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 лабораторной работе

**«Таблицы»**

**Выполнила:** студентка группы 381603-1

Прохорова О. Д.

**Проверила:** кандидат тех. наук, старший преподаватель каф. МОСТ института ИТММ

Кустикова В.Д.

Нижний Новгород

2018

Оглавление

[Введение 3](#_Toc515817358)

[1. Постановка задачи 4](#_Toc515817359)

[2. Руководство пользователя 5](#_Toc515817360)

[3. Руководство программиста 6](#_Toc515817361)

[1. Описание алгоритмов 6](#_Toc515817362)

[2. Описание структуры программы 9](#_Toc515817363)

[3. Описание структур данных 10](#_Toc515817364)

[Заключение 12](#_Toc515817365)

[Литература 13](#_Toc515817366)

[Приложение 14](#_Toc515817367)

[main.cpp 14](#_Toc515817368)

[table.h 16](#_Toc515817369)

[hash.h 20](#_Toc515817370)

[scan.h 26](#_Toc515817371)

[sorted.h 28](#_Toc515817372)

# Введение

Таблица — особая форма передачи содержания, которую отличает от текста организация слов и чисел в колонки (графы) и горизонтальные строки таким образом, что каждый элемент является одновременно составной частью и строки, и колонки.

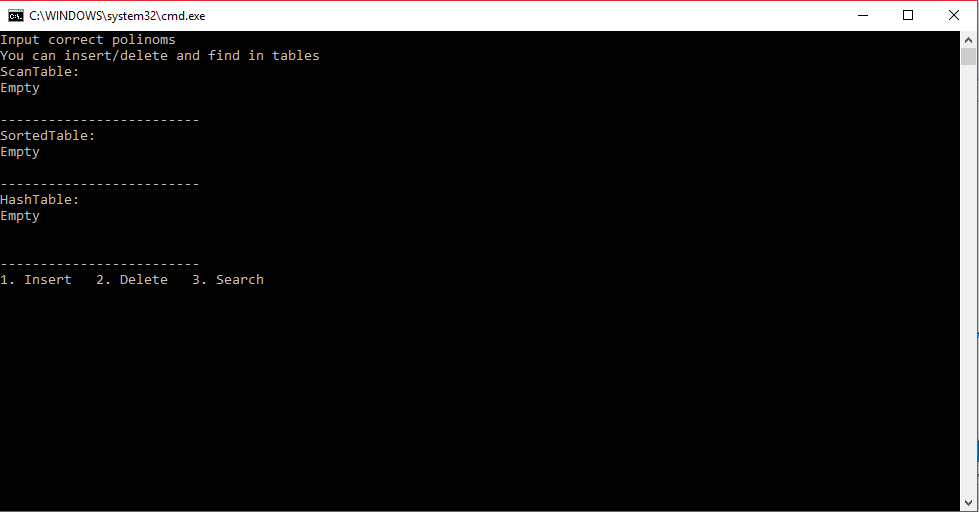
Хэш-таблица — это [структура данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), реализующая интерфейс [ассоциативного массива](https://ru.wikipedia.org/wiki/%D0%90%D1%81%D1%81%D0%BE%D1%86%D0%B8%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2), а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу. Важное свойство хеш-таблиц состоит в том, что все три операции (поиск, вставка, удаление элементов) в среднем выполняются за время меньшее, чем в других способах реализации таблицO ( 1 ) {\displaystyle O(1)}.

