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**

**ЛАБОРАТОРНАЯ РАБОТА №7**

**«Список на массивах»**

**Выполнил:** студент группы 381903-3

Пасека Иван Евгеньевич

**Руководитель:**

Ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

Нижний Новгород

2020

Содержание

[1.Введение 3](#_Toc536814977)

[2. Цели и задачи 5](#_Toc536814978)

[2.1. Используемые инструменты 5](#_Toc536814979)

[3. Руководство пользователя 7](#_Toc536814980)

[4. Руководство программиста 13](#_Toc536814981)

[4.1. Описание структуры программы 13](#_Toc536814982)

[4.2. Описание функций и процедур, их алгоритмов 13](#_Toc536814983)

[5. Эксперименты 15](#_Toc536814984)

[6. Заключение 16](#_Toc536814985)

[7. Литература 17](#_Toc536814986)

[8. Приложения 18](#_Toc536814987)

[8.1. Приложение 1:Класс TExсeption 18](#_Toc536814988)

[8.2. Приложение 2:Класс TArrList 18](#_Toc536814989)

[8.3. Приложение 3:Класс TQueue 22](#_Toc536814990)

[8.4. Приложение 4:Класс TStack 24](#_Toc536814991)

[8.5. Приложение 5:Код программы тестирования и экспериментов 26](#_Toc536814992)

[8.4. Приложение 4:Тесты для класса 29](#_Toc536814993)

# 1.Введение

Списки являются чрезвычайно гибкой структурой, так как их легко сделать большими или меньшими, и их элементы доступны для вставки или удаления в любой позиции списка. Списки также можно объединять или разбивать на меньшие списки. Списки регулярно используются в приложениях, например в программах информационного поиска, трансляторах программных языков или при моделировании различных процессов.

В математике список представляет собой последовательность элементов определенного типа ( elementtype ). Представим список в виде последовательности элементов, разделенных запятыми: a1, а2, ..., аn, где n≥ 0 и всё ai имеют тип elementtype. Количество элементов n - длина списка. Если n ≥ 1, то а1 называется первым элементом, а аn—последним элементом списка. В случае n = 0 имеем пустой список, который не содержит элементов.

Важное свойство списка заключается в том, что его элементы можно линейно упорядочить в соответствии с их позицией в списке. Говорим, что элемент ai предшествует ai+1 для і = 1, 2, ..., n — 1 и ai следует за ai-1 для і =2, 3, .., n. Также будем говорить, что элемент аi имеет позицию і.

Реализация списков посредством массивов

При реализации списков с помощью массивов элементы списка располагаются в смежных ячейках массива. Это представление позволяет легко просматривать содержимое списка и вставлять новые элементы в его конец. Но вставка нового элемента в середину списка требует перемещения всех, последующих элементов на одну позицию к концу массива, чтобы освободить место для нового элемента. Удаление элемента также требует перемещения элементов, чтобы закрыть освободившуюся ячейку.

|  |
| --- |
|  |
|  | http://bookwu.net/imgs/1432734111image011.jpg |

# 2. Цели и задачи

В рамках лабораторной работы ставится задача эффективной реализации структуры данных – линейный односвязный список на массивах и выполнение основных операций над ним:

* добавления элемента в начало списка,
* добавления элемента в конец списка,
* добавления, извлечения с удалением промежуточного элемента списка,
* извлечения с удалением элемента из начала списка,
* извлечения с удалением элемента из конца списка,
* проверка списка на полноту/пустоту.

