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

**Таблицы**

**Выполнила:** студентка группы 381603-1

Фадеева А.М.

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

Кустикова В.Д.

Нижний Новгород  
2018

**Содержание**

[Введение 3](#_Toc515555865)

[1 Постановка задачи 4](#_Toc515555866)

[2 Руководство пользователя 5](#_Toc515555867)

[3 Руководство программиста 10](#_Toc515555868)

[3.1 Описание структуры программы 10](#_Toc515555869)

[3.2 Описание структуры данных 11](#_Toc515555870)

[3.3 Описание программной реализации 14](#_Toc515555871)

[Заключение 19](#_Toc515555872)

[Список литературы 20](#_Toc515555873)

[Приложение 21](#_Toc515555874)

[Приложение А. Программная реализация линейного односвязного списка 21](#_Toc515555875)

[Приложение Б. Программная реализация таблиц 26](#_Toc515555876)

[Приложение В. Программная реализация приложения 33](#_Toc515555877)

# Введение

Таблица — форма представления информации, облегчающая её визуальное восприятие. В качестве элемента в таблице хранится пара (ключ, данные), а доступ к данным осуществляется по ключу. Благодаря использованию таблиц намного проще работать с данными.

В данной работе в качестве данных используются полиномы, работа с которыми была подробно описана в предыдущей лабораторной работе. В представленной работе можно не только добавлять и удалять строки таблицы, но также можно производить различные манипуляции с полиномами, хранящимися в таблице, – складывать, вычитать, умножать. Если результат операций необходимо добавить в таблицу, то пользователь сразу сможет это сделать.

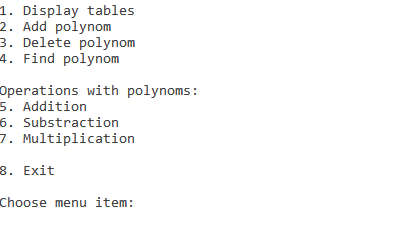
# Постановка задачи

Разработать программу, позволяющую хранить данные в таблицах трех видов: неупорядоченные, упорядоченные, хэш-таблицы. Должны быть реализованы поиск, вставка и удаление. Работоспособность программы необходимо проверить с помощью Google Tests. Кроме того, необходимо разработать пользовательское консольное приложение.

# Руководство пользователя

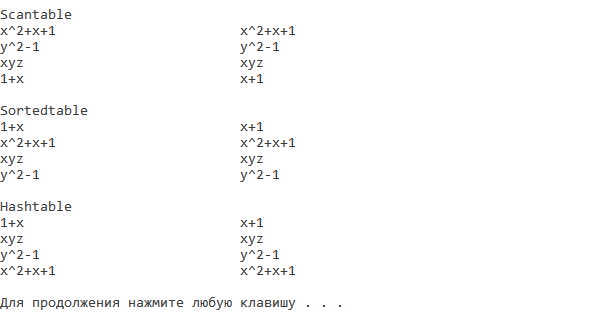
Запустить программу Table.exe.

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



1. Главное меню

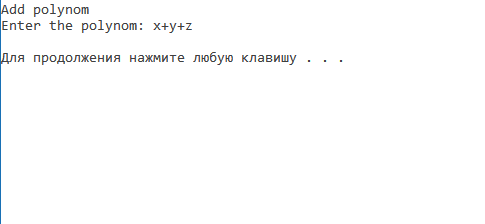
Пункт 1 главного меню позволяет просмотреть все таблицы (рис. 2). Первый столбец – ключ, второй – полином.



1. Просмотр таблиц

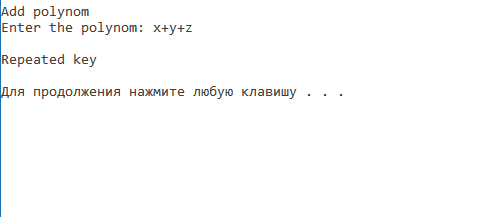
Пункт 2 меню позволяет добавить полином в таблицы.

После выбора данного пункта меню следует ввести полином, который будет добавлен во все таблицы (рис. 3).



1. Добавление полинома

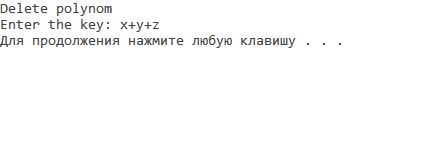
Нельзя добавить полином, который будет записан точно так же, как один из уже хранящихся в таблице. В этом случае пользователь получит сообщение об ошибке и полином повторно добавлен в таблицы не будет (рис.4).



