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

**Способы организации таблиц**

**Выполнила:** студентка группы 381603-1

Лакшина А.Р.

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

Кустикова В.Д.

Нижний Новгород

2018

**Содержание**

[Введение 3](#_Toc515595919)

[1 Постановка задачи 4](#_Toc515595920)

[2 Руководство пользователя 5](#_Toc515595921)

[3 Руководство программиста 10](#_Toc515595922)

[3.1 Структура программы 10](#_Toc515595923)

[3.2 Структуры данных 11](#_Toc515595924)

[3.3 Программная реализация 14](#_Toc515595925)

[Заключение 19](#_Toc515595926)

[Литература 20](#_Toc515595927)

[Приложение 21](#_Toc515595928)

[Приложение А. Программная реализация линейного списка 21](#_Toc515595929)

[Приложение Б. Программная реализация таблиц 26](#_Toc515595930)

[Приложение Б. Программная реализация приложения 33](#_Toc515595931)

# Введение

Существует множество способов хранения и представления данных на ЭВМ. Один из них — в виде ассоциативных массивов. Ассоциативный массив — абстрактный тип данных, позволяющий хранить пары вида «(ключ, значение)» и поддерживающий операции добавления пары, а также поиска и удаления пары по ключу.

Ассоциативный массив можно организовать различными способами. В данной работе рассмотрено три из них: упорядоченные и неупорядоченные по ключу таблицы и хэш-таблицы. В качестве «значения» используются полиномы, реализация которых была описана в предыдущем отчёте.

В данном отчёте приводится постановка задачи, реализованные алгоритмы и структуры данных, их программная реализация и код программы, решающий поставленную задачу.

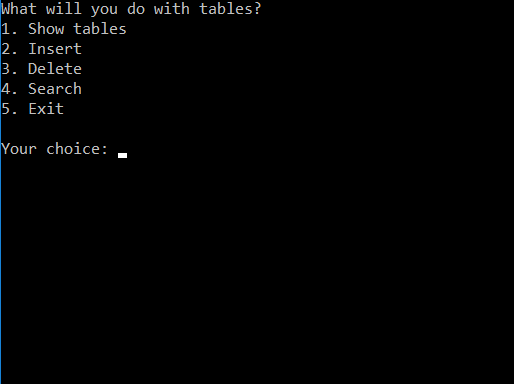
# Постановка задачи

Требуется разработать программу, позволяющую хранить данные в таблицах трех видов: неупорядоченные, упорядоченные и хэш-таблицы. Данные для хранения — полиномы. Для таблиц необходимо реализовать функции: вставка, поиск и удаление, которые выполняются параллельно во всех таблицах. Так же требуется консольное приложение, иллюстрирующее работу программы.

# Руководство пользователя

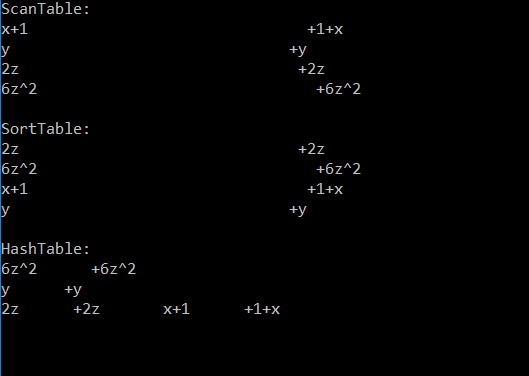
Для работы с программой необходимо запустить на исполнение файл Table.exe.

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



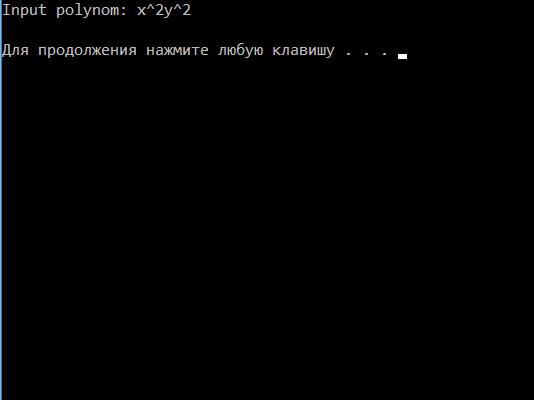
1. Главное меню

Пункт 1 позволяет просмотреть все таблицы, где первый столбец — ключи, второй — соответствующие полиномы (рис. 2).



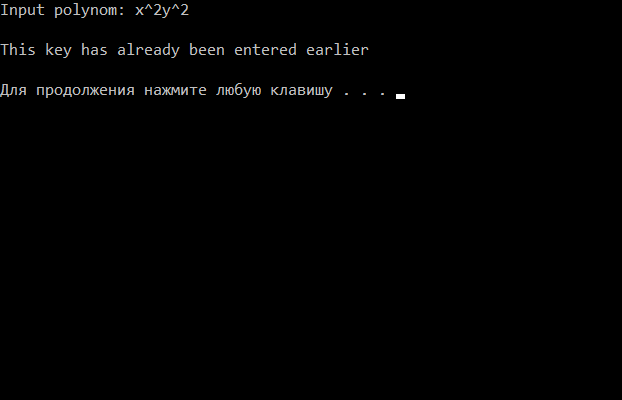
1. Просмотр таблиц

Пункт 2 меню позволяет вставить полином в таблицы. После выбора данного пункта нужно ввести полином (рис. 3).



