МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное автономное образовательное учреждение

высшего образования

**«Национальный исследовательский**

**Нижегородский государственный университет им. Н.И. Лобачевского»**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения суперкомпьютерных технологий**

Направление подготовки «Прикладная математика и информатика»

**ОТЧЕТ**

по учебной практике

**Организация доступа к данным по ключу**

**Выполнила:** студент группы 381603-1  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Захарова Д. Д.

**Проверила:** к.т. н., доцент каф. МОСТ института ИТММ  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Кустикова В.Д.

Нижний Новгород

2018

**Содержание**

[Введение 3](#_Toc515149136)

[1 Постановка задачи 4](#_Toc515149137)

[2 Руководство пользователя 5](#_Toc515149138)

[3 Руководство программиста 6](#_Toc515149139)

[3.1 Описание структуры программы 6](#_Toc515149140)

[3.2 Описание структуры данных 7](#_Toc515149141)

[3.3 Описание программной реализация 8](#_Toc515149142)

[Заключение 12](#_Toc515149143)

[Список литературы 13](#_Toc515149144)

[Приложение 14](#_Toc515149145)

[Приложение A. table.h. 14](#_Toc515149146)

[Приложение B. tableUnorder.h 15](#_Toc515149147)

[Приложение C. tableOrder.h 16](#_Toc515149148)

[Приложение D. tableHash.h 17](#_Toc515149149)

[Приложение E. main\_table.cpp 20](#_Toc515149150)

# Введение

Таблицы являются одними из наиболее распространенных структур данных, используемых при создании математического обеспечения. Таблицы широко применяются во многих областях науки, так как являются удобным и наглядным средством хранения и отображения информации.

Под таблицей следует понимать динамическую структуру данных, которая в каждый момент выполнения вычислений состоит из конечного набора строк (записей), структура строк может быть различной, но обязательном является поле, задающее имя (ключ) записи.

Однако данные в таблицах могут располагаться разными способами. Поэтому в данной лабораторной работе мы рассмотрим основные типы представления данных в таблицах.

# Постановка задачи

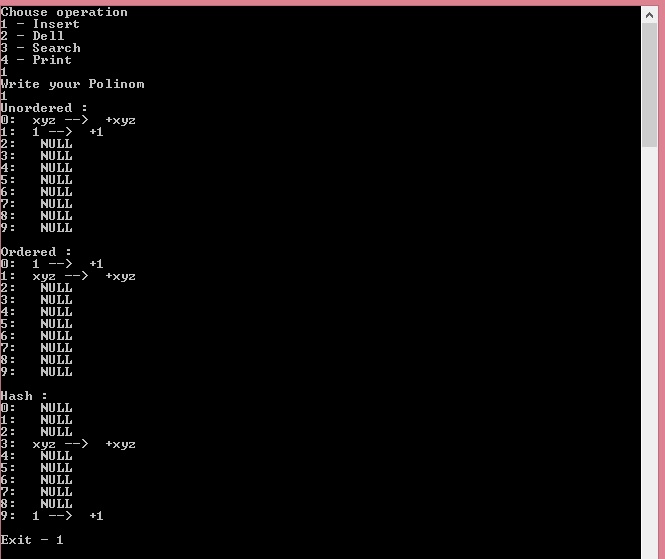
*Формулировка задачи:*

Необходимо разработать программу, выполняющую следующие операции в таблицах: вставка, удаление, поиск элементов (полиномов). Таблицы могут быть трёх видов: несортированные, сортированные и хэш-таблицы, которые хранят в себе значение—полином и ключ—строка. Полином может состоять из трех переменных (x, y, z). Степень переменных полинома принимает значения от 0 до 9. Коэффициенты перед мономами—вещественные числа. В конечном итоге, программа должна вывести пользователю результат.

*Исходные данные:*

Вводится размер таблиц и полиномы, которые будут храниться в таблицах.

*Требуемый результат:*

Программа правильно выполняет вставку, поиск и удаление элементов, заданных пользователем.

1. Пример работы программы.

# Руководство пользователя

1. Для начала работы программы необходимо открыть файл sample.exe.

Допустимые символы: x, y, z

1, 2, 3, 4, 5, 6, 7, 8, 9, 0

+, -, ^

1. Вводится размер таблицы.
2. Выбирается операция.
3. Вводится полином.
4. ****Пользователь получает ответ (вывод таблиц/вывод элемента/сообщение об ошибке) .
5. Выбор операции

# Руководство программиста

## Описание структуры программы

Программа состоит из 4 проектов.

