using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

namespace Homework31.\_10.\_23

{

//4

/\* class Enterprise

{

string name;

DateTime foundationDate;

int nWorkers;

public Enterprise(string name, DateTime foundationDate, int nWorkers)

{

this.name = name;

this.foundationDate = foundationDate;

this.nWorkers = nWorkers;

}

public string Name { get => name; set => name = value; }

public DateTime FoundationDate { get => foundationDate; set => foundationDate = value; }

public int NWorkers { get => nWorkers; set => nWorkers = value; }

public override string ToString()

{

return string.Format($"name:{name}, foundationDate:{foundationDate.ToShortDateString()}, nWorkers:{nWorkers}");

}

}

class ProccesEnterprise

{

public string[] GetInfo(Enterprise[] A)

{

string[] s = (from n in A

select n.ToString()).ToArray();

return s;

}

public string[] GetCountWorkers(Enterprise[] A)

{

string[] s = (from n in A

where n.NWorkers > 100

select n.Name).ToArray();

return s;

}

}

\*/

delegate void TOutput();

interface IWord

{

string GetWord();

int GetWordCount { get; }

}

enum DelimiterChars

{

Space = ' ', // Пробіл

Dot = '.', // крапка

Semicolon = ';', // крапка з комою

NewLine = '\n' // новий рядок

}

class WordFrequence : IWord,IComparable

{

string word;

int count;

public WordFrequence(string word)

{

this.word = word;

count= 0;

}

public string GetWord()

{

return word;

}

public int GetWordCount

{

get=>count;

}

public override string ToString()

{

return $"{word}:{count}";

}

public static WordFrequence operator ++(WordFrequence e)

{

e.count++;

return e;

}

public static WordFrequence operator --(WordFrequence e)

{

e.count--;

return e;

}

public int CompareTo(object obj)

{

if (obj is WordFrequence)

{

if (count > ((WordFrequence)obj).GetWordCount) //у меня сортуеться за спаданням

return -1;

else if (count == ((WordFrequence)obj).GetWordCount)

return 0;

else

return 1;

}

else

throw new ArgumentException();

}

}

class DictionaryFrequence

{

Dictionary<string, int> AW;

public DictionaryFrequence()

{

AW= new Dictionary<string, int>();

}

public void ReadFromFile(string name)

{

using(FileStream f = new FileStream(name, FileMode.Open, FileAccess.Read))

{

using(StreamReader r = new StreamReader(f))

{

if (File.Exists(name)) {

string buff ="";

while (r.Peek() != -1) {

buff += r.ReadToEnd();

}

DelimiterChars[] values = (DelimiterChars[])Enum.GetValues(typeof(DelimiterChars));

char[] chars = (from i in values

select (char)i).ToArray();

string[] ar = buff.Split(chars);

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

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

{

string currobj = ar[i];

int count = 0;

for (int j = 0; j < ar.Length; j++)

{

if (currobj == ar[j])

count++;

}

if (i == 0)

{

AW.Add(currobj, count);

used.Add(currobj);

}

else

{

bool flag = true;

foreach (var curr in used)

if (curr == currobj)

flag = false;

if (flag)

{

AW.Add(currobj, count);

used.Add(currobj);

}

}

count = 0;

}

}

}

}

}

public void SaveResultsToFile(string name)

{

using(FileStream f = new FileStream(name, FileMode.Create,FileAccess.Write)) {

using (StreamWriter w = new StreamWriter(f)) {

foreach (var currobj in AW)

{

w.WriteLine( string.Format($"{currobj.Key} : {currobj.Value}"));

}

}

}

}

public int this[string key]

{

get

{

if (AW.ContainsKey(key))

{

return AW[key];

}

else

{

Console.WriteLine("KEY DOESN`t EXIST");

return -1;

}

}

}

public void Print()

{

TOutput output = () =>

{

foreach (var currobj in AW)

{

Console.WriteLine(currobj.Key + " : " + currobj.Value);

}

};

output.Invoke();

}

public WordFrequence[] ToWordFrequence()

{

WordFrequence[] cur = new WordFrequence[AW.Count];

int i = 0;

foreach (var currobj in AW)

{

cur[i] = new WordFrequence(currobj.Key);

for(int j = 0; j < currobj.Value; j++)

{

cur[i]++;

}

i++;

}

return cur;

}

public override string ToString()

{

string str = "";

TOutput output = () =>

{

foreach (var currobj in AW)

{

str += string.Format($"{currobj.Key} : {currobj.Value}");

}

};

output.Invoke();

return str;

}

}

class ProcessDictionary

{

public void SortAlphabet(WordFrequence[] WF)

{

WF = (from i in WF

orderby i.GetWord()

select i).ToArray();

}

public WordFrequence[] SortFrequence(DictionaryFrequence df)

{

WordFrequence[]f = df.ToWordFrequence();

Array.Sort(f);

return f;

}

public void SortAlphabetLINQ(WordFrequence[] WF)

{

WF = (from i in WF

orderby i.GetWord() descending

select i).ToArray();

}

}

internal class Program

{

static void Main(string[] args)

{

/\*Enterprise[] enterprise =

{

new Enterprise("DevilStudio",new DateTime(2000,8,13),125),

new Enterprise("Angel2Free",new DateTime(1988,4,22),1000),

new Enterprise("GGG",new DateTime(2020,4,21),15),

new Enterprise("GGG222222",new DateTime(2021,4,21),5),

};

ProccesEnterprise proccesEnterprise = new ProccesEnterprise();

Console.WriteLine(string.Join("\n",proccesEnterprise.GetInfo(enterprise)));

Console.WriteLine("-----------------");

Console.WriteLine(string.Join("\n", proccesEnterprise.GetCountWorkers(enterprise)));\*/

DictionaryFrequence d = new DictionaryFrequence();

d.ReadFromFile("file.txt");

d.SaveResultsToFile("file2.txt");

ProcessDictionary processDictionary = new ProcessDictionary();

WordFrequence[] word;

word = processDictionary.SortFrequence(d);

Console.WriteLine(word[0]);

}

}

}