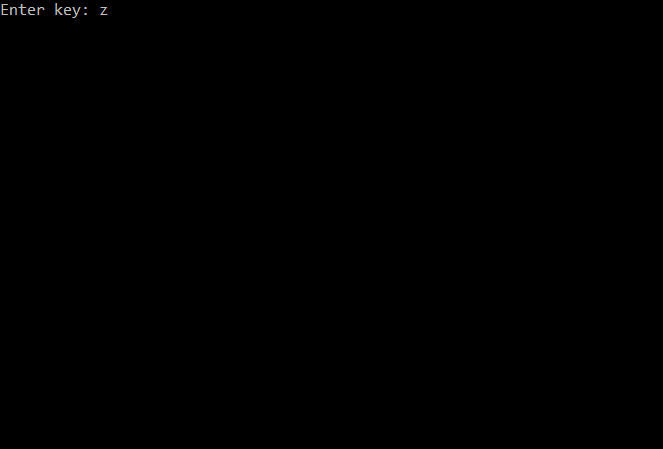
1. Вставка

Если попытаться вставить полином с ключом, который уже хранится в таблицах, выведется сообщение об ошибке (рис. 4).



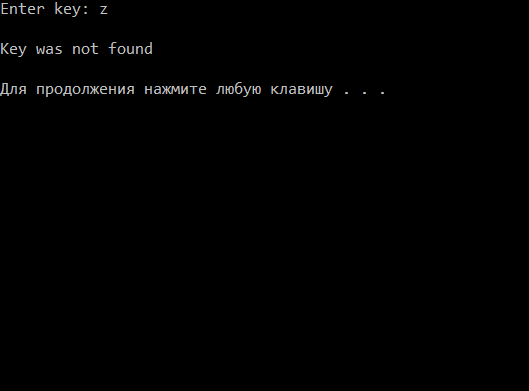
1. Сообщение об ошибке при вставке

Пункт 3 меню позволяет осуществить удаление полинома из всех таблиц. При выборе этого пункта нужно ввести ключ удаляемого элемента (рис. 5).



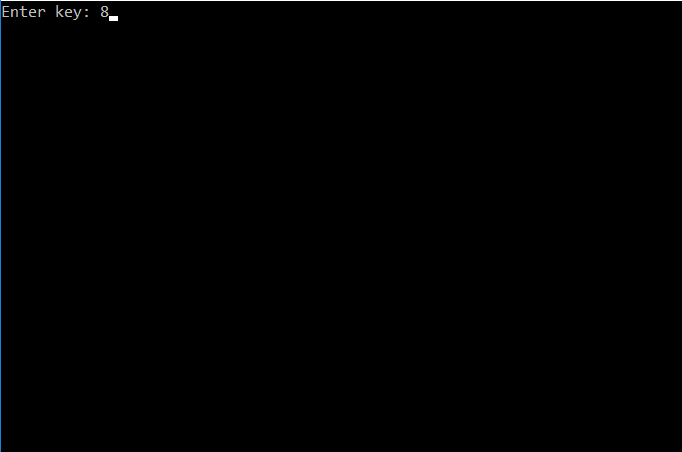
1. Удаление

При попытке удалить полином по ключу, которого нет в таблицах, выведется сообщение об ошибке (рис. 6).



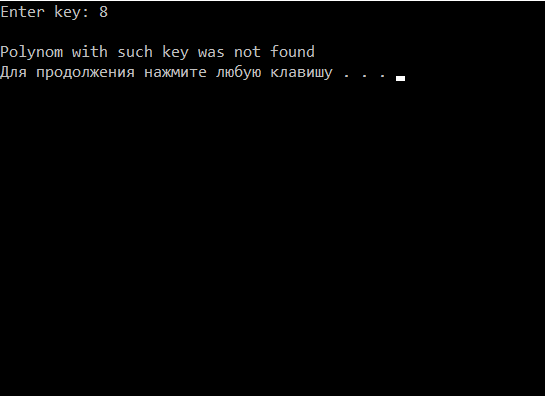
1. Сообщение об ошибке при удалении

Пункт 4 меню позволяет осуществить поиск полинома. При выборе этого пункта нужно ввести ключ элемента, который нужно найти (рис. 7).



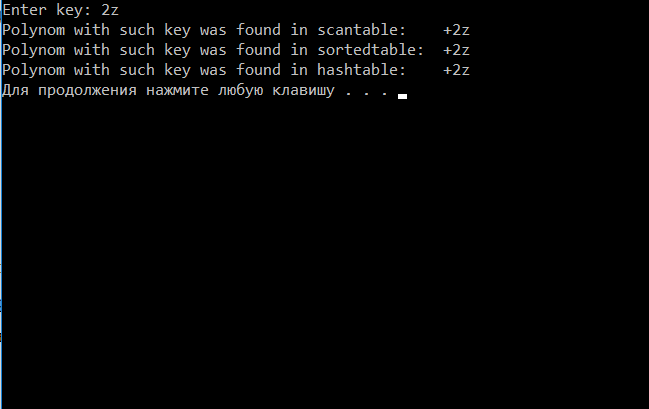
1. Поиск

При попытке найти полином по ключу, которого нет в таблицах, выведется сообщение об ошибке (рис. 8).



1. Сообщение об ошибке при поиске

Если полином с введённым ключом найден, на экран выведется соответствующий результат (рис. 9).



1. Результат поиска

Для выхода из программы необходимо выбрать 5 пункт меню.

