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

Логинов В.А.

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

Кустикова В.Д.

Нижний Новгород  
2018**Содержание**

[Введение 4](#_Toc515021198)

[1 Постановка задачи 5](#_Toc515021199)

[2 Руководство пользователя 6](#_Toc515021200)

[3 Руководство программиста 9](#_Toc515021201)

[3.1 Структура программы 9](#_Toc515021202)

[3.2 Структуры данных 9](#_Toc515021203)

[3.2.1 RingList 9](#_Toc515021204)

[3.2.2 TPolinom 9](#_Toc515021205)

[3.2.3 ScanTable 10](#_Toc515021206)

[3.2.4 SortedTable 10](#_Toc515021207)

[3.2.5 HashTable 10](#_Toc515021208)

[3.3 Программная реализация 11](#_Toc515021209)

[3.3.1 Структура TMonom 11](#_Toc515021210)

[3.3.2 Класс TLink 11](#_Toc515021211)

[3.3.3 Класс RingList: 12](#_Toc515021212)

[3.3.4 Класс TPolinom 12](#_Toc515021213)

[3.3.5 Абстрактный класс Table 13](#_Toc515021214)

[3.3.6 Класс ScanTable 13](#_Toc515021215)

[3.3.7 Класс SortedTable 13](#_Toc515021216)

[3.3.8 Класс HashTable 14](#_Toc515021217)

[Заключение 15](#_Toc515021218)

[Литература 16](#_Toc515021219)

[Приложение 17](#_Toc515021220)

[Приложение А. Программная реализация монома 17](#_Toc515021221)

[Приложение Б. Программная реализация звена списка 18](#_Toc515021222)

[Приложение В. Программная реализация кольцевого списка. 18](#_Toc515021223)

[Приложение В. Программная реализация полиномов 22](#_Toc515021224)

[Приложение Г. Основная программа 28](#_Toc515021225)

[Приложение Г. Программная реализация абстрактного класса таблиц 32](#_Toc515021226)

[Приложение Д. Программная реализация класса просматриваемых таблиц. 32](#_Toc515021227)

[Приложение Е. Программная реализация класса упорядоченных таблиц. 34](#_Toc515021228)

[Приложение Ж. Программная реализация класса хэш-таблиц 35](#_Toc515021229)

# Введение

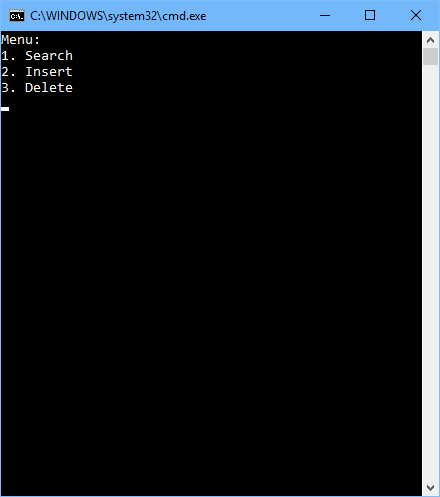
Лабораторная работа посвящена хранению различной информации в таблицах, в частности полиномов. Таблицы имеют большое практическое значение в жизни человека, так как он стремится структурировать все, что есть вокруг. Информация в виде таблицы воспринимается проще.

# Постановка задачи

Требуется разработать программу, осуществляющую хранение, поиск, вставку и удаление информации в трех таблицах: в просматриваемых (неупорядоченных), в упорядоченных и в таблицах с вычисляемым входом (хэш-таблицах). Работоспособность программы нужно проверить в google тестах, кроме того требуется разработать консольное приложение для работы с таблицами.

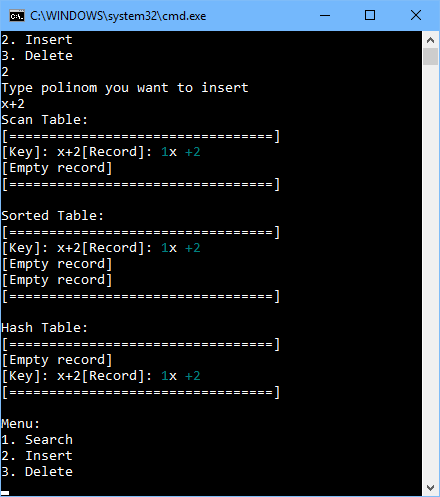
# Руководство пользователя

После запуска программы доступно следующее меню:



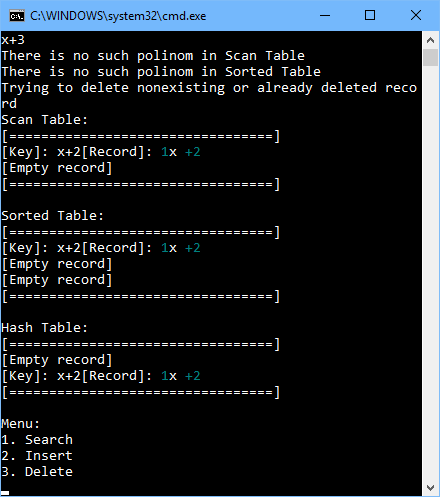
1. Главное меню программы

Для того, чтобы искать полином, добавить новый или удалить полином, нужно выбрать соответствующий пункт, при этом программа предложит Вам ввести полином. После ввода полинома программа выведет актуальное состояние всех трех таблиц:



1. Добавление нового полинома

Пункты с удалением и поиском полиномов аналогичны. Если вы попытаетесь удалить несуществующий в таблице полином, программа уведомит вас об этом:



1. Пример удаления несуществующего полинома

# Руководство программиста

## Структура программы

Программа состоит из следующих модулей: статическая библиотека Tables, где реализована структура хранения полинома и операции с ними, статической библиотеки gtest, отвечающей за тестирование программы, и двух исполняемых файлов: консольного приложения sample для работы пользователя с программой и приложения test для тестирования программы с помощью google-тестов.

Библиотека Polinom состоит из:

TMonom – реализация структуры хранения монома и операций с ним;

TLink – реализация шаблонного звена списка и операций с ним;

RingList – односвязный кольцевой (указатель с последнего звена указывает на первое звено списка) список с шаблонными звеньями;

Polinoms – реализация структуры хранения полинома и операций с ним на основе кольцевого списка со звеном типа TMonom.

Table – абстрактный класс, объединяющий в себе общие черты всех трех таблиц.

ScanTable – класс просматриваемых таблиц

SortedTable – класс упорядоченных таблиц

HashTable – класс хэш-таблиц.

Исполняемый файл sample состоит из одного файла main, где реализовано консольное приложение

gtest содержит test\_main и test\_list, где реализованы все тесты для программы.

## Структуры данных

### RingList

Описание алгоритмов:

Поиск. На вход поступает шаблонное искомое. Создается временный указатель на звено и приравнивается к указателю на первое звено списка. Пока список не кончился или искомое не было найдено, звено передвигается на 1 вперед. Если следующий за звеном указатель указывает на голову, возвращается нулевой указатель, иначе возвращается найденное звено.

Удаление. На вход поступает шаблонное удаляемое. Создается два указателя-«ходилки»: предыдущее звено и текущее. Первый приравнивается к голове, второй – к первому звену. Далее, если список не пуст, пока следующий – не первый элемент и текущий – не удаляемое, оба указателя передвигаются на один вперед. Если цикл остановился, потому что звено найдено, оно удаляется: следующий указатель с предыдущего присваивается на следующий с текущего звена, а текущее удаляется.

Вставка. На вход поступает вставляемое шаблонного типа. Если список пуст, данные вставляются в следующее за головой звено, указатель с первого звена на следующее ставится на голову. Иначе создается два указателя на звено: текущий и следующий. Аналогично поиску находится место для вставляемого элемента, указатель с предыдущего на следующее звено устанавливается на вставленное звено, указатель со вставленного на следующее устанавливается на текущее.

