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

Жбанова Надежда Сергеевна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Руководитель:**

Ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2018

Содержание

[1.Введение 3](#_Toc534316445)

[2. Цели и задачи 4](#_Toc534316446)

[2.1. Используемые инструменты 4](#_Toc534316447)

[3. Руководство пользователя 6](#_Toc534316448)

[4. Руководство программиста 12](#_Toc534316449)

[4.1. Описание структуры программы 12](#_Toc534316450)

[4.2. Описание функций и процедур, их алгоритмов 12](#_Toc534316451)

[5. Эксперименты 15](#_Toc534316452)

[6. Заключение 16](#_Toc534316453)

[7. Литература 17](#_Toc534316454)

[8. Приложения 18](#_Toc534316455)

[8.1. Приложение 1:Класс TExсeption 18](#_Toc534316456)

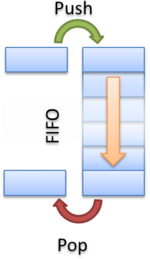
[8.2. Приложение 2:Класс TStack 18](#_Toc534316457)

[8.3. Приложение 3:Класс TQueue 21](#_Toc534316458)

[8.4. Приложение 4:Код программы тестирования и экспериментов 23](#_Toc534316459)

[8.5. Приложение 5:Тесты для классов 25](#_Toc534316460)

# Введение

[](http://neerc.ifmo.ru/wiki/index.php?title=%D0%A4%D0%B0%D0%B9%D0%BB:Fifo_new.png)

Очередь (англ. queue) — это структура данных, добавление и удаление элементов в которой происходит путём операций push и pop соответственно. Притом первым из очереди удаляется элемент, который был помещен туда первым, то есть в очереди реализуется принцип «первым вошел — первым вышел» (англ. first-in, first-out — FIFO). У очереди имеется голова (англ. head) и хвост (англ. tail). Когда элемент ставится в очередь, он занимает место в её хвосте. Из очереди всегда выводится элемент, который находится в ее голове. Очередь поддерживает следующие операции:

* Empty — проверка очереди на наличие в ней элементов,
* Push (запись в очередь) — операция вставки нового элемента,
* Pop (снятие с очереди) — операция удаления нового элемента,
* Size — операция получения количества элементов в очереди.

# 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) и фрэймворк для разработки автоматических тестов [Google Test](https://github.com/google/googletest).

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.

## 2.1. Используемые инструменты

* Система контроля версий [Git](https://git-scm.com/book/ru/v2). Рекомендуется использовать один из следующих клиентов на выбор студента:
  + [Git](https://git-scm.com/downloads)
  + [GitHub Desktop](https://desktop.github.com/)
* Фреймворк для написания автоматических тестов [Google Test](https://github.com/google/googletest).
* Среда разработки Microsoft Visual Studio (2008 или старше).
* Опционально. Утилита [CMake](http://www.cmake.org/) для генерации проектов по сборке исходных кодов. Может быть использована для генерации решения для среды разработки, отличной от Microsoft Visual Studio 2008 или 2010.

# 3. Руководство пользователя

Запускаем программу из файла queue\_sample.cpp (Рис.1):

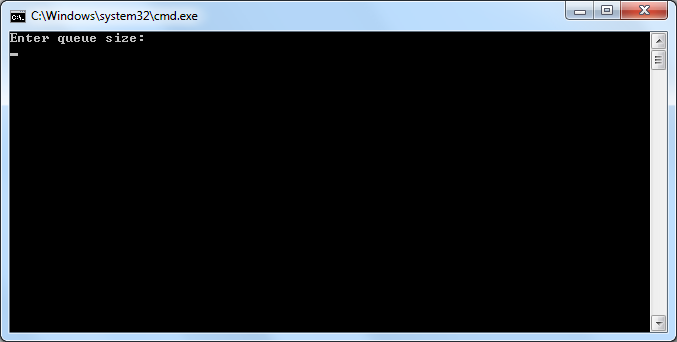


Рис.1.Ввод размера очереди.

