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 831703-1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Шибаева И. В.

**Проверил:** ассистент кафедры

МОСТ ИИТММ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Волокитин В.Д.

# Содержание

[**Содержание**](#_f7iuj18fq39k) **2**

[**Введение**](#_30j0zll) **3**

[**Постановка учебно-практической задачи**](#_1fob9te) **4**

[**2. Руководство пользователя**](#_3038lilfa7vp) **5**

[**3. Руководство программиста**](#_f3h51fhyknfo) **6**

[3.1 Описание структур данных](#_oz8vw5f709ra) 6

[3.3 Общее описание структур программного комплекса](#_dby2v5is38gn) 13

[**Заключение**](#_macektxxkt3z) **14**

[**Список литературы**](#_jb2w7zhntuej) **15**

# 

# **Введение**

Таблица - структура хранения для однотипных данных, состоящая из строк и столбцов. Хранение информации в виде таблиц существенно ускоряет доступ данным, избавляет от проблем с их распределением и представляет содержимое в доступной для пользователя форме.

В данном отчете приведен вариант программы для реализации таблиц трех типов, руководство пользователя, а также описание структур данных и алгоритмов.

# Постановка учебно-практической задачи

*Формулировка задачи:*

Разработать программу, которая предоставляет возможность хранить данные в таблицах 3-х типов:

1. Неупорядоченная таблицы
2. Упорядоченная таблица
3. Хэш-таблица

В качестве данных использовать полиномы из лабораторной работы “Арифметические операции с полиномами”.

Работоспособность программы необходимо проверить с помощью Google Test-ов. Кроме того, необходимо разработать пользовательское консольное приложение.

*Исходные данные:*

Комплект полиномов с именами из латинских символов и\или цифр.

*Требуемый результат:*

1. Вставка в три таблицы, вывод результатов логирования
2. Удаление из таблиц, вывод результатов логирования
3. Поиск в трех таблицах, вывод результатов логирования

*Контрольный пример:*

Пусть программа сгенерирует два случайных полинома с названиями PPP и AAA, произведёт вставку, поиск и удаление AAA.

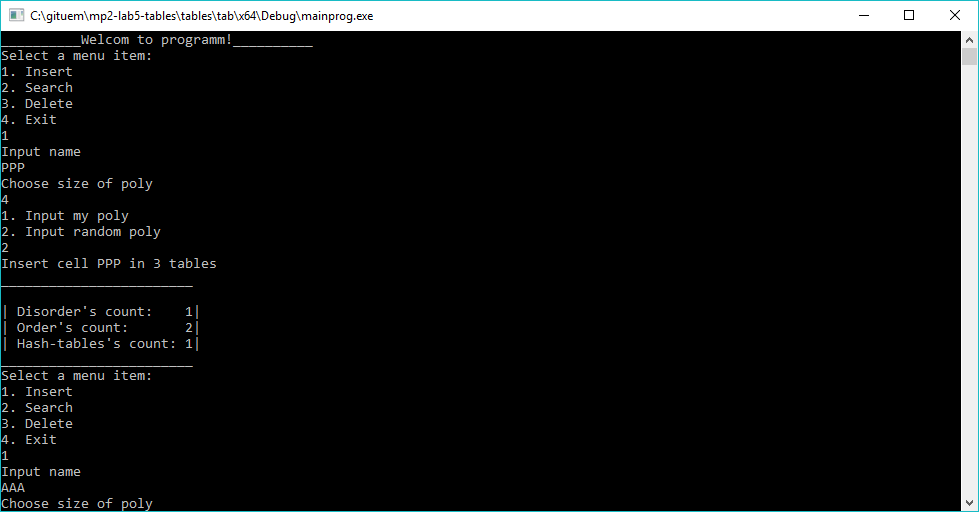


Рис. 1. Результат контрольного примера №1

# 

# Рис.2 Результат контрольного примера №2

# 

# Рис.3 Результаты контрольного примера №3

# 

# 2. Руководство пользователя

Для запуска программы необходимо открыть файл mainprog.exe. На экране консоли появится Меню выбора действий над таблицами.

Программа выведет на экран имя операции и результаты логирования, для вставки же будет предложено ввести полином или сгенерировать его.

# 

# 3. Руководство программиста

### **3.1 Описание структур данных**

В программе имеются структуры для реализации полиномов и для реализации таблиц.

