

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский
Нижегородский государственный университет им. Н.И. Лобачевского»
(ННГУ)

Институт информационных технологий, математики и механики

Направление подготовки: «Программная инженерия»

ОТЧЕТ

по лабораторной работе №4

***ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ СИСТЕМЫ ОБСЛУЖИВАНИЯ ПОТОКА
ЗАДАНИЙ НА ЭВМ (ОЧЕРЕДИ)***

Выполнил: студент группы
3822Б1ПР2

Е. А. Соловьева
Подпись

Нижний Новгород
2023

Содержание

1.ВВЕДЕНИЕ:.....	3
2. ПОСТАНОВКА ЗАДАЧИ:.....	4
3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ.....	5
3.1. <i>ВИЗУАЛЬНОЕ ПРИЛОЖЕНИЕ:</i>	5
3. РУКОВОДСТВО ПРОГРАММИСТА:.....	6
3.1 <i>ОПИСАНИЕ СТРУКТУР:</i>	6
3.2. <i>ОПИСАНИЕ ПРОГРАММЫ:</i>	7
4.ЗАКЛЮЧЕНИЕ	8
5.ЛИТЕРАТУРА:.....	9
6.ПРИЛОЖЕНИЕ:	10

1.Введение:

Цель лабораторной работы заключается в освоении динамической структуры данных очередь путем изучения различных способов хранения очереди и разработки методов и программ для решения задач с использованием очередей. Основной областью применения является эффективная организация выполнения задач на вычислительных системах. Очередь характеризуется порядком обработки значений, при котором новые элементы вставляются в конец очереди, а извлечение происходит из начала.

Имитационное моделирование реально существующих объектов и явлений-физических, химических, биологических, социальных процессов, живых и неживых систем, инженерных конструкций, конструируемых объектов – представляет собой построение математической модели, которая описывает изучаемое явление с достаточной точностью, и последующую реализацию разработанной модели на ЭВМ для проведения вычислительных экспериментов с целью изучения свойств моделируемых явлений. Использование имитационного моделирования позволяет проводить изучение исследуемых объектов и явлений без проведения реальных экспериментов.

Очередь — это абстрактный тип данных, который представляет собой список элементов, где добавление новых элементов происходит в конец очереди, а удаление — из начала. Это означает, что элемент, добавленный первым, будет удален первым (First-In-First-Out, FIFO).

2. Постановка задачи:

Для вычислительной системы (ВС) с одним процессором и однопрограммным последовательным режимом выполнения поступающих заданий требуется разработать программную систему для имитации процесса обслуживания заданий в ВС. При построении модели функционирования ВС должны учитываться следующие основные моменты обслуживания заданий:

- генерация нового задания;
- постановка задания в очередь для ожидания момента освобождения процессора;
- выборка задания из очереди при освобождении процессора после обслуживания очередного задания.

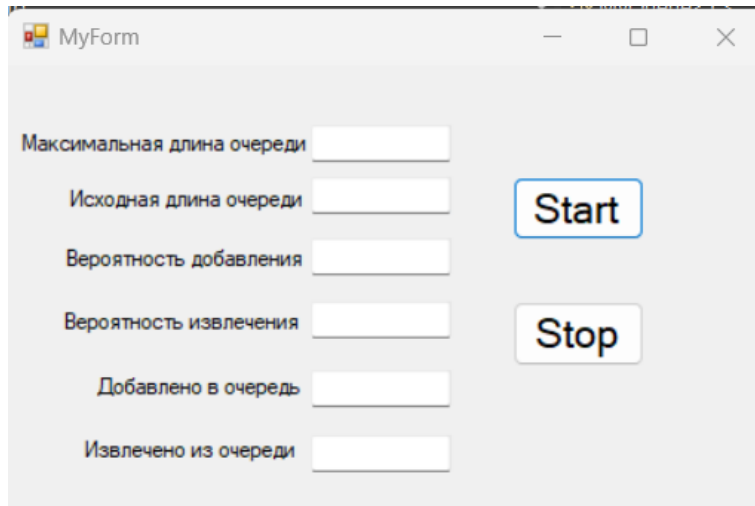
По результатам проводимых вычислительных экспериментов система имитации должна выводить информацию об условиях проведения эксперимента (интенсивность потока заданий, размер очереди заданий, производительность процессора, число тактов имитации) и полученные в результате имитации показатели функционирования вычислительной системы, в т.ч:

- количество поступивших в ВС заданий;
- количество отказов в обслуживании заданий из-за переполнения очереди;
- среднее количество тактов выполнения заданий;
- количество тактов простоя процессора из-за отсутствия в очереди заданий для обслуживания.

