МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского» (ННГУ)

Институт информационных технологий, математики и механики

Направление подготовки «Прикладная математика и информатика»

**Отчёт по учебной практике**

**«Таблицы (несортированные, сортированные, хэш)»**

**Выполнил:** студент группы 381603-1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Капустин Д.О.

**Проверила:** к.т. н., доцент каф. МОСТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кустикова В.Д.

Нижний Новгород

2018 г.

**Содержание**

[Введение 3](#_Toc514109797)

[1 Постановка задачи 4](#_Toc514109798)

[2 Руководство пользователя 5](#_Toc514109799)

[3 Руководство программиста 6](#_Toc514109800)

[3.1 Структура программы 6](#_Toc514109801)

[3.2 Структуры данных 7](#_Toc514109802)

[3.3 Программная реализация](#_Toc514109804) 7

Заключение [1](#_Toc514109806)0

[Литература 1](#_Toc514109807)1

[Приложения 1](#_Toc514109808)2

[Приложение 1. Pol\_Link.h 1](#_Toc514109809)2

Приложение 2. Pol\_List.h. [1](#_Toc514109810)2

[Приложение 3. Pol\_Mon.h 1](#_Toc514109811)7

[Приложение 4. Pol.h 17](#_Toc514109812)

[Приложение 5. Pol\_Mon.cpp 20](#_Toc514109813)

[Приложение 6. Pol.cpp 2](#_Toc514109814)1

[Приложение 7. Main.cpp 26](#_Toc514109815)

Приложение 8. Tab.h ...................................................................................................28

Приложение 9. UnSort\_Tab.h ......................................................................................29

Приложение 10. Sort\_Tab.h .........................................................................................31

Приложение 11. Hash\_Tab.h .......................................................................................32

Приложение 12. Test\_pol.cpp ......................................................................................35

[Приложение 13. test\_tab.cpp 4](#_Toc514109816)5

# Введение

Целью работы является создание программы, способной выполнять операции вставки, удаления и поиска элемента в таблицах (несортированных, сортированных, хэш), хранящих в себе ключ и значение.

Программа реализуется на языке С++.

Разработанная программа может быть использована как самостоятельный продукт непосредственно для выполнения заданных пользователем операций с таблицами, так и в качестве составной части при решении более сложных задач обработки больших массивов информации (бухгалтерский и банковский учет, планирование и распределение ресурсов), в которых для представления и хранения информации используются таблицы как наиболее удобное средство для быстрого поиска информации, её обработки и сортировки по различным фильтрам и пользовательским настройкам.

# Постановка задачи

Разработать программу, которая позволяет пользователю выполнять операции вставки, удаления и поиска элемента в таблицах (несортированных, сортированных, хэш), хранящих в себе ключ (строка) и значение (полином). В ходе работы программы необходимо обеспечить уникальность ключа и быстродействие указанных выше операций, для чего таблицы при необходимости могут менять свой размер.

# Руководство пользователя

Для начала работы необходимо открыть файл Sample.exe.

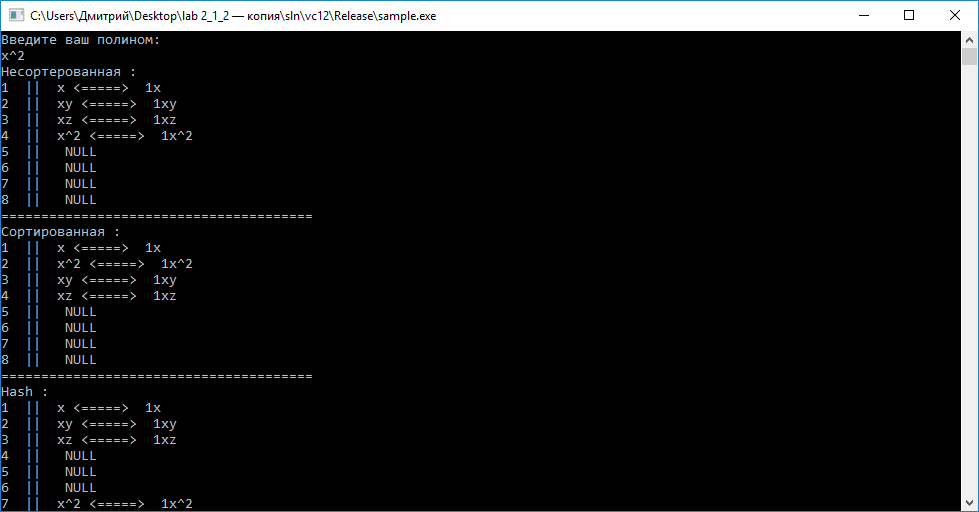
Затем на запрос программы ввести натуральное число, которое будет использовано в качестве размера автоматически создаваемых таблиц.

После чего выбрать желаемую операцию из предложенного меню:

1. Ввод элемента
2. Удаление элемента
3. Поиск элемента
4. Печать таблиц

При выборе пунктов 1, 2, 3 после предложения ввести Ваш полином осуществить ввод полинома трех переменных с вещественными коэффициентами (допустимые символы: x, y, z,^,0,1,2,3,4,5,6,7,8,9,+.-). Далее программа производит вывод результата выполнения операции (полученные таблицы/найдеyный элемент) и (по желанию пользователя) продолжает либо завершает свою работу.

В случае выбора пункта 4 меню производится печать имеющихся таблиц.



1. Пример

# Руководство программиста

## Структура программы

Pol\_Link.h

Pol\_List.h

Pol\_Mon.h

Pol.h

Pol\_Mon.cpp

Pol.cpp

Main.cpp

Tab.h

UnSort\_Tab.h

Sort\_Tab.h

Hash\_Tab.h

Test\_pol.cpp

Test\_tab.cpp

Программа состоит из пяти проектов.

*gtest*

Содержит в себе gtest.h и gtest-all.cpp. Файлы содержат google тесты, которые необходимы для проверки корректности реализованных классов.

*polinoms*

Содержит в себе:

1. Pol\_Link.h– реализация шаблонного класса звена.
2. Pol\_Mon.h– объявление полей и методов, которые отвечают за степень и коэффициент звена(монома)
3. Pol.h – объявление полей и методов класса полином
4. Pol\_List.h– реализация шаблонного класса список
5. Pol\_Mon.cpp– реализация полей и методов класса Monom
6. Pol.cpp – реализация полей и методов класса полином

*tab*

Содержит в себе:

Tab.h- заголовочный файл, содержащий объявление абстрактного базового виртуального класса Table и его реализацию.

UnSort\_Tab.h- заголовочный файл, содержащий объявление и реализацию неупорядоченных таблиц.

Sort\_Tab.h- заголовочный файл, содержащий объявление и реализацию упорядоченных таблиц.

Hash\_Tab.h- заголовочный файл, содержащий объявление и реализацию хэш-таблиц.

*tests*

Содержит в себе test\_tab.cpp, test\_pol.cpp. Реализация тестов для проверки корректности реализованных классов.

*sample*

Содержит Main.cpp– реализация меню консольного приложения.

## Структуры данных

Структура DataTable:

Хранит два поля: ключ(Key) и данные(Data).

Абстрактный, шаблонный класс таблиц – Table содержит реализацию методов навигации и переупаковки, при которой размер таблиц увеличивается и данные копируются в новый массив большего размера.

Класс несортированных таблиц – UnSortTab

Вставка:

Пользователь вводит ключ, который является полиномом. Если таблица заполнена на половину, происходит увеличение размера массива (таблицы) в 2 раза, старые данные будут сохранены и занесены в новую увеличенный массив (таблицу). Происходит поиск совпадающих ключей, если таких нет, то запись вставляется в первую свободную ячейку, начиная сначала, иначе срабатывает исключение с информацией о том, что существует запись с таким же ключом. Увеличиваем Top на единицу.

Поиск:

Пользователь вводит ключ. Далее, массив элементов просматривается на совпадение ключей. Если была найдена запись с таким же ключом, то функция возвращается адрес на структуру DataTable, иначе срабатывает исключение с информацией о том, что записей с таким же ключом не обнаружено.

Удаление:

Пользователь вводит ключ. Происходит поиск элемента, если он найден, то происходит его удаление и обнуление адреса данной ячейки, иначе срабатывает исключение с информацией о том, что записей с таким же ключом не обнаружено.

Класс сортированных таблиц – SortTab

Вставка:

Пользователь вводит ключ, который является полиномом. Если таблица заполнена на половину, происходит увеличение размера массива (таблицы) в 2 раза, старые данные будут сохранены и занесены в новую увеличенный массив (таблицу). Функция вставки, с помощью алгоритма бинарного поиска, находит место (индекс), куда будет вставлена наша запись. Если ячейка с данным индексом не заполнена, то элемент занимает своё место, иначе происходит смещение всех элементов массива на одну позицию, начиная с конца. Увеличиваем Top на единицу. При просмотре ячейки, также сравниваются ключи, если они совпали, срабатывает исключение с информацией о том, что существует запись с таким же ключом.

Поиск:

Пользователь вводит ключ. Далее, с помощью алгоритма бинарного поиска, куда был передан наш ключ, возвращается индекс, где предположительно находится наша запись. Если была найдена запись с таким же ключом, то функция возвращается адрес на структуру DataTable, иначе срабатывает исключение с информацией о том, что записей с таким же ключом не обнаружено.