# 1. Постановка задачи

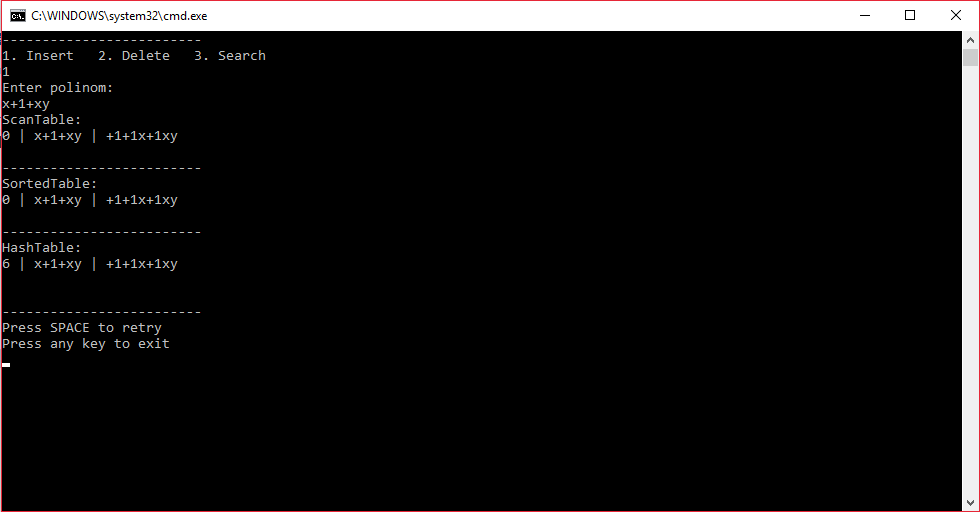
Написать программу, позволяющую записывать, удалять и искать данные в трёх видах таблиц: сортируемых, несортируемых и хэш-таблицах. Данными являются полиномы. Для реализации использовать шаблонный класс таблиц с виртуальными методами и три класса-наследника: сортированные, несортированные и хэш-таблицы.

# 2. Руководство пользователя

В начале работы программа выведет три таблицы и предложит выбрать одно из тёх действий: вставить, удалить или найти элемент.



После выбора действия нужно ввести полином. Для продолжения работы с программой необходимо нажать пробел, для выхода, любую другую кнопку.



После нажатия на пробел можно выбрать следующее действие.

# 3. Руководство программиста

## 1. Описание алгоритмов

**Несортируемые (Просматриваемые) таблицы**

Записи в таблице расположены беспорядочно.

*Вставка*

* Проверяется, полна ли таблица. Если да, происходит перераспределение памяти.
* Запись вставляется в конец таблицы.

*Поиск*

* Запись находится при помощи обхода массива указателей и удаляется.
* На её место ставиться запись из конца таблицы.

*Удаление*

* Запись находится при помощи обхода массива указателей, и выводятся её данные.

**Сортированные таблицы**

Записи в таблице расположены в порядке возрастания ключей.

*Вставка*

* Проверяется, полна ли таблица. Если да, происходит перераспределение памяти.
* Бинарным поиском по ключу находится место, на которое нужно вставить запись, чтобы не нарушить порядок.
* Все записи с большим ключом сдвигаются вниз.
* Запись вставляется на свободное место.

*Удаление*

* Бинарным поиском по ключу находится нужная запись таблицы и удаляется.
* Записи с большим ключом сдвигаются вверх.

*Бинарный поиск*

* Определяется значение ключа в середине массива указателей.
* Ключ сравнивается с этим значением и если он больше, то поиск продолжается во второй половине элементов, если меньше – в первой.
* Поиск сводится к тому, что вновь определяется значение серединного ключа массива указателей и сравнивается и нужным ключом.
* Процесс продолжается до тех пор, пока не будет найден элемент с необходимым значением ключа или интервал для поиска не станет пустым.

**Хэш-таблицы**

Записи вставляются, удаляются и ищутся в таблице при помощи некоторого хэш-алгоритма.

*Вставка*

* Проверяется, полна ли таблица. Если да, происходит перераспределение памяти.
* Значение хэш-функции от нужного ключа – номер элемента в массиве указателей.
* Если нет коллизий, записываем данные в эту ячейку таблицы и обозначаем её непустой.

*Поиск*

* Значение хэш-функции от нужного ключа – номер элемента в массиве указателей.
* Если нет коллизий, выводим запись.

