11 The scores for each competitor in an archery competition are recorded.

The one-dimensional (1D) array `CompetitorName[]` is used to store the names of the competitors.

The maximum number of competitors is 30.

There are 10 rounds in the competition with a maximum score of 30 per round.

The two-dimensional (2D) array `CompetitorScore[]` stores the score in each of the 10 rounds for each competitor.

The first index of the array `CompetitorScore[]` is the same as the index of the competitor in `CompetitorName[]`

After the 10 rounds have been completed, the highest score and the lowest score for each competitor are discarded.

The overall score for each competitor is the total of the remaining 8 scores (after discarding the highest and lowest scores) .

A competitor with an overall score of 210 or more moves onto the next competition. A competitor with an overall score between 180 and 209 inclusive becomes a reserve competitor. A competitor with an overall score below 180 does not qualify.

Write a program that meets the following requirements:

- inputs the total score (out of 30) for each competitor in each round
- validates each input is between 0 and 30 inclusive
- calculates the overall score for each competitor discarding the highest and lowest scores
- outputs the name of each competitor and if they move onto the next competition, are a reserve or do not qualify
- calculates and outputs the number of competitors who:
  - have qualified for the next competition
  - are reserve competitors
  - do **not** qualify.

You must use pseudocode or program code **and** add comments to explain how your code works.

You do **not** need to declare any arrays or variables; assume that this has already been done.

All inputs and outputs must contain suitable messages.

.....

.....

.....

.....

.....

.....

.....

[1]

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

[2]

[15]

2 This pseudocode algorithm is intended to input a first name from a user and search for it in a two-dimensional (2D) array. If the name is found, the algorithm outputs all the remaining details for that name. The algorithm will continue to allow other names to be entered for searching, until the user input stops it.

```
01 DECLARE Contacts : ARRAY[1:500, 1:4] OF REAL
02 DECLARE Row : INTEGER
03 DECLARE Column : INTEGER
04 DECLARE Continue : BOOLEAN
05 DECLARE Stop : BOOLEAN
06 DECLARE FirstName : STRING
07 Continue ← TRUE
08 Stop ← FALSE
09 WHILE Continue
10     Row ← 1
11     OUTPUT "Enter a first name "
12     OUTPUT FirstName
13     REPEAT
14         IF Contacts[Row, 1] = FirstName
15           THEN
16             FOR Column ← 1 TO 4
17                 OUTPUT Contacts[Row, 1]
18             NEXT Column
19             Stop ← TRUE
20           ELSE
21             Row ← Row + 1
22         ENDIF
23     NEXT Stop OR Row > 500
24     OUTPUT "Search for another name? (Y or N)"
25     INPUT Answer
26     IF Answer = 'N' OR Answer = 'n'
27       THEN
28         Continue ← TRUE
29     ENDIF
30 ENDWHILE
```

(a) Identify the line numbers of **five** errors in the pseudocode and suggest a correction for each error.

Error 1 line number .....

Correction .....

.....

Error 2 line number .....

Correction .....

.....

Error 3 line number .....

Correction .....

.....

Error 4 line number .....

Correction .....

.....

Error 5 line number .....

Correction .....

.....

[5]

(b) A procedure `NewData` allows data for new contacts to be entered and stored in the 2D array `Contacts[]`. A parameter, `Number`, is used to pass values from the main algorithm to the procedure for the number of new contacts. Next, four pieces of contact data for each new contact are entered and stored at the start of the array.

(i) Complete the pseudocode for `PROCEDURE NewData`

```
PROCEDURE NewData(.....)
    FOR Row ← 1 TO .....
        FOR Column ← 1 TO 4
            INPUT .....
            .....
    .....
ENDPROCEDURE
```

[4]

(ii) Write the pseudocode to input the number of new contacts and to use procedure `NewData`. A variable declaration is **not** required.

.....

.....

.....

..... [2]

(c) A first name is used as the search key in the algorithm on page 2. If the name is found, all the data for that name is output and the algorithm stops searching.

(i) Explain why the given algorithm may **not** always give the expected results.

.....

.....

.....

..... [2]

(ii) Explain how you could improve the algorithm so that the search is more likely to find the required results.