Пользователю предлагается ввести максимальный размер создаваемого очереди. Если ввести отрицательное число, бросится исключение (Рис.2):

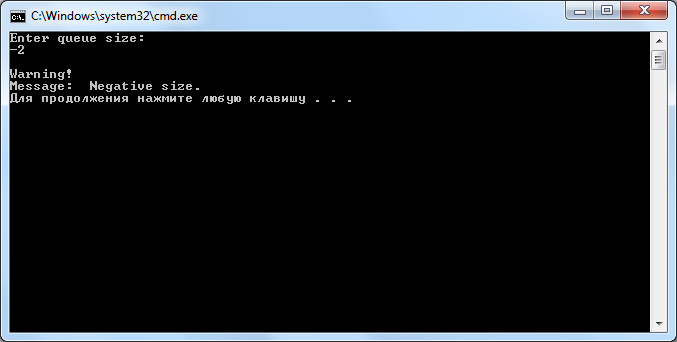


Рис.2. Ошибка.

При положительном значении создастся очередь заданного размера для хранения значений любого типа. Затем пользователю предлагается заполнить все её ячейки. (Рис.3):

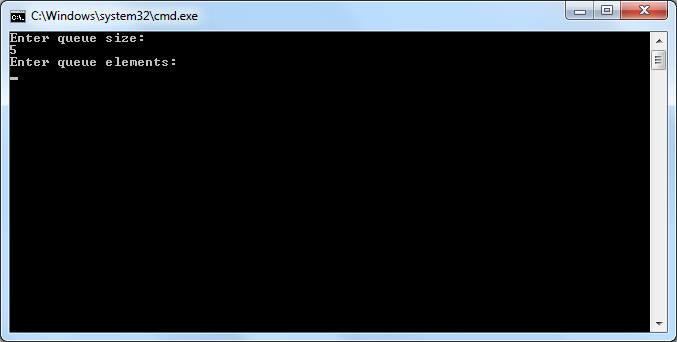


Рис.3. Заполнение очереди.

После заполнения происходит печать очереди на консоль, и предоставляется выбор операций: добавление элемента в очередь, удаление элемента, проверка на пустоту и полноту, копирование соответственно. (Рис.4):

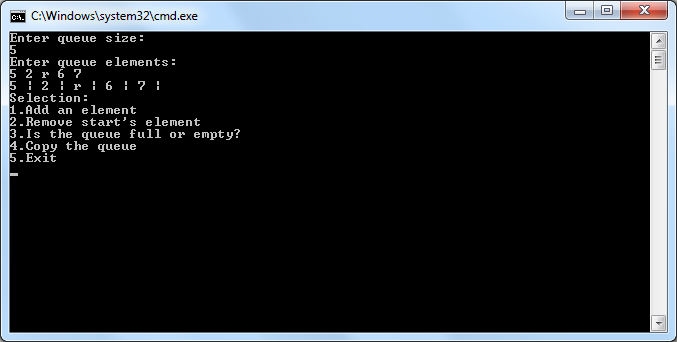


Рис.4. Меню операций.

Выбрав ‘Exit’, можно выйти из программы. (Рис.5):

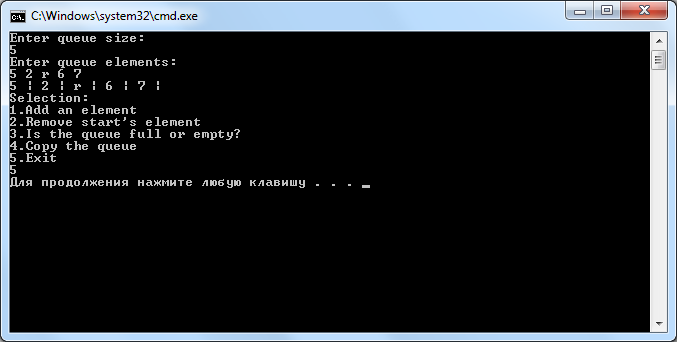


Рис.5.Выход из программы.

Если удалять из пустой очереди или добавлять в полную, программа выведет ошибку и не выполнит операцию. (Рис.6, 7):

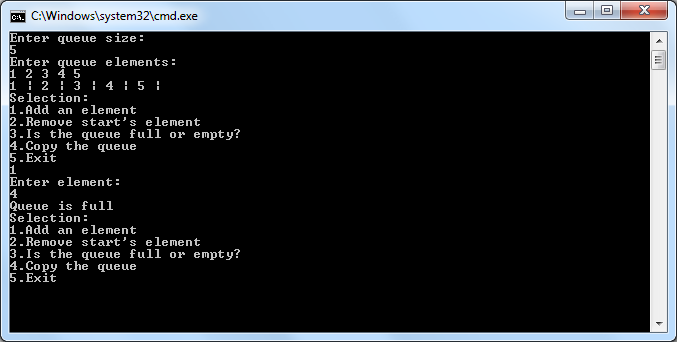


Рис.6. Ошибка – очередь заполнена.

