8. Programming

8.2 Arrays

Marking Scheme

Q1)

Question	Answer	Marks
	One mark for explanation of dimension **One** mark for explanation of index **One** mark for inclusion of an example The dimension is the number of indexes required to access an element. The index is the position of the element in an array For example A[25] is the 25th element of a one-dimensional array.	3

Q2)

Question	Answer	Marks

(a)	**One** mark for:
	• Use of `FOR` loop
	• Working loop with correct number of Iterations
	• Correct assignment
	`FOR Count ← 1 TO 20`
	`  dataArray[Count] ← 0`
	`NEXT (Count)`
		3

(b)	(A FOR loop has) a fixed number of repetitions // No need to manage the loop counter // no need to use another variable for the array index	1

Q3)

Question	Answer	Marks
(a)	**One** mark per mark point, max **two** • a list / one column of items • … of the same data type • … stored under a **single** identifier • … with a single index to identify each element **One** mark for an example of a declaration • example e.g. `DECLARE MyArray[1:10] OF INTEGER`	3

Question	Answer	Marks
(b)	**One** mark per mark point, max **two** • using a counter to index the array • so that the same code can be repeatedly used to check every element // every element can be checked in a loop	2

Q4)

- `FOR (… TO … NEXT)`
- `REPEAT (… UNTIL)`
- `WHILE (… DO … ENDWHILE)` [3]

Q5)

(a) **1** mark for `FOR … TO … NEXT` **1** mark for `INPUT`

```
FOR Count ← 1 TO 1000
  INPUT A[Count]
NEXT (Count)
```
[2]

(b) 4 marks
– initialisation
– start of loop
– update loop counter
– end of loop

Example1

```
Count ← 1                    (1 mark)
REPEAT                       (1 mark)
  INPUT A[Count]
  Count ← Count + 1          (1 mark)
UNTIL Count > 1000           (1 mark)
```

Example2

```
Count ← 0                    (1 mark)
WHILE Count < 1000           (1 mark)
  DO
  Count ← Count + 1          (1 mark)
  INPUT A[Count]
ENDWHILE                     (1 mark)
```
[4]

Q6)

– FOR (… TO … NEXT)…
– … a set number of iterations
– WHILE (… DO … ENDWHILE) …
– … used where the loop may never be executed/whilst a specified condition exists

[4]

Q7)

For each example **1 mark** for **correct structure**, **1 mark** for **appropriate content** inside loop and **1 mark** for **reason.** There are many correct answers these are only samples

```
REPEAT
..INPUT Number
  Total ← Total + Number
UNTIL Number = 0
```

– at least one repeat is required

```
WHILE Number <> -1 DO
..INPUT Number
   Total ← Total + Number
ENDWHILE
```

– the loop may never be executed [6]

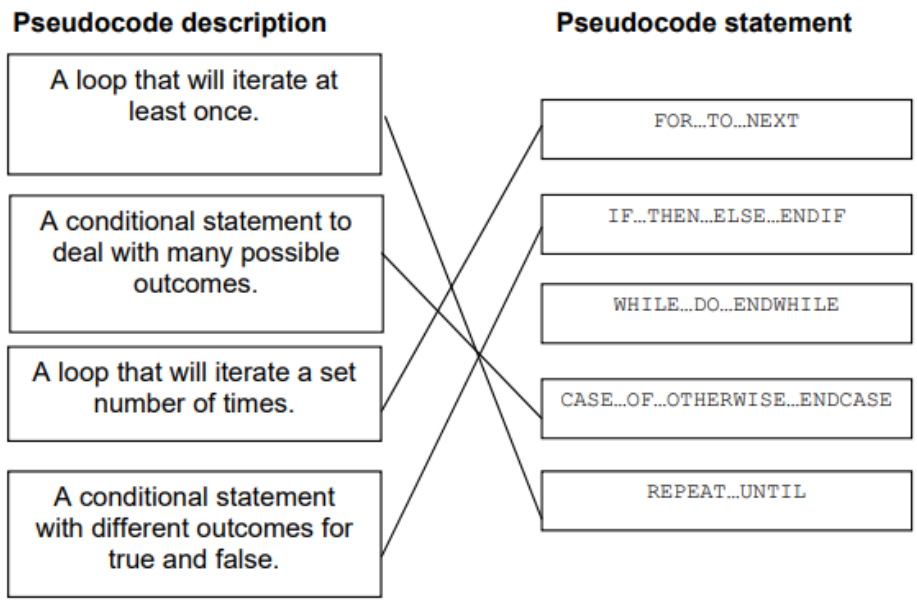
Q8)

Question	Answer	Marks
(a)	Any **two** from: - Loop with 300 repetitions (starting at 1) / Loops from 1 to 300 - Values input/stored (in consecutive/different locations) in an array (at position I) - Increases the loop counter/I value by 1 (and returns to the start of the loop)	**2**

(b)	Any **one** from:
	`REPEAT (… UNTIL)`
	`WHILE (… DO … ENDWHILE)`
		1

(c)	- Prompt and input number (1)
	- Checking the input number is between 0 and 100 - both limits (1)
	- Correct error message (1)
	
	Many correct algorithms. This is an example only.
	
	`OUTPUT "Enter a number between 0 and 100 "`
	`INPUT Number`
	`IF Number < 0 OR Number > 100`
	`  THEN`
	`  OUTPUT "The number you have entered is outside the specified range"`
	`ENDIF`
		3

Q9)

Question	Answer	Marks
(a)	1 mark for each correct line  	4

`(b)	1 mark per bullet:
	∞ Appropriate loop controls
	∞ Read from array
	∞ Print from array (the last two points can be in one statement)
	Note reading and printing **MUST** be within the same loop
	Example algorithm:
	`Count ← 0`
	`WHILE Count < 50 DO`
	`  OUTPUT Name[Count]`
	`  Count ← Count + 1`
	`ENDWHILE`
		3

Q10)

Question	Answer	Marks
	∞ FOR … TO … NEXT ∞ fixed number of repetitions ∞ REPEAT … UNTIL ∞ always executed // condition tested at end ∞ WHILE … DO … ENDWHILE ∞ may not be executed // condition tested at beginning	6

Q11)

