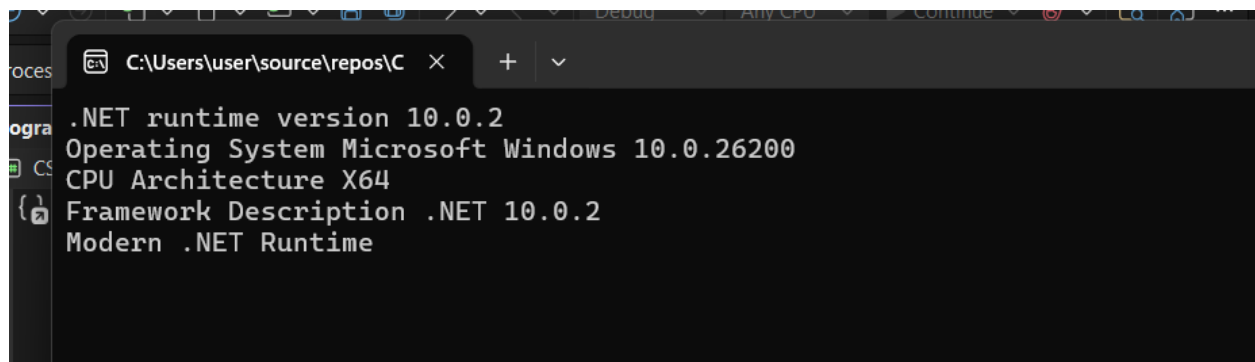


Question 1:

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Runtime.InteropServices;
namespace CSharp.ConsoleApp1
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine(".NET runtime version "+Environment.Version);
            Console.WriteLine("Operating System " +
RuntimeInformation.OSDescription);
            Console.WriteLine("CPU Architecture
"+RuntimeInformation.OSArchitecture);
            Console.WriteLine("Framework Description " +
RuntimeInformation.FrameworkDescription);
            var s = RuntimeInformation.FrameworkDescription;
            switch (s)
            {
                case string r when r.Contains(".NET"):
                    Console.WriteLine("Modern .NET Runtime");
                    break;
                default:
                    Console.WriteLine("Legacy Runtime");
                    break;
            }
            Console.ReadKey();
        }
    }
}
```

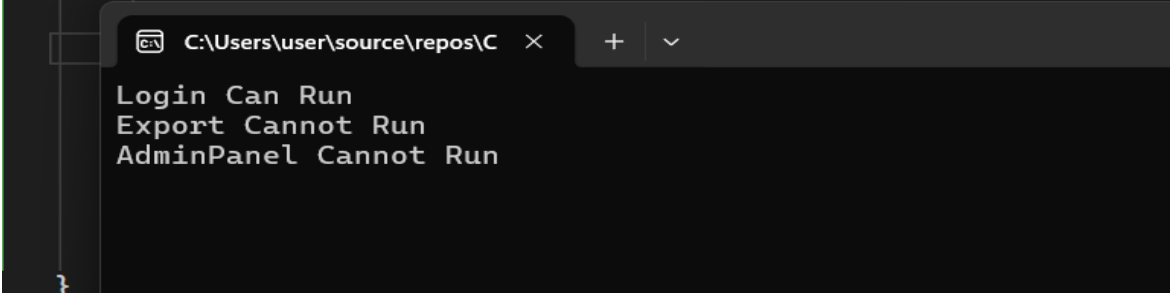


```
C:\Users\user\source\repos\C x + v
.NET runtime version 10.0.2
Operating System Microsoft Windows 10.0.26200
CPU Architecture X64
Framework Description .NET 10.0.2
Modern .NET Runtime
```

Question 2:

```
using System.ComponentModel.DataAnnotations;
using System.Collections.Generic;
namespace CSharp.ConsoleApp1
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Feature Login = new Feature("Login", true, 1.2);
            Login.DisplayCanRun();
            Feature Export = new Feature("Export", false, 1.1);
            Export.DisplayCanRun();
            Feature AdminPanel = new Feature("AdminPanel", true, 1.3);
            AdminPanel.DisplayCanRun();
            Console.ReadKey();
        }
    }

    public class Feature
    {
        const double Version = 1.2;
        private string name;
        private bool enabled;
        private readonly double minimumRequiredVersion;
        public Feature(string name, bool enabled, double minimumRequiredVersion)
        {
            this.name = name;
            this.enabled = enabled;
            this.minimumRequiredVersion = minimumRequiredVersion;
        }
        public bool CanRun()
        {
            return (enabled && minimumRequiredVersion <= Version);
        }
        public void DisplayCanRun()
        {
            Console.Write(name + " ");
            Console.Write(CanRun() ? "Can Run" : "Cannot Run");
            Console.WriteLine();
        }
    }
}
```



The screenshot shows a Windows command prompt window titled "C:\Users\user\source\repos\C". The output of the program is as follows:

