МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

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»

Профиль подготовки: «Общий»

**Отчет по лабораторной работе**

на тему:

**«Очередь»**

**Выполнил:** студент группы 3822Б1ПР2

Чернова Наталья Алексеевна

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись

Нижний Новгород  
2023

**Содержание**

[1. ВВЕДЕНИЕ 3](#_Toc4)

[2. ПОСТАНОВКА ЗАДАЧИ 4](#_Toc5)

[3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 5](#_Toc6)

[4. РУКОВОДСТВО ПРОГРАММИСТА 7](#_Toc7)

[4.1 Описание структуры программы 7](#_Toc8)

[4.2 Описание структур данных 7](#_Toc9)

[5. ОПИСАНИЕ АЛГОРИТМОВ 9](#_Toc10)

[6. ЭКСПЕРИМЕНТЫ 10](#_Toc11)

[7. ЗАКЛЮЧЕНИЕ 11](#_Toc12)

[8.ЛИТЕРАТУРА 12](#_Toc13)

[9.ПРИЛОЖЕНИЕ 13](#_Toc14)

# 1. ВВЕДЕНИЕ

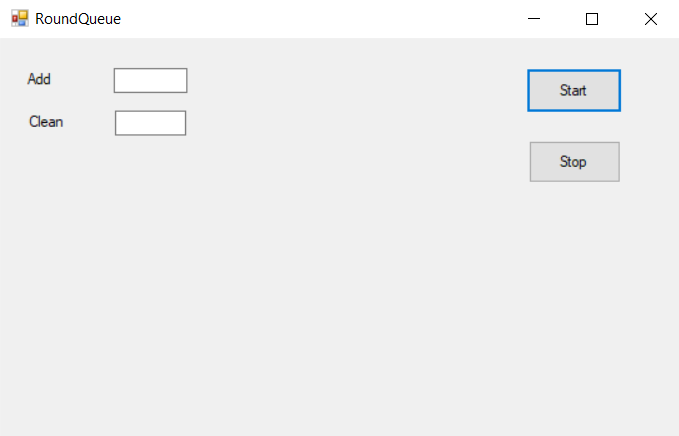
Очередь – это абстрактный тип данных, где элементы хранятся по принципу FIFO (от англ. “first in – first out”, «первым вошёл, первым вышел»). Это означает, что элементы можно добавлять только в конец очереди, извлекать – только из начала. В данной лабораторной работе будет выполнена реализация динамического класса очереди, а также реализация очереди на кольцевом буфере в визуальном приложении.

# 2. ПОСТАНОВКА ЗАДАЧИ

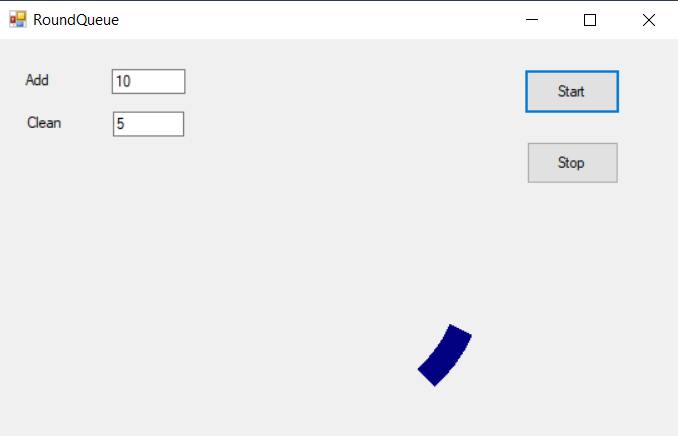
Задача данной работы – реализовать шаблонный класс очередь, а также реализовать очередь на кольцевом буфере в визуальном приложении с возможностью задания параметров для увеличения и уменьшения длины очереди.

# 3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

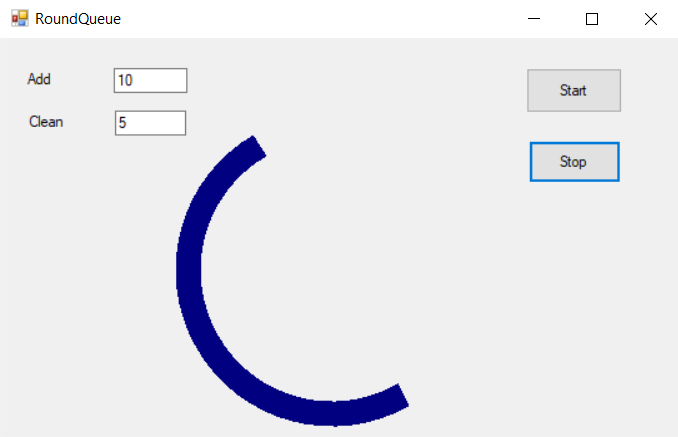
Изначально у нас есть окно, в которое нужно вводить количество добавляемых и удаляемых членов очереди.



