

**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 № 06

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

Выполнил Фокин Е.А., 6103

Проверил Котенева С.Э.

Самара  
2024

ЗАДАНИЕ

Задание 0.

Прочитать теоретический материал.

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.Runtime.Serialization.Formatters.Binary;

delegate void MenuAction();

namespace Lab6

{

public class Program

{

public static void Main()

{

while (true)

{

MenuAction menuAction = WriteMenuActions;

menuAction();

string selectedAction = Console.ReadLine()!.Trim();

switch (selectedAction)

{

case "0":

return;

case "1":

menuAction += TestInputOutputVectors;

break;

case "2":

menuAction += TestWriteReadVectros;

break;

case "3":

menuAction += TestSerialization;

break;

case "4":

menuAction += Run4LabMenu;

break;

default:

Console.WriteLine("Не выбран ни один пункт меню");

break;

}

menuAction();

}

}

private static void TestInputOutputVectors()

{

Console.Clear();

string FILE\_NAME = "test-io-vectors.bin";

IVectorable[] vectors = Utility.GetRadnomVectors();

Console.WriteLine("Исходный массив векторов:");

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

{

Console.WriteLine(i + ") " + vectors[i]);

}

FileStream outputStream = File.Create(FILE\_NAME);

Vectors.OutputVectors(vectors, outputStream);

outputStream.Close();

FileStream inputStream = File.OpenRead(FILE\_NAME);

IVectorable[] newVectors = Vectors.InputVectors(inputStream);

inputStream.Close();

Console.WriteLine();

Utility.TestVectorsEquality(vectors, newVectors);

}

private static void TestWriteReadVectros()

{

Console.Clear();

string FILE\_NAME = "test-wr-vectors.txt";

IVectorable[] vectors = Utility.GetRadnomVectors();

Console.WriteLine("Исходный массив векторов:");

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

{

Console.WriteLine(i + ") " + vectors[i]);

}

StreamWriter streamWriter = File.CreateText(FILE\_NAME);

Vectors.WriteVectors(vectors, streamWriter);

streamWriter.Close();

StreamReader streamReader = File.OpenText(FILE\_NAME);

IVectorable[] newVectors = Vectors.ReadVectors(streamReader);

streamReader.Close();

Console.WriteLine();

Utility.TestVectorsEquality(vectors, newVectors);

}

private static void TestSerialization()

{

Console.Clear();

string FILE\_NAME\_AV = "test-serialization-av.bat";

string FILE\_NAME\_LLV = "test-serialization-llv.bat";

IVectorable vectorA = Utility.GetRandomAV();

IVectorable vectorLL = Utility.GetRandomLLV();

Console.WriteLine("Исходный ArrayVectror: " + vectorA);

Console.WriteLine("Исходный LinkedListVector: " + vectorLL);

FileStream fileStreamA = File.Create(FILE\_NAME\_AV);

FileStream fileStreamLL = File.Create(FILE\_NAME\_LLV);

BinaryFormatter serializerA = new BinaryFormatter();

BinaryFormatter serializerLL = new BinaryFormatter();

serializerA.Serialize(fileStreamA, vectorA);

serializerLL.Serialize(fileStreamLL, vectorLL);

fileStreamA.Close();

fileStreamLL.Close();

fileStreamA = File.OpenRead(FILE\_NAME\_AV);

fileStreamLL = File.OpenRead(FILE\_NAME\_LLV);

IVectorable newVectorA = (IVectorable)serializerA.Deserialize(fileStreamA)!;

IVectorable newVectorLL = (IVectorable)serializerA.Deserialize(fileStreamLL)!;

fileStreamA.Close();

fileStreamLL.Close();

Console.WriteLine("Десериализованный ArrayVectror: " + newVectorA);

Console.WriteLine("Десериализованный LinkedListVector: " + newVectorLL);

Utility.TestVectorsEquality([vectorA, vectorLL], [newVectorA, newVectorLL]);

}

private static void Run4LabMenu()

{

Console.Clear();

RunMenu(GetPrevLabMenu());

}

private static void WriteMenuActions()

{

Console.WriteLine("\nВыберите один из пунтов меню");

Console.WriteLine("0 - Осознанное завершение");

Console.WriteLine("1 - Input/Output Vectors");

Console.WriteLine("2 - Write/Read Vectors");

Console.WriteLine("3 - Сериализация");

Console.WriteLine("4 - Запуск меню 4ой лаборторной работы");

}

private static class Utility

{

public static void FillVectorRandomValues(IVectorable vector)