1. **gtest** – статическая библиотека google-тестов.

gtest.h – заголовочный файл для google—тестов.

gtest.cpp – source-файл для google—тестов.

1. **table** – статическая библиотека для полиномов.

node.h—заголовочный файл, который содержит объявление шаблонного класса node.

monom.h—заголовочный файл, который содержит объявление шаблонного класса monom.

list.h—заголовочный файл, который содержит объявление и реализацию шаблонного класса list.

polynom.h—заголовочный файл, который содержит объявление шаблонного класса polynom.

table.h—заголовочный файл, который содержит объявление шаблонный, абстрактный класс таблиц.

tableUnorder.h—заголовочный файл, который содержит объявление шаблонного класса, унаследованного от абстрактного класса таблиц.

tableOrder.h—заголовочный файл, который содержит объявление шаблонного класса, унаследованного от абстрактного класса таблиц.

tableHash.h—заголовочный файл, который содержит объявление шаблонного класса, унаследованного от абстрактного класса таблиц.

polynom.cpp—файл, который содержит реализацию шаблонного класса polynom.

1. **samples**—консольное приложение.

main\_table.cpp—файл, который содержит функцию main().

1. **test**—консольное приложение.

test\_main.cpp—файл, который содержит реализацию тестов для проверки корректности работы функции main().

test\_list.cpp— файл, который содержит реализацию тестов для проверки корректности работы класса list.

test\_polynom.cpp— файл, который содержит реализацию тестов для проверки корректности работы класса polynom.

test\_table.cpp— файл, который содержит реализацию тестов для проверки корректности работы классов таблиц.

## Описание структуры данных

**RecordTable:**

Структура, содержит два поля: ключ и данные.

**Table:**

Абстрактный, шаблонный класс таблиц. Содержит реализацию методов навигации и переупаковки.

**TableUnorder:**

Класс несортированных таблиц.

*Вставка:*

Вводятся входные данные – полином, его строка—ключ.

*Поиск:*

Вводится ключ. Далее массив элементов просматривается для нахождения совпадений ключей. Если ключ совпал, то функция возвращает адрес на структуру RecordTable. Если ключ не совпал, то вызывается исключение.

*Удаление:*

Вводится ключ, начинается поиск элемента. Если элемент найден, то происходит его удаление и обнуление адреса данной ячейки. Если элемент не найден, то вызывается исключение.

**TableOrder:**

Класс сортированных таблиц.

*Вставка:*

Вводятся входные данные – полином, его строка—ключ. Функция вставки ищет индекс, куда необходимо вставить элемент, используя бинарный поиск. Если ячейка пуста, то элемент занимает своё место. Если ячейка занята, то все элементы массива сдвигаются вниз на одну позицию, начиная с конца. Если при просмотре ячейки на наличие элемента совпали ключи, то вызывается исключение..

*Поиск:*

Вводится ключ. Он передаётся в функцию бинарного поиска, которая возвращает индекс предполагаемого места, где находится элемент. Если ключи совпали, то возвращается адрес RecordTable, который хранится в ячейке. Если ключи не совпали, то вызывается исключение.

*Удаление:*

Вводится ключ. Ключ передаётся в функцию бинарного поиска, которая возвращает индекс предполагаемого места, где находится элемент. Если ключи совпали, то происходит удаление элемента и сдвиг всех элементов массива на одну ячейку, начиная с места удаления.

**HashTable:**

Класс hash—таблиц.

*Вставка:*

Вводятся входные данные – полином, его строка—ключ. Hash—функция, которая принимает ключ, переводит его в число, которое используется как индекс для вставки элемента в массив. Если ячейка свободна, то элемент вставляется и у ячейки появляется статус—занята. Переменная, которая хранит количество вставленных элементов, увеличивается на единицу. Если ячейка была удалена, то элемент вставляется; у ячейки появляется статус—занята; переменная, хранящая количество вставленных элементов не меняется. Если ячейка занята и ключи равны, то вызывается исключение.

*Удаление:*

Вводится ключ. Ключ передаётся в hash—функцию. Если по вычисленному индексу ячейка свободна, то вызывается исключение. Если ячейка занята и ключи совпали, то функция возвращает адрес на RecordTable, который находился по этому индексу. Если ключи не совпали или ячейка была удалена, то вычисляется предполагаемое место до тех пор, пока ячейка будет искомым ключом, или пустая, то в этом случаи вызывается исключение.

