**Задание 4 вариант 1**

using System;

// Классы исключений

public class CounterUnderflowException : Exception

{

public CounterUnderflowException(string message) : base(message) { }

}

public class CounterOverflowException : Exception

{

public CounterOverflowException(string message) : base(message) { }

}

public class InvalidCounterRangeException : Exception

{

public InvalidCounterRangeException(string message) : base(message) { }

}

// Класс десятичного счетчика

public class DecimalCounter

{

private int \_current;

private readonly int \_min;

private readonly int \_max;

// Конструктор по умолчанию (0-9, старт = 0)

public DecimalCounter()

{

\_min = 0;

\_max = 9;

\_current = 0;

}

// Пользовательский конструктор

public DecimalCounter(int min, int max, int initialValue)

{

if (min >= max) throw new InvalidCounterRangeException("Минимум должен быть меньше максимума");

if (initialValue < min || initialValue > max)

throw new InvalidCounterRangeException("Начальное значение вне диапазона");

\_min = min;

\_max = max;

\_current = initialValue;

}

// Свойство для получения текущего значения

public int Current => \_current;

// Увеличение счетчика

public void Increment()

{

if (\_current >= \_max)

throw new CounterOverflowException($"Достигнут максимум {\_max}");

\_current++;

}

// Уменьшение счетчика

public void Decrement()

{

if (\_current <= \_min)

throw new CounterUnderflowException($"Достигнут минимум {\_min}");

\_current--;

}

}

class Program

{

static void Main()

{

try

{

// Тест 1: Счетчик по умолчанию

var defaultCounter = new DecimalCounter();

Console.WriteLine($"По умолчанию: {defaultCounter.Current}");

defaultCounter.Increment();

Console.WriteLine($"После увеличения: {defaultCounter.Current}");

// Тест 2: Пользовательский счетчик

var customCounter = new DecimalCounter(5, 15, 10);

Console.WriteLine($"\nКастомный: {customCounter.Current}");

customCounter.Increment();

Console.WriteLine($"После увеличения: {customCounter.Current}");

// Тест 3: Попытка выйти за пределы

var edgeCounter = new DecimalCounter(0, 2, 2);

Console.WriteLine($"\nГраничный: {edgeCounter.Current}");

edgeCounter.Increment(); // Выбросит исключение

}

catch (Exception ex)

{

Console.WriteLine($"\nОшибка: {ex.Message}");

}

// Тест 4: Некорректная инициализация (исправленная версия)

try

{

new DecimalCounter(10, 5, 7); // Убрана неиспользуемая переменная

}

catch (InvalidCounterRangeException ex)

