# 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»

**ОТЧЕТ**

по учебной практике

**Таблицы**

**Выполнила:** студентка группы 381703-1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гитлина З. М.

Подпись

**Проверил:** ассистент кафедры МОСТ ИИТММ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Волокитин В. Д.

Подпись

Нижний Новгород  
2019

Оглавление

Введение3

1. Постановка учебно-практической задачи4

2. Руководство пользователя6

3. Руководство программиста8

3.1. Описание структур данных 8

3.2. Описание алгоритмов вычисления арифметического выражения 10

3.3. Описание структуры программного комплекса12

Заключение14

Список литературы15

Приложение16

**Введение**

Задача использования таблиц для хранения и операций с данными является очень важной, т.к. имеет очень широкое применение. Например, таблицы используются при:

* организации доступа по имени для управления информацией в привычной для человека форме;
* представлении данных во многих задачах из разных областей приложений (таблицы идентификаторов, номенклатура изделий, словари и т. п.);
* абстрагировании от проблем распределения памяти при размещении данных;
* отображении на ЭВМ такого важного математического понятия как множества.

В отчете приводится постановка задачи выполнения операций над данными таблиц трех типов, а именно над многочленами от трех переменных, и вычислений, связанных с полиномами из таблиц, описание алгоритмов операций, а также дается описание программы и правил ее использования, прилагается текст программы и результаты выполнения данных операций.

**1. Постановка учебно-практической задачи**

*Формулировка задачи:*

Разработать программу, которая предоставляет возможность хранить данные в таблицах 3-х типов: неупорядоченные таблицы, упорядоченные таблицы, хэш-таблицы. В качестве данных необходимо использовать полиномы. Также должно производиться логирование количества произведенных операций во время работ с таблицами.

*Исходные данные:*

* row r – структура, хранящая имя полинома и указатель на сам многочлен от трех переменных,
* Hash H, Ordered O, Disordered D – таблицы, которые содержат структуры row.

*Требуемый результат:*

1. Hash H, Ordered O, Disordered D – таблицы, полученные после вставки/удаления из них полиномов (измененные).
2. Polynom A – многочлен от трех переменных, полученный при поиске полинома в таблице по его имени;
3. Polynom A – многочлен от трех переменных, полученный при выполнении операций над многочленами из таблицы.

Также после каждого действия с таблицами на экран будет выводиться количество произведенных операций для этого.

*Контрольный пример:*

Результаты выполнения операций над таблицами (и многочленами в них) приведены на Рис. 2 – 5.



