Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/43**

Paper 4 Practical **October/November 2025**

2 hours 30 minutes

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

0089200834

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 **16** pages. Any blank pages are indicated.

You have been supplied with the following source files:

`BinaryData.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 2**. The file is called **`BinaryData.txt`**

1 A program stores data for a board game using Object-Oriented Programming (OOP). The game has objects that are placed on the board. Each object has a string code (for example `"A"`) and an integer value (for example, `2`).

The class `BoardObject` stores the data about the objects that can be placed on the board:

BoardObject	
`Code : String`	stores the board object's code
`Value : Integer`	stores the integer value of the board object
`Constructor()`	initialises the attributes to the parameter values
`GetCode()`	returns `Code`
`GetValue()`	returns `Value`

(a) (i) Write program code to declare the class `BoardObject` 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 **Question1_N25**.

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

[5]

(ii) The methods `GetCode()` and `GetValue()` return the appropriate attribute.

Write program code for `GetCode()` and `GetValue()`

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

[3]

(iii) The table shows the code and value of **five** board objects. The table has the variable identifier where each of the objects are stored.

Variable identifier	Code	Value
Object1	"A"	2
Object2	"B"	3
Object3	"C"	5
Object4	"D"	2
Object5	"E"	7

Write program code for the main program to instantiate each of the **five** board objects and store them in the variables with the identifiers given.

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

[3]

(b) The board objects are placed on a board that is represented by a 0-indexed 2D array:

- the board is a 10 x 10 grid
- each position on the board is identified by a row and column number
- rows are numbered 0 to 9
- columns are numbered 0 to 9
- board objects can be placed at a row and column position
- each board position is initialised with an empty `BoardObject`, an empty `BoardObject` has `Code = "-"` and `Value = 0`

For example, the element highlighted in the given board is in row 1 and column 3.

Row	Column									
	0	1	2	3	4	5	6	7	8	9
0										
1				■						
2										
3										
4										
5										
6										
7										
8										
9										

The class `Board` stores the data about the board and where the board objects are placed.

Board	
`TheBoard : ARRAY[0:9, 0:9] of BoardObject`	stores the board contents as a 2D array of 10 x 10 elements of type `BoardObject`
`Constructor()`	initialises each of `TheBoard` elements to an empty `BoardObject` with `Code = "-"` and `Value = 0`
`GetObject()`	takes a row and column number as parameters and returns the `BoardObject` at the row, column position
`SetObject()`	takes a `BoardObject`, row number and column number as parameters. Stores the `BoardObject` in `TheBoard` at the given row, column position
`DisplayBoard()`	outputs the code of each `BoardObject` stored in `TheBoard`, one row at a time

(i) Write program code to declare the class `Board` 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 **1(b)(i)** in the evidence document.

[4]

(ii) The method `GetObject()` takes a row number and column number as parameters.

The method returns the `BoardObject` stored at the parameter position.

Write program code for `GetObject()`

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

[2]

(iii) The method `SetObject()` takes **three** parameters: a `BoardObject`, row number and column number.

The method stores the `BoardObject` parameter in the row, column position in `TheBoard`

Write program code for `SetObject()`

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

[2]

(iv) The method `DisplayBoard()` outputs the `Code` of each `BoardObject` stored in `TheBoard` using `GetCode()`

Each row in `TheBoard` is output on one line with a space between each `Code`

For example, the following board contains these four board objects:

- one object has code `"A"` in row 0 column 7
- one object has code `"B"` in row 0 column 9
- one object has code `"C"` in row 1 column 1
- one object has code `"E"` in row 6 column 5

The other board objects are empty. The output for this board will be:

```
- - - - - - - A - B
- C - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
- - - - - E - - - -
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - - -
```

Write program code for `DisplayBoard()`

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

[3]

(c) (i) The table gives the row and column position on the board to store each of the five objects created in part **1(a)(iii)**.

Object identifier	row position	column position
`Object1`	0	0
`Object2`	9	9
`Object3`	4	5
`Object4`	2	2
`Object5`	8	7

Write program code to amend the main program to:

- declare a new instance of `Board()`
- store each `BoardObject` in the position given in the table
- call `DisplayBoard()` for the new board object.

Save your program.

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

[1]

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

- repeatedly take a row position as input until it is between 0 and 9 inclusive
- repeatedly take a column position as input until it is between 0 and 9 inclusive
- use the appropriate method(s) to identify if there is an object in the array position input
- output `"Miss"` if there is an empty `BoardObject` in that position
- output the `Code` and `Value` if there is a non-empty `BoardObject` in that position.

All outputs must include appropriate messages.

Save your program.

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

[4]

(ii) Test your program by entering this test data in the order given:

Row position first input:	10
Row position second input:	4
Column position first input:	–1
Column position second input:	5

Take a screenshot of the output(s).

Save your program.

