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 №14**

**«Упорядоченная таблица»**

**Выполнил:** студент группы 381706-2

Жбанова Надежда Сергеевна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Руководитель:**

Ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2019

Содержание

[1.Введение 3](#_Toc9515559)

[2. Цели и задачи 4](#_Toc9515560)

[2.1. Используемые инструменты 4](#_Toc9515561)

[3. Руководство пользователя 6](#_Toc9515562)

[4. Руководство программиста 9](#_Toc9515563)

[4.1. Описание структуры программы 9](#_Toc9515564)

[4.2. Описание функций и процедур, их алгоритмов 9](#_Toc9515565)

[6. Заключение 14](#_Toc9515566)

[7. Литература 15](#_Toc9515567)

[8. Приложения 16](#_Toc9515568)

[8.1. Приложение 1:Класс TExсeption 16](#_Toc9515569)

[8.2. Приложение 2:Класс TString 16](#_Toc9515570)

[8.3. Приложение 3:Класс TElement 18](#_Toc9515571)

[8.4. Приложение 4:Класс TTableSee 20](#_Toc9515572)

[8.5. Приложение 5:Класс TSortTable 22](#_Toc9515573)

[8.6. Приложение 6:Код программы тестирования 28](#_Toc9515574)

[8.7. Приложение 7:Тесты для классов 29](#_Toc9515575)

# 1.Введение

Понятие таблицы:

**Таблица**— набор элементов одинаковой организации, каждый из которых можно представить в виде двойки <K, V>, где K— ключ, а V— тело (информационная часть) элемента.

Ключ уникален для каждого элемента, то есть в таблице нет элементов с одинаковыми ключами. Ключ используется для доступа к элементам при выполнении операций.

Виды таблиц:

Существует три вида таблиц: неупорядоченная, упорядоченная и хеш-таблица. В данной лабораторной работе будет реализован следующий тип таблицы:

Упорядоченная таблица

В такой таблице все элементы **упорядочиваются по возрастанию** значения ключа. Для поиска элемента с заданным ключом применяются алгоритмы линейного, бинарного или **блочного поиска**. После выполнения операции включения, элементы таблицы должны остаться упорядоченными, поэтому перед выполнением операции включения необходимо найти элемент, после которого нужно включить элемент.

Почему сортировке уделяется большое внимание можно понять, прочитав следующую цитату:

"Даже если бы сортировка была почти бесполезна, нашлась бы масса причин заняться ею! Изобретательные методы сортировки говорят о том, что она и сама по себе интересна как объект исследования." (Д. Кнут)

# 2. Цели и задачи

В рамках лабораторной работы ставится задача эффективной реализации структуры данных - упорядоченная таблица и выполнение основных операций над ней:

* инициализация
* включение элемента с заданным ключом
* исключение элемента с заданным ключом
* поиск элемента
* сортировка элементов

Для реализации этой структуры данных будем использовать массив элементов таблицы, состоящих из хранимого элемента и его ключа.

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 элемента таблицы TElement согласно заданному интерфейсу, с использованием класса строки TString.
2. Реализация методов шаблонного класса упорядоченной таблицы TSortTable согласно заданному интерфейсу, на основе класса просмотровой таблицы TTableSee.
3. Реализация класса для обработки исключений– TException, которые могут возникнуть при выполнении различных операций, согласно заданному интерфейсу.
4. Обеспечение работоспособности тестов и примера использования.
5. Реализация заготовок тестов, покрывающих все методы созданных классов.
6. Модификация примера использования в тестовое приложение, позволяющее осуществлять основные операции над таблицей.

## 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\_tablesee.cpp (Рис.1).

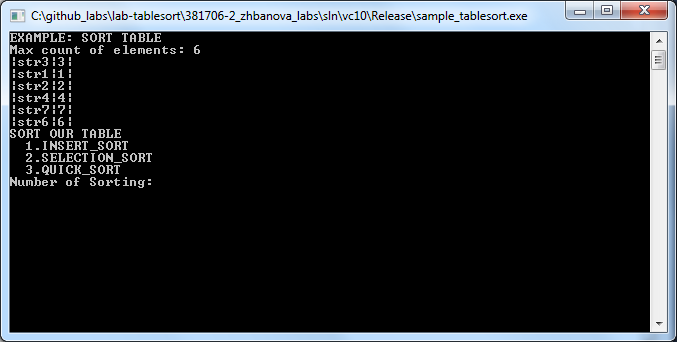


Рис.1.Неупорядоченная таблица.

Пользователю нужно выбрать номер сортировки, чтобы упорядочить предлагаемую таблицу. (Рис.2):

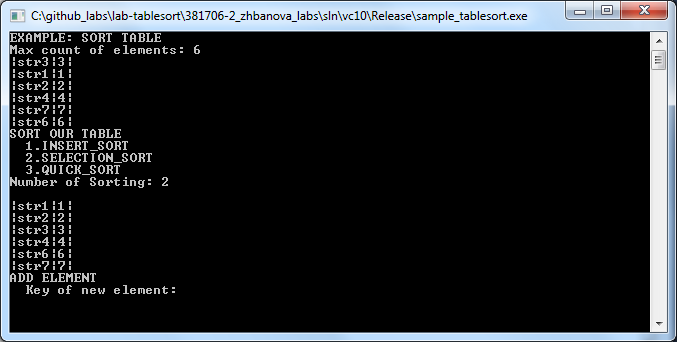


Рис.2.Сортировка.

В уже полную отсортированную таблицу пользователю предлагается добавить еще один элемент, указав его ключ и значение (Рис.3), а потом удалить любой элемент таблицы по ключу (Рис.4).

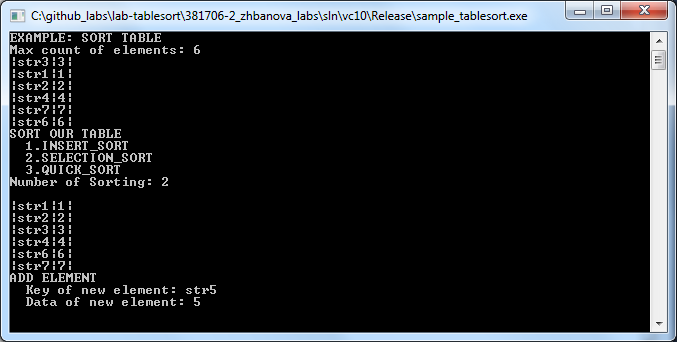


Рис.3.Добавление элемента.

