3 A text file `OldFile.txt` contains IDs, names and email addresses for members of a club. Three information items are stored for each member and each item is stored on a separate line.

The example shows the information for the first two members in the first six lines of the file:

Line in file	Information item	Example data
1	member 1 ID	`"AB1234"`
2	member 1 name	`"Freddie Jones"`
3	member 1 email address	`"FreddieJ909@Cambridge.org"`
4	member 2 ID	`"BC2345"`
5	member 2 name	`"Sue Smith"`
6	member 2 email address	`"Sue1024@Cambridge.org"`

The member ID string is always two letters followed by four digits.

The file design is to be changed so that information for each user is stored as a single line of the file, with the character `'\'` used as a separator between data items.

For example, the single line for member 1 will be:

```
"AB1234\Freddie Jones\FreddieJ909@Cambridge.org"
```

An algorithm will produce a new file `NewFile.txt` from the contents of `OldFile.txt`

Assume:

- `LineX`, `LineY` and `LineZ` are of type `STRING` and are used to store the three items of information for each member
- `NewString` is a temporary variable of type `STRING`
- `OldFile.txt` exists and contains valid data.

(a) The algorithm to create the new file is expressed in steps.

Complete the following numbered steps:

Step 1 : open the file ..... in .....

Step 2 : open the file ..... in .....

Step 3 : read a line from ..... and store in .....

Step 4 : read a line from ..... and store in .....

Step 5 : read a line from ..... and store in .....

Step 6 : set `NewString` to .....

Step 7 : write `NewString` to .....

Step 8 : repeat from step ..... until .....

Step 9 : close both files. (1)

[2]

(b) The character `'\'` has been chosen as a separator.

Explain why this is a suitable character.

.....

..... [1]

(c) Two new information items are to be stored for each member. Both new items are encrypted; they can each contain any character (including `'\'`) and can be of any length up to a maximum of 99 characters.

For example:

Information item	Example data
member 1 ID	`"AB1234"`
member 1 name	`"Freddie Jones"`
member 1 email address	`"FreddieJ909@Cambridge.org"`
member 1 new information 1	`"En/98&*(|?\/D7iP"`
member 1 new information 2	`"23\CoboL"`

Explain the additional file design changes needed so that:

- all the information items for each member are stored on a single line of `NewFile.txt`
- it is possible to extract each information item after the line is read.

Assume the `'\'` character is not used in any email address.

.....

.....

.....

.....

.....

.....

.....

.....

..... [3]

8 A program is being developed to manage student book loans from a college library.

The programmer has defined a record type to define each loan.

The data items are:

Data item	Data type	Comment
`StudentID`	`STRING`	the unique ID of the student who has borrowed the book
`BookID`	`STRING`	the unique ID of the book being borrowed
`OnLoan`	`BOOLEAN`	`TRUE` if the book has **not** been returned

The programmer has defined a global array `Loan` to store 7000 loan records.

There are more elements in the array than books in the library. Unused elements have the `StudentID` set to an empty string. These may occur anywhere in the array.

The programmer has defined a program module:

Module	Description
`CountLoans()`	• called with a parameter of type `STRING` representing a `StudentID` • counts the number of books currently on loan to the specified student • counts the number of books that the student has already returned • output both counts together with a suitable message

(a) Write pseudocode for module `CountLoans()`

[7]

(b) As a reminder, a global array `Loan` stores 7000 loan records with data items for each loan record:

Data item	Data type	Comment
`StudentID`	`STRING`	the unique ID of the student who has borrowed the book
`BookID`	`STRING`	the unique ID of the book being borrowed
`OnLoan`	`BOOLEAN`	`TRUE` if the book has **not** been returned

A new module is defined:

Module	Description
`NewLoan()`	• called with two parameters of type `STRING` representing a `StudentID` and a `BookID` • searches the array for an unused loan record • if found, updates the loan record and returns `TRUE`, otherwise returns `FALSE`

Write efficient pseudocode for `NewLoan()`

.... [7]

(c) When a book is returned, the loan record will have its `OnLoan` data set to `FALSE`

A new procedure `Archive()` will mark these records as unused after first saving them for future reference.

