Course Title: MSc. Information System with Computing

(Jan 2019)

Lecturer Name: Harnaik Dhoot

Subject Title: Software Engineering (B9IS118)

Assignment Title: Software Development.

By

Paul Dinesh Augustine (10515229)

Arden Dias (10510105)

Table of contents

INTRODUCTION 3

DESIGN OF THE APPLICATION 4

APPROACHES FOLLOWED 5

METHODOLOGY 20

CLASS DIAGRAM 23

SEQUENCE DIAGRAM A. 24

SEQUENCE DIAGRAM B. 24

LIMITATIONS 27

ASSIGNMENT SPECIFICATION: WHO DID WHAT- 26

CONCLUSION 28

INTRODUCTION

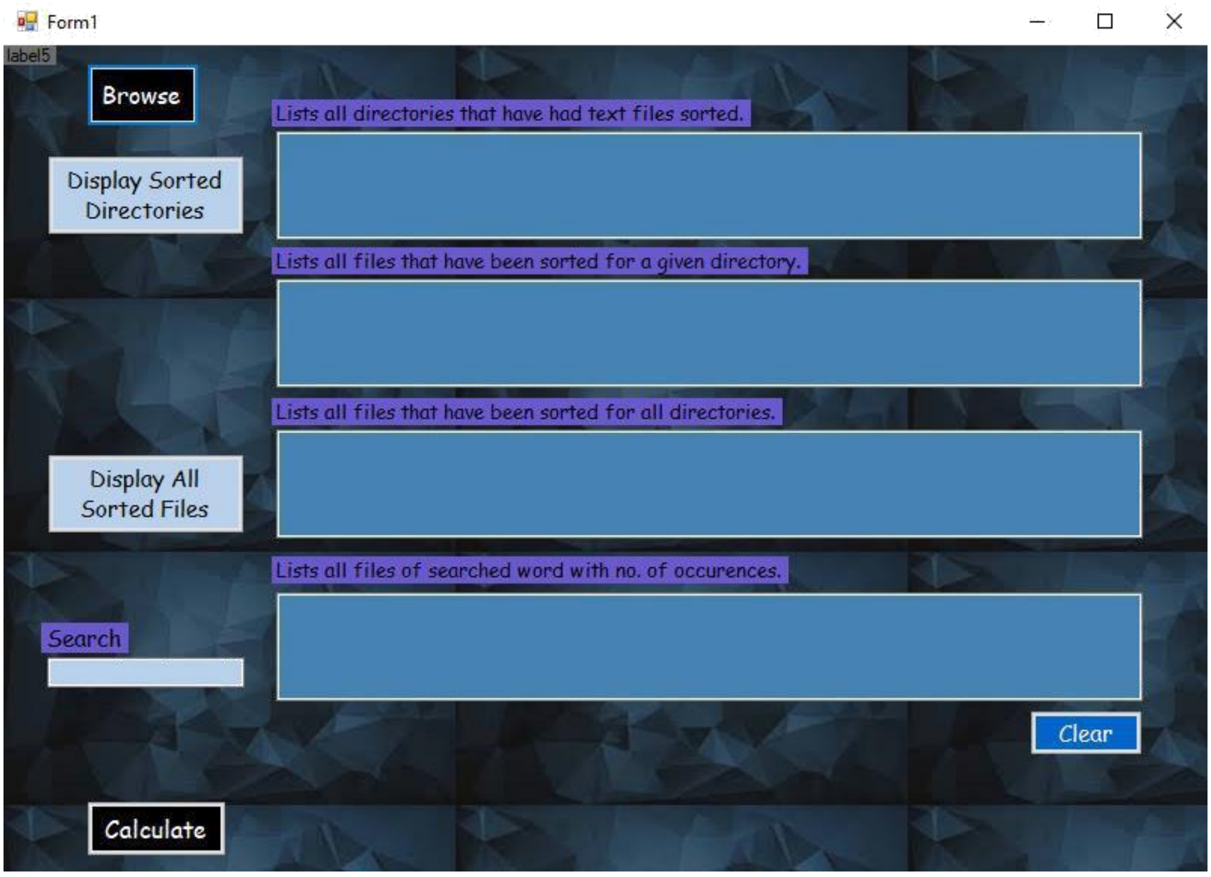
This report describes the process of constructing a Windows form application using C#. The system built, focuses on sorting the text and solves the arithmetic expressions from the files selected by the user in any directories. The system allows the user to select any directories with the help of the browse button. On selection of directory, every **.text** file present in that folder will be read by the system, sorted and a new file will be created with the word “Sorted” prepended to the original file name (ex. ABC.text will be SortedABC.text) with the data all sorted as per requirements 3, 4, and 5. The system keeps tracks of all the selected directories and the sorted files. The system also asks the users to list all selected text file directories, list all sorted files for a given directory, list all sorted files for all directories and display the sorted files based on the search key words entered by the user.

The second major functionality allows the system to process a text file of extension **.calc** with multiple calculations present. The mechanism processes the arithmetic expressions from the multiple text files and copies the same along with the answers of the expressions in the new file with the extension **.answ**. Few edge cases for the arithmetic expressions are considered by the system. The mechanism of the entire system with all its requirements are presented with the help of windows form.

DESIGN OF THE APPLICATION

There is a two main buttons on the top left and bottom left of the dialog screen namely B**rowse** and **Calculate** respectively. These are the two major functionalities of the system. There is a **Display Sorted Directories** button and a list box to display the same. There is a second list box that displays all sorted files for a given directory when selecting the listed directory form the list box. The **Display All Sorted Files** buttons will display all the files sorted in all the directories in the following list box.

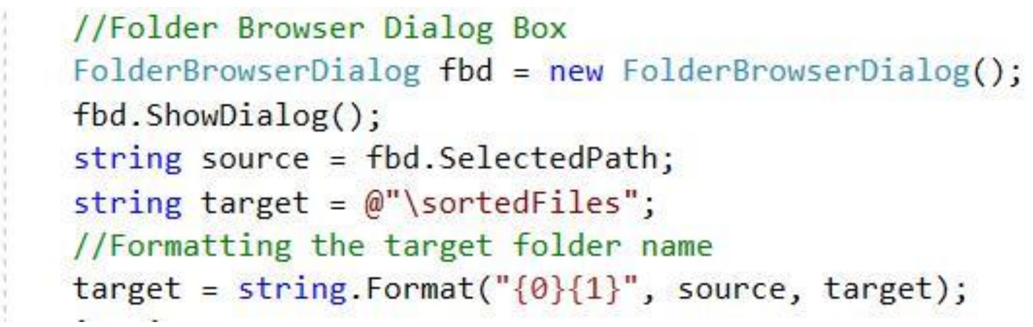
There is also a S**earch** field and a list box to display the directory of the file and the occurrence of the search term in that file. There is a **Clear** button on the bottom right of the form to clear the list boxes at any point in time. On clicking the **Calculate** button a dialog box appears to select the directory and message box appears when the arithmetic expressions are solved.


Form1
label5
Browse
Lists all directories that have had text files sorted.
Display Sorted Directories
Lists all files that have been sorted for a given directory.
Lists all files that have been sorted for all directories.
Display All Sorted Files
Lists all files of searched word with no. of occurences.
Search
Clear
Calculate

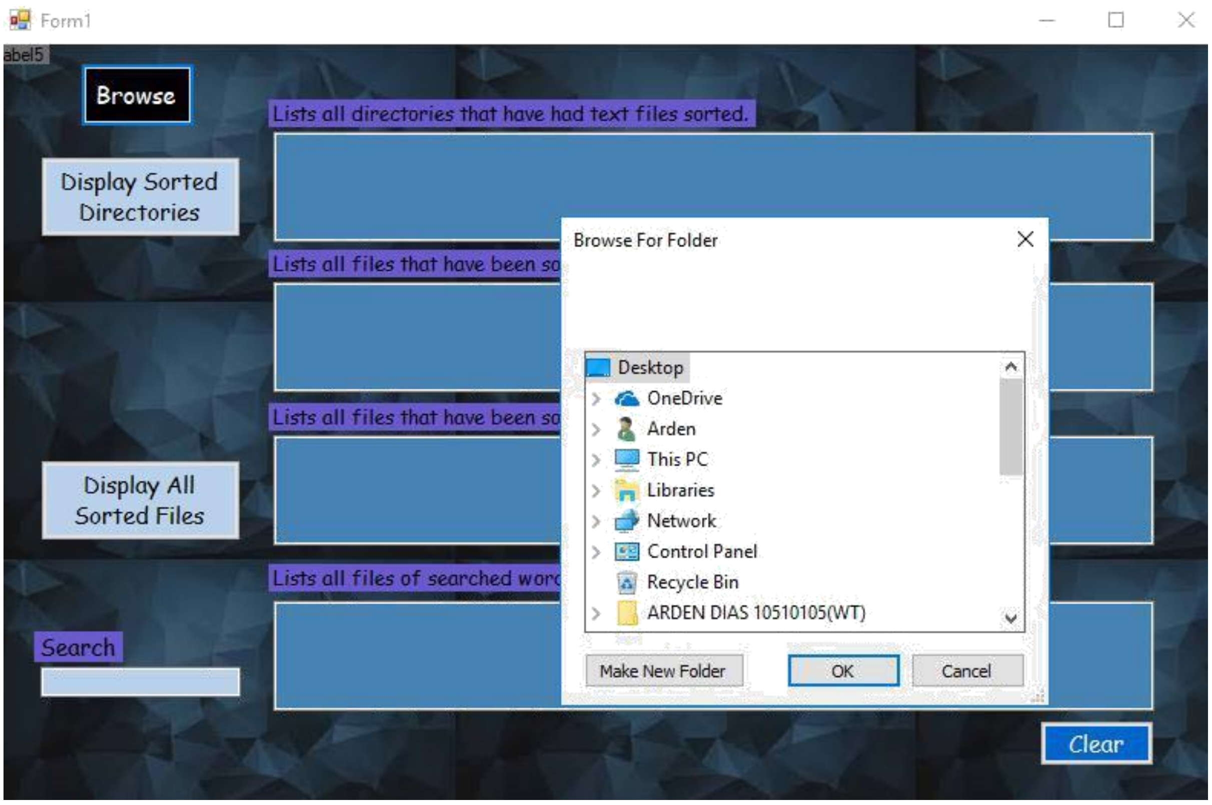
Approaches followed

Requirements 1: Allow the user to select directories on their computers.

Folder browser dialog Class is used to select a directory by the user. When the user clicks the Browse button a dialog Box appears which allows the user to select a directory.