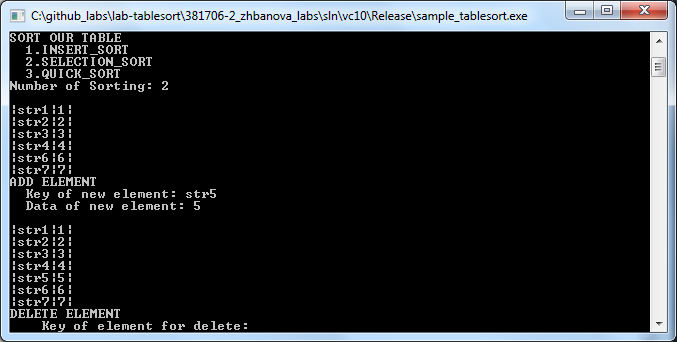


Рис.4.Удаление элемента.

Результат операций автоматически выведется на экран (Рис.5).

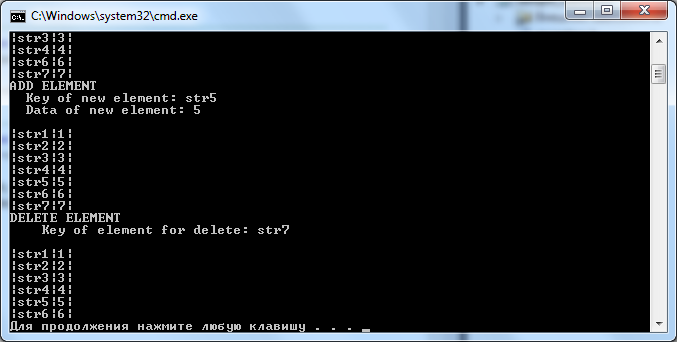


Рис.5.Результат.

Алгоритм можно повторить несколько раз.

# 4. Руководство программиста

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

В данной работе будет использовано несколько классов:

* Класс «Строка» (TString).
* Класс «Элемент таблицы» (TElement).
* Класс «Просмотровая таблица» (TTableSee).
* Класс «Упорядоченная таблица» (TSortTable).
* Класс исключения (TExсeption).

## 4.1. Описание структуры программы

Модульная структура программы:

1. ttablesort.h - модуль с классом TSortTable, в котором определен интерфейс шаблонного класса «Упорядоченная таблица» и реализация его методов.
2. tablesee.h– модуль с классом TTableSee, в котором определен интерфейс шаблонного класса «Просмотровая таблица» и реализация его методов.
3. telement.h– модуль с классом TElement, в котором определен интерфейс шаблонного класса «Элемент таблицы» и реализация его методов.
4. tstring.h– модуль с классом TString, в котором определен интерфейс класса «Строка» и реализация его методов.
5. exсeption.h – модуль с классом исключения TExсeption.
6. sample\_ttablesort.cpp – модуль программы тестирования, с которым работает пользователь.
7. test\_main.cpp, test\_ttablesort.cpp – модуль с функциями тестирования для созданных классов. Содержат 10 тестов для класса TString, 9 - для класса TElement и 13 - для класса TTableSee, 17 - для класса TSortTable.

## 4.2. Описание функций и процедур, их алгоритмов

Рассмотрим реализацию методов класса TString:

class TString

char\* memory – указатель на массив символов строки.

int count – кол-во символов в строке.

1. TString() - конструктор по умолчанию.

2. TString(const char\* A) - конструктор с параметрами. Принимает указатель на массив символов.

3. TString(const TString& A) - конструктор копирования.

Принимает ссылку на объект класса TString.

4. ~TString() - деструктор.

5. int GetCount() const – метод, возвращающий кол-во символов строки.

6. char\* GetMemory() – метод, возвращающий указатель на массив символов строки.

7. TString& operator=(const TString& A) - перегрузка оператора присваивания.

8. bool operator==(const TString& A) const - перегрузка оператора сравнения“==”.

9. bool operator!=(const TString& A) const - перегрузка оператора сравнения“!=”.

10. bool operator<(const TString& A) const - перегрузка оператора “<”.

11. bool operator>(const TString& A) const - перегрузка оператора “>”.

12. char& operator[](const int a) const - перегрузка оператора индексации.

Принимает номер элемента массива символов.

Рассмотрим реализацию методов шаблонного класса TElement:

template <typename ElemType>

class TElement

ElemType data – значение элемента таблицы.

TString key - ключ элемента таблицы.

1. TElement() - конструктор по умолчанию.

2. TElement(const ElemType& A, const TString& B) - конструктор с параметрами. Принимает два параметра: значение элемента таблицы и его ключ.

3. TElement(const ElemType& A) - конструктор копирования.

Принимает ссылку на объект класса TElement.

4. ~TElement() - деструктор.

5. TElement& operator=(const TElement<ElemType>& A) - перегрузка оператора присваивания.

6. bool operator==(const TElement<ElemType>& A) - перегрузка оператора сравнения“==”.

Возвращает true, если элементы таблицы равны, и false – если не равны.

7. bool operator!=(const TElement<ElemType>& A) - перегрузка оператора сравнения“!=”.

Возвращает true, если элементы таблицы не равны, и false – если равны.

8. ElemType& GetData() - метод, возвращающий значение элемента таблицы.

9. TString& GetKey() – метод, возвращающий ключ элемента таблицы.

10. void SetData(const ElemType& A) - метод, устанавливающий значение элемента таблицы.

11. void SetKey(const TString& A) - метод, устанавливающий ключ элемента таблицы.

12. template <typename Operator>

friend ostream& operator<<(ostream& out, TElement< Operator >& A) - дружественный метод, перегрузка оператора потокового вывода элемента таблицы.

Рассмотрим реализацию методов шаблонного класса TTableSee:

template <typename TableSeeType>

class TTableSee

static TElement<TableSeeType> st - первоначальный пустой элемент таблицы.

TElement<TableSeeType>\* node – указатель на массив элементов таблицы.

int size – кол-во элементов таблицы.

int count – кол-во используемых элементов таблицы.

1. void Expansion(const int \_size) – защищенный метод, увеличивающий максимальное кол-во элементов созданной таблицы.

2. TTableSee(const int \_size = 1) – конструктор, принимающий кол-во элементов таблицы. По умолчанию кол-во элементов таблицы = 1;

3. TTableSee(const TTableSee<TableSeeType>& A) - конструктор копирования.

Принимает ссылку на объект класса TTableSee.

4. ~TTableSee() - деструктор.

5. int GetCount() const – метод, возвращающий кол-во используемых элементов таблицы.

