****

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

Институт (факультет) Информатики и кибернетики

Кафедра Программных систем

**ОТЧЁТ**

**по лабораторной работе**

№6 «Потоки»  
по дисциплине «Основы программирования»

Выполнил Мананников М.А. 6103- 020302D

Проверил Котенева Светлана Эдуардовна

Самара  
2022

ЗАДАНИЕ

Задание 1.

Прочитать теоретический материал.

Задание 2.

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

Задание 3.

Добавить в класс Vectors следующие методы:

* записи вектора в символьный поток void WriteVector(Vector v, TextWriter out),
* чтения вектора из символьного потока Vector ReadVector(TextReader in).

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

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

Задание 4.

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

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

Задание 5.

Протестировать работу приложения в классе Program, доработав интерфейс пользователя из лабораторной работы 4.

Задание 6.

Подготовить отчет о работе.

**КОДПРОГРАММЫ**

using System;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

namespace ЛР\_06

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Лабораторная работа №6: Потоки\nВыполнил Мананников Максим, студент группы 6103-020302D");

while (true)

{

Console.WriteLine("Выберите пункт меню:\n0 - Выход\n1 - Запись векторов в байтовый поток\n2 - Чтение векторов из байтового потока\n3 - Запись векторов в символьный поток\n4 - Чтение из символьного потока\n5 - Сериализация векторов");

switch (Console.ReadLine())

{

case "1":

Task1();

break;

case "2":

Task2();

break;

case "3":

Task3();

break;

case "4":

Task4();

break;

case "5":

Task5();

break;

case "0":

return;

default:

Console.WriteLine("Ошибка ввода! Повторите попытку!");

break;

}

}

}

static void Task1()

{

try

{

Console.WriteLine("Запись вектора в байтовый поток\n");

IVectorable[] vector = Massive();

try

{

FileStream file\_1 = new FileStream("bytes.txt", FileMode.Create, FileAccess.Write);

Vectors vec = new Vectors();

vec.OutputVector(vector, file\_1);

file\_1.Close();

Console.WriteLine("Вектор записан в файл успешно!");

}

catch (IOException)

{

Console.WriteLine("Ошибка! Повторите попытку!");

}

}

catch

{

throw new Exception("Ошибка! Повторите попытку!");

}

}

static void Task2()

{

try

{

try

{

using (FileStream file\_2 = new FileStream("bytes.txt", FileMode.Open, FileAccess.Read))

{

Vectors vec = new Vectors();

IVectorable[] vector = vec.InputVector(file\_2);

Console.WriteLine("Чтение вектора с файла через байтовый поток");

Console.WriteLine();

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

{

Console.WriteLine(vector[i].ToString());

}

}

}

catch (IOException)

{

Console.WriteLine("Ошибка! Файл пустой!");

}

}

catch (Exception)

{

Console.WriteLine("Ошибка! Повторите попытку!");

}

}

static void Task3()

{

try

{

Console.WriteLine("Запись вектора в символьный поток\n");

IVectorable[] vector = Massive();

try

{

StreamWriter file\_3 = new StreamWriter("symbols.txt", true);

Vectors vec = new Vectors();

vec.WriteVector(vector, file\_3);

file\_3.Close();

Console.WriteLine("Вектор записан в файл успешно!");

}

catch (IOException)

{

Console.WriteLine("Ошибка! Повторите попытку!");

}

}

catch (Exception)

{

Console.WriteLine("Ошибка! Повторите попытку!");

}

}

static void Task4()

{

try

{

try

{

using (StreamReader file\_4 = new StreamReader("symbols.txt"))

{

Vectors vec = new Vectors();

IVectorable[] vector = vec.ReadVector(file\_4);

Console.WriteLine("Чтение вектора с файла через символьный поток");

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

{

Console.WriteLine(vector[i].ToString());

}

}

}

catch (IOException)

{

Console.WriteLine("Ошибка! Файл пустой!");

}

}

catch (Exception)

{

Console.WriteLine("Ошибка! Повторите попытку!");

}

}

static void Task5()

{

try

{

Console.Write("Введите количество координат вектора: ");

int len = int.Parse(Console.ReadLine());