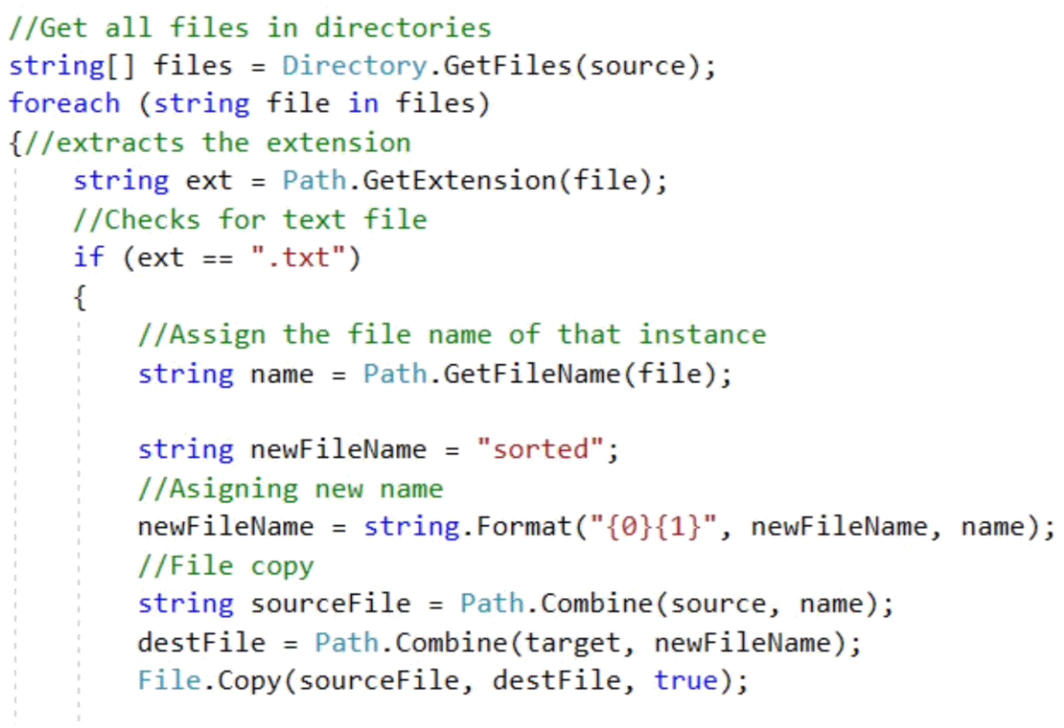


```
//Folder Browser Dialog Box
FolderBrowserDialog fbd = new FolderBrowserDialog();
fbd.ShowDialog();
string source = fbd.SelectedPath;
string target = @"\sortedFiles";
//Formatting the target folder name
target = string.Format("{0}{1}", source, target);
```

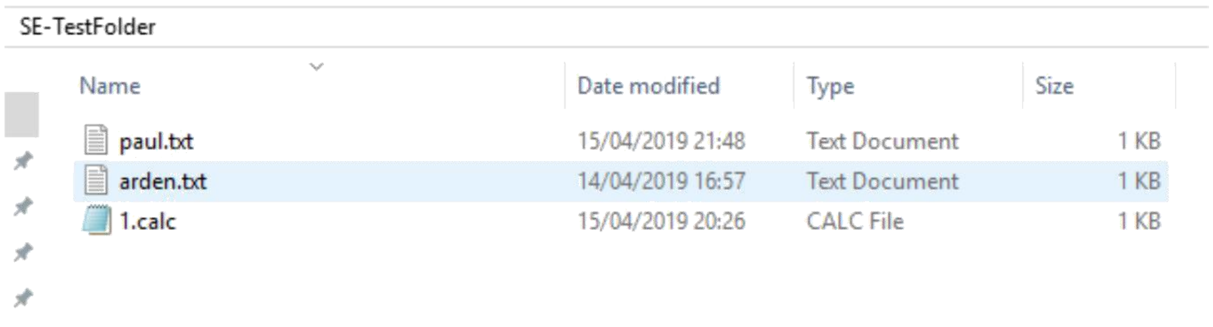


Requirements 2: For each text file located in a selected directory, you must create another text file that must have the same name as the original, but with the word "sorted" prepended to the file name. (E.g., if the original file was called mydiary.txt, the copy will be called sortedmydiary.txt).

The application creates a new directory "sorted" inside the selected directory. In this Sorted directory, files which would be sorted in the next step are kept. Then the application copies all the file from the selected directory to the sorted directory with a new file name using **File.Copy** method. The new file name contains the word "sorted" prepended to the existing name e.g., if the name of the file is **test.txt** the new file name would be **sortedtest.txt**. Also, it copies the files along with the original contents to the newly created directory sorted.

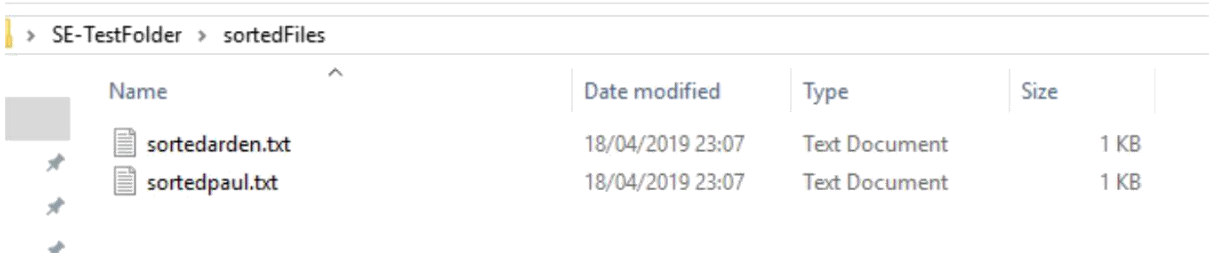


```
//Get all files in directories
string[] files = Directory.GetFiles(source);
foreach (string file in files)
{//extracts the extension
    string ext = Path.GetExtension(file);
    //Checks for text file
    if (ext == ".txt")
    {
        //Assign the file name of that instance
        string name = Path.GetFileName(file);

        string newFileName = "sorted";
        //Asigning new name
        newFileName = string.Format("{0}{1}", newFileName, name);
        //File copy
        string sourceFile = Path.Combine(source, name);
        destFile = Path.Combine(target, newFileName);
        File.Copy(sourceFile, destFile, true);
```



SE-TestFolder

Name	Date modified	Type	Size
paul.txt	15/04/2019 21:48	Text Document	1 KB
arden.txt	14/04/2019 16:57	Text Document	1 KB
1.calc	15/04/2019 20:26	CALC File	1 KB

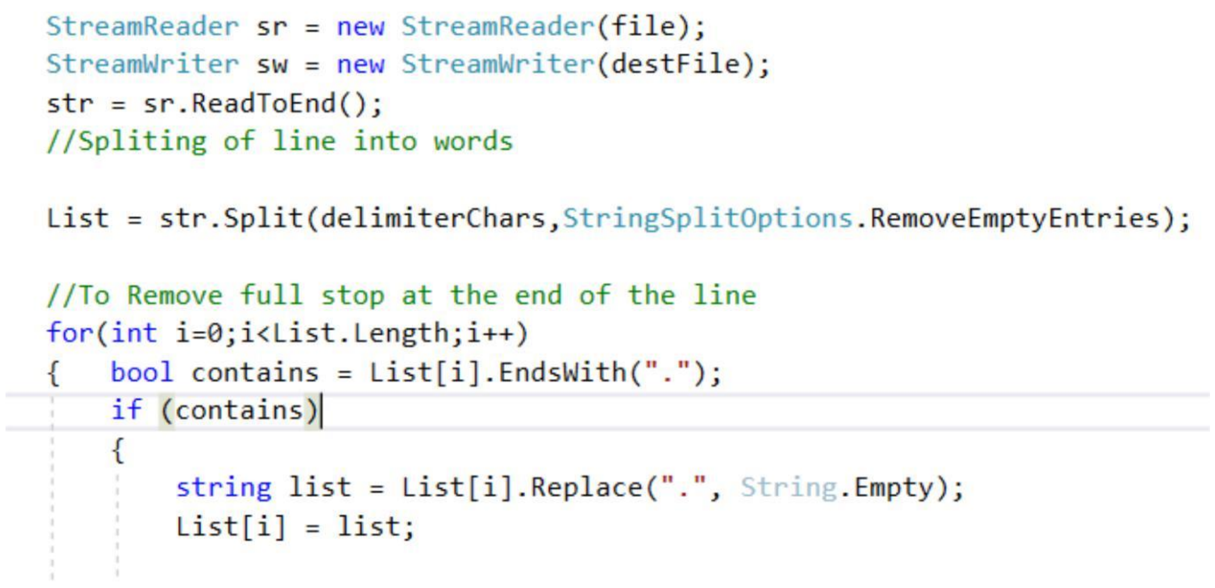


› SE-TestFolder › sortedFiles

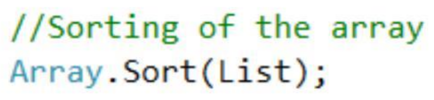
Name	Date modified	Type	Size
sortedarden.txt	18/04/2019 23:07	Text Document	1 KB
sortedpaul.txt	18/04/2019 23:07	Text Document	1 KB

Requirements 3: This "sorted" text file's contents are the words located in the original text file in alphabetical order, with one word/number appearing per line. Any numerical values must be stored in numerical order at the start of the file. In addition, if a given word appears more than once, the frequency count should be stored alongside the entry.

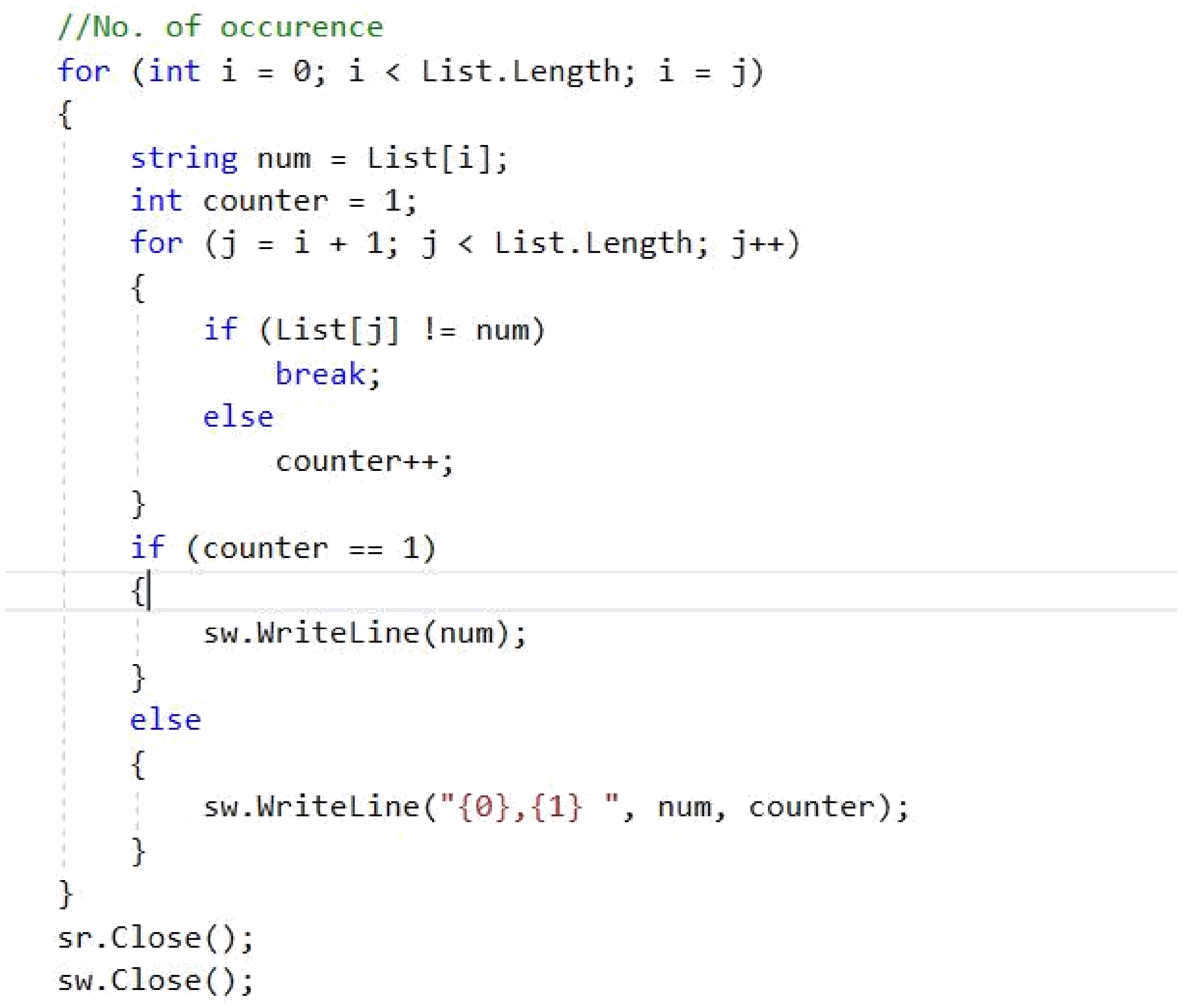
The application checks for the text files in the selected directory and reads the contents of the file using **Stream Reader Class** and the contents are stored in a string variable. Then it splits the contents into words by using **Split** method and the contents are stored in an array. Then a sort method is invoked to the array which contains all the contents of the file. To find the occurrence of the contents in an array a search functionality is defined and the occurrence of every element in an array is found. Then by using **Stream Writer Class** the sorted contents and their occurrence are written in the respective files using **WriteLine** method. An If statement is included at the beginning to check whether the file has .txt extension by invoking **Path.GetExtension** method.



