Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/43**

Paper 4 Practical **May/June 2024**

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 (SL) 329378/3

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 **`Trees.txt`**

1 A program needs to take integer numbers as input, sort the numbers and then search for a specific number.

(a) The integer numbers will be stored in the global 1D array, `DataStored`, with space for up to 20 integers.

The global variable `NumberItems` stores the quantity of items the array contains.

Write program code to declare `DataStored` and `NumberItems`.

Save your program as **Question1_J24**. Copy and paste the program code into part **1(a)** in the evidence document.

[1]

(b) The procedure `Initialise()`:

- prompts the user to input the quantity of numbers the user would like to enter
- reads the input and validates it is between 1 and 20 (inclusive)
- prompts the user to input each number and stores each number in `DataStored`.

Write program code for `Initialise()`.

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

[5]

(c) The main program stores `0` in `NumberItems`, calls `Initialise()` and then outputs the contents of `DataStored`.

(i) Write program code for the main program.

Save your program.

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

[2]

(ii) Test your program by inputting the following data in the order given:

30

5

3

9

4

1

2

Take a screenshot of the output(s).

Save your program.

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

[2]

(d) The procedure `BubbleSort()` uses a bubble sort to sort the data in `DataStored` into ascending numerical order.

(i) Write program code for `BubbleSort()`.

Save your program.

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

[4]

(ii) Write program code to amend the main program to call `BubbleSort()` and then output the contents of `DataStored`.

Save your program.

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

[1]

(iii) Test your program by inputting the following data in the order given:

5

3

9

4

1

2

Take a screenshot of the output(s).

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

[1]

(e) The function `BinarySearch()`:

- takes the integer parameter `DataToFind` to search for in the array
- performs an iterative binary search on the array `DataStored`
- returns the index where `DataToFind` is found in `DataStored`. If `DataToFind` is **not** found, the function returns `-1`.

(i) Write program code for the iterative function `BinarySearch()`.

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

[6]

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

- take a number as input from the user
- call `BinarySearch()` with the number input
- output the value returned from the function call as its parameter.

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

[3]

(iii) Test your program twice with the following inputs:

Test 1: 5 1 6 2 8 10 2

Test 2: 5 1 6 2 8 10 7

Take a screenshot of the output(s).

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

[2]

2 A computer program will store data about trees.

The user can enter their requirements for a tree and a suitable tree will be selected.

The program is written using object-oriented programming.

The class `Tree` stores data about the trees.

Tree	
`TreeName : STRING`	stores the name of the tree
`HeightGrowth : INTEGER`	stores the number of cm the tree will grow each year
`MaxHeight : INTEGER`	stores the maximum height in cm that the tree will grow
`MaxWidth : INTEGER`	stores the maximum width in cm that the tree will grow
`Evergreen : STRING`	stores whether the tree keeps its leaves as `"Yes"`, or loses its leaves as `"No"`
`Constructor()`	initialises `TreeName`, `HeightGrowth`, `MaxHeight`, `MaxWidth` and `Evergreen` to its parameter values
`GetTreeName()`	returns the name of the tree
`GetGrowth()`	returns the number of cm the tree will grow each year
`GetMaxHeight()`	returns the maximum height in cm that the tree will grow
`GetMaxWidth()`	returns the maximum width in cm that the tree will grow
`GetEvergreen()`	returns whether the tree keeps its leaves or loses its leaves

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

Do **not** declare the other methods.

Use the appropriate constructor for your programming language.

All attributes must be private.

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

Save your program as **Question2_J24**.

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

[4]

(ii) The get methods `GetTreeName()`, `GetGrowth()`, `GetMaxHeight()`, `GetMaxWidth()` and `GetEvergreen()` each return the relevant attribute.

Write program code for the get methods.

Save your program.

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

[3]

(b) The text file `Trees.txt` stores data about 9 trees.

The data in the file is stored in the format:

Tree name,Height growth each year,Maximum height,Maximum width,Evergreen

For example, the first row of data is:

Beech,30,400,200,No

The tree is a Beech. It can grow 30 cm each year. It has a maximum height of 400 cm. It has a maximum width of 200 cm. It is **not** evergreen (it loses its leaves).

The function `ReadData()`:

- creates an array of type `Tree`
- reads the data from the file
- raises an exception if the file is **not** found
- creates a new object of type `Tree` for each tree in the file
- appends each object to the array
- returns the array.

Write program code for `ReadData()`.

Save your program.

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

[7]

(c) The procedure `PrintTrees()` takes a `Tree` object as a parameter and outputs the tree's name, height growth each year, maximum height, maximum width and whether it is evergreen.

The output message changes depending on whether it is evergreen.

If it is evergreen, it is in the format:

> `TreeName` has a maximum height `MaxHeight` a maximum width `MaxWidth` and grows `HeightGrowth` cm a year. It does not lose its leaves.

If it is **not** evergreen, it is in the format:

> `TreeName` has a maximum height `MaxHeight` a maximum width `MaxWidth` and grows `HeightGrowth` cm a year. It loses its leaves each year.

Write program code for `PrintTrees()`.