### TPolinom

Описание алгоритмов:

Сложение полиномов. На вход поступает второй полином. Создается результирующий пустой полином. Далее алгоритм сложения аналогичен алгоритму слияния двух отсортированных списков: Оба полинома просматриваются одновременно в цикле, пока один из них не кончился. Если степень текущего монома первого меньше, чем степень текущего монома второго, в результат вставляется текущий первого, иначе второго. В случае, если мономы одинаковые, а сумма их коэффициентов не равна нулю, вставляется моном с коэффициентом, равным сумме коэффициентов первого и второго монома. После выхода из цикла оставшиеся мономы вставляются в конец. Возвращается результат сложения

Умножение полиномов. На вход поступает второй полином. Создается результирующий пустой полином. В цикле с первого по последний мономы просматривается первый полином. В этом цикле вложен второй такой же цикл по второму полиному. В результат вставляются мономы, являющиеся результатом произведения каждого монома из первого полинома с каждым мономом из второго полинома, если только коэффициент результирующего монома не равен нулю. Возвращается результирующий полином.

### ScanTable

Описание алгоритмов:

1. Поиск. На вход поступает ключ. Все элементы таблиц просматриваются слева направо, каждый элемент сравнивается с ключом. При совпадении возвращается указатель на элемент, если же таблица была просмотрена до конца и совпадений не было найдено, возвращается нулевой указатель.
2. Удаление. На вход поступает ключ. Производится поиск элемента по алгоритму, описанному выше, при нахождении элемента в эту ячейку копируется элемент с конца таблицы, а последний элемент удаляется.
3. Вставка. На вход поступает пара ключ-данные и вставляется в конец таблицы.

### SortedTable

1. Поиск. На вход поступает ключ. Просматривается элемент из середины таблицы. Если он равен значению ключа, возвращается указатель на элемент, соответствующий этому ключу. Если элемент из середины меньше ключа, выбирается середина левой половины таблицы, если больше – правой половины, затем все действия повторяются, пока элемент не будет найден или пока от таблицы не останется один элемент.
2. Удаление. На вход поступает ключ. По ключу находится элемент, если такой элемент существует, то происходит сдвиг всех элементов таблицы справа налево на одну позицию.
3. Вставка. На вход поступает пара ключ-данные. Далее для новой пары ищется место по ключу, когда такое место было найдено, все элементы, начиная с этого места сдвигаются вправо на одну позицию.

### HashTable

1. Поиск. На вход поступает ключ. Ключ отправляется в хэш-функцию, вычисляется значение, полученное значение кладется в зерно генератора случайных чисел. Далее производится кручение генератора и вычисляется место элемента в таблице. Производится сравнение ключа, хранящегося в этом месте с искомым ключом. Если значения не совпали, генератор крутится еще раз. Если значения совпали, возвращается указатель на элемент. Если генератор был раскручен столько же раз, сколько элементов в таблице, возвращается нулевой указатель. Если в процессе последовательности кручений генератора было обнаружено, что элемент на вычисленной позиции был удален, или позиция свободна, так же возвращается нулевой указатель.
2. Вставка. На вход поступает ключ. Ключ отправляется в хэш-функцию, вычисляется значение, полученное значение кладется в зерно генератора случайных чисел. Далее производится кручение генератора и вычисляется место нового элемента в таблице. Если текущая ячейка помечена как свободная или удаленная, элемент ставится на это место, метка ячейки меняется на занятую, иначе производится новое кручение генератора случайных чисел. Если в начальный момент алгоритма вставки отношение занятых записей к общему числу записей составляло более 70 процентов, то производится перехеширование таблицы: размер таблицы увеличивается на полтора, для каждого элемента из старой таблицы производится вставка в новую таблицу.
3. Удаление. На вход поступает ключ. Производится поиск элемента по алгоритму, описанному в п. 1. Если в процессе кручения генератора был найден элемент с искомым ключом, производится его удаление, ячейка помечается как удаленная. Если же в последовательности переходов был получен элемент с меткой «удаленный» или «свободный», выводится соответствующее исключение.

## Программная реализация

### Структура TMonom

Поля:

double cf – коэффициент при мономе

unsigned int abc – обобщенная степень

Методы:

Tmonom(const double d = 0.0, const unsigned int m = 0) – конструктор с параметрами

Tmonom(const Tmonom& \_Tmonom); - конструктор копирования

const Tmonom& operator = (const Tmonom& \_Tmonom); - оператор присваивания

bool operator < (const Tmonom& \_tmonom) const; - оператор сравнения

bool operator != (const Tmonom& \_tmonom) const; - оператор сравнения

bool operator == (const Tmonom& \_tmonom) const; - оператор сравнения

bool operator > (const Tmonom& \_tmonom) const; - оператор сравнения

Tmonom operator+ (const Tmonom& \_tmonom); - оператор сложения;

### Класс TLink

Поля:

T data; - данные

TLink<T>\* pNext; - указатель на следующее звено

Методы:

TLink() {}; - конструктор по умолчанию

~TLink() {}; - деструктор

TLink(const T datum, TLink<T>\* link = nullptr) - конструктор с параметрами

TLink<T>& operator=(const TLink<T>& \_TLink); - оператор присваивания

bool operator == (const TLink<T>& link) const – оператор сравнения

### Класс RingList:

Поля:

TLink <T> \*head; - указатель на первое звено списка

TLink <T> \*current; - указатель на текущее звено списка

Методы:

TRingList(); - конструктор по умолчанию

TRingList(const TRingList <T> &); - конструктор копирования

~TRingList(); - деструктр

void Insert(const T \_data); - вставка в список данных

void InsertInEnd(const T datum); - вставка в конец

void Reset(); - перевод указателя на текущее звено на начало списка

void Clean(); - удаление всех звеньев списка, кроме первого

bool IsEnded() const; - проверка на то, что указатель на текущее звено списка указывает на последнее звено

TRingList<T>& operator=(const TRingList<T>& \_TRingList); - оператор присваивания

bool operator==(const TRingList<T>& list2) const; - оператор сравнения

bool operator<(const TRingList<T>& list2) const; - оператор сравнения

bool operator>(const TRingList<T>& list2) const; - оператор сравнения

bool operator!=(const TRingList<T>& list2) const - оператор сравнения

TLink<T>\* GetNext(); - передвинуть указатель на текущее звено на один элемент вперед и вернуть это звено

TLink<T>\* Search(const TLink<T>& d); - поиск в списке данных

void Delete(const T & d); -удаление данных

TLink<T>\* GetCurrent() – возвращение указателя на текущее звено списка

### Класс TPolinom

Поля

RingList<Tmonom> monoms; - список мономов

Методы:

TPolinom() - конструктор по умолчанию

TPolinom(const string & s); - конструктор с параметром (по строке)

TPolinom(const TPolinom &); - конструктор копирования

~TPolinom(); - деструктор

TPolinom operator+ (const TPolinom & \_polinom); - оператор сложения полиномов

TPolinom operator- (const TPolinom & \_polinom) - оператор вычитания полиномов

TPolinom& operator+= (const TPolinom & \_polinom); - оператор сложения

TPolinom operator\* (const TPolinom & \_polinom); - оператор умножения двух полиномов

TPolinom operator\* (double d); - оператор умножения на константу

TPolinom& operator = (const TPolinom& \_polinom); - оператор присваивания

bool operator == (const TPolinom& pol) const - операторы сравнения

bool operator != (const TPolinom& pol) const

bool operator < (const TPolinom& pol) const { return (\*this).monoms < pol.monoms; };

bool operator > (const TPolinom& pol) const { return (\*this).monoms > pol.monoms; };

friend ostream& operator<< (ostream &out, const TPolinom& \_polinom) – потоковый вывод

