ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА (САМАРСКИЙ УНИВЕРСИТЕТ)»

Институт Информатики и кибернетики
Кафедра Программных систем

ОТЧЁТ

по лабораторной работе

№3 «Язык программирования C#: Полиморфизм (наследование, интерфейсы)»

по дисциплине «Языки программирования и структуры данных»

Выполнил Фадеев А.М. 6101

Проверил Котенёва С.Э.

Самара

2024

ЗАДАНИЕ

Задание 0.

Прочитать теоретический материал.

Задание 1.

Описать интерфейс «IVectorable», содержащий следующие элементы:

- индексатор для организации доступа к элементам массива/списка;
- свойство для чтения числа координат вектора Length;
- метод получения модуля вектора GetNorm().

Задание 2.

Сделать классы ArrayVector и LinkListVector реализующими интерфейс IVectorable и привести их в соответствие с описанной структурой наследования.

Поменять в классе со статическими методами Vectors типы параметров так, чтобы методы работали со ссылками типа интерфейс IVectorable.

Задание 3.

В классах ArrayVector и LinkListVector переопределить унаследованный от класса Object метод ToString(), который преобразует вектор в строку формата «<число координат вектора><пробел><координаты вектора через пробел>». Например, 5 1 2 3 4 5. В дальнейшем, для вывода информации о векторе использовать вызов данного метода.

Задание 4.

Добавить в индексаторы классов ArrayVector и LinkListVector выброс исключительной ситуации (возможно, IndexOutOfRangeException) в случае выхода параметра индексатора за пределы индексирования координат вектора.

Задание 5.

Протестировать работу приложения в классе Program, разработать адекватный интерфейс пользователя, отлавливать и обрабатывать все (!) возможные исключения.

Воспользоваться пользовательским интерфейсом из лабораторной работы 2.

Примечание: в меню предусмотреть пункт, позволяющий сложить вектора разных типов.

Задание 6.

Подготовить отчет о работе.