# Руководство программиста

## Структура программы

Проекты и файлы, из которых состоит программа:

1. gtest — проект, содержащий Google Tests Framework:

gtest.h — заголовочный файл для Google Tests Framework

gtest-all.cc — файл с кодом Google Tests Framework

1. list — проект, содержащий реализацию линейного односвязного списка, и списка «с головой» (второй был описан в предыдущем отчёте):

line\_list.h — заголовочный файл, содержащий объявление узла линейного списка, объявление и реализацию шаблонного класса линейного списка

line\_list.cpp — файл исходного кода класса линейного списка

1. tables — проект, содержащий реализацию трёх таблиц:

tab\_record.h — заголовочный файл, содержащий объявление и реализацию структуры — записи таблицы

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

scan\_table.h — заголовочный файл, содержащий объявление и реализацию класса неупорядоченных таблиц

sorted\_table.h — заголовочный файл, содержащий объявление и реализацию класса упорядоченных таблиц

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

1. sample — проект, содержащий реализацию пользовательского интерфейса:

main.cpp — файл исходного кода консольного приложения

1. tests — проект, содержащий тесты на правильность реализации трёх классов таблиц (а также полином и списка «с головой», которые описаны в предыдущем отчёте):

test\_main.cpp — файл для запуска всех тестов

test\_tables.cpp — файл, содержащий тесты на правильность реализации классов упорядоченных, неупорядоченных и хэш-таблиц

## Структуры данных

### Класс линейный односвязный список

Поля класса:

* указатель на первый элемент
* указатель на текущий элемент

### Структура «запись таблицы»

Поля класса:

* указатель на данные
* ключ

### Абстрактный класс таблиц

Поля класса:

* максимальный размер таблицы
* текущий размер таблицы
* индекс текущего элемента

### Класс неупорядоченных таблиц

Поля класса:

* массив строк

Методы класса:

*Поиск*

На вход подается ссылка на ключ.

1. Идём по таблице, пока она не кончится, или пока не нашли нужный ключ.
2. Если ключ найден, возвращаем указатель на данные, связанные с этим ключом, а если дошли до конца таблицы и не нашли нужный ключ, бросаем исключение.

*Вставка*

На вход подаются ссылка на ключ и ссылка на данные.

1. Проверяем, есть ли уже такой ключ. Если да, то бросаем исключение, а если нет, то переходим к пункту 2.
2. Если текущий размер таблицы равен максимальному, то создаем новую таблицу с большим максимальным размером и копируем в неё исходную.
3. Вставляем строку (входные данные) в конец.
4. Увеличиваем текущий размер на 1.

*Удаление*

На вход подается ссылка на ключ.

1. Обходим таблицу, пока она не кончится, или пока не нашли нужный ключ.
2. Если нашли нужный ключ, то уменьшаем текущий размер на 1, на место удаленной строки вставляем последнюю строку таблицы. Если дошли до конца и не нашли нужный ключ, то бросаем исключение.

### Класс упорядоченных таблиц

Поля класса:

* массив строк

Методы класса:

*Поиск*

На вход подается ссылка на ключ.

1. Находим строку, в которой данные с данным ключом.
2. Если ключ, стоящий в этой строке, равен данному и номер этой строки в пределах текущего размера таблицы, то возвращаем указатель на данные, связанные с этим ключом.

Иначе, бросаем исключение.

*Вставка*

На вход ссылка на ключ и ссылка на данные.

1. Если текущий размер равен максимальному, то создаем новую таблицу с большим максимальным размером и копируем в неё исходную.
2. Находим номер строки, на которой должны быть данные с данным ключом.
3. Если ключ, стоящий в этой строке, равен данному, то бросаем исключение.
4. Сдвигаем все строки на одну вниз, увеличиваем текущий размер таблицы на 1, вставляем на нужное место строку (входные данные).

*Удаление*

На вход подается ссылка на ключ.

1. Находим строку, в которой данный ключ.
2. Если ключ, стоящий в этой строке, равен данному и номер этой строки в пределах текущего размера таблицы, то сдвигаем все строки, начиная с данной на одну вверх, уменьшаем на 1 текущий размер таблицы.

Иначе, бросаем исключение.

### Класс хэш-таблиц

Поля класса:

* массив линейных списков из строк

Методы класса:

*Хэш-функция*

Используется аддитивный метод реализации.

На вход подается ключ.

Складываем коды всех символов данного ключа и берем остаток от деления на максимальный размер таблицы.

Получаем номер строки, в которой будет находиться данный ключ.

*Поиск*

На вход подается ссылка на ключ.

1. С помощью хэш-функции вычисляем номер списка, в котором должен быть данный ключ.
2. Если в этом списке найдена строка с данным ключом, то возвращаем указатель на данные, связанные с этим ключом.

Иначе бросаем исключение.

*Вставка*

На вход подаются ссылка на ключ и ссылка на данные.

1. С помощью хэш-функции вычисляем номер списка, в который необходимо вставить входные данные.
2. Если в этом списке не найдена строка с данным ключом:
   1. Если этот список пустой, то увеличиваем текущий размер таблицы на 1, иначе переходим к пункту 5.
   2. Если таблица заполнена больше, чем на 60%, то делаем перепаковку.
   3. Вставляем строку, состоящую из входных данных, в начало линейного списка.

Иначе, бросаем исключение.

*Удаление*

На вход подается ссылка на ключ.

1. С помощью хэш-функции вычисляем номер списка, из которого необходимо удалить строку с данным ключом.
2. Если в этом списке не найдена строка с данным ключом, то исключение.
3. Удаляем строку из списка.
4. Если список стал пустой, то уменьшаем текущий размер таблицы на 1.

