4(a)

Example solution - conditional:

```
PROCEDURE Store()
   DECLARE ThisNum, LastNum : REAL
   DECLARE Index : INTEGER // index to global array

   INPUT ThisNum
   LastNum ← ThisNum
   Number[1] ← ThisNum

   INPUT ThisNum
   Index ← 2

   WHILE Index < 21 AND ThisNum > LastNum
      Number[Index] ← ThisNum
      LastNum ← ThisNum
      Index ← Index + 1
      IF Index < 21 THEN
         INPUT ThisNum
      ENDIF
   ENDWHILE

   OUTPUT Index - 1, " values were stored in the array"
ENDPROCEDURE
```

Alternative loop:

```
   // Loop without using LastNum
   WHILE Index < 21 AND ThisNumNum > Number[Index - 1]
      Number[Index] ← ThisNum
      Index ← Index + 1
      IF Index < 21 THEN      //skip input if array full
         INPUT ThisNum
      ENDIF
   ENDWHILE
```

Mark as follows:

1 Declare local variables `Index` as integer, `ThisNum` and `LastNum` as real
2 Input value and assign to first element
3 **Attempt** at conditional loop including **both** tests
4 **Completely correct** conditional loop including **both** tests
5 Assign input value to current element if greater than previous element **and** input next value **in a loop**
6 Final output giving attempted count of elements stored plus message **after a loop**

4(a)

Alternative FOR loop example solution:

```
PROCEDURE Store()
   DECLARE ThisNum, LastNum : REAL
   DECLARE Index : INTEGER // index to global array
   DECLARE Count : INTEGER
   INPUT ThisNum
   LastNum ← ThisNum
   Number[1] ← ThisNum
   Count ← 1
   INPUT ThisNum
   FOR Index ← 2 TO 20
      IF ThisNum > LastNum THEN
         Number[Index] ← ThisNum
         LastNum ← ThisNum
         Count ← Count + 1
         IF Index < 20 THEN
            INPUT ThisNum
         ENDIF
      ELSE
         BREAK
      ENDIF
   NEXT Index
   OUTPUT Count, " values were stored in the array"
ENDPROCEDURE
```

Alternative loop:

```
   Count ← 1
   INPUT ThisNum
   FOR Index ← 2 TO 20
      IF ThisNum > Number[Index - 1] THEN
         Number[Index] ← ThisNum
         Count ← Count + 1
         IF Index < 20 THEN
            INPUT ThisNum
         ENDIF
      ELSE
         BREAK
      ENDIF
   NEXT Index
```

Mark as follows:

1 Declare local variables `Index` as integer, `ThisNum` and `LastNum` as real
2 Input value and assign to first element
3 Count-controlled loop
4 Count-controlled loop **with BREAK** if current value greater than previous value
5 ... Assign input value to current element **following correct comparison in a loop**
6 Final output giving count of elements stored plus message following a reasonable attempt **after loop**

4(b)(i)	One mark per point: 1 The amount of values **stored** (in a text file) is **not fixed** / **not limited** (other than by secondary storage available) // The amount values **stored** is **not limited** to the size of the array 2 The data is stored **permanently** / **non-volatile** // Data can be restored / reused without **re-entering** values
4(b)(ii)	Each (real) value would have to be **converted** to a string

5(a)(i)	It is a value that is **invalid as an** (array) **index** // not in the **range** of valid (index) values
5(a)(ii)	To test whether the **current element / value at index** matches the **given / search** string / data / **parameter**
5(a)(iii)	The value of `FoundAt` is only updated to the (current) index if it currently stores the initial value / -1 // `FoundAt` is not updated when it currently stores any other value than -1
5(b)(i)	The loop continues after the (first instance) of the **search** value / string / **matching element** has been found Comparisons/tests continue to be made even if the **search** value **/ string / matching element** has been found
5(b)(ii)	One mark per point Construct: A conditional loop Justification: The loop / search can be terminated as soon as the (first occurrence of) **`SearchString`** / **required** value / **required** string / **matching** element is found

4(a)

Example solution:

```
DECLARE CheckTotal : ARRAY [0:35] OF BOOLEAN
DECLARE Index : INTEGER

FOR Index ← 0 TO 35
   CheckTotal[Index] ← FALSE
NEXT Index
```