6. int GetSize() const – метод, возвращающий максимальное кол-во элементов таблицы.

7. TElement<TableSeeType>\* GetNode() - метод, возвращающий указатель на массив элементов таблицы.

8. bool Add(const TElement<TableSeeType>& A) - метод, добавляющий элемент в таблицу.

Принимает ссылку на объект класса TTableSee.

9. bool Delete(TElement<TableSeeType>& A) - метод, удаляющий элемент таблицы.

Принимает ссылку на объект класса TTableSee.

10. TString& Add(const TableSeeType& A) - метод, добавляющий элемент в таблицу.

В качестве аргумента принимает значение, ключ нового элемента установится в“Key”.

11. bool Delete(const TString& A) – метод, удаляющий элемент таблицы.

В качестве аргумента принимает его ключ.

12. TElement<TableSeeType>& Search(const TString& A) const – метод, осуществляющий поиск элемента в таблице по его ключу.

13. TableSeeType& operator[](const TString& A) const - перегрузка оператора индексации.

Возвращает элемент таблицы по его ключу.

14. template <typename Op>

friend ostream& operator<<(ostream& ostr, const TTableSee<Op>& A) - дружественный метод, перегрузка оператора потокового вывода просмотровой таблицы.

Рассмотрим реализацию методов шаблонного класса TSortTable:

template <typename SortTableType>

class TSortTable

static TElement<SortTableType> stt - первоначальный пустой элемент таблицы.

TElement<SortTableType>\* node - указатель на массив элементов таблицы.

int count - кол-во используемых элементов таблицы.

int size - кол-во элементов таблицы.

1. void Expansion(const int \_size) – защищенный метод, увеличивающий максимальное кол-во элементов созданной таблицы.

2. TSortTable (const int \_size = 1) - конструктор, принимающий кол-во элементов таблицы.

По умолчанию кол-во элементов таблицы = 1;

3. TSortTable (const TTableSee<SortTableType>& A, const int num) - конструктор, принимающий просмотровую таблицу и номер сортировки.

4. TSortTable (const TSortTable<SortTableType>& A) - - конструктор копирования.

Принимает ссылку на объект класса TSortTable.

5. ~TSortTable() - деструктор.

6. int GetCount() const – метод, возвращающий кол-во используемых элементов таблицы.

7. int GetSize() const - метод, возвращающий максимальное кол-во элементов таблицы.

8. bool Add(TElement<SortTableType>& A) - метод, добавляющий элемент в таблицу.

Принимает ссылку на объект класса TSortTable.

9. TString& Add(const SortTableType& A) - метод, добавляющий элемент в таблицу.

В качестве аргумента принимает значение.

10. bool Delete(TElement<SortTableType>& A) - метод, удаляющий элемент таблицы.

Принимает ссылку на объект класса TSortTable.

11. bool Delete(const TString& A) - метод, удаляющий элемент таблицы.

В качестве аргумента принимает его ключ.

12. TElement<SortTableType>& LineSearch(const TString& A) const - метод, осуществляющий линейный поиск элемента в таблице по его ключу.

13. TElement<SortTableType>& BinarySearch(const TString& A) const - метод, осуществляющий бинарный поиск элемента в таблице по его ключу.

14. static void InsertSort(TTableSee<SortTableType>& A) – статический метод, реализующий сортировку вставками.

Принимает ссылку на объект класса просмотровой таблицы.

15. static void SelectionSort(TTableSee<SortTableType>& A) - статический метод, реализующий сортировку прямым выбором.

Принимает ссылку на объект класса просмотровой таблицы.

16. static void QuickSort(TTableSee<SortTableType>& A, const int low, const int high) - статический метод, реализующий быструю сортировку.

Принимает ссылку на объект класса просмотровой таблицы.

17. SortTableType& operator[](const TString& A) const - перегрузка оператора индексации.

Возвращает элемент таблицы по его ключу.

18. template <typename Type>

friend ostream& operator<<(ostream& ostr, const TSortTable<Type>& A) - дружественный метод, перегрузка оператора потокового вывода просмотровой таблицы.

# 

# 6. Заключение

В результате лабораторной работы была разработана структура данных – упорядоченная таблица, 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).

Созданные классы был протестированы с использованием Google Tests.

# 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://mvblog.ru/archives/343/>
3. КнутД.Э. “Искусство программирования. Том 3. Сортировка и поиск”. Вильямс, 2012

# 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 TString

|  |
| --- |
| **tstring.h** |
| #pragma once  #include <iostream>  #include <cstring>  #include "exception.h"  class TString  {  protected:  char\* memory;  int count;  public:  TString();  TString(const char\* A);  TString(const TString& A);  ~TString();  int GetCount() const { return count; }  char\* GetMemory() { return memory; }  TString& operator=(const TString& A);  bool operator==(const TString& A) const;  bool operator!=(const TString& A) const;  bool operator<(const TString& A) const;  bool operator>(const TString& A) const;  char& operator[](const int a) const;  };  //----------------------------------------------------------------------  TString::TString() : count(0), memory(NULL)  {  }  //----------------------------------------------------------------------  TString::TString(const char\* A)  {  count = strlen(A);  memory = new char[count + 1];  for (int i = 0; i < count; i++)  memory[i] = A[i];  memory[count] = '\0';  }  //----------------------------------------------------------------------  TString::TString(const TString& A)  {  count = A.count;  memory = new char[count + 1];  for (int i = 0; i < count; i++)  memory[i] = A.memory[i];  memory[count] = '\0';  }  //----------------------------------------------------------------------  TString::~TString()  {  count = 0;  delete[] memory;  }  //----------------------------------------------------------------------  TString& TString::operator=(const TString& A)  {  count = A.count;  delete[] memory;  memory = new char[count + 1];  for (int i = 0; i < count; i++)  memory[i] = A.memory[i];  memory[count] = '\0';  return \*this;  }  //----------------------------------------------------------------------  bool TString::operator==(const TString& A) const  {  if (count != A.count)  return false;  for (int i = 0; i < count; i++)  if (memory[i] != A.memory[i])  return false;  return true;  }  //----------------------------------------------------------------------  bool TString::operator!=(const TString& A) const  {  if (count != A.count)  return true;  for (int i = 0; i < count; i++)  if (memory[i] != A.memory[i])  return true;  return false;  }  //----------------------------------------------------------------------  bool TString::operator<(const TString& A) const  {  bool res = false;  if (count < A.count)  return true;  else if (count > A.count)  return false;  else  {  for (int i = 0; i < count; i++)  if (memory[i] < A.memory[i])  {  res = true;  break;  }  }  return res;  }  //----------------------------------------------------------------------  bool TString::operator>(const TString& A) const  {  bool res = false;  if (count < A.count)  return false;  else if (count > A.count)  return true;  else  for (int i = 0; i < count; i++)  if (memory[i] > A.memory[i])  {  res = true;  break;  }  return res;  }  //----------------------------------------------------------------------  char& TString::operator[](const int a) const  {  if (a < 0 || a > count - 1)  throw TException("Error");  return memory[a];  } |