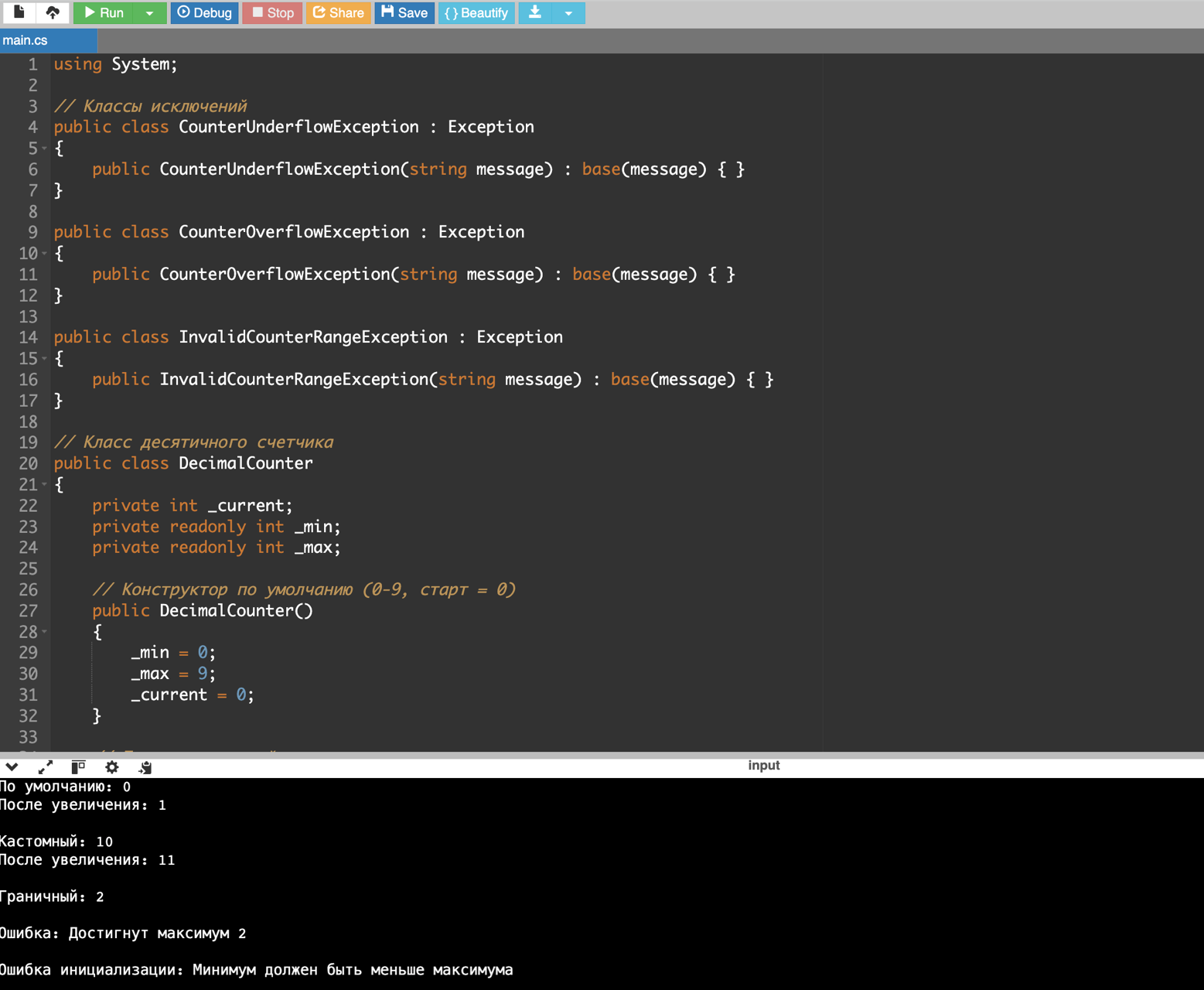
{

Console.WriteLine($"\nОшибка инициализации: {ex.Message}");

}

}

}



**Задание 8 вариант 1**

using System;

// Классы исключений

public class CounterUnderflowException : Exception

{

public CounterUnderflowException(string message) : base(message) { }

}

public class CounterOverflowException : Exception

{

public CounterOverflowException(string message) : base(message) { }

}

public class InvalidCounterRangeException : Exception

{

public InvalidCounterRangeException(string message) : base(message) { }

}

// Класс десятичного счетчика

public class DecimalCounter

{

private int \_current;

private readonly int \_min;

private readonly int \_max;

// Конструктор по умолчанию (0-9, старт = 0)

public DecimalCounter()

{

\_min = 0;

\_max = 9;

\_current = 0;

}

// Пользовательский конструктор

public DecimalCounter(int min, int max, int initialValue)

{

if (min >= max) throw new InvalidCounterRangeException("Минимум должен быть меньше максимума");

if (initialValue < min || initialValue > max)

throw new InvalidCounterRangeException("Начальное значение вне диапазона");

\_min = min;

\_max = max;

\_current = initialValue;

}

// Свойство для получения текущего значения

public int Current => \_current;

// Увеличение счетчика

public void Increment()

{

if (\_current >= \_max)

throw new CounterOverflowException($"Достигнут максимум {\_max}");

\_current++;

}

// Уменьшение счетчика

public void Decrement()

{

if (\_current <= \_min)

throw new CounterUnderflowException($"Достигнут минимум {\_min}");

\_current--;

}

}

class Program

{

static void Main()

{

try

{

// Тест 1: Счетчик по умолчанию

var defaultCounter = new DecimalCounter();

Console.WriteLine($"По умолчанию: {defaultCounter.Current}");

defaultCounter.Increment();

Console.WriteLine($"После увеличения: {defaultCounter.Current}");

// Тест 2: Пользовательский счетчик

var customCounter = new DecimalCounter(5, 15, 10);

Console.WriteLine($"\nКастомный: {customCounter.Current}");

customCounter.Increment();

Console.WriteLine($"После увеличения: {customCounter.Current}");

// Тест 3: Попытка выйти за пределы

var edgeCounter = new DecimalCounter(0, 2, 2);

Console.WriteLine($"\nГраничный: {edgeCounter.Current}");

edgeCounter.Increment(); // Выбросит исключение

}

catch (Exception ex)

{

Console.WriteLine($"\nОшибка: {ex.Message}");

}

// Тест 4: Некорректная инициализация (исправленная версия)

try

{

var \_ = new DecimalCounter(10, 5, 7); // Фиктивная переменная

}

catch (InvalidCounterRangeException ex)

