**ASSESSMENT 14-04-2023**

**Write C# code for the following problems.**

**Use OOP concepts wherever applicable. EXCEPTIONS to be provided for CONSTRAINTS given (User**

**defined / System defined).**

**1. Write a program to**

**a. create and write some line of text into a file which does not contain a given set of phrase in a line.**

**b. Read a specific line from a file**

using Basic\_Program;

using System.Net.Http.Headers;

class Demo1

{

public static void Main(string[] args)

{

FileOperations fileOperations = new FileOperations();

if (fileOperations.createfile()==false)

{

Console.WriteLine("The file exists/created and operable");

}

else

{

Console.WriteLine("The file is not operable");

}

fileOperations.writingToFile();

fileOperations.readFromFile();

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Basic\_Program

{

internal class FileOperations

{

public bool createfile()

{

/\*FileInfo f1 = new FileInfo("D:\\C# training\\Sample2.txt");

if (f1.Exists)

{

Console.WriteLine("File Exists");

Console.WriteLine(f1.IsReadOnly);

}\*/

bool Locked = false;

try

{

FileStream fs = File.Open("D:\\C# training\\Sample2.txt", FileMode.OpenOrCreate,FileAccess.ReadWrite, FileShare.None);

fs.Close();

}

catch (IOException ex)

{

Locked = true;

}

finally

{

Console.WriteLine("Constraints has been checked");

}

return Locked;

}

public void writingToFile()

{

FileStream fs = new FileStream("D:\\C# training\\Sample2.txt", FileMode.Open, FileAccess.Write);

StreamWriter sw = new StreamWriter(fs);

Console.WriteLine("Input the string to ignore the line");

string word = Console.ReadLine();

Console.WriteLine("Input number of lines to write in the file");

int num = Convert.ToInt32(Console.ReadLine());

Console.WriteLine($"Input {num} strings below :");

for (int i=0;i<num;i++)

{

Console.WriteLine($"Input line {i+1} :");

string sentence = Console.ReadLine();

if (!sentence.Contains(word))

{

sw.WriteLine(sentence);

}

}

sw.Flush();

sw.Close();

fs.Close();

}

public void readFromFile()

{

Console.WriteLine("Enter the line number to read a specific line from the file");

int l = Convert.ToInt32(Console.ReadLine());

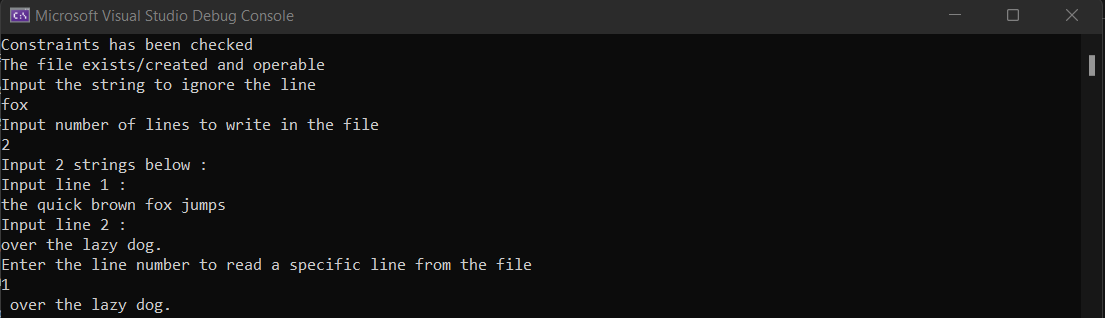
string[] lines = File.ReadAllLines("D:\\C# training\\Sample2.txt");

Console.WriteLine(" {0}", lines[l - 1]);

}

}

}



**2. Given a string, str = s1,s2,...sn , consisting of lowercase English characters (a-z), remove all of the**

**characters that occurred previously in the string.**

using Basic\_Program;

using System.Net.Http.Headers;

class Demo1

{

public static void Main(string[] args)

{

stringDuplication stringDuplication = new stringDuplication();

stringDuplication.findDuplicate();

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Basic\_Program

{

internal class stringDuplication

{

public void findDuplicate()

{

Console.Write("Enter a String : ");

string inputString = Console.ReadLine();

string resultString = string.Empty;

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

{

if (!resultString.Contains(inputString[i]))

{

resultString += inputString[i];

}

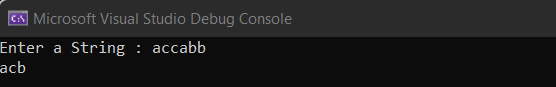
}

Console.WriteLine(resultString);

}

}

}



**3. Joseph and Jane are making a contest for apes. During the process, they have to communicate frequently with each other. Since they are not completely human, they cannot speak properly. They have to transfer messages using postcards of small sizes. To save space on the small postcards, they devise a string compression algorithm.**

using Basic\_Program;

using System.Net.Http.Headers;

class Demo1

{

public static void Main(string[] args)

