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

**«Структура данных типа таблица»**

**Выполнил:** студент группы 381603-1  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Вихрев И.Б.

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

Нижний Новгород

2018

Содержание

[Введение 3](#_Toc515641873)

[1 Постановка учебно-практической задачи 4](#_Toc515641874)

[2 Руководство пользователя 5](#_Toc515641875)

[3 Руководство программиста 7](#_Toc515641876)

[3.1 Описание структуры программы 7](#_Toc515641877)

[3.2 Описание структур данных и алгоритмов 8](#_Toc515641878)

[3.2.1 Шаблон структуры TabRecord 8](#_Toc515641879)

[3.2.2 Абстрактный класс Table. 8](#_Toc515641880)

[3.2.3 Шаблон класса неупорядоченной таблицы. 8](#_Toc515641881)

[3.2.4 Шаблон класса упорядоченной таблицы. 8](#_Toc515641882)

[3.2.5 Шаблон класса хэш-таблицы 9](#_Toc515641883)

[3.3 Описание программной реализации 11](#_Toc515641884)

[Заключение 13](#_Toc515641885)

[Список литературы 14](#_Toc515641886)

[4 Приложение 15](#_Toc515641887)

[4.1 Приложение 1. Заголовочный файл table.h 15](#_Toc515641888)

[4.2 Приложение 2. Заголовочный файл unord\_table.h 16](#_Toc515641889)

[4.3 Приложение 3. Заголовочный файл ord\_table.h 18](#_Toc515641890)

[4.4 Приложение 4. Заголовочный файл hash\_table.h 20](#_Toc515641891)

[4.5 Приложение 5. Файл sample.cpp 24](#_Toc515641892)

# Введение

Таблица — набор элементов одинаковой организации, каждый из которых можно представить в виде пары **<K, V>,** где **K** — ключ, а **V** — тело (информационная часть) элемента.

Ключ уникален для каждого элемента, то есть в таблице нет элементов с одинаковыми ключами. Ключ используется для доступа к элементам при выполнении операций.

Существует три вида таблиц: просматриваемые(неупорядоченные), упорядоченные и хэш-таблицы. Данная работа направлена на изучение этих структур данных и их особенностей.

В отчёте приводится постановка задачи, описание использующихся алгоритмов, описание программы и правила её использования, а также прилагается код программы, решающей поставленную задачу.

# Постановка учебно-практической задачи

*Формулировка задачи*:

Реализовать 3 класса таблиц: просматриваемые, упорядоченные и хэш-таблицы. Строка таблицы содержит данные и ключ. Таблицы должны поддерживать три операции по работе с данными: вставка, поиск и удаление.

В просматриваемых таблицах данные должны храниться в порядке записи, в упорядоченных необходимо сортировать данные по ключу. В хэш-таблицах данные записываются в строку, индекс которой — результат работы хэш-функции.

В таблицах хранить полиномы, реализованные в лабораторной работе “Арифметические операции с полиномами”. Ключом выступает строковое представление полиномов.

Поиск, вставка и удаление данных происходит одновременно во все три таблицы. Результат выполнения операции должен быть выведен на экран для всех трёх таблиц.

Работоспособность программы проверить с помощью Google Tests Framework.

*Исходные данные*:

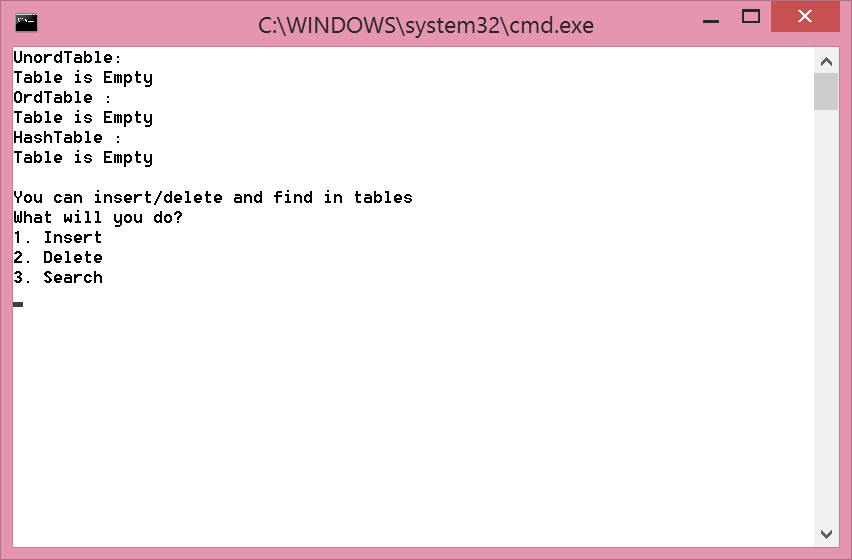
Строковое представление полинома.