*Удаление*

* Значение хэш-функции от нужного ключа – номер элемента в массиве указателей.
* Если нет коллизий, удаляем запись и обозначаем ячейку пустой.

*Хэш-функция*

Для одного ключа всегда возвращает одно значение. Для разных ключей может вернуть одинаковое значение. Такой случай называется коллизией.

Разрешение коллизий.

При вставке – запись создаётся в следующей свободной строке таблицы.

При удалении и поиске:

* Значение хэш-функции от нужного ключа – номер элемента в массиве указателей.
* Если элемент массива по этому номеру указывает на запись с другим ключом, то последовательно идем по массиву указателей дальше и ищем элемент массива, указывающий на запись, ключ которой совпадает с нужным ключом.
* Если таких ключей нет, то вызывается исключение.
* Если запись нашлась, то удаляем ее и обозначаем пустой или выводим данные записи.

Хэш-функция реализована следующим образом:

* Каждый символ ключа переводится в десятичное число и прибавляется к некоторой переменной.
* Эта переменная и есть значение хэш-функции.

## 2. Описание структуры программы

Программа содержит:

* статическую библиотеку polinome\_lib, содержащую реализацию действий над полиномами, а также тесты;
* table.h – файл с объявлением и реализацией шаблонного класса table и класса TabRecord – строк таблицы;
* scan.h – файл с объявлением и реализацией класса просматриваемых таблиц – наследника класса table;
* sorted.h – файл с объявлением и реализацией класса сортируемых таблиц – наследника класса table;
* hash.h – файл с объявлением и реализацией класса хэш-таблиц – наследника класса table;
* main.cpp – реализация пользовательского приложения;
* test\_hash.cpp – тесты для хэш-таблиц;
* test\_scan.cpp – тесты для просматриваемых таблиц;
* test\_sorted.cpp – тесты для сортированных таблиц;

## 3. Описание структур данных

**Класс TabRecord**

Поля:

string name – ключ записи;

type data – данные записи.

Методы:

TabRecord() – конструктор записи по умолчанию;

TabRecord(const TabRecord& tr) – конструктор копирования;

TabRecord& operator=(const TabRecord& tr) – оператор присваивания;

TabRecord(const string& KEY, const type& DATA) – конструктор записи.

**Класс Table**

Поля:

TabRecord<type>\*\* Records – массив указателей на записи таблицы;

int Max – максимально возможное количество записей в таблице;

int CurrSize – текущее количество записей;

int CurrIndex – номер текущей записи таблицы.

Методы:

Table(int i = 10) – конструктор;

Table(const Table<type>& Tab) – конструктор копирования;

virtual ~Table() – деструктор;

virtual void Realloc() – перераспределение памяти;

virtual void Reset() – установление CurrIndex на начало таблицы;

virtual int IsEnded() const – проверка, кончилась ли таблица;

virtual type& GetCurr() const – возвращает данные текущей записи таблицы;

virtual void SetNext() – передвигает текущую запись таблицы на следующую;

virtual type& Search(const string& KEY) const = 0 - поиск;

virtual void Insert(const string& KEY, const type& DATA) = 0 - вставка;

virtual void Delete(const string& KEY) = 0 - удаление;

template<class type> friend ostream& operator<< (std::ostream& os, const Table<type>& Tab) – вывод таблицы на экран.

**Класс ScanTable**

Поля и методы унаследованы от класса Table.

**Класс SortedTable**

Поля и методы унаследованы от класса Table.

Метод int BinarySearch(const string& KEY) const – бинарный поиск по ключу.

**Класс HashTable**

Поля и методы унаследованы от класса Table.

Помимо унаследованных:

Поле int\* H – массив, хранящий данные о пустоте/занятости каждой ячейки таблицы.

Метод int Hash(const string& KEY) const – хэш-функция от ключа.

