**C# ASSIGNMENT**

//Question 1

internal class FileOperations

{

public bool createfile()

{

bool operate = false;

try

{

FileStream fs = File.Open("C:\\Users\\kbrka\\OneDrive\\Desktop\\practical training\\C#\\Assignment.txt", FileMode.OpenOrCreate, FileAccess.ReadWrite, FileShare.None);

fs.Close();

}

catch (IOException ex)

{

operate = true;

}

finally

{

Console.WriteLine("Checked Sucessfully");

}

return operate;

}

public void writingToFile()

{

FileStream fs = new FileStream("C:\\Users\\kbrka\\OneDrive\\Desktop\\practical training\\C#\\Assignment.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("Total 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("C:\\Users\\kbrka\\OneDrive\\Desktop\\practical training\\C#\\Assignment.txt");

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

}

}

//Question 2

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);

}

}

//Question 3

internal class Repetition

{

public void Display\_Rep(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());

}

}

//Question 4

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}");

}

}

//Question 5

internal class Sequence

{

public bool Colors(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;

}

}

//Question 6

internal class Digit

{

public int Digits(long n)

{

if (n < 10)

{

return (int)n;

}

else

{

long digitSum = 0;

while (n > 0)

{

digitSum += n % 10;

n /= 10;

}

return Digits(digitSum);

}

}

}

//Question 7

internal class inimates

{

private int[] parent;

private int[] size;

public int[] Parent { get => parent; set => parent = value; }

public int[] Size { get => size; set => size = value; }

public int Find(int x)

{

if (Parent[x] == x)

{

return x;

}

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

}

public 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];

}

}

}

//Question 8

internal class Country

{

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

int n = int.Parse(line1[0]);

int q = int.Parse(line1[1]);

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

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

{

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

}

int[] maxCombat = new int[n];

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

{

maxCombat[i] = int.MinValue;

}

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

{

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

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

if (type == 1)

{

int army = int.Parse(line[1]) - 1;

Console.WriteLine(maxCombat[army]);

}

else if (type == 2)

{

int army = int.Parse(line[1]) - 1;

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)

{

int army = int.Parse(line[1]) - 1;

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

armies[army].Add(combat);

if (combat > maxCombat[army])

{

maxCombat[army] = combat;

}

}

else

{

int army1 = int.Parse(line[1]) - 1;

int army2 = int.Parse(line[2]) - 1;

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

armies[army2] = null;

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

maxCombat[army2] = int.MinValue;

}

}

}

}

**DRIVER CODE :**

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();

stringDuplication stringDuplication = new stringDuplication();

stringDuplication.findDuplicate();

Repetition rep = new Repetition();

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

string input = Console.ReadLine();

rep.Display\_Rep(input);

string s1 = Console.ReadLine();

string s2 = Console.ReadLine();

DataTransfer dt = new DataTransfer();

dt.process(s1, s2);

Sequence seq = new Sequence();

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

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

{

string sequence = Console.ReadLine();

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

}

Digit d=new Digit();

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

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

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

long digitSum = 0;

while (num > 0)

{

digitSum += n % 10;

n /= 10;

}

digitSum \*= k;

Console.WriteLine(d.Digits(digitSum));

inimates ini=new inimates();

int num1 = int.Parse(Console.ReadLine());

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

parent = new int[num1 + 1];

size = new int[num1 + 1];

for (int i = 1; i <= num1; 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]);

ini.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);

}