МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

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

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

Направление подготовки «Прикладная математика и информатика»

**ОТЧЕТ**

по учебной практике

**Организация доступа к данным по ключу**

**Выполнил:** студент группы 381603-1

Репин В.И.

**Проверила:** к.т.н.,доцент каф. МОСТ института ИТММ

Кустикова В.Д.

Нижний Новгород  
2018

**Содержание**

[Введение 3](#_Toc514846624)

[1 Постановка задачи 4](#_Toc514846625)

[2 Руководство пользователя 5](#_Toc514846626)

[3 Руководство программиста 7](#_Toc514846627)

[3.1 Описание структуры программы 7](#_Toc514846628)

[3.2 Описание структуры данных 8](#_Toc514846629)

[3.3 Описание программной реализации 9](#_Toc514846630)

[Заключение 13](#_Toc514846631)

[Список литературы 14](#_Toc514846632)

[Приложение 15](#_Toc514846633)

[Приложение А. Программная реализация таблиц 15](#_Toc514846634)

[Приложение Б. Программная реализация приложения 22](#_Toc514846635)

# Введение

Данные в компьютере можно представлять множеством различных способов, однако в каждом отдельном случае есть более или менее удобные и надёжные варианты. Одним из самых простых, разумеется, является массив, однако с точки зрения человека работа с массивом не всегда удобна. Предлагается к рассмотрению модифицированный вариант – ассоциативные массивы, где в качестве элемента хранится пара ключ-данные, а доступ к данным осуществляется не по порядковому индексу, а по имени – ключу. Внутренняя организация такой структуры данных может быть разной, в зависимости от целей, так например: неупорядоченные по ключу таблицы, упорядоченные по ключу таблицы, таблицы с вычислимым входом (хэш-таблицы) имеют разные характеристики. Задача состоит в том, чтобы оценить и сравнить эти способы, понять их преимущества и недостатки.

В отчёте приводится постановка задачи, разработанные алгоритмы и структуры данных, описание программы и правила её использования, а также прилагается код программы, решающей поставленную задачу.

Поскольку использованные в консольном приложении классы polynom и list были подробно описаны и рассмотрены в отчёте “Арифметические операции с полиномами”, документация по ним в данном отчёте не приводится.

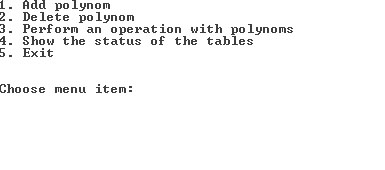
# Постановка задачи

Разработать программу, позволяющую хранить данные в трёх различных видах таблиц: упорядоченных, неупорядоченных, хэш-таблицах. Кроме того, необходимо разработать пользовательское консольное приложение, где в качестве данных выступают полиномы, действия над которыми и хранение происходят синхронно в трёх видах таблиц. Работоспособность программы необходимо проверить с помощью Google Tests Framework.

# Руководство пользователя

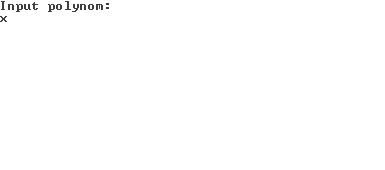
Для работы с программой необходимо запустить на исполнение файл ui2.exe.

После запуска на экране появится главное меню программы. (рис. 1)



1. Главное меню программы

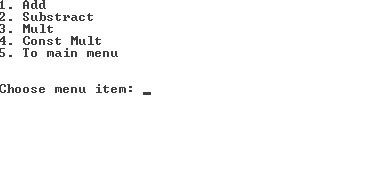
Пункт 1 позволяет добавить полином по имени в каждую из таблиц (рис. 2)



1. Вставка полинома в таблицы

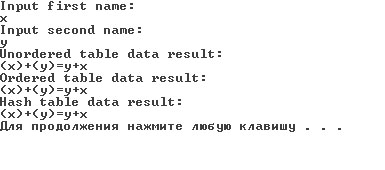
Пункт 2 позволяет удалить полином по имени из таблиц, аналогично вставке.

Пункт 3 – меню где можно проводить операции над полиномами.(рис 3).



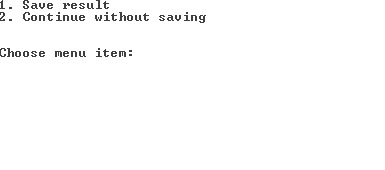
1. Меню работы с полиномами

Пункты 1–4 позволяют выполнять соответственно сложение, вычитание, умножение, и умножение на константу полиномов (рис 4).



1. Результат сложения полиномов с доступом по имени.

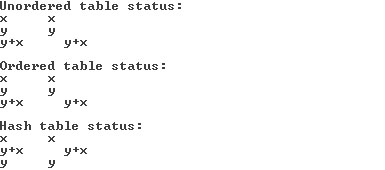
После выполнения операции пользователю будет предложено сохранить результат в таблицу. (рис. 5)



1. Меню сохранения результата

Для выхода в главное меню программы необходимо выбрать 5.