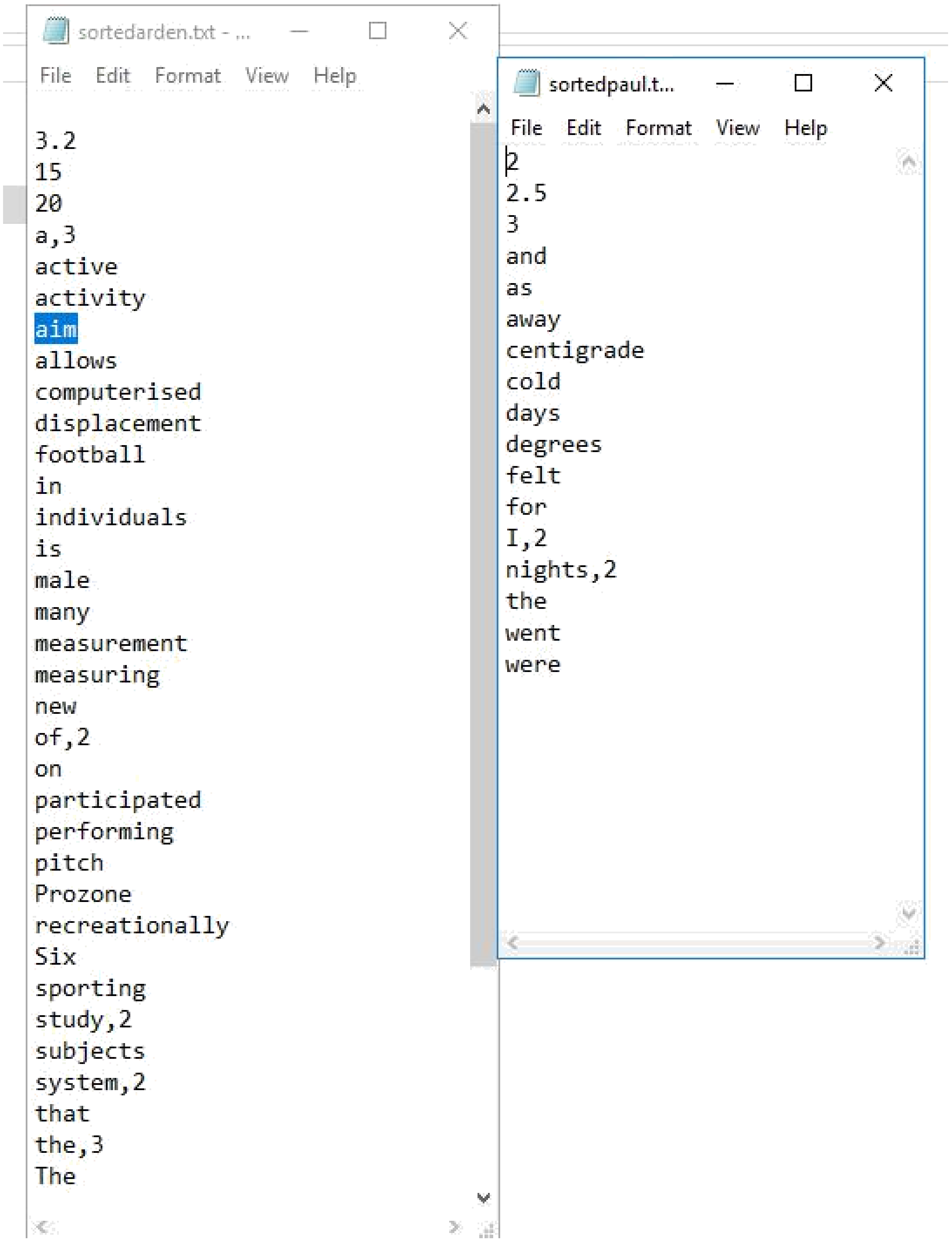
```
StreamReader sr = new StreamReader(file);
StreamWriter sw = new StreamWriter(destFile);
str = sr.ReadToEnd();
//Spliting of line into words

List = str.Split(delimiterChars,StringSplitOptions.RemoveEmptyEntries);

//To Remove full stop at the end of the line
for(int i=0;i<List.Length;i++)
{   bool contains = List[i].EndsWith(".");
    if (contains)
    {
        string list = List[i].Replace(".", String.Empty);
        List[i] = list;
```



```
//Sorting of the array
Array.Sort(List);
```



```
//No. of occurence
for (int i = 0; i < List.Length; i = j)
{
    string num = List[i];
    int counter = 1;
    for (j = i + 1; j < List.Length; j++)
    {
        if (List[j] != num)
            break;
        else
            counter++;
    }
    if (counter == 1)
    {
        sw.WriteLine(num);
    }
    else
    {
        sw.WriteLine("{0},{1} ", num, counter);
    }
}
sr.Close();
sw.Close();
```



```
3.2
15
20
a,3
active
activity
aim
allows
computerised
displacement
football
in
individuals
is
male
many
measurement
measuring
new
of,2
on
participated
performing
pitch
Prozone
recreationally
Six
sporting
study,2
subjects
system,2
that
the,3
The
```

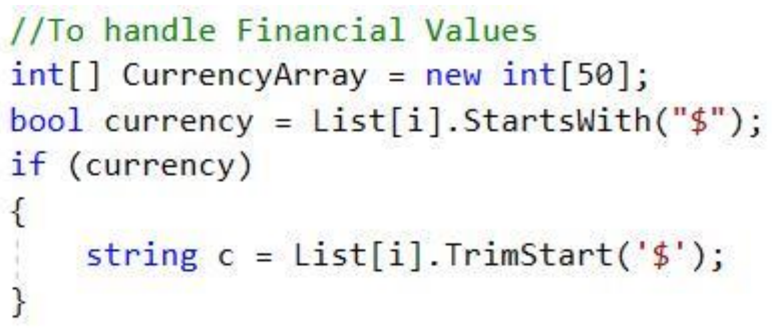
```
2
2.5
3
and
as
away
centigrade
cold
days
degrees
felt
for
I,2
nights,2
the
went
were
```

Requirements 4: Please be aware of sort order. ASCII sorting will place words starting with capital letters ahead of those with lower case. However, you should be sorting in the order a typical dictionary sorts in.

The contents written in the Sorted file are sorted as per alphabetical order and are not done by ASCII values.

Requirements 5: Other things you will need to consider are: handling financial values: €2.75, temperatures (21.6°C).

The logic for handling financial values and temperatures is given below.



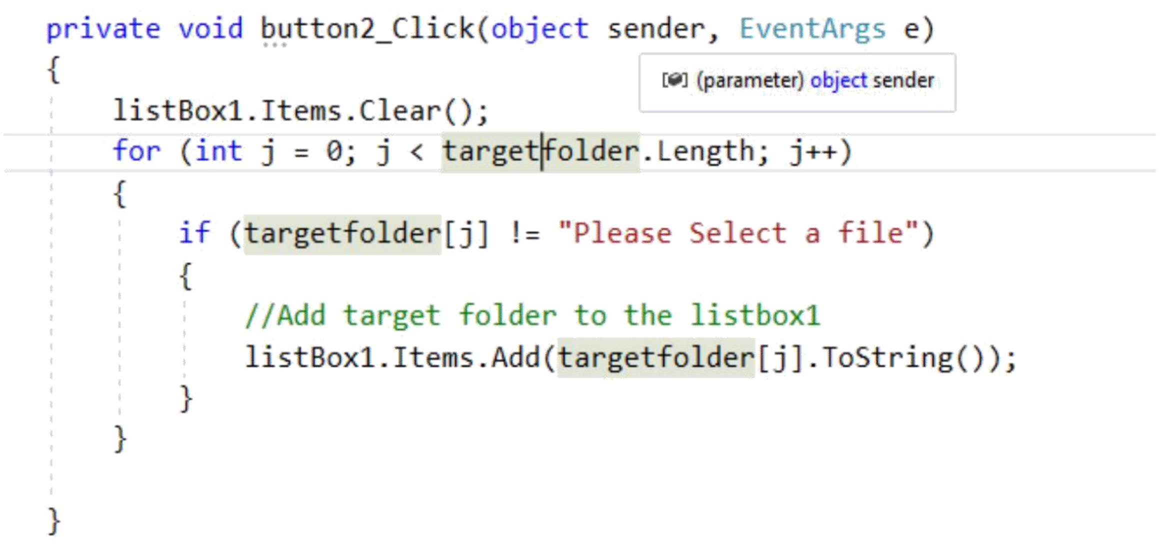
```
//To handle Financial Values
int[] CurrencyArray = new int[50];
bool currency = List[i].StartsWith("$");
if (currency)
{
    string c = List[i].TrimStart('$');
}
```

By Extracting the number from the Array List based on currency using **StartsWith** and **TrimStart** method and the extracted numbers can be stored in another array and can be separately sorted with **Array.Sort** method and the sorted elements can be written in the file by using **WriteLine** method in **Stream Writer** class.

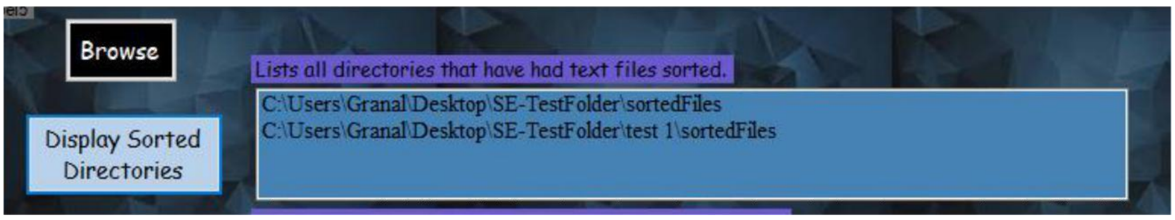
Requirements 6: Your system must keep a track of all directories that were selected by the user and all sorted files—where they are and what they are called. This is so that users can ask the system to:

a. List all directories that have had text files sorted.