Mark as follows:

MP1 Declaration of `CheckTotal` array
MP2 Declaration of a loop counter
MP3 Loop – with correct syntax and number of iterations
MP4 Assignment of array element to `FALSE`

4(b)

```
DECLARE EasyQ, HardQ : INTEGER

FOR EasyQ ← 0 TO 15 STEP 3
          MP1          MP2

   FOR HardQ ← 0 TO 20 STEP 5
                MP3     MP4

      CheckTotal[HardQ + EasyQ] ← TRUE
                        MP5
   NEXT HardQ

NEXT EasyQ
```

Mark as follows:
One mark per gap (boldened)

4(c)

MP1 Check that the parameter value is in the range 0 to 35
MP2 ... and if not, return `FALSE`
MP3 Use the **parameter** value as **the index to array** `CheckTotal`
MP4 Return the value from given index position

4

Example solution:

```
PROCEDURE Store()
   DECLARE Index, StartPos, ThisNum : INTEGER
   DECLARE ThisString : STRING

   Index ← 1
   StartPos ← 1

   WHILE StartPos < LENGTH(NumString) AND Index < 21
      ThisString ← MID(NumString, StartPos, 3)
      ThisNum ← STR_TO_NUM(ThisString)
      Data[Index] ← ThisNum
      Index ← Index + 1
      StartPos ← StartPos + 4
   ENDWHILE

ENDPROCEDURE
```

Mark as follows:

1 Declare all local variables used
2 Conditional loop while end of string not reached
3 ... and array not full
4 Attempted use of `MID()` // Attempted use of `Right()` and `Left()`
5 Correct extraction and use next three letter substring from `NumString` **in a loop**
6 Convert to integer **and** assign value to array element **in a loop**
7 Correct increment of array index by 1 **and** start position of substring by 4

Max 6

Alternative FOR loop solution:

```
PROCEDURE Store()
   DECLARE Index, StartPos, ThisNum : INTEGER
   DECLARE ThisString : STRING
   StartPos ← 1
   FOR Index ← 1 TO 20
      ThisString ← MID(NumString, StartPos, 3)
      ThisNum ← STR_TO_NUM(ThisString)
      Data[Index] ← ThisNum
      StartPos ← StartPos + 4
      IF StartPos > LENGTH(NumString) THEN
         Break
      ENDIF
   NEXT Index
ENDPROCEDURE
```

4	Alternative where number of 3-digit substrings calculated
	
	`StartPos ← 1`
	`NumOfSubStrings ← ((LENGTH(NumString) + 1)DIV 4))`
	
	`FOR Index ← 1 TO NumOfSubString`
	`  ThisString ← MID(NumString, StartPos, 3)`
	`  ThisNum ← STR_TO_NUM(ThisString)`
	`  Data[Index] ← ThisNum`
	`  StartPos ← StartPos + 4`
	`  IF Index = 20 THEN`
	`  Break`
	`  ENDIF`
	`NEXT Index`
	
	Mark as follows:
	1 Declare all variables used
	2 Loop for 20 iterations // number of 3-digit numbers in the `NumString`
	3 Attempted use of `MID()` // Attempted use of `Right()` and `Left()`
	4 Correct extraction and use next three letter substring from `NumString` **in a loop**
	5 Convert to integer **and** assign value to array element **in a loop**
	6 Increment `StartPos` by 4 **in a loop**
	7 Test for end of `NumString` **and** `Break` // Test for end of array **and** Break
	
	Max 6

4	Alternative Solution extracting one character at a time **and** testing for comma
	
	`PROCEDURE Store()`
	`  DECLARE Index, Position : INTEGER`
	`  DECLARE SubString : STRING`
	`  Decalre ThisCharacter : CHAR`
	`  Index ← 1`
	`  Position ← 0`
	`  SubString ← ""`
	`  WHILE Index <= 20 AND Position <= LENGTH(NumString)`
	`  Charcter ← MID(NumString, Position, 1)`
	`  IF Character = ','`
	`  Data[Index] ← STR_TO_NUM(SubString)`
	`  Index ← Index + 1`
	`  SubString ← ""`
	`  ELSE`
	`  SubString ← SubString & Character`
	`  ENDIF`
	`  Position ← Position + 1`
	`  ENDWHILE`
	`  Data[Index] ← STR_TO_NUM(SubString) //last 3 digit string`
	