*Требуемый результат*:

Вывести на экран результат применения указанной операции. При вставке, и удалении – отобразить состояние всех трёх таблиц. При поиске вывести на экран полином или сообщение об его отсутствии в таблицах.

# Руководство пользователя

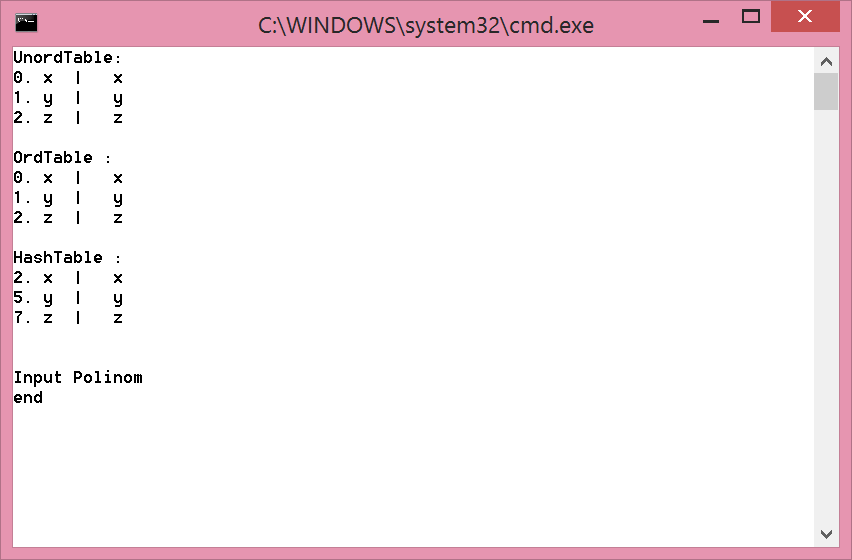
Для выполнения лабораторной работы необходимо запустить файл **sample.exe.** На экране появится пользовательский интерфейс: в верхней части экрана отображаются 3 вида таблиц (неупорядоченные, упорядоченные и хэш-таблицы), ниже расположено пользовательское меню (Рис. 1).



1. Начало работы программы.

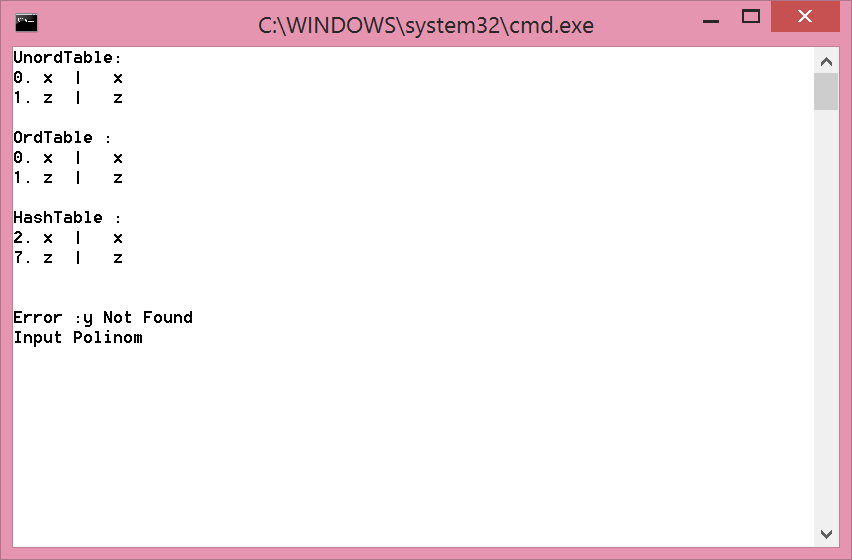
Навигация по меню осуществляется с помощью ввода соответствующих цифр. Для возвращения в меню необходимо ввести **end**.