{

Random random = new Random();

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

{

vector[i + 1] = random.Next(-1000, 1001);

}

}

public static IVectorable[] GetRadnomVectors()

{

IVectorable vector1 = new ArrayVector(3);

FillVectorRandomValues(vector1);

IVectorable vector2 = new ArrayVector(4);

FillVectorRandomValues(vector2);

IVectorable vector3 = new LinkedListVector(2);

FillVectorRandomValues(vector3);

IVectorable vector4 = new LinkedListVector(7);

FillVectorRandomValues(vector4);

IVectorable vector5 = new ArrayVector(6);

FillVectorRandomValues(vector5);

IVectorable vector6 = new LinkedListVector(2);

FillVectorRandomValues(vector6);

IVectorable vector7 = new LinkedListVector(7);

FillVectorRandomValues(vector7);

return new IVectorable[]

{

vector1,

vector2,

vector3,

vector4,

vector5,

vector6,

vector7

};

}

public static IVectorable[] GetVectorsArray()

{

IVectorable vector1 = new ArrayVector(3);

vector1[1] = -1000;

vector1[2] = 2000;

vector1[3] = 3000;

IVectorable vector2 = new ArrayVector(4);

vector2[1] = 1;

vector2[2] = 2;

vector2[3] = 3;

vector2[4] = 40;

IVectorable vector3 = new LinkedListVector(2);

vector3[1] = 100;

vector3[2] = 200;

IVectorable vector4 = new LinkedListVector(7);

vector4[1] = 1;

vector4[2] = 2;

vector4[3] = 3;

vector4[4] = 4;

vector4[5] = 5;

vector4[6] = 6;

vector4[7] = 7;

IVectorable vector5 = new ArrayVector(6);

vector5[1] = 2;

vector5[2] = 4;

vector5[3] = 6;

vector5[4] = 8;

vector5[5] = 10;

vector5[6] = 12;

IVectorable vector6 = new LinkedListVector(2);

vector6[1] = 1;

vector6[2] = 3;

IVectorable vector7 = new LinkedListVector(7);

vector7[1] = 1;

vector7[2] = 3;

vector7[3] = 5;

vector7[4] = 7;

vector7[5] = 9;

vector7[6] = 11;

vector7[7] = 13;

return new IVectorable[]

{

vector1,

vector2,

vector3,

vector4,

vector5,

vector6,

vector7

};

}

public static LinkedListVector GetRandomLLV()

{

LinkedListVector vector = new LinkedListVector(7);

FillVectorRandomValues(vector);

return vector;

}

public static ArrayVector GetRandomAV()

{

ArrayVector vector = new ArrayVector(5);

FillVectorRandomValues(vector);

return vector;

}

public static void TestVectorsEquality(IVectorable[] vectors, IVectorable[] newVectors)

{

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

{

if (vectors[i].Equals(newVectors[i]))

{

Console.WriteLine(i + ") (+) " + "Вектор { " + newVectors[i] + " } прошел проверку методом Equals после чтения из файла");

}

else

{

Console.WriteLine(i + ") (-) " + "Вектор { " + newVectors[i] + " } не прошел проверку методом Equals после чтения из файла");

}

}

}

}

private static Dictionary<string, Action> GetPrevLabMenu()

{

return new Dictionary<string, Action>{

{"Работа с классом ArrayVector", () => {

Console.WriteLine("Введите значения вектора через пробел");

int[] vectorElems = Console.ReadLine()!.Trim().Split(" ").Select((el) => int.Parse(el)).ToArray();

ArrayVector vector = new ArrayVector(vectorElems.Length);

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

vector[i+1] = vectorElems[i];

}

Console.WriteLine("Модуль вектора: " + vector.GetNorm());

Console.WriteLine("Размерность вектора: ", vector.Length);

Console.WriteLine("Введите индекс элемента, который хотите получить");

int index = int.Parse(Console.ReadLine()!);

RunWithCatch("Элемент с индексом " + index + ": ", () => vector[index]);

Console.WriteLine("Введите индекс элемента, который хотите изменить");

index = int.Parse(Console.ReadLine()!);

RunWithCatch("Найден элемент с индексом " + index + ": ", () => vector[index]);

Console.WriteLine("Введите значение (целое число), на которое хотите изменить");

int value = int.Parse(Console.ReadLine()!);

RunWithCatch("Значение изменено, теперь вектор выглядит так: ", () => {

vector[index] = value;

return vector;

});

