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 очереди»**

**Выполнил:** студент группы 381706-1

Силенко Дмитрий Игоревич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

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

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc533165735)

[2. Постановка задачи 4](#_Toc533165736)

[3. Руководство пользователя 5](#_Toc533165737)

[4. Руководство программиста 6](#_Toc533165738)

[4.1. Описание структуры программы 6](#_Toc533165739)

[4.2. Описание структур данных 6](#_Toc533165740)

[4.3. Описание алгоритмов 6](#_Toc533165741)

[5. Заключение 8](#_Toc533165742)

# Введение

**Основная цель данной работы** – разработка структуры хранения очереди.

Но для начала необходимо разобраться, что такое очередь и как она работает.

**Очередь** — это структура данных, представляющая собой упорядоченный набор элементов и построенная по принципу **LILO** (last in — last out: последним пришел — последним вышел). В очереди, если вы добавите элемент, который вошел самый первый, то он выйдет тоже самым первым. Другими словами, у очереди есть, так называемые, голова и хвост. Кладется новый элемент всегда в конец – хвост. Вывод производится наоборот из головы.

Чтобы понять принцип работы очереди можно представить себе магазинную очередь. В ней вы стоите посередине, чтобы оказаться напротив кассы, сначала необходимо дождаться всех впереди стоящих людей. Новый человек встает в конец и ему для выхода из магазина нужно, чтобы кассир обслужил всех людей кроме него самого.

# Постановка задачи

Выполнение работы предполагает решение следующих задач:

1. Разработка и реализация класса стека - TQueue.
2. Пример программы, демонстрирующая работу класса TQueue.
3. Написание набора автоматических тестов с использованием Google C++ Testing Framework и проверка работоспособности методов классов.

# Руководство пользователя

При запуске программы пользователя просят ввести размер очереди.



Рисунок 1 Ввод длины очереди

После чего, очередь заполняется значениями от 0 до введенной длины – 1. Как только он полностью заполнится, программа выведет его на экран. Причем поэлементно.

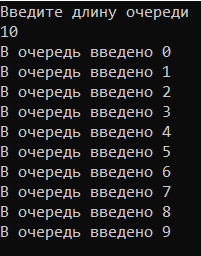


Рисунок 2 Вывод заполненного стека

Далее, программа выводит все элементы очереди. Важно отметить, что выводятся элементы в правильном порядке.

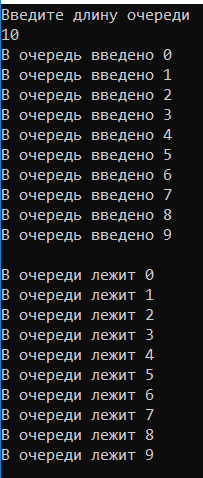


Рисунок 3 Вывод элементов очереди

# Руководство программиста

## Описание структуры программы

Программа состоит из следующих модулей:

* Модуль QueueLib. Статическая библиотека. Включает в себя заголовочный файл Queue.h, в котором описаны методы с реализаций шаблонного класса *TQueue.*
* Модуль QueueTest. Набор тестов для класса TQueue. Включает в себя файл *QueueTest.cpp.* Разработаны они с помощью использования Google C++ Testing Framework.
* Модуль Queue. Пример использования очереди. Включает в себя файл с реализацией *main\_Queue.cpp.*

## Описание структур данных

*Класс TQueue наследуется от класса TStack:*

Поля:

int start; - начало очереди

int count; - кол-во элементов

Конструкторы:

TQueue(int len = 0);

TQueue(TQueue <T> &Que);

Методы:

void Put(T el); - положить в конец

T Get(); - взять первый элемент

bool IsFull(); - проверка на полноту

bool IsEmpty(); - проверка на пустоту

## Описание алгоритмов

Добавление элемента в очередь:

Добавляя новый элемент, размещаем его в свободную ячейку (если она имеется) массива на которую указывает *top* – вершина очереди. Поскольку после выхода из очереди хотя бы одного элемента, массив, в котором она хранится будет выглядеть как на рисунке 4, значение *top* рассчитывается по следующей формуле:

|  |  |
| --- | --- |
|  |  |

Подобная реализация поиска последнего элемента в очереди необходима, чтобы в случае, как на рисунке 4, можно было заполнить все свободные ячейки в массиве, в котором хранится очередь.