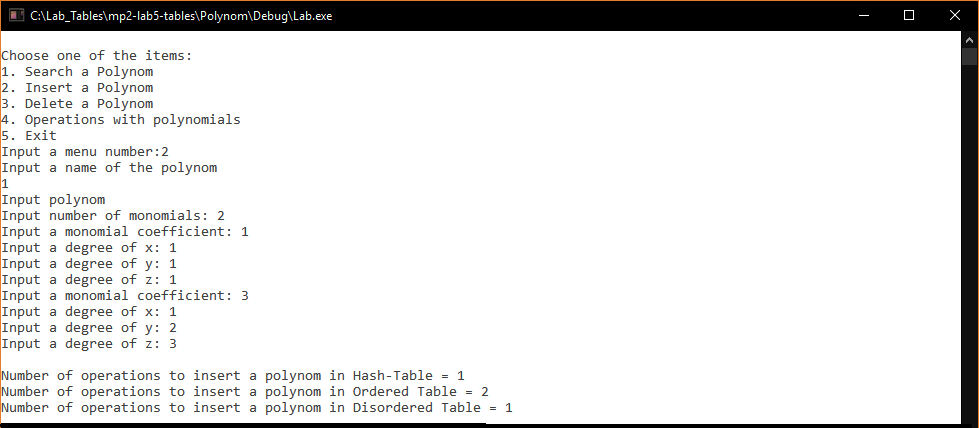


Рис. 1. Результат вставки полинома в таблицы

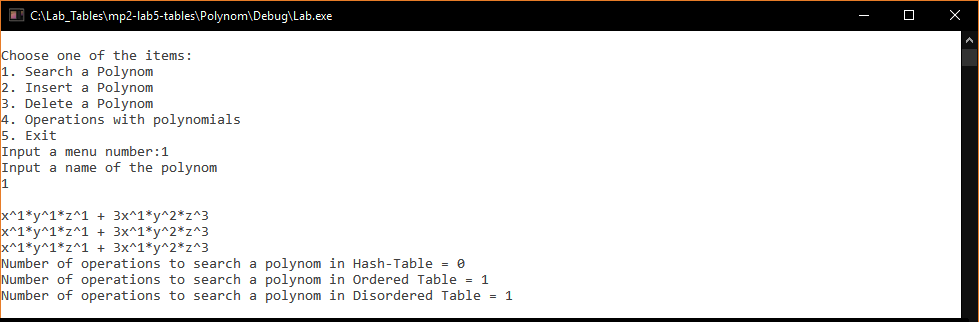


Рис. 2. Результат поиска полинома по имени в таблицах

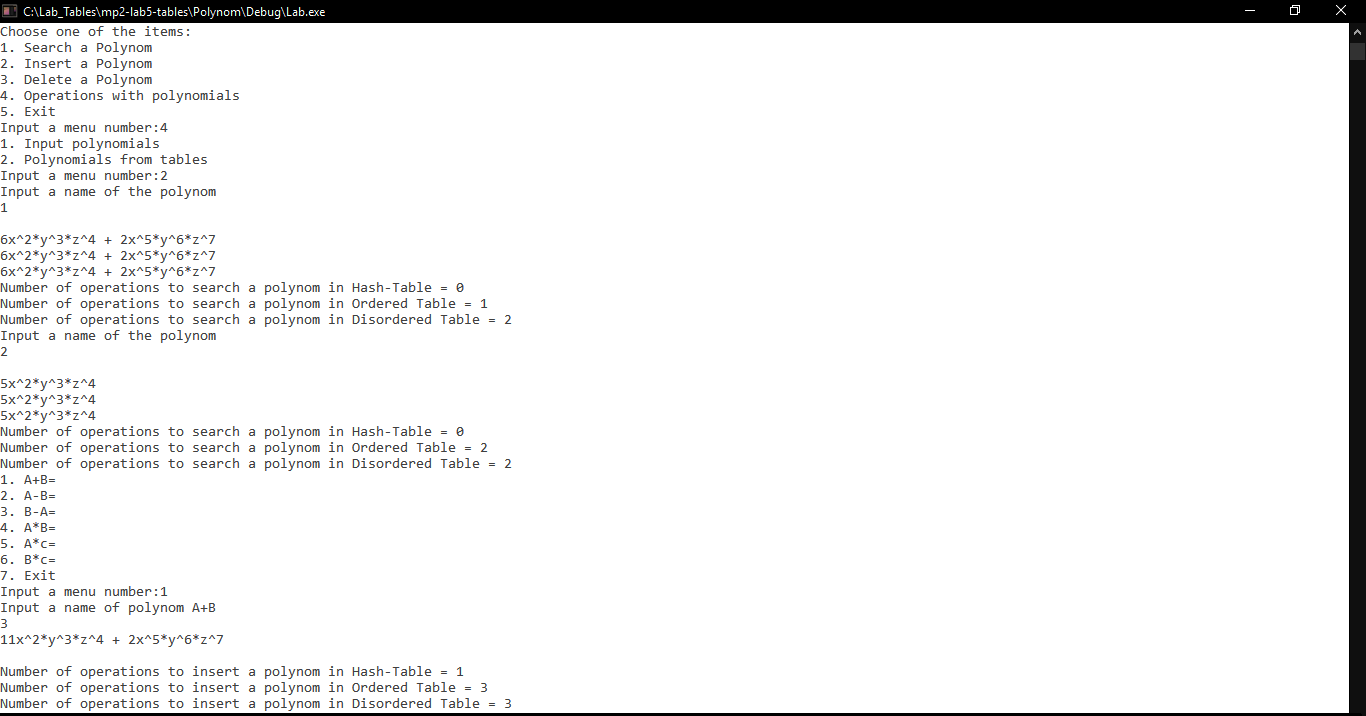


Рис. 3. Результат выполнения вычисления над полиномами из таблиц

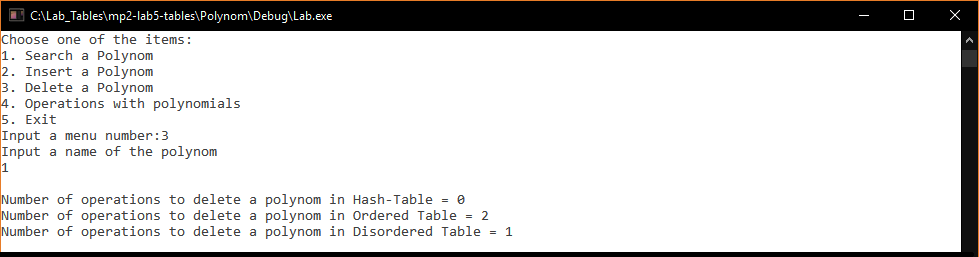


Рис. 4. Результат удаления полинома из таблиц

**2. Руководство пользователя**

Для выполнения лабораторной работы по использованию таблиц для хранения и операций над данными ( а именно многочленов от трех переменных) необходимо запустить на выполнение файл **Lab.exe**. На экране монитора после запуска программы открывается главное меню (Рис. 5).



Рис. 5. Главное меню программы

На экране дисплея появляется запрос "Input a menu number:".

При выборе 1-ого пункта меню программа предлагает следующее: «Search a Polynom». Следует ввести название полинома. При вводе имени многочлена, имеющегося в таблицах, на экране монитора появится сам многочлен с данным именем, если же не существует полинома с таким именем, то программа сообщит об этом, выведя сообщение «There is not such polynom in … table».

При выборе 2-ого пункта меню программа предлагает следующее: «Insert a Polynom». Следует ввести название и сам полином. Если в таблицах уже существует полином с таким именем, то программа сообщит об этом, выведя сообщение об ошибке «This key is taken in … table».

При выборе 3-ого пункта меню программа предлагает следующее: «Delete a Polynom». Следует ввести название полинома. При вводе имени многочлена, имеющегося в таблицах, программа удалит данный полином, если же не существует многочлена с таким именем, то программа сообщит об этом, выведя сообщение «There is not such polynom in … table».

При выборе 4-ого пункта меню программа предлагает следующее: «Operations with polynomials». Затем на экране монитора появляются пункты меню: «Input polynomials», «Polynomials from tables». Следует выбрать один из них. И далее после ввода многочленов или выбора уже существующих из таблиц для выполнения операций над ними необходимо выбрать один из пунктов появившегося меню с 1-ого по 6-ой. В качестве результатов будет выводиться полученный полином.