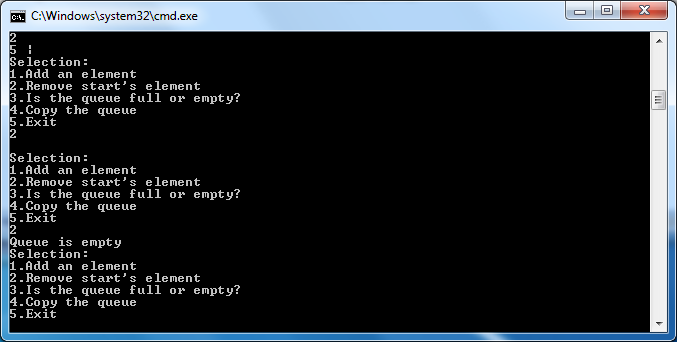


Рис.7.Ошибка – очередь пуста.

Чтобы удалить элемент – нужно нажать “2”. При удалении элемента из очереди – всегда берется ее первый элемент. (Рис.8):

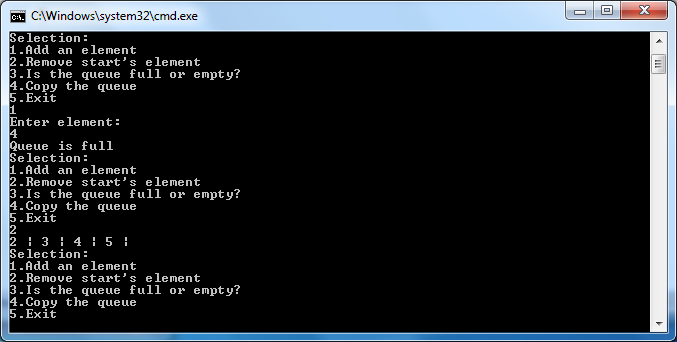


Рис.8.Удаление элемента.

Чтобы добавить элемент в очередь нужно нажать “1”. Далее требуется ввести добавляемый элемент, который добавится в конец очереди. (Рис.9):

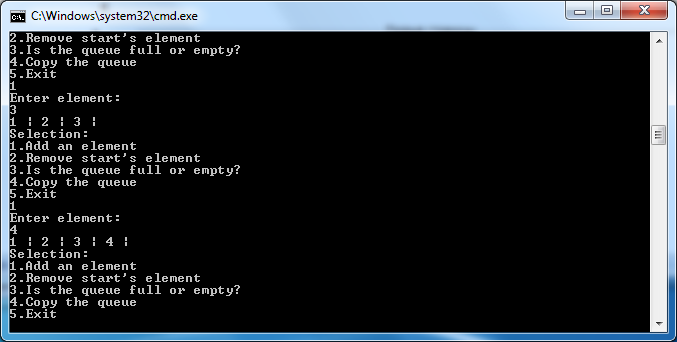


Рис.9.Добавление элемента.

Чтобы проверить очередь на полноту или пустоту, нужно выбрать “3”. Программа выведет “Full queue”, если очередь полна, и “Empty queue”, если очередь пуста.  
(Рис.10, 11):

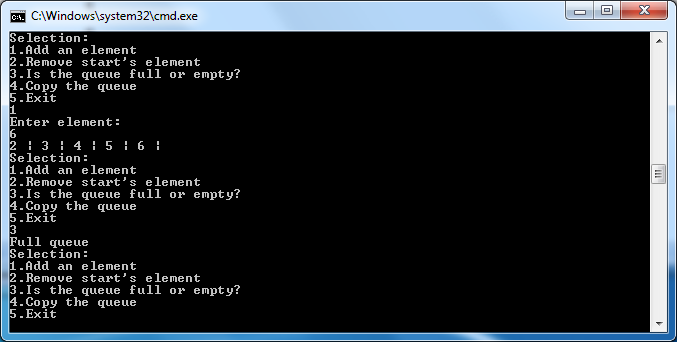


Рис.10.Очередь полная.