## 8.3. Приложение 3:Класс TElement

|  |
| --- |
| **telement.h** |
| #pragma once  #include "tstring.h"  #include <iostream>  using namespace std;  template <typename ElemType>  class TElement  {  protected:  ElemType data;  TString key;  public:  TElement();  TElement(const ElemType& A, const TString& B);  TElement(const ElemType& A);  ~TElement(){}  TElement& operator=(const TElement<ElemType>& A);  bool operator==(const TElement<ElemType>& A);  bool operator!=(const TElement<ElemType>& A);  ElemType& GetData() { return data; }  TString& GetKey() { return key; }  void SetData(const ElemType& A) { data = A; }  void SetKey(const TString& A) { key = A; }  template <typename Operator>  friend ostream& operator<<(ostream& ostr, TElement<Operator>& A);  };  //----------------------------------------------------------------------  template <typename ElemType>  TElement<ElemType>::TElement() : data(0), key(" ")  {  }  //----------------------------------------------------------------------  template <typename ElemType>  TElement<ElemType>::TElement(const ElemType& A, const TString& B) : data(A), key(B)  {  }  //----------------------------------------------------------------------  template <typename ElemType>  TElement<ElemType>::TElement(const ElemType& A) : data(A.data), key(A.key)  {  }  //----------------------------------------------------------------------  template <typename ElemType>  TElement<ElemType>& TElement<ElemType>::operator=(const TElement<ElemType>& A)  {  if (this != &A)  {  data = A.data;  key = A.key;  }  return \*this;  }  //----------------------------------------------------------------------  template <typename ElemType>  bool TElement<ElemType>::operator==(const TElement<ElemType>& A)  {  if (data == A.data && key == A.key)  return true;  return false;  }  //----------------------------------------------------------------------  template <typename ElemType>  bool TElement<ElemType>::operator!=(const TElement<ElemType>& A)  {  if (data == A.data && key == A.key)  return false;  return true;  }  //----------------------------------------------------------------------  template <typename Operator>  ostream& operator<<(ostream& ostr, TElement<Operator>& A)  {  ostr << "|" << A.GetKey().GetMemory() << "|";  ostr << A.GetData() <<"|";  return ostr;  } |

## 8.4. Приложение 4:Класс TTableSee

|  |
| --- |
| **tablesee.h** |
| #pragma once  #include "telement.h"  using namespace std;  template <typename TableSeeType>  class TTableSee  {  protected:  static TElement<TableSeeType> st;  TElement<TableSeeType>\* node;  int size;  int count;  void Expansion(const int \_size);  public:  TTableSee(const int \_size = 1);  TTableSee(const TTableSee<TableSeeType>& A);  ~TTableSee();  int GetCount() const { return count; }  int GetSize() const { return size; }  TElement<TableSeeType>\* GetNode() { return node; }  bool Add(const TElement<TableSeeType>& A);  bool Delete(TElement<TableSeeType>& A);  TString& Add(const TableSeeType& A);  bool Delete(const TString& A);  TElement<TableSeeType>& Search(const TString& A) const;  TableSeeType& operator[](const TString& A) const;  template <typename Op>  friend ostream& operator<<(ostream& ostr, const TTableSee<Op>& A);  };  //----------------------------------------------------------------------  template <typename TableSeeType>  void TTableSee<TableSeeType>::Expansion(const int \_size)  {  if (\_size > size)  {  TElement<TableSeeType>\* tmp = new TElement<TableSeeType>[\_size];  for (int i = 0; i < count; i++)  tmp[i] = node[i];  for (int i = count; i < \_size; i++)  tmp[i] = st;  size = \_size;  delete[] node;  node = tmp;  }  else  throw TException("Error");  }  //----------------------------------------------------------------------  template <typename TableSeeType>  TElement<TableSeeType> TTableSee<TableSeeType>::st;  //----------------------------------------------------------------------  template <typename TableSeeType>  TTableSee<TableSeeType>::TTableSee(const int \_size)  {  if (\_size < 0)  throw TException("Error");  size = \_size;  count = 0;  node = new TElement<TableSeeType>[size];  for (int i = 0; i < size; i++)  node[i] = st;  }  //----------------------------------------------------------------------  template <typename TableSeeType>  TTableSee<TableSeeType>::TTableSee(const TTableSee<TableSeeType>& A)  {  size = A.size;  count = A.count;  node = new TElement<TableSeeType>[size];  for (int i = 0; i < size; i++)  node[i] = A.node[i];  }  //----------------------------------------------------------------------  template <typename TableSeeType>  TTableSee<TableSeeType>::~TTableSee()  {  count = size = 0;  delete[] node;  }  //----------------------------------------------------------------------  template <typename TableSeeType>  bool TTableSee<TableSeeType>::Add(const TElement<TableSeeType>& A)  {  if (size == count)  Expansion(count \* 2);  node[count] = A;  count++;  return true;  }  //----------------------------------------------------------------------  template <typename TableSeeType>  TString& TTableSee<TableSeeType>::Add(const TableSeeType& A)  {  if (size == count)  Expansion(count \* 2);  node[count].SetData(A);  TString string("Key");  node[count].SetKey(string);  count++;  return node[count - 1].GetKey();  }  //----------------------------------------------------------------------  template <typename TableSeeType>  bool TTableSee<TableSeeType>::Delete(TElement<TableSeeType>& A)  {  if (count == 0)  return false;  bool flag = false;  for (int i = 0; i < count; i++)  if (node[i].GetKey() == A.GetKey())  {  for (int j = i; j < count - 1; j++)  node[j] = node[j + 1];  flag = true;  break;  }  if (flag == true)  count--;  return flag;  }  //----------------------------------------------------------------------  template <typename TableSeeType>  bool TTableSee<TableSeeType>::Delete(const TString& A)  {  if (count == 0)  return false;  bool flag = false;  for (int i = 0; i < count; i++)  if (node[i].GetKey() == A)  {  for (int j = i; j < count - 1; j++)  node[j] = node[j + 1];  flag = true;  break;  }  if (flag == true)  count--;  return flag;  }  //----------------------------------------------------------------------  template <typename TableSeeType>  TElement<TableSeeType>& TTableSee<TableSeeType>::Search(const TString& A) const  {  for (int i = 0; i < count; i++)  if (node[i].GetKey() == A)  return node[i];  return st;  }  //----------------------------------------------------------------------  template <typename TableSeeType>  TableSeeType& TTableSee<TableSeeType>::operator[](const TString& A) const  {  return Search(A).GetData();  }  //----------------------------------------------------------------------  template <typename Op>  ostream& operator<<(ostream& ostr, const TTableSee<Op>& A)  {  for (int i = 0; i < A.GetCount(); i++)  ostr << A.node[i] << endl;  return ostr;  } |

