ALPHA COLLEGE

| | |
|---|---|
| **Candidate Name** | |
| **Student ID** | |

---

**COMPUTER SCIENCE** **9618**

**MOCK Examination – A2 Level**

**Paper 4 Practical**

**March/April 2024**

**2 Hours 30 Minutes**

**Maximum Marks 75**

---

**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 [ ].

**Question No. 1**

A teacher is designing a program to perform simple syntax checks on programs written by students.

Two global 1D arrays are used to store the syntax error data. Both arrays contain 10 elements.

- Array `ErrCode` contains integer values that represent an error number in the range 1 to 80.
- Array `ErrText` contains string values that represent an error description.

The following diagram shows an example of the arrays.

| Index | ErrCode | ErrText |
|---|---|---|
| 0 | 10 | "Invalid identifier name" |
| 1 | 20 | "Bracket mismatch" |
| 2 | 25 | "Undeclared variable" |
| 3 | 30 | "Missing colon" |
| 4 | 40 | "Missing semicolons" |
| 5 | 45 | "Quotation mark mistakes" |
| 6 | 50 | "Incorrect use of operators" |
| 7 | 60 | "Incorrect path" |
| 8 | 80 | "Data type mismatch" |
| 9 | 99 | "" |

**Note:**

- There may be unused elements in both arrays. Unused elements in `ErrCode` have the value `99`, the corresponding value of unused elements in `ErrText` is undefined.

- Values in the `ErrCode` array are stored in ascending order but not all values may be present, for example, there may be no error code `35`.

The teacher has defined two program modules as follows:

| Module | Description |
|---|---|
| `OutputError()` | • takes two parameters as integers:<br>○ a line number in the student's program<br>○ an error number<br>• searches for the error number in the `ErrCode` array:<br>○ if found, outputs the corresponding error description and the line number, for example: `"Bracket mismatch on line 34"`<br>○ if not found, outputs the line number and a warning, for example: `"Unknown error on line 34"` |
| `SortArrays()` | sorts the arrays into ascending order of `ErrCode` |

**(a)** Write program code for a **new program** to:

- declare the global 1D arrays, `ErrCode` and `ErrText`, with ten elements
- initialise `ErrCode` and `ErrText` in the main program using the data values shown

Save your program as **question1**.
Copy and paste the program code into **part 1(a)** in the evidence document.
[3]

**(b)** Write program code for **module** `OutputError()`.

Save your program.
Copy and paste the program code into **part 1(b)** in the evidence document.
[6]

**(c)** Write bubble sort algorithm in program code for **module** `SortArrays()`.

Save your program.
Copy and paste the program code into **part 1(c)** in the evidence document.
[5]

**(d)** The following pseudocode algorithm performs a binary search on the sorted array `ErrCode`.

There are **six** incomplete statements.

The algorithm returns either the location of `SearchItem` in the array, or `-1` if is not in the array.

The function `DIV` returns the integer value of the division, for example, `11 DIV 2` returns `5`.

```
FUNCTION BinarySearch(ThisArray[], LowerBound, UpperBound,
                      SearchItem : INTEGER) RETURNS INTEGER

DECLARE Flag : INTEGER

DECLARE Mid : INTEGER

Flag ← -2

WHILE Flag <> -1

   Mid ← LowerBound + ((UpperBound – LowerBound) DIV 2)

   IF .................................... < ....................................

      THEN

         RETURN ....................................

      ELSE

         IF ThisArray[Mid] > SearchItem

            THEN

              UpperBound ← Mid ....................................

            ELSE

              IF ThisArray[Mid] < SearchItem

                 THEN

                    LowerBound ← Mid ....................................

                 ELSE

                    RETURN ....................................

              ENDIF

         ENDIF

   ENDIF

ENDWHILE

ENDFUNCTION
```

Write **recursive** algorithm for the `BinarySearch()` function in program code, to search an item in array `ErrCode`.

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

[6]

**(e) (i)** Write program code for the main program to:

- call the procedure `SortArrays()`
- allow the user to input an integer value (an error code)
- pass the value to `BinarySearch()` as the parameter
- output an appropriate message.

Save your program.
Copy and paste the program code into **part 1(e)(i)** in the evidence document.
[4]

**(ii)** Test your program by inputting one error code that is in the array and one error code that is not in the array.

Save your program.
Copy and paste the screenshots into **part 1(e)(ii)** in the evidence document.
[2]

**(f)** Two 1D arrays were described at the beginning of the question. Both arrays contain 10 elements.

- Array `ErrCode` contains integer values that represent an error number in the range 1 to 80.
- Array `ErrText` contains string values that represent an error description.

The two arrays will be replaced by a single array. A user-defined data type (record structure) has been declared as follows in pseudocode:

```
TYPE ErrorRec
   DECLARE ErrCode : INTEGER
   DECLARE ErrText : STRING
ENDTYPE
```

Write program code to declare the record type `ErrorRec`.

Save your program.
Copy and paste the program code into **part 1(f)** in the evidence document.
[3]

**Question No. 2**

**(a)** The number of cars that cross a bridge is recorded each hour. This number is placed in a circular queue before being processed.

The queue is stored as an array, `NumberQueue`, with eight elements. The function `AddToQueue` adds a number to the queue. `EndPointer` and `StartPointer` are global variables.