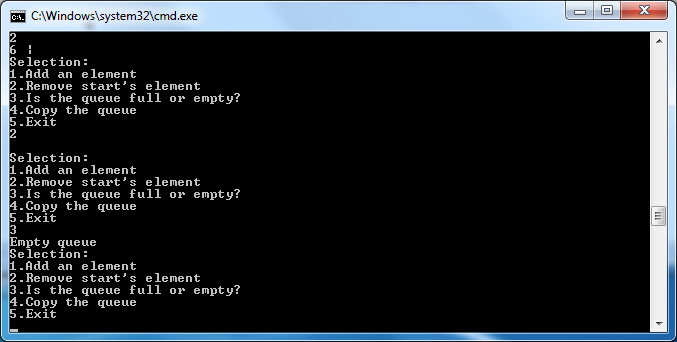


Рис.11.Очередь пустая.

Чтобы скопировать очередь нужно выбрать “4”. (Рис.12):

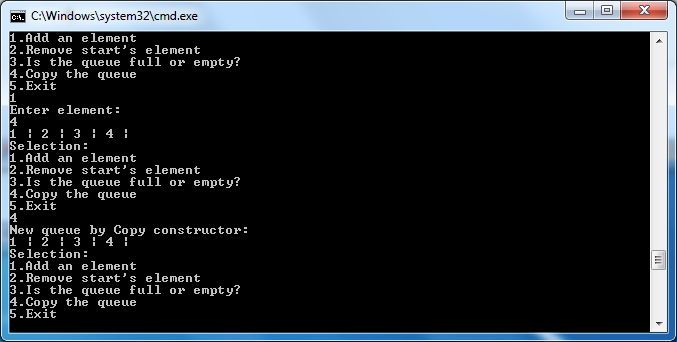


Рис.12.Копирование очереди.

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. (Рис.5) По приведенным данным можно сделать вывод, что практическая сложность выполнения алгоритма равна теоретической.

Рис.5. График зависимости времени выполнения операции присвоения от размера очереди.

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) и фрэймворк для разработки автоматических тестов [Google Test](https://github.com/google/googletest).

Созданный шаблонный класс был протестированы с использованием Google Tests, а также были проведены эксперименты для сравнения теоретической и практической сложности выполнения операций на методе класса.

# 7. Литература

1. Лабораторный практикум: Учебно-методическое пособие / Мееров И.Б. [и др.] – Нижний Новгород: Нижегородский госуниверситет , 2017. – 105с.
2. Тестирование с использованием Google Test

(http://neerc.ifmo.ru/wiki/index.php?title=Тестирование\_с\_использованием\_Google\_Test#.D0.A4.D1.83.D0.BD.D0.BA.D1.86.D0.B8.D1.8F\_main.28.29)

1. Касперски К. Техника оптимизации программ. Эффективное использование памяти. — СПб.: БХВ-Петербург, 2003. — 464 с.: ил.
2. http://neerc.ifmo.ru/wiki/index.php?title=Очередь

# 8. Приложения

## 8.1. Приложение 1:Класс TExсeption

|  |
| --- |
| **exception.h** |
| #pragma once  #include <iostream>  #include <string>  class TException  {  private:  std::string str;  public:  TException(std::string \_str);  void Show();  };  TException::TException(std::string \_str) : str(\_str) {}  void TException::Show()  {  std::cout << "\nWarning! \nMessage: " << str << std::endl;  } |