Explain how these records can be saved for future reference.

.....

.....

.....

..... [2]

(d) It is decided to extend the program so that a new module `Reminder()` will send an email to the student three days before the book is due to be returned.

Outline the changes that will need to be made to the data stored and how this data will be used to generate the reminder email.

Data .....

.....

.....

.....

Use .....

.....

.....

.....

[2]

8 A program is being developed to manage student book loans from a college library.

The programmer has defined a record type to define each loan.

The data items are:

Data item	Data type	Comment
`StudentID`	`STRING`	the unique ID of the student who has borrowed the book The first three characters of a `StudentID` represent a tutor ID. Each student has one tutor.
`BookID`	`STRING`	the unique ID of the book being borrowed
`OnLoan`	`BOOLEAN`	`TRUE` if the book has **not** been returned

The programmer has defined a global array `Loan` to store 8000 loan records.

There are more elements in the array than books in the library. Unused elements have the `StudentID` set to an empty string. These may occur anywhere in the array.

The programmer has defined a program module:

Module	Description
`LoanStatus()`	• called with two parameters of type `STRING` representing a `StudentID` and a `BookID` • outputs a message saying whether a given loan has been returned or not • outputs a warning message if a record of the given loan is **not** found

(a) Write efficient pseudocode for the module `LoanStatus()`

Assume that each combination of `StudentID` and `BookID` can occur only once.

[8]

(b) A second module is defined:

Module	Description
`LoansPerTutor()`	• called with a parameter of type `STRING` representing a tutor ID (as a reminder, the first three characters of a `StudentID` represent a tutor ID) • returns an integer value representing the number of books currently on loan to students who have the given tutor

Reminder: unused elements have the `StudentID` set to an empty string. These may occur anywhere in the array.

Write pseudocode for the module `LoansPerTutor()`

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....[7]

(c) It is decided to mark as unused all records for book loans that have been returned. The data for these records will first be written to a new text file for archive purposes.

(i) The archive program will automatically generate a meaningful filename each time it is run.

Outline a meaningful format to use for the filename.

.....

.....

.....

..... [2]

(ii) There is a problem that will need to be overcome before the data items can be written to a text file.

As a reminder, the data items are:

Data item	Data type	Comment
`StudentID`	`STRING`	the unique ID of the student who has borrowed the book The first three characters of a `StudentID` represent a tutor ID. Each student has one tutor.
`BookID`	`STRING`	the unique ID of the book being borrowed
`OnLoan`	`BOOLEAN`	`TRUE` if the book has **not** been returned

Explain the problem.

.....

..... [1]

(iii) One way of storing the data items in a text file is to store each data item on a separate line.

Identify **one** benefit and **one** drawback of this way of storing the data.

Benefit .....

.....

Drawback .....

.....

[2]

2 A program reads string data from a text file into a linear queue and then compresses the data.

The queue is created using the global 1D array `Queue`. The queue can store up to 100 elements. Each element is initialised to the empty string `""`

The queue has two pointers that are global to the program:

- `QueueHead` that points to the index of the first element in the queue, initialised to `-1`
- `QueueTail` that points to the index of the last element in the queue, initialised to `-1`

The global variable `NumberItems` stores the number of elements stored in the queue, initialised to `0`

(a) Write program code to declare and initialise `Queue`, `QueueHead`, `QueueTail` and `NumberItems`

Save your program as **Question2_N25**.

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

[2]

(b) The function `Enqueue()` takes a string as a parameter. The function checks if the queue is full, and returns Boolean `FALSE` if the queue is full.

If the queue is not full, the parameter is inserted into the next position in `Queue`. The function updates the appropriate pointer(s), updates `NumberItems` and then returns Boolean `TRUE`

Write program code for `Enqueue()`

Save your program.

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

[5]

(c) The function `Dequeue()` returns the string `"False"` if the queue is empty.

If the queue is not empty, the function returns the next element in the queue. The function updates the appropriate pointer(s) and updates `NumberItems`

Write program code for `Dequeue()`

Save your program.