КОД ПРОГРАММЫ

```
namespace Lab03;

public class LinkedListVector : IVectorable
{
    private Node head;

    private class Node
    {
        public int value = 0;
        public Node next = null;

        public Node(int value)
        {
            this.value = value;
            next = null;
        }
    }

    public LinkedListVector()
    {
        var r = new Random();

        head = new Node(r.Next(100));
        Node cur = head;

        for (int i = 0; i < 5; i++)
        {
            cur.next = new Node(r.Next(100));
            cur = cur.next;
        }
    }

    public LinkedListVector(int length)
    {
        var r = new Random();

        head = new Node(r.Next(100));
        Node cur = head;

        for (int i = 0; i < length; i++)
        {
            cur.next = new Node(r.Next(100));
            cur = cur.next;
        }
    }

    public int this[int idx]
    {
        get
        {
            if (0 <= idx && idx <= Length)
            {
                Node cur = head;
                for (int i = 0; i < idx; i++)
                {
                    cur = cur.next;
                }

                return cur.value;
            }
            else
            {
```

```
                throw new IndexOutOfRangeException("Индекс за границами
связного списка");
            }
        }
        set
        {
            if (0 <= idx && idx <= Length)
            {
                Node cur = head;
                for (int i = 0; i < idx; i++)
                {
                    cur = cur.next;
                }

                cur.value = value;
            }
            else
            {
                throw new IndexOutOfRangeException("Индекс за границами
связного списка");
            }
        }
    }

    public int Length
    {
        get
        {
            if (head == null)
            {
                return -1;
            }

            int length = 0;
            Node cur = head;
            while (cur.next != null)
            {
                cur = cur.next;
                length++;
            }

            return length;
        }
    }

    public double GetNorm()
    {
        double acc = 0;
        Node cur = head;
        for (int i = 0; i < Length; i++)
        {
            acc += Math.Pow(cur.value, 2);
            cur = cur.next;
        }

        return Math.Sqrt(acc);
    }

    public void InsertByIndex(int idx, int value)
    {
        if (idx < 0 || idx > Length) throw new
IndexOutOfRangeException("Индекс за границами связного списка");

        Node node = new Node(value);

        if (idx == 0) {
```

```
            node.next = head;
            head = node;
            return;
        }

        Node cur = head;
        int curIndex = 0;
        while (cur != null && curIndex < idx - 1) {
            cur = cur.next;
            curIndex++;
        }

        if (cur == null) throw new IndexOutOfRangeException("Индекс за
границами связного списка");

        node.next = cur.next;
        cur.next = node;
    }

    public void InsertToStart(int value)
    {
        InsertByIndex(0, value);
    }

    public void InsertToEnd(int value)
    {
        InsertByIndex(Length, value);
    }

    public void DeleteByIndex(int idx)
    {
        if (head == null) throw new Exception("Связный список пуст");
        if (idx < 0 || idx >= Length) throw new
IndexOutOfRangeException("Индекс за границами связного списка");

        Node cur = head;

        if (idx == 0)
        {
            head = cur.next;
            return;
        }

        for (int i = 0; cur != null && i < idx - 1; i++)
        {
            cur = cur.next;
        }

        if (cur == null || cur.next == null) return;

        cur.next = cur.next.next;
    }

    public void DeleteFromStart()
    {
        DeleteByIndex(0);
    }

    public void DeleteFromEnd()
    {
        DeleteByIndex(Length - 1);
    }

    public void Log(string message = "")
    {
        if (message != "") Console.Write($"{message}: ");
```

```
        var cur = head;
        Console.Write("{");
        while (cur.next != null)
        {
            if (cur.next.next == null)
            {
                Console.Write(cur.value);
            }
            else
            {
                Console.Write(cur.value + ", ");
            }
            cur = cur.next;
        }
        Console.WriteLine("}");
    }
}
namespace Lab03;

public interface IVectorable
{
    int this[int index] { get; set; }

    int Length { get; }

    double GetNorm();

    void Log(string message = "");
}
namespace Lab03;

public static class Program
{
    public static void Main()
    {
        List<IVectorable> vectors = new List<IVectorable>();
        string inp;

        vectors.Add(GetVectorFromUserInput());
        LogVectors(vectors);

        while (true)
        {
            Console.WriteLine("Выберете действие:\n\n" +
                              "\t1 - Сумма векторов\n" +
                              "\t2 - Скалярное умножение\n" +
                              "\t3 - Умножение на число\n" +
                              "\t4 - Рассчитать модуль вектора\n" +
                              "\t5 - Добавить вектор в список\n" +
                              "\t6 - Удалить вектор из списка\n" +
                              "\t0 - Выход\n");

            inp = Console.ReadLine();

            switch (inp)
            {
                case "0":
                    Console.WriteLine("Выход из программы...");
                    return;
                case "1":
                {
                    LogVectors(vectors);