Для просмотра состояния таблиц необходимо выбрать пункт 4. (рис 6)



1. Просмотр состояния таблиц

Для выхода из программы необходимо выбрать пункт 5.

# Руководство программиста

## Описание структуры программы

Проекты и файлы, из которых состоит программа:

Tables – проект, содержащий реализацию трёх видов таблиц

table.h – заголовочный файл, содержащий объявление и реализацию абстрактного базового класса таблиц.

ord\_table.h – заголовочный файл, содержащий объявление и реализацию упорядоченных таблиц.

unord\_table.h – заголовочный файл, содержащий объявление и реализацию неупорядоченных таблиц.

hash\_table.h – заголовочный файл, содержащий объявление и реализацию хэш-таблиц.

table.cpp – файл исходного кода таблиц (пустой)

gtest – проект, содержащий Google Tests Framework:

gtest.h – заголовочный файл для Google Tests Framework.

gtest-all.cc – файл с исходным кодом Google Tests Framework.

ui2 – проект, содержащий реализацию консольного приложения:

main\_ui.cpp – файл исходного кода консольного приложения.

tests – проект, содержащий тесты для трёх классов таблиц.

test\_main.cpp – файл для запуска тестов с помощью Google Test Framework

test\_table.cpp – файл, содержащий тесты для классов unord\_table,ord\_table,hash\_table.

## Описание структуры данных

### Абстрактный класс таблиц

Поля класса:

* Массив строк таблицы
* Максимальный размер
* Текущий размер
* Текущий индекс

Описание алгоритмов:

Отсутствует.

### Класс неупорядоченных таблиц

Поля класса:

* Унаследован от абстрактного класса, других полей не имеет.

Описание алгоритмов:

*Вставка пары ключ-данные*

Вход: ключ key и данные data

Алгоритм:

1. Если текущий размер равен максимальному, то перепаковать массив;
2. Обходить массив от начала, до того момента пока либо ключ не станет равным входному, либо он не закончится;
3. Если такой ключ уже присутствует в массиве, то исключение. Иначе добавить элемент в конец, увеличить текущий размер.

Выход: нет.

*Поиск данных по ключу*

Вход: ключ key

Алгоритм:

1. Обходить массив от начала, до того момента пока либо ключ не станет равным входному, либо он не закончится;
2. Если такой ключ уже присутствует в массиве, то вернуть указатель на данные по этому ключу. Иначе – исключение.

Выход: данные по ключу или исключение, если их нет.

*Удаление данных по ключу*

Вход: ключ key

Алгоритм:

1. Обходить массив от начала, до того момента пока либо ключ не станет равным входному, либо он не закончится;
2. Если такой ключ уже присутствует в массиве, то перезаписать последний элемент на место удаляемого, уменьшить размер на единицу. Иначе – исключение.

Выход: нет или исключение, если данных нет.

### Класс упорядоченных таблиц

Поля класса:

* Унаследован от абстрактного класса, других собственных полей не имеет.

Описание алгоритмов:

*Вставка пары ключ-данные*

Вход: ключ key и данные data

Алгоритм:

1. Если текущий размер равен максимальному, то перепаковать массив;
2. Вызвать бинарный поиск для данного ключа, получить индекс ind;
3. Если такой ключ уже присутствует в массиве по индексу ind, то исключение. Иначе сдвинуть все строки начиная с индекса ind на единицу вправо, на освободившееся место записать новую строку, увеличить текущий размер.

Выход: нет.

*Поиск данных по ключу*

Вход: ключ key

Алгоритм:

1. Вызвать бинарный поиск для данного ключа, получить индекс ind;
2. Если такой ключ присутствует в массиве по индексу ind, то вернуть указатель на данные по этому ключу, иначе – исключение.

Выход: данные по ключу или исключение, если их нет.

*Удаление данных по ключу*

Вход: ключ key

Алгоритм:

1. Вызвать бинарный поиск для данного ключа, получить индекс ind;
2. Если такой ключ присутствует в массиве по индексу ind, то сдвинуть все строки начиная с индекса ind на единицу влево, затирая элемент по ind, уменьшить текущий размер на единицу иначе – исключение.

Выход: нет или исключение, если данных нет.

### Класс хэш-таблиц

Поля класса:

* Унаследован от абстрактного класса.
* Массив состояний строк таблицы.

Описание алгоритмов:

*Хэш-функция*

Вход: ключ key, флаг поиска fd

Алгоритм:

1. Сгенерировать беззнаковое целое число из ключа;
2. Проинициализировать генератор псевдослучайных чисел с помощью этого числа.
3. Если fd = true, то найти первую пустую строку, или строку с входным ключом;
4. Если fd = false, то найти первую удалённую, пустую, или с входным ключом строку;
5. Вернуть индекс найденной строки

Выход: индекс в массиве, в зависимости от флага либо место, куда можно записать новую строку, либо место, где должна быть искомая.