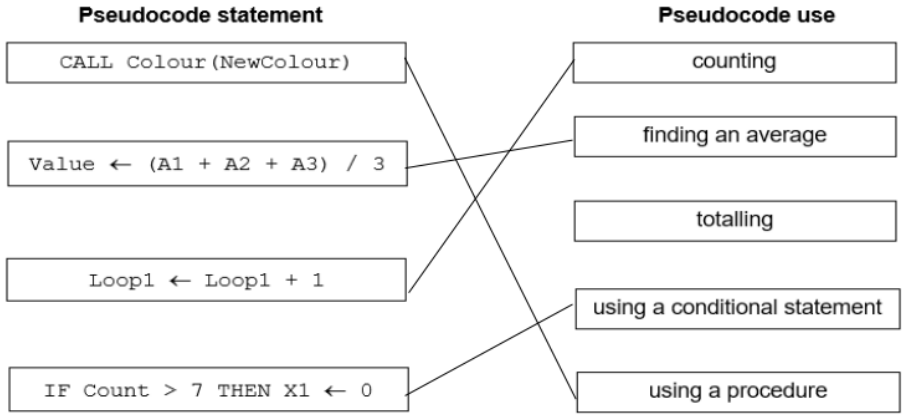
Question	Answer	Marks
	• FOR (… TO … NEXT) loop • WHILE (… DO … ENDWHILE) loop • REPEAT (… UNTIL) loop	3

Q12)

Question	Answer	Marks

	Any **six** from:
	MP1 Initialisation of `Higher` to `0` before the loop
	MP2 Use of IF statement
	MP3 Correct condition in IF statement
	MP4 Correct counting statement inside loop
	MP5 `OUTPUT/PRINT` statement with correct reference to `Higher`
	MP6 Appropriate message in output
	MP7 Correct location of `OUTPUT` and `IF` statements
	
	`Higher ← 0`
	`FOR Count ← 1 TO 5000`
	`  INPUT Number[Count]`
	`  IF Number[Count] > 500`
	`  THEN`
	`  Higher ← Higher + 1`
	`  ENDIF`
	`NEXT Count`
	`OUTPUT "There are ", Higher, " values that are greater than 500"`
		6

Q13)

Question	Answer	Marks
(a)	**One** mark for each correct line.  	4

Question	Answer	Marks

(b)	**One** mark per mark point, max **four**
	MP1 initialise a variable to store the lowest number set to a high value (at least 100) / first value in the array
	MP2 loop structure to iterate 25 times
	MP3 use of `IF` to check if the array element is less than the current lowest value
	MP4 …if it is, set this to the lowest value
	MP5 output the result (with an appropriate message) after the loop
	
	Example answer:
	`Min ← 100`
	`FOR Count ← 1 TO 25`
	`  IF Temperatures[Count] < Min`
	`  THEN`
	`  Min ← Temperatures[Count]`
	`  ENDIF`
	`NEXT Count`
	`OUTPUT "The lowest temperature is ", Min`
		4

Q14)

Question	Answer	Marks

	One mark per mark point, max **five**
	MP1 storing string in `Quote`
	MP2 correct assignment for `Start` // sending correct start value
	MP3 correct assignment for `Number` // sending correct number of characters
	MP4 use of `SUBSTRING` function with first parameter as `Quote`, or equivalent, plus two other parameters
	MP5 correct use of `LCASE` function
	MP6 two correct outputs
	
	For example:
	`Quote ← "Learning Never Exhausts The Mind"`
	`Start ← 25`
	`Number ← 8`
	`OUTPUT SUBSTRING(Quote, Start, Number)`
	`OUTPUT LCASE(Quote)`
		5

Pseudocode, Program code and Arrays

Q1)

Question	Answer	Marks
	Read the whole answer: Check if each requirement listed below has been met. Requirements may be met using a suitable built-in function from the programming language used (Python, VB.NET or Java). On place a SEEN mark if requirement met, cross if no attempt seen, omission mark and/or comment if partially met (see marked scripts). Use the tables for AO2 and AO3 below to award a mark in a suitable band using a best fit approach, then add up the total: • AO2 (maximum 9 marks) • AO3 (maximum 6 marks) **Data structures required:** The names underlined must match those given in the scenario: Arrays or lists `Days[]`, `Readings[]`, `AverageTemp[]` Variables `WeekLoop`, `DayLoop`, `InTemp`, `TotalDayTemp`, `TotalWeekTemp`, `AverageWeekTemp` **Requirements (techniques):** **R1** Input and store hourly temperatures and validation of input temperatures for each day (with prompts, range check and (nested)iteration) **R2** Calculate, round to one decimal place and store daily average temperatures and calculate the weekly average temperature rounded to one decimal place (iteration, totalling and rounding) **R3** Convert all average temperatures to Fahrenheit (to one decimal place) and output the average temperatures in both Celsius and Fahrenheit. Output with appropriate messages. (output and rounding)	**15**

Question	Answer	Marks

	Example 15 mark answer in pseudocode
	
	`// meaningful identifiers and appropriate data structures for`
	`// all data required`
	`DECLARE Days : ARRAY[1:7] OF STRING`
	`DECLARE Readings : ARRAY[1:7, 1:24] OF REAL`
	`DECLARE AverageTemp : ARRAY[1:7] OF REAL`
	`DECLARE WeekLoop : INTEGER`
	`DECLARE DayLoop : INTEGER`
	`DECLARE InTemp : REAL`
	`DECLARE TotalDayTemp : REAL`
	`DECLARE TotalWeekTemp : REAL`
	`DECLARE AverageWeekTemp : REAL`
	`// initial population of Days[] array`
	`// input and a loop are also acceptable`
	`Days[1] ← "Sunday"`
	`Days[2] ← "Monday"`
	`Days[3] ← "Tuesday"`
	`Days[4] ← "Wednesday"`
	`Days[5] ← "Thursday"`
	`Days[6] ← "Friday"`
	`Days[7] ← "Saturday"`
	`// input temperatures inside nested loop`
	`FOR WeekLoop ← 1 TO 7`
	`  TotalDayTemp ← 0`
	`  FOR DayLoop ← 1 TO 24`
	`  OUTPUT "Enter temperature ", DayLoop, " for ", Days[WeekLoop]`	

Question	Answer	Marks

	INPUT InTemp
	// validation of input for between -20 and +50 inclusive
	WHILE InTemp < -20.0 OR InTemp > 50.0 DO
	OUTPUT "Your temperature must be between -20.0 and +50.0 inclusive. Please try again"
	INPUT InTemp
	ENDWHILE
	Readings[WeekLoop, DayLoop] ← InTemp
	// totalling of temperatures during the day
	TotalDayTemp ← TotalDayTemp + ROUND(InTemp, 1)
	NEXT DayLoop
	
	// average temperature for the day
	AverageTemp[WeekLoop] ← ROUND(TotalDayTemp / 24,1)
	NEXT WeekLoop
	// calculate the average temperature for the week
	TotalWeekTemp ← 0
	FOR WeekLoop ← 1 TO 7
	TotalWeekTemp ← TotalWeekTemp + AverageTemp[WeekLoop]
	NEXT WeekLoop
	