1. Сообщение об ошибке при добавлении

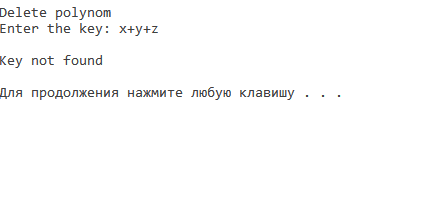
Пункт 3 меню позволяет удалить полином. Удаление происходит по ключу. Ключ – строка, при вводе которой данный полином был добавлен в таблицы.

После выбора данного пункта меню следует ввести ключ полинома, который необходимо удалить (рис.5).



1. Удаление полинома

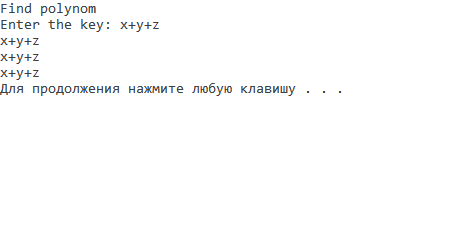
Нельзя удалить полином по ключу, которого нет в таблицах. В этом случае пользователь получит сообщение об ошибке, ничего удалено из таблиц не будет (рис.6).



1. Сообщение об ошибке при удалении

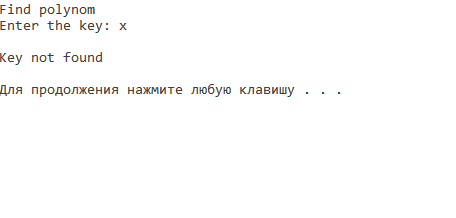
Пункт 4 главного меню позволяет найти полином по ключу.

При выборе данного пункта следует ввести ключ, который соответствует интересующему пользователя полиному (рис. 7). После этого на экран будет выведен полином с этим ключом, хранящий в каждой таблице.



1. Поиск полинома

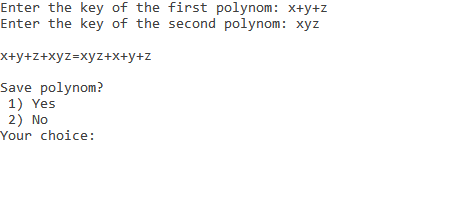
Нельзя найти полином по ключу, которого нет. В этом случае будет выведено сообщение об ошибке (рис. 8).



1. Сообщение об ошибке при поиске

Пункты 5-7 позволяют выбрать действие, которое необходимо совершить над полиномами, хранящимися в таблице.

После выбора одного из этих пунктов следует ввести ключи полиномов, над которыми необходимо совершить действие (рис. 9).



1. Сложение полиномов

Далее на экран будет выведен результат выбранной операции.

Затем пользователю будет предложено сохранить результирующий полином. Если выбрать 1 пункт, тогда результирующий полином будет добавлен в таблицы в данном виде. Если же данный полином ненужно сохранять, то следует выбрать 2 пункт.

Для выхода из программы необходимо выбрать 8 пункт меню.

# Руководство программиста

## Описание структуры программы

Проекты и файлы, из который состоит программа:

1. List – проект, содержащий реализацию линейного односвязного списка:

linelist.h – заголовочный файл, содержащий объявление узла списка, объявление и реализацию шаблонного класса списка.

linelist.cpp – файл исходного кода класса списка.

1. Table – проект, содержащий реализацию таблиц:

line.h – заголовочный файл, в котором находится объявление строки таблицы.

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

scantable.h – заголовочный файл, содержащий объявление и реализацию наследуемого класса неупорядоченных таблиц.

sortedtable.h – заголовочный файл, содержащий объявление и реализацию наследуемого класса упорядоченных таблиц.

hashtable.h – заголовочный файл, содержащий объявление и реализацию наследуемого класса хэш-таблиц.

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. ui – проект, содержащий реализацию пользовательского интерфейса:

main.cpp – файл исходного кода пользовательского интерфейса.

1. tests – проект, содержащий тесты для таблиц:

test\_main.cpp – файл для запуск всех тестов

test\_table.cpp – файл, содержащий тесты для классов упорядоченных, неупорядоченных и хэш-таблиц.

## Описание структуры данных

### Класс линейный односвязный список

Поля класса:

* указатель на «голову»
* указатель на текущий элемент

### Абстрактный класс таблиц

Поля класса:

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

### Класс неупорядоченных таблиц

Поля класса:

* массив указателей на строки (данные и ключ)

Методы класса:

*Вставка*

На вход подаются ключ и данные.