1. **Insert (Вставка).**

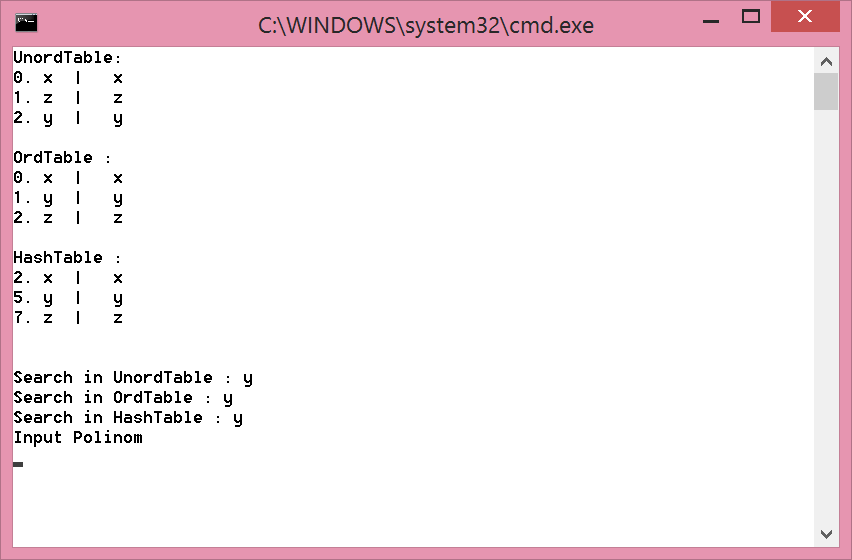


1. Вставка данных в таблицы.
2. **Delete (Удаление)**

Удаление происходит по ключу. Если полином не был найден, выведется соответствующее сообщение (Рис. 3).



1. Попытка удаления элемента, которого нет в таблице.
2. **Search (Поиск)**



1. Поиск в таблице.

При выходе в меню вам будет предложено либо продолжить работу с таблицами, либо завершить ее. Для продолжения необходимо нажать любую клавишу***,*** для завершения ***Esc***.

# Руководство программиста

## Описание структуры программы

*Структура решения*:

1. Статическая библиотека **gtest**.
2. Статическая библиотека **polinom\_lib**.
3. Статическая библиотека **table\_lib.**
4. Проект **table\_test**.
5. Проект **sample**.

Библиотека **gtest** необходима для работы Google Tests Framework.

Библиотека **polinom\_lib** была описана в отчёте “Арифметические операции с полиномами”.

Библиотека **table\_lib** состоит из нескольких программных модулей:

1. Заголовочный файл **table.h** содержит объявление абстрактного класса **Table.**
2. Заголовочный файл **unord\_table.h**содержит объявление шаблонного класса *просматриваемых таблиц* и его реализацию.
3. Заголовочный файл **ord\_table.h** содержит объявление класса *упорядоченных таблиц* и его реализацию.
4. Заголовочный файл **hash\_table.h** содержит объявление и реализацию класса *хэш-таблиц*.

Проект **table\_test** содержит тесты функций библиотеки **table\_lib**.

Проект **sample** содержит реализацию пользовательского интерфейса.

## Описание структур данных и алгоритмов

### Шаблон структуры TabRecord

Содержит 2 поля:

1. Строка-ключ
2. Данные.

Перегружен оператор присваивания.

### Абстрактный класс Table.

Содержит 4 поля:

1. Массив указателей на **TabRecord.**
2. Максимальное количество записей в таблице.
3. Текущее количество записей в таблице.
4. Текущий индекс.

Абстрактный класс содержит виртуальные методы: увеличение размера таблицы, вставка, поиск и удаление, некоторые методы навигации. Часть виртуальных методов определена в абстрактном классе и используется 2 классами-наследниками.

### Шаблон класса неупорядоченной таблицы.

Поля класса унаследованы от класса **Table**.

В данном классе реализованы вставка, поиск и удаление.

* Вставка происходит неупорядоченно, т.е. элементы записываются друг за другом в массив.
* Поиск — полный перебор.
* На место удаленного элемента записываем последний элемент массива, при этом уменьшаем количество текущих записей на 1.

### Шаблон класса упорядоченной таблицы.

Поля класса унаследованы от класса **Table**.

В данном классе реализованы вставка, поиск и удаление.

Вставка происходит упорядоченно:

1. Сравниваем элемент, который хотим вставить, с текущим элементом.
2. Если текущий больше, — вставляем элемент на его место. Этот элемент и все последующие сдвигаем на одно место вперед.

Если текущий меньше — сравниваем со следующим элементом.

1. Алгоритм продолжается до тех пор, пока массив элементов не закончится или элемент не будет вставлен.

Удаление также не должно нарушать упорядоченность массива. При удалении происходит перепаковка массива: Все элементы, следующие за удаленным сдвигаются на одно место влево.

Используется алгоритм бинарного поиска:

1. Определение значения элемента в середине структуры данных. Полученное значение сравнивается с ключом.
2. Если ключ меньше значения середины, то поиск осуществляется в первой половине элементов, иначе — во второй.
3. Поиск сводится к тому, что вновь определяется значение серединного элемента в выбранной половине и сравнивается с ключом.
4. Процесс продолжается до тех пор, пока не будет найден элемент со значением ключа или не станет пустым интервал для поиска.

### Шаблон класса хэш-таблицы

К полям класса, помимо унаследованных от класса **Table,** добавлен целочисленный массив flag, в котором хранится признак состояния записи таблицы:

1. 0 — пустая запись.
2. 1 — занятая запись.
3. -1 — удаленная запись.

Хэш-таблица организована таким образом, что поиск, вставка и удаление используют хэш-функцию.