{

Consecutives consecutives= new Consecutives();

Console.WriteLine("Enter the string : ");

string input = Console.ReadLine();

consecutives.display\_consecutives(input);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Basic\_Program

{

internal class Consecutives

{

public void display\_consecutives(string input)

{

string output = "";

int count = 1;

for (int i = 1; i < input.Length; i++)

{

if (input[i] == input[i - 1])

{

count++;

}

else

{

output += input[i - 1] + count.ToString();

count = 1;

}

}

output += input[input.Length - 1] + count.ToString();

Console.WriteLine("Output : ");

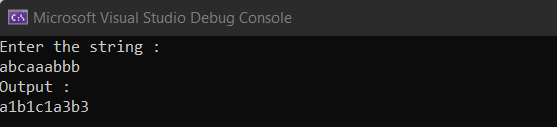
Console.WriteLine(output.ToLower());

}

}

}

OUTPUT:



**4. You are in charge of data transfer between two Data Centers. Each set of data is represented by a pair of strings. Over a period of time you have observed a trend: most of the times both strings share some prefix. You want to utilize this observation to design a data compression algorithm which will be used to reduce amount of data to be transferred.**

using Basic\_Program;

using System.Net.Http.Headers;

class Demo1

{

public static void Main(string[] args)

{

string s1 = Console.ReadLine();

string s2 = Console.ReadLine();

DataTransfer dt = new DataTransfer();

dt.process(s1, s2);

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Basic\_Program

{

internal class DataTransfer

{

int length;

int index;

public void process(string s1, string s2)

{

int prefixLen = 0;

for (int i = 0; i < Math.Min(s1.Length, s2.Length); i++)

{

if (s1[i] == s2[i])

{

prefixLen++;

}

else

{

break;

}

}

string compressedS1 = s1.Substring(prefixLen);

string compressedS2 = s2.Substring(prefixLen);

Console.WriteLine($"{prefixLen} {s1.Substring(0, prefixLen)}");

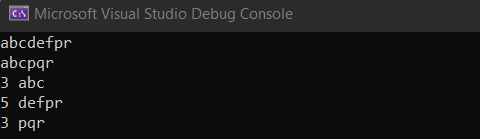
Console.WriteLine($"{compressedS1.Length} {compressedS1}");

Console.WriteLine($"{compressedS2.Length} {compressedS2}");

}

}

}



**5. You are given a sequence of balls in 4 colors: red, green, yellow and blue.** **The sequence is full of colors if and only if all of the following conditions are true:**

**• There are as many red balls as green balls.**

**• There are as many yellow balls as blue balls.**

**• Difference between the number of red balls and green balls in every prefix of the sequence is at most 1.**

**• Difference between the number of yellow balls and blue balls in every prefix of the sequence is at most 1.**

**Your task is to write a program, which for a given sequence prints True if it is full of colors, otherwise it prints False.**

using System;

class Program

{

static bool IsFullOfColors(string sequence)

{

int redCount = 0;

int greenCount = 0;

int yellowCount = 0;

int blueCount = 0;

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

{

switch (sequence[i])

{

case 'R':

redCount++;

break;

case 'G':

greenCount++;

break;

case 'Y':

yellowCount++;

break;

case 'B':

blueCount++;

break;

}

if (Math.Abs(redCount - greenCount) > 1 || Math.Abs(yellowCount - blueCount) > 1)

{

return false;

}

}

return redCount == greenCount && yellowCount == blueCount;

}

static void Main(string[] args)

{

int n = int.Parse(Console.ReadLine());

for (int i = 0; i < n; i++)

{

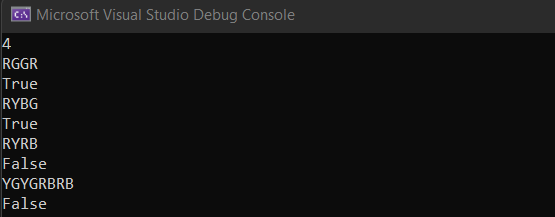
string sequence = Console.ReadLine();

Console.WriteLine(IsFullOfColors(sequence) ? "True" : "False");

}

}

}



**6. We define super digit of an integer using the following rules:**

**• If has only digit, then its super digit is .**

**• Otherwise, the super digit of is equal to the super digit of the digit-sum of . Here, digit-sum of a number is defined as the sum of its digits.**

using System;

class Program

{

static int SuperDigit(long n)

{

if (n < 10)

{

return (int)n;

}

else

{

long digitSum = 0;

while (n > 0)

{

digitSum += n % 10;

n /= 10;

}

return SuperDigit(digitSum);

}

}

static void Main(string[] args)

{

string[] inputs = Console.ReadLine().Split(' ');

long n = long.Parse(inputs[0]);

int k = int.Parse(inputs[1]);

long digitSum = 0;

while (n > 0)