### Абстрактный класс Table

Поля

std::vector<std::pair<key, data\*> > records; - массив записей

size\_t max\_size; - максимальный размер таблицы

size\_t current\_size; - текущий размер таблицы

size\_t current; - номер текущего элемента

Методы

Table(size\_t ms) конструктор с параметром

virtual ~Table() {}; - деструктор

virtual void GoNext() = 0; - переход к следующей записи

virtual data\* GetCurrent() const = 0; - вернуть указатель на текущую запись

virtual data\* Search(key &k) = 0; - поиск по ключу

virtual void Insert(key &k, data \*d) = 0; - вставка новой пары ключ-значение

virtual void Delete(key &k) = 0; - удаление по ключу

virtual void Reset() – перевод номера текущего элемента на 0

virtual bool IsEnded() const проверка на совпадение номера текущего элемента с номером последнего

virtual bool IsEmpty() const проверка на совпадение номера текущего элемента с нулем

### Класс ScanTable

Методы:

**ScanTable() : Table<key, data>(7)** конструктор по умолчанию

**ScanTable(size\_t ms) : Table<key, data>(ms)** конструктор с параметром

**~ScanTable() {};-** деструктор

**data\* Search(key &k) override; -** поиск элемента

**void Insert(key &k, data\* d) override; -** вставка элемента

**void Delete(key& k) override;** удаление элемента

**void GoNext() override; -** перейти к следующему

**data\* GetCurrent() const override;** - вернуть указатель на текущий элемент

**size\_t GetSize() const -** получить текущий размер таблицы

**template <class key, class data>**

**friend std::ostream& operator<< (ostream &out, const ScanTable<key, data>& \_table)** – потоковый вывод

### Класс SortedTable

Методы (аналогичны полям из предыдущего класса):

**SortedTable() : Table <key, data>(7)**

**SortedTable(size\_t ms) : Table <key, data>(ms)**

**SortedTable(const ScanTable<key, data> &b);**

**~SortedTable() {};**

**data\* Search(key& k) override;**

**void Insert(key& k, data\* d) override;**

**void Delete(key& k) override;**

**void GoNext() override;**

**data\* GetCurrent() const override;**

**size\_t GetSize() const { return records.size(); }**

**template <class key, class data>**

**friend std::ostream& operator<< (ostream &out, const SortedTable<key, data>& \_table)**

### Класс HashTable

Поля:

**std::vector<short unsigned int> signs;** - массив меток (0 – свободно, 1- занято, 2 – удалено)

**void ReHash(); -** перехеширование таблицы

Методы**:**

**HashTable() : Table <std::string, data>(7)**

**HashTable(size\_t ms) : Table <std::string, data>(ms)**

**~HashTable() {};**

**unsigned int murmurHash2(std::string &dat);** - хэш-функция

**data\* Search(std::string &s) override;**

**void Insert(std::string &s, data \*d) override;**

**void Delete(std::string &s) override;**

**void GoNext() override;**

**data\* GetCurrent() const override;**

**size\_t GetSize() const { return records.size()-1; }**

**bool Check\_Overload1() const;** - проверка отношения числа занятых к общему числу записей

**bool Check\_Overload2() const;** - проверка отношения числа занятых и удаленных к общему числу записей

**template <class data>**

**friend std::ostream& operator<< (ostream &out, const HashTable<data>& \_table)**

**{**

# Заключение

Полученная программа удовлетворяет поставленным задачам. Правильность ее работы была протестирована с помощью google tests.

# Литература

Гергель В.П. и др. Методы программирования. Учебное пособие. – Н.Новгород.: ННГУ. – 2016. – 211с.

# Приложение

## Приложение А. Программная реализация монома

TMonom:

#ifndef TMONOM\_H

#define TMONOM\_H

struct Tmonom {

double cf;

unsigned int abc;

Tmonom(const double d = 0.0, const unsigned int m = 0) { cf = d; abc = m; };

Tmonom(const Tmonom& \_Tmonom);

const Tmonom& operator = (const Tmonom& \_Tmonom);

bool operator < (const Tmonom& \_tmonom) const;

bool operator != (const Tmonom& \_tmonom) const;

bool operator == (const Tmonom& \_tmonom) const;

bool operator > (const Tmonom& \_tmonom) const;

Tmonom operator+ (const Tmonom& \_tmonom);

};

#endif

TMonom.cpp:

#include "TMonom.h"

Tmonom::Tmonom(const Tmonom& \_Tmonom)

{

cf = \_Tmonom.cf;

abc = \_Tmonom.abc;

}

const Tmonom& Tmonom::operator = (const Tmonom& \_Tmonom)

{

cf = \_Tmonom.cf;

abc = \_Tmonom.abc;

return \*this;

}

bool Tmonom::operator < (const Tmonom& \_tmonom) const

{

return abc < \_tmonom.abc;

}

bool Tmonom::operator != (const Tmonom& \_tmonom) const

{

return abc != \_tmonom.abc;

}

bool Tmonom::operator == (const Tmonom& \_tmonom) const

{

return abc == \_tmonom.abc;

}

bool Tmonom::operator > (const Tmonom& \_tmonom) const

{

return abc > \_tmonom.abc;

}

Tmonom Tmonom::operator+ (const Tmonom& \_tmonom)

{

Tmonom res;

if ((\*this).abc == \_tmonom.abc)

{

res.abc = (\*this).abc;

res.cf = (\*this).cf + \_tmonom.cf;

}

else throw "unable to sum monoms!";

return res;

}

## Приложение Б. Программная реализация звена списка

TLink.h:

#ifndef TLINK\_H

#define TLINK\_H

template <typename T>

class TLink {

public:

T data;

TLink<T>\* pNext;

TLink() {};

~TLink() {};

TLink(const T datum, TLink<T>\* link = nullptr) { data = datum; pNext = link; }

TLink<T>& operator=(const TLink<T>& \_TLink)

{

data = \_TLink.data;

pNext = \_TLink.pNext;

return \*this;

};

bool operator == (const TLink<T>& link) const

{

return \*this->pNext == link.pNext;

};

};

#endif

TLink.cpp:

#include "TLink.h"

## Приложение В. Программная реализация кольцевого списка.

RingList.h:

#ifndef TRINGLIST\_H

#define TRINGLIST\_H

#include "TLink.h"

template <typename T>

class TRingList {

private:

TLink <T> \*head;

TLink <T> \*current;

public:

TRingList();

TRingList(const TRingList <T> &);

~TRingList();

void Insert(const T \_data);

void InsertInEnd(const T datum);

void Reset();

void Clean();

bool IsEnded() const; //const

TRingList<T>& operator=(const TRingList<T>& \_TRingList);

bool operator==(const TRingList<T>& list2) const;

bool operator<(const TRingList<T>& list2) const;

bool operator>(const TRingList<T>& list2) const;

bool operator!=(const TRingList<T>& list2) const { return !((\*this) == list2); }

TLink<T>\* GetNext();

TLink<T>\* Search(const TLink<T>& d);

void Delete(const T & d);

TLink<T>\* GetCurrent() { return current; }

};

template <typename T>

TRingList<T>::TRingList()

{

head = new TLink<T>();

head->pNext = head;

current = head;

};

template <typename T>

TLink<T>\* TRingList<T>::Search(const TLink<T> & d)

{

TLink<T>\* tmp = head;

while ((tmp->pNext != head) && (tmp->pNext->data != d.data))

{

tmp = tmp->pNext;

}

if (tmp->pNext == head)

