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 №16**

**«Хеш таблица с разрешением коллизий»**

**Выполнил:** студент группы 381706-2

Жбанова Надежда Сергеевна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Руководитель:**

Ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2019

Содержание

[1.Введение 3](#_Toc9860279)

[Простое представление хеш-таблиц 3](#_Toc9860280)

[2. Цели и задачи 6](#_Toc9860281)

[2.1. Используемые инструменты 6](#_Toc9860282)

[3. Руководство пользователя 8](#_Toc9860283)

[4. Руководство программиста 10](#_Toc9860284)

[4.1. Описание структуры программы 10](#_Toc9860285)

[4.2. Описание функций и процедур, их алгоритмов 10](#_Toc9860286)

[6. Заключение 14](#_Toc9860287)

[7. Литература 15](#_Toc9860288)

[8. Приложения 16](#_Toc9860289)

[8.1. Приложение 1:Класс TExсeption 16](#_Toc9860290)

[8.2. Приложение 2:Класс TString 16](#_Toc9860291)

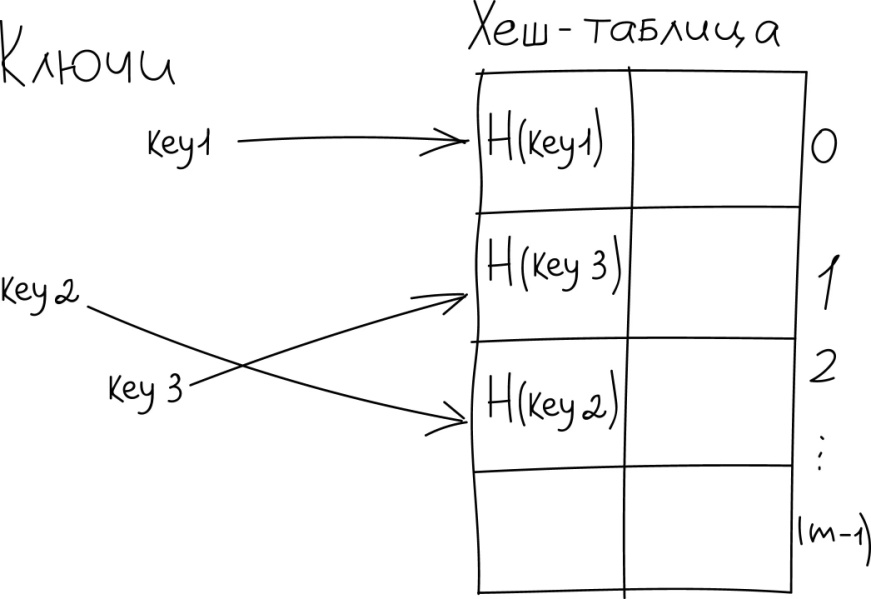
[8.3. Приложение 3:Класс TElement 18](#_Toc9860292)

[8.4. Приложение 4:Класс THashTable 20](#_Toc9860293)

[8.5. Приложение 5:Код программы тестирования 22](#_Toc9860294)

[8.6. Приложение 6:Тесты для классов 23](#_Toc9860295)

# 1.Введение

Хеш-таблица (hash table) — это специальная структура данных для хранения пар ключей и их значений. По сути это ассоциативный массив, в котором ключ представлен в виде хеш-функции.

Пожалуй, главное свойство hash-таблиц — все три операции вставка, поиск и удаление в среднем выполняются за время O(1), среднее время поиска по ней также равно O(1) и O(n) в худшем случае.

## Простое представление хеш-таблиц

Чтобы разобраться, что такое хеш-таблицы, представьте, что вас попросили создать библиотеку и заполнить ее книгами. Но вы не хотите заполнять шкафы в произвольном порядке.

Первое, что приходит в голову — разместить все книги в алфавитном порядке и записать все в некий справочник. В этом случае не придется искать нужную книгу по всей библиотеке, а только по справочнику.

А можно сделать еще удобнее. Если изначально отталкиваться от названия книги или имени автора, то лучше использовать некий алгоритм хеширования, который обрабатывает входящее значение и выдает номер шкафа и полки для нужной книги.

