3(a)

1 mark each
- Class header (and end) **and** constructor header (and end) in class
- Constructor takes two parameters **and** stores each in attributes

Example program code

Java
```
class Record{
      public Integer Key;
      public String Data;

      public Record(Integer pKey, String pData){
            Key = pKey;
            Data = pData;
      }
}
```

VB.NET
```
Class Record
    Dim Key As Integer
    Dim Data As String
    Sub New(pKey, pData)
        Key = pKey
        Data = pData
    End Sub
End Class
```

Python
```
class Record:
    def __init__(self, pKey, pData):
        self.Key = pKey #integer
        self.Data = pData #string
```

3(b)

1 mark each
- 2D array of 100 × 10 elements of type `Record`
- Procedure `InitialiseHashTable()` header (and end) **and** initialises each element in the 2D array to an empty/null record in procedure

Example program code

Java
```
public static Record[][] HashTable = new Record[100][10];
public static void InitialiseHashTable(){
      Record EmptyRecord = new Record(-1,"-1");

      for(Integer X = 0; X < 100; X++){
            for(Integer Y = 0; Y < 10; Y++){
                  HashTable[X][Y] = EmptyRecord;
            }
      }
}
```

VB.NET
```
Dim HashTable(99, 9) As Record
Sub InitialiseHashTable()
    Dim EmptyRecord As Record = New Record(-1, "")
    For X = 0 To 99
        For Y = 0 To 9
            HashTable(X, Y) = EmptyRecord
        Next
    Next
End Sub
```

Python
```
HashTable = []
def InitialiseHashTable():
    global HashTable
    HashTable = [[Record(-1,"")]*10 for i in range(100)]
```

3(c)

1 mark each
- Function header (and end) taking one parameter **and** returning calculated hash
- …. hash calculated correctly from parameter

Example program code

Java
```
public static Integer Hash(Integer TheKey){

      return(TheKey % 100);
}
```

VB.NET
```
Function Hash(Key)
    Return Key Mod 100
End Function
```

Python
```
def Hash(Key):
    return Key % 100
```

3(d)

1 mark each
- Procedure header (and end) taking one `Record` parameter
- Calling `Hash()` using key from parameter **and** storing/using return value
- Accessing `HashTable[return][0]` **and** storing parameter if no collision … … if collision: iterating through 2nd dimension to find empty index **and** store parameter in that position

Example program code

Java
```
public static void InsertData(Record RecordData){

      Integer HashValue = Hash(RecordData.Key);

      for(Integer X = 0; X < 10; X++){

            if(HashTable[HashValue][X].Key.equals(-1)){
                  HashTable[HashValue][X] = RecordData;
                  X = 10;
            }
      }
}
```

VB.NET
```
Function InsertData(RecordData)
   Dim HashValue As Integer = Hash(RecordData.Key)

    For X = 0 To 9
        If HashTable(HashValue, X).Key = -1 Then
            HashTable(HashValue, X) = RecordData
            X= 10
        End If
    Next X
End Function
```

3(d)

Python

```
def InsertData(RecordData):
    global HashTable

    HashValue = Hash(RecordData.Key)

    for X in range(0, 10):
        if HashTable[HashValue][X].Key == -1:
            HashTable[HashValue][X] = RecordData
```

3(e)

1 mark each to max 5

- Procedure header (and end), opening file **and** closing file (in appropriate place)
- Iterating through each line in file // reading each line in from file
- Splitting each line read in by comma …
- … creating `Record` object with **each** key and data as arguments …
- … calling `InsertData()` with each **object**
- Try, catch with appropriate output and all file access within try

Example program code

Java

```
public static void ReadData(){
      String[] Data = new String[3];
      Integer NewKey;
      Integer NewItem1;
      Integer NewItem2;
      Record TheRecord;
      try{
            FileReader File = new FileReader("HashTableData.txt");
            try{
                  BufferedReader Reader = new BufferedReader(File);
                  String Line= Reader.readLine();
                  while (Line != null){
                        Line = Line.replace("\n","");
                        Data = Line.split(",");
                        TheRecord = new Record(Integer.parseInt(Data[0]), Data[1]);

                        InsertData(TheRecord);
                        Line= Reader.readLine();
                  }
                  Reader.close();
            }catch(IOException ex){}

      }catch(FileNotFoundException e){System.out.println("File not found");}
}
```

3(e)

VB.NET

```
Sub ReadData()
    Dim Line As String
    Dim Data(3) As String
    Dim TheRecord As Record

    Dim FileReader As New System.IO.StreamReader("HashTableData.txt")
    While Not FileReader.EndOfStream
        Line = FileReader.ReadLine()
        Data = Split(Line, ",")
        TheRecord = New Record(Integer.Parse(Data(0)), Data(1))
        InsertData(TheRecord)
    End While
    FileReader.Close()

End Sub
```

Python

```
def ReadData():
    global HashTable
    File = open("HashTableData.txt")
    for Line in File:
        Data = Line.strip()
        Data = Line.split(",")
        InsertData(Record(int(Data[0]), Data[1]))
    File.close()
```

3(f)

1 mark each

- Function header (and end) taking one parameter **and** returning string in all cases
- Calling `Hash()` with parameter **and** storing/using return value
- Iterating through 2nd dimension at `HashTable[return value]` **and** comparison to parameter …
- … returning data if found/equal
- … returning `"Not found"` if not found by the end of the dimension

Example program code

Java

```
public static String GetRecord(Integer Key){
      Integer HashValue = Hash(Key);
      for(Integer X = 0; X < 10; X++){
            if(HashTable[HashValue][X].Key.equals(Key)){
                  return(HashTable[HashValue][X].Data);
            }
      }
      return "Not found";
}
```