Удаление:

Пользователь вводит ключ. Далее, с помощью алгоритма бинарного поиска, куда был передан наш ключ, возвращается индекс, где предположительно находится наша запись. Если была найдена запись с таким же ключом, то происходит удаление элемента и происходит смещение всех элементов массива на одну позицию, начиная с места удаления. Top уменьшается на единицу.

Класс хэш таблиц – HashTab

Вставка:

Пользователь вводит ключ, который является полиномом. Если таблица заполнена на половину, происходит увеличение размера массива (таблицы) в 2 раза, старые данные будут сохранены и занесены в новую увеличенный массив (таблицу). Хэш-функция, принимает ключ и переводит его в число, это число будет использоваться в качестве индекса для вставки в массив. Если ячейка с данным индексом не заполнена (имеет статус FREE(свободна)), то элемент занимает своё место, переменная, которая хранит количество вставленных элементов, увеличивает своё значение на единицу, занятая ячейка получает статус BUSY (занята). Если ячейка была удалена, происходят аналогичные действия, за исключение того, что переменная, которая хранит количество вставленных элементов, не увеличивает своё значение на единицу. При просмотре ячейки, также сравниваются ключи, если они совпали, срабатывает исключение с информацией о том, что существует запись с таким же ключом. Иначе происходит поиск свободных ячеек с помочью функции rand(), в которую мы передаем индекс занятой ячейки, это происходит до тех пор, пока элемент, находящийся по новому (полученному с помощью rand()) индексу занят. Далее ячейка имеет статус либо FREE (свободна), либо (DELETE) удалена, алгоритм для этих случаев приведен в пункте «Вставка».

Удаление:

Пользователь вводит ключ. Далее, с помощью хэш-функции получаем индекс. Если по полученному индексу ячейка имеет статус FREE (свободна), то вызывается исключение, иначе если была найдена запись с таким же ключом, то происходит удаление элемента, а эта ячейка получает статус DELETE(удалена).

Поиск:

Аналогично удалению, только возвращается адрес, лежащий по индексу найденной ячейки.

## Программная реализация

### Структура Monom

Поля:

double cf–коэффициент монома

unsigned int stmonom – степень монома

Методы:

Monom(const Monom&\_Tmonom) –конструктор копирования

Monom(const double d=0.0; const int m=0) –конструктор с параметрами

~Monom() – деструктор

Monom operator + (constTMonom&monom) –сумма мономов

Monom operator = (constTMonom&monom) –оператор копирования

bool operator == (constTMonom&monom) const –сравнение мономов

bool operator != (constTMonom&monom) const –сравнение мономов

bool operator > (constTMonom&monom) const –сравнение мономов

bool operator < (constTMonom&monom) const –сравнение мономов

### Класс TLink

Поля:

Tdata –данные

TLink<T>\* pNext – указатель на следующее звено

Методы:

TLink() – конструктор по умолчанию

TLink(const T datum, TLink<T>\* link = nullptr) – конструктор по параметрам

~TLink() –деструктор

TLink<T>& operator = (const TLink<T>& \_TLink) –оператор копирования

bool operator == (const TLink<T>& link) const –оператор сравнения

### Класс TRingList

Поля:

TLink<T> \*head – голова списка

TLink<T> \*current – текущий элемент списка

Методы:

TRingList() –конструктор по умолчанию

TRingList(constTRingList<T>&) –конструктор копирования

~TRingList() –деструктор

TRingList<T>& operator = (constTRingList<T>& TRingList2) –оператор копирования

void Clean() –очистка списка

void Insert(const T \_data) – упорядоченная вставка в список

void InsertInEnd(const T datum) – вставка в конец

void Reset() – текущий элемент указывает следующее звено после головы

void Delete(const T & d) – удаление данных

TLink<T>Search(const TLink<T>& d) – поиск в списке данных

void GetNext() – двигаем указатель текущего элемента на следующий

bool IsEnded() const – проверка на конце списка

TLink<T>\* GetCurrent() – вернуть текущее звено

bool operator == (constTRingList<T>& list2) –оператор сравнения

bool operator > (constTRingList<T>& list2) –оператор сравнения

bool operator < (constTRingList<T>& list2) –оператор сравнения

bool operator != (constTRingList<T>& list2) –оператор сравнения

### Класс TPolinom

Поля:

TRingList<Monom>curmon – список

Методы:

TPolinom() – конструктор по умолчанию

TPolinom(const string &s) – конструктор по строке

TPolinom(const TPolinom&) –конструткор копирования

~TPolinom() –деструктор

TPolinom operator + (constTPolinom& \_pol) –оператор суммы

TPolinom& operator += (constTPolinom& \_pol) –оператор суммы

TPolinom operator - (constTPolinom& \_pol) –оператор разности

TPolinom operator \* (constTPolinom& \_pol) – оператор произведения

TPolinomoperator \* (double d) – оператор умножения на константу

friendTPolinomoperator \* (doublecf2, constTPolinom&poli2) – оператор произведения константы и полинома

TPolinom& operator = (constTpolinom&\_pol) –оператор копирвоания

bool operator == (constTPolinom& curpol) –оператор сравнения

bool operator != (constTPolinom& curpol) –оператор сравнения

bool operator > (constTPolinom& curpol) –оператор сравнения

bool operator < (constTPolinom& curpol) –оператор сравнения

voidPrivPod() – функция приведения подобных звеньев

friendostream& operator <<(ostream&out, constTPolinom& \_pol) –перегрузка вывода полинома

### Класс Table

Поля:

DataTable<KeyField, DataField> \*\* DT– массив из указателей на DataTable

Int MaxSize; – размер таблицы

Virtual void Realloc() = 0 – изменение размера таблицы

Int current– индекс текущего элемента

Int Top– индеек - количество последнего или вставленных элемента(ов).

Методы:

Table(int SizeTable = MIN\_SIZE);– конструктор

virtual ~Table();– деструктор

virtual void Insert(const KeyField &KEY, const DataField &DATA) =0;– вставка

virtual void Delete(const KeyFiled &KEY) = 0;– удаление

virtual DataTable<KeyField, DataField> \* Search(const KeyField &KEY) const = 0– поиск

void Reset();– переход в начало таблицы

boolIsFull() const– проверка на полноту таблицы

void GoNext()– увеличение текущего индекса

DataFieldGetCurData() const;– получение данных в текущей ячейке

friend std::ostream& operator<<(std::ostream& out, const Table<KeyField, DataField>&P);– оператор вывода

Класс UnSortTab – класс таблиц, наследник (потомок) класса Table не содержит новых полей и методов.

Класс SortTab – класс таблиц, наследник потомок класса Table

Новые методы:

int BinSearch(const KeyField &KEY) const;– бинарный поиск

Класс HashTable – класс таблиц, наследник (потомок) класса Table

Новые поля:

int \*Status;– массив – хранит статус ячейки (DELETE(-1,удалена), BUSY (1,занята), FREE (0,свободна))

Новые методы:

Int ReHash(const KeyField&KEY) const;– хэш-функция

# Заключение

Представленная программа позволяет пользователю выполнять операции вставки, удаления и поиска элемента в таблицах (несортированных, сортированных, хэш), хранящих в себе ключ (строка) и значение (полином). Реализован алгоритм, позволяющий обеспечить уникальность ключа и разрешить проблемы при возникновении коллизий в хэш-таблицах.

Таким образом, цель работы достигнута.

# Литература

1. *Гергель В.П.* Рабочие материалы студента по общему курсу «Методы программирования» - Н.Новгород, 2015.

# Приложения

**Приложение 1. Pol\_Link.h**

#pragma once;

template <Fieldname T>

class TLink {

public:

T data;

TLink<T>\* pNext;

TLink() {};

~TLink() {};

TLink(const T datum, TLink<T>\* link = nullptr) { data = datum; pNext = link; }

TLink<T>& operator=(const TLink<T>& \_TLink)

{

data = \_TLink.data;

pNext = \_TLink.pNext;

return \*this;

};

bool operator == (const TLink<T>& link) const

{

return \*this->pNext == link.pNext;

};

## };

## Приложение 2. Pol\_List.h

#pragma once;

#include "Pol\_Link.h"

template <Fieldname T>

class TRingList {

private:

TLink <T> \*head;

TLink <T> \*current;

public:

TRingList();

TRingList(const TRingList <T> &);

~TRingList();

void Insert(const T \_data);

void InsertInEnd(const T datum);

void Reset();

void Clean();

bool IsEnded() const;

TRingList<T>& operator=(const TRingList<T>& \_TRingList);

bool operator==(const TRingList<T>& list2) const;

bool operator<(const TRingList<T>& list2) const;

bool operator>(const TRingList<T>& list2) const;

bool operator!=(const TRingList<T>& list2) const { return !((\*this) == list2); }

TLink<T>\* GetNext();

void GetNext() { current = current->pNext; };

//TLink<T>\* Search(const TLink<T>& d);

void Delete(const T & d);

TLink<T>\* GetCurrent() { return current; }

};

template <Fieldname T>

TRingList<T>::TRingList()

{

head = new TLink<T>();

head->pNext = head;

current = head;

};

template <Fieldname T>

TLink<T>\* TRingList<T>::Search(const TLink<T> & d)

{

TLink<T>\* tmp = head;

while ((tmp->pNext != head) && (tmp->pNext->data != d.data))

{

tmp = tmp->pNext;

}

if (tmp->pNext == head)

{

return nullptr;

}

return tmp->pNext;

};

template<Fieldname T>

void TRingList<T>::Delete(const T & d)

{

TLink<T>\* tmp = head;

TLink<T>\* tmp2 = head->pNext;

if (tmp != tmp2)