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 методов шаблонного класса TArrList согласно заданному интерфейсу.
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 TArrList.
5. Модификация примера использования в тестовое приложение, позволяющее задавать списки 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 sample\_tarrlist.cpp (Рис.1):

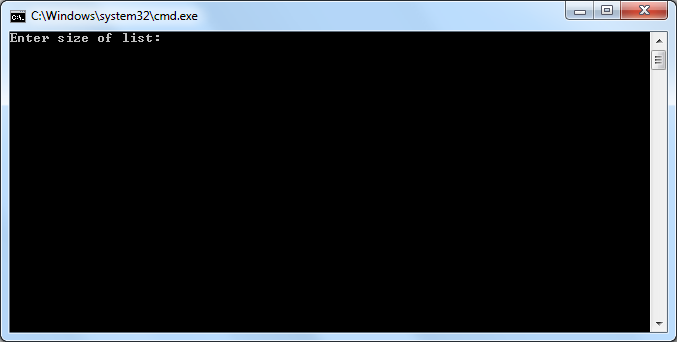


Рис.1.Ввод размера списка.

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

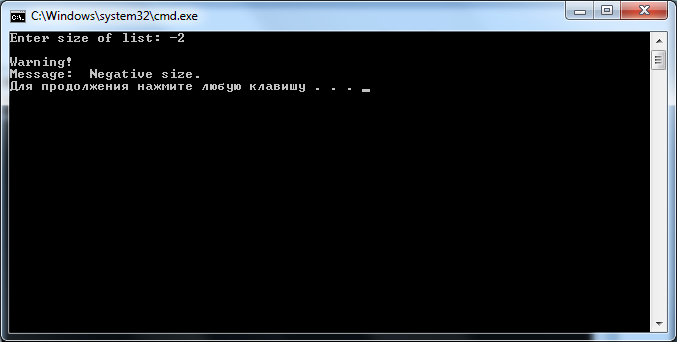


Рис.2.Ошибка.

При положительном значении создастся список заданного размера с пустыми ячейками, и пользователю предоставляется выбор операций со списком. (Рис.3):

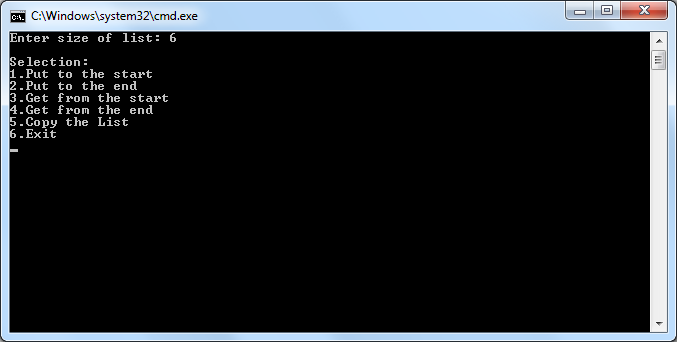


Рис.3.Меню операций.

Выбрав ‘Exit’, можно выйти из программы. (Рис.4):

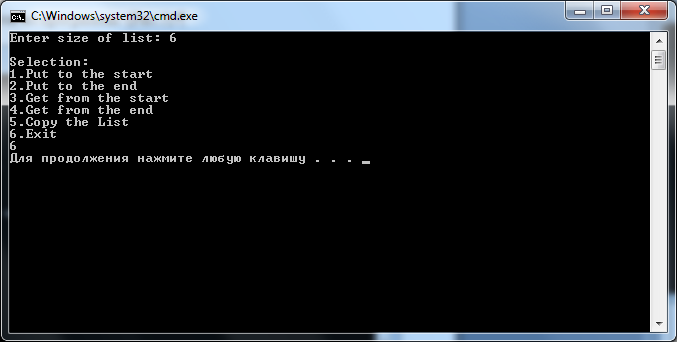


Рис.4.Выход из программы.

