



**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

№5 «Язык программирования C#: Потоки данных»

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.

В класс `Vectors` добавить следующие методы:

- записи вектора в байтовый поток:
`void OutputVector(Vector v, Stream out);`
- чтения вектора из байтового потока:
`Vector InputVector(Stream in).`

Записанный вектор должен представлять собой последовательность чисел, первым из которых является размерность вектора, а остальные числа являются значениями координат вектора.

Проверить работоспособность методов в классе `Program`, в качестве байтового потока используя файловый поток (создать файл данных в текущей папке).

Задание 2.

Добавить в класс `Vectors` следующие методы:

- записи вектора в символьный поток:
`void WriteVector(Vector v, TextWriter out);`
- чтения вектора из символьного потока:
`Vector ReadVector(Reader in).`

В данном случае рекомендуется считать, что один вектор записывается в одну строку (числа разделены пробелами, можно использовать переопределенный метод `ToString()`). Для чтения вектора из символьного потока рекомендуется использовать метод `Split()` класса `String`.

Проверить работоспособность методов в классе `Program`, в качестве текстового потока используя файловый поток (создать текстовый файл в текущей папке).

Задание 3.

Модифицировать классы `ArrayVector` и `LinkedListVector` таким образом, чтобы они были сериализуемыми.

Продемонстрировать возможности сериализации в классе `Program`, записав в файл объект, затем считав и сравнив его с исходным, для чего вывести содержимое обоих объектов на экран, можно также использовать метод `Equals()`, унаследованный от класса `Object`.

Задание 4.

Протестировать работу приложения в классе `Program`, разработать адекватный интерфейс пользователя. Необходимо отлавливать и обрабатывать все возможные исключения. Воспользоваться пользовательским интерфейсом из лабораторной работы 4.

Задание 5.

Подготовить отчет о работе.

КОД ПРОГРАММЫ

```
namespace Lab05_net6._0;

[Serializable]
public class ArrayVector : IVectorable, IComparable, ICloneable
{
    private int[] vector;

    public ArrayVector(int length)
    {
        vector = new int[length];
    }

    public ArrayVector()
    {
        vector = new int[5];
    }

    public int this[int idx]
    {
        get
        {
            if (idx < 0 || idx >= Length)
            {
                throw new IndexOutOfRangeException("Индекс за границами вектора");
            }
            return vector[idx];
        }
        set
        {
            if (idx < 0 || idx >= Length)
            {
                throw new IndexOutOfRangeException("Индекс за границами вектора");
            }
            vector[idx] = value;
        }
    }

    public int Length
    {
        get
        {
            return vector.Length;
        }
    }

    public double GetNorm()
    {
        double acc = 0;
        for (int i = 0; i < Length; i++)
        {
            acc += Math.Pow(vector[i], 2);
        }

        return Math.Sqrt(acc);
    }

    public int SumPositivesWithEvenIndex()
    {
        int acc = 0;
        for (int i = 1; i < Length; i += 2)
```

```

        {
            if (vector[i] > 0)
            {
                acc += vector[i];
            }
        }

        return acc;
    }

    public int SumLessAverageAbsoluteWithOddIndex()
    {
        if (Length == 0)
        {
            return 0;
        }

        int average = 0;
        for (int i = 0; i < Length; i++)
        {
            average += Math.Abs(vector[i]);
        }

        average /= Length;

        int acc = 0;
        for (int i = 0; i < Length; i += 2)
        {
            if (vector[i] < average)
            {
                acc += vector[i];
            }
        }

        return acc;
    }

    public int MultiplyEven()
    {
        int result = 0;
        for (int i = 1; i < Length; i+=2)
        {
            if (vector[i] > 0 && vector[i] % 2 == 0)
            {
                if (result == 0) result = 1;
                result *= vector[i];
            }
        }

        return result;
    }

    public int MultiplyOdd()
    {
        int result = 0;
        for (int i = 0; i < Length; i+=2)
        {
            if (vector[i] % 2 != 0 && vector[i] % 3 != 0)
            {
                if (result == 0) result = 1;
                result *= vector[i];
            }
        }

        return result;
    }