{

while ((tmp2->pNext != head) && (tmp2->data != d))

{

tmp = tmp2;

tmp2 = tmp2->pNext;

}

if (tmp2->data == d)

{

tmp->pNext = tmp2->pNext;

delete tmp2;

}

else throw "element does not exist in your list";

}

else throw "element does not exist in your list";

};

template <Fieldname T>

TRingList<T>::TRingList(const TRingList <T> & Ring)

{

if (Ring.head->pNext != Ring.head)

{

head = new TLink<T>(Ring.head->data);

TLink<T> \*temp = head;

TLink<T> \*temp2 = Ring.head->pNext;

while (temp2 != Ring.head)

{

temp->pNext = new TLink<T>(temp2->data, head);

temp = temp->pNext;

temp2 = temp2->pNext;

}

}

else {

head = new TLink<T>(Ring.head->data);

current = head;

head->pNext = head;

}

};

template<Fieldname T>

TRingList<T>::~TRingList()

{

Clean();

delete head;

};

template <Fieldname T>

void TRingList<T>::Insert(const T datum)

{

if (head->pNext == head)

{

TLink<T>\* tmp = new TLink<T>(datum, head);

head->pNext = tmp;

}

else

{

TLink<T>\* prev = head;

TLink<T>\* tmp = head->pNext;

while ((tmp != head) && (datum < tmp->data))

{

prev = tmp;

tmp = tmp->pNext;

}

prev->pNext = new TLink<T>(datum, tmp);

}

};

template <Fieldname T>

void TRingList<T>::InsertInEnd(const T datum)

{

TLink<T>\* tmp = head;

while (tmp->pNext != head)

tmp = tmp->pNext;

tmp->pNext = new TLink<T>(datum, head);

};

template <Fieldname T>

void TRingList<T>::Reset()

{

current = head->pNext;

};

template<Fieldname T>

void TRingList<T>::Clean()

{

TLink<T> \*temp = head->pNext;

TLink<T> \*temp2 = head;

while (temp != head)

{

temp2 = temp->pNext;

delete temp;

temp = temp2;

}

head->pNext = head;

};

template<Fieldname T>

bool TRingList<T>::IsEnded() const

{

return (current == head);

};

template<Fieldname T>

TRingList<T>& TRingList<T>::operator=(const TRingList<T> & Ring)

{

if ((\*this) != Ring)

{

Clean();

if (Ring.head->pNext != Ring.head)

{

head = new TLink<T>(Ring.head->data);

TLink<T> \*temp = head;

TLink<T> \*temp2 = Ring.head->pNext;

while (temp2 != Ring.head)

{

temp->pNext = new TLink<T>(temp2->data, head);

temp = temp->pNext;

temp2 = temp2->pNext;

}

}

else {

head = new TLink<T>(Ring.head->data);

current = head;

head->pNext = head;

}

}

return \*this;

};

template<Fieldname T>

bool TRingList<T>::operator==(const TRingList<T> & list2) const

{

TLink<T> \*temp = head;

TLink<T> \*temp2 = list2.head;

bool f = true;

if ((temp->pNext == head) && (temp2->pNext == list2.head))

f = true;

else if (((temp->pNext != head) && (temp2->pNext == list2.head)) || ((temp2->pNext != list2.head) && (temp->pNext == head)))

f = false;

while ((temp->pNext != head) && (temp2->pNext != list2.head) && (f))

{

if (temp->data != temp2->data)

f = false;

temp = temp->pNext;

temp2 = temp2->pNext;

}

return f;

};

template<Fieldname T>

bool TRingList<T>::operator<(const TRingList<T> & list2) const

{

TLink<T> \*temp = head;

TLink<T> \*temp2 = list2.head;

bool f = true;

if ((temp->pNext == head) && (temp2->pNext == list2.head))

f = false;

else if ((temp->pNext != head) && (temp2->pNext == list2.head))

f = false;

else if ((temp2->pNext != list2.head) && (temp->pNext == head))

f = true;

while ((temp->pNext != head) && (temp2->pNext != list2.head) && (f))

{

if ((temp->data > temp2->data) || (temp->data == temp2->data))

f = false;

temp = temp->pNext;

temp2 = temp2->pNext;

}

return f;

};

template<Fieldname T>

bool TRingList<T>::operator>(const TRingList<T> & list2) const

{

TLink<T> \*temp = head;

TLink<T> \*temp2 = list2.head;

bool f = true;

if ((temp->pNext == head) && (temp2->pNext == list2.head))

f = true;

else if ((temp->pNext != head) && (temp2->pNext == list2.head))

f = true;

else if ((temp2->pNext != list2.head) && (temp->pNext == head))

f = false;

while ((temp->pNext != head) && (temp2->pNext != list2.head) && (f))

{

if ((temp->data < temp2->data) || (temp->data == temp2->data))

f = false;

temp = temp->pNext;

temp2 = temp2->pNext;

}

return f;

};

template <Fieldname T>

TLink<T>\* TRingList<T>::GetNext()

{

current = current->pNext;

return current;

};

## 

## Приложение 3. Pol\_Mon.h

#pragma once

struct Monom {

double cf;

unsigned int stmonom;

Monom(const double d = 0.0, const int m = 0) { cf = d; stmonom = m; };

Monom(const Monom& \_Tmonom);

const Monom& operator = (const Monom& \_Tmonom);

bool operator < (const Monom& \_tmonom) const;

bool operator != (const Monom& \_tmonom) const;

bool operator == (const Monom& \_tmonom) const;

bool operator > (const Monom& \_tmonom) const;

Monom operator+ (const Monom& \_tmonom);

};

## 

## Приложение 4. Pol.h

#include <iostream>

#include <cstdio>

#include <string>

#include "Pol\_List.h"

#include "Pol\_Mon.h"

using namespace std;

class TPolinom {

private:

TRingList<Monom> curmon;

public:

TPolinom() { }

TPolinom(const string &s);

TPolinom(const TPolinom &);

~TPolinom();

TPolinom operator+ (const TPolinom & \_pol);

TPolinom operator- (const TPolinom & \_pol)

{

TPolinom res = (\*this) + (-1.0) \* \_pol;

return res;

}

TPolinom& operator+= (const TPolinom & \_pol);

TPolinom operator\* (const TPolinom & \_pol);

TPolinom operator\* (double d);

TPolinom& operator = (const TPolinom& \_pol);

bool operator == (const TPolinom& curpol) const { return (\*this).curmon == curpol.curmon; };

bool operator != (const TPolinom& curpol) const { return !((\*this) == curpol); };

bool operator < (const TPolinom& curpol) const { return (\*this).curmon <curpol.curmon; };

bool operator > (const TPolinom& curpol) const { return (\*this).curmon > curpol.curmon; };

friend ostream& operator<< (ostream &out, const TPolinom& \_pol)

{

TPolinom copy\_of\_polinom(\_pol);

copy\_of\_polinom.curmon.Reset();

cout << copy\_of\_polinom.curmon.GetCurrent()->data.cf;

if ((copy\_of\_polinom.curmon.GetCurrent()->data.cf==1) && (copy\_of\_polinom.curmon.GetCurrent()->data.stmonom / 100 != 0) && (copy\_of\_polinom.curmon.GetCurrent()->data.stmonom / 10) % 10 != 0 && (copy\_of\_polinom.curmon.GetCurrent()->data.stmonom) % 10 != 0)

out << copy\_of\_polinom.curmon.GetCurrent()->data.cf;

if ((copy\_of\_polinom.curmon.GetCurrent()->data.stmonom / 100) != 0) {

if ((copy\_of\_polinom.curmon.GetCurrent()->data.stmonom / 100) == 1)

out << "x";

else

out << "x^" << (copy\_of\_polinom.curmon.GetCurrent()->data.stmonom / 100);

}

if ((copy\_of\_polinom.curmon.GetCurrent()->data.stmonom / 10) % 10 != 0) {

if ((copy\_of\_polinom.curmon.GetCurrent()->data.stmonom / 10) % 10 == 1)

out << "y";

else out << "y^" << (copy\_of\_polinom.curmon.GetCurrent()->data.stmonom / 10) % 10;

}

if ((copy\_of\_polinom.curmon.GetCurrent()->data.stmonom) % 10 != 0) {

if ((copy\_of\_polinom.curmon.GetCurrent()->data.stmonom % 10) == 1)

out << "z";

else out << "z^" << (copy\_of\_polinom.curmon.GetCurrent()->data.stmonom) % 10;

}

TLink<Monom>\* cur = copy\_of\_polinom.curmon.GetNext();

for (cur; !copy\_of\_polinom.curmon.IsEnded(); copy\_of\_polinom.curmon.GetNext())

{

if (copy\_of\_polinom.curmon.GetCurrent()->data.cf > 0)

{

out << " +" << copy\_of\_polinom.curmon.GetCurrent()->data.cf;

}

else out << " " << copy\_of\_polinom.curmon.GetCurrent()->data.cf;

if ((copy\_of\_polinom.curmon.GetCurrent()->data.stmonom / 100) != 0) {

if ((copy\_of\_polinom.curmon.GetCurrent()->data.stmonom / 100) == 1)

out << "x";

else

out << "x^" << (copy\_of\_polinom.curmon.GetCurrent()->data.stmonom / 100);

}

if ((copy\_of\_polinom.curmon.GetCurrent()->data.stmonom / 10) % 10 != 0)

{

if ((copy\_of\_polinom.curmon.GetCurrent()->data.stmonom / 10) % 10 == 1)

out << "y";

else out << "y^" << (copy\_of\_polinom.curmon.GetCurrent()->data.stmonom / 10) % 10;

