Cambridge Assessment International Education

Cambridge International AS & A Level

COMPUTER SCIENCE **9618/42**

Paper 4 Practical **October/November 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 [ ].

This document has **16** 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 3.** The file is called **`HighScoreTable.txt`**

1 A computer game is designed for users to select characters. Each character can take part in a group of events. Each group has five events. There are four types of event: jump, swim, run, drive.

The program is written using object-oriented programming.

(a) The class `EventItem` stores data about the events.

EventItem	
`EventName : STRING`	stores the name of the event
`Type : STRING`	stores the type of event, either: jump, swim, run or drive
`Difficulty : INTEGER`	stores the difficulty of the event from 1 (easiest) to 5 (hardest)
`Constructor()`	initialises `EventName`, `Type` and `Difficulty` to its parameter values
`GetName()`	returns the name of the event
`GetDifficulty()`	returns the difficulty of the event
`GetEventType()`	returns the type of event

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

Do **not** declare the other methods.

Use your programming language's appropriate constructor.

All attributes must be private.

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

Save your program as **Question1_N24**.

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

[4]

(ii) The get methods `GetName()`, `GetDifficulty()` and `GetEventType()` each return the relevant attribute.

Write program code for the **three** get methods.

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

[3]

(b) The array `Group` stores five objects of type `EventItem`.

(i) Write program code to declare `Group` local to the main program.

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

[1]

(ii) One group has the following events:

Event name	Event type	Event difficulty
Bridge	jump	3
Water wade	swim	4
100 mile run	run	5
Gridlock	drive	2
Wall on wall	jump	4

Write program code to create an instance of `EventItem` for each of the **five** events, and store them in `Group`.

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

[3]

[Turn over

(c) The class `Character` stores data about the characters in the game.

Each character has a skill level for each type of event. The skill level is an integer between 1 and 5 inclusive. Skill level 1 is the lowest skill level, and skill level 5 is the highest skill level.

Character	
`CharacterName : STRING`	stores the name of the character
`Jump : INTEGER`	stores the character's skill level at events of type jump
`Swim : INTEGER`	stores the character's skill level at events of type swim
`Run : INTEGER`	stores the character's skill level at events of type run
`Drive : INTEGER`	stores the character's skill level at events of type drive
`Constructor()`	initialises `CharacterName`, `Jump`, `Swim`, `Run` and `Drive` to its parameter values
`GetName()`	returns the name of the character
`CalculateScore()`	takes the type of event and difficulty as parameters. Calculates and returns the chance of the character completing the event

Write program code to declare the class `Character`, its constructor and get method.

Use your programming language's appropriate constructor.

All attributes must be private.

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

Save your program.

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

[4]

(d) The method `CalculateScore()` in the `Character` class calculates and returns the percentage chance of a character completing an event.

When a character's skill level is greater than or equal to the difficulty of that event, the percentage chance of completing the event is 100%.

When a character's skill level is less than the difficulty of that event, the character's skill level is subtracted from the difficulty of that event. This difference is used to identify the percentage chance of success using this table:

Difference	Percentage chance of success
1	80
2	60
3	40
4	20

For example:

- A character has a skill level of 3 for events of type run.
- An event of type run has a difficulty level of 5
- The character's skill level is less than the difficulty, therefore the difference is calculated.
- The difference is the character's skill level subtracted from the event difficulty, 5 – 3 = 2
- The difference is 2, therefore the percentage chance of succeeding is 60%

Write program code for the method `CalculateScore()` to:

- take the type of event and difficulty as parameters
- calculate the percentage chance of the character completing the event
- return the percentage chance of completing the event as an integer number, for example 60

Save your program.

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

[4]

(e) Two characters are attempting each event in the group you created in part **1(b)**.

One character has the name Tarz and the skill levels:

Jump 5

Swim 3

Run 5

Drive 1

The second character has the name Geni and the skill levels:

Jump 2

Swim 2

Run 3

Drive 4

Each `Character` object is stored in a variable.

(i) Amend the main program to declare and create an instance of a `Character` object for Tarz and for Geni.

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

[2]

(ii) Both characters Tarz and Geni take part in each event in the group you created in part **1(b)**.

These steps are repeated for all **five** events:

- The percentage chance of each character completing the event is calculated and compared.
- The character with the highest percentage chance of completing the event gets 1 point. Their character name is output together with a message telling them they have won that event.
- If both characters have the same percentage chance of completing the event, the scores do **not** change, and a message is output stating that the event is a draw.

When the total score for each character has been calculated, the name of the character with the highest score is output stating they have won and the number of points they have. If both characters have the same number of points, a message is output telling them the group is a draw.

Amend the main program to:

- calculate the score for each character in each event
- output the name of the character that wins each event or that the event is a draw
- output the name of the character that has the most points for the group or that the group is a draw.

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

[7]

(iii) Test your program.

Take a screenshot of the output.

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

[1]

2 A linear queue data structure is designed using a record structure.

The record structure `Queue` has the following fields:

- `QueueArray`, a 1D array of up to 100 integer values
- `Headpointer`, a variable that stores the index of the first data item in `QueueArray`
- `Tailpointer`, a variable that stores the index of the next free location in `QueueArray`.