IVectorable vec\_1 = new ArrayVector(len);

Console.WriteLine("Введите координаты вектора: ");

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

{

vec\_1[i] = int.Parse(Console.ReadLine());

}

BinaryFormatter bin\_format = new BinaryFormatter();

using (FileStream stream = new FileStream("task\_5.bin", FileMode.Create, FileAccess.Write))

{

bin\_format.Serialize(stream, vec\_1);

}

Console.WriteLine();

Console.WriteLine("Вектор сохранен в файл! Сериализация прошла успешно!");

using (FileStream stream\_2 = new FileStream("task\_5.bin", FileMode.Open, FileAccess.Read))

{

IVectorable vec\_2 = (IVectorable)bin\_format.Deserialize(stream\_2);

Console.WriteLine();

Console.Write("Прочитанный с потока вектор: " + vec\_2.ToString());

Console.WriteLine();

}

}

catch (Exception)

{

Console.WriteLine("Ошибка! Повторите попытку!");

}

}

static IVectorable[] Massive()

{

while (true)

{

Console.Write("Введите количество векторов: ");

int count = int.Parse(Console.ReadLine());

if (count > 0)

{

IVectorable[] massive = new IVectorable[count];

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

{

while (true)

{

Console.WriteLine("Выберите тип вектора:\n1 - Массив\n2 - Список");

switch (Console.ReadLine())

{

case "1":

try

{

while (true)

{

Console.WriteLine("Вектор номер " + (i + 1));

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

int len\_vec = int.Parse(Console.ReadLine());

if (len\_vec > 0)

{

Console.WriteLine("Ваш Вектор: ");

massive[i] = new ArrayVector(len\_vec);

for (int j = 0; j < len\_vec; j++)

{

massive[i][j] = int.Parse(Console.ReadLine());

}

Console.WriteLine();

break;

}

else

{

throw new Exception("Количество коорднат должно быть больше нуля!");

}

}

}

catch (Exception)

{

Console.WriteLine("Ошибка! Повторите попытку!");

}

break;

case "2":

try

{

while (true)

{

Console.WriteLine("Вектор номер " + (i + 1));

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

int len\_vec = int.Parse(Console.ReadLine());

if (len\_vec > 0)

{

Console.WriteLine("Ваш Вектор: ");

massive[i] = new LinkedListVector(len\_vec);

for (int j = 0; j < len\_vec; j++)

{

massive[i][j] = int.Parse(Console.ReadLine());

}

Console.WriteLine();

break;

}

else

{

throw new Exception("Количество коорднат должно быть больше нуля!");

}

}

}

catch (Exception)

{

Console.WriteLine("Ошибка! Повторите попытку!");

}

break;

default:

Console.WriteLine("Ошибка ввода! Повторите попытку!");

break;

}

break;

}

}

return massive;

}

else

{

Console.WriteLine("Ошибка! Повторите попытку!");

}

}

}

}

}

using System;

using System.IO;

using System.Collections.Generic;

namespace ЛР\_06

{

[Serializable]

class ArrayVector : IVectorable, ICloneable, IComparable

{

int[] arr;

public ArrayVector(int len) // конструктор с параметрами

{

arr = new int[len];

}

public ArrayVector() // конструктор без парам (по умолчанию 5)

{

arr = new int[5];

}

public void SetElement(int index, int digit) // метод установки элемента массива

{

arr[index] = digit;

}

public int Length

{

get

{

return arr.Length;

}

}

public int GetElement(int index)

{

return arr[index];

}

public int this[int index] // индексатор для организации доступа к элементам массива, выбрасывающий искл. при некорректном индексе

{

get

{

try

{

return GetElement(index);

}

catch (IndexOutOfRangeException)

{

Console.WriteLine("Ошибка! Выход за границу массива!");

return 0;

}

}

set

{

try

{

SetElement(index, value);

}

catch (IndexOutOfRangeException) // объект исключения

{

Console.WriteLine("Ошибка! Выход за границу массива!");

}

}

}

public double GetNorm() // модуль вектора

{

double modul = 0;