1. Если такого ключа еще нет в таблице, то переходим к пункту 2, иначе исключение.
2. Если текущий размер равен максимальному, то делаем перепаковку.
3. Вставляем строку, состоящую из входных данных и ключа, в конец.
4. Увеличиваем текущий размер на 1.

*Поиск*

На вход подается ключ.

1. Обходим таблицу, пока она не кончится, или не встретим интересующий нас ключ.
2. Если таблица кончилась, то исключение.

Если нашли ключ, то возвращаем указатель на данные с этим ключом.

*Удаление*

На вход подается ключ.

1. Обходим таблицу, пока она не кончится, или не встретим интересующий нас ключ.
2. Если таблица кончилась, то исключение.

Иначе уменьшаем текущий размер на 1, на место удаленной строки вставляем строку с последнего места.

### Класс упорядоченных таблиц

Поля класса:

* массив указателей на строки (данные и ключ)

Методы класса:

*Вставка*

На вход подаются ключ и данные.

1. Находим номер строки, на которой должен стоять данный ключ.
2. Если ключ, стоящий в этой строке, равен данному, то исключение.
3. Если текущий размер равен максимальному, то делаем перепаковку.
4. Сдвигаем строки на одну вниз, вставляем на нужное место строку, состоящую из входных данных и ключа.

*Поиск*

На вход подается ключ.

1. Находим номер строки, в которой должен стоять данный ключ.
2. Если ключ, стоящий в этой строке, равен данному, то возвращаем указатель на данные с этим ключом.

Если ключ, стоящий в этой строке, не равен данному, то исключение.

*Удаление*

На вход подается ключ.

1. Находим номер строки, на которой должен стоять данный ключ.
2. Если ключ, стоящий в этой строке, равен данному, то уменьшаем размер на 1, удаляем строку, сдвигаем строки, стоящие ниже, на 1 вверх.

Если ключ, стоящий в этой строке, не равен данному, то исключение.

### Класс хэш-таблиц

Поля класса:

* массив линейных списков из строк (ключ и данные)

Методы класса:

*Хэш-функция*

На вход подается ключ.

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

Получили номер строки, в которой будет находится данный ключ.

*Вставка*

На вход подаются ключ и данные.

1. С помощью хэш-функции вычисляем номер списка, в который необходимо вставить строку с ключом и данными.
2. Если в этом списке найдена строка с данным ключом, то исключение.
3. Если этот список пустой, то увеличиваем текущий размер таблицы на 1, иначе переходим к пункту 5.
4. Если таблица заполнена больше, чем на 65%, то делаем перепаковку.
5. Вставляем строку, состоящую из входных данных, в начало списка.

*Поиск*

На вход подается ключ.

1. С помощью хэш-функции вычисляем номер списка, в котором должна быть строка.
2. Если в этом списке найдена строка с данным ключом, то возвращаем указатель на данные, иначе исключение.

*Удаление*

На вход подается 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.

## Описание программной реализации

### Node

Node – шаблон структуры, параметризованный DataType, представляет собой звено списка.

*Поля структуры*:

Node\* next – указатель на следующий элемент

DataType data – данные, хранящиеся в звене

*Методы структуры:*

Node(DataType d = DataType(), Node<DataType>\* n = NULL) – конструктор

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

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

bool operator!=( const Node<DataType>& node2) – оператор сравнения

### List

List – шаблон класса, параметризованный DataType, представляет собой линейный односвязный список.

*Поля класса:*

Node<DataType>\* head – указатель на "голову"

Node<DataType>\* cur – указатель на текущий элемент

*Реализованные методы:*

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

bool IsEmpty() – проверка на пустоту

void Reset() – ставит указатель на «голову»

void GetNext() – текущим элементом становиться следующий

bool IsEnded() – проверка на конец списка

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

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

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

### **line**

line – шаблон структуры, параметризованный type, представляет собой строку таблицы.

*Поля структуры*:

type\* data – указатель на данные

string key – ключ

*Методы структуры:*

line() – конструктор

line(string k, type d) – конструктор с параметрами

line(string k) – конструктор с параметром

line<type>& operator=(const line<type>& l) – оператор =

~line() – деструктор

line (const line<type>& l) – конструктор копирования

bool operator==(const line<type>&) const – оператор сравнения

bool operator!=(const line<type>& l) const – оператор сравнения

### Абстрактный класс table

table – шаблон абстрактного базового класса, параметризованный type.

