3(a)

Mark as follows:

MP1	1		open the file **OldFile.txt** in **read (mode)**
MP2	2		open the file **NewFile.txt** in **write (mode)**
	3	⌐	read a line from **OldFile.txt** and store in **`LineX`**
MP3	4	⌐	read a line from **OldFile.txt** and store in **`LineY`**
	5	⌐	read a line from **OldFile.txt** and store in **`LineZ`**
MP4	6		set `NewString` to **`LineX & '\' & LineY & '\' & LineZ`**
MP5	7		write `NewString` to **NewFile.txt**
MP6	8		repeat from step **3** until **end of OldFile.txt**
	9		close both files

3(b)

It does not appear in (any of) the data items
// it does not appear in the ID **and** Name

3(c)

MP1 Add a (**single)** new separator **at the end of the third (original) item**
MP2 Calculate the **length** of **both new data items**
MP3 Store / append the **length** (of the new each items)
MP4 as a fixed **length** / separated digit string // by example

Max 3

8(a)

Example solution:

```
PROCEDURE CountLoans(ThisStudentID : STRING)
   DECLARE Index, LCount, RCount : INTEGER

   LCount ← 0
   RCount ← 0

   FOR Index ← 1 TO 7000
      IF Loan[Index].StudentID = ThisStudentID THEN
         IF Loan[Index].OnLoan = FALSE THEN
            RCount ← RCount + 1
         ELSE
            LCount ← LCount + 1
         ENDIF
      ENDIF
   NEXT Index

   OUTPUT "Student has ", LCount, " books on loan"
   OUTPUT "and has returned ", RCount, " books"

ENDPROCEDURE
```

Mark as follows:

MP1 Declaration of Index **and** two count variables
MP2 Loop for 7000 iterations
MP3 `Index` references an **indexed data** item with correct **.dot notation**
MP4 Attempt to compare the `StudentID` field = **parameter in a loop**
MP5 Test of the `OnLoan` field **and** correct logic for both outcomes
MP6 Both count variables initialised **and** increment appropriate count **in a loop**
MP7 Output includes both counts with a <u>suitable</u> message

8(b)

Example solution:

```
FUNCTION NewLoan(ThisStudentID, ThisBookID : STRING)
                                                    RETURNS
BOOLEAN
   DECLARE Index : INTEGER
   DECLARE IsStored : BOOLEAN
   CONSTANT Blank = ""

   Index ← 1
   IsStored ← FALSE

   WHILE Index <= 7000 AND NOT IsStored
      IF Loan[Index].StudentID = Blank THEN
         Loan[Index].StudentID ← ThisStudentID
         Loan[Index].BookID ← ThisBookID
         Loan[Index].OnLoan ← TRUE
         IsStored ← TRUE
      ENDIF
      Index ← Index + 1
   ENDWHILE

   RETURN IsStored
ENDFUNCTION
```

Mark as follows:

MP1 Loop through 7000 elements in `Loan` array – correct loop structure
MP2 or until the first unused element is found // Correct `RETURN` inside a `FOR` loop
MP3 Correct use of the indexed dot notation
MP4 Test if record is unused (`.StudentID = ""`) **in a loop**
MP5 Two item assignment statements correct
MP6 `Loan[Index].OnLoan` assigned TRUE
MP7 Correct logic to return `BOOLEAN` in both cases (position found / no position available)

Alternative solution: loop only (no 'found' mechanism)

```
   FOR Index ← 1 TO 7000
      IF Loan[Index].StudentID = Blank THEN
         Loan[Index].StudentID ← ThisStudentID
         Loan[Index].BookID ← ThisBookID
         Loan[Index].OnLoan ← TRUE
         RETURN TRUE
      ENDIF
    NEXT
    RETURN FALSE
```

8(c)	**MP1** Save to/Open a **(text) file** **MP2** **Single line** – form a string with separator characters and write to the file // **Multiple lines** - write each data item to a separate line in the file
8(d)	Additional date(s) solution **MP1** Store the **return date** of each loan // the **start date <u>and</u>** the **loan period** **MP2** Algorithm will do a **calculation** involving the MP1 data item(s) OR The student email address solution **MP3** Store the student <u>email address</u> **MP4** In order **to send** the email // the student is alerted to return the book **Max 2**