Зная этот алгоритм хэширования, вы быстро найдете нужную книгу по ее названию.

Хеш-функция должна иметь следующие свойства:

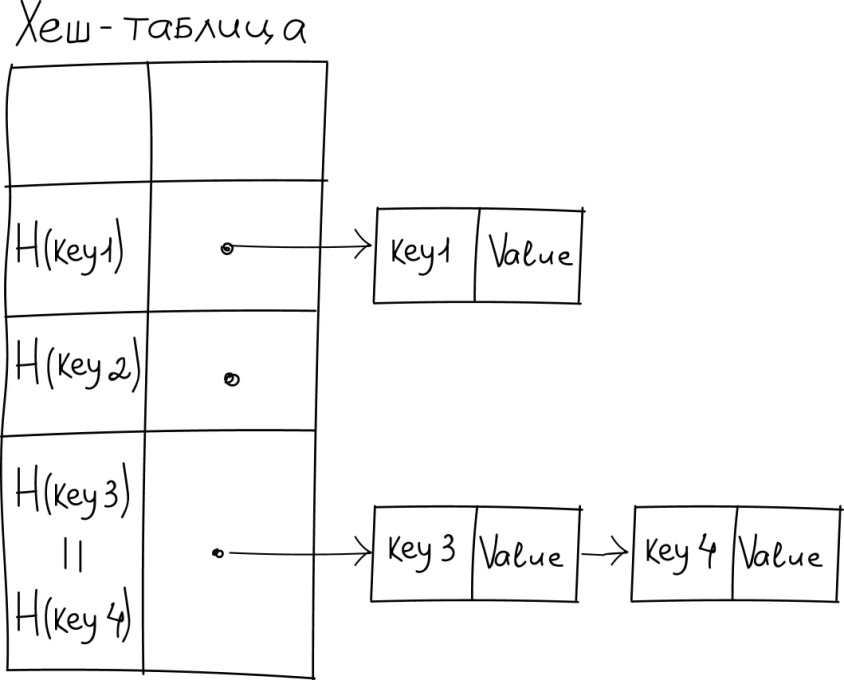
* Всегда возвращать один и тот же адрес для одного и того же ключа;
* Не обязательно возвращает разные адреса для разных ключей;
* Использует все адресное пространство с одинаковой вероятностью;
* Быстро вычислять адрес.

Борьба с коллизиями (они же столкновения)

В идеальном случае, когда заранее известны все пары ключ-значение, достаточно легко реализовать идеальную хеш-таблицу, в которой время поиска будет постоянным (используется идеальная хеш-функция, которая определяет положения в таблице по целым значениям и без столкновений).

Но в большинстве случаев приходится бороться с коллизиями. Обычно применяются методы цепочек и открытой индексации.

