8(a)

Example solution:

```
PROCEDURE OKToBorrow(ThisStudentID : STRING)
   DECLARE Index, Count : INTEGER

   CONSTANT Max = 5
   Count ← 0
   Index ← 1 / 0

   WHILE Index <= 5000 / Index <= 4999 AND Count < Max
      IF Loan[Index].StudentID = ThisStudentID AND __
                                 Loan[Index].OnLoan =
TRUE THEN
         Count ← Count + 1
      ENDIF
      Index ← Index + 1
   ENDWHILE

   IF Count < Max THEN
      OUTPUT "Student may borrow another book"
   ELSE
      OUTPUT "Student may not borrow another book"
   ENDIF

ENDPROCEDURE
```

Mark as follows:
1 Procedure heading, parameter and ending
2 Loop through **all elements** in `Loan` array
3 ... terminating if `Max` reached

4 Test for correct `StudentID` field **in a loop**
5 ... and that `OnLoan = TRUE` **in a loop**
... if so, then increment `Count` **in a loop**
6 Output an appropriate **message** in both cases **after the loop**

8(b)

Example solution:

```
FUNCTION ReturnBook(ThisStudentID, ThisBookID : STRING)
                                                RETURNS BOOLEAN
   DECLARE Index : INTEGER

   Index ← 1

   WHILE Index < 5001
      IF Loan[Index].StudentID = ThisStudentID  AND __
         Loan[Index].BookID = ThisBookID THEN
         IF Loan[Index].OnLoan = TRUE THEN // Not
                                                  returned
            Loan[Index].OnLoan  ← FALSE // Mark as
                                                returned now
            RETURN TRUE  //Found and not already returned
         ELSE
            RETURN FALSE  // Found and already returned
         ENDIF
      ENDIF
      Index ← Index + 1
   ENDWHILE

   RETURN False // loan not found
ENDFUNCTION
```

Mark as follows:
1 Loop through **all elements** in `Loan` array
2 ...and book loan not found
3 Attempt to reference an individual data item **in a loop**
4 Correct test for `StudentID` **and** correct `BookID` **in a loop**
5 **...** If book not returned yet, then set `OnLoan` field to `FALSE`
6 RETURN TRUE if book loan Found and not already returned
7 RETURN FALSE if book loan Found and loan already returned
8 RETURN FALSE if book loan not found

Alterantive example solution – not immediate return

```
FUNCTION ReturnBook(ThisStudentID, ThisBookID : STRING)
                                                RETURNS BOOLEAN
   DECLARE Index : INTEGER
   DECLARE Found, OK : BOOLEAN

   Index ← 1
   Found ← FALSE
   OK ← TRUE
```

8(b)

```
   WHILE Index < 5001 AND NOT Found
      IF Loan[Index].StudentID = ThisStudentID  AND __
         Loan[Index].BookID = ThisBookID THEN
         Found ← TRUE
         IF Loan[Index].OnLoan = FALSE THEN // Already
                                                returned
            OK ← FALSE
         ELSE
            Loan[Index].OnLoan  ← FALSE // Mark as
                                          returned now
         ENDIF
      ENDIF
      Index ← Index + 1
   ENDWHILE

   RETURN Found AND OK
ENDFUNCTION
```

8(c)

Award marks for reference to following design features:

Change to existing Record:
1 Store date due back as a new data item / in the loan record
2 Store date of loan **and** the loan length category as new data items / in the loan record
Max 1

New Record
3 **New** Records with Category (and Loan length field)

Change to Program:
4 (Use the loan length category of a book to) calculate the date due date
5 Compare due date with today's date (to check if overdue / to calculate fine)
6 Map loan length category to length of loan / date due back
Max 1

Max 2 marks

3(a)

1 mark each
- (global) `TreeArray` declared as a 2D array with 50 × 3 elements …
- … all initialised to `-1`
- (global) `RootPointer` initialised to `-1` **and** `FreeNode` initialised to 0