{

digitSum += n % 10;

n /= 10;

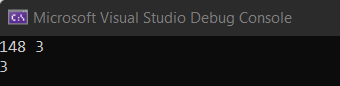
}

digitSum \*= k;

Console.WriteLine(SuperDigit(digitSum));

}

}



**7. There are N inmates numbered between [1, N] in a prison. These inmates have superhuman strength because they have drunk a special concoction made by Dr. Evil. They have to be transported by some buses to a new facility. But they are bound by special chains which are made from strong carbon fibres. Each inmate is either chained alone or is chained in a group along with one or more inmates. A group of inmates are those who are directly or indirectly connected to each other. Only one group can be transported per bus.**

using System;

using System.Collections.Generic;

class Program

{

static int[] parent;

static int[] size;

static int Find(int x)

{

if (parent[x] == x)

{

return x;

}

return parent[x] = Find(parent[x]);

}

static void Union(int x, int y)

{

int rootX = Find(x);

int rootY = Find(y);

if (rootX != rootY)

{

if (size[rootX] < size[rootY])

{

int temp = rootX;

rootX = rootY;

rootY = temp;

}

parent[rootY] = rootX;

size[rootX] += size[rootY];

}

}

static void Main(string[] args)

{

int n = int.Parse(Console.ReadLine());

int m = int.Parse(Console.ReadLine());

parent = new int[n + 1];

size = new int[n + 1];

for (int i = 1; i <= n; i++)

{

parent[i] = i;

size[i] = 1;

}

for (int i = 0; i < m; i++)

{

string[] line = Console.ReadLine().Split();

int x = int.Parse(line[0]);

int y = int.Parse(line[1]);

Union(x, y);

}

int[] groupSize = new int[n + 1];

for (int i = 1; i <= n; i++)

{

groupSize[Find(i)]++;

}

int cost = 0;

int remaining = n;

for (int i = 1; i <= n; i++)

{

if (groupSize[i] > 0)

{

int k = (int)Math.Ceiling(Math.Sqrt(groupSize[i]));

cost += k;

remaining -= k \* k;

}

}

if (remaining > 0)

{

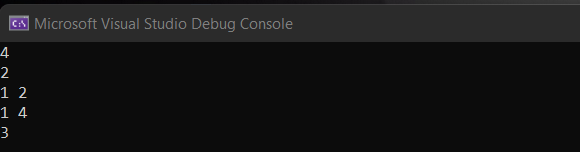
cost += (int)Math.Ceiling((double)remaining / Math.Sqrt(remaining));

}

Console.WriteLine(cost);

}

}



**8. Your country is at war!**

using System;

using System.Collections.Generic;

class Solution

{

static void Main(string[] args)

{

string[] line1 = Console.ReadLine().Split();

int n = int.Parse(line1[0]); // number of armies

int q = int.Parse(line1[1]); // number of events

// initialize the armies with empty soldier lists

List<int>[] armies = new List<int>[n];

for (int i = 0; i < n; i++)

{

armies[i] = new List<int>();

}

// keep track of the soldiers' combat abilities in each army

int[] maxCombat = new int[n];

for (int i = 0; i < n; i++)

{

maxCombat[i] = int.MinValue;

}

// handle each event

for (int i = 0; i < q; i++)

{

string[] line = Console.ReadLine().Split();

int type = int.Parse(line[0]);

if (type == 1)

{ // print maximum combat ability in army

int army = int.Parse(line[1]) - 1; // 0-based indexing

Console.WriteLine(maxCombat[army]);

}

else if (type == 2)

{ // remove soldier with max combat ability

int army = int.Parse(line[1]) - 1; // 0-based indexing

int maxCombatIndex = armies[army].Count - 1;

for (int j = armies[army].Count - 2; j >= 0; j--)

{

if (armies[army][j] > armies[army][maxCombatIndex])

{

maxCombatIndex = j;

}

}

armies[army].RemoveAt(maxCombatIndex);

if (armies[army].Count > 0)

{

maxCombat[army] = armies[army][armies[army].Count - 1];

}

else

{

maxCombat[army] = int.MinValue;

}

}

else if (type == 3)

{ // add soldier with combat ability

int army = int.Parse(line[1]) - 1; // 0-based indexing

int combat = int.Parse(line[2]);

armies[army].Add(combat);

if (combat > maxCombat[army])

{

maxCombat[army] = combat;

}

}

else

{ // merge armies

int army1 = int.Parse(line[1]) - 1; // 0-based indexing

int army2 = int.Parse(line[2]) - 1; // 0-based indexing

armies[army1].AddRange(armies[army2]);

armies[army2] = null; // mark army2 as removed

maxCombat[army1] = Math.Max(maxCombat[army1], maxCombat[army2]);

maxCombat[army2] = int.MinValue;

}

}

}

}