После каждого действия с таблицами на экран будет выводиться количество произведенных операций для этого.

Для выхода из программы необходимо выбрать 5 пункт меню.

**3. Руководство программиста**

**3. 1. Описание структур данных**

В ходе выполнения данной лабораторной работы был создан абстрактный класс таблиц (class Table). В каждой таблице хранятся записи в виде структур (struct row), содержащих имя полинома и указатель на него. Имя полинома играет роль некоторого наименования, по которому мы идентифицируем полиномы во время работ с таблицами. Название полинома, то есть его ключ определяет расположение данных в таблицах. Указатель на полином – это указатель на данные, с которыми мы работаем. Интерфейс использовавшегося класса приведен ниже:

class Table {

protected:

int counter;//счетчик произведенных операций

public:

Table() {}//конструктор по умолчанию

virtual ~Table() {}//виртуальный деструктор

virtual void Insert(row&) = 0;//чисто виртуальная функция вставки записи в таблицу

virtual void Delete(string&) = 0;//чисто виртуальная функция удаления записи из таблицы

virtual Polynom\* Search(string&) = 0;//чисто виртуальная функция поиска записи в таблице

int Count() { return counter; }//логирование количества произведенных операций

};

Для представления упорядоченных, неупорядоченных и хэш-таблиц были реализованы классы наследники class Table – class Disordered, class Ordered, class Hash.

class Disordered : public Table

{

private:

vector<row> v;//массив структур row, хранящих имя полинома и указатель на него

public:

Disordered();//конструктор по умолчанию

~Disordered() {}//деструктор

void Insert(row&);//вставка в таблицу «записи»

void Delete(string&);//удаление из таблицы «записи»

Polynom\* Search(string&);//поиск полинома в таблице

};

class Ordered : public Table

{

private:

vector<row> v; //массив структур row, хранящих имя полинома и указатель на него

int Binary\_search(string&);//бинарный поиск в массиве по строке (имени полинома)

public:

Ordered();//конструктор по умолчанию

~Ordered() {}//деструктор

void Insert(row&);//вставка в таблицу «записи»

void Delete(string&);//удаление из таблицы «записи»

Polynom \*Search(string&);// поиск полинома в таблице

};

class Hash : public Table

{

private:

vector<list<row>> v;//массив списков структур row, хранящих имя полинома и указатель на него

unsigned int MurmurHash2(string&);//хэш-функция

public:

Hash();//конструктор по умолчанию

~Hash() {}//деструктор

void Insert(row&);//вставка в таблицу «записи»

void Delete(string&);//удаление из таблицы «записи»

Polynom\* Search(string&);// поиск полинома в таблице

};

В данной лабораторной работе также потребовалось создание структуры (struct row) для хранения имени полинома и указателя на него. Интерфейс структуры приведен ниже:

struct row {

string key;//имя полинома

Polynom\* data;//указатель на полином

};

**3.2. Описание алгоритмов операций над многочленами**

В программах комплекса реализованы алгоритмы операций вставки, поиска и удаления записей в таблицы. Приведем общее описание методов выполнения этих операций.

**Операция вставки записи в таблицу**

void Insert(row&) – метод класса, реализующий вставку записи в таблицу, который на вход принимает ссылку на запись.

*Вставка в неупорядоченную таблицу*

Идея операции вставки записи в неупорядоченную таблицу состоит во вставке записи в конец таблицы, то есть в конец массива структур row. Необходимо учитывать, что вставка полинома может выполняться, только если в таблице еще не существует полинома с данным именем. При этом считается, что произведена одна операция.

*Вставка в упорядоченную таблицу*

Идея операции вставки записи в упорядоченную таблицу состоит в определении нужной строки таблицы для вставки, а затем помещении записи в нее. Вначале выполняется бинарный поиск по названию полинома, являющегося ключом, нужной строки для вставки записи в массив структур row, а затем непосредственно выполняется вставка структуры на найденную позицию в массиве (таблице). Необходимо учитывать, что вставка полинома может выполняться, только если в таблице еще не существует полинома с данным именем. При этом считается, что выполнено количество операций, на одно большее, чем количество итераций цикла в бинарном поиске.

*Вставка в хэш-таблицу*

Идея операции вставки записи в хэш-таблицу состоит в определении нужной строки таблицы для вставки, а затем помещении записи в конец списка, находящегося в ней. Вначале вычисляется значение нужной строки для вставки записи в массив списков структур row при помощи хэш-функции от названия полинома, являющегося ключом, а затем непосредственно выполняется вставка полинома в конец списка структур, находящегося на найденной позиции в массиве (таблице). Необходимо учитывать, что вставка полинома может выполняться, только если в таблице еще не существует полинома с данным именем. При этом считается, что выполнено количество операций, на одно большее, чем количество итераций цикла во время прохода по списку в данной строке таблицы.

**Операция удаления записи из таблицы**

void Delete(string&) – метод класса, реализующий удаление записи из таблицы, который на вход принимает ссылку на имя полинома, то есть ключ.

*Удаление из неупорядоченной таблицы*

Идея операции удаления записи из неупорядоченной таблицы состоит в последовательном прохождении в цикле по таблице, то есть по массиву структур row, до тех пор, пока не будет найден полином с данным ключом, или пока не закончится таблица. Если полином с искомым именем не будет найден, то никакие действия не выполняются. При этом считается, что произведено столько операций, сколько выполнено итераций в цикле прохода по таблице до момента нахождения полинома с нужным ключом.

*Удаление из упорядоченной таблицы*

Идея операции удаления записи из упорядоченной таблицы состоит в определении нужной строки таблицы для удаления, а затем исключения записи из нее. Вначале выполняется бинарный поиск по названию полинома, являющегося ключом, нужной строки для удаления записи из массива структур row, а затем непосредственно выполняется удаление структуры row из таблицы, то есть удаление ячейки массива по полученному индексу. Если полином с искомым именем не будет найден, то никакие действия не выполняются. При этом считается, что выполнено количество операций, на одно большее, чем количество итераций цикла в бинарном поиске.