                    int firstVectorIdx, secondVectorIdx;
```

```
                    do
                    {
                        Console.Write("Введите индекс первого вектора: ");
                        inp = Console.ReadLine();
                    } while (!int.TryParse(inp, out firstVectorIdx) ||
firstVectorIdx <= 0 || firstVectorIdx > vectors.Count);

                    do
                    {
                        Console.Write("Введите индекс второго вектора: ");
                        inp = Console.ReadLine();
                    } while (!int.TryParse(inp, out secondVectorIdx) ||
secondVectorIdx <= 0 || secondVectorIdx > vectors.Count);

                    try
                    {
                        var result = Vectors.Sum(vectors[firstVectorIdx - 1],
vectors[secondVectorIdx - 1]);
                        result.Log($"Результат сложения {firstVectorIdx}-го и
{secondVectorIdx}-го векторов");
                    }
                    catch (Exception e)
                    {
                        Console.WriteLine("Длины векторов не совпадают");
                    }

                    break;
                }
                case "2":
                {
                    LogVectors(vectors);

                    int firstVectorIdx, secondVectorIdx;

                    do
                    {
                        Console.Write("Введите индекс первого вектора: ");
                        inp = Console.ReadLine();
                    } while (!int.TryParse(inp, out firstVectorIdx) ||
firstVectorIdx <= 0 || firstVectorIdx > vectors.Count);

                    do
                    {
                        Console.Write("Введите индекс второго вектора: ");
                        inp = Console.ReadLine();
                    } while (!int.TryParse(inp, out secondVectorIdx) ||
secondVectorIdx <= 0 || secondVectorIdx > vectors.Count);

                    try
                    {
                        var result =
Vectors.ScalarMultiply(vectors[firstVectorIdx - 1], vectors[secondVectorIdx -
1]);
                        Console.WriteLine($"Результат скалярного умножения:
{result}");
                    }
                    catch (Exception e)
                    {
                        Console.WriteLine("Длины векторов не совпадают");
                    }

                    break;
                }
                case "3":
                {
                    LogVectors(vectors);
```

```
                    int idx, value;

                    do
                    {
                        Console.Write("Введите индекс вектора: ");
                        inp = Console.ReadLine();
                    } while (!int.TryParse(inp, out idx) ||  idx <= 0 || idx
> vectors.Count);

                    do
                    {
                        Console.Write("Введите значение на которое умножить
число: ");
                        inp = Console.ReadLine();
                    } while (!int.TryParse(inp, out value));

                    var result = Vectors.MultiplyByNumber(vectors[idx - 1],
value);
                    result.Log($"Результат сложения умножения вектора на
число");

                    break;
                }
                case "4":
                {
                    LogVectors(vectors);

                    int idx;

                    do
                    {
                        Console.Write("Введите номер вектора для удаления:
");
                        inp = Console.ReadLine();
                    } while (!Int32.TryParse(inp, out idx) || idx < 1 || idx
> vectors.Count);

                    double norm = vectors[idx - 1].GetNorm();

                    Console.WriteLine($"Модуль вектора: {norm}");

                    break;
                }
                case "5":
                {
                    vectors.Add(GetVectorFromUserInput());
                    LogVectors(vectors);
                    break;
                }
                case "6":
                {
                    LogVectors(vectors);
                    int idx;

                    do
                    {
                        Console.Write("Введите номер вектора для удаления:
");
                        inp = Console.ReadLine();
                    } while (!Int32.TryParse(inp, out idx) || idx < 1 || idx
> vectors.Count);

                    vectors.Remove(vectors[idx - 1]);

                    LogVectors(vectors);
```

```
                    break;
                }
                default:
                    Console.WriteLine("Нет такого пункта в меню");
                    break;
            }
        }
    }

    public static IVectorable GetVectorFromUserInput()
    {
        IVectorable vec;
        string inp;
        do
        {
            Console.WriteLine("Выберете тип вектора:\n\n" +
                              "\t1 - Вектор\n" +
                              "\t2 - Связный список\n");
            inp = Console.ReadLine();
        } while (inp != "1" && inp != "2");

        if (inp == "1")
        {
            vec = ArrayVector.GetFromUserInput();
        }
        else
        {
            int length;
            do
            {
                Console.Write("Введите длину связного списка: ");
                inp = Console.ReadLine();
            } while (!Int32.TryParse(inp, out length));

            vec = new LinkedListVector(length);
        }

        return vec;
    }

    public static void LogVectors(List<IVectorable> vectors)
    {
        for (int i = 0; i < vectors.Count; ++i)
        {
            vectors[i].Log($"Вектор {i + 1}");
        }
    }
}
```

