МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

**Отчет по ЛАБОРАТОРНОЙ РАБОТЕ №4**

**«Структура хранения данных: Очередь»**

**Выполнил:** студент группы 381903-3

Зинкин Кирилл Сергеевич

**Проверил:**

ассистент каф. МОСТ ИИТММ

Лебедев Илья Геннадьевич

Нижний Новгород

2020.

**Оглавление**

[Введение. 2](#_Toc532943564)

[Постановка задачи. 3](#_Toc532943565)

[Руководство пользователя. 4](#_Toc532943566)

[Руководство программиста. 5](#_Toc532943567)

[Описание структуры программы. 5](#_Toc532943568)

[Описание структур данных. 5](#_Toc532943569)

[Описание алгоритмов. 6](#_Toc532943570)

[Заключение. 7](#_Toc532943583)

[Литература. 8](#_Toc532943584)

# Введение

Лабораторная работа направлена на практическое освоение динамической структуры данных Очередь. С этой целью в лабораторной работе изучаются различные варианты структуры хранения очереди и разрабатываются методы и программы решения задач с использованием очередей. В качестве области приложений выбрана тема эффективной организации выполнения потока заданий на вычислительных системах.

Очередь характеризуется таким порядком обработки значений, при котором вставка новых элементов производится в конец очереди, а извлечение – из начала. Подобная организация данных широко встречается в различных приложениях. Рассматриваемая в рамках лабораторной работы схема имитации является одной из наиболее простых моделей обслуживания заданий в вычислительной системе и обеспечивает тем самым лишь начальное ознакомление с проблемами моделирования и анализа эффективности функционирования реальных вычислительных систем.

# Постановка задачи

В данной лабораторной работе нужно разработать эффективную библиотеку для хранения и работы с такой структурой данных, как очередь.

Для этого нам нужно:

* Описать и реализовать класс **TQueue**.
* Протестировать класс **TQueue** с помощью Google Test.
* Реализовать класс **TMyException** для обработки исключений, которые могут возникнуть в результате работы класса **TQueue**.
* Продемонстрировать работу класса **TQueue**.

# Руководство пользователя

Программа работает следующим образом:

* Создается очередь (в данном примере её размер равен 10).
* Проверяем очередь на пустоту.
* Заполняем очередь значениями от 1 до 10.
* Проверяем очередь на полноту.
* Забираем элемент из очереди.
* Проверяем очередь на полноту и пустоту.

# Руководство программиста

# Описание структуры программы

Программа состоит из модулей:

* **Queue** – содержит в себе файл **queue\_main.cpp** с реализацией примера использования класса **TQueue.**
* **QueueLib –** содержит в себе файл **Queue.h**, в котором описан и реализован класс **TQueue**.
* **QueueTest** – содержит в себе файл **test\_queue.cpp**, в котором находится набор тестов, для проверки работоспособности класса **TQueue**.
* **MyExceptionLib** –содержитв себе файл **MyException.h** с реализацией класса исключений **TMyException**.

# Описание структур данных

**Класс TQueue.**

Класс **TQueue** является шаблонным наследником от стека и содержит два поля со спецификатором **protected**:

* **int start** – индекс,куда помещается новый элемент.
* **int count** – количество элементов в очереди.

Далее идут методы класса со спецификатором доступа **public**:

* **TQueue(int \_size=0)** – конструктор инициализатор.
* **TQueue(TQueue<T> &obj)** – конструктор копирования.
* **void Put(T elem)** – функция, помещающая элемент в начало очереди.
* **T Get()** – функция, возвращающая элемент из конца очереди.
* **bool IsEmpty()** – функция, проверяющая очередь на пустоту.
* **bool IsFull()** – функция, проверяющая очередь на полноту.

**Класс TMyException.**

Класс **TMyException** содержит одно поле со спецификатором доступа **private**:

* **string str** – строка, хранящая сообщение об ошибке.

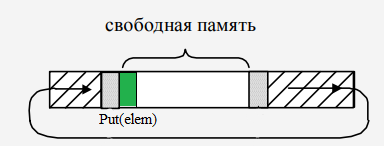
Далее идут методы класса со спецификатором доступа **public**:

* **TMyException(std::string \_str)** – конструктор инициализатор.
* **void what()** – метод вывода ошибки на экран.

# Описание алгоритмов

**Добавление элемента в очередь.**

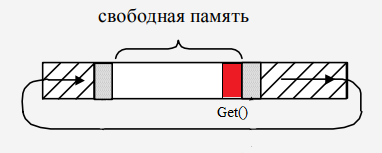
Добавляя элемент в очередь, мы помещаем его начало очереди, на которое указывает поле **start**. После добавления элемента в очередь, значение поля **start** увеличивается на 1 следующим образом: . Это нужно для того, чтобы мы могли заполнить всю свободную память, как показано на *Рис. 2*. Количество элементов в очереди **count** увеличиваем на 1.



*Рис. 2 Добавление элемента в очередь.*

**Удаление элемента из очереди.**

Удаляя элемент из очереди, мы забираем его из конца очереди. На конец очереди указывает поле **top**. После чего, значение поле **top** уменьшаем на 1 следующим образом: . Количество элементов в очереди **count** уменьшаем на 1.



*Рис. 3 Удаление элемента из очереди.*

# Заключение

В данной лабораторной работе мне удалось реализовать библиотеку для хранения и работы с очередью, а именно:

* Удалось реализовать класс **TQueue**.
* Удалось протестировать методы класса **TQueue**, а также обеспечить их работоспособность.
* Удалось реализовать класс для обработки исключений **TMyException**.

Таким образом, я смог реализовать структуру данных под названием очередь, использующую кольцевой буфер.

# Литература

1. Ссылка из Википедии про очередь:

<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)>

1. Гергель В.П. Методические материалы по курсу «Методы программирования 2», 2015.