}

if ((copy\_of\_polinom.curmon.GetCurrent()->data.stmonom) % 10 != 0) {

if ((copy\_of\_polinom.curmon.GetCurrent()->data.stmonom % 10) == 1)

out << "z";

else out << "z^" << (copy\_of\_polinom.curmon.GetCurrent()->data.stmonom) % 10;

}

}

out << endl;

return out;

}

friend TPolinom operator\* (double d, const TPolinom &p)

{

TPolinom res(p);

res.curmon.Reset();

if (d == 0)

res.curmon.Clean();

else {

for (res.curmon.GetCurrent(); !res.curmon.IsEnded(); res.curmon.GetNext())

{

res.curmon.GetCurrent()->data.cf \*= d;

}

}

return res;

};

};

## 

## Приложение 5. Pol\_Mon.cpp

#include "Pol\_Mon.h"

Monom::Monom(const Monom& \_Tmonom)

{

cf = \_Tmonom.cf;

stmonom = \_Tmonom.stmonom;

}

const Monom& Monom::operator = (const Monom& \_Tmonom)

{

cf = \_Tmonom.cf;

stmonom = \_Tmonom.stmonom;

return \*this;

}

bool Monom::operator < (const Monom& \_tmonom) const

{

return stmonom < \_tmonom.stmonom;

}

bool Monom::operator != (const Monom& \_tmonom) const

{

return stmonom != \_tmonom.stmonom;

}

bool Monom::operator == (const Monom& \_tmonom) const

{

return stmonom == \_tmonom.stmonom;

}

bool Monom::operator > (const Monom& \_tmonom) const

{

return stmonom > \_tmonom.stmonom;

}

Monom Monom::operator+ (const Monom& \_tmonom)

{

Monom res;

if ((\*this).stmonom == \_tmonom.stmonom)

{

res.stmonom = (\*this).stmonom;

res.cf = (\*this).cf + \_tmonom.cf;

}

else throw "unable to sum monoms!";

return res;

}

## Приложение 6. Pol.cpp

#include "Pol.h"

TPolinom::TPolinom(const std::string& s)

{

string str = s;

if (str[0] != '\0')

{

while (str.length() > 0)

{

string monom = "";

Monom temp;

string coeff = "";

double number = 0;

int i = 0, j = 1;

j = 1;

while (j < str.length() && str[j] != '+' && str[j] != '-')

j++;

monom = str.substr(0, j);

str.erase(0, j);

j = 0;

while (j < monom.length() && !isalpha(monom[j]))

j++;

coeff = monom.substr(0, j);

if (coeff == "" || coeff == "+")

temp.cf = 1;

else if (coeff == "-")

temp.cf = -1;

else temp.cf = atof(coeff.c\_str());

monom.erase(0, j);

j = 0;

while (j < monom.length())

{

if (isalpha(monom[j]) && monom[j + 1] == '\0')

{

switch (monom[j])

{

case 'x':

temp.stmonom += 1 \* 100;

break;

case 'y':

temp.stmonom += 1 \* 10;

break;

case 'z':

temp.stmonom += 1;

break;

default:

throw "Input error:end of string";

break;

}

j++;

}

else if (isalpha(monom[j]) && isalpha(monom[j + 1]))

{

switch (monom[j])

{

case 'x':

temp.stmonom += 1 \* 100;

break;

case 'y':

temp.stmonom += 1 \* 10;

break;

case 'z':

temp.stmonom += 1;

break;

default:

throw "Input error:end of string";

break;

}

j++;

}

else if (isalpha(monom[j]) && monom[j + 1] == '^' && monom[j + 2] != '\0' && isdigit(monom[j + 2]))

{

switch (monom[j])

{

case 'x':

temp.stmonom += atoi(&monom[j + 2]) \* 100;

j += 3;

break;

case 'y':

temp.stmonom += atoi(&monom[j + 2]) \* 10;

j += 3;

break;

case 'z':

temp.stmonom += atoi(&monom[j + 2]) \* 1;

j += 3;

break;

}

}

}

if ((temp.stmonom != 0 || temp.stmonom == 0) && temp.cf != 0) {

TLink<Monom>\* link = curmon.Search(temp);

if (link != NULL)

{

if (link->data.cf + temp.cf != 0)

{

link->data.cf += temp.cf;

}

else curmon.Delete(link->data);

}

else

curmon.Insert(temp);

}

}

}

}

TPolinom::TPolinom(const TPolinom & \_polinom)

{

curmon = \_polinom.curmon;

}

TPolinom::~TPolinom()

{

}

TPolinom TPolinom::operator+ (const TPolinom & \_polinom)

{

(\*this).curmon.Reset();

TPolinom copy\_of\_polinom(\_polinom);

copy\_of\_polinom.curmon.Reset();

TPolinom res;

res.curmon.Reset();

while (!copy\_of\_polinom.curmon.IsEnded() && !(\*this).curmon.IsEnded())

{

if (copy\_of\_polinom.curmon.GetCurrent()->data < (\*this).curmon.GetCurrent()->data)

{

res.curmon.InsertInEnd((\*this).curmon.GetCurrent()->data);

(\*this).curmon.GetNext();

}

else if (copy\_of\_polinom.curmon.GetCurrent()->data == (\*this).curmon.GetCurrent()->data)

{

if ((\*this).curmon.GetCurrent()->data.cf + copy\_of\_polinom.curmon.GetCurrent()->data.cf != 0.0)

{

res.curmon.InsertInEnd((\*this).curmon.GetCurrent()->data + copy\_of\_polinom.curmon.GetCurrent()->data);

}

(\*this).curmon.GetNext();

copy\_of\_polinom.curmon.GetNext();

}

else

{

res.curmon.InsertInEnd(copy\_of\_polinom.curmon.GetCurrent()->data);

copy\_of\_polinom.curmon.GetNext();

}

}

while (!copy\_of\_polinom.curmon.IsEnded())

{

res.curmon.InsertInEnd(copy\_of\_polinom.curmon.GetCurrent()->data);

copy\_of\_polinom.curmon.GetNext();

}

while (!(\*this).curmon.IsEnded())

{

res.curmon.InsertInEnd((\*this).curmon.GetCurrent()->data);

(\*this).curmon.GetNext();

}

return res;

}

TPolinom & TPolinom::operator+=(const TPolinom & \_polinom)

{

TPolinom copy\_of\_polinom(\_polinom);

copy\_of\_polinom.curmon.Reset();

for (copy\_of\_polinom.curmon.GetCurrent(); !copy\_of\_polinom.curmon.IsEnded(); copy\_of\_polinom.curmon.GetNext())

{

TLink<Monom>\* tmp2 = (\*this).curmon.Search(copy\_of\_polinom.curmon.GetCurrent()->data);

if (tmp2 != NULL)

{

if (tmp2->data.cf + copy\_of\_polinom.curmon.GetCurrent()->data.cf != 0)

{

tmp2->data.cf += copy\_of\_polinom.curmon.GetCurrent()->data.cf;

}

else

(\*this).curmon.Delete(tmp2->data);

}

else

(\*this).curmon.Insert(copy\_of\_polinom.curmon.GetCurrent()->data);

}

return \*this;

}

TPolinom TPolinom::operator\* (const TPolinom & \_polinom)

{

(\*this).curmon.Reset();

TPolinom copy\_of\_polinom(\_polinom);

copy\_of\_polinom.curmon.Reset();

TPolinom res;

res.curmon.Reset();

for ((\*this).curmon.GetCurrent(); !(\*this).curmon.IsEnded(); (\*this).curmon.GetNext())

{

for (copy\_of\_polinom.curmon.GetCurrent(); !copy\_of\_polinom.curmon.IsEnded(); copy\_of\_polinom.curmon.GetNext())

{

if (((\*this).curmon.GetCurrent()->data.stmonom + copy\_of\_polinom.curmon.GetCurrent()->data.stmonom < 1000) &&

(((\*this).curmon.GetCurrent()->data.stmonom / 10 % 10 + copy\_of\_polinom.curmon.GetCurrent()->data.stmonom / 10 % 10) < 10) &&

(((\*this).curmon.GetCurrent()->data.stmonom % 10 + copy\_of\_polinom.curmon.GetCurrent()->data.stmonom % 10) < 10))

{

unsigned int tmpabc = (\*this).curmon.GetCurrent()->data.stmonom + copy\_of\_polinom.curmon.GetCurrent()->data.stmonom;

Monom tmp((\*this).curmon.GetCurrent()->data.cf \* copy\_of\_polinom.curmon.GetCurrent()->data.cf, tmpabc);

TLink<Monom>\* tmp2 = res.curmon.Search(tmp);

if (tmp2 != NULL)

if ((tmp2->data.cf += (\*this).curmon.GetCurrent()->data.cf \* copy\_of\_polinom.curmon.GetCurrent()->data.cf) == 0)

{

res.curmon.Delete(tmp2->data);

}

else tmp2->data.cf += (\*this).curmon.GetCurrent()->data.cf \* copy\_of\_polinom.curmon.GetCurrent()->data.cf;

else

res.curmon.Insert(tmp);

}

else throw "Power is more then expected";

}

copy\_of\_polinom.curmon.Reset();

}

return res;

}

TPolinom TPolinom::operator\* (double d)

{

TPolinom res = (\*this);

res.curmon.Reset();

if (d == 0)

res.curmon.Clean();

else {

while (!res.curmon.IsEnded())

{

res.curmon.GetCurrent()->data.cf \*= d;

res.curmon.GetNext();

}

}

return res;

}

TPolinom& TPolinom::operator =(const TPolinom& \_polinom)

{

curmon = \_polinom.curmon;

return \*this;

}

## Приложение 7. Main.cpp

#include <cstdio>

#include <string>

#include <iostream>

#include "Pol.h"

#include "Pol\_Mon.h"

#include"Hash\_Tab.h"

#include"UnSort\_Tab.h"

#include"Sort\_Tab.h"

using namespace std;

int main()

{

int c = 0;

int menu;

int SIZE;

string a;

setlocale(LC\_ALL, "Russian");

cout << "Создание таблиц,введите размер ваших таблиц:" << endl;

cin >> SIZE;

UnSortTab <string, TPolinom> A(SIZE);

SortTab<string, TPolinom> B(SIZE);

HashTab <string, TPolinom> C(SIZE);

while (c != -1)

{

system("cls");

cout << "Выберите операцию:" << endl;

cout << "1.Ввод элемента" << endl;

cout << "2.Удаление элемента" << endl;

cout << "3.Поиск элемента" << endl;

cout << "4.Печать таблиц" << endl;

cin >> menu;

switch (menu)

{

case 1:

{

string str;

cout << "Введите ваш полином:" << endl;

cin >> str;

TPolinom a(str);

try

{

A.Insert(str, a);

}

catch (...)