*Удаление из хэш-таблицы*

Идея операции удаления записи из хэш-таблицы состоит в определении нужной строки таблицы для удаления, а затем нахождения необходимой записи в списке, находящемся в ней. Вначале вычисляется значение нужной строки для удаления записи из массива списков структур row при помощи хэш-функции от названия полинома, являющегося ключом, а затем в цикле совершается последовательный проход по списку структур, находящемуся на найденной позиции в массиве, до тех пор, пока не будет найден полином с данным ключом, или пока не закончится список, далее непосредственно выполняется удаление данного звена списка. Если полином с искомым именем не будет найден, то никакие действия не выполняются. При этом считается, что выполнено количество операций, на одно большее, чем количество итераций цикла во время прохода по списку в данной строке таблицы.

**Операция поиска записи в таблице**

Polynom\* Search(string&) – метод класса, реализующий поиск записи в таблице, который на вход принимает ссылку на имя полинома, то есть ключ.

*Поиск в неупорядоченной таблице*

Идея операции поиска записи в неупорядоченной таблице состоит в последовательном прохождении в цикле по таблице, то есть по массиву структур row, до тех пор, пока не будет найден полином с данным ключом, или пока не закончится таблица. При этом считается, что произведено столько операций, сколько выполнено итераций в цикле прохода по таблице до момента нахождения полинома с нужным ключом.

*Поиск в упорядоченной таблице*

Идея операции поиска записи в упорядоченной таблице состоит в определении нужной строки таблицы при помощи бинарного поиска по названию полинома, являющегося ключом. При этом считается, что выполнено количество операций, равное количеству итераций цикла в бинарном поиске.

*Поиск в хэш-таблице*

Идея операции поиска записи в хэш-таблице состоит в определении нужной строки таблицы для поиска, а затем нахождения необходимой записи в списке, находящемся в ней. Вначале вычисляется значение нужной строки для поиска записи из массива списков структур row при помощи хэш-функции от названия полинома, являющегося ключом, а затем в цикле совершается последовательный проход по списку структур, находящемуся на найденной позиции в массиве, до тех пор, пока не будет найден полином с данным ключом, или пока не закончится список. При этом считается, что выполнено количество операций, равное количеству итераций цикла во время прохода по списку в данной строке таблицы.

**3.3 .Описание структуры программного комплекса**

Программный комплекс представлен программой, состоящей из пятнадцати файлов:

1. В файле Table.h представлены интерфейс класса Table и структуры row;
2. В файле Disordered.h представлен интерфейс класса Disordered;
3. В файле Ordered.h представлен интерфейс класса Ordered;
4. В файле Hash.h представлен интерфейс класса Hash;
5. В файле Disordered.cpp реализованы методы класса Disordered;
6. В файле Ordered.cpp реализованы методы класса Ordered;
7. В файле Hash.cpp реализованы методы класса Hash;
8. Файл Test\_ Disordered.cpp содержит тесты для класса Disordered;
9. Файл Test\_ Ordered.cpp содержит тесты для класса Ordered;
10. Файл Test\_ Hash.cpp содержит тесты для класса Hash;
11. В файле Polynom.h представлены интерфейс классов Polynom и Monom, а также структуры m и прототипы дружественных функций ввода и вывода;
12. В файле Polynom.cpp реализованы методы класса Polynom и функции ввода и вывода;
13. Файл Test\_Polynom.cpp содержит тесты для класса Polynom.
14. Файл Tests.cpp содержит вызов всех google tests.
15. В файле Lab.cpp находится основная программа.

**Заключение**

В лабораторной работе разработан и создан программный комплекс, включающий в себя возможность выполнения операций вставки, поиска, удаления записей в таблицу, а также вычислений с полиномами из таблиц. Программный комплекс позволяет в режиме диалога выбирать операцию, которую пользователь хочет выполнить. После каждой операции программа выводит количество потребовавшихся для ее исполнения действий в каждой из таблиц. Исходя из этого, можно сделать вывод, что при большом количестве записей наиболее эффективно работают хэш-таблицы, а наименее эффективно – неупорядоченные. Работоспособность данного программного комплекса была проверена с помощью Google-Test.

**Список литературы**

1. Т. Кормен, Ч. Лейзерсон, Р. Ривест Алгоритмы: построение и анализ. М.: МЦНМО, 1999.-960 с., 263 ил.
2. А. Шень Программирование: теоремы и задачи.- 6-е изд., дополненное.- М.: МЦНМО, 2017.-320с. : ил.