## 8.2. Приложение 2:Класс TStack

|  |
| --- |
| **tstack.h** |
| #pragma once  #include <iostream>  #include "exception.h"  using namespace std;  template <typename StackType>  class TStack  {  protected:  int size;  int top;  StackType\* memory;  public:  int GetSize() { return size; }  TStack(int n = 0);  TStack(TStack<StackType> &S);  TStack& operator=(const TStack<StackType>& stack);  ~TStack() { delete[] memory; }  StackType Get();  void Put(StackType A);  bool IsFull();  bool IsEmpty();  void PrintStack();  bool operator!=(const TStack<StackType>& stack) const;  bool operator==(const TStack<StackType>& stack) const;  };  //----------------------------------------------------------------------  template <class StackType>  TStack <StackType> ::TStack(int n)  {  if ( n < 0 )  throw TException(" Negative size.");  else if ( n == 0 )  {  size = 0;  top = 0;  memory = NULL;  }  else  {  size = n;  top = 0;  memory = new StackType[size];  for ( int i = 0; i < size; i++ )  memory[i] = 0;  }  }  //----------------------------------------------------------------------  template <class StackType>  TStack <StackType> ::TStack(TStack <StackType> &S)  {  size = S.size;  top = S.top;  if ( size == 0 )  memory = NULL;  else  {  memory = new StackType[size];  for ( int i = 0; i < size; i++ )  memory[i] = S.memory[i];  }  }  //----------------------------------------------------------------------  template <class StackType>  TStack<StackType>& TStack<StackType> ::operator=(const TStack<StackType>& stack)  {  if ( this != &stack )  {  top = stack.top;  if ( size != stack.size )  {  size = stack.size;  delete[] memory;  memory = new StackType[size];  }  for ( int i = 0; i < size; i++ )  {  memory[i] = stack.memory[i];  }  }  return \*this;  }  //----------------------------------------------------------------------  template <class StackType>  void TStack<StackType> ::Put(StackType A)  {  if ( IsFull() )  {  throw TException("Stack is full");  }  else  {  memory[top] = A;  top++;  }  }  //----------------------------------------------------------------------  template <class StackType>  StackType TStack<StackType> ::Get()  {  if ( IsEmpty() )  throw TException("Stack is empty");  else  {  top--;  return memory[top];  }  }  //----------------------------------------------------------------------  template <class StackType>  bool TStack<StackType> ::IsFull()  {  //ternary operator  return top >= size ? true : false;  }  //----------------------------------------------------------------------  template <class StackType>  bool TStack<StackType> ::IsEmpty()  {  return !top;  }  //----------------------------------------------------------------------  template <class StackType>  bool TStack<StackType> ::operator==(const TStack<StackType>& stack) const  {  if ( top != stack.top || size != stack.size )  return false;  for ( int i = 0; i < top; i++ )  {  if ( memory[i] != stack.memory[i] )  return false;  }  return true;  }  //----------------------------------------------------------------------  template <class StackType>  bool TStack<StackType> ::operator!=(const TStack<StackType>& stack) const  {  return !( \*this == stack );  }  //----------------------------------------------------------------------  template <class StackType>  void TStack<StackType>:: PrintStack()  {  for ( int i = top-1; i >= 0; i-- )  cout << "\t|" << memory[i] << "|" << endl;  } |