	AverageWeekTemp ← ROUND(TotalWeekTemp / 7,1)
	// outputs in Celsius and Fahrenheit
	FOR WeekLoop ← 1 TO 7
	OUTPUT "The average temperature on ", Days[WeekLoop], " was ", AverageTemp[WeekLoop], " Celsius and ",
	ROUND(AverageWeekTemp * 9 / 5 + 32), 1, " Fahrenheit"
	NEXT WeekLoop
	
	OUTPUT "The average temperature for the week was ",
	AverageWeekTemp," Celsius and ", ROUND(AverageWeekTemp * 9 / 5 + 32, 1),"
	Fahrenheit"	

Q2)

Question	Answer	Marks

	Read the whole answer:
	Check if each requirement listed below has been met. Requirements may be met using a suitable built-in function from the programming language used (Python, VB.NET or Java)
	Mark SEEN on script if requirement met, cross if no attempt seen, NE if partially met (see marked scripts).
	Use the tables for A02 and A03 below to award a mark in a suitable band using a best fit approach
	Then add up the total.
	Marks are available for:
	• AO2 (maximum 9 marks)
	• AO3 (maximum 6 marks)
	
	Data Structures required names shown underlined must be used as given in the scenario
	Arrays or lists Account, AccDetails
	Variable Size, AccountNumber
	
	Requirements (techniques)
	R1 Check account number and password (iteration and validation, selection, input, output)
	R2 Display menu and make a selection (output, input and selection)
	R3 Perform actions selected (use of arrays and procedures with parameters)
	
	Example 15 mark answer in pseudocode
	`// Procedures to be called`
	`PROCEDURE CheckDetails(AccID : INTEGER)`
	`  DECLARE Name, Password : STRING // local variables`
	`  Valid ← FALSE`
	`  IF AccID <0 OR AccID > Size`
	`  THEN`
	`  OUTPUT "Invalid Account Number"`
	`  ELSE`
	`  OUTPUT "Please Enter Name "`
	`  INPUT Name`
	`  OUTPUT "Please Enter Password "`
	`  INPUT Password`
	`  IF Name <> Account[AccID,1] OR Password <> Account[AccID,2]`
	`  THEN`
	`  OUTPUT "Invalid name or password"`
	`  ELSE`
		15

Question	Answer	Marks

```
            Valid ← True
        ENDIF
    ENDIF
ENDPROCEDURE

PROCEDURE Balance(AccID : INTEGER)
    OUTPUT "Your balance is ", AccDetails[AccID,1]
ENDPROCEDURE

PROCEDURE WithDrawal(AccID : INTEGER)
    DECLARE Amount : REAL // local variable
    REPEAT
        OUTPUT "Please enter amount to withdraw "
        INPUT Amount
        IF Amount > AccDetails[AccID,3]
          THEN
            OUTPUT "Amount greater than withdrawal limit"
        ENDIF
        IF Amount > AccDetails[AccID,2] + AccDetails[AccID,1]
          THEN
            OUTPUT "Amount greater than cash available"
        ENDIF
        IF Amount <= AccDetails[AccID,3] AND Amount < AccDetails[AccID,2] +
          AccDetails[AccID,1]
          THEN
            AccDetails[AccID,1] ← AccDetails[AccID,1] - Amount
        ENDIF
    UNTIL Amount<= AccDetails[AccID,3] AND Amount > AccDetails[AccID,2] +
          AccDetails[AccID,1] AND Amount > 0
ENDPROCEDURE

PROCEDURE Deposit(AccID : INTEGER)
    DECLARE Amount : REAL // local variable
    REPEAT
        OUTPUT "Please enter a positive amount to deposit "
        INPUT Amount
    UNTIL Amount >0
    AccDetails[AccID,1] ← AccDetails[AccID,1] + Amount
```

Question	Answer	Marks

```
ENDPROCEDURE

// Declarations of global variables for information - not required in candidate responses
DECLARE AccountNumber, Choice : INTEGER
DECLARE Valid, Exit : BOOLEAN

OUTPUT "Please enter your account number "
INPUT AccountNumber
CheckDetails(AccountNumber)

IF Valid
  THEN
    REPEAT
        OUTPUT "Menu"
        OUTPUT "1. display balance"
        OUTPUT "2. withdraw money"
        OUTPUT "3. deposit money"
        OUTPUT "4. exit"
        OUTPUT "please choose 1, 2, 3 or 4"
        INPUT Choice
        CASE OF Choice
          1 : Balance(AccountNumber)
          2 : Withdrawal(AccountNumber)
          3 : Deposit(AccountNumber)
          4 : Exit ← TRUE
          OTHERWISE OUTPUT "Invalid choice"
        ENDCASE
    UNTIL Choice = 4
  ELSE
    OUTPUT "Invalid account number "
ENDIF
```

Q3)

Question	Answer	Marks
	Read the whole answer: Check if each requirement listed below has been met. Requirements may be met using a suitable built-in function from the programming language used (Python, VB.NET or Java). Mark SEEN on script if requirement met, cross if no attempt seen, NE if partially met (see marked scripts). Use the tables for A02 and A03 below to award a mark in a suitable band using a best fit approach. Then add up the total. Marks are available for: • AO2 (maximum 9 marks) • AO3 (maximum 6 marks) **Data structures required:** The names underlined must match those given in the scenario: Arrays or lists `Contacts[]` Variables `CurrentSize`, `Cont`, `Choice`, `NewContacts`, `Count`, `Count2`, `Flag` **Requirements (techniques):** **R1** Output menu and input choice, with validation (range check, output with messages, input with prompts). **R2** Input number of new entries, within limits, update current size of contacts, input new data and sort the array (range check, totalling, iteration and bubble sort). **R3** Output array whole contents and delete contents of array (iteration, output with labelling/messages, array initialisation).	**15**

Question	Answer	Marks

	Example 15 mark answer in pseudocode
	
	`// meaningful identifiers and appropriate data structures for`
	`// all data required`
	`DECLARE Contacts : ARRAY[1:100, 1:2] OF STRING`
	`DECLARE CurrentSize : INTEGER`
	`DECLARE Cont : BOOLEAN`
	`DECLARE Choice : INTEGER`
	`DECLARE NewContacts : INTEGER`
	`DECLARE Count : INTEGER`
	`DECLARE Count2 : INTEGER`
	`DECLARE Flag : BOOLEAN`
	`DECLARE Temp1 : STRING`
	`DECLARE Temp2 : STRING`
	