8(a) Example solution:

```
PROCEDURE LoanStatus(ThisStudentID, ThisBookID : STRING)
   DECLARE Index : INTEGER
   DECLARE Found : BOOLEAN

   Index ← 1 / 0
   Found ← FALSE

   WHILE Index <= 8000 / 7999 AND NOT Found
      IF Loan[Index].StudentID = ThisStudentID AND
                     Loan[Index].BookID = ThisBookID THEN
         IF Loan[Index].OnLoan = FALSE THEN
            OUTPUT "Loan has been returned."
         ELSE
            OUTPUT "Loan has not been returned."
         ENDIF
         Found ← TRUE
      ENDIF
      Index ← Index + 1
   ENDWHILE

   IF NOT Found THEN
      OUTPUT "Warning - Loan not found."
   ENDIF
ENDPROCEDURE
```

Mark as follows:

1 Procedure heading, parameters and ending
2 Loop through **all elements** in `Loan` array
3 ... or until record loan is found
4 Test for correct `StudentID` **in a loop**
5 ... and correct `BookID`
6 ... check `OnLoan` data item
7 ... and output <u>both</u> messages as appropriate regarding book loan status **once**
8 If no matching loan found, Output 'not found' **after the loop**

8(b)	Example solution:
	
	`FUNCTION LoansPerTutor(TutorID : STRING) RETURNS INTEGER`
	`  DECLARE Index, Count : INTEGER`
	
	`  Count ← 0`
	
	`  FOR Index ← 1 / 0 TO 8000 / 7999`
	`  IF LEFT(Loan[Index].StudentID, 3) = TutorID AND`
	`  Loan[Index].OnLoan = TRUE THEN`
	`  Count ← Count + 1`
	`  ENDIF`
	`  NEXT Index`
	
	`  RETURN Count`
	
	`ENDFUNCTION`
	
	Mark as follows:
	1 Initialise local integer for `Count`
	2 Loop through **all elements** in the `Loan` array
	3 Attempt at referencing a data item **in a loop**
	4 Extract `TutorID` from `Loan[Index].StudentID` **in a loop**
	5 Test for Matching Tutor **AND**
	test if the book has been returned using correct dot notation **in a loop**
	6 ... if true then increment `Count` **in a loop**
	7 Return count

8(c)(i)	One mark per point: 1 Include (a suffix string based on) the date of the archive / the next number in sequence 2 Concatenate with a root filename such as 'Archive'
8(c)(ii)	Boolean data cannot be written to a text file // `OnLoan` is not a string // `OnLoan` will have to be converted to text
8(c)(iii)	One mark per point: Benefit: Algorithm to **store / extract** a record is easier // unpacking of data is not required // No need for string concatenation Drawback: More file **accesses needed** (to read / write a complete record)

2(a)

1 mark each

- `Queue` declared as a (global) 1D array of 100 string elements all initialised to `""`
- `QueueHead` and `QueueTail` initialised with `-1`, `NumberItems` initialised to `0`

Example program code

Java

```
public static String[] Queue = new String[100];
public static Integer QueueHead;
public static Integer QueueTail;
public static Integer NumberItems;
for(int x = 0; x < 100; x++){
      Queue[x] = "";
}
QueueHead = -1;
QueueTail = -1;
NumberItems = 0;
```

VB.NET

```
    Dim Queue(99) As String
    Dim QueueHead As Integer
    Dim QueueTail As Integer
    Dim NumberItems As Integer
For x = 0 To 99
     Queue(x) = ""
 Next
 QueueHead = -1
 QueueTail = -1
 NumberItems = 0
```

2(a)

Python

```
global Queue, QueueHead, QueueTail, NumberItems
Queue = []
    for x in range(100):
        Queue.append("")
    QueueHead = -1
    QueueTail = -1
    NumberItems = 0
```

2(b)

1 mark each

- Function header (and end where appropriate) taking one (string) parameter **and** checking full **and** returning `FALSE`
- (otherwise) Storing parameter in `QueueTail + 1`
- Incrementing `QueueTail` and `NumberItems`
- (Dealing with first element) If `QueueHead = -1` store `0` in `QueueHead` / increment `QueueHead`
- Return `TRUE` in **all** cases when inserted

Example program code.