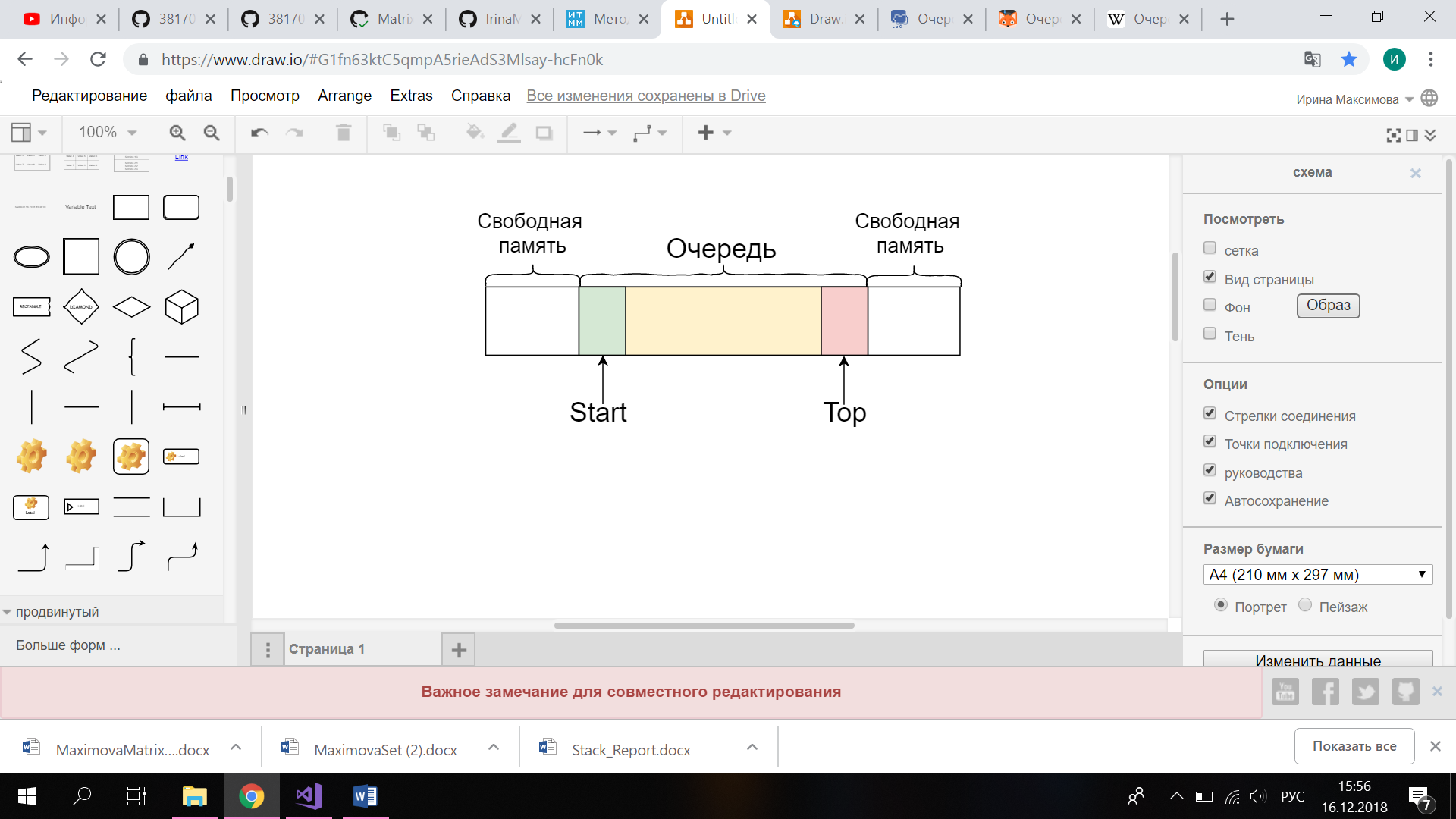


Рисунок 4 Представление очереди в памяти компьютера на n-й операции добавления и изъятия элементов

Удаление элемента из очереди:

При изъятии элемента из очереди (если таковой имеется), возвращается элемент с индексом начала очереди *start*, а сам индекс *start* рассчитывается по формуле:

|  |  |
| --- | --- |
|  |  |

Такая реализация поиска следующего начального элемента очереди необходима, чтобы в случае, изображенном на рисунке 5, можно было забрать все элементы очереди.

