Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/42**

Paper 4 Practical **October/November 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 [ ].

DC (PQ) 343263/2

You have been supplied with the following source files:

`TreeData.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 3**. The file is called **`TreeData.txt`**

1 A program stores data about birds using Object-Oriented Programming (OOP).

The class `Bird` stores the data about the birds:

Bird	
`DistancePerHour : Real`	stores the number of kilometres per hour (km/h) the bird can fly (between 0.0 and 100.0 inclusive)
`Species : String`	stores the species of the bird, for example Pigeon
`XPosition : Real`	stores the current horizontal position of the bird
`YPosition : Real`	stores the current vertical position of the bird
`Constructor()`	initialises `Species` and `DistancePerHour` to the parameter values; initialises `XPosition` and `YPosition` to 500.0
`GetPosition()`	returns a string message that contains the horizontal and vertical position of the bird
`GetSpecies()`	returns the species of the bird
`Move()`	takes a direction and number of minutes flying (between 0 and 500 inclusive) as parameters and updates the appropriate horizontal or vertical position

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

Do **not** declare the other methods.

All attributes should be private.

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.

[4]

(ii) The method `GetSpecies()` returns the species of the bird.

Write program code for `GetSpecies()`

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

[2]

(iii) The method `GetPosition()` returns a string of the bird's data in the format:

```
"X = " & <XPosition> & " Y = " & <YPosition>
```

An example string for a bird is:

```
X = 500.0 Y = 250.0
```

Write program code for `GetPosition()`

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

[3]

(iv) If the bird is travelling north, the vertical position increases. If the bird is travelling south, the vertical position decreases.

If the bird is travelling east, the horizontal position increases. If the bird is travelling west, the horizontal position decreases.

The method `Move()`:

- takes the direction of travel as a parameter in the form `'N'` for north, `'S'` for south, `'E'` for east or `'W'` for west
- takes the number of minutes that the bird has been flying (to the nearest minute) as a parameter
- calculates the distance travelled by the bird using the formula:
  distance travelled = (distance per hour / 60) * minutes flying
- updates the vertical or horizontal position using the distance calculated.

You do **not** need to validate the parameters.

Write program code for `Move()`

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

[5]

(b) The main program creates two instances of `Bird`

The first bird is the species 'Cockatiel' and flies 71.0 km/h.

The second bird is the species 'Macaw' and flies 56.0 km/h.

Write program code to declare and initialise the **two** birds.

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

[3]

(c) (i) The main program needs to:

- output a message that includes the species and current X position and Y position for each bird
- prompt the user to select one of the birds to move and take this as an input
- prompt the user to enter the direction the bird has been travelling and take this as an input
- prompt the user to enter the time to the nearest minute that the bird has been travelling and take this as an input
- call the appropriate method to update the chosen bird's position
- output an update on the bird's new position.

Each input needs to repeat until valid data is entered. All outputs must be meaningful.

Write program code to amend the main program.

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

[8]

(ii) Test your program **four** times to meet these criteria:

Test 1: The Cockatiel travels north for 60 minutes.

Test 2: The Macaw travels south for 30 minutes.

Test 3: The Cockatiel travels west for 30 minutes.

Test 4: The Macaw travels east for 60 minutes.

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.

[3]

2 A program stores 20 unique random integers between 0 and 100 (inclusive) in a 1D array that is local to the main program.

(a) Write program code to declare the array local to the main program and store 20 **unique** random numbers between 0 and 100 (inclusive) in the array.

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

[3]

(b) The procedure `PrintArray()` takes an integer array as a parameter. The procedure outputs the array contents on a single line with a space between each integer.

Write the program code for `PrintArray()`

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

[3]

(c) The function `BubbleSort()`:

- takes an integer array as a parameter
- sorts the data into ascending order using a bubble sort
- returns the sorted array.

The function needs to work for an array of any length.

Do **not** use an inbuilt sorting method.

Write program code for `BubbleSort()`

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

[5]

(d) (i) The main program:

- outputs the contents of the array using `PrintArray()`
- sorts the array using `BubbleSort()`
- outputs `"Sorted"`
- outputs the contents of the sorted array using `PrintArray()`

Write program code for the main program.

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

[1]

(e) The recursive function `RecursiveBinarySearch()` takes four parameters:

- an integer array
- the lower bound of the array
- the upper bound of the array
- the value to find in the array.

The recursive function performs a binary search to find the index of the value in the array.

The function returns the index of the value if it is found. The function returns `-1` if the value is not found.

Write program code for `RecursiveBinarySearch()`

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

[6]

(f) **(i)** The main program:

- prompts the user to enter an integer
- takes the integer as input
- calls `RecursiveBinarySearch()` with the sorted array, appropriate lower bound, appropriate upper bound and the user's input as parameters
- outputs `"Not found"` if the input is not within the array
- outputs `"Found at position"` and the index if the input is within the array.

Write program code to amend the main program.

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

[3]

(ii) Test your program **three** times with each of the inputs described:

Test 1: the smallest number in the array

Test 2: the largest number in the array

Test 3: a number **not** in the array

Take a screenshot of each output.

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

[2]

BLANK PAGE

3 A program stores integers in ascending order in an ordered binary tree. The tree is implemented as a 2D array.

Each node is stored with three values:

- a pointer to the left node
- the data
- a pointer to the right node.

All null values are stored as `-1`

The binary tree can store up to 50 nodes.

Nodes cannot be deleted from the binary tree.

(a) The binary tree is stored as a global array with the identifier `TreeArray`. The left pointer, the data and the right pointer of each node are initialised to `-1`

The global variable `RootPointer` stores the index of the root node in the tree, initialised to `-1`

The global variable `FreeNode` stores the index of the next free node in the array, initialised to `0`

Write program code to declare and initialise `TreeArray`, `RootPointer` and `FreeNode`

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

[3]

(b) The procedure `AddNode()`:

- takes an integer to store in the binary tree as a parameter
- stores the parameter in the next free node in the array
- finds the position to store the data in the tree by following the appropriate pointers
- updates the pointer of the new node's parent node.

The procedure outputs `"The tree is full"` if the parameter cannot be stored because the tree is full.

Write program code for `AddNode()`

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

[7]

(c) The text file `TreeData.txt` stores 50 integers. Each integer is on a new line in the file.

The main program reads each integer from the file and stores each integer in the binary tree in the order they are read.

Write program code for the main program.

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

[4]

(d) The procedure `WriteAllToFile()` stores the content of `TreeArray` in a new text file with the filename `Tree.txt`. The file `Tree.txt` is not provided.

Each node in `TreeArray` is stored on one line with a comma separating each value.

For example, the current contents of `TreeArray` are:

Index	**0**	**1**	**2**
0	−1	20	1
1	−1	30	−1
…			
49	−1	−1	−1

After writing `TreeArray` to the text file, `Tree.txt` will contain:

```
-1,20,1
-1,30,-1
…
-1,-1,-1
```

Write program code for `WriteAllToFile()`

Include exception handling when writing to the file.

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

[5]

(e) (i) Amend the main program to call `WriteAllToFile()`

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

[1]

(ii) Test your program.

Take a screenshot that shows all of the content stored in the file `Tree.txt`

In this screenshot you need to make sure the filename is visible.

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

[1]