**KOLEGJI UNIVERSUM**

****

Programi Shkenca Kompjuterike / Viti I / Semestri 2

Lënda: Hyrje ne Struktura te te Dhenave

Chapter 15 – Fundamentals of Computer Programming with C#

Studentja: Prof.Dr:

Elsira Kabashi Muzafer Shala

Asistenti: Laberion Zebica

Prishtinë 2021

1.     Write a program that reads a text file and **prints its odd lines** on the console.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

class Program

{

static void Main(string[] args)

{

string path = "../../OddLines.txt";

Console.WriteLine("> Prints odd lines of text file...\n");

using (StreamReader reader = new StreamReader(path))

{

int lineCount = 1;

while (!reader.EndOfStream)

{

string line = reader.ReadLine();

if (lineCount++ % 2 != 0) Console.WriteLine(line);

}

}

}

}

2.     Write a program that **joins two text files** and records the results in a third file.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp2

{

class Program

{

static void Main(string[] args)

{

string pathText1 = "../../text1.txt";

string pathText2 = "../../text2.txt";

string pathResult = "../../result.txt";

ReadFileContent(pathText1, pathResult);

ReadFileContent(pathText2, pathResult);

PrintResultContent(pathResult);

}

static void ReadFileContent(string pathText, string pathResult)

{

using (StreamWriter result = new StreamWriter(pathResult, true))

{

using (StreamReader reader = new StreamReader(pathText))

{

while (!reader.EndOfStream) result.WriteLine(reader.ReadLine());

}

}

}

static void PrintResultContent(string path)

{

Console.WriteLine("> Result:\n");

using (StreamReader reader = new StreamReader(path))

{

while (!reader.EndOfStream) Console.WriteLine(reader.ReadLine());

}

Console.WriteLine("\n> End of file...\n");

}

}

}

3.     Write a program that reads the contents of a text file and **inserts the line numbers** at the beginning of each line, then rewrites the file contents.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp3

{

class Program

{

static void Main(string[] args)

{

string pathText = "../../text.txt";

string pathResult = "../../result.txt";

InsertLinesToResultFile(pathText, pathResult);

PrintResultContent(pathResult);

}

static void InsertLinesToResultFile(string pathText, string pathResult)

{

int lineCount = 1;

using (StreamWriter result = new StreamWriter(pathResult))

{

using (StreamReader reader = new StreamReader(pathText))

{

while (!reader.EndOfStream) result.WriteLine("Line {0}: {1}", lineCount++, reader.ReadLine());

}

}

}

static void PrintResultContent(string path)

{

Console.WriteLine("> Result:\n");

using (StreamReader reader = new StreamReader(path))

{

while (!reader.EndOfStream) Console.WriteLine(reader.ReadLine());

}

Console.WriteLine("\n> End of file...\n");

}

}

}

4.     Write a program that **compares two text files** with the same number of rows line by line and prints the number of equal and the number of different lines.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp4

{

class Program

{

static void Main(string[] args)

{

static int sameLinesCount = 0, diffLinesCount = 0;

static void Main()

{

string pathText1 = "../../text1.txt";

string pathText2 = "../../text2.txt";

CompareTextFiles(pathText1, pathText2);

Console.WriteLine("Same lines: {0}", sameLinesCount);

Console.WriteLine("Different lines: {0}\n", diffLinesCount);

}

static void CompareTextFiles(string pathText1, string pathText2)

{

using (StreamReader reader1 = new StreamReader(pathText1))

{

using (StreamReader reader2 = new StreamReader(pathText2))

{

// Text files have equal number of lines

while (!reader1.EndOfStream || !reader2.EndOfStream)

{

string line1 = reader1.ReadLine();

string line2 = reader2.ReadLine();

if (line1.CompareTo(line2) == 0) sameLinesCount++;

else diffLinesCount++;

}

}

}

5.     Write a program that reads a square matrix of integers from a file and **finds the sub-matrix with size 2 × 2 that has the maximal sum** and writes this sum to a separate text file. The first line of input file contains the size of the recorded matrix (N). The next N lines contain N integers separated by spaces.

