Министерство образования и науки Российской Федерации

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

Институт Информационных технологий, математики и механики

Отчёт по лабораторной работе

Таблицы

Выполнил:

студент гр. 381603-01

Федотова Е.А.

Проверил:

ассистент каф. МОСТ, ИИТММ

Кустикова В.Д.

Нижний Новгород

2018 г

Содержание

[Введение 3](#_Toc515832534)

[1 Постановка задачи 4](#_Toc515832535)

[2 Руководство пользователя 5](#_Toc515832536)

[3 Руководство программиста 6](#_Toc515832537)

[3.1 Структура программы 6](#_Toc515832538)

[3.2 Структуры данных 6](#_Toc515832539)

[3.2.1 Описание класса Tabrecord 6](#_Toc515832540)

[3.2.2 Описание класса Table 6](#_Toc515832541)

[3.2.3 Описание класса SortedTable 7](#_Toc515832542)

[3.2.4 Описание класса Scantable 7](#_Toc515832543)

[3.2.5 Описание класса Hashtable 7](#_Toc515832544)

[3.3 Описание алгоритмов 8](#_Toc515832545)

[3.3.1 Алгоритмы класса SortedTable 8](#_Toc515832546)

[3.3.2 Алгоритмы класса Scantable 8](#_Toc515832547)

[3.3.3 Алгоритмы класса Hashtable 9](#_Toc515832548)

[Заключение 11](#_Toc515832549)

[Литература 12](#_Toc515832550)

[Приложение 13](#_Toc515832551)

[Приложение 1. Программная реализация шаблонного класса Tabкecord и виртуального класса Table 13](#_Toc515832552)

[table.h 13](#_Toc515832553)

[Приложение 2. Программная реализация дочернего класса Scantable 16](#_Toc515832554)

[Приложение 3. Программная реализация дочернего класса SortedTable 17](#_Toc515832555)

[Приложение 4. Программная реализация дочернего класса Hashtable 19](#_Toc515832556)

# Введение

С физической точки зрения таблица представляет собой вектор, элементами которого являются записи. Характерной логической особенностью таблиц, которая определяет их отдельное рассмотрение, является то, что доступ к элементам таблицы производится не по номеру (индексу), а по ключу - по значению одного из свойств объекта, описываемого элементом таблицы. Ключ - это свойство, идентифицирующее данную запись во множестве однотипных записей. Как правило, к ключу предъявляется требование уникальности в данной таблице. Ключ может включаться в состав записи и быть одним из ее полей, но может и не включаться в запись, а вычисляться по положению записи. Таблица может иметь один или несколько ключей.

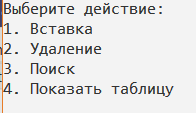
Хеш-таблицей называется структура данных, предназначенная для реализации ассоциативного массива, такого в котором адресация реализуется посредством хеш-функции. Хеш**-**функция – это функция, преобразующая ключ **key** в некоторый индекс iравныйHashFunc(key), где HashFunc(key) – хеш-код key. Весь процесс получения индексов хеш-таблицы называется хешированием.

# Постановка задачи

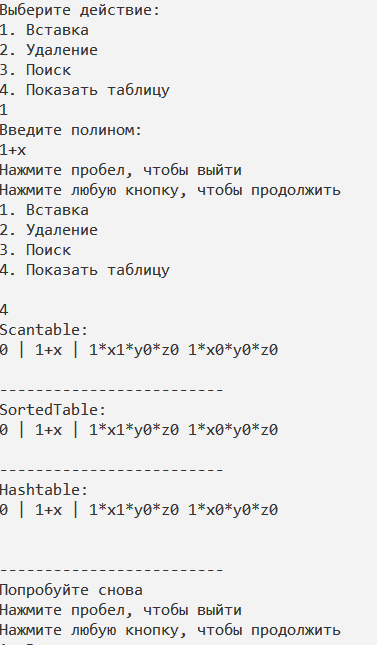
В лабораторной работе нужно реализовать виртуальный шаблонный класс таблиц, в котором поддерживаются операции сложения, умножения и поиска. Также нужно реализовать 3 дочерних класса от этого. Первый – сортируемые таблицы, второй – несортируемые и третий – хеш – таблицы. В каждом их них должны быть реализованы операции сложения, умножения и поиска. Для каждого из этих 3 видов таблиц надо написать тесты на проверку правильности их работы.