When the D**isplay Sorted Directories** button is clicked the directories, which contains the sorted text files are listed in the first list box. When the targets directories are created, the directories are stored in an array **targetfolder**. When the display button is clicked the elements in the **targetfolder** array are displayed in the list box.



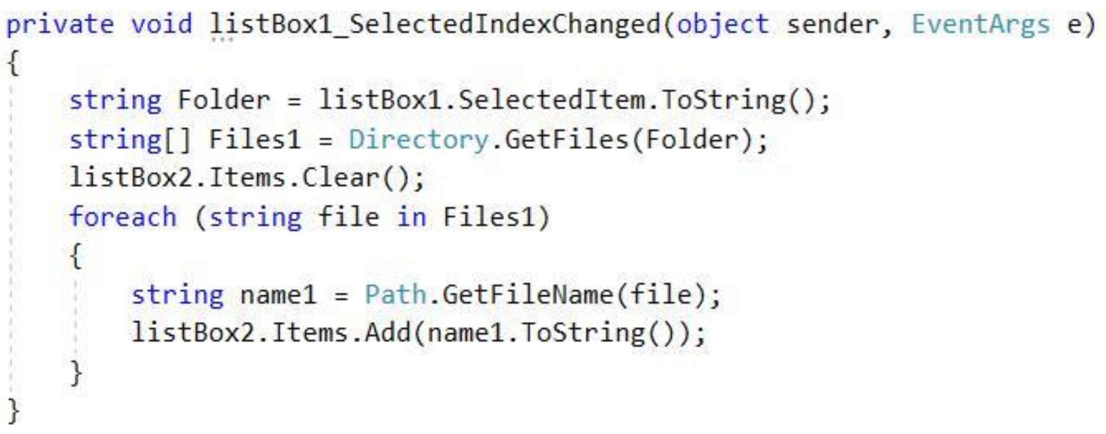
```
private void button2_Click(object sender, EventArgs e)
{
    listBox1.Items.Clear();
    for (int j = 0; j < targetfolder.Length; j++)
    {
        if (targetfolder[j] != "Please Select a file")
        {
            //Add target folder to the listbox1
            listBox1.Items.Add(targetfolder[j].ToString());
        }
    }

}
```


Browse
Display Sorted Directories
Lists all directories that have had text files sorted.

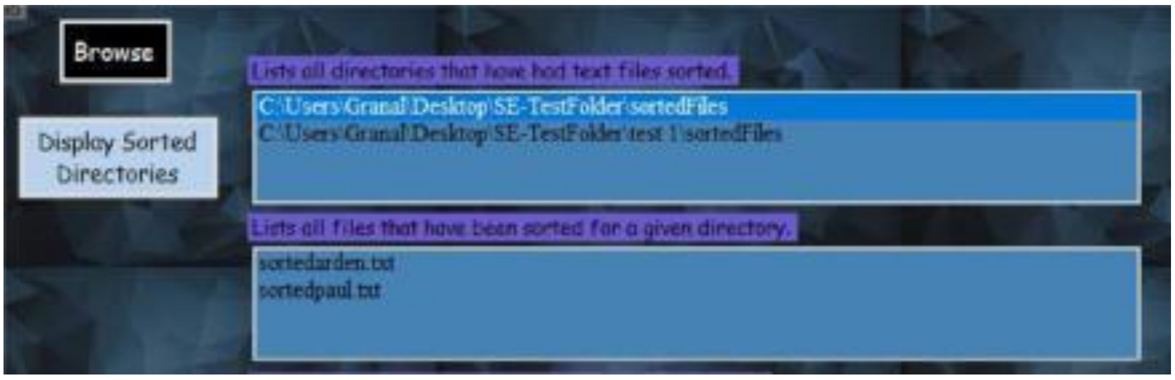
```
C:\Users\Granal\Desktop\SE-TestFolder\sortedFiles
C:\Users\Granal\Desktop\SE-TestFolder\test 1\sortedFiles
```

b. List all files that have been sorted for a given directory

In order to list all files that have been sorted for a given directory the user must select a directory listed in the first list box. When the directory is selected the **Directory.GetFiles** method is invoked for the selected directory and the names of all the files are retrieved by using Path.GetFileName method and all file names are added to the list box.



```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    string Folder = listBox1.SelectedItem.ToString();
    string[] Files1 = Directory.GetFiles(Folder);
    listBox2.Items.Clear();
    foreach (string file in Files1)
    {
        string name1 = Path.GetFileName(file);
        listBox2.Items.Add(name1.ToString());
    }
}
```



Lists all directories that have had text files sorted.

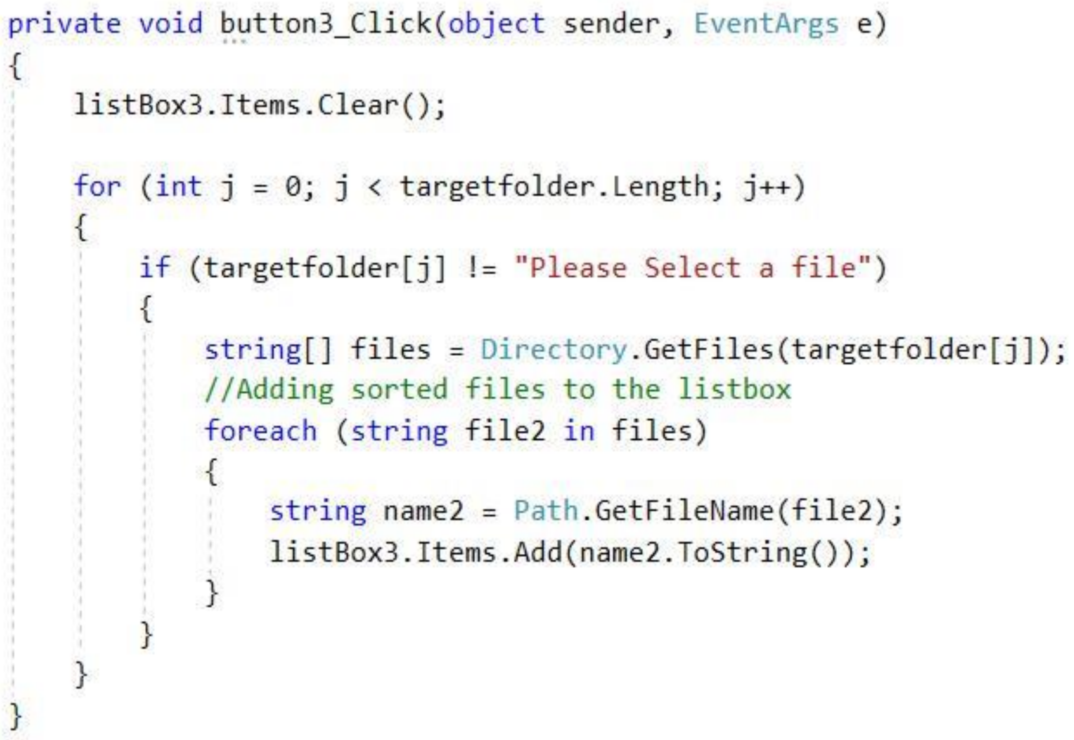
```
C:\Users\Granal\Desktop\SE-TestFolder\sortedFiles
C:\Users\Granal\Desktop\SE-TestFolder\test 1\sortedFiles
```

Lists all files that have been sorted for a given directory.

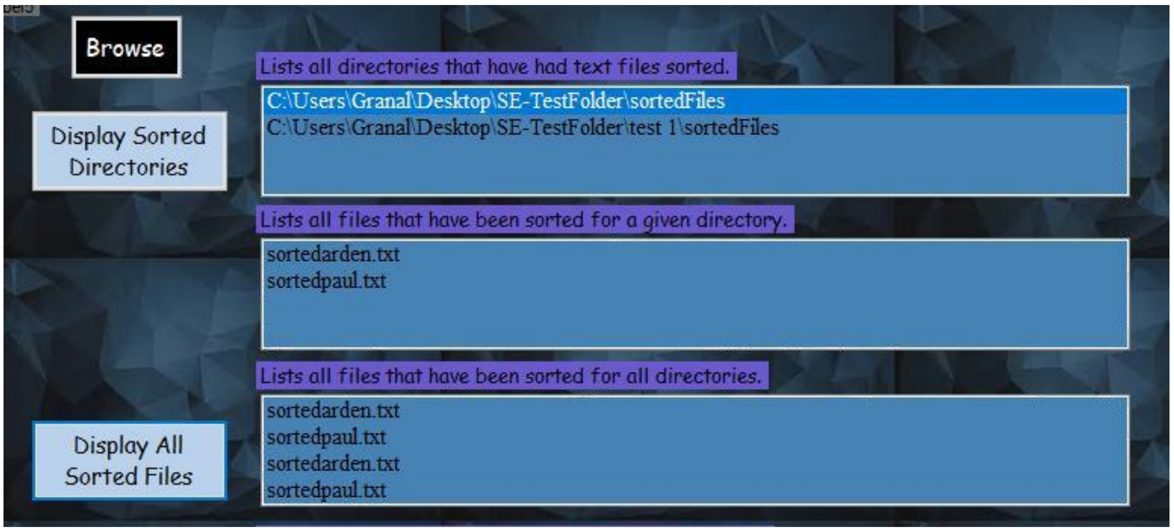
```
sortedarden.txt
sortedpaul.txt
```

c. List all files that have been sorted for all directories.

When the user clicks the **Display All Files** button the **Directory.GetFiles** method and **Path.GetFileName** method is invoked for the target directories and all files that have been sorted for all directories are listed in the third list box.



```
private void button3_Click(object sender, EventArgs e)
{
    listBox3.Items.Clear();

    for (int j = 0; j < targetfolder.Length; j++)
    {
        if (targetfolder[j] != "Please Select a file")
        {
            string[] files = Directory.GetFiles(targetfolder[j]);
            //Adding sorted files to the listbox
            foreach (string file2 in files)
            {
                string name2 = Path.GetFileName(file2);
                listBox3.Items.Add(name2.ToString());
            }
        }
    }
}
```



Lists all directories that have had text files sorted.

C:\Users\Granal\Desktop\SE-TestFolder\sortedFiles
C:\Users\Granal\Desktop\SE-TestFolder\test 1\sortedFiles

Lists all files that have been sorted for a given directory.

