Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/41**

Paper 4 Practical **October/November 2023**

2 hours 30 minutes

You will need: Candidate source files (listed on page 2)
evidence.doc

8815737793

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 evidence document, **evidence.doc**

Make sure that your name, centre number and candidate number 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 **`QueueData.txt`**

1 This iterative pseudocode algorithm for the function `IterativeVowels()` takes a string as a parameter and counts the number of lower-case vowels in this string.
The vowels are the letters a, e, i, o and u.

```
FUNCTION IterativeVowels(Value : STRING) RETURNS INTEGER
   DECLARE Total : INTEGER
   DECLARE LengthString : INTEGER
   DECLARE FirstCharacter : CHAR
   Total ← 0
   LengthString ← LENGTH(Value)
   FOR X ← 0 TO LengthString - 1
      FirstCharacter ← MID(Value, 0, 1)
      IF FirstCharacter = 'a' OR FirstCharacter = 'e' OR
         FirstCharacter = 'i' OR FirstCharacter = 'o' OR
         FirstCharacter = 'u' THEN
            Total ← Total + 1
      ENDIF
      Value ← MID(Value, 1, LENGTH(Value)-1)
   NEXT X
   RETURN Total
ENDFUNCTION
```

The pseudocode function `MID(X, Y, Z)` returns `Z` number of characters from string `X`, starting at the character in position `Y`. The first character in a string is in position 0, for example:

`MID("computer", 0, 3)` returns `"com"`

The pseudocode function `LENGTH(X)` returns the number of characters in the string `X`, for example:

`LENGTH("computer")` returns 8

(a) (i) Write program code for the function `IterativeVowels()`.

Save your program as **Question1_N23**. Copy and paste the program code into part **1(a)(i)** in the evidence document.

[5]

(ii) Write program code to call the function `IterativeVowels()` with the parameter `"house"` from the main program.

Output the return value.

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

[2]

(iii) Test your program.

Take a screenshot of the output.

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

[1]

(b) (i) Rewrite the function `IterativeVowels()` as a recursive function with the identifier `RecursiveVowels()`.

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

[6]

(ii) Write program code to call the function `RecursiveVowels()` with the parameter `"imagine"` from the main program.

Output the return value.

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

[1]

(iii) Test your program.

Take a screenshot of the output.

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

[1]

2 A linear queue is implemented using the 1D array, `Queue`. The index of the first element in the array is 0.

(a) (i) Write program code to declare:

- `Queue` — a global array with space to store 50 IDs of type string
- `HeadPointer` — a global variable to point to the first element in the queue, initialised to −1
- `TailPointer` — a global variable to point to the next available space in the queue, initialised to 0.

Save your program as **Question2_N23**.

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

[2]

(ii) The procedure `Enqueue()` takes a string parameter.

If the queue is full, the procedure outputs a suitable message. If the queue is not full, the procedure inserts the parameter into the queue and updates the relevant pointer(s).

Write program code for `Enqueue()`.

Save your program.

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

[4]

(iii) The function `Dequeue()` checks if the queue is empty.

If the queue is empty, the function outputs a suitable message and returns the string `"Empty"`.
If the queue is not empty, the function returns the first element in the queue and updates the relevant pointer(s).

Write program code for `Dequeue()`.

Save your program.

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

[4]

(b) A shop sells computer games. Each game has a unique identifier (ID) of string data type.

The text file `QueueData.txt` contains a list of game IDs.

The procedure `ReadData()` reads the data from the text file and inserts each item of data into the array `Queue`.

Write program code for the procedure `ReadData()`.

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

[6]

(c) Some game IDs appear in the text file more than once.

The program needs to total the number of times each game ID appears in the text file.

The record structure `RecordData` has the following fields:

- `ID` — a string to store the game ID
- `Total` — an integer to store the total number of times that game ID appears in the text file.

(i) Write program code to declare the record structure `RecordData`.

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

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

[2]

(ii) The global 1D array `Records` stores up to 50 items of type `RecordData`.

The global variable `NumberRecords` stores the number of records currently in the array `Records` and is initialised to 0.

Write program code to declare `Records` and `NumberRecords`.