}},

{"Работа с классом Vectors", () => {

Console.WriteLine("Выберите тип 1го вектора");

Console.WriteLine("1 - ArrayVector");

Console.WriteLine("2 - LinkedListVector");

int selectedVectorType = int.Parse(Console.ReadLine()!);

Console.WriteLine("Введите значения 1го вектора через пробел");

int[] vectorElems = Console.ReadLine()!.Trim().Split(" ").Select((el) => int.Parse(el)).ToArray();

IVectorable vector1;

switch (selectedVectorType)

{

case 1:

{

vector1 = new ArrayVector(vectorElems.Length);

break;

}

case 2: {

vector1 = new LinkedListVector(vectorElems.Length);

break;

}

default:

Console.WriteLine("Выбрал неверный тип вектора");

return;

}

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

vector1[i+1] = vectorElems[i];

}

Console.WriteLine("Выберите тип 2го вектора");

Console.WriteLine("1 - ArrayVector");

Console.WriteLine("2 - LinkedListVector");

int selectedVectorType2 = int.Parse(Console.ReadLine()!);

Console.WriteLine("Введите значения 2го вектора через пробел");

int[] vectorElems2 = Console.ReadLine()!.Trim().Split(" ").Select((el) => int.Parse(el)).ToArray();

IVectorable vector2;

switch (selectedVectorType2)

{

case 1:

{

vector2 = new ArrayVector(vectorElems.Length);

break;

}

case 2: {

vector2 = new LinkedListVector(vectorElems.Length);

break;

}

default:

Console.WriteLine("Выбрал неверный тип вектора");

return;

}

for(int i = 0; i < vectorElems2.Length; i++) {

vector2[i+1] = vectorElems2[i];

}

RunWithCatch("Сумма векторов: ", () => Vectors.Sum(vector1, vector2));

RunWithCatch("Скалярное произведение: ", () => Vectors.Scalar(vector1, vector2));

}},

{"Работа с классом LinkedListVector", () => {

Console.Write("Введите размерность: ");

int size = int.Parse(Console.ReadLine()!);

LinkedListVector list = new LinkedListVector(size);

Console.WriteLine("Список: " + list);

RunMenu(new Dictionary<string, Action> {

{"Вывести список и размерность", () => {

Console.WriteLine("Список: " + list );

Console.WriteLine("Размерность: " + list.Length);

}},

{"Вставить в начало", () => {

Console.Write("Введите число, которое вставить: ");

int num = int.Parse(Console.ReadLine()!);

list.InsertStart(num);

Console.WriteLine("Список: " + list);

Console.WriteLine("Размерность списка: " + list.Length);

}},

{"Удалить из начала", () => {

list.DeleteStart();

Console.WriteLine("Список: " + list);

Console.WriteLine("Размерность списка: " + list.Length);

}},

{"Вставить в конец", () => {

Console.Write("Введите число, которое вставить: ");

int num = int.Parse(Console.ReadLine()!);

list.InsertEnd(num);

Console.WriteLine("Список: " + list);

Console.WriteLine("Размерность списка: " + list.Length);

}},

{"Удалить из конца", () => {

list.DeleteEnd();

Console.WriteLine("Список: " + list);

Console.WriteLine("Размерность списка: " + list.Length);

}},

{"Вставить по индексу", () => {

Console.Write("Введите индекс: ");

int index = int.Parse(Console.ReadLine()!);

Console.WriteLine("Найден элемент с индексом " + index + ": " + list[index]);

Console.Write("Введите значение (целое число): ");

int value = int.Parse(Console.ReadLine()!);

list.InsertByIndex(index, value);

Console.WriteLine("Вставка произошла успешно, новый список: " + list);

}},

{"Удалить по индексу", () => {

Console.Write("Введите индекс: ");

int index = int.Parse(Console.ReadLine()!);

list.DeleteByIndex(index);

Console.WriteLine("Удаление произошла успешно, новый список: " + list);

}},

{"Модуль списка", () => {

Console.WriteLine("Модуль списка: " + list.GetNorm());

}},

{"Получить элемент по индексу", () => {

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

int index = int.Parse(Console.ReadLine()!);

RunWithCatch("Элемент с индексом " + index + ": ", () => list[index]);

}},

{"Изменить элемент по индексу", () => {

Console.WriteLine("Введите индекс элемента, который хотите изменить");

int index = int.Parse(Console.ReadLine()!);

Console.WriteLine("Найден элемент с индексом " + index + ": " + list[index]);