## 8.5. Приложение 5:Класс TSortTable

|  |
| --- |
| **ttablesort.h** |
| #pragma once  #include "tablesee.h"  using namespace std;  #define INSERT\_SORT 1  #define SELECTION\_SORT 2  #define QUICK\_SORT 3  template <typename SortTableType>  class TSortTable  {  protected:  static TElement<SortTableType> stt;  TElement<SortTableType>\* node;  int count;  int size;  void Expansion(const int \_size);  public:  TSortTable(const int \_size = 1);  TSortTable(const TTableSee<SortTableType>& A, const int num);  TSortTable(const TSortTable<SortTableType>& A);  ~TSortTable();  int GetCount() const { return count; }  int GetSize() const { return size; }  bool Add(TElement<SortTableType>& A);  TString& Add(const SortTableType& A);  bool Delete(TElement<SortTableType>& A);  bool Delete(const TString& A);  TElement<SortTableType>& LineSearch(const TString& A) const;  TElement<SortTableType>& BinarySearch(const TString& A) const;  static void InsertSort(TTableSee<SortTableType>& A);  static void SelectionSort(TTableSee<SortTableType>& A);  static void QuickSort(TTableSee<SortTableType>& A, const int low, const int high);  SortTableType& operator[](const TString& A) const;  template <typename Type>  friend ostream& operator<<(ostream& ostr, const TSortTable<Type>& A);  };  //----------------------------------------------------------------------  template <typename SortTableType>  void TSortTable<SortTableType>::Expansion(const int \_size)  {  if ( \_size > size )  {  TElement<SortTableType>\* tmp = new TElement<SortTableType>[\_size];  for ( int i = 0; i < count; i++ )  tmp[i] = node[i];  for ( int i = count; i < \_size; i++ )  tmp[i] = stt;  size = \_size;  delete[] node;  node = tmp;  }  else  throw TException( "Error" );  }  //----------------------------------------------------------------------  template <typename SortTableType>  TElement<SortTableType> TSortTable<SortTableType>::stt;  //----------------------------------------------------------------------  template <typename SortTableType>  TSortTable<SortTableType>::TSortTable(const int \_size)  {  if ( \_size <= 0 )  throw 2;  size = \_size;  count = 0;  node = new TElement<SortTableType>[size];  for ( int i = 0; i < size; i++ )  node[i] = stt;  }  //----------------------------------------------------------------------  template <typename SortTableType>  TSortTable<SortTableType>::TSortTable(const TSortTable<SortTableType>& A)  {  size = A.size;  count = A.count;  node = new TElement<SortTableType>[size];  for ( int i = 0; i < size; i++ )  node[i] = A.node[i];  }  //----------------------------------------------------------------------  template <typename SortTableType>  TSortTable<SortTableType>::TSortTable(const TTableSee<SortTableType>& A, const int num)  {  size = A.GetSize();  count = A.GetCount();  TTableSee<SortTableType> copy\_t(A);  if (num == INSERT\_SORT)  TSortTable<SortTableType>::InsertSort(copy\_t);  else if (num == SELECTION\_SORT)  TSortTable<SortTableType>::SelectionSort(copy\_t);  else if (num == QUICK\_SORT)  TSortTable<SortTableType>::QuickSort(copy\_t, 0, copy\_t.GetCount() - 1);  node = new TElement<SortTableType>[size];  for (int i = 0; i < count; i++)  node[i] = copy\_t.GetNode()[i];  }  //----------------------------------------------------------------------  template <typename SortTableType>  TSortTable<SortTableType>::~TSortTable()  {  count = size = 0;  delete[] node;  }  //----------------------------------------------------------------------  template <typename SortTableType>  bool TSortTable<SortTableType>::Add(TElement<SortTableType>& A)  {  bool flag = false;  if ( count == 0 )  {  node[count] = A;  count++;  return true;  }  if ( count == size )  Expansion( count \* 2 );  TElement<SortTableType> tmp;  for ( int i = 0; i <= count; i++ )  {  if ( node[i].GetKey() < A.GetKey() && i != count )  continue;  else if ( node[i].GetKey() > A.GetKey() && flag == false )  {  flag = true;  tmp = node[i];  node[i] = A;  continue;  }  else if ( flag == true )  {  TElement<SortTableType> temp = node[i];  node[i] = tmp;  tmp = temp;  }  else if ( i == count && flag == false )  {  node[count] = A;  flag = true;  }  }  count++;  return flag;  }  //----------------------------------------------------------------------  template <typename SortTableType>  TString& TSortTable<SortTableType>::Add(const SortTableType& A)  {  if ( size == count )  Expansion( count \* 2 );  node[count].SetData(A);  TString string( "Key" );  if ( count == 0 )  {  node[count].SetKey(string);  count++;  return node[count - 1].GetKey();  }  else  {  TString string1( &( node[count - 1].GetKey().GetMemory()[0] ) );  string = string1;  node[count].SetKey(string);  TElement<SortTableType> temp = node[count - 1];  node[count - 1] = node[count];  node[count] = temp;  }  count++;  return node[count - 2].GetKey();  }  //----------------------------------------------------------------------  template <typename SortTableType>  bool TSortTable<SortTableType>::Delete(TElement<SortTableType>& A)  {  int left = 0;  int right = count;  int current = count / 2;  bool flag = false;  while ( right - left >= 1 )  if ( node[current].GetKey() < A.GetKey() )  {  left = current;  current = (right + left) / 2;  }  else if ( node[current].GetKey() > A.GetKey() )  {  right = current;  current = (right - left) / 2;  }  else if ( node[current].GetKey() == A.GetKey() )  {  for ( int i = current; i < count - 1; i++ )  node[i] = node[i + 1];  count--;  flag = true;  break;  }  return flag;  }  //----------------------------------------------------------------------  template <typename SortTableType>  bool TSortTable<SortTableType>::Delete(const TString& A)  {  int left = 0;  int right = count;  int current = count / 2;  bool flag = false;  while ( right - left >= 1 )  if ( node[current].GetKey() < A )  {  left = current;  current = (right + left) / 2;  }  else if (node[current].GetKey() > A )  {  right = current;  current = (right - left) / 2;  }  else if ( node[current].GetKey() == A )  {  for ( int i = current; i < count - 1; i++ )  node[i] = node[i + 1];  count--;  flag = true;  break;  }  return flag;  }  //----------------------------------------------------------------------  template <typename SortTableType>  TElement<SortTableType>& TSortTable<SortTableType>::LineSearch(const TString& A) const  {  for ( int current = 0; current < count; current++ )  {  if ( node[current].GetKey() == A )  return node[current];  }  return stt;  }  //----------------------------------------------------------------------  template <typename SortTableType>  TElement<SortTableType>& TSortTable<SortTableType>::BinarySearch(const TString& A) const  {  int left = 0;  int right = count;  int cur = count / 2;  int tmp = count / 2;  while (right - left >= 1)  {  if (node[cur].GetKey() < A)  {  left = cur;  cur = (right + left) / 2;  }  else if (node[cur].GetKey() > A)  {  right = cur;  cur = (right - left) / 2;  }  else if (node[cur].GetKey() == A)  return node[cur];  if (right - left == 1)  break;  }  return stt;  }  //----------------------------------------------------------------------  template <typename SortTableType>  void TSortTable<SortTableType>::InsertSort(TTableSee<SortTableType>& A)  {  for ( int i = 1; i < A.GetCount(); i++ )  {  TElement<SortTableType> temp = A.GetNode()[i];  int j = i;  while ( j > 0 && A.GetNode()[j - 1].GetKey() > temp.GetKey() )  {  A.GetNode()[j] = A.GetNode()[j - 1];  j--;  }  A.GetNode()[j] = temp;  }  }  //----------------------------------------------------------------------  template <typename SortTableType>  void TSortTable<SortTableType>::QuickSort(TTableSee<SortTableType>& A, const int low, const int high)  {  int i = low, j = high;  TElement<SortTableType> d;  TElement<SortTableType> m = A.GetNode()[(low + high) / 2];  while (i <= j)  {  for (; A.GetNode()[i].GetKey() < m.GetKey(); i++);  for (; A.GetNode()[j].GetKey() > m.GetKey(); j--);  if (i <= j)  {  d = A.GetNode()[i];  A.GetNode()[i++] = A.GetNode()[j];  A.GetNode()[j--] = d;  }  }  if (low < j)  QuickSort(A, low, j);  if (i < high)  QuickSort(A, i, high);  }  //----------------------------------------------------------------------  template <typename SortTableType>  void TSortTable<SortTableType>::SelectionSort(TTableSee<SortTableType>& A)  {  TElement<SortTableType> temp;  int min;  for ( int i = 0; i < A.GetCount() -1 ; i++ )  {  min = i;  for ( int j = i + 1; j < A.GetCount(); j++ )  {  if (A.GetNode()[j].GetKey() < A.GetNode()[min].GetKey())  min = j;  }  temp = A.GetNode()[i];  A.GetNode()[i] = A.GetNode()[min];  A.GetNode()[min] = temp;  }  }  //----------------------------------------------------------------------  template <typename SortTableType>  SortTableType& TSortTable<SortTableType>::operator[](const TString& A) const  {  return LineSearch(A).GetData();  }  //----------------------------------------------------------------------  template <typename Type>  ostream& operator<<(ostream& ostr, const TSortTable<Type>& A)  {  for ( int i = 0; i < A.GetCount(); i++ )  ostr << A.node[i] << endl;  return ostr;  } |