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

[1]

2 A program reads string data from a text file into a linear queue and then compresses the data.

The queue is created using the global 1D array `Queue`. The queue can store up to 100 elements. Each element is initialised to the empty string `""`

The queue has two pointers that are global to the program:

- `QueueHead` that points to the index of the first element in the queue, initialised to `-1`
- `QueueTail` that points to the index of the last element in the queue, initialised to `-1`

The global variable `NumberItems` stores the number of elements stored in the queue, initialised to `0`

(a) Write program code to declare and initialise `Queue`, `QueueHead`, `QueueTail` and `NumberItems`

Save your program as **Question2_N25**. Copy and paste the program code into part **2(a)** in the evidence document.

[2]

(b) The function `Enqueue()` takes a string as a parameter. The function checks if the queue is full, and returns Boolean `FALSE` if the queue is full.

If the queue is not full, the parameter is inserted into the next position in `Queue`. The function updates the appropriate pointer(s), updates `NumberItems` and then returns Boolean `TRUE`

Write program code for `Enqueue()`

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

[5]

(c) The function `Dequeue()` returns the string `"False"` if the queue is empty.

If the queue is not empty, the function returns the next element in the queue. The function updates the appropriate pointer(s) and updates `NumberItems`

Write program code for `Dequeue()`

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

[3]

(d) The text file `BinaryData.txt` stores individual binary digits, `'1'` and `'0'`. Each digit is on a new line. For example, the first line in the text file stores `'1'`, the second line stores `'1'`

The procedure `ReadData()` reads in each line from the text file `BinaryData.txt` and inserts it into the queue using the appropriate method.

The procedure needs to work for a text file with any number of lines up to a maximum of 100.

Write program code for `ReadData()`

Save your program.

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

[5]

(e) The string data in the text file is compressed.

The compression algorithm counts the number of times each binary digit appears consecutively, then stores the binary digit followed by the number of times it appears. The algorithm stores the compressed data in a single string.

For example, if the text file contains the data:

```
1
1
0
0
0
1
1
1
```

The compression algorithm will create the string `"120313"` because there are two `'1'` digits, followed by three `'0'` digits, followed by three `'1'` digits.

The procedure `Compress()` uses `Dequeue()` to remove each element from the queue in turn. The procedure then compresses the data following the compression algorithm described. The new compressed string is stored in the global variable `NewString`

You can assume that one binary digit will never appear more than nine times consecutively in the sequence.

You can assume that there will always be at least one item in the queue.

Write program code for `Compress()`

Save your program.

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

[6]

(f) **(i)** Write program code for the main program to:

- call `ReadData()`
- call `Compress()`
- output the content of the compressed string.

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(s) into part **2(f)(ii)** in the evidence document.

[1]

3 A program is written to perform different individual processes using arrays.

(a) A 1D array stores 10 integers.

(i) A recursive function `RecursiveCount()` takes three parameters:

- `ArrayCopy`, an array of integers
- `NumberElements`, the number of elements in the array of integers
- `DataToFind`, an integer data to find in the array.

The function counts and returns the number of times `DataToFind` is in `ArrayCopy`

The recursive algorithm:

- returns `0` if there are no elements in `ArrayCopy`
- compares the first element in `ArrayCopy` with `DataToFind`
  - if the element matches, the function returns `1` added to the return value from a recursive call (passing the array without the first element)
  - if the element does **not** match, the function returns the return value from a recursive call (passing the array without the first element).

Write program code for `RecursiveCount()`

Save your program as **Question3_N25**. Copy and paste the program code into part **3(a)(i)** in the evidence document.

[6]

(ii) The main program stores the following data in a 1D array of integers in the order given:

```
0  5  1  2  5  9  9  6  5  0
```

The main program calls `RecursiveCount()` with the parameters:

- `0` as the data to find
- `10` as the number of elements
- the array of 10 integers.

The return value is output.

Write program code for the main program.

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

[3]

(iii) Test your program.

Take a screenshot of the output(s).

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

[1]

(b) The program stores the following string. The string has four statements that are each terminated by a semi-colon `';'`

`"x=0;y=1;x=x+y;y++;"`

(i) Write program code to amend the main program to store the string `"x=0;y=1;x=x+y;y++;"` in a local variable.

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

[1]

(ii) The function `SplitData()` splits a string parameter into **four** individual lines of code.

The function creates an array of the strings where each line is stored in a new array element without the semi-colon.

The function returns the array of strings.

Do **not** use an inbuilt function to split the string.

Write program code for `SplitData()`

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

[6]

(iii) The main program needs to call `SplitData()` with the string from part **3(b)(i)**. The main program then needs to output each element from the returned array on a new line.

Write program code to amend the main program.

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

[2]

(iv) Test your program.

Take a screenshot of the output(s).

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

[1]

BLANK PAGE

BLANK PAGE

BLANK PAGE