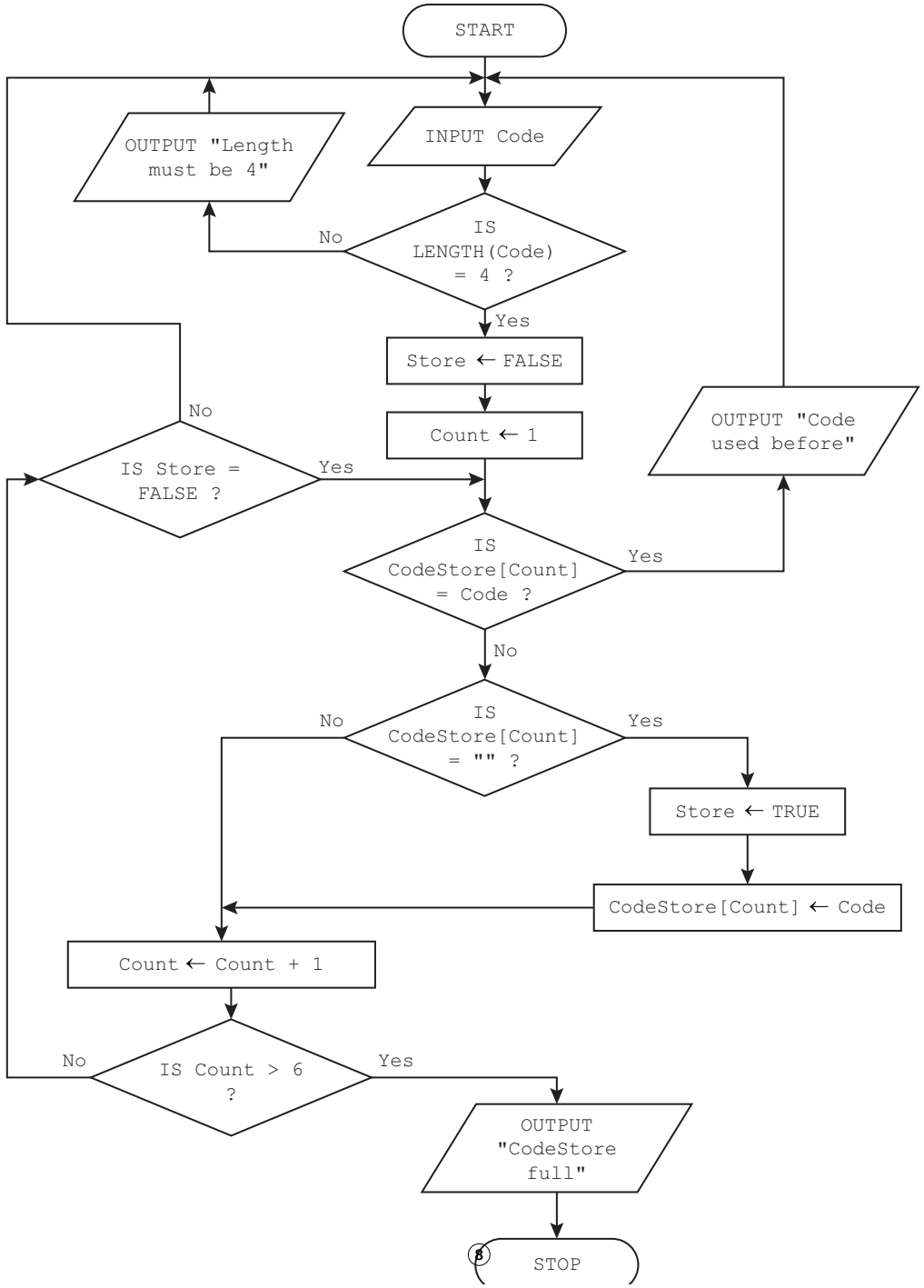
.....

.....

.....

..... [2]

5 This flowchart represents an algorithm.



(a) The first data row of the trace table shows the values currently stored in `CodeStore[]`

Complete the trace table for the algorithm, using the input data:

7686, Face, Speed, 432U, Yp79

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

[6]

(b) Describe the processes in the algorithm on page 6.

.....

.....

.....

.....

.....

..... [3]

(c) The algorithm on page 6 requires the array `CodeStore[]` to be declared as a one-dimensional (1D) string array with six elements that are initially set to the null string `""`.

Write the pseudocode required to declare and initialise the array `CodeStore[]`
You do need to declare any variables used.

.....

.....

.....

.....

.....

.....

.....

..... [4]

3 The following pseudocode algorithm uses a count-controlled loop to read in 10 names and store them in the array `Names[]`

```
FOR Count ← 1 TO 10
    OUTPUT "Please enter a name"
    INPUT Name
    Names[Count] ← Name
NEXT Count
```

Rewrite the algorithm in pseudocode, using a condition-controlled loop instead of a count-controlled loop.

.....

.....

.....

.....

.....

.....

.....

.....

..... [4]

3 The purpose of this pseudocode algorithm is to carry out a bubble sort to sort, in descending order, 1000 numbers stored in a one-dimensional (1D) array.

```
01 DECLARE Values : ARRAY[1:1000] OF REAL
02 DECLARE Index : CHAR
03 DECLARE Stop : BOOLEAN
04 DECLARE Hold : REAL
05 Stop ← FALSE
06 WHILE NOT Stop DO
07     Stop ← FALSE
08     FOR Index ← 1 TO 50
09         IF Values[Index + 1] > Values[Index]
10           THEN
11             Hold ← Values[Index]
12             Values[Index] ← Values[Index + 1]
13             Values[Index + 1] ← Hold
14             Stop ← FALSE
15         ENDIF
16     NEXT Index
17 NEXT Stop
```

(a) Identify the line numbers of **four** errors in the pseudocode and suggest a correction for each error.