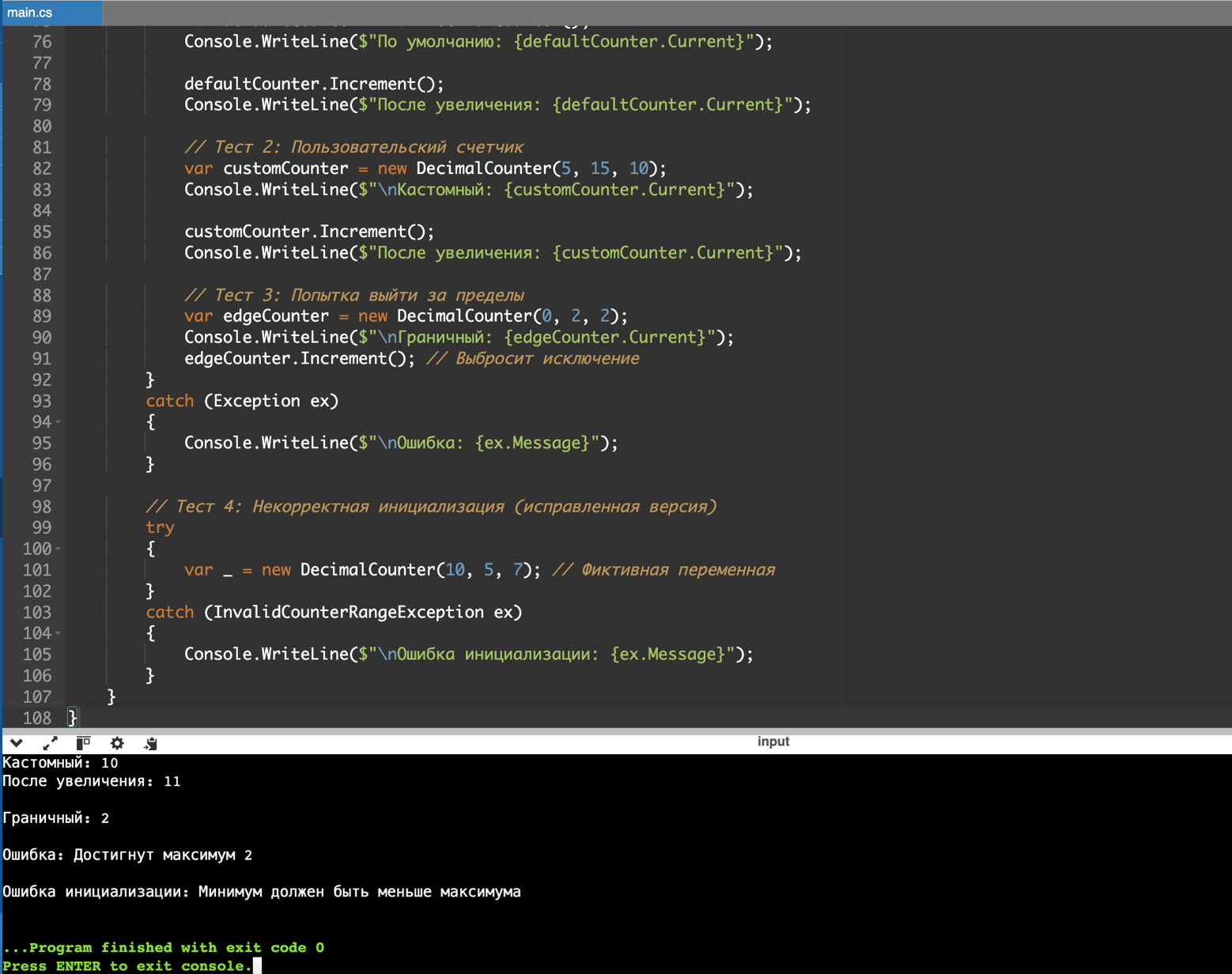
{

Console.WriteLine($"\nОшибка инициализации: {ex.Message}");

}

}

}



**Задание 9 вариант 1**

using System;

using System.Collections.Generic;

public class Point

{

public double X { get; set; }

public double Y { get; set; }

public Point() : this(0, 0) { }

public Point(double x, double y) { X = x; Y = y; }

public override bool Equals(object obj)

{

return obj is Point other && X == other.X && Y == other.Y;

}

public override int GetHashCode()

{

return HashCode.Combine(X, Y);

}

}

public class ColoredPoint : Point

{

public string Color { get; set; }

public ColoredPoint() : base() => Color = "Black";

public ColoredPoint(double x, double y, string color) : base(x, y) => Color = color;

public override bool Equals(object obj)