**Структуры полиномов**

Представим наши объекты в виде двусвязного списка, в котором звенья были бы структурами, содержащими степень в виде числа XYZ от 0 до 999, коэффициентов и указателей. Сам полином реализован с помощью класса Poly и имеет одно поле - указатель на голову.

**struct Mon**

double coef; - хранение вещественных коэффициентов

int pow; - хранение степени в формате XYZ

Mon\* next; - хранение указателя на следующий элемент в списке

Mon\* prev; - хранение указателя на предыдущий элемент списка

**class Poly**

Mon\*head; - указатель на “голову” списка

public:

Poly(); - конструктор по умолчанию

Poly(const Poly&); - конструктор копирования

Poly(double\*, int\*, int); - конструктор инициализации

void Insert(double coef, int pow); - функция вставки

void Del(int); - функция удаления

bool operator==(const Poly&) const; - оператор сравнения

Poly& operator=(const Poly&); - оператор присваивания

Poly operator+(Poly &); - оператор сложения

Poly operator-(Poly &); - оператор вычитания

Poly operator\*(double); - оператор, осуществляющий умножение на константу

Poly operator\*(Poly &); - оператор умножения многочлен на многочлен

friend ostream& operator<<(ostream&, Poly &); - дружественная функция вывода

~Poly(); - деструктор;

**Структуры таблиц**

Для таблиц реализован абстрактный класс с основными методами: вставкой, удалением, поиском, а также возвращением результатов логирования. В качестве строки таблицы используется структура cell, содержащая ключ полинома и полином.

Для синхронизации работы с таблицами реализован класс TableComm.

**struct cell**

string name; - ключ

Poly\* pol; - указатель на полином

**class Table**

protected:

int count; - счетчик

public:

Table() {} - конструктор

virtual ~Table() {} - виртуальный деструктор

virtual void Insert(cell) = 0; - виртуальный метод вставки

virtual void Delete(string) = 0; - виртуальный метод удаления

virtual Poly\* Search(string&) = 0; - виртуальный метод поиска

int GetCount() { return count; } - метод возвращения счетчика

void CleanCount() { count = 0; } - удаление значения счетчика

**Неупорядоченные таблицы**

**class Dorder: public Table**

vector<cell> tbl1; - вектор, выполняющий роль неупорядоченной таблицы

public:

Dorder(); - конструктор

~Dorder() {}; - деструктор

void Insert(cell); - вставка в конец

void Delete(string); - удаление

Poly\* Search(string&); - поиск с возвращением полинома

**Упорядоченные таблицы**

**class Order : public Table**

vector<cell> tbl2; - вектор, выполняющий роль упорядоченной таблицы

public:

Order(); - конструктор

~Order() {}; - деструктор

void Insert(cell); - вставка

void Delete(string); - поиск в таблице и удаление

Poly\* Search(string&); - поиск с возвращением полинома

int Position(string); - бинарный поиск по ключу

**Хэш-таблица**

**class Hash : public Table**

vector<list<cell>> tbl3; - вектор из списков

public:

Hash(); - конструктор

~Hash() {}; - деструктор

void Insert(cell); - вставка с помощью хэш-функции

void Delete(string); - удаление

Poly\* Search(string&); - поиск и возвращение полинома

int Hashing(string&); - хэш-функция

**Глобальная таблица полиномов**

**class TableComm**

Order or;

Dorder dor;

Hash hash;

public:

TableComm(); - конструктор

void Insert(cell&); - вставка в 3 таблицы

void Delete(string); - удаление из 3 таблиц

Poly\* Search(string); - возвращение полинома

void RetCounter(); - возвращение всех 3 счетчиков

void CleanCounter(); - очистка всех счетчиков

~TableComm(); - деструктор

**3.2 Описание алгоритмов арифметических действий и работы перегрузок**

**Реализация полиномов**

Для решения поставленной задачи были реализованы алгоритмы и перегрузки арифметических операций, а также сравнение и присваивание.

*1) Функция вставки*

В программе обозначена как void Insert(double coef, int pow). Принимает на вход коэффициент и степень в формате XYZ. Функция находит нужное место для монома и вставляет его. Учитываются случаи, когда список пуст или существует такой моном(в этом случае складывает коэффициенты).

*2) Удаление*

В коде представлено как void Del(int). Последовательно ищет моном, соответствующий запросу, и исключает его из списка. Также выдает сообщение об ошибке, если такого не нашлось.