## Программная реализация

### Структура Node

Node — шаблонная структура, представляющая собой звено линейного списка.

*Поля структуры:*

DataType data — данные в звене

Node\* next — указатель на следующий элемент списка

*Методы структуры:*

Node(DataType d = 0, Node<DataType>\* n = nullptr) — конструктор

Node(const Node<DataType>& node2) — конструктор с параметром

bool operator==(const Node<DataType>& node2) const — оператор =

~Node() — деструктор

### Класс List

List — шаблонные класс, представляющий собой линейный список.

*Поля класса:*

Node<DataType>\* head — указатель на голову списка

Node<DataType>\* cur — указатель на текущий элемент списка

*Методы класса:*

List() — конструктор

List(const List<DataType>& list2) — конструктор копирования

List<DataType>& operator=(const List<DataType>& list2) — оператор =

~List()— деструктор

void InsertToHead(const DataType& d) — вставить элемент d первым

void InsertToTail(const DataType& d) — вставить элемент d последним

void InsertAfter(Node<DataType>\* node, const DataType& d) — вставить элемент d после звена node

void Delete(const DataType& d) — удалить звено со значением data = d

Node<DataType>\* Search(const DataType& d) — найти указатель на звено со значением data = d

void Clean() — удалить все звенья

int GetSize() — узнать число звеньев в списке

Node<DataType>\* GetHead() — получить указатель на первое звено списка

void Inverse() — инвертировать список

List<DataType> Merge(Node<DataType>\* node, const List& list2) —создать список3, добавив список2 в текущий список после указателя node

List<DataType> Merge(const List<DataType>& list2) — создать список3, добавив в конец текущего списка список2

friend ostream& operator<<(ostream& os, const List& l) — вывод

bool operator==(const List& list2) const — списки равны, если элементы в них идут в одинаковом порядке

bool IsEmpty() const — проверить, пуст ли список

void Reset() — поставить укзатель на голову списка

void GetNext() — переместить указатель на следующий

bool IsEnded() const — проверка на попадание в конец списка

Node<DataType>\* GetCur() — получить указатель на текущий элемент списка

### Структура record

Record — шаблонная структура, представляющая собой строку таблицы.

*Поля структуры*:

type\* data — указатель на данные

string key — ключ

*Методы структуры:*

record() — конструктор без параметров

record(string k) — конструктор с параметром

record(string s, type t) — конструктор с параметрами

record(const record<type>& r) — конструктор копирования

record<type>& operator=(const record<type>& r) — оператор присваивания

~record() — деструктор

bool operator==(const record<type>& r) const — оператор =

bool operator!=(const record<type>& r) — оператор !=

### Класс table

table — абстрактный базовый класс для таблиц.

*Поля класса:*

unsigned int maxsize — максимальный размен таблицы

unsigned int cursize — текущий размен таблицы

int idx — индекс текущего элемента

*Методы класса:*

virtual void Realloc() = 0  — перепаковка таблицы

table(int any\_size = S) — конструктор с параметром

virtual ~table() — деструктор

virtual type\* Search(const string& k) = 0 — поиск записи по ключу

virtual void Insert(const string& k, const type& d) = 0 — вставка в таблицу строки, состоящей из ключа k и данных d

virtual void Delete(const string& k) = 0 — удалить строку по ключу

virtual void Reset() — переместить указатель на первую строку

virtual bool IsTabEnded() const — проверка на достижение конца таблицы

virtual void GetNext() — переместить указатель на следующую строку

virtual record<type> GetCurrent() const = 0 — получить текущую строку

virtual bool IsEmpty() — проверка, пуста ли таблица

template<class type> friend ostream& operator<<(ostream& os, const table<type>& tab) = 0 — вывод таблицы на экран

### Класс scantable

Scantable — шаблонный класс, унаследованный от базового абстрактного класса table, представляющий собой неупорядоченную таблицу.

*Поля класса:*

record<type>\* z — массив строк

*Методы класса:*

void Realloc() override — перепаковка

scantable(int any\_size = S) — конструктор с параметром

~scantable() —деструктор

type\* Search(const string& k) override —поиск записи по ключу

void Insert(const string& k, const type& d) override — вставка в таблицу строки, состоящей из ключа k и данных d

void Delete(const string& k) overrid —удалить из таблицы строку по ключу

record<type> GetCurrent() const override — получить текущую строку

template<class t> friend ostream& operator<<(ostream& os, scantable<t>& tab)— вывод таблицы на экран

### Класс sortedtable

sortedtable — шаблонный класс, унаследованный от базового абстрактного класса table, представляющий собой неупорядоченную таблицу.

*Поля класса:*

record<type>\* z — массив строк

*Методы класса:*

int BSearch(const string k) — бинарный поиск данных по ключу

void Realloc() override — перепаковка таблицы

sortedtable(int any\_size = S)  — конструктор с параметром

sortedtable(const scantable<type>& st) конструктор копирования

~sortedtable()() —деструктор

type\* Search(const string& k) override —поиск записи по ключу

void Insert(const string& k, const type& d) override — вставка в таблицу строки, состоящей из ключа k и данных d

void Delete(const string& k) overrid —удалить из таблицы строку по ключу

record<type> GetCurrent() const override — получить текущую строку

template<class t> friend ostream& operator<<(ostream& os, sortedtable<t>& tab) — вывод таблицы на экран

### Класс hashtable

hashtable — шаблонный класс, унаследованный от базового абстрактного класса table, представляющий собой хэш-таблицу.