## 8.3. Приложение 3:Класс TQueue

|  |
| --- |
| **tqueue.h** |
| #pragma once  #include "tstack.h"  template <typename QueueType>  class TQueue : public TStack <QueueType>  {  protected:  int start;  int count;  public:  TQueue(int n = 0);  TQueue(TQueue <QueueType> &Q);  TQueue& operator=(const TQueue<QueueType>& queue);  QueueType Get();  void Put(QueueType Q);  bool IsFull();  bool IsEmpty();  void PrintQueue();  bool operator!=(const TQueue<QueueType>& queue) const;  bool operator==(const TQueue<QueueType>& queue) const;  };  //-----------------------------------------------------------------  template <typename QueueType>  TQueue<QueueType>::TQueue(int n) : TStack<QueueType>(n)  {  start = 0;  count = 0;  }  //-----------------------------------------------------------------  template <typename QueueType>  TQueue<QueueType>::TQueue(TQueue<QueueType> &Q) : TStack<QueueType>(Q)  {  start = Q.start;  count = Q.count;  }  //-----------------------------------------------------------------  template <typename QueueType>  TQueue<QueueType>& TQueue<QueueType> ::operator=(const TQueue<QueueType>& queue)  {  if ( this == &queue ) return \*this;  start = queue.start;  count = queue.count;  TStack<QueueType>::top = queue.top;  if ( TStack<QueueType>::size != queue.size )  {  TStack<QueueType>::size = queue.size;  delete[] TStack<QueueType>::memory;  TStack<QueueType>::memory = new QueueType[TStack<QueueType>::size];  }  for ( int i = start; i <count+start; i++ )  {  TStack<QueueType>::memory[i % TStack<QueueType>::size] = queue.memory[i % TStack<QueueType>::size];  }  return \*this;  }  //-----------------------------------------------------------------  template <typename QueueType>  void TQueue<QueueType>::Put(QueueType el)  {  if (IsFull())  throw TException("Queue is full.");  else  {  TStack<QueueType>::memory[TStack<QueueType>::top] = el;  TStack<QueueType>::top = ++TStack<QueueType>::top % TStack<QueueType>::size;  count++;  }  }  //-----------------------------------------------------------------  template <typename QueueType>  QueueType TQueue<QueueType>::Get()  {  if (IsEmpty())  throw TException("Queue is empty.");  else  {  QueueType buf = TStack<QueueType>::memory[start];  start = ++start % TStack<QueueType>::size;  count--;  return buf;  }  }  //-----------------------------------------------------------------  template <typename QueueType>  bool TQueue<QueueType>::IsFull()  {  return count == TStack<QueueType>::size ? true: false;  }  //-----------------------------------------------------------------  template <typename QueueType>  bool TQueue<QueueType>::IsEmpty()  {  return count == 0 ? true: false;  }  //-----------------------------------------------------------------  template <typename QueueType>  bool TQueue<QueueType> ::operator==(const TQueue<QueueType>& queue) const  {  if ( TStack<QueueType>::top != queue.top || TStack<QueueType>::size != queue.size ||  count != queue.count || start !=queue.start )  return false;  for ( int i = start; i < count + start; i++ )  {  if ( TStack<QueueType>::memory[i % TStack<QueueType>::size] != queue.memory[i % TStack<QueueType>::size] )  return false;  }  return true;  }  //-----------------------------------------------------------------  template <typename QueueType>  bool TQueue<QueueType> ::operator!=(const TQueue<QueueType>& queue) const  {  return !( \*this == queue );  }  template <typename QueueType>  void TQueue<QueueType>:: PrintQueue()  {  for ( int i = start; i <count+start; i++ )  {  cout<<TStack<QueueType>::memory[i % TStack<QueueType>::size]<<" | ";  }  cout << endl;  } |

## 8.4. Приложение 4:Код программы тестирования и экспериментов

|  |
| --- |
| **queue\_sample.cpp** |
| #include <iostream>  #include <locale.h>  #include "tqueue.h"  using namespace std;  int main()  {  try  {  int n;  int input;  cout << "Enter queue size:"<< endl;  cin >> n;  TQueue<char> s1(n);  char symbol;  cout << "Enter queue elements:"<< endl;  for ( int i = 0; i < n; i++ )  {  cin >> symbol;  s1.Put(symbol);  }  s1.PrintQueue();  for ( int i = 0; i <= 100; i++)  {  cout << "Selection:"<< endl;  cout << "1.Add an element"<< endl;  cout << "2.Remove start's element"<< endl;  cout << "3.Is the queue full or empty?"<< endl;  cout << "4.Copy the queue"<< endl;  cout << "5.Exit"<< endl;  cin >> input;  switch(input)  {  case 1:  {  char elem;  cout << "Enter element:"<< endl;  cin >> elem;  if (!s1.IsFull())  {  s1.Put(elem);  s1.PrintQueue();  }  else  {  cout<<"Queue is full"<<endl;  }  break;  }  case 2:  {  if (!s1.IsEmpty())  {  s1.Get();  s1.PrintQueue();  }  else  {  cout<<"Queue is empty"<<endl;  }  break;  }  case 3:  {  if ( s1.IsFull() )  cout << "Full queue"<< endl;  else if (s1.IsEmpty())  cout << "Empty queue"<< endl;  break;  }  case 4:  {  cout << "New queue by Copy constructor:"<< endl;  TQueue<char> s2(s1);  s2.PrintQueue();  break;  }  case 5:  {  return 0;  }  default:  cout << "Invalid Input!"<< endl;  }  }  }  catch (TException exp)  {  exp.Show();  }  return 0;  } |

|  |
| --- |
| **sample\_performance\_check.cpp** |
| #include <iostream>  #include "tqueue.h"  #include "time.h"  using namespace std;  int main()  {  unsigned max\_count = 0;  cout << "Enter number of checks - ";  cin >> max\_count;  clock\_t time;  clock\_t average\_time;  for ( unsigned size = 10; size < 1000000; size \*= 10 )  {  TQueue<int> s1(size);  TQueue<int> s2(size);  for ( unsigned i = 0; i < size; i++ )  {  s1.Put(i);  }  average\_time = 0;  for (unsigned count = 0; count < max\_count; count++)  {  time = clock();  s1 = s2;  average\_time += clock() - time;  }  average\_time /= max\_count;  cout<< "Time is: " <<average\_time<<endl;  }  return 0;  } |