{

cout << "Повторный ключ" << endl;

}

try

{

B.Insert(str, a);

}

catch (...)

{

cout << "Повторный ключ" << endl;

}

try

{

C.Insert(str, a);

}

catch (...)

{

cout << "Повторный ключ" << endl;

}

cout << "Несортерованная :" << endl;

cout << A << endl;

cout << "Сортированная :" << endl;

cout << B << endl;

cout << "Hash :" << endl;

cout << C << endl;

break;

}

case 2:

{

string str;

cout << "Введите ваш полином:" << endl;

cin >> str;

TPolinom a(str);

A.Delete(str);

B.Delete(str);

C.Delete(str);

cout << "Несортированная :" << endl;

cout << A << endl;

cout << "Сортированная :" << endl;

cout << B << endl;

cout << "Hash :" << endl;

cout << C << endl;

break;

}

case 3:

{

string str;

cout << "Введите ваш полином" << endl;

cin >> str;

TPolinom a(str);

try

{

if (A.Search(str) != nullptr)

cout << "Найден: " << A.Search(str)->Data << endl;

else

cout << "Not found!" << endl;

}

catch (...) { cout << "Не найден" << endl; }

try

{

if (B.Search(str) != nullptr)

cout << "Найден: " << B.Search(str)->Data << endl;

else

cout << "Not found!" << endl;

}

catch (...) { cout << "Не найден" << endl; }

try

{

if (C.Search(str) != nullptr)

cout << "Найден: " << C.Search(str)->Data << endl;

else

cout << "Not found!" << endl;

}

catch (...) { cout << "Не найден" << endl; }

break;

}

case 4:

{

cout << "Несортированная :" << endl;

cout << A << endl;

cout << "Сортированная :" << endl;

cout << B << endl;

cout << "Hash :" << endl;

cout << C << endl;

break;

}

default:

{

cout << "Ошибка, попробуйте снова" << endl;

break;

}

}

cout << "Для продолжения работы введите '1'" << endl;

cout << "Для выхода введите '- 1' " << endl;

cin >> c;

}

return 0;

}

## Приложение 8. Tab.h

#pragma once

#define MIN\_SIZE 10

#include <iostream>

#include <ostream>

template <class KeyField, class DataField>

struct DataTable

{

KeyField Key;

DataField Data;

DataTable(KeyField Key\_k = KeyField(), DataField Data\_d = DataField()) { Key = Key\_k; Data = Data\_d; }

};

template <class KeyField, class DataField>

class Table

{

public:

Table(int SizeTable = MIN\_SIZE);

virtual ~Table();

virtual void Insert(const KeyField &KEY, const DataField &DATA) = 0;

virtual void Delete(const KeyField &KEY) = 0;

virtual DataTable<KeyField, DataField> \* Search(const KeyField &KEY) const = 0;

void Reset() { current = 0; };

bool IsFull() const

{

if (Top == 0)

return true;

if (current == Top - 1) return 1;

else return 0;

};

void GoNext() { if (!IsFull())current++; };

DataField GetCurData() const { return (\*(this->DATATAB[current])).Data; };

template <class KeyField, class DataField>

friend std::ostream& operator<<(std::ostream& os, const Table<KeyField, DataField>&P);

protected:

DataTable <KeyField, DataField> \*\* DATATAB;

int MaxSize;

virtual void Realloc() = 0;

int current;

int Top;

};

template <class KeyField, class DataField>

Table<KeyField, DataField>::Table(int SizeTable)

{

MaxSize = SizeTable;

DATATAB = new DataTable<KeyField, DataField> \*[SizeTable];

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

DATATAB[i] = NULL;

Top = 0;

current = 0;

};

template <class KeyField, class DataField>

Table<KeyField, DataField>::~Table()

{

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

delete DATATAB[i];

delete[] DATATAB;

};

template <class KeyField, class DataField>

std::ostream& operator << (std::ostream& out, const Table<KeyField, DataField>& P)

{

for (int i = 0; i<P.MaxSize; i++)

{

if (P.DATATAB[i] != NULL)

out << i+1 << " || " << (\*(P.DATATAB[i])).Key << " <=====> " << " " << (\*(P.DATATAB[i])).Data;

else out << i+1 << " || " << " NULL" << endl;

}

cout << "=======================================";

return out;

}

## Приложение 9. UnSort\_Tab.h

#pragma once

#include "Tab.h"

#include "math.h"

#include <string>

using namespace std;

template <class KeyField, class DataField>

class UnSortTab : public Table <KeyField, DataField>

{

public:

UnSortTab(int SIZE = MIN\_SIZE) : Table<KeyField, DataField>(SIZE) {};

virtual ~UnSortTab() {};

virtual void Insert(const KeyField &KEY, const DataField &DATA);

virtual void Delete(const KeyField &KEY);

virtual int GetMaxSize() { return MaxSize; };

virtual int GetCurrent() { return current; };

virtual DataTable<KeyField, DataField>\* Search(const KeyField &KEY) const;

private:

virtual void Realloc();

};

template <class KeyField, class DataField>

void UnSortTab <KeyField, DataField> ::Realloc()

{

int NextSize = (this->MaxSize)\*2;

DataTable<KeyField, DataField> \*\* NewDATATAB = new DataTable<KeyField, DataField>\*[NextSize];

for (int i = 0; i<this->MaxSize; i++)

NewDATATAB[i] = this->DATATAB[i];

for (int i = this->MaxSize; i<NextSize; i++)

NewDATATAB[i] = NULL;

delete[] this->DATATAB;

this->DATATAB = NewDATATAB;

this->MaxSize = NextSize;

};

template <class KeyField, class DataField>

void UnSortTab< KeyField, DataField> ::Insert(const KeyField &KEY, const DataField &DATA)

{

if ((double)this->Top / (double)this->MaxSize > 0.5)

Realloc();

for (int i = 0; i < this->Top; i++)

{

if ((\*(this->DATATAB[i])).Key == KEY)

{

throw "duplicated key";

}

}

this->DATATAB[this->Top++] = new DataTable<KeyField, DataField>(KEY, DATA);

};

template <class KeyField, class DataField>

void UnSortTab< KeyField, DataField> ::Delete(const KeyField &KEY)

{

int i = 0;

int flag = 0;

while ((i < this->Top) && (flag == 0)) {

if ((\*(this->DATATAB[i])).Key == KEY) flag = 1;

i++;

}

if (flag == 1) {

i--;

delete this->DATATAB[i];

while (i < this->Top - 1) {

this->DATATAB[i] = this->DATATAB[i + 1];

i++;

}

this->DATATAB[i] = NULL;

this->Top--;

}

else cout<< "Элемента нет в АнСорт"<<'\n';

};

template <class KeyField, class DataField>

DataTable<KeyField, DataField>\* UnSortTab< KeyField, DataField> ::Search(const KeyField &KEY) const

{

int i = 0;

while (i < this->Top)

{

if ((\*(this->DATATAB[i])).Key == KEY)

return this->DATATAB[i];

i++;

}

throw "element doesn't exist";

};

## Приложение 10. Sort\_Tab.h

#pragma once

#include "Tab.h"

template <class KeyField, class DataField>

class SortTab : public Table<KeyField, DataField>

{

public:

SortTab(int SIZE = MIN\_SIZE) : Table<KeyField, DataField>(SIZE) {};

virtual ~SortTab() {};

virtual void Insert(const KeyField &KEY, const DataField &DATA);

virtual void Delete(const KeyField &KEY);

virtual int GetCurrent() { return current; };

virtual DataTable<KeyField, DataField>\* Search(const KeyField &KEY) const;

private:

virtual void Realloc();

int BinSearch(const KeyField &Key) const;

};

template <class KeyField, class DataField>

void SortTab <KeyField, DataField> ::Realloc()

{

int NextSize = (this->MaxSize)\*2;

DataTable <KeyField, DataField> \*\* NewDataTab = new DataTable <KeyField, DataField>\*[NextSize];

for (int i = 0; i<this->MaxSize; i++)

NewDataTab[i] = this->DATATAB[i];

for (int i = this->MaxSize; i<NextSize; i++)

