Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/04**

Paper 4 Practical **For examination from 2021**

SPECIMEN PAPER **2 hours 30 minutes**

0123456789

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 your 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 **10** pages. Blank pages are indicated.

Candidate source files are used to answer question 3. The files are: **DataToAdd.txt** and **SecondData.txt**.

1 A 1-dimensional array, `TheData`, stores the following data:

20	3	4	8	12	99	4	26	4

(a) Write a program to declare the array as a local variable, and initialise the values with the data given. Give the array the identifier `TheData`.

Save your program as **question1**.

Copy and paste the program code into the **evidence.doc**

[2]

(b) Pseudocode for an insertion sort is shown below with **three** pieces of code missing.

```
PROCEDURE InsertionSort(TheData())
    FOR Count ← FirstElement TO ……
        DataToInsert ← TheData(i)
        Inserted ← 0
        NextValue ← Count - 1
        WHILE (NextValue >= 0 AND …… <> 1)
            IF (DataToInsert < TheData(NextValue)) THEN
                TheData(NextValue + 1) ← TheData(NextValue)
                NextValue ← NextValue - 1
                TheData(NextValue + 1) ← ……
            ELSE
                Inserted ← 1
            ENDIF
        ENDWHILE
    NEXT
ENDPROCEDURE
```

In the program you wrote for **part 1(a)**, write the procedure `InsertionSort` to perform an insertion sort on the data in `TheData`. Pass the array you declared in **part 1(a)** by reference. Follow the pseudocode given and insert the missing pieces of code in your program.

Save your program.

Copy and paste the program code into the **evidence.doc**

[7]

(c) Write a procedure to output all the contents of the array `TheData` to the screen. The procedure should use iteration. Pass the array into the procedure as a parameter.

Save your program.

Copy and paste the program code into the **evidence.doc**

[3]

(d) (i) The main program needs to output the data before the data has been sorted, and after the date has been sorted.

Use the subroutines you declared in **part (b)** and **part (c)**.

Edit the main program so it:

- outputs all the data in `TheData` before sorting
- sorts the data in `TheData`
- outputs the data after sorting
- outputs appropriate headings to identify the outputs before and after sorting.

Save your program. Copy and paste the program code into the **evidence.doc**

[3]

(ii) Test your program and take a screenshot to show the result.

Save your program. Copy and paste the screenshots into the **evidence.doc**

[2]

(e) (i) Write a function that:

- takes a whole number as input from the user
- if the number is in `TheData` outputs 'found' and returns true
- if the number is not in `TheData` outputs 'not found' and returns false.

Save your program. Copy and paste the program code into the **evidence.doc**

[6]

(ii) Test the function using the number 8 as input, then the number 9 as input.

Take a screenshot of the results of each test, making sure the input is visible in the screenshot.

Save your program. Copy and paste the screenshots into the **evidence.doc**

[2]

2 'Hidden Boxes' is a game where players hide boxes in a virtual world. Other players search for the boxes. Object-oriented techniques must be used to program the game.

The program has a class named `HiddenBox`.

The class has the following properties:

Property	Description
`BoxName`	The name of the box, entered by the creator of the box e.g. `blueBox1`
`Creator`	The player name of the creator of the box e.g. `girl25`
`DateHidden`	The date the box was created e.g. `01/01/2019`

`GameLocation`	The location of the box, in the format two letters followed by four numbers e.g.
	`LL4561`
	`YE4561`

`LastFinds`	A two-dimensional array that stores the player name of the last ten players to find the box, and a comment that each player leaves about the box
`Active`	A Boolean value, `True` means the box can be found (active), `False` means the box cannot be found (inactive)

(a) Write the program code for the class, `HiddenBox`. Declare the properties as private.

Do **not** write the constructor or any other methods.

If you are using Python, add a comment for each property to give its identifier and data type.

Save your program as **question2**

Copy and paste the program code into the **evidence.doc**

[4]

(b) The constructor:

- takes the box name, creator, date hidden and game location as parameters
- sets the box to be inactive
- initialises all the array elements as empty.