*Поля класса:*

int maxsize – максимальный размер таблицы

int size – текущий размер таблицы

int ind – индекс текущего элемента

*Реализованные методы:*

table(int sizeT) – конструктор

virtual ~table() – деструктор

virtual void Insert(const string k, const type& d)=0 – вставка в таблицу строки с данными d и ключом k

virtual void Delete(const string k) = 0 – удаление из таблицы строки по ключу

virtual type\* Search(const string k) = 0 – поиск данных по ключу

virtual void Reset() = 0 – переместить указатель на первую строку

virtual bool IsTabEnded() const = 0 – проверка на конец

virtual void GetNext()=0 – переместить указатель на следующую строку

virtual line<type> GetCurrent() const = 0 – получить текущую строку

bool IsEmpty() const – проверка на пустоту

bool IsFull() const – проверка на полноту

int GetSize() const – текущий размер

virtual void Realloc() = 0 – перепаковка таблицы

### Класс scantable

scantable – шаблон класса, унаследованный от базового абстрактного класса table, параметризованный type, представляет собой неупорядоченную таблицу.

*Поля класса:*

line<type>\*\* stab – массив указателей на строки

*Реализованные методы:*

void Realloc()override – перепаковка таблицы

scantable(int sizeT = DEFAULTSIZE) – конструктор

~scantable() – деструктор

void Insert(const string k, const type& d) override – вставка в таблицу строки с данными d и ключом k

void Delete(const string k) override – удаление из таблицы строки по ключу

type\* Search(const string k) override – поиск данных по ключу

bool BSearch(const string k) – поиск данных по ключу, возвращает true если строка с таким ключом найдена

void Reset() override – переместить указатель на первую строку

bool IsTabEnded() const override – проверка на конец

void GetNext()override – переместить указатель на следующую строку

line<type> GetCurrent() const override – получить текущую строку

template<class t> friend ostream& operator<<(ostream& os, scantable<t>& t) – оператор вставки в поток

### Класс sortedtable

sortedtable – шаблон класса, унаследованный от базового абстрактного класса table, параметризованный type, представляет собой неупорядоченную таблицу.

*Поля класса:*

line<type>\*\* stab – массив указателей на строки

*Реализованные методы:*

int BinSearch(const string k) – бинарный поиск строки по ключу, возвращает номер строки, в которой должен стоять искомый элемент

sortedtable(const scantable<type> & t) – преобразование таблицы типа scantable к sortedtable

void Realloc()override – перепаковка таблицы

sortedtable(int sizeT = DEFAULTSIZE) – конструктор

~sortedtable() – деструктор

void Insert(const string k, const type& d) override – упорядоченная по ключу вставка в таблицу строки с данными d и ключом k

void Delete(const string k) override – удаление из таблицы строки по ключу

type\* Search(const string k) override – поиск данных по ключу

void Reset() override – переместить указатель на первую строку

bool IsTabEnded() const override – проверка на конец

void GetNext()override – переместить указатель на следующую строку

line<type> GetCurrent() const override – получить текущую строку

template<class t> friend ostream& operator<<(ostream& os, sortedtable<t>& t) – оператор вставки в поток

### Класс hashtable

hashtable – шаблон класса, унаследованный от базового абстрактного класса table, параметризованный type, представляет собой хэш-таблицу.

*Поля класса:*

List<line<type>>\* htab – массив списков из строк

*Реализованные методы:*

int hashf(const string k) – хэш-функция, возвращает номер списка, в котором должна стоять строка с данным ключом

void Realloc()override – перехэширование таблицы

hashtable(int sizeT = DEFAULTSIZE) – конструктор

~hashtable() – деструктор

void Insert(const string k, const type& d) override – вставка в таблицу строки с данными d и ключом k

void Delete(const string k) override – удаление из таблицы строки по ключу

type\* Search(const string k) override – поиск данных по ключу

void Reset() override – переместить указатель на первую строку

bool IsTabEnded() const override – проверка на конец

void GetNext()override – переместить указатель на следующую строку

line<type> GetCurrent() const override – получить текущую строку

template<class t> friend ostream& operator<<(ostream& os, hashtable<t>& t) – оператор вставки в поток

# Заключение

Была разработана программа, позволяющая хранить данные в таблицах трех видов: неупорядоченные, упорядоченные, хэш-таблицы. Реализованы поиск, вставка и удаление. Все функции тестируются с помощью Google Tests. Разработано пользовательское консольное приложение.