*3) Оператор сравнения*

bool operator==(const Poly&) const. Возвращает true, если последовательное сравнение степеней и коэффициентов полинома не выдало несостыковок. В противном случае вернет false.

*4) Оператор присваивания*

В коде - Poly& operator=(const Poly&). Сравнивает “головы” двух выбранных списков. Если не равны, левый операнд очищается, и в него поэлементно кладутся мономы правого операнда.

*5) Сложение полиномов*

Представлено оператором Poly operator+(Poly &). Выполняется как последовательное слияние двух списков в третий. За каждый шаг функция сравнивает элементы из первого и из второго полинома и вставляет элемент того, чья степень оказалась меньше. После окончания элементов одного из полинома, в полином суммы сохраняется “хвост” второго, если таковой имеется.

*6) Умножение на константу*

Выполняется через оператор Poly operator\*(double). Коэффициент каждого монома умножается на заданную константу.

*7) Вычитание полиномов*

Выполняется при помощи оператора Poly operator-(Poly &). Реализовано на основе сложения, но предварительно все коэффициенты умножены на -1 с помощью умножения на константу.

*8) Умножение полинома на полином.*

Обозначено в коде программы как Poly operator\*(Poly &). Последовательно умножает каждый моном первого на каждый моном второго полинома: коэффициенты перемножаются, числа степеней складываются(с учетом концепции, где число не может быть больше 999)

**Реализация таблиц**

Было реализовано несколько вариантов функций вставки, удаления, поиска и специфические для таблиц функции.

*1)Функция поиска*

Реализованы в функции Poly\* Search(string). Возвращают указатель на полином.

В неупорядоченной таблице:

Последовательно ищет ключ в таблице;

В упорядоченной таблице:

Использует бинарный поиск в функции int Position(string a), а затем возвращает полином;

В хэш-таблице:

Функция int Hashing(string& a) вычисляет значение как произведение золотого сечения на сумму первого и последнего элемента имени полинома, переведенную в double.

С помощью функции int Hashing(string& a) получает номер в таблице. Затем в списке, лежащем под этим номером, производится поиск нужного элемента. Возвращается полином, если он есть в списке.

*2) Функция вставки*

Выполнены в void Insert(cell a). Вставляет клетку в нужное место.

В неупорядоченной таблице:

Проверяет, нет ли полинома в таблице с помощью функции поиска, и если нет, вставляет в конец таблицы;

В упорядоченной таблице:

Проверяет, есть ли полином в таблице с помощью функции поиска, и если нет, вставляет его в подходящее место;

В хэш-таблице:

Получает номер с помощью int Hashing(string& a) из названия полинома, затем вставляет полином в список по подходящему номеру, предварительно проверив, нет ли повторений.

*3) Функция удаления*

В неупорядоченных таблицах:

С помощью функции поиска ищет нужный номер в таблице, затем удаляет полином. Если такого полинома нет, вернет сообщение об ошибке;

В упорядоченных таблицах:

Аналогично функции в неупорядоченных.

В хэш-таблице:

Через функцию подсчета номера получает место в таблице, затем ищет полином в списке и удаляет его.

Каждая из реализаций снабжена счетчиком операций для итоговой проверки сложности.

**Реализация глобальной таблицы полиномов**

Присутствуют функции поиска, вставки и удаления сразу для трех таблиц. Функция void RetCounter() возвращает счетчики всех операций, а void CleanCounter() очищает их.

### 

### **3.3 Общее описание структур программного комплекса**

Приложение представлено следующими файлами:

1. В файле List.h объявлены структура Mon и класс Poly и прототипы функций для осуществления операций;
2. В файле List.cpp реализованы методы класса Poly и оператор вывода;
3. В файле Tabl.h реализован класс виртуальных функций для таблиц и структура cell;
4. В файлах Dorder.h и Dorder.cpp написан класс Неупорядоченные таблицы;
5. В файлах Order.h и Order.cpp реализован класс Упорядоченные таблицы;
6. В файлах Hash.h и Hash.cpp находится класс хэш-таблиц;
7. В файлах TableComm.h и TableComm.cpp реализованы поля и методы глобального класса таблиц;
8. В файле mainprog.cpp содержится интерфейс пользовательского приложения;
9. Тесты для всех классов содержатся в файлах tDor.cpp, tOrder.cpp и tHash.cpp;
10. Файл tmain.cpp запускает все тесты сразу.