	`ENDPROCEDURE`
	
	Mark as follows:
	1 Declare all variables used
	2 Conditional loop while end of string not reached
	3 ... and array not full
	4 Correct extraction and use of a character from `NumString` **in a loop**
	5 Test for comma **in a loop**
	6 If true convert substring to number … **and** store in `Data` array **and** set `SubString` to `""` **and** increment `Index`
	7 Otherwise concatenate next character from `NumString` to end of Substring
	
	Max 6

5(a)

Index	CaseVar	Count	Num[1]	Num[2]	Num[3]	Num[4]
1		0	1	2	5	3
	1					
2		1	2			
3						
	5					
		20				
4						
	3					
7		19				6
4						
	6					
		20				
1						
	2					
3		22	3			
4						

Zone 1

Zone 2 Zone 3 Zone 4 Zone 5

Zones 1 to 4 : One mark for each set of values as shown
Zone 5: Num[4] set to 6 when Index is 7
and
Num[1] set to 3 when index is 3 for **second** time

5(b)(i)

1 and 2

5(b)(ii)

```
1 TO 2 : Num[Index] ← Num[Index] + Index
         Index ← Index + CaseVar
         Count ← Count + CaseVar
```

One mark per point:
- use of variable `CaseVar` rather than literal values
- completely correct clause

6(a)

One mark per point:

1 Fewer lines of code are needed
2 The program is simpler/ less complex // Program is easier to design / code / maintain / modify / test / debug
3 Direct access to days in a month / data (using month number as index) // Can use index / month to (directly) access days in month / data

Max 2 marks

6(b)

Example Solution

```
FUNCTION GetDate(ProductionDate : DATE, ShelfLife :
                                        INTEGER) RETURNS DATE
   DECLARE NewDate : DATE
   DECLARE DD, MM, YY, LastDay : INTEGER

   DD ← DAY(ProductionDate)
   MM ← MONTH(ProductionDate)
   YY ← YEAR(ProductionDate)

   DD ← DD + ShelfLife
   LastDay ← DaysInMonth[MM]

   IF DD > LastDay THEN
      MM ← MM + 1
      DD ← DD - LastDay
      IF MM = 13 THEN
         MM ← 1
         YY ← YY + 1
      ENDIF
   ENDIF
   NewDate ← SETDATE(DD, MM, YY)
   RETURN NewDate

ENDFUNCTION
```

Mark as follows:

1 Declare three variables of `TYPE INTEGER` to store `DD`, `MM` and `YY`
2 Correct use of date functions from Insert to extract three integer values representing DD, MM and YY and use/assign to a variable
3 Add parameter `ShelfLife` to `DD` and use / assign result to a variable
4 Extract `LastDay` from `DaysInMonth` using `MM` as `Index`
5 Test for new `DD` after end of month / `DaysInMonth[MM]`
… and if `DD` > `DaysInMonth[MM]`:
6 `MM` is incremented by 1 and `DD` is set to `DD - DaysInMonth[MM]`
7 If `MM = 13 / MM > 12` then increment `YY` **and** set `MM` to 1
8 Use of `SETDATE` to assign value to `NewDate`, `NewDate` must have been delared as a type `DATE`
9 Return date

Max 7 marks

8(a)

Loop example solution

```
FUNCTION CheckMark(Mark : INTEGER) RETURNS BOOLEAN
   DECLARE Index, Lower, Upper : INTEGER

   FOR Index ← 1 TO 5
      Lower ← GradeBoundary[Index] - 2
      Upper ← GradeBoundary[Index] + 2
      IF Mark >= Lower AND Mark <= Upper THEN
         RETURN TRUE
      ENDIF
   NEXT Index

   RETURN FALSE
ENDFUNCTION
```