## 8.5. Приложение 5:Тесты для классов

|  |
| --- |
| **test\_main.cpp** |
| #include <gtest.h>  int main(int argc, char \*\*argv)  {  ::testing::InitGoogleTest(&argc, argv);  return RUN\_ALL\_TESTS();  } |

|  |
| --- |
| **test\_tstack.cpp** |
| #include "tstack.h"  #include <gtest.h>  TEST(TStack, can\_create\_stack)  {  ASSERT\_NO\_THROW(TStack<int> s1);  }  TEST(TStack, new\_stack\_is\_empty)  {  TStack<int> s1(3);  EXPECT\_EQ(true, s1.IsEmpty());  }  TEST(TStack, can\_create\_stack\_with\_positive\_size)  {  ASSERT\_NO\_THROW(TStack<int> s1(5));  }  TEST(TStack, throw\_when\_create\_stack\_with\_negative\_size)  {  ASSERT\_ANY\_THROW(TStack<int> s1(-5));  }  TEST(TStack, can\_create\_copied\_stack)  {  TStack<int> s1(5);  ASSERT\_NO\_THROW(TStack<int> s2(s1));  }  TEST(TStack, can\_put\_elem\_in\_stack)  {  TStack<int> s1(3);  ASSERT\_NO\_THROW(s1.Put(1));  }  TEST(TStack, stack\_with\_elem\_isnt\_empty)  {  TStack<int> s1(3);  s1.Put(1);  EXPECT\_FALSE(s1.IsEmpty());  }  TEST(TStack, cant\_put\_in\_full\_stack)  {  TStack<int> s1(3);  for ( int i = 0; i < 3; i++ )  {  s1.Put(i);  }  ASSERT\_ANY\_THROW(s1.Put(3));  }  TEST(TStack, can\_get\_elem\_from\_stack)  {  TStack<int> s1(1);  s1.Put(1);  int res = s1.Get();  EXPECT\_EQ(1, res);  }  TEST(TStack, cant\_get\_from\_empty\_stack)  {  TStack<int> s1(3);  ASSERT\_ANY\_THROW(s1.Get());  }  TEST(TStack, get\_returns\_last\_put\_elem)  {  TStack<int> s1(3);  int elem1 = 1, elem2 = 2, res;  s1.Put(elem1); s1.Put(elem2);  res = s1.Get();  EXPECT\_TRUE((res = elem2) && (res != elem1));  }  TEST(TStack, not\_empty\_assignment)  {  TStack<int> s1(10);  TStack<int> s2(10);  s1.Put(10);  s1.Put(20);  s1.Put(40);  s2 = s1;  EXPECT\_EQ(40, s2.Get());  EXPECT\_EQ(s2.Get(), 20);  EXPECT\_EQ(s2.Get(), 10);  }  TEST(TStack, can\_use\_isempty\_correctly)  {  TStack<int> s1(5);  ASSERT\_TRUE(s1.IsEmpty());  }  TEST(TStack, can\_use\_isempty\_incorrectly)  {  TStack<int> s1(2);  s1.Put(1);  ASSERT\_FALSE(s1.IsEmpty());  }  TEST(TStack, can\_use\_isfull\_correctly)  {  TStack<int> s1(1);  s1.Put(10);  ASSERT\_TRUE(s1.IsFull());  }  TEST(TStack, can\_use\_isfull\_incorrectly)  {  TStack<int> s1(5);  ASSERT\_FALSE(s1.IsFull());  }  TEST(TStack, compare\_equal\_stacks\_return\_true)  {  TStack<int> s1(2), s2(2);  s1.Put(10);  s1.Put(20);  s2 = s1;  EXPECT\_TRUE(s2==s1);  }  TEST(TStack, compare\_stack\_with\_itself\_return\_true)  {  TStack<int> s1(5);  EXPECT\_TRUE(s1==s1);  }  TEST(TStack, stacks\_with\_different\_size\_are\_not\_equal)  {  TStack<int> s1(5), s2(4);  EXPECT\_FALSE(s1==s2);  }  TEST(TStack, compare\_non\_equal\_stacks\_return\_false)  {  TStack<int> s1(2), s2(2);  s1.Put(10);  s1.Put(10);  s2.Put(10);  s2.Put(20);  EXPECT\_FALSE(s1==s2);  } |