*Поиск:*

Аналогично удалению. Только возвращается адрес, который лежит по индексу найденной ячейки.

## Описание программной реализация

**Node**:

*Поля:*

TP data;—данные в узле

node<TP>\* next; —указатель на следующий узел.

*Методы:*

node();—конструктор по умолчанию

node(TP a1, node<TP>\* a2 = NULL); —конструктор с параметром

bool operator< (const node<TP>& a) const; —оператор сравнения

bool operator> (const node<TP>& a) const;—оператор сравнения

**Monom:**

*Поля:*

double cff; —коэффициент перед мономом

int svr; —свёрнутая степень

*Методы:*

monom(const monom& a);

monom(const double index\_cff = 0.0, const unsigned int index\_svr = 0); —конструктор

const monom& operator=(const monom& a); —оператор присваивания

bool operator< (const monom& a) const; —оператор сравнения

bool operator> (const monom& a) const— оператор сравнения

bool operator==(const monom& a) const;—перегрузка ==

bool operator!=(const monom& a) const; —перегрузка !=

**List:**

*Поля:*

node<TP>\* head; —голова

node<TP>\* Curr; —указатель на текущий элемент

*Методы:*

list();—конструктор по умолчанию

~list();—деструктор

list(const list<TP>& a); —конструктор

void Clean();—очистка списка

void InsertInOrder(const TP& a); —вставить в упорядоченный список

void InsertAfter(node<TP>\* a1, const TP& a2); —вставить после

void InsertToTail(const TP a); —вставить в конец

bool IsEmpty() const; —проверка на пустоту

TP& GetCurr() const; —получение текущего адреса

void Reset();—установка на начало

void gonext();—переход на следующее звено

bool IsOver();—проверка на конец

const list<TP>& operator=(const list<TP>& a);—перегрузка оператора =

bool operator==(const list<TP>& a) const;—оператор сравнения ==

bool operator!=(const list<TP>& a) const; —оператор сравнения не равно

**Polynom:**

*Поля:*

list<monom> list\_monom;—список мономов

list<monom> such(list <monom> & op);—объединение подобные

*Методы:*

polynom(const polynom& p);

const polynom& operator=(const polynom &p);

polynom(list<monom> &in\_list) : list\_monom(in\_list); —конструктор по списку

polynom(const string op = "");— констурктор по строке

polynom operator-(const polynom& p) const;­— бинарный минус

polynom operator-() const; — унарный минус

bool operator==(const polynom& p) const— оператор сравнения

bool operator!=(const polynom& p) const; — оператор сравнения

polynom operator+(const polynom& p) const; — оператор сложения полиномов

polynom operator\*(const polynom& p) const; — оператор умножения полиномов

polynom operator\*(const double k) const; — умножение на константу

friend polynom operator\*(const double k, const polynom& p); —умножение на константу с другой стороны

friend std::ostream& operator<<(std::ostream &str, const polynom &p); — оператор вставки в поток

**Table:**

*Поля:*

RecordTable <key\_T, data\_T> \*\* A\_T; — массив указателей на записи

int Max\_Dim; — размер таблицы

int Index; — верхний элемент

int \_Dim; — индекс фактической записи

virtual void Repack() = 0; — переупаковка

*Методы:*

Table(int Dim\_Tb = MIN\_DIM); — конструктор

virtual ~Table();— деструктор

virtual void Insert(const key\_T &\_k, const data\_T &\_d) = 0; — вставка

virtual void Dell(const key\_T &\_k) = 0; — удаление

virtual RecordTable<key\_T, data\_T> \* Search(const key\_T &\_k) const = 0; — поиск

//Методы навигации

void Reset(); — переход в начало таблицы

bool IsTableEnd() const; — проверка на конец

void GoNext();— установка на следующий индекс

data\_T GetCurr() const; — просмотр текущего элемента

template <class key\_T, class data\_T> friend std::ostream& operator<<(std::ostream& os, const Table<key\_T, data\_T>& P); — вставка в поток

**TableUnorder:**

Класс таблиц, потомок класса table не содержит новый полей и методов.

**TableOrder**:

Класс таблиц, потомок класса table.

*Новые методы:*

int BinarSearch(const key\_T &Key) const; — бинарный поиск

**TableHash:**

Класс таблиц, потомок класса table.