После ввода данных и нажатия на кнопку «Start» появится графический элемент в виде части окружности, показывающий заполненность очереди.



При нажатии кнопки «Stop» программа приостанавливает свою работу.



# 4. РУКОВОДСТВО ПРОГРАММИСТА

## 4.1 Описание структуры программы

Программа состоит из нескольких проектов:

* Проект queue, содержащий файл main.cpp, где находится проверка работоспособности очереди
* Проект queuelib, содержащий файл MyQueue.h, где находится реализация класса очереди.
* Проект queuetest, содержащий тесты для проверки корректности работы класса очереди. Реализовано с помощью фреймворка GoogleTest.
* Графический проект RoundQueue, содержащий реализацию вывода на экран

## 4.2 Описание структур данных

TQueue (шаблонный класс)

p

protected:

int size; //размер очереди

int start; //индекс начального элемента

int end; //индекс последнего элемента

int count; //количество запомненных элементов

T\* mas; //массив размера size

public:

TQueue(int n = 0); //конструктор

TQueue(TQueue <T>& queue); //конструктор копирования

~TQueue(); //деструктор

void Push(T a); //добавить элемент в очередь

T Get(); //получить элемент

T Pop(); //получить элемент и удалить элемент из очереди

bool IsFull(); //проверка, полна ли очередь

bool IsEmpty(); //проверка, пуста ли очередь

int GetSize(); //получить размер

int GetStart(); //получить индекс начального элемента

int GetEnd(); //получить индекс последнего элемента

int GetCount(); //получить количество элементов

TQueue& operator=(const TQueue<T>& queue); //оператор присваивания

bool operator!=(const TQueue<T>& queue); //оператор сравнения

bool operator==(const TQueue<T>& queue); //оператор сравнения

friend ostream& operator<<(ostream& ostr, const TQueue<T>& queue) noexcept //оператор вывода

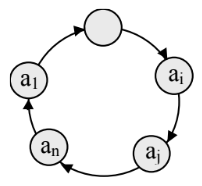
friend istream& operator>>(istream& istr, const TQueue<T>& queue) noexcept //оператор ввода

# 5. ОПИСАНИЕ АЛГОРИТМОВ

Очередь – это динамическая структура, где 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 на основе принципа FIFO, от англ. “first in – first out”, «первым вошёл, первым вышел»).

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

Как и в случае со стеком, в качестве структуры хранения очереди в данной работе используется одномерный 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(an,a1).



# 6. ЭКСПЕРИМЕНТЫ

Эксперименты проводились на ПК с следующими параметрами:

1. Операционная система: Windows 10

2. Процессор: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx, 2100 МГц

3. Версия Visual Studio: 2022

|  |  |
| --- | --- |
| Кол-во элементов (n) | Время работы оператора добавления элементов (ms) |
| 100000 | 1 |
| 300000 | 2,5 |
| 600000 | 6 |

Здесь представлено время работы программы, добавляющей в очередь n элементов. При количестве элементов n<100000 время работы программы менее миллисекунды. При большем количестве элементов сложность работы оператора примерно равна O(n).

# 7. ЗАКЛЮЧЕНИЕ

В ходе работы были реализован класс очереди (TQueue). Также был реализован визуальный интерфейс, для более удобного представления результатов работы.

# 8.ЛИТЕРАТУРА

[Очередь (программирование) — Википедия (wikipedia.org)](https://ru.wikipedia.org/wiki/%D0%9E%D1%87%D0%B5%D1%80%D0%B5%D0%B4%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))

[Связный список — Википедия (wikipedia.org)](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D1%8F%D0%B7%D0%BD%D1%8B%D0%B9_%D1%81%D0%BF%D0%B8%D1%81%D0%BE%D0%BA#%D0%9B%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D1%8B%D0%B9_%D1%81%D0%B2%D1%8F%D0%B7%D0%BD%D1%8B%D0%B9_%D1%81%D0%BF%D0%B8%D1%81%D0%BE%D0%BA)

[Pract\_ADS.pdf - Google Диск](https://drive.google.com/file/d/1aZEfnRSgA7IhMccdLTvbJ5noZlXPC1BK/view)

Лабораторный практикум. Составители:Барышева И.В., Мееров И.Б., Сысоев А.В., Шестакова Н.В. Под редакцией Гергеля В.П. Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017. – 105с.