NewDataTab[i] = NULL;

delete[] this->DATATAB;

this->DATATAB = NewDataTab;

this->MaxSize = NextSize;

}

template <class KeyField, class DataField>

int SortTab <KeyField, DataField> ::BinSearch(const KeyField &Key) const

{

int tail = this->Top - 1;

int mid = 0;

int start = 0;

while (start <= tail)

{

mid = (tail + start) / 2;

if ((\*(this->DATATAB[mid])).Key < Key)

start = mid + 1;

else if ((\*(this->DATATAB[mid])).Key > Key)

tail = mid - 1;

else return mid;

}

if (start > tail)

mid = start;

return mid;

}

template <class KeyField, class DataField>

void SortTab <KeyField, DataField> ::Insert(const KeyField &KEY, const DataField &DATA)

{

int area = BinSearch(KEY);

int j;

if ((this->DATATAB[area] == NULL) || ((\*(this->DATATAB[area])).Key != KEY))

{

if ((double)this->Top / (double)this->MaxSize > 0.7)

Realloc();

for (j = this->Top; j > area; j--)

this->DATATAB[j] = this->DATATAB[j - 1];

this->DATATAB[j] = new DataTable<KeyField, DataField>(KEY, DATA);

this->Top++;

}

else throw "duplicate key";

}

template <class KeyField, class DataField>

void SortTab <KeyField, DataField> ::Delete(const KeyField &KEY)

{

int area = BinSearch(KEY);

if ((this->DATATAB[area] != NULL) && ((\*(this->DATATAB[area])).Key == KEY))

{

delete this->DATATAB[area];

for (int i = area; i < this->Top; i++)

this->DATATAB[i] = this->DATATAB[i + 1];

this->Top -= 1;

}

else

cout<< "Элемента нет в Сорт"<<'\n';

}

template <class KeyField, class DataField>

DataTable<KeyField, DataField>\* SortTab <KeyField, DataField> ::Search(const KeyField &KEY) const

{

int area = BinSearch(KEY);

if ((this->DATATAB[area] != NULL) && ((\*(this->DATATAB[area])).Key == KEY))

return this->DATATAB[area];

else throw "element doesn't exist";

}

## Приложение 11. Hash\_Tab.h

#pragma once

#include "Tab.h"

#include <string>

#include <cstdlib>

#include <cmath>

using namespace std;

#define FREE 0

#define BUSY 1

#define DELETE -1

template <class KeyField, class DataField>

class HashTab : public Table <KeyField, DataField>

{

private:

void Realloc();

int ReHash(const KeyField &k) const;

int \*STATUS;

public:

HashTab(int SIZE = 20) : Table<KeyField, DataField>(SIZE)

{

STATUS = new int[SIZE];

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

STATUS[i] = FREE;

};

~HashTab() {};

void Insert(const KeyField &KEY, const DataField &DATA);

void Delete(const KeyField &KEY);

void GoNext();

virtual int GetMaxSize() { return MaxSize; };

virtual int GetCurrent() { return current; };

DataTable<KeyField, DataField>\* Search(const KeyField &KEY) const;

};

template <class KeyField, class DataField>

void HashTab<KeyField, DataField>::GoNext()

{

bool flag = true;

int i = current;

while (flag == true)

{

if (STATUS[i] == BUSY)

{

flag = false;

current = i;

}

i++;

}

}

template <class KeyField, class DataField> int HashTab<KeyField, DataField>::ReHash(const KeyField& k) const

{

int g = 17;

int hash = 0;

for (int i = 0; i < k.length(); i++)

hash = g \* hash + k[i];

return (hash % (this->MaxSize));

}

template <class KeyField, class DataField>

void HashTab<KeyField, DataField>::Realloc()

{

int i = 0;

int NextSize = (this->MaxSize)\*2;

DataTable<KeyField, DataField> \*\* NewDATATAB = new DataTable<KeyField, DataField> \*[NextSize];

int \* STATUS1 = new int[NextSize];

for (int i = 0; i < this->MaxSize; i++)

STATUS1[i] = STATUS[i];

for (int i = this->MaxSize; i < NextSize; i++)

STATUS1[i] = FREE;

delete[]STATUS;

STATUS = STATUS1;

//

for (int i = 0; i<this->MaxSize; i++)

NewDATATAB[i] = this->DATATAB[i];

for (int i = this->MaxSize; i < NextSize; i++)

NewDATATAB[i] = NULL;

delete[] this->DATATAB;

this->DATATAB = NewDATATAB;

this->MaxSize = NextSize;

}

template <class KeyField, class DataField>

void HashTab<KeyField, DataField>::Insert(const KeyField &KEY, const DataField &DATA)

{

if ((double)this->current / (double)this->MaxSize > 0.5)

Realloc();

int place = ReHash(KEY);

if (STATUS[place] == FREE)

{

this->DATATAB[place] = new DataTable<KeyField, DataField>(KEY, DATA); STATUS[place] = BUSY; this->current++;

}

else if (STATUS[place] == DELETE)

{

this->DATATAB[place] = new DataTable<KeyField, DataField>(KEY, DATA); STATUS[place] = BUSY;

}

else

{

if ((\*(this->DATATAB[place])).Key == KEY)

{

throw "duplicated key";

}

srand(place);

int laker = rand() % (this->MaxSize);

while (STATUS[laker] == BUSY)

{

laker = rand() % (this->MaxSize);

if ((this->DATATAB[laker] != NULL) && ((\*(this->DATATAB[laker])).Key == KEY))

{

throw "duplicated key";

}

}

this->DATATAB[laker] = new DataTable<KeyField, DataField>(KEY, DATA);

if (STATUS[laker] == 0)

this->current++;

STATUS[laker] = BUSY;

}

}

template <class KeyField, class DataField>

void HashTab<KeyField, DataField>::Delete(const KeyField &KEY)

{

int place = ReHash(KEY);

if (STATUS[place] == 0)

cout<< "Элемента нет в Хэш"<<'\n';

if ((this->DATATAB[place] != NULL) && ((\*(this->DATATAB[place])).Key == KEY))

{

delete this->DATATAB[place];

this->DATATAB[place] = NULL;

STATUS[place] = DELETE;

}

else

{

srand(place);

while ((STATUS[place] == -1) || ((STATUS[place] == 1) && ((\*(this->DATATAB[place])).Key != KEY)))

place = rand() % (this->MaxSize);

if (STATUS[place] == 1)

{

delete this->DATATAB[place];

this->DATATAB[place] = NULL;

STATUS[place] = DELETE;

}

else cout << "Элемента нет в Хэш" << '\n';

}

}

template <class KeyField, class DataField>

DataTable<KeyField, DataField>\* HashTab<KeyField, DataField>::Search(const KeyField &KEY) const

{

int place = ReHash(KEY);

if (STATUS[place] == 0)

throw "element doesn't exist";

if ((this->DATATAB[place] != NULL) && ((\*(this->DATATAB[place])).Key) == KEY)

{

return this->DATATAB[place];

}

else

{

srand(place);

while ((STATUS[place] == -1) || ((STATUS[place] == 1) && ((\*(this->DATATAB[place])).Key != KEY)))

place = rand() % (this->MaxSize);

if (STATUS[place] == 1)

{

return this->DATATAB[place];

}

else

throw "element doesn't exist";

}

}

## Приложение 12. Test\_pol.cpp

#include "Pol\_List.h"

#include "Pol.h"

#include "POl\_Mon.h"

#include "Pol\_Link.h"

#include <gtest.h>

TEST(TPolinom, Create\_Pol)

{

ASSERT\_NO\_THROW(TPolinom a(""));

}

TEST(TPolinom, Two\_Empty\_Pol\_Equal)

{

TPolinom b("");

TPolinom a("");

EXPECT\_EQ(a, b);

}

TEST(TPolinom, Build\_Pol)

{

ASSERT\_NO\_THROW(TPolinom a("5x^3"));

}

TEST(TPolinom, Create\_Pol\_After\_Existing)

{

TPolinom a("5x^3");

ASSERT\_NO\_THROW(TPolinom b(a));

}

TEST(TPolinom, Create\_Pol\_With\_EQ\_Mon)

{

TPolinom a("2x^5+3x^5");

TPolinom b("5x^5");

ASSERT\_NO\_THROW(a, b);

}

TEST(TPolinom, Summa\_1)

{

TPolinom a("10x^2");

TPolinom b("12y^2");

ASSERT\_NO\_THROW(a + b);

}

TEST(TPolinom, Summa\_2)

{

TPolinom a("x^3+2y^2-z");

TPolinom b("y^3-3xy+2xy^3");

ASSERT\_NO\_THROW(a + b);

}

TEST(TPolinom, Summa\_is\_Correct\_1)

{

TPolinom a("5x^2");

TPolinom b("6x^7");

TPolinom c("6x^7+5x^2");

EXPECT\_EQ(c, a + b);

}

TEST(TPolinom, Summa\_is\_Correct\_2)

{

TPolinom a("x+y+z");

TPolinom b("x-2y+z^2");

TPolinom c("2x-y+z^2+z");

EXPECT\_EQ(c, a + b);

}

TEST(TPolinom, Substract\_1)

{

TPolinom a("x+1");

TPolinom b("x-1");

ASSERT\_NO\_THROW(a + b);

}

TEST(TPolinom, Substract\_2)

{

TPolinom a("x^2+5");

TPolinom b("x^2-4");

ASSERT\_NO\_THROW(a + b);

}

TEST(TPolinom, Substract\_is\_Correct\_1)

{

TPolinom a("x+1");

