****

**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**

Лабораторная работа №6

Основы языка С#: Делегатыпо дисциплине «Языки программирования и структуры данных»

Выполнил Маликов Н.Д.

Проверил Котенева С.Э.

Самара  
2024

ЗАДАНИЕ

Задание 1.

Организовать работу меню в программе из лабораторной работы 5 с помощью механизма делегатов.

* описать делегат без параметров, возвращающий void;
* в классе Program все пункты меню реализовать как статические методы, сигнатура методов совпадает с сигнатурой, заданной делегатом;
* запросить у пользователя траекторию выполнения программы – набор пунктов меню и добавить в делегат, в зависимости от набора пунктов меню, введенного пользователем, список статических методов, выполняющих соответствующие пункты меню;
* в методе Main() вызывать делегат на исполнение.

Задание 2.

Протестировать работу приложения в классе Program, разработать адекватный интерфейс пользователя. Необходимо отлавливать и обрабатывать все возможные исключения. Воспользоваться пользовательским интерфейсом из лабораторной работы 5.

Задание 3.

Подготовить отчет о работе.

КОД ПРОГРАММЫ

using System.Collections;

namespace lab06;

public class Comparer

{

public int Compare(object? obj1, object? obj2)

{

IVectorable vec1 = (IVectorable)obj1;

IVectorable vec2 = (IVectorable)obj2;

if (vec2.GetNorm() == vec1.GetNorm())

{

return 0;

}

else if (vec1.GetNorm() < vec2.GetNorm())

{

return -1;

}

else

{

return 1;

}

}

}

using System;

using System.Collections.Generic;

using System.Data;

using System.Dynamic;

using System.Linq;

using System.Runtime.Intrinsics;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Threading.Tasks;

namespace lab06

{

public class ArrayVector : IVectorable, IComparable

{

public int[] coordinates { get; set; }

public ArrayVector(int lenght)

{

if (lenght < 0)

{

throw new Exception("Пространство не может быть меньше чем 0!");

}

else

{

coordinates = new int[lenght];

}

}

public ArrayVector()

{

coordinates = new int[5];

}

public int this[int index]

{

get

{

if (index < 0 || index >= Length)

{

throw new Exception("Индекс за границами массива!");

}

else

{

return coordinates[index];

}

}

set

{

if (index < 0 || index >= Length)

{

throw new Exception("Индекс за границами массива!");

}

else

{

coordinates[index] = value;

}

}

}

public int Length

{

get

{

return coordinates.Length;

}

}

public double GetNorm()

{

double numbersSquared = 0;

for (int i = 0; i < coordinates.Length; i++)

{

numbersSquared += Math.Pow(coordinates[i], 2);

}

return Math.Sqrt(numbersSquared);

}

public void Log(string message = "")

{

if (message != "")

{

Console.Write(message + ": ");

}

Console.Write("{");

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

{

if (i == Length - 1)

{

Console.Write(coordinates[i]);

}

else

{

Console.Write(coordinates[i] + ", ");

}

}

Console.WriteLine("}");

}

public static ArrayVector UserInput()

{

string inp;

int length;

do

{

Console.Write("Введите длину вектора: ");

} while (!int.TryParse(Console.ReadLine(), out length) || length <= 0);

ArrayVector vec = new ArrayVector(length);

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

{

int value;

do

{

Console.Write($"Введите значение координаты {{{i + 1}}}: ");

inp = Console.ReadLine();

} while (!int.TryParse(inp, out value));

vec[i] = value;

}

return vec;

}

public int CompareTo(object? obj)

{

if (!(obj is IVectorable))

{

throw new Exception("Можно сравнить только объекты типа IVectorable");

}

IVectorable vec = (IVectorable)obj;

if (vec.Length == this.Length)

{

return 0;

}

else if (vec.Length > this.Length)

{

return -1;

}

else

{

return 1;

}

}

public override bool Equals(object? obj)

{

if (!(obj is IVectorable))

{

throw new Exception("Можно сравнивать только объекты типа IVectorable");

}

IVectorable vec = (IVectorable)obj;

if (vec.Length != this.Length)

{

return false;

}

for (int i = 0; i < vec.Length; i++)

{

if (vec[i] != this[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;

}

public override string ToString()

{

string result = Length.ToString() + ' ';

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

{

result = result + this[i].ToString() + ' ';

}

return result;