# Список литературы

1. Павловская Т.А. С/С++. Программирование на языке высокого уровня. - СПб.: Питер. - 2003. - 460с.
2. Кормен Т., Лейзерсон Ч., Риверст Р., Штайн К. Алгоритмы. Построение и анализ. - М.: Издательский дом "Вильямс". - 2005. - 1290с.
3. Гергель В.П. и др. Методы программирования. Учебное пособие. – Н.Новгород.: ННГУ. – 2016. – 211с.

# Приложение

## Приложение А. Программная реализация линейного односвязного списка

### **linelist.h**

#pragma once

#include <iostream>

#include "line.h"

using namespace std;

template <class DataType>

struct Node

{

DataType data;

Node\* next;

Node(DataType d = DataType(), Node<DataType>\* n = NULL)

{

data = d;

next = n;

}

Node(const Node<DataType>& node2)

{

data = node2.data;

next = NULL;

}

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

{

return (data == node2.data && next == node2.next);

}

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

{

return !(\*this == node2);

}

};

template <class DataType>

class List

{

// любые поля

Node<DataType>\* cur;

Node<DataType>\* head;

public:

List() { head = NULL;}

List(const List<DataType>& list2);

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

~List();

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);

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

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

void Clean(); // удалить все звенья

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

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

bool IsEmpty() {return head == NULL; }

void Reset() { cur = head; }

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

bool IsEnded() { return cur->next == NULL; }

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

void Inverse();

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

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

bool operator==(const List& list2) const;

};

template <class DataType>

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

{

if (list2.head)

{

Node<DataType>\* cur = new Node<DataType>(list2.head->data);

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

head = cur;

while (temp->next)

{

temp = temp->next;

cur->next = new Node<DataType>(temp->data);

cur = cur->next;

}

}

else

head = NULL;

}

template <class DataType>

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

{

if (head != list2.head)

{

Clean();

if (list2.head)

{

Node<DataType>\* cur = new Node<DataType>(list2.head->data);

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

head = cur;

while (temp->next)

{

temp = temp->next;

cur->next = new Node<DataType>(temp->data);

cur = cur->next;

}

}

else

head = NULL;

}

return \*this;

}

template <class DataType>

List<DataType>::~List()

{

Clean();

}

template <class DataType>

void List<DataType>::Clean()

{

Node<DataType>\* cur;

Node<DataType>\* temp = head;

if (head)

while (temp)

{

cur = temp->next;

delete temp;

temp = cur;

}

head = NULL;

}

template <class DataType>

int List<DataType>::GetSize()

{

int res = 0;

Node<DataType>\* cur = head;

while (cur)

{

res++;

cur = cur->next;

}

return res;

}

template <class DataType>

Node<DataType>\* List<DataType>::GetHead()

{

return head;

}

template <class DataType>

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

{

Node<DataType>\* cur1 = head;

Node<DataType>\* cur2 = list2.head;

bool res = true;

if (cur1 != NULL && cur2 != NULL && cur1 != cur2)

{

while (res && cur1->next && cur2->next)

if (cur1->data == cur2->data)

{

cur1 = cur1->next;

cur2 = cur2->next;

}

else

res = false;

if (cur1->next || cur2->next)

res = false;

}

else

if (cur1 != cur2)

res = false;

return res;

}

template <class DataType>

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

{

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

head = temp;

}

template <class DataType>

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

{

Node<DataType>\*cur = head;

if (cur)

{

while (cur->next)

cur = cur->next;

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

}

else

head = new Node<DataType>(d);

}

template <class DataType>

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

{

if (node && head)

{

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

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

}

else

if (head == NULL)

throw "";

}

template <class DataType>

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

{

Node<DataType>\* cur = head;

Node<DataType>\* temp;

if (head)

{

if (head->data == d)

{

temp = head->next;

delete head;

head = temp;

}

else

{

while (cur->next)

{

if (cur->next->data != d)

cur = cur->next;

else

{

temp = cur->next;

cur->next = temp->next;

delete temp;

}

}

}

}

}

template <class DataType>

void List<DataType>::Inverse()

{

if (head)

{

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

head->next = NULL;

Node<DataType>\* cur = head;

Node<DataType>\* p;

while (temp)

{

p = temp->next;

temp->next = cur;

cur = temp;

temp = p;

}

head = cur;