*Вставка пары ключ-данные*

Вход: ключ key и данные data

Алгоритм:

1. Если таблица занята более чем на 75%, то перепаковать её;
2. Вызвать хэш-функцию для входного ключа key и без флага поиска. Полученный индекс – ind;
3. Если по полученному индексу присутствует входной ключ, то исключение. Иначе записать новую строку на этот индекс, увеличить текущий размер на единицу.

Выход: нет.

*Поиск данных по ключу*

Вход: ключ key

Алгоритм:

1. Вызвать хэш-функцию для входного ключа key и с флагом поиска. Полученный индекс – ind;
2. Если по полученному индексу строка занята, то вернуть её. Иначе – исключение;

Выход: данные по ключу или исключение, если их нет.

*Удаление данных по ключу*

Вход: ключ key

Алгоритм:

1. Вызвать хэш-функцию для входного ключа key и с флагом поиска. Полученный индекс – ind;
2. Если по полученному индексу строка занята, то удалить её, уменьшить текущий размер на единицу. Иначе – исключение;

Выход: нет или исключение, если данных нет.

## Описание программной реализации

### Структура row

row – шаблонная структура, представляет собой строку таблицы ключ-данные, с параметрами KeyType – тип ключа, DataType – тип хранящихся данных.

*Поля:*

KeyType key – ключ

DataType\* data; – данные

*Методы:*

row(const KeyType& i\_key, const DataType& i\_data) – конструктор по ключу и данным

row() – конструктор пустой строки

row(const row& src) – конструктор копирования

row& operator=(const row& src) – оператор присваивания

~row() – деструктор

### **Класс table**

table – абстрактный базовый класс для таблиц.

*Поля:*

row <KeyType, DataType>\* mt – массив строк для хранения данных

int MaxSize – максимально возможное количество элементов в таблице, по умолчанию 50

int CurSize – текущее количество элементов в таблице

int Curr – индекс текущего элемента

*Методы:*

virtual void Realloc() = 0 – перепаковка таблицы, если она заполнена

table(int i\_size = MIN\_SIZE) – конструктор пустой таблицы заданного размера

virtual ~table() – деструктор

virtual void Insert(const KeyType& key, const DataType& data) = 0 – вставка в таблицу пары ключ-данные

virtual DataType\* Find(const KeyType& key) const = 0 – поиск данных по ключу

virtual void Delete(const KeyType& key) = 0 – удаление данных по ключу

virtual void Reset() – смещение на начало таблицы

virtual bool IsEnded() const – проверка, что таблица не кончилась

virtual void Move() - смещение на следующую строку в таблице

virtual row<KeyType, DataType> GetCurr() const; – получить текущую строку таблицы

virtual bool IsFull() const – проверка что таблица не заполнена

int GetCurSize() const – получить текущее количество элементов в таблице

int GetMaxSize() const – получить максимальное количество элементов в таблице

bool IsEmpty() const – проверка таблицы на пустоту

### **Класс unord\_table**

Класс упорядоченных таблиц, унаследован от базового абстрактного класса table, параметризован аналогично. Для него является дружественным класс упорядоченных таблиц, чтобы было возможно приведение неупорядоченных таблиц к упорядоченным.

*Поля:*

Не имеет.

*Методы:*

void Realloc() override – перепаковка таблицы, если она заполнена

unord\_table(int i\_size = MIN\_SIZE) – конструктор пустой таблицы заданного размера

~unord\_table() – деструктор

void Insert(const KeyType& key, const DataType& data) override – вставка в таблицу пары ключ-данные

DataType\* Find(const KeyType& key) const override – поиск данных по ключу

void Delete(const KeyType& key) override – удаление данных по ключу

template<class K, class D> friend std::ostream& operator<<(std::ostream& os, unord\_table<K, D> & tab) – оператор вставки в поток

### **Класс ord\_table**

Класс упорядоченных таблиц, унаследован от базового абстрактного класса table, параметризован аналогично.

*Поля:*

Не имеет.

*Методы:*

void Realloc() override перепаковка таблицы, если она заполнена

int binsearch(const KeyType& key) const – бинарный поиск по упорядоченной таблице, возвращает индекс позиции где есть элемент, или где он должен быть

~ord\_table()– деструктор

ord\_table(int i\_size = MIN\_SIZE) – конструктор пустой таблицы заданного размера

ord\_table(const unord\_table<KeyType, DataType>& src) – конструктор преобразования типа от unord\_table к ord\_table

void Insert(const KeyType& key, const DataType& data) override – вставка в таблицу пары ключ-данные

DataType\* Find(const KeyType& key) const override – поиск данных по ключу

void Delete(const KeyType& key) override – удаление данных по ключу

template<class K, class D> friend std::ostream& operator<<(std::ostream& os, ord\_table<K, D> & tab) – оператор вставки в поток

### **Класс hash\_table**

Класс хэш-таблиц, унаследован от базового абстрактного класса table, параметризован аналогично с добавлением ещё одного параметра – HashGen, класс, единственная функция которого – получить беззнаковое целое число из произвольного объекта типа данных KeyType. По умолчанию предоставлен, значение получено как сумма всех байт объекта, интерпретированных как целые беззнаковые числа.