}

}

}

namespace lab06;

public interface IVectorable

{

int Length

{

get;

}

int this[int index]

{

get;

set;

}

double GetNorm();

void Log(string message = "");

}

using System.Xml.Linq;

namespace lab06;

public class LinkedListVector : IVectorable

{

private class Node

{

public int Data { get; set; }

public Node Next { get; set; }

public Node()

{

Data = 0;

Next = null;

}

public Node(int data)

{

Data = data;

Next = null;

}

}

private Node head { get; set; }

private int length;

public LinkedListVector(int length)

{

this.length = length;

head = new Node();

Node current = head;

for (int i = 1; i < length; i++)

{

current.Next = new Node();

current = current.Next;

}

}

public LinkedListVector() : this(5) { }

public int this[int index]

{

get

{

if (index < 0 || index >= length)

{

throw new IndexOutOfRangeException("Invalid index");

}

Node current = head;

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

{

current = current.Next;

}

return current.Data;

}

set

{

if (index < 0 || index >= length)

{

throw new IndexOutOfRangeException("Invalid index");

}

Node current = head;

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

{

current = current.Next;

}

current.Data = value;

}

}

public double GetNorm()

{

double norm = 0;

Node current = head;

while (current != null)

{

norm += Math.Pow(current.Data, 2);

current = current.Next;

}

return Math.Sqrt(norm);

}

public int Length

{

get { return length; }

}

public void AddElementToEnd(int element)

{

Node current = head;

while (current.Next != null)

{

current = current.Next;

}

current.Next = new Node(element);

length++;

}

public void AddElementToStart(int element)

{

Node newHead = new Node(element);

newHead.Next = head;

head = newHead;

length++;

}

public void AddElementAtPosition(int element, int position)

{

if (position < 0 || position > length)

{

throw new IndexOutOfRangeException("Invalid position");

}

if (position == 0)

{

AddElementToStart(element);

return;

}

Node current = head;

for (int i = 1; i < position; i++)

{

current = current.Next;

}

Node newNode = new Node(element);

newNode.Next = current.Next;

current.Next = newNode;

length++;

}

public void RemoveElementFromEnd()

{

if (head == null || head.Next == null)

{

throw new InvalidOperationException("Cannot remove from an empty list");

}

Node current = head;

while (current.Next.Next != null)

{

current = current.Next;

}

current.Next = null;

length--;

}

public void RemoveElementFromStart()

{

if (head == null)

{

throw new InvalidOperationException("Cannot remove from an empty list");

}

head = head.Next;

length--;

}

public void RemoveElementAtPosition(int position)

{

if (position < 0 || position >= length)

{

throw new IndexOutOfRangeException("Invalid position");

}

if (position == 0)

{

RemoveElementFromStart();

return;

}

Node current = head;

for (int i = 1; i < position; i++)

{

current = current.Next;

}

current.Next = current.Next.Next;

length--;

}

public static LinkedListVector UserInput()

{

string inp;

int length;

do

{

Console.Write("Введите длину вектора: ");

} while (!int.TryParse(Console.ReadLine(), out length) || length <= 0);

LinkedListVector vec = new LinkedListVector(length);

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

{

int value;

do

{

Console.Write($"Введите значение координаты {{{i + 1}}}: ");

inp = Console.ReadLine();

} while (!int.TryParse(inp, out value));

vec[i] = value;

}

return vec;

}

public override bool Equals(object? obj)

{

if (!(obj is IVectorable))

{

throw new Exception("Можно сравнивать только объекты типа IVectorable");

}

IVectorable vec = (IVectorable)obj;

if (vec.Length != this.Length)

{

return false;

}

for (int i = 0; i < vec.Length; i++)

{

if (vec[i] != this[i])

{

return false;

}

}

return true;

}

public int CompareTo(object? obj)

{

if (!(obj is IVectorable))

{

throw new Exception("Можно сравнить только объекты типа IVectorable");

}

IVectorable vec = (IVectorable)obj;

if (vec.Length == this.Length)

{

return 0;

}

else if (vec.Length > this.Length)

{

return -1;

}

else

{

return 1;

}

}

public object Clone()

{

ArrayVector clone = new ArrayVector(Length);

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

{

clone[i] = this[i];

}

return clone;

}

public override string ToString()

{

string result = length.ToString() + ' ';

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

{

result = result + this[i].ToString() + ' ';

}

return result;

}

public void Log(string message = "")