# 

# Заключение

В ходе работы была реализована программа, проверенная с помощью Google Test’ов, и выполнены следующие требования:

1. Работа происходит сразу с таблицами всех типов.
2. Пользователь может положить в таблицу свой полином, с заданием ему некоторого наименования (ключа).
3. Пользователь может запросить (найти в/извлечь из) таблицы полином с заданным наименованием.
4. В ходе выполнения программы у пользователя должна оставаться возможность производить операции с полиномами, а также размещать полученный результат в таблицу.
5. Во время работы программы происходит логирования количества произведенных операций. Пример: Пользователь ввел свой полином и запросил положить его в таблица, программа в ответ выводит на экран или в файл, какое действие произошло и сколько операций потребовалось на нее у разных типов таблиц. Аналогично с поиском и извлечением.

# 

# Список литературы

1. Шилдт, Г. С++ для начинающих: самоучитель / Шилдт, Г. - Изд-во: Эком, 2013г.
2. Т. Кормен, Ч. Лейзерсон, Р. Ривест Алгоритмы: построение и анализ, М.:МЦНМО, 1999.-960 с.б 263 ил.

**Приложение**

**Файл Tabl.h**

struct cell

{

string name;

Poly\* pol;

};

class Table

{

protected:

int count;

public:

Table() {}

virtual ~Table() {}

virtual void Insert(cell) = 0;

virtual void Delete(string) = 0;

virtual Poly\* Search(string&) = 0;

int GetCount() { return count; }

void CleanCount() { count = 0; }

};

**Файл Dorder.h**

class Dorder: public Table

{

vector<cell> tbl1;

public:

Dorder();

~Dorder() {};

void Insert(cell);

void Delete(string);

Poly\* Search(string&);

};

**Файл Order.h**

class Order : public Table

{

vector<cell> tbl2;

public:

Order();

~Order() {};

void Insert(cell);

void Delete(string);

Poly\* Search(string&);

int Position(string);

};

**Файл Hash.h**

class Hash : public Table

{

vector<list<cell>> tbl3;

public:

Hash();

~Hash() {};

void Insert(cell);

void Delete(string);

Poly\* Search(string&);

int Hashing(string&);

};

**Файл TableComm.h**

class TableComm

{

Order or;

Dorder dor;

Hash hash;

public:

TableComm();

void Insert(cell&);

void Delete(string);

Poly\* Search(string);

void RetCounter();

void CleanCounter();

~TableComm();

};

**Файл Dorder.cpp**

void Dorder::Insert(cell el)

{

if (Search(el.name) != NULL)

{

cout <<"This poly in table now" << endl ;

return;

}

tbl1.push\_back(el);

count++;

}

Poly\* Dorder::Search(string& el)

{

for (int i = 0; i < tbl1.size(); i++, count++)

{

if (el == tbl1[i].name)

return tbl1[i].pol;

}

return NULL;

}

void Dorder::Delete(string el)

{

for (int i = 0; i < tbl1.size(); i++, count++)

{

if (el == tbl1[i].name)

{

tbl1.erase(tbl1.begin() + i);

return;

}

}

}

**Файл Order.cpp**

void Order::Insert(cell a)

{

int pp = Position(a.name);

if (pp == -1)

{

if (tbl2.empty())

{

count++;

tbl2.push\_back(a);

return;

}

for (int i = 0; i < tbl2.size(); i++, count++)

{

if (a.name < tbl2[i].name)

{

tbl2.insert(tbl2.begin() + i, a);

return;

}

}

tbl2.push\_back(a);

}

else

{

if (a.name != tbl2[pp].name)

{

count++;

tbl2.insert(tbl2.begin() + pp, a);

}

else

cout << "Polynom in table now";

}

}

Poly\* Order::Search(string& a)

{

int pp = Position(a);

if (pp >= 0)

return tbl2[pp].pol;

else return NULL;

}

int Order::Position(string a)

{

if (!tbl2.empty())

{

int mid = 0, st = 0, fn = tbl2.size() - 1;

while (st <= fn)

{

count++;

mid = (st + fn) / 2;

if (a < tbl2[mid].name)

fn = mid-1;

else

if (a > tbl2[mid].name)

st = mid + 1;

else

return mid;

}

return -1;

}

else

{

count++;

return -1;

}

}

void Order::Delete(string a)

{

int pp = Position(a);

if (pp != -1)

{

if (tbl2[pp].name == a)

{

count++;

tbl2.erase(tbl2.begin() + pp);