Java

```
public static Boolean Enqueue(String TheData){
      if(QueueHead == -1){
            Queue[0] = TheData;
            QueueHead = 0;
            QueueTail = 0;
            NumberItems++;
            return true;
      }else if(QueueTail >= 99){
            return false;
      }else{
            Queue[QueueTail+1] = TheData;
            QueueTail++;
            NumberItems++;
            return true;
      }

}
```

2(b)

VB.NET

```
Function Enqueue(TheData)
    If QueueHead = -1 Then
        Queue(0) = TheData
        QueueHead = 0
        QueueTail = 0
        NumberItems += 1
        Return True
    ElseIf QueueTail >= 99 Then
        Return False
    Else
        Queue(QueueTail + 1) = TheData
        QueueTail += 1
        NumberItems += 1
        Return True
    End If
End Function
```

Python

```
def Enqueue(TheData):
    global Queue, QueueHead, QueueTail, NumberItems
    if(QueueHead == -1){
         Queue[0] = TheData
         QueueHead = 0
         QueueTail = 0
         NumberItems +=1
         return True
    elif QueueTail >= 99:
        return False
    else:
        Queue[QueueTail+1] = TheData
        QueueTail +=1
        NumberItems +=1
        return True
```

2(c)

1 mark each

- Function header (and end), checking if `Queue` is empty (`NumberItems = 0` or `QueueHead > QueueTail`) **and** returning "`False`"
- (otherwise) returning `Queue[QueueHead]`
- Incrementing `QueueHead`, decrementing `NumberItems`

Example program code

Java

```
public static String Dequeue(){
      String ReturnValue;
      if(NumberItems == 0){
            return "False";
      }else{
            ReturnValue = Queue[QueueHead];
            QueueHead++;
            NumberItems--;
            return ReturnValue;
      }
}
```

VB.NET

```
Function Dequeue()

    If NumberItems = 0 Then
        Return "False"
    Else
        Dequeue = Queue(QueueHead)
        QueueHead += 1
        NumberItems -= 1

    End If
End Function
```

2(c)

Python

```
def Dequeue():
    global Queue, QueueHead, QueueTail, NumberItems
    if NumberItems == 0:
        return "False"
    else:
        ReturnData = Queue[QueueHead]
        QueueHead += 1
        NumberItems -=1
        return ReturnData
```

2(d)

1 mark each to max 5

- Procedure header (and end where appropriate)
- Opening the file **and** closing the file (in appropriate place)
- Looping until EOF // looping through each line …
- … read in each value …
- … calling `Enqueue()` with read in value and store/use return value
- Exception handling with try except and appropriate output with all file access within try

Example program code

Java

```
public static void ReadData(){
           Boolean ReturnValue;
           Boolean FinishLoop = false;
           try{
                Scanner Scanner1 = new Scanner(new File("BinaryData.txt"));
                while(Scanner1.hasNextLine() && FinishLoop == false){
                     ReturnValue = Enqueue(Scanner1.nextLine());
                     if(ReturnValue.equals("False")){
                          FinishLoop = true;
                     }
                }
                Scanner1.close();
           } catch(FileNotFoundException ex){
                System.out.println("No file found");
           }
     }
```

2(d)

VB.NET

```
Sub ReadData()
     Dim ReturnValue As String
     Dim DataReader As New System.IO.StreamReader("BinaryData.txt")
     Dim FinishLoop As Boolean = True
     Do Until DataReader.EndOfStream Or FinishLoop = False
         ReturnValue = Enqueue(DataReader.ReadLine())
         If ReturnValue = False Then
             FinishLoop = False
         End If
     Loop
     DataReader.Close()
 End Sub
```

Python

```
def ReadData():

    TheFile = open("BinaryData.txt")
    for Line in TheFile:
        ReturnValue = Enqueue(Line.strip())
        if ReturnValue == False:
            break

    TheFile.close()
```

2(e)

