11

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:
CompetitorName, CompetitorScore

Requirements (techniques)
R1 inputs the score for each round validating the input is in the range 0 to 30 inclusive (input, output, nested iteration, validation)
R2 calculates the score of each competitor discarding the highest and lowest scores of the 10 rounds (nested iteration, output and selection)
R3 outputs the name of the competitor and whether they have qualified, are a reserve or not qualified, output the number of competitors who have qualified, are reserve and not qualified (totalling, output and selection)

11

Example 15-mark answer in pseudocode.

```
// programming techniques of iteration, selection, counting, totalling and output are used
// Candidates did not need to declare any arrays or variables.
DECLARE Score : INTEGER
DECLARE Highest : INTEGER
DECLARE Lowest : INTEGER
DECLARE Qualified : INTEGER
DECLARE Reserve : INTEGER
DECLARE NotQualified : INTEGER

Qualified ← 0
Reserve ← 0
NotQualified ← 0

FOR X ← 1 TO 30  //looping for number of competitors
    FOR Round ← 1 TO 10  //looping for number of rounds
        OUTPUT CompetitorName[X]
        OUTPUT "Enter the score for the round: ", Round
        INPUT Score
        // validation of input for between 0 and 30 inclusive
        WHILE Score < 0 OR Score > 30 DO
            OUTPUT "The score must be between 0 and 30 inclusive. Please try again."
            INPUT Score
        ENDWHILE
        CompetitorScore[X, Round] ← Score  // adding score to 2-D array
    NEXT Round
NEXT X


// or suitable method of determining how many competitors
FOR X ← 1 TO 30
    Count ← 0
    Highest ← 0
    Score ← 0
    Lowest ← 999 // resetting variables for each competitor
```

11

```
    FOR Y ← 1 TO 10  // loop through rounds
        Score ← Score + CompetitorScore[X, Y]
        IF CompetitorScore[X, Y] > Highest  // finds highest score
          THEN
            Highest ← CompetitorScore[X, Y]
        ENDIF
        IF CompetitorScore[X, Y] < Lowest  // finds lowest score
          THEN
            Lowest ← CompetitorScore[X, Y]
        ENDIF
    NEXT Y
    Score ← Score - Highest - Lowest // calculating score to subtract highest and lowest

    IF Score >= 210  // competitor has qualified
      THEN
        OUTPUT "Competitor ", CompetitorName[X], " has qualified"
        Qualified ← Qualified + 1
    ELSE
      IF Score >= 180 AND Score <= 209  // competitor is a reserve competitor
        THEN
          OUTPUT "Competitor ", CompetitorName[X], " are a reserve"
          Reserve ← Reserve + 1
        ELSE
          OUTPUT "Competitor ", CompetitorName[X], " has not qualified"  // competitor not
                                                                            qualified
          NotQualified ← NOTQualified + 1
      ENDIF
NEXT X
OUTPUT "The number of competitors qualified is ", Qualified    // output of results
OUTPUT "The number of competitors who are a reserve is ", Reserve
OUTPUT "The number of competitors not qualified is ", NotQualified
```

11

Marking Instructions in italics			
AO2: Apply knowledge and understanding of the principles and concepts of computer science to a given context, including the analysis and design of computational or programming problems			
0	**1–3**	**4–6**	**7–9**
No creditable response.	At least one programming technique has been used. *Any use of selection, iteration, counting, totalling, input and output.*	Some programming techniques used are appropriate to the problem. *More than one technique seen applied to the scenario, check the list of techniques needed.*	The range of programming techniques used is appropriate to the problem. *All criteria stated for the scenario have been covered by the use of appropriate programming techniques, check the list of techniques needed.*
	Some data has been stored but not appropriately. *Any **use** of variables or arrays or other language dependent data structures e.g. Python lists.*	Some of the data structures chosen are appropriate and store some of the data required. *More than one data structure **used** to store data required by the scenario.*	The data structures chosen are appropriate and store all the data required. *The data structures **used** store all the data required by the scenario.*