*Поля:*

char\* exist – массив для реализации метода открытого перемешивания. Элементы массива могут принимать значение -1,0,1, соответственно строка удалена, строка пуста, строка заполнена.

*Методы:*

void Realloc() override – перепаковка таблицы, если она заполнена

int rhash(const KeyType& key, bool fd) const – хэш функция, если fd = false, то ищется место для вставки, если fd = true, то ищется позиция уже существующей строки.

hash\_table(int i\_size = MIN\_SIZE) – конструктор пустой таблицы заданного размера

~hash\_table() – деструктор

void Insert(const KeyType& key, const DataType& data) override – вставка в таблицу пары ключ-данные

DataType\* Find(const KeyType& key) const override – поиск данных по ключу

void Delete(const KeyType& key) override – удаление данных по ключу

bool IsEnded() const override – проверка, что таблица не кончилась

void Reset() override – смещение на начало таблицы

void Move() override - смещение на следующую строку в таблице

row<KeyType, DataType> GetCurr() const override – получить текущую строку таблицы

template<class K, class D,class H> friend std::ostream& operator<<(std::ostream& os, hash\_table<K, D, H> & tab) – оператор вставки в поток

# Заключение

В ходе учебной практики была разработана программа, полностью удовлетворяющая поставленным требованиям в смысле функциональности. Получившееся консольное приложение обладает дружественным интерфейсом, работоспособность всех его функций была протестирована с помощью Google Test Framework. Различные таблицы с точки зрения пользователя работают одинаково, однако хэш-таблицы более выгодно использовать, когда количество хранимых данных не слишком велико, а упорядоченные – когда операции вставки и удаления происходят реже, чем операция поиска. Неупорядоченные таблицы наоборот – когда удаление и вставка часты, а поиск – нет.

# Список литературы

1. Кормен Т., Лейзерсон Ч., Риверст Р., Штайн К. Алгоритмы. Построение и анализ. – М.: Издательский дом "Вильямс". – 2005. – 1290с.
2. Гергель В.П. и др. Методы программирования. Учебное пособие. – Н.Новгород.: ННГУ. – 2016. – 211с.
3. Страуструп Б. Язык программирования С++. – М.:Бином. – 2001. – 1099с.

# Приложение

## Приложение А. Программная реализация таблиц

**table.h**

#pragma once

#include <cmath>

#include <iostream>

#define MIN\_SIZE 50

#define GROWTHCOEF 1.7

template <class KeyType, class DataType>

struct row

{

KeyType key;

DataType\* data;

row(const KeyType& i\_key, const DataType& i\_data) { key = i\_key; data = new DataType(i\_data); }

row() { key = KeyType(); data = nullptr; }

row(const row& src) { key = src.key; data = new DataType(\*(src.data)); }

row& operator=(const row& src) { key = src.key; delete data; data = new DataType(\*(src.data)); return \*this; }

~row() { delete data; }

};

template<class KeyType, class DataType>

class table

{

protected:

row <KeyType, DataType>\* mt;

int MaxSize;

int CurSize;

virtual void Realloc() = 0;

int Curr;

public:

table(int i\_size = MIN\_SIZE);

virtual ~table() { delete[] mt; }

virtual void Insert(const KeyType& key, const DataType& data) = 0;

virtual DataType\* Find(const KeyType& key) const = 0;

virtual void Delete(const KeyType& key) = 0;

virtual void Reset();

virtual bool IsEnded() const;

virtual void Move();

virtual row<KeyType, DataType> GetCurr() const;

virtual bool IsFull() const {return CurSize >= MaxSize; }

int GetCurSize() const { return CurSize; }

int GetMaxSize() const { return MaxSize; }

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

};

template <class KeyType, class DataType>

table<KeyType, DataType>::table(int i\_size)

{

CurSize = 0;

MaxSize = i\_size;

mt = new row<KeyType, DataType>[MaxSize];

}

template <class KeyType, class DataType>

void table<KeyType, DataType>::Reset()

{

if (CurSize == 0)

Curr = -1;

else

Curr = 0;

}

template <class KeyType, class DataType>

bool table<KeyType, DataType>::IsEnded() const

{

return(Curr >= CurSize - 1);

}

template <class KeyType, class DataType>

void table<KeyType, DataType>::Move()

{

Curr++;

if (Curr == CurSize)

Reset();