|  |
| --- |
| **test\_tqueue.cpp** |
| #include "tqueue.h"  #include <gtest.h>  TEST(TQueue, can\_create\_queue)  {  ASSERT\_NO\_THROW(TQueue<int> s1);  }  TEST(TQueue, new\_queue\_is\_empty)  {  TQueue<int> s1(3);  EXPECT\_EQ(true, s1.IsEmpty());  }  TEST(TQueue, can\_create\_queue\_with\_positive\_size)  {  ASSERT\_NO\_THROW(TQueue<int> s1(5));  }  TEST(TQueue, throw\_when\_create\_queue\_with\_negative\_size)  {  ASSERT\_ANY\_THROW(TQueue<int> s1(-5));  }  TEST(TQueue, can\_create\_copied\_queue)  {  TQueue<int> s1(5);  ASSERT\_NO\_THROW(TQueue<int> s2(s1));  }  TEST(TQueue, can\_put\_elem\_in\_queue)  {  TQueue<int> s1(3);  ASSERT\_NO\_THROW(s1.Put(1));  }  TEST(TQueue, queue\_with\_elem\_isnt\_empty)  {  TQueue<int> s1(3);  s1.Put(1);  EXPECT\_FALSE(s1.IsEmpty());  }  TEST(TQueue, cant\_put\_in\_full\_queue)  {  TQueue<int> s1(3);  for ( int i = 0; i < 3; i++ )  {  s1.Put(i);  }  ASSERT\_ANY\_THROW(s1.Put(3));  }  TEST(TQueue, can\_get\_elem\_from\_queue)  {  TQueue<int> s1(1);  s1.Put(1);  int res = s1.Get();  EXPECT\_EQ(1, res);  }  TEST(TQueue, cant\_get\_from\_empty\_queue)  {  TQueue<int> s1(3);  ASSERT\_ANY\_THROW(s1.Get());  }  TEST(TQueue, get\_returns\_first\_put\_elem)  {  TQueue<int> s1(3);  int elem1 = 1, elem2 = 2, res;  s1.Put(elem1); s1.Put(elem2);  res = s1.Get();  EXPECT\_TRUE((res = elem1) && (res != elem2));  }  TEST(TQueue, not\_empty\_assignment)  {  TQueue<int> s1(10);  TQueue<int> s2(10);  s1.Put(10);  s1.Put(20);  s1.Put(40);  s2 = s1;  EXPECT\_EQ(s2.Get(), 10);  EXPECT\_EQ(s2.Get(), 20);  EXPECT\_EQ(s2.Get(), 40);  }  TEST(TQueue, can\_use\_isempty\_correctly)  {  TQueue<int> s1(5);  ASSERT\_TRUE(s1.IsEmpty());  }  TEST(TQueue, can\_use\_isempty\_incorrectly)  {  TQueue<int> s1(2);  s1.Put(1);  ASSERT\_FALSE(s1.IsEmpty());  }  TEST(TQueue, can\_use\_isfull\_correctly)  {  TQueue<int> s1(1);  s1.Put(10);  ASSERT\_TRUE(s1.IsFull());  }  TEST(TQueue, can\_use\_isfull\_incorrectly)  {  TQueue<int> s1(5);  ASSERT\_FALSE(s1.IsFull());  }  TEST(TQueue, compare\_equal\_queues\_return\_true)  {  TQueue<int> s1(2), s2(2);  s1.Put(10);  s1.Put(20);  s2 = s1;  EXPECT\_TRUE(s2==s1);  }  TEST(TQueue, compare\_queue\_with\_itself\_return\_true)  {  TQueue<int> s1(5);  EXPECT\_TRUE(s1==s1);  }  TEST(TQueue, queues\_with\_different\_size\_are\_not\_equal)  {  TQueue<int> s1(5), s2(4);  EXPECT\_FALSE(s1==s2);  }  TEST(TQueue, compare\_non\_equal\_queues\_return\_false)  {  TQueue<int> s1(2), s2(2);  s1.Put(10);  s1.Put(10);  s2.Put(10);  s2.Put(20);  EXPECT\_FALSE(s1==s2);  } |