2 A program stores data in a 1D array of records, `HashTable`. The array has space for 200 records. Each data item is stored in a specific index of the array that is calculated using an algorithm. The index is calculated from the key field using the formula: `Key MOD 200`

If two key fields generate the same index, there is a collision. Any records that have a collision are stored in a second array, `Spare`. This array has space for 100 records. When a collision is detected, the record is stored in the next free space in `Spare`.

(a) The pseudocode record format is:

```
TYPE NewRecord

  DECLARE Key : INTEGER

  DECLARE Item1 : INTEGER

  DECLARE Item2 : INTEGER

ENDTYPE
```

Write program code to declare the record type `NewRecord`

If your chosen programming language does **not** support record formats, a class can be used instead.

Save your program as **Question2_J25**.

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

[2]

(b) **(i)** Write program code to declare the global arrays `HashTable` and `Spare`

Save your program.

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

[1]

(ii) An empty record has the integer `-1` stored in each field.

The procedure `Initialise()` stores an empty record in each element in `HashTable` and `Spare`

Write program code for `Initialise()`

Save your program.

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

[2]

(c) The hash value is calculated from the key field using the formula: `Key MOD 200`

The function `CalculateHash()` takes a key field as a parameter and calculates and returns the hash value for the key field.

Write program code for `CalculateHash()`

Save your program.

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

[2]

(d) The procedure `InsertIntoHash()`:

- takes a record of type `NewRecord` as a parameter
- uses `CalculateHash()` to calculate the hash value for the key field in the record
- checks if the hash value index in `HashTable` currently stores an empty record
  - if the index stores an empty record, store the parameter in this index
  - if the index does **not** store an empty record, store the parameter in `Spare`

You can assume there will always be enough space in `Spare` to store any collisions.

Write program code for `InsertIntoHash()`

Save your program.

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

[6]

(e) The text file `HashData.txt` stores up to 200 rows of data to be stored into the hash table. Each row contains three integer numbers separated by commas.

The first number is the key field, the second number is item 1 and the third number is item 2.

For example:

The first row in the text file contains: `646, 12, 568`

The key field is `646`, item 1 is `12` and item 2 is `568`

The procedure `CreateHashTable()`:

- opens the file `HashData.txt`
- creates a record for each row of data in the file
- calls `InsertIntoHash()` with each record.

Write program code for `CreateHashTable()`

Save your program.

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

[5]

(f) **(i)** The procedure `PrintSpare()` outputs the key field of each element in the array `Spare` that does **not** contain an empty record.

Write program code for `PrintSpare()`

Save your program.

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

[2]

(ii) The main program should call the procedure to initialise the arrays, call the procedure to create the hash table and then call the procedure to output the contents of the array `Spare`

Write program code for the main program.

Save your program.

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

[1]

(iii) Test your program.

Take a screenshot of the output(s).

Save your program.

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

[1]