# Заключение

Была разработана программа, позволяющая вводить, искать и удалять полиномы из таблиц трёх разных типов: сортированных, несортированных и хэш-таблиц. Программа реализована на основе шаблонного класса и трёх его классов-наследников.

# Литература

1. *Тимофеев В. В.* Самоучитель по С++ - М.: Издательство Бином, 2014. -338 с., ил.

2. *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы: построение и анализ, Москва 2013.

3. Рабочие материалы студента по общему курсу «Методы программирования

# Приложение

## main.cpp

#include "polinom.h"

#include <conio.h>

#include "scan.h"

#include "sorted.h"

#include "hash.h"

void main()

{

cout << "Input correct polinoms" << endl;

cout << "You can insert/delete and find in tables" << endl;

char c = 0, str;

ScanTable<TPolinome> Tab1;

SortedTable<TPolinome> Tab2;

HashTable<TPolinome> Tab3;

string key;

TPolinome P;

while (!c || (c == ' '))

{

cout << "ScanTable: " << endl << Tab1 << endl;

cout << "-------------------------" << endl;

cout << "SortedTable:" << endl << Tab2 << endl;

cout << "-------------------------" << endl;

cout << "HashTable:" << endl << Tab3 << endl << endl;

cout << "-------------------------" << endl;

cout << "1. Insert 2. Delete 3. Search" << endl;

cin >> str;

switch (str)

{

case '1':

{

cout << "Enter polinom:" << endl;

getline(cin, key);

if (key == "")

getline(cin, key);

P = TPolinome(key);

Tab1.Insert(key, P);

Tab2.Insert(key, P);

Tab3.Insert(key, P);

cout << "ScanTable: " << endl << Tab1 << endl;

cout << "-------------------------" << endl;

cout << "SortedTable:" << endl << Tab2 << endl;

cout << "-------------------------" << endl;

cout << "HashTable:" << endl << Tab3 << endl << endl;

cout << "-------------------------" << endl;

break;

}

case '2':

{

cout << "Enter polinom:" << endl;

getline(cin, key);

if (key == "")

getline(cin, key);

Tab1.Delete(key);

Tab2.Delete(key);

Tab3.Delete(key);

cout << "ScanTable: " << endl << Tab1 << endl;

cout << "-------------------------" << endl;

cout << "SortedTable:" << endl << Tab2 << endl;

cout << "-------------------------" << endl;

cout << "HashTable:" << endl << Tab3 << endl << endl;

cout << "-------------------------" << endl;

break;

}

case '3':

{

cout << "Enter polinom:" << endl;

getline(cin, key);

if (key == "")

getline(cin, key);

Tab1.Search(key);

Tab2.Search(key);

Tab3.Search(key);

cout << "ScanTable: " << endl << Tab1 << endl;

cout << "-------------------------" << endl;

cout << "Search in ScanTable: " << Tab1.Search(key) << endl;

cout << "-------------------------" << endl;

cout << "SortedTable:" << endl << Tab2 << endl;

cout << "-------------------------" << endl;

cout << "Search in SortedTable: " << Tab2.Search(key) << endl;

cout << "-------------------------" << endl;

cout << "HashTable:" << endl << Tab3 << endl << endl;

cout << "-------------------------" << endl;

cout << "Search in HashTable: " << Tab3.Search(key) << endl;

cout << "-------------------------" << endl;

break;

}

default:

cout << "try again" << endl;

}

cout << "Press SPACE to retry" << endl;

cout << "Press any key to exit" << endl;

c = \_getch();

}

}

## table.h

#pragma once

#include <string>

#include <iostream>

#define Cf 5

using namespace std;

template <typename type>

class TabRecord

{

public:

string name;

type data;

TabRecord()

{

name = "emp";

};

TabRecord(const TabRecord& tr)

{

name = tr.name;

data = tr.data;

};

TabRecord& operator=(const TabRecord& tr)

{

name = tr.name;

data = tr.data;

};

TabRecord(const string& KEY, const type& DATA)

