МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ 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**

**Отчет по лабораторной работе**

**«Работа с таблицами»**

**Выполнил:**

студент группы 382003-1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лапин Д.А.

**Проверил:**

к.т.н., сотрудник каф. МОСТ ИИТММ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Арисова А. Н.

Нижний Новгород  
2022

**Содержание**

[**Содержание** 2](#_Toc100056167)

[Введение 3](#_Toc100056168)

[**1** Постановка задачи 4](#_Toc100056169)

[**2** Руководство программиста 5](#_Toc100056170)

[**2.1** Описание структуры программы 5](#_Toc100056171)

[Заключение 9](#_Toc100056177)

[Список литературы 10](#_Toc100056178)

[**Приложение** 11](#_Toc100056179)

# Введение

В лабораторной работе рассматривается вопрос работы с неупорядоченными, упорядоченными и хеш-таблицами.

Разработка приложения для работы с таблицами является практически значимой задачей, поскольку оно позволит облегчить, ускорить и оптимизировать многие задачи.

# Постановка задачи

В данной лабораторной работе необходимо разработать программу, выполняющую арифметические операции с полиномами и работу с таблицами, в которой должна быть реализованы:

Работа с мономами от трех переменных (x, y, z), чья степень находится в пределах от 0 до 9. Cтепень хранится в свернутом виде, а именно в виде трехзначного числа (разряд сотен отвечает за степень переменной x, разряд десятков за степень переменной y, соответственно разряд единиц за степень z). Коэффициентами полинома являются вещественные числа.

Примеры мономов: xyz, zyx, y, x^2y^3z^4, xyyxxzx.

Операций сложения, вычитания, умножения на константу, умножения двух полиномов.

Пользователь имеет право работать с полиномами за счёт выполнения трёх перечисленных операций (с контролем приведения подобных слагаемых, удаления монома с нулевым коэффициентом и сохранением ограничений на степень каждой переменной в составе монома). Операцию умножения полинома на константу можно представить, как умножение полинома на полином, где в качестве одного из аргументов на вход передана константа – частный случай полинома. Должна быть гарантирована корректность выполняемых действий.

Работа с неупорядоченными, упорядоченными и хеш-таблицами. У пользователя должна быть возможность добавить полином в таблицу, удалить его из таблицы, проводить над этим полиномом арифметические операции и заносить результат арифметических операций в таблицу.

# Руководство программиста

## Описание структуры программы

С учетом сформулированных выше тезисов к реализации целесообразной представляется следующая структура программы:

* Table – абстрактный класс, реализующий интерфейс работы над таблицей. С классом работают 1 вспомогательный класс: record – класс, кторый реализует ячейку таблицы.
* UnorderedTable – класс, отвечающий непосредственно за работу с неупорядоченными таблицами, наследник класса Table.
* OrderedTable – класс, отвечающий непосредственно за работу с упорядоченными таблицами, наследник класса Table.
* HashTable – класс, отвечающий непосредственно за работу с хеш-таблицами, наследник класса Table

В каждом классе реализованы методы deleteRecord, find и insert. Которые выполняют удаление записи, поиск и вставку.

Кроме того, есть модуль программ тестирования test\_table.cpp.

Реализация классов находится в приложении.

# Заключение

В ходе данной лабораторной работы было разработано приложение для работы с полиномами, которое поддерживает работу с различными видами таблиц.

# Список литературы

1. Кормен Т. и др. Алгоритмы. Построение и анализ:[пер. с англ.]. – Издательский дом Вильямс, 2009.
2. Шилдт, Герберт. С++: базовый курс, 3-е изд. : Пер. с англ. – СПб.: ООО «Диалектика», 2019.-624 с.
3. Барышева И.В. и др. Под руководством Гергеля В.П. Учебно-методическое пособие. Нижний Новгород: Нижегородский университет, 2017.

Приложение

#pragma once

#include<vector>

#include<algorithm>

#include"TList.h"

#include <fstream>

#define LOGGING

#ifdef LOGGING

void log\_insert(int num) {

std::ofstream out;

out.open("log.txt", std::ios::app);

out << std::endl;

out << "Number of commands, used to do insert:\n";

out << num << std::endl;

out.close();

}

void log\_find(int num) {

std::ofstream out;

out.open("log.txt", std::ios::app);

out << std::endl;

out << "Number of commands, used to do find:\n";

out << num << std::endl;

out.close();

}

void log\_delete(int num) {

std::ofstream out;

out.open("log.txt", std::ios::app);

out << std::endl;

out << "Number of commands, used to do delete:\n";

out << num << std::endl;

out.close();

}

#endif

template <class TElement>

class record {

private:

int key;

