11 The one-dimensional (1D) array `TeamName[]` contains the names of teams in a sports league. The two-dimensional (2D) array `TeamPoints[]` contains the points awarded for each match. The position of each team's data in the two arrays is the same. For example, the team stored at index 10 in `TeamName[]` and `TeamPoints[]` is the same.

The variable `LeagueSize` contains the number of teams in the league. The variable `MatchNo` contains the number of matches played. All teams have played the same number of matches.

The arrays and variables have already been set up and the data stored.

Each match can be played at home or away. Points are recorded for the match results of each team with the following values:

- 3 – away win
- 2 – home win
- 1 – drawn match
- 0 – lost match.

Write a program that meets the following requirements:

- calculates the total points for all matches played for each team
- counts the total number of away wins, home wins, drawn matches and lost matches for each team
- outputs for each team:
  - name
  - total points
  - total number of away wins, home wins, drawn matches and lost matches
- finds and outputs the name of the team with the highest total points
- finds and outputs the name of the team with the lowest total points.

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

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

All inputs and outputs must contain suitable messages.

You do **not** need to initialise the data in the arrays `TeamName[]` and `TeamPoints[]` or the variables `LeagueSize` and `MatchNo`

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

11 A one-dimensional (1D) array `Days[]` contains the names of the days of the week. A two-dimensional (2D) array `Readings[]` is used to store 24 temperature readings, taken once an hour, for each of the seven days of the week. A 1D array `AverageTemp[]` is used to store the average temperature for each day of the week.

The position of any day's data is the same in all three arrays. For example, if Wednesday is in index 4 of `Days[]`, Wednesday's temperature readings are in index 4 of `Readings[]` and Wednesday's average temperature is in index 4 of `AverageTemp[]`

The temperature readings are in Celsius to one decimal place. Temperatures can only be from –20.0 °C to +50.0 °C inclusive.

Write a program that meets the following requirements:

- input and validate the hourly temperatures for one week
- calculate and store the average temperature for each day of the week
- calculate the average temperature for the whole week
- convert all the average temperatures from Celsius to Fahrenheit by using the formula Fahrenheit = Celsius * 9/5 + 32
- output the average temperature in Celsius and in Fahrenheit for each day
- output the overall average temperature in Celsius and in Fahrenheit for the whole week.

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

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

All inputs and outputs must contain suitable messages.

All data output must be rounded to one decimal place.

You will need to initialise and populate the array `Days[]` at the start of the program.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

12 A two-dimensional (2D) array `Account[]` contains account holders' names and passwords for a banking program.

A 2D array `AccDetails[]` has three columns containing the following details:

- column one stores the balance – the amount of money in the account, for example 250.00
- column two stores the overdraft limit – the maximum total amount an account holder can borrow from the bank after the account balance reaches 0.00, for example 100.00
- column three stores the withdrawal limit – the amount of money that can be withdrawn at one time, for example 200.00

The amount of money in a bank account can be negative (overdrawn) but **not** by more than the overdraft limit.
For example, an account with an overdraft limit of 100.00 must have a balance that is greater than or equal to –100.00

Suitable error messages must be displayed if a withdrawal cannot take place, for example if the overdraft limit or the size of withdrawal is exceeded.

The bank account ID gives the index of each account holder's data held in the two arrays.
For example, account ID 20's details would be held in:
`Account[20,1]` and `Account[20,2]`
`AccDetails[20,1]` `AccDetails[20,2]` and `AccDetails[20,3]`

The variable `Size` contains the number of accounts.

The arrays and variable `Size` have already been set up and the data stored.

Write a program that meets the following requirements:

- checks the account ID exists and the name and password entered by the account holder match the name and password stored in `Account[]` before any action can take place
- displays a menu showing the four actions available for the account holder to choose from:
    1. display balance
    2. withdraw money
    3. deposit money
    4. exit
- allows an action to be chosen and completed. Each action is completed by a procedure with a parameter of the account ID.

You must use pseudocode or program code **and** add comments to explain how your code works.
All inputs and outputs must contain suitable messages.

You only need to declare any local arrays and local variables that you use.

You do **not** need to declare and initialise the data in the global arrays `Account[]` and `AccDetails[]` and the variable `Size`

.....

.....

.....

.....

.....

.....

11 A two-dimensional (2D) array `Contacts[]` is used to store names and telephone numbers. All the data is stored as strings. The array must have the capacity to store 100 contacts in the form of:

- column 1 – contact names as: last name, first name
  for example: Smith, John
- column 2 – telephone numbers.