}

}

template <class DataType>

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

{

Node<DataType>\*cur = head;

while (cur)

{

if (cur->data == d)

return cur;

cur = cur->next;

}

return NULL;

}

template <class DataType>

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

{

List<DataType> res;

Node<DataType>\* cur1 = head;

if (head)

{

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

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

while (cur1->next && cur1 != node)

{

temp->next = new Node<DataType>(cur1->next->data);

temp = temp->next;

cur1 = cur1->next;

}

if (cur1 == node)

{

if (list2.head)

{

Node<DataType>\* cur2 = list2.head;

while (cur2)

{

temp->next = new Node<DataType>(cur2->data);

cur2 = cur2->next;

temp = temp->next;

}

}

while (cur1->next)

{

temp->next = new Node<DataType>(cur1->next->data);

cur1 = cur1->next;

temp = temp->next;

}

}

temp->next = NULL;

}

return res;

}

template <class DataType>

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

{

List res(\*this);

if (res.head && list2.head)

{

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

while (cur->next)

cur = cur->next;

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

while (temp)

{

cur->next = new Node<DataType>(temp->data);

cur = cur->next;

temp = temp->next;

}

cur->next = NULL;

}

else if (res.head == NULL) res = List(list2);

return res;

}

### linelist.cpp

#include "linelist.h"

## Приложение Б. Программная реализация таблиц

### line.h

#pragma once

#include <string>

using namespace std;

template <class type>

struct line {

type\* data;

string key;

line() { key = ""; data = NULL; }

line(string k, type d) { key = k; data =new type(d); }

line(string k) { key = k; data = NULL; }

line<type>& operator=(const line<type>& l);

~line() { delete data; }

line (const line<type>& l);

bool operator==(const line<type>&) const;

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

};

template <class type> line<type>::line(const line<type>& l) {

key = l.key;

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

}

template <class type> line<type>& line<type>::operator=(const line<type>& l) {

delete data;

key = l.key;

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

return \*this;

}

template <class type> bool line<type>::operator==(const line<type>& l) const {

return (key == l.key);

}

### table.h

#pragma once

#include "line.h"

#include <iostream>

#include <iomanip>

using namespace std;

#define WIDTHFIRSTCOLUMN 30

#define DEFAULTSIZE 10

template <class type>

class table {

protected:

int maxsize;

int size;

int ind;

virtual void Realloc() = 0;

public:

table(int sizeT) { maxsize = sizeT; size = 0; ind = 0; }

virtual ~table() {};

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

virtual void Delete(const string k) = 0;

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

virtual void Reset() = 0;

virtual bool IsTabEnded() const = 0;

virtual void GetNext()=0;

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

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

bool IsFull() const { return size == maxsize; }

int GetSize() const { return size; }

};

### scantable.h

#pragma once

#include "table.h"

template <class type>

class scantable :public table <type> {

protected:

line<type>\*\* stab;

void Realloc()override;

bool BSearch(const string k); //true если элемент найдет. для Insert

public:

scantable(int sizeT = DEFAULTSIZE);

~scantable();

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

void Delete(const string k) override;

type\* Search(const string k) override;

void Reset() override;

bool IsTabEnded() const override;

void GetNext()override;

line<type> GetCurrent() const override;

template<class type> friend class sortedtable; //класс sortedtable может залезать в protected поля

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

};

template <class type> scantable<type>::scantable(int sizeT = DEFAULTSIZE) : table(sizeT) {

stab = new line<type>\*[maxsize];

}

template <class type> scantable<type>::~scantable() {

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

delete stab[i];

delete[] stab;

}

template<class type> void scantable<type>::Reset() {

if (!IsEmpty())

ind = 0;

else

throw "empty table";

}

template <class type> void scantable<type>::GetNext() {

if (!IsTabEnded())

ind++;

else

Reset();

}

template<class type> bool scantable<type>::IsTabEnded() const {

return (ind >= size - 1);

}

template<class type> line<type> scantable<type>::GetCurrent() const {

if (ind <= size - 1)

return \*(stab[ind]);

else

throw "not found";

}

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

int i = 0;

while (i < size && stab[i]->key != k)

i++;

if (i == size)

throw "not found";

else

return stab[i]->data;

}

template <class type> bool scantable<type>::BSearch(const string k) {

int i = 0;

bool res = false;

while (i < size && stab[i]->key != k)

i++;

if (i != size && stab[i]->key == k)

res = true;

return res;

}

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

if (!BSearch(k))

{

if (size == maxsize)