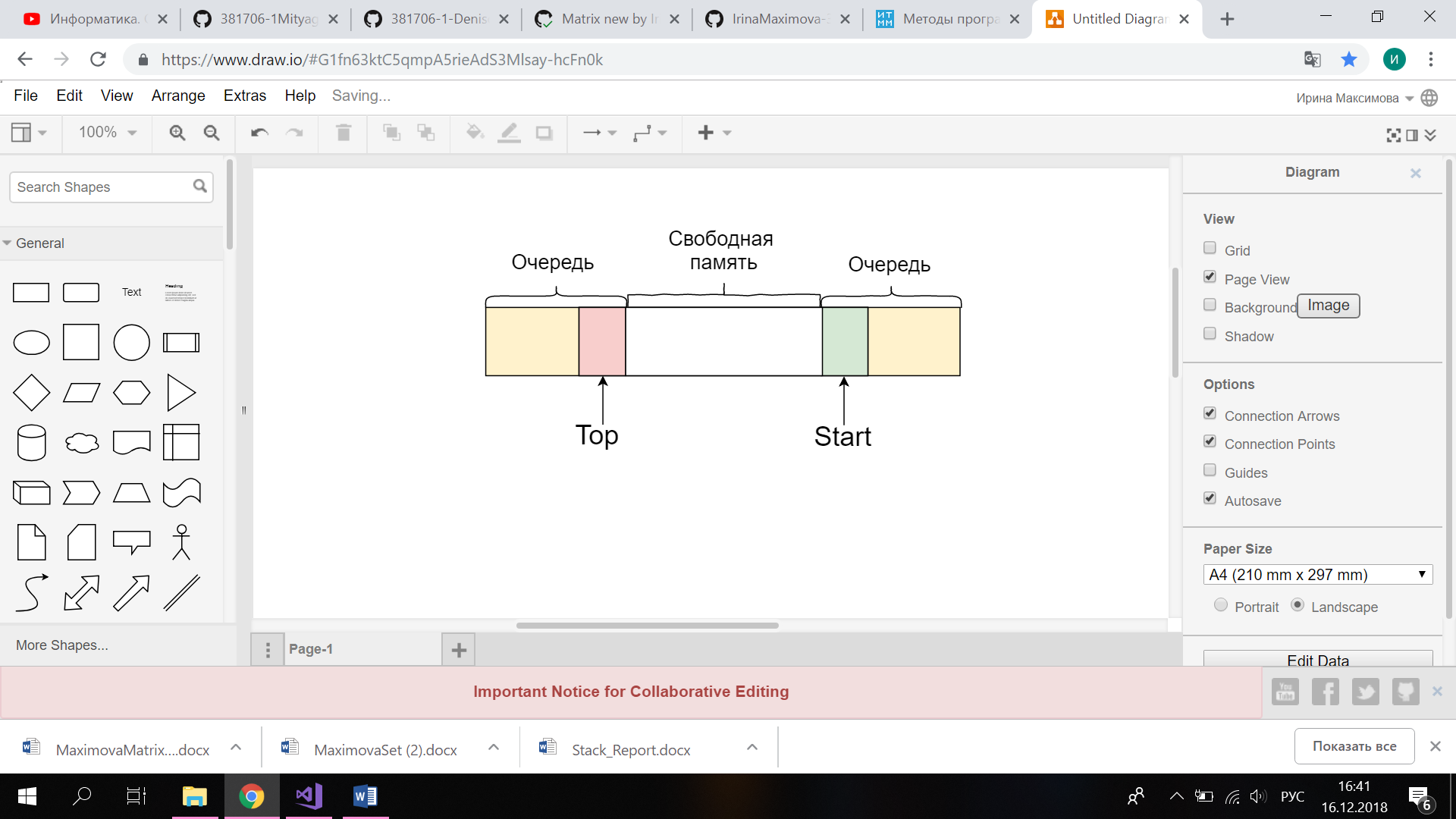


Рисунок 5 Представление очереди в памяти компьютера на n-й операции добавления и изъятия элементов

Само значения изъятого элемента не затирается, т.к. оно будет перезаписано при следующем добавлении другого элемента.

# Заключение

Эта лабораторная работа дала возможность более детально разобраться с устройством работы очереди. В том числе, реализовать один из методов представления стека. В ходе выполнения, был реализован шаблонный класс очереди TQueue, описанный в специально разработанной библиотеке QueueLib. Помимо этого, для проверки работоспособности всех методом, были написаны автоматические тесты, реализованные с использованием Google C++ Testing Framework. Пример реализации очереди для пользователя также написан и успешно работает.

1. **Литература**
2. Гергель В.П. Методические материалы по курсу «Методы программирования 2»: [http://www.itmm.unn.ru/files/2018/10/Primer-1.4.-Struktury-hraneniya-ocheredi.pdf], 2015.
3. Википедия: свободная электронная энциклопедия: на русском языке:

[https://ru.wikipedia.org/wiki/Очередь\_(программирование)]