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 учебной практике**

**«Структура хранения данных: Очередь»**

**Выполнил:** студент группы 381706-1

Денисов Владислав Львович

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

ассистент каф. МОСТ ИИТММ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лебедев И.Г

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc532752145)

[2. Постановка задачи 4](#_Toc532752146)

[3. Руководство пользователя 5](#_Toc532752147)

[4. Руководство программиста 6](#_Toc532752148)

[4.1 Описание структуры программы 6](#_Toc532752149)

[4.2 Описание структур данных 6](#_Toc532752150)

[4.3 Описание алгоритмов 7](#_Toc532752151)

[5. Заключение 9](#_Toc532752152)

[6. Литература 10](#_Toc532752153)

# Введение

Лабораторная работа направлена на практическое освоение динамической структуры данных под названием очередь.

Очередь – абстрактный тип данных с дисциплиной доступа к элементам «первый пришёл — первый вышел» (FIFO, англ.first in, first out). Добавление элемента возможно лишь в конец очереди, извлечение – только из начала очереди, при этом выбранный элемент из очереди удаляется.

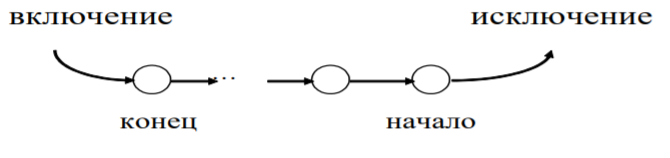


Рисунок 1 Схема работы очереди.

# Постановка задачи

В рамках лабораторной работы ставится задача эффективной реализации структуры данных – очередь.

Хранить данную структуру в этой лабораторной работе будем с помощью массива. Такая организация очереди удобна, если элемент информации занимает в памяти фиксированное количество длины, например, какую-то одну условную единицу. При этом отпадает необходимость хранения в элементе очереди явного указателя на следующий элемент очереди, что экономит память.

Однако обратим внимание на то, что использоваться будет не просто массив, а так называемый кольцевой буфер, расположение данных в котором будет представлено следующим образом:

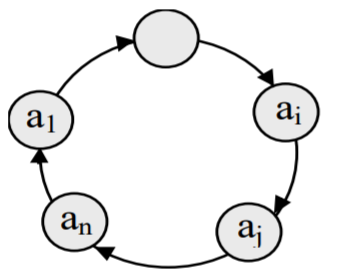


Рисунок 1 Кольцевой буфер.

Для работы с очередью будут реализованы операции:

* добавления элемента в очередь,
* извлечения элемента из очереди (с удалением),
* проверка очереди на полноту/пустоту,
* печать очереди на консоль.

Программное решение будет выглядеть следующим образом:

1. Класс очереди – TQueue, построенный на основе существующего класса TStack.
2. Класс для обработки исключений – TException, которые могут возникнуть при выполнении различных операций.
3. Программа, демонстрирующая работу класса TQueue.
4. Набор автоматических тестов с использованием Google C++ Testing Framework.

# Руководство пользователя

В качестве примера использования очереди предлагается следующее. После запуска тестовой программы пользователь вводит максимальный размер очереди. Создается очередь заданного размера для хранения значений целого типа.

Затем пользователю предлагается заполнить ее значениями. После этого происходит печать очереди на консоль. Происходит удаление одного элемента. Снова выполняется печать того, что осталось в очереди. Затем пользователю предлагается снова добавить элемент. Дополненная очередь будет выведена на экран.

Наконец, создается точно такая же очередь с помощью конструктора копирования для демонстрации его работы. Выводятся адреса исходной и скопированной очереди, для наглядности того, что копия действительно была создана.

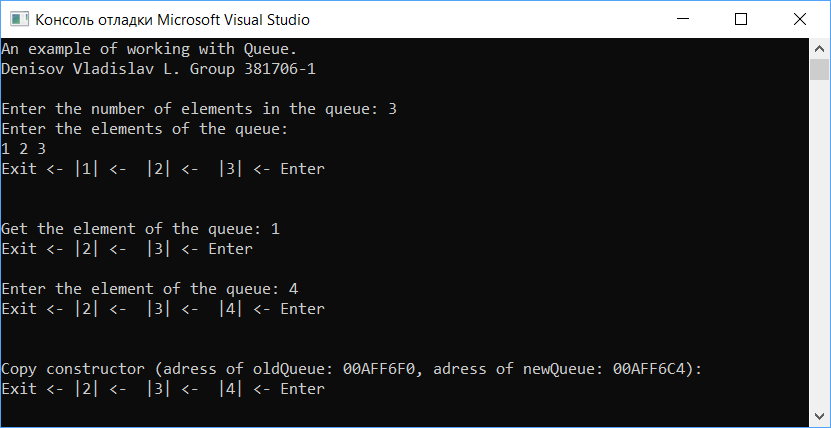


Рисунок 1 Пример работы демонстрационной программы

# Руководство программиста

## 4.1 Описание структуры программы

Программа состоит из следующих модулей:

* Модуль Queue. Содержит пример использования очереди. Реализация в файле *main\_ Queue.cpp.*
* Модуль QueueLib – статическая библиотека. Содержит файл Queue.h, в котором описан интерфейс и реализация шаблонного класса *TQueue*.
* Модуль QueueTest. Содержит 7 тестов, описанных в файле *QueueTest.cpp* и разработанных с помощью использования Google C++ Testing Framework.
* Модуль ExceptionLib – библиотека, позволяющая создавать собственные исключения.

## Описание структур данных

#### Класс TException – класс исключений.

Класс содержит 1 **private** поле *std::string msg* – переменная, хранящая сообщение об ошибке в виде строки.

И содержит 2 **public** элемента:

*TException(std::string \_str) : msg(\_msg)* – конструктор с одним параметром.

*void Print()* – метод отображения ошибки на консоль.

#### Класс TQueue – очередь.

Рассмотрим класс *TQueue* подробно.

template <class T> class TQueue : protected TStack<T> {…} – класс очереди является шаблонным, что позволяет использовать его для хранения данных любого типа. Кроме того, очередь построена с помощью наследования разработанного в прошлой лабораторной работе класса стека TStack. Используется спецификатор protected, чтобы запретить использование public-методов класса TStack и тем самым сохранить логику работы очереди.

**Наследуются из класса TStack:**

int size – размер очереди.

int top – индекс ячейки в массиве, в которой расположен элемент на выход.

T\* mas – указатель на область памяти, выделенную под хранение элементов очереди.

**Элементы класса, объявленные со спецификатором protected:**

int start – индекс ячейки в массиве, в которую можно добавить очередной элемент.

int count – количество элементов в очереди.

**Элементы класса, объявленные со спецификатором public:**

TQueue(int n = 0) – конструктор с одним параметром.

TQueue(TQueue <T> &q) – конструктор копирования.

virtual ~TQueue() – деструктор.

void Put(T a) – метод, позволяющий добавить новый элемент в очередь.

T Get() – метод изъятия элемента из очереди с удалением.

void Print() – метод отображения текущих элементов очереди.

bool IsFull() – метод проверки очереди на полноту.

bool IsEmpty() – метод проверки очереди на пустоту.

## Описание алгоритмов

Рассмотрим некоторые алгоритмы, работа которых не очевидна на первый взгляд.

**Добавление элемента в очередь.**

При добавлении элемента в очередь обращаемся к элементу массива с индексом *start*, записываем туда полученное значение *a.* Значение *start* увеличиваем согласно формуле *start = (start + 1) % size*. Такой подход обусловлен тем, что мы используем принцип хранения данных – кольцевой буфер. Запоминаем увеличение числа элементов в очереди путем изменения значения *count++*.

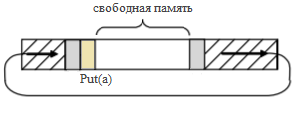


Рисунок 1 Добавление элемента в очередь.

**Удаление элемента из очереди.**

При удалении элемента из очереди обращаемся к переменной *top* и забираем значение по этому индексу. Значение *top* увеличиваем согласно формуле *top = (top + 1) % size.* Уменьшаем значение *count* на 1. Само значение в ячейке с индексом, по которому производилось возращение никак не затирается, т.к. оно будет перезаписано при следующем добавлении другого элемента.



Рисунок 2 Удаление элемента из очереди

# Заключение

В результате лабораторной работы была разработана библиотека, реализующая шаблонный класс очереди. Она позволяет создать объект класса очереди и выполнить с ним простейшие операции, задача реализации которых была поставлена в начале данной лабораторной работы.

Были разработаны и доведены до успешного выполнения тесты, разработанные для данного программного проекта с использованием Google C++ Testing Framework.

Программное решение было продемонстрировано с помощью простейшего набора операций над очередью. Описание примера работы с очередью было представлено в разделе «Руководство пользователя».

# Литература

1. Википедия: свободная электронная энциклопедия: на русском языке [Электронный ресурс] // URL: https://ru.wikipedia.org/wiki/Очередь\_(программирование) (дата обращения: 16.12.2018)
2. Методы программирования [Электронный ресурс] // URL: <http://www.itmm.unn.ru/files/2018/10/Primer-1.4.-Struktury-hraneniya-ocheredi.pdf> (дата обращения: 16.12.2018)