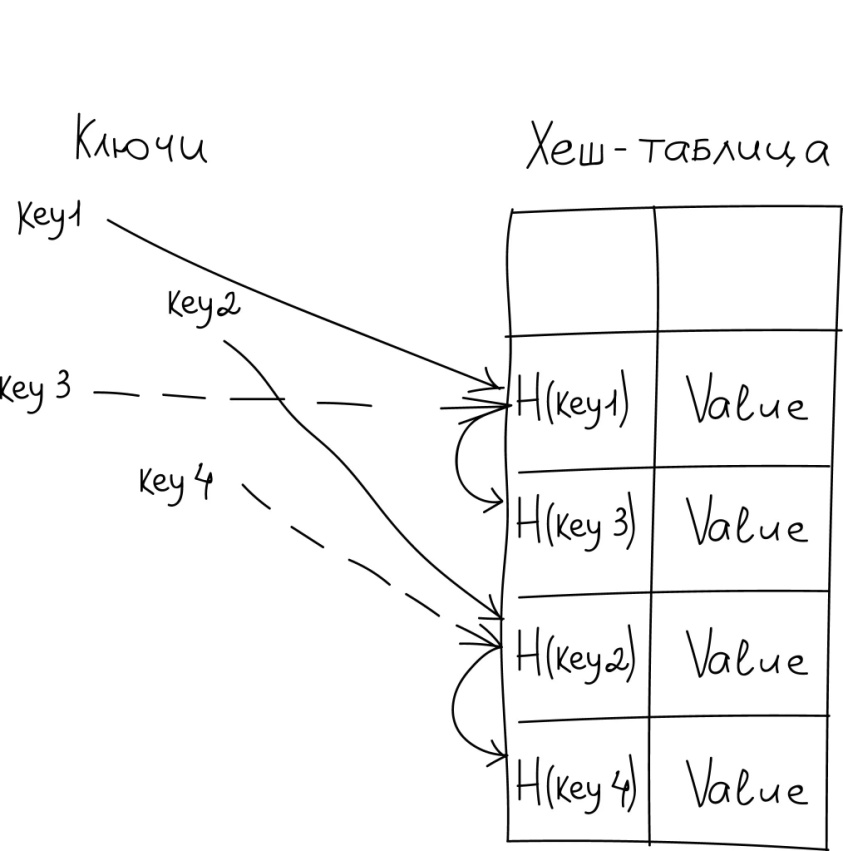
Метод цепочек

Этот метод часто называют **открытым хешированием**. Его суть проста — элементы с одинаковым хешем попадают в одну ячейку в виде связного списка.

То есть, если ячейка с хешем уже занята, но новый ключ отличается от уже имеющегося, то новый элемент вставляется в список в виде пары ключ-значение.

Если выбран метод цепочек, то вставка нового элемента происходит за O(1), а время поиска зависит от длины списка и в худшем случае равно O(n). Если количество ключей n , а распределяем по m -ячейкам, то соотношение n/m будет коэффициентом заполнения.

Открытая индексация (или закрытое хеширование)

Второй распространенный метод — открытая индексация. Это значит, что пары ключ-значение хранятся непосредственно в хеш-таблице. А алгоритм вставки проверяет ячейки в некотором порядке, пока не будет найдена пустая ячейка. 

Порядок вычисляется на лету.

Самая простая в реализации последовательность проб — линейное пробирование (или линейное исследование). Здесь все просто — в случае коллизии, следующие ячейки проверяются линейно, пока не будет найдена пустая ячейка.

А алгоритм поиска ищет ячейки в том же порядке, что и при вставке, пока не найдет нужный элемент или пустую ячейку, которая говорит о том, что ключ отсутствует. В случае, если таблица будет заполнена, ее придется динамически расширять.

# 2. Цели и задачи

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

* инициализация
* добавление/удаление элемента
* поиск элемента
* разрешение коллизий

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 THashTable согласно заданному интерфейсу.
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).

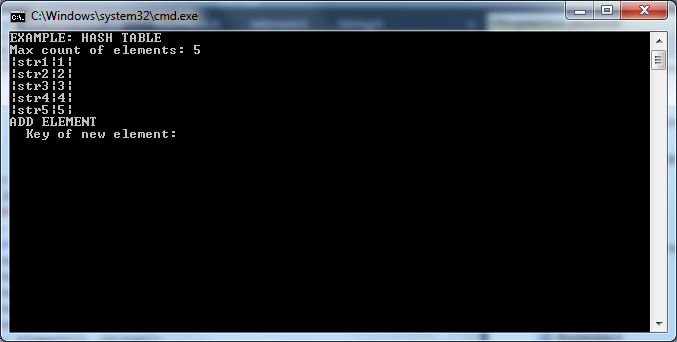


Рис.1.Пример таблицы.

На консоль выводится пример хеш-таблицы. Пользователю предлагается добавить новый элемент в таблицу, указав ключ и значение. (Рис.2):

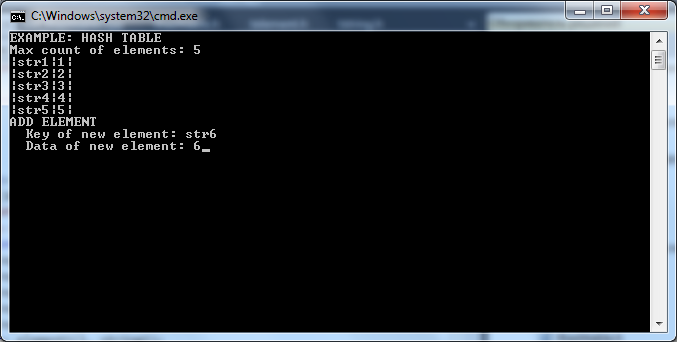


Рис.2. Добавление элемента..