VB.NET

```
Function GetRecord(Key)
    Dim HashValue As Integer = Hash(Key)

    For X = 0 To 9
        If HashTable(HashValue, X).Key = Key Then
            Return HashTable(HashValue, X).Data
        End If
    Next X
    Return "Not found"
End Function
```

3(f)

Python

```
def GetRecord(Key):
    global HashTable

    HashValue = Hash(Key)
    for X in range(0, 10):
        if HashTable[HashValue][X].Key == Key:
            return HashTable[HashValue][X].Data

    return "Not found"
```

3(g)(i)

1 mark each

- Calling `InitialiseHashTable()` then `ReadData()`
- Taking five (integer) inputs
- Calling `GetRecord()` with **each** input **and** outputting return value

Example program code

Java

```
public static void main(String args[]){
      InitialiseHashTable();
      ReadData();
      Scanner scanner = new Scanner(System.in);
      for(Integer X = 0; X < 5; X++){
            System.out.println("Enter key field");
            System.out.println(GetRecord(Integer.parseInt(scanner.nextLine())));

      }
}
```

VB.NET

```
Sub Main(args As String())
    InitialiseHashTable()
    ReadData()

    For X = 0 To 5
        Console.WriteLine("Enter key field")
        Console.WriteLine(GetRecord(Console.ReadLine()))
    Next
End Sub
```

3(g)(i)

Python

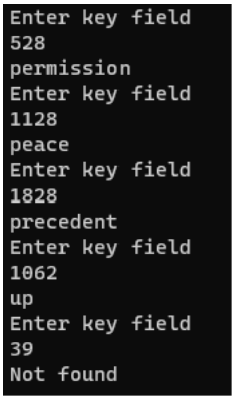
```
InitialiseHashTable()
ReadData()

for x in range(5):
    Key = int(input("Enter key field "))
    print(GetRecord(Key))
```

3(g)(ii)

1 mark each for screenshot(s) showing

- Input of the 4 integers and matching word output
  `528  permission`
  `1128  peace`
  `1828  precedent`
  `1062  up`
- Input of `39` and output of `Not found`

e.g.



12

One mark for each correctly completed line (**Max 5**)

```
DECLARE Location : INTEGER
DECLARE Item : STRING
DECLARE Continue : BOOLEAN
DECLARE Answer : CHAR
Continue ← TRUE
OPENFILE "StockList.dat" FOR RANDOM
WHILE Continue
   OUTPUT "Enter a location between 1 and 500: "
   INPUT Location
   SEEK "StockList.dat", Location
   GETRECORD "StockList.dat", Item
   IF Item = "" THEN
      OUTPUT "This record is missing"
   ELSE
      OUTPUT "The item in stock is ", Item
   ENDIF
   OUTPUT "Another location (Y or N)?"
   INPUT Answer
   IF Answer <> 'Y' THEN
      Continue ← FALSE
   ENDIF
ENDWHILE
CLOSEFILE "StockList.dat"
OUTPUT "End of program"
```

13	**One** mark for each correctly completed line
	
	`DECLARE Grade : StudentResult`
	`DECLARE Position : INTEGER`
	`OPENFILE "CurrentResults.dat" FOR RANDOM`
	`OPENFILE "StoredResults.dat" FOR RANDOM`
	`FOR Position ← 1 TO 50`
	`  SEEK "CurrentResults.dat", Position`
	`  GETRECORD "CurrentResults.dat", Grade`
	`  IF Grade.ExamGrade = "" THEN`
	`  Grade.ExamGrade ← "Missing grade"`
	`  ENDIF`
	`  SEEK "StoredResults.dat", Position`
	`  PUTRECORD "StoredResults.dat", Grade`
	`NEXT Position`
	`CLOSEFILE "CurrentResults.dat"`
	`CLOSEFILE "StoredResults.dat"`

11	**One** mark for each correctly completed line
	`DECLARE Location : INTEGER`
	`DECLARE NewStock : STRING`
	`DECLARE CurrentStock : STRING`
	`DECLARE Stored : BOOLEAN`
	`DECLARE Max : INTEGER`
	`Max ← 100000`
	`Stored ← FALSE`
	`Location ← 1`
	`OPENFILE "StockList.dat" FOR RANDOM`
	`OUTPUT "Enter the new item you wish to store"`
	`INPUT NewStock`
	`WHILE NOT Stored AND Location <= Max`
	`  SEEK "StockList.dat", Location`
	`  GETRECORD "StockList.dat", CurrentStock`
	`  IF CurrentStock = "" THEN`
	`  PUTRECORD "StockList.dat", NewStock`
	`  Stored ← TRUE`
	`  ELSE`
	`  Location ← Location + 1`
	`  ENDIF`
	`ENDWHILE`
	`IF Stored = FALSE THEN`
	`  OUTPUT "The new stock item has not been stored as the file was full"`
	`ENDIF`
	`CLOSEFILE "StockList.dat"`

2(a)	**One** mark per mark point (**Max 2**) MP1 Records are stored one after the other as they are collected // records are stored in chronological order MP2 New records are appended to the end of the file.
2(b)	**One** mark for a use (**Max 1**) MP1 Creating unsorted / temporary transaction files MP2 Creating data logging files

5(a)	**One** mark per mark point (**Max 3**) MP1 A hashing algorithm is used in direct access methods on random and sequential files MP2 It is a mathematical formula MP3 … used to perform a calculation applied to the key field of the record being searched / stored MP4 The result of the calculation gives the address where the record should be found / stored.
5(b)	**One** mark per mark point (**Max 2**) MP1 The record is stored in the next free memory space after the one identified by the hashing algorithm // Use linear progression MP2 An overflow area is set up and the record is stored in the next free memory space in the overflow area.