Хэш-функция на основе ключа генерирует индекс строки таблицы, в которую будут записаны данные и ключ. Коллизии разрешаются способом открытого перемешивания.

Хэш-функция:

1. Идем по строке и суммируем все символы из строки.
2. Умножаем сумму на , где n – количество символов в строке.( 13 ближайшее простое число к кол-ву цифр в используемом алфавите).
3. Устанавливаем полученное число как зерно для генератора псевдослучайных чисел.
4. Возвращаем сгенерированное число, которое мы ограничили размерами таблицы.

Вставка:

1. На вход подается элемент, который необходимо вставить в таблицу.
2. По ключу элемента вычисляется хэш-функция, она возвращает индекс строки таблицы, в которой должен быть записан элемент.
3. Если текущая строка свободна происходит запись элемента в таблицу. Если строка занята, проверяем свободна ли следующая строка и т.д. Как только попадается свободная строка, записываем в нее элемент.

Удаление:

1. Используется функция поиска в таблице. Побочный эффект от использования функции – индекс элемента в таблице.
2. Удаляем элемент, находящийся по индексу, который вернул поиск.

Поиск:

1. На вход подается ключ элемента, который надо найти
2. По ключу элемента вычисляется хэш-функция, она возвращает индекс строки таблицы, в которой должен быть записан искомый элемент.
3. Сравниваем ключ искомого элемента и элемента, находящего в строке, индекс которой вернула хэш-функция. Если она равны – возвращаем указатель на элемент. Если не равны, то поиск вырождается в линейный поиск.
4. Если ключ не был найден в результате линейного поиска, выдается сообщение об отсутствии ключу в таблице.

Если хэш-таблица была заполнена, вызывается функция увеличения ее размера и перехеширования.

## Описание программной реализации

1. Шаблон структуры **TabRecord** – строка таблицы

template <typename T>

class TabRecord

{

public:

string Key; // Ключ

T Data; // Данные

TabRecord();//Конструктор по умолчанию

TabRecord(string K , T D );//Конструктор с параметрами

TabRecord(const TabRecord& TR);//Конструктор копирования

TabRecord<T>& operator=(const TabRecord<T>& TabR);//оператор присваивания

}

1. Шаблон абстрактного класса **Table**

template <typename T>

class Table

{

protected:

TabRecord<T>\*\* Rows; // массив указателей на строки

unsigned int MaxRecords; // максимальное число элементов

unsigned int CurrRecords; // текущее число элементов

int CurrIndex; //текущий индекс

virtual void Realloc() = 0; // реаллок

public:

Table(unsigned int i = 10); ();//Конструктор по умолчанию

Table(const Table<T>& TabToCopy);//Конструктор копирования

virtual ~Table();//деструктор

//Общие методы

virtual void Insert(const T Data, const string Key) = 0;//вставка

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

virtual T\* Search(const string Key) = 0;//поиск

//Методы навигации

Virtual void Reset();//сброс индекса

virtual bool IsTabEnded() const;//Проверка на конец таблицы

bool IsFull() const;//проверка на полноту

bool IsEmpty() const;//проверка на пустоту

virtual void SetNext();//установить индекс на следующий элемент

virtual T\* GetCurr() const;// получить указатель на текущий элемент

int GetCurrRecords() const ;// получить кол-во записей

int GetMaxRecords() const;// получить макс кол-во записей

};

1. Шаблон класса UnordTable

template <typename T>

class UnordTable : public Table<T>

{

protected:

void Realloc();

public:

// Конструкторы и деструктор

UnordTable(unsigned int i = 10) : Table(i) {};

UnordTable(const UnordTable<T>& TabToCopy);

~UnordTable() { };

//Общие Методы

void Insert(const T Data, const string Key);

T\* Search(const string Key);

void Delete(const string Key);

template<class T> friend std::ostream& operator<< (std::ostream& os, const UnordTable<T>& Tab); // оператор вывода

};

1. Шаблон класса OrdTable

template <typename T>

class OrdTable : public Table<T>

{

protected:

void Realloc();

public:

// Конструкторы и деструктор

OrdTable(unsigned int i = 10);

OrdTable(const OrdTable<T>& TabToCopy);

~OrdTable() {};

//Методы

void Insert(const T Data, const string Key);

T\* Search(const string Key);

void Delete(const string Key);

template<class T> friend std::ostream& operator<< (std::ostream& os, const OrdTable<T>& Tab); // оператор вывода

};

1. Шаблон класса HashTable

template <typename T>

class HashTable : public Table<T>

{

protected:

int\* flag;//массив признаков пуст, полон, удалён

int HashFunc(string Key) const;// хэш-функция

void Realloc();//реаллок

public:

// Конструкторы и деструктор

HashTable(unsigned int i = 10);

HashTable(const HashTable<T>& TabToCopy);