*Новые поля:*

int \*P\_; — массив, хранящий статус ячейки(свободна/занята/удалена)

*Новые методы:*

int Hash\_function(const key\_T &k) const; — hash-функция

# Заключение

В лабораторной работе была реализована программа, выполняющая вставку, поиск, удаления элементов, заданных пользователем. Реализован алгоритм, позволяющий разрешить проблемы при возникновении коллизий в хэш таблицах. Алгоритмы обеспечивают уникальность ключа во всех видах таблиц.

# Список литературы

1. Рабочие материалы к учебному курсу « Методы программирования», В.П. Гергель.

2. Лабораторный практикум. Барышева И.В., Мееров И.Б., Сысоев А.В., Шестакова Н.В. Под редакцией Гергеля В.П. Учебно-методическое пособие.

# Приложение

## Приложение A. table.h.

#pragma once

#define MIN\_DIM 50

#include <iostream>

#include <ostream>

template <class key\_T, class data\_T>

struct RecordTable //Данные, которые хранит таблица

{

key\_T key; //Ключ

data\_T data; //Данные, которые хранятся в таблице-указатель

RecordTable(key\_T \_k = key\_T(), data\_T \_d = data\_T()) { key = \_k; data = \_d; }//к.

};

template <class key\_T, class data\_T>

class Table //Класс таблиц (абстрактный)

{

protected:

RecordTable <key\_T, data\_T> \*\* A\_T; //Массив указателей на записи

int Max\_Dim; //Размер таблицы

int Index; //Верхний элемент

int \_Dim; //Индекс фактической записи

virtual void Repack() = 0; //Переупаковка

public:

Table(int Dim\_Tb = MIN\_DIM); //К.

virtual ~Table() {for (int i = 0; i < Max\_Dim; i++) delete A\_T[i]; delete[] A\_T; };//деструктор

virtual void Insert(const key\_T &\_k, const data\_T &\_d) = 0;

virtual void Dell(const key\_T &\_k) = 0;

virtual RecordTable<key\_T, data\_T> \* Search(const key\_T &\_k) const = 0; //Методы навигации

void Reset() { \_Dim = 0; }; //Текущий элемент = 0

//virtual void Move() { \_Dim++; if (Index == \_Dim) Reset(); };

bool IsTableEnd() const { if (Index == 0) return true; if (\_Dim == Index - 1) return 1; else return 0; }; //Проверяем на конец

void GoNext() { if (!IsTableEnd())\_Dim++; };//Установка следующего индекса

data\_T GetCurr() const { return (\*(this->A\_T[\_Dim])).data; };//Просмотр текущего элемента

template <class key\_T, class data\_T> friend std::ostream& operator<<(std::ostream& os, const Table<key\_T, data\_T>& P); //Вставка в поток

};

template <class key\_T, class data\_T>

Table<key\_T, data\_T>::Table(int Dim\_Tb)

{

Max\_Dim = Dim\_Tb;

A\_T = new RecordTable<key\_T, data\_T> \*[Dim\_Tb];

for (int i = 0; i < Dim\_Tb; i++)

A\_T[i] = NULL;

Index = 0;

\_Dim = 0;

};

template <class key\_T, class data\_T>

std::ostream& operator << (std::ostream& os, const Table<key\_T, data\_T>& P)

{

for (int i = 0; i<P.Max\_Dim; i++)

{

if (P.A\_T[i] != NULL)

os << i << ": " << (\*(P.A\_T[i])).key << " --> " << " " << (\*(P.A\_T[i])).data << endl;

else os << i << ": " << " NULL" << endl;

}

return os;

## }

## Приложение B. tableUnorder.h

#pragma once

#include "table.h"

#include "math.h"

#include <string>

using namespace std;

template <class key\_T, class data\_T>

class TableUnorder : public Table <key\_T, data\_T>

{

public:

TableUnorder(int Dim = MIN\_DIM) : Table<key\_T, data\_T>(Dim) {};//К.

virtual ~TableUnorder() {}; //Деструктор

virtual void Insert(const key\_T &\_k, const data\_T &\_d); //Вставка

virtual void Dell(const key\_T &\_k); //Удаление элемента

virtual RecordTable<key\_T, data\_T>\* Search(const key\_T &\_k) const;//Поиск элемента private:

virtual void Repack();//Переупаковка

};

template <class key\_T, class data\_T>

void TableUnorder< key\_T, data\_T> ::Insert(const key\_T &\_k, const data\_T &\_d)