{

if (message != "") Console.Write($"{message}: ");

var current = head;

Console.Write("{");

while (current != null)

{

if (current.Next == null)

{

Console.Write(current.Data);

}

else

{

Console.Write(current.Data + ", ");

}

current = current.Next;

}

Console.WriteLine("}");

}

}

using lab06;

using System;

using System.Text.Json;

namespace lab06;

public static class Program

{

public delegate void TestFunctionProgramMenu(List<IVectorable> arr);

public static void Main(string[] args)

{

Console.WriteLine("Лабораторная работа #6\n 6104-020302D - Маликов Николай\n");

Console.WriteLine("Выберете класс для работы:\n\n" +

"\t1. Добавить вектор\n" +

"\t2. Убрать вектор\n" +

"\t3. Вывести вектора с минимальным и максимальным количеством координат\n" +

"\t4. Отсортировать массив векторов по возрастанию их модулей\n" +

"\t5. Клонировать вектор\n" +

"\t6. Сложение векторов\n" +

"\t7. Модуль вектора\n" +

"\t8. Скалярное произведение вектора\n" +

"\t9. Запись вектора в бинарный поток\n" +

"\t10. Запись вектора в текстовый файл\n" +

"\t11. Сериализация вектора\n");

List<IVectorable> arr = new List<IVectorable>();

TestFunctionProgramMenu menu = null;

Console.Write("Введите порядок выполнения действий программы через пробел: ");

string[] inp = Console.ReadLine().Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

foreach (string comand in inp)

{

switch (comand)

{

case "1":

menu += AddVector;

break;

case "2":

menu += RemoveVector;

break;

case "3":

menu += VectorsMinAndMaxCountCoordinates;

break;

case "4":

menu += SortArrayVectorsForUpsModules;

break;

case "5":

menu += CloneVector;

break;

case "6":

menu += SummVectors;

break;

case "7":

menu += ModuleVectors;

break;

case "8":

menu += ScalarMultVectors;

break;

case "9":

menu += BinaryFlow;

break;

case "10":

menu += FileFlow;

break;

case "11":

menu += SerializedeMethod;

break;

default:

Console.WriteLine("Нет такого пункта в меню");

break;

}

}

menu(arr);

Console.WriteLine("Программа завершила свое выполнение!");

Console.ReadKey();

}

public static void AddVector(List<IVectorable> arr)

{

try

{

arr.Add(CreateVector());

}

catch (Exception e)

{

Console.WriteLine("В данный момент это действие не возможно!");

throw;

}

}

public static void RemoveVector(List<IVectorable> arr)

{

try

{

Console.WriteLine("Укажите индекс по которому хотитете удалить вектор:");

int inp1 = Convert.ToInt32(Console.ReadLine());

arr.Remove(arr[inp1 - 1]);

LogVectors(arr);

}

catch (Exception e)

{

Console.WriteLine("В данный момент это действие не возможно!");

throw;

}

}

public static void VectorsMinAndMaxCountCoordinates(List<IVectorable> arr)

{

try

{

int minLength = arr[0].Length;

int maxLength = arr[0].Length;

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

{

if (maxLength < arr[i].Length)

{

maxLength = arr[i].Length;

}

if (minLength > arr[i].Length)

{

minLength = arr[i].Length;

}

}

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

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

{

if (arr[i].Length == minLength)

{

Console.WriteLine($"{arr[i].ToString()}");

}

}

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

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

{

if (arr[i].Length == maxLength)

{

Console.WriteLine($"{arr[i].ToString()}");

}

}

}

catch (Exception e)

{

Console.WriteLine("Вектора отсутствуют");

throw;

}

}

public static void SortArrayVectorsForUpsModules(List<IVectorable> arr)

{

IVectorable temp;

Comparer comparer = new Comparer();

for (int i = 0; i < arr.Count - 1; i++)

{

for (int j = i + 1; j < arr.Count; j++)

{

if (comparer.Compare(arr[i], arr[j]) > 0)

{

temp = arr[j];

arr[j] = arr[i];

arr[i] = temp;

}

}

}

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

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

{

IVectorable vec1 = arr[i];

vec1.Log($"{i + 1} Модуль: {vec1.GetNorm()}");

}

}

public static void CloneVector(List<IVectorable> arr)

{

LogVectors(arr);

int idx;

string input;

do

{

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

input = Console.ReadLine();