	`// the number of contacts in the array`
	`CurrentSize ← 0`
	
	`// to allow program to continue indefinitely`
	`Cont ← TRUE`
	`WHILE Cont DO`
	`// display menu`
	`  OUTPUT "Please choose one of the following: "`
	`  OUTPUT "Press 1 to enter new contacts "`
	`  OUTPUT "Press 2 to display your contacts "`
	`  OUTPUT "Press 3 to delete all contacts "`
	`  INPUT Choice`
	`// validate choice as 1, 2 or 3`
	`  WHILE Choice = 1 AND CurrentSize = 100 DO`
	`  OUTPUT "Your contacts are full, please enter 2 or 3"`
	`  INPUT Choice`
	`  ENDWHILE`
	`  WHILE Choice < 1 OR Choice > 3 DO`
	`  OUTPUT "Incorrect entry - please enter 1, 2, or 3"`
	`  INPUT Choice`
	`  ENDWHILE`	

Question	Answer	Marks

	`// enter new contacts`
	`  IF Choice = 1`
	`  THEN`
	`  OUTPUT "How many contacts (1 to 5 only)?"`
	`  INPUT NewContacts`
	`// validates new contacts input`
	`  WHILE NewContacts < 1 OR NewContacts > 5 DO`
	`  OUTPUT "You may only enter between 1 and 5 contacts. Please try again"`
	`  INPUT NewContacts`
	`  ENDWHILE`
	`// checks the maximum size is not exceeded`
	`  WHILE CurrentSize + NewContacts > 100`
	`  OUTPUT "Not enough space in your contacts"`
	`  OUTPUT "The maximum number you may input is ", 100 - CurrentSize`
	`  INPUT NewContacts`
	`  ENDWHILE`
	`  FOR Count ← CurrentSize + 1 TO CurrentSize + NewContacts`
	`  OUTPUT "Enter the contact name as last name, first name"`
	`  INPUT Contacts[Count, 1]`
	`  OUTPUT "Enter the telephone number"`
	`  INPUT Contacts[Count, 2]`
	`  NEXT Count`
	`  CurrentSize ← CurrentSize + NewContacts`
	`// bubble sort to sort array if it contains 2 or more contacts`
	`  IF CurrentSize >= 2`
	`  THEN`
	`  REPEAT`
	`  Flag ← FALSE`
	`  FOR Count ← 1 TO CurrentSize-1`
	`  IF Contacts[Count + 1, 1] <`
	`  Contacts[Count, 1]`
	`  THEN`
	`  Flag ← TRUE`
	`  Temp1 ← Contacts[Count, 1]`
	`  Temp2 ← Contacts[Count, 2]`	

Question	Answer	Marks

	`  Contacts[Count, 1] ← Contacts[Count + 1, 1]`
	`  Contacts[Count, 2] ← Contacts[Count + 1, 2]`
	`  Contacts[Count + 1, 1] ← Temp1`
	`  Contacts[Count + 1, 2] ← Temp2`
	
	`  ENDIF`
	`  NEXT Count`
	`  UNTIL NOT Flag`
	`  ENDIF`
	`  ENDIF`
	`// display all contacts`
	`  IF Choice = 2`
	`  THEN`
	`  IF CurrentSize > 0`
	`  THEN`
	`  OUTPUT "Name and Telephone Number"`
	`  FOR Count ← 1 TO CurrentSize`
	`  OUTPUT Contacts[Count, 1], "  ", Contacts[Count, 2]`
	`  NEXT Count`
	`  ENDIF`
	`  ENDIF`
	`// delete all contacts`
	`  IF Choice = 3`
	`  THEN`
	`  FOR Count ← 1 TO 100`
	`  FOR Count2 ← 1 TO 2`
	`  Contacts[Count, Count2] ← ""`
	`  NEXT Count2`
	`  NEXT Count`
	`  ENDIF`
	`ENDWHILE`	

Q4)

Question	Answer	Marks

	• AO2 (maximum 9 marks)
	• AO3 (maximum 6 marks)
	
	Data Structures required names shown underlined must be used as given in the scenario
	2D Array or list `Evening[1:10, 1:20] / Evening[0:9, 0:19]`
	Variables `Counter, SeatCounter, NumSeats, Row, Column`
	
	Requirements (techniques)
	R1 Find number of seats available for each performance and output (searching, nested iteration, output)
	R2 Inputs and validates number of seats (input, iteration, and selection)
	R3 Checking if seats available (selection, assignment, output with appropriate messages)
	
	Example 15-mark answer in pseudocode
	
	`// meaningful identifier names and appropriate data structures to store the data required`
	`DECLARE Counter, SeatCounter, NumSeats, Row, Column : INTEGER`
	
	`CONSTANT HouseFull = 200`
	`CONSTANT MaxRow = 10`
	`CONSTANT MaxColumn = 20`
	
	`SeatCounter1 ← 0 // initialise seat counter for performance 1`
		15

Question	Answer	Marks

	`FOR Row ← 1 TO 10`
	`  FOR Column ← 1 TO 20`
	`  IF Evening[Row, Column]`
	`  THEN`
	`  SeatCounter ← SeatCounter + 1`
	`  ENDIF`
	`  NEXT Column`
	`NEXT Row`
	
	`// validate input`
	`OUTPUT "How many seats do you want to book? 1, 2, 3 or 4 "`
	`INPUT NumSeats`
	
	`WHILE 1 < NumSeats OR NumSeats > 4 OR NumSeats <> ROUND(NumSeats, 0)`
	`  OUTPUT "Please enter 1, 2, 3 or 4 for the number of seats "`
	`  INPUT NumSeats`
	`ENDWHILE`
	
	`IF SeatCounter + NumSeats > 200)// check for house full`
	`  THEN`
	`  OUTPUT "House full"`
	`  ELSE`
	`  IF SeatCounter + NumSeats > 200 // checks for not enough seats`
	`  THEN`
	`  OUTPUT "Only ", SeatCounter + NumSeats - 200, " seats left"`
	`  ELSE`
	`  FOR Counter ← 1 TO NumSeats // book required number of seats for performance`
	`  Evening[MOD(SeatCounter + Counter, MaxColumn), DIV(SeatCounter +`
	`  Counter), MaxColumn] ← TRUE`
	`  OUTPUT "Row ", MOD(SeatCounter + Counter, MaxColumn), " seat ",`
	`  DIV(SeatCounter + Counter, MaxColumn)," booked"`
	`  NEXT Counter`
	`  ENDIF`	

Q5)

