CAMBRIDGE
International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/43**

Paper 4 Practical **May/June 2026**

2 hours 30 minutes

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

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.

DC (WW/SG) 356387/2

[Turn over

You have been supplied with the following source file:

`Characters.txt`

Open the evidence document, **evidence.docx**

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 **`Characters.txt`**

1 A program stores 15 strings in the 1D array `DataArray`

(a) Write program code to declare the global 1D array `DataArray` with space for **15** string elements.

Save your program as **Question1_J26**. Copy and paste the program code into **1(a)** in the evidence document.

[1]

(b) The procedure `CreateArray()` takes **15** strings as input from the user. Each input is converted to lower case and stored in `DataArray` in the order input.

Write program code for `CreateArray()`

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

[4]

(c) The procedure `OutputArray()` outputs the data in each element in `DataArray` on **one** line with a space between each element.

Write program code for `OutputArray()`

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

[2]

(d) (i) Write program code for the main program to call `CreateArray()` and then `OutputArray()`

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

[1]

(ii) Test your program. Enter the following data for the inputs in the order given:

1

One

2

TWO

three

??

house

99987

popping

green

eleven

12

Twenty

THIRTY

16

Take a screenshot of the output(s).

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

[1]

(e) The procedure `FindWords()` takes a string as input from the user and converts it to lower case. The procedure counts and outputs the number of times the input is stored in `DataArray`

Write program code for `FindWords()`. Do **not** use an in-built function to count the number of times the input is stored in `DataArray`

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

[5]

(f) **(i)** The procedure `SortLength()` uses a bubble sort to order the elements in `DataArray` into ascending order by the length of each string.

The bubble sort needs to be efficient and stop iterating as soon as it identifies that `DataArray` is in the correct order.

Write program code for `SortLength()`. Do **not** use an in-built function to sort the array or check if the array is in the correct order.

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

[5]

(ii) Write program code to extend the main program by calling `FindWords()` then `SortLength()` then `OutputArray()`

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

[1]

(iii) Test your program. Enter the following data to be stored in the array in the order given:

one

One

1

three

three

One

house

99987

popping

green

eleven

One

Twenty

THIRTY

One

Enter `"one"` as the string to find in the array.

Take a screenshot of the output(s).

Save your program. Copy and paste the screenshot into **1(f)(iii)** in the evidence document.

[1]

2 A prototype for a game is being developed using Object-Oriented Programming (OOP).

The game has a set of characters that have data stored about their characteristics, for example if the character can fly or not fly. A user has to select their requirements, for example a character that can fly, and the program will output the characters that meet the user's requirements.

The class `Character` stores the data about the characters. The attributes are public.

Character	
`CharacterName : String`	stores the character's name
`Colour : String`	stores the colour of the character, for example `"Green"`
`SuperStrength : String`	stores whether the character has super strength (`"Yes"`) or not (`"No"`)
`Fly : String`	stores whether the character can fly (`"Yes"`) or not (`"No"`)
`Invisible : String`	stores whether the character can become invisible (`"Yes"`) or not (`"No"`)
`Constructor()`	initialises the attributes to its parameter values
`GetCharacterName()`	returns `CharacterName`
`GetColour()`	returns `Colour`
`GetSuperStrength()`	returns `SuperStrength`
`GetFly()`	returns `Fly`
`GetInvisible()`	returns `Invisible`

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

Use your programming language appropriate constructor.

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

Save your program as **Question2_J26**.

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

[4]

(ii) The get methods `GetCharacterName()`, `GetColour()`, `GetSuperStrength()`, `GetFly()` and `GetInvisible()` each return the appropriate attribute.

Write program code for `GetCharacterName()`, `GetColour()`, `GetSuperStrength()`, `GetFly()` and `GetInvisible()`

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

[3]

(b) The text file `Characters.txt` stores data about 15 characters. Each character is on a new line in the file. The data is stored in the order: Character name, colour, super strength, fly, invisible. Each value is separated by a comma.

For example, the first line in the file is:

```
Jet,Black,Yes,No,No
```

The name of the character is Jet. The colour of the character is Black. The character has super strength. The character cannot fly. The character cannot become invisible.

The function `ReadData()`:

- opens the file `Characters.txt` and reads in the data
- creates a new `Character` object for each character in the file
- stores each `Character` object in a local array
- returns the populated array.

The function uses exception handling when opening and reading from the file.

Write program code for `ReadData()`

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

[7]

(c) The procedure `OutputCharacters()` takes an array of `Character` objects as a parameter. The procedure outputs the data for each character on a new line, in the format:

```
Name: <Character name> Colour: <Colour> Super strength: <Super
strength> Fly: <Fly> Invisible: <Invisible>
```

For example:

```
Name: Jet Colour: Black Super strength: Yes Fly: No Invisible: No
```

Write program code for `OutputCharacters()`

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

[3]

(d) **(i)** The main program calls `ReadData()` and then `OutputCharacters()` with the returned array.

Write program code for the main program.