{

return nullptr;

}

return tmp->pNext;

};

template<typename T>

void TRingList<T>::Delete(const T & d)

{

TLink<T>\* tmp = head;

TLink<T>\* tmp2 = head->pNext;

if (tmp != tmp2)

{

while ((tmp2->pNext != head) && (tmp2->data != d))

{

tmp = tmp2;

tmp2 = tmp2->pNext;

}

if (tmp2->data == d)

{

tmp->pNext = tmp2->pNext;

delete tmp2;

}

else throw "element does not exist in your list";

}

else throw "element does not exist in your list";

};

template <typename T>

TRingList<T>::TRingList(const TRingList <T> & Ring)

{

if (Ring.head->pNext != Ring.head)

{

head = new TLink<T> (Ring.head->data);

TLink<T> \*temp = head;

TLink<T> \*temp2 = Ring.head->pNext;

while (temp2 != Ring.head)

{

temp->pNext = new TLink<T>(temp2->data, head);

temp = temp->pNext;

temp2 = temp2->pNext;

}

}

else {

head = new TLink<T>(Ring.head->data);

current = head;

head->pNext = head;

}

};

template<typename T>

TRingList<T>::~TRingList()

{

Clean();

delete head;

};

template <typename T>

void TRingList<T>::Insert(const T datum)

{

if (head->pNext == head)

{

TLink<T>\* tmp = new TLink<T>(datum, head);

head->pNext = tmp;

}

else

{

TLink<T>\* prev = head;

TLink<T>\* tmp = head->pNext;

while ((tmp != head) && (datum < tmp->data))

{

prev = tmp;

tmp = tmp->pNext;

}

prev->pNext = new TLink<T>(datum, tmp);

}

};

template <typename T>

void TRingList<T>::InsertInEnd(const T datum)

{

TLink<T>\* tmp = head;

while (tmp->pNext != head)

tmp = tmp->pNext;

tmp->pNext = new TLink<T>(datum, head);

};

template <typename T>

void TRingList<T>::Reset()

{

current = head->pNext;

};

template<typename T>

void TRingList<T>::Clean()

{

TLink<T> \*temp = head->pNext;

TLink<T> \*temp2 = head;

while (temp != head)

{

temp2 = temp->pNext;

delete temp;

temp = temp2;

}

head->pNext = head;

};

template<typename T>

bool TRingList<T>::IsEnded() const

{

return (current == head);

};

template<typename T>

TRingList<T>& TRingList<T>::operator=(const TRingList<T> & Ring)

{

if ((\*this) != Ring)

{

Clean();

if (Ring.head->pNext != Ring.head)

{

head = new TLink<T>(Ring.head->data);

TLink<T> \*temp = head;

TLink<T> \*temp2 = Ring.head->pNext;

while (temp2 != Ring.head)

{

temp->pNext = new TLink<T>(temp2->data, head);

temp = temp->pNext;

temp2 = temp2->pNext;

}

}

else {

head = new TLink<T>(Ring.head->data);

current = head;

head->pNext = head;

}

}

return \*this;

};

template<typename T>

bool TRingList<T>::operator==(const TRingList<T> & list2) const

{

TLink<T> \*temp = head;

TLink<T> \*temp2 = list2.head;

bool f = true;

if ((temp->pNext == head) && (temp2->pNext == list2.head))

f = true;

else if (((temp->pNext != head) && (temp2->pNext == list2.head)) || ((temp2->pNext != list2.head) && (temp->pNext == head)))

f = false;

while ((temp->pNext != head) && (temp2->pNext != list2.head) && (f))

{

if (temp->data != temp2->data)

f = false;

temp = temp->pNext;

temp2 = temp2->pNext;

}

return f;

};

template<typename T>

bool TRingList<T>::operator<(const TRingList<T> & list2) const

{

TLink<T> \*temp = head;

TLink<T> \*temp2 = list2.head;

bool f = true;

if ((temp->pNext == head) && (temp2->pNext == list2.head))

f = false;

else if ((temp->pNext != head) && (temp2->pNext == list2.head))

f = false;

else if ((temp2->pNext != list2.head) && (temp->pNext == head))

f = true;

while ((temp->pNext != head) && (temp2->pNext != list2.head) && (f))

{

if ((temp->data > temp2->data) || (temp->data == temp2->data))

f = false;

temp = temp->pNext;

temp2 = temp2->pNext;

}

return f;

};

template<typename T>

bool TRingList<T>::operator>(const TRingList<T> & list2) const

{

TLink<T> \*temp = head;

TLink<T> \*temp2 = list2.head;

bool f = true;

if ((temp->pNext == head) && (temp2->pNext == list2.head))

f = true;

else if ((temp->pNext != head) && (temp2->pNext == list2.head))

f = true;

else if ((temp2->pNext != list2.head) && (temp->pNext == head))

f = false;

while ((temp->pNext != head) && (temp2->pNext != list2.head) && (f))

{

if ((temp->data < temp2->data) || (temp->data == temp2->data))

f = false;

temp = temp->pNext;

temp2 = temp2->pNext;

}

return f;

};

template <typename T>

TLink<T>\* TRingList<T>::GetNext()

{

current = current->pNext;

return current;

};

#endif

RingList.cpp:

#include "RingList.h"

## Приложение В. Программная реализация полиномов

Polinoms.h:

#ifndef TPOLINOM\_H

#define TPOLINOM\_H

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <string>

#include <Windows.h>

#include "RingList.h"

#include "TMonom.h"

using namespace std;

class TPolinom {

private:

TRingList<Tmonom> monoms;

public:

TPolinom() { }

TPolinom(const string & s);

TPolinom(const TPolinom &);

~TPolinom();

TPolinom operator+ (const TPolinom & \_polinom);

TPolinom operator- (const TPolinom & \_polinom)

{

TPolinom res = (\*this) + (-1.0) \* \_polinom;

return res;

}

TPolinom& operator+= (const TPolinom & \_polinom);

TPolinom operator\* (const TPolinom & \_polinom);

TPolinom operator\* (double d);

TPolinom& operator = (const TPolinom& \_polinom);

bool operator == (const TPolinom& pol) const { return (\*this).monoms == pol.monoms; };

bool operator != (const TPolinom& pol) const { return !((\*this) == pol); };

bool operator < (const TPolinom& pol) const { return (\*this).monoms < pol.monoms; };

bool operator > (const TPolinom& pol) const { return (\*this).monoms > pol.monoms; };

friend ostream& operator<< (ostream &out, const TPolinom& \_polinom)

{

TPolinom copy\_of\_polinom(\_polinom);

copy\_of\_polinom.monoms.Reset();

// ЧТоб было K P A C U B O;

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hStdOut, (WORD)((0 << 4) | 3));

out << copy\_of\_polinom.monoms.GetCurrent()->data.cf;

SetConsoleTextAttribute(hStdOut, (WORD)((0 << 4) | 15));

if ((copy\_of\_polinom.monoms.GetCurrent()->data.abc / 100) != 0) {

if ((copy\_of\_polinom.monoms.GetCurrent()->data.abc / 100) == 1)

out << "x";

else

out << "x^" << (copy\_of\_polinom.monoms.GetCurrent()->data.abc / 100);

}

if ((copy\_of\_polinom.monoms.GetCurrent()->data.abc / 10) % 10 != 0) {

if ((copy\_of\_polinom.monoms.GetCurrent()->data.abc / 10) % 10 == 1)

out << "y";

else out << "y^" << (copy\_of\_polinom.monoms.GetCurrent()->data.abc / 10) % 10;