~HashTable() { delete[] flag;};

//Общие Методы

void Insert(const T Data, const string Key);

T\* Search(const string Key);

void Delete(const string Key);

//Методы навигации

void SetNext();

T\* GetCurr() const;

void Reset();

bool IsTabEnded() const { return false; };

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

};

# Заключение

В лабораторной работе была реализована программа для работы с 3 видами таблиц. В ходе работы были выяснены недостатки и преимущества каждой из реализаций. Так, например, упорядоченные таблицы подходят для случаев, когда вставка и удаление происходят редко, иначе будут возникать частые перепаковки, что негативно скажется на времени работы программы. Хэш-таблицы же оптимальны в общем случае, т.к. при приемлемых условиях, все три операции в среднем выполняются за константное время.

# Список литературы

1. *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К*. Алгоритмы: построение и анализ, Москва 2013.
2. Рабочие материалы к учебному курсу «Методы программирования». *Гергель В.П*. 2002 г.

# Приложение

## Приложение 1. Заголовочный файл table.h

#pragma once

#include <string>

#include <iostream>

using std::string;

using std::endl;

using std::cout;

template <typename T>

class TabRecord

{

public:

string Key;

T Data;

TabRecord() { Key = "EMPTY RECORD"; Data = 0;}

TabRecord(string K , T D ) { Key = K; Data = D;} //

TabRecord(const TabRecord& TR) { Key = TR.Key; Data = TR.Data; }

TabRecord<T>& operator=(const TabRecord<T>& TabR) { Data = TabR.Data; Key = TabR.Key; return \*this; }

};

template <typename T>

class Table

{

protected:

TabRecord<T>\*\* Rows; // массив указателей

unsigned int MaxRecords;

unsigned int CurrRecords;

int CurrIndex;

virtual void Realloc() = 0;

public:

Table(unsigned int i = 10);

Table(const Table<T>& TabToCopy);

virtual ~Table() { delete[] Rows; }; //?

//Общие методы

virtual void Insert(const T Data, const string Key) = 0;

virtual void Delete(const string key) = 0;

virtual T\* Search(const string Key) = 0;

//Методы навигации

virtual void Reset();

virtual bool IsTabEnded() const { return CurrIndex == CurrRecords || CurrIndex == -1; };

bool IsFull() const { return CurrRecords == MaxRecords; };

bool IsEmpty() const { return CurrRecords == 0; };

virtual void SetNext();

virtual T\* GetCurr() const;

int GetCurrRecords() const { return CurrRecords; };

int GetMaxRecords() const { return MaxRecords; };

};

//..........................................................

template<typename T>

Table<T>::Table(unsigned int i)

{

MaxRecords = i;

CurrRecords = 0;

CurrIndex = -1;

Rows = new TabRecord<T>\*[MaxRecords];

}

//..........................................................

template<typename T>

void Table<T>::Reset()

{

if(CurrRecords > 0)

CurrIndex = 0;

else

CurrIndex = -1;

}

//..........................................................

template<typename T>

T\* Table<T>::GetCurr() const

{

T\* tmp;

if (CurrIndex >= 0 && CurrIndex<CurrRecords)

{

tmp = &(Rows[CurrIndex]->Data);

}

else

{

throw (string)"Table Is Empty";

}

return tmp;

}

//..........................................................

template<typename T>

void Table<T>::SetNext()

{

if (CurrIndex != -1)

CurrIndex++;

else

throw (string)"Table Is Empty";

if (IsTabEnded())

Reset();

}

template <typename T>

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

{

MaxRecords = TabToCopy.MaxRecords;

CurrIndex = TabToCopy.CurrIndex;

CurrRecords = TabToCopy.CurrRecords;

delete[] Rows;

Rows = new TabRecord<T>\*[MaxRecords];

if (!IsEmpty())

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

Rows[i] = new TabRecord<T>(\*(TabToCopy.Rows[i]));

}

## Приложение 2. Заголовочный файл unord\_table.h

#pragma once

#include "table.h"

template <typename T>

class UnordTable : public Table<T>

{

protected:

void Realloc();

public:

// Конструкторы и деструктор

UnordTable(unsigned int i = 10) : Table(i) {};

UnordTable(const UnordTable<T>& TabToCopy);

~UnordTable() { };

//Общие Методы

void Insert(const T Data, const string Key);

T\* Search(const string Key);

void Delete(const string Key);

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

};

//............................................................................

template <typename T>

UnordTable<T>::UnordTable(const UnordTable<T>& TabToCopy) : Table(TabToCopy)

{

}

//............................................................................

template <typename T>

void UnordTable<T>::Realloc()

{

unsigned int NewMaxRecords = (unsigned int)(MaxRecords\*1.5);

TabRecord<T>\*\* tmp = new TabRecord<T>\*[NewMaxRecords];