TElement value;

public:

record() {}

record(int key, TElement value) :key(key), value(value) {}

record(const std::pair<int, TElement>& p) :key(p.first), value(p.second) {}

record(const record& c) :key(c.key), value(c.value) {}

record& operator=(const record& c) {

key = c.key;

value = c.value;

return \*this;

}

TElement& getValue() {

return value;

}

template <class T>

friend class Table;

template <class U>

friend class UnorderedTable;

template <class P>

friend class OrderedTable;

template <class Q>

friend class HashTable;

template <class T>

friend bool operator==(const record<T>& c, const record<T>& v);

template <class T>

friend bool operator!=(const record<T>& c, const record<T>& v);

};

template <class T>

bool operator==(const record<T>& c, const record<T>& v) {

return (v.key == c.key && v.value == c.value);

}

template <class T>

bool operator!=(const record<T>& c, const record<T>& v) {

return (v.key != c.key && v.value != c.value);

}

template <class TElement>

class Table {

public:

virtual void clear() = 0;

virtual int getSize() = 0;

virtual void deleteRecord(int key) = 0;

virtual void insert(int key, const TElement& value) = 0;

virtual void insert(record<TElement>& value) = 0;

virtual ~Table() = default;

virtual record<TElement> \* find(int key) = 0;

};

template <class TElement>

class UnorderedTable: public Table <TElement> {

private:

std::vector<record<TElement>> data;

int size = 0;

public:

UnorderedTable(){}

UnorderedTable(UnorderedTable const& a):data(a.data), size(a.size) {}

UnorderedTable(const std::initializer\_list<record<TElement>>& l) : data(l), size(l.size()) {}

UnorderedTable& operator=(const UnorderedTable& a) {

if (this != &a) {

data = a.data;

size = a.size;

}

return \*this;

}

record<TElement>\* find(int key) {

for (int i = 0; i < size; i++) {

if (data[i].key == key) {

#ifdef LOGGING

std::ofstream out;

out.open("log.txt", std::ios::app);

out << "\nUNORDERED TABLE";

out.close();

log\_find(i+1);

#endif

return &data[i];

}

}

return 0;

}

int getSize() {

return size;

}

void insert(int key, const TElement& value) {

if (find(key) == 0) {

data.push\_back(record<TElement>(key, value));

size++;

#ifdef LOGGING

std::ofstream out;

out.open("log.txt", std::ios::app);

out << "\nUNORDERED TABLE";

out.close();

log\_insert(1);

#endif

}

else throw std::invalid\_argument("table contains element with this key");

}

void insert(record<TElement>& t) {

insert(t.key, t.value);

}

void clear() {

data.clear();

size = 0;

}

~UnorderedTable() {

data.clear();

}

void deleteRecord(int key) {

if (size != 0) {

for (int i = 0; i < size; i++) {

if (data[i].key == key) {

#ifdef LOGGING

std::ofstream out;

out.open("log.txt", std::ios::app);

out << "\nUNORDERED TABLE";

out.close();

log\_delete(i + 1);

#endif

data[i] = data[size - 1];

size--;

data.resize(size);

return;

}

}

}

else {

throw std::invalid\_argument("table is empty");

}

throw std::invalid\_argument("no element with this key in the table");

}

TElement& operator[](int key) {

auto t = find(key);

if (t != 0) return t->value;

else throw std::invalid\_argument("no record in table with this key");

}

};

template <class TElement>

class OrderedTable : public Table <TElement> {

private:

std::vector<record<TElement>> data;

int size = 0;

public:

OrderedTable() {}

OrderedTable(OrderedTable const& a) :data(a.data), size(a.size) {}

OrderedTable(const std::initializer\_list<record<TElement>>& l) :data(l), size(l.size()) {

std::sort(data.begin(), data.end(), [](const record<TElement>& first, const record<TElement>& second) {

return first.key < second.key;

});

}

TElement& operator[](int key) {

auto t = find(key);

if (t != 0) return t->value;

else throw std::invalid\_argument("no record in table with this key");

}

int getSize() {

return size;

}

OrderedTable& operator=(const OrderedTable& a) {

if (this != &a) {

data = a.data;

size = a.size;

}

return \*this;

}

void clear() {

data.clear();

size = 0;

}

~OrderedTable() {

data.clear();

}

record<TElement>\* find(int key) {

auto it = std::lower\_bound(data.begin(), data.end(), record<TElement>(key, TElement()),

[](const record<TElement>& first, const record<TElement>& second) {

return first.key < second.key;

});

#ifdef LOGGING

std::ofstream out;

out.open("log.txt", std::ios::app);

out << "\nORDERED TABLE";

out.close();

log\_find(log2(size));