3. Руководство пользователя

3.1. ВИЗУАЛЬНОЕ ПРИЛОЖЕНИЕ:

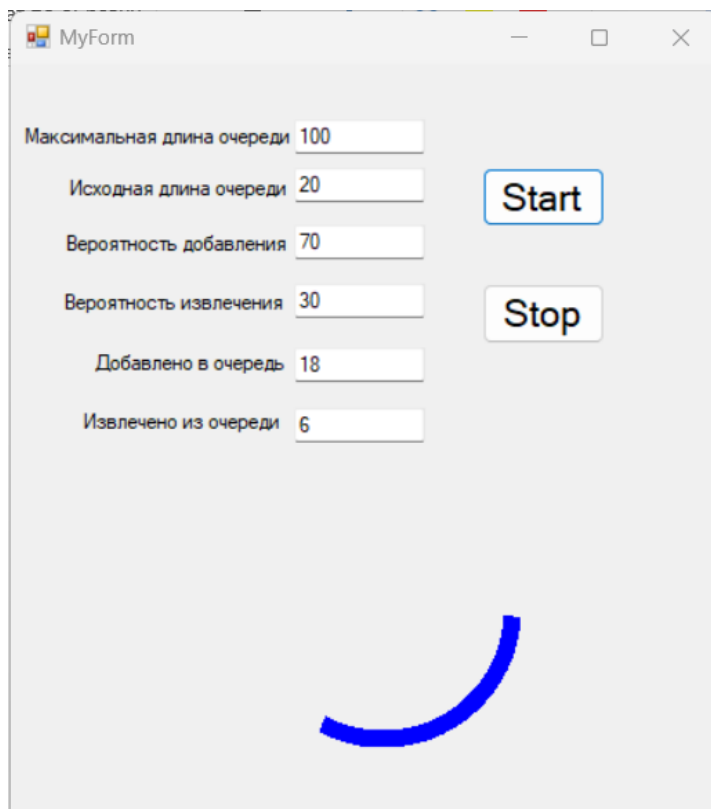
В визуальном приложении (рис.1) мы видим данное окно:



The screenshot shows a window titled "MyForm" with a light gray background. It contains six text input fields arranged vertically on the left, each with a label in Russian: "Максимальная длина очереди" (Maximum queue length), "Исходная длина очереди" (Initial queue length), "Вероятность добавления" (Probability of addition), "Вероятность извлечения" (Probability of extraction), "Добавлено в очередь" (Added to queue), and "Извлечено из очереди" (Extracted from queue). To the right of these fields are two buttons: "Start" (highlighted with a blue border) and "Stop".

(Рис.1)

Вводить данные мы можем только в первые 4 TextBox. Пользователь вводит максимальную длину очереди, исходную длину очереди, вероятность добавления и вероятность извлечения. Далее, нажав на кнопку «Start» появляется окно (рис.2).



The screenshot shows the same "MyForm" window after the "Start" button has been clicked. The first four input fields now contain numerical values: "Максимальная длина очереди" is 100, "Исходная длина очереди" is 20, "Вероятность добавления" is 70, and "Вероятность извлечения" is 30. The "Start" button is still highlighted with a blue border. The "Stop" button is also visible. Below the input fields, a thick blue arc is drawn, representing a queue or a process flow.

(Рис.2)

Если нажать на кнопку «Stop» то программа прекратит выполнение и данные в TextBox(5 и 6) обнулятся.

3. Руководство программиста:

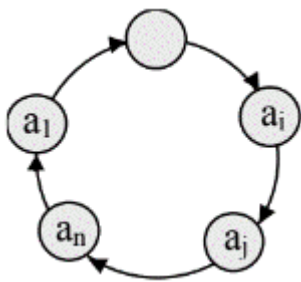
3.1 ОПИСАНИЕ СТРУКТУР:

Напомним, что динамическая структура есть математическая структура, которой соответствует частично-упорядоченное (по включению) базовое множество M , операции вставки и удаления элементы которого являются структурами данных. При этом отношения включения индуцируются операциями преобразования структуры данных.