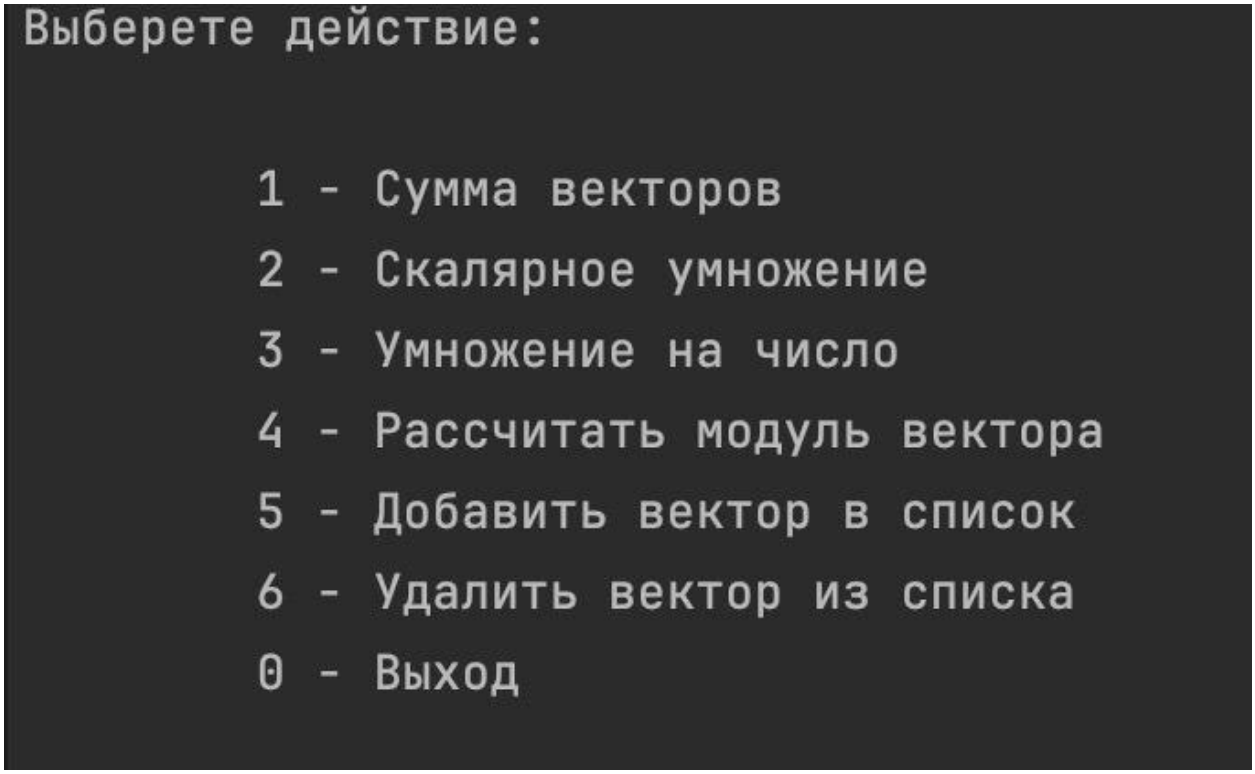

Рисунок 1 – Главное меню программы

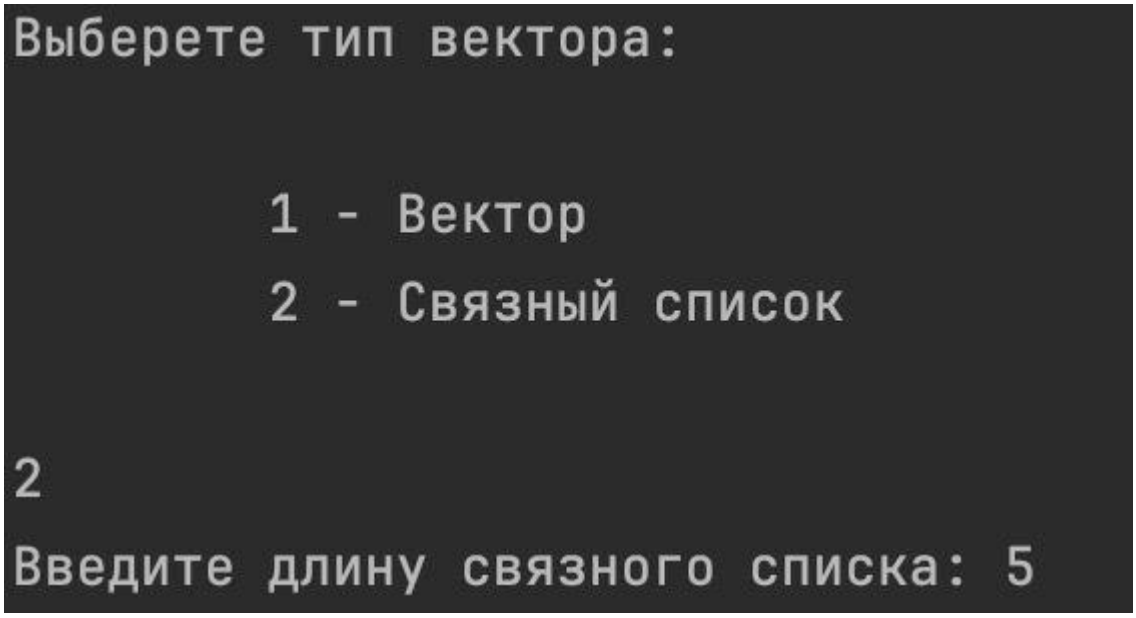

Рисунок 2 – Добавление вектора в общий список

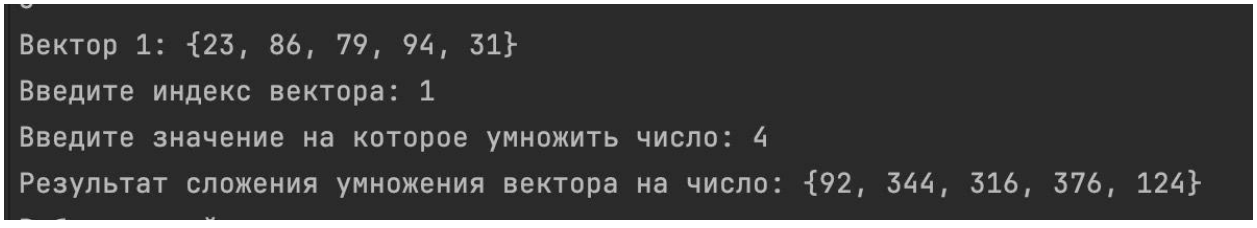

Рисунок 3 – Умножение вектора на число

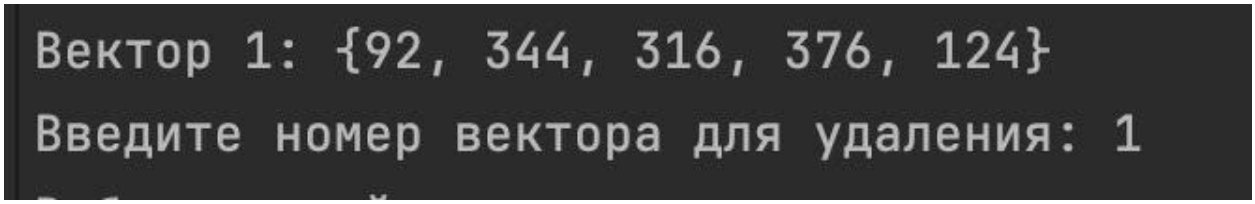

Рисунок 4 – Удаление вектора из списка

ВЫВОДЫ

В лабораторной работе были использованы конструкции языка:

- форматированный вывод информации на консоль;
- оператор switch;
- условные операторы;
- функции;
- классы;
- конструкторы класса;
- поля класса;
- статические и динамические методы класса;
- интерфейсы;
- индексаторы;
- конструкция try-catch.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Павловская Т.А. C#. Программирование на языке высокого уровня. Учебник для вузов [Текст]/Т.А. Павловская. – СПб.: Питер, 2007. – 432 с.