}

}

}

**Файл Hash.cpp**

void Hash::Insert(cell a)

{

int mp = Hashing(a.name);

if (Search(a.name) != NULL)

{

cout << "This cell in the table now";

return;

}

count++;

tbl3[mp].push\_back(a);

}

void Hash::Delete(string str)

{

int mi = Hashing(str);

for (auto it = tbl3[mi].begin(); it != tbl3[mi].end(); it++, count++)

{

if ((\*it).name == str)

{

tbl3[mi].erase(it);

return;

}

}

}

Poly\* Hash::Search(string& str)

{

int mi = Hashing(str);

if (tbl3.empty())

{

count++;

return NULL;

}

for (auto it = tbl3[mi].begin(); it != tbl3[mi].end(); it++, count++)

{

if ((\*it).name == str)

return (\*it).pol;

}

return NULL;

}

int Hash::Hashing(string& a)

{

double rez, summ;

double k = 0.6180339887499;

rez = k\*(double)(a[0] + a[a.size()-1]);

return (int)rez%256;

}

**Файл TableComm.cpp**

TableComm::~TableComm()

{

}

void TableComm::Insert(cell& \_cell)

{

dor.Insert(\_cell);

or.Insert(\_cell);

hash.Insert(\_cell);

cout << "Insert cell " <<\_cell.name<<" in 3 tables"<< endl;

}

void TableComm::Delete(string str)

{

dor.Delete(str);

or .Delete(str);

hash.Delete(str);

cout << "Delete cell " << str << " from 3 tables" << endl;

}

Poly\* TableComm::Search(string str)

{

Poly\* rez;

rez = dor.Search(str);

or .Search(str);

hash.Search(str);

cout << "Search "<< str << endl;

return rez;

}

void TableComm::RetCounter()

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl<<endl;

cout<< "| Disorder's count: " << dor.GetCount()<<"|"<< endl;

cout << "| Order's count: " << or .GetCount() <<"|"<< endl;

cout << "| Hash-tables's count: " << hash.GetCount() <<"|"<< endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

}

void TableComm::CleanCounter()

{

dor.CleanCount();

or .CleanCount();

hash.CleanCount();

}

**Файл tDor.cpp**

TEST(Dorder, can\_create)

{

ASSERT\_NO\_THROW(Dorder A);

}

TEST(Dorder, can\_insert)

{

Poly A;

Dorder AA;

cell aaa;

aaa.name = "P1";

aaa.pol = &A;

ASSERT\_NO\_THROW(AA.Insert(aaa));

}

TEST(Dorder, can\_delete)

{

Poly A,B,C;

Dorder AA;

cell aaa,bbb,ccc;

aaa.name = "P1";

bbb.name = "P2";

ccc.name = "P3";

aaa.pol = &A;

bbb.pol = &B;

ccc.pol = &C;

ASSERT\_NO\_THROW(AA.Delete(aaa.name));

}

TEST(Dorder, can\_get\_counter\_value)

{

Poly A, B;

Dorder AA;

cell aaa, bbb;

aaa.name = "P1";

bbb.name = "P2";

aaa.pol = &A;

bbb.pol = &B;

AA.Insert(aaa);

AA.Insert(bbb);

EXPECT\_EQ(3,AA.GetCount());

}

TEST(Dorder, can\_find\_unknown\_poly)

{

Poly A,B;

Dorder AA;

cell aaa,bbb,ccc;

aaa.name = "P1";

bbb.name = "P2";

ccc.name = "Nnnn";

aaa.pol = &A;

bbb.pol = &B;

AA.Insert(aaa);

AA.Insert(bbb);

int\* deg1 = new int[3];

double\* coef1 = new double[3];

for (int i = 0; i < 3; i++)

{

coef1[i] = i;

deg1[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly C(coef1, deg1, 3);

ccc.pol = &C;

EXPECT\_EQ(NULL, AA.Search(ccc.name));

}

TEST(Dorder, can\_find\_poly)

{

Dorder AA;

cell aaa, bbb, ccc;

aaa.name = "P1";

bbb.name = "P2";

ccc.name = "Nnnn";

int\* deg1 = new int[3];

int\* deg2 = new int[3];

int\* deg3 = new int[3];

double\* coef1 = new double[3];

double\* coef2 = new double[3];

double\* coef3 = new double[3];

for (int i = 0; i < 3; i++)

