Set E Solution

1. **Write a program to find and display all prime numbers within a given range. The user should input the starting and ending values of the range, and the program should output all the prime numbers in that range.**

```
using System;

class Program
{
    static void Main()
    {
        Console.Write("Enter the starting value of the range: ");
        int start = Convert.ToInt32(Console.ReadLine());

        Console.Write("Enter the ending value of the range: ");
        int end = Convert.ToInt32(Console.ReadLine());

        Console.WriteLine($"Prime numbers between {start} and {end} are:");

        for (int i = start; i <= end; i++)
        {
            if (IsPrime(i))
            {
                Console.Write(i + " ");
            }
        }
    }
    static bool IsPrime(int number)
    {
        if (number <= 1)
        {
            return false;
        }
        for (int i = 2; i <= Math.Sqrt(number); i++)
        {
            if (number % i == 0)
                return false;
        }
        return true;
    }
}
```

2. Write a program to find and display all prime numbers within a given range. The user should input the starting and ending values of the range, and the program should output all the prime numbers in that range.

```
using System;

class Student
{
    public int RollNumber;
    public string Name;
    public int[] Marks = new int[5];

    public void InputDetails()
    {
        Console.Write("Enter Roll Number: ");
        RollNumber = Convert.ToInt32(Console.ReadLine());

        Console.Write("Enter Name: ");
        Name = Console.ReadLine();

        Console.WriteLine("Enter Marks for 5 Subjects:");
        for (int i = 0; i < Marks.Length; i++)
        {
            Console.Write($"Subject {i + 1}: ");
            Marks[i] = Convert.ToInt32(Console.ReadLine());
        }
    }

    public void DisplayDetails()
    {
        Console.WriteLine("\nStudent Details:");
        Console.WriteLine($"Roll Number: {RollNumber}");
        Console.WriteLine($"Name: {Name}");
        Console.WriteLine("Marks:");
        for (int i = 0; i < Marks.Length; i++)
        {
            Console.WriteLine($"Subject {i + 1}: {Marks[i]}");
        }
    }

    public int CalculateTotalMarks()
    {
        int total = 0;
        foreach (int mark in Marks)
        {
            total += mark;
        }
        return total;
    }

    public double CalculatePercentage()
```

```
    {
        int totalMarks = CalculateTotalMarks();
        return (double)totalMarks / (Marks.Length * 100) * 100;
    }

    public string DetermineGrade()
    {
        double percentage = CalculatePercentage();
        if (percentage >= 90)
            return "A";
        else if (percentage >= 75)
            return "B";
        else if (percentage >= 50)
            return "C";
        else
            return "F";
    }
}

class Program
{
    static void Main()
    {
        Student student = new Student();

        student.InputDetails();

        student.DisplayDetails();

        int totalMarks = student.CalculateTotalMarks();
        Console.WriteLine($"\nTotal Marks: {totalMarks}");

        double percentage = student.CalculatePercentage();
        Console.WriteLine($"Percentage: {percentage:F2}%");

        string grade = student.DetermineGrade();
        Console.WriteLine($"Grade: {grade}");
    }
}
```

3. Create a base class Vehicle with attributes Make, Model, and Year, then derive Car with an additional attribute FuelType and DisplayDetails() method, and Truck with an additional attribute LoadCapacity and DisplayDetails() method to show specific details of each vehicle.

```
using System;

class Vehicle
{
    public string Make;
    public string Model;
    public int Year;
```

```
    public void InputDetails()
    {
        Console.Write("Enter Make: ");
        Make = Console.ReadLine();

        Console.Write("Enter Model: ");
        Model = Console.ReadLine();

        Console.Write("Enter Year: ");
        Year = Convert.ToInt32(Console.ReadLine());
    }

    public virtual void DisplayDetails()
    {
        Console.WriteLine($"Make: {Make}");
        Console.WriteLine($"Model: {Model}");
        Console.WriteLine($"Year: {Year}");
    }
}

class Car : Vehicle
{
    public string FuelType;

    public void InputCarDetails()
    {
        InputDetails();
        Console.Write("Enter Fuel Type (Petrol/Diesel/Electric): ");
        FuelType = Console.ReadLine();
    }

    public override void DisplayDetails()
    {
        base.DisplayDetails();
        Console.WriteLine($"Fuel Type: {FuelType}");
    }
}

class Truck : Vehicle
{
    public double LoadCapacity;

    public void InputTruckDetails()
    {
        InputDetails();
        Console.Write("Enter Load Capacity (in tons): ");
        LoadCapacity = double.Parse(Console.ReadLine());
    }

    public override void DisplayDetails()
    {
        base.DisplayDetails();
```

```
        Console.WriteLine($"Load Capacity: {LoadCapacity} tons");
    }
}

class Program
{
    static void Main()
    {
        Console.WriteLine("Enter details for Car:");
        Car car = new Car();
        car.InputCarDetails();
        Console.WriteLine("\nCar Details:");
        car.DisplayDetails();

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

        Console.WriteLine("Enter details for Truck:");
        Truck truck = new Truck();
        truck.InputTruckDetails();
        Console.WriteLine("\nTruck Details:");
        truck.DisplayDetails();
    }
}
```