## 8.6. Приложение 6:Код программы тестирования

|  |
| --- |
| **sample\_tablesee.cpp** |
| #include <iostream>  #include <locale.h>  #include "ttablesort.h"  using namespace std;  int main()  {  int data;  int num;  TElement<int> element;  char memory[100];  cout << "EXAMPLE: SORT TABLE"<< endl;  cout << "Max count of elements: 6";  TTableSee<int> look(3);  TString string1("str1");  TString string2("str2");  TString string3("str3");  TString string4("str4");  TString string7("str7");  TString string6("str6");  TElement<int> element1(1, string1);  TElement<int> element2(2, string2);  TElement<int> element3(3, string3);  TElement<int> element4(4, string4);  TElement<int> element7(7, string7);  TElement<int> element6(6, string6);  look.Add(element3);  look.Add(element1);  look.Add(element2);  look.Add(element4);  look.Add(element7);  look.Add(element6);  cout << endl << look;  cout << "SORT OUR TABLE\n";  cout << " 1.INSERT\_SORT \n 2.SELECTION\_SORT \n 3.QUICK\_SORT \n";  cout << "Number of Sorting: ";  cin >> num;  TSortTable<int> sort(look, num);  cout << endl << sort;  cout << "ADD ELEMENT\n ";  cout << "Key of new element: ";  cin >> memory;  TString key1(memory);  element.SetKey(key1);  cout << " Data of new element: ";  cin >> data;  element.SetData(data);  sort.Add(element);  cout << endl << sort;  cout << "DELETE ELEMENT\n ";  cout << " Key of element for delete: ";  cin >> memory;  TString key2(memory);  sort.Delete(key2);  cout << endl << sort;  return 0;  } |