1 mark each

- Procedure header (and end where appropriate), storing final string compressed data in global variable
- Call `Dequeue()` **and** storing return value …
- … repeatedly until the return value == "False"
- Comparing new value with previous …
- … maintaining counter of number of occurrences …
- … appending digit and number of occurrences to a string without overriding

Example program code

Java

```
public static void Compress(){
           String First = Dequeue();
           String NewLine = "";
           Integer Count;
           String NextChar;
           while(NumberItems > 0 && First != "False"){
                 Count = 1;
                 NextChar = Dequeue();
                 while(NextChar.equals(First)){
                       Count++;
                       First = NextChar;
                       NextChar = Dequeue();
                 }
                 NewLine = First + Count;
                 NewString = NewString + NewLine;
                 First = NextChar;
           }}
```

2(e)

VB.NET

```
Sub Compress()
     Dim First As String = Dequeue()
     Dim NewLine As String = ""
     Dim Count As Integer
     Dim NextChar As String
     While NumberItems > 0 And First <> "False"
         Count = 1
         NextChar = Dequeue()

         While NextChar = First
             Count += 1
             First = NextChar
             NextChar = Dequeue()
         End While

         NewLine = First & Count
         NewString = NewString & NewLine
         First = NextChar
     End While
```

Python

```
def Compress():
    global NewString
    First = Dequeue()
    NewString = ""
    while NumberItems > 0 and First != "False":
        Count = 1
        NextChar = Dequeue()
        while NextChar == First:
            Count += 1
            First = NextChar
            NextChar = Dequeue()
        NewLine = First + str(Count)
        NewString = NewString + NewLine
        First = NextChar
```

2(f)(i)

1 mark each

- Calling `ReadData()` then `Compress()`
- Outputting compressed string

Example program code

Java

```
public static void main(String args[]){
           NewString = "";
           for(int x = 0; x < 100; x++){
                     Queue[x] = "";
                        }
                     QueueHead = -1;
                     QueueTail = -1;
                    NumberItems = 0;

           ReadData();
           Compress();
           System.out.println(NewString);
     }
```

VB.NET

```
Sub Main()
     NewString = ""
     For x = 0 To 99
       Queue(x) = ""
     Next
     QueueHead = -1
     QueueTail = -1
     NumberItems = 0

     ReadData()
     Compress()
     Console.WriteLine(NewString)
     Console.ReadLine()
 End Sub
```

2(f)(i)

Python

```
NewString = ""
Queue = []
for x in range(100):
        Queue.append("")
QueueHead = -1
QueueTail = -1
NumberItems = 0
ReadData()
Compress()
print(NewString)
```

2(f)(ii)

1 mark for screenshot showing output

Example

```
19051306
```

7(a)

Example solution:

```
FUNCTION CustomerOrder(Number, Points : INTEGER) RETURNS
                                                        INTEGER

   DECLARE NewPoints, NumberFree: INTEGER

   NewPoints ← Points + Number
   NumberFree ← 0
   WHILE NewPoints >= 11 AND Number > 0
      NumberFree ← NumberFree + 1
      NewPoints ← NewPoints -11
       Number ← Number - 1
   END WHILE

   OUTPUT "Number of free coffees is: ", NumberFree
   RETURN NewPoints
ENDFUNCTION
```

MP1 Function header **and** ending **and** parameters **and** return type
MP2 Number of coffees ordered added to points

MP3 Correct calculation of one free coffee

MP4 Attempted calculation of multiple free coffees **and** reduced `NewPoints`

MP5 Output number of free drinks with suitable message

MP6 Return of `NewPoints`

7(b)(i)

Example solution:

```
PROCEDURE AddNewCustomers(NumToAdd : INTEGER)

   DECLARE CustomerID : INTEGER
   DECLARE Count : INTEGER
   DECLARE Line : STRING
   DECLARE NewLine : STRING
   OPENFILE "Loyalty.txt" FOR READ

   WHILE NOT EOF("Loyalty.txt")
      READFILE "Loyalty.txt", Line
   ENDWHILE
   CLOSEFILE "Loyalty.txt"

   OPENFILE "Loyalty.txt" FOR APPEND

   CustomerID ← STR_TO_NUM((LEFT(Line, 6))

   FOR Count ← 1 TO NumToAdd
      CustomerID ← CustomerID + 1
      OUTPUT CustomerID
      NewLine ← NUM_TO_STR(CustomerID) & ",0"
      WRITEFILE "Loyalty.txt", NewLine
   NEXT Count

   CLOSEFILE "Loyalty.txt"
ENDPROCEDURE
```