```

```

    }

    public void SortUp()
    {
        int n = Length;
        for (int i = 0; i < n - 1; i++)
        {
            for (int j = 0; j < n - i - 1; j++)
            {
                if (vector[j] > vector[j + 1])
                {
                    (vector[j], vector[j + 1]) = (vector[j + 1], vector[j]);
                }
            }
        }
    }

    public void SortDown()
    {
        int n = Length;
        for (int i = 0; i < n - 1; i++)
        {
            for (int j = 0; j < n - i - 1; j++)
            {
                if (vector[j] < vector[j + 1])
                {
                    (vector[j], vector[j + 1]) = (vector[j + 1], vector[j]);
                }
            }
        }
    }

    public void Log(string message = "")
    {
        if (message != "")
        {
            Console.Write(message + ": ");
        }
        Console.WriteLine(ToString());
    }

    public override string ToString()
    {
        string s = Length.ToString() + ' ';

        for (int i = 0; i < Length; i++)
        {
            s += this[i].ToString();
            if (i != Length - 1) s += ' ';
        }
        return s;
    }

    public static ArrayVector GetFromUserInput()
    {
        int length;
        do
        {
            Console.Write("Введите длину вектора: ");
        } while (!int.TryParse(Console.ReadLine(), out length) || length <=
0);

        ArrayVector vec = new ArrayVector(length);

        Random r = new Random();

```

```

        for (int i = 0; i < length; i++)
        {
            vec[i] = r.Next(100);
        }

        return vec;
    }

    public int CompareTo(object? obj)
    {
        if (!(obj is IVectorable))
        {
            throw new Exception("Можно сравнить только объекты типа
IVectorable");
        }

        IVectorable other = obj as IVectorable;

        if (Length < other.Length) return -1;
        if (Length > other.Length) return 1;
        return 0;
    }

    public override bool Equals(object? obj)
    {
        if (!(obj is IVectorable))
        {
            throw new Exception("Можно сравнивать только объекты типа
IVectorable");
        }

        IVectorable other = obj as IVectorable;

        if (Length != other.Length) return false;

        for (int i = 0; i < Length; i++)
        {
            if (this[i] != other[i]) return false;
        }

        return true;
    }

    public object Clone()
    {
        ArrayVector clone = new ArrayVector(Length);

        for (int i = 0; i < Length; i++)
        {
            clone[i] = this[i];
        }

        return clone;
    }
}
namespace Lab05_net6._0;

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

    int Length { get; }

    double GetNorm();

    void Log(string message = "");
}

```

```

}
namespace Lab05_net6._0;

[Serializable]
public class LinkedListVector : IVectorable, IComparable, ICloneable
{
    private Node head;

    [Serializable]
    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 связанного 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 связанного 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}: ");
}

```

```

        Console.WriteLine(ToString());
    }

    public override string ToString()
    {
        string s = Length.ToString() + ' ';

        for (int i = 0; i < Length; i++)
        {
            s += this[i].ToString();
            if (i != Length - 1) s += ' ';
        }
        return s;
    }

    public int CompareTo(object? obj)
    {
        if (!(obj is IVectorable))
        {
            throw new Exception("Можно сравнить только объекты типа
IVectorable");
        }

        IVectorable other = obj as IVectorable;

        if (Length < other.Length) return -1;
        if (Length > other.Length) return 1;
        return 0;
    }

    public override bool Equals(object? obj)
    {
        if (!(obj is IVectorable))
        {
            throw new Exception("Можно сравнивать только объекты типа
IVectorable");
        }

        IVectorable other = obj as IVectorable;

        if (Length != other.Length) return false;

        for (int i = 0; i < Length; i++)
        {
            if (this[i] != other[i]) return false;
        }

        return true;
    }

    public object Clone()
    {
        LinkedListVector clone = new LinkedListVector(Length);

        for (int i = 0; i < Length; i++)
        {
            clone[i] = this[i];
        }

        return clone;
    }
}

using System.Runtime.Serialization.Formatters.Binary;
using System.Text.Json;

