**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

Курсовая РАБОТА

**по дисциплине «Объектно-ориентированное программирование»**

Тема: Разработка приложения на основе объектно-ориентированного подхода

|  |  |  |
| --- | --- | --- |
| Студент гр. 7301 |  | Литвинов К.Л. |
| Преподаватель |  | Новакова Н.Е. |

Санкт-Петербург

2019

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Литвинов К.Л. | | |
| Группа 7301 | | |
| Тема работы: разработка приложения на основе объектно-ориентированного подхода | | |
| Исходные данные:  Задание в соответствии с вариантом (части 1- 5)  Язык программирования С# (.NET 4.6.2) | | |
| Содержание пояснительной записки:  «Содержание», «Введение», «Первый раздел», «Второй раздел», «Третий раздел», «Четвёртый раздел», «Пятый раздел», «Заключение», «Список использованных источников» | | |
| Предполагаемый объем пояснительной записки:  Не менее 59 страниц. | | |
| Дата выдачи задания: 02.10.2019 | | |
| Дата сдачи реферата: | | |
| Дата защиты реферата: | | |
| Студент |  | Литвинов К.Л. |
| Преподаватель |  | Новакова Н.Е. |

**Аннотация**

В данной курсовой работе созданы 5 программ, которые показывают применение объектно-ориентированного подхода в различных сферах разработки. Все приложения написаны на языке C#, тем самым в заданиях были использованы преимущества данного языка.

**Summary**

In this course work five applications were developed. They show use of object-orient programing in different subjects. All applications were developed using C# language and the advantages of this language were used in development.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 6](#_Toc28029523)

[1. ПЕРВЫЙ РАЗДЕЛ КУРСОВОЙ РАБОТЫ 8](#_Toc28029524)

[1.1. Вариант задания: 8](#_Toc28029525)

[1.2. Цель первого задания курсовой работы: 8](#_Toc28029526)

[1.3. Задание: 8](#_Toc28029527)

[1.4. Формализация задачи 8](#_Toc28029528)

[1.5. Спецификация программы 9](#_Toc28029529)

[1.6. Руководства оператора 10](#_Toc28029530)

[1.7. Руководство программиста 15](#_Toc28029531)

[1.8. Контрольный пример 16](#_Toc28029532)

[1.9. Листинг программы 17](#_Toc28029533)

[2. ВТОРОЙ РАЗДЕЛ КУРСОВОЙ РАБОТЫ 22](#_Toc28029534)

[2.1. Вариант задания: 22](#_Toc28029535)

[2.2. Цель второго задания курсовой работы: 22](#_Toc28029536)

[2.3. Задание: 22](#_Toc28029537)

[2.6. Спецификация программы 23](#_Toc28029538)

[2.7. Руководства оператора 24](#_Toc28029539)

[2.8. Руководство программиста 28](#_Toc28029540)

[2.9. Контрольный пример 29](#_Toc28029541)

[2.10. Листинг программы 30](#_Toc28029542)

[3. ТРЕТИЙ РАЗДЕЛ КУРСОВОЙ РАБОТЫ 33](#_Toc28029543)

[3.1. Вариант задания: 33](#_Toc28029544)

[3.2. Цель третьего задания курсовой работы 33](#_Toc28029545)

[3.3. Задание: 33](#_Toc28029546)

[3.4. Теоретический аспект задачи: 33](#_Toc28029547)

[3.5. Формализация задачи: 34](#_Toc28029548)

[3.6. Спецификация программы 35](#_Toc28029549)

[3.7. Руководства оператора 36](#_Toc28029550)

[3.8. Руководство программиста 39](#_Toc28029552)

[3.9. Контрольный пример 40](#_Toc28029553)

[3.10. Листинг программы 40](#_Toc28029554)

[4. ЧЕТВЁРТЫЙ РАЗДЕЛ КУРСОВОЙ РАБОТЫ 43](#_Toc28029555)

[4.1. Вариант задания: 43](#_Toc28029556)

[4.2. Цель четвёртого задания курсовой работы: 43](#_Toc28029557)

[4.3. Задание: 43](#_Toc28029558)

[4.4. Формализация 43](#_Toc28029559)

[4.5. Спецификация программы 47](#_Toc28029560)

[4.6. Руководства оператора 50](#_Toc28029561)

[4.7. Руководство программиста 54](#_Toc28029562)

[4.8. Руководство программиста 55](#_Toc28029563)

[4.9. Листинг программы 56](#_Toc28029564)

[5. ПЯТЫЙ РАЗДЕЛ КУРСОВОЙ РАБОТЫ 62](#_Toc28029565)

[5.1. Цель пятого задания курсовой работы 62](#_Toc28029566)

[5.2. Задание: 62](#_Toc28029567)

[5.3. Теоретический аспект задачи: 62](#_Toc28029568)

[5.4. Формализация 63](#_Toc28029569)

[5.5. Спецификация программы 64](#_Toc28029570)

[5.6. Руководства оператора 66](#_Toc28029571)

[5.7. Руководство программиста 70](#_Toc28029572)

[5.8. Контрольный пример 71](#_Toc28029573)

[5.9. Вывод 73](#_Toc28029574)

[5.10. Листинг 73](#_Toc28029575)

# ВВЕДЕНИЕ

В рамках курсовой работы мы рассмотрим несколько сфер применения объектно-ориентированного программирования, в том числе в реализации парсера, методов оптимизации, задачах имитационного моделирования, задачах на графах и задачах теории принятия решения.

**Цель курсовой работы**: Целью курсовой работы является закрепление теоретических знаний и получение практических навыков разработки программного обеспечения на основе объектно-ориентированного подхода. Выполнение курсовой работы должно базироваться на объектной модели, являющейся концептуальной базой для объектно-ориентированной парадигмы. В курсовой работе должны быть отражены ключевые концепций объектной модели: абстрагирование, передача сообщений, инкапсуляция, модульность, полиморфизм и наследование.

# ПЕРВЫЙ РАЗДЕЛ КУРСОВОЙ РАБОТЫ

1.1. Вариант задания:28

1.2. Цель первого задания курсовой работы: закрепление теоретических знаний и практических навыков разработки программ на основе объектно-ориентированного подхода и разработке объектной модели.

1.3. Задание:создать класс человек, имеющий имя (указатель на строку), возраст, вес. Определить конструкторы, деструктор и запись в файл. Создать public- производный класс - совершеннолетний, имеющий номер паспорта. Определить конструкторы по умолчанию и конструкторы с разным числом параметров, деструкторы, функцию вывода данных в файл. Определить функции переназначения возраста и номера паспорта.

## 1.4. Формализация задачи

Программа должна позволять пользователю ввести совершеннолетнего, имеющего паспорта или просто человека. Имеющий паспорт будет наследником человека, а совершеннолетний будет наследником имеющего паспорт.

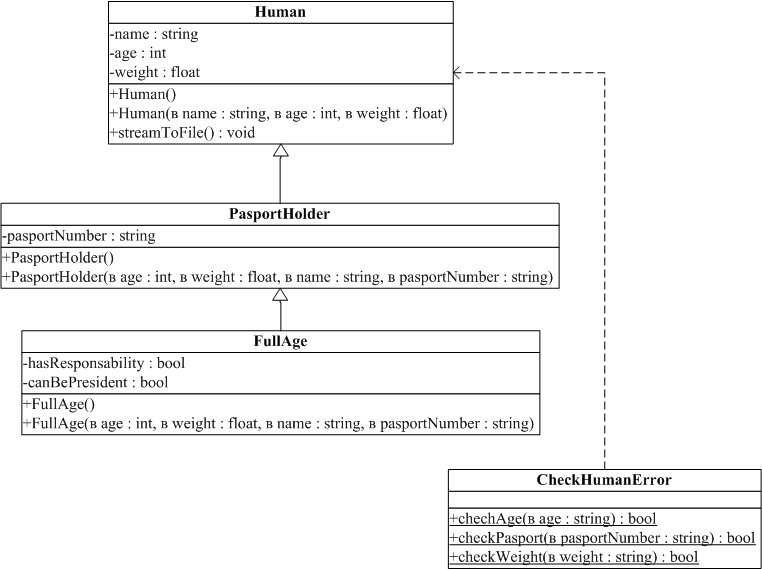


Рисунок 1.

## 1.5. Спецификация программы

Используемые классы:

* Human, для представления человека
* PasportHoler для представления держателя паспорта
* FullAge для представления совершеннолетнего

Таблица 1.1. Описание атрибутов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Класс** | **Название атрибута** | **Тип атрибута** | **Модификатор доступа** | **Описание** |
| Human | name | string | private | Имя человека |
| age | int | private | Возраст человека |
| weight | float | private | Вес человека |
| PasportHolder | pasportNumber | string | private | Номер паспорта держателя паспорта |
| FullAge | hasResponsability | bool | private | Наличие у совершеннолетнего ответственности |
| canBePresident | bool | private | Наличие возможности у совершеннолетнего баллотироваться в президенты |

Таблица 1.2. Описание методов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Класс** | **Название метода** | **Модификатор доступа** | **Входные параметры** | **Выходные параметры** | **Описание** |
| Human | streamToFile | public | - | - | Вывод класса человек в txt файл |

## 1.7. Руководство программиста

**Назначение и условие применения программы**

Программа позволяет пользователю вводить различные данные о классах, представленных в программе. Для запуска программы требуется операционная система Windows и платформа .NET Framework не ниже версии 4.6.2. На устройстве программиста/оператора должна быть поддержка внешних устройств ввода «клавиатура», «мышь», устройства вывода «монитор». Пользователь должен обладать навыками работы с графическим интерфейсом операционной системы.

**Характеристика программы**

Программа следит за правильностью заполнения данных и выводит ошибку, если какое-то поле было введено неправильно.

**Обращение к программе**

Программу можно запустить двумя способами:

1. Через .exe файл, расположенный в папке проекта по пути bin\Debug
2. Через программу Visual Studio, при запуске отладки программы.

**Входные и выходные данные**

Программа принимает на вход поля, заполненные в программе, на выход программа выдаёт сообщение об успешном/неуспешном выполнение той или иной операции. Также программа может по запросу пользователя выдать .txt файл.

**Сообщения**

Программа выводит на экран пользователя, сообщение об ошибке, если тот неправильно ввёл какие-либо данные. Так же программа выводит сообщения об успешном добавление, если все данные были введены корректно и сообщение об успешном выводе данных в txt файл, если пользователь решил нажать на соответствующую кнопку.

При успешном вводе класса программа выдаст сообщение об успешном вводе (рисунок 1.7.).

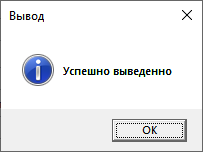


Рисунок 1.7.