Mark as follows:

MP1 All variables used are declared using the correct type including a string **and** an integer

MP2 Open file in read mode **and** close (before opening in append mode)

MP3 Conditional loop with `EOF("Loyalty.txt")`

MP4 Read `Line` from file // Count number of records in the file

MP5 Open the file in append mode **and** subsequently close

MP6 Extract `CustomerID` from `Line` **and** convert to integer // use count from MP4 to generate last `CustomerID` stored

MP7 A (count controlled) loop for the number of customers to add

MP8 Increment `CustomerID` **and** output the new `CustomerID` in loop

MP9 Create string for new `CustomerID` **and** write to file in loop

Max 8

7(b)(ii)	**MP1** Check if the text file `loyalty.txt` exists / is empty **MP2** If file does not exist it must be created (using write mode) **MP3** If the file has to be created / is empty then set the first `CustomerID` to `100001` **MP4** Write `"100001,0"` to `loyalty.txt` (using write mode) **MP5** For all but the first customer (to be added to the empty file) **use the existing code / module** **Max 4**

4(a)	**MP1** **Open** the file (`Sales.txt`) in **read mode and** subsequently close **MP2** **Read <u>a line</u>** (from the text file) **MP3** Convert the line read (string) to a number **MP4** If the number is greater than 500 **and** then output week number **MP5** Increment week number (must have been Initialised **…**) **MP6** Repeat from step 2 for 52 times // Repeat from step 2 until end of file **Max 5**
4(b)	**MP1** Going through the program **a line / statement at a time** **//** doing a **<u>dry run</u>** // **<u>single-stepping</u>** **(**at run-time with an IDE) // Peer testing/review is carried out **MP2** Creating a **<u>trace table</u>** **MP3** Draw up **test data** // draw up list **of inputs with expected output** **//** boundary, normal, abnormal data is used **MP4** Errors will be indicated when a **variable** / **output** gives an **unexpected value** **MP5** Faults in the logic of the program can be detected // Errors may be indicated by an unexpected path through the program **Max 3**

7(a)(i)	Example solution:
	`PROCEDURE UpdateVisit(BYREF LastVisitDate : DATE,__`
	`BYREF LoyaltyPoints : INTEGER)`
	
	`IF MONTH(TODAY()) = MONTH(LastVisitDate) THEN`
	`LoyaltyPoints ← LoyaltyPoints + 4`
	`ELSE`
	`LoyaltyPoints ← LoyaltyPoints + 1`
	`ENDIF`
	
	`LastVisitDate ← TODAY()`
	
	`ENDPROCEDURE`
	
	MP1 Procedure heading **and** ending **and** two parameters of correct type
	MP2 … both passed `BYREF`
	
	MP3 Use of `TODAY()` function to obtain current date
	MP4 Use `MONTH` function (×2) **and** one parameter is the header parameter **and** 2nd parameter is `TODAY()` / assigned variable
	
	MP5 If visit is same month, increase `LoyaltyPoints` by 4 otherwise increase by 1
	MP6 `LastVisitDate` set to `TODAY()` / 'current date' variable (assigned or unassigned)
	Max 5

7(a)(ii)	`DAYINDEX(LastVisitDate) = DAYINDEX(TODAY())`

7(b)(i)

Example solution:

```
FUNCTION (MondayCheck(CustomerID : STRING) RETURNS
                                                    INTEGER
   DECLARE LoyaltyPoints : INTEGER
   DECLARE LoyaltyPointsString : STRING
   DECLARE Line : STRING

   Line ← FindCustomer(CustomerID)
   LoyaltyPointsString ← MID(Line, 8, LENGTH(Line) -
                                                     16))
   LoyaltyPoints ← STR_TO_NUM(LoyaltyPointsString)

IF DAYINDEX(TODAY()) = 2 THEN
      LoyaltyPoints ← LoyaltyPoints + 10
   ENDIF

   RETURN LoyaltyPoints
ENDFUNCTION
```