```

```

namespace Lab05_net6._0;

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

        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" +
                "\t7 - Клонировать вектор\n" +
                "\t8 - Вывести вектора с минимальным и
максимальным количеством координат\n" +
                "\t9 - Отсортировать вектора по количеству
координат\n" +
                "\t10 - Отсортировать вектора по модулю\n" +
                "\t11 - Сравнить вектора\n" +
                "\t12 - Потоки\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;
    }
    case "7":
    {
        LogVectors(vectors);
        int idx;

        do
        {
            Console.Write("Введите номер вектора для
клонирования: ");

            inp = Console.ReadLine();
        } while (!Int32.TryParse(inp, out idx) || idx < 1 || idx
> vectors.Count);

        var vec = vectors[idx - 1];
        IVectorable clone;
        if (vec is ArrayVector)
        {
            clone = (vec as ArrayVector).Clone() as IVectorable;
        }
        else

```

```

        {
            clone = (vec as LinkedListVector).Clone() as
IVectorable;
        }

        vectors.Add(clone);

        LogVectors(vectors);

        break;
    }
    case "8":
    {
        LogVectors(vectors);

        try
        {
            int minLength = vectors[0].Length;
            int maxLength = vectors[0].Length;

            for (int i = 0; i < vectors.Count; i++)
            {
                if (minLength > vectors[i].Length)
                {
                    minLength = vectors[i].Length;
                }

                if (maxLength < vectors[i].Length)
                {
                    maxLength = vectors[i].Length;
                }
            }

            string type;
            Console.WriteLine("Вектора с минимальным значением
координат: ");

            for (int i = 0; i < vectors.Count; i++)
            {
                if (vectors[i].Length == minLength)
                {
                    type = vectors[i] is ArrayVector ?
"ArrayVector" : "LinkedListVector";
                    Console.WriteLine($"{type}:
{vectors[i].ToString()}");
                }
            }

            Console.WriteLine("Вектора с максимальным значением
координат: ");

            for (int i = 0; i < vectors.Count; i++)
            {
                if (vectors[i].Length == maxLength)
                {
                    type = vectors[i] is ArrayVector ?
"ArrayVector" : "LinkedListVector";
                    Console.WriteLine($"{type}:
{vectors[i].ToString()}");
                }
            }
        }
        catch (Exception e)
        {
            Console.WriteLine("Нет векторов");
        }

        break;
    }
}

```

```

    }
    case "9":
    {
        IVectorable tmp;
        VectorsComparer comparer = new VectorsComparer();

        for (int i = 0; i < vectors.Count - 1; i++)
        {
            for (int j = i + 1; j < vectors.Count; j++)
            {
                int compareResult;
                if (vectors[i] is ArrayVector)
                {
                    compareResult = (vectors[i] as
ArrayVector).CompareTo(vectors[j]);
                }
                else
                {
                    compareResult = (vectors[i] as
LinkedListVector).CompareTo(vectors[j]);
                }

                if (compareResult > 0)
                {
                    tmp = vectors[i];
                    vectors[i] = vectors[j];
                    vectors[j] = tmp;
                }
            }
        }

        Console.WriteLine("Список векторов после сортировки по
длине: ");

        LogVectors(vectors);

        break;
    }
    case "10":
    {
        IVectorable tmp;
        VectorsComparer comparer = new VectorsComparer();

        for (int i = 0; i < vectors.Count - 1; i++)
        {
            for (int j = i + 1; j < vectors.Count; j++)
            {
                if (comparer.Compare(vectors[i], vectors[j]) > 0)
                {
                    tmp = vectors[j];
                    vectors[j] = vectors[i];
                    vectors[i] = tmp;
                }
            }
        }

        Console.WriteLine("Список векторов после сортировки по
модулю: ");

        for (int i = 0; i < vectors.Count; ++i)
        {
            IVectorable vec = vectors[i];

            string typeView = vec is ArrayVector ? "ArrayVector"
: "LinkedListVector";