{

name = KEY;

data = DATA;

};

};

template <typename type>

class Table

{

protected:

TabRecord<type>\*\* Records;

int Max;

int CurrSize;

int CurrIndex;

public:

Table(int i = 10)

{

Max = i;

CurrSize = 0;

CurrIndex = -1;

Records = new TabRecord<type>\*[Max];

};

Table(const Table<type>& Tab);

virtual ~Table()

{

delete[] Records;

};

virtual void Realloc();

virtual void Reset()

{

if (CurrSize) CurrIndex = 0;

else CurrIndex = -1;

};

virtual int IsEnded() const

{

return CurrIndex == -1 || ((CurrIndex + 1) == CurrSize);

};

virtual type& GetCurr() const

{

if (CurrSize) return Records[CurrIndex]->data;

else throw "Empty";

};

virtual void SetNext()

{

if (CurrSize) CurrIndex++;

else throw "Empty";

if (CurrIndex == CurrSize) Reset();

};

virtual type& Search(const string& KEY) const = 0;

virtual void Insert(const string& KEY, const type& DATA) = 0;

virtual void Delete(const string& KEY) = 0;

template<class type> friend ostream& operator<< (std::ostream& os, const Table<type>& Tab);

};

template<typename type>

Table<type>::Table(const Table<type>& Tab)

{

Max = Tab.Max;

CurrSize = Tab.CurrSize;

CurrIndex = Tab.CurrIndex;

Records = new TabRecord<type>\*[Max];

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

Records[i] = Tab.Records[i];

}

template<typename type>

void Table<type>::Realloc()

{

int NewMax = Max \* Cf;

TabRecord<type>\*\* temp = new TabRecord<type>\*[NewMax];

Reset();

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

temp[i] = Records[i];

delete[] Records;

Records = temp;

Max = NewMax;

};

template <typename type>

ostream& operator<< (ostream& os, const Table<type>& Tab)

{

if (Tab.CurrSize)

for (int i = 0; i < Tab.CurrSize; i++)

os << i << " | " << Tab.Records[i]->name << " | " << Tab.Records[i]->data << endl;

else

os << "Empty" << endl;

return os;

}

## hash.h

#pragma once

#include "table.h"

template<typename type>

class HashTable :public Table<type>

{

protected:

int\* H;

int Hash(const string& KEY) const;

virtual void Realloc();

public:

HashTable(int i = 10);

HashTable(const HashTable& Tab);

~HashTable()

{

delete[] H;

};

virtual void Reset();

virtual type& GetCurr() const;

virtual void SetNext();

virtual type& Search(const string& KEY) const;

virtual void Insert(const string& KEY, const type& DATA);

virtual void Delete(const string& KEY);

template<class type> friend ostream& operator<< (std::ostream& os, const HashTable<type>& Tab);

};

template<typename type>

HashTable<type>::HashTable(int i = 10) : Table(i)

{

H = new int[i];

for (int j = 0; j < Max; j++)

H[j] = 0;

};

template<typename type>

int HashTable<type>::Hash(const string& KEY) const

{

int seed = 0;

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

seed = seed + int(KEY[i]);

return seed % Max;

}

template<typename type>

void HashTable<type>::Realloc()

{

int NewMax = Max \* Cf;

TabRecord<type>\*\* temp = new TabRecord<type>\*[NewMax];

int\* tempH = new int[NewMax];

int i = 0;

for (i; i < Max; i++)

{

temp[i] = Records[i];

tempH[i] = H[i];

}

delete [] Records;

delete [] H;

Records = temp;

H = tempH;

Max = NewMax;

}

template<typename type>

HashTable<type>::HashTable(const HashTable& Tab)

{

Max = Tab.Max;

CurrSize = Tab.CurrSize;

CurrIndex = Tab.CurrIndex;

Records = new TabRecord<type>\*[Max];

H = new int[Max];

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

{

H[i] = Tab.H[i];

Records[i] = Tab.Records[i];