11

Marking Instructions in italics			
AO3: Provide solutions to problems by: • **evaluating computer systems** • **making reasoned judgements** • **presenting conclusions**			
0	**1–2**	**3–4**	**5–6**
No creditable response.	Program seen without relevant comments.	Program seen with some relevant comment(s).	The program has been fully commented.
	Some identifier names used are appropriate. *Some of the data structures used have meaningful names.*	The majority of identifiers used are appropriately named. *Most of the data structures used have meaningful names.*	Suitable identifiers with names meaningful to their purpose have been used throughout. *All of the data structures used have meaningful names.*
	The solution is illogical.	The solution contains parts that may be illogical.	The program is in a logical order.
	The solution is inaccurate in many places. *Solution contains few lines of code with errors that attempt to perform a task given in the scenario.*	The solution contains parts that are inaccurate. *Solution contains lines of code with some errors that logically perform tasks given in the scenario. Ignore minor syntax errors.*	The solution is accurate. *Solution logically performs all the tasks given in the scenario. Ignore minor syntax errors.*
	The solution attempts at least one of the requirements. *Solution contains lines of code that attempt at least one task given in the scenario.*	The solution meets most of the requirements. *Solution contains lines of code that attempts most tasks given in the scenario.*	The solution meets all the requirements given in the question. *Solution performs all the tasks given in the scenario.*

2(a)

One mark per mark point, **max five**

- Line 01 / `DECLARE Contacts : ARRAY[1:500, 1:4] OF REAL`
should be `DECLARE Contacts : ARRAY[1:500, 1:4] OF STRING`

- Line 12 / `OUTPUT FirstName`
should be `INPUT FirstName`

- Line 17 / `OUTPUT Contacts[Row, 1]`
should be `OUTPUT Contacts[Row, Column]`

- Line 23 / `NEXT Stop OR Row > 500`
should be `UNTIL Stop OR Row > 500`

- Line 25 / 26 / Answer variable not declared
should be `DECLARE Answer : CHAR` before line 25

- Line 28 / `Continue ← TRUE`
should be `Continue ← FALSE`

2(b)(i)	**One** mark per mark point, **max four**
	
	MP1 Appropriate parameter in `PROCEDURE` line
	MP2 Correct index outer loop termination value; must use `Number` as stated in question
	MP3 Correct array reference in `INPUT`
	MP4 Correct termination of both loops.
	
	Example:
	
	`PROCEDURE NewData(Number : INTEGER)`
	`  FOR Row ← 1 TO Number`
	`  FOR Column ← 1 TO 4`
	`  INPUT Contacts[Row, Column]`
	`  NEXT Column`
	`  NEXT Row`
	`ENDPROCEDURE`

2(b)(ii)	**One** mark per mark point, **max two**
	
	MP1 Correct input statement
	MP2 Correct call statement for `NewData`
	
	Example:
	
	`INPUT MyNumber`
	`CALL NewData(MyNumber)`

2(c)(i)	**One** mark per mark point, **max two** MP1 There may be more than one person with the given first name // The name may not exist/found in the array MP2 The search will stop at the first occurrence and **output the data for the wrong person** MP3 If the name is not present there could be no output / the loop will continue.

2(c)(ii)	**One** mark per mark point, **max two** MP1 Introduce additional search criteria into the algorithm **to make the search more specific to each person in the array** MP2 … such as the last name to go with the first name // … such as some form of ID number / date of birth / contact number / post code MP3 The algorithm should be able to search the whole array to find all available matches MP4 … by removing the stop variable MP5 Use selection to confirm the name has not been found in the whole array MP6 If the name is not present, output an appropriate message.