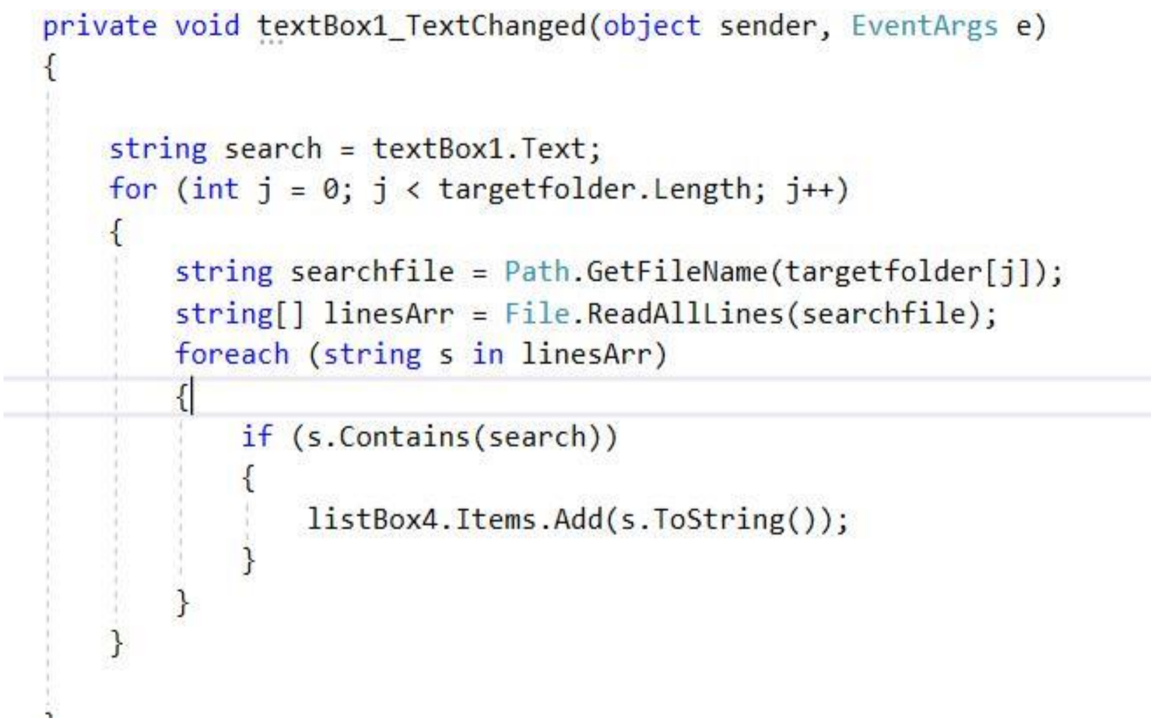
sortedarden.txt
sortedpaul.txt

Lists all files that have been sorted for all directories.

sortedarden.txt
sortedpaul.txt
sortedarden.txt
sortedpaul.txt

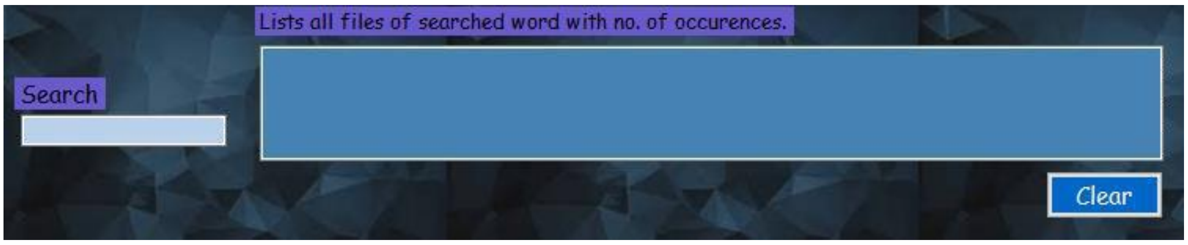
d. **Allow the user to search sorted files for a given word or number. The system should display all files that contain the searched for word along with the number of times the given word or number is displayed.**

The logic that allows the user to search sorted files for a given word or number is given below.



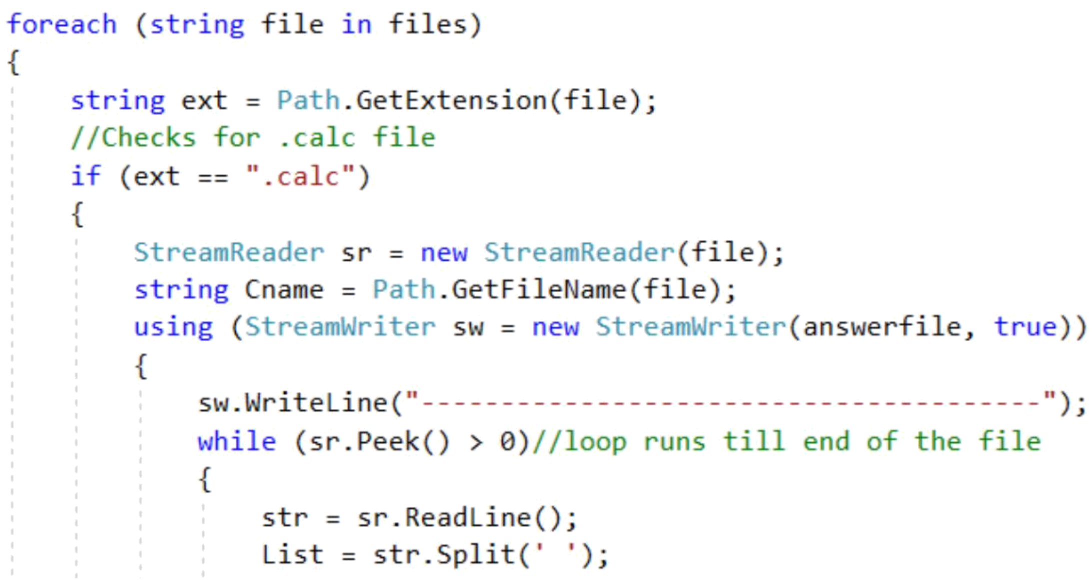
```
private void textBox1_TextChanged(object sender, EventArgs e)
{

    string search = textBox1.Text;
    for (int j = 0; j < targetfolder.Length; j++)
    {
        string searchfile = Path.GetFileName(targetfolder[j]);
        string[] linesArr = File.ReadAllLines(searchfile);
        foreach (string s in linesArr)
        {
            if (s.Contains(search))
            {
                listBox4.Items.Add(s.ToString());
            }
        }
    }
```

When user enters a search term in the search text box, it checks with all files in each sorted directories by using **Contains** method. If the search term is fold then it adds to the list box along with number of occurrence which can be retrieved from the sorted file.

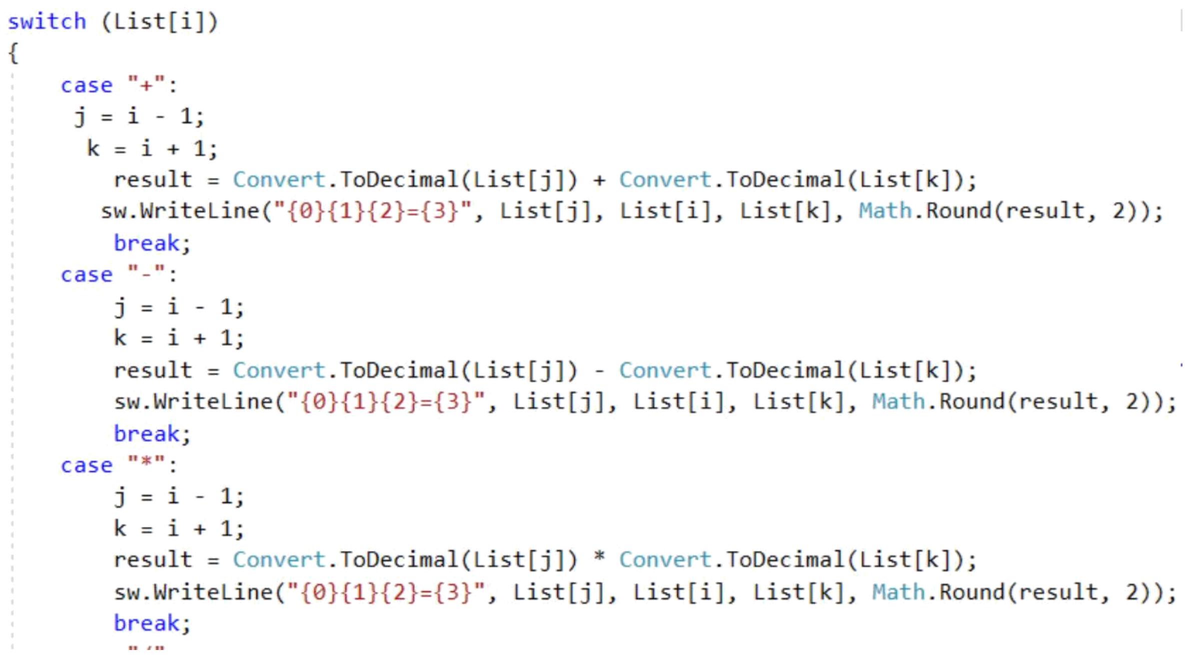


Requirements 7: The second piece of major functionality that your application must support is the ability to process a text file with several calculations present and your application should carry out the calculations listed and produce an output file. Note: all calculations should be rounded to 2 decimal places.

Arithmetic Expressions are stored in a file. When the button is clicked the user can select the directory where the file contents of Arithmetic Expressions are present. Selection of directory are done by invoking **FolderBrowserDialogBox** method. The selected file is read by using **StreamReader** class and stored it in an array. By using switch case branching statements the system identifies the cases where operators are **+,-,*,/,%,^** and performs the respective arithmetic operation to the value before and after the operator. The result is stored in another variable. The Output along with the arithmetic expression are written in newly created file. The new file is created by invoking **File.CreateText** method in the **StreamWriter** Class.

