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**

**ЛАБОРАТОРНАЯ РАБОТА №4**

**«Структура хранения данных: Очередь»**

**Выполнил:** студент группы 381706-2

Макарихин Семён Андреевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Руководитель:**

Ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2019

Содержание

[1.Введение 3](#_Toc534316445)

[2. Цели и задачи 4](#_Toc534316446)

[3. Руководство пользователя](#_Toc534316448) 5

[4. Руководство программиста 1](#_Toc534316449)1

[4.1. Описание структуры программы 1](#_Toc534316450)1

[4.2. Описание функций и процедур, их алгоритмов 1](#_Toc534316451)1

[5. Эксперименты 1](#_Toc534316452)4

[6. Заключение 1](#_Toc534316453)5

[7. Литература 1](#_Toc534316454)6

[8. Приложения 1](#_Toc534316455)7

# Введение

Очередью называется упорядоченный набор элементов, которые могут удаляться с её начала и помещаться в её конец.

Очередь организована, в отличие от стека, согласно дисциплине обслуживания FIFO:

*FIRST — первый*

*INPUT — вошел*

*FIRST — первый*

*OUTPUT — вышел*

Очередь в программировании используется, как и в реальной жизни, когда нужно совершить какие-то действия в порядке их поступления, выполнив их последовательно. Примером может служить организация событий в Windows. Когда пользователь оказывает какое-то действие на приложение, то в приложении не вызывается соответствующая процедура (ведь в этот момент приложение может совершать другие действия), а ему присылается сообщение, содержащее информацию о совершенном действии, это сообщение ставится в очередь, и только когда будут обработаны сообщения, пришедшие ранее, приложение выполнит необходимое действие.

Существует несколько способов реализации очереди:

*с помощью одномерного массива;*

*с помощью связанного списка;*

*с помощью класса объектно-ориентированного программирования.*

# 2. Цели и задачи

В рамках лабораторной работы ставится задача эффективной реализации структуры данных – очередь и выполнение основных операций над ней:

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

Также как и при работе со стеками, хранить данную структуру в этой лабораторной работе будем с помощью массива. Однако использоваться будет не просто массив, а так называемый кольцевой буфер.

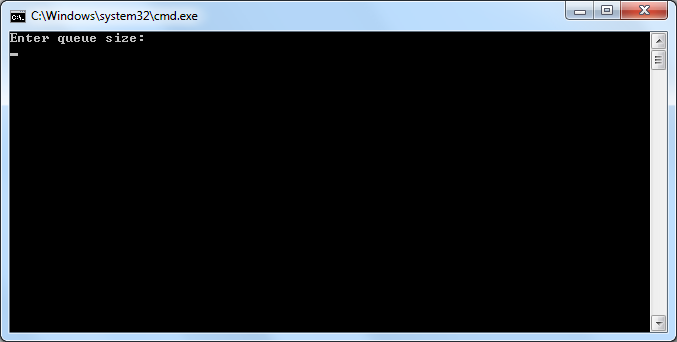
В процессе выполнения лабораторной работы требуется использовать систему контроля версий [Git](https://git-scm.com/book/ru/v2).

Выполнение работы предполагает решение следующих задач:

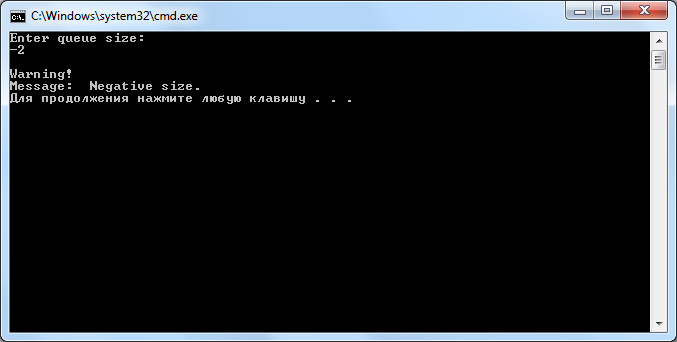
1. Реализация класса TQueue, построенного на основе существующего класса TStack.
2. Реализация класса для обработки исключений– TException, которые могут возникнуть при выполнении различных операций, согласно заданному интерфейсу.
3. Обеспечение работоспособности тестов и примера использования.
4. Реализация заготовок тестов, покрывающих все методы класса TQueue.
5. Модификация примера использования в тестовое приложение, позволяющее задавать очереди и осуществлять основные операции над ними.