4. **Create an abstract class Employee with an abstract method CalculateSalary(), an interface Bonus with a method CalculateBonus(), and derive two classes: Manager that implements CalculateSalary() and CalculateBonus() to calculate salary and bonus for a manager, and Worker that implements CalculateSalary() and CalculateBonus() to calculate salary and bonus for a worker, demonstrating inheritance, abstract class, and interface implementation.**

```
using System;

abstract class Employee
{
    public string Name;
    public int EmployeeID;

    public abstract double CalculateSalary();
}

interface IBonus
{
    double CalculateBonus();
}

class Manager : Employee, IBonus
{
    public double BaseSalary;
    public double Allowances;

    public override double CalculateSalary()
```

```
    {
        return BaseSalary + Allowances;
    }

    public double CalculateBonus()
    {
        return BaseSalary * 0.10;
    }
}

class Worker : Employee, IBonus
{
    public double HourlyRate;
    public int HoursWorked;

    public override double CalculateSalary()
    {
        return HourlyRate * HoursWorked;
    }

    public double CalculateBonus()
    {
        return HourlyRate * 20;
    }
}

class Program
{
    static void Main()
    {
        Console.WriteLine("Enter details for Manager:");
        Manager manager = new Manager
        {
            Name = "Alice",
            EmployeeID = 101,
            BaseSalary = 50000,
            Allowances = 20000
        };

        Console.WriteLine($"Manager {manager.Name} (ID: {manager.EmployeeID}):");
        Console.WriteLine($"Salary: {manager.CalculateSalary():C}");
        Console.WriteLine($"Bonus: {manager.CalculateBonus():C}");

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

        Console.WriteLine("Enter details for Worker:");
        Worker worker = new Worker
        {
            Name = "Bob",
            EmployeeID = 202,
            HourlyRate = 50,
            HoursWorked = 160
```

```
        };

        Console.WriteLine($"Worker {worker.Name} (ID: {worker.EmployeeID}):");
        Console.WriteLine($"Salary: {worker.CalculateSalary():C}");
        Console.WriteLine($"Bonus: {worker.CalculateBonus():C}");
    }
}
```

5. **Create a custom exception NegativeAgeException. In your main program, write logic to accept the age of a user and throw the NegativeAgeException if the age is negative. Display a custom error message when the exception is caught.**

```
using System;

class NegativeAgeException : Exception
{
    public NegativeAgeException(string message) : base(message) { }
}

class Program
{
    static void Main()
    {
        try
        {
            Console.Write("Enter your age: ");
            int age = Convert.ToInt32(Console.ReadLine());

            if (age < 0)
            {
                throw new NegativeAgeException("Age cannot be negative.");
            }

            Console.WriteLine($"Your age is: {age}");
        }
        catch (NegativeAgeException ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
        }
    }
}
```

6. Write a program that counts the number of vowels, consonants, digits, and special characters in a given string using string functions.

```
using System;

class Program
{
   static void Main()
   {
      Console.Write("Enter a string: ");
      string input = Console.ReadLine();

      int vowels = 0, consonants = 0, digits = 0, specialCharacters = 0;

      foreach (char ch in input)
      {
         if (char.IsDigit(ch))
         {
            digits++;
         }
         else if (char.IsLetter(ch))
         {
            char lowerChar = char.ToLower(ch);
            if ("aeiou".Contains(lowerChar))
            {
               vowels++;
            }
            else
            {
               consonants++;
            }
         }
         else if (!char.IsWhiteSpace(ch))
         {
            specialCharacters++;
         }
      }

      Console.WriteLine($"\nVowels: {vowels}");
      Console.WriteLine($"Consonants: {consonants}");
      Console.WriteLine($"Digits: {digits}");
      Console.WriteLine($"Special Characters: {specialCharacters}");
   }
}
```

7. Create a program that demonstrates the use of a copy constructor with a Time object. The Time class should store time in hours, minutes, and seconds, and provide functionality for adding times.

```
using System;

class Time
{
    public int Hours;
    public int Minutes;
    public int Seconds;

    public Time(int hours = 0, int minutes = 0, int seconds = 0)
    {
        Hours = hours;
        Minutes = minutes;
        Seconds = seconds;
        NormalizeTime();
    }

    public Time(Time other)
    {
        Hours = other.Hours;
        Minutes = other.Minutes;
        Seconds = other.Seconds;
    }

    private void NormalizeTime()
    {
        Minutes += Seconds / 60;
        Seconds %= 60;

        Hours += Minutes / 60;
        Minutes %= 60;
    }

    public Time Add(Time other)
    {
        return new Time(Hours + other.Hours, Minutes + other.Minutes, Seconds + other.Seconds);
    }

    public void Display()
    {
        Console.WriteLine($"{Hours:D2}:{Minutes:D2}:{Seconds:D2}");
    }
}

class Program
{
    static void Main()
    {
        Time time1 = new Time(1, 45, 50);
```

```
        Console.WriteLine("Original Time:");
        time1.Display();

        Time timeCopy = new Time(time1);
        Console.WriteLine("\nCopied Time:");
        timeCopy.Display();

        Time time2 = new Time(2, 20, 30);
        Console.WriteLine("\nAnother Time:");
        time2.Display();

        Time timeSum = time1.Add(time2);
        Console.WriteLine("\nSum of Times:");
        timeSum.Display();
    }
}
```