Question	Answer	Marks
	Requirements may be met using a suitable built-in function from the programming language used (Python, VB.NET or Java) Tables for AO2 and AO3 are used to award a mark in a suitable band using a best fit approach. Marks are available for: • AO2 (maximum 9 marks) • AO3 (maximum 6 marks) **Data Structures required** with names as given in the scenario The names underlined must be used as they are provided in the scenario: Arrays or lists `WoodType[]`, `Price[]`, `Customers[]`, `Quotations[]` **Requirements (techniques)** **R1** Input and store customer name, room length and width, with validation of input for room dimensions, including error message and repeated input (Input with prompts, range check and iteration). **R2** Initialise wood arrays. Calculate room area, select and store wood required. Determine cost of wood type and calculate price of wood to purchase. Round and store all data to relevant array (array initialisation, rounding, data retrieval from array, calculation and storage of results). **R3** Output full details: name of customer, choice of wood and quotation price with appropriate messages. Program continues for next customer (Output with messages, iteration of whole program).	**15**

Question	Answer	Marks

	Example 15-mark answer in pseudocode
	
	// declarations not required in the answer
	// initial population of WoodType[] and Price[] arrays
	// input and loops are also acceptable
	WoodType[1] ← "Laminate"
	WoodType[2] ← "Pine"
	WoodType[3] ← "Oak"
	Price[1] ← 29.99
	Price[2] ← 39.99
	Price[3] ← 54.99
	// initialises starting customer in sales arrays
	CurrentCustomer ← 1
	// to allow program to continue to next customer
	Cont ← TRUE
	WHILE Cont DO
	// input customer name
	OUTPUT "Input the customer's name "
	INPUT Customers[CurrentCustomer]
	// input of room dimensions with validation
	OUTPUT "What is the length of your room? "
	INPUT RoomLength
	// validate RoomLength
	WHILE RoomLength < 1.5 OR RoomLength > 10.0
	OUTPUT "The measurement must be in the range 1.5 to 10.0 inclusive, please try again "
	INPUT RoomLength
	ENDWHILE
	OUTPUT "What is the width of your room? "
	INPUT RoomWidth
	// validate RoomWidth
	
	WHILE RoomWidth < 1.5 OR RoomWidth > 10.0
	OUTPUT "The measurement must be in the range 1.5 to 10.0 inclusive, please try again "
	INPUT RoomWidth
	ENDWHILE
	RoomArea ← ROUND(RoomLength, 1) * ROUND(RoomWidth, 1)
	RoomArea ← ROUND (RoomArea + 0.5, 0)
	// show the wood available and prices
	OUTPUT "the wood choices available are:"
	OUTPUT "Number Wood Type Price($)"
	FOR Count ← 1 TO 3
	OUTPUT Count, " ", WoodType[Count], " ",	

Question	Answer	Marks

	Price[Count]
	  Next Count
	// input wood choice
	  OUTPUT "Input a number from 1 to 3 "
	  INPUT WoodChoice
	// validate wood choice
	  WHILE WoodChoice < 1 OR WoodChoice > 3
	  OUTPUT "Your input is out of range, please try again "
	  INPUT WoodChoice
	  ENDWHILE
	// to calculate the total cost of the wood
	  WoodCost ← RoomArea * Price[WoodChoice]
	// to store the relevant data in Quotations[]
	  Quotations[CurrentCustomer, 1] ← RoomLength
	  Quotations[CurrentCustomer, 2] ← RoomWidth
	  Quotations[CurrentCustomer, 3] ← RoomArea
	  Quotations[CurrentCustomer, 4] ← WoodChoice
	  Quotations[CurrentCustomer, 5] ← WoodCost
	
	// final output of quotation
	  OUTPUT "Customer name: ", Customers[CurrentCustomer]
	  OUTPUT "The wood you have chosen is: ", WoodType[WoodChoice]
	  OUTPUT "Your total price is: ", Quotations[CurrentCustomer,5]
	// ready for next customer
	  CurrentCustomer ← CurrentCustomer + 1
	// resets CurrentCustomer to beginning of array when array
	// limit reached
	  IF CurrentCustomer > 100
	  THEN
	  CurrentCustomer ← 1
	  ENDIF
	ENDWHILE	

Q6)

Question	Answer	Marks

	• AO2 (maximum 9 marks)
	• AO3 (maximum 6 marks)
	
	Data Structures required names shown underlined must be used as given in the scenario
	2D Array or list Temperatures
	Variables MaxDay, MinDay, AvDay, MaxWeek, MinWeek, AvWeek
	
	Requirements (techniques)
	
	R1 Find maximum and minimum temperatures for each day and calculates the average daily temperature (searching, totalling)
	R2 Find maximum and minimum temperatures for week and calculates the average weekly temperature (nested searching, totalling)
	R3 outputs for each day name, the rounded values for maximum temperature, minimum temperatures and average temperature. Outputs for the week the rounded values for maximum temperature, minimum temperatures and average temperature (output with appropriate messages and rounded values)
	
	Example 15-mark answer in pseudocode:
	
	`// meaningful identifier names and appropriate data structures to store the data required`
	`DECLARE DayCounter, HourCounter : INTEGER`
	`DECLARE AvDay, AvWeek, MaxDay, MinDay, MaxWeek, MinWeek : REAL`
	`DECLARE DayTotal, WeekTotal : REAL`
	`DECLARE Day : STRING`
	
	`CONSTANT Hours ← 24`
	`CONSTANT Days ← 7`
		15

Question	Answer	Marks

	`MaxWeek ← -1000// initialise max and min temperatures and total for the week`
	`MinWeek ← 1000`
	`WeekTotal ← 0`
	
	`FOR DayCounter ← 0 TO Days - 1`
	`  MaxDay ← -1000// initialise max and min temperatures and total for each day`
	`  MinDay ← 1000`
	`  DayTotal ← 0`
	`  FOR HourCounter ← 0 TO Hours - 1`
	`  DayTotal ← DayTotal + Temperatures(HourCounter, DayCounter)`
	`  // update total maximum and minimum`
	`  IF Temperatures(HourCounter, DayCounter) > MaxDay`
	`  THEN`
	`  MaxDay ← Temperatures(HourCounter, DayCounter)`
	`  ENDIF`
	`  IF Temperatures(HourCounter, DayCounter) < MinDay`
	`  THEN`
	`  MinDay ← Temperatures(HourCounter, DayCounter)`
	`  ENDIF`
	`  NEXT HourCounter`
	
	`  CASE OF DayCounter  // select message for day`
	`  0 : Day ← "Monday"`
	`  1 : Day ← "Tuesday"`
	`  2 : Day ← "Wednesday"`
	`  3 : Day ← "Thursday"`
	`  4 : Day ← "Friday"`
	`  5 : Day ← "Saturday"`
	`  6 : Day ← "Sunday"`
	