}

if ((copy\_of\_polinom.monoms.GetCurrent()->data.abc) % 10 != 0) {

if ((copy\_of\_polinom.monoms.GetCurrent()->data.abc % 10) == 1)

out << "z";

else out << "z^" << (copy\_of\_polinom.monoms.GetCurrent()->data.abc) % 10;

}

TLink<Tmonom>\* cur = copy\_of\_polinom.monoms.GetNext();

for (cur; !copy\_of\_polinom.monoms.IsEnded(); copy\_of\_polinom.monoms.GetNext())

{

if (copy\_of\_polinom.monoms.GetCurrent()->data.cf > 0)

{

SetConsoleTextAttribute(hStdOut, (WORD)((0 << 4) | 3)); //system("color 0B");

out << " +" << copy\_of\_polinom.monoms.GetCurrent()->data.cf;

SetConsoleTextAttribute(hStdOut, (WORD)((0 << 4) | 15));//system("color 0F");

}

else out << " " << copy\_of\_polinom.monoms.GetCurrent()->data.cf;

if ((copy\_of\_polinom.monoms.GetCurrent()->data.abc / 100) != 0) {

if ((copy\_of\_polinom.monoms.GetCurrent()->data.abc / 100) == 1)

out << "x";

else

out << "x^" << (copy\_of\_polinom.monoms.GetCurrent()->data.abc / 100);

}

if ((copy\_of\_polinom.monoms.GetCurrent()->data.abc / 10) % 10 != 0)

{

if ((copy\_of\_polinom.monoms.GetCurrent()->data.abc / 10) % 10 == 1)

out << "y";

else out << "y^" << (copy\_of\_polinom.monoms.GetCurrent()->data.abc / 10) % 10;

}

if ((copy\_of\_polinom.monoms.GetCurrent()->data.abc) % 10 != 0) {

if ((copy\_of\_polinom.monoms.GetCurrent()->data.abc % 10) == 1)

out << "z";

else out << "z^" << (copy\_of\_polinom.monoms.GetCurrent()->data.abc) % 10;

}

}

out << endl;

return out;

}

friend TPolinom operator\* (double d, const TPolinom &p)

{

TPolinom res(p);

res.monoms.Reset();

if (d == 0)

res.monoms.Clean();

else {

for (res.monoms.GetCurrent(); !res.monoms.IsEnded(); res.monoms.GetNext())

{

res.monoms.GetCurrent()->data.cf \*= d;

}

}

return res;

};

};

#endif

Polinoms.cpp:

#include "Polinoms.h"

TPolinom::TPolinom(const std::string& s)

{

string str = s;

if (str[0] != '\0')

{

while (str.length() > 0)

{

string monom = "";

Tmonom temp;

string coeff = "";

double number = 0;

int i = 0, j = 1;

j = 1;

while (j < str.length() && str[j] != '+' && str[j] != '-')

j++;

monom = str.substr(0, j);

str.erase(0, j);

j = 0;

while (j < monom.length() && !isalpha(monom[j]))

j++;

coeff = monom.substr(0, j);

if (coeff == "" || coeff == "+")

temp.cf = 1;

else if (coeff == "-")

temp.cf = -1;

else temp.cf = atof(coeff.c\_str());

monom.erase(0, j);

j = 0;

while (j < monom.length()) // формирование степени

{

if (isalpha(monom[j]) && monom[j + 1] == '\0') // последняя буква в строке

{

switch (monom[j])

{

case 'x':

temp.abc += 1 \* 100;

break;

case 'y':

temp.abc += 1 \* 10;

break;

case 'z':

temp.abc += 1;

break;

default:

throw "Input error:end of string";

break;

}

j++;

}

else if (isalpha(monom[j]) && isalpha(monom[j + 1])) // 2 буквы подряд

{

switch (monom[j])

{

case 'x':

temp.abc += 1 \* 100;

break;

case 'y':

temp.abc += 1 \* 10;

break;

case 'z':

temp.abc += 1;

break;

default:

throw "Input error:end of string";

break;

}

j++;

}

else if (isalpha(monom[j]) && monom[j + 1] == '^' && monom[j + 2] != '\0' && isdigit(monom[j + 2]))

{

switch (monom[j])

{

case 'x':

temp.abc += atoi(&monom[j + 2]) \* 100;

j += 3;

break;

case 'y':

temp.abc += atoi(&monom[j + 2]) \* 10;

j += 3;

break;

case 'z':

temp.abc += atoi(&monom[j + 2]) \* 1;

j += 3;

break;

}

}

}

if (temp.abc != 0 && temp.cf != 0) {

TLink<Tmonom>\* link = monoms.Search(temp);

if (link != NULL)

{

if (link->data.cf + temp.cf != 0)

{

link->data.cf += temp.cf;

}

else monoms.Delete(link->data);

}

else

monoms.Insert(temp);

}

else if (temp.abc == 0 && temp.cf != 0)

{

monoms.InsertInEnd(temp);

}

}

}

}

TPolinom::TPolinom(const TPolinom & \_polinom)

{

monoms = \_polinom.monoms;

}

TPolinom::~TPolinom()

{

}

TPolinom TPolinom::operator+ (const TPolinom & \_polinom)

{

(\*this).monoms.Reset();

TPolinom copy\_of\_polinom(\_polinom);

copy\_of\_polinom.monoms.Reset();

TPolinom res;

res.monoms.Reset();

while (!copy\_of\_polinom.monoms.IsEnded() && !(\*this).monoms.IsEnded())

{

if (copy\_of\_polinom.monoms.GetCurrent()->data < (\*this).monoms.GetCurrent()->data)

{

res.monoms.InsertInEnd((\*this).monoms.GetCurrent()->data);

(\*this).monoms.GetNext();

}

else if (copy\_of\_polinom.monoms.GetCurrent()->data == (\*this).monoms.GetCurrent()->data)

{

if ((\*this).monoms.GetCurrent()->data.cf + copy\_of\_polinom.monoms.GetCurrent()->data.cf != 0.0)

{

res.monoms.InsertInEnd((\*this).monoms.GetCurrent()->data + copy\_of\_polinom.monoms.GetCurrent()->data);

}

(\*this).monoms.GetNext();

copy\_of\_polinom.monoms.GetNext();

}

else

{

res.monoms.InsertInEnd(copy\_of\_polinom.monoms.GetCurrent()->data);

copy\_of\_polinom.monoms.GetNext();

}

}

while (!copy\_of\_polinom.monoms.IsEnded())

{

res.monoms.InsertInEnd(copy\_of\_polinom.monoms.GetCurrent()->data);

copy\_of\_polinom.monoms.GetNext();

}

while (!(\*this).monoms.IsEnded())

{

res.monoms.InsertInEnd((\*this).monoms.GetCurrent()->data);

(\*this).monoms.GetNext();

}

return res;

}

TPolinom & TPolinom::operator+=(const TPolinom & \_polinom)

{

TPolinom copy\_of\_polinom(\_polinom);

copy\_of\_polinom.monoms.Reset();

for (copy\_of\_polinom.monoms.GetCurrent(); !copy\_of\_polinom.monoms.IsEnded(); copy\_of\_polinom.monoms.GetNext())

{

TLink<Tmonom>\* tmp2 = (\*this).monoms.Search(copy\_of\_polinom.monoms.GetCurrent()->data);

if (tmp2 != NULL)

{

if (tmp2->data.cf + copy\_of\_polinom.monoms.GetCurrent()->data.cf != 0)

{

tmp2->data.cf += copy\_of\_polinom.monoms.GetCurrent()->data.cf;

}

else

(\*this).monoms.Delete(tmp2->data);

}

else

(\*this).monoms.Insert(copy\_of\_polinom.monoms.GetCurrent()->data);

}

return \*this;

}

TPolinom TPolinom::operator\* (const TPolinom & \_polinom)

