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 381903-3

Стеценко Данил Михайлович

**Научный руководитель:**

ассистент каф. МОСТ ИИТММ

Лебедев Илья Геннадьевич

Нижний Новгород

2020

Оглавление

[1. Введение. 2](#_Toc532943564)

[2. Постановка задачи. 3](#_Toc532943565)

[3. Руководство пользователя. 4](#_Toc532943566)

[4. Руководство программиста. 5](#_Toc532943567)

[4.1. Описание структуры программы. 5](#_Toc532943568)

[4.2. Описание структур данных. 5](#_Toc532943569)

[4.3. Описание алгоритмов. 6](#_Toc532943570)

[5. Заключение. 7](#_Toc532943583)

[6. Литература. 8](#_Toc532943584)

**7.** Приложения………………………………………………………………………….9

# Введение.

**Очередь** — абстрактный тип данных с дисциплиной доступа к элементам «первый пришёл — первый вышел». Добавление элемента возможно лишь в конец очереди, выборка — только из начала очереди, при этом выбранный элемент из очереди удаляется.

Очередь в программировании в основном применяется, когда нужно провести какие-то действия с элементами последовательно.

# Постановка задачи.

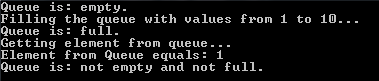
В данной лабораторной работе нужно разработать эффективную библиотеку для хранения и работы с такой структурой данных, как очередь.

Для этого нам нужно:

* Описать и реализовать класс **TQueue**.
* Протестировать класс **TQueue** с помощью Google Test.
* Продемонстрировать работу класса **TQueue**.

# Руководство пользователя.

Пример работы класса **TQueue**:

****

*Рис 1. Пример работы программы*

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.cpp** с реализацией примера использования класса **TQueue.**
* **QueueLib –** содержит в себе файл **Queue.h**, в котором описан и реализован класс **TQueue**.
* **Test** – содержит в себе файл **test\_queue.cpp**, в котором находится набор тестов, для проверки работоспособности класса **TQueue**.

# 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.

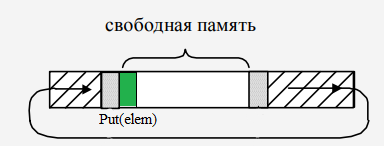
Далее идут методы класса со спецификатором доступа **public**:

* **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.

# Описание алгоритмов.

**Добавление элемента в очередь.**

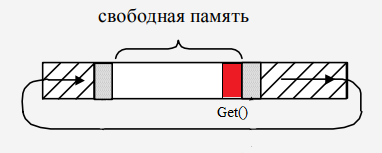
Добавляя элемент в очередь, мы помещаем его начало очереди, на которое указывает поле **start**. После добавления элемента в очередь, значение поля **start** увеличивается на 1 следующим образом: . Это нужно для того, чтобы мы могли заполнить всю свободную память, как показано на *Рис. 2*. Количество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

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.

