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 №13**

**«Просмотровая таблица»**

**Выполнил:** студент группы 381706-2

Макарихин Семён Андреевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Руководитель:**

Ассистент кафедры МОСТ

Лебедев Илья Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2019

Содержание

[1.Введение 3](#_Toc9348620)

[2. Цели и задачи 4](#_Toc9348621)

[3. Руководство программиста](#_Toc9348624) 5

[3.1. Описание структуры программы](#_Toc9348625) 5

[3.2. Описание функций и процедур, их алгоритмов](#_Toc9348626) 5

[4. Заключение 1](#_Toc9348627)0

[5. Литература 1](#_Toc9348628)1

[6. Приложение 1](#_Toc9348629)2

# 1.Введение

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 множество видов таблиц, рассмотрим основные: **неупорядоченная, упорядоченная и хеш-таблица.**

# 2. Цели и задачи

Целью лабораторной работы является создание структуры хранения типа «Просмотровая таблица» и методов работы с ним, таких как:

* Добавление элементов в таблицу;
* Удаление элементов из таблицы;
* Получение доступа к элементу.

Для реализации алгоритмов будет использоваться 4 класса:

* TString;
* TTableSee;
* TExсeption;
* TElement.

Для проверки правильности работы этих классов будут написаны тесты с использованием фреймворка Google Test, а также тестовый образец программы, которая будет использует класс список.**4. Руководство программиста**

Разработка системы вычисления проводились в среде “Microsoft Visual Studio 2017”.

В данной работе будет использовано несколько классов:

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

## 4.1. Описание структуры программы

Модульная структура программы:

1. tablesee.h– модуль с классом TTableSee, в котором определен интерфейс шаблонного класса «Просмотровая таблица» и реализация его методов.
2. element.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\_tablesee.cpp – модуль программы тестирования, с которым работает пользователь.
6. test\_main.cpp, test\_tablesee.cpp – модуль с функциями тестирования для созданных классов. Содержат 10 тестов для класса TString, 9 - для класса TElement и 13 - для класса TTableSee.

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

Принимает ссылку на объект класса TTableSee.

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

Принимает ссылку на объект класса TTableSee.

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

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

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

В качестве аргумента принимает его ключ.

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

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

Возвращает элемент таблицы по его ключу.

13. template <typename Op>

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

# 

# 6. Заключение

В результате лабораторной работы была разработана структура данных – просмотровая таблица, а также освоены такие инструменты разработки программного обеспечения, как система контроля версий [Git](https://git-scm.com/book/ru/v2).

# 7. Литература

1. [https://ru.stackoverflow.com](https://ru.stackoverflow.com/)
2. Касперски К. Техника оптимизации программ. Эффективное использование памяти. — СПб.: БХВ-Петербург, 2003. — 464 с.: ил.
3. <http://www.cyberforum.ru/cpp-beginners/thread697660.html>
4. <http://qaru.site/questions/1776876/table-implementation-in-c>
5. <https://toster.ru/q/340182>

# 8. Приложениe

[**TableSee.cpp**](https://github.com/SamMakarikhin/All-labs/blob/tables/TableSee/TableSee/TableSee.cpp)

#include "pch.h"

#include <iostream>

#include <locale.h>

#include "tsee.h"

using namespace std;

int main()

{

int data;

int max;

TTableSee<int> look;

TElement<int> element;

char memory[100];

cout << "EXAMPLE: LOOKUP TABLE" << endl;

cout << "Max count of elements: ";

cin >> max;

for (int i = 0; i < max; i++)

{

cout << " Key of " << i + 1 << " element: ";

cin >> memory;

TString key(memory);

element.SetKey(key);

cout << " Data of " << i + 1 << " element: ";

cin >> data;

element.SetData(data);

look.Add(element);

}

cout << "TABLE IS FULL" << endl;

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 << "ADD ONE ELEMENT WITHOUT KEY\n ";

cout << " Data of new element: ";

cin >> data;

look.Add(data);

cout << endl << look;

cout << "DELETE ONE ELEMENT\n ";

cout << " Key of element for delete: ";

cin >> memory;

TString key2(memory);

look.Delete(key2);

cout << endl << look;

return 0;

}

[**element.cpp**](https://github.com/SamMakarikhin/All-labs/blob/tables/TableSee/TableSee/element.cpp)

#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();

TString& GetKey();

void SetData(const ElemType& A);

void SetKey(const TString& A);

template <typename Operator>

friend ostream& operator<<(ostream& out, TElement<Operator>& A);

};

template <typename ElemType>

TElement<ElemType>::TElement()

{

key = " ";

data = 0;

}

template<typename ElemType>

inline TElement<ElemType>::TElement(const ElemType & A, const TString & B)

{

}

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 ElemType>

ElemType& TElement<ElemType>::GetData()

{

return data;

}

template <typename ElemType>

TString& TElement<ElemType>::GetKey()

{

return key;

}

template <typename ElemType>

void TElement<ElemType>::SetData(const ElemType& A)

{

data = A;

}

template <typename ElemType>

void TElement<ElemType>::SetKey(const TString& A)

{

key = A;

}

template <typename Operator>

ostream& operator<<(ostream& ostr, TElement<Operator>& A)

{

ostr << "|" << A.GetKey().GetMemory() << "|";

ostr << A.GetData() << "|";

return ostr;

}

[**exception.cpp**](https://github.com/SamMakarikhin/All-labs/blob/tables/TableSee/TableSee/exception.cpp)

#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;

}

[**tsee.cpp**](https://github.com/SamMakarikhin/All-labs/blob/tables/TableSee/TableSee/tsee.cpp)

#pragma once

#include "element.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;

int GetSize() const;

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>

int TTableSee<TableSeeType>::GetCount() const

{

return count;

}

template <typename TableSeeType>

int TTableSee<TableSeeType>::GetSize() const

{

return size;

}

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;

}

[**tstring.cpp**](https://github.com/SamMakarikhin/All-labs/blob/tables/TableSee/TableSee/tstring.cpp)

#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;

char\* GetMemory();

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;

}

int TString::GetCount() const

{

return count;

}

char\* TString::GetMemory()

{

return 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];

}

**[test\_tsee.cpp](https://github.com/381706-2-zhbanova-nadezhda/381706-2_zhbanova_labs/pull/14/files" \l "diff-002104bff4fbf35e279bc2c2f79b699b" \o "test/test_tablesee.cpp)**

|  |
| --- |
|  |
|  |  |
|  | #include "tsee.h" |
|  |  |
|  |  |
|  | 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)); |
|  | } |