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

[1]

(e) The main program needs to give the user two choices to limit the characters that they can choose from. For example, they could choose to only see characters that are green and that can fly.

(i) The main program needs extending to:

- prompt the user to enter **two** criteria and take both as input. For example, the colour green and that they have super strength.
- create a new array with only the characters that meet both of the criteria.

You do **not** need to validate the inputs. You can assume that each criteria that the user inputs will be for a different characteristic of the character.

The main program should then:

- output the number of characters that meet both criteria
- use `OutputCharacters()` to output the characters that meet both criteria.

Write program code to extend the main program as described.

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

[8]

(ii) Test your program. Enter the options given when prompted:

first choice is characters with the colour black

second choice is characters that can fly.

Take a screenshot of the output(s).

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

[1]

3 A binary tree is designed to store up to 100 positive integers. The binary tree stores data in ascending numerical order.

The data in the binary tree is created as a global 2D array with the identifier `BinaryTree`

Each node has three values:

- a pointer to the array index with the left node
- the integer data
- a pointer to the array index with the right node.

A null pointer is represented by the integer `-1`. All nodes are initialised with the null data `-1`

The table has example data for a binary tree:

index	left pointer	data	right pointer
0	1	20	2
1	4	10	-1
2	3	26	-1
3	-1	22	-1
4	-1	8	-1

The binary tree has two global pointers:

- `TreeRootPointer`, initialised to `-1`, stores the index of the root node of the binary tree
- `FirstFreeNodePointer`, initialised to `0`, stores the index of the first node that does **not** store any data.

Data cannot be removed or replaced from the tree. `FirstFreeNodePointer` is incremented each time a node is added to the binary tree.

(a) The main program initialises all 100 nodes, `TreeRootPointer` and `FirstFreeNodePointer`

Write program code for the main program.

Save your program as **Question3_J26**.

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

[3]

(b) The procedure `AddNode()`:

- takes an integer parameter to be stored as the data in a new node
- stores the data in the first element and updates the root pointer when the tree is empty
- outputs `"The tree is full"` if the tree is full
- stores the data in the next free node if the tree is **not** full, then:
  - checks if the data is less than or greater than the root node
  - follows the left or right pointer(s) repeatedly until it reaches the appropriate leaf node
  - updates the appropriate parent node pointer to the new node.

Write program code for `AddNode()`

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

[7]

(c) The procedure `OutputArray()` outputs the content of the first 10 elements in the binary tree. Each node is output on one line in the format:

```
<left pointer> <data> <right pointer>
```

For example:

```
1  20  4
```

Write program code for `OutputArray()`

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

[2]

(d) (i) Write program code to extend the main program to:

- take 10 integers as input
- store each input in the binary tree in the order they are entered using the appropriate procedure
- output the content of the binary tree using `OutputArray()`

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

[3]

(ii) Test your program. Enter the following numbers in the order shown:

50
26
120
236
2
16
67
49
165
15

Take a screenshot of the output(s).

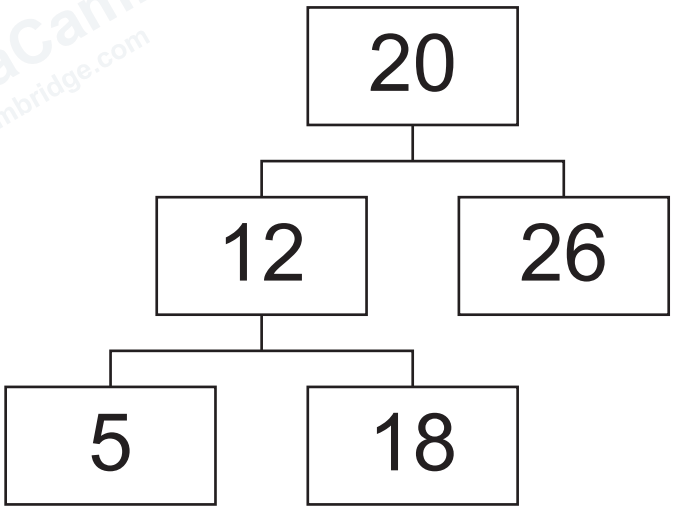
Save your program. Copy and paste the screenshot into **3(d)(ii)** in the evidence document.

[1]

(e) An in-order tree traversal outputs the contents of the binary tree in ascending numerical order by:

- visiting the left node
- output the data
- visiting the right node.

For example, an in-order traversal on the following binary tree will output: 5 12 18 20 26



(i) The recursive procedure `InOrder()`:

- takes the root pointer as a parameter
- makes a recursive call with the left pointer as the parameter if the left pointer is **not** null
- outputs the data at the parameter
- makes a recursive call with the right pointer as the parameter if the right pointer is **not** null.

Write program code for `InOrder()`

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

[7]

(ii) Write program code to extend the main program to call `InOrder()`

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

[1]

(iii) Test your program. Enter the following numbers in the order given:

50
26
120
236
2
16
67
49
165
15

Take a screenshot of the output(s).

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

[1]

BLANK PAGE

BLANK PAGE