Example program code

Java
```
public static Integer FreeNode;
public static Integer RootPointer;
public static Integer[][] TreeArray = new Integer[50][3];
public static void main(String args[]){
     for(Integer X = 0; X < 50; X++){
          TreeArray[X][0] = -1;
          TreeArray[X][1] = -1;
          TreeArray[X][2] = -1;
     }
     RootPointer = -1;
     FreeNode = 0;
}
```

VB.NET
```
Dim FreeNode As Integer
Dim TreeArray(0 To 49, 0 To 2) As Integer
Dim RootPointer As Integer
Sub Main(args As String())
    For X = 0 To 49
        TreeArray(X, 0) = -1
        TreeArray(X, 1) = -1
        TreeArray(X, 2) = -1
    Next
    RootPointer = -1
    FreeNode = 0
End Sub
```

3(a)

Python
```
TreeArray = []
for x in range(50):
    TreeArray.append([-1,-1,-1])
RootPointer = -1
FreeNode = 0
```

3(b)

1 mark each to max 7

- Procedure header (and end) taking one (integer) parameter **and** storing parameter in array in index `TreeArray[FreeNode][1]`
- Checking if tree is full (`FreeNode = 50`) **and** outputting "`The tree is full`"
- Checking if tree is empty (`RootPointer = -1 // FreeNode = 0`) **and** if so, storing 0 in `RootPointer`

- (if not empty) Comparing parameter to data at index `TreeArray[RootPointer][1]` …
- … if less than, accessing left node …
- … if greater than, accessing right node …
- … until location found …
- … updating parent node's appropriate pointer

- Incrementing `FreeNode`

Example program code

Java

```
public static void AddNode(Integer NodeData){
      Boolean Placed;
      Integer CurrentNode;
      if(FreeNode <= 49){
            TreeArray[FreeNode][0] = -1;
            TreeArray[FreeNode][1] = NodeData;
            TreeArray[FreeNode][2] = -1;
            if(RootPointer == -1){
                  RootPointer = 0;
            }else{
                  Placed = false;
                  CurrentNode = RootPointer;
                  while(Placed == false){
                        if(NodeData < TreeArray[CurrentNode][1]){
                              if(TreeArray[CurrentNode][0] == -1){
                                    TreeArray[CurrentNode][0] = FreeNode;
                                    Placed = true;
                              }else{
```


3(b)

```
                                    CurrentNode = TreeArray[CurrentNode][0];
                              }
                        }else{
                              if(TreeArray[CurrentNode][2] == -1){
                                    TreeArray[CurrentNode][2] = FreeNode;
                                    Placed = true;
                              }else{
                                    CurrentNode = TreeArray[CurrentNode][2];
                              }
                        }
                  }
            }
            FreeNode++;
      }else{
            System.out.println("The tree is full");
      }
}
```

VB.NET

```
Sub AddNode(NodeData)
    Dim Placed As Boolean
    Dim CurrentNode As Integer
    If FreeNode <= 49 Then
        TreeArray(FreeNode, 0) = -1
        TreeArray(FreeNode, 1) = NodeData
        TreeArray(FreeNode, 2) = -1

        If RootPointer = -1 Then
            RootPointer = 0
        Else
            Placed = False
            CurrentNode = RootPointer
        While Placed = False
            If NodeData < TreeArray(CurrentNode, 1) Then
                    If TreeArray(CurrentNode, 0) = -1 Then
                        TreeArray(CurrentNode, 0) = FreeNode
```

3(b)

```
                        Placed = True
                    Else
                        CurrentNode = TreeArray(CurrentNode, 0)
                    End If
                Else
                    If TreeArray(CurrentNode, 2) = -1 Then
                        TreeArray(CurrentNode, 2) = FreeNode
                        Placed = True
                    Else
                        CurrentNode = TreeArray(CurrentNode, 2)
                    End If
                End If
            End While
        End If
        FreeNode = FreeNode + 1
    Else
        Console.WriteLine("The tree is full")
    End If
End Sub
```

Python

```
def AddNode(NodeData):
    global FreeNode
    global TreeArray
    global RootPointer
    if FreeNode <= 49:
        TreeArray[FreeNode][0] = -1
        TreeArray[FreeNode][1] = NodeData
        TreeArray[FreeNode][2] = -1
        if RootPointer == -1:
            RootPointer = 0
        else:
            Placed = False
            CurrentNode = RootPointer
            while Placed == False:
                if NodeData < TreeArray[CurrentNode][1]:
```