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

tmp[i] = Rows[i];

MaxRecords = NewMaxRecords;

delete[] Rows;

Rows = tmp;

}

//............................................................................

template <typename T>

void UnordTable<T>::Insert(const T Data, const string Key)

{

if (IsFull())

Realloc();

TabRecord<T>\* Row = new TabRecord<T>(Key, Data);

Reset();

while (!IsTabEnded() && Key != Rows[CurrIndex]->Key)

CurrIndex++;

if (IsEmpty())

{

CurrIndex++;

}

if (CurrIndex == CurrRecords)

{

Rows[CurrIndex] = Row;

CurrRecords++;

Reset();

}

else

{

Reset();

string s = "Key: " + Key + " - isn`t unique";

throw s;

}

}

//............................................................................

template <typename T>

T\* UnordTable<T>::Search(const string Key)

{

Reset();

T\* tmp = nullptr;

if (IsEmpty())

throw (string)"Cant Search In Empty Table";

while (!IsTabEnded() && Key != Rows[CurrIndex]->Key)

CurrIndex++;

if (!IsTabEnded())

tmp = &(Rows[CurrIndex]->Data);

else

{

string s = Key + " Not Found";

throw s;

}

return tmp;

}

//............................................................................

template <typename T>

void UnordTable<T>::Delete(string Key)

{

Reset();

if (IsEmpty())

throw (string)"Cant Delete From Empty Table";

Search(Key);

if (CurrRecords > 1)

Rows[CurrIndex] = Rows[--CurrRecords];

else

CurrRecords = 0;

}

//............................................................................

//спросить почему так

template <typename T>

std::ostream& operator<< (std::ostream& os, const UnordTable<T>& Tab)

{

unsigned int i = 0;

while(i < Tab.CurrRecords)

{

os << i << ". " << Tab.Rows[i]->Key<< " | " << Tab.Rows[i]->Data << std::endl;

i++;

}

if (Tab.CurrRecords == 0)

os << "Table is Empty";

return os;

}

## Приложение 3. Заголовочный файл ord\_table.h

#include "table.h"

template <typename T>

class OrdTable : public Table<T>

{

protected:

void Realloc();

public:

// Конструкторы и деструктор

OrdTable(unsigned int i = 10);

OrdTable(const OrdTable<T>& TabToCopy);

~OrdTable() {};

//Методы

void Insert(const T Data, const string Key);

T\* Search(const string Key);

void Delete(const string Key);

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

};

//............................................................................

template <typename T>

OrdTable<T>::OrdTable(unsigned int i) : Table(i)

{

}

//............................................................................

template <typename T>

OrdTable<T>::OrdTable(const OrdTable<T>& TabToCopy) : Table(TabToCopy)

{

}

//............................................................................

template <typename T>

void OrdTable<T>::Realloc()

{

unsigned int NewMaxRecords = (unsigned int)(MaxRecords\*1.5);

TabRecord<T>\*\* tmp = new TabRecord<T>\*[NewMaxRecords];

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

tmp[i] = Rows[i];

MaxRecords = NewMaxRecords;

delete[] Rows;

Rows = tmp;

}

//............................................................................

template <typename T>

void OrdTable<T>::Insert(const T Data, const string Key)

{

if (IsFull())

Realloc();

TabRecord<T>\* Row = new TabRecord<T>(Key, Data);

Reset();

Reset();

while (!IsTabEnded() && Row->Key >= Rows[CurrIndex]->Key)

{

if (Rows[CurrIndex]->Key == Row->Key)

{

string s = "Key: " + Key + " - isn`t unique";

throw s;

}

CurrIndex++;

}

if (IsEmpty())

CurrIndex++;

CurrRecords++;

for (int i = CurrRecords-1; i > CurrIndex; i--)

{

Rows[i] = Rows[i - 1];

}

Rows[CurrIndex] = Row;

Reset();

}

//............................................................................

template <typename T>

T\* OrdTable<T>::Search(const string Key)

{

Reset();

T\* tmp = nullptr;

if (!IsEmpty())

{

int i = -1, j = CurrRecords;

int mid;

while (i < j -1 ) //

{

mid = (j + i) / 2;

if (Key >= Rows[mid]->Key)

{

i = mid;

}

else

{

j = mid;

}

}

if (Key == Rows[i]->Key)

{

CurrIndex = i;

tmp = &(Rows[CurrIndex]->Data);

}

else

{

string s = Key + " Not Found";

throw s;

}

}

else

throw (string)"Table Is Empty";

return tmp;

}

//............................................................................

template <typename T>

void OrdTable<T>::Delete(const string Key)

{

Reset();

if (IsEmpty())

throw (string)"Cant Delete From Empty Table";

Search(Key);

if (CurrRecords > 1)