## 8.7. Приложение 7:Тесты для классов

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

|  |
| --- |
| **test\_tablesee.cpp** |
| #include "tablesee.h"  #include <gtest.h>  //TESTS FOR CLASS TString  TEST(TString, can\_create\_string\_without\_symbols)  {  ASSERT\_NO\_THROW(TString string);  }  TEST(TString, can\_create\_string\_with\_symbols)  {  ASSERT\_NO\_THROW(TString string("Program"));  }  TEST(TString, can\_create\_copy\_of\_string)  {  TString string("Program");  TString string2(string);  EXPECT\_EQ(string.GetMemory()[0], 'P');  EXPECT\_EQ(string.GetMemory()[6], 'm');  }  TEST(TString, can\_get\_count)  {  TString string("Program");  EXPECT\_EQ(string.GetCount(), 7);  }  TEST(TString, can\_get\_memory)  {  TString string("Program");  ASSERT\_NO\_THROW(string.GetMemory());  EXPECT\_EQ(string.GetMemory()[0], 'P');  EXPECT\_EQ(string.GetMemory()[6], 'm');  }  TEST(TString, can\_use\_assign\_operator)  {  TString string("Program");  TString string2;  string2 = string;  EXPECT\_EQ(string.GetMemory()[0], 'P');  EXPECT\_EQ(string.GetMemory()[6], 'm');  }  TEST(TString, can\_compare\_strings\_part1)  {  TString string("One");  TString string2("Two");  TString string3("One");  EXPECT\_TRUE(string == string3);  EXPECT\_TRUE(string2 != string3);  EXPECT\_FALSE(string == string2);  EXPECT\_FALSE(string != string3);  }  TEST(TString, can\_compare\_strings\_part2)  {  TString string("AAA");  TString string2("AAAAAAA");  EXPECT\_TRUE(string < string2);  EXPECT\_FALSE(string > string2);  EXPECT\_TRUE(string2 > string);  EXPECT\_FALSE(string2 < string);  }  TEST(TString, throw\_when\_use\_wrong\_index\_of\_element)  {  TString string("Program");  ASSERT\_ANY\_THROW(string[-1]);  ASSERT\_ANY\_THROW(string[10]);  }  TEST(TString, can\_return\_element)  {  TString string("Program");  EXPECT\_EQ(string[0], 'P');  EXPECT\_EQ(string[6], 'm');  }  //TESTS FOR CLASS TElement  TEST(TElement, can\_create\_element\_without\_parametres)  {  ASSERT\_NO\_THROW(TElement<int> elem);  TElement<int> element;  EXPECT\_EQ(element.GetData(), 0);  EXPECT\_EQ(element.GetKey(), " ");  }  TEST(TElement, can\_create\_element\_with\_parametres)  {  TString string("new");  ASSERT\_NO\_THROW(TElement<int> element(1, string));  }  TEST(TElement, can\_create\_copy\_of\_element)  {  TString string("new");  TElement<int> element(1, string);  TElement<int> element2(element);  EXPECT\_EQ(element.GetData(), 1);  EXPECT\_EQ(element.GetKey(), "new");  }  TEST(TElement, can\_use\_assign\_operator)  {  TString string("new");  TElement<int> element(1, string);  TElement<int> element2;  element2 = element;  EXPECT\_EQ(element.GetData(), 1);  EXPECT\_EQ(element.GetKey(), "new");  }  TEST(TElement, can\_get\_data)  {  TString string("new");  TElement<int> element(1, string);  EXPECT\_EQ(element.GetData(), 1);  }  TEST(TElement, can\_set\_data)  {  TString string("new");  TElement<int> element(1, string);  element.SetData(2);  EXPECT\_EQ(element.GetData(), 2);  }  TEST(TElement, can\_get\_key)  {  TString string("new");  TElement<int> element(1, string);  EXPECT\_EQ(element.GetKey(), "new");  }  TEST(TElement, can\_set\_key)  {  TString string("old");  TString string1("new");  TElement<int> element(1, string);  element.SetKey(string1);  EXPECT\_EQ(element.GetKey(), string1);  }  TEST(TElement, can\_compare\_elements)  {  TString string("old");  TString string1("old");  TString string2("new");  TElement<int> element(1, string);  TElement<int> element1(1, string1);  TElement<int> element2(2, string2);  EXPECT\_TRUE(element == element1);  EXPECT\_FALSE(element == element2);  EXPECT\_TRUE(element != element2);  EXPECT\_FALSE(element != element1);  }  //TESTS FOR CLASS TTableSee  TEST(TTableSee, can\_create\_table\_without\_parametres)  {  TTableSee<int> table;  EXPECT\_EQ(table.GetSize(), 1);  }  TEST(TTableSee, can\_create\_table\_with\_parametres)  {  TTableSee<int> table(10);  EXPECT\_EQ(table.GetSize(), 10);  }  TEST(TTableSee, can\_create\_copy\_of\_table)  {  TTableSee<int> table(10);  TTableSee<int> table2(table);  EXPECT\_EQ(table2.GetSize(), 10);  }  TEST(TTableSee, can\_get\_size)  {  TTableSee<int> table(10);  EXPECT\_EQ(table.GetSize(), 10);  }  TEST(TTableSee, can\_get\_count)  {  TTableSee<int> table(10);  EXPECT\_EQ(table.GetCount(), 0);  TString string("new");  TElement<int> element(1, string);  table.Add(element);  EXPECT\_EQ(table.GetCount(), 1);  }  TEST(TTableSee, can\_add\_element)  {  TString string("new");  TElement<int> element(1, string);  TTableSee<int> table(10);  table.Add(element);  EXPECT\_EQ(table[string], 1);  }  TEST(TTableSee, can\_use\_resize)  {  TString string("one");  TElement<int> element(1, string);  TString string2("two");  TElement<int> element2(1, string2);  TTableSee<int> table(1);  table.Add(element);  ASSERT\_NO\_THROW(table.Add(element2));  }  TEST(TTableSee, can\_use\_operator\_for\_index)  {  TString string("new");  TElement<int> element(1, string);  TTableSee<int> table(10);  table.Add(element);  EXPECT\_EQ(table[string], 1);  }  TEST(TTableSee, can\_search\_element)  {  TString string("one");  TElement<int> element(1, string);  TString string2("two");  TElement<int> element2(1, string2);  TTableSee<int> table(1);  table.Add(element);  table.Add(element2);  EXPECT\_TRUE(table.Search(string) == element);  }  TEST(TTableSee, can\_add\_element\_without\_key)  {  TTableSee<int> table(5);  TString string = table.Add(15);  EXPECT\_EQ(string.GetMemory()[0], 'K');  EXPECT\_EQ(string.GetMemory()[1], 'e');  EXPECT\_EQ(string.GetMemory()[2], 'y');  TString string2 = "Key";  TElement<int> element2(15, string2);  EXPECT\_TRUE(table.Search(string) == element2);  }  TEST(TTableSee, cad\_delete\_element\_by\_two\_ways)  {  TString string("one");  TElement<int> element(1, string);  TString string2("two");  