Console.WriteLine("Введите значение (целое число), на которое хотите изменить");

int value = int.Parse(Console.ReadLine()!);

list[index] = value;

Console.WriteLine("Значение изменено, теперь список выглядит так: " + list);

}}

});

}},

{"Сравнение векторов", () => {

IVectorable[] vectors = Utility.GetVectorsArray();

IVectorable minCoordinatesVector = vectors[0];

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

{

if (vectors[i].CompareTo(minCoordinatesVector) < 0)

{

minCoordinatesVector = vectors[i];

}

}

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

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

if (vectors[i].Length == minCoordinatesVector.Length) {

minCoordinates.Add(vectors[i]);

}

}

Console.WriteLine("Вектора с минимальным числом координат: ");

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

Console.WriteLine(i + 1 + ") " + minCoordinates[i]);

}

IVectorable maxCoordinatesVector = vectors[0];

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

{

if (vectors[i].CompareTo(maxCoordinatesVector) > 0)

{

maxCoordinatesVector = vectors[i];

}

}

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

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

if (vectors[i].Length == maxCoordinatesVector.Length) {

maxCoordinates.Add(vectors[i]);

}

}

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

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

Console.WriteLine(i + 1 + ") " + maxCoordinates[i]);

}

Array.Sort(vectors, new VectorAscComparer());

Console.WriteLine("Отсортированный по возрастанию модуля массив векторов: ");

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

{

Console.WriteLine("Вектор: " + vectors[i] + "; Модуль: " + vectors[i].GetNorm());

}

Console.WriteLine();

}},

{"Клонирование векторов", () => {

ArrayVector vector = new ArrayVector(3);

vector[1] = 1;

vector[2] = 2;

vector[3] = 3;

IVectorable clone = vector.Clone() as IVectorable;

Console.WriteLine("Вектор: " + vector + "; " + "клон: " + clone);

Console.WriteLine("Сравнение оригинала и клона с помощью == выдает: " + (vector == clone));

Console.WriteLine("Сравнение оригинала и клона с помощью == Equals(): " + (vector.Equals(clone)));

clone[1] = -1;

Console.WriteLine("Вектор: " + vector + "; " + "клон: " + clone);

Console.WriteLine("Сравнение оригинала и измененного клона с помощью == выдает: " + (vector == clone));

Console.WriteLine("Сравнение оригинала и измененного клона с помощью == Equals(): " + (vector.Equals(clone)));

}},

};

}

private static void RunMenu(Dictionary<string, Action> menuActions)

{

while (true)

{

Console.WriteLine("\nВыберите один из пунтов меню");

Console.WriteLine("0 - Осознанное завершение");

for (int index = 0; index < menuActions.Count; index++)

{

KeyValuePair<string, Action> item = menuActions.ElementAt(index);

Console.WriteLine($"{index + 1} - {item.Key}");

}

string menuAction = Console.ReadLine()!.Trim();

bool parsed = int.TryParse(menuAction, out int actionIndex);

if (!parsed || 0 > actionIndex || actionIndex > menuActions.Count)

{

Console.WriteLine("Не выбран ни один пункт меню\n");

continue;

}

else if (actionIndex == 0)

{

Console.Clear();

return;

}

else

{

try

{

Console.Clear();

Console.WriteLine(menuActions.ElementAt(actionIndex - 1).Key + "\n");

menuActions.ElementAt(actionIndex - 1).Value();

}

catch (IndexOutOfRangeException)

{

Console.WriteLine("Ошибка: индекс вышел за переделы");

}

catch (Exception e)

{

Console.WriteLine("Ошибка: " + e.Message);

}

}

}

}

private static void RunWithCatch(string message, Func<object> action)

{

try

{

object actionResult = (object)action();

Console.WriteLine(message + actionResult);

}

catch (IndexOutOfRangeException)

{

Console.WriteLine("Ошибка: индекс вышел за переделы");

}

catch (Exception e)

{

Console.WriteLine("Ошибка: " + e.Message);

}

}

}

}

namespace Lab6

{

public class VectorAscComparer : IComparer<IVectorable>

{

public int Compare(IVectorable? vector1, IVectorable? vector2)

{

if (vector1 == null || vector2 == null)

{

return -1;

}

return vector1.GetNorm().CompareTo(vector2.GetNorm());

}

}

}

namespace Lab6

{

[Serializable]

public class ArrayVector : IVectorable

{

private int[] vector;

public ArrayVector() : this(5) { }

public ArrayVector(int size)