Теперь пользователю нужно ввести ключ элемента, который следует удалить (Рис.3).

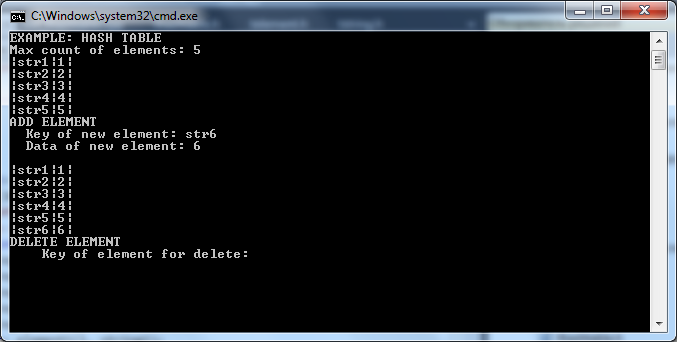


Рис.3.Удаление элемента.

Результат операций автоматически выведется на экран (Рис.4).

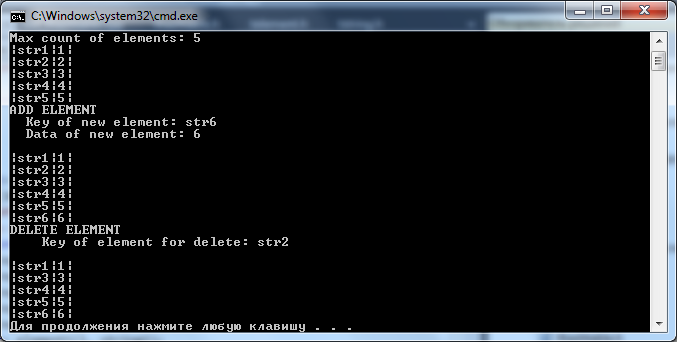


Рис.4.Результат.

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 «Хэш-таблица» (THashTable).
* Класс исключения (TExсeption).

## 4.1. Описание структуры программы

Модульная структура программы:

1. thashtable.h– модуль с классом THashTable, в котором определен интерфейс шаблонного класса «Хэш-таблица» и реализация его методов.
2. telement.h– модуль с классом TElement, в котором определен интерфейс шаблонного класса «Элемент таблицы» и реализация его методов.
3. tstring.h– модуль с классом TString, в котором определен интерфейс класса «Строка» и реализация его методов.
4. exсeption.h – модуль с классом исключения TExсeption.
5. sample\_thashtable.cpp – модуль программы тестирования, с которым работает пользователь.
6. test\_main.cpp, test\_thashtable.cpp – модуль с функциями тестирования для созданных классов. Содержат 10 тестов для класса TString, 9 - для класса TElement и 12 - для класса THashTable.

## 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 THashTable:

template <typename HashTableType>

class THashTable

static TElement<HashTableType> tht - первоначальный пустой элемент таблицы.

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

int count - кол-во используемых элементов таблицы.

int size – максимальное кол-во элементов таблицы.

int m – шаг обхода массива элементов таблицы.

1. void Expansion(int \_size) – защищенный метод, увеличивающий максимальный размер таблицы.

2. int Function(TString& A) – защищенный метод хэш-функции, возвращающей целочисленное число.

Принимает ссылку на объект класса TString.

3. THashTable(const int \_size = 1) – конструктор, принимающий кол-во элементов таблицы.

По умолчанию кол-во элементов таблицы = 1.

4. THashTable(const THashTable<HashTableType>& A) - конструктор копирования.

Принимает ссылку на объект класса THashTable.

5. ~THashTable() - деструктор.

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

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

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

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

В качестве параметров принимает ключ и значение элемента.

10. void Add(TElement<HashTableType>& A) - метод, добавляющий элемент в таблицу.