Если номер введённый номер паспорта будет длинною больше или меньше 10, , то при попытке нажать кнопку «Добавить» программа выдаст ошибку (рисунок 1.8.)

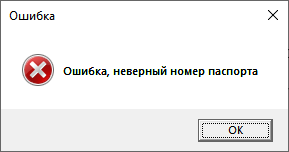


Рисунок 1.8.

Если введённый возраст для классов «Держатель паспорта» и «Совершеннолетний» будет меньше 18, а также если попытаются ввести символы вместо числа, то при попытке нажать кнопку «Добавить» программа выдаст ошибку (рисунок 1.9.).

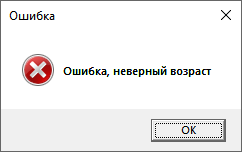


Рисунок 1.9.

Если введённый вес будет содержать какие-либо символы, помимо чисел, , то при попытке нажать кнопку «Добавить» программа выдаст ошибку (рисунок 1.10.)

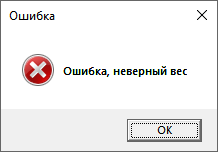


Рисунок 1.10.

## 1.8. Контрольный пример

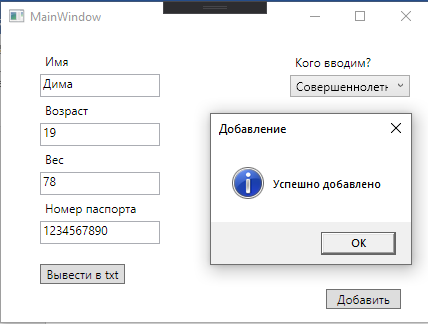


Рисунок 1.

На данном изображение проиллюстрирован успешный ввод объекта «Совершеннолетний»

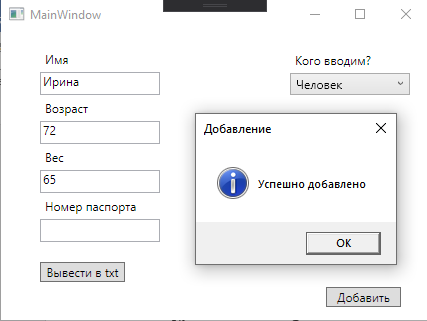


Рисунок 1.

На данном изображение проиллюстрирован успешный ввод объекта «Человек»

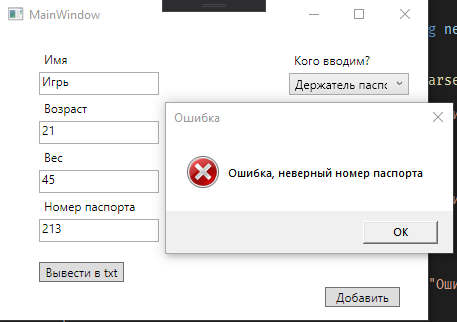


Рисунок 1.

На данном изображение проиллюстрирован неправильный ввод объекта «Держатель паспорта» (ошибка в неверном номере паспорта)

## 1.9. Листинг программы

Классы Human, PasportHolder, FullAge

public class Human

{

private string name; //Name of the human

private int age { get; set; } //Age of the human

private float weight; //Weight of the human

//Constructor by default

public Human() {

name = "";

age = 0;

weight = 0;

}

//Constructor with all values

public Human(string name="", int age=0, float weight=0)

{

this.name = name;

this.age = age;

this.weight = weight;

}

//Override output

public override string ToString()

{

return "Name: " + name + "\nAge: " +

age + "\nWeight: " + weight;

}

//Output all data in txt file

public void streamToFile()

{

string docPath =

Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

using (StreamWriter outputFile =

File.AppendText("Humans.txt"))

{

outputFile.Write(ToString());

}

}

}

public class PasportHolder: Human

{

private string pasportNumber { get; set; } //Number of pasport

public PasportHolder()

{

pasportNumber = "";

}

public PasportHolder(string name = "", int age = 0, float weight = 0,

string pasportNumber = "")

: base(name, age, weight)

{

this.pasportNumber = pasportNumber;

}

public override string ToString()

{

return base.ToString() + "\nPassport Number: " + pasportNumber;

}

}

public class FullAge: PasportHolder

{

private bool hasResponsability;

private bool canBePresident;

//Constructor by default

public FullAge()

{

hasResponsability = true;

canBePresident = false;

}

//Constructor with all values

public FullAge(string name="", int age=0, float weight=0,

string passportNumber= "")

: base(name, age, weight, passportNumber)

{

this.hasResponsability = true;

if (age >= 35)

{

this.canBePresident = true;

}

else

{

this.canBePresident = false;

}

}

//Override output

public override string ToString()

{

return base.ToString() + "\nHas Responsability: " + hasResponsability.ToString() +

"\nCan be president: "+ canBePresident.ToString();

}

}

# 2. ВТОРОЙ РАЗДЕЛ КУРСОВОЙ РАБОТЫ

2.1. Вариант задания:9-5-2

2.2. Цель второго задания курсовой работы: решение задачи из области теории принятия решений и реализация её в программном виде.

2.3. Задание:найти максимальную цену эксперимента, определив минимальное значение риска по формуле

 ()





Сделать ввод данных через графический интерфейс (Windows Forms). Вывод результатов выполнить в форму. Организовать сохранение результатов в файле.

**2.4. Теоретический аспект:**

Для того, чтобы найти максимальную цену эксперимента нам надо построить матрицу рисков. Она строится на основе матрицы выигрышей, только вместо значения uij мы ставим значение rij, которое находим по формуле . Далее мы находим по формуле (1) мы находим значение средних рисков и выбираем минимальное из полученных значений. Это и будет максимальная цена эксперимента

**2.5. Формализация:**

Так как для задачи требуется ввод через матрицу выигрышей, а вывод черех матрицу рисков, то было решено разбить здазачу на три подзадачи: ввод матрицы выигрешй, перевод матрицы выигрешей в матрицу рисков, поиск максимальнрй цены эксперимента для матрицы рисков. Тем самым для рещения задачи было решено разработать 3 класса: Playoff, Risk, Converter. Класс Playoff отвечает за хранение и создание матрицы выигрешей, класс Risk, отвечает за хранение матрицы рисков, а так же значения средних рисков. Класс Converter отвечает за перевод матрицы выигрешей в матрицу рисков.

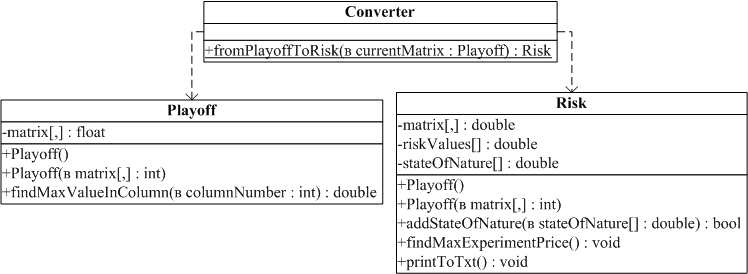


Рисунок 2.1

## 2.6. Спецификация программы

Используемые классы:

* Playoff, для представления матрицы выигрышей и методов, для взаимодействия с ней
* Risk для представления матрицы рисков, а так же хранения значений рисков и вероятностей. В дополнение реализует методы для нахождения максимальной цены эксперимента
* Converter для перевода матрицы выигрышей в матрицу рисков

Таблица 2.1. Описание атрибутов классов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Класс** | **Название атрибута** | **Тип атрибута** | **Модификатор доступа** | **Описание** |
| Playoff | matrix | float[,] | private | Матрица выигрышей |
| Risk | matrix | double[] | private | Матрица рисков |
| riskValues | double[] | private | Массив значений рисков |
| stateOfNature | double[] | private | Массив значений вероятностей |

Таблица 2.2. Описание методов классов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Класс** | **Название метода** | **Модификатор доступа** | **Входные параметры** | **Выходные параметры** | **Описание** |
| Playoff | findMaxValueInColumn | public | int columnNumber | double | Поиск наибольшего значения в стобце |
| Converter | fromPlayoffToRisk | public static | Playoff currentMatrix | Risk | Перевод матрицы выигрышей в матрицу рисков |
| Risk | addStateOfNature | public | double[] stateOfNature | bool | Добавление массива значений вероятности для класса |
| findMaxExperimentPrice | public | - | - | Нахождение максимальной цены эксперимента |
| printToTxt | public | - | - | Вывод результатов в. txt файл |

## 2.8. Руководство программиста

**Назначение и условие применения программы**

Программа позволяет пользователю позволяет находить максимальную цену эксперимента из матрицы выигрышей и массива значений вероятностей. Для запуска программы требуется операционная система Windows и платформа .NET Framework не ниже версии 4.6.2. На устройстве программиста/оператора должна быть поддержка внешних устройств ввода «клавиатура», «мышь», устройства вывода «монитор». Пользователь должен обладать навыками работы с графическим интерфейсом операционной системы.

**Характеристика программы**

Программа следит за правильностью заполнения данных и выводит ошибку, если какое-то поле было введено неправильно.

**Обращение к программе**

Программу можно запустить двумя способами:

1. Через .exe файл, расположенный в папке проекта по пути bin\Debug
2. Через программу Visual Studio, при запуске отладки программы.

**Входные и выходные данные**

Программа принимает на вход размерность матрицы выигрышей, значения полей матрицы и значения вероятностей. На выход программа выводит матрицу рисков, массив значений рисков и значение цены максимального эксперимента.

**Сообщения**

Программа выводит на экран пользователя, сообщение об ошибке, если тот неправильно ввёл какие-либо данные.

Если пользователь ввёл символы, вместо цифр в поля, под «Количество строк», «Количество столбцов», то при попытке нажать кнопку «Создать матрицу» программа выдаст ошибку (рисунок 2.6.).

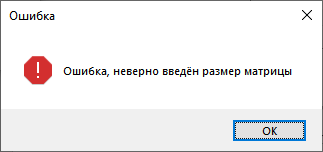


Рисунок 2.6.

Если пользователь ввёл символы, вместо числа с плавающей запятой в поля матрицы рисков или не до конца её заполнил, то при попытке нажать кнопку «Найти максимальную цену эксперимента» программа выдаст ошибку (рисунок 2.7.).

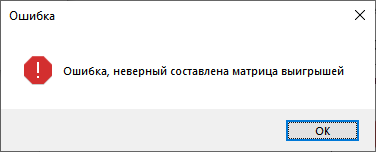


Рисунок 2.7.

Если пользователь ввёл символы, вместо числа с плавающей запятой в поля значений вероятностей или не до конца их заполнил, то при попытке нажать кнопку «Найти максимальную цену эксперимента» программа выдаст ошибку (рисунок 2.8.).