Realloc();

stab[size] = new line<type>(k, d);

size++;

}

else

throw "repeated key";

}

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

int i = 0;

while (i < size && stab[i]->key != k)

i++;

if (i == size)

throw "not found";

else

{

size--;

delete stab[i];

stab[i] = stab[size];

}

}

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

maxsize \*= 2;

line<type>\*\* temp;

temp = new line<type>\*[maxsize];

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

temp[i] = new line<type>;

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

temp[i] = stab[i];

delete[] stab;

stab = temp;

}

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

if (t.size != 0)

{

t.Reset();

os << setw(WIDTHFIRSTCOLUMN) << left << t.GetCurrent().key << \*(t.GetCurrent().data) << endl;

while (!t.IsTabEnded())

{

t.GetNext();

os << setw(WIDTHFIRSTCOLUMN) << left << t.GetCurrent().key << \*(t.GetCurrent().data) << endl;

}

}

return os;

}

### sortedtable.h

#pragma once

#include "table.h"

#include "scantable.h"

template <class type>

class sortedtable :public table <type> {

protected:

int BinSearch(const string k);

void Realloc()override;

line<type>\*\* stab;

public:

sortedtable(int sizeT = DEFAULTSIZE);

~sortedtable();

sortedtable(const scantable<type> & t); // преобрпзование таблицы типа scantable к sortedtable

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

void Delete(const string k) override;

type\* Search(const string k) override;

void Reset() override;

bool IsTabEnded() const override;

void GetNext()override;

line<type> GetCurrent() const override;

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

};

template <class type> sortedtable<type>::sortedtable(int sizeT = DEFAULTSIZE) : table(sizeT) {

stab = new line<type>\*[maxsize];

}

template <class type> sortedtable<type>::~sortedtable() {

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

delete stab[i];

delete[] stab;

}

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

stab = new line<type>\*[maxsize];

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

stab[i] = new line<type>;

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

Insert(t.stab[i]->key, \*(t.stab[i]->data));

}

template <class type> void sortedtable<type>::Reset() {

if (!IsEmpty())

ind = 0;

else

throw "empty table";

}

template <class type> void sortedtable<type>::GetNext() {

if (!IsTabEnded())

ind++;

else

Reset();

}

template<class type> bool sortedtable<type>::IsTabEnded() const {

return (ind >= size - 1);

}

template<class type> line<type> sortedtable<type>::GetCurrent() const {

if (ind <= size - 1)

return \*(stab[ind]);

else

throw "not found";

}

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

int st = 0, fn = size - 1, mid;

while (st <= fn)

{

mid = (st + fn) / 2;

if (k > stab[mid]->key)

st = mid + 1;

else fn = mid - 1;

}

return st;

}

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

int pos = BinSearch(k);

if ((pos < size && stab[pos]->key != k) || pos==size)

{

if (size == maxsize)

Realloc();

for (int i = size - 1; i >= pos; i--)

stab[i + 1] = stab[i];

size++;

stab[pos] = new line<type>(k, d);

}

else

throw "repeated key";

}

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

int pos = BinSearch(k);

if (pos < size && stab[pos]->key == k)

return stab[pos]->data;

else

throw "not found";

}

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

int pos = BinSearch(k);

if (pos<size && stab[pos]->key == k)

{

size--;

delete stab[pos];

for (int i = pos; i < size; i++)

stab[i] = stab[i + 1];

}

else

throw "not found";

}

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

maxsize \*= 2;

line<type>\*\* temp;

temp = new line<type>\*[maxsize];

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

temp[i] = new line<type>;

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

temp[i] = stab[i];

delete[] stab;

stab = temp;

}

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

if (t.size != 0)

{

t.Reset();

os << setw(WIDTHFIRSTCOLUMN) << left << t.GetCurrent().key << \*(t.GetCurrent().data) << endl;

while (!t.IsTabEnded())

{

t.GetNext();

os << setw(WIDTHFIRSTCOLUMN) << left << t.GetCurrent().key << \*(t.GetCurrent().data) << endl;

}

}

return os;

}

### hashtable.h

#pragma once

#include "table.h"

#include "linelist.h"

#define LOADFACTOR 0.65

template <class type>

class hashtable :public table <type> {

protected:

List<line<type>>\* htab;

void Realloc()override;

int hashf(const string k);

public:

hashtable(int sizeT = DEFAULTSIZE);

~hashtable();

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;

line<type> GetCurrent() const override;