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

{

modul += Math.Pow(arr[i], 2);

}

modul = Math.Sqrt(modul);

return modul;

}

public override string ToString()

{

string str = Length + " ";

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

{

if (arr[i] != 0)

{

str += arr[i] + " ";

}

}

return str;

}

public override bool Equals(object obj) // переопреление метода Equals класса Object

{

int count = 0;

if (obj is IVectorable a)

{

if (vectorable.Length == arr.Length) // если равны по длине

{

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

{

if (vectorable[i] == arr[i]) // если равны покоординатно

{

count += 1;

}

}

if (arr.Length == count) // если равны по кол-ву координат

{

return true;

}

else

{

return false;

}

}

else

{

return false;

}

}

else

{

return false;

}

}

public override int GetHashCode() // получение хэш-кода

{

return base.GetHashCode();

}

public object Clone() // клонирование объектов

{

ArrayVector array = new ArrayVector(Length);

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

{

array[i] = this[i];

}

return array;

}

public int CompareTo(object obj) // метод сравнения по числу координат

{

if (obj is IVectorable vectorable)

{

if (Length == arr.Length)

{

return 0;

}

else if (Length > arr.Length)

{

return 1;

}

else

{

return -1;

}

}

else

{

throw new Exception("Невозможно сравнить!");

}

}

}

}

using System;

using System.IO;

using System.Collections.Generic;

namespace ЛР\_06

{

class Vectors

{

public static ArrayVector Sum(IVectorable vec1, IVectorable vec2)

{

if (vec1.Length == vec2.Length)

{

ArrayVector result = new ArrayVector();

int sum;

Console.Write("Сумма векторов: ");

for (int i = 0; i < vec1.Length; i++)

{

sum = vec1[i] + vec2[i];

result[i] = sum;

Console.Write(result[i] + " ");

}

return result;

}

else

{

throw new FormatException("Количество координат должно быть одинаково!");

}

}

public static double Scalar(IVectorable vec1, IVectorable vec2)

{

double result = 0;

if (vec1.Length == vec2.Length)

{

Console.Write("Скалярное произведние: ");

for (int i = 0; i < vec1.Length; i++)

{

result += vec1[i] \* vec2[i];

}

}

else

{

throw new FormatException("Количество координат должно быть одинаково!");

}

return result;

}

public static double GetNorm(IVectorable vec)

{

double len = 0;

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

{

len += Math.Pow(vec[i], 2);

}

len = Math.Sqrt(len);

return len;

}

public void OutputVector(IVectorable[] vec\_1, Stream vec\_2)

{

for (int i = 0; i < vec\_1.Length; i++)

{

byte[] array = BitConverter.GetBytes(vec\_1[i].Length);

vec\_2.Write(array, 0, array.Length);

for (int j = 0; j < vec\_1[i].Length; j++)

{

byte[] array\_2 = BitConverter.GetBytes(vec\_1[i][j]);

vec\_2.Write(array\_2, 0, array\_2.Length);

}

}

}

public IVectorable[] InputVector(Stream vec\_2)

{

if (vec\_2.Length == 0)

{

throw new Exception("Массив не содержит векторы!");

}

else

{

vec\_2.Seek(0, SeekOrigin.Begin);

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

while (vec\_2.Position != vec\_2.Length)

{

byte[] length = new byte[4];

vec\_2.Read(length, 0, 4);

IVectorable vector = new ArrayVector(BitConverter.ToInt32(length, 0));

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

{

length = new byte[4];

vec\_2.Read(length, 0, 4);

vector[i] = BitConverter.ToInt32(length, 0);

}

list.Add(vector);

}

IVectorable[] mass\_vec = list.ToArray();

return mass\_vec;

}

}

public void WriteVector(IVectorable[] vec\_1, TextWriter vec\_2)

{

for (int i = 0; i < vec\_1.Length; i++)

{

for (int j = 0; j < vec\_1[i].Length; j++)

{

vec\_2.Write(vec\_1[i][j] + " ");

}

vec\_2.WriteLine();

}

}

public IVectorable[] ReadVector(TextReader vector)