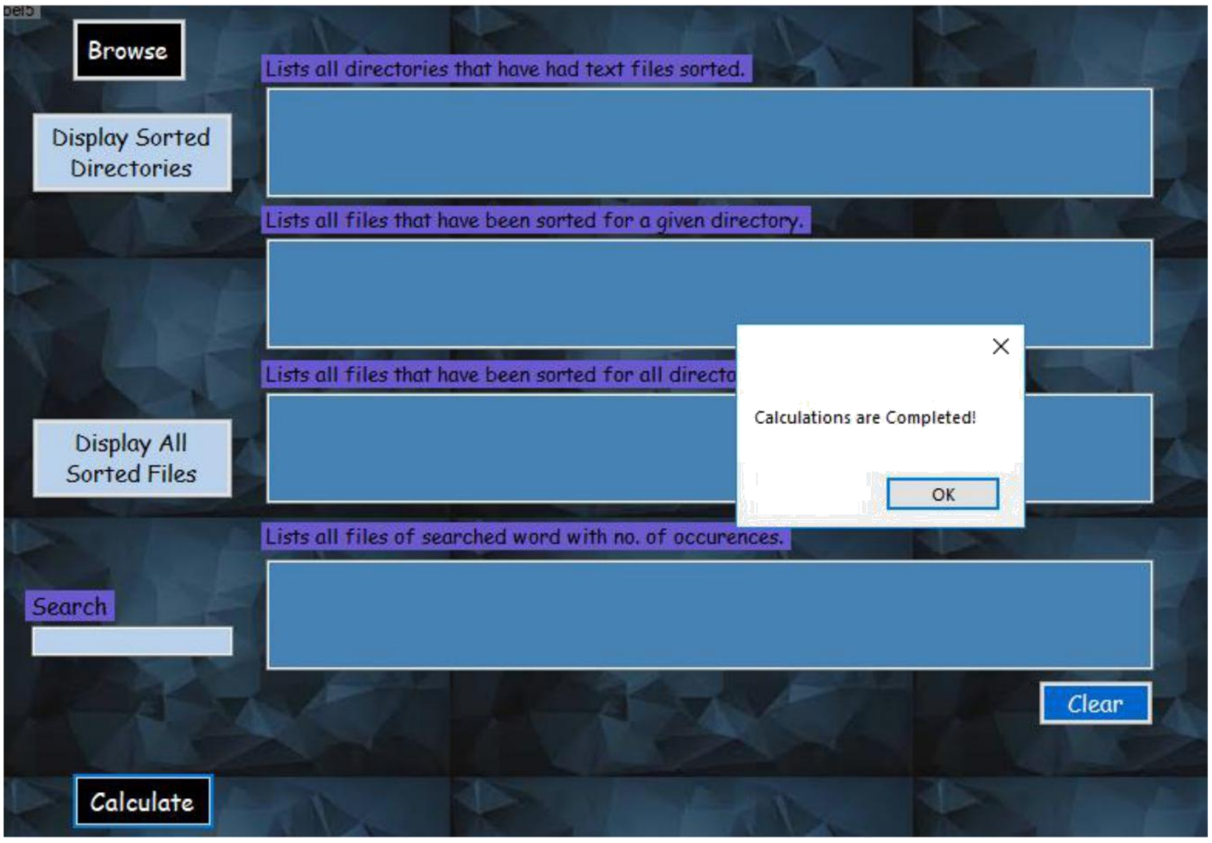


```
foreach (string file in files)
{
    string ext = Path.GetExtension(file);
    //Checks for .calc file
    if (ext == ".calc")
    {
        StreamReader sr = new StreamReader(file);
        string Cname = Path.GetFileName(file);
        using (StreamWriter sw = new StreamWriter(answerfile, true))
        {
            sw.WriteLine("---------------------------------------");
            while (sr.Peek() > 0)//loop runs till end of the file
            {
                str = sr.ReadLine();
                List = str.Split(' ');
```



```
switch (List[i])
{
    case "+":
     j = i - 1;
      k = i + 1;
        result = Convert.ToDecimal(List[j]) + Convert.ToDecimal(List[k]);
       sw.WriteLine("{0}{1}{2}={3}", List[j], List[i], List[k], Math.Round(result, 2));
        break;
    case "-":
        j = i - 1;
        k = i + 1;
        result = Convert.ToDecimal(List[j]) - Convert.ToDecimal(List[k]);
        sw.WriteLine("{0}{1}{2}={3}", List[j], List[i], List[k], Math.Round(result, 2));
        break;
    case "*":
        j = i - 1;
        k = i + 1;
        result = Convert.ToDecimal(List[j]) * Convert.ToDecimal(List[k]);
        sw.WriteLine("{0}{1}{2}={3}", List[j], List[i], List[k], Math.Round(result, 2));
        break;
```

```
case "/":
    j = i - 1;
    k = i + 1;
    result = Convert.ToDecimal(List[j]) / Convert.ToDecimal(List[k]);
    sw.WriteLine("{0}{1}{2}={3}", List[j], List[i], List[k], Math.Round(result, 2));
    break;
case "%":
    j = i - 1;
    k = i + 1;
    result = Convert.ToDecimal(List[j]) % Convert.ToDecimal(List[k]);
    sw.WriteLine("{0}{1}{2}={3}", List[j], List[i], List[k], Math.Round(result, 2));
    break;
case "^":
    j = i - 1;
    k = i + 1;
    double resul = Math.Pow(Convert.ToDouble(List[j]), Convert.ToDouble(List[k]));
    sw.WriteLine("{0}{1}{2}={3}", List[j], List[i], List[k], Math.Round(resul, 2));
    break;
```



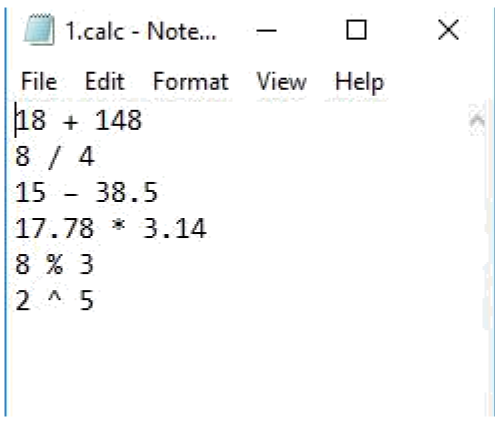
Lists all directories that have had text files sorted.

Lists all files that have been sorted for a given directory.

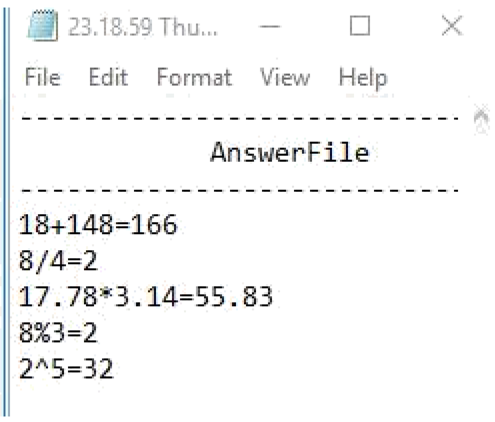
Lists all files that have been sorted for all directo

Lists all files of searched word with no. of occurences.

Calculations are Completed!



```
18 + 148
8 / 4
15 - 38.5
17.78 * 3.14
8 % 3
2 ^ 5
```



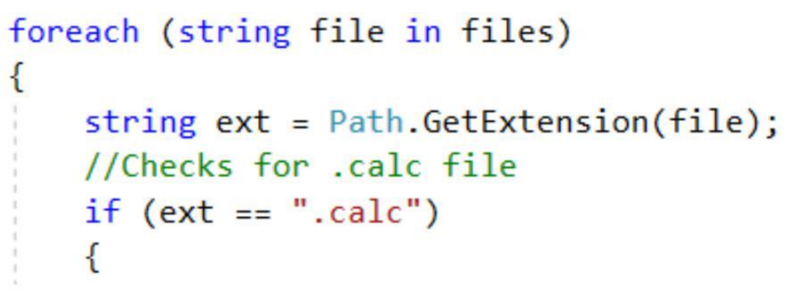
```
----------------------------
         AnswerFile
----------------------------
18+148=166
8/4=2
17.78*3.14=55.83
8%3=2
2^5=32
```

Requirements 8: There are edge cases you will need to consider for the calculator. I will not list them here—you should be able to identify these and discuss them in your report.
Below are the edge cases identified for arithmetic expression:

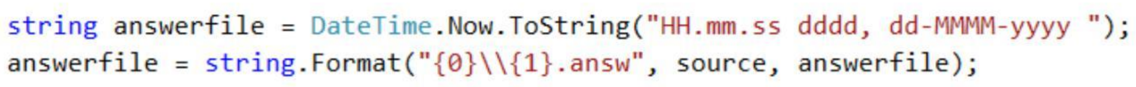
1. Divided by 0 (10/0)
2. Operator present at first(*3+2)
3. More than one Decimal point in a number(8..2 or 8.2.2)
4. Use of Two operators sequentially (2+ *3)
5. Alphabets are present instead of numbers (2+a)
6. Operand at the end of expression (2+3-)

Requirements 9: Your calculation should support the ability to process any number of calculation files located in a given directory. These files should be identified by the extension .calc. Your answers should be output to a file whose name is the current data and time with an extension of .answ.

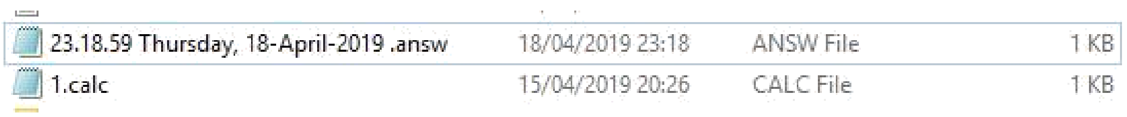
File name of the answer file is renamed to the current date and time by invoking **DateTime.Now.ToString**. An If statement is included to check whether the file has **.calc** extension by invoking **Path.GetExtension** method.



```
foreach (string file in files)
{
    string ext = Path.GetExtension(file);
    //Checks for .calc file
    if (ext == ".calc")
    {
```



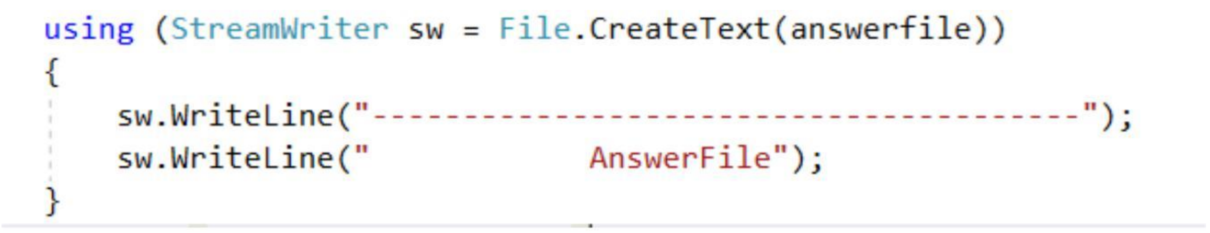
```
string answerfile = DateTime.Now.ToString("HH.mm.ss dddd, dd-MMMM-yyyy ");
answerfile = string.Format("{0}\\{1}.answ", source, answerfile);
```



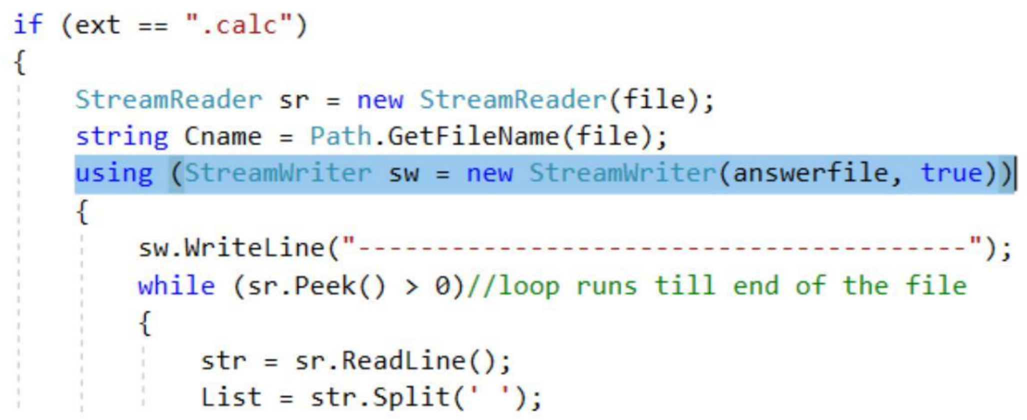
```
23.18.59 Thursday, 18-April-2019 .answ    18/04/2019 23:18    ANSW File    1 KB
1.calc                                    15/04/2019 20:26    CALC File    1 KB
```

Requirements 10: If a user selects the calculator functionality of your application and selects a directory that contains 3 files with the .calc extension, your application will process each .calc file in turn, but will only produce one .answ file containing the calculations and their respective answers.