TPolinom b("x-1");

TPolinom c("-2");

EXPECT\_EQ(c, b - a);

}

TEST(TPolinom, Substract\_is\_Correct\_2)

{

TPolinom a("x+2y");

TPolinom b("x-z");

TPolinom c("2y+z");

EXPECT\_EQ(c, a - b);

}

TEST(TPolinom, Mult\_Two\_Pol\_1)

{

TPolinom a("x^3");

TPolinom b("y^4");

ASSERT\_NO\_THROW(a \* b);

}

TEST(TPolinom, Mult\_Two\_Pol\_2)

{

TPolinom a("10x^2");

TPolinom b("12y^2");

ASSERT\_NO\_THROW(a \* b);

}

TEST(TPolinom, Mult\_Two\_Pol\_is\_Correct\_1)

{

TPolinom a("2x^2");

TPolinom b("2y^2");

TPolinom c("4x^2y^2");

EXPECT\_EQ(c, a \* b);

}

TEST(TPolinom, Mult\_Two\_Pol\_is\_Correct\_2)

{

TPolinom a("x+1");

TPolinom b("x-1");

TPolinom c("x^2-1");

EXPECT\_EQ(c, a \* b);

}

TEST(TPolinom, ERROR\_Mult\_St\_Over\_10\_1)

{

TPolinom a("3x^4");

TPolinom b("x^8");

ASSERT\_ANY\_THROW(a \* b);

}

TEST(TPolinom, ERROR\_Mult\_St\_Over\_10\_2)

{

TPolinom a("y^2");

TPolinom b("y^8");

ASSERT\_ANY\_THROW(a \* b);

}

TEST(TPolinom, Group\_Equal\_St\_1)

{

TPolinom a("xyz+xy");

TPolinom b("z+2");

TPolinom c("xyz^2+xyz+2xyz+2xy");

TPolinom d("xyz^2+3xyz+2xy");

EXPECT\_EQ(c, a \* b);

EXPECT\_EQ(c, d);

}

TEST(TPolinom, Group\_Equal\_St\_2)

{

TPolinom a("x+yz");

TPolinom b("2x+yz");

TPolinom c("2x^2+2xyz+xyz+y^2z^2");

TPolinom d("2x^2+3xyz+y^2z^2");

EXPECT\_EQ(c, a \* b);

EXPECT\_EQ(c, d);

}

TEST(TPolinom, Mult\_Pol\_On\_Double\_1)

{

TPolinom a("x^2");

double b = 20.9;

ASSERT\_NO\_THROW(a \* b);

}

TEST(TPolinom, Mult\_Pol\_On\_Double\_2)

{

TPolinom a("x^2+y-z");

double b = 5.5;

ASSERT\_NO\_THROW(a \* b);

}

TEST(TPolinom, Mult\_Pol\_On\_Double\_is\_Correct\_1)

{

TPolinom a("100x^3");

double b = 19.5;

TPolinom c("1950x^3");

EXPECT\_EQ(c, a \* b);

}

TEST(TPolinom, Mult\_Pol\_On\_Double\_is\_Correct\_2)

{

TPolinom a("10x^2-2y+4z");

double b = 3.5;

TPolinom c("35x^2-7y+14z");

EXPECT\_EQ(c, a \* b);

}

TEST(TPolinom, Mult\_Pol\_On\_Double\_OtherSide\_1)

{

double b = 20.9;

TPolinom a("x^2");

ASSERT\_NO\_THROW(b \* a);

}

TEST(TPolinom, Mult\_Pol\_On\_Double\_OtherSide\_2)

{

double b = 5.5;

TPolinom a("x^2+y-z");

ASSERT\_NO\_THROW(b \* a);

}

TEST(TPolinom, Mult\_Pol\_On\_Double\_OtherSide\_is\_Correct\_1)

{

double b = 19.5;

TPolinom a("100x^3");

TPolinom c("1950x^3");

EXPECT\_EQ(c, b\* a);

}

TEST(TPolinom, Mult\_Pol\_On\_Double\_OtherSide\_is\_Correct\_2)

{

double b = 3.5;

TPolinom a("10x^2-2y+4z");

TPolinom c("35x^2-7y+14z");

EXPECT\_EQ(c, b \* a);

}

TEST(TMonom, can\_create\_node)

{

ASSERT\_NO\_THROW(Monom a);

}

TEST(TRingList, can\_create\_list)

{

ASSERT\_NO\_THROW(TRingList<int> l);

}

class EmptyList : public testing::Test

{

protected:

TRingList<int> Example;

public:

EmptyList() {};

~EmptyList() {};

};

TEST\_F(EmptyList, Copy\_Empty)

{

ASSERT\_NO\_THROW(TRingList<int>variable(Example));

}

TEST\_F(EmptyList, Copy\_Empty\_Correct)

{

TRingList<int>variable(Example);

variable.Reset();

EXPECT\_EQ(Example.GetCurrent(), Example.GetNext());

}

TEST\_F(EmptyList, Eq\_Two\_Empty)

{

TRingList<int> variable;

EXPECT\_EQ(true, Example == variable);

}

TEST\_F(EmptyList, Assign\_Two\_Empty)

{

TRingList<int> variable;

ASSERT\_NO\_THROW(Example = variable);

}

TEST\_F(EmptyList, Assign\_Two\_Emptyi\_Correct1)

{

TRingList<int> variable;

Example = variable;

EXPECT\_EQ(Example.GetCurrent(), Example.GetNext());

}

TEST\_F(EmptyList, Insert\_To\_Tail\_For\_Empty)

{

Example.InsertInEnd(5);

Example.Reset();

EXPECT\_EQ(5, Example.GetCurrent()->data);

EXPECT\_EQ(Example.GetCurrent(), Example.GetCurrent()->pNext->pNext);

}

TEST\_F(EmptyList, Search\_For\_Empty)

{

ASSERT\_NO\_THROW(Example.Search(5));

}

TEST\_F(EmptyList, Search\_For\_Empty\_Correct)

{

TLink<int>\* p = Example.Search(6);

EXPECT\_EQ(NULL, p);

}

TEST\_F(EmptyList, Clean\_Empty)

{

ASSERT\_NO\_THROW(Example.Clean());

}

TEST\_F(EmptyList, Can\_Not\_Del\_From\_Empty)

{

ASSERT\_ANY\_THROW(Example.Delete(5));

}

TEST\_F(EmptyList, Del\_From\_Empty\_Correct)

{

Example.InsertInEnd(5);

Example.Delete(5);

Example.Reset();

EXPECT\_EQ(Example.GetCurrent(), Example.GetCurrent()->pNext);

}

TEST\_F(EmptyList, Can\_Not\_Del\_Not\_Memb)

{

Example.InsertInEnd(5);

ASSERT\_ANY\_THROW(Example.Delete(6));

}

TEST\_F(EmptyList, Clean\_Empty\_Correct)

{

Example.Clean();

Example.Reset();

EXPECT\_EQ(Example.GetCurrent(), Example.GetCurrent()->pNext);

}

class ThreeNodesList : public testing::Test

{

protected:

TRingList<int> Example;

public:

ThreeNodesList()

{

Example.InsertInEnd(1);

Example.InsertInEnd(2);

Example.InsertInEnd(3);

}

~ThreeNodesList() {};

};

TEST\_F(ThreeNodesList, 3\_Nodes\_Correct)

{

Example.Reset();

EXPECT\_EQ(1, Example.GetCurrent()->data);

EXPECT\_EQ(2, Example.GetCurrent()->pNext->data);

EXPECT\_EQ(3, Example.GetCurrent()->pNext->pNext->data);

EXPECT\_EQ(Example.GetCurrent()->pNext->pNext->pNext, Example.GetCurrent()->pNext->pNext->pNext->pNext->pNext->pNext->pNext);

}

TEST\_F(ThreeNodesList, Copy\_Not\_Empty)

{

ASSERT\_NO\_THROW(TRingList<int> variable(Example));

}

TEST\_F(ThreeNodesList, Copy\_Of\_Not\_Empty\_Correct)

{

TRingList<int> variable(Example);

variable.Reset();

EXPECT\_EQ(1, variable.GetCurrent()->data);

EXPECT\_EQ(2, variable.GetCurrent()->pNext->data);

EXPECT\_EQ(3, variable.GetCurrent()->pNext->pNext->data);

EXPECT\_EQ(variable.GetCurrent()->pNext->pNext->pNext, variable.GetCurrent()->pNext->pNext->pNext->pNext->pNext->pNext->pNext);

}

TEST\_F(ThreeNodesList, Insert\_In\_End\_For\_Not\_Empty)

{

Example.InsertInEnd(5);

Example.Reset();

EXPECT\_EQ(5, Example.GetCurrent()->pNext->pNext->pNext->data);

EXPECT\_EQ(Example.GetCurrent()->pNext->pNext->pNext->pNext, Example.GetCurrent()->pNext->pNext->pNext->pNext->pNext->pNext->pNext->pNext->pNext);

}

TEST\_F(ThreeNodesList, Search\_For\_Not\_Empty)

{

ASSERT\_NO\_THROW(Example.Search(3));

}

TEST\_F(ThreeNodesList, sSearch\_For\_Not\_Empty\_Correct)

{

Example.Reset();

TLink<int> \* p = Example.Search(3);

TLink<int> \* ptrue = Example.GetCurrent()->pNext->pNext;

EXPECT\_EQ(ptrue, p);

}

TEST\_F(ThreeNodesList, Search\_Not\_a\_Member\_For\_Not\_Empty)