# 3. Руководство пользователя

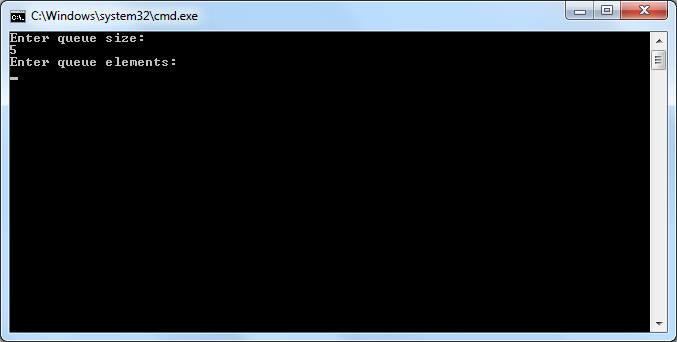
Запускаем программу из файла queue.cpp:



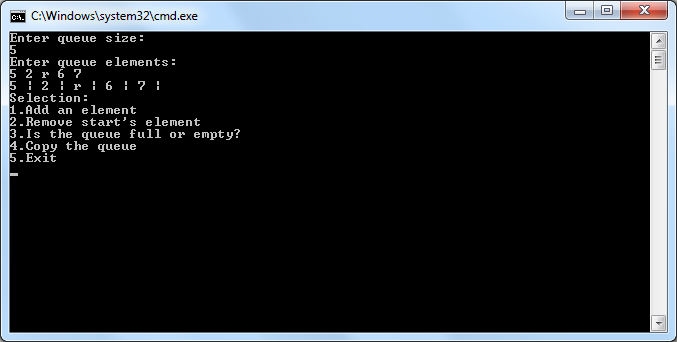
Пользователю предлагается ввести максимальный размер создаваемого очереди. Если ввести отрицательное число, бросится исключение:



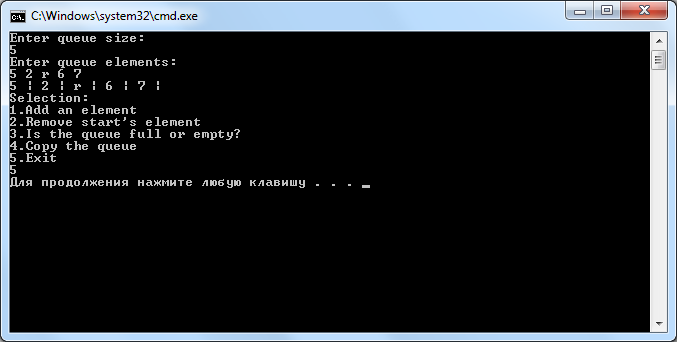
При положительном значении создастся очередь заданного размера для хранения значений любого типа. Затем пользователю предлагается заполнить все её ячейки.



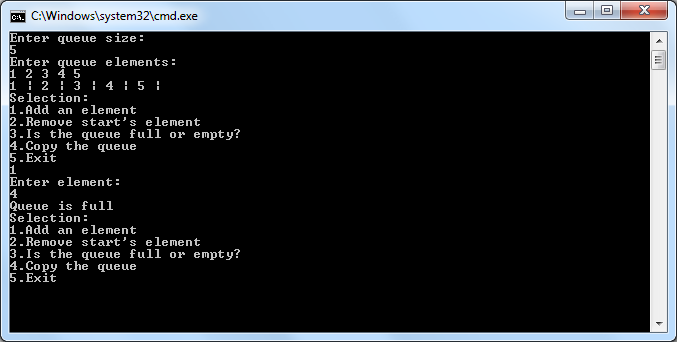
После заполнения происходит печать очереди на консоль, и предоставляется выбор операций: добавление элемента в очередь, удаление элемента, проверка на пустоту и полноту, копирование соответственно.

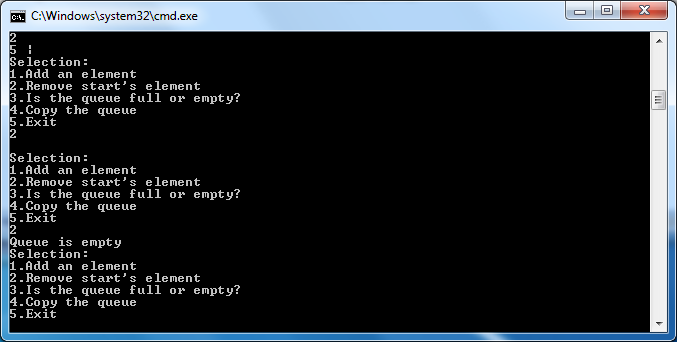


Выбрав ‘Exit’, можно выйти из программы.