	`  ENDCASE`
	
	`  DayAverage ← DayTotal / Hours  // output results for day`
	`  OUTPUT Day  // Results from a day`
	`  OUTPUT "Maximum temperature ", MaxDay`
	`  OUTPUT "Minimum temperature ", MinDay`
	`  OUTPUT "Average temperature ", ROUND(DayAverage,2)`	

Question	Answer	Marks
	IF MaxDay > MaxWeek // update total maximum and minimum THEN MaxWeek ← MaxDay ENDIF IF MinDay > MinWeek THEN MinWeek ← MinDay ENDIF WeekTotal ← WeekTotal + DayTotal // update total for week NEXT DayCounter WeekAverage ← WeekTotal / Days OUTPUT "Maximum temperature for week ", MaxWeek// output results for week OUTPUT "Minimum temperature for week ", MinWeek OUTPUT "Average temperature for Week ", ROUND(WeekAverage,2)	

Q7)

	Data structures required: The names underlined must match those given in the scenario: Arrays or lists Clubs[], Statistics[], Points[], TieIndex[] Variables Matches, Won, Drawn, Lost, Counter, Maximum, TieCheck, OutLoop, Max, Count **Requirements (techniques):** **R1** Input and store number of matches, cricket club names, number of matches won, drawn and lost. Validation of number of matches played and matches won, lost and drawn against number of matches played (with prompts, iteration, storing data in variables, validation, 1D and 2D arrays). **R2** Calculate and store the number of points for each cricket club. Finding the index of the array with the highest score / highest score (calculation, finding the maximum, iteration). **R3** Identifying cricket clubs with highest number of points and whether there is a tie for the top position. Output with appropriate messages (iteration, selection and output).	**15**

Question	Answer	Marks

	Example 15-mark answer in pseudocode
	
	`// meaningful identifiers and appropriate data structures`
	`// similar variables grouped to save space as in HL languages`
	`DECLARE Clubs : ARRAY[1:12] OF STRING`
	`DECLARE Statistics : ARRAY[1:12, 1:3] OF INTEGER`
	`DECLARE Points : ARRAY[1 : 12] OF INTEGER`
	`DECLARE TieIndex : ARRAY[1 : 12] OF INTEGER`
	`DECLARE Matches, Won, Drawn, Lost : INTEGER`
	`DECLARE Counter, Maximum, TieCheck, OutLoop : INTEGER`
	`DECLARE Max, Count : INTEGER`
	`// input number of matches`
	`REPEAT`
	`  OUTPUT "How many matches have been played (Maximum 22)? "`
	`  INPUT Matches`
	`UNTIL Matches <= 22`
	`// input of data as a single loop - multiple loops for data entry`
	`// is also acceptable.`
	`FOR Counter ← 1 TO 12`
	`  OUTPUT "Enter the name of the cricket club"`
	`  INPUT Clubs[Counter]`
	`  // input of match results with validation`
	`  REPEAT`
	`  OUTPUT "Enter the number of matches won, drawn and lost for ", Clubs[Counter]`
	`  INPUT Won, Drawn, Lost`
	`  IF Won + Drawn + Lost <> Matches`
	`  THEN`
	`  OUTPUT "Your inputs must total ", Matches, " please try again"`
	`  ENDIF`
	`  UNTIL Matches = Won + Drawn + Lost`
	`  Statistics[Counter, 1] ← Won`
	`  Statistics[Counter, 2] ← Drawn`
	`  Statistics[Counter, 3] ← Lost`
	`  // calculating and storing points`	

Question	Answer	Marks

	Points[Counter] ← Statistics[Counter, 1] * 12 + Statistics[Counter, 2] * 5
	NEXT Counter
	// finding the highest points
	Max ← -100
	FOR Maximum ← 1 TO 12
	IF Points[Maximum] > Max
	THEN
	Max ← Points[Maximum]
	ENDIF
	NEXT Maximum
	// finding the winner(s)
	Count ← 0
	FOR TieCheck ← 1 TO 12
	IF Points[TieCheck] = Max
	THEN
	Count ← Count + 1
	TieIndex[Count] ← TieCheck
	ENDIF
	Next TieCheck
	// outputting the results
	FOR OutLoop ← 1 TO Count
	OUTPUT "Winning Club(s): ", Clubs[TieIndex[OutLoop]]
	OUTPUT "Number of wins: ", Statistics[TieIndex[OutLoop], 1]
	Next OutLoop
	OUTPUT "Winning Points: ", Max	

Q8)

Question	Answer	Marks

	Data Structures required with names as given in the scenario:
	Arrays or lists `Grid`
	
	Requirements (techniques)
	R1 Set up game – generate random cell, clear all other cells in array, set player start position and start player moves counter (iteration, use of arrays and library routines (round and random))
	R2 Input and check move – is it valid? (input, output, iteration and selection)
	R3 Decide outcome – has move found the X? If so, give appropriate output. If not increment counter and continue. If 10 moves exceeded, give appropriate output (use of arrays, iteration, selection and output).
	
	Example 15-mark answer in pseudocode
	`// Set up game`
	`FOR Row ← 1 TO 5`
	`  FOR Column ← 1 TO 5`
	`  Grid[Row, Column] ← ''// set grid cells to be empty`
	`  NEXT Column`
	`NEXT Row`
		15

Question	Answer	Marks

	`REPEAT  // not in cell 1,1`
	`  XRow ← ROUND ((RANDOM() * 4) + 1, 0) // Random row position between 1 and 5 in GRID`
	`  XColumn ← ROUND ((RANDOM() * 4) + 1, 0) // Random column position between 1 and 5 in GRID`
	`UNTIL XRow <> 1 and XColumn <> 1 // not in cell 1,1`
	
	`Grid [XRow, XColumn] ← 'X'`
	`MaxMove ← 10`
	`NumberMoves ← 0`
	`PlayerRow ← 1`
	`PlayerColumn ← 1`
	`Win ← FALSE`
	
	`// during game`
	`WHILE NumberMoves < MaxMove AND NOT Win`
	`  MoveError ← FALSE`
	`  OUTPUT "Please enter your move, L - Left, R - Right, U - Up or D - Down"`
	`  INPUT UPPER(PlayerMove)`
	`  REPEAT`
	`  CASE OF PlayerMove`
	`  'L' : TempColumn ← PlayerColumn - 1`
	`  'R' : TempColumn ← PlayerColumn + 1`
	`  'U' : TempRow ← PlayerRow - 1`
	`  'D' : TempRow ← PlayerRow + 1`
	`  OTHERWISE MoveError ← TRUE`
	`  ENDCASE`
	
	`// check for out-of-range moves`
	`  IF TempColumn < 1 or TempColumn > 5`
	`  THEN`
	`  MoveError ← TRUE`
	`  ELSE`
	`  PlayerColumn ← TempColumn`
	`  ENDIF`	