*Поля класса:*

List<record<type>>\* HT — массив списков из строк

*Методы класса:*

int hf (const string s)  — хэш-функция

void Realloc() override — перепаковка

hashtable(int any\_size = S)  — конструктор с параметром

~hashtable()()() —деструктор

type\* Search(const string& k) override —поиск записи по ключу

void Insert(const string& k, const type& d) override — вставка в таблицу строки, состоящей из ключа k и данных d

void Delete(const string& k) overrid —удалить из таблицы строку по ключу

void Reset() override — переместить указатель на первую непустую строку таблицы

void GetNext()override — переместить указатель на следующую непустую строку таблицы

bool IsTabEnded() const override — проверить, дошли ли до конца таблицы

record<type> GetCurrent() const override — получить текущую строку

template<class t> friend ostream& operator<<(ostream& os, hashtable<t>& tab)  — вывод таблицы на экран

# Заключение

В ходе выполнения данной работы была разработана программа, позволяющая хранить данные в таблицах трех видов: неупорядоченные, упорядоченные, хэш-таблицы. Как и требовалось, были реализованы функции поиска, вставки и удаления для каждой таблицы. Так же было разработано пользовательское пользовательское приложение с доступным интерфейсом.

# Литература

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 А. Программная реализация линейного списка

**line\_list.h**

#pragma once

#include <iostream>

using namespace std;

template <class DataType>

struct Node

{

DataType data;

Node\* next;

Node(DataType d = 0, Node<DataType>\* n = nullptr) { data = d; next = n; }

Node(const Node<DataType>& node2) { data = node2.data; next = nullptr; }

bool operator==(const Node<DataType>& node2) const { if (data == node2.data && next == node2.next) return true; else return false; }

~Node() {}

};

template <class DataType>

class List

{

// любые поля

Node<DataType>\* head;

Node<DataType>\* cur;

public:

List() { head = nullptr; }

List(const List<DataType>& list2);

List<DataType>& operator=(const List<DataType>& list2);

~List() { Clean(); }

void InsertToHead(const DataType& d); // вставить элемент d первым

void InsertToTail(const DataType& d); // вставить элемент d последним

void InsertAfter(Node<DataType>\* node, const DataType& d); // вставить элемент d после звена node

void Delete(const DataType& d); // удалить звено со значением data = d

Node<DataType>\* Search(const DataType& d); // найти указатель на звено со значением data = d

void Clean(); // удалить все звенья

int GetSize(); // узнать число звеньев в списке

Node<DataType>\* GetHead() { return head; } // получить указатель на первое звено списка

void Inverse(); // инвертировать список, т.е. звенья должны идти в обратном порядке

List<DataType> Merge(Node<DataType>\* node, const List& list2); // создать список3, добавив список2 в текущий список после указателя node

List<DataType> Merge(const List<DataType>& list2); // создать список3, добавив в конец текущего списка список2

friend ostream& operator<<(ostream& os, const List& l);

bool operator==(const List& list2) const; // списки равны, если элементы в них идут в одинаковом порядке

bool IsEmpty() const { return head == nullptr; }

void Reset() { cur = head; }

void GetNext() { cur = cur->next; }

bool IsEnded() const { return cur == nullptr; } //

Node<DataType>\* GetCur() { return cur; }

};

template <class DataType>

List<DataType>::List(const List<DataType>& list2)

{

if (list2.head)

{

head = new Node<DataType>(list2.head->data, list2.head);

Node<DataType> \*u = head;

Node<DataType> \*u2 = list2.head->next;

while (u2 != nullptr)

{

u->next = new Node<DataType>(u2->data, u2->next);

u = u->next;

u2 = u2->next;

}

}

else

head = list2.head;;

}

template <class DataType>

List<DataType>& List<DataType>:: operator=(const List<DataType>& list2)

{

if (this != &list2)

{

Clean();

if (list2.head)

{

head = new Node<DataType>(list2.head->data, list2.head);

Node<DataType> \*u = head;

Node<DataType> \*u1 = list2.head->next;

while (u1)

{

u->next = new Node<DataType>(u1->data);

u = u->next;

u1 = u1->next;

}

}

else

{

Clean();

head = list2.head;

}

}

return \*this;

}

template <class DataType>

void List<DataType>::InsertToHead(const DataType& d)

{

if (head != nullptr)

{

Node<DataType>\* u = new Node<DataType>(d, head);

head = u;

}

else

head = new Node<DataType>(d, nullptr);

}

template <class DataType>

void List<DataType>::InsertToTail(const DataType& d)

{

Node<DataType>\*u = head;

if (u)

{

while (u->next)

u = u->next;

u->next = new Node<DataType>(d);

}

else

head = new Node<DataType>(d, nullptr);

}

template <class DataType>

void List<DataType>::InsertAfter(Node<DataType>\* node, const DataType& d)

{

if (node && head)

{

Node<DataType>\* u = node->next;

node->next = new Node<DataType>(d, u);

}

else

if (head == nullptr)

throw 1;

}

template <class DataType>

void List<DataType>::Delete(const DataType& d)

{

Node<DataType>\* u = head;

Node<DataType>\* u2 = u;

if (u)

{

while ((u->next) && (u->data != d))

{

u2 = u;

u = u->next;

}

if (u->data == d)

{

u2->next = u->next;

if (u == head)

head = head->next;

delete u;

}

}

}

template <class DataType>

Node<DataType>\* List<DataType>::Search(const DataType& d)