Если удалять из пустого списка или добавлять в полный, программа выведет ошибку и не выполнит операцию. (Рис.5, 6):

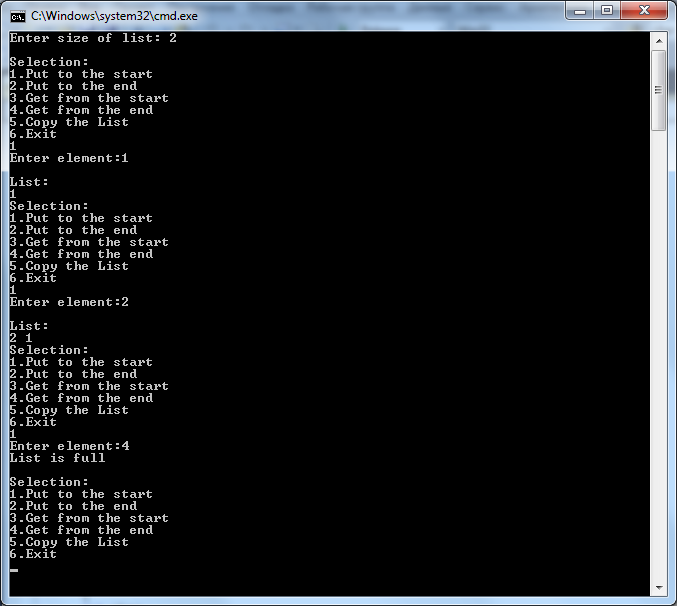


Рис.5.Ошибка – список заполнен.

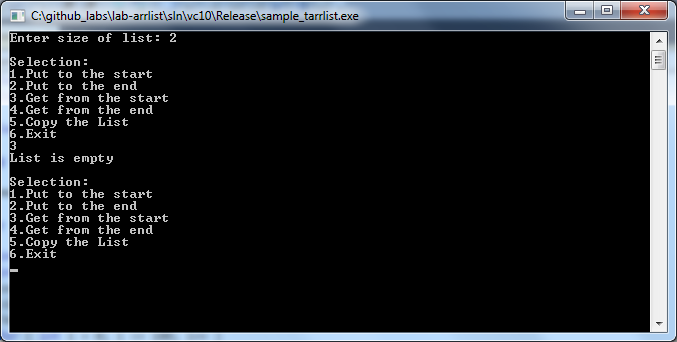


Рис.6.Ошибка - список пуст.

Чтобы добавить элемент в начало списка, нужно выбрать “1” и ввести новый элемент (Рис.7).

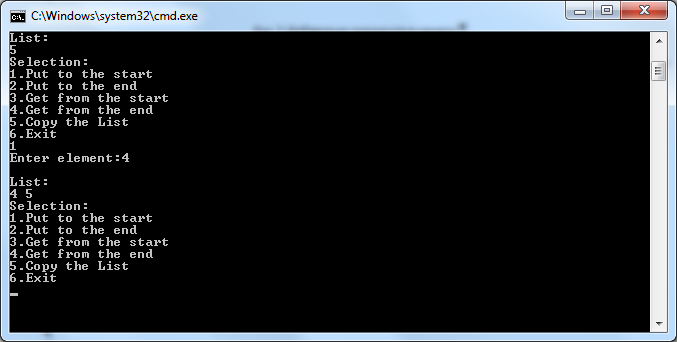


Рис.7.Добавлене элемента в начало.

Чтобы добавить элемент в конец списка, нужно выбрать “2”” и ввести новый элемент (Рис.8).

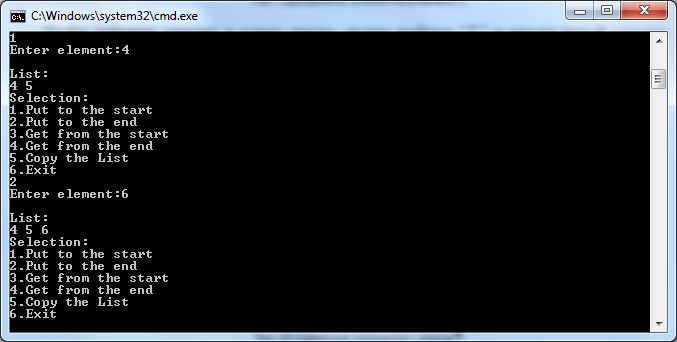


Рис.8.Добавлене элемента в конец.