} while (!Int32.TryParse(input, out idx) || idx < 1 || idx > arr.Count);

var vec = arr[idx - 1];

IVectorable clone;

if (vec is ArrayVector)

{

clone = (vec as ArrayVector).Clone() as IVectorable;

}

else

{

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

}

arr.Add(clone);

LogVectors(arr);

}

public static void SummVectors(List<IVectorable> arr)

{

IVectorable vect = CreateVector();

vect.Log();

IVectorable vect1 = CreateVector();

vect1.Log();

try

{

IVectorable resulta = Vectors.Sum(vect, vect1);

resulta.Log();

}

catch (Exception e)

{

Console.WriteLine("Вектора разного размера!");

}

}

public static void BinaryFlow(List<IVectorable> arr)

{

IVectorable vec = CreateVector();

FileStream inpStream;

try

{

inpStream = new FileStream("forVector.bin", FileMode.CreateNew);

}

catch

{

inpStream = new FileStream("forVector.bin", FileMode.Truncate, FileAccess.Write, FileShare.None);

}

Vectors.OutputVector(vec, inpStream);

inpStream.Close();

Console.WriteLine("Успешно записан!");

FileStream outputStream = new FileStream("forVector.bin", FileMode.Open, FileAccess.Read, FileShare.None);

Console.WriteLine("\nПрочитанный вектор: " + Vectors.InputVector(outputStream).ToString());

outputStream.Close();

}

public static void FileFlow(List<IVectorable> arr)

{

IVectorable vec = CreateVector();

FileStream inpStream;

try

{

inpStream = new FileStream("forVector.txt", FileMode.CreateNew, FileAccess.Write, FileShare.None);

}

catch

{

inpStream = new FileStream("forVector.txt", FileMode.Truncate, FileAccess.Write, FileShare.None);

}

StreamWriter print = new StreamWriter(inpStream);

Vectors.WriteVector(vec, print);

Console.WriteLine("Успешно записан в .txt!");

inpStream.Close();

FileStream outputStream = new FileStream("forVector.txt", FileMode.Open, FileAccess.Read, FileShare.None);

StreamReader reader = new StreamReader(outputStream);

try

{

Console.WriteLine("Прочитанный вектор:" + Vectors.ReadVector(reader));

}

catch (System.FormatException e)

{

Console.WriteLine(e.Message);

}

outputStream.Close();

}

public static void SerializedeMethod(List<IVectorable> arr)

{

ArrayVector vec = ArrayVector.UserInput();

string serializedVector = JsonSerializer.Serialize(vec);

FileStream inpStream;

try

{

inpStream = new FileStream("forVector.json", FileMode.CreateNew, FileAccess.Write, FileShare.None);

}

catch

{

inpStream = new FileStream("forVector.json", FileMode.Truncate, FileAccess.Write, FileShare.None);

}

StreamWriter print = new StreamWriter(inpStream);

print.Write(serializedVector);

print.Close();

inpStream.Close();

Console.WriteLine("Успешно записан!");

FileStream outputStream = new FileStream("forVector.json", FileMode.Open, FileAccess.Read, FileShare.None);

Console.WriteLine("Прочитанный вектор: " + JsonSerializer.Deserialize<ArrayVector>(outputStream));

outputStream.Close();

}

public static void ModuleVectors(List<IVectorable> arr)

{

IVectorable vecc = CreateVector();

vecc.Log();

double resullt = Vectors.GetNormSt(vecc);

Console.WriteLine($"Модуль равен: {resullt}");

}

public static void ScalarMultVectors(List<IVectorable> arr)

{

IVectorable vectorr = CreateVector();

vectorr.Log();

IVectorable vectorr1 = CreateVector();

vectorr1.Log();

double result = Vectors.Scalar(vectorr, vectorr1);

Console.WriteLine($"Скаляр равен: {result}");

}

public static void TestLinkedListVectorClass(LinkedListVector vec)

{

string inp;

vec.Log("Созданный вектор");

while (true)

{

Console.WriteLine("Выберете действие:\n\n" +

"1. Добавить элемент в начало списка\n" +

"2. Добавить элемент в конец списка\n" +

"3. Добавить элемент по индексу\n" +

"4. Удалить первый элемент\n" +

"5. Удалить последний элемент\n" +

"6. Удалить элемент по индексу\n" +

"0. Продолжить\n");

inp = Console.ReadLine();

switch (inp)