Question	Answer	Marks

	IF TempRow < 1 or TempRow > 5
	  THEN
	  MoveError ← TRUE
	  ELSE
	  PlayerRow ← TempRow
	  ENDIF
	
	// check win if X Found
	  IF Grid [PlayerRow, PlayerColumn] = 'X'
	  THEN
	  OUTPUT "You Win"
	  Win ← TRUE
	  ELSE
	  IF NOT MoveError
	  THEN
	  NumberMoves ← NumberMoves + 1
	  ENDIF
	  ENDIF
	  UNTIL NOT MoveError
	ENDWHILE
	
	IF NOT Win
	  THEN
	  OUTPUT "You Lose"
	ENDIF	

Q9)

Question	Answer	Marks
	• AO2 (maximum 9 marks) • AO3 (maximum 6 marks) **Data Structures required** with names as given in the scenario: Arrays or lists `Teams[]`, `Results[]` Variables `Played` **Requirements (techniques):** **R1** Input and store number of games played, teams, number of games won, drawn and lost, with validation for input of numbers (iteration, range check, input, output). **R2** Calculate and store the number of points. Sort the arrays by number of points (calculation, sort, (nested) iteration). **R3** Finding and outputting top team(s) (finding max, counting and output).	**15**

Question	Answer	Marks

	Example 15-mark answer in pseudocode
	
	`// input number of games`
	`REPEAT`
	`  OUTPUT "How many games have been played (Maximum 18)? "`
	`  INPUT Played`
	`UNTIL Played <=18`
	`// input of data as a single loop - a loop for the teams and`
	`// another loop for the data is also acceptable.`
	`FOR InLoop ← 1 TO 10`
	`  OUTPUT "Enter the name of the team"`
	`  INPUT Teams[InLoop]`
	`  // input of games results with validation`
	`  REPEAT`
	`  OUTPUT "Enter the number of games won, drawn and lost for ", Teams[InLoop]`
	`  INPUT Won, Drawn, Lost`
	`  IF Won + Drawn + Lost <> Played`
	`  THEN`
	`  OUTPUT "Your inputs must total ", Played, " please try again"`
	`  ENDIF`
	`  UNTIL Played = Won + Drawn + Lost`
	`  Results[InLoop, 1] ← Won`
	`  Results[InLoop, 2] ← Drawn`
	`  Results[InLoop, 3] ← Lost`
	`  // calculating and storing points`
	`  Results[InLoop, 4] ← Results[InLoop, 1] * 3 + Results[InLoop, 2]`
	`NEXT InLoop`
	`// sorting section`
	`Flag ← TRUE`
	`WHILE Flag DO`
	`  Flag ← FALSE`
	`  FOR Sort ← 1 TO 9`
	`  IF Results[Sort, 4] < Results[Sort + 1, 4]`
	`  THEN`	

Question	Answer	Marks

	// swapping if points not higher then next element
	TempString ← Teams[Sort]
	Temp1 ← Results[Sort, 1]
	Temp2 ← Results[Sort, 2]
	Temp3 ← Results[Sort, 3]
	Temp4 ← Results[Sort, 4]
	Teams[Sort] ← Teams[Sort + 1, 1]
	Results[Sort, 1] ← Results[Sort + 1, 1]
	Results[Sort, 2] ← Results[Sort + 1, 2]
	Results[Sort, 3] ← Results[Sort + 1, 3]
	Results[Sort, 4] ← Results[Sort + 1, 4]
	Teams[Sort + 1] ← TempString
	Results[Sort + 1, 1] ← Temp1
	Results[Sort + 1, 2] ← Temp2
	Results[Sort + 1, 3] ← Temp3
	Results[Sort + 1, 4] ← Temp4
	Flag ← TRUE
	ENDIF
	NEXT Sort
	ENDWHILE
	// checking for tie
	Count ← 1
	Finish ← FALSE
	REPEAT
	IF Results[Count, 4] = Results[Count + 1, 4]
	THEN
	Count ← Count + 1
	ELSE
	Finish ← TRUE
	ENDIF
	UNTIL Finish	

Question	Answer	Marks
	// outputting the results FOR OutLoop ← 1 TO Count OUTPUT "Winning Team(s): ", Teams[OutLoop] NEXT OutLoop OUTPUT "Winning Points: ", Results[1, 4]	

Q10)

Question	Answer	Marks
	• AO2 (maximum 9 marks) • AO3 (maximum 6 marks) **Data Structures required** names shown underlined must be used as given in the scenario. 1D Array or list `MemberName[], MemberTime[], MemberCertificate[], Position[]` Variables `Index, Last, ClubSize, TempTime, TempName, Swap, Count` **Requirements (techniques)** **R1** Input and verify the members times (input and iteration) **R2** sort the MemberName[] and MemberTime[] arrays in ascending order of time and outputs the top three members and their times (nested iteration, sorting, selection and output) **R3** Storing the members names who will receive a certificate and outputting the number of certificates (iteration, selection, counting and output)	**15**

Question	Answer	Marks

	Example 15-mark answer in pseudocode
	
	`CONSTANT ClubSize = 200  // setting the number of members in the club`
	`DECLARE Array Position[1:3} STRING`
	`DECLARE Position : ARRAY[1:3] OF STRING`
	
	`Position[1] ← "First"`
	`Position[2] ← "Second"`
	`Position[3] ← "Third"`
	
	`FOR Index ← 1 TO ClubSize // 200`
	`    REPEAT`
	`        PRINT "Please enter the time for ", MemberName[Index]`
	`        INPUT Time1`
	`        PRINT "Please re-enter the time"`
	`        INPUT Time2`
	`        IF Time1 <> Time2`
	`          THEN`
	`            PRINT "Incorrect input, the times should be the same, please re-enter"`
	`        ENDIF`
	`    UNTIL Time1 = Time2`
	`    MemberTime[Index] ← Time1`
	`NEXT Index`
	
	`Last ←ClubSize`
	`REPEAT`
	`    Swap ← FALSE`
	`    FOR Index ← 1 TO ClubSize - 1`
	`        IF MemberTime[Index] > MemberTime[Index + 1]`
	`          THEN`
	`            TempTime ← MemberTime[Index]`
	`            MemberTime[Index] ← MemberTime[Index + 1]`
	`            MemberTime[Index + 1] ← TempTime`
	`            TempName ← MemberName[Index]`
	`            MemberName[Index] ← MemberName[Index + 1]`
	`            MemberName[Index + 1] ← TempName`	