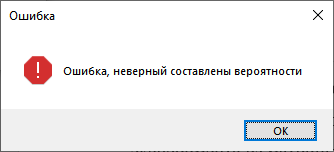


Рисунок 2.8.

## 2.9. Контрольный пример

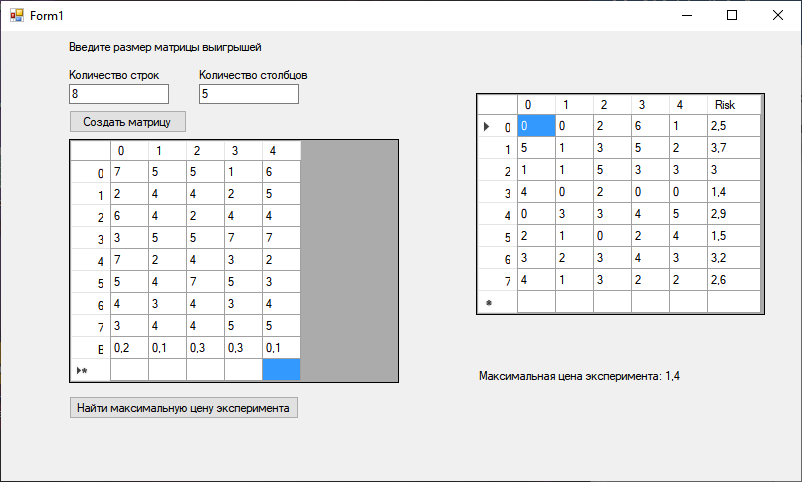


Рисунок 2.9.

На рисунке 2.9. продемонстрирована работа программа на матрицы выигрышей 8\*5. Справа мы видим заполненную матрицу со значениями вероятностей, слева мы видим матрицу рисков, со значениями рисков. Под матрицей рисков приведено найденное значение максимального эксперимента.

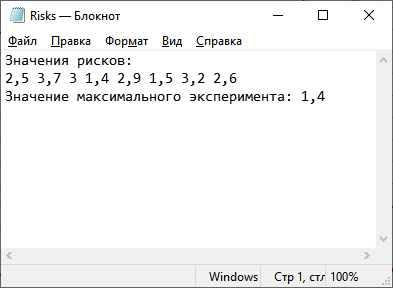


Рисунок 2.10.

На рисунке 2.10. мы видим результат выполнения программы, выведенный в txt файл.

## 2.10. Листинг программы

Класс Playoff

//Playoff matrix

public class Playoff

{

private double[,] matrix;

public double[,] Matrix => matrix;

Playoff()

{

}

public Playoff(double[,] matrix)

{

this.matrix = matrix;

}

//Find max value in row

public double findMaxValueInColumn(int columnNumber)

{

double max = 0;

for (int i = 0 ; i < matrix.GetLength(0); i++)

{

if (max < matrix[i, columnNumber])

max = matrix[i, columnNumber];

}

return max;

}

}

Класс Risk

public class Risk

{

double[,] matrix; //Risk matrix

public double[,] Matrix => matrix;

double[] riskValues; //Risk values

public double[] RiskValues => riskValues;

double[] stateOfNature; //State of nature

public double[] StateOfNature => stateOfNature;

double maxExperiment; //Max experiment price

public double MaxExperiment => maxExperiment;

public Risk(int n, int m)

{

matrix = new double[n, m];

riskValues = null;

stateOfNature = null;

maxExperiment = 0;

}

public Risk(double[,] matrix)

{

this.matrix = matrix;

this.riskValues = new double[matrix.GetLength(0)];

}

//Finding max experiment price

public void findMaxExperimentPrice()

{

int n = matrix.GetLength(0);

int m = matrix.GetLength(1);

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

{

riskValues[i] = 0;

//Multipling each cell element on current state of nature

//To find risk values of current row

for (int j = 0; j < m; j++)

{

riskValues[i] += matrix[i, j] \* stateOfNature[j];

}

}

//Finding maximum value of all risks

maxExperiment = riskValues.Min();

}

}

Класс Converter

//Converter from playoff to risk matrix

public class Converter

{

public static Risk fromPlayoffToRisk(Playoff playoff)

{

}

}

# 3. ТРЕТИЙ РАЗДЕЛ КУРСОВОЙ РАБОТЫ

3.1. Вариант задания:Г-42-2

3.2. Цель третьего задания курсовой работы**:** решение задачи на теории графах и реализацию её в программном виде.

3.3. Задание:найти максимальный поток в сети с помощью алгоритма Форда- Фалкерсона. Задачу решить в общем виде.

Ввод можно через матрицу (Ввод матрицы случайным образом и редактор матрицы с проверкой ошибок)

## 3.4. Теоретический аспект задачи:

В решение используется модифицированный метод Форда-Фалкерсона: Эдмондсона-Карпа [2].

Инициализация: обнуляем все потоки. Остаточная сеть будет изначально совпадает с исходной сетью.

В остаточной сети находим кратчайший путь из источника в сток. Если такого пути нет, останавливаемся. Пускаем через найденный путь максимально возможный поток: на найденном пути ищем ребро с минимальной пропускной способностью Сmin, для каждого ребра на найденном пути увеличиваем поток на Сmin, а в противоположном ему — уменьшаем на Сmin.

Модифицируем остаточную сеть. Для всех рёбер на найденном пути, а также для противоположных им рёбер, вычисляем новую пропускную способность. Если она стала ненулевой, добавляем ребро к остаточной сети, а если обнулилась, стираем его. Повторяем алгоритм до тех пор, пока находим кратчайший путь.

Для поиска кратчайшего пути в графе, воспользуемся поиском в ширину:

Создаём очередь вершин Queue. Кладём в очередь исток (вершину s). Отметим вершину s как посещённую, без родителя, а все остальные как не посещённые.

Пока очередь не пуста, выполняем следующие шаги:

* Удаляем из очереди текущую первую вершину.
* Для всех дуг, исходящих из удалённой вершины, которые ещё не посещены, выполняем следующие шаги:
  + Отмечаем эту вершину как посещённую, с родителем удалённая вершина.
  + Добавляем эту вершину в конец очереди.
  + Если данная вершина является стоком, то выходим из обоих циклов: мы нашли кратчайший путь.

Если очередь пуста, возвращаем ответ, что пути нет вообще и останавливаемся.

Если нет, идём от стока к истоку, каждый раз переходя к родителю. Возвращаем путь в обратном порядке.

## 3.5. Формализация задачи:

Для реализации алгоритма, было решено разбить задачу на 2 класса и 1 интерфейс. Интерфейс IFlow, который определяет функционал для нахождения максимального потока. Классы: Graph и Vertices. Graph хранит в себе матрицу остатков, а так же реализует функционал интерфейса, а именно реализацию алгоритма Эдмондсона-Карпа, и вспомогательную для него функцию поиска в ширину. Класс Vertices, определяет уникальные вершин, а также запоминает их имя и позицию. В нашем случае это используется для реализации стока и истока. Решение создать интерфейс было принято, так как для дальнейшей модификации мы сможем создать классы, которые будут реализовывать метод нахождения максимального потока интерфейса IFlow, другими способами, нежели только методом Форда-Фалкерсона.

Так как все атрибуты классов закрыты от изменений, но при этом сами классы доступны для дальнейшей модификации, то данное свойство отвечает требованию открытости-закрытости.

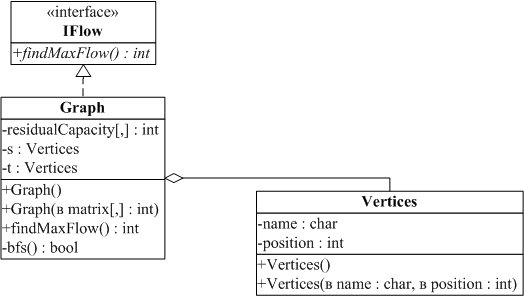


Рисунок 3.1

## 3.6. Спецификация программы

Используемые классы:

* Graph, для реализации метода Форда-Фалкерсона с использованием матрицы остатков
* Vertices, для описывания уникальных вершин графа

Таблица 3.1. Описание атрибутов классов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Класс** | **Название атрибута** | **Тип атрибута** | **Модификатор доступа** | **Описание** |
| Graph | residualCapacity | int[,] | private | Матрица остатков |
| s | Vertices | private | Вершина, описывающая исток |
| t | Vertices | private | Вершина, описывающая сток |
| Vertices | name | char | private | Название вершины |
| position | int | private | Позиция вершина в матрице остатков |

Таблица 3.2. Описание методов классов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Класс** | **Название метода** | **Модификатор доступа** | **Входные параметры** | **Выходные параметры** | **Описание** |
| Graph | findMaxFlow | public | - | int | Нахождение максимального потока в сети |
| bfs | private | - | bool | Обход в ширину для метода Форда-Фалкерсона |

## 3.7. Руководства оператора

**Назначение программы:**

Данная программа позволяет найти максимальный поток сети из известной матрицы смежности.

**Условие выполнение программы:**

Для запуска программы требуется операционная система Windows и платформа .NET Framework не ниже версии 4.6.2. На устройстве программиста/оператора должна быть поддержка внешних устройств ввода «клавиатура», «мышь», устройства вывода «монитор». Пользователь должен обладать навыками работы с графическим интерфейсом операционной системы.

**Выполнение программы**

Для запуска программы достаточно запустить .exe файл, который хранится в папке проекта по пути bin\Debug. При запуске появляется интерфейс программы.

На рисунке 3.2. продемонстрирован интерфейс программы.

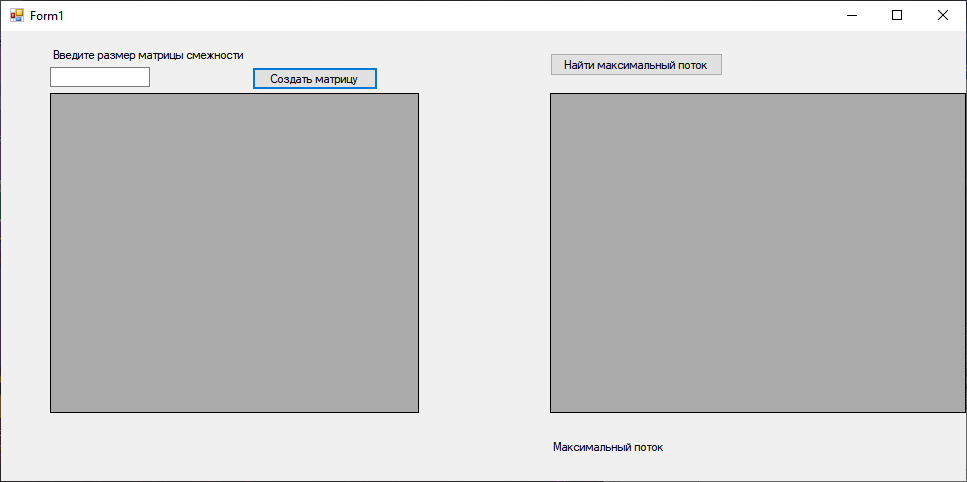


Рисунок 3.2.