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

};

template <class type> hashtable<type>::hashtable(int sizeT = DEFAULTSIZE) : table(sizeT) {

htab = new List<line<type>>[maxsize];

}

template <class type> hashtable<type>::~hashtable(){

delete[] htab;

}

template <class type> void hashtable<type>::Reset() {

int i = 0;

if (size != 0)

{

while (htab[i].IsEmpty())

i++;

ind = i;

htab[ind].Reset();

}

else

throw "empty table";

}

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

bool res = false;

int i = ind;

if (size != 0)

{

if (htab[ind].IsEnded())

{

i++;

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

i++;

if (i == maxsize)

res = true;

}

}

else res = true;

return res;

}

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

if (!htab[ind].IsEnded())

htab[ind].GetNext();

else

{

ind++;

while (ind<maxsize && htab[ind].IsEmpty())

ind++;

if (ind==maxsize)

Reset();

else

htab[ind].Reset();

}

}

template <class type> line<type> hashtable<type>::GetCurrent() const {

return htab[ind].GetCur()->data;

}

template <class type> int hashtable<type>::hashf(const string k) {

int res = 0;

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

res += int(k[i]);

res %= maxsize;

return res;

}

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

int s = hashf(k);

Node<line<type>>\* el = htab[s].Search(line<type>(k));

if (el != NULL)

return (el->data).data;

else

throw "not found";

}

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

int i = 0;

Node<line<type>>\* el = htab[hashf(k)].Search(line<type>(k));

if (el == NULL)

{

if (htab[hashf(k)].IsEmpty())

{

size++;

if (double(size) / maxsize > LOADFACTOR)

Realloc();

}

htab[hashf(k)].InsertToHead(line<type>(k, d));

}

else

throw "repeated key";

}

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

int s = hashf(k);

Node<line<type>>\* el = htab[s].Search(line<type>(k));

if (el != NULL)

{

htab[s].Delete(line<type>(k));

if (htab[s].IsEmpty())

size--;

}

else

throw "not found";

}

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

int nmaxsize = maxsize \* 2;

List<line<type>>\* temp;

temp = new List<line<type>>[nmaxsize];

Reset();

temp[hashf(GetCurrent().key)].InsertToHead(GetCurrent());

while (!IsTabEnded())

{

GetNext();

temp[hashf(GetCurrent().key)].InsertToHead(GetCurrent());

}

delete[] htab;

maxsize = nmaxsize;

htab = temp;

}

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

if (t.size != 0)

{

t.Reset();

os << setw(WIDTHFIRSTCOLUMN) << left << t.GetCurrent().key << \*(t.GetCurrent().data) << endl;

while (!t.IsTabEnded())

{

t.GetNext();

os << setw(WIDTHFIRSTCOLUMN) << left << t.GetCurrent().key << \*(t.GetCurrent().data) << endl;

}

}

return os;

}

## Приложение В. Программная реализация приложения

### main.cpp

#include "polynom.h"

#include "hashtable.h"

#include "sortedtable.h"

#include <iostream>

#include <sstream>

char main\_menu()

{

system("cls");

char res;

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

cout << "2. Add polynom" << endl;

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

cout << "4. Find polynom" << endl;

cout << endl;

cout << "Operations with polynoms: "<< endl;

cout << "5. Addition" << endl;

cout << "6. Substraction" << endl;

cout << "7. Multiplication" << endl;

cout << endl;

cout << "8. Exit" << endl;

cout << endl;

cout << "Choose menu item: ";

cin >> res;

return res;

}

void main() {

char pmenu;

scantable<polynom> scant;

sortedtable<polynom> sortt;

hashtable<polynom>hasht;

polynom p, res;

string polynom, yn;

string k, k1;

stringstream b;

do {

if (yn == "1")

{

system("cls");

b << res;

k = b.str();

try

{

scant.Insert(k, res);

sortt.Insert(k, res);

hasht.Insert(k, res);

}

catch (...)

{

cout << "Repeated key" << endl << endl;

}

}

yn = "";

pmenu = main\_menu();

switch (pmenu)

{

case '1':

{

system("cls");

cout << endl;

cout << "Scantable" << endl << scant << endl;

cout << "Sortedtable" << endl << sortt << endl;

cout << "Hashtable" << endl << hasht << endl;

system("pause");

break;

}

case '2':

{

system("cls");

cout << "Add polynom" << endl << "Enter the polynom: ";

cin >> polynom;

cout << endl;

p = polynom;

k = polynom;

try