{

case "1":

{

int value;

do

{

Console.Write("Введите значение для добавления: ");

inp = Console.ReadLine();

} while (!int.TryParse(inp, out value));

vec.AddElementToStart(value);

vec.Log("Обновленный вектор");

break;

}

case "2":

{

int value;

do

{

Console.Write("Введите значение для добавления: ");

inp = Console.ReadLine();

} while (!int.TryParse(inp, out value));

vec.AddElementToEnd(value);

vec.Log("Обновленный вектор");

break;

}

case "3":

{

int value;

do

{

Console.Write("Введите позицию для добавления: ");

inp = Console.ReadLine();

} while (!int.TryParse(inp, out value));

int idx;

do

{

Console.Write("Введите значение для добавления элемента: ");

inp = Console.ReadLine();

} while (!int.TryParse(inp, out idx));

try

{

vec.AddElementAtPosition(idx, value);

vec.Log("Обновленный вектор");

}

catch (IndexOutOfRangeException e)

{

Console.WriteLine("Введенный индекс выходит за рамки связного списка");

}

break;

}

case "4":

{

try

{

vec.RemoveElementFromStart();

vec.Log("Обновленный вектор");

}

catch

{

Console.WriteLine("Пустой вектор");

}

break;

}

case "5":

{

try

{

vec.RemoveElementFromEnd();

vec.Log("Обновленный вектор");

}

catch

{

Console.WriteLine("Пустой вектор");

}

break;

}

case "6":

{

int idx;

do

{

Console.Write("Введите индекс элемента который хотите удалить: ");

inp = Console.ReadLine();

} while (!int.TryParse(inp, out idx));

try

{

vec.RemoveElementAtPosition(idx);

vec.Log("Обновленный вектор");

}

catch (IndexOutOfRangeException e)

{

Console.WriteLine("Введенный индекс выходит за рамки связного списка");

}

break;

}

case "0":

{

return;

}

default:

{

Console.WriteLine("Нет такого пункта в меню");

break;

}

}

}

}

public static IVectorable CreateVector()

{

while (true)

{

Console.WriteLine("Вектор какого типа вы хотите создать?\n" +

"\t1. LinkedListVector\n" +

"\t2. ArrayVector\n");

string Input = Console.ReadLine();

switch (Input)

{

case "1":

string inp;

int length;

LinkedListVector vec = LinkedListVector.UserInput();

TestLinkedListVectorClass(vec);

return vec;

break;

case "2":

ArrayVector vector = ArrayVector.UserInput();

return vector;

break;

default:

Console.WriteLine("Такого пункта нет в меню");

break;

}

}

}

public static void LogVectors(List<IVectorable> vectors)

{

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

{

IVectorable vec = vectors[i];

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

vec.Log($"{i + 1}: {typeView}");

}

}

}

namespace lab06

{

public static class Vectors

{

public static IVectorable Sum(IVectorable vectorFirst, IVectorable vectorSecond)

{

if (vectorFirst.Length != vectorSecond.Length)

{

throw new Exception("Вектора разного размера!");

}

else

{

ArrayVector Sum = new ArrayVector(vectorFirst.Length);

for (int i = 0; i < vectorFirst.Length; i++)

{

Sum[i] = vectorFirst[i] + vectorSecond[i];

}

return Sum;

}

}

public static double Scalar(IVectorable vectorFirst, IVectorable vectorSecond)

{

if (vectorFirst.Length != vectorSecond.Length)

{

throw new Exception("Вектора разного размера!");

}

else

{

double multi = 0;

for (int i = 0; i < vectorFirst.Length; i++)

{

multi += vectorFirst[i] \* vectorSecond[i];

}

return multi;

}

}

public static double GetNormSt(IVectorable vector)

{

return vector.GetNorm();

}

public static void OutputVector(IVectorable vector, Stream stream)

{

BinaryWriter bw = new BinaryWriter(stream);

bw.Write(vector.Length);

for (int i = 0; i < vector.Length; i++)

{

bw.Write(vector[i]);

}

bw.Close();

}

public static IVectorable InputVector(Stream stream)

{

BinaryReader br = new BinaryReader(stream);

int length = br.ReadInt32();

ArrayVector vector = new ArrayVector(length);

for (int i = 0; i < vector.Length; i++)

{

vector[i] = br.ReadInt32();

}

br.Close();

return (IVectorable)vector;