Таким образом, очередь есть динамическая структура, операции вставки и удаления переводят очередь из одного состояния в другое, при этом добавление новых элементов осуществляется в конец очереди, а извлечение – из начала очереди (дисциплина обслуживания «первым пришел – первым обслужен»).

Важной задачей при реализации системы обслуживания очереди является выбор структуры хранения, обеспечивающей решение проблемы эффективного использования памяти без перепакетов и без использования связных списков (требующих дополнительных затрат памяти на указатели).

Как и в случае со стеком, в качестве структуры хранения очереди предлагается использовать одномерный (одноиндексный) массив, размещаемый в динамической области памяти. В связи с характером обработки значений, располагаемых в очереди, для указания хранимых в очереди данных необходимо иметь два указателя – на начало и конец очереди. Эти указатели увеличивают свое значение: один при вставке, другой при извлечении элемента. Таким образом, в ходе функционирования очереди может возникнуть ситуация, когда оба указателя достигнут своего наибольшего значения и дальнейшее пополнение очереди станет невозможным, несмотря на наличие свободного пространства в очереди. Одним из решений проблемы «движения» очереди является организация на одномерном массиве кольцевого буфера. Кольцевым буфером называется структура хранения, получаемая из вектора расширением отношения следования парой $p(a \text{ от } n, a \text{ от } 1)$.



(Рис.3) Общая схема структуры хранения вида кольцевой буфер.

3.2.ОПИСАНИЕ ПРОГРАММЫ:

Программа состоит из следующего:

Класс очередь:

Класс очередь – шаблонный класс с изменяемым типом хранения содержащий:

```
int MaxSize; // – максимальный размер
int begin; // – индекс элемента массива, в котором хранится первый элемент очереди
int end; // – индекс элемента массива, в котором хранится последний элемент очереди
size_t size; // – размер выделенной памяти
T* pMem; // – указатель на память, выделенную для кольцевого буфера
int cnt; // – (count) количество заполненных в очереди значений
```

MyQueue() – конструктор

~MyQueue() – деструктор

MyQueue(size_t s) – конструктор с размером s

size_t GetSize() – метод получения размера очереди

int GetMaxSize() – метод получения максимального размера очереди

bool IsEmpty() – метод проверить очередь на пустоту

bool IsFull() – метод проверить очередь на полноту

void Push(T val) – метод добавить элемент

T Pop() – метод удалить элемент

T GetFirst() – метод извлекает значение из массива "pMem" по индексу "begin"

T GetEnd() – метод извлекает значение из массива "pMem" по индексу "end"

4. Заключение

В ходе выполнения этой лабораторной работы, я успешно справилась со всеми поставленными задачами. Реализовала класс очередь и визуальный интерфейс.

5.Литература:

1. Википедия/Очередь - <https://ru.wikipedia.org/wiki/Очередь>
2. Википедия/Список - https://ru.wikipedia.org/wiki/Связный_список
3. Учебно-методическое пособие - <https://drive.google.com/drive>

6. Приложение:

Queue.h

```
#pragma once
#include <iostream>

const int MaxQueueSize = 100;
const int MinQueueSize = 1;

template <class T>
class MyQueue {
    int MaxSize; // - максимальный размер
    int begin; // - индекс элемента массива, в котором хранится первый элемент оче-
реди
    int end; // - индекс элемента массива, в котором хранится последний элемент оче-
реди
    size_t size; // - размер выделенной памяти
    T* pMem; // - указатель на память, выделенную для кольцевого буфера
    int cnt; // - (count) количество заполненных в очереди значений

public:
    MyQueue() {
        begin = 0;
        end = 0;
        size = 0;
        cnt = 0;
        MaxSize = 0;
        pMem = new T[size];
    }

    ~MyQueue() {
        delete[] pMem;
    }

    MyQueue(size_t s) {
        if (s < MinQueueSize || s > MaxQueueSize)
            throw "Неправильный размер очереди";
        size = s;
        begin = 0;
        end = 0;
        cnt = 0;
        MaxSize = s;
        pMem = new T[size];
    }

    size_t GetSize() {
        return cnt;
    }

    int GetMaxSize() {
        return MaxSize;
    }

    bool IsEmpty() { //- проверить очередь на пустоту
        if (cnt == 0)
            return true;
        return false;
    }

    bool IsFull() { //- проверить очередь на полноту
        if (cnt == size)
```

```

        return true;
    return false;
}

void Push(T val) { //добавить элемент
    if (IsFull())
        throw "Очередь заполнена";
    pMem[end++] = val;
    cnt++;
    end %= size; // выполняет модульное деление индекса "end" на размер мас-
сива,
    //чтобы гарантировать, что он остается в пределах допустимых значений
}

T Pop() { //удалить элемент
    if (IsEmpty()) {
        throw "Очередь пуста";
    }
    T tmp = pMem[begin];
    begin = (begin + 1) % size;
    cnt--;
    return tmp;
}

T GetFirst() { //извлекает значение из массива "pMem" по индексу "begin"
    if (IsEmpty()) {
        throw "Очередь пуста";
    }
    return pMem[begin];
}

T GetEnd() { //извлекает значение из массива "pMem" по индексу "end"
    if (IsFull()) {
        throw "Очередь заполнена";
    }
    return pMem[end];
}
};

```

Main.cpp

```

#include <iostream>
#include "Queue.h"
using namespace std;

int main()
{
    MyQueue<int> a(3);
    a.Push(1);
    a.Push(2);
    a.Push(3);
    for (int i = 0; i < 3; i++) {
        cout << a.Pop() << endl;
    }
}

```

QueueTest.cpp

```
#include "Queue.h"
#include <gtest.h>
TEST(MyQueue, can_create_queue)
{
    ASSERT_NO_THROW(MyQueue<int> q(5));
}

TEST(MyQueue, cant_create_queue_with_negative_size)
{
    ASSERT_ANY_THROW(MyQueue<int> q(-4));
}

TEST(MyQueue, can_check_queue_is_empty)
{
    MyQueue<int> q(5);
    EXPECT_TRUE(q.IsEmpty());
}

TEST(MyQueue, can_get_size)
{
    MyQueue<int> q(5);
    q.Push(2);
    q.Push(3);
    q.Push(4);
    EXPECT_EQ(q.GetSize(), 3);
}

TEST(MyQueue, can_put_elem)
{
    MyQueue<int> q(5);
    ASSERT_NO_THROW(q.Push(2));
}

TEST(MyQueue, can_check_first_element_withuot_remove)
{
    MyQueue<int> q(5);
    q.Push(2);
    q.Push(3);
    q.Push(4);
    EXPECT_EQ(q.GetFirst(), 2);
}
```

MyForm.cpp

```
#include "MyForm.h"

using namespace System;
using namespace System::Windows::Forms;

[STAThread]
void Main(cli::array<String^>^ args)
{
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);

    Gpafqueue::MyForm form;
```

```
        Application::Run(% form);  
    }
```

MyForm.h

```
#pragma once  
  
#include "../queuelib/Queue.h"  
  
namespace Gpafqueue {  
  
    using namespace System;  
    using namespace System::ComponentModel;  
    using namespace System::Collections;  
    using namespace System::Windows::Forms;  
    using namespace System::Data;  
    using namespace System::Drawing;  
  
    /// <summary>  
    /// Сводка для MyForm  
    /// </summary>  
    public ref class MyForm : public System::Windows::Forms::Form  
    {  
    public:  
        int MaxSize;  
        int Size;  
        double p;  
        double q;  
        int PopCount;  
        int PushCount;  
  
        MyQueue<int>*pQueue = nullptr;  
  
        int CenterX, CenterY, Width, Height;  
        Random^ rnd1;  
        Graphics^ gr;  
        Pen^ BlackPen;
```

```

        Pen^ WhitePen;

private: System::Windows::Forms::Label^ label1;
public:
private: System::Windows::Forms::Label^ label2;
private: System::Windows::Forms::Label^ label3;
private: System::Windows::Forms::Label^ label4;
private: System::Windows::Forms::Label^ label5;
private: System::Windows::Forms::Label^ label6;
private: System::Windows::Forms::TextBox^ textBox1;
private: System::Windows::Forms::TextBox^ textBox2;
private: System::Windows::Forms::TextBox^ textBox3;
private: System::Windows::Forms::TextBox^ textBox4;
private: System::Windows::Forms::TextBox^ textBox5;
private: System::Windows::Forms::TextBox^ textBox6;

        bool Flag;

public:
        MyForm()
        {
                InitializeComponent();
                gr = CreateGraphics();
                rnd1 = gcnew Random();
                BlackPen = gcnew Pen(Color::Blue);
                BlackPen->Width = 10.0F;
                WhitePen = gcnew Pen(SystemColors::Control);
                WhitePen->Width = 10.0F;
                CenterX = 140;
                CenterY = 240;
                Width = Height = 150;
                PopCount = PushCount = 0;
                Flag = false;
        }

protected:
        /// <summary>
        /// Освободить все используемые ресурсы.