В первую очередь надо ввести размер матрицы смежностей под надписью: «Введите размер матрицы смежности». В результате мы получим таблицу для ввода матрицы смежности (рисунок 3.3.)

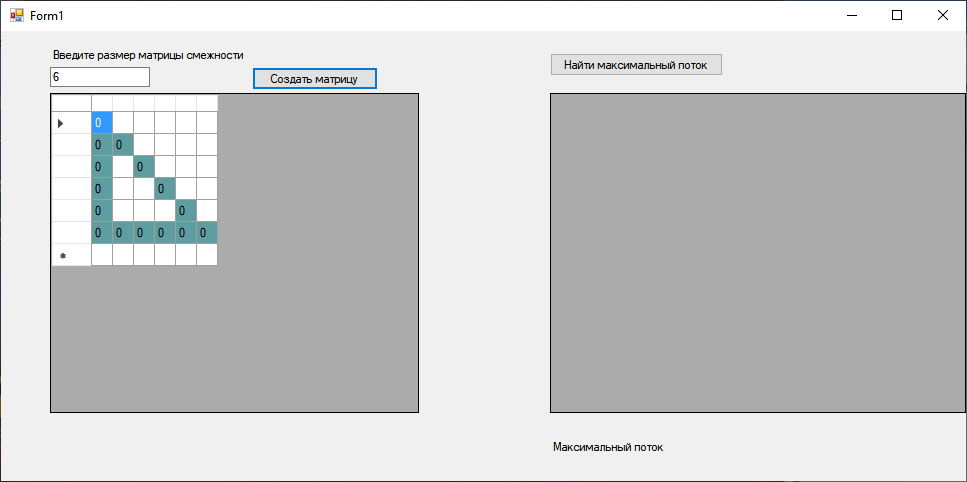


Рисунок 3.3.

В выведенную таблицу, в поля, которые отмечены белым цветом, надо ввести значения матрицы смежности, после чего нажать кнопку «Найти максимальный поток». После нажатия на кнопку мы получаем матрицу остатков. Под матрицей будет выведен максимальный поток введённой сети (рисунок 3.4.).

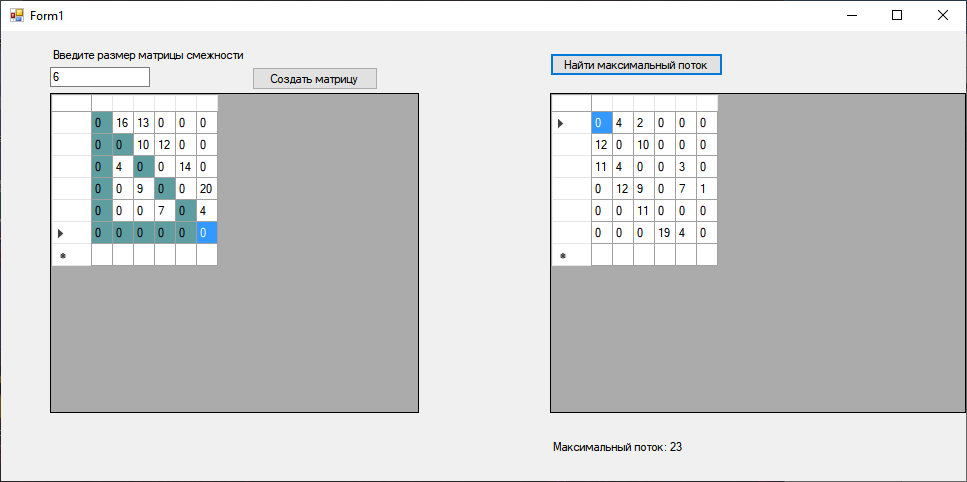


Рисунок 3.4.

**Сообщения оператору:**

Программа выводит на экран пользователя, сообщение об ошибке, если тот неправильно ввёл какие-либо данные.

Если пользователь введёт любой символ, кроме числа в поле под надписью: «Введите размер матрицы», то при попытке нажатия кнопки «» программа выдаст ошибку (рисунок 3.5.)

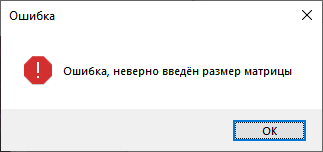


Рисунок 3.5.

## Если пользователь ввёл любой символ, кроме числа в поля матрицы смежности или же не до конца её заполнит, то при попытке нажатия кнопки «Найти максимальный поток» программа выдаст ошибку (рисунок 3.6.)

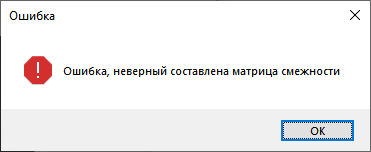


Рисунок 3.6.

## 3.8. Руководство программиста

**Назначение и условие применения программы**

Программа позволяет пользователю позволяет находить максимальную цену эксперимента из матрицы выигрышей и массива значений вероятностей. Для запуска программы требуется операционная система Windows и платформа .NET Framework не ниже версии 4.6.2. На устройстве программиста/оператора должна быть поддержка внешних устройств ввода «клавиатура», «мышь», устройства вывода «монитор». Пользователь должен обладать навыками работы с графическим интерфейсом операционной системы.

**Характеристика программы**

Программа следит за правильностью заполнения данных и выводит ошибку, если какое-то поле было введено неправильно.

**Обращение к программе**

Программу можно запустить двумя способами:

1. Через .exe файл, расположенный в папке проекта по пути bin\Debug
2. Через программу Visual Studio, при запуске отладки программы.

**Входные и выходные данные**

Программа принимает на вход размерность матрицы смежности. Далее программа принимает заполненную матрицу смежности. На выход выводятся матрица остатков и значение максимального потока.

**Сообщения**

Сообщения программисту совпадают с сообщениями оператора (см. пункт 3.7. сообщения оператору)

## 3.9. Контрольный пример

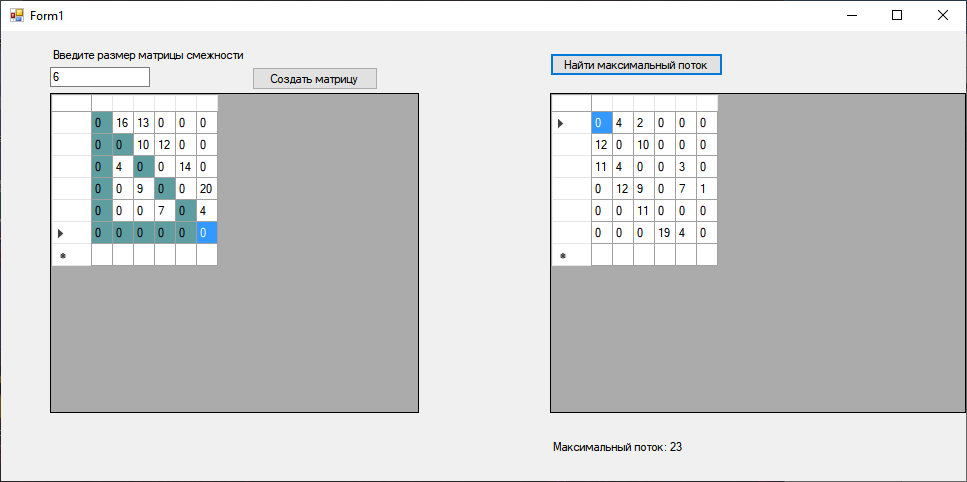


Рисунок 3.7.

На рисунке 3.7. продемонстрирована работа программы. Слева мы видим матрицу смежности, данную по заданию. Справа мы выели найденную матрицу остатков. Под матрицей остатков мы видим найденный максимальный поток. Результат совпадает с верным ответом.

## 3.10. Листинг программы

Класс Graph

public class Graph: IFlow

{

//Matrix of residual capacity

public int[,] residualCapacity { get; }

//Unique vertex

private Vertex s, t;

//Constructor from matrix parameter

public Graph(int[,] residualCapacity)

{

int n = residualCapacity.GetLength(0);

this.residualCapacity = new int[n, n];

//Copying values from given matrix

Array.Copy(residualCapacity, 0 ,this.residualCapacity, 0, residualCapacity.Length);

this.s = new Vertex("S", 0);

this.t = new Vertex("T",

Convert.ToInt32(Math.Sqrt(residualCapacity.Length)) - 1);

}

bool bfs(int []parent)

{

while(queue.Count != 0)

{

//Tacking first item from he queue

int u = queue[0]; //This will be first vertices

queue.RemoveAt(0);

for (int v = 0; v < N; v++)

{

//If we current vertices isn't visited, and free path from first vertices to current exist

if (visited[v] == false && residualCapacity[u, v] > 0)

{

queue.Add(v); //Adding this vertices to the queue

parent[v] = u; //Rembering the path

visited[v] = true; //Marking it as visited

}

}

}

return visited[t.position] == true;

}

public int findMaxFlow()

{

int u, v;

int N = residualCapacity.GetLength(0);

int[] parent = new int[N];

int maxFlow = 0;

while(bfs(parent)) //While there is a path from source to sink

{

int path\_flow = int.MaxValue;

//Finding the maximum flow through current path

for (v = t.position; v != s.position; v = parent[v])

{

u = parent[v];

path\_flow = Math.Min(path\_flow, residualCapacity[u, v]);

}

//Changing the residual capacity through current path

for (v = t.position; v != s.position; v = parent[v])

{

u = parent[v];

residualCapacity[u, v] -= path\_flow; residualCapacity[v, u] += path\_flow;

}

maxFlow += path\_flow;

}

return maxFlow;

}

}

Класс Vertex

class Vertex {

public string name {get;} //Name of unique vertex

public int position {get;} //Position of unique vertex

//Constructor by default

public Vertex() {

name = "";

position = 0;

}

//Constructor from name and position parameters

public Vertex(string name, int position) {

this.name = name;

this.position = position;

}

}

# 4. ЧЕТВЁРТЫЙ РАЗДЕЛ КУРСОВОЙ РАБОТЫ

4.1. Вариант задания:10

4.2. Цель четвёртого задания курсовой работы: решить задачу с помощью использования имитационной модели и реализации её в программном виде.