```



```

        vec.Log($"{i + 1}: {typeView}, Модуль:
{vec.GetNorm():0.00}");
    }

    break;
}
case "11":
{
    int firstIdx, secondIdx;

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

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

    if (vectors[firstIdx-1].Equals(vectors[secondIdx-1]))
    {
        Console.WriteLine("Вектора равны");
    }
    else
    {
        Console.WriteLine("Вектора не равны");
    }
    break;
}
case "12":
{
    RunStreamSubmenu(vectors);
    break;
}
default:
    Console.WriteLine("Нет такого пункта в меню");
    break;
}
}

}

public static void RunStreamSubmenu(List<IVectorable> vectors)
{
    while (true)
    {
        Console.Write("Выберете поток:\n\n" +
            "\t1. Байтовый поток\n" +
            "\t2. Символьный поток\n" +
            "\t3. Сериализация\n" +
            "\t0. Выход в главное меню\n");

        string inp = Console.ReadLine();

        switch (inp)
        {
            case "0":
                return;
            case "1":
                RunByteStream(vectors);
                break;
            case "2":

```

```

        RunSymbolStream(vectors);
        break;
    case "3":
        RunSerialization(vectors);
        break;
    default:
        Console.WriteLine("Нет такого пункта в меню");
        break;
    }
}

public static void RunByteStream(List<IVectorable> vectors)
{
    string path = "../../vectors.bin";
    if (File.Exists(path)) File.Delete(path);

    using (FileStream fs = new FileStream(path, FileMode.Append,
FileAccess.Write))
    {
        Vectors.WriteVectors(fs, vectors);
    }

    Console.WriteLine("Запись в файл `vectors.bin` выполнена");

    List<IVectorable> vectorsRead = new List<IVectorable>();
    using (FileStream fs = new FileStream(path, FileMode.Open))
    {
        vectorsRead = Vectors.ReadVectors(fs);
    }

    Console.WriteLine("Чтение из файла `vectors.bin` выполнено");

    Console.WriteLine("\nИсходный список векторов:");
    LogVectors(vectors);
    Console.WriteLine("\nСписок векторов считанный из файла:");
    LogVectors(vectorsRead);
}

public static void RunSymbolStream(List<IVectorable> vectors)
{
    string path = "../../vectors.txt";
    if (File.Exists(path)) File.Delete(path);

    using (TextWriter w = File.AppendText(path))
    {
        for (int i = 0; i < vectors.Count; i++)
        {
            Vectors.WriteVector(w, vectors[i]);
        }
        Console.WriteLine("Запись в файл `vectors.txt` выполнена");
    }

    TextReader r = File.OpenText(path);
    List<IVectorable> vectorsRead = new List<IVectorable>();
    for (int i = 0; i < vectors.Count; i++)
    {
        vectorsRead.Add(Vectors.ReadVector(r));
    }
    r.Close();

    Console.WriteLine("Чтение из файла `vectors.txt` выполнено");

    Console.WriteLine("\nИсходный список векторов:");
    LogVectors(vectors);
    Console.WriteLine("\nСписок векторов считанный из файла:");
}

```

```

        LogVectors(vectorsRead);
    }

    public static void RunSerialization(List<IVectorable> vectors)
    {
        string path = "./serialized.bin";
        List<IVectorable> vectorsRead = new List<IVectorable>();
        for (int i = 0; i < vectors.Count; i++)
        {
            FileStream serialized = File.Create(path);
            BinaryFormatter formatter = new BinaryFormatter();
            formatter.Serialize(serialized, vectors[i]);
            serialized.Close();

            FileStream deserialized = File.Open(path, FileMode.Open,
            FileAccess.Read);

            vectorsRead.Add((IVectorable)formatter.Deserialize(deserialized));
            deserialized.Close();
            Console.WriteLine($"Исходный вектор:
{vectors[i].ToString()}");
            Console.WriteLine($"Десеарилизованный вектор:
{vectorsRead[i].ToString()}");

            if (vectors[i].Equals(vectorsRead[i]))
            {
                Console.WriteLine("Вектора равны");
            }
            else
            {
                Console.WriteLine("Вектора не равны");
            }
        }
    }

    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)
    {
        IVectorable vec = vectors[i];

        string typeView = vec is ArrayVector ? "Array" : "LinkedList";

        vec.Log($"{i + 1}: {typeView}, 10}");
    }
}
}
using System.Text;

namespace Lab05_net6._0;

public class Vectors
{
    public static IVectorable Sum(IVectorable a, IVectorable b)
    {
        if (a.Length != b.Length)
        {
            throw new Exception("Vectors norms are not equals");
        }

        IVectorable vec = new ArrayVector(a.Length);
        for (int i = 0; i < vec.Length; i++)
        {
            vec[i] = a[i] + b[i];
        }

        return vec;
    }

    public static double ScalarMultiply(IVectorable a, IVectorable b)
    {
        if (a.Length != b.Length)
        {
            throw new Exception("Vectors norms are not equal");
        }

        int result = 0;
        for (int i = 0; i < a.Length; i++)
        {
            result += a[i] * b[i];
        }

        return result;
    }

    public static IVectorable MultiplyByNumber(IVectorable vector, int
number)
    {
        for (int i = 0; i < vector.Length; i++)
        {
            vector[i] *= number;
        }

        return vector;
    }

    public static double GetNormSt(IVectorable vector)
    {
        return vector.GetNorm();
    }
}