```

```

    /// </summary>
    ~MyForm()
    {
        if (components)
        {
            delete components;
        }
    }

private: System::Windows::Forms::Button^ button1;
private: System::Windows::Forms::Timer^ timer1;
private: System::Windows::Forms::Button^ button2;
private: System::ComponentModel::IContainer^ components;

protected:

private:
    /// <summary>
    /// Обязательная переменная конструктора.
    /// </summary>

```

#pragma region Windows Form Designer generated code

```

    /// <summary>
    /// Требуемый метод для поддержки конструктора – не изменяйте
    /// содержимое этого метода с помощью редактора кода.
    /// </summary>
    void InitializeComponent(void)
    {
        this->components = (gcnew System::ComponentModel::Container());
        this->button1 = (gcnew System::Windows::Forms::Button());
        this->timer1 = (gcnew System::Windows::Forms::Timer(this->components));
        this->button2 = (gcnew System::Windows::Forms::Button());
        this->label1 = (gcnew System::Windows::Forms::Label());
        this->label2 = (gcnew System::Windows::Forms::Label());
    }

```

```

this->label3 = (gcnew System::Windows::Forms::Label());
this->label4 = (gcnew System::Windows::Forms::Label());
this->label5 = (gcnew System::Windows::Forms::Label());
this->label6 = (gcnew System::Windows::Forms::Label());
this->textBox1 = (gcnew System::Windows::Forms::TextBox());
this->textBox2 = (gcnew System::Windows::Forms::TextBox());
this->textBox3 = (gcnew System::Windows::Forms::TextBox());
this->textBox4 = (gcnew System::Windows::Forms::TextBox());
this->textBox5 = (gcnew System::Windows::Forms::TextBox());
this->textBox6 = (gcnew System::Windows::Forms::TextBox());
this->SuspendLayout();
//
// button1
//
this->button1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 16, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
this->button1->Location = System::Drawing::Point(546, 116);
this->button1->Margin = System::Windows::Forms::Padding(4, 5, 4, 5);
this->button1->Name = L"button1";
this->button1->Size = System::Drawing::Size(142, 66);
this->button1->TabIndex = 0;
this->button1->Text = L"Start";
this->button1->UseVisualStyleBackColor = true;
this->button1->Click += gcnew System::EventHandler(this,
&MyForm::button1_Click);
//
// timer1
//
this->timer1->Tick += gcnew System::EventHandler(this,
&MyForm::timer1_Tick);
//
// button2
//
this->button2->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 16, System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));