3(b)

```
                    if TreeArray[CurrentNode][0] == -1:
                        TreeArray[CurrentNode][0] = FreeNode
                        Placed = True
                    else:
                        CurrentNode = TreeArray[CurrentNode][0]
                else:
                    if TreeArray[CurrentNode][2] == -1:
                        TreeArray[CurrentNode][2] = FreeNode
                        Placed = True
                    else:
                        CurrentNode = TreeArray[CurrentNode][2]
        FreeNode = FreeNode + 1
    else:
        print("The tree is full")
```

3(c)

1 mark each to max 4

- Opening the file to read **and** closing the file in an appropriate place
- Looping 50 times/through file/through each line/until EOF …
- … reading in **each line** …
- … calling `AddNode()` with each value read in
- Exception handling try, catch, except with appropriate message

Example program code

Java

```
Integer Line;
String ReadData;
try{
      FileReader f = new FileReader("TreeData.txt");
      try{
            BufferedReader Reader = new BufferedReader(f);
            ReadData = Reader.readLine();
            while (ReadData != null){
                  Line = Integer.parseInt(ReadData);
                  AddNode(Line);
                  ReadData = Reader.readLine();
            }
            Reader.close();
      }catch(IOException ex){
      }
}catch(FileNotFoundException e){
      System.out.println("File not found");
}
```


3(c)

VB.NET

```
Try
    Dim FileReader As New System.IO.StreamReader("TreeData.txt")
    While Not FileReader.EndOfStream
        AddNode(FileReader.ReadLine())
    End While
    FileReader.Close()
Catch ex As Exception
    Console.WriteLine("Cannot open file")
End Try
```

Python

```
try:
    File= open("TreeData.txt")
    for Line in File:
        AddNode(int(Line.strip()))
    File.close()
except:
    print("Error cannot open file")
```

3(d)

1 mark each

- Procedure header (and end) **and** with exception handling for writing to file: try, catch, except with appropriate message
- Opening the file (`Tree.txt`) to write **and** closing the file in appropriate place
- Looping through each element in array …
- … creating correct string
- … writing each string to the file

Example program code

Java

```
public static void WriteAllToFile(){
      File TheFile = new File("Tree.txt");
      String Line;
      try{
            FileWriter FW = new FileWriter(TheFile, true);
            for(Integer X = 0; X < 50; X++){
                  Line = TreeArray[X][0] + "," + TreeArray[X][1] + "," + TreeArray[X][2];
                  FW.write(Line);
                  FW.write("\n");
            }
            FW.close();
      }catch(IOException ex){
            System.out.println("Cannot open file");
      }
}
```


3(d)

VB.NET

```
Sub WriteAllToFile()
    Dim FileWriter As IO.StreamWriter = New IO.StreamWriter("Tree.txt", False)
    Dim Line As String
    Try
        For x = 0 To 49
            Line = TreeArray(x, 0) & "," & TreeArray(x, 1) & "," & TreeArray(x, 2)
            FileWriter.WriteLine(Line)
        Next
        FileWriter.Close()
    Catch ex As Exception
        Console.WriteLine("Cannot open or write to file")
    End Try
End Sub
```

Python

```
def WriteAllToFile():
    try:
        File = open("Tree.txt","a+")
        for x in range(0, 50):
            Line = str(TreeArray[x][0]) + "," + str(TreeArray[x][1])+ "," +
str(TreeArray[x][2]) + "\n"
            File.write(Line)
        File.close()
    except:
        print("Cannot write to file")
```

3(e)(i)	1 mark for calling `WriteAllToFile()`
	
	Example program code
	
	Java
	`WriteAllToFile();`
	
	VB.NET
	`WriteAllToFile()`
	
	Python
	`WriteAllToFile()`

3(e)(ii)	1 mark for a screenshot that shows correct data stored, each node on a new line (in correct format). The screenshot must include the filename. e.g.