4.3. Задание:Имитация работы речного порта. Речной порт имеет 4 места для причаливания (4 причала). В порт заходят различные речные суда — некоторые из них небольшие (60% от общего количества судов) и занимают только 1 причал, другие — средние (30% от общего количества судов) и занимают 2 причала, есть также большие суда (10% от общего количества судов), которые одновременно занимают три причала. Суда заходят в порт через каждые A±B часов. Небольшие суда обслуживаются в течение C±D часов, средние суда обслуживаются в течение E±F часов, большие суда обслуживаются G±H часов. При этом A<E<G. Если судну некуда причалить (не хватает нужного количества причалов), оно ждёт на рейде порта в очереди. Порт работает круглосуточно. Проанализировать процесс обслуживания 50 судов.

В программе должен быть предусмотрен вывод подсчитанной статистики.

## 4.4. Формализация

Задача представлена в данном виде. Мы создаём очередь для каждого типа кораблей (в очереди небольших 30 судов, в очереди средних 15 судов, в очереди больших 5 судов), потом мы задаём время начало работы как 0 часов Затем мы генерируем время прибытия нового корабля (равномерно распределённая величина в пределах от А до В). Если текущее время совпадает с временем прибытия нового корабля, то мы случайным образом выбираем какой из 3х типов кораблей приплывает. Если выбранный корабль занимает причалов меньше, чем свободных причалов, то он будет добавлен в список обслуживания(время обслуживание определяется случайным образом по равномерному распределению в пределах, характерных для вида корабля (C±D, E±F, G±H)), а так же ему будет присвоено время выхода (время выхода – время обслуживания + текущее время), иначе он пойдёт в очередь на обслуживание. Каждый раз, когда в корабль идёт на обслуживание, мы уменьшаем число причалов, доступных в данный момент. Далее мы проходимся по списку обслуживаемых кораблей, если он не пустой. Когда у взято корабля из очереди его время выхода будет равнять текущему времени, то мы уберём текущий корабль из списка и увеличим количество свободных портов. Пройдясь по списку обслуживания, мы переходим к очереди на обслуживание. Если очередь не пустая, то мы берём элемент из очереди и так же пытаемся добавить его добавить на обслуживание (по описанному алгоритму). При успешном добавление на обслуживание, мы удаляем текущий корабль из очереди. После всех проделанных действий мы генерируем время прибытия нового корабля. В самом конце увеличиваем время на + 1 час. Выполняем весь алгоритм, пока не пройдут все 50 кораблей. (Алгоритм проиллюстрирован на рисунке 4.1, 4.2)

Для данная задачи были определены несколько сущностей.

* Порт – хранит в себе общее количество причалов, количество доступных причалов в данный момент, а также очередь из кораблей, ожидающих освобождение места в порту.
* Корабль, который хранит в себе время, которое он находится в порту, время которое он затратит на обслуживание и количество причалов, которое он занимает. Также имеется абстрактный метод для определения времени, которое корабль потратить на обслуживание.
* Малый, Средний и Большой Корабль - наследники Корабля, они обладает всеми атрибутами головного класса, а также имеют поля минимального и максимального времени, которое корабль может провести на обслуживание. Эти поля являются статическими, так как для всех экземпляров класса они будут всегда одинаковые. Реализует метод определения времени в порту, которой выбирает случайное значение в промежутке от минимального до максимального времени обслуживания.

Все классы показаны на рисунке 4.3.

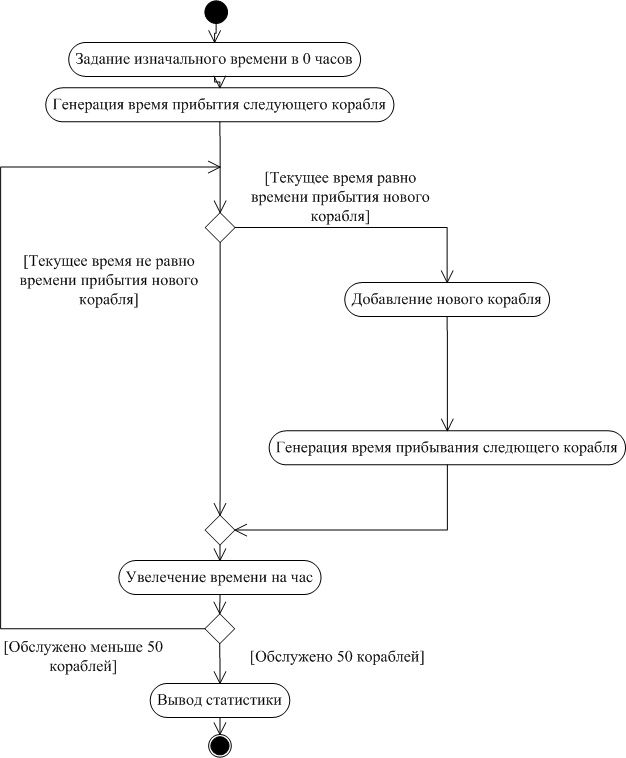


Рисунок 4.1

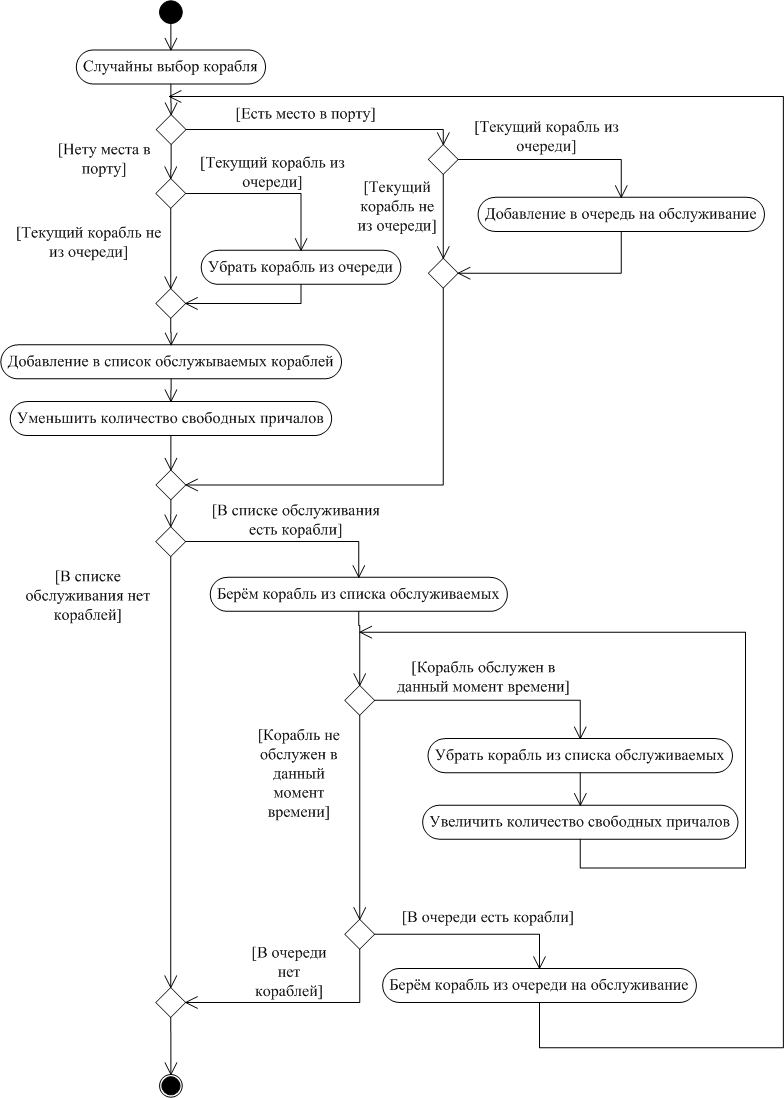


Рисунок 4.2

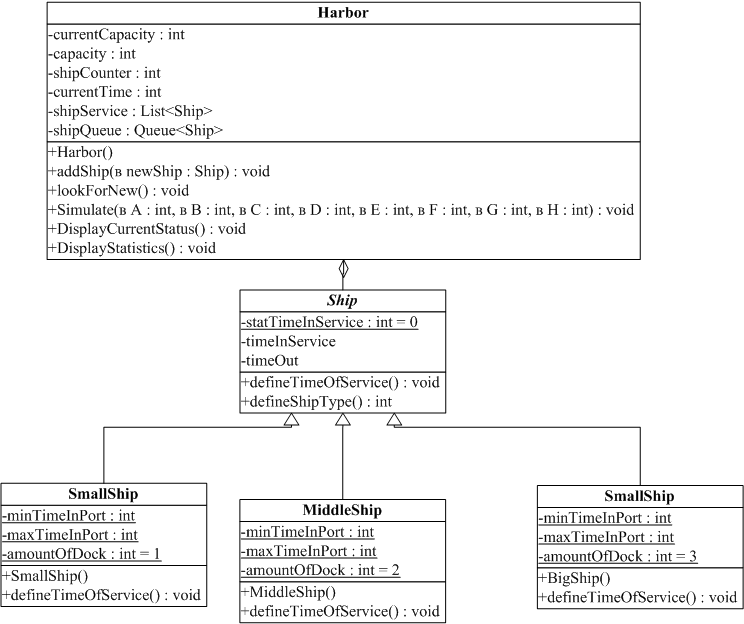


Рисунок 4.3

## 4.5. Спецификация программы

Используемые классы:

* Harbor, для представления порта и имитации его работы
* Ship, для представления общего типа корабля

Таблица 4.1. Описание атрибутов классов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Класс** | **Название атрибута** | **Тип атрибута** | **Модификатор доступа** | **Описание** |
| Harbor | currentCapacity | int | private | Текущая вместимость порта |
| capacity | int | private | Общая вместимость порта |
| shipCounter | int | private | Количество кораблей, прошедших обслуживание в порту. |
| Таблица 4.1. Описание атрибутов классов | | | | |
|  | currentTime | int | private | Текущее время в порту |
| shipService | List<Ship> | private | Корабли, которые находятся на обслуживание |
| shipQueue | Queue<Ship> | private | Корабли, которые стоят в очереди на обслуживание |
| Ship | statTimeInService | char | private | Название вершины |
| timeInService | int | private | Позиция вершина в матрице остатков |
| timeOut | int | private | Время выхода корабля с обслуживания |
| SmallShip | minTimeInPort | int | private static | Минимальное время, обслуживания корабля |
| maxTimeInPort | int | private static | Максимальное время, обслуживания корабля |
| amountOfDock | int | private static | Количество причалов, которое занимает корабль |
| MiddleShip | minTimeInPort | int | private static | Минимальное время, обслуживания корабля |
| maxTimeInPort | int | private static | Максимальное время, обслуживания корабля |
| amountOfDock | int | private static | Количество причалов, которое занимает корабль |
| BigShip | minTimeInPort | int | private static | Минимальное время, обслуживания корабля |
| maxTimeInPort | int | private static | Максимальное время, обслуживания корабля |
| amountOfDock | int | private static | Количество причалов, которое занимает корабль |

Таблица 4.2. Описание методов классов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Класс** | **Название метода** | **Модификатор доступа** | **Входные параметры** | **Выходные параметры** | **Описание** |
| Harbor | addShip | private | Ship newShip | - | Попытка добавления нового корабля на обслуживание |
| lookForNew | private | - | - | Просмотр текущих кораблей на обслуживание и попытка добавить корабль из очереди на обслуживание |
| Simulate | public | int A  int B  int C  int D  int E  int F  int G  int H | - | Запуск имитационной модели |
| DisplayCurrentStauts | public | - | - | Вывод текущей статистики по имитационной модели |
| DisplayStatistics | public | - | - | Вывод общей статистики имитационной модели |

## 4.7. Руководство программиста

**Назначение и условие применения программы**

Программа позволяет пользователю запустить имитационную модель речного порта, с параметрами, которые задаёт пользователь. Для запуска программы требуется операционная система Windows и платформа .NET Framework не ниже версии 4.6.2. На устройстве программиста/оператора должна быть поддержка внешних устройств ввода «клавиатура», «мышь», устройства вывода «монитор». Пользователь должен обладать навыками работы с графическим интерфейсом операционной системы.