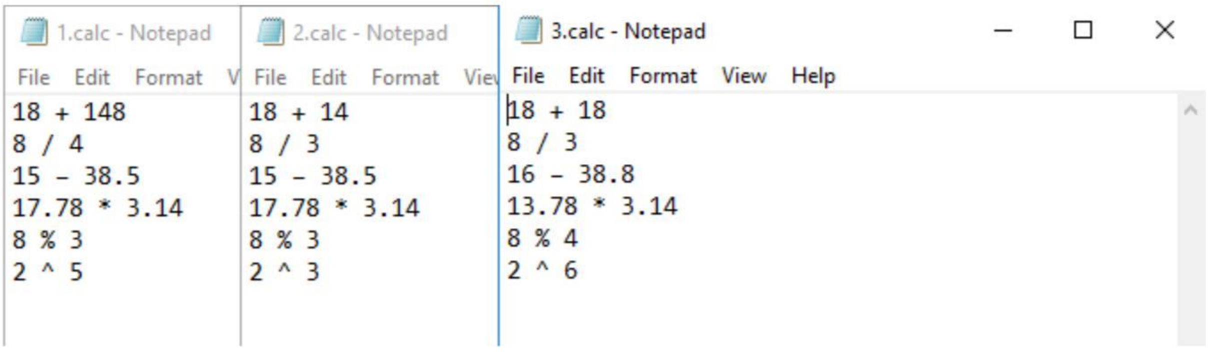
By using a forloop as given below all the files in the selected directory are processed which has **.calc** extension to find the solution of each arithmetic expressions present in the file.



```
using (StreamWriter sw = File.CreateText(answerfile))
{
    sw.WriteLine("--------------------------------------");
    sw.WriteLine("              AnswerFile");
}
```



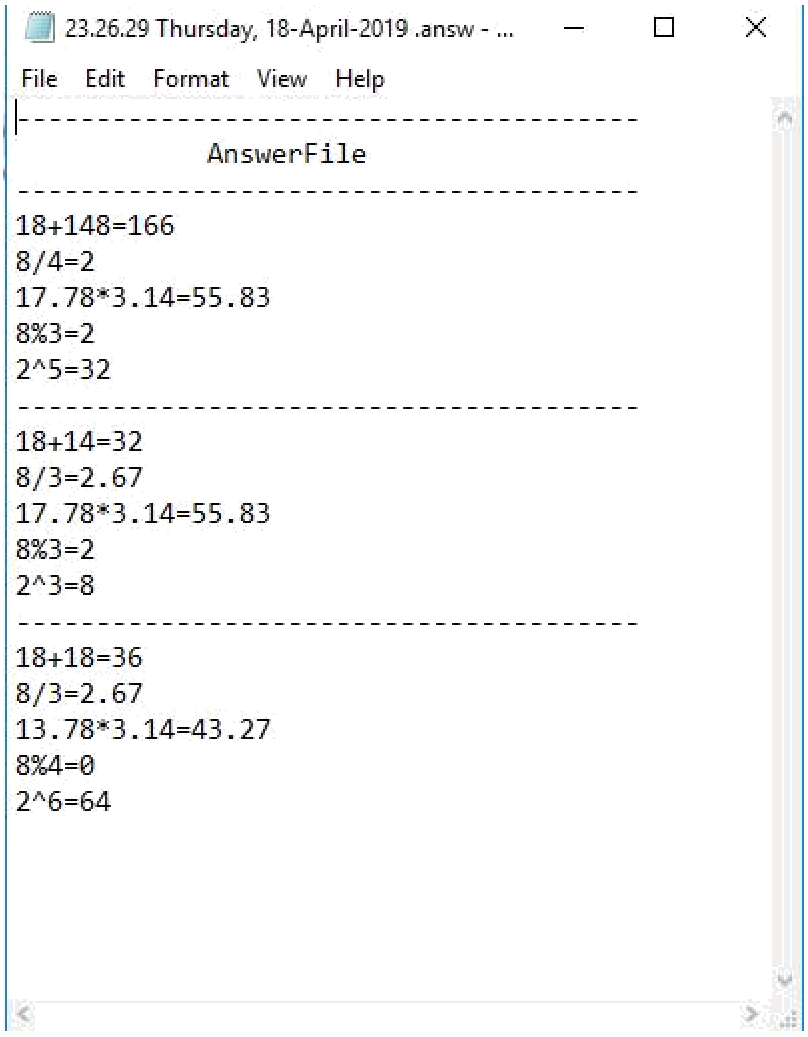
```
if (ext == ".calc")
{
    StreamReader sr = new StreamReader(file);
    string Cname = Path.GetFileName(file);
    using (StreamWriter sw = new StreamWriter(answerfile, true))
    {
        sw.WriteLine("--------------------------------------");
        while (sr.Peek() > 0)//loop runs till end of the file
        {
            str = sr.ReadLine();
            List = str.Split(' ');
```



```
1.calc - Notepad
18 + 148
8 / 4
15 - 38.5
17.78 * 3.14
8 % 3
2 ^ 5
```

```
2.calc - Notepad
18 + 14
8 / 3
15 - 38.5
17.78 * 3.14
8 % 3
2 ^ 3
```

```
3.calc - Notepad
18 + 18
8 / 3
16 - 38.8
13.78 * 3.14
8 % 4
2 ^ 6
```


23.26.29 Thursday, 18-April-2019 .answ - ...
File Edit Format View Help

```
--------------------------------------
            AnswerFile
--------------------------------------
18+148=166
8/4=2
17.78*3.14=55.83
8%3=2
2^5=32
--------------------------------------
18+14=32
8/3=2.67
17.78*3.14=55.83
8%3=2
2^3=8
--------------------------------------
18+18=36
8/3=2.67
13.78*3.14=43.27
8%4=0
2^6=64
```

METHODOLOGY

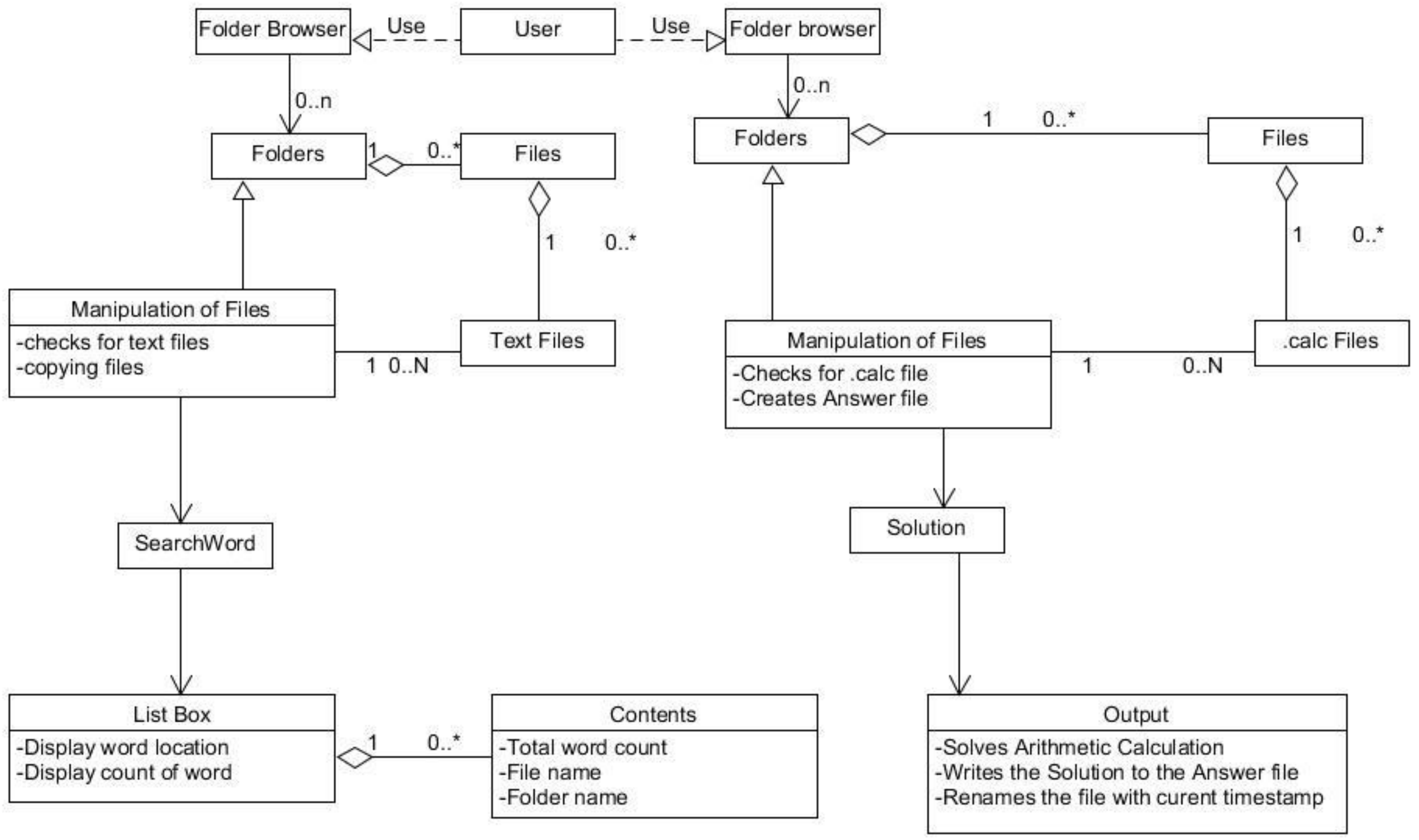
A Tracking methodology was adopted to monitor each other’s activity. The tracking of the requirements is done by monitoring who works on the requirements and their completion status at initial phase, halfway phase and final phase. This tracking system gave us proper segregation of task management as well as systematic approach for completion of our assignment.

Requirnments(Initial Phase)	***Arden***	***Paul***
Requirnment 1: allow user to select derectories	🟩	
Requirnment 2:create a "sorted" file	🟩	
Requirnment 3: Sort the OG file contents and store in sorted file		🟩
Requirnment 4: sort fulfilling all the requirments.		🟧
Requirnment 5:handling financial values and temp.		🟥
Requirnment 6		
6.A: List all directories that have had text files sorted.	🟥	
6.B:List all files that have been sorted for a given directory.		
6.C:List all files that have been sorted for all directories.		
6.D: search sorted files by keywords.		
Requirnment 7: calculations		🟥
Requirnment 8: Edge cases		
Requirnment 9:support n number of calculations.		
Requirnment 10: store in .**answ**		
🟥 ***Started***		
🟧 ***Half done***		
🟩 ***Completed***		