В качестве параметра принимает ссылку на объект класса TElement.

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

В качестве аргумента принимает его ключ.

12. HashTableType& Search(TString& A) - метод, осуществляющий поиск элемента в таблице по его ключу.

13. bool IsSimple(const int num) – метод, проверяющий простое ли число.

Если число простое - возвращает true, иначе false.

14. template <typename Type>

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

# 

# 6. Заключение

В результате лабораторной работы была разработана структура данных – хэш - 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. <https://miassats.ru/5437/>

# 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 THashTable

|  |
| --- |
| **thashtable.h** |
| #pragma once  #include "telement.h"  using namespace std;  template <typename HashTableType>  class THashTable  {  protected:  static TElement<HashTableType> tht;  TElement<HashTableType>\* node;  int count;  int size;  int m;  void Expansion(int \_size);  int Function(TString& A);  public:  THashTable(const int \_size = 1);  THashTable(const THashTable<HashTableType>& A);  ~THashTable();  int GetCount() const { return count; }  int GetSize() const { return size; }  TElement<HashTableType>\* GetNode() { return node; }  void Add(TString& A, const HashTableType& H);  void Add(TElement<HashTableType>& A);  bool Delete(TString& A);  HashTableType& Search(TString& A);  bool IsSimple(const int num);  template <typename Type>  friend ostream& operator<<(ostream& ostr, const THashTable<Type>& A)  {  for (int i = 0; i < A.size; i++)  if (A.node[i] != tht)  ostr << A.node[i] << endl;  return ostr;  }  };  //----------------------------------------------------------------------  template <typename HashTableType>  TElement<HashTableType> THashTable<HashTableType>::tht;  //----------------------------------------------------------------------  template <typename HashTableType>  int THashTable<HashTableType>::Function(TString& A)  {  unsigned int n = 0;  for (int i = 0; i < A.GetCount(); i++)  n = (n >> 1) + A[i];  return n;  }  //----------------------------------------------------------------------  template <typename HashTableType>  void THashTable<HashTableType>::Expansion(int \_size)  {  if(\_size <= size)  throw TException("Error");  else if(\_size > size)  {  while (IsSimple(\_size) == 0)  \_size++;  TElement<HashTableType>\* tmp = new TElement<HashTableType>[\_size];  for (int i = 0; i < size; i++)  tmp[i] = node[i];  for (int i = size; i < \_size; i++)  tmp[i] = tht;  size = \_size;  delete[] node;  node = tmp;  }  }  //----------------------------------------------------------------------  template <typename HashTableType>  THashTable<HashTableType>::THashTable(const int \_size)  {  if (\_size <= 0)  throw TException("Error");  count = 0;  size = \_size;  node = new TElement<HashTableType>[size];  for (int i = 0; i < size; i++)  node[i] = tht;  }  //----------------------------------------------------------------------  template <typename HashTableType>  THashTable<HashTableType>::THashTable(const THashTable<HashTableType>& A)  {  count = A.count;  size = A.size;  node = new TElement<HashTableType>[size];  for (int i = 0; i < size; i++)  node[i] = A.node[i];  }  //----------------------------------------------------------------------  template <typename HashTableType>  THashTable<HashTableType>::~THashTable()  {  count = 0;  size = 0;  delete[] node;  }  //----------------------------------------------------------------------  template <typename HashTableType>  void THashTable<HashTableType>::Add(TString& A, const HashTableType& H)  {  m = 2;  if (count == size)  Expansion(count \* 2);  int i = Function(A);  if (i > size)  Expansion(i + 10);  while (node[i] != tht)  i = (i + m) % size;  node[i].SetKey(A);  node[i].SetData(H);  count++;  }  //----------------------------------------------------------------------  template <typename HashTableType>  void THashTable<HashTableType>::Add(TElement<HashTableType>& A)  {  if (count == size)  Expansion(count \* 2);  int i = Function(A.GetKey());  if (i > size)  Expansion(i + 10);  while (node[i] != tht)  i = (i + m) % size;  node[i] = A;  count++;  }  //----------------------------------------------------------------------  template <typename HashTableType>  bool THashTable<HashTableType>::Delete(TString& A)  {  m = 2;  int i = Function(A);  while (node[i].GetKey() != A)  {  if (node[i] == tht)  return false;  i = (i + m) % size;  }  node[i] = tht;  return true;  }  //----------------------------------------------------------------------  template <typename HashTableType>  HashTableType& THashTable<HashTableType>::Search(TString& A)  {  m = 2;  int i = Function(A);  while (node[i].GetKey() != A)  {  i = (i + m) % size;  if (node[i] == tht)  break;  }  return node[i].GetData();  }  //----------------------------------------------------------------------  template <typename HashTableType>  bool THashTable<HashTableType>::IsSimple(const int num)  {  for (int i = 2; i < num / 2; i++)  if (num%i == 0) return false;  return true;  } |