**Характеристика программы**

Программа следит за правильностью заполнения данных и выводит ошибку, если какое-то поле было введено неправильно.

**Обращение к программе**

Программу можно запустить двумя способами:

1. Через .exe файл, расположенный в папке проекта по пути bin\Debug
2. Через программу Visual Studio, при запуске отладки программы.

**Входные и выходные данные**

Программа на вход принимает параметры имитационной модели, которые вводит пользователь. На выход программа выдаёт статистику по имитационной модели, а также txt файл, в котором содержится процесс работы имитационной модели.

**Сообщения**

Программа выводит на экран пользователя, сообщение об ошибке, если тот неправильно ввёл какие-либо данные.

Если пользователь ввёл символы в одно из полей программы, отличные от цифр, то программа выдаст ошибку (рисунок 4.8.)

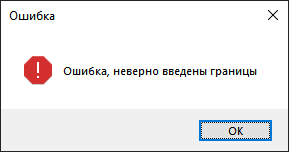


Рисунок 4.8.

## 4.8. Контрольный пример

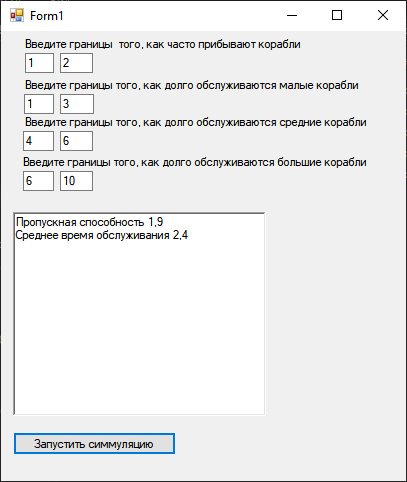


Рисунок 4.9.

На рисунке 4.9. продемонстрирован пример запуска программы. На данном изображении мы видим в самом внизу краткую статистику по работе причала. Для получение более подробной статистике обратимся к файлу simulationAllStats (рисунок 4.10.)

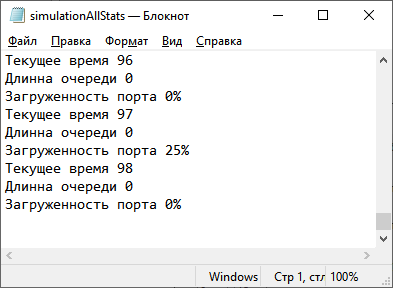


Рисунок 4.10.

На нём мы видим загруженность порта и длину очереди на каждый момент времени. Если мы хотим рассмотреть процесс работы порта в каждый момент времени, то можем посмотреть файл simulation.txt (рисунок 4.11.).

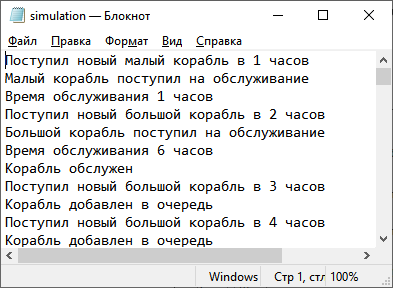


Рисунок 4.11.

## 4.9. Листинг программы

Класс Ship

abstract class Ship {

public static float statTimeInService = 0; //Time in service of all ships

public int timeInService; //Time in service of current ship

public int timeOut;//Time, when service of the ship will be done

abstract protected void defineTimeOfService(); //Defining time in service of current ship

}

Класс SmallShip

class SmallShip: Ship {

public static int minTimeInPort { get; set; } //Minimum time in port of current ship

public static int maxTimeInPort { get; set; } //Maximum time in port of current ship

private static int amountOfDock = 1; //Amount of dock, that ship needs for service

public SmallShip() {

defineTimeOfService();

}

protected override void defineTimeOfService() {

Random newRand = new Random();

this.timeInService = newRand.Next(minTimeInPort, maxTimeInPort);

}

}

Класс Harbor

class Harbor

{

//All events

delegate void HarborEvent(string message);

event HarborEvent NotifyAll;

event HarborEvent NotifyOrder;

event HarborEvent NotifyCurrentStat;

event HarborEvent NotifyStat;

private int currentCapacity; //Current free capacity of harbor

private int capacity; //Capacity of harbor

private int shipCounter; //Counter of how many ship passed service

private int currentTime; //Current time in harbor

private Queue<Ship> shipQueue; //Queue of ships, that waiting for service

private List<Ship> shipService; //List of ships that are on service

public Harbor()

{

currentCapacity = 4;

capacity = 4;

shipCounter = 0;

shipQueue = new Queue<Ship>();

shipService = new List<Ship>();

}

private void addShip(Ship newShip)

{

int n;

n = newShip.defineShipType();

//If current ship amount of docks is higher, than current harbor capacity

if (n > currentCapacity)

{

//Add to queue, if it isn't already ship from the queue

if ((shipQueue.Count == 0) || (newShip != shipQueue.Peek()))

{

NotifyOrder?.Invoke("Корабль добавлен в очередь");

shipQueue.Enqueue(newShip);

}

}

else

{

//If it is ship from the queue, than remove it from the queue

if ((shipQueue.Count > 0) && (newShip == shipQueue.Peek()))

{

shipQueue.Dequeue();

if (newShip is SmallShip)

{

NotifyOrder?.Invoke("Малый корабль вышел из очереди и поступил на обслуживание");

}

else if (newShip is MiddleShip)

{

NotifyOrder?.Invoke("Средний корабль вышел из очереди и поступил на обслуживание");

}

else

{

NotifyOrder?.Invoke("Большой корабль вышел из очереди и поступил на обслуживание");

}

}

//Reducing current harbor capacity

currentCapacity -= n;

if (newShip is SmallShip)

{

NotifyAll?.Invoke("Малый корабль поступил на обслуживание");

}

else if (newShip is MiddleShip)

{

NotifyAll?.Invoke("Средний корабль поступил на обслуживание");

}

else

{

NotifyAll?.Invoke("Большой корабль поступил на обслуживание");

}

NotifyAll?.Invoke($"Время обслуживания {newShip.timeInService} часов");

//Defining time out of current ship

newShip.timeOut = currentTime + newShip.timeInService;

//Adding current ship to service list

shipService.Add(newShip);

}

}

private void lookForNew()

{

int i = 0;

//Looking through ships on service

while (i < shipService.Count)

{

//If current ship finished service, than remove it from service

if (currentTime == shipService[i].timeOut)

{

currentCapacity += shipService[i].defineShipType();

Ship.statTimeInService += shipService[i].timeInService;

shipService.RemoveAt(i);

NotifyAll?.Invoke("Корабль обслужен");

shipCounter++;

}

else

{

i++;

}

}

//Checking if we can add on service ship from queue

if (shipQueue.Count > 0)

{

for (int j = 0; j < shipQueue.Count; j++)

addShip(shipQueue.Peek());

}

}

public void Simulate(int A, int B, int C, int D, int E, int F, int G, int H)

{

//Defing event hadelers

NotifyAll += DisplayAllMessage;

NotifyOrder += DisplayQueueMessage;

NotifyCurrentStat += DisplayCurrentStatMessage;

NotifyStat += DisplayStatMessage;

currentTime = 0;

Random random = new Random();

//Creating queue for all types of ships

Queue<SmallShip> smallShipList = new Queue<SmallShip>();

Queue<MiddleShip> middleShipList = new Queue<MiddleShip>();

Queue<BigShip> bigShipList = new Queue<BigShip>();

//Filling queue for all types of ships

for (int i = 0; i < 30; i++)

{

SmallShip newShip = new SmallShip();

smallShipList.Enqueue(newShip);

}

for (int i = 0; i < 15; i++)

{

MiddleShip newShip = new MiddleShip();

middleShipList.Enqueue(newShip);

}

for (int i = 0; i < 5; i++)

{

BigShip newShip = new BigShip();

bigShipList.Enqueue(newShip);

}

//Defing time intervals for all types of ships

SmallShip.minTimeInPort = C;

SmallShip.maxTimeInPort = D;

MiddleShip.minTimeInPort = E;

MiddleShip.maxTimeInPort = F;

BigShip.minTimeInPort = G;

BigShip.maxTimeInPort = H;

//Defing time intervals for ship arrival

int randMin = A;

int randMax = B;

//Finding time of arrival of first ship

int timeOfNextShip = random.Next(randMin, randMax);

do

{

//Checking if current time is time of arrival of new ship

if (currentTime == timeOfNextShip)

{

//Defining which type of ship has arrived

switch (random.Next(1, 4))

{

case 1:

if (smallShipList.Count > 0)

{

NotifyAll?.Invoke($"Поступил новый малый корабль в {currentTime} часов");

this.addShip(smallShipList.Dequeue());

}

break;

case 2:

if (middleShipList.Count > 0)

{

NotifyAll?.Invoke($"Поступил новый средний корабль в {currentTime} часов");

this.addShip(middleShipList.Dequeue());

}

break;

case 3:

if (bigShipList.Count > 0)

{

NotifyAll?.Invoke($"Поступил новый большой корабль в {currentTime} часов");

this.addShip(bigShipList.Dequeue());

}

break;

default:

break;

}

//Changing arrive intervals

randMin = currentTime + 1;

randMax = currentTime + 2;

//Finding time of arrival of next ship

timeOfNextShip = random.Next(randMin, randMax);

}

this.lookForNew();

//Increasing current time

currentTime++;

//Displaing current statistics

DisplayCurrentStatus();

//Thread.Sleep(10);

} while (this.shipCounter < 50);

}

# 5. ПЯТЫЙ РАЗДЕЛ КУРСОВОЙ РАБОТЫ

5.1. Цель пятого задания курсовой работы**:** обеспечить реализацию алгоритма оптимизации. Найти минимум и максимум функции.