}

template <class KeyType, class DataType>

row<KeyType, DataType> table<KeyType, DataType>::GetCurr() const

{

if (Curr < CurSize && Curr > -1)

return mt[Curr];

else throw "Element doesn't exist";

}

**unord\_table.h**

#pragma once

#include "table.h"

#include "ord\_table.h"

template<class KeyType, class DataType>

class unord\_table : public table <KeyType, DataType>

{

private:

void Realloc() override;

public:

unord\_table(int i\_size = MIN\_SIZE) : table(i\_size) {}

~unord\_table() {}

void Insert(const KeyType& key, const DataType& data) override;

DataType\* Find(const KeyType& key) const override;

void Delete(const KeyType& key) override;

template<class KeyType,class DataType> friend class ord\_table;

template<class K, class D>

friend std::ostream& operator<<(std::ostream& os, unord\_table<K, D> & tab);

};

template <class KeyType, class DataType>

std::ostream& operator<<(std::ostream &ostr, unord\_table<KeyType, DataType> & tab)

{

if (tab.GetCurSize() != 0)

{

tab.Reset();

ostr << tab.GetCurr().key << " " << \*tab.GetCurr().data << endl;

while (!tab.IsEnded())

{

tab.Move();

ostr << tab.GetCurr().key << " " << \*tab.GetCurr().data << endl;

}

}

return ostr;

}

template <class KeyType, class DataType>

void unord\_table<KeyType, DataType>::Realloc()

{

int nSize = (int)((MaxSize + 10)\*GROWTHCOEF);

row <KeyType, DataType>\* temp = new row<KeyType, DataType>[nSize];

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

temp[i] = mt[i];

delete mt;

mt = temp;

MaxSize = nSize;

}

template <class KeyType, class DataType>

void unord\_table<KeyType, DataType>::Insert(const KeyType& key, const DataType& data)

{

if (CurSize == MaxSize)

Realloc();

int i = 0;

while (mt[i].key != key && i<CurSize)

i++;

if (i == CurSize)

{

mt[CurSize] = row<KeyType, DataType>(key, data);

CurSize++;

}

else

throw "Duplicated key";

}

template <class KeyType, class DataType>

DataType\* unord\_table<KeyType, DataType>::Find(const KeyType& key) const

{

int i = 0;

while (mt[i].key != key && i<CurSize)

i++;

if (i < CurSize)

return mt[i].data;

else

throw "Data doesn't exist";

}

template <class KeyType, class DataType>

void unord\_table<KeyType, DataType>::Delete(const KeyType& key)

{

int i = 0;

while (mt[i].key != key && i<CurSize)

i++;

if (i < CurSize)

if (CurSize > 1)

mt[i] = mt[--CurSize];

else

CurSize = 0;

else

throw "Data doesn't exist";

}

**ord\_table.h**

#pragma once

#include "table.h"

#include "unord\_table.h"

template<class KeyType, class DataType>

class ord\_table : public table <KeyType, DataType>

{

private:

void Realloc() override;

int binsearch(const KeyType& key) const;

public:

~ord\_table() {}

ord\_table(int i\_size = MIN\_SIZE) : table(i\_size) {}

ord\_table(const unord\_table<KeyType, DataType>& src);

void Insert(const KeyType& key, const DataType& data) override;

DataType\* Find(const KeyType& key) const override;

void Delete(const KeyType& key) override;

template<class K, class D>

friend std::ostream& operator<<(std::ostream& os, ord\_table<K, D> & tab);

};

template <class KeyType, class DataType>

std::ostream& operator<<(std::ostream &ostr, ord\_table<KeyType, DataType> & tab)

{

if (tab.GetCurSize() != 0)

{

tab.Reset();

ostr << tab.GetCurr().key << " " << \*tab.GetCurr().data << endl;

while (!tab.IsEnded())

{

tab.Move();

ostr << tab.GetCurr().key << " " << \*tab.GetCurr().data << endl;

}

}

return ostr;

}

template <class KeyType, class DataType>

ord\_table<KeyType, DataType>::ord\_table(const unord\_table<KeyType, DataType>& src) : table(src.MaxSize)

{

for (int i = 0; i < src.CurSize; i++)

Insert(src.mt[i].key, \*(src.mt[i].data));

Reset();

}

template <class KeyType, class DataType>

void ord\_table<KeyType, DataType>::Realloc()

{

int nSize = (int)((MaxSize + 10)\*GROWTHCOEF);

row <KeyType, DataType>\* temp = new row<KeyType, DataType>[nSize];

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

temp[i] = mt[i];

delete mt;

mt = temp;

MaxSize = nSize;

}

template <class KeyType, class DataType>

int ord\_table<KeyType, DataType>::binsearch(const KeyType& key) const

{

int st = 0;

int fn = CurSize - 1;

int mid;

while (st <= fn)

{

mid = (st + fn) / 2;

if (key > mt[mid].key)

st = mid + 1;

else

fn = mid - 1;

}

return st;