**Приложение**

// Файл Disordered.cpp

#include "Disordered.h"

Disordered::Disordered()

{

vector<row> temp;

v = temp;

counter = 0;

}

Polynom\* Disordered::Search(string& \_key) {

for (int i = 0,counter=0; i < v.size(); i++,counter++)

if (v[i].key == \_key)

return v[i].data;

return NULL;

}

void Disordered::Insert(row& str) {

Polynom\* temp = Search(str.key);

if (temp == NULL) {

v.push\_back(str);

counter++;

}

else {

cout << "This key is taken in Disordered table" << endl;

return;//throw("This key is taken");

}

}

void Disordered::Delete(string& \_key) {

for (int i = 0,counter=0; i < v.size(); i++,counter++)

if (v[i].key == \_key) {

v.erase(v.begin() + i);

break;

}

}

// Файл Ordered.cpp

#include "Ordered.h"

Ordered::Ordered()

{

vector<row> temp;

v = temp;

counter = 0;

}

int Ordered::Binary\_search(string& \_key) {

counter = 0;

if (v.empty()) {

counter++;

return -1;//an empty table

}

else {

int pos;

int mid, st, fn;

st = 0;

fn = v.size() - 1;

mid = 0;

while (st <= fn) {

counter++;

mid = (st + fn) / 2;

if (\_key < v[mid].key)

fn = mid - 1;

else

if (\_key > v[mid].key)

st = mid + 1;

else

return mid;

}

return -1;//st > fn

}

}

void Ordered::Insert(row& str) {

int pos = Binary\_search(str.key);

if (pos == -1) {

counter++;

v.push\_back(str);

}

else

if (v[pos].key == str.key) {

cout << "This key is taken in Ordered table" << endl;

return;

}//throw("This key is taken");

else {

counter++;

v.insert(v.begin() + pos, str);

}

}

void Ordered::Delete(string&\_key) {

int pos = Binary\_search(\_key);

if (pos != -1) {

if (v[pos].key == \_key) {

v.erase(v.begin() + pos);

counter++;

}

}

}

Polynom\* Ordered::Search(string &\_key) {

int pos = Binary\_search(\_key);

if ((pos == -1) || (v[pos].key != \_key))

return NULL;

else

return (v[pos].data);

}

// Файл Hash.cpp

#include"Hash.h"

unsigned int Hash::MurmurHash2(string& \_key) {

{

unsigned int len = \_key.size();

const unsigned int m = 0x5bd1e995;

const unsigned int seed = 0;

const int r = 24;

unsigned int h = seed ^ len;

const unsigned char \* data = (const unsigned char \*)\_key.c\_str();

unsigned int k;

while (len >= 4)

{

k = data[0];

k |= data[1] << 8;

k |= data[2] << 16;

k |= data[3] << 24;

k \*= m;

k ^= k >> r;

k \*= m;

h \*= m;

h ^= k;

data += 4;

len -= 4;

}

switch (len)

{

case 3:

h ^= data[2] << 16;

case 2:

h ^= data[1] << 8;

case 1:

h ^= data[0];

h \*= m;

};

h ^= h >> 13;

h \*= m;

h ^= h >> 15;

return h % 1024;

}

}

Hash::Hash() {

vector<list<row>> temp(1024);

v = temp;

list<row>a ;

for (int i = 0; i < 1024; i++)

v[i]=a;

counter=0;

}

void Hash::Insert(row& str) {

counter = 0;

unsigned int h = MurmurHash2(str.key);

if (!v[h].empty())

for (auto it = v[h].begin(); it != v[h].end(); it++,counter++) {

if ((\*it).key == str.key) {

cout << "This key is taken in Hash-Table" << endl;

return;

}//throw("This key is taken");

}

v[h].push\_back(str);

counter++;

}

void Hash::Delete(string&\_key) {

counter = 0;

unsigned int h = MurmurHash2(\_key);

for (auto it = v[h].begin(); it != v[h].end(); it++,counter++) {

if ((\*it).key == \_key) {

v[h].erase(it);

break;

}

}

}

Polynom\* Hash::Search(string&\_key) {

counter = 0;

unsigned int h = MurmurHash2(\_key);

for (auto it = v[h].begin(); it != v[h].end(); it++,counter++) {

if ((\*it).key == \_key)

return (\*it).data;

}

return NULL;

}