MP1 Value is returned from `FindCustomer(CustomerID)` and stored

MP2 Function `MID` used
MP3 Attempt to extract `LoyaltyPoints` from the string returned by `FindCustomer()`
MP4 Correct extraction of `LoyaltyPoints` from the string returned by `FindCustomer()`

MP5 Conversion of `LoyaltyPoints` extracted to an integer

MP6 Check - is today's date a Monday using **`TODAY()`**
MP7 … increase `LoyaltyPoints` by 10
MP8 Return `LoyaltyPoints`

7(b)(i)

Alternative example solution:

Uses a loop to calculate the number of digits in the `LoyaltyPoints` string

```
FUNCTION MondayCheck(CustomerID : STRING) RETURNS INTEGER

   DECLARE LoyaltyPoints : INTEGER
   DECLARE LoyaltyPointsString : STRING
   DECLARE Index : INTEGER
   DECLARE Line : STRING

   Line ← FindCustomer(CustomerID)
   Index ← 9

   WHILE MID(Line, Index, 1) <> ','
     Index ← Index + 1
   ENDWHILE
   // calculate the number of digits in LoyaltyPoints
                                                    string
   LoyaltyPointsString ← MID(Line, 8, Index - 8)
   LoyaltyPoints ← STR_TO_NUM(LoyaltyPointsString)

   IF DAYINDEX(TODAY()) = 2 THEN
     LoyaltyPoints ← LoyaltyPoints + 10
   ENDIF

   RETURN LoyaltyPoints

ENDFUNCTION
```

7(b)(ii)

Example solution:

```
DayInt ← STR_TO_NUM(LEFT(LastVisitDateString, 2))
MonthInt ← STR_TO_NUM(MID(LastVisitDateString, 3, 2))
YearInt ← STR_TO_NUM(RIGHT(LastVisitDateString, 4))

LastVisitDate ← SETDATE(DayInt, MonthInt, YearInt)
```

Mark as follows:

MP1 Use of one substring function
MP2 `DayInt`, `MonthInt` and `YearInt` all correctly extracted

MP3 Use of `STR_TO_NUM` x 3

MP4 `SETDATE()` function used to convert to a date type **and** assigned to `LastVisitDate`

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"
```


3(a)

1 mark each

- `HighScores` created as 2D array, (of strings) with (min) 7 × 3 elements (local to main) …
- … all elements initialised to empty string ("")

e.g.

Python

```
HighScores = [] #String, 7 x 3
HighScores = [['' for x in range(3)] for y in range(7)]
```

VB.NET

```
Dim HighScores(7, 3) As String
For(X = 0 to 7)
    For(Y = 0 to 3)
       HighScores(X, Y) = ""
  Next Y
Next X
```

Java

```
String[][] HighScores = new String[7][3];
for(Int X = 0; X <7; X++){
    for(Int Y = 0; Y < 3; Y++){
       HighScores[X][Y] = "";
  }
}
```

3(b)

1 mark each

- Function header (and end where appropriate) that returns populated array
- Opening text file to read and closing the file in an appropriate place
- Looping through 7 players/to EOF/21 times …
- … reading in each group of 3 data items and storing each in separate element in 2D array for each player
- Exception handling with all file handling within the try, appropriate catch and an output

3(b)

e.g.

Python

```
def ReadData():
    Temp = []
    HighScores = []
    try:
        File = open("HighScoreTable.txt")
        Temp = File.read().split("\n")
        File.close()
    except:
        print("No file found")
    NumberRecords = len(Temp)-1
    Counter = 0

    while Counter < NumberRecords:

        HighScores.append([Temp[Counter], Temp[Counter+1], Temp[Counter+2]])
        Counter = Counter + 3

    return HighScores
```

VB.NET

```
  Function ReadData()
        Dim TextFile As String = "HighScoreTable.txt"

        Dim HighScores(7, 3) As String
        Try
            Dim FileReader As New System.IO.StreamReader(TextFile)
            Dim Counter As Integer = 0
```

3(b)