# Руководство пользователя

1. При запуске программы пользователь может выбрать действие:



1. Варианты действия
2. Выбрав одно из них, нажав при этом 1, 2 или 3, он может вставить полином, удалить его из таблицы, если она заполнена, и найти в ней полином. Нажав 4 можно увидеть текущее состояние таблицы.



1. Пример работы программы
2. Получив результат, пользователь может либо выйти из программы, нажав пробел, либо продолжить работу, нажав любую клавишу.

# Руководство программиста

## Структура программы

Программа разделена на несколько файлов: node.h, list.h, monom.h, polinom.h, polinom.cpp, sample.cpp, test\_list.cpp, test\_polinom.cpp, table.h, sorted\_table.h, scan\_table.h, hash\_table.h, table.cpp, scan\_table.cpp, sorted\_table.cpp, hash\_table.cpp, scan\_table\_test.cpp, sorted\_table\_test.cpp, hash\_table\_test.cpp.

Про статическую библиотеку класса TPolinom написано в первом отчете.

В файле table.h содержится описание и реализация шаблонного класса Tabrecord, который представляет собой запись – элемент таблицы, а также описание и реализацию шаблонного класса Table, который представляет собой таблицу из записей.

Этот файл подключен к файлам sorted\_table.h, который представляет дочерний класс SortedTable сортируемых таблиц, scan\_table.h, который представляет дочерний класс Scantable несортируемых таблиц, hash\_table.h, который представляет дочерний класс Hashtable хеш – таблиц.

В файле sample.cpp находится «руководство пользователя».

В файлах scan\_table\_test.cpp, sorted\_table\_test.cpp, hash\_table\_test.cpp находятся тесты для соответствующих классов.

## Структуры данных

### Описание класса Tabrecord

Поля:

T data – данные записи

String key – ключ записи

Методы:

Tabrecord – конструктор по умолчанию;

Tabrecord(const Tabrecord<T> &t) – конструктор копирования;

Tabrecord(const string& key1, const T& data1) – конструктор записи таблицы;

Tabrecord& operator=(const Tabrecord<T>& t) – перегрузка операции равно;

### **Описание класса Table**

Поля:

Tabrecord<T>\*\* tab – массив указателей на записи;

int currind – текущий индекс в таблице;

int currec – текущее количество записей в таблице;

int maxrec – максимальное количество записей в таблице;

Методы:

Table(int j = 2)- конструктор по умолчанию;

Table(const Table<T> &l) – конструктор копирования;

Virtual ~Table()- деструктор;

Virtual void Realloc() – переопределение памяти;

Virtual T& Search(const string& key1)- поиск;

virtual void Delete(const string& key1) = 0 – удаление;

virtual void Insert(const T& data1, const string& key1) = 0 – вставка;

virtual void Reset() – переход в начало;

virtual bool TIsended() const – проверка на конец;

bool Isfull() const – проверка на полноту;

bool Isempty() const – проверка на пустоту;

virtual void TGetnext() – переводит текущий индекс на следующий;

virtual T& Getcurr() const – получить данные по текущему индексу;

virtual int Getcurrrec() const – получить currrec;

virtual int Getcurrind() const – получить currind;

template <class T> friend ostream& operator<< (ostream& os, const Table<T>& t) – перегрузка вывода таблицы на экран;

### **Описание класса SortedTable**

Все поля и методы унаследованы от класса Table.

Метод:

int BinSearch(const string& key1) const – бинарный поиск по ключу;

### **Описание класса Scantable**

Все поля и методы унаследованы от класса Table.

### Описание класса Hashtable

Поля:

int\* flag – массив, состоящий из 0 и 1. 0 – пустая запись, 1 – запись занята.

Остальные поля унаследованы от класса Table.

Методы:

int HashFunc(const string& key) const – хеш-функция по ключу;

Все остальные методы унаследованы от класса Table.