{

coef1[i] = i;

deg1[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

coef2[i] = i+1;

deg2[i] = (i + 2) \* 100 + (i + 1) \* 10 + i + 1;

coef3[i] = i+2;

deg3[i] = (i + 3) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly A(coef1, deg1, 3);

Poly B(coef2, deg2, 3);

Poly C(coef3, deg3, 3);

aaa.pol = &A;

bbb.pol = &B;

ccc.pol = &C;

AA.Insert(aaa);

AA.Insert(bbb);

AA.Insert(ccc);

EXPECT\_EQ(C, \*AA.Search(ccc.name));

}

TEST(Dorder, can\_find\_deleted\_poly)

{

Dorder AA;

cell aaa, bbb, ccc;

aaa.name = "P1";

bbb.name = "P2";

ccc.name = "P3";

Poly A, B, C;

aaa.pol = &A;

bbb.pol = &B;

ccc.pol = &C;

AA.Insert(bbb);

AA.Insert(aaa);

AA.Insert(ccc);

AA.Delete(bbb.name);

EXPECT\_EQ(NULL, AA.Search(bbb.name));

}

**Файл tOrder.cpp**

TEST(Order, can\_create)

{

ASSERT\_NO\_THROW(Order A);

}

TEST(Order, can\_insert)

{

Poly A;

Order AA;

cell aaa;

aaa.name = "P1";

aaa.pol = &A;

ASSERT\_NO\_THROW(AA.Insert(aaa));

}

TEST(Order, can\_delete)

{

Order AA;

cell aaa, bbb;

aaa.name = "P1";

bbb.name = "P2";

int\* deg1 = new int[3];

int\* deg2 = new int[3];

double\* coef1 = new double[3];

double\* coef2 = new double[3];

for (int i = 0; i < 3; i++)

{

coef1[i] = i;

deg1[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

coef2[i] = i + 1;

deg2[i] = (i + 2) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly A(coef1, deg1, 3);

Poly B(coef2, deg2, 3);

aaa.pol = &A;

bbb.pol = &B;

AA.Insert(aaa);

AA.Insert(bbb);

ASSERT\_NO\_THROW(AA.Delete(aaa.name));

}

TEST(Order, can\_check\_counter)

{

Order AA;

cell aaa, bbb, ccc;

aaa.name = "P1";

bbb.name = "P2";

ccc.name = "P3";

Poly A, B, C;

aaa.pol = &A;

bbb.pol = &B;

ccc.pol = &C;

AA.Insert(aaa);

AA.Insert(bbb);

AA.Insert(ccc);

EXPECT\_EQ(8, AA.GetCount());

}

TEST(Order, can\_find\_poly)

{

Order AA;

cell aaa, bbb, ccc;

aaa.name = "P1";

bbb.name = "P2";

ccc.name = "P3";

int\* deg1 = new int[3];

int\* deg2 = new int[3];

int\* deg3 = new int[3];

double\* coef1 = new double[3];

double\* coef2 = new double[3];

double\* coef3 = new double[3];

for (int i = 0; i < 3; i++)

{

coef1[i] = i;

deg1[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

coef2[i] = i + 1;

deg2[i] = (i + 2) \* 100 + (i + 1) \* 10 + i + 1;

coef3[i] = i + 2;

deg3[i] = (i + 3) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly A(coef1, deg1, 3);

Poly B(coef2, deg2, 3);

Poly C(coef3, deg3, 3);

aaa.pol = &A;

bbb.pol = &B;

ccc.pol = &C;

AA.Insert(bbb);

AA.Insert(aaa);

AA.Insert(ccc);

EXPECT\_EQ(C, \*AA.Search(ccc.name));

}

TEST(Order, can\_find\_unknown\_poly)

{

Order AA;

cell aaa, bbb, ccc;

aaa.name = "P1";

bbb.name = "P2";

ccc.name = "P3";

int\* deg1 = new int[3];

int\* deg2 = new int[3];

double\* coef1 = new double[3];

double\* coef2 = new double[3];

for (int i = 0; i < 3; i++)

{

coef1[i] = i;

deg1[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

coef2[i] = i + 1;

deg2[i] = (i + 2) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly A(coef1, deg1, 3);

Poly B(coef2, deg2, 3),C;

aaa.pol = &A;

bbb.pol = &B;

ccc.pol = &C;

AA.Insert(bbb);

AA.Insert(aaa);

EXPECT\_EQ(NULL, AA.Search(ccc.name));