{

if (size < 1)

{

throw new Exception("Размерность не может быть отрицательной");

}

vector = new int[size];

}

public int Length

{

get

{

return vector.Length;

}

}

public int this[int index]

{

get

{

return vector[index - 1];

}

set

{

vector[index - 1] = value;

}

}

public double GetNorm()

{

int sum = 0;

foreach (int el in vector)

{

sum += el \* el;

}

return Math.Sqrt(sum);

}

public override string ToString()

{

string res = Length + "";

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

{

res += " " + this[i + 1];

}

return res;

}

public override bool Equals(Object? obj)

{

IVectorable vector = obj as IVectorable;

if (obj == null || vector.Length != Length)

{

return false;

}

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

{

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

{

return false;

}

}

return true;

}

public int CompareTo(object? other)

{

if (other == null)

{

return -1;

}

return Length.CompareTo((other as IVectorable).Length);

}

public object Clone()

{

ArrayVector clone = new ArrayVector(Length);

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

{

clone[i] = this[i];

}

return clone;

}

}

}

namespace Lab6

{

public interface IVectorable : IComparable, ICloneable

{

int this[int index] { get; set; }

int Length { get; }

double GetNorm();

}

}

namespace Lab6

{

[Serializable]

public class LinkedListVector : IVectorable

{

private Node? head;

public LinkedListVector(int length)

{

head = new Node();

Node node = head;

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

{

node.next = new Node();

node = node.next;

}

}

public LinkedListVector() : this(5) { }

public int Length

{

get

{

int length = 0;

Node? node = head;

while (node != null)

{

node = node.next;

length++;

}

return length;

}

}

public int this[int index]

{

get

{

Node node = GetNodeByIndex(index - 1);

return node.value;

}

set

{

Node node = GetNodeByIndex(index - 1);

node.value = value;

}

}

public double GetNorm()

{

int sum = 0;

Node? node = head;

while (node != null)

{

sum += node.value \* node.value;

node = node.next;

}

return Math.Sqrt(sum);

}

public void InsertStart(int value)

{

Node node = new Node(value, head);

head = node;

}

public void DeleteStart()

{

if (Length == 0)

{

return;

}

head = head?.next;

}

public void InsertEnd(int value)

{

if (head == null)

{

head = new Node(value);

}

else

{

Node node = GetNodeByIndex(Length - 1);

node.next = new Node(value);

}

}

public void DeleteEnd()

{

if (head == null)

{

return;

}

if (head.next == null)

{

head = null;

return;

}

Node node = GetNodeByIndex(Length - 2);

node.next = null;

}

public void InsertByIndex(int index, int value)

{

if (index == Length)

{

Node prevCurrNode = GetNodeByIndex(index - 2);

prevCurrNode.next = new Node(value, prevCurrNode.next);

}

else if (index == 1)

{

InsertStart(value);

}

else

{

Node prevCurrNode = GetNodeByIndex(index - 2);

prevCurrNode.next = new Node(value, prevCurrNode.next);

}

}

public void DeleteByIndex(int index)

{

if (index < 1 || index > Length)

{

throw new IndexOutOfRangeException();

}

if (index == Length)

{

DeleteEnd();

}

else if (index == 1)

{

DeleteStart();

}

else

{

Node prevCurrNode = GetNodeByIndex(index - 2);

prevCurrNode.next = prevCurrNode.next.next;

}

}

public override string ToString()

{

string res = Length + "";

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

{

res += " " + GetNodeByIndex(i).value;

}

return res;

}

public override bool Equals(Object? obj)

{

IVectorable vector = obj as IVectorable;

if (obj == null || vector.Length != Length)

{

return false;

}

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

{

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

{

return false;

}

}

return true;

}

public int CompareTo(object? other)

{

if (other == null)

{

return -1;

}

return Length.CompareTo((other as IVectorable).Length);

}

public object Clone()

{

LinkedListVector clone = new LinkedListVector(Length);

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

{

clone[i] = this[i];

}

return clone;

}

private Node GetNodeByIndex(int index)

{

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

{

throw new IndexOutOfRangeException();

}

Node? node = head;

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

{

node = node?.next;

}

return node;

}

[Serializable]

private class Node

{

public int value;

public Node? next;

public Node(int nodeValue = 0, Node? nextNode = null)

{

value = nodeValue;

next = nextNode;

}

}

}

}

using System.Text;

namespace Lab6

{

public class Vectors

{

public static IVectorable Sum(IVectorable vector1, IVectorable vector2)