## 8.5. Приложение 5:Код программы тестирования

|  |
| --- |
| **sample\_tablesee.cpp** |
| #include "thashtable.h"  using namespace std;  int main()  {  int data;  TElement<int> element;  char memory[100];  cout << "EXAMPLE: HASH TABLE"<< endl;  cout << "Max count of elements: 5";  THashTable<int> look(3);  TString string1("str1");  TString string2("str2");  TString string3("str3");  TString string4("str4");  TString string5("str5");  TElement<int> element1(1, string1);  TElement<int> element2(2, string2);  TElement<int> element3(3, string3);  TElement<int> element4(4, string4);  TElement<int> element5(5, string5);  look.Add(element3);  look.Add(element1);  look.Add(element2);  look.Add(element4);  look.Add(element5);  cout << endl << look;  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);  look.Add(element);  cout << endl << look;  cout << "DELETE ELEMENT\n ";  cout << " Key of element for delete: ";  cin >> memory;  TString key2(memory);  look.Delete(key2);  cout << endl << look;  return 0;  } |

## 8.6. Приложение 6: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 THashTable  TEST(THashTable, can\_create\_hashtable\_without\_parametres)  {  THashTable<int> table;  EXPECT\_EQ(table.GetSize(), 1);  }  TEST(THashTable, can\_create\_hashtable\_with\_parametres)  {  THashTable<int> table(10);  EXPECT\_EQ(table.GetSize(), 10);  }  TEST(THashTable, can\_create\_copy\_of\_table)  {  THashTable<int> table(10);  THashTable<int> table2(table);  EXPECT\_EQ(table2.GetSize(), 10);  }  TEST(THashTable, can\_get\_size)  {  THashTable<int> table(10);  EXPECT\_EQ(table.GetSize(), 10);  }  TEST(THashTable, can\_get\_count)  {  THashTable<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(THashTable, can\_get\_node)  {  THashTable<int> table(10);  EXPECT\_NO\_THROW(table.GetNode());  }  TEST(THashTable, can\_add\_element\_1)  {  TString string("new");  TElement<int> element(1, string);  THashTable<int> table(10);  EXPECT\_NO\_THROW(table.Add(element));  }  TEST(THashTable, can\_add\_element\_2)  {  TString string("new");  THashTable<int> table(10);  EXPECT\_NO\_THROW(table.Add(string, 5));  }  TEST(THashTable, can\_use\_resize)  {  TString string("one");  TElement<int> element(1, string);  TString string2("two");  TElement<int> element2(1, string2);  THashTable<int> table(1);  table.Add(element);  ASSERT\_NO\_THROW(table.Add(element2));  }  TEST(THashTable, can\_search\_element)  {  TString string("one");  TElement<int> element(1, string);  TString string2("two");  TElement<int> element2(2, string2);  THashTable<int> table(1);  table.Add(element);  table.Add(element2);  EXPECT\_EQ(table.Search(string), 1);  }  TEST(THashTable, can\_not\_delete\_element\_from\_empty\_table)  {  THashTable<int> table(3);  TString string("one");  TElement<int> element(1, string);  EXPECT\_FALSE(table.Delete(string));  }  TEST(THashTable, can\_delete\_element\_from\_not\_empty\_table)  {  THashTable<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(string2));  } |