Edit your program from **part 2(a)** and write the constructor for `HiddenBox`.

Do **not** write any of the other methods.

Save your program.

Copy and paste the program code into the **evidence.doc**

[4]

(c) The class `HiddenBox` has two getter methods: `GetBoxName` and `GetGameLocation`.

Edit your program and write the getter methods `GetBoxName` and `GetGameLocation` in the class `HiddenBox`.

Save your program. Copy and paste the program code into the **evidence.doc**

[3]

(d) **(i)** The main program declares a 1D array named `TheBoxes` of type `HiddenBox`.

The main program can store up to 10 000 elements as a local variable.

Write program code for the main program.

Save your program. Copy and paste the program code into the **evidence.doc**

[2]

(ii) The procedure `NewBox`:

- takes the Name, Creator, Date Hidden and Game Location of the box as input
- creates an instance of `HiddenBox` and appends it to the end of `TheBoxes`

Write program code for the procedure, `NewBox`.

Save your program. Copy and paste the program code into the **evidence.doc**

[4]

(iii) After declaring the array, the main program calls `NewBox`.

Edit the main program to call `NewBox`.

Save your program. Copy and paste the program code into the **evidence.doc**

[1]

(e) A new type of box is created. The new type of box is a puzzle box.

The class `PuzzleBox`, inherits from `HiddenBox`.

`PuzzleBox` has the additional properties:

- `PuzzleText` as String
- `Solution` as String

The constructor for `PuzzleBox` extends the constructor from `HiddenBox` to also take the values for `PuzzleText` and `Solution`.

Write program code for the class, `PuzzleBox`.

If you are using Python, add a comment for each property to give its identifier and data type. Save your program. Copy and paste the program code into the **evidence.doc**

[3]

BLANK PAGE

3 `QueueData` is a queue that stores up to 20 string values. The queue has a start pointer to identify the first element in the queue, and an end pointer to identify that last element in the queue.

(a) Write a program that defines `QueueData` and its pointers.

Save your program as **program 3**. Copy and paste the program code into the **evidence.doc**

[3]

(b) Write program code for a function, `Enqueue`.

The function should take an item to be added to the queue as a parameter:

- if the element was successfully added to the queue, the function should return `TRUE`
- if the queue was full and the item could not be added to the queue the function should returns `FALSE`
- each time an item is successfully added, the function should updates the pointers.

Save your program. Copy and paste the program code into the **evidence.doc**

[6]

(c) Write program code for a function, `ReadFile`, which:

- asks the user to input a filename
- reads the data from the text file into `QueueData`.

The function returns the following values when:

- all of the data is successfully read into the queue, the function returns the value 2
- the queue is full and not all the data could be inserted into the queue, the function returns the value 1
- the text file could not be found, the function returns the value –1.

Save your program. Copy and paste the program code into the **evidence.doc**

[8]

(d) (i) The main program calls `ReadFile`.

`ReadFile` outputs an appropriate message to identify if:

- the text file could not be found
- the queue was full
- all items were added to the queue.

Edit the main program to call `ReadFile` and output the appropriate message.

Save your program. Copy and paste the program code into the **evidence.doc**

[4]

(ii) You have **two** text files `DataToAdd.txt` and `SecondData.txt`.

Use the filenames below as input to test your program:

- `DataToAdd.txt`
- `SecondData.txt`
- `ThirdData.txt` (this file does not need to be created)

Take a screenshot to show the result from each test. Make sure you show the name of the text file being input in your screenshots.

Save your program. Copy and paste the **three** screenshots into the **evidence.doc**

[3]

(e) The function, `Remove`:

- removes the first two elements from the queue
- concatenates the elements with a space in between them.
  For example, if the first item in the queue is `"Hello"` and the second is `"World"`, the function creates `"Hello World"`
- returns the concatenated string
- returns `"No Items"` if there are insufficient items in the queue.

Write program code for the function `Remove`.

Save your program. Copy and paste the program code into the **evidence.doc**

[5]

BLANK PAGE