# Приложения

|  |
| --- |
| **Queue.h** |
| #pragma once  #ifndef \_QUEUE\_  #define \_QUEUE\_  #include <iostream>  #include <fstream>  using namespace std;  template <class T>  class TQueue  {  protected:  int length;  T\* x;  int ind;  int end;  int count;  public:  TQueue(int size = 0);  TQueue(TQueue<T>& \_v);  ~TQueue();  TQueue<T>& operator =(TQueue<T>& \_v);  void Push(T d);  T Get();  int Length();  int IsEmpty(void) const;  inline int min\_elem();  inline int max\_elem();  template <class T1>  friend ostream& operator<< (ostream& ostr, const TQueue<T1>& A);  template <class T1>  friend istream& operator >> (istream& istr, TQueue<T1>& A);  };  template <class T1>  ostream& operator<< (ostream& ostr, const TQueue<T1>& A)  {  for (int i = A.ind; i < A.end; i++)  {  ostr << A.x[i] << endl;  }  return ostr;  }  template <class T1>  istream& operator >> (istream& istr, TQueue<T1>& A)  {  int count;  istr >> count;  for (int i = 0; i < A.count; i++) {  T1 d;  istr >> d;  A.Push(d);  }  return istr;  }  template<class T>  inline TQueue<T>::TQueue(int size)  {  if (size > 0)  {  this->length = size;  x = new T[length];  for (int i = 0; i < length; i++)  {  x[i] = 0;  }  this->ind = 0;  this->end = 0;  this->count = 0;  }  else  {  throw logic\_error("ERROR");  }  }  template <class T>  TQueue<T>::TQueue(TQueue<T>& \_v)  {  length = \_v.length;  ind = \_v.ind;  end = \_v.end;  count = \_v.count;  x = new T[length];  for (int i = 0; i < length; i = i + 1)  {  x[i] = \_v.x[i];  }  }  template <class T>  TQueue<T>::~TQueue()  {  length = 0;  if (x != 0)  delete[] x;  x = 0;  }  template <class T>  TQueue<T>& TQueue<T>::operator =(TQueue<T>& \_v)  {  if (this == &\_v)  return \*this;  this->length = \_v.length;  if (x != NULL)  {  delete[] x;  }  this->x = new T[length];  for (int i = 0; i < length; i++)  x[i] = \_v.x[i];  this->ind = \_v.ind;  this->end = \_v.end;  this->count = \_v.count;  return \*this;  }  template<class T>  inline void TQueue<T>::Push(T d)  {  if (count >= length)  {  ind++;  count--;  }  x[end] = d;  end = (end + 1) % length;  count++;  }  template<class T>  inline T TQueue<T>::Get()  {  if (count == 0)  {  throw logic\_error("ERROR");  }  T d = x[ind];  ind = (ind + 1) % length;  return d;  }  template <class T>  int TQueue<T>::Length()  {  return length;  }  template<class T>  inline int TQueue<T>::IsEmpty(void) const  {  return count == 0;  }  template<class T>  inline int TQueue<T>::min\_elem()  {  int res = x[0];  for (int i = 1; i < length; i++)  {  if (x[i] < res)  {  res = x[i];  }  }  return res;  }  template<class T>  inline int TQueue<T>::max\_elem()  {  int res = x[0];  for (int i = 1; i < length; i++)  {  if (x[i] > res)  {  res = x[i];  }  }  return res;  }  #endif |

|  |
| --- |
| **Test\_TQueue** |
| #include "Queue.h"  #include "Queue.cpp"  #include "../gtest/gtest.h"  TEST(TQueue, can\_create\_queue\_with\_positive\_length)  {  ASSERT\_NO\_THROW(TQueue<int> qe(3));  }  TEST(TQueue, cant\_create\_queue\_with\_null)  {  ASSERT\_ANY\_THROW(TQueue<int> qe(0));  }  TEST(TQueue, create\_empty\_queue)  {  TQueue<int> qe(3);  EXPECT\_EQ(true, qe.IsEmpty());  }  TEST(TQueue, can\_push\_if\_ok)  {  TQueue<int> qe(3);  ASSERT\_NO\_THROW(qe.Push(3));  }  TEST(TQueue, can\_get\_elem)  {  TQueue<int> qe(2);  qe.Push(4);  EXPECT\_EQ(4, qe.Get());  }  TEST(TQueue, cant\_get\_if\_empty)  {  TQueue<int> qe(1);  ASSERT\_ANY\_THROW(qe.Get());  }  TEST(TQueue, can\_copy\_queue)  {  TQueue<int> qe1(2);  qe1.Push(1);  ASSERT\_NO\_THROW(TQueue<int> qe2(qe1));  }  TEST(TQueue, can\_assing\_queue)  {  TQueue<int> qe1(2);  qe1.Push(1);  TQueue<int> qe2(2);  qe2 = qe1;  EXPECT\_EQ(1, qe2.Get());  }  TEST(TQueue, test\_ring\_queue)  {  TQueue<int> qe(2);  qe.Push(1);  qe.Push(2);  qe.Push(3);  EXPECT\_EQ(2, qe.Get());  }  TEST(TQueue, can\_get\_Lenght)  {  TQueue<int> qe(2);  EXPECT\_EQ(2, qe.Length());  }  TEST(TQueue, can\_get\_MIN\_elem)  {  TQueue<int> qe(3);  qe.Push(1);  qe.Push(14);  qe.Push(7);  EXPECT\_EQ(1, qe.min\_elem());  }  TEST(TQueue, can\_get\_MAX\_elem)  {  TQueue<int> qe(3);  qe.Push(1);  qe.Push(14);  qe.Push(7);  EXPECT\_EQ(14, qe.max\_elem());  } |