#endif

if (it == data.end() || it->key != key)

return nullptr;

else return &(\*it);

}

void insert(int key, const TElement& value) {

auto it = std::lower\_bound(data.begin(), data.end(), record<TElement>(key, TElement()),

[](const record<TElement>& first, const record<TElement>& second) {

return first.key < second.key;

});

if (it == data.end() || it->key != key) {

int pos = it - data.begin();

data.push\_back(record<TElement>(key, value));

for (int i = size; i > pos; i--) {

data[i] = data[i - 1];

}

data[pos] = record<TElement>(key, value);

size++;

#ifdef LOGGING

std::ofstream out;

out.open("log.txt", std::ios::app);

out << "\nORDERED TABLE";

out.close();

log\_insert(log2(size)+pos);

#endif

}

else throw std::invalid\_argument("table contains element with this key");

}

void insert(record<TElement>& t) {

insert(t.key, t.value);

}

void deleteRecord(int key) {

if (size != 0) {

auto it = std::lower\_bound(data.begin(), data.end(), record<TElement>(key, TElement()),

[](const record<TElement>& first, const record<TElement>& second) {

return first.key < second.key;

});

if (it != data.end() && it->key == key) {

int pos = it - data.begin();

for (int i = pos; i < size - 1; i++) {

data[i] = data[i + 1];

}

size--;

data.resize(size);

#ifdef LOGGING

std::ofstream out;

out.open("log.txt", std::ios::app);

out << "\nORDERED TABLE";

out.close();

log\_delete(log2(size) + pos);

#endif

}

else throw std::invalid\_argument("no element with this key in the table");

}

else {

throw std::invalid\_argument("table is empty");

}

}

};

template <class TElement>

class HashTable : public Table <TElement> {

private:

std::vector<TList<record<TElement>>> data;

int size = 0;

int NUM\_SLOTS=1000;

int PRIME = 1013;

int hashfunction(int key) {

return ((5 \* key + 7) % PRIME) % NUM\_SLOTS;

}

public:

HashTable() {

data.resize(NUM\_SLOTS);

}

HashTable(HashTable const& a) :data(a.data), size(a.size) {}

HashTable(const std::initializer\_list<record<TElement>>& l) {

data.resize(NUM\_SLOTS);

for (auto it = l.begin(); it != l.end(); it++)

insert((\*it).key, (\*it).value);

}

record<TElement>\* find(int key) {

int temp = hashfunction(key);

#ifdef LOGGING

int op = 0;

#endif

for (auto t = data[temp].begin(); t != data[temp].end(); t++) {

#ifdef LOGGING

op++;

#endif

if ((\*t).key == key) {

#ifdef LOGGING

std::ofstream out;

out.open("log.txt", std::ios::app);

out << "\nHASH TABLE";

out.close();

log\_find(op);

#endif

return &(\*t);

}

}

return 0;

}

void insert(int key, const TElement& value) {

if (find(key) == 0) {

size++;

data[hashfunction(key)].push\_back(record<TElement>(key,value));

#ifdef LOGGING

std::ofstream out;

out.open("log.txt", std::ios::app);

out << "\nHASH TABLE";

out.close();

log\_insert(1);

#endif

}

else throw std::invalid\_argument("element with such key exists in table");

}

TElement& operator[](int key) {

auto t = find(key);

if (t != 0) return t->value;

else throw std::invalid\_argument("no record in table with this key");

}

void insert(record<TElement>& t) {

insert(t.key, t.value);

}

int getSize() {

return size;

}

HashTable& operator=(const HashTable& a) {

if (this != &a) {

data = a.data;

size = a.size;

}

return \*this;

}

void clear() {

data.clear();

size = 0;

}

~HashTable() {

clear();

}

void deleteRecord(int key) {

if (size != 0) {

if (!data[hashfunction(key)].empty()) {

#ifdef LOGGING

int op = 1;

#endif

if ((\*data[hashfunction(key)].begin()).key == key) {

data[hashfunction(key)].remove\_first();

size--;

return;

}

for (auto it = data[hashfunction(key)].begin(); it.next() != data[hashfunction(key)].end(); it++) {

#ifdef LOGGING

op++;

#endif

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

data[hashfunction(key)].erase(it);

size--;

#ifdef LOGGING

std::ofstream out;

out.open("log.txt", std::ios::app);

out << "\nHASH TABLE";

out.close();

log\_delete(op);

#endif

return;

}

}

throw std::invalid\_argument("no element in list with such key");

}

else throw std::invalid\_argument("no element in list with such key");

}

else {

throw std::invalid\_argument("table is empty");

}

}

};