{

Node<DataType>\* u = head;

while (u)

{

if (u->data == d)

return u;

u = u->next;

}

return nullptr;

}

template <class DataType>

void List<DataType>::Clean()

{

Node<DataType>\* u = head;

Node<DataType>\* u2;

if (u != nullptr)

{

while (u->next != nullptr)

{

u2 = u->next;

delete u;

u = u2;

}

delete u;

head = nullptr;

}

}

template <class DataType>

int List<DataType>::GetSize()

{

int res = 0;

Node<DataType>\* u = head;

while (u)

{

res++;

u = u->next;

}

return res;

}

template <class DataType>

void List<DataType>::Inverse()

{

if (head)

{

Node<DataType>\* u = head->next;

head->next = nullptr;

Node<DataType>\* u2 = head;

Node<DataType>\* u3;

while (u)

{

u3 = u->next;

u->next = u2;

u2 = u;

u = u3;

}

head = u2;

}

else

head = nullptr;

}

template <class DataType>

List<DataType> List<DataType>::Merge(Node<DataType>\* node, const List<DataType>& list2)

{

List res;

Node<DataType>\* u = head;

if (head)

{

res.head = new Node<DataType>(head->data);

Node<DataType>\* u1 = res.head;

while (u->next && u != node)

{

u1->next = new Node<DataType>(u->next->data);

u = u->next;

u1 = u1->next;

}

if (u == node)

{

if (list2.head)

{

Node<DataType>\* u2 = list2.head;

while (u2)

{

u1->next = new Node<DataType>(u2->data);

u2 = u2->next;

u1 = u1->next;

}

}

while (u->next)

{

u1->next = new Node<DataType>(u->next->data);

u = u->next;

u1 = u1->next;

}

}

u1->next = nullptr;

}

return res;

}

template <class DataType>

List<DataType> List<DataType>::Merge(const List<DataType>& list2)

{

List res(\*this);

if (res.head && list2.head)

{

Node<DataType>\* u = res.head;

while (u->next)

u = u->next;

Node<DataType>\* u1 = list2.head;

while (u1)

{

u->next = new Node<DataType>(u1->data);

u = u->next;

u1 = u1->next;

}

u->next = nullptr;

}

else

if (!(res.head))

res = List(list2);

return res;

}

template <class DataType>

ostream& operator<<(ostream& os, const List<DataType>& l)

{

Node<DataType>\* u = l.head;

while (u)

{

os << u->data << " ";

u = u->next;

}

return os;

}

template <class DataType>

bool List<DataType>:: operator==(const List<DataType>& list2) const

{

Node<DataType>\* u = head;

Node<DataType>\* u1 = list2.head;

bool res = true;

if ((u && !u1) || (!u && u1))

res = false;

if (res)

{

while (u && u1)

{

if (u->data != u1->data)

{

res = false;

u = u->next;

u1 = u1->next;

}

else

{

u = u->next;

u1 = u1->next;

}

}

}

return res;

}

**line\_list.cpp**

#include "line\_list.h"

## Приложение Б. Программная реализация таблиц

**tab\_record.h**

#pragma once

#include <string>

using namespace std;

template <class type>

struct record

{

type\* data;

string key;

record() {key = ""; data = nullptr;} //Конструктор без параметров

record(string k) { key = k; data = nullptr; }

record(string s, type t) {key = s; data = new type(t);} //Конструктор с парметрами

record(const record<type>& r) { key = r.key; data = new type(\*(r.data)); } //Конструктор копирования

record<type>& operator=(const record<type>& r); //Присваивание

~record() { delete data; };

bool operator==(const record<type>& r) const { return key == r.key; }

bool operator!=(const record<type>& r) const { return !(\*this == r); }

};

template <class type> record<type>& record<type>::operator=(const record<type>& r)

{

key = r.key;

delete data;

data = new type(\*(r.data));

return \*this;

}

**table.h**

#pragma once

#include "list.h"

#include "tab\_record.h"

#define S 10

template <class type>

class table

{

protected:

unsigned int maxsize;

unsigned int cursize;

int idx;

virtual void Realloc() = 0;

public:

table(int any\_size = S) { maxsize = any\_size; cursize = 0; idx = -1; } //Конструктор

virtual ~table() {};

virtual type\* Search(const string& k) = 0;

virtual void Insert(const string& k, const type& d) = 0;

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

virtual void Reset();

virtual bool IsTabEnded() const;

virtual void GetNext();

virtual record<type> GetCurrent() const = 0;

virtual bool IsEmpty() { return cursize == 0; }

template<class type> friend ostream& operator<<(ostream& os, const table<type>& tab) = 0;

};

template<class type> void table<type>::Reset()

{

if (!IsEmpty())

idx = 0;

else

throw "Table is empty";

}

template<class type> bool table<type>::IsTabEnded() const

{

return (idx >= cursize - 1 ); //нулевая запись

}

template <class type> void table<type>::GetNext()

{

if (!IsTabEnded())

idx++;

else

//

Reset();

//throw "Next record is not found";

}

**table.cpp**

#include "tab\_record.h"

#include "table.h"

#include "scan\_table.h"

#include "sorted\_table.h"

#include "hash\_table.h"

**scan\_table.h**

#pragma once

#include "table.h"

template <class type>

class scantable :public table <type>

{

protected:

record<type>\* z;

void Realloc() override;

public:

scantable(int any\_size = S) : table(any\_size) { z = new record<type>[maxsize]; }