{

return obj is ColoredPoint other &&

base.Equals(other) &&

Color == other.Color;

}

public override int GetHashCode()

{

return HashCode.Combine(base.GetHashCode(), Color);

}

}

public class Line : Point

{

public Point End { get; set; }

public Line() : base() => End = new Point();

public Line(double sx, double sy, double ex, double ey) : base(sx, sy) => End = new Point(ex, ey);

public void Rotate(double degrees)

{

double radians = degrees \* Math.PI / 180;

double dx = End.X - X;

double dy = End.Y - Y;

End.X = X + dx \* Math.Cos(radians) - dy \* Math.Sin(radians);

End.Y = Y + dx \* Math.Sin(radians) + dy \* Math.Cos(radians);

}

public override bool Equals(object obj)

{

return obj is Line line &&

base.Equals(obj) &&

End.Equals(line.End);

}

public override int GetHashCode()

{

return HashCode.Combine(base.GetHashCode(), End);

}

}

public class ColoredLine : Line

{

public string Color { get; set; }

public ColoredLine() : base() => Color = "Black";

public ColoredLine(double sx, double sy, double ex, double ey, string color)

: base(sx, sy, ex, ey) => Color = color;

public override bool Equals(object obj)

{

return obj is ColoredLine cl &&

base.Equals(obj) &&

Color == cl.Color;

}

public override int GetHashCode()

{

return HashCode.Combine(base.GetHashCode(), Color);

}

}

public class PolyLine : Line

{

public List<Point> Points { get; set; } = new List<Point>();

public void Scale(double factor)

{

for (int i = 0; i < Points.Count; i++)

{

Points[i].X = X + (Points[i].X - X) \* factor;

Points[i].Y = Y + (Points[i].Y - Y) \* factor;

}

End.X = X + (End.X - X) \* factor;

End.Y = Y + (End.Y - Y) \* factor;

}

public override bool Equals(object obj)

{

return obj is PolyLine pl &&

base.Equals(obj) &&

EqualityComparer<List<Point>>.Default.Equals(Points, pl.Points);

}

public override int GetHashCode()

{

return HashCode.Combine(base.GetHashCode(), Points);

}

}

class Program

{

static void Main()

{

List<Point> shapes = new List<Point>

{

new ColoredPoint(2, 3, "Red"),

new Point(1, 1),

new Line(0, 0, 5, 5),

new ColoredLine(1, 2, 3, 4, "Blue"),

new PolyLine()

{

X = 0, Y = 0,

End = new Point(10, 0),

Points = { new Point(2, 2), new Point(3, 3) }

}

};

foreach (var shape in shapes)

{

Console.WriteLine($"Type: {shape.GetType().Name}");

Console.WriteLine($"Position: ({shape.X}, {shape.Y})");

if (shape is Line line)

{

Console.WriteLine($"End: ({line.End.X}, {line.End.Y})");

line.Rotate(45);

Console.WriteLine($"After rotation: ({line.End.X}, {line.End.Y})");

if (shape is ColoredLine cl)

Console.WriteLine($"Color: {cl.Color}");

if (shape is PolyLine pl)

{

pl.Scale(2);

Console.WriteLine($"After scaling: ({pl.End.X}, {pl.End.Y})");

}

}

if (shape is ColoredPoint cp)