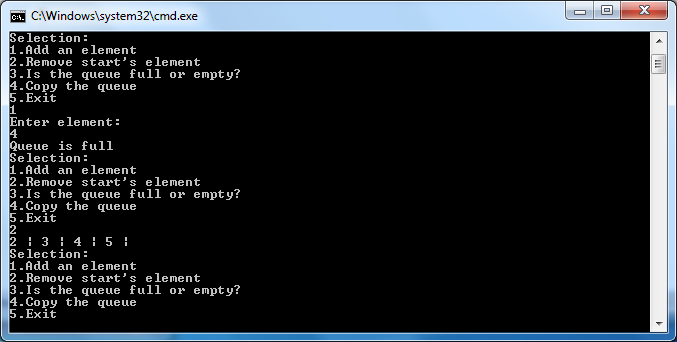


Если удалять из пустой очереди или добавлять в полную, программа выведет ошибку и не выполнит операцию.

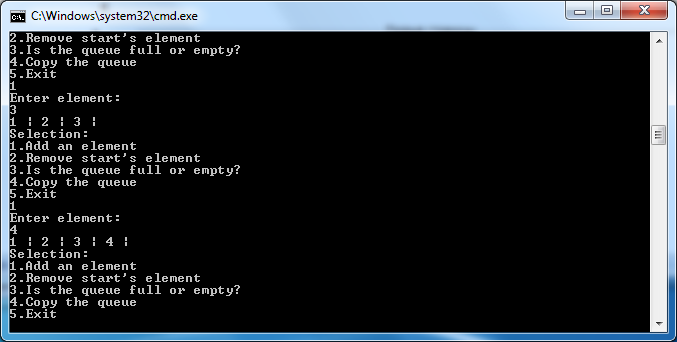




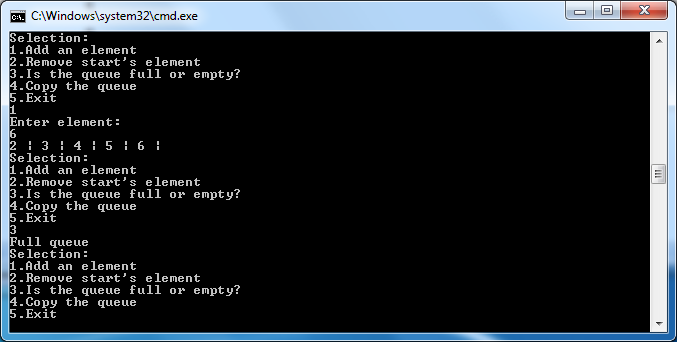
Чтобы удалить элемент – нужно нажать “2”. При удалении элемента из очереди – всегда берется ее первый элемент.

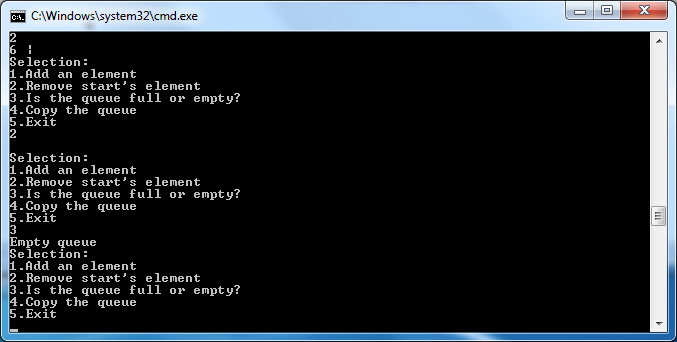


Чтобы добавить элемент в очередь нужно нажать “1”. Далее требуется ввести добавляемый элемент, который добавится в конец очереди.

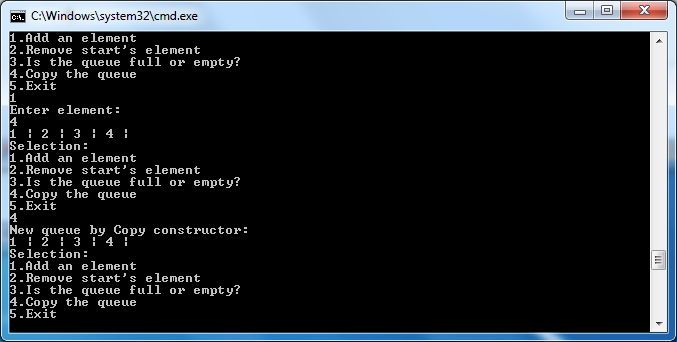


Чтобы проверить очередь на полноту или пустоту, нужно выбрать “3”. Программа выведет “Full queue”, если очередь полна, и “Empty queue”, если очередь пуста.





Чтобы скопировать очередь нужно выбрать “4”.



Алгоритм можно повторять много раз, не выходя из программы.

# 4. Руководство программиста

Разработка системы вычисления проводились в среде “Microsoft Visual Studio 2010”.

В данной работе будет использовано 3 класса:

* Класс «Стек» (TStack), реализованный с использованием массива.
* Класс «Очередь» (TQueue), построенного на основе класса TStack.
* Класс исключения (TExсeption).

