8 A program is being developed to manage student book loans from a college library. Students may borrow up to five books at a time from the library.

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

The record 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 5000 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 the first program module:

Module	Description
`OKToBorrow()`	• called with a parameter of type `STRING` representing a `StudentID` • search the array for loan records for the specified student • output a suitable message to say whether the student may, or may not, borrow another book

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

[7]

(b) A second module is defined:

Module	Description
`ReturnBook()`	• called with two parameters of type `STRING` representing a `StudentID` and a `BookID` • searches the array for the relevant loan record • when found, sets `OnLoan` to `FALSE` and returns `TRUE` • if a loan record is **not** found, or the book has already been returned, then returns `FALSE`

Write efficient pseudocode for the module `ReturnBook()`

Assume that each student is only allowed to borrow each book only once. That means that there will be no more than **one** loan record for a given combination of student and book.

[8]

(c) It is decided to introduce a system of fines for books that have been borrowed for too long.

Two new requirements are defined:

1. Each loan has a maximum length, represented as a number of days.
2. Each book in the library will be assigned one of three categories. Each category has a different maximum loan length.

Record structure and program design changes are needed to meet these two requirements.

Outline the changes that are necessary to meet the **two** requirements.

[2]

3 A program stores integers in ascending order in an ordered binary tree. The tree is implemented as a 2D array.

Each node is stored with three values:

- a pointer to the left node
- the data
- a pointer to the right node.

All null values are stored as `-1`

The binary tree can store up to 50 nodes.

Nodes cannot be deleted from the binary tree.

(a) The binary tree is stored as a global array with the identifier `TreeArray`. The left pointer, the data and the right pointer of each node are initialised to `-1`

The global variable `RootPointer` stores the index of the root node in the tree, initialised to `-1`

The global variable `FreeNode` stores the index of the next free node in the array, initialised to `0`

Write program code to declare and initialise `TreeArray`, `RootPointer` and `FreeNode`

Save your program as **Question3_N25**.

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

[3]

(b) The procedure `AddNode()`:

- takes an integer to store in the binary tree as a parameter
- stores the parameter in the next free node in the array
- finds the position to store the data in the tree by following the appropriate pointers
- updates the pointer of the new node's parent node.

The procedure outputs `"The tree is full"` if the parameter cannot be stored because the tree is full.

Write program code for `AddNode()`

Save your program.

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

[7]

(c) The text file `TreeData.txt` stores 50 integers. Each integer is on a new line in the file.

The main program reads each integer from the file and stores each integer in the binary tree in the order they are read.

Write program code for the main program.

Save your program.

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

[4]

(d) The procedure `WriteAllToFile()` stores the content of `TreeArray` in a new text file with the filename `Tree.txt`. The file `Tree.txt` is not provided.

Each node in `TreeArray` is stored on one line with a comma separating each value.

For example, the current contents of `TreeArray` are:

Index	0	1	2
0	-1	20	1
1	-1	30	-1
…			
49	-1	-1	-1

After writing `TreeArray` to the text file, `Tree.txt` will contain:

```
-1,20,1
-1,30,-1
…
-1,-1,-1
```

Write program code for `WriteAllToFile()`

Include exception handling when writing to the file.

Save your program.

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

[5]

(e) (i) Amend the main program to call `WriteAllToFile()`

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

[1]

(ii) Test your program.

Take a screenshot that shows all of the content stored in the file `Tree.txt`

In this screenshot you need to make sure the filename is visible.

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

[1]