~scantable() {delete[] z;}

type\* Search(const string& k) override;

void Insert(const string& k, const type& d) override;

void Delete(const string& k) override;

record<type> GetCurrent() const override;

template<class t> friend ostream& operator<<(ostream& os, scantable<t>& tab);

};

template <class type> void scantable<type>::Realloc()

{

maxsize \*= 1.5;

record<type>\* new\_z = new record<type>[maxsize];

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

new\_z[i] = z[i];

delete[] z;

z = new\_z;

}

template <class type> type\* scantable<type>::Search(const string& k)

{

int i = 0;

while (i < cursize && z[i].key != k)

i++;

if (i < cursize)

return z[i].data;

else

throw "Data with this key was not found";

}

template <class type> void scantable<type>::Insert(const string& k, const type& d)

{

int i = 0;

bool r = true;

while (i < cursize && z[i].key != k)

i++;

if (i != cursize && z[i].key == k)

r = false;

if (r)

{

if (cursize == maxsize)

Realloc();

z[cursize] = record<type>(k, d);

cursize++;

}

else

throw "This key has already been entered earlier";

}

template <class type> void scantable<type>::Delete(const string& k)

{

int i = 0;

while (i < cursize && z[i].key != k)

i++;

if (i < cursize)

z[i] = z[--cursize];

else

throw "Data with this key was not found";

}

template<class type> record<type> scantable<type>::GetCurrent() const

{

if (idx < cursize )

return z[idx];

else

throw "Not found";

}

template<class type> ostream& operator<<(ostream& os, scantable<type>& tab)

{

if (tab.cursize)

{

tab.Reset();

os << tab.GetCurrent().key << " " << \*(tab.GetCurrent().data) << endl;

while (!tab.IsTabEnded())

{

tab.GetNext();

os << tab.GetCurrent().key << " " << \*(tab.GetCurrent().data) << endl;

}

}

else

os << "Empty table" << endl;

return os;

}

**sorted\_table.h**

#pragma once

#include "table.h"

#include <iostream>

using namespace std;

template <class type>

class sortedtable :public table <type>

{

protected:

record<type>\* z;

int BSearch(const string k);

void Realloc() override;

public:

sortedtable(int any\_size = S) : table(any\_size) { z = new record<type>[maxsize]; }

sortedtable(const scantable<type>& st); //Конструктор копирования

~sortedtable() {delete[] z;}

type\* Search(const string& k) override;

void Insert(const string& k, const type& d) override;

void Delete(const string& k) override;

record<type> GetCurrent() const override;

template<class t> friend ostream& operator<<(ostream& os, sortedtable<t>& tab);

};

template <class type> void sortedtable<type>::Realloc()

{

maxsize \*= 1.5;

record<type>\* new\_z = new record<type>[maxsize];

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

new\_z[i] = z[i];

delete[] z;

z = new\_z;

}

template <class type> int sortedtable<type>::BSearch(const string k)

{

int start = 0;

int finish = cursize - 1;

int centre;

while (start <= finish) // 0...n

{

centre = (start + finish) / 2;

if (k <= z[centre].key)

finish = centre - 1;

else

start = centre + 1;

}

return start;

}

template <class type> sortedtable<type>::sortedtable(const scantable<type> & st) : table(st.maxsize)

{

for (int i = 0; i < st.cursize; i++)

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

Reset();

}

template <class type> type\* sortedtable<type>::Search(const string& k)

{

int p = BSearch(k);

if (z[p].key == k && p < cursize)

return z[p].data;

else

throw "Data with this key was not found";

}

template <class type> void sortedtable<type>::Insert(const string& k, const type& d)

{

if (cursize == maxsize)

Realloc();

int p = BSearch(k);

if ( z[p].key != k)

{

for (int i = cursize; i > p; i--)

z[i] = z[i-1];

cursize++;

z[p] = record<type>(k, d);

}

else

throw "this key already exists";

}

template <class type> void sortedtable<type>::Delete(const string& k)

{

int p = BSearch(k);

if (p < cursize && z[p].key == k)

{

for (int i = p; i < cursize - 1; i++)

z[i] = z[i + 1];

cursize--;

}

else

throw "Data with this key was not found";

}

template<class type> record<type> sortedtable<type>::GetCurrent() const

{

if (idx < cursize)

return z[idx];

else

throw "Not found";

}

template<class type> ostream& operator<<(ostream& os, sortedtable<type>& tab)

{

if (tab.cursize)

{

tab.Reset();

os << tab.GetCurrent().key << " " << \*(tab.GetCurrent().data) << endl;

while (!tab.IsTabEnded())

{

tab.GetNext();

os << tab.GetCurrent().key << " " << \*(tab.GetCurrent().data) << endl;

}

}

else

os << "Empty table" << endl;

return os;

}

**hash\_ table.h**

#pragma once

#include "table.h"

#include "line\_list.h"

template <class type>

class hashtable :public table <type>

{

protected:

int hf (const string s);

void Realloc() override;

List<record<type>>\* HT;

public:

hashtable(int any\_size = S) : table(any\_size) { HT = new List<record<type>>[any\_size]; }

~hashtable() { delete[] HT; }

type\* Search(const string& k) override;

void Insert(const string& k, const type& d) override;

void Delete(const string& k) override;

void Reset() override;

void GetNext()override;

bool IsTabEnded() const override;

record<type> GetCurrent() const override { return HT[idx].GetCur()->data;}

template<class t> friend ostream& operator<<(ostream& os, hashtable<t>& tab);

};

template <class type> int hashtable<type>::hf (const string s)