## Описание алгоритмов

### Алгоритмы класса SortedTable

1. Вставка полинома:

а) Перераспределяем память, если таблица полна;

б) Ищем место, куда нужно вставить запись так, чтобы не нарушалась упорядоченность в таблице;

в) Те записи, у которых ключ больше, смещаем.

г) Вставляем новую запись на найденное место.

2. Удаление полинома:

а) Переставляем текущий индекс в начало;

б) Ищем по ключу ту запись, которую надо удалить;

в) Если нашли, то все элементы, ключ которых больше ключа удаляемой записи, сдвигаем вверх, а запись удаляем.

3. Поиск полинома:

а) Сравниваем ключ той записи, которую надо найти, с ключами остальных записей в таблице;

б) Если такой нашли, то возвращаем данные той записи, по ключу которой искали.

### Алгоритмы класса Scantable

1. Вставка полинома:

а) Перераспределяем память, если таблица полна;

б) Переставляем текущий индекс до индекса текущего количества записей в таблице;

в) На следующее место после последней записи вставляется новая запись.

2. Удаление полинома:

а) Переставляем текущий индекс в начало;

б) Ищем по ключу ту запись, которую надо удалить;

в) Если нашли, то удаляем, и на ее место ставим последнюю запись в таблице;

г) Количество текущих записей уменьшаем на 1.

3. Поиск полинома:

а) Сравниваем ключ той записи, которую надо найти, с ключами остальных записей в таблице;

б) Если такой нашли, то возвращаем данные той записи, по ключу которой искали.

### Алгоритмы класса Hashtable

1. Хеш-функция:

а) Каждый элемент ключа переводится в число. Сумма этих чисел и есть значение хеш-функции.

\*\*\* Коллизией (иногда конфликтом или столкновением) хеш-функции называются такие два входных блока данных, которые дают одинаковые хеш-коды.

Существует два основных метода используемых в хеш-таблицах:

1. Метод цепочек (метод прямого связывания)
2. Метод открытой адресации

Второй метод состоит в том, что в массиве таблицы хранятся пары ключ-значение. Таким образом мы полностью отказываемся от ссылок и просто просматриваем записи таблицы, пока не найдем нужный ключ key или пустую позицию.

2. Вставка полинома:

а) Перераспределяем память, если таблица полна;

б) Текущий индекс запоминаем как число, полученное хеш-функцией от ключа той записи, которую нужно вставить;

в) Создаем в ячейке таблицы запись и делаем флаг 1, то есть запоминаем, что эта ячейка теперь занята. Если же случилась коллизия, то запись создается в следующей свободной ячейке таблицы;

3. Удаление полинома:

а) Текущий индекс запоминаем как число, полученное хеш-функцией от ключа той записи, которую нужно удалить;

б) Удаляем запись, если нет коллизии, и запоминаем, что эта ячейка свободна. Если коллизия есть, то мы смотрим, совпадает ли ключ нашей записи с той, которую нашли. Если нет, то идем по массиву дальше и проверяем. Если запись нашлась, то удаляем ее.

4. Поиск полинома:

\*с коллизией тот же самый принцип\*

а) Текущий индекс запоминаем как число, полученное хеш-функцией от ключа той записи, которую нужно найти;

б) Выводим данные этой записи.

# Заключение

В данной лабораторной работе мне удалось реализовать виртуальный шаблонный класс таблиц и 3 его дочерних, которые поддерживают 3 главные операции. Могу сделать вывод, что программа работает правильно, на основе верно выполненных тестов.

# Литература

1. Лекции «Алгоритмы и структуры данных» Гергель В.П. ННГУ им. Лобачевского Н. И.