# 9.ПРИЛОЖЕНИЕ

MyQueue.h

//#ifndef \_MY\_STACK\_

#define \_MY\_STACK\_

#include <iostream>

using namespace std;

template <class T>

class TQueue {

protected:

int size;

int start;

int end;

int count;

T\* mas;

public:

TQueue(int n = 0);

TQueue(TQueue <T>& queue);

~TQueue();

void Push(T a);

T Get();

T Pop();

bool IsFull();

bool IsEmpty();

int GetSize();

int GetStart();

int GetEnd();

int GetCount();

TQueue& operator=(const TQueue<T>& queue);

bool operator!=(const TQueue<T>& queue);

bool operator==(const TQueue<T>& queue);

//операторы вводы и выводы

friend ostream& operator<<(ostream& ostr, const TQueue<T>& queue) noexcept {

ostr << "My Queue: " << endl;

for (int i = 0; i < queue.size; i++) {

ostr << queue.mas[i] << endl;

}

return ostr;

}

friend istream& operator>>(istream& istr, const TQueue<T>& queue) noexcept {

for (int i = 0; i < queue.size; i++) {

istr >> queue.mas[i];

}

return istr;

}

};

template <class T>

TQueue<T>::TQueue(int n) {

if (n > 0)

{

size = n;

start = 0;

end = 0;

count = 0;

mas = new T[size];

}

else

{

throw "Negative size";

}

}

template <class T>

TQueue<T>::TQueue(TQueue<T>& queue) {

if (queue.mas == nullptr)

{

size = 0;

start = 0;

end = 0;

count = 0;

mas = nullptr;

}

else

{

size = queue.size;

start = queue.start;

end = queue.end;

count = queue.count;

mas = new T[size];

for (int i = 0; i < size; i++) {

mas[i] = queue.mas[i];

}

}

}

template <class T>

TQueue<T>::~TQueue() {

delete[]mas;

size = 0;

start = 0;

end = 0;

count = 0;

}

template <class T>

bool TQueue<T>::IsEmpty() {

if (count == 0) {

return true;

}

else {

return false;

}

}

template <class T>

bool TQueue<T>::IsFull() {

if (count == size) {

return true;

}

else {

return false;

}

}

template <class T>

void TQueue<T>::Push(T el) {

if (IsFull()) {

throw "Queue is full";

}

mas[end] = el;

count++;

end = (end + 1) % size;

}

template <class T>

T TQueue<T>::Get() {

if (IsEmpty()) {

throw "Queue is empty";

}

return mas[start];

}

template <class T>

T TQueue<T>::Pop() {

if (IsEmpty()) {

throw "Queue is empty";

}

T p = mas[start];

mas[start] = 0;

count--;

start = (start + 1) % size;

return p;

}

template <class T>

int TQueue<T>::GetSize() {

return size;

}

template <class T>

int TQueue<T>::GetStart() {

return start;

}

template <class T>

int TQueue<T>::GetEnd() {

return end;

}

template <class T>

int TQueue<T>::GetCount() {

return count;

}

template <class T>

TQueue<T>& TQueue<T>::operator=(const TQueue<T>& queue) {

if (this == &queue) {

return \*this;

}

delete[]mas;

size = queue.size;

start = queue.start;

end = queue.end;

count = queue.count;

mas = new T[size];

for (int i = 0; i < size; i++) {

mas[i] = queue.mas[i];

}

}

template <class T>

bool TQueue<T>::operator==(const TQueue<T>& queue) {

if (this == &queue) {

return true;

}

if (size != queue.size) {

return false;

}

for (int i = 0; i < size; i++) {

if (mas[i] != queue.mas[i]) {

return false;

}

}

return true;

}

template <class T>

bool TQueue<T>::operator!=(const TQueue<T>& queue) {

if (this == &queue) {

return false;

}

if (size != queue.size) {

return true;

}

for (int i = 0; i < size; i++) {

if (mas[i] != queue.mas[i]) {

return true;

}

}

return false;

}

RoundQueue.h

#pragma once

#include <iostream>

#include <string>

#include <msclr\marshal\_cppstd.h>

#include <MyQueue.h>

const int k = 30;

namespace RoundQueue {

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 RoundQueue

/// </summary>

public ref class RoundQueue : public System::Windows::Forms::Form

{

public:

RoundQueue(void)

{

InitializeComponent();

//

//TODO: добавьте код конструктора

//

}

protected:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~RoundQueue()