## 5.2. Задание:

Найти минимум функции с помощью метода комплексов Бокса. Реализовать парсер для считывания функции.

## 5.3. Теоретический аспект задачи:

Метод Больцано (дихотомии)[4]: Метод Больцано является вариацией метода деления отрезка пополам. Он ищет минимум на интервале [a, b]. Чтобы метод работал, надо чтобы производная в точках a и b были разных знаков.

Алгоритм:

1. Берём точку x = (a+b)/2
2. Если производная в точке x больше 0, то a присваиваем значение x, иначе точке b присваиваем значение x
3. Продолжаем алгоритм пока  и , где и это малые положительные числа.

Парсер: парсер основан на методе обратной польской записи [3] (далее обозначим как ОПЗ). Особенностью ОПЗ является то, что все аргументы расположены перед знаком операции. Алгоритм перевода из инфиксной нотации таков:

1. Пока есть символы для чтения
   1. Читаем новый символ
   2. Если символ число или постфиксная функция, то добавляем к выходной строке
   3. Если символ префиксная функция, то помещаем её в стек
   4. Если символ оператор, то если у оператора в стеке приоритет выше или равный, то достаём из стека и кладём в входную очередь, иначе добавляем оператор в стек.
   5. Если символ открывающая скобка, то помещаем в стек
   6. Если символ закрывающаяся скобка, то выталкиваем элементы их стека до тех пор, пока не дойдём до открывающейся скобки, при этом символ скобки удаляется из стека.
2. Если входная строка закончилась, то выталкиваем все символы из стека.

## 5.4. Формализация

Для метода Больцано было разработан 1 класс. Он содержит метод поиска производной в точке, а также саму реализацию метода Больцано.

Для парсера было разработано 3 класса

* Класс checkInput имеет метод checkInputString, которые проверяет входную строку (в нашем случае формулу) на отсутствие в ней ошибок.
* Класс SignChecker, который имеет методы, проверяющие символ на его принадлежность к какому-либо набору символов, будь то значение переменной, знак операции. Также имеет метод, для определения приоритета операции и использование операторов, т.е. использование какой-либо операции на двумя значениями.
* Класс RPN, который имеет методы для взаимодействия с обратной польской записью, например метод перевод с инфиксной нотации в обратную польскую запись, нахождение всех переменных в обратной польской записи, и подсчитывание формулы, записанной в обратной польской записи.

Классы показаны на рисунке 5.1.

В связи с тем, что каждый метод класса выполняет свою задачу, то у нас отсутствует причина модифицировать классы, кроме случая, если мы хотим использовать другой парсер для метода Больцано, что тем самым отвечают требованию единственности ответственности.

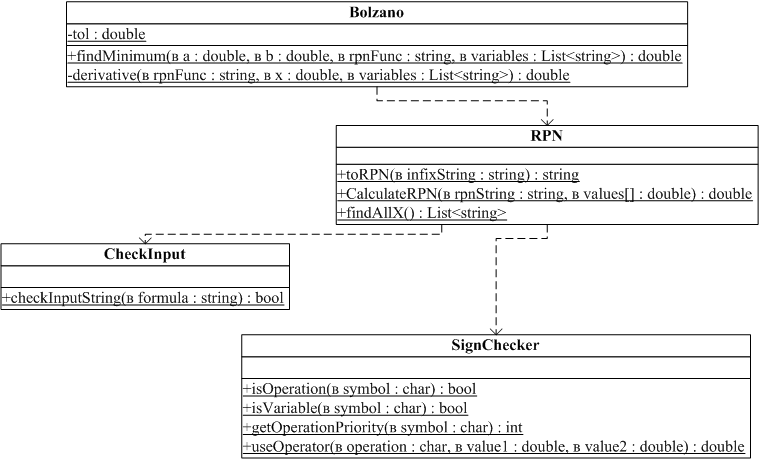
****

Рисунок 5.1

## 5.5. Спецификация программы

Используемые классы:

* Bolzano, для реализации метода Больцано
* RPN для реализации методов работы с обратной польской записью

Таблица 5.1. Описание атрибутов классов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Класс** | **Название атрибута** | **Тип атрибута** | **Модификатор доступа** | **Описание** |
| Bolzano | tol | int | private static | Точность поиска минимума функции |

Таблица 5.2. Описание методов классов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Класс** | **Название метода** | **Модификатор доступа** | **Входные параметры** | **Выходные параметры** | **Описание** |
| Bolzano | findMinimum | public static | double a;  double b;  string rpnFunc;  List<string>  variables | double | Нахождение минимума функции методом Больцано (дихотомии) |
| derivative | private static | string rpnFunc;  double x;  List<string> variables | double | Нахождение производной функции в точке x |
| RPN | toRPN | public static | string infixString | string | Перевод инфиксной нотации в обратную польскую запись |
| CalculateRPN | public static | string rpnString;  double[] values;  List<string>  variables | double | Заменяет все переменные на их значение и находит значение функции |
| findAllX | public static | string rpnString | List<string> | Находит все переменные в функции |
| Таблица 5.2. Описание методов классов | | | | | |
| CheckInput | checkInputString | public static | string formula | bool | Проверяет введённую формула на отсутствие ошибок. |
| SignChecker | isOperation | public static | char symbol | bool | Проверяет символ на принадлежность к символам операций |
| isVariable | public static | char symbol | bool | Проверяет символ на принадлежность к символам переменных |
| getOperationPriority | public static | char symbol | int | Возвращает приоритет операции |
| useOperation | public static | char operation;  double value1;  double value 2; | double | Возвращает результат применения операции над двумя числами |

## 5.7. Руководство программиста

**Назначение и условие применения программы**

Программа позволяет пользователю два варианта работы: нахождение минимума функции с помощью метода Больцано, нахождение значения функции в заданной точке. Для запуска программы требуется операционная система Windows и платформа .NET Framework не ниже версии 4.6.2. На устройстве программиста/оператора должна быть поддержка внешних устройств ввода «клавиатура», «мышь», устройства вывода «монитор». Пользователь должен обладать навыками работы с графическим интерфейсом операционной системы.

**Характеристика программы**

Программа следит за правильностью заполнения данных и выводит ошибку, если какое-то поле было введено неправильно.

**Обращение к программе**

Программу можно запустить двумя способами:

1. Через .exe файл, расположенный в папке проекта по пути bin\Debug
2. Через программу Visual Studio, при запуске отладки программы.

**Входные и выходные данные**

Для сценария «нахождение значение в точке» программа принимает на вход функцию и значение точки для функции. На выход программа выдаёт значение функции в данной точке.

Для сценария «нахождение минимума функции» программа принимает на вход функцию и границы поиска минимума. На выход программа выдаёт найденный минимум функции.

**Сообщения**

Программа выводит на экран пользователя, сообщение об ошибке, если тот неправильно ввёл какие-либо данные.

Для обоих сценариев, если функция будет введена неправильно (незакрытые скобки, два знака операции подряд и т.д.), то после нажатия кнопки «Считать функцию» для первого сценария, и кнопки «Найти минимум» для второго сценария программа выдаст ошибку (рисунок 5.6.).

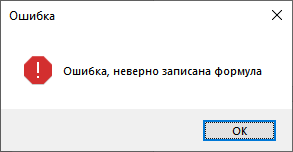


Рисунок 5.6.

Для первого сценария, если значение точки введено неверно (присутствуют символы, вместо чисел), то при нажатии кнопки «Найти значение» программа выдаст ошибку (рисунок 5.7.)

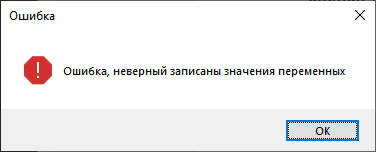


Рисунок 5.7.

Для второго сценария, если в полях под надписью: «Введите границы поиска минимума» будут какие-либо символы, кроме цифр, то программа выдаст ошибку (рисунок 5.8.)

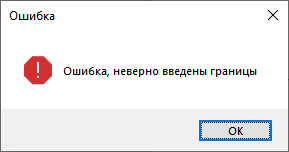


Рисунок 5.8.

Так как метод Больцано требует, чтобы производная функции имела разные знаки на границах поиска, то при несоблюдение этого условия программа выдаст ошибку (рисунок 5.9.)

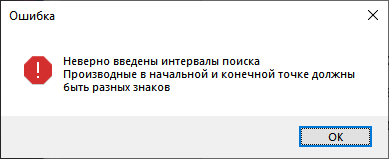


Рисунок 5.9.

## 5.8. Контрольный пример

На рисунках 5.10. и 5.1. приведено нахождения минимума функции  в сделанной программе и Matlab соответственно. Как мы видим значения минимума в обоих случаях примерно близки к истинному значению 3.

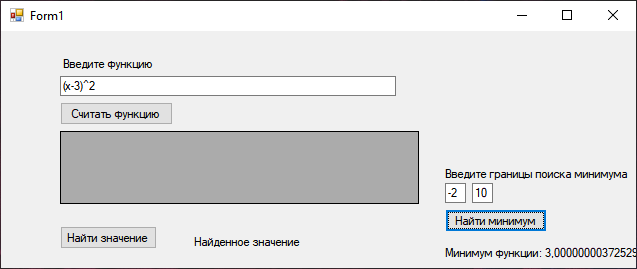


Рисунок 5.10.

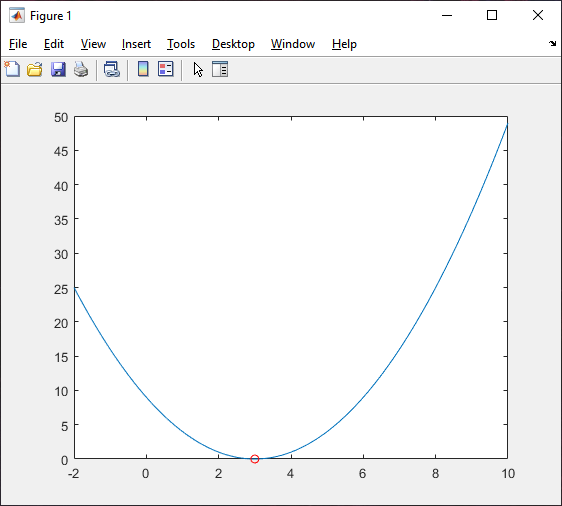


Рисунок 5.11.

