Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/42**

Paper 4 Practical **May/June 2022**

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 [ ].

This document has **12** pages. Any blank pages are indicated.

Open the 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.

A source file is used to answer **Question 3**. The file is called **`CardValues.txt`**

1 A program needs to use a stack data structure. The stack can store up to 10 integer elements.

A 1D array `StackData` is used to store the stack globally. The global variable `StackPointer` points to the next available space in the stack and is initialised to 0.

(a) Write program code to declare the array and pointer as global data structures. Initialise the pointer to 0.

Save your program as **Question1_J22**. Copy and paste the program code into **part 1(a)** in the evidence document.

[3]

(b) Write a procedure to output all 10 elements in the stack **and** the value of `StackPointer`.

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

[3]

(c) The function `Push()` takes an integer parameter and returns FALSE if the stack is full. If the stack is not full, it puts the parameter value on the stack, updates the relevant pointer and returns TRUE.

Write program code for the function `Push()`.

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

[6]

(d) (i) Edit the main program to test the `Push()` function. The main program needs to:

- allow the user to enter 11 numbers and attempt to add these to the stack
- output an appropriate message when a number is added to the stack
- output an appropriate message when a number is not added to the stack if it is full
- output the contents of the stack after attempting to add all 11 numbers.

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

[5]

(ii) Test your program from **part 1(d)(i)** with the following 11 inputs:

11 12 13 14 15 16 17 18 19 20 21

Take a screenshot to show the output.

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

[1]

(e) The function `Pop()` returns `-1` if the stack is empty. If the stack is not empty, it returns the element at the top of the stack and updates the relevant pointer.

(i) Write program code for the function `Pop()`.

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

[5]

(ii) After the code you wrote in the main program for **part 1(d)(i)**, add program code to:

- remove two elements from the stack using `Pop()`
- output the updated contents of the stack.

Test your program and take a screenshot to show the output.

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

[2]

2 A 2D array stores data entered by a user.

(a) The main program declares a 2D array of 10 by 10 integer elements.

The array is initialised with a random number between 1 and 100 in each element.

Write program code for the main program.

Save your program as **Question2_J22**.
Copy and paste the program code into **part 2(a)** in the evidence document.

[4]

(b) The following bubble sort pseudocode algorithm sorts the data in the first dimension of the 2D array into ascending numerical order.

```
ArrayLength ← 10
FOR X ← 0 TO ArrayLength - 1
  FOR Y ← 0 TO ArrayLength - 2
    FOR Z ← 0 TO ArrayLength - Y - 2
      IF ArrayData[X, Z] > ArrayData[X, Z + 1] THEN
        TempValue ← ArrayData[X, Z]
        ArrayData[X, Z] ← ArrayData[X, Z+1]
        ArrayData[X, Z + 1] ← TempValue
      ENDIF
    NEXT Z
  NEXT Y
NEXT X
```

(i) Amend your main program by writing program code to implement the bubble sort algorithm after the initialisation of the array elements.

You must **not** use any built-in sorting functions for your programming language.

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

[5]

(ii) Write program code for a procedure to output all the values in the 2D array. The values should be output as a 2D grid, with values in rows and columns.

Call the procedure before and after your bubble sort code.

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

[3]

(iii) Test your program.

Take a screenshot to show the output.

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

[1]

(c) The following pseudocode function uses recursion to perform a binary search in the first row of the array, for the value `SearchValue` in the array `SearchArray`.

The function returns `-1` if the item was not found, or it returns the index where it is found.

There are **six** incomplete statements.

```
FUNCTION BinarySearch(SearchArray, Lower, Upper, SearchValue)RETURNS
                                                                INTEGER
  IF Upper >= Lower THEN
    Mid ← (Lower + (Upper - 1)) DIV ……
    IF SearchArray[0, Mid] = …… THEN
      RETURN ……
    ELSE
      IF SearchArray[0, Mid] > SearchValue THEN
        RETURN BinarySearch(SearchArray, ……, Mid - 1,
                                                        SearchValue)
      ELSE
        RETURN BinarySearch(SearchArray, Mid + 1, ……,
                                                        SearchValue)
      ENDIF
    ENDIF
  ENDIF
  RETURN ……
ENDFUNCTION
```

Note: the arithmetic operator `DIV` performs integer division, e.g. the result of `10 DIV 3` will be `3`.

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

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

[8]

(ii) In the main program, test the function `BinarySearch()` twice, outputting the returned value each time.

One test should be for a number that is in the first line of the array.
One test should be for a number that is not in the first line of the array.

Take a screenshot to show the output.

Copy and paste the screenshot into **part 2(c)(ii)** in the evidence document.

[2]

3 A programmer is designing a computer game that uses a set of cards.

Each card has a number and a colour. The cards are saved in the text file `CardValues.txt`

The program has a class named `Card`. The class has the following attributes and methods.

Card	
`Number : INTEGER`	The number of the card
`Colour : STRING`	The colour of the card
`Constructor()`	Takes two values as parameters and sets them to the private attributes
`GetNumber()`	Returns the number of the card
`GetColour()`	Returns the colour of the card

(a) The constructor takes the number and colour of the card as parameters and sets them to the private attributes.

Write program code to declare the class `Card` and its constructor. Do **not** write any other methods.

Use your programming language appropriate constructor.

All attributes should be private. If you are writing in Python, include attribute declarations using comments.

Save your program as **Question3_J22**.

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

[5]

(b) The two get methods return the associated attribute.

Write program code for the get methods `GetNumber()` and `GetColour()`.

Save your program.

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

[3]

(c) The text file `CardValues.txt` stores the data for 30 cards, in the order: number, colour.

For example, the first card in the text file:

`1` is the number
`red` is the colour.

A 1D array of type `Card` is declared to store all the cards read in from `CardValues.txt`

Write the main program to:

- declare an array of type `Card` with 30 elements
- read in the data for the 30 cards from `CardValues.txt` and assign each to the array.

Save your program.

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

[7]

(d) The program needs to allow all players (maximum of 5) to select 4 cards from the 30 available. A card can only be selected once, so the program needs to record which cards have already been selected.

The function, `ChooseCard()`:

- takes as input an integer to represent an array index from 1 to 30
- validates that the value is between 1 and 30 inclusive
- checks if the card is available (it has not already been selected)
- loops until an available card is selected
- returns the index of the card if it is available.

Amend the program to store which cards have already been selected **and** write program code for the function `ChooseCard()`.

Save your program.

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

[6]

(e) The main program needs to allow one player to select all their 4 cards.

(i) Amend the main program to:

- create an array, `Player1`, for player 1 of type `Card`
- ask player 1 to input 4 integers using the function from **part 3(d)**
- store the cards in `Player1`
- output the number and colour of the 4 cards in `Player1`.

Save your program.

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

[5]

(ii) Test your program with the following test data:

Test 1: **1** **5** **9** **10**

Test 2: **2** **2** **3** **4** **4** **5**

Take a screenshot to show the output.

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

[1]

BLANK PAGE

BLANK PAGE

BLANK PAGE