```



```

this->button2->Location = System::Drawing::Point(546, 245);
this->button2->Margin = System::Windows::Forms::Padding(4, 5, 4, 5);
this->button2->Name = L"button2";
this->button2->Size = System::Drawing::Size(142, 67);
this->button2->TabIndex = 1;
this->button2->Text = L"Stop";
this->button2->UseVisualStyleBackColor = true;
this->button2->Click += gcnew System::EventHandler(this,
&MyForm::button2_Click);
//
// label1
//
this->label1->AutoSize = true;
this->label1->Location = System::Drawing::Point(10, 67);
this->label1->Margin = System::Windows::Forms::Padding(4, 0, 4, 0);
this->label1->Name = L"label1";
this->label1->Size = System::Drawing::Size(322, 25);
this->label1->TabIndex = 2;
this->label1->Text = L"Максимальная длина очереди ";
//
// label2
//
this->label2->AutoSize = true;
this->label2->Location = System::Drawing::Point(62, 125);
this->label2->Margin = System::Windows::Forms::Padding(4, 0, 4, 0);
this->label2->Name = L"label2";
this->label2->Size = System::Drawing::Size(263, 25);
this->label2->TabIndex = 3;
this->label2->Text = L"Исходная длина очереди";
//
// label3
//
this->label3->AutoSize = true;
this->label3->Location = System::Drawing::Point(58, 186);
this->label3->Margin = System::Windows::Forms::Padding(4, 0, 4, 0);

```

```

this->label3->Name = L"label3";
this->label3->Size = System::Drawing::Size(263, 25);
this->label3->TabIndex = 4;
this->label3->Text = L"Вероятность добавления";
//
// label4
//
this->label4->AutoSize = true;
this->label4->Location = System::Drawing::Point(57, 252);
this->label4->Margin = System::Windows::Forms::Padding(4, 0, 4, 0);
this->label4->Name = L"label4";
this->label4->Size = System::Drawing::Size(261, 25);
this->label4->TabIndex = 5;
this->label4->Text = L"Вероятность извлечения";
this->label4->Click += gcnew System::EventHandler(this, &MyForm::la-
bel4_Click);
//
// label5
//
this->label5->AutoSize = true;
this->label5->Location = System::Drawing::Point(93, 320);
this->label5->Margin = System::Windows::Forms::Padding(4, 0, 4, 0);
this->label5->Name = L"label5";
this->label5->Size = System::Drawing::Size(228, 25);
this->label5->TabIndex = 6;
this->label5->Text = L"Добавлено в очередь";
//
// label6
//
this->label6->AutoSize = true;
this->label6->Location = System::Drawing::Point(78, 384);
this->label6->Margin = System::Windows::Forms::Padding(4, 0, 4, 0);
this->label6->Name = L"label6";
this->label6->Size = System::Drawing::Size(238, 25);
this->label6->TabIndex = 7;

```

```

this->label6->Text = L"Извлечено из очереди";
//
// textBox1
//
this->textBox1->Location = System::Drawing::Point(327, 61);
this->textBox1->Margin = System::Windows::Forms::Padding(4, 5, 4,
5);

this->textBox1->Name = L"textBox1";
this->textBox1->Size = System::Drawing::Size(148, 31);
this->textBox1->TabIndex = 8;
//
// textBox2
//
this->textBox2->Location = System::Drawing::Point(327, 116);
this->textBox2->Margin = System::Windows::Forms::Padding(4, 5, 4,
5);

this->textBox2->Name = L"textBox2";
this->textBox2->Size = System::Drawing::Size(148, 31);
this->textBox2->TabIndex = 9;
//
// textBox3
//
this->textBox3->Location = System::Drawing::Point(327, 178);
this->textBox3->Margin = System::Windows::Forms::Padding(4, 5, 4,
5);

this->textBox3->Name = L"textBox3";
this->textBox3->Size = System::Drawing::Size(148, 31);
this->textBox3->TabIndex = 10;
//
// textBox4
//
this->textBox4->Location = System::Drawing::Point(327, 245);
this->textBox4->Margin = System::Windows::Forms::Padding(4, 5, 4,
5);

this->textBox4->Name = L"textBox4";
this->textBox4->Size = System::Drawing::Size(148, 31);
this->textBox4->TabIndex = 11;