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

[3]

(d) The text file `BinaryData.txt` stores individual binary digits, `'1'` and `'0'`. Each digit is on a new line. For example, the first line in the text file stores `'1'`, the second line stores `'1'`

The procedure `ReadData()` reads in each line from the text file `BinaryData.txt` and inserts it into the queue using the appropriate method.

The procedure needs to work for a text file with any number of lines up to a maximum of 100.

Write program code for `ReadData()`

Save your program.

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

[5]

(e) The string data in the text file is compressed.

The compression algorithm counts the number of times each binary digit appears consecutively, then stores the binary digit followed by the number of times it appears. The algorithm stores the compressed data in a single string.

For example, if the text file contains the data:

```
1
1
0
0
0
1
1
1
```

The compression algorithm will create the string `"120313"` because there are two `'1'` digits, followed by three `'0'` digits, followed by three `'1'` digits.

The procedure `Compress()` uses `Dequeue()` to remove each element from the queue in turn. The procedure then compresses the data following the compression algorithm described. The new compressed string is stored in the global variable `NewString`

You can assume that one binary digit will never appear more than nine times consecutively in the sequence.

You can assume that there will always be at least one item in the queue.

Write program code for `Compress()`

Save your program.

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

[6]

(f) **(i)** Write program code for the main program to:

- call `ReadData()`
- call `Compress()`
- output the content of the compressed string.

Save your program.

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

[2]

(ii) Test your program.

Take a screenshot of the output(s).

Save your program.

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

[1]

7 A program is being developed to implement a customer loyalty scheme for a coffee shop.

The programmer has decided that the following data items need to be stored for each customer:

Data item	Description
customer ID	a unique six-digit string
points	an integer value that is increased by one for every cup of coffee the customer orders

When a customer visits the shop and orders coffee, the scheme operates as follows:

- The total number of points is increased by the number of coffees ordered.
- If just one cup of coffee is ordered and the number of points goes above 10, then:
  - the cup of coffee they have just ordered is given to them free of charge
  - the number of points is reduced by 11.
- If the order is for multiple coffees and the number of points goes above 10, then:
  - they get one coffee free of charge for every 11 points
  - the number of points is reduced by 11 for each free coffee.

For example, the:

- customer currently has 9 points
- customer orders 16 cups of coffee
- total number of points now becomes 25
- customer gets 2 free coffees and now has 3 points left.

The programmer has defined a program module that is called every time a customer places an order:

Module	Description

`CustomerOrder()`	• called with two integer parameters:
	  ◦ the number of coffees ordered
	  ◦ the current number of points
	• output a suitable message giving the number of free coffees
	• return a value for the new points total.

(a) Write pseudocode for module `CustomerOrder()`

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [6]

(b) A text file `Loyalty.txt` will be used to store the data items for the loyalty scheme. The data items for each customer will be stored on a separate line of the text file where each data item is separated by a comma:

```
<CustomerID>,<Points>
```

The contents of the text file `Loyalty.txt` will always be stored in ascending order by customer ID.

When the data items are read from or written to the text file `Loyalty.txt`, they may need to be converted to the appropriate data type.

Each customer has a unique customer ID starting at `"100001"` with this value increasing by one each time a new customer joins the loyalty scheme.

When a customer joins the loyalty scheme, they are assigned the next customer ID and value of points is set to 0.

For example, if the loyalty scheme has 204 customers and a new customer joins the loyalty scheme, the following line is added to the text file `Loyalty.txt`:

```
"100205,0"
```

You can assume that the number of customers in the loyalty scheme will never be more than 9000.

The programmer has defined a program module as follows:

Module	Description
`AddNewCustomers()`	• called with an integer parameter representing the number of new customers to be added to the loyalty scheme • adds a new line, containing the required information, to the text file `Loyalty.txt` for each customer added to the loyalty scheme • outputs each new customer ID added to the loyalty scheme

(i) Write pseudocode for module `AddNewCustomers()`

Assume that there is at least **one** customer already in the loyalty scheme.

.....