}

}

template<typename type>

void HashTable<type>::Reset()

{

if (CurrSize)

{

CurrIndex = 0;

while (H[CurrIndex] != 1)

CurrIndex++;

}

else

CurrIndex = -1;

}

template<typename type>

type& HashTable<type>::GetCurr() const

{

if (CurrSize)

return Records[CurrIndex]->data;

else

throw "Empty";

}

template<typename type>

void HashTable<type>::SetNext()

{

if (CurrSize)

{

CurrIndex++;

while (H[CurrIndex] != 1)

CurrIndex = (CurrIndex + 1) % Max;

}

else

throw "Empty";

}

template<typename type>

type& HashTable<type>::Search(const string& KEY) const

{

HashTable<type> Temp(\*this);

Temp.Reset();

if (Temp.CurrSize)

{

Temp.CurrIndex = Temp.Hash(KEY);

int l = Temp.CurrIndex;

if (Temp.Records[Temp.CurrIndex]->name == KEY)

return Temp.Records[Temp.CurrIndex]->data;

else

{

while (Temp.H[Temp.CurrIndex] && ((Temp.CurrIndex + 1) != l) && (Temp.Records[Temp.CurrIndex]->name != KEY))

Temp.CurrIndex = (Temp.CurrIndex + 1) % Temp.Max;

if(Temp.Records[Temp.CurrIndex]->name == KEY)

return Temp.Records[Temp.CurrIndex]->data;

else throw "Key isn't exist";

}

}

else throw "Empty";

}

template<typename type>

void HashTable<type>::Insert(const string& KEY, const type& DATA)

{

if (CurrSize == Max) Realloc();

CurrIndex = Hash(KEY);

if (!H[CurrIndex])

{

Records[CurrIndex] = new TabRecord<type>(KEY, DATA);

CurrSize++;

H[CurrIndex] = 1;

}

else

if (Records[CurrIndex]->name != KEY)

{

int l = CurrIndex;

while (H[CurrIndex] && ((CurrIndex + 1) != l))

CurrIndex = (CurrIndex + 1) % Max;

Records[CurrIndex] = new TabRecord<type>(KEY, DATA);

CurrSize++;

H[CurrIndex] = 1;

}

else throw "key already exists";

}

template<typename type>

void HashTable<type>::Delete(const string& KEY)

{

Reset();

if (CurrSize)

{

CurrIndex = Hash(KEY);

int l = CurrIndex;

if (H[CurrIndex])

{

if (Records[CurrIndex]->name != KEY)

{

while (H[CurrIndex] && ((CurrIndex + 1) != l) && (Records[CurrIndex]->name != KEY))

CurrIndex = (CurrIndex + 1) % Max;

if (!H[CurrIndex] || Records[CurrIndex]->name != KEY)

throw "Key does not exist";

else

{

Records[CurrIndex] = new TabRecord<type>;

H[CurrIndex] = 0;

CurrSize--;

}

}

else

{

Records[CurrIndex] = new TabRecord<type>;

H[CurrIndex] = 0;

CurrSize--;

}

}

else

throw "Key isn't exist";

}

else

throw "Empty";

}

template <typename type>

ostream& operator<< (ostream& os, const HashTable<type>& Tab)

{

if (Tab.CurrSize)

{

for (int i = 0; i < Tab.Max; i++)

if (Tab.H[i])

os << i << " | " << Tab.Records[i]->name << " | " << Tab.Records[i]->data << endl;

}

else

os << "Empty" << endl;

return os;

}

## scan.h

#pragma once

#include "table.h"

template<typename type>

class ScanTable :public Table<type>

{

public:

ScanTable(int i = 10) : Table(i) {};

ScanTable(const ScanTable<type>& Tab) : Table(Tab) {};

~ScanTable() {};

virtual type& Search(const string& KEY) const;

virtual void Insert(const string& KEY, const type& DATA);

virtual void Delete(const string& KEY);