```
            While Counter < 8
                HighScores(Counter, 0) = FileReader.ReadLine()
                HighScores(Counter, 1) = FileReader.ReadLine()
                HighScores(Counter, 2) = FileReader.ReadLine()

                Counter = Counter + 1


            End While
            FileReader.Close()
        Catch ex As Exception
            Console.WriteLine("No file found")
        End Try
        Return HighScores
  End Function
```

Java

```
public static String[][] ReadData(){
      String TextFile = "HighScoreTable.txt";
      String[][] HighScores = new String[7][3];

      try{
            FileReader f = new FileReader(TextFile);
            BufferedReader Reader = new BufferedReader(f);
            for(Integer X = 0; X < 7; X++){
                  try{
```

3(b)

```
                    HighScores[X][0] = Reader.readLine();
                    HighScores[X][1] = Reader.readLine();
                    HighScores[X][2] = Reader.readLine();
                }catch(IOException ex){}
            }
            try{
                Reader.close();
            }catch(IOException ex){}
            return HighScores;
        }catch(FileNotFoundException e){
            System.out.println("File not found");
        }
        return HighScores;
}
```

3(c)

1 mark each

- Procedure (header and end) taking (min) 1 parameter (2D array), looping through each of the first dimension in array …
- … outputting all data in correct format

e.g.
Python

```
def OutputHighScores(HighScores):
    for x in range(0, len(HighScores)):
        print(HighScores[x][0], "reached level", HighScores[x][1], "with a score of",
HighScores[x][2])
```

VB.NET

```
  Sub OutputHighScores(HighScores(,))
        For x = 0 To 6
            Console.WriteLine(HighScores(x, 0) & " reached level " & HighScores(x, 1) & "
with a score of " & HighScores(x, 2))
        Next
  End Sub
```

3(c)

Java

```
public static void OutputHighScores(String[][] HighScores){
       for(Integer x = 0; x < 7; x++){
           System.out.println(HighScores[x][0] + " reached level " + HighScores[x][1] + "
with a score of " + HighScores[x][2]);
       }
}
```

3(d)

1 mark each

- Function header (and end taking array as parameter) returning sorted array.
- Comparing the levels and swapping **all** dimensions when in incorrect order
- Comparing scores when levels are the same and swapping **all** dimensions when in incorrect order
- Correct loops and comparisons to put data in correct order

e.g.
Python

```
def SortScores(HighScores):
    Counter = 0
    ArrayLength = len(HighScores)

    for x in range(ArrayLength-1):
        for y in range(0, ArrayLength-x-1):
            if int(HighScores[y][1]) < int(HighScores[y + 1][1]):
                HighScores[y], HighScores[y + 1] = HighScores[y + 1], HighScores[y]
            elif int(HighScores[y][1]) == int(HighScores[y+1][1]):
                if int(HighScores[y][2]) < int(HighScores[y+1][2]):
                    HighScores[y], HighScores[y + 1] = HighScores[y + 1], HighScores[y]
    return HighScores
```

VB.NET

```
  Function SortScores(HighScores)
        Dim ArrayLength As Integer = 6
        Dim Temp1 As String
```

3(d)

```
        Dim Temp2 As String
        Dim Temp3 As String
        For x = 0 To ArrayLength - 1
            For y = 0 To ArrayLength - x - 1
                If Integer.Parse(HighScores(y, 1)) < Integer.Parse(HighScores(y + 1, 1))
Then
                    Temp1 = HighScores(y, 0)
                    Temp2 = HighScores(y, 1)
                    Temp3 = HighScores(y, 2)
                    HighScores(y, 0) = HighScores(y + 1, 0)
                    HighScores(y, 1) = HighScores(y + 1, 1)
                    HighScores(y, 2) = HighScores(y + 1, 2)
                    HighScores(y + 1, 0) = Temp1
                    HighScores(y + 1, 1) = Temp2
                    HighScores(y + 1, 2) = Temp3
                ElseIf Integer.Parse(HighScores(y, 1)) = Integer.Parse(HighScores(y + 1,
1)) Then
                    If Int(HighScores(y, 2)) < Int(HighScores(y + 1, 2)) Then
                        Temp1 = HighScores(y, 0)
                        Temp2 = HighScores(y, 1)
                        Temp3 = HighScores(y, 2)
                        HighScores(y, 0) = HighScores(y + 1, 0)
                        HighScores(y, 1) = HighScores(y + 1, 1)
                        HighScores(y, 2) = HighScores(y + 1, 2)
                        HighScores(y + 1, 0) = Temp1
                        HighScores(y + 1, 1) = Temp2
                        HighScores(y + 1, 2) = Temp3
                    End If
                End If
            Next y
        Next x
        Return HighScores
    End Function
```

Java