Sample input file:

|  |
| --- |
| 4  2 3 3 4  0 2 3 4  **3 7** 1 2  **4 3** 3 2 |

Sample output: 17.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp5

{

class Program

{

static void Main(string[] args)

{

static int[,] matrix;

static int bestSum = 0;

static void Main()

{

string pathMatrix = "../../matrix.txt";

string pathResult = "../../result.txt";

InitializeMatrix(pathMatrix);

FindBestSumOfSquareInMatrix();

WriteBestSumToTextFile(pathResult);

Console.WriteLine("Best sum of square: {0}\n", bestSum);

}

static void InitializeMatrix(string pathMatrix)

{

using (StreamReader reader = new StreamReader(pathMatrix))

{

int row = 0;

int length = int.Parse(reader.ReadLine());

matrix = new int[length, length];

while (!reader.EndOfStream)

{

string[] tokens = reader.ReadLine().Split(new char[] { ' ' },

StringSplitOptions.RemoveEmptyEntries);

for (int i = 0; i < tokens.Length; i++)

matrix[row, i] = int.Parse(tokens[i]);

row++;

}

}

}

static void FindBestSumOfSquareInMatrix()

{

for (int row = 0; row < matrix.GetLongLength(0) - 1; row++)

{

int currentSum = 0;

for (int col = 0; col < matrix.GetLongLength(1) - 1; col++)

{

currentSum = matrix[row, col] + matrix[row, col + 1] +

matrix[row + 1, col] + matrix[row + 1, col + 1];

if (currentSum > bestSum) bestSum = currentSum;

}

}

}

static void WriteBestSumToTextFile(string pathResult)

{

using (StreamWriter result = new StreamWriter(pathResult))

{

result.WriteLine(bestSum);

}

}

}

}

}

6.     Write a program that **reads a list of names** from a text file, arranges them in **alphabetical order**, and writes them to another file. The lines are written one per row.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp6

{

class Program

{

static List<string> strings = new List<string>();

static void Main()

{

string pathText = "../../strings.txt";

string pathResult = "../../result.txt";

SeparateStringsFromTextFile(pathText);

Console.WriteLine("Strings: {0}", string.Join(", ", strings));

strings.Sort();

SaveSortedStringsToTextFile(pathResult);

Console.WriteLine("\nSorted strings: {0}\n", string.Join(", ", strings));

}

static void SeparateStringsFromTextFile(string pathText)

{

using (StreamReader reader = new StreamReader(pathText))

{

while (!reader.EndOfStream)

{

string[] tokens = reader.ReadLine().Split(new char[] { ' ', ',', '\n' },

StringSplitOptions.RemoveEmptyEntries);

for (int i = 0; i < tokens.Length; i++)

strings.Add(tokens[i]);

}

}

}

static void SaveSortedStringsToTextFile(string pathResult)

{

using (StreamWriter result = new StreamWriter(pathResult))

{

for (int i = 0; i < strings.Count; i++)

result.WriteLine(strings[i]);

}

}

}

}

7.     Write a program that **replaces every occurrence of the substring** "**start**" with "**finish**" in a text file. Can you rewrite the program to replace whole words only? Does the program work for large files (e.g. 800 MB)?

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp7

{

class Program

{

static void Main(string[] args)

{

string pathText = "../../text.txt";

string pathResult = "../../result.txt";

ReplaceSubstrings(pathText, pathResult);

}

static void ReplaceSubstrings(string pathText, string pathResult)

{

using (StreamWriter result = new StreamWriter(pathResult))

{

using (StreamReader reader = new StreamReader(pathText))

{

while (!reader.EndOfStream) result.WriteLine(reader.ReadLine().Replace("start", "finish"));

}

}

}

}

}

8.     Write the previous program so that it changes only the **whole words** (not parts of the word).

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp8

{

class Program

{

static void Main(string[] args)