Mark as follows:
1 Loop through all elements in `GradeBoundary` array
2 Attempt to calculate range, both `Lower` and `Upper`, **in a loop**
3 Completely correct range calculation **in a loop**
4 Test if given mark / parameter, is within range **in a loop**
5 Set variable / immediate RETURN if recheck required **in a loop**
6 Return Boolean in both cases following a reasonable attempt

Selection example solution

```
FUNCTION CheckMark(Mark : INTEGER) RETURNS BOOLEAN

   CASE OF Mark
      GradeBoundary[1] - 2 TO GradeBoundary[1] + 2 :
                                                RETURN TRUE
      GradeBoundary[2] - 2 TO GradeBoundary[2] + 2 :
                                                RETURN TRUE
      GradeBoundary[3] - 2 TO GradeBoundary[3] + 2 :
                                                RETURN TRUE
      GradeBoundary[4] - 2 TO GradeBoundary[4] + 2 :
                                                RETURN TRUE
      GradeBoundary[5] - 2 TO GradeBoundary[5] + 2 :
                                                RETURN TRUE
      OTHERWISE : RETURN FALSE
   ENDCASE

ENDFUNCTION
```

Mark as follows:
1 Correct syntax used for selection structure(s)
2 Correct checks for two ranges
3 Correct checks for three ranges
4 Correct checks for four ranges
5 Correct checks for all five ranges
6 Return Boolean in both cases following a reasonable attempt

8(b)	Example Solution:
	
	`PROCEDURE CheckAll(CNum : INTEGER)`
	`  DECLARE Index, Count, ThisMark: INTEGER`
	
	`  Count ← 0`
	`  OPENFILE "GRList.txt" FOR WRITE`
	`  FOR Index ← 1 to CNum`
	`  ThisMark ← Result[Index, 1] // 2D array: mark + ID`
	`  IF CheckMark(ThisMark) = TRUE THEN`
	`  WRITE "GRList.txt", NUM_TO_STR(Result[Index, 2])`
	`  Count ← Count + 1`
	`  ENDIF`
	`  NEXT Index`
	`  CLOSEFILE "GRList.txt"`
	`  OUTPUT "There are ", Count, " papers to check"`
	
	`ENDPROCEDURE`
	
	Mark as follows:
	1 Procedure heading, parameter and ending.
	2 Open file in write mode and subsequently close
	3 Loop through all `CNum` candidates
	4 Extract candidate mark from `Result` array **in a loop**
	5 Use of `CheckMark()` with candidate mark as parameter **in a loop**
	6 Test return value and if `TRUE` then increment `Count` **in a loop**
	7 ... write ID to file ...
	8 ... after conversion to a string **in a loop**
	9 Final output of message giving number of papers to check **not in a loop**
	
	Max 8 marks

8(c)	The file will need to be opened in <u>APPEND</u> mode

2(a)

Example 'loop solution':

```
FUNCTION Conceal(CardNumber : STRING) RETURNS STRING
   DECLARE MaskedString : STRING
   DECLARE Count : INTEGER
   CONSTANT Asterisk = '*'

   MaskedString ← RIGHT(CardNumber, 4)
   FOR Count ← 1 TO LENGTH(CardNumber) - 4
       MaskedString ← Asterisk & MaskedString
   NEXT Count

   RETURN MaskedString

ENDFUNCTION
```

Mark as follows:
MP1 Function heading **and** parameter **and** ending **and** return type
MP2 Declaration of all local variables used - including the loop counter
MP3 Calculate number of digits to mask / number of asterisks required

MP4 use of a 'relevant' Loop
MP5 Correct number of iterations
MP6 Concatenate asterisk to start/end of `MaskedString` **in a loop**

MP7 Assign last four characters to `MaskedString //` Concatenate the retained original last four digits
MP8 Return masked string

Max 6 marks

ALTERNATIVE 'non loop' solution:

```
FUNCTION Conceal(CardNumber : STRING) RETURNS STRING
   DECLARE MaskedString : STRING
   DECLARE Count : INTEGER
   CONSTANT Asterisks = "********************" //20
                                                asterisks

   Count ← LENGTH(CardNumber) - 4
   MaskedString ← LEFT(Asterisks, Count) &
                                       RIGHT(CardNumber, 4)

   RETURN MaskedString

ENDFUNCTION
```