Чтобы удалить элемент из начало списка, нужно выбрать “3” (Рис.9).

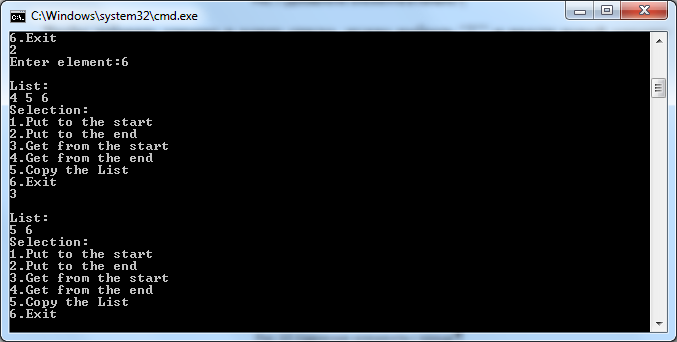


Рис.9.Удаление элемента из начала.

Чтобы удалить элемент с конца списка, нужно выбрать “4” (Рис.10).

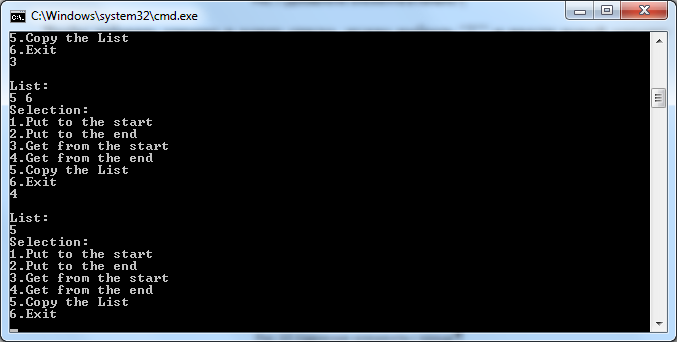


Рис.10.Удаление элемента с конца.

Чтобы скопировать список, нужно выбрать “5” (Рис.11).

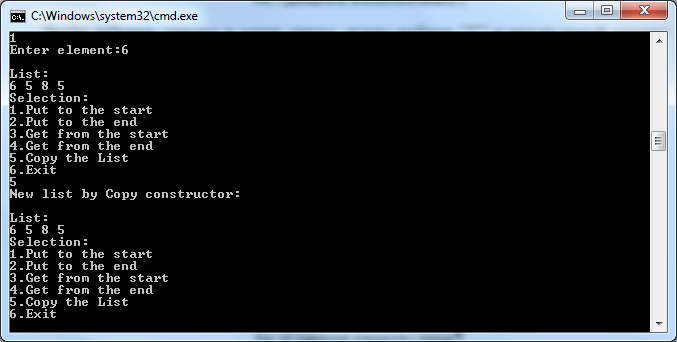


Рис.11.Копирование.

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 4 класса:

* Класс «Список» (TArrList), реализованный с использованием массивов и использующий класс TQueue.
* 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:

1. tartlist.h– модуль с классом TArrList, в котором определен интерфейс шаблонного класса Список и реализация его методов.
2. tstack.h– модуль с классом TStack, в котором определен интерфейс шаблонного класса Стек и реализация его методов.
3. tqueue.h– модуль с классом TQueue, в котором определен интерфейс шаблонного класса Очередь и реализация его методов.
4. exсeption.h – модуль с классом исключения TExсeption.
5. sample\_tattlist.cpp, sample\_performance\_check.cpp– модуль программы тестирования, с которым работает пользователь, в котором проводятся эксперименты.
6. test\_main.cpp, test\_tarrlist.cpp – модуль с функциями тестирования для созданных классов. Содержит 24 теста для класса TArrList.