# Приложение

## Приложение 1. Программная реализация шаблонного класса Tabrecord и виртуального класса Table

## table.h

#pragma once

#include <string>

#include <iostream>

using namespace std;

template <typename T>

class Tabrecord

{

public:

T data;

string key;

Tabrecord();

Tabrecord(const string& key1, const T& data1);

Tabrecord(const Tabrecord<T>& t);

Tabrecord& operator=(const Tabrecord<T>& t);

};

template <typename T>

class Table

{

protected:

Tabrecord<T>\*\* tab;

int currind; //текущий индекс

int maxrec; //максимальное количество

int currrec; //текущее количество

virtual void Realloc(); ///////

public:

Table(int j = 2);

Table(const Table<T>& t);

virtual ~Table();

virtual T& Search(const string& key1) const = 0; //поиск

virtual void Delete(const string& key1) = 0; //удаление

virtual void Insert(const T& data1, const string& key1) = 0; //вставка

virtual void Reset(); //переход в начало

virtual bool TIsended() const; //проверка на конец /////const

bool Isfull() const; //проверка на заполненность

bool Isempty() const; //проверка на пустоту

virtual void TGetnext();

virtual T& Getcurr() const;

virtual int Getcurrrec() const;

virtual int Getcurrind() const;

template <class T> friend ostream& operator<< (ostream& os, const Table<T>& t);

};

//реализация класса Tabrecord

template<typename T>

Tabrecord<T>::Tabrecord()

{

key = "empty";

}

template<typename T>

Tabrecord<T>::Tabrecord(const string& key1, const T& data1)

{

key = key1;

data = data1;

}

template<typename T>

Tabrecord<T>::Tabrecord(const Tabrecord& t)

{

key = t.key;

data = t.data;

}

template<typename T>

Tabrecord<T>& Tabrecord<T>::operator=(const Tabrecord<T>& t)

{

key = t.key;

data = t.data;

}

//реализация класса Table

template <typename T>

void Table<T>::Realloc()

{

int maxrec1 = maxrec \* 2;

Tabrecord<T>\*\* temp = new Tabrecord<T>\*[maxrec1];

Reset();

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

temp[i] = tab[i];

delete[] tab;

maxrec = maxrec1;

tab = temp;

}

template<typename T>

Table<T>::Table(int j)

{

maxrec = j;

currind = -1;

currrec = 0;

tab = new Tabrecord<T>\*[maxrec];

}

template<typename T>

Table<T>::Table(const Table<T>& t)

{

currrec = t.currrec;

maxrec = t.maxrec;

currind = t.currind;

delete[] tab;

tab = new Tabrecord<T>\*[maxrec];

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

tab[i] = t.tab[i];

}

template<typename T>

Table<T>::~Table()

{

delete[] tab;

}

template<typename T>

void Table<T>::Reset()

{

if (currrec > 0)

currind = 0;

else

currind = -1;

}

template<typename T>

bool Table<T>::TIsended() const

{

if (((currind + 1) == currrec) || (currind == -1))

return true;

else

return false;

}

template<typename T>

bool Table<T>::Isfull() const

{

if (currrec == maxrec)

return true;

else

return false;

}

template<typename T>

bool Table<T>::Isempty() const

{

if (currrec == 0)

return true;

else

return false;

}

template<typename T>

void Table<T>::TGetnext()

{

if (currrec)

currind++;

else

throw "error";

if (currind == currrec)

Reset();

}

template<typename T>

T& Table<T>::Getcurr() const

{

if (currrec)

return tab[currind]->data;

else

throw "error";

}

template<typename T>

int Table<T>::Getcurrrec() const

{

return currrec;

}

template<typename T>

int Table<T>::Getcurrind() const

{

return currind;

}

template <typename T>

ostream& operator<< (ostream& os, const Table<T>& t)

{

if (t.currrec)

for (int i = 0; i < t.currrec; i++)

os << i << " | " << t.tab[i]->key << " | " << t.tab[i]->data << endl;

else

os << "error" << endl;

return os;

}

## Приложение 2. Программная реализация дочернего класса Scantable

#pragma once

#include "table.h"

template<typename T>

class Scantable : public Table<T>

{

public:

Scantable(int j = 2) : Table(j) {};

Scantable(const Scantable<T>& t) : Table(t) {}

~Scantable() {}

void Insert(const T& data1, const string& key1) override;

T& Search(const string& key1) const override;

void Delete(const string& key1) override;

};

template<typename T>

void Scantable<T>::Insert(const T& data1, const string& key1)

{

if(Isfull())

Realloc();

Reset();

if (currrec)