{

ASSERT\_NO\_THROW(Example.Search(5));

}

TEST\_F(ThreeNodesList, Search\_Not\_a\_Member\_for\_Not\_Empty\_Correct)

{

TLink<int> \* p = Example.Search(7);

EXPECT\_EQ(nullptr, p);

}

TEST\_F(ThreeNodesList, can\_clean\_not\_empty\_list)

{

ASSERT\_NO\_THROW(Example.Clean());

}

TEST\_F(ThreeNodesList, Clean\_Not\_Empty\_Correct)

{

Example.Clean();

Example.Reset();

EXPECT\_EQ(Example.GetCurrent(), Example.GetCurrent()->pNext);

}

TEST\_F(ThreeNodesList, Assign\_2\_Not\_Empty)

{

TRingList<int> variable;

variable.InsertInEnd(22);

variable.InsertInEnd(33);

ASSERT\_NO\_THROW(Example = variable);

}

TEST\_F(ThreeNodesList, Assign\_2\_Not\_Empty\_Of\_EQ\_SIZE)

{

TRingList<int> variable;

variable.InsertInEnd(11);

variable.InsertInEnd(22);

variable.InsertInEnd(33);

Example = variable;

Example.Reset();

EXPECT\_EQ(11, Example.GetCurrent()->data);

EXPECT\_EQ(22, Example.GetCurrent()->pNext->data);

EXPECT\_EQ(33, Example.GetCurrent()->pNext->pNext->data);

EXPECT\_EQ(Example.GetCurrent()->pNext->pNext->pNext, Example.GetCurrent()->pNext->pNext->pNext->pNext->pNext->pNext->pNext);

}

TEST\_F(ThreeNodesList, Compare\_2\_EQ\_Correct)

{

TRingList<int> variable(Example);

EXPECT\_EQ(true, Example == variable);

}

class TwoListsTest : public testing::Test

{

protected:

TRingList<int> Example;

TRingList<int> variable;

TRingList<int> z;

public:

TwoListsTest()

{

Example.InsertInEnd(1);

Example.InsertInEnd(2);

Example.InsertInEnd(3);

variable.InsertInEnd(11);

variable.InsertInEnd(22);

}

~TwoListsTest() {}

};

TEST\_F(TwoListsTest, Assign\_2\_Not\_Empty\_Of\_Not\_EQ\_SIZE)

{

ASSERT\_NO\_THROW(Example = variable);

}

TEST\_F(TwoListsTest, Assign\_2\_Not\_Empty\_Of\_Not\_EQ\_SIZE\_Correct)

{

Example = variable;

Example.Reset();

EXPECT\_EQ(11, Example.GetCurrent()->data);

EXPECT\_EQ(22, Example.GetCurrent()->pNext->data);

EXPECT\_EQ(Example.GetCurrent()->pNext->pNext, Example.GetCurrent()->pNext->pNext->pNext->pNext->pNext);

}

TEST\_F(TwoListsTest, Assign\_To\_Empty\_Correct)

{

Example = z;

Example.Reset();

EXPECT\_EQ(Example.GetCurrent()->pNext, Example.GetCurrent());

}

TEST\_F(TwoListsTest, Assign\_Empty\_To\_Not\_Empty\_Correct)

{

z = Example;

EXPECT\_EQ(Example, z);

}

TEST\_F(TwoListsTest, Compare\_2\_Not\_Empty)

{

ASSERT\_NO\_THROW(Example == variable);

}

TEST\_F(TwoListsTest, Compare\_2\_Not\_EQ\_of\_EQ\_SIZE)

{

variable.Insert(3);

EXPECT\_EQ(false, Example == variable);

}

TEST\_F(TwoListsTest, Compare\_2\_Not\_EQ\_With\_Bigger\_Left)

{

EXPECT\_EQ(false, Example == variable);

}

TEST\_F(TwoListsTest, Compare\_2\_Not\_EQ\_With\_Bigger\_Right)

{

EXPECT\_EQ(false, variable == Example);

}

TEST\_F(TwoListsTest, Compare\_With\_Empty\_Left)

{

EXPECT\_EQ(false, z == variable);

}

TEST\_F(TwoListsTest, Compare\_With\_Empty\_Right)

{

int s;

EXPECT\_EQ(false, variable == z);

}

## Приложение 13. test\_tab.cpp

#include "gtest.h"

#include "Tab.h"

#include "Pol.h"

#include "UnSort\_Tab.h"

#include "Sort\_Tab.h"

#include "Hash\_Tab.h"

TEST(UnSortTab, Create)

{

UnSortTab<string, TPolinom> A(20);

TPolinom a("1+x");

ASSERT\_NO\_THROW(A.Insert("1+x", a));

}

class TableTest : public ::testing::Test

{

public:

HashTab <string, TPolinom> HT;

SortTab <string, TPolinom> ST;

UnSortTab <string, TPolinom> UT;

TableTest() : HT(4), ST(4), UT(4)

{

HT.Insert("x", TPolinom("x"));

ST.Insert("x", TPolinom("x"));

UT.Insert("x", TPolinom("x"));

HT.Insert("10+xy", TPolinom("10+xy"));

ST.Insert("10+xy", TPolinom("10+xy"));

UT.Insert("10+xy", TPolinom("10+xy"));

HT.Insert("x^5yz", TPolinom("x^5yz"));

ST.Insert("x^5yz", TPolinom("x^5yz"));

UT.Insert("x^5yz", TPolinom("x^5yz"));

};

};

TEST\_F(TableTest, Insert\_UnSort\_Tab\_1)

{

TPolinom p("xyz");

ASSERT\_NO\_THROW(UT.Insert("xyz", p));

}

TEST\_F(TableTest, Insert\_UnSort\_Tab\_2)

{

TPolinom p("x+y");

ASSERT\_NO\_THROW(UT.Insert("x+y", p));

}

TEST\_F(TableTest, Insert\_Sort\_Tab\_1)

{

TPolinom p("xyz");

ASSERT\_NO\_THROW(ST.Insert("xyz", p));

}

TEST\_F(TableTest, Insert\_Sort\_Tab\_2)

{

TPolinom p("x+y");

ASSERT\_NO\_THROW(ST.Insert("x+y", p));

}

TEST\_F(TableTest, Insert\_Hash\_Tab\_1)

{

TPolinom p("xyz");

ASSERT\_NO\_THROW(HT.Insert("xyz", p));

}

TEST\_F(TableTest, Insert\_Hash\_Tab\_2)

{

TPolinom p("x+y");

ASSERT\_NO\_THROW(UT.Insert("x+y", p));

}

TEST\_F(TableTest, Delete\_UnSort\_Tab\_1)

{

TPolinom p("xyz");

UT.Insert("xyz", p);

ASSERT\_NO\_THROW(UT.Delete("xyz"));

}

TEST\_F(TableTest, Delete\_UnSort\_Tab\_2)

{

TPolinom p("x+z");

UT.Insert("x+z", p);

ASSERT\_NO\_THROW(UT.Delete("x+z"));

}

TEST\_F(TableTest, Delete\_Sort\_Tab\_1)

{

TPolinom p("xyz");

ST.Insert("xyz", p);

ASSERT\_NO\_THROW(ST.Delete("xyz"));

}

TEST\_F(TableTest, Delete\_Sort\_Tab\_2)

{

TPolinom p("x+z");

ST.Insert("x+z", p);

ASSERT\_NO\_THROW(ST.Delete("x+z"));

}

TEST\_F(TableTest, Delete\_Hash\_Tab\_1)

{

TPolinom p("xyz");

HT.Insert("xyz", p);

ASSERT\_NO\_THROW(HT.Delete("xyz"));

}

TEST\_F(TableTest, Delete\_Hash\_Tab\_2)

{

TPolinom p("x+z");

HT.Insert("x+z", p);

ASSERT\_NO\_THROW(HT.Delete("x+z"));

}

TEST\_F(TableTest, Search\_UnSort\_Tab\_1)

{

ASSERT\_NO\_THROW(UT.Search("x"));

}

TEST\_F(TableTest, Search\_UnSort\_Tab\_2)

{

ASSERT\_NO\_THROW(UT.Search("10+xy"));

}

TEST\_F(TableTest, Search\_Sort\_Tab\_1)

{

ASSERT\_NO\_THROW(ST.Search("x"));

}

TEST\_F(TableTest, Search\_Sort\_Tab\_2)

{

ASSERT\_NO\_THROW(ST.Search("10+xy"));

}

TEST\_F(TableTest, Search\_Hash\_Tab\_1)

{

ASSERT\_NO\_THROW(HT.Search("x"));

}

TEST\_F(TableTest, Search\_Hash\_Tab\_2)

{

ASSERT\_NO\_THROW(HT.Search("10+xy"));

}

TEST\_F(TableTest, Reset\_in\_UnSort\_Tab)

{

UT.Reset();

EXPECT\_EQ(UT.GetCurrent(), 0);

}

TEST\_F(TableTest, Reset\_Sort\_Tab)

{

ST.Reset();

EXPECT\_EQ(ST.GetCurrent(), 0);

}

TEST\_F(TableTest, Reset\_Hash\_Tab)

{

HT.Reset();

EXPECT\_EQ(HT.GetCurrent(), 0);

}

TEST\_F(TableTest, Go\_Next\_Sort\_Tab)

{

ASSERT\_NO\_THROW(ST.GoNext());

}

TEST\_F(TableTest, Go\_Next\_UnSort\_Tab)

{

ASSERT\_NO\_THROW(UT.GoNext());

}