## 4.2. Описание функций и процедур, их алгоритмов

Рассмотрим реализацию методов шаблонного класса TArrList:

template <class ArrListType>

class TArrList

1. TArrList(int \_size = 10) – конструктор класса, принимающий размер списка. По умолчанию создается список размера 10.

2. TArrList(TArrList<ArrListType> &List) - конструктор копирования.

Принимает ссылку на объект класса TArrList.

3. ~TArrList() - деструктор. Освобождает выделенную память.

4. void Put(int n, ArrListType elem) - метод, позволяющий добавить новый элемент в список на определенную позицию.

5. void PutStart(ArrListType elem) - метод, позволяющий добавить новый элемент в начало списка.

6. void PutEnd(ArrListType elem) - метод, позволяющий добавить новый элемент в конец списка.

7. ArrListType Get(int n) - метод изъятия элемента на определенной позиции из списка с удалением.

8. ArrListType GetStart() - метод изъятия элемента из начала списка с удалением.

9. ArrListType GetEnd() - метод изъятия элемента с конца списка с удалением.

10. bool IsFull() - метод проверки списка на полноту.

11. bool IsEmpty() - метод проверки списка на пустоту.

12. void Print() - метод отображения текущих элементов списка.

# 5. Эксперименты

В качестве примера рассмотрим операцию добавления элемента в конец и в начало списка для класса TArrList.

Теоретическая сложность выполнения алгоритмов O(1).

Мы провели измерение скорости добавления элемента в конец и в начало списка при разном количестве элементов: 10, 100, …, 1000000. Ниже вы можете увидеть таблицу зависимости времени выполнения операции от количества элементов списка. (Таблица.1)

По приведенным данным можно сделать вывод, что практическая сложность выполнения алгоритма равна теоретической.

|  |  |  |
| --- | --- | --- |
| Кол-во элементов | метод PutEnd() | метод PutStart() |
| 10 | 0 | 0 |
| 100 | 0 | 0 |
| 1000 | 0 | 0 |
| 10000 | 0 | 0 |
| 100000 | 0 | 0 |

Таблица.1. Зависимость времени выполнения операций от размера списка.

# 6. Заключение

В результате лабораторной работы была разработана структура данных – односвязный линейный список на массивах, 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://bookwu.net/book_algoritmy-i-struktury-dannyh_1245/11_2.1.-abstraktnyj-tip-dannyh-spisokquot>
3. Д. Кнут. Искусство программирования. (3-е издание) Т.1.

