2 A program reads data from a text file, splits the data depending on its content and stores the separated data into different files.

The text file `TheData.txt` contains 72 lines of data. Each line of data has an integer number and a string colour that are separated by a comma. For example, the first line in the file is:

```
10,red
```

The integer is `10` and the string is `"red"`

The file contains six different colours: red, green, blue, orange, yellow, pink.

(a) The function `ReadData()`:

- prompts the user to enter a filename and reads this filename from the user
- opens the file and reads each line of data into a 1D array
- returns the populated 1D array.

The function needs to work for a file that contains an unknown number of lines.

Write program code for `ReadData()`

Save your program as **Question2_J25**.

Copy and paste the program code into part **2(a)** in the evidence document.

[7]

(b) The procedure `SplitData()` takes a 1D string array as a parameter with the identifier `DataArray`

The procedure declares six 1D arrays: one array for each colour that appears in the file (red, green, blue, orange, yellow, pink).

The procedure accesses each string in `DataArray`. The data in each string is split into the integer and the colour. The integer is stored in the array that matches the colour.

For example, the first string in `DataArray` has the integer `10` and the colour `red`, so the integer `10` is stored in the array for the colour red.

Write program code for `SplitData()`

Save your program.

Copy and paste the program code into part **2(b)** in the evidence document.

[6]

(c) The procedure `StoreData()`:

- takes **two** parameters: a 1D array `DataToStore` and a filename
- opens the text file with the filename that is passed as a parameter
- appends each item of data from `DataToStore` to a new line in the text file
- uses exception handling when opening and writing data to the text file.

Write program code for `StoreData()`

Save your program.

Copy and paste the program code into part **2(c)** in the evidence document.

[5]

(d) Each of the six colours has a blank text file where the numbers will be stored. The names of these six text files are:

- `Blue.txt`
- `Green.txt`
- `Orange.txt`
- `Pink.txt`
- `Red.txt`
- `Yellow.txt`

The procedure `SplitData()` needs amending to call `StoreData()` **six** times, with each of the **six** colour arrays and the name of the text file that corresponds to that colour.

For example, `StoreData()` will be called with the red array and the file name `"Red.txt"`

Write program code to amend `SplitData()`

Save your program.

Copy and paste the program code into part **2(d)** in the evidence document.

[2]

(e) The main program calls `ReadData()` and then `SplitData()`

(i) Write program code for the main program.

Save your program.

Copy and paste the program code into part **2(e)(i)** in the evidence document.

[3]

(ii) Test your program. Input the text `"TheData.txt"` when prompted.

Take a screenshot of the output(s) and a screenshot showing the content of the file that stores the red numbers.

Save your program.

Copy and paste the screenshot(s) into part **2(e)(ii)** in the evidence document.

[2]

3 The text file `HighScoreTable.txt` stores the top seven player scores for a game.

The data in the file is stored in the order:

Player ID

Game level

Score

Each item of data is stored on a new line. For example, the first set of data in the file is:

Player ID: GHEH

Game level: 3

Score: 10

(a) The data about the players and their scores is stored in a 2D array of strings with the identifier `HighScores`.

The first dimension of the array has seven elements: one for each player. The second dimension of the array has three elements: one for the player ID, one for the game level and one for the score. All data is stored in the array as strings.

`HighScores` is declared local to the main program, and all elements are initialised to an empty string, for example `""`.

Write program code to declare and initialise `HighScores`.

Save your program as **Question3_N24**.

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

[2]

(b) The function `ReadData()` reads the data from `HighScoreTable.txt` and stores the data in a 2D array. The function returns the 2D array.

The function uses exception handling when opening and reading data from the file.

Write program code for `ReadData()`.

Save your program.

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

[5]

(c) The procedure `OutputHighScores()` takes a 2D array as a parameter. It outputs each player's ID, level and score in the order they are in the array. The outputs are in the format:

GHEH reached level 3 with a score of 10

Write program code for `OutputHighScores()`.

Save your program.

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

[2]

(d) The scores in `HighScoreTable.txt` are **not** in order.

The player scores are first sorted by the game level they reached. For example, all players that reached level 5 are higher in the high score table than players that reached level 4.

The players that reached the same level are then sorted in descending order of score.

An example sorted high score table is:

Position	Player ID	Game level	Score
1	GFED	5	25
2	HJKM	4	21
3	RERR	4	19
4	TTYU	4	15
5	WSVG	3	20
6	PPTR	3	15
7	SNQT	2	10

The function `SortScores()` uses a bubble sort to sort the data as described and returns the sorted array.

Write program code for `SortScores()`.

Save your program.

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

[4]

(e) (i) Amend the main program to implement these steps in the order given:

- call `ReadData()` and store the returned array in `HighScores`
- output `"Before"`
- call `OutputHighScores()` with `HighScores` as a parameter
- call `SortScores()` and store the returned array in `HighScores`
- output `"After"`
- call `OutputHighScores()` with `HighScores` as a parameter.

Save your program.

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

[2]

(ii) Test your program

Take a screenshot of the output.

Save your program.

Copy and paste the screenshot into part **3(e)(ii)** in the evidence document.

[2]