{

string pathText = "../../text.txt";

string pathResult = "../../result.txt";

ReplaceAllWholeWords(pathText, pathResult);

}

static void ReplaceAllWholeWords(string pathText, string pathResult)

{

using (StreamWriter result = new StreamWriter(pathResult))

{

using (StreamReader reader = new StreamReader(pathText))

{

while (!reader.EndOfStream)

result.WriteLine(Regex.Replace(reader.ReadLine(), @"\bstart\b", "finish"));

}

}

}

}

9.     Write a program that **deletes all the odd lines** of a text file.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp9

{

class Program

{

static string pathText = "../../text.txt";

static void Main()

{

WriteOddLines(ReadOddLines());

}

static List<string> ReadOddLines()

{

List<string> oddLines = new List<string>();

int lineCount = 1;

using (StreamReader reader = new StreamReader(pathText))

{

while (!reader.EndOfStream)

{

string line = reader.ReadLine();

if (lineCount++ % 2 == 0) oddLines.Add(line);

}

}

return oddLines;

}

static void WriteOddLines(List<string> oddLines)

{

using (StreamWriter result = new StreamWriter(pathText))

{

for (int i = 0; i < oddLines.Count; i++)

result.WriteLine(oddLines[i]);

}

}

}

}

10.   Write a program that extracts from an XML file the **text only** (without the tags). Sample input file:

|  |
| --- |
| <?xml version="1.0"><student><name>**Peter**</name> <age>**21**</age><interests count="3"><interest> **Games**</interest><interest>**C#**</interest><interest> **Java**</interest></interests></student> |

Sample output:

|  |
| --- |
| Peter  21  Games  C#  Java |

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp10

{

class Program

{

static StringBuilder textWithoutTags = new StringBuilder();

static void Main()

{

string pathXML = "../../text.xml";

ExtractTextWithoutTags(pathXML);

Console.WriteLine(textWithoutTags);

}

static void ExtractTextWithoutTags(string pathTextFile)

{

using (StreamReader reader = new StreamReader(pathTextFile))

{

while (!reader.EndOfStream)

{

string line = Regex.Replace(reader.ReadLine(), @"<[^>]\*>", String.Empty).Trim();

if (line != "") textWithoutTags.AppendLine(line);

}

}

}

}

}

11.   Write a program that **deletes all words** that begin with the word "**test**". The words will contain only the following chars: 0…9, a…z, A…Z.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp11

{

class Program

{

static void Main(string[] args)

{

string pathText = "../../text.txt";

string pathResult = "../../result.txt";

ExtractTextWithoutTags(pathText, pathResult);

}

static void ExtractTextWithoutTags(string pathText, string pathResult)

{

using (StreamWriter result = new StreamWriter(pathResult))

{

using (StreamReader reader = new StreamReader(pathText))

{

while (!reader.EndOfStream)

{

string line = Regex.Replace(reader.ReadLine(), @"\btest\S\*", String.Empty).Trim();

result.Write(line);

}

}

}

}

}

}

12.   A text file **words.txt** is given, containing a list of words, one per line. Write a program that **deletes in the file text.txt all the words that occur in the other file**. Catch and handle all possible exceptions.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp12

{

class Program

{

static List<string> blackList = new List<string>();

static void Main()

{

try

{

string pathText = "../../text.txt";

string pathBlackList = "../../blacklist.txt";

string pathResult = "../../result.txt";

GetBlackWords(pathBlackList);

ExtractTextWithoutTags(pathText, pathResult);

}

catch (DriveNotFoundException driveError)

{

PrintErrorMessage(driveError);

}

catch (DirectoryNotFoundException directoryError)

{

PrintErrorMessage(directoryError);

}

catch (EndOfStreamException eose)

{

PrintErrorMessage(eose);

}

catch (FileNotFoundException fnfe)

{

PrintErrorMessage(fnfe);

}

catch (FileLoadException fle)

