3 A program stores records in the 2D array `HashTable`. Each record is stored at a specific index of the array that is calculated using a hashing algorithm with the record's key field.

The array has 100 × 10 elements. The hashing algorithm uses the key to generate an index between 0 and 99 (inclusive). If two key fields generate the same index, there is a collision. Any records that have a collision are stored in the next space in the same index.

For example: In this table two record keys generated the same hash value of 1. Four record keys generated the same hash value of 3.

Index	**0**	**1**	**2**	**3**	**4**	**5**	**…**	**9**
0	record							
1	record	record						
2								
3	record	record	record	record				
4								
…								
99								

The program uses Object-Oriented Programming (OOP).

(a) The class `Record` stores data about the records:

Record	
`Key : Integer`	stores the integer key field for the data
`Data : String`	stores the string data
`Constructor()`	initialises `Key` and `Data` to its parameter values

The attributes `Key` and `Data` are public.

Write program code to declare the class `Record` and its constructor.

Use your programming language appropriate constructor.

Save your program as **Question3_N25**.

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

[2]

(b) The procedure `InitialiseHashTable()` initialises each element in the array to an empty or null record.

Write program code to declare the global 2D array `HashTable` and the procedure `InitialiseHashTable()`

Save your program.

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

[2]

(c) The function `Hash()`:

- takes an integer key field as a parameter
- calculates and returns the hash value of the key field.

The hash value is the result from the formula: `key MOD 100`

Write program code for `Hash()`

Save your program.

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

[2]

(d) The procedure `InsertData()`:

- takes an object of type `Record` as a parameter
- calculates the hash value for the parameter using the appropriate function
- stores the parameter in the correct position in `HashTable`

You can assume there will be no more than 10 objects that generate the same hash value.

Write program code for `InsertData()`

Save your program.

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

[4]

(e) The file `"HashTableData.txt"` stores 200 key values and string data items in the format:

```
key,string
```

For example, the first row in the text file is:

```
528,permission
```

The key is `528` and the string data is `"permission"`

The procedure `ReadData()`:

- opens the text file and reads each line
- splits each line into the key and data
- calls `InsertData()` with an object containing each key and matching data.

Write program code for `ReadData()`

Save your program.

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

[5]

(f) The function `GetRecord()`:

- takes an integer key field as a parameter
- calculates the hash value for the key field using the appropriate function
- searches the hash table for the record with the matching key field
- returns the data for the record if the record is found
- returns `"Not found"` if the record is not found.

Write program code for `GetRecord()`

Save your program.

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

[5]

(g) The main program:

- calls `InitialiseHashTable()` and `ReadData()`
- takes **five** integer key fields as input from the user
- calls `GetRecord()` with each input and outputs the return value.

(i) Write program code for the main program.

Save your program.

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

[3]

(ii) Test your program with the following **five** inputs in the order given:

528

1128

1828

1062

39

Take a screenshot of the output(s).

Save your program.

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

[2]

11 The pseudocode algorithm below allows a user to input a new stock item. The random file is searched for the next empty location in the file and the new item is inserted there.
A suitable message is displayed if the file is full.

Complete this pseudocode.

```
DECLARE Location : INTEGER
DECLARE NewStock : STRING
DECLARE CurrentStock : STRING
DECLARE Stored : BOOLEAN
DECLARE Max : INTEGER
Max ← 100000
Stored ← FALSE
Location ← 1

.....
OUTPUT "Enter the new item you wish to store: "
INPUT NewStock
WHILE NOT Stored AND Location <= Max

   .....

   GETRECORD "StockList.dat", .....
   IF CurrentStock = "" THEN

      ..... "StockList.dat", NewStock
      Stored ← TRUE
   ELSE
      Location ← Location + 1
   ENDIF
ENDWHILE

..... THEN
   OUTPUT "The new item has not been stored as the file was full."
ENDIF
CLOSEFILE "StockList.dat"
```

[5]

2 **(a)** Describe **serial file organisation** as a method of storing data records in a file.

.....

.....

.....

..... [2]

(b) State **one** example of a use for serial file organisation.

.....

..... [1]

5 **(a)** Explain what is meant by a hashing algorithm in the context of file access.

.....

.....

.....

.....

.....

..... [3]

(b) The use of a hashing algorithm can result in the same storage location being identified for more than one record.

Outline **two** methods of overcoming this issue.

1 .....

.....

2 .....

.....

[2]