Question	Answer	Marks
	Swap ← TRUE ENDIF NEXT Index Last ← Last - 1 UNTIL NOT Swap or Last = 1 FOR Index ← 1 TO 3 OUTPUT Position[Index], MemberName[Index], " with a time of ", MemberTime[Index] NEXT Index Count ← 0 FOR Index ← 1 TO ClubSize // 200 IF MemberTime[Index] < 240 THEN Count ← Count + 1 MemberCertificate[Count] ← MemberName[Index] ENDIF NEXT Index OUTPUT "Number of certificates to be printed is ", Count	

Q11)

Question	Answer	Marks

	• AO2 (maximum 9 marks)
	• AO3 (maximum 6 marks)
	
	Data Structures required with names as given in the scenario:
	Arrays or lists `Rooms[]`, `Dimensions[]`
	
	Variables `Number`
	
	Requirements (techniques):
	
	R1 Input and store number of rooms, the names of the rooms and their dimensions, including validation of number of rooms (input with prompts, (nested) iteration, use of variables, 1D and 2D arrays, validation).
	R2 Calculate and store the area of each room, the total area of the house and the average room area rounded to two decimal places. Find the smallest and largest rooms (calculation, totalling, rounding, finding maximum and minimum values, iteration).
	R3 Output the results, including contents of the arrays and the calculated data (iteration, output).
	
	Example 15-mark answer in pseudocode
	
	`// input and validation of number of rooms`
	`REPEAT`
	`  OUTPUT "How many rooms are in your house (enter a number between 3 and 20 inclusive)?"`
	`  INPUT Number`
	`  IF Number < 3 OR Number > 20`
	`  THEN`
	`  OUTPUT "The number of rooms must be between 3 and 20 inclusive, please try again"`
	`  ENDIF`
	`UNTIL Number >= 3 AND Number <= 20`
	
	`// input of room names and dimensions - could be more than one loop`
	`FOR InLoop ← 1 TO Number`
	`  OUTPUT "Enter the name of room ", InLoop`
	`  INPUT Rooms[InLoop]`
	`  OUTPUT "Enter the length of the room in metres"`
	`  INPUT Dimensions[InLoop, 1]`
	`  OUTPUT "Enter the width of the room in metres"`
	`  INPUT Dimensions[InLoop, 2]`
	`  Dimensions[InLoop, 3] ← ROUND(Dimensions[InLoop, 1] * Dimensions[InLoop, 2], 2)`
	`NEXT InLoop`
	
	`// calculates total area and average area - rounded values have been stored`
	`TotArea ← 0`
	`FOR Total ← 1 TO Number`
	`  TotArea ← TotArea + Dimensions[Total, 3]`
	`NEXT Total`
	`AvArea ← ROUND(TotArea / Number, 2)`
		15

Question	Answer	Marks

	`// finding largest and smallest rooms`
	`Larea ← 0`
	`Sarea ← 100000`
	`Lindex ← 1`
	`Sindex ← 1`
	`FOR Count ← 1 TO Number`
	`  IF Dimensions[Count, 3] > Larea`
	`  THEN`
	`  Larea ← Dimensions[Count, 3]`
	`  Lindex ← Count`
	`  ENDIF`
	`  IF Dimensions[Count, 3] < Sarea`
	`  THEN`
	`  Sarea ← Dimensions[Count, 3]`
	`  Sindex ← Count`
	`  ENDIF`
	`NEXT Count`
	
	`// outputting the results`
	`FOR OutLoop ← 1 TO Number`
	`  OUTPUT "Room: ", Rooms[Outloop]`
	`  OUTPUT "Length: ", Dimensions[OutLoop, 1], " metres"`
	`  OUTPUT "Width: ", Dimensions[OutLoop, 2], " metres"`
	`  OUTPUT "Area: ", Dimensions[OutLoop, 3], " square metres"`
	`Next OutLoop`
	`OUTPUT "The largest room is: ", Rooms[Lindex]`
	`OUTPUT "The smallest room is: ", Rooms[Sindex]`
	`OUTPUT "The total area of the house is: ", TotArea, " square metres"`
	`OUTPUT "The average area of the rooms is: ", AvArea, " square metres"`	

Q12)

Question	Answer	Marks

	• AO2 (maximum 9 marks)
	• AO3 (maximum 6 marks)
	
	Data Structures required names shown underlined must match those given in the scenario
	2D Array or list `PickerName[], PickedWeight[], PickerCertificate[]`
	Variables
	
	Requirements (techniques)
	R1 Input and validate the weights (input and iteration)
	R2 sort the `PickerName[]`, and `PickedWeight[]` arrays in descending order of weight (nested iteration and sorting)
	R3 Output top two members and the weights. Storing the members names who will receive a certificate and outputting the number of certificates (iteration, selection, assignment, counting and output with appropriate messages)
	
	Example 15-mark answer in pseudocode
	
	`CONSTANT GroupSize = 200  // setting the number of members in the club`
	`DECLARE Position : Array[1:2} OF STRING`
	
	`Position[1] ← "Best in Group"`
	`Position[2] ← "Second best in Group"`
	
	`FOR Index ← 1 TO GroupSize`
	`  REPEAT`
	`  PRINT "Please enter the weight for ", PickerName[Index],`
	`  INPUT PickedWeight[Index]`
	`  UNTIL PickedWeight[Index] > 0 AND PickedWeight[Index] < 15`
	`NEXT Index`
	`Last ← GroupSize`
	`REPEAT`
	`  Swap ← FALSE`
		15

Question	Answer	Marks

	`  FOR Index ← 1 TO GroupSize -1`
	`  IF PickedWeight[Index] < PickedWeight[Index + 1]`
	`  THEN`
	`  TempWeight ← PickedWeight[Index]`
	`  PickedWeight[Index] ← PickedWeight[Index + 1]`
	`  PickedWeight [Index + 1] ← TempWeight`
	`  TempName ← PickerName[Index]`
	`  PickerName[Index] ← PickerName[Index + 1]`
	`  PickerName[Index + 1] ← TempName`
	`  Swap ← TRUE`
	`  ENDIF`
	`  NEXT Index`
	`  Last ← Last - 1`
	`UNTIL NOT Swap or Last = 1`
	`FOR Index ← 1 TO 2`
	`  OUTPUT Position[Index], PickerName[Index], " with a weight of ", PickedWeight[Index]`
	`NEXT Index`
	`Index ← 1`
	`Count ← 0`
	`WHILE PickedWeight[Index] > 3 DO // count number of certificates required and store names`
	`  Count ← Count + 1`
	`  PickerCertificate[Count] ← PickerName[Index]`
	`  Index ← Index + 1`
	`ENDWHILE`
	`OUTPUT "Number of certificates to be printed is ", Count`	