{

string stroka;

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

while ((stroka = vector.ReadLine()) != null)

{

string[] result = stroka.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

IVectorable vector\_2 = new ArrayVector(result.Length);

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

{

vector\_2[i] = Convert.ToInt32(result[i]);

}

list.Add(vector\_2);

}

var array = list.ToArray();

return array;

}

}

}

using System;

using System.Collections.Generic;

using System.IO;

namespace ЛР\_06

{

[Serializable]

class LinkedListVector : IVectorable, IComparable, ICloneable

{

public Node Head;

public class Node

{

public Node link; // null

public int value; // целое число

public Node() // констуктор по умолчанию

{

value = 0;

link = null;

}

public Node Link // свойство

{

get { return link; }

set { link = value; }

}

}

public int Length // свойство для чтения элентов

{

get

{

int i = 0;

Node node = Head;

while (node != null)

{

node = node.link;

i++;

}

return i;

}

}

public LinkedListVector(int len) // конструктор с парамаметром

{

try

{

if (len != 0)

{

Head = new Node();

Node node = Head;

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

{

node.Link = new Node();

node = node.Link;

}

}

}

catch (Exception)

{

Console.WriteLine("Длина не может равняться 0!");

}

}

public LinkedListVector() : this(5) { }

public Node GetElement(int num)

{

Node node = Head;

if (num > -1 && num < Length)

{

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

{

node = node.Link; //добавление ссылки

}

}

else

{

throw new IndexOutOfRangeException(); // принудительный выброс объекта исключения

}

return node;

}

public double GetNorm() // длина вектора (модуль)

{

double result = 0;

Node node = Head;

while (node != null)

{

result += Math.Pow(node.value, 2);

node = node.Link;

}

result = Math.Sqrt(result);

return result;

}

public int this[int num] // индексатор

{

get { return GetElement(num).value; }

set { GetElement(num).value = value; }

}

public override string ToString() // переопределение метода ToString класса Object

{

string str = Length + " ";

Node node = Head;

while (node != null)

{

str += node.value + " ";

node = node.Link;

}

return str;

}

public override bool Equals(object obj) // переопреление метода Equals класса Object

{

int count = 0;

if (obj is IVectorable vectorable)

{

if (vectorable.Length == Length) // если равны по длине

{

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

{

if (vectorable[i] == GetElement(i).value) // если равны покоординатно

{

count += 1;

}

}

if (Length == count) // если равны по кол-ву координат

{

return true;

}

else

{

return false;

}

}

else

{

return false;

}

}

else

{

return false;

}

}

public override int GetHashCode()

{

return base.GetHashCode();

}

public object Clone() // клонирование объектов

{

LinkedListVector list = new LinkedListVector(Length);

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

{

list[i] = this[i];

}

return list;

}

public int CompareTo(object obj)

{

if (obj is IVectorable vectorable)

{

if (Length == vectorable.Length)

{

return 0;

}

else if (Length > vectorable.Length)

{

return 1;

}

else

{

return -1;

}

}

else

{

throw new Exception("Невозможно сравнить!");

}

}

// ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ

public void AddElenentInBegin(int value)

{

Node newnode = new Node(); // создание нового объекта

newnode.value = value;

newnode.link = Head;

Head = newnode; // переписваивание главного элемента новым элементом

Node node = Head;

while (node != null)

{

Console.WriteLine(node.value);

node = node.link;

}

}

public void AddElementInEnd(int value)

{

Node newnode = new Node(); // создание нового объекта

newnode.value = value;

Node node = Head;

for (int i = 0; i < Length - 1; i++)

{

node = node.link; // добавление ссылки элементу

}

node.link = newnode; // присваивание ссылки новому элементу

node = Head;

while (node != null)

{

Console.WriteLine(node.value);

node = node.link;

}

}

public void DeleteElementInBegin()

{

Node node = Head;

node = node.link;

Head = node;

while (node != null)

{

Console.WriteLine(node.value);

node = node.link;

}

}

public void DeleteElementInEnd()

{

Node node = Head;

for (int i = 0; i < Length - 2; i++)

{

node = node.link;