Error 1 line number .....

Correction .....

.....

Error 2 line number .....

Correction .....

.....

Error 3 line number .....

Correction .....

.....

Error 4 line number .....

Correction .....

.....

[4]

(b) The swap section in lines 11 to 13 of the existing code are to be changed to a call statement for a procedure. This procedure will include two parameters representing the indexes of the array elements to be swapped.

(i) Complete the pseudocode for `PROCEDURE Swap`

```
PROCEDURE Swap(.....)
    DECLARE Hold : REAL
    Hold ← Values[.....]
    .....
    .....
ENDPROCEDURE
```

[4]

(ii) Write the pseudocode to transfer control to `PROCEDURE Swap`

.....

..... [2]

(c) Explain the difference between global variables and the variable declared in **3(b)(i)**.

.....

.....

.....

..... [2]

12 A one-dimensional (1D) array `Rooms[]` contains the names of up to 20 rooms in a house.
A two-dimensional (2D) array `Dimensions[]` is used to store the length, width and area of each room.

The position of any room's data is the same in both arrays. For example, the data in index 5 of `Dimensions[]` belongs to the room in index 5 of `Rooms[]`

The variable `Number` stores the number of rooms for which data is to be input. There must be at least 3 rooms but no more than 20.

Write a program that meets the following requirements:

- allows the number of rooms for which data is required to be input, stored and validated
- allows the name of the room and the length and width of the room, in metres, to be entered and stored
- allows the area of each room to be calculated as length multiplied by width and stored as square metres rounded to two decimal places
- calculates the average size of all the rooms by area, in square metres, rounded to two decimal places
- finds the largest room and smallest room by area
- outputs the names of all rooms with their dimensions and area
- outputs the names of the largest room and smallest room by area
- outputs the total area of the house and the average size of all the rooms by area.

You must use pseudocode or program code **and** add comments to explain how your code works.

You do **not** need to declare any arrays or variables; you may assume that this has already been done.

All inputs and outputs must contain suitable messages.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

[15]

10 The one-dimensional (1D) array `Clubs[]` is used to store the names of 12 cricket clubs in a local sports league.

The two-dimensional (2D) array `Statistics[]` is used to store, for each cricket club, the number of:
- matches won
- matches drawn
- matches lost.

The 1D array `Points[]` is used to store the total number of points each cricket club has been awarded.

The position of any cricket club's data is the same in all three arrays. For example, the data in index 2 of `Statistics[]` and index 2 of `Points[]` belongs to the cricket club in index 2 of `Clubs[]`

The variable `Matches` stores the number of matches played by each team. Each team plays the same number of matches.

Points are awarded for:
- a win – 12 points
- a draw – 5 points
- a loss – 0 points.

Write a program that meets the following requirements:
- allows the number of matches played to be input and stored, with a maximum of 22 matches
- validates the number of matches played
- allows the names of the cricket clubs to be input and stored
- allows the number of matches won, drawn and lost to be input and stored for each team
- validates the number of matches won, drawn or lost against the number of matches played
- asks the user to re-enter the number of matches won, drawn or lost if the total does not match the number of matches played
- calculates and stores the total number of points for each club
- finds the cricket club or clubs with the highest number of points
- outputs the name or names of the winning club or clubs, the number of wins and the total number of points awarded.

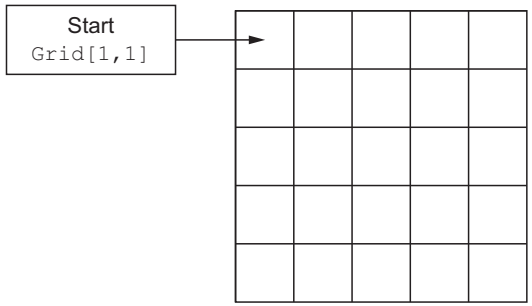
You must use pseudocode or program code **and** add comments to explain how your code works.

You do **not** need to declare any arrays or variables; you may assume that this has already been done.

All inputs and outputs must contain suitable messages.

[15]

11 A one-player game uses the two-dimensional (2D) array `Grid[]` to store the location of a secret cell to be found by the player in 10 moves. Each row and column has 5 cells.



At the start of the game:

- The program places an 'X' in a random cell (**not** in `Grid[1,1]`) and empties all the other cells in the grid.
- The player starts at the top left of the grid.
- The player has 10 moves.

During the game:

- The player can move left, right, up or down by one cell and the move must be within the grid.
- "You Win" is displayed if the player moves to the cell with 'X' and has played 10 moves or less.
- "You Lose" is displayed if the player has made 10 moves without finding the 'X'.

Write a program that meets these requirements.

You must use pseudocode or program code **and** add comments to explain how your code works.

You do **not** need to declare any arrays or variables; you may assume that this has already been done.

All inputs and outputs must contain suitable messages.