```

```

public static void WriteVectors(FileStream fs, List<IVectorable> vectors)
{
    fs.Seek(0, SeekOrigin.End);
    for (int i = 0; i < vectors.Count; i++)
    {
        string line = vectors[i].ToString();
        if (i != vectors.Count - 1) line += '\n';

        byte[] buffer = Encoding.ASCII.GetBytes(line);
        fs.Write(buffer, 0, buffer.Length);
    }
}

public static List<IVectorable> ReadVectors(FileStream fs)
{
    List<IVectorable> vectors = new List<IVectorable>();
    string content = "";
    int bytesRead;

    do
    {
        byte[] buffer = new byte[1024];
        bytesRead = fs.Read(buffer, 0, buffer.Length);

        content += Encoding.ASCII.GetString(buffer, 0, bytesRead);
    } while (bytesRead > 0);

    string[] lines = content.Split('\n');
    for (int i = 0; i < lines.Length; i++)
    {
        string[] data = lines[i].Split(' ');
        IVectorable vector = new ArrayVector(Convert.ToInt32(data[0]));
        for (int j = 0; j < vector.Length; j++)
        {
            vector[j] = Convert.ToInt32(data[j + 1]);
        }
        vectors.Add(vector);
    }

    return vectors;
}

public static void WriteVector(TextWriter w, IVectorable vec)
{
    w.WriteLine(vec.ToString());
}

public static IVectorable ReadVector(TextReader r)
{
    string[] coordinates = r.ReadLine().Split(' ');
    int length = int.Parse(coordinates[0]);

    ArrayVector vector = new ArrayVector(length);
    for (int i = 0; i < vector.Length; i++)
    {
        vector[i] = int.Parse(coordinates[i + 1]);
    }

    return vector;
}
}

```

Выберете действие:

- 1 - Сумма векторов
- 2 - Скалярное умножение
- 3 - Умножение на число
- 4 - Рассчитать модуль вектора
- 5 - Добавить вектор в список
- 6 - Удалить вектор из списка
- 7 - Клонировать вектор
- 8 - Вывести вектора с минимальным и максимальным количеством координат
- 9 - Отсортировать вектора по количеству координат
- 10 - Отсортировать вектора по модулю
- 11 - Сравнить вектора
- 12 - Потоки
- 0 - Выход

Рисунок 1 – Главное меню программы

Выберете поток:

- 1. Байтовый поток
- 2. Символьный поток
- 3. Сериализация
- 0. Выход в главное меню

Рисунок 2 – Меню работы с потоками

```
Запись в файл `vectors.txt` выполнена  
Чтение из файла `vectors.txt` выполнено
```

```
Исходный список векторов:
```

```
1:      Array: 2 69 18  
2: LinkedList: 5 48 56 95 73 49
```

```
Список векторов считанный из файла:
```

```
1:      Array: 2 69 18  
2:      Array: 5 48 56 95 73 49
```

Рисунок 3 – Запись в символьный поток

```
3  
Исходный вектор:           2 69 18  
Десеаризованный вектор: 2 69 18  
Вектора равны  
Исходный вектор:           5 48 56 95 73 49  
Десеаризованный вектор: 5 48 56 95 73 49  
Вектора равны
```

Рисунок 4 – Сериализация

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;
- байтовые и символьные потоки;
- конструкция try-catch.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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