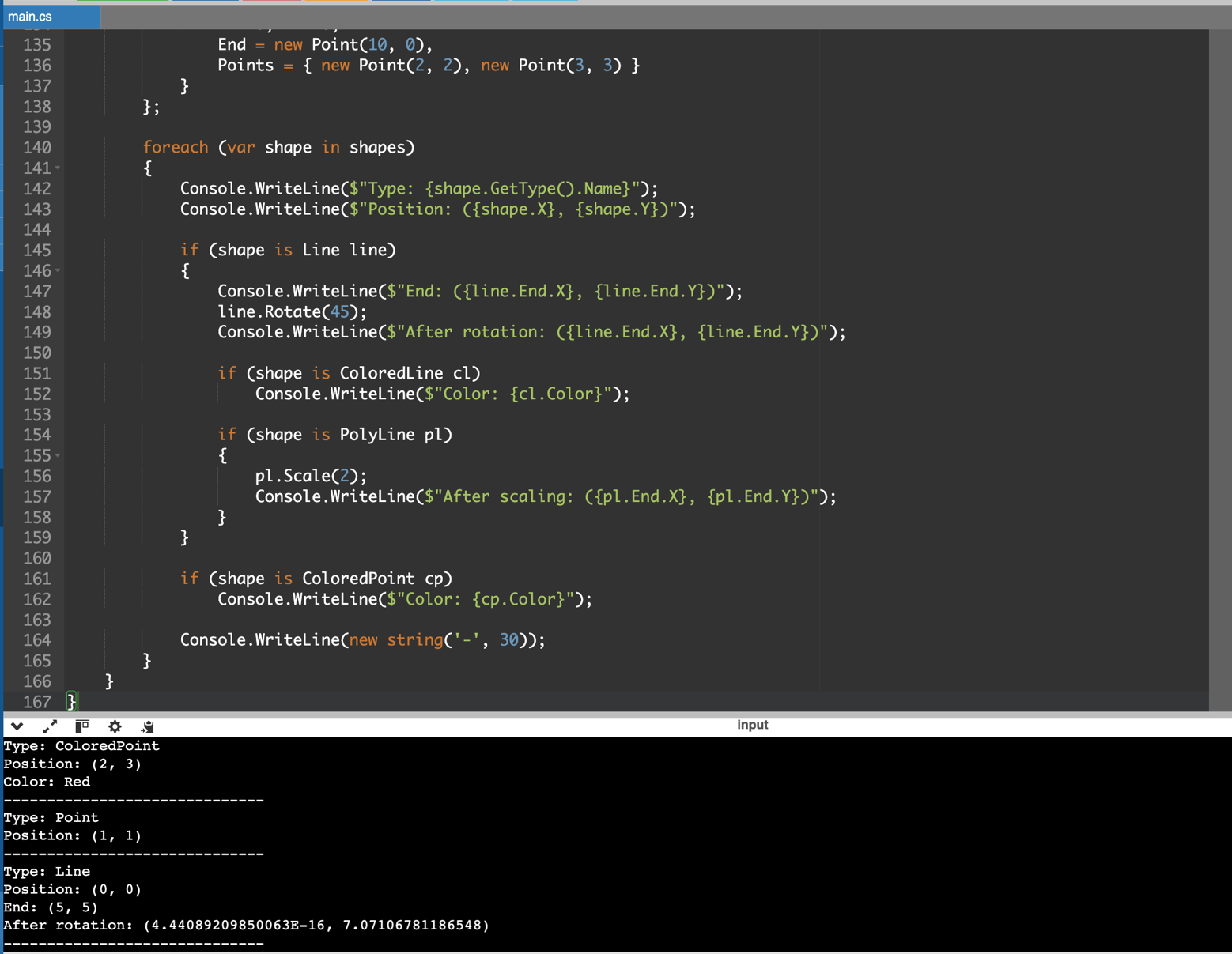
Console.WriteLine($"Color: {cp.Color}");

Console.WriteLine(new string('-', 30));

}

}

}



Задание 10 вариант 1

using System;

using System.Linq;

public struct STUDENT

{

public string FullName;

public int GroupNumber;

public int[] Grades;

public STUDENT(string fullName, int groupNumber, int[] grades)

{

FullName = fullName;

GroupNumber = groupNumber;

Grades = grades;

}

}

class Program

{

static void Main()

{

STUDENT[] students = new STUDENT[10];

// Ввод данных

for (int i = 0; i < 10; i++)

{

Console.WriteLine($"\nСтудент {i + 1}:");

Console.Write("Фамилия и инициалы: ");

string name = Console.ReadLine();

int group;

while (true)

{

Console.Write("Номер группы: ");

if (int.TryParse(Console.ReadLine(), out group)) break;

Console.WriteLine("Ошибка! Введите целое число.");

}

int[] grades = new int[5];

for (int j = 0; j < 5; j++)

{

while (true)

{

Console.Write($"Оценка {j + 1}: ");

if (int.TryParse(Console.ReadLine(), out grades[j])) break;

Console.WriteLine("Ошибка! Введите целое число.");

}

}

students[i] = new STUDENT(name, group, grades);

}

// Сортировка по номеру группы

Array.Sort(students, (s1, s2) => s1.GroupNumber.CompareTo(s2.GroupNumber));

// Поиск студентов с средним баллом > 4.0

bool hasHonorStudents = false;

Console.WriteLine("\nРезультаты:");

foreach (var student in students)

{

double average = student.Grades.Average();

if (average > 4.0)

{

Console.WriteLine($"{student.FullName} (Группа: {student.GroupNumber})");

hasHonorStudents = true;

}

}

if (!hasHonorStudents)

{

Console.WriteLine("Студентов со средним баллом выше 4.0 не найдено.");

}

}

}