(a) Write program code to declare the record structure `Queue` and its fields.

If your programming language does **not** support record structures, a class can be declared instead.

If you are writing in Python, use comments to declare the appropriate data types.

Save your program as **Question2_N24**. Copy and paste the program code into part **2(a)** in the evidence document.

[3]

(b) The main program creates a new `Queue` record with the identifier `TheQueue`. The head pointer is initialised to –1. The tail pointer is initialised to 0. Each element in the array is initialised with –1.

Write program code for the main program.

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

[3]

(c) The pseudocode function `Enqueue()` inserts an integer value into the queue.

The function is incomplete. There are **three** incomplete statements.

```
FUNCTION Enqueue(BYREF AQueue : Queue, BYVAL TheData : INTEGER)
                 RETURNS INTEGER

   IF AQueue.Headpointer = -1 THEN

      AQueue.QueueArray[AQueue.Tailpointer] ← .....

      AQueue.Headpointer ← 0

      AQueue.Tailpointer ← AQueue.Tailpointer + 1

      RETURN 1

   ELSE

      IF AQueue.Tailpointer > ..... THEN

         RETURN -1

      ELSE

         AQueue.QueueArray[AQueue.Tailpointer] ← TheData

         AQueue.Tailpointer ← AQueue.Tailpointer .....

         RETURN 1

      ENDIF

   ENDIF

ENDFUNCTION
```

Write program code for `Enqueue()`.

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

[5]

(d) The function `ReturnAllData()` accesses `TheQueue`. It concatenates all the integer values that have been inserted into the queue's array, starting from the value stored at `HeadPointer`, with a space between each integer value. The string of concatenated values is returned.

None of the integer values are removed from the queue.

Write program code for `ReturnAllData()`.

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

[3]

(e) (i) The main program asks the user to enter 10 integers with values of 0 or greater. It reads each input repeatedly until a valid number is entered.

All 10 valid inputs are added to the queue, using `Enqueue()`.

If the value returned from `Enqueue()` is –1, a message is output to state that the queue is full, otherwise a message is output to state that the item has been added to the queue.

The function `ReturnAllData()` is called once all 10 integers have been entered and the return value from the function call is output.

Amend the main program to perform these actions.

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

[5]

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

10 9 –1 8 7 6 5 4 3 2 1

Take a screenshot of the output.

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

[2]

(f) The function `Dequeue()` accesses `TheQueue`. The function returns –1 if the queue is empty. If the queue is **not** empty, the function returns the next item in the queue and updates the relevant pointer(s).

The data is **not** replaced or deleted from the queue.

Write program code for `Dequeue()`.

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

[4]

(g) (i) The main program calls `Dequeue()` twice, and each time it either outputs 'Queue empty' if there is no data in the queue or outputs the return value.

The main program then calls `ReturnAllData()` a second time.

Amend the main program.

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

[3]

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

10 9 8 7 6 5 4 3 2 1

Take a screenshot of the output.

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

[1]

3 The text file `HighScoreTable.txt` stores the top seven player scores for a game.

The data in the file is stored in the order:

Player ID

Game level

Score

Each item of data is stored on a new line. For example, the first set of data in the file is:

Player ID: GHEH

Game level: 3

Score: 10

(a) The data about the players and their scores is stored in a 2D array of strings with the identifier `HighScores`.

The first dimension of the array has seven elements: one for each player. The second dimension of the array has three elements: one for the player ID, one for the game level and one for the score. All data is stored in the array as strings.

`HighScores` is declared local to the main program, and all elements are initialised to an empty string, for example `""`.

Write program code to declare and initialise `HighScores`.

Save your program as **Question3_N24**.
Copy and paste the program code into part **3(a)** in the evidence document.

[2]

(b) The function `ReadData()` reads the data from `HighScoreTable.txt` and stores the data in a 2D array. The function returns the 2D array.

The function uses exception handling when opening and reading data from the file.

Write program code for `ReadData()`.

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

[5]

(c) The procedure `OutputHighScores()` takes a 2D array as a parameter. It outputs each player's ID, level and score in the order they are in the array. The outputs are in the format:

GHEH reached level 3 with a score of 10

Write program code for `OutputHighScores()`.

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

[2]

(d) The scores in `HighScoreTable.txt` are **not** in order.

The player scores are first sorted by the game level they reached. For example, all players that reached level 5 are higher in the high score table than players that reached level 4.

The players that reached the same level are then sorted in descending order of score.

An example sorted high score table is:

Position	Player ID	Game level	Score
1	GFED	5	25
2	HJKM	4	21
3	RERR	4	19
4	TTYU	4	15
5	WSVG	3	20
6	PPTR	3	15
7	SNQT	2	10

The function `SortScores()` uses a bubble sort to sort the data as described and returns the sorted array.

Write program code for `SortScores()`.

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

[4]

(e) (i) Amend the main program to implement these steps in the order given:

- call `ReadData()` and store the returned array in `HighScores`
- output `"Before"`
- call `OutputHighScores()` with `HighScores` as a parameter
- call `SortScores()` and store the returned array in `HighScores`
- output `"After"`
- call `OutputHighScores()` with `HighScores` as a parameter.

Save your program.

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

[2]

BLANK PAGE

BLANK PAGE