{

while ((currind < currrec) && (tab[currind]->key != key1))

currind++;

if (currind == currrec)

{

tab[currind] = new Tabrecord<T>(key1, data1);

currind++;

}

else

throw "error";

}

else

{

currind++;

tab[currind] = new Tabrecord<T>(key1, data1);

currrec++;

}

}

template<typename T>

T& Scantable<T>::Search(const string &key1) const

{

Scantable<T> temp(\*this);

temp.Reset();

if (temp.currind > -1)

{

while((temp.tab[temp.currind]->key != key1) && (temp.currind < temp.currrec))

temp.TGetnext();

if ((temp.currind < temp.currrec) && (temp.tab[temp.currind]->key == key1))

return temp.tab[temp.currind]->data;

else

throw "error";

}

else

throw "error";

}

template<typename T>

void Scantable<T>::Delete(const string& key1)

{

Reset();

if (currind > -1)

{

while ((currind < currrec) && (tab[currind]->key != key1))

currind++;

if (currrec && (currind < currrec))

{

if (currrec > 1)

tab[currind] = tab[--currrec];

else

currrec = 0;

}

else

throw "error";

}

else

throw "error";

}

## Приложение 3. Программная реализация дочернего класса SortedTable

#pragma once

#include "table.h"

template <typename T>

class SortedTable : public Table<T>

{

protected:

int BinSearch(const string& key1) const;

public:

SortedTable(int j = 2) : Table(j) {}

SortedTable(const SortedTable<T>& t) : Table(t) {}

~SortedTable() {};

void Insert(const T& data1, const string& key1) override;

T& Search(const string& key1) const override;

void Delete(const string& key1) override;

};

///////////////////////////////////////////////////////////

template<typename T>

int SortedTable<T>::BinSearch(const string& key1) const

{

int i = 0;

int j = currrec - 1;

int ser;

while (i <= j)

{

ser = (j + i) / 2;

if (key1 > tab[ser]->key)

i = ser + 1;

else

j = ser - 1;

}

return i;

}

template<typename T>

void SortedTable<T>::Insert(const T& data1, const string& key1)

{

if(Isfull())

Realloc();

Reset();

if (currrec)

{

int k = BinSearch(key1);

if (k != currrec)

{

if (tab[k]->key != key1)

{

for (int i = currrec; i > k; i--)

tab[i] = tab[i - 1];

tab[k] = new Tabrecord<T>(key1, data1);

currrec++;

}

else

throw "error";

}

else

{

tab[k] = new Tabrecord<T>(key1, data1);

currrec++;

}

}

else

{

currind++;

tab[currind] = new Tabrecord<T>(key1, data1);

currrec++;

}

}

template<typename T>

T& SortedTable<T>::Search(const string& key1) const

{

SortedTable<T> temp(\*this);

temp.Reset();

if (currind > -1)

{

int i = BinSearch(key1);

if (key1 == temp.tab[i]->key)

return temp.tab[i]->data;

else

throw "error";

}

else

throw "error";

}

template<typename T>

void SortedTable<T>::Delete(const string& key1)

{

Reset();

if (currrec)

{

int k = BinSearch(key1);

if ((k < currrec) && (tab[k]->key == key1))

{

for (int i = k; k < currrec; i++)

tab[i] = tab[i + 1];

currrec--;

}

else

throw "error";

}

else

throw "error";

}

## Приложение 4. Программная реализация дочернего класса Hashtable

#pragma once

#include "table.h"

template <typename T>

class Hashtable : public Table<T>

{

protected:

int\* flag;

int HashFunc(const string& key) const;

void Realloc() override;

public:

Hashtable(int j = 2);

Hashtable(const Hashtable<T>& t);

~Hashtable();

void Insert(const T& data1, const string& key1) override;

T& Search(const string& key1) const;

void Delete(const string& key1);

void TGetnext() override;

T& Getcurr() const override;

void Reset() override;

bool TIsended() const;

template <class T> friend ostream& operator<<(ostream& os, const Hashtable<T>& t);

};

template<typename T>

Hashtable<T>::Hashtable(int j = 2) : Table(j)

{

flag = new int[j];