```
Login Can Run
Export Cannot Run
AdminPanel Cannot Run
```

Question 3:

```
using System.ComponentModel.DataAnnotations;
using System.Collections.Generic;
namespace CSharp.ConsoleApp1
{
    internal class Program
    {
        static void Main(string[] args)
        {
            List<int> numbers = new List<int> { 2, 3, 4, 5, 6, 7, 8, 9, 10 };

            Numbers num = new Numbers();
            num.DisplayNumbers(numbers);
            Console.ReadKey();
        }
    }

    public class NumberResult
    {
        public List<int> evennumders = new List<int>();
        public List<int> oddnumders = new List<int>();
        public List<int> primenumders = new List<int>();
    }

    public class Numbers
    {
        public NumberResult GetNumbers(List<int> l)
        {
            NumberResult result = new NumberResult();

            foreach (var x in l)
            {
                if (x % 2 == 0)
                    result.evennumders.Add(x);
                else result.oddnumders.Add(x);

                if(Prime(x))
                    result.primenumders.Add(x);
            }
            return result;
        }

        private bool Prime (int x)
        {
            bool check = true;
            if (x <= 1) check = false;
            else
            {
                for (int i = 2; i < x; i++)
                {
                    if (x % i == 0)
                    {
                        check = false;
                        break;
                    }
                }
            }
        }
    }
}
```

```

    }
    return check;
}
public void DisplayNumbers(List<int> l)
{
    var v=GetNumbers(l);

    Console.Write("The odd numbers = ");
    foreach (var x in v.oddnumbers)
    {
        Console.Write(x + " ");
    }
    Console.WriteLine();
    Console.Write("The even numbers = ");
    foreach (var x in v.evennumbers)
    {
        Console.Write(x + " ");
    }
    Console.WriteLine();
    Console.Write("The prime numbers = ");
    foreach (var x in v.primenumbers)
    {
        Console.Write(x + " ");
    }
    Console.WriteLine();
}
}
}

```

```

C:\Users\user\source\repos\C  +  v
The odd numbers = 3 5 7 9
The even numbers = 2 4 6 8 10
The prime numbers = 2 3 5 7

```

Question 4:

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Runtime.InteropServices;
namespace CSharp.ConsoleApp1
{
    internal class Program
    {
        static void Main(string[] args)
        {
            User user = new User();
            UserSnapshot usersnapshot = new UserSnapshot();
            ModifyData(user, usersnapshot);
            Console.WriteLine("Without ref");
            Console.WriteLine($"User name {user.Name} UserSnapshot name {usersnapshot.Name}");

            ModifyDataRef(user, ref usersnapshot);
            Console.WriteLine("With ref");
            Console.WriteLine($"User name {user.Name} UserSnapshot name {usersnapshot.Name}");

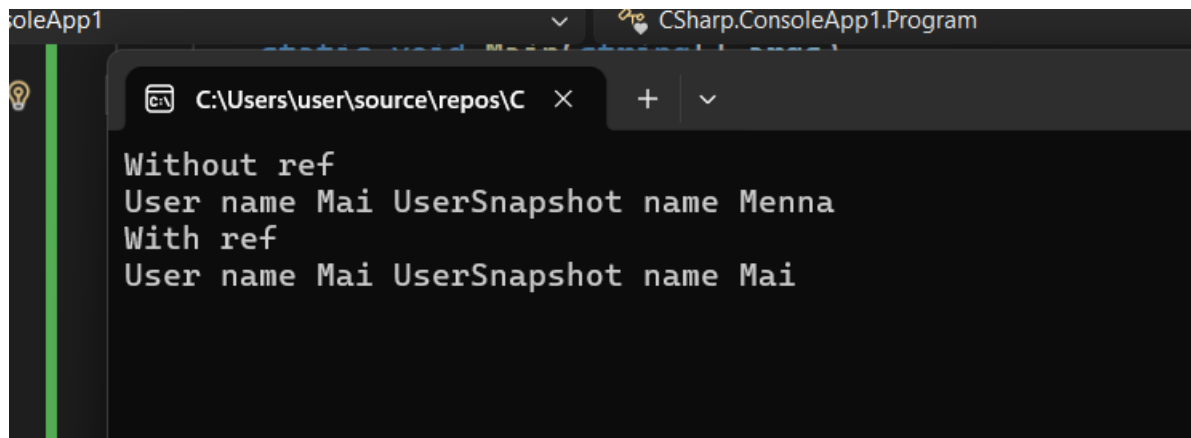
            Console.ReadKey();
        }

        static void ModifyData(User user, UserSnapshot userSnapshot)
        {
            user.Name = "Mai";
            userSnapshot.Name = "Mai";
        }

        static void ModifyDataRef(User user, ref UserSnapshot userSnapshot)
        {
            user.Name = "Mai";
            userSnapshot.Name = "Mai";
        }
    }

    public class User
    {
        public string Name = "Menna";
    }

    public struct UserSnapshot
    {
        public string Name;
        public UserSnapshot()
        {
            Name = "Menna";
        }
    }
}
```