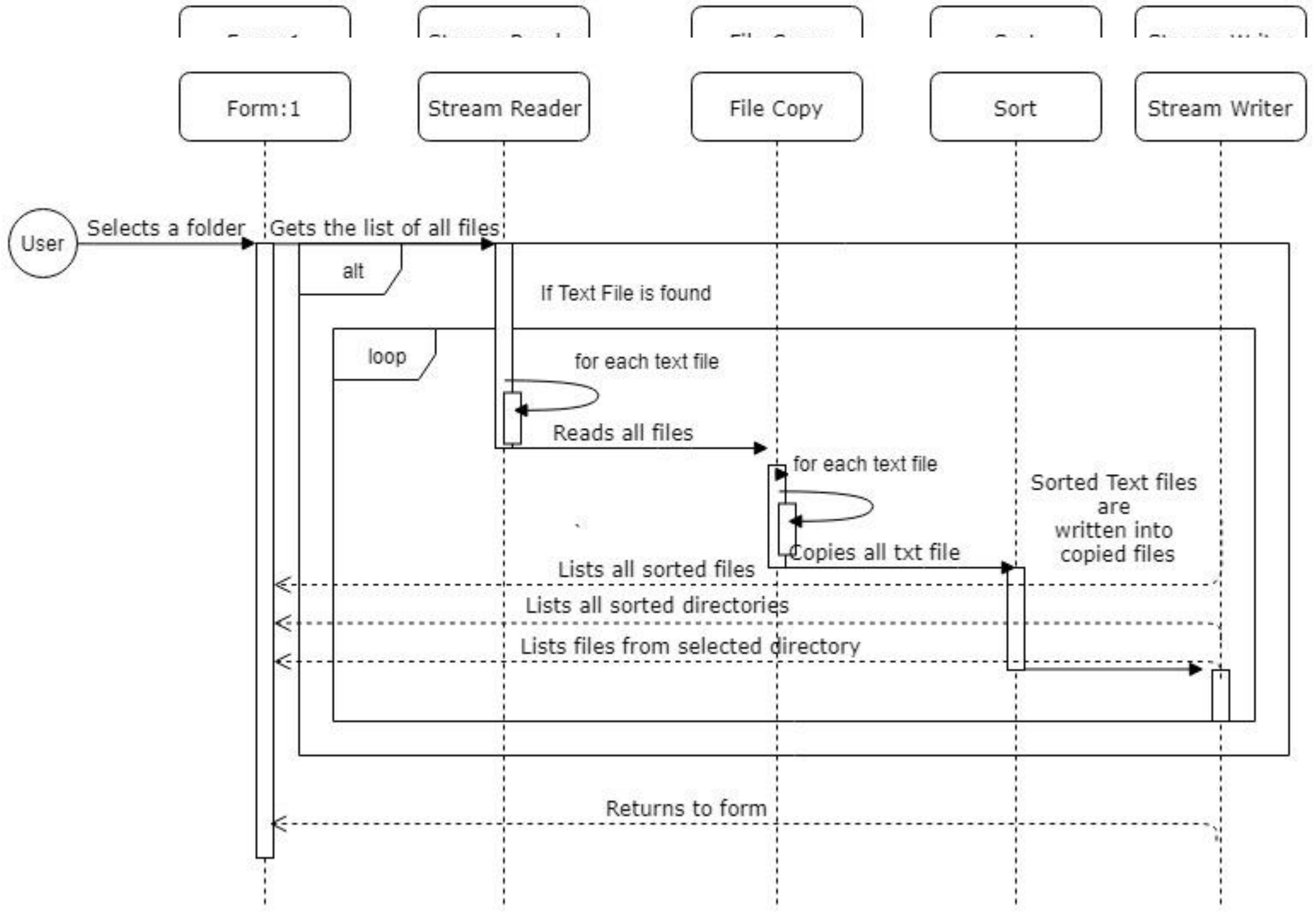
Requirnments(half way Phase)	***Arden***	***Paul***
Requirnment 1: allow user to select derectories	🟩	
Requirnment 2:create a "sorted" file	🟩	
Requirnment 3: Sort the OG file contents and store in sorted file		🟩
Requirnment 4: sort fulfilling all the requirments.		🟩
Requirnment 5:handling financial values and temp.		🟧
Requirnment 6		
6.A: List all directories that have had text files sorted.	🟩	
6.B:List all files that have been sorted for a given directory.	🟥	
6.C:List all files that have been sorted for all directories.		
6.D: search sorted files by keywords.		
Requirnment 7: calculations	🟥	🟧
Requirnment 8: Edge cases		
Requirnment 9:support n number of calculations.		
Requirnment 10: store in .**answ**		
🟥 ***Started***		
🟧 ***Half done***		
🟩 ***Completed***		

Requirnments(final Phase)	***Arden***	***Paul***
Requirnment 1: allow user to select derectories	🟩	
Requirnment 2:create a "sorted" file	🟩	
Requirnment 3: Sort the OG file contents and store in sorted file		🟩
Requirnment 4: sort fulfilling all the requirments.		🟩
Requirnment 5:handling financial values and temp.		🟧
Requirnment 6		
6.A: List all directories that have had text files sorted.	🟩	
6.B:List all files that have been sorted for a given directory.	🟩	
6.C:List all files that have been sorted for all directories.	🟩	
6.D: search sorted files by keywords.		🟧
Requirnment 7: calculations	🟩	🟩
Requirnment 8: Edge cases		🟩
Requirnment 9:support n number of calculations.	🟩	🟩
Requirnment 10: store in .**answ**	🟩	🟩
🟥 ***Started***		
🟧 ***Half done***		
🟩 ***Completed***		

CLASS DIAGRAM

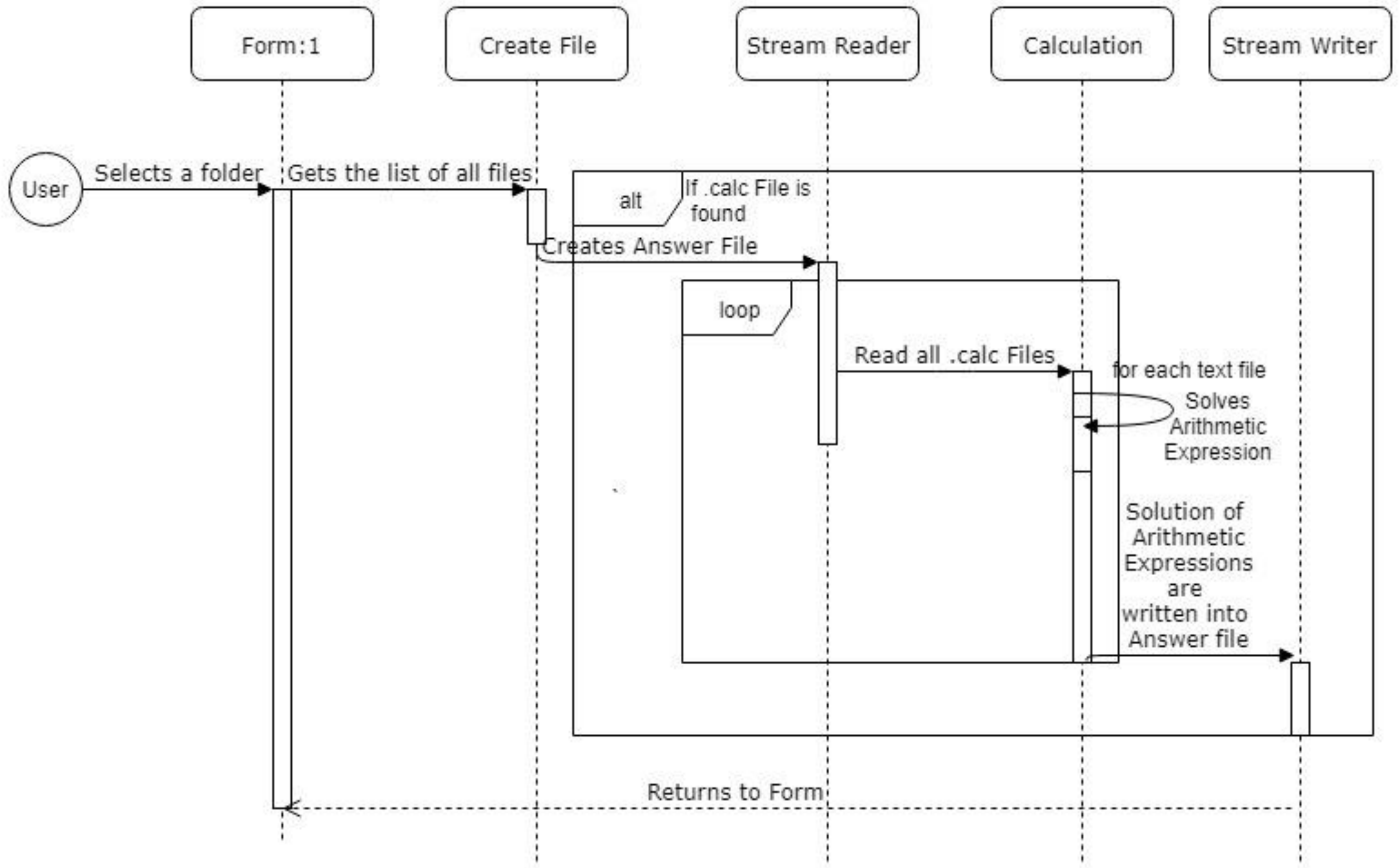


SEQUENCE DIAGRAM A. (Sort File Manipulation)



SEQUENCE DIAGRAM B.

(Evaluation of Arithmetic Expressions)



ASSIGNMENT SPECIFICATION: WHO DID WHAT -

Create a Windows Forms (Design Arden/Paul) application that satisfies the following requirements:

Requirements 1:

Allow the user to select directories on their computers. (Arden)

Requirements 2:

For each text file located in a selected directory, you must create another text file that must have the same name as the original, but with the word "sorted" prepended to the file name. (E.g., if the original file was called mydiary.txt, the copy will be called sortedmydiary.txt). (Arden)

Requirements 3:

This "sorted" text file's contents are the words located in the original text file in alphabetical order, with one word/number appearing per line. Any numerical values must be stored in numerical order at the start of the file. In addition,
if a given word appears more than once, the frequency count should be stored alongside the entry. (Paul)

Requirements 4:

Please be aware of sort order. ASCII sorting will place words starting with capital letters ahead of those with lower case. However, you should be sorting in the order a typical dictionary sorts in. (Paul)

Requirements 5:

Other things you will need to consider are: handling financial values: €2.75, temperatures (21.6°C). (Paul)

Requirements 6:

Your system must keep a track of all directories that were selected by the user and all sorted files—where they are and what they are called. This is so that users can ask the system to: (Arden/Paul)

- **List all directories that have had text files sorted. (Arden)**
- **List all files that have been sorted for a given directory. (Arden)**
- **List all files that have been sorted for all directories. (Arden)**

- **Allow the user to search sorted files for a given word or number. The system should display all files that contain the searched for word along with the number of times the given word or number is displayed. (Paul)**

Requirements 7:

The second piece of major functionality that your application must support is the ability to process a text file with several calculations present and your application should carry out the calculations listed and produce an output file.

Note: all calculations should be rounded to 2 decimal places. (Arden/Paul)

Requirements 8:

There are edge cases you will need to consider for the calculator. I will not list them here—you should be able to identify these and discuss them in your report.

Requirements 9:Your calculation should support the ability to process any number of calculation files located in a given directory. These files should be identified by the extension .calc. Your answers should be output to a file whose name is the current data and time with an extension of .answ. (Paul/ Arden) Requirements 10:

If a user selects the calculator functionality of your application and selects a directory that contains 3 files with the .calc extension, your application will process each .calc file in turn, but will only produce one .answ file containing the calculations and their respective answers. (Paul/ Arden)

LIMITATIONS

These are the Limitations of our application:

We tried to perfect the few things such as

- handling special values such as financial and temperature.
- The search functionality which allows the user to search sorted files for a given word or number.

These functionality were not able to fully justify the requirements of the assignment.

CONCLUSION

We fulfilled the assignment brief requirements by dividing the work between each individual member. Firstly, every individual in the group tried to tackle the requirements by themselves and tried to implement the code which then was optimised by other person. Most of the requirements were completed by both the members together. This assignment tested our programming skills, which enabled us to learn and implement the windows form application in C#.

We referred some external sources such as books and websites to complete the assignment in the best way possible. Tracking and monitoring system was created in 'spreadsheet' to collect the requirements and assigned it to a team member and to track and monitor every individual activity through various phases. The testing was done with various test cases on completion of every functionality. The application built, satisfies the requirements of the assignments.