}

template <class KeyType, class DataType>

void ord\_table<KeyType, DataType>::Insert(const KeyType& key, const DataType& data)

{

if (CurSize == MaxSize)

Realloc();

int i = binsearch(key);

if (mt[i].key != key)

{

for (int j = CurSize; j > i; j--)

mt[j] = mt[j - 1];

mt[i] = row<KeyType, DataType>(key, data);

CurSize++;

}

else

throw "Duplicated key";

}

template <class KeyType, class DataType>

DataType\* ord\_table<KeyType, DataType>::Find(const KeyType& key) const

{

int res = binsearch(key);

if (mt[res].key == key && res<CurSize)

return mt[res].data;

else

throw "Data doesn't exist";

}

template <class KeyType, class DataType>

void ord\_table<KeyType, DataType>::Delete(const KeyType& key)

{

int del = binsearch(key);

if (mt[del].key == key)

{

for (int i = del; i < CurSize - 1; i++)

mt[i] = mt[i + 1];

CurSize--;

}

else

throw "Data doesn't exist";

}

**hash\_table.h**

#pragma once

#include "table.h"

#include <string>

#include <random>

#define LOADFACTOR\_LIMIT 0.75

template <typename KeyType> class Hash

{

public:

static int get(const KeyType& key)

{

int res = 0;

unsigned char\* ptr = (unsigned char\*)&key;

for (int i = 0; i < sizeof(key); i++)

res += ptr[i];

return res;

}

};

template<> class Hash<std::string>

{

public:

static int get(const std::string& key)

{

int res = 0;

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

res += (unsigned char)key[i];

return res;

}

};

template<class KeyType, class DataType, class HashGen = Hash<KeyType>>

class hash\_table : public table <KeyType, DataType>

{

private:

char\* exist;

void Realloc() override;

int rhash(const KeyType& key, bool fd) const;

public:

hash\_table(int i\_size = MIN\_SIZE) : table(i\_size) {exist = new char[i\_size](); }

~hash\_table() { delete[] exist; }

void Insert(const KeyType& key, const DataType& data) override;

DataType\* Find(const KeyType& key) const override;

void Delete(const KeyType& key) override;

bool IsEnded() const override;

void Reset() override;

void Move() override;

row<KeyType, DataType> GetCurr() const override;

template<class K, class D,class H>

friend std::ostream& operator<<(std::ostream& os, hash\_table<K, D, H> & tab);

};

template <class KeyType, class DataType,class HashGen>

std::ostream& operator<<(std::ostream &ostr, hash\_table<KeyType, DataType,HashGen> & tab)

{

if (tab.GetCurSize() != 0)

{

tab.Reset();

ostr << tab.GetCurr().key << " " << \*tab.GetCurr().data << endl;

while (!tab.IsEnded())

{

tab.Move();

ostr << tab.GetCurr().key << " " << \*tab.GetCurr().data << endl;

}

}

return ostr;

}

template <class KeyType, class DataType,class HashGen>

void hash\_table<KeyType, DataType, HashGen>::Move()

{

do

{

Curr++;

} while (Curr < MaxSize && exist[Curr] != 1);

if (Curr == MaxSize)

Reset();

}

template <class KeyType, class DataType, class HashGen>

row<KeyType, DataType> hash\_table<KeyType, DataType,HashGen>::GetCurr() const

{

if (exist[Curr] == 1)

return mt[Curr];

else throw "Element doesn't exist";

}

template <class KeyType, class DataType, class HashGen>

bool hash\_table<KeyType, DataType, HashGen>::IsEnded() const

{

int t = Curr + 1;

while (t < MaxSize && exist[t] != 1)

t++;

return t == MaxSize;

}

template <class KeyType, class DataType, class HashGen>

void hash\_table<KeyType, DataType, HashGen>::Reset()

{

if (CurSize == 0)

Curr = -1;

else

{

Curr = 0;

while (exist[Curr] != 1)

Curr++;

}

}

template <class KeyType, class DataType, class HashGen>

int hash\_table<KeyType, DataType, HashGen>::rhash(const KeyType& key, bool fd) const

{

std::mt19937 gen;

int ind, val=HashGen::get(key);

gen.seed(val);

if (!fd)

do

ind = gen() % MaxSize;

while (exist[ind] == 1 && mt[ind].key != key);

else

do

ind = gen() % MaxSize;

while (exist[ind] !=0 && mt[ind].key != key);

return ind;

}

template <class KeyType, class DataType, class HashGen>

void hash\_table<KeyType, DataType, HashGen>::Insert(const KeyType& key, const DataType& data)

{

if ((double)CurSize / MaxSize > LOADFACTOR\_LIMIT)

Realloc();

int ind = rhash(key,false);

if (mt[ind].key != key)

{

mt[ind] = row<KeyType, DataType>(key, data);

exist[ind] = 1;

CurSize++;

}

else

throw "Duplicate key";