}

public static void WriteVector(IVectorable vector, TextWriter print)

{

print.Write(vector);

print.Close();

}

public static IVectorable ReadVector(TextReader reader)

{

string[] stringVector = reader.ReadToEnd().Trim().Split(' ');

ArrayVector vector = new ArrayVector(stringVector.Length - 1);

try

{

for (int i = 1; i < stringVector.Length; i++)

{

vector[i - 1] = int.Parse(stringVector[i]);

}

}

catch

{

throw new FormatException("Ошибка, несуществующие значения");

}

reader.Close();

return (IVectorable)vector;

}

}

}

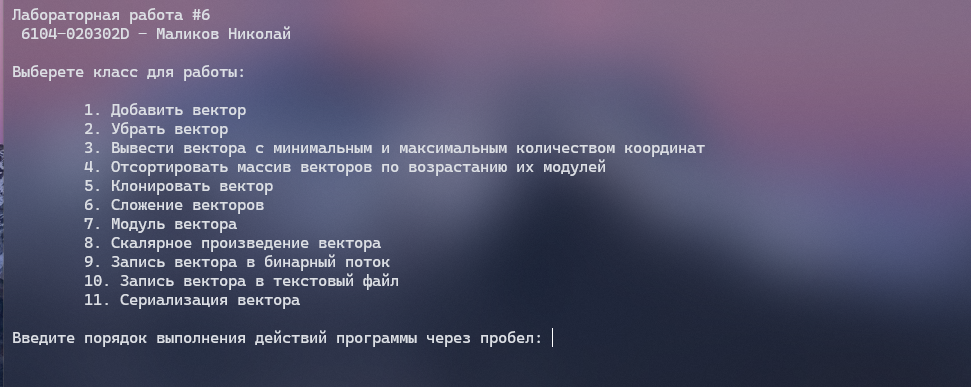


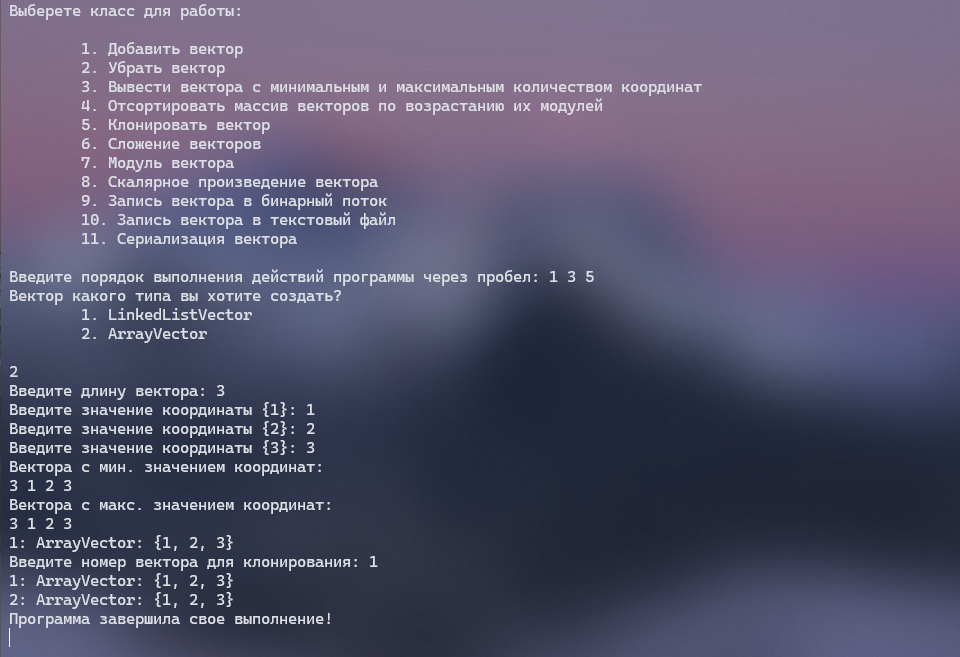
Рисунок 1 – Главное меню 

Рисунок 2 – Работа делегата

ВЫВОДЫ

В лабораторной работе были использованы конструкции языка:

* пользовательские классы;
* методы;
* свойства;
* конструкторы;
* индексаторы;
* циклы while, for, foreach;
* конструкция switch;
* пользовательские интерфейсы;
* переопределение методов родительских классов;
* потоки данных;
* делегирование.

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.