{

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: Random^ rnd1;

private: System::Windows::Forms::TextBox^ textBox1;

protected:

private: System::Windows::Forms::TextBox^ textBox2;

private: System::Windows::Forms::Label^ label1;

private: System::Windows::Forms::Label^ label2;

protected: Graphics^ gr;

protected: TQueue<int>\* queue;

protected: int add, clean = 0;

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->textBox1 = (gcnew System::Windows::Forms::TextBox());

this->textBox2 = (gcnew System::Windows::Forms::TextBox());

this->label1 = (gcnew System::Windows::Forms::Label());

this->label2 = (gcnew System::Windows::Forms::Label());

this->SuspendLayout();

//

// button1

//

this->button1->Location = System::Drawing::Point(561, 30);

this->button1->Name = L"button1";

this->button1->Size = System::Drawing::Size(103, 44);

this->button1->TabIndex = 0;

this->button1->Text = L"Start";

this->button1->UseVisualStyleBackColor = true;

this->button1->Click += gcnew System::EventHandler(this, &RoundQueue::button1\_Click);

//

// timer1

//

this->timer1->Enabled = true;

this->timer1->Tick += gcnew System::EventHandler(this, &RoundQueue::timer1\_Tick);

//

// button2

//

this->button2->Location = System::Drawing::Point(564, 101);

this->button2->Name = L"button2";

this->button2->Size = System::Drawing::Size(99, 42);

this->button2->TabIndex = 1;

this->button2->Text = L"Stop";

this->button2->UseVisualStyleBackColor = true;

this->button2->Click += gcnew System::EventHandler(this, &RoundQueue::button2\_Click);

//

// textBox1

//

this->textBox1->Location = System::Drawing::Point(121, 30);

this->textBox1->Name = L"textBox1";

this->textBox1->Size = System::Drawing::Size(77, 22);

this->textBox1->TabIndex = 2;

//

// textBox2

//

this->textBox2->Location = System::Drawing::Point(122, 72);

this->textBox2->Name = L"textBox2";

this->textBox2->Size = System::Drawing::Size(75, 22);

this->textBox2->TabIndex = 3;

//

// label1

//

this->label1->AutoSize = true;

this->label1->Location = System::Drawing::Point(25, 32);

this->label1->Name = L"label1";

this->label1->Size = System::Drawing::Size(32, 16);

this->label1->TabIndex = 4;

this->label1->Text = L"Add";

//

// label2

//

this->label2->AutoSize = true;

this->label2->Location = System::Drawing::Point(28, 74);

this->label2->Name = L"label2";

this->label2->Size = System::Drawing::Size(42, 16);

this->label2->TabIndex = 5;

this->label2->Text = L"Clean";

//

// RoundQueue

//

this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(727, 393);

this->Controls->Add(this->label2);

this->Controls->Add(this->label1);

this->Controls->Add(this->textBox2);

this->Controls->Add(this->textBox1);

this->Controls->Add(this->button2);

this->Controls->Add(this->button1);

this->Name = L"RoundQueue";

this->Text = L"RoundQueue";

this->ResumeLayout(false);

this->PerformLayout();

this->gr = CreateGraphics();

this->queue = new TQueue<int>(k);

this->rnd1 = gcnew Random();

}

#pragma endregion

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e) {

timer1->Enabled = true;

string a = msclr::interop::marshal\_as<std::string>(textBox1->Text);

add = stoi(a, 0);

string b = msclr::interop::marshal\_as<std::string>(textBox1->Text);

clean = stoi(b, 0);

}

private: System::Void timer1\_Tick(System::Object^ sender, System::EventArgs^ e) {

//Graphics^ gr = this->CreateGraphics();

Pen^ MyPen = gcnew Pen(Color::Navy);

MyPen->Width = 20;

Pen^ MyPenW = gcnew Pen(SystemColors::Control);

MyPenW->Width = 30;

if ((!(queue->IsFull())) && (rnd1->Next() % 100 < add) && add) {

queue->Push(rnd1->Next());

}

if ((!(queue->IsEmpty())) && (rnd1->Next() % 100 < clean) && clean) {

queue->Pop();

}

gr->DrawArc(MyPenW, 150, 70, 230, 230, 0, 360);

gr->DrawArc(MyPen, 150, 70, 230, 230, 360\*((queue->GetStart()\*1.0)/k), 360 \* ((queue->GetCount()\*1.0) / k));

}

private: System::Void button2\_Click(System::Object^ sender, System::EventArgs^ e) {

timer1->Enabled = false;

}

};

}

RoundQueue.cpp

#include "RoundQueue.h"

using namespace System;

using namespace System::Windows::Forms;

[STAThread]

void Main(cli::array<System::String^>^ args)

{

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

RoundQueue::RoundQueue form;

Application::Run(% form);

}