```

```

//
// textBox5
//
this->textBox5->Location = System::Drawing::Point(327, 316);
this->textBox5->Margin = System::Windows::Forms::Padding(4, 5, 4,
5);

this->textBox5->Name = L"textBox5";
this->textBox5->Size = System::Drawing::Size(148, 31);
this->textBox5->TabIndex = 12;
//
// textBox6
//
this->textBox6->Location = System::Drawing::Point(327, 383);
this->textBox6->Margin = System::Windows::Forms::Padding(4, 5, 4,
5);

this->textBox6->Name = L"textBox6";
this->textBox6->Size = System::Drawing::Size(148, 31);
this->textBox6->TabIndex = 13;
//
// MyForm
//
this->AutoScaleDimensions = System::Drawing::SizeF(12, 25);
this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
this->ClientSize = System::Drawing::Size(823, 1236);
this->Controls->Add(this->textBox6);
this->Controls->Add(this->textBox5);
this->Controls->Add(this->textBox4);
this->Controls->Add(this->textBox3);
this->Controls->Add(this->textBox2);
this->Controls->Add(this->textBox1);
this->Controls->Add(this->label6);
this->Controls->Add(this->label5);
this->Controls->Add(this->label4);
this->Controls->Add(this->label3);
this->Controls->Add(this->label2);
this->Controls->Add(this->label1);

```

```

        this->Controls->Add(this->button2);
        this->Controls->Add(this->button1);
        this->Name = L"MyForm";
        this->Text = L"MyForm";
        this->Load += gcnew System::EventHandler(this,
&MyForm::MyForm_Load);
        this->ResumeLayout(false);
        this->PerformLayout();

    }

#pragma endregion

private: System::Void MyForm_Load(System::Object^ sender, System::EventArgs^ e)
{
}

void DrawQueue() {
    //Вычисляем стартовый угол
    if (!pQueue->IsEmpty()) {
        int start = 360 * pQueue->GetFirst() / pQueue->GetMaxSize();
        int finish = 360 * pQueue->GetSize() / pQueue->GetMaxSize();
        //нарисовать дугу от tail до head
        gr->DrawArc(BlackPen, CenterX, CenterY, Width, Height, start, fin-
ish);
    }
}

void Clean() {
    //Вычисляем стартовый угол
    if (!pQueue->IsEmpty()) {
        int start = 360 * pQueue->GetFirst() / pQueue->GetMaxSize();
        int finish = 360 * pQueue->GetSize() / pQueue->GetMaxSize();
        //нарисовать дугу цветом фона от tail до head
        gr->DrawArc(WhitePen, CenterX, CenterY, Width, Height, start, fin-
ish);
    }
}

private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e)
{

```

```

        if (!Flag) {
            try {
                MaxSize = System::Convert::ToInt32(textBox1->Text);
                Size = System::Convert::ToInt32(textBox2->Text);
                pQueue = new MyQueue<int>(MaxSize);
                for (int i = 1; i <= Size; ++i)
                    pQueue->Push(1);

                p = System::Convert::ToDouble(textBox3->Text);
                q = System::Convert::ToDouble(textBox4->Text);
            }
            catch (Exception^ ex) {
                MessageBox::Show(ex->Message);
            }

            Flag = true;
            timer1->Enabled = true;
        }
        else
        {
            MessageBox::Show("!");
            return;
        }

        //Graphics^ gr = this -> CreateGraphics();
        ///gr->DrawLine(Pens::Black, 10,10,100,200);
        //Pen^ BlackPen = gcnew Pen(Color::DeepPink);
        //BlackPen->Width = 10.0F;
        //gr->DrawArc(BlackPen,10,10,200,200,0,90);
        //timer1->Enabled = true;
    }

private: System::Void timer1_Tick(System::Object^ sender, System::EventArgs^ e)
{
    //Стереть очередь
    Clean();
    try {

```

```

        int temp = std::rand() % 100;
        //Обновить состояние очереди
        if (temp < p) {
            pQueue->Push(1);
            PushCount++;
        }
        temp = std::rand() % 100;
        if (temp < q) {
            pQueue->Pop();
            PopCount++;
        }
    }
    catch (Exception^ ex) {

    }

    //Нарисовать очередь
    DrawQueue();
    //Обновить статистику
    textBox5->Text = Convert::ToString(PushCount);
    textBox6->Text = Convert::ToString(PopCount);
}

private: System::Void button2_Click(System::Object^ sender, System::EventArgs^ e)
{
    timer1->Enabled = false;
    Clean();

    Flag = false;
    delete pQueue;

    PushCount = 0;
    PopCount = 0;

    textBox5->Text = "";
    textBox6->Text = "";
}

```

```
        this->Refresh();
    }
    private: System::Void label4_Click(System::Object^ sender, System::EventArgs^ e)
    {
    };
}
```