{

if ((double)this->Index / (double)this->Max\_Dim > 0.7)

Repack();

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

{

if ((\*(this->A\_T[i])).key == \_k)

{

throw "key - duplicate"; }

}

this->A\_T[this->Index++] = new RecordTable<key\_T, data\_T>(\_k, \_d);

};

template <class key\_T, class data\_T>

void TableUnorder< key\_T, data\_T> ::Dell(const key\_T &\_k)

{

int i = 0;

int FLAG = 0;

while ((i < this->Index) && (FLAG == 0)) {

if ((\*(this->A\_T[i])).key == \_k) FLAG = 1;

i++;

}

if (FLAG == 1) {

i--;

delete this->A\_T[i];

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

this->A\_T[i] = this->A\_T[i + 1];

i++;

}

this->A\_T[i] = NULL;

this->Index--;

}

else throw "item does not exist";

};

template <class key\_T, class data\_T>

RecordTable<key\_T, data\_T>\* TableUnorder< key\_T, data\_T> ::Search(const key\_T &\_k) const

{

int i = 0;

while (i < this->Index)

{

if ((\*(this->A\_T[i])).key == \_k)

return this->A\_T[i];

i++;

}

throw "item does not exist";

}

template <class key\_T, class data\_T>

void TableUnorder <key\_T, data\_T> ::Repack()

{

int Nxt\_Dim = (int)(this->Max\_Dim + 20)\*1.65;

RecordTable<key\_T, data\_T> \*\* New\_A = new RecordTable<key\_T, data\_T>\*[Nxt\_Dim];

for (int i = 0; i<this->Max\_Dim; i++)

New\_A[i] = this->A\_T[i];

for (int i = this->Max\_Dim; i<Nxt\_Dim; i++)

New\_A[i] = NULL;

delete[] this->A\_T;

this->A\_T = New\_A;

this->Max\_Dim = Nxt\_Dim;

## };

## Приложение C. tableOrder.h

#pragma once

#include "table.h"

template <class key\_T, class data\_T>

class TableOrder : public Table<key\_T, data\_T>

{

public:

TableOrder(int Dim = MIN\_DIM) : Table<key\_T, data\_T>(Dim) {};

virtual ~TableOrder() {};

virtual void Insert(const key\_T &\_k, const data\_T &\_d);//Вставка

virtual void Dell(const key\_T &\_k);//Удаление

virtual RecordTable<key\_T, data\_T>\* Search(const key\_T &\_k) const;//Поиск

private:

virtual void Repack();//Переупаковка

int BinarSearch(const key\_T &Key) const; //Бинарный поиск

};

template <class key\_T, class data\_T>

void TableOrder <key\_T, data\_T> ::Insert(const key\_T &\_k, const data\_T &\_d)

{

int pa = BinarSearch(\_k);

int i;

if ((this->A\_T[pa] == NULL) || ((\*(this->A\_T[pa])).key != \_k))

{

if ((double)this->Index / (double)this->Max\_Dim > 0.7)

Repack();

for (i = this->Index; i > pa; i--)

this->A\_T[i] = this->A\_T[i - 1];

this->A\_T[i] = new RecordTable<key\_T, data\_T>(\_k, \_d);

this->Index++;

}

else throw "key - duplicate";

}

template <class key\_T, class data\_T>

void TableOrder <key\_T, data\_T> ::Dell(const key\_T &\_k)

{

int pa = BinarSearch(\_k);

if ((this->A\_T[pa] != NULL) && ((\*(this->A\_T[pa])).key == \_k))

{

delete this->A\_T[pa];

for (int i = pa; i < this->Index; i++)

this->A\_T[i] = this->A\_T[i + 1];

this->Index -= 1;

}

else

throw "item does not exist";

}

template <class key\_T, class data\_T>

RecordTable<key\_T, data\_T>\* TableOrder <key\_T, data\_T> ::Search(const key\_T &\_k) const

{

int pa = BinarSearch(\_k);

if ((this->A\_T[pa]!=NULL)&&((\*(this->A\_T[pa])).key == \_k))

return this->A\_T[pa];

else

throw "item does not exist";

}

template <class key\_T, class data\_T>

void TableOrder <key\_T, data\_T> ::Repack()

{

int Nxt\_Dim = (int)(this->Max\_Dim + 20)\*1.65;