{

(\*this).monoms.Reset();

TPolinom copy\_of\_polinom(\_polinom);

copy\_of\_polinom.monoms.Reset();

TPolinom res;

res.monoms.Reset();

for ((\*this).monoms.GetCurrent(); !(\*this).monoms.IsEnded(); (\*this).monoms.GetNext())

{

for (copy\_of\_polinom.monoms.GetCurrent(); !copy\_of\_polinom.monoms.IsEnded(); copy\_of\_polinom.monoms.GetNext())

{

if (((\*this).monoms.GetCurrent()->data.abc + copy\_of\_polinom.monoms.GetCurrent()->data.abc < 1000) &&

(((\*this).monoms.GetCurrent()->data.abc / 10 % 10 + copy\_of\_polinom.monoms.GetCurrent()->data.abc / 10 % 10) < 10) &&

(((\*this).monoms.GetCurrent()->data.abc % 10 + copy\_of\_polinom.monoms.GetCurrent()->data.abc % 10) < 10))

{

unsigned int tmpabc = (\*this).monoms.GetCurrent()->data.abc + copy\_of\_polinom.monoms.GetCurrent()->data.abc;

Tmonom tmp((\*this).monoms.GetCurrent()->data.cf \* copy\_of\_polinom.monoms.GetCurrent()->data.cf, tmpabc);

TLink<Tmonom>\* tmp2 = res.monoms.Search(tmp);

if (tmp2 != NULL)

if ((tmp2->data.cf += (\*this).monoms.GetCurrent()->data.cf \* copy\_of\_polinom.monoms.GetCurrent()->data.cf) == 0)

{

res.monoms.Delete(tmp2->data);

}

else tmp2->data.cf += (\*this).monoms.GetCurrent()->data.cf \* copy\_of\_polinom.monoms.GetCurrent()->data.cf;

else

res.monoms.Insert(tmp);

}

else throw "Power is more then expected";

}

copy\_of\_polinom.monoms.Reset();

}

return res;

}

TPolinom TPolinom::operator\* (double d)

{

TPolinom res = (\*this);

res.monoms.Reset();

if (d == 0)

res.monoms.Clean();

else {

while (!res.monoms.IsEnded())

{

res.monoms.GetCurrent()->data.cf \*= d;

res.monoms.GetNext();

}

}

return res;

}

TPolinom& TPolinom::operator =(const TPolinom& \_polinom)

{

monoms = \_polinom.monoms;

return \*this;

}

## Приложение Г. Основная программа

main.cpp:

#include <cstdio>

#include <cstdlib>

#include <algorithm>

#include <string>

#include <iostream>

#include <vector>

#include <Windows.h>

#include "HashTable.h"

#include "ScanTable.h"

#include "SortedTable.h"

#include "Polinoms.h"

#include "TMonom.h"

int main() {

system("color 0F");

std::string s;

TPolinom zero("");

std::vector<TPolinom> Vector\_of\_Polinoms;

TPolinom temp1("-50xy^3+xyz+4zxy-5y+y+z");

Vector\_of\_Polinoms.push\_back(temp1);

TPolinom temp2("25x^2-34y^2z^6x^4");

Vector\_of\_Polinoms.push\_back(temp2);

TPolinom temp3("58x^1y^2z^3");

Vector\_of\_Polinoms.push\_back(temp3);

int menu = 1;

int nPolinoms = 1;

int a, b, c;

std::cout << "Menu:" << endl;

std::cout << "1. Add new polinom" << endl;

std::cout << "2. Delete existing polinom" << endl;

std::cout << "3. Sum two polinoms" << endl;

std::cout << "4. Multiple two polinoms" << endl;

std::cout << "5. Multiple polinom on number" << endl;

std::cout << "6. Print polinom" << endl;

std::cout << "7. Sum two polinoms (+=)" << endl;

std::cout << "8. Substract two polinoms" << endl;

std::cin >> menu;

while (menu < 9 && menu > 0)

{

switch (menu)

{

case 1:

{

std::cout << "Enter your polinom. It must be like '25x^2+34x^2y^8z^4' " << std::endl;

std::cin >> s;

TPolinom temp1(s);

Vector\_of\_Polinoms.push\_back(temp1);

nPolinoms++;

std::cout << "Menu:" << endl;

std::cout << "1. Add new polinom" << endl;

std::cout << "2. Delete existing polinom" << endl;

std::cout << "3. Sum two polinoms" << endl;

std::cout << "4. Multiple two polinoms" << endl;

std::cout << "5. Multiple polinom on number" << endl;

std::cout << "6. Print polinom" << endl;

std::cout << "7. Sum two polinoms (+=)" << endl;

std::cout << "8. Substract two polinoms" << endl;

std::cin >> menu;

break;

}

case 2:

do {

std::cout << "Enter number of the polinom you want to delete " << std::endl;

std::cin >> a;

} while (a < 0 || a > Vector\_of\_Polinoms.size());

Vector\_of\_Polinoms[a]=zero;

std::cout << "Menu:" << endl;

std::cout << "1. Add new polinom" << endl;

std::cout << "2. Delete existing polinom" << endl;

std::cout << "3. Sum two polinoms" << endl;

std::cout << "4. Multiple two polinoms" << endl;

std::cout << "5. Multiple polinom on number" << endl;

std::cout << "6. Print polinom" << endl;

std::cout << "7. Sum two polinoms (+=)" << endl;

std::cout << "8. Substract two polinoms" << endl;

std::cin >> menu;

break;

case 3:

do {

std::cout << "Enter number of the first polinom you want to sum " << std::endl;

std::cin >> a;

} while (a < 0 || a > Vector\_of\_Polinoms.size());

do {

std::cout << "Enter number of the second polinom you want to sum " << std::endl;

std::cin >> b;

} while (b < 0 || b > Vector\_of\_Polinoms.size());

do {

std::cout << "Enter number of the polinom you want to save the result. If you dont want to save, enter '-1' " << std::endl;

std::cin >> c;

} while (c < -1 || c > (int)Vector\_of\_Polinoms.size());

if (c == -1)

{

TPolinom res;

res = Vector\_of\_Polinoms[a] + Vector\_of\_Polinoms[b];

std::cout << res;

}

else if (c > -1)

Vector\_of\_Polinoms[c] = Vector\_of\_Polinoms[a] + Vector\_of\_Polinoms[b];

std::cout << "Menu:" << endl;

std::cout << "1. Add new polinom" << endl;

std::cout << "2. Delete existing polinom" << endl;

std::cout << "3. Sum two polinoms" << endl;

std::cout << "4. Multiple two polinoms" << endl;

std::cout << "5. Multiple polinom on number" << endl;

std::cout << "6. Print polinom" << endl;

std::cout << "7. Sum two polinoms (+=)" << endl;

std::cout << "8. Substract two polinoms" << endl;

std::cin >> menu;

break;

case 4:

do {

std::cout << "Enter number of the first polinom you want to multiple " << std::endl;

std::cin >> a;

} while (a < 0 || a > Vector\_of\_Polinoms.size());

do {

std::cout << "Enter number of the second polinom you want to multiple " << std::endl;

std::cin >> b;

} while (b < 0 || b > Vector\_of\_Polinoms.size());

do {

std::cout << "Enter number of the polinom you want to save the result. If you dont want to save, enter '-1' " << std::endl;

std::cin >> c;

} while (c < -1 || c > (int)Vector\_of\_Polinoms.size());

if (c == -1)

{

TPolinom res;

res = Vector\_of\_Polinoms[a] \* Vector\_of\_Polinoms[b];

std::cout << res;

}

else if (c > -1)