A screenshot of a C# console application window titled 'CSharp.ConsoleApp1.Program'. The window shows the output of a program. The first part of the output is 'Without ref', followed by 'User name Mai UserSnapshot name Menna'. The second part is 'With ref', followed by 'User name Mai UserSnapshot name Mai'. This demonstrates that without the 'ref' keyword, the 'UserSnapshot' field of the 'User' struct is not updated when passed to a method, even if the method's parameter is declared as a 'ref' parameter. The 'User' struct has a 'name' property, and the 'UserSnapshot' struct has a 'name' property. The 'User' struct is passed to a method, and the 'UserSnapshot' struct is passed to another method. The 'UserSnapshot' struct is modified in the second method, but the 'User' struct's 'UserSnapshot' field remains unchanged without the 'ref' keyword.

```
Without ref
User name Mai UserSnapshot name Menna
With ref
User name Mai UserSnapshot name Mai
```

What changed? Why?

The Name of Struct 'UserSnapshot' did not actually change when passed to the method normally because struct is value type any modification inside the method is not applied to the original struct without using ref keyword.

Stack vs Heap

Stack:

Fast memory allocation

Stores local variables and value types

Memory is automatically released when the method ends

Heap:

Slower than stack

Stores objects and class instances (reference types)

Managed by the Garbage Collector

Question 5:

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Runtime.InteropServices;
namespace CSharp.ConsoleApp1
{
    internal class Program
    {
        static void Main(string[] args)
        {
            PaymentSystem payment = new PaymentSystem(1001, "Menna", 2299, 70000);
            try
            {
                payment.MakePayment(8000, 7);
                payment.MakePayment(5000, 7);
                payment.MakePayment(9000, 7);
                //payment.MakePayment(90000, 7);
                payment.MakePayment(9000, 11);
            }

            catch (InsufficientBalanceException ex)
            {
                Console.WriteLine(ex.Message);
            }
            catch (PaymentTimeoutException ex)
            {
                Console.WriteLine(ex.Message);
            }

            finally
            {
                Console.WriteLine("End");
            }

            Console.ReadKey();
        }
    }
}

public class PaymentSystem
{
    const int MaxTime = 10;
    private static int OperationCounter = 0;
    public int ID { get; set; }
    public string Name { get; set; }
    public int AccountNumber { get; set; }
    public int Balance { get; set; }
    public PaymentSystem(int id, string name, int accountnumber, int balance)
    {
        ID = id;
        Name = name;
        AccountNumber = accountnumber;
        Balance = balance;
    }
    public void MakePayment(int balance, int paymenttime)
    {
        int currentOperation = ++OperationCounter;
        if (balance > Balance)
        {
            Console.WriteLine("Insufficient Balance");
        }
        else if (paymenttime > MaxTime)
        {
            Console.WriteLine("Payment Timeout");
        }
        else
        {
            Console.WriteLine("Payment Successful");
        }
    }
}
```

```

        throw new InsufficientBalanceException($"Payment {currentOperation}
Insufficient Balance");
    }
    if(paymenttime > MaxTime)
    {
        throw new PaymentTimeoutException($"Payment {currentOperation}
Payment Time out");
    }
    Balance -= balance;
    Console.WriteLine($"Payment {currentOperation} done successfully ,
Remaining balance = {Balance}");
}
}
public class InsufficientBalanceException : Exception
{
    public InsufficientBalanceException(string message) : base(message) { }
}
public class PaymentTimeoutException : Exception
{
    public PaymentTimeoutException(string message) : base(message) { }
}
}

```

```

oleA
C:\Users\user\source\repos\C x + v
Payment 1 done successfully , Remaining balance = 62000
Payment 2 done successfully , Remaining balance = 57000
Payment 3 done successfully , Remaining balance = 48000
Payment 4 Insufficient Balance
End

```

```

C:\Users\user\source\repos\C x + v
Payment 1 done successfully , Remaining balance = 62000
Payment 2 done successfully , Remaining balance = 57000
Payment 3 done successfully , Remaining balance = 48000
Payment 4 Payment Time out
End

```


Part 2:

Problem 1:

```
public class Solution
{
    public string LongestCommonPrefix(string[] strs)
    {
        if (strs == null || strs.Length == 0) return "";
        else
        {
            string s=strs[0];
            for (int i = 1; i < strs.Length; i++)
            {
                int j = 0;
                while (j < s.Length && j < strs[i].Length && s[j] == strs[i][j] )
                {
                    j++;
                }
                s=s.Substring(0,j);
                if (s == "")
                    break;
            }
            return s;
        }
    }
}
```

The screenshot displays the LeetCode submission page for the "14. Longest Common Prefix" problem. The problem description on the left states: "Write a function to find the longest common prefix string amongst an array of strings. If there is no common prefix, return an empty string \"\"." It includes two examples: Example 1 with input ["flower", "flow", "flight"] and output "fl", and Example 2 with input ["dog", "racecar", "car"] and output "", with an explanation that there is no common prefix. Constraints specify that the number of strings and the length of each string are between 1 and 200.

The central code editor shows the C# solution:

```
1 public class Solution
2 {
3     public string LongestCommonPrefix(string[] strs)
4     {
5         if (strs == null || strs.Length == 0)
6             return "";
7         else
8         {
9             string s=strs[0];
10            for (int i = 1; i < strs.Length; i++)
11            {
12                int j = 0;
```

Below the code editor, the "Test Result" section shows "Accepted" with a runtime of 0 ms. It lists two test cases, both of which are passed. The input for Case 1 is ["flower", "flow", "flight"] and the output is "fl".

On the right, the "Accepted" tab shows performance metrics: 43.42 MB of memory and 48.02% of the best runtime. A bar chart indicates the runtime for various test cases, with the highest being around 2ms. At the bottom right, a snippet of the C# code is visible.

Problem 2:

```
public class Solution
{
    public bool ContainsDuplicate(int[] nums)
    {
        Dictionary<int, int> freq = new Dictionary<int, int>();

        foreach (int num in nums)
        {
            if (freq.ContainsKey(num))
                return true;

            freq[num] = 1;
        }

        return false;
    }
}
```

The screenshot displays the LeetCode submission interface for the problem "Contains Duplicate". The interface is divided into several panels:

- Description:** Shows the problem statement, examples, and constraints. Example 2: Input: `nums = [1,2,3,4]`, Output: `false`. Example 3: Input: `nums = [1,1,1,3,3,4,3,2,4,2]`, Output: `true`. Constraints: $1 \leq \text{nums.length} \leq 10^5$, $-10^9 \leq \text{nums}[i] \leq 10^9$.
- Code:** Displays the C# solution code. The code uses a `Dictionary<int, int>` to track the frequency of each number. It iterates through the array `nums`, and if a number is already in the dictionary, it returns `true`. Otherwise, it adds the number to the dictionary with a frequency of 1. If the loop completes, it returns `false`.
- Test Result:** Shows the results of the test cases. All three cases (Case 1, Case 2, Case 3) are marked as "Accepted". The runtime is 0 ms.
- Runtime and Memory:** Shows the performance metrics. The runtime is 9 ms, which beats 88.88% of other solutions. The memory usage is 71.07 MB, which beats 54.10% of other solutions. A graph shows the memory usage over time.

Problem 3:

```
public class Solution
{
    public bool IsAnagram(string s, string t)
    {
        if (s.Length != t.Length)
            return false;
        int[] freq = new int[26];
        for (int i = 0; i < s.Length; i++)
        {
            freq[s[i] - 'a']++;
            freq[t[i] - 'a']--;
        }
        for (int i = 0; i < 26; i++)
        {
            if (freq[i] > 0)
                return false;
        }
        return true;
    }
}
```

The screenshot shows the LeetCode interface for problem 242, "Valid Anagram". The problem description states: "Given two strings s and t, return true if t is an anagram of s, and false otherwise." Example 1: Input: s = "anagram", t = "nagaram", Output: true. Example 2: Input: s = "rat", t = "car", Output: false. Constraints: 1 <= s.length, t.length <= 5 * 10^4; s and t consist of lowercase English letters.

The code editor shows the following C# code:

```
1 public class Solution {
2     public bool IsAnagram(string s, string t) {
3         if (s.Length != t.Length)
4             return false;
5         int[] freq = new int[26];
6         for (int i = 0; i < s.Length; i++)
7         {
8             freq[s[i] - 'a']++;
9             freq[t[i] - 'a']--;
10        }
11        for (int i = 0; i < 26; i++)
12        {
```

The test result shows "Accepted" with a runtime of 0 ms. The input for the test case is s = "anagram" and t = "nagaram".

The runtime and memory usage are displayed on the right:

- Runtime: 0 ms, Beats 100.00%
- Memory: 43.86 MB, Beats 54.83%

A bar chart shows the distribution of runtime results, with most submissions completing within 10ms.