RecordTable <key\_T, data\_T> \*\* New\_A = new RecordTable <key\_T, data\_T>\*[Nxt\_Dim];

for (int i = 0; i<this->Max\_Dim; i++)

New\_A[i] = this->A\_T[i];

for (int i = this->Max\_Dim; i<Nxt\_Dim; i++)

New\_A[i] = NULL;

delete[] this->A\_T;

this->A\_T = New\_A;

this->Max\_Dim = Nxt\_Dim;

}

template <class key\_T, class data\_T>

int TableOrder <key\_T, data\_T> ::BinarSearch(const key\_T &k) const

{

int right = this->Index - 1;

int mid = 0;

int left = 0;

while (right >= left)

{

mid = (right + left) / 2;

if ((\*(this->A\_T[mid])).key < k)

left = mid + 1;

else if ((\*(this->A\_T[mid])).key > k)

right = mid - 1;

else return mid; }

if (left > right)

mid = left;

return mid;

}

## Приложение D. tableHash.h

#pragma once

#include "table.h"

#include <string>

using namespace std;

//0 - свободен;

//1 - место занято;

//-1 - был удалён;

template <class key\_T, class data\_T>

class TableHash : public Table <key\_T, data\_T>

{

public:

TableHash(int Dim = 20) : Table<key\_T, data\_T>(Dim) { P\_ = new int[Dim]; for (int i = 0; i < Dim; i++) P\_[i] = 0; }; //К.

~TableHash() {};

void Insert(const key\_T &\_k, const data\_T &\_d); //Вставка

void Dell(const key\_T &\_k); //Удаление элемента

RecordTable<key\_T, data\_T>\* Search(const key\_T &\_k) const; //Поиск

private:

int \*P\_;

void Repack();//Перехеширование

int Hash\_function(const key\_T &k) const;

};

template<class key\_T>

int Get\_Key(const key\_T& k) { return 0; }

template<>

int Get\_Key<string>(const string &k) //Хэш - функции (от string)

{

int g = 31;

int hash = 0;

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

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

return hash;

}

template <class key\_T, class data\_T>//Хэш-функция МЕТОД

int TableHash<key\_T, data\_T>::Hash\_function(const key\_T& k) const

{

int temp = abs(Get\_Key(k));

return (temp % (this->Max\_Dim));

}

template <class key\_T, class data\_T>

void TableHash<key\_T, data\_T>::Repack()

{

int i = 0;

int Nxt\_Dim = (int)(this->Max\_Dim + 20)\*1.5;

RecordTable<key\_T, data\_T> \*\* New\_A = new RecordTable<key\_T, data\_T> \*[Nxt\_Dim];

int \* PM\_ = new int[Nxt\_Dim];

for (int i = 0; i < this->Max\_Dim; i++)

PM\_[i] = P\_[i];

for (int i = this->Max\_Dim; i < Nxt\_Dim; i++)

PM\_[i] = 0;

delete[]P\_;

P\_ = PM\_;

for (int i = 0; i<this->Max\_Dim; i++)

New\_A[i] = this->A\_T[i];

for (int i = this->Max\_Dim; i < Nxt\_Dim; i++)

New\_A[i] = NULL;

delete[] this->A\_T;

this->A\_T = New\_A;

this->Max\_Dim = Nxt\_Dim;

}

template <class key\_T, class data\_T>

void TableHash<key\_T, data\_T>::Insert(const key\_T &\_k, const data\_T &\_d)

{

if ((double)this->\_Dim / (double)this->Max\_Dim > 0.7)

Repack();

int loc = Hash\_function(\_k);

if (P\_[loc] == 0)

{

this->A\_T[loc] = new RecordTable<key\_T, data\_T>(\_k, \_d);

P\_[loc] = 1;

this->\_Dim++;

}

else if (P\_[loc] == -1)

{

this->A\_T[loc] = new RecordTable<key\_T, data\_T>(\_k, \_d);

P\_[loc] = 1;

}

else

{

if ((\*(this->A\_T[loc])).key == \_k)

{

throw "key - duplicated";

}

int label = loc + 1;

while (P\_[label] == 1)

{

label = label + 1;

if ((this->A\_T[label] != NULL) && ((\*(this->A\_T[label])).key == \_k))

{

throw "key - duplicated";

