1)

```
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]);
        }
    }
}
```

```
Constraints has been checked
The file exists/created and operable
Input the string to ignore the line
fox
Input number of lines to write in the file
2
Input 2 strings below :
Input line 1 :
the quick brown fox jumps
Input line 2 :
over the lazy dog.
Enter the line number to read a specific line from the file
1
 over the lazy dog.
```

2)

```
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);
    }
  }
}
```

```
Enter a String : accabb
acb
```

3)

```
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());
    }
  }
}
```

```
Enter the string :
abcaaabbb
Output :
a1b1c1a3b3
```

4)

```
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}");
        }
    }
}
```

```
abcdefpr
abcpqr
3 abc
5 defpr
3 pqr
```

5)

```
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");
    }
  }
}
```

```
4
RGGR
True
RYBG
True
RYRB
False
YGYGRBRB
False
```

6)

```
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));
    }
}
```

```
148 3
3
```

7)

```
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);
    }
}
```

```
4
2
1 2
1 4
3
```

8)

```
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;
        }
    }
  }
}
```

```
2 6
3 1 10
3 2 20
4 1 2
1 1
20
2 1
1 1
10
```