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

flag[i] = 0;

}

template<typename T>

Hashtable<T>::Hashtable(const Hashtable<T>& t)

{

maxrec = t.maxrec;

currind = t.currind;

currrec = t.currrec;

delete[] flag;

delete[] tab;

tab = new Tabrecord<T>\*[maxrec];

flag = new int[maxrec];

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

{

tab[i] = t.tab[i];

flag[i] = t.flag[i];

}

}

template<typename T>

Hashtable<T>::~Hashtable()

{

delete[] flag;

}

template<typename T>

int Hashtable<T>::HashFunc(const string& key) const

{

int h = 0;

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

h = h + int(key[i]);

return h % maxrec;

}

template<typename T>

void Hashtable<T>::Realloc()

{

int maxrec1 = maxrec \* 2;

int\* flag1 = new int[maxrec1];

Tabrecord<T>\*\* temp = new Tabrecord<T>\*[maxrec1];

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

{

temp[i] = tab[i];

flag1[i] = flag[i];

}

delete[] tab;

delete[] flag;

tab = temp;

flag = flag1;

maxrec = maxrec1;

}

template<typename T>

void Hashtable<T>::TGetnext()

{

if(currrec)

{

currind++;

while (flag[currind] != 1)

currind = (currind + 1) % maxrec;

}

else

throw "error";

}

template<typename T>

T& Hashtable<T>::Getcurr() const

{

if (currrec)

return tab[currind]->data;

else

throw "error";

}

template<typename T>

void Hashtable<T>::Reset()

{

if (currrec)

{

currind = 0;

while (flag[currind] != 1)

currind++;

}

else

currind = -1;

}

template<typename T>

bool Hashtable<T>::TIsended() const

{

if (currind == currrec || currind == -1)

return true;

else

return false;

}

template<typename T>

void Hashtable<T>::Insert(const T& data1, const string& key1)

{

if (Isfull())

Realloc();

currind = HashFunc(key1);

if (!flag[currind])

{

tab[currind] = new Tabrecord<T>(key1, data1);

currrec++;

flag[currind] = 1;

}

else

if (tab[currind]->key != key1)

{

int i = currind;

while (flag[currind] && ((currind + 1) != i))

currind = (currind + 1) % maxrec;

tab[currind] = new Tabrecord<T>(key1, data1);

currrec++;

flag[currind] = 1;

}

else

throw "error";

}

template<typename T>

T& Hashtable<T>::Search(const string& key1) const

{

Hashtable<T> temp(\*this);

temp.Reset();

if (temp.currrec)

{

temp.currind = temp.HashFunc(key1);

int ind = temp.currind;

if (temp.tab[temp.currind]->key == key1)

return temp.tab[temp.currind]->data;

else

{

while (temp.flag[temp.currind] && ((temp.currind + 1) != ind) && (temp.tab[temp.currind]->key != key1))

temp.currind = (temp.currind + 1) % temp.maxrec;

if (temp.tab[temp.currind]->key == key1)

return temp.tab[temp.currind]->data;

else

throw "error";

}

}

else

throw "error";

}

template<typename T>

void Hashtable<T>::Delete(const string& key1)

{

Reset();

if (currrec)

{

currind = HashFunc(key1);

int ind = currind;

if (flag[currind])

{

if (tab[currind]->key != key1)

{

while (flag[currind] && ((currind + 1) != ind) && (tab[currind]->key == key1))

currind = (currind + 1) % maxrec;

if (!flag[currind] || tab[currind]->key != key1)

throw "error";

else

{

tab[currind] = new Tabrecord<T>;

flag[currind] = 0;

currrec--;

}

}

else

{

tab[currind] = new Tabrecord<T>;

flag[currind] = 0;

currrec--;

}

}

else

throw "error";

}

else

throw "error";

}

template <typename T>

ostream& operator<<(ostream& os, const Hashtable<T>& t)

{

if (t.currrec)

{

for (int i = 0; i < t.maxrec; i++)

if (t.flag[i])

os << i << " | " << t.tab[i]->key << " | " << t.tab[i]->data << endl;

}

else

os << "error" << endl;

return os;

}