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

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

[2]

[Turn over

(iii) The pseudocode algorithm for the procedure `TotalData()`:

- uses `Dequeue()` to remove an ID from the queue
- checks whether a `RecordData` with the returned ID exists in `Records`
- increments the total for that ID in the record if the ID already exists
- creates a new record and stores it in `Records` if the ID does not exist.

```
PROCEDURE TotalData()
   DECLARE DataAccessed : STRING
   DECLARE Flag : BOOLEAN
   DataAccessed ← Dequeue()
   Flag ← FALSE
   IF NumberRecords = 0 THEN
      Records[NumberRecords].ID ← DataAccessed
      Records[NumberRecords].Total ← 1
      Flag ← TRUE
      NumberRecords ← NumberRecords + 1
    ELSE
      FOR X ← 0 TO NumberRecords - 1
         IF Records[X].ID = DataAccessed THEN
            Records[X].Total ← Records[X].Total + 1
            Flag ← TRUE
         ENDIF
       NEXT X
   ENDIF
   IF Flag = FALSE THEN
      Records[NumberRecords].ID ← DataAccessed
      Records[NumberRecords].Total ← 1
      NumberRecords ← NumberRecords + 1
   ENDIF
ENDPROCEDURE
```

Write program code for the procedure `TotalData()`.

Save your program.

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

[5]

(d) The procedure `OutputRecords()` outputs the ID and total of each record in `Records` in the format:

```
ID  1234   Total   4
```

Write program code for `OutputRecords()`.

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

[1]

(e) The main program needs to:

- call `ReadData()`
- call `TotalData()` for each element in the queue
- call `OutputRecords()`.

(i) Write program code for the main program.

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

[2]

(ii) Test your program.

Take a screenshot of the output.

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

[1]

[Turn over

3 A computer game is written using object-oriented programming.

The game has multiple characters that can move around the screen.

The class `Character` stores data about the characters. Each character has a name, a current X (horizontal) position and a current Y (vertical) position.

Character	
`Name : STRING`	stores the name of the character as a string
`XPosition : INTEGER`	stores the X position as an integer
`YPosition : INTEGER`	stores the Y position as an integer
`Constructor()`	initialises `Name`, `XPosition` and `YPosition` to its parameter values
`GetXPosition()`	returns the X position
`GetYPosition()`	returns the Y position
`SetXPosition()`	adds the parameter to the X position validates that the new X position is between 0 and 10000 inclusive
`SetYPosition()`	adds the parameter to the Y position validates that the new Y position is between 0 and 10000 inclusive
`Move()`	takes a direction as a parameter and calls either `SetXPosition` or `SetYPosition` with an integer value

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

Do **not** declare the other methods.

Use your programming language's appropriate constructor.

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

Save your program as **Question3_N23**.

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

[4]

(ii) The get methods `GetXPosition()` and `GetYPosition()` each return the relevant attribute.

Write program code for the get methods.

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

[3]

(iii) The set methods `SetXPosition()` and `SetYPosition()` each take a value as a parameter and add this to the current X or Y position.

If the new value exceeds 10 000, it is limited to 10 000.

If the new value is below 0, it is limited to 0.

Write program code for the set methods.

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

[4]

(iv) The method `Move()` takes a string parameter: `"up"`, `"down"`, `"left"` or `"right"`.

The table shows the change each direction will make to the X or Y position.

Use the appropriate method to change the position value.

Direction	Value change
up	Y position + 10
down	Y position − 10
left	X position − 10
right	X position + 10

Write program code for `Move()`.

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

[4]

(b) Write program code to declare a new instance of `Character` with the identifier `Jack`.

The starting X position is 50 and the starting Y position is 50, the character's name is Jack.

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

[2]

[Turn over

(c) The class `BikeCharacter` inherits from the class `Character`.

BikeCharacter	
`Constructor()`	takes `Name`, `XPosition` and `YPosition` as parameters calls its parent class constructor with the appropriate values
`Move()`	overrides the method `Move()` from the parent class by changing either the X position or the Y position by 20 instead of 10

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

Do **not** declare the other method.

Use your programming language's appropriate constructor.

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

Save your program.

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

[3]

(ii) The method `Move()` overrides the method from the parent class.

The table shows the change each direction will make to the X or Y position.

Direction	Value change
up	Y position + 20
down	Y position − 20
left	X position − 20
right	X position + 20

Write program code for `Move()`.

Save your program.

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

[2]

(d) Write program code to declare a new instance of `BikeCharacter` with the identifier `Karla`.

The starting X position is 100, the starting Y position is 50 and the character's name is Karla.

Save your program.

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

[1]

(e) (i) Write program code to:

- take as input which of the two characters the user would like to move
- take as input the direction the user would like the character to move
- call the appropriate method to move the character
- output the character's new X and Y position in an appropriate format, for example:

```
"Karla's new position is X = 100 Y = 200"
```

All inputs require appropriate prompts and must be validated.

Save your program.

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

[5]

(ii) Test your program twice with the following inputs.

Test 1: jack right

Test 2: karla down

Take a screenshot of the output.

Save your program.

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

[2]

BLANK PAGE