Mark as follows:
MP1 Function heading **and** parameter **and** ending **and** return type
MP2 Declaration of all local variables used
MP3 Calculate number of digits to mask / number of asterisks required

2(a)	**MP4** Trim asterisk string to number calculated in **MP3** **MP5** Extract the last four characters **MP6** Concatenate trimmed asterisk string with last four characters of `CardNumber` **MP7** Return masked string **Max 6 marks**
2(b)(i)	`DECLARE CardNumber : ARRAY [1:100, 1:2] OF STRING` **MP1** Correct dimensions **MP2** All other parts of the statement correct
2(b)(ii)	Any reference to BYREF // 'by reference'

1(a)

1 mark each to max 6:

- Function declaration (and close where appropriate)
- Declaration/use of an array (with space/initialised with 45 spaces/strings)
- Opening the file Data.txt for read and closing in an appropriate place
- Looping through all file contents/Looping 45 times and reading each line …
- … storing all items from file into array
- Returning the populated array
- Exception handling with suitable try, catch and output

e.g.

Python

```
def ReadData():
    Colours = []
    try:
        File = open("Data.txt")
        Colours = File.read().split("\n")
        File.close()
        return Colours
    except:
        print("No file found")
```

VB.NET

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

        Dim Colours(45) As String
        Try
            Dim FileReader As New System.IO.StreamReader(TextFile)
            For x = 0 To 45
                Colours(x) = FileReader.ReadLine()
```

1(a)

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

Java

```
public static String[] ReadData(){
       String TextFile = "Data.txt";
       String Colours[] = new String[45];

       try{
           FileReader f = new FileReader(TextFile);
           BufferedReader Reader = new BufferedReader(f);
           for(Integer X = 0; X < 45; X++){
                try{
                     Colours[X] = Reader.readLine();
                }catch(IOException ex){}
           }
           try{
                Reader.close();
           }catch(IOException ex){}
           return Colours;
       }catch(FileNotFoundException e){
           System.out.println("File not found");
       }
       return Colours;
}
```

1(b)(i)

1 mark each

- Function header (and end where appropriate) taking (min) one parameter
- Looping through each parameter array element, concatenating with space and returning

Python

```
def FormatArray(DataArray):
    OutputText = ""
    for x in range(0, 45):
        OutputText = OutputText + DataArray[x] + " "
    return OutputText
```

VB.NET

```
Function FormatArray(DataArray)
    Dim OutputText As String = ""
    For X = 0 To 44
        OutputText = OutputText & DataArray(X) & " "
    Next
    Return OutputText
End Function
```

Java

```
public static String FormatArray(String[] DataArray){
       String OutputText = "";
       for(Integer X = 0; X < 45; X++){
           OutputText = OutputText + DataArray[X] + " ";
       }
       return OutputText;
}
```

1(b)(ii)	1 mark each:
	• Calling `ReadData()` and storing returned array …
	• … calling `FormatArray()` with returned array
	• Outputting return value from `FormatArray()`
	
	Python
	`Colours = ReadData() #string array`
	`print(FormatArray(Colours))`
	
	VB.NET
	`Dim Colours(45) As String`
	`Colours = ReadData()`
	`Console.WriteLine(FormatArray(Colours))`
	
	Java
	`String[] Colours = new String[45];`
	`Colours = ReadData();`
	`System.out.println(FormatArray(Colours));`

1(b)(iii)	1 mark for output showing all colours in one string e.g. beige green scarlet silver bronze slate yellow orange jade lavender magenta magnolia turquoise black grey russet mango maroon mint purple red pink white cream navy olive brown violet cyan amber aqua azure copper fawn fuschia gold indigo ivory mauve mulberry peach periwinkle plum rose sage

1(c)	1 mark each
	• Function header (and close where appropriate) taking (min) two parameters and returns a value in all cases
	• Looping through each character in each string parameter …
	• … return 1 when first parameter < second
	• … return 2 when first parameter > second
	
	e.g.
	Python
	`def CompareStrings(First, Second):`
	`  Count = 0`
	`  while True:`
	`  if First[Count] < Second[Count]:`
	`  return 1`
	`  elif First[Count] > Second[Count]:`
	`  return 2`
	`  else:`
	`  Count = Count + 1`
	
	VB.NET
	`Function CompareStrings(FirstS, SecondS)`
	`  Dim Count As Integer = 1`
	`  While (True)`
	`  If Mid(FirstS, Count, 1) < Mid(SecondS, Count, 1) Then`
	`  Return 1`
	`  ElseIf Mid(FirstS, Count, 1) > Mid(SecondS, Count, 1) Then`
	`  Return 2`
	`  Else`
	`  Count = Count + 1`
	`  End If`
	`  End While`
	`End Function`

1(c)

Java