{

CurrRecords--;

for (int i = CurrIndex; i < CurrRecords; i++)

{

Rows[i] = Rows[i + 1];

}

Reset();

}

else

CurrRecords = 0;

}

//............................................................................

template <typename T>

std::ostream& operator<< (std::ostream& os, const OrdTable<T>& Tab)

{

unsigned int i = 0;

while (i < Tab.CurrRecords)

{

os << i << ". " << Tab.Rows[i]->Key << " | " << Tab.Rows[i]->Data << std::endl;

i++;

}

if (Tab.CurrRecords == 0)

os << "Table is Empty";

return os;

}

## Приложение 4. Заголовочный файл hash\_table.h

#pragma once

#include "table.h"

template <typename T>

class HashTable : public Table<T>

{

protected:

int\* flag;

int HashFunc(string Key) const;

void Realloc();

public:

// Конструкторы и деструктор

HashTable(unsigned int i = 10);

HashTable(const HashTable<T>& TabToCopy);

~HashTable() { delete[] flag;};

//Общие Методы

void Insert(const T Data, const string Key);

T\* Search(const string Key);

void Delete(const string Key);

//Методы навигации

void SetNext();

T\* GetCurr() const;

void Reset();

bool IsTabEnded() const { return false; };

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

};

//............................................................................

template <typename T>

HashTable<T>::HashTable(unsigned int i) : Table(i)

{

flag = new int[i];

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

flag[j] = 0; // 0 пустые ячейки, 1 -занято, -1 - удалено

}

//............................................................................

template <typename T>

HashTable<T>::HashTable(const HashTable<T>& TabToCopy)

{

MaxRecords = TabToCopy.MaxRecords;

CurrIndex = TabToCopy.CurrIndex;

CurrRecords = TabToCopy.CurrRecords;

delete[] Rows;

delete[] flag;

Rows = new TabRecord<T>\*[MaxRecords];

flag = new int[MaxRecords];

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

flag[j] = TabToCopy.flag[j];

if (!IsEmpty())

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

{

if(flag[i] == 1)

Rows[i] = new TabRecord<T>(\*(TabToCopy.Rows[i]));

}

}

//............................................................................

template <typename T>

void HashTable<T>::Reset()

{

if (!IsEmpty())

{

CurrIndex = 0;

while (flag[CurrIndex] != 1)

CurrIndex++;

}

else

CurrIndex = -1;

}

//............................................................................

template <typename T>

T\* HashTable<T>::GetCurr() const

{

T\* tmp = nullptr;

if(!IsEmpty())

{

tmp = &(Rows[CurrIndex]->Data);

}

else

{

throw (string)"Table Is Empty";

}

return tmp;

}

//............................................................................

template <typename T>

void HashTable<T>::SetNext()

{

if (!IsEmpty())

{

CurrIndex++;

while (flag[CurrIndex] != 1)

CurrIndex = (CurrIndex+1) % MaxRecords;

}

else

throw (string)"Table Is Empty";

}

//............................................................................

template <typename T>

void HashTable<T>::Realloc()

{

unsigned int OldMaxRecords = MaxRecords;

MaxRecords \*= 1.5;

TabRecord<T>\*\* tmp = new TabRecord<T>\*[OldMaxRecords];

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

{

tmp[i] = Rows[i];

}

delete[] Rows;

CurrRecords = 0;

CurrIndex = -1;

delete[] flag;

flag = new int[MaxRecords];

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

flag[i] = 0;

Rows = new TabRecord<T>\*[MaxRecords];

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

{

Insert(tmp[i]->Data, tmp[i]->Key); // ?

}

}

//............................................................................

template<typename T>

int HashTable<T>::HashFunc(string Key) const

{

unsigned int i= 0;

for (int j = 0; j < Key.length(); j++)

i += (char)Key[j];

for (int j = 0; j < Key.length(); j++)

i \*= 13;

srand(i);

unsigned int h = rand();

return h % MaxRecords;

}

//............................................................................

template <typename T>

void HashTable<T>::Insert(const T Data, const string Key)

{

if (IsFull())

Realloc();

TabRecord<T>\* Row = new TabRecord<T>(Key, Data);

Reset();

CurrIndex = HashFunc(Key);

if (flag[CurrIndex] == 0 || flag[CurrIndex] == -1 )

{

Rows[CurrIndex] = Row;

CurrRecords++;

flag[CurrIndex] = 1;

Reset();

}

else

{

if (Rows[CurrIndex]->Key != Key)

{

int ind = CurrIndex;

while (flag[CurrIndex] == 1 && CurrIndex+1 != ind)

CurrIndex = (CurrIndex+1) % MaxRecords;

Rows[CurrIndex] = Row;

CurrRecords++;

flag[CurrIndex] = 1;

Reset();