```
public static String[][] SortScores(String[][] HighScores){
```

3(d)

```
      Integer ArrayLength = 6;
      String Temp1;
      String Temp2;
      String Temp3;

      for(Integer x = 0; x < ArrayLength; x++){
            for(Integer y = 0; y < ArrayLength - x; y++){
                  if(Integer.parseInt(HighScores[y][1]) <
Integer.parseInt(HighScores[y+1][1])){
                        Temp1 = HighScores[y][0];
                        Temp2 = HighScores[y][1];
                        Temp3 = HighScores[y][2];
                        HighScores[y][0] = HighScores[y+1][0];
                        HighScores[y][1] = HighScores[y+1][1];
                        HighScores[y][2] = HighScores[y+1][2];
                        HighScores[y+1][0] = Temp1;
                        HighScores[y+1][1] = Temp2;
                        HighScores[y+1][2] = Temp3;
                  }else if(Integer.parseInt(HighScores[y][1]) ==
Integer.parseInt(HighScores[y+1][1])){
                        if(Integer.parseInt(HighScores[y][2]) <
Integer.parseInt(HighScores[y+1][2])){
                              Temp1 = HighScores[y][0];
                              Temp2 = HighScores[y][1];
                              Temp3 = HighScores[y][2];
                              HighScores[y][0] = HighScores[y+1][0];
                              HighScores[y][1] = HighScores[y+1][1];
                              HighScores[y][2] = HighScores[y+1][2];
                              HighScores[y+1][0] = Temp1;
                              HighScores[y+1][1] = Temp2;
                              HighScores[y+1][2] = Temp3;
                        }
                  }
```

3(d)

```
            }
      }
      return HighScores;
}
```

3(e)(i)

1 mark each

- Code statements in order:

```
HighScores = ReadData()
HighScores = SortScores(HighScores) // HighScores = SortScores()
```

- Code statements in order:

```
OUTPUT "Before"
OutputHighScores(HighScores)
OUTPUT "After"
OutputHighScores(HighScores)
```

unsorted

sorted

e.g.

Python

```
HighScores = []
HighScores = ReadData()
print("Before")
OutputHighScores(HighScores)
HighScores = SortScores(HighScores)
print("After")
OutputHighScores(HighScores)
```

VB.NET

```
  Sub Main(args As String())
        Dim HighScores(7, 3) As String
        HighScores = ReadData()
        Console.WriteLine("Before")
        OutputHighScores(HighScores)
        HighScores = SortScores(HighScores)
        Console.WriteLine("After")
        OutputHighScores(HighScores)
  End Sub
```

3(e)(i)

Java

```
public static void main(String args[]){
      String[][] HighScores = new String[7][3];
      HighScores = ReadData();
      System.out.println("Before");
      OutputHighScores(HighScores);
      HighScores = SortScores(HighScores);
      System.out.println("After");
      OutputHighScores(HighScores);
}
```

3(e)(ii)

Output showing ‘Before’ and players and scores in correct format before sorting
Output showing ‘After’ players and scores in correct order and format after sorting

e.g.

```
Before
GHEH got to level 3 with a score of 10
KWQW got to level 4 with a score of 20
MMND got to level 4 with a score of 18
RFOO got to level 5 with a score of 20
XXHD got to level 3 with a score of 19
QWSD got to level 3 with a score of 15
JGHF got to level 5 with a score of 22
After
JGHF got to level 5 with a score of 22
RFOO got to level 5 with a score of 20
KWQW got to level 4 with a score of 20
MMND got to level 4 with a score of 18
XXHD got to level 3 with a score of 19
QWSD got to level 3 with a score of 15
GHEH got to level 3 with a score of 10
```