```
public static Integer CompareStrings(String First, String Second){
       Integer Count = 0;
       while(true){
           if(First.substring(Count, Count + 1).compareTo(Second.substring(Count, Count +
1)) < 0){
           return 1;
           }else if(First.substring(Count, Count + 1).compareTo(Second.substring(Count,
Count + 1))>0){
                 return 2;
           }else{
                 Count++;
           }
       }
}
```

1(d)(i)

1 mark each

- Bubble sort function header taking array parameter and returns **sorted** array in all cases
- Comparing strings using `CompareStrings()` **and** correctly swapping values when needed
- Correct bubble sort that sorts the data correctly

Python

```
def Bubble(DataArray):
    ArrayLength = len(DataArray)

    for x in range(ArrayLength - 1):
        for y in range(0, ArrayLength - x - 1):
            Result = CompareStrings(DataArray[y], DataArray[y + 1])
            if Result == 2:
                DataArray[y], DataArray[y+1] = DataArray[y+1], DataArray[y]
    return DataArray
```

1(d)(i)

VB.NET

```
Function Bubble(DataArray)
        Dim ArrayLength As Integer = 45
        Dim Result As Integer
        Dim Temp As String

        For X = 0 To ArrayLength - 1
            For Y = 0 To ArrayLength - X - 2
                Result = CompareStrings(DataArray(Y), DataArray(Y + 1))
                If Result = 2 Then
                    Temp = DataArray(Y)
                    DataArray(Y) = DataArray(Y + 1)
                    DataArray(Y + 1) = Temp
                End If
            Next
        Next
        Return DataArray

End Function
```

Java

```
public static String[] Bubble(String[] DataArray){
       Integer ArrayLength = 45;
       Integer Result;
       String Temp;
       for(Integer X = 0; X < ArrayLength ; X++){
           for(Integer Y = 0; Y < ArrayLength - X - 1; Y++){
                 Result = CompareStrings(DataArray[Y], DataArray[Y+1]);
```

1(d)(i)

```
                if (Result == 2){
                    Temp = DataArray[Y];
                    DataArray[Y] = DataArray[Y+1];
                    DataArray[Y+1] = Temp;
                }
            }
        }
        return DataArray;
}
```

1(d)(ii)

1 mark each

- Calling `Bubble()` with array as parameter **and** using/storing return value
- Calling `FormatArray()` with **return value** from `Bubble()` **and** outputting return value

Python

```
BubbleSorted = Bubble(Colours)
print(FormatArray(BubbleSorted))
```

VB.NET

```
Dim BubbleSorted(45) As String
BubbleSorted = Bubble(Colours)
Console.WriteLine(FormatArray(BubbleSorted))
```

Java

```
String[] BubbleSorted = new String[45];
BubbleSorted = Bubble(Colours);
System.out.println(FormatArray(BubbleSorted));
```

1(d)(iii)

1 mark for sorted data

e.g.

beige green scarlet silver bronze slate yellow orange jade lavender magenta magnolia turquoise black grey russet mango maroon mint purple red pink white cream navy olive brown violet cyan amber aqua azure copper fawn fuschia gold indigo ivory mauve mulberry peach periwinkle plum rose sage
amber aqua azure beige black bronze brown copper cream cyan fawn fuschia gold green grey indigo ivory jade lavender magenta magnolia mango maroon mauve mint mulberry navy olive orange peach periwinkle pink plum purple red rose russet sage scarlet silver slate turquoise violet white yellow

1(a)

1 mark each to max 6:

- Function declaration (and close where appropriate)
- Declaration/use of an array (with space/initialised with 45 spaces/strings)
- Opening the file Data.txt for read and closing in an appropriate place
- Looping through all file contents/Looping 45 times and reading each line …
- … storing all items from file into array
- Returning the populated array
- Exception handling with suitable try, catch and output

e.g.

Python