};

template<typename type>

type& ScanTable<type>::Search(const string& KEY) const

{

ScanTable<type> Temp(\*this);

Temp.Reset();

if (Temp.CurrIndex > -1)

{

while ((Temp.CurrIndex > -1) && (Temp.Records[Temp.CurrIndex]->name != KEY) && (Temp.CurrIndex < Temp.CurrSize))

Temp.SetNext();

if (Temp.CurrIndex < Temp.CurrSize)

return Temp.Records[Temp.CurrIndex]->data;

else

throw "key isn't exist";

}

else throw "Empty";

}

template<typename type>

void ScanTable<type>::Insert(const string& KEY, const type& DATA)

{

if (CurrSize == Max)

Realloc();

Reset();

if (CurrSize)

{

while ((CurrIndex < CurrSize) && (Records[CurrIndex]->name != KEY))

CurrIndex++;

if (CurrIndex == CurrSize)

{

Records[CurrIndex] = new TabRecord<type>(KEY, DATA);

CurrSize++;

}

else throw "Key already exists";

}

else

{

CurrIndex++;

Records[CurrIndex] = new TabRecord<type>(KEY, DATA);

CurrSize++;

}

}

template<typename type>

void ScanTable<type>::Delete(const string& KEY)

{

Reset();

while ((CurrIndex > -1) && (Records[CurrIndex]->name != KEY) && (CurrIndex < CurrSize))

CurrIndex++;

if (CurrSize && (CurrIndex < CurrSize))

{

if (CurrSize > 1)

Records[CurrIndex] = Records[--CurrSize];

else

CurrSize = 0;

}

else

throw "Key isn't exist";

}

## sorted.h

#pragma once

#include "table.h"

template<typename type>

class SortedTable : public Table<type>

{

public:

SortedTable(int i = 10) : Table(i) {};

SortedTable(const SortedTable<type>& Tab) : Table(Tab) {};

~SortedTable() { };

int BinarySearch(const string& KEY) const;

virtual type& Search(const string& KEY) const;

virtual void Insert(const string& KEY, const type& DATA);

virtual void Delete(const string& KEY);

};

template<typename type>

int SortedTable<type>::BinarySearch(const string& KEY) const

{

int i = 0, j = CurrSize - 1;

int mid;

while (i <= j)

{

mid = (i + j) / 2;

if (KEY > Records[mid]->name)

i = mid + 1;

else

j = mid - 1;

}

return i;

}

template<typename type>

type& SortedTable<type>::Search(const string& KEY) const

{

SortedTable<type> Temp(\*this);

Temp.Reset();

if (CurrIndex > -1)

{

int k = Temp.BinarySearch(KEY);

if (Temp.Records[k]->name == KEY)

return Temp.Records[k]->data;

else

throw "Key isn't exist";

}

else

throw "Empty";

}

template<typename type>

void SortedTable<type>::Insert(const string& KEY, const type& DATA)

{

if (CurrSize == Max) Realloc();

Reset();

if (CurrSize)

{

int k = BinarySearch(KEY);

if (k != CurrSize)

{

if (Records[k]->name != KEY)

{

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

Records[i] = Records[i - 1];

Records[k] = new TabRecord<type>(KEY, DATA);

CurrSize++;

}

else

throw "Key already exists";

}

else

{

Records[k] = new TabRecord<type>(KEY, DATA);

CurrSize++;

}

}

else

{

CurrIndex++;

Records[CurrIndex] = new TabRecord<type>(KEY, DATA);

CurrSize++;

}

}

template<typename type>

void SortedTable<type>::Delete(const string& KEY)

{

Reset();

if (CurrSize)

{

int k = BinarySearch(KEY);

if (Records[k]->name == KEY)

{

for (int i = k; i < CurrSize - 1; i++)

Records[i] = Records[i + 1];

CurrSize--;

}

else throw "Key isn't exist";

}

else throw "Empty";

}