`
`GetRight()`	returns `RightNode`
`GetData()`	returns `NodeData`
`SetLeft()`	takes an object of type `Node` as a parameter and stores it in `LeftNode`
`SetRight()`	takes an object of type `Node` as a parameter and stores it in `RightNode`

(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 **three** get methods.

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

[3]

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

The method `SetRight()` takes an object of type `Node` as a parameter. The method stores the parameter in the attribute `RightNode`

Write program code for `SetLeft()` and `SetRight()`

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

[3]

(b) Write the main program to declare **five** objects of type `Node` :

- Node 1 with the data `10`
- Node 2 with the data `20`
- Node 3 with the data `5`
- Node 4 with the data `15`
- Node 5 with the data `7`

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

[2]

(c) The class `Tree` stores the tree.

Tree	
`FirstNode : Node`	stores the root node in the tree
`Constructor()`	initialises `FirstNode` to its parameter value
`GetRootNode()`	returns the node stored in `FirstNode`
`Insert()`	stores its parameter node in the correct position in the tree

(i) Write program code to declare the class `Tree` 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(c)(i)** in the evidence document.

[2]

(ii) Write program code for `GetRootNode()`

Save your program.

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

[1]

(iii) The method `Insert()` takes a node as a parameter and then searches the tree to find the position to insert the new node by:

- checking whether the node's data is less than or greater than the root node's data
- moving to the left node if the data is less than the root node's data
- moving to the right node if the data is greater than or equal to the root node's data
- repeating until the final position of the node is found and stores the node in that position.

Write program code for `Insert()`

Save your program.

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

[6]

(d) The recursive procedure `OutputInOrder()` outputs the data stored in the binary tree in ascending numerical order.

The procedure takes a `Node` object as a parameter and then:

- checks if the left node is null. If there is a left node, the function calls itself with the left node
- outputs the data of the current node
- checks if the right node is null. If there is a right node, the function calls itself with the right node.

Write program code for `OutputInOrder()`

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

[5]

(e) **(i)** Write program code to extend the main program to:

- create a new object of type `Tree` with the node containing the data 10 as the first node
- insert the nodes with the values 20, 5, 15 and 7 into the tree in the order given
- call `OutputInOrder()` with the tree's root node as a parameter.

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 of the output(s).

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

[1]

BLANK PAGE

BLANK PAGE

BLANK PAGE