{

//additive method

int sum = 0;

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

sum += (int)(s[i]);

return (sum % maxsize);

}

template <class type> void hashtable<type>::Realloc()

{

maxsize \*= 1.5;

List<record<type>>\* new\_HT = new List<record<type>>[maxsize];

Reset();

new\_HT[hf(GetCurrent().key)].InsertToHead(GetCurrent());

while (!IsTabEnded())

{

GetNext();

new\_HT[hf(GetCurrent().key)].InsertToHead(GetCurrent());

}

delete[] HT;

HT = new\_HT;

}

template <class type> type\* hashtable<type>::Search(const string& k)

{

if (HT[hf(k)].Search(record<type>(k)) != nullptr)

return ((HT[hf(k)].Search(record<type>(k)))->data).data;

else

throw "Not found";

}

template <class type> void hashtable<type>::Insert(const string& k, const type& d)

{

int pos = hf(k);

if (HT[pos].Search(record<type>(k)) == nullptr)

{

if (HT[pos].IsEmpty())

{

cursize++;

if (double(cursize) / maxsize > 0.6)

Realloc();

}

HT[pos].InsertToHead(record<type>(k, d));

}

else

throw "Repeat key";

}

template <class type> void hashtable<type>::Delete(const string& k)

{

if (HT[hf(k)].Search(record<type>(k)) != nullptr)

{

HT[hf(k)].Delete(record<type>(k));

if (HT[hf(k)].IsEmpty())

cursize--;

}

else

throw "Not found";

}

template <class type> void hashtable<type>::Reset() // устанавливает idx на первую цепочку и идёт в первый элеммент цепочки

{

int i = 0;

if (cursize)

{

while (HT[i].IsEmpty())

i++;

idx = i;

HT[idx].Reset();

}

else

throw "Table is empty";

}

template <class type> void hashtable<type>::GetNext()

{

if (!HT[idx].IsEnded())

HT[idx].GetNext();

else

{

idx++;

while (idx < maxsize && HT[idx].IsEmpty())

idx++;

if (idx == maxsize)

Reset();

else

HT[idx].Reset();

}

}

template <class type> bool hashtable<type>::IsTabEnded() const

{

bool res = false;

int i = idx;

if (cursize)

{

if (HT[idx].IsEnded())

{

i++;

while (i < maxsize && HT[i].IsEmpty())

i++;

if (i == maxsize)

res = true;

}

}

else

res = true;

return res;

}

template<class type> ostream& operator<<(ostream& os, hashtable<type>& tab)

{

if (tab.cursize)

{

tab.Reset();

int i = tab.idx; // первый непустой

if (!tab.IsTabEnded())

{

do

{

tab.HT[tab.idx].Reset();

while (!tab.HT[tab.idx].IsEnded())

{

os << tab.HT[tab.idx].GetCur()->data.key << " " << \*(tab.HT[tab.idx].GetCur()->data.data) << " ";

tab.HT[tab.idx].GetNext();

}

os << endl;

tab.GetNext();

} while (!tab.IsTabEnded() && i < tab.idx);

}

}

else

os << "Empty table";

return os;

}

## Приложение Б. Программная реализация приложения

**main.cpp**

#include "scan\_table.h"

#include "sorted\_table.h"

#include "hash\_table.h"

#include <iostream>

#include "polinom.h"

char main\_menu()

{

system("cls");

char str;

cout << "What will you do with tables?" << endl;

cout << "1. Show tables" << endl;

cout << "2. Insert" << endl;

cout << "3. Delete" << endl;

cout << "4. Search" << endl;

cout << "5. Exit" << endl;

cout << endl;

cout << "Your choice: ";

cin >> str;

return str;

}

void main()

{

char pmenu;

scantable<polynom> ScT;

sortedtable<polynom> SrT;

hashtable<polynom> HsT;

polynom pol;

string inp\_pol;

do {

pmenu = main\_menu();

switch (pmenu)

{

case '1':

{

system("cls");

cout << "ScanTable: " << endl << ScT << endl;

cout << "SortTable: " << endl << SrT << endl;

cout << "HashTable: " << endl << HsT << endl;

cout << endl << endl;

system("pause");

break;

}

case '2':

{

system("cls");

cout << "Input polynom: ";

cin >> inp\_pol;

cout << endl;

pol = inp\_pol;

try

{

ScT.Insert(inp\_pol, pol);

SrT.Insert(inp\_pol, pol);

HsT.Insert(inp\_pol, pol);

}

catch (...)

{

cout << "This key has already been entered earlier" << endl << endl;

}

system("pause");

break;

}

case '3':

{

system("cls");

cout << "Enter key: ";

cin >> inp\_pol;

try

{

ScT.Delete(inp\_pol);

SrT.Delete(inp\_pol);

HsT.Delete(inp\_pol);

}

catch (...)

{

cout << endl << "Key was not found" << endl << endl;

}

system("pause");

break;

}

case '4':

{

system("cls");

cout << "Enter key: ";

cin >> inp\_pol;

try

{

cout << "Polynom with such key was found in scantable: " << \*(ScT.Search(inp\_pol)) << endl;

cout << "Polynom with such key was found in sortedtable: " << \*(SrT.Search(inp\_pol)) << endl;

cout << "Polynom with such key was found in hashtable: " << \*(HsT.Search(inp\_pol)) << endl;

}

catch (...)

{

cout << endl << "Polynom with such key was not found" << endl;

}

system("pause");

break;

}

}

} while (pmenu != '5');

}