TElement<int> element2(2, string2);  TString string3("three");  TElement<int> element3(3, string3);  TTableSee<int> table(3);  table.Add(element);  table.Add(element2);  table.Add(element3);  table.Delete(element3);  table.Delete(string2);  EXPECT\_EQ(table.GetCount(), 1);  }  TEST(TTableSee, can\_not\_delete\_element\_from\_empty\_table)  {  TTableSee<int> table(3);  TString string("one");  TElement<int> element(1, string);  EXPECT\_FALSE(table.Delete(element));  EXPECT\_FALSE(table.Delete(string));  }  TEST(TTableSee, can\_delete\_element\_from\_not\_empty\_table)  {  TTableSee<int> table(3);  TString string("one");  TElement<int> element(1, string);  table.Add(element);  TString string2("two");  TElement<int> element2(2, string2);  table.Add(element2);  EXPECT\_TRUE(table.Delete(element));  EXPECT\_TRUE(table.Delete(string2));  }  //TESTS FOR CLASS TTableSort  TEST(TSortTable, can\_create\_table\_without\_parametres)  {  TSortTable<int> table;  EXPECT\_EQ(table.GetSize(), 1);  }  TEST(TSortTable, can\_create\_table\_with\_parametres)  {  TSortTable<int> table(10);  EXPECT\_EQ(table.GetSize(), 10);  }  TEST(TSortTable, can\_use\_resize)  {  TString string("one");  TElement<int> element(1, string);  TString string2("two");  TElement<int> element2(1, string2);  TSortTable<int> table(1);  table.Add(element);  ASSERT\_NO\_THROW(table.Add(element2));  }  TEST(TSortTable, can\_create\_sort\_table\_using\_different\_sortings)  {  TTableSee<int> table(3);  TString string("one");  TElement<int> element(1, string);  TString string2("two");  TElement<int> element2(2, string2);  TString string3("three");  TElement<int> element3(3, string3);  table.Add(element3);  table.Add(element);  table.Add(element2);  ASSERT\_NO\_THROW(TSortTable<int> temp(table, 1));  ASSERT\_NO\_THROW(TSortTable<int> temp1(table, 2));  ASSERT\_NO\_THROW(TSortTable<int> temp2(table, 3));  }  TEST(TSortTable, can\_create\_copy\_of\_table)  {  TSortTable<int> table(10);  TSortTable<int> table2(table);  EXPECT\_EQ(table2.GetSize(), 10);  }  TEST(TSortTable, can\_get\_size)  {  TSortTable<int> table(10);  EXPECT\_EQ(table.GetSize(), 10);  }  TEST(TSortTable, can\_get\_count)  {  TSortTable<int> table(10);  EXPECT\_EQ(table.GetCount(), 0);  TString string("new");  TElement<int> element(1, string);  table.Add(element);  EXPECT\_EQ(table.GetCount(), 1);  }  TEST(TSortTable, can\_add\_element)  {  TSortTable<int> table(10);  TString string("One");  TElement<int> element(1, string);  table.Add(element);  EXPECT\_EQ(table[string], 1);  }  TEST(TSortTable, can\_add\_element\_without\_key)  {  TSortTable<int> table(5);  TString string = table.Add(15);  EXPECT\_EQ(string.GetMemory()[0], 'K');  EXPECT\_EQ(string.GetMemory()[1], 'e');  EXPECT\_EQ(string.GetMemory()[2], 'y');  TString string2 = "Key";  TElement<int> element2(15, string2);  EXPECT\_TRUE(table.LineSearch(string) == element2);  }  TEST(TSortTable, cad\_delete\_element\_by\_two\_ways)  {  TString string("one");  TElement<int> element(1, string);  TString string2("two");  TElement<int> element2(2, string2);  TString string3("three");  TElement<int> element3(3, string3);  TSortTable<int> table(3);  table.Add(element);  table.Add(element2);  table.Add(element3);  table.Delete(element3);  table.Delete(string2);  EXPECT\_EQ(table.GetCount(), 1);  }  TEST(TSortTable, can\_not\_delete\_element\_from\_empty\_table)  {  TSortTable<int> table(3);  TString string("one");  TElement<int> element(1, string);  EXPECT\_FALSE(table.Delete(element));  EXPECT\_FALSE(table.Delete(string));  }  TEST(TSortTable, can\_delete\_element\_from\_not\_empty\_table)  {  TSortTable<int> table(3);  TString string("one");  TElement<int> element(1, string);  table.Add(element);  TString string2("two");  TElement<int> element2(2, string2);  table.Add(element2);  EXPECT\_TRUE(table.Delete(element));  EXPECT\_TRUE(table.Delete(string2));  }  TEST(TSortTable, can\_use\_operator\_for\_index)  {  TString string("new");  TElement<int> element(1, string);  TSortTable<int> table(10);  table.Add(element);  EXPECT\_EQ(table[string], 1);  }  TEST(TSortTable, can\_search\_element)  {  TString string("one");  TElement<int> element(1, string);  TString string2("two");  TElement<int> element2(2, string2);  TString string3("three");  TElement<int> element3(3, string3);  TSortTable<int> table(1);  table.Add(element);  table.Add(element2);  table.Add(element3);  EXPECT\_TRUE(table.LineSearch(string2) == element2);  EXPECT\_TRUE(table.BinarySearch(string2) == element2);  }  TEST(TSortTable, can\_sort\_table\_by\_insert\_sort)  {  TTableSee<int> table(3);  TString string("one");  TElement<int> element(1, string);  TString string2("two");  TElement<int> element2(2, string2);  TString string3("three");  TElement<int> element3(3, string3);  table.Add(element3);  table.Add(element);  table.Add(element2);  TSortTable<int>::InsertSort(table);  EXPECT\_TRUE(table.GetNode()[0].GetKey() == "one");  }  TEST(TSortTable, can\_sort\_seetable\_by\_selection\_sort)  {  TTableSee<int> table(3);  TString string("one");  TElement<int> element(1, string);  TString string2("two");  TElement<int> element2(2, string2);  TString string3("three");  TElement<int> element3(3, string3);  table.Add(element3);  table.Add(element);  table.Add(element2);  TSortTable<int>::SelectionSort(table);  EXPECT\_TRUE(table.GetNode()[0].GetKey() == "one");  }  TEST(TSortTable, can\_sort\_seetable\_by\_quick\_sort)  {  TTableSee<int> table(3);  TString string("one");  TElement<int> element(1, string);  TString string2("two");  TElement<int> element2(2, string2);  TString string3("three");  TElement<int> element3(3, string3);  table.Add(element3);  table.Add(element);  table.Add(element2);  TSortTable<int>::QuickSort(table, 0, table.GetCount() - 1);  EXPECT\_TRUE(table.GetNode()[0].GetKey() == "one");  } |