```
def ReadData():
    Colours = []
    try:
        File = open("Data.txt")
        Colours = File.read().split("\n")
        File.close()
        return Colours
    except:
        print("No file found")
```

VB.NET

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

        Dim Colours(45) As String
        Try
            Dim FileReader As New System.IO.StreamReader(TextFile)
            For x = 0 To 45
                Colours(x) = FileReader.ReadLine()
```

1(a)

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

Java

```
public static String[] ReadData(){
       String TextFile = "Data.txt";
       String Colours[] = new String[45];

       try{
           FileReader f = new FileReader(TextFile);
           BufferedReader Reader = new BufferedReader(f);
           for(Integer X = 0; X < 45; X++){
                try{
                     Colours[X] = Reader.readLine();
                }catch(IOException ex){}
           }
           try{
                Reader.close();
           }catch(IOException ex){}
           return Colours;
       }catch(FileNotFoundException e){
           System.out.println("File not found");
       }
       return Colours;
}
```

1(b)(i)

1 mark each

- Function header (and end where appropriate) taking (min) one parameter
- Looping through each parameter array element, concatenating with space and returning

Python

```
def FormatArray(DataArray):
    OutputText = ""
    for x in range(0, 45):
        OutputText = OutputText + DataArray[x] + " "
    return OutputText
```

VB.NET

```
Function FormatArray(DataArray)
    Dim OutputText As String = ""
    For X = 0 To 44
        OutputText = OutputText & DataArray(X) & " "
    Next
    Return OutputText
End Function
```

Java

```
public static String FormatArray(String[] DataArray){
       String OutputText = "";
       for(Integer X = 0; X < 45; X++){
           OutputText = OutputText + DataArray[X] + " ";
       }
       return OutputText;
}
```

1(b)(ii)	1 mark each:
	• Calling `ReadData()` and storing returned array …
	• … calling `FormatArray()` with returned array
	• Outputting return value from `FormatArray()`
	
	Python
	`Colours = ReadData() #string array`
	`print(FormatArray(Colours))`
	
	VB.NET
	`Dim Colours(45) As String`
	`Colours = ReadData()`
	`Console.WriteLine(FormatArray(Colours))`
	
	Java
	`String[] Colours = new String[45];`
	`Colours = ReadData();`
	`System.out.println(FormatArray(Colours));`

1(b)(iii)	1 mark for output showing all colours in one string e.g. beige green scarlet silver bronze slate yellow orange jade lavender magenta magnolia turquoise black grey russet mango maroon mint purple red pink white cream navy olive brown violet cyan amber aqua azure copper fawn fuschia gold indigo ivory mauve mulberry peach periwinkle plum rose sage

1(c)	1 mark each
	• Function header (and close where appropriate) taking (min) two parameters and returns a value in all cases
	• Looping through each character in each string parameter …
	• … return 1 when first parameter < second
	• … return 2 when first parameter > second
	
	e.g.
	Python
	`def CompareStrings(First, Second):`
	`    Count = 0`
	`    while True:`
	`        if First[Count] < Second[Count]:`
	`            return 1`
	`        elif First[Count] > Second[Count]:`
	`            return 2`
	`        else:`
	`            Count = Count + 1`
	
	VB.NET
	`Function CompareStrings(FirstS, SecondS)`
	`    Dim Count As Integer = 1`
	`    While (True)`
	`        If Mid(FirstS, Count, 1) < Mid(SecondS, Count, 1) Then`
	`            Return 1`
	`        ElseIf Mid(FirstS, Count, 1) > Mid(SecondS, Count, 1) Then`
	`            Return 2`
	`        Else`
	`            Count = Count + 1`
	`        End If`
	`    End While`
	`End Function`

1(c)