На рисунках 5.12 и 5.13 мы видим поиски минимума функции  в сделанной программе и Matlab соответственно. Как мы видим значения минимума в обоих случаях примерно близки к истинному значению 2.718.

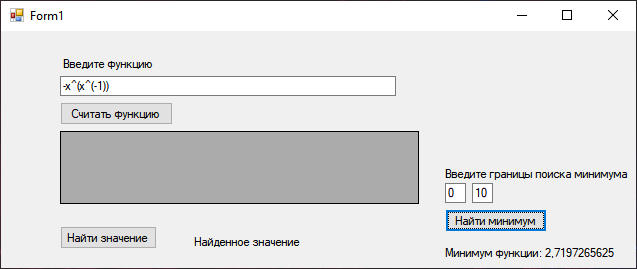


Рисунок 5.12.

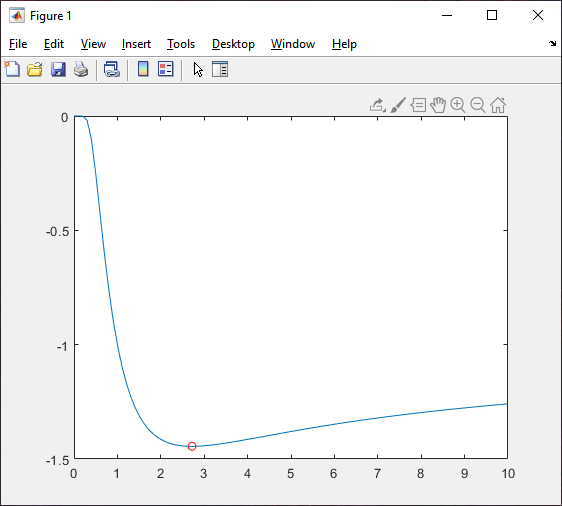


Рисунок 5.13.

## 5.9. Вывод

По результаты тестирования метода можно сказать, что когда выполнено условие для его работы, и при низкой заданной точности метод находит правильно точку минимума, близко к той, которую требуется найти.

## 5.10. Листинг

Класс Bolzano

class Bolzano

{

//Tolerance of minimum

static double tol = 0.00000001;

//Finding minimum

public static double findMinimum(double a, double b, string rpnFunc, List<string> variables)

{

//Throws exception if we have unsatisfying points

if (derivative(rpnFunc, a, variables) \* derivative(rpnFunc, b, variables) > 0)

throw new Exception("Неверно введены интервалы поиска");

double x; //Current point

do

{

x = (a + b) / 2; //Finding x at this position

//If derivative in x point is bigger than 0

if (derivative(rpnFunc, x, variables) > 0)

//Than b becomes x

b = x;

else

//Else a becomes x

a = x;

} while ((Math.Abs(derivative(rpnFunc, x, variables)) >= tol) && (Math.Abs(b - a) >= tol));

return x;

}

//Findin derivative in x point

private static double derivative(string rpnFunc, double x, List<string> variables)

{

//Making an array for x + derivative tolerance

double[] dx = new double[1];

dx[0] = x + 1E-12;

//Making an array for x point

double[] xArr = new double[1];

xArr[0] = x;

//Finding f(x) and f(x+dx)

double fxdx = RPN.CalculateRPN(rpnFunc, variables, dx);

double fx = RPN.CalculateRPN(rpnFunc, variables, xArr);

//Finding derivative

double result = (fxdx - fx) / 1E-12;

return result;

}

}

Класс RPN

class RPN

{

public static string toRPN(string infixString)

{

infixString = infixString.Replace(" ", ""); //Replacing all spaces

if (!CheckInput.checkInputString(infixString)) //If we have wrong input string

throw new ArgumentException(); //Than throw exception

Stack<char> operationStack = new Stack<char>(); //Stack for all operations

char lastOperation; //Keeping last operation

string resultString = ""; //String with result

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

{

//If it is number, than add to resultString

if (SignChecker.isVariable(infixString[i]))

{

resultString += infixString[i];

}

//If it is operation

else if (SignChecker.isOperation(infixString[i]))

{

//If it is unary minus< then add 0 to result string

if (infixString[i] == '-' && (((i == 0) || (infixString[i-1] == '(')) && (SignChecker.isVariable(infixString[i+1]))))

{

resultString += 0;

}

//If we have operations in stack

if (operationStack.Count != 0)

{

//Taking last operation of stack

lastOperation = operationStack.Peek();

//If priority of operation from stack is higher, then pirority of current operation

if (SignChecker.getOperationPriority(lastOperation) < SignChecker.getOperationPriority(infixString[i]))

{

//Add current operation to stack

operationStack.Push(infixString[i]);

resultString += " ";

}

else

{

//Add operation from stack to result string

resultString += operationStack.Pop();

//Add current operation to stack

operationStack.Push(infixString[i]);

resultString += " ";

}

}

else

{

//Add current operation to stack

resultString += " ";

operationStack.Push(infixString[i]);

}

}

//If current symbol is (, then add it to stack

else if(infixString[i] == '(') {

operationStack.Push(infixString[i]);

}

//If current symbol is ), then return all symbols from stack, until (

else if(infixString[i] == ')')

{

while (operationStack.Peek() != '(')

{

resultString += operationStack.Pop();

}

operationStack.Pop();

}

}

//In the end, return all the values from stack

while (operationStack.Count != 0)

{

resultString += operationStack.Pop() + " ";

}

return resultString;

}

//Calculating function

public static Double CalculateRPN(string rpnString, List<string> variables, double[] values)

{

Stack<Double> numbersStack = new Stack<Double>(); //stack for numbers

Double value1, value2; //first and second values

string variable = ""; //Current variable

//Replaces all variables from rpn string to variables values

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

{

rpnString = rpnString.Replace(variables[i], values[i].ToString());

}

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

{

//If current symbol is variable

if (SignChecker.isVariable(rpnString[i]))

{

//Add to variable

variable += rpnString[i];

}

else if(rpnString[i] == ' ')

{

//If current symbol is space and we have something in variable

if (variable != "")

{

//Add variable to stack

numbersStack.Push(double.Parse(variable));

}

variable = "";

}

else if(rpnString[i] == '-' && i != rpnString.Length - 1 && SignChecker.isVariable(rpnString[i + 1]))

{

variable += rpnString[i];

}

else {

//If variable has something

if (variable != "")

{

//Add to stack

numbersStack.Push(double.Parse(variable));

variable = "";

}

//Take two values from stack

value2 = numbersStack.Pop();

value1 = numbersStack.Pop();

//Calculate them

numbersStack.Push(SignChecker.useOperator(rpnString[i], value1, value2));

}

}

if (variable != "")

{

numbersStack.Push(double.Parse(variable));

}

return numbersStack.Pop();

}

//Finding all variables in rpnString

public static List<string> findAllX(string rpnString)

{

Stack<Double> numbersStack = new Stack<Double>();//Stack for numbers

List<string> stringList = new List<string>(); //List of all variables

string variable = "";

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

{

//If current symbol is variable

if (SignChecker.isVariable(rpnString[i]))

{

//Add to current variable

variable += rpnString[i];

}

else

{

//If variable has symbols in itself

if (variable.Any(c => char.IsLetter(c)))

{

//Add variable to variable list

stringList.Add(variable);

}

variable = "";

}

}

if (variable != "")

{

//If variable has symbols in itself

if (variable.Any(c => char.IsLetter(c)))

{

//Add variable to variable list

stringList.Add(variable);

}

}

//Return list of variables

return stringList.Distinct().ToList();

}

}

Класс SignChecker

class SignChecker

{

public static bool isOperation(char symbol) //Checking if the symbol is operation

{

if (symbol == '+' ||

symbol == '\*' ||

symbol == '-' ||

symbol == '/' ||

symbol == '^')

{

return true;

}

else return false;

}

public static bool isVariable(char symbol) //Checking if symbol is variable

{

if (Char.IsDigit(symbol) ||

Char.IsLetter(symbol) ||

symbol == '.' ||

symbol == ',' ||

symbol == '—')

{

return true;

}

else return false;

}

public static int getOperationPriority(char symbol) //Returns the priority of the symbol

{

switch (symbol)

{

case '+': return 1;

case '-': return 1;

case '\*': return 2;

case '/': return 2;

case '^': return 3;

default: return 0;

}

}

public static double useOperator(char operation, Double value1, Double value2) //Calculating two values

{

switch (operation)

{

case '+': return value1 + value2;

case '-': return value1 - value2;

case '\*': return value1 \* value2;

case '/': return value1 / value2;

case '^': return Math.Pow(value1, value2);

default: return 0;

}

}

}

Класс CheckInput

class CheckInput

{

//Checks function

public static bool checkInputString(string formula)

{

int counter1=0,

counter2=2;

//Returns false if first symbol in function is operation (besides minus)

if (SignChecker.isOperation(formula[0]) && formula[0] != '-')

return false;

//Returs false if last symbol in string is operation

if (SignChecker.isOperation(formula[formula.Length-1]) )

return false;

//Counts ammount of opening and closing brackets

counter1 = formula.ToCharArray().Where(x => x == '(').Count();

counter2 = formula.ToCharArray().Where(x => x == ')').Count();

//If ammount of brackets doesn't equal, then returns false

if (counter1 != counter2)

return false;

//Checks if there is no two operations side by side

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

{

if (SignChecker.isOperation(formula[i]) && SignChecker.isOperation(formula[i - 1]))

return false;

}

return true;

}

}

**заключение**

В ходе выполнения курсового проекта были достигнуты все поставленные цели и задачи. Были разработаны приложения на языке C#, а также закреплены знания, полученные в течение семестра.

**список использованных источников**

1. MSDN – сеть разработчиков Microsoft.URL: https://msdn.microsoft.com/ru-ru/library/saxz13w4.aspx
2. Статья: Алгоритм Эдмондса-Карпа // Википедия. Свободная Эниклопедия. URL: https://ru.wikipedia.org/wiki/Алгоритм\_Эдмондса\_-\_Карпа (дата обращения: 20.11.2019).
3. Статья: Обратная польская запись // Википедия. Свободная Эниклопедия. URL: https://ru.wikipedia.org/wiki/Обратная\_польская\_запись (дата обращения: 03.12.2019).
4. Статья: Дихотомия // Википедия. Свободная Эниклопедия. URL: https://ru.wikipedia.org/wiki/ Дихотомия (дата обращения: 20.12.2019).