Vector\_of\_Polinoms[c] = Vector\_of\_Polinoms[a] \* Vector\_of\_Polinoms[b];

std::cout << "Menu:" << endl;

std::cout << "1. Add new polinom" << endl;

std::cout << "2. Delete existing polinom" << endl;

std::cout << "3. Sum two polinoms" << endl;

std::cout << "4. Multiple two polinoms" << endl;

std::cout << "5. Multiple polinom on number" << endl;

std::cout << "6. Print polinom" << endl;

std::cout << "7. Sum two polinoms (+=)" << endl;

std::cout << "8. Substract two polinoms" << endl;

std::cin >> menu;

break;

case 5:

double d;

do {

std::cout << "Enter number of the first polinom you want to multiple " << std::endl;

std::cin >> a;

} while (a < 0 || a > Vector\_of\_Polinoms.size());

std::cout << "Enter number you want your polinom to multiple" << std::endl;

std::cin >> d;

do {

std::cout << "Enter number of the polinom you want to save the result. If you dont want to save, enter '-1' " << std::endl;

std::cin >> c;

} while (c < -1 || c > (int)Vector\_of\_Polinoms.size());

if (c == -1)

{

TPolinom res;

res = Vector\_of\_Polinoms[a] \* d;

std::cout << res;

}

else if (c > -1)

Vector\_of\_Polinoms[c] = Vector\_of\_Polinoms[a] \* d;

std::cout << "Menu:" << endl;

std::cout << "1. Add new polinom" << endl;

std::cout << "2. Delete existing polinom" << endl;

std::cout << "3. Sum two polinoms" << endl;

std::cout << "4. Multiple two polinoms" << endl;

std::cout << "5. Multiple polinom on number" << endl;

std::cout << "6. Print polinom" << endl;

std::cout << "7. Sum two polinoms (+=)" << endl;

std::cout << "8. Substract two polinoms" << endl;

std::cin >> menu;

break;

case 6:

for(auto polinom : Vector\_of\_Polinoms)

std::cout << "Polinom = " << polinom;

std::cout << "Menu:" << endl;

std::cout << "1. Add new polinom" << endl;

std::cout << "2. Delete existing polinom" << endl;

std::cout << "3. Sum two polinoms" << endl;

std::cout << "4. Multiple two polinoms" << endl;

std::cout << "5. Multiple polinom on number" << endl;

std::cout << "6. Print polinom" << endl;

std::cout << "7. Sum two polinoms (+=)" << endl;

std::cout << "8. Substract two polinoms" << endl;

std::cin >> menu;

break;

case 7:

do {

std::cout << "Enter number of the first polinom you want to sum from 0 to" << Vector\_of\_Polinoms.size() << std::endl;

std::cin >> a;

} while (a < 0 || a > Vector\_of\_Polinoms.size());

do {

std::cout << "Enter number of the second polinom you want to sum from 0 to" << Vector\_of\_Polinoms.size() << std::endl;

std::cin >> b;

} while (b < 0 || b > Vector\_of\_Polinoms.size());

Vector\_of\_Polinoms[a] += Vector\_of\_Polinoms[b];

std::cout << "Menu:" << endl;

std::cout << "1. Add new polinom" << endl;

std::cout << "2. Delete existing polinom" << endl;

std::cout << "3. Sum two polinoms" << endl;

std::cout << "4. Multiple two polinoms" << endl;

std::cout << "5. Multiple polinom on number" << endl;

std::cout << "6. Print polinom" << endl;

std::cout << "7. Sum two polinoms (+=)" << endl;

std::cout << "8. Substract two polinoms" << endl;

std::cin >> menu;

break;

case 8:

do {

std::cout << "Enter number of the first polinom you want to sum " << std::endl;

std::cin >> a;

} while (a < 0 || a > Vector\_of\_Polinoms.size());

do {

std::cout << "Enter number of the second polinom you want to sum " << std::endl;

std::cin >> b;

} while (b < 0 || b > Vector\_of\_Polinoms.size());

do {

std::cout << "Enter number of the polinom you want to save the result. If you dont want to save, enter '-1' " << std::endl;

std::cin >> c;

} while (c < -1 || c >(int)Vector\_of\_Polinoms.size());

if (c == -1)

{

TPolinom res;

res = Vector\_of\_Polinoms[a] - Vector\_of\_Polinoms[b];

std::cout << res;

}

else if (c > -1)

Vector\_of\_Polinoms[c] = Vector\_of\_Polinoms[a] - Vector\_of\_Polinoms[b];

std::cout << "Menu:" << endl;

std::cout << "1. Add new polinom" << endl;

std::cout << "2. Delete existing polinom" << endl;

std::cout << "3. Sum two polinoms" << endl;

std::cout << "4. Multiple two polinoms" << endl;

std::cout << "5. Multiple polinom on number" << endl;

std::cout << "6. Print polinom" << endl;

std::cout << "7. Sum two polinoms (+=)" << endl;

std::cout << "8. Substract two polinoms" << endl;

std::cin >> menu;

break;

default:exit(1);

}

}

return 0;

}

### Приложение Г. Программная реализация абстрактного класса таблиц

Table.h:

#ifndef TABLE\_H

#define TABLE\_H

#include <vector>

template <class key, class data>

class Table

{

protected:

std::vector<std::pair<key, data\*> > records;

size\_t max\_size;

size\_t current\_size;

size\_t current;

public:

Table(size\_t ms) { max\_size = ms; current = current\_size = 0; };

virtual ~Table() {};

virtual void GoNext() = 0;

virtual data\* GetCurrent() const = 0;

virtual data\* Search(key &k) = 0;

virtual void Insert(key &k, data \*d) = 0;

virtual void Delete(key &k) = 0;

virtual void Reset() { current = 0; }

virtual bool IsEnded() const { return current == current\_size; }

virtual bool IsEmpty() const { return current == 0; }

};

#endif

### Приложение Д. Программная реализация класса просматриваемых таблиц.

ScanTable.h:

#pragma once

#include <cstdlib>

#include <cstdio>

#include <vector>

#include "Tables.h"

#include "Polinoms.h"

template <class key, class data>

class ScanTable : public Table <key, data>

{

public:

ScanTable() : Table<key, data>(7) { max\_size = 7; records.resize(7); current = 0; }

ScanTable(size\_t ms) : Table<key, data>(ms) { max\_size = ms; records.resize(ms); current = 0; }

~ScanTable() {};

data\* Search(key &k) override;

void Insert(key &k, data\* d) override;

void Delete(key& k) override;

void GoNext() override;;

data\* GetCurrent() const override;

size\_t GetSize() const { return records.size(); }

template <class key, class data>

friend std::ostream& operator<< (ostream &out, const ScanTable<key, data>& \_table)

{

out << "[=================================]" << std::endl;

for (auto rec : \_table.records)

{

if (rec.second == nullptr)

out << "[Empty record]" << endl;

else

out <<"[Key]: " << rec.first << "[Record]: " << \*(rec.second);

}

out << "[=================================]" << std::endl;

return out;

}

};

template<class key, class data>

data \* ScanTable<key, data>::Search(key& k)

{

Reset();

while ((current < (records.size()-1)) && (records[current].first != k))

GoNext();

if (records[current].first == k)

return records[current].second;

else

return nullptr;

}

template<class key, class data>

void ScanTable<key, data>:: Insert(key &k, data\* d)

{

if (current\_size < max\_size)

{

records[current++] = std::make\_pair(k, d);

current\_size++;

}

else

{

max\_size \*= 1.5;

records.resize(max\_size);

records[current++] = std::make\_pair(k, d);

current\_size++;

}

}

template<class key, class data>

void ScanTable<key, data>::Delete(key& k)

{

Search(k);

if (records[current].first == k)