.....

.....

.....

.....

.....

.....

[8]

(ii) The requirements for `AddNewCustomers()` are changed. There may be no existing customers in the loyalty scheme.

Explain the changes that will need to be made to the module `AddNewCustomers()`

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [4]

4 A programmer is developing a new stock control program for a shop owner to produce sales reports.

(a) A text file `Sales.txt` stores strings representing the number of items sold.

The file contains 52 lines, each representing the number of items sold in each week of the year.

For example:

File line number	File data
1	"350"
2	"502"
3	"434"
49	"492"
50	"872"
51	"772"
52	"201"

The example shows that 502 items were sold in week 2.

The owner requires a module to output all week numbers where the number of items sold is greater than 500.

Describe an algorithm that produces the required module.

Do **not** include pseudocode statements in your answer.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [5]

(b) Once the program has been completed, it is tested using the walkthrough method.

Describe the walkthrough method and explain how it can be used to identify errors.

.....

.....

.....

.....

.....

.....

.....

..... [3]

7 A program is being developed to implement a customer loyalty scheme for a coffee shop.

The programmer has decided that the following data items need to be stored for each customer:

Data item	Description
customer ID	a unique six-digit string
loyalty points	an integer value that depends on how often the customer visits the coffee shop
last visit date	the date the customer last visited the coffee shop

The programmer has defined a program module:

Module	Description
`UpdateVisit()`	• called with two parameters: 1. loyalty points of type `INTEGER` 2. last visit date of type `DATE` • change the loyalty points: ○ increase by **four** if the last visit date and the current date are in the same month ○ otherwise increase by **one** • change the last visit date to the current date • the changed values must be available to the code that follows the call to `UpdateVisit()`

(a) (i) Write pseudocode for module `UpdateVisit()`

Assume the customer has made at least **one** previous visit.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

[5]

(ii) The programmer decides to amend the `UpdateVisit()` module so that loyalty points are further increased if the customer visits the coffee shop the same day of the week as their last visit.

Write the pseudocode for this condition.

.....

..... [1]

(b) A text file `Loyalty.txt` will be used to store the data items for the loyalty scheme.

The text file `Loyalty.txt` contains only one line of data for each customer.

Each data item is separated by a comma:

`<CustomerID>,<LoyaltyPointsString>,<LastVisitDateString>`

`LastVisitDateString` is always eight characters in length in the format: DDMMYYYY.

An example of how a line of data will be stored in the text file `Loyalty.txt` is:

```
"100123,132,05032025"
```

This example shows that the customer with an ID of 100123 had 132 loyalty points following their last visit to the coffee shop on 05/03/2025.

If a customer visits the coffee shop on a Monday, the value of their loyalty points is increased by 10.

The programmer has defined **two** more program modules:

Module	Description
`FindCustomer()` (is already written)	• called with a parameter of type `STRING` that represents a customer ID • returns the line of data read from the text file `Loyalty.txt` containing the data items for that customer
`MondayCheck()`	• called with a parameter of type `STRING` that represents the customer ID of a customer • calls `FindCustomer()` • extracts the loyalty points from the string returned by `FindCustomer()` • increases the loyalty points for the current customer by 10 if the day visited is a Monday • returns an integer value representing the loyalty points

(i) Write pseudocode for module `MondayCheck()`

The module `FindCustomer()` must be used and assume it returns a valid string for the current customer.

.... [8]

(ii) A date is stored in `LastVisitDateString` in the format:

DDMMYYYY

DD is a 2-digit string
MM is a 2-digit string
YYYY is a 4-digit string.

For example, the date 03/09/2024 is stored as `"03092024"`

An algorithm is needed to convert the string stored in `LastVisitDateString` to data type `DATE` which is then stored in the variable `LastVisitDate`.

Complete the pseudocode for this algorithm:

Assume that `LastVisitDateString` has been declared and contains valid data.

```
DECLARE DayInt, MonthInt, YearInt : INTEGER

DECLARE LastVisitDate : DATE
```

.....

.....

.....

.....

.....

.....

.....

.....

[4]

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]