The variable `CurrentSize` shows how many contacts are in the array.

Write a program that meets the following requirements:

- display a menu of choices:
  - enter new contact details
  - display all the contact details
  - delete all the contact details
- validate the menu input
- allow up to a maximum of five new contacts to be added to the array at any one time
- do **not** allow more than 100 contacts in total
- after new contacts have been added, sort the array by contact name, as long as there are at least two contacts in the array
- output the whole of the array
- delete the contents of the array.

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

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

All inputs and outputs must contain suitable messages.

You do **not** need to initialise the data in the array `Contacts[]` and the variable `CurrentSize`

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

10 Drama students put on a performance of a play for one evening. Seats in a small theatre can be booked for this performance.

The theatre has 10 rows of 20 seats. The status of the seat bookings for the evening is held in the two-dimensional (2D) Boolean array `Evening[]`

Each element contains `FALSE` if the seat is available and `TRUE` if the seat is booked.

Up to and including four seats can be booked at one time. Seats are allocated in order from those available. A row or seat number cannot be requested.

The array `Evening[]` has already been set up and some data stored.

Write a program that meets the following requirements:

- counts and outputs the number of seats already booked for the evening
- allows the user to input the number of seats required
- validates the input
- checks if enough seats are available:
  - if they are available
    - changes the status of the seats
    - outputs the row number and seat number for each seat booked
  - if they are **not** available:
    - outputs a message giving the number of seats left or 'House full' if the theatre is fully booked.

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.
You do **not** need to initialise the data in the array `Evening[]`
All inputs and outputs must contain suitable messages.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

11 A wood flooring company stores the names of up to 100 customers in a one-dimensional (1D) array `Customers[]`. A two-dimensional (2D) array `Quotations[]` stores details of each customer's quotation:

- length of room (one decimal place)
- width of room (one decimal place)
- area of wood required (rounded up to next whole number)
- choice of wood index (whole number)
- price of wood required in dollars (two decimal places).

The floor measurements (room length and room width) are taken in metres. All floors are rectangles and room measurements must be between 1.5 and 10.0 inclusive.

The index of any customer's data is the same in both arrays. For example, a customer named in index 4 of `Customers[]` corresponds to the data in index 4 of `Quotations[]`

The wood choices available are:

Index	Wood type	Price per square metre ($)
1	Laminate	29.99
2	Pine	39.99
3	Oak	54.99

The data are stored in two 1D arrays named `WoodType[]` and `Price[]`. The index of the wood type and price in their arrays share the same index number.

Write a program that meets the following requirements:

- input a new customer's name, room length and room width
- check that each measurement is valid
- output an error message and require the measurement to be re-entered until it is valid
- calculate the area of the room by multiplying together the length of the room and the width of the room
- input the choice of wood and find its price per square metre
- calculate the price of the wood needed
- store all data in the relevant array
- output the customer's quotation to include: the name of the customer, the choice of wood and the calculated price of the wood required
- continue to accept the next customer.

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.

You will need to initialise `WoodType[]` and `Price[]`

All inputs and outputs must contain suitable messages.

.....

.....

.....

.....

.....

10 A weather station takes temperature readings once an hour for a week. These temperatures are stored in a two-dimensional (2D) array `Temperatures[]`
Each column contains 24 readings for a single day. The first temperature is recorded at 00:00 and the final temperature at 23:00. There are seven columns, one for each day of the week, starting with Monday and ending with Sunday.

The variables `MaxDay`, `MinDay` and `AvDay` are used to store the maximum, minimum, and average temperatures for a day. The variables `MaxWeek`, `MinWeek` and `AvWeek` are used to store the maximum, minimum, and average temperatures for the week.

The array has already been set up and the data stored.

Write a program that meets the following requirements:

- finds the maximum and minimum temperatures for each day
- calculates the average temperature for each day
- outputs for each day:
  - name of the day, for example Monday
  - maximum temperature
  - minimum temperature
  - average temperature
- finds the maximum and minimum temperatures for the week
- calculates the average temperature for the week
- outputs:
  - maximum temperature for the week
  - minimum temperature for the week
  - average temperature for the week.

All temperatures output must be rounded to two decimal places.

You must use pseudocode or program code **and** add comments to explain how your code works.
All inputs and outputs must contain suitable messages.

You do **not** need to declare any arrays or variables; you may assume that this has already been done.
You do **not** need to initialise the data in the array `Temperatures[]`

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....