{

records[current] = records[records.size()-1];

records.pop\_back();

}

}

template<class key, class data>

void ScanTable<key, data>::GoNext()

{

++current;

}

template<class key, class data>

data\* ScanTable<key, data>::GetCurrent() const

{

return records[current].second;

}

### Приложение Е. Программная реализация класса упорядоченных таблиц.

SortedTable.h:

#pragma once

#include <vector>

#include <cstdlib>

#include <iostream>

#include "Polinoms.h"

#include "Tables.h"

#include "ScanTable.h"

template <class key, class data>

class SortedTable : public Table <key, data>

{

public:

SortedTable() : Table <key, data>(7) { max\_size = 7; records.resize(7); };

SortedTable(size\_t ms) : Table <key, data>(ms) { max\_size = ms; records.resize(ms); };

SortedTable(const ScanTable<key, data> &b);

~SortedTable() {};

data\* Search(key& k) override;

void Insert(key& k, data\* d) override;

void Delete(key& k) override;

void GoNext() override;

data\* GetCurrent() const override;

size\_t GetSize() const { return records.size(); }

template <class key, class data>

friend std::ostream& operator<< (ostream &out, const SortedTable<key, data>& \_table)

{

out << "[=================================]" << std::endl;

for (auto rec : \_table.records)

{

if (rec.second == nullptr)

out << "[Empty record]" << endl;

else

out << "[Key]: " << rec.first << "[Record]: " << \*(rec.second);

}

out << "[=================================]" << std::endl;

return out;

}

};

template<class key, class data>

SortedTable<key, data>::SortedTable(const ScanTable<key, data> &b)

{

size\_t ms = b.GetSize();

records.resize(ms);

signs.resize(ms);

b.Reset();

for (b.GetCurrent(); !b.IsEnded(); b.GoNext())

{

(\*this).Insert(b.GetCurrent().first, b.GetCurrent().second);

}

current = 0;

}

template <class key, class data>

data \* SortedTable<key, data>::Search(key& k)

{

size\_t j = GetSize()-1, i = 0;

size\_t mid = j - i;

while (mid > 0)

{

mid = (j - i) >> 1;

if (records[mid].first == k)

{

current = mid;

return records[current].second;

}

else if (records[mid].first < k)

i += mid;

else j -= mid+1;

}

return nullptr;

}

template <class key, class data>

void SortedTable<key, data>::Insert(key& k, data\* d)

{

Search(k);

records.insert(records.begin() + current, std::make\_pair(k, d));

}

template <class key, class data>

void SortedTable<key, data>::Delete(key &k)

{

Search(k);

records.erase(records.begin() + current);

}

template <class key, class data>

void SortedTable<key, data>::GoNext()

{

++current;

}

template <class key, class data>

data\* SortedTable<key, data>::GetCurrent() const

{

return records[current].second;

}

### Приложение Ж. Программная реализация класса хэш-таблиц

HashTable.h:

#pragma once

#include <vector>

#include <string>

#include "Tables.h"

#include "Polinoms.h"

template <class data>

class HashTable : public Table<std::string, data>

{

protected:

std::vector<short unsigned int> signs;

void ReHash();

public:

HashTable() : Table <std::string, data>(7) { max\_size = 7; records.resize(7); signs.resize(7); }

HashTable(size\_t ms) : Table <std::string, data>(ms) { max\_size = current\_size = ms; records.resize(ms); signs.resize(ms); }

~HashTable() {};

unsigned int murmurHash2(std::string &dat);

data\* Search(std::string &s) override;

void Insert(std::string &s, data \*d) override;

void Delete(std::string &s) override;

void GoNext() override;

data\* GetCurrent() const override;

size\_t GetSize() const { return records.size()-1; }

bool Check\_Overload1() const;

bool Check\_Overload2() const;

template <class data>

friend std::ostream& operator<< (ostream &out, const HashTable<data>& \_table)

{

out << "[=================================]" << std::endl;

for (auto rec : \_table.records)

{

if (rec.second == nullptr)

out << "[Empty record]" << endl;

else

out << "[Key]: " << rec.first << "[Record]: " << \*(rec.second);

}

out << "[=================================]" << std::endl;

return out;

}

};

template <class data>

unsigned int HashTable<data>::murmurHash2(std::string &name)

{

std::string st = name;

unsigned int len = st.length();

const unsigned int m = 0x5bd1e995;

const unsigned int seed = 0;

const int r = 24;

unsigned int h = seed ^ len;

unsigned int k;

while (len >= 4)

{

k = st[0];

k |= st[1] << 8;

k |= st[2] << 16;

k |= st[3] << 24;

k \*= m;

k ^= k >> r;

k \*= m;

h \*= m;

h ^= k;

st += 4;

len -= 4;

}

switch (len)

{

case 3:

h ^= st[2] << 16;

case 2:

h ^= st[1] << 8;

case 1:

h ^= st[0];

h \*= m;

};

h ^= h >> 13;

h \*= m;

h ^= h >> 15;

return h;

}

template <class data>

data \* HashTable<data>::Search(std::string &name)

{

srand(murmurHash2(name));

size\_t i =0;

current = rand() % records.size();

while ((signs[current] == 1) && (i < (records.size()-1))) {

if (records[current].first == name)

return records[current].second;

else

{

current = rand() % records.size();

i++;

}

}

if ((signs[i] == 0) || (signs[i] == 2) || i >= (records.size()-1))

return nullptr;

}

template <class data>

void HashTable<data>::Insert(std::string &name, data \*d)

{

if (Check\_Overload1())

ReHash();

auto res = std::make\_pair(name, d);

srand(murmurHash2(name));

current = rand() % records.size();

while (signs[current] == 1)

current = rand() % records.size();

if ((signs[current] == 0) || (signs[current] == 2))

{

records[current] = res;

signs[current] = 1;

}

}

template <class data>

void HashTable<data>::Delete(std::string &name)

{

srand(murmurHash2(name));

size\_t i = 0;

current = rand() % records.size();

bool del = false;

while ((signs[current] == 1) && (i < records.size() - 1)) {

if (records[current].first == name)

{

signs[current] = 2;

records[current].first = "";

records[current].second = nullptr;

del = true;

}

else

{

current = rand() % records.size();

i++;

}

}

if (signs[current] == 2 && !del)

throw "Record is deleted";

if (signs[current] == 0 || (i>=records.size()-1))

throw "Record is empty";

}

template <class data>

void HashTable<data>::ReHash()

{

current\_size \*= 1.5;

auto records\_new(records);

auto signs\_new(signs);

/\*records\_new.resize(current\_size);

signs\_new.resize(current\_size);\*/

records.clear();

signs.clear();

records.resize(current\_size);

signs.resize(current\_size);

for (auto rec : records\_new)

{

srand(murmurHash2(rec.first));

current = rand() % current\_size;

while (signs[current] == 1)

current = rand() % records\_new.size();

records[current] = rec;

signs[current] = 1;

}

}

template <class data>

void HashTable<data>::GoNext()

{

++current;

}

template <class data>

data\* HashTable<data>::GetCurrent() const

{

return records[current].second;

}

template <class data>

bool HashTable<data>::Check\_Overload1() const

{

unsigned short int Rec\_b = 0, Rec\_e = (\*this).signs.size();

for (auto a : signs)

{

if (a == 1)

Rec\_b++;

}

return Rec\_b / Rec\_e >= 0.7;

}

template <class data>

bool HashTable<data>::Check\_Overload2() const

{

unsigned short int Rec\_bd = 0, Rec\_e = (\*this).signs.size();

for (auto a : signs)

{

if ((a == 1) || (a == 2))

Rec\_bd++;

}

return Rec\_bd / Rec\_e >= 0.7;

};