}

else

{

Reset();

string s = "Key: " + Key + " - isn`t unique";

throw s;

}

}

}

//............................................................................

template <typename T>

T\* HashTable<T>::Search(const string Key)

{

Reset();

T\* tmp = nullptr;

if (IsEmpty())

throw (string)"Cant Search In Empty Table";

CurrIndex = HashFunc(Key);

if(Rows[CurrIndex]->Key == Key)

tmp = &(Rows[CurrIndex]->Data);

else

{

int ind = CurrIndex;

while (flag[CurrIndex] == 1 && CurrIndex+1 != ind)

{

CurrIndex = (CurrIndex + 1) % MaxRecords;

if (Rows[CurrIndex]->Key == Key)

{

tmp = &(Rows[CurrIndex]->Data);

break;

}

}

if (tmp == nullptr)

{

Reset();

string s = Key + " Not Found";

throw s;

}

}

return tmp;

}

//............................................................................

template <typename T>

void HashTable<T>::Delete(string Key)

{

Reset();

if (IsEmpty())

throw (string)"Cant Delete From Empty Table";

Search(Key);

TabRecord<T> A;

flag[CurrIndex] = -1;

Rows[CurrIndex] = new TabRecord<T>(A);

CurrRecords--;

Reset();

}

//............................................................................

//спросить почему так

template <typename T>

std::ostream& operator<< (std::ostream& os, const HashTable<T>& Tab)

{

unsigned int i = 0;

while (i < Tab.MaxRecords)

{

if (Tab.flag[i] == 1)

{

os << i << ". " << Tab.Rows[i]->Key << " | " << Tab.Rows[i]->Data << std::endl;

}

i++;

}

if (Tab.CurrRecords == 0)

os << "Table is Empty";

return os;

}

## Приложение 5. Файл sample.cpp

#include <conio.h>

#include "unord\_table.h"

#include "ord\_table.h"

#include "hash\_table.h"

#include "polinom.h"

int main()

{

system("color F0");

UnordTable<Polinom> Utab;

OrdTable<Polinom> Otab;

HashTable<Polinom> Htab;

string Key,Str;

Polinom P;

char ch = 0;

while (ch != 27)

{

system("cls");

cout << "UnordTable: " << endl << Utab << endl;

cout << "OrdTable :" << endl << Otab << endl;

cout << "HashTable :" << endl << Htab << endl << endl;

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

cout << "What will you do?" << endl << "1. Insert" << endl << "2. Delete" <<endl << "3. Search" << endl;

cin >> Str;

int i = 0;

if (isdigit(Str[0]))

i = stoi(Str);

switch (i)

{

case 1:

{

while(true)

{

cout << "Input Polinom" << endl;

cin >> Key;

if (Key == "end") break;

P = Polinom(Key);

try

{

system("cls");

Utab.Insert(P, Key);

Otab.Insert(P, Key);

Htab.Insert(P, Key);

cout << "UnordTable: " << endl << Utab << endl;

cout << "OrdTable :" << endl << Otab << endl;

cout << "HashTable :" << endl << Htab << endl << endl;

}

catch (string s)

{

cout << "UnordTable: " << endl << Utab << endl;

cout << "OrdTable :" << endl << Otab << endl;

cout << "HashTable :" << endl << Htab << endl << endl;

cout << "Error :" << s << endl;

}

}

}

break;

case 2:

{

while (true)

{

cout << "Input Polinom" << endl;

cin >> Key;

if (Key == "end") break;

try

{

system("cls");

Utab.Delete(Key);

Otab.Delete(Key);

Htab.Delete(Key);

cout << "UnordTable: " << endl << Utab << endl;

cout << "OrdTable :" << endl << Otab << endl;

cout << "HashTable :" << endl << Htab << endl << endl;

}

catch (string s)

{

cout << "UnordTable: " << endl << Utab << endl;

cout << "OrdTable :" << endl << Otab << endl;

cout << "HashTable :" << endl << Htab << endl << endl;

cout << "Error :" << s << endl;

}

}

}

break;

case 3:

{

while (true)

{

cout << "Input Polinom" << endl;

cin >> Key;

if (Key == "end") break;

system("cls");

cout << "UnordTable: " << endl << Utab << endl;

cout << "OrdTable :" << endl << Otab << endl;

cout << "HashTable :" << endl << Htab << endl << endl;

try

{

cout << "Search in UnordTable : " << \*(Utab.Search(Key)) << endl;

cout << "Search in OrdTable : " << \*(Otab.Search(Key)) << endl;

cout << "Search in HashTable : " << \*(Htab.Search(Key)) << endl;

}

catch (string s)

{

cout << "Error :" << s << endl;

}

}

}

break;

default:

*cout* << "try again" << endl;

break;

}

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

cout << "Esc to exit " << endl;

ch = \_getch();

}

return 0;

}