}

}

this->A\_T[label] = new RecordTable<key\_T, data\_T>(\_k, \_d);

if (P\_[label] == 0)

this->\_Dim++;

P\_[label] = 1;

}

}

template <class key\_T, class data\_T>

void TableHash<key\_T, data\_T>::Dell(const key\_T &\_k)

{

int loc = Hash\_function(\_k);

if (P\_[loc] == 0)

throw "item does not exist";

if ((this->A\_T[loc] != NULL) && ((\*(this->A\_T[loc])).key == \_k))

{

delete this->A\_T[loc];

this->A\_T[loc] = NULL;

P\_[loc] = -1;

}

else

{

while ((P\_[loc] == -1) || ((P\_[loc] == 1) && ((\*(this->A\_T[loc])).key != \_k)))

loc = loc + 1;

if (P\_[loc] == 1)

{

delete this->A\_T[loc];

this->A\_T[loc] = NULL;

P\_[loc] = -1;

}

else throw "item does not exist";

}

}

template <class key\_T, class data\_T>

RecordTable<key\_T, data\_T>\* TableHash<key\_T, data\_T>::Search(const key\_T &\_k) const

{

int loc = Hash\_function(\_k);

if (P\_[loc] == 0)

throw "item does not exist";

if ((this->A\_T[loc] != NULL) && ((\*(this->A\_T[loc])).key) == \_k)

{

return this->A\_T[loc];

}

else

{

while ((P\_[loc] == -1) || ((P\_[loc] == 1) && ((\*(this->A\_T[loc])).key != \_k)))

loc = loc + 1;

if (P\_[loc] == 1)

{

return this->A\_T[loc];

}

else

throw "item does not exist";

}

}

## Приложение E. main\_table.cpp

#include "monom.h"

#include "polynom.h"

#include <iostream>

#include <ostream>

#include "tableUnorder.h"

#include "tableOrder.h"

#include "tableHash.h"

using namespace std;

int main()

{ int c = 0;

int k;

int SIZE;

cout << "Enter the size of the table" << endl;

cin >> SIZE;

TableUnorder<string, polynom> A(SIZE);

string a;

TableOrder<string, polynom> B(SIZE);

TableHash<string, polynom> C(SIZE);

while (c != 1)

{ system("cls");

cout << "Chouse operation" << endl;

cout << "1 - Insert" << endl;

cout << "2 - Dell" << endl;

cout << "3 - Search" << endl;

cout << "4 - Print" << endl;

cin >> k;

switch (k)

{

case 1:

{ string str;

cout << "Write your Polinom" << endl;

cin >> str;

polynom a(str);

try { A.Insert(str, a); }

catch (...) { cout << "duplicated key" << endl; }

try { B.Insert(str, a); }

catch (...) { cout << "duplicated key" << endl; }

try { C.Insert(str, a); }

catch (...) { cout << "duplicated key" << endl; }

cout << "Unordered :" << endl;

cout << A << endl;

cout << "Ordered :" << endl;

cout << B << endl;

cout << "Hash :" << endl;

cout << C << endl;

break;

}

case 2:

{ string str;

cout << "Write polynom" << endl;

cin >> str;

polynom a(str);

try { A.Dell(str); }

catch (...) { cout << "element doesn't exist" << endl; }

try { B.Dell(str); }

catch (...) { cout << "element doesn't exist" << endl; }

try { C.Dell(str); }

catch (...) { cout << "element doesn't exist" << endl; }

cout << "Unordered :" << endl;

cout << A << endl;

cout << "Ordered :" << endl;

cout << B << endl;

cout << "Hash :" << endl;

cout << C << endl;

break;

}

case 3:

{ string str;

cout << "Write polynom" << endl;

cin >> str;

polynom a(str);

try { cout << "finded: " << A.Search(str)->data; }

catch (...) { cout << "element doesn't exist" << endl; }

try { cout << "finded: " << B.Search(str)->data; }

catch (...) { cout << "element doesn't exist" << endl; }

try { cout << "finded: " << C.Search(str)->data; }

catch (...) { cout << "element doesn't exist" << endl; }

break;

}

case 4:

{ cout << "Unordered :" << endl;

cout << A << endl;

cout << "Ordered :" << endl;

cout << B << endl;

cout << "Hash :" << endl;

cout << C << endl;

break;

}

default:

{

cout << "ERROR, choose again" << endl;

break;

}

}

cout << "Exit - 1" << endl;

cin >> c;

}

return 0;

}