CAMBRIDGE
International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/42**

Paper 4 Practical **May/June 2026**

2 hours 30 minutes

5655643136

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 **12** pages. Any blank pages are indicated.

DC (PQ) 356382/2

[Turn over

You have been supplied with the following source file:

`AvatarFile.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 **`AvatarFile.txt`**

1 A program stores **nine** lower-case strings in a 1D array `TheData`. The following strings are stored in the order shown:

```
laptop
tablet
printer
computer
processor
monitor
memory
cable
touchscreen
```

(a) Write program code to declare the array `TheData` in the main program. Store the given strings in the array.

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

[2]

(b) (i) The procedure `FindLongestShortest()`:

- takes an array of strings as a parameter
- finds and outputs the longest string in the array of strings
- finds and outputs the shortest string in the array of strings.

All outputs must have appropriate messages. Do **not** use in-built functions to find the longest and shortest string in the array. You may use an in-built length function to find the length of a string.

You can assume that there will only be **one** string that has the shortest length.

You can assume that there will only be **one** string that has the longest length.

Write program code for `FindLongestShortest()`

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

[5]

(ii) Write program code to extend the main program to call `FindLongestShortest()` with `TheData`

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

[1]

(iii) Test your program

Take a screenshot of the output(s).

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

[1]

(c) (i) The function `InsertionSort()`:

- takes an array of strings as a parameter
- sorts the strings in the array into ascending alphabetical order using an insertion sort
- returns the sorted array.

The function must work for an array of any length. Do **not** use an in-built function to sort the array.

Write program code for `InsertionSort()`

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

[5]

(ii) Write program code to extend the main program to perform the following in the order shown:

- output the string `"Before sorting"`
- output the contents of the array `TheData`
- call `InsertionSort()` with `TheData` as a parameter
- output the string `"After sorting"`
- output the contents of the returned sorted array.

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

[1]

2 A program stores data about avatars in a program using Object-Oriented Programming (OOP). An avatar is a character that a user can control in a game.

The class `Avatar` stores the data about avatars.

Avatar	
`TheName : String`	stores the name of the avatar
`Glasses : Boolean`	stores `TRUE` if the avatar wears glasses stores `FALSE` if the avatar does **not** wear glasses
`Hair : String`	stores the colour of the avatar's hair
`Age : Integer`	stores the age of the avatar
`Constructor()`	initialises all attributes to its parameter values
`GetName()`	returns the name of the avatar
`GetAge()`	returns the age of the avatar
`MeetCriteria()`	takes the glasses choice and hair choice as parameters and returns whether the avatar matches the choices

(a) (i) Write program code to declare the class `Avatar` and its constructor. All attributes must be private.

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 **Question2_J26**.

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

[4]

(ii) The get methods `GetName()` and `GetAge()` each return the relevant attribute.

Write program code for `GetName()` and `GetAge()`

Save your program.

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

[3]

(iii) The method `MeetCritera()` takes two parameters:

- a Boolean value to identify whether the avatar wears glasses
- a string value to identify the hair colour.

The method checks whether this avatar's attributes match both of the parameters. Boolean `TRUE` is returned when both of the attributes match the parameters, otherwise Boolean `FALSE` is returned.

Write program code for `MeetCriteria()`

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

[2]

(b) (i) The text file `AvatarFile.txt` stores data for 30 avatars. Each avatar is on a new line in the file. The data is stored in the order: name, glasses, hair colour, age. Each value is separated by a space.

For example, the first line in the file is:

```
Kit True blonde 21
```

The name of the avatar is Kit, the avatar wears glasses, the avatar has blonde hair, the avatar is 21 years old.

The procedure `LoadAvatars()` reads the data from the text file into the program. The function creates a new object for each avatar in the file and stores the avatar objects in a global array with the identifier `AvailableAvatars`

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

Write program code for `LoadAvatars()`

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

[7]

(ii) Write program code for the main program to call `LoadAvatars()`

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

[1]

(c) (i) The procedure `EnterChoices()` prompts the user to enter whether they want an avatar that wears glasses and the hair colour they want.

The procedure takes these two values as input until they are valid:

- wearing glasses can only be yes or no
- the hair colour can be black, blonde, brown, pink, red or green.

The procedure calls the method `MeetCriteria()` for each avatar in `AvailableAvatars` and stores the avatars that match the criteria in a global array.

Write program code for `EnterChoices()`

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

[6]

(ii) Write program code to extend the main program to:

- call `EnterChoices()`
- output the quantity of avatars that match the criteria
- output the name and age of each avatar that matches the criteria.

All outputs must include an appropriate message.

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

[5]

(d) (i) Test your program with the following data as input:

Glasses choice: yes

Hair colour first input: blue

Hair colour second input: black

Take a screenshot of the output(s).

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

[1]

(ii) Test your program with the following data as input:

Glasses choice: no

Hair colour: green

Take a screenshot of the output(s).

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

[1]

3 A 2D array is used to store the linked list, `LinkedList`. The linked list stores a maximum of **eight** data items. Each data item is an integer.

The array elements not currently used in the linked list are stored as the empty list.

`LinkedList` has two pointers:

- `StartLinkedList` stores a pointer to the first node in the linked list
- `StartEmptyList` stores a pointer to the first node in the empty list.

A data value of `-1` indicates there is no data in the node.

A pointer of `-1` indicates it is the end of the linked list, or the end of the empty list.

The linked list and pointers all have global scope.

The linked list already has the following data stored:

array index	data	pointer
0	9512	5
1	5541	3
2	1999	0
3	6651	-1
4	-1	6
5	5612	1
6	-1	-1
7	-1	4

`StartLinkedList` currently stores `2`

`StartEmptyList` currently stores `7`

(a) Write program code to declare and initialise the linked list and the pointers.

Save your program as **Question3_J26**.

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

[5]

(b) (i) The procedure `PrintListItems()` outputs the data in each node of the linked list. The procedure starts with the first node in the linked list and follows the pointers until it has reached the end of the linked list.

Write program code for `PrintListItems()`

Save your program.

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

[4]

(ii) The procedure `PrintEmptyList()` outputs the index of each node in the empty list. The procedure starts with the first node in the empty list and follows the pointers until it has output the index of each node in the empty list.

Write program code for `PrintEmptyList()`

Save your program.

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

[3]

(iii) Write program code to extend the main program to:

- output `"Linked list"` and then call `PrintListItems()`
- output `"Empty list"` and then call `PrintEmptyList()`

Save your program.

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

[2]

(c) New nodes are inserted at the end of the linked list.

The procedure `AddItem()` takes an integer parameter, inserts the parameter in the space identified by `StartEmptyList` and updates the appropriate pointer(s).

If there is no space in the linked list, the procedure outputs `"List is full"`

(i) Write program code for `AddItem()`

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

[8]

(ii) Write program code to extend the main program to:

- use `AddItem()` to insert a new node into the linked list with the data `8451`
- output the new contents of the linked list using `PrintListItems()` with a suitable heading
- output the new contents of the empty list using `PrintEmptyList()` with a suitable heading.

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

[2]

(iii) Test your program.

Take a screenshot of the output(s).

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

[1]

BLANK PAGE

BLANK PAGE