}

TEST(Order, can\_find\_deleted\_poly)

{

Order AA;

cell aaa, bbb, ccc;

aaa.name = "P1";

bbb.name = "P2";

ccc.name = "P3";

Poly A, B, C;

aaa.pol = &A;

bbb.pol = &B;

ccc.pol = &C;

AA.Insert(bbb);

AA.Insert(aaa);

AA.Insert(ccc);

AA.Delete(bbb.name);

EXPECT\_EQ(NULL, AA.Search(bbb.name));

}

**Файл tHash.cpp**

TEST(Hash, can\_create)

{

ASSERT\_NO\_THROW(Hash A);

}

TEST(Hash, can\_insert)

{

Poly A;

Hash AA;

cell aaa;

aaa.name = "P1";

aaa.pol = &A;

ASSERT\_NO\_THROW(AA.Insert(aaa));

}

TEST(Hash, can\_delete)

{

Hash AA;

cell aaa, bbb;

aaa.name = "P1";

bbb.name = "P2";

int\* deg1 = new int[3];

int\* deg2 = new int[3];

double\* coef1 = new double[3];

double\* coef2 = new double[3];

for (int i = 0; i < 3; i++)

{

coef1[i] = i;

deg1[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

coef2[i] = i + 1;

deg2[i] = (i + 2) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly A(coef1, deg1, 3);

Poly B(coef2, deg2, 3);

aaa.pol = &A;

bbb.pol = &B;

AA.Insert(aaa);

AA.Insert(bbb);

ASSERT\_NO\_THROW(AA.Delete(aaa.name));

}

TEST(Hash, can\_find\_poly)

{

Hash AA;

cell aaa, bbb, ccc;

aaa.name = "P1";

bbb.name = "P2";

ccc.name = "P3";

int\* deg1 = new int[3];

int\* deg2 = new int[3];

int\* deg3 = new int[3];

double\* coef1 = new double[3];

double\* coef2 = new double[3];

double\* coef3 = new double[3];

for (int i = 0; i < 3; i++)

{

coef1[i] = i;

deg1[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

coef2[i] = i + 1;

deg2[i] = (i + 2) \* 100 + (i + 1) \* 10 + i + 1;

coef3[i] = i + 2;

deg3[i] = (i + 3) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly A(coef1, deg1, 3);

Poly B(coef2, deg2, 3);

Poly C(coef3, deg3, 3);

aaa.pol = &A;

bbb.pol = &B;

ccc.pol = &C;

AA.Insert(bbb);

AA.Insert(aaa);

AA.Insert(ccc);

EXPECT\_EQ(C, \*AA.Search(ccc.name));

}

TEST(Hash, can\_find\_unknown\_poly)

{

Hash AA;

cell aaa, bbb, ccc;

aaa.name = "P1";

bbb.name = "P2";

ccc.name = "P3";

int\* deg1 = new int[3];

int\* deg2 = new int[3];

double\* coef1 = new double[3];

double\* coef2 = new double[3];

for (int i = 0; i < 3; i++)

{

coef1[i] = i;

deg1[i] = (i + 1) \* 100 + (i + 1) \* 10 + i + 1;

coef2[i] = i + 1;

deg2[i] = (i + 2) \* 100 + (i + 1) \* 10 + i + 1;

}

Poly A(coef1, deg1, 3);

Poly B(coef2, deg2, 3), C;

aaa.pol = &A;

bbb.pol = &B;

ccc.pol = &C;

AA.Insert(bbb);

AA.Insert(aaa);

EXPECT\_EQ(NULL, AA.Search(ccc.name));

}

TEST(Hash, can\_find\_deleted\_poly)

{

Hash AA;

cell aaa, bbb, ccc;

aaa.name = "P1";

bbb.name = "P2";

ccc.name = "P3";

Poly A, B, C;

aaa.pol = &A;

bbb.pol = &B;

ccc.pol = &C;

AA.Insert(bbb);

AA.Insert(aaa);

AA.Insert(ccc);

AA.Delete(bbb.name);

EXPECT\_EQ(NULL, AA.Search(bbb.name));

}

TEST(Hash, correct\_hash)

{

Hash EX;

cell ex;

Poly A;

string str= "Random\_string";

ex.name = str;

ex.pol = &A;

int ex\_h=EX.Hashing(ex.name);

EXPECT\_EQ(ex\_h, EX.Hashing(str));

}