**(i)** Write program code for a **new program** to:

- declare the global 1D array, `NumberQueue`, with eight elements
- declare and initialise `EndPointer` and `StartPointer`

Save your program as **question2**.
Copy and paste the program code into **part 2(a)(i)** in the evidence document.
[3]

**(ii)** The following **pseudocode** algorithm is for the function `AddToQueue`.

There are **five** incomplete statements.

```
FUNCTION AddToQueue(Number : INTEGER) RETURNS BOOLEAN
    DECLARE TempPointer : INTEGER
    CONSTANT FirstIndex = 0
    CONSTANT LastIndex = ....7..................................................
    TempPointer ← EndPointer + 1
    IF ..............TempPointer.................................. > LastIndex
        THEN
            TempPointer ← ..............FirstIndex.........................
    ENDIF
    IF TempPointer = StartPointer
        THEN
            RETURN ......FALSE.............................................
        ELSE
            EndPointer ← TempPointer
            NumberQueue[EndPointer] ← .......Number.......................................
            RETURN TRUE
    ENDIF
ENDFUNCTION
```

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

Save your program.
Copy and paste the program code into **part 2 (a)(ii)** in the evidence document.
[5]

**(b)** Sandy is writing a program to process data in a stack. The stack is implemented as a 1D array, `NumberQueue`, declared in Question 2(a) (i), which has up to 8 elements.

The function `Push(Value)` stores `Value` on the stack and returns `TRUE` if `Value` was added to the stack, or `FALSE` if the stack is full.

The function `Pop()` returns the item at the top of the stack, or returns `-1` if the stack is empty.

`NumberQueue` and `TopPointer` are declared as global.

**(i)** Write **program code** to declare and initialise `TopPointer`.

Save your program.
Copy and paste the program code into **part 2(b)(i)** in the evidence document.
[1]

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

Save your program.
Copy and paste the program code into **part 2(b)(ii)** in the evidence document.
[4]

**(iii)** Write **program code** for the function `Pop()`.

Save your program.
Copy and paste the program code into **part 2(b)(iii)** in the evidence document.
[4]

**Question No. 3**

A computer games club wants to run a competition. The club needs a system to store the scores achieved in the competition.

Players complete one game to place them into a category for the competition. The games club wants to implement a program to place players into the correct category. The programmer has decided to use object-oriented programming (OOP).

The highest score that can be achieved in the game is 150. Any score less than 50 will not qualify for the competition. Players will be placed in a category based on their score.

The following diagram shows the design for the class `Player`. This includes the properties and methods.

```
                              Player
--------------------------------------------------------------------
Score    : INTEGER // initialised to 0
Category : STRING  // "Beginner", "Intermediate",
                   // "Advanced" or "Not Qualified", initialised
                   // to "Not Qualified"
PlayerID : STRING  // initialised with the parameter InputPlayerID
--------------------------------------------------------------------
Create()         // method to create and initialise an object using
                 // language-appropriate constructor
SetScore()       // checks that the Score parameter has a valid value
                 // if so, assigns it to Score
SetCategory()    // sets Category based on player's Score
SetPlayerID()    // allows a player to change their PlayerID
                 // validates the new PlayerID
GetScore()       // returns Score
GetCategory()    // returns Category
GetPlayerID()    // returns PlayerID
```

**(a)** The constructor receives the parameter `InputPlayerID` to create the `PlayerID`. Other properties are initialised as instructed in the class diagram.

Write **program code** for the **constructor** method.

Save your program as **question3**.
Copy and paste the program code into **part 3(a)** in the evidence document.
[5]

**(b)** Write **program code** for the following **three** get methods.

**(i)** `GetScore()`

Save your program.
Copy and paste the program code into **part 3(b) (i)** in the evidence document.
[1]

**(ii)** `GetCategory()`

Save your program.
Copy and paste the program code into **part 3(b) (ii)** in the evidence document.
[1]

**(iii)** `GetPlayerID()`

Save your program.
Copy and paste the program code into **part 3(b) (iii)** in the evidence document.
[1]

**(c)** The method `SetPlayerID()` asks the user to input the new player ID and reads in this value.

It checks that the length of the `PlayerID` is less than or equal to 15 characters and greater than or equal to 4 characters. If the input is valid, it sets this as the `PlayerID`, otherwise it loops until the player inputs a valid `PlayerID`.

Use suitable input and output messages.

Write **program code** for `SetPlayerID()`.

Save your program.
Copy and paste the program code into **part 3(c)** in the evidence document.
[4]

**(d)** The method `SetScore()` checks that its `INTEGER` parameter `ScoreInput` is valid. If it is valid, it is then set as `Score`. A valid `ScoreInput` is greater than or equal to 0 and less than or equal to 150.

If the `ScoreInput` is valid, the method sets `Score` and returns `TRUE`.
If the `ScoreInput` is not valid, the method does not set `Score`, displays an error message, and it returns `FALSE`.

Write **program code** for `SetScore(ScoreInput : INTEGER)RETURNS BOOLEAN`.

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

[5]

**(e)** Write **program code** for the method `SetCategory()`. Use the properties and methods in the original class definition.

Players will be placed in one of the following categories.

| Category | Criteria |
|---|---|
| Advanced | Score is greater than 120 |
| Intermediate | Score is greater than 80 and less than or equal to 120 |
| Beginner | Score is greater than or equal to 50 and less than or equal to 80 |
| Not Qualified | Score is less than 50 |

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

[4]

**(f)** Joanne has played the first game to place her in a category for the competition. The procedure `CreatePlayer()` performs the following tasks.

- allows the player ID and score to be input with suitable prompts
- creates an instance of `Player` with the identifier `JoannePlayer`
- sets the score for the object
- sets the category for the object
- outputs the category for the object

Write **program code** for the `CreatePlayer()` procedure.

Save your program.
Copy and paste the program code into **part 3(f)** in the evidence document.
[6]

**(g)** Amend the main program to call the procedure `CreatePlayer()`.

Test your program using the following inputs:

- player ID `"ALPHA001"`
- score `100`

Save your program.
Copy and paste the screenshot into **part 3(g)** in the evidence document.
[2]