}

node.link = null;

node = Head;

while (node != null)

{

Console.WriteLine(node.value);

node = node.link;

}

}

}

}

using System;

using System.Collections.Generic;

using System.Diagnostics.CodeAnalysis;

namespace ЛР\_06

{

class Sravnenie : IComparer<IVectorable>

{

public int Compare([AllowNull] IVectorable vec1, [AllowNull] IVectorable vec2)

{

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

{

return 0;

}

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

{

return 1;

}

else

{

return -1;

}

}

}

}

using System;

namespace ЛР\_06

{

interface IVectorable : IComparable, ICloneable

{

int this[int num] { get; set; } // индексатор

int Length { get; } // свойство для чтения

double GetNorm(); // получение модуля вектора

}

}

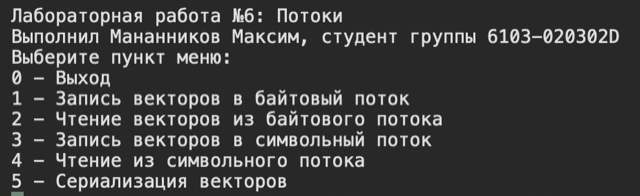


Рисунок 1 - Начало работы программы

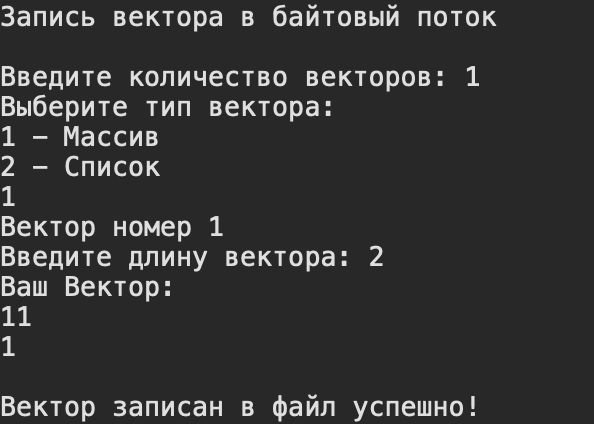


Рисунок 2 - Пример выполнения задания 1

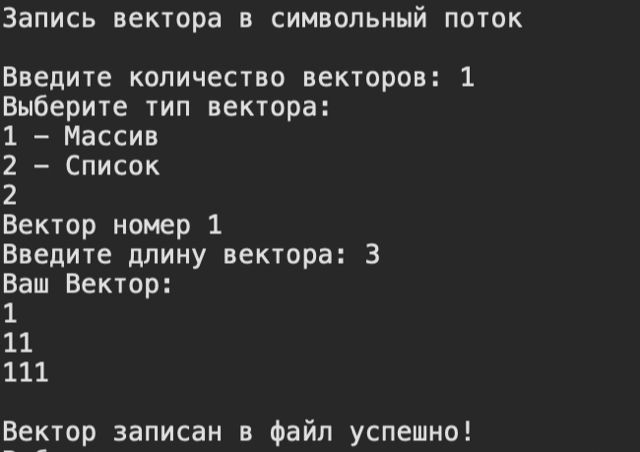


Рисунок 3 - Пример выполнения задания 2

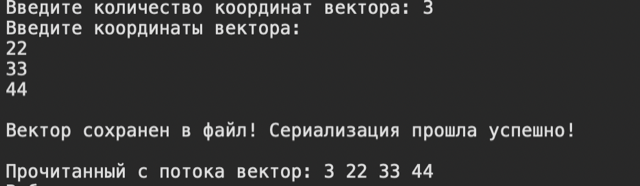


Рисунок 4 - Пример выполнения задания 3

**ВЫВОДЫ**

В лабораторной работе были использованы конструкции языка:

* условия;
* циклы;
* классы;
* конструкторы классов;
* различные методы классов;
* оператор switch;
* оператор 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.
2. Шилдт, Герберт. С# 4.0 : полное руководство / Герберт Шилдт ; [пер. с англ. и ред. И. В. Берштейна]. – Москва [и др.] : Вильямс, 2015. - 1056 с.