}

template <class KeyType, class DataType, class HashGen>

DataType\* hash\_table<KeyType, DataType, HashGen>::Find(const KeyType& key) const

{

int ind = rhash(key,true);

if (exist[ind] == 1)

return mt[ind].data;

else

throw "Data doesn't exist";

}

template <class KeyType, class DataType, class HashGen>

void hash\_table<KeyType, DataType, HashGen>::Delete(const KeyType& key)

{

int ind = rhash(key, true);

if (mt[ind].key == key)

{

CurSize--;

exist[ind] = -1;

}

else

throw "Data doesn't exist";

}

template <class KeyType, class DataType, class HashGen>

void hash\_table<KeyType, DataType, HashGen>::Realloc()

{

int oldSize = MaxSize;

MaxSize = (int)((MaxSize + 10)\*GROWTHCOEF);

row <KeyType, DataType>\* oldmt = mt;

char\* oldEx = exist;

exist = new char[MaxSize]();

mt = new row<KeyType, DataType>[MaxSize];

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

if (oldEx[i] == 1)

Insert(oldmt[i].key, \*(oldmt[i].data));

delete[] oldmt;

delete[] oldEx;

}

## Приложение Б. Программная реализация приложения

**main.cpp**

#include "ord\_table.h"

#include "hash\_table.h"

#include "unord\_table.h"

#include "polynom.h"

#include <iostream>

using namespace std;

char main\_menu()

{

system("cls");

char res;

cout << "1. Add polynom\n";

cout << "2. Delete polynom\n";

cout << "3. Perform an operation with polynoms\n";

cout << "4. Show the status of the tables\n";

cout << "5. Exit\n";

cout << "\n\n";

cout << "Choose menu item: ";

cin >> res;

return res;

}

char calc\_menu()

{

system("cls");

char res;

cout << "1. Add\n";

cout << "2. Substract\n";

cout << "3. Mult\n";

cout << "4. Const Mult\n";

cout << "5. To main menu\n";

cout << "\n\n";

cout << "Choose menu item: ";

cin >> res;

return res;

}

char save\_menu()

{

system("cls");

char res;

cout << "1. Save result\n";

cout << "2. Continue without saving\n";

cout << "\n\n";

cout << "Choose menu item: ";

cin >> res;

return res;

}

void main() {

unord\_table<string, polynom> ut;

ord\_table<string, polynom> ot;

hash\_table<string, polynom> ht;

polynom t;

char pmenu;

do {

pmenu = main\_menu();

switch (pmenu)

{

case '1':

{

system("cls");

string pname;

string inp;

try

{

cout << "Input polynom:\n";

cin >> inp;

t = polynom(inp);

ut.Insert(t.str() ,t);

ot.Insert(t.str(), t);

ht.Insert(t.str(), t);

}

catch (...)

{

system("cls");

cout << "Duplicated name\n";

system("pause");

}

break;

}

case '2':

{

system("cls");

string pname;

try

{

cout << "Input polynom:\n";

cin >> pname;

ut.Delete(pname);

ot.Delete(pname);

ht.Delete(pname);

}

catch (...)

{

system("cls");

cout << "There's no such polynom in tables!\n";

system("pause");

}

break;

}

case '3':

{

system("cls");

do

{

pmenu = calc\_menu();

switch (pmenu)

{

case '1':

{

string pn1, pn2;

system("cls");

try

{

cout << "Input first name:\n";

cin >> pn1;

cout << "Input second name:\n";

cin >> pn2;

cout << "Unordered table data result:\n";

cout << '(' << \*ut.Find(pn1) << ')' << '+' << '(' << \*ut.Find(pn2) << ')' << '=' << \*ut.Find(pn1) + \*ut.Find(pn2) << '\n';

cout << "Ordered table data result:\n";

cout << '(' << \*ot.Find(pn1) << ')' << '+' << '(' << \*ot.Find(pn2) << ')' << '=' << \*ot.Find(pn1) + \*ot.Find(pn2) << '\n';

cout << "Hash table data result:\n";

cout << '(' << \*ht.Find(pn1) << ')' << '+' << '(' << \*ht.Find(pn2) << ')' << '=' << \*ht.Find(pn1) + \*ht.Find(pn2) << '\n';

system("pause");

}

catch (...)

{

system("cls");

cout << "There's no such polynom in tables!\n";

system("pause");

break;

}

do

{

pmenu = save\_menu();

switch (pmenu)

{

case '1':

{

polynom res;

try

{

res = \*ot.Find(pn1) + \*ot.Find(pn2);

ut.Insert(res.str(), res);

ot.Insert(res.str(), res);

ht.Insert(res.str(), res);

}

catch (...)