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

[4]

(d) The main program calls `ReadData()`, stores the return value and calls `PrintTrees()` with the first object in the returned array.

(i) 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.

[2]

(ii) Test your program.

Take a screenshot of the output(s).

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

[1]

(e) The procedure `ChooseTree()` takes an array of `Tree` objects as a parameter.

The procedure prompts the user to input their requirements for a tree.
The user needs to enter:

- the maximum height the tree can be in cm
- the maximum width the tree can be in cm
- whether they want the tree to be evergreen, or **not** evergreen.

A tree meets the requirements if:

- the tree's maximum height is **not** more than the user's input

and

- the tree's maximum width is **not** more than the user's input

and

- the tree matches their evergreen input.

The procedure creates a new array of all the `Tree` objects that meet all the requirements.

The procedure calls `PrintTrees()` for each `Tree` object that meets all the requirements. If there are no trees that meet all the requirements, a suitable message is output.

(i) Write program code for `ChooseTree()`.

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

[6]

(ii) The procedure `ChooseTree()` needs amending. After the procedure has output the list of trees that meet all the requirements, the procedure needs to:

- take as input the name of one of the trees that the user would like to buy from those that meet all the requirements
- take as input the height of the tree in cm when it is bought
- calculate and output how many years it will take the tree to grow to its maximum height.

For example, the user inputs the tree, Beech. The tree's height is 40 cm when bought. The tree will take 12 years to reach its maximum height of 400 cm.

Write program code to amend `ChooseTree()`.

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

[2]

(iii) Write program code to amend the main program to call `ChooseTrees()`.

Test your program with the following tree requirements:

- a maximum height of 400 cm
- a maximum width of 200 cm
- a tree that is evergreen (does **not** lose its leaves).

When asked for the tree selection, use the following data:

- first tree name entered is 'Blue Conifer'
- starting height is 100 cm.

Take a screenshot of the outputs.

Save your program.

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

[2]

3 A program reads data from the user and stores the data that is valid in a linear queue.

The queue is stored as a global 1D array, `QueueData`, of string values. The array needs space for 20 elements.

The global variable `QueueHead` stores the index of the first element in the queue.
The global variable `QueueTail` stores the index of the last element in the queue.

(a) The main program initialises all the elements in `QueueData` to a suitable null value, `QueueHead` to −1 and `QueueTail` to −1.

Write program code for the main program.

Save your program as **Question3_J24**. Copy and paste the program code into part **3(a)** in the evidence document.

[1]

(b) The function `Enqueue()` takes the data to insert into the queue as a parameter.

If the queue is **not** full, it inserts the parameter in the queue, updates the appropriate pointer(s) and returns `TRUE`. If the queue is full, it returns `FALSE`.

Write program code for `Enqueue()`.

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

[4]

(c) The function `Dequeue()` returns `"false"` if the queue is empty. If the queue is **not** empty, it returns the next item in the queue and updates the appropriate pointer(s).

Write program code for `Dequeue()`.

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

[3]

(d) The string values to be stored in the queue are 7 characters long. The first 6 characters are digits and the 7th character is a check digit. The check digit is calculated from the first 6 digits using this algorithm:

- multiply the digits in position 0, position 2 and position 4 by 1
- multiply the digits in position 1, position 3 and position 5 by 3
- calculate the sum of the products (add together the results from all of the multiplications)
- divide the sum of the products by 10 and round the result down to the nearest integer to get the check digit
- if the check digit equals 10 then it is replaced with '`X`'.

Example:

Data is 954123

Character position	**0**	**1**	**2**	**3**	**4**	**5**
Digit	9	5	4	1	2	3
Multiplier	1	3	1	3	1	3
Product	9	15	4	3	2	9

Sum of products = 9 + 15 + 4 + 3 + 2 + 9 = 42

Divide sum of products by 10: 42 / 10 = 4 (rounded down)

The check digit = 4. This is inserted into character position 6.

The data including the check digit is: **9541234**

A 7-character string is valid if the 7th character matches the check digit for that data. For example, the data 9541235 is invalid because the 7th character (5) does **not** match the check digit for 954123.

(i) The subroutine `StoreItems()` takes ten 7-character strings as input from the user and uses the check digit to validate each input.

Each valid input has the check digit removed and is stored in the queue using `Enqueue()`.
An appropriate message is output if the item is inserted. An appropriate message is output if the queue is already full.

Invalid inputs are **not** stored in the queue.

The subroutine counts and outputs the number of invalid items that were entered.

`StoreItems()` can be a procedure or a function as appropriate.

Write program code for `StoreItems()`.

Save your program.

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

[6]

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

- call `StoreItems()`
- call `Dequeue()`
- output a suitable message if the queue was empty
- output the returned value if the queue was **not** empty.

Save your program.

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

[1]

(iii) Test the program with the following inputs in the order given:

999999X

1251484

5500212

0033585

9845788

6666666

3258746

8111022

7568557

0012353

Take a screenshot of the output(s).

Save your program.

Copy and paste the screenshot into part **3(d)(iii)** in the evidence document.

[2]

BLANK PAGE

BLANK PAGE

BLANK PAGE