{

if (vector1.Length != vector2.Length)

{

throw new FormatException("Размерность векторов должна быть одинаковой");

}

ArrayVector result = new ArrayVector(vector1.Length);

for (int i = 1; i < vector1.Length + 1; i++)

{

result[i] = vector1[i] + vector2[i];

}

return result;

}

public static double Scalar(IVectorable vector1, IVectorable vector2)

{

if (vector1.Length != vector2.Length)

{

throw new FormatException("Размерность векторов должна быть одинаковой");

}

double result = 0;

for (int i = 1; i < vector1.Length + 1; i++)

{

result += vector1[i] \* vector2[i];

}

return result;

}

public static double GetNormSt(IVectorable vector)

{

return vector.GetNorm();

}

public static void OutputVectors(IVectorable[] vectors, Stream stream)

{

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

{

byte[] buffer = Encoding.UTF8.GetBytes(vectors[i].ToString() + "\n");

stream.Write(buffer, 0, buffer.Length);

}

}

public static IVectorable[] InputVectors(Stream stream)

{

byte[] buffer = new byte[stream.Length];

stream.Read(buffer, 0, buffer.Length);

return ParseVectors(Encoding.UTF8.GetString(buffer));

}

public static void WriteVectors(IVectorable[] vectors, TextWriter writer)

{

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

{

writer.WriteLine(vectors[i]);

}

}

public static IVectorable[] ReadVectors(TextReader reader)

{

string vectorsString = reader.ReadToEnd();

return ParseVectors(vectorsString);

}

private static IVectorable[] ParseVectors(string vectorsStr)

{

vectorsStr = vectorsStr.Trim();

string[] vectorsStrArr = vectorsStr.Split("\n");

IVectorable[] vectors = new IVectorable[vectorsStrArr.Length];

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

{

string[] vectorStrArr = vectorsStrArr[i].Trim().Split();

int size = int.Parse(vectorStrArr[0]);

IVectorable vector = new ArrayVector(size);

for (int j = 1; j < vectorStrArr.Length; j++)

{

vector[j] = int.Parse(vectorStrArr[j]);

}

vectors[i] = vector;

}

return vectors;

}

}

}

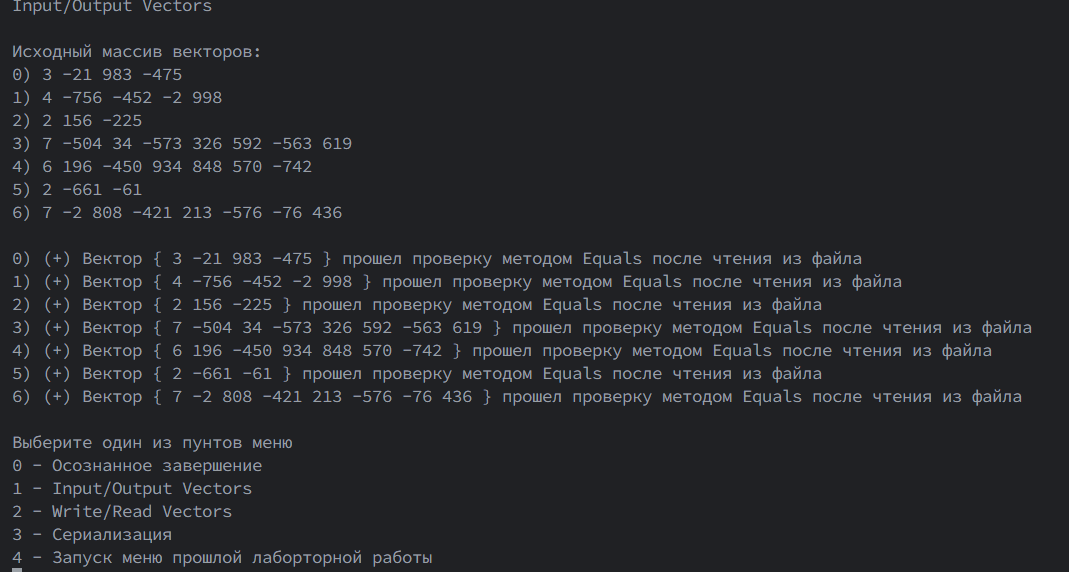


Рисунок 1 – Пример работы программыВЫВОДЫ

В лабораторной работе были использованы возможности языка C#:

* ветвления if;
* цикл с параметром for;
* цикл foreach;
* методы;
* классы;
* индексатор.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

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