{

system("cls");

cout << "Duplicatated name!\n";

system("pause");

}

break;

}

case '2': break;

}

} while (pmenu != '1' && pmenu != '2');

break;

}

case '2':

{

string pn1, pn2;

system("cls");

try

{

cout << "Input first name:\n";

cin >> pn1;

cout << "Input second name:\n";

cin >> pn2;

cout << "Unordered table data result:\n";

cout << '(' << \*ut.Find(pn1) << ')' << '-' << '(' << \*ut.Find(pn2) << ')' << '=' << \*ut.Find(pn1) - \*ut.Find(pn2) << '\n';

cout << "Ordered table data result:\n";

cout << '(' << \*ot.Find(pn1) << ')' << '-' << '(' << \*ot.Find(pn2) << ')' << '=' << \*ot.Find(pn1) - \*ot.Find(pn2) << '\n';

cout << "Hash table data result:\n";

cout << '(' << \*ht.Find(pn1) << ')' << '-' << '(' << \*ht.Find(pn2) << ')' << '=' << \*ht.Find(pn1) - \*ht.Find(pn2) << '\n';

system("pause");

}

catch (...)

{

system("cls");

cout << "There's no such polynom in tables!\n";

system("pause");

break;

}

do

{

pmenu = save\_menu();

switch (pmenu)

{

case '1':

{

polynom res;

try

{

res = \*ot.Find(pn1) - \*ot.Find(pn2);

ut.Insert(res.str(), res);

ot.Insert(res.str(), res);

ht.Insert(res.str(), res);

}

catch (...)

{

system("cls");

cout << "Duplicatated name!\n";

system("pause");

}

break;

}

case '2': break;

}

} while (pmenu != '1' && pmenu != '2');

break;

}

case '3':

{

string pn1, pn2;

system("cls");

try

{

cout << "Input first name:\n";

cin >> pn1;

cout << "Input second name:\n";

cin >> pn2;

cout << "Unordered table data result:\n";

cout << '(' << \*ut.Find(pn1) << ')' << '\*' << '(' << \*ut.Find(pn2) << ')' << '=' << \*ut.Find(pn1) \* \*ut.Find(pn2) << '\n';

cout << "Ordered table data result:\n";

cout << '(' << \*ot.Find(pn1) << ')' << '\*' << '(' << \*ot.Find(pn2) << ')' << '=' << \*ot.Find(pn1) \* \*ot.Find(pn2) << '\n';

cout << "Hash table data result:\n";

cout << '(' << \*ht.Find(pn1) << ')' << '\*' << '(' << \*ht.Find(pn2) << ')' << '=' << \*ht.Find(pn1) \* \*ht.Find(pn2) << '\n';

system("pause");

}

catch (...)

{

system("cls");

cout << "There's no such polynom in tables!\n";

system("pause");

break;

}

do

{

pmenu = save\_menu();

switch (pmenu)

{

case '1':

{

polynom res;

try

{

res = \*ot.Find(pn1) \* \*ot.Find(pn2);

ut.Insert(res.str(), res);

ot.Insert(res.str(), res);

ht.Insert(res.str(), res);

}

catch (...)

{

system("cls");

cout << "Duplicatated name!\n";

system("pause");

}

break;

}

case '2': break;

}

} while (pmenu != '1' && pmenu != '2');

break;

}

case '4':

{

string pn1;

double c;

system("cls");

try

{

cout << "Input first name:\n";

cin >> pn1;

cout << "Input constant value:\n";

cin >> c;

cout << "Unordered table data result:\n";

cout << '(' << \*ut.Find(pn1) << ')' << '\*' << '(' << c << ')' << '=' << \*ut.Find(pn1) \* c << '\n';

cout << "Ordered table data result:\n";

cout << '(' << \*ot.Find(pn1) << ')' << '\*' << '(' << c << ')' << '=' << \*ot.Find(pn1) \* c << '\n';

cout << "Hash table data result:\n";

cout << '(' << \*ht.Find(pn1) << ')' << '\*' << '(' << c << ')' << '=' << \*ht.Find(pn1) \* c << '\n';

system("pause");

}

catch (...)

{

system("cls");

cout << "There's no such polynom in tables!\n";

system("pause");

break;

}

do

{

pmenu = save\_menu();

switch (pmenu)

{

case '1':

{

polynom res;

try

{

res = \*ot.Find(pn1) \*c;

ut.Insert(res.str(), res);

ot.Insert(res.str(), res);

ht.Insert(res.str(), res);

}

catch (...)

{

system("cls");

cout << "Duplicatated name!\n";

system("pause");

}

break;

}

case '2': break;

}

} while (pmenu != '1' && pmenu != '2');

break;

}

case '5': break;

}

} while (pmenu != '5');

pmenu = '1';

break;

}

case '4':

{

system("cls");

cout << "Unordered table status:\n";

cout << ut<<endl;

cout << "Ordered table status:\n";

cout << ot << endl;

cout << "Hash table status:\n";

cout << ht << endl;

system("pause");

break;

}

case '5': break;

}

} while (pmenu != '5');

**}**