Java

```
public static Integer CompareStrings(String First, String Second){
       Integer Count = 0;
       while(true){
           if(First.substring(Count, Count + 1).compareTo(Second.substring(Count, Count +
1)) < 0){
           return 1;
           }else if(First.substring(Count, Count + 1).compareTo(Second.substring(Count,
Count + 1))>0){
                 return 2;
           }else{
                 Count++;
           }
       }
}
```

1(d)(i)

1 mark each

- Bubble sort function header taking array parameter and returns **sorted** array in all cases
- Comparing strings using `CompareStrings()` **and** correctly swapping values when needed
- Correct bubble sort that sorts the data correctly

Python

```
def Bubble(DataArray):
    ArrayLength = len(DataArray)

    for x in range(ArrayLength - 1):
        for y in range(0, ArrayLength - x - 1):
            Result = CompareStrings(DataArray[y], DataArray[y + 1])
            if Result == 2:
                DataArray[y], DataArray[y+1] = DataArray[y+1], DataArray[y]
    return DataArray
```

1(d)(i)

VB.NET

```
Function Bubble(DataArray)
        Dim ArrayLength As Integer = 45
        Dim Result As Integer
        Dim Temp As String

        For X = 0 To ArrayLength - 1
            For Y = 0 To ArrayLength - X - 2
                Result = CompareStrings(DataArray(Y), DataArray(Y + 1))
                If Result = 2 Then
                    Temp = DataArray(Y)
                    DataArray(Y) = DataArray(Y + 1)
                    DataArray(Y + 1) = Temp
                End If
            Next
        Next
        Return DataArray

End Function
```

Java

```
public static String[] Bubble(String[] DataArray){
       Integer ArrayLength = 45;
       Integer Result;
       String Temp;
       for(Integer X = 0; X < ArrayLength ; X++){
           for(Integer Y = 0; Y < ArrayLength - X - 1; Y++){
                 Result = CompareStrings(DataArray[Y], DataArray[Y+1]);
```

1(d)(i)

```
                if (Result == 2){
                    Temp = DataArray[Y];
                    DataArray[Y] = DataArray[Y+1];
                    DataArray[Y+1] = Temp;
                }
            }
        }
        return DataArray;
}
```

1(d)(ii)

1 mark each

- Calling `Bubble()` with array as parameter **and** using/storing return value
- Calling `FormatArray()` with **return value** from `Bubble()` **and** outputting return value

Python

```
BubbleSorted = Bubble(Colours)
print(FormatArray(BubbleSorted))
```

VB.NET

```
Dim BubbleSorted(45) As String
BubbleSorted = Bubble(Colours)
Console.WriteLine(FormatArray(BubbleSorted))
```

Java

```
String[] BubbleSorted = new String[45];
BubbleSorted = Bubble(Colours);
System.out.println(FormatArray(BubbleSorted));
```

1(d)(iii)

1 mark for sorted data

e.g.

beige green scarlet silver bronze slate yellow orange jade lavender magenta magnolia turquoise black grey russet mango maroon mint purple red pink white cream navy olive brown violet cyan amber aqua azure copper fawn fuschia gold indigo ivory mauve mulberry peach periwinkle plum rose sage

amber aqua azure beige black bronze brown copper cream cyan fawn fuschia gold green grey indigo ivory jade lavender magenta magnolia mango maroon mauve mint mulberry navy olive orange peach periwinkle pink plum purple red rose russet sage scarlet silver slate turquoise violet white yellow

4

Example:

```
FUNCTION Check() RETURNS STRING
   DECLARE Odd, Even, Index : INTEGER

   Odd ← 0
   Even ← 0

   FOR Index ← 1 TO 100
      IF Index MOD 2 = 0 THEN
         Even ← Even + Data[Index]
      ELSE
         Odd ← Odd + Data[Index]
      ENDIF
   NEXT Index

ENDFUNCTION
```

Mark as follows:

1. Function heading, ending and return type
2. Declare local variables `Odd`, `Even` and `Index` as integers
3. Initialise `Odd` and `Even`
4. Loop for 100 `iterations` // through array
5. Sum `Odd` and `Even` element values **in a loop**
6. Compare `Odd` and `Even` after the loop and Return appropriate string