5(a)

One mark per mark point, **max six**

MP1 Correct `Code` column
MP2 Correct `Store` column
MP3 Correct `Count` column
MP4 First four `CodeStore` columns left unchanged
MP5 Correct `CodeStore[5]` and `CodeStore[6]` columns
MP6 Correct OUTPUT column

			CodeStore[]						
Code	**Store**	**Count**	**[1]**	**[2]**	**[3]**	**[4]**	**[5]**	**[6]**	**OUTPUT**
			aBcd	**1532**	**Face**	**63q8**			
7686	FALSE	1							
		2							
		3							
		4							
	TRUE	5					7686		
		6							
Face	FALSE	1							
		2							

5(a)

			CodeStore[]						
Code	**Store**	**Count**	**[1]**	**[2]**	**[3]**	**[4]**	**[5]**	**[6]**	**OUTPUT**
		3							Code used before
Speed									Length must be 4
432U	FALSE	1							
		2							
		3							
		4							
		5							
	TRUE	6						432U	
		7							CodeStore full

5(b)

One mark per mark point, **max three**

MP1 Checks that the length of the code input is 4 characters.
MP2 Checks the code against the array contents **to see if it has been used before** (in the last 6 codes)
MP3 If the code passes the tests, it is stored in the array
MP4 Stores the code in the lowest available index
MP5 Checks that the code store isn't full before storing a new code // Outputs a code store is full message if appropriate // Algorithm continues until the code store is full.

5(c)

One mark per mark point, **max four**

MP1 Declaration of array using correct name and data type
MP2 Declaration of loop counter
MP3 Use of appropriate loop for six iterations (allow any type of loop)
MP4 Assignment of all array elements to null string (using loop counter as array index)

Example:

```
DECLARE CodeStore : ARRAY[1:6] OF STRING
DECLARE Index : INTEGER
FOR Index ← 1 TO 6
    CodeStore[Index] ← ""
NEXT Index
```


3

One mark per bullet point

- Initialising a count variable outside of loop
- Correct use of loop with condition for 10 times (`WHILE (DO)...ENDWHILE, REPEAT...UNTIL`) with close
- Incrementing the count variable inside the loop
- Rest of algorithm correct

Examples

```
Count ← 1
REPEAT
    OUTPUT "Please enter a name"
    INPUT Name
    Names[Count] ←Name
    Count ← Count + 1
UNTIL Count > 10
```

Or

```
Count ← 1
WHILE Count <= 10 DO
    OUTPUT "Please enter a name"
    INPUT Name
    Names[Count] ←Name
    Count ← Count + 1
ENDWHILE
```

3(a)	**One** mark per mark point • Line 02 / `DECLARE Index : CHAR` should be `DECLARE Index : INTEGER` • Line 07 / `Stop ← FALSE` should be `Stop ← TRUE` • Line 08 / `FOR Index ← 1 TO 50` should be `FOR Index ← 1 TO 999` // `FOR Index ← 1 TO 1000 - 1` • Line 17 / `NEXT Stop` should be `ENDWHILE`

3(b)(i)	**One** mark per mark point
	
	MP1 Two integer parameters in `PROCEDURE` line
	MP2 Correct index assigned to `Hold ← Values[Index1]`
	MP3 Array element correctly assigned to new array element (`Index2` to `Index1`)
	MP4 Value in `Hold` assigned to correct array element (`Index2`).
	
	For example,
	
	`PROCEDURE Swap(`**`Index1 : INTEGER, Index2 : INTEGER`**`)`
	`  DECLARE Hold : REAL`
	`  Hold ← Values[`**`Index1`**`]`
	`  Values[Index1] ← Values[Index2]`
	`  Values[Index2] ← Hold`
	`ENDPROCEDURE`

3(b)(ii)	**One** mark per mark point MP1 Correct call command and use of `Swap` for procedure name MP2 Two parameters referring to the correct indices of array elements to be swapped, as shown in the original algorithm. `Index + 1` and `Index`. **For example,** `CALL Swap(Index + 1, Index)`
3(c)	**One** mark per mark point MP1 Global variables can be used throughout the program and its procedures // The memory used by the variables is not recovered until the program terminates. MP2 The variable declared in part 3(b)(i) is a local variable and **can only be used in that procedure** // The memory used by a local variable is **recovered at the end of the procedure**.

12

- 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)
```

12

```
// 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"
```


10

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).

10 **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
```

10

```
    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
```

11

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
```

11

```
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
```

11

```
        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
```