{

PrintErrorMessage(fle);

}

catch (PathTooLongException ptle)

{

PrintErrorMessage(ptle);

}

}

static void PrintErrorMessage(Exception error)

{

Console.Error.WriteLine("-> Error! {0}\n", error.Message);

}

static void GetBlackWords(string pathBlackList)

{

using (StreamReader reader = new StreamReader(pathBlackList))

{

while (!reader.EndOfStream)

{

string[] tokens = reader.ReadLine().Split(new char[] { ' ', ',', '\n' },

StringSplitOptions.RemoveEmptyEntries);

for (int i = 0; i < tokens.Length; i++)

if (!blackList.Contains(tokens[i])) blackList.Add(tokens[i]);

}

}

}

static void ExtractTextWithoutTags(string pathText, string pathResult)

{

using (StreamWriter result = new StreamWriter(pathResult))

{

using (StreamReader reader = new StreamReader(pathText))

{

while (!reader.EndOfStream)

{

string line = reader.ReadLine();

for (int i = 0; i < blackList.Count; i++)

line = Regex.Replace(line, "\\b" + blackList[i] + "\\b", String.Empty);

result.WriteLine(line);

}

}

}

}

}

13.   Write a program that **reads a list of words** from a file called **words.txt**, **counts how many times each of these words is found in another file** **text.txt**, and records the results in a third file – **result.txt**, but before that, sorts them by the number of occurrences in descending order. Handle all possible exceptions.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp13

{

class Program

{

static Dictionary<string, int> wordOccurs = new Dictionary<string, int>();

static void Main()

{

try

{

string pathText = "../../test.txt";

string pathSearchedWords = "../../words.txt";

string pathResult = "../../result.txt";

GetSearchedWords(pathSearchedWords);

GetWordOccurs(pathText, pathResult);

}

catch (DriveNotFoundException driveError)

{

PrintErrorMessage(driveError);

}

catch (DirectoryNotFoundException directoryError)

{

PrintErrorMessage(directoryError);

}

catch (EndOfStreamException eose)

{

PrintErrorMessage(eose);

}

catch (FileNotFoundException fnfe)

{

PrintErrorMessage(fnfe);

}

catch (FileLoadException fle)

{

PrintErrorMessage(fle);

}

catch (PathTooLongException ptle)

{

PrintErrorMessage(ptle);

}

catch (InvalidOperationException ioe)

{

PrintErrorMessage(ioe);

}

}

static void PrintErrorMessage(Exception error)

{

Console.Error.WriteLine("-> Error! {0}\n", error.Message);

}

static void GetSearchedWords(string pathSearchedWords)

{

using (StreamReader reader = new StreamReader(pathSearchedWords))

{

while (!reader.EndOfStream)

{

string[] tokens = reader.ReadLine().Split(new char[] { ' ', ',', '\n' },

StringSplitOptions.RemoveEmptyEntries);

for (int i = 0; i < tokens.Length; i++)

if (!wordOccurs.ContainsKey(tokens[i])) wordOccurs.Add(tokens[i], 0);

}

}

}

static void GetWordOccurs(string pathText, string pathResult)

{

string allContent = string.Empty;

using (StreamWriter result = new StreamWriter(pathResult))

{

using (StreamReader reader = new StreamReader(pathText))

{

allContent = reader.ReadToEnd();

for (int i = 0; i < wordOccurs.Count; i++)

{

KeyValuePair<string, int> word = wordOccurs.ElementAt(i);

int index = allContent.IndexOf(word.Key);

while (index != -1)

{

wordOccurs[word.Key]++;

index = allContent.IndexOf(word.Key, index + 1);

}

}

}

WriteWordOccursToFile(result);

}

}

static void WriteWordOccursToFile(StreamWriter result)

{

foreach (KeyValuePair<string, int> word in wordOccurs.OrderByDescending(key => key.Value))

result.WriteLine(word.Key + " -> " + word.Value + " time(s).");

}

}

}

}