# 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 TArrList

|  |
| --- |
| **tarrlist.h** |
| #pragma once  #include <iostream>  #include "exception.h"  #include "tqueue.h"  using namespace std;  template <class ArrListType>  class TArrList  {  private:  int size;  int count;  int start;  int end;  int \*after;  int \*before;  ArrListType\* memory;  TQueue <int> nullElem;  public:  TArrList(int \_size = 10);  TArrList(TArrList<ArrListType> &List);  ~TArrList();  void Put(int n, ArrListType elem);  void PutStart(ArrListType elem);  void PutEnd(ArrListType elem);  ArrListType Get(int n);  ArrListType GetStart();  ArrListType GetEnd();  bool IsFull();  bool IsEmpty();  void Print();  };  //-----------------------------------------------------------------  template <class ArrListType>  TArrList<ArrListType>::TArrList(int \_size) : nullElem(\_size)  {  if ( \_size <= 0 )  throw TException( "False size list." );  size = \_size;  count = 0;  start = -1;  end = -1;  memory = new ArrListType[size];  after = new int[size];  before = new int[size];  for ( int i = 0; i < size; i++ )  {  after[i] = -2;  before[i] = -2;  nullElem.Put(i);  }  }  //-----------------------------------------------------------------  template <class ArrListType>  TArrList<ArrListType>::TArrList(TArrList<ArrListType> &List)  {  start = List.start;  end = List.end;  size = List.size;  count = List.count;  memory = new ArrListType[size];  after = new int[size];  before = new int[size];  nullElem = List.nullElem;  for ( int i = 0; i < size; i++ )  {  memory[i] = List.memory[i];  after[i] = List.after[i];  before[i] = List.before[i];  }  }  //-----------------------------------------------------------------  template<class ArrListType>  inline TArrList<ArrListType>::~TArrList()  {  delete[] memory;  delete[] after;  delete[] before;  }  //-----------------------------------------------------------------  template<class ArrListType>  void TArrList<ArrListType>::Put(int n, ArrListType elem)  {  if (IsFull())  throw TException( "List is full." );  if (n < 1 || n > count - 1)  throw TException( "Uncurrent index." );  int t = nullElem.Get();  memory[t] = elem;  int first = start;  int second = after[start];  for ( int i = 0; i < n - 1; i++ )  {  first = second;  second = after[second];  }  after[t] = second;  after[first] = t;  before[t] = first;  before[second] = t;  count++;  }  //-----------------------------------------------------------------  template <class ArrListType>  void TArrList<ArrListType>::PutStart(ArrListType elem)  {  if ( IsFull() )  throw TException( "List is full." );  int t = nullElem.Get();  memory[t] = elem;  after[t] = start;  if ( start != -1 )  before[start] = t;  else  end = t;  start = t;  count++;  }  //-----------------------------------------------------------------  template <class ArrListType>  void TArrList<ArrListType>::PutEnd(ArrListType elem)  {  if ( IsFull() )  throw TException( "List is full." );  int t = nullElem.Get();  memory[t] = elem;  if ( end != -1 )  after[end] = t;  else  {  start = t;  before[t] = -1;  }  before[t] = end;  end = t;  count++;  }  //-----------------------------------------------------------------  template<class ArrListType>  ArrListType TArrList<ArrListType>::Get(int n)  {  if ( IsEmpty() )  throw TException( "List is empty." );  if ( n < 1 || n > count - 1 )  throw TException( "Uncurrent index." );  int t = start;  for ( int i = 0; i < n; i++ )  t = after[t];  after[before[t]] = after[t];  before[after[t]] = before[t];  ArrListType temp = memory[t];  nullElem.Put(t);  count--;  return temp;  }  //-----------------------------------------------------------------  template <class ArrListType>  ArrListType TArrList<ArrListType>::GetStart()  {  if (IsEmpty())  throw TException( "List is empty." );  ArrListType elem = memory[start];  nullElem.Put(start);  int newstart = after[start];  after[start] = before[start] = -2;  if (newstart != -1)  before[newstart] = -1;  count--;  start = newstart;  return elem;  }  //-----------------------------------------------------------------  template <class ArrListType>  ArrListType TArrList<ArrListType>::GetEnd()  {  if ( IsEmpty() )  throw TException( "List is empty." );  ArrListType elem = memory[end];  int newfinish = before[end];  before[end] = after[end] = -2;  nullElem.Put(end);  end = newfinish;  if ( newfinish != -1 )  after[newfinish] = -1;  else  start = -1;  count--;  return elem;  }  //-----------------------------------------------------------------  template <class ArrListType>  bool TArrList<ArrListType>::IsFull()  {  if ( count == size )  return true;  return false;  }  //-----------------------------------------------------------------  template <class ArrListType>  bool TArrList<ArrListType>::IsEmpty()  {  if ( count == 0 )  return true;  return false;  }  //-----------------------------------------------------------------  template <class ArrListType>  void TArrList<ArrListType>::Print()  {  int t = start;  for ( int i = 0; i < count; i++ )  {  cout << memory[t] << " ";  t = after[t];  }  } |

## 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: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;  } |

## 8.5. Приложение 5:Код программы тестирования и экспериментов

|  |
| --- |
| **sample\_tarrlist.cpp** |
| #include <iostream>  #include <locale.h>  #include "tarrlist.h"  using namespace std;  int main()  {  try  {  int size;  int input;  cout << "Enter size of list: ";  cin >> size;  TArrList<int> Arrlist(size);  for ( int i = 0; i <= 100; i++ )  {  cout << endl << "Selection:" << endl;  cout << "1.Put to the start " << endl;  cout << "2.Put to the end " << endl;  cout << "3.Get from the start"<< endl;  cout << "4.Get from the end"<< endl;  cout << "5.Copy the List"<< endl;  cout << "6.Exit"<< endl;  cin >> input;  switch ( input )  {  case 1:  {  int elem;  cout << "Enter element:";  cin >> elem;  if ( !Arrlist.IsFull() )  {  Arrlist.PutStart(elem);  cout << endl << "List: "<< endl;  Arrlist.Print();  }  else  cout<<"List is full"<<endl;  break;  }  case 2:  {  int elem;  cout << "Enter element:";  cin >> elem;  if ( !Arrlist.IsFull() )  {  Arrlist.PutEnd(elem);  cout << endl << "List: "<< endl;  Arrlist.Print();  }  else  cout<<"List is full"<<endl;  break;  }  case 3:  {  if( !Arrlist.IsEmpty() )  {  Arrlist.GetStart();  cout << endl << "List: "<< endl;  Arrlist.Print();  }  else  cout<<"List is empty"<<endl;  break;  }  case 4:  {  if( !Arrlist.IsEmpty() )  {  Arrlist.GetEnd();  cout << endl << "List: "<< endl;  Arrlist.Print();  }  else  cout<<"List is empty"<<endl;  break;  }  case 5:  {  cout << "New list by Copy constructor:"<< endl;  TArrList<int> Arrlist2(Arrlist);  cout << endl << "List: "<< endl;  Arrlist2.Print();  break;  }  case 6:  {  return 0;  }  default:  cout << "Invalid Input!"<< endl;  }  }  }  catch (TException exp)  {  exp.Show();  }  return 0;  } |

|  |
| --- |
| **sample\_performance\_check.cpp** |
| #include <iostream>  #include "tarrlist.h"  #include "time.h"  using namespace std;  int main()  {  clock\_t time\_start;  clock\_t total\_time;  for ( unsigned size = 10; size < 1000000; size \*= 10 )  {  TArrList<int> Arrlist1(size);  for ( unsigned i = 0; i < size - 1; i++ )  {  Arrlist1.PutStart(i);  }  total\_time = 0;  time\_start = clock();  Arrlist1.PutEnd(1);  //Arrlist1.PutStart(1);  total\_time += clock() - time\_start;  /\* cout<< "Size is: " << size << " Time(put to the start) is: " << total\_time <<endl;\*/  cout<< "Size is: " << size << " Time(put to the end) is: " << total\_time <<endl;  }  return 0;  } |

## 8.4. Приложение 4:Тесты для класса

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

|  |
| --- |
| **test\_tarrlist.cpp** |
| #include <gtest.h>  #include "tarrlist.h"  TEST(TArrList, can\_create\_arrlist)  {  ASSERT\_NO\_THROW(TArrList <int> list);  }  TEST(TArrList, can\_create\_list\_with\_positive\_size)  {  ASSERT\_NO\_THROW(TArrList <int> list(5));  }  TEST(TArrList, throw\_when\_create\_arrlist\_with\_negative\_size)  {  ASSERT\_ANY\_THROW(TArrList <int> list(-5));  }  TEST(TArrList, throw\_when\_create\_arrlist\_with\_size\_eq\_zero)  {  ASSERT\_ANY\_THROW(TArrList <int> list(0));  }  TEST(TArrList, can\_put\_elem\_to\_the\_start\_of\_arrlist)  {  TArrList <int> list(5);  ASSERT\_NO\_THROW(list.PutStart(1));  }  TEST(TArrList, can\_put\_elem\_to\_the\_end\_of\_arrlist)  {  TArrList <int> list(5);  ASSERT\_NO\_THROW(list.PutEnd(1));  }  TEST(TArrList, can\_put\_elem\_into\_arrlist)  {  TArrList <int> list(5);  list.PutStart(4);  list.PutStart(3);  list.PutStart(2);  list.Put(2, 1);  ASSERT\_EQ(list.Get(2), 1);  }  TEST(TArrList, arrlist\_with\_elem\_isnt\_empty)  {  TArrList <int> list(5);  list.PutStart(1);  EXPECT\_EQ(false, list.IsEmpty());  }  TEST(TArrList, cant\_put\_to\_the\_start\_into\_full\_arrlist)  {  TArrList <int> list(5);  for ( int i = 0; i < 5; i++ )  {  list.PutStart(1);  }  ASSERT\_ANY\_THROW(list.PutStart(1));  }  TEST(TArrList, cant\_put\_to\_the\_end\_into\_full\_arrlist)  {  TArrList <int> list(5);  for ( int i = 0; i < 5; i++ )  {  list.PutStart(1);  }  ASSERT\_ANY\_THROW(list.PutEnd(1));  }  TEST(TArrList, cant\_put\_into\_full\_list)  {  TArrList <int> list(5);  for ( int i = 0; i < 5; i++ )  {  list.PutStart(1);  }  ASSERT\_ANY\_THROW(list.Put(1, 6));  }  TEST(TArrList, can\_get\_elem\_from\_the\_start\_of\_arrlist)  {  TArrList <int> list(5);  list.PutStart(2);  list.PutStart(1);  int res = list.GetStart();  EXPECT\_EQ(1, res);  }  TEST(TArrList, can\_get\_elem\_from\_the\_end\_of\_arrlist)  {  TArrList <int> list(5);  list.PutStart(2);  list.PutStart(1);  int res = list.GetEnd();  EXPECT\_EQ(2, res);  }  TEST(TArrList, can\_get\_elem\_from\_arrlist)  {  TArrList <int> list(5);  for ( int i = 1; i < 5; i++ )  {  list.PutEnd(i);  }  ASSERT\_EQ(list.Get(1), 2);  }  TEST(TArrList, throw\_when\_get\_last\_element\_from\_empty\_arrlist)  {  TArrList <int> list(5);  ASSERT\_ANY\_THROW(list.GetEnd());  }  TEST(TArrList, throw\_when\_get\_start\_element\_from\_empty\_arrlist)  {  TArrList <int> list(5);  ASSERT\_ANY\_THROW(list.GetStart());  }  TEST(TArrList, throws\_when\_get\_elem\_from\_empty\_arrlist)  {  TArrList <int> list(5);  ASSERT\_ANY\_THROW(list.Get(1));  }  TEST(TArrList, can\_create\_right\_copy\_arrlist)  {  TArrList <int> list1(5);  list1.PutStart(1);  TArrList <int> list2(list1);  ASSERT\_EQ(list1.GetStart(), list2.GetStart());  }  TEST(TArrList, can\_use\_isempty\_correctly)  {  TArrList <int> list(5);  ASSERT\_TRUE(list.IsEmpty());  }  TEST(TArrList, can\_use\_isempty\_incorrectly)  {  TArrList <int> list(5);  list.PutStart(1);  ASSERT\_FALSE(list.IsEmpty());  }  TEST(TArrList, can\_use\_isfull\_correctly)  {  TArrList <int> list(1);  list.PutStart(1);  ASSERT\_TRUE(list.IsFull());  }  TEST(TArrList, can\_use\_isfull\_incorrectly)  {  TArrList <int> list(5);  ASSERT\_FALSE(list.IsFull());  }  TEST(TArrList, throw\_when\_put\_with\_incurrent\_index)  {  TArrList <int> list(5);  ASSERT\_ANY\_THROW(list.Put(10, 6));  }  TEST(TArrList, throw\_when\_get\_with\_incurrent\_index)  {  TArrList <int> list(5);  list.PutStart(1);  list.PutStart(1);  ASSERT\_ANY\_THROW(list.Get(10));  } |