## 4.1. Описание структуры программы

Модульная структура программы:

* tstack.h– модуль с классом TStack, в котором определен интерфейс шаблонного класса Стек и реализация его методов.
* exсeption.h – модуль с классом исключения TExсeption.
* tqueue.h– модуль с классом TQueue, в котором определен интерфейс шаблонного класса Очередь и реализация его методов.
* queue\_sample.cpp , sample\_performance\_check.cpp– модуль программы тестирования, с которым работает пользователь, в котором проводятся эксперименты.
* test\_main.cpp, test\_tqueue.cpp, test\_tstack.cpp – модуль с функциями тестирования для созданных классов. Содержат 20 тестов для класса TStack и 20 теста для класса TQueue.

## 4.2. Описание функций и процедур, их алгоритмов

Рассмотрим реализацию методов шаблонного класса TStack:

template <typename StackType>

class TStack

1. TStack(int n = 0) - конструктор класса, принимающий размер стека. По умолчанию создается стек размера 0 с позицией вершины стека 0.

2. TStack(TStack<StackType> &S) - конструктор копирования.

Принимает ссылку на объект класса TStack.

3. ~TStack() - деструктор. Освобождает выделенную под вектор память.

4. TStack& operator=(const TStack<StackType>& stack) – перегрузка оператора присваивания одного стека другому.

Присваивает полям первого объекта класса поля второго объекта класса.

5. int GetSize() - возвращает размер стека.

6. StackType Get() – метод изъятия элемента из вершины стека с удалением.

7. void Put(StackType A) - метод, позволяющий добавить новый элемент в стек.

8. bool IsFull() – метод проверки стека на полноту.

9. bool IsEmpty() – метод проверки стека на пустоту.

10. void PrintStack() - метод отображения текущих элементов стека.

11. bool operator==(const TStack<StackType>& stack) const – перегрузка оператора сравнения.

Выполняется проверка стеков на равенство. Возвращает true, если равенство выполняется, false в противном случае.

12. bool operator!=(const TStack<StackType>& stack) const - перегрузка оператора сравнения.

Выполняется проверка стеков на неравенство. Возвращает true, если неравенство выполняется, false в противном случае.

Рассмотрим реализацию методов шаблонного класса TQueue:

template <typename QueueType>

class TQueue : public TStack <QueueType>

1. TQueue(int n = 0) - конструктор класса с одним параметром.

2. TQueue(TQueue <QueueType> &Q) - конструктор копирования.

Принимает ссылку на объект класса TQueue.

3. TQueue& operator=(const TQueue<QueueType>& queue) - перегрузка оператора присваивания одной очереди другому.

Присваивает полям первого объекта класса поля второго объекта класса.

4. QueueType Get() - метод изъятия элемента из начала очереди с удалением.

5. void Put(QueueType Q) - метод, позволяющий добавить новый элемент в конец очереди.

6. bool IsFull() - метод проверки очереди на полноту.

7. bool IsEmpty() - метод проверки очереди на пустоту.

8. void PrintQueue() - метод отображения текущих элементов очереди.

9. bool operator!=(const TQueue<QueueType>& queue) const - перегрузка оператора сравнения.

Выполняется проверка очередей на равенство. Возвращает true, если равенство выполняется, false в противном случае.

10. bool operator==(const TQueue<QueueType>& queue) const - перегрузка оператора сравнения.

Выполняется проверка очередей на неравенство. Возвращает true, если неравенство выполняется, false в противном случае.

# 5. Эксперименты

В качестве примера рассмотрим перегрузку оператора присвоения для класса очереди (TQueue).

Теоретическая сложность выполнения алгоритма O(1).

Мы провели измерение присваивая очереди разное количество элементов: 10, 100, …, 1000000 элементов. Ниже вы можете увидеть график зависимости времени выполнения операции присвоения от количества элементов очереди. По приведенным данным можно сделать вывод, что практическая сложность выполнения алгоритма равна теоретической.

По горизонтали – количество присваиваемых элементов.

По вертикали - время выполнения программы.

# 6. Заключение

В результате лабораторной работы была разработана структура данных - очередь, а также освоены такие инструменты разработки программного обеспечения, как система контроля версий [Git](https://git-scm.com/book/ru/v2).

# 7. Литература

1. Лабораторный практикум: Учебно-методическое пособие / Мееров И.Б. [и др.] – Нижний Новгород: Нижегородский госуниверситет , 2017. – 105с.
2. Касперски К. Техника оптимизации программ. Эффективное использование памяти. — СПб.: БХВ-Петербург, 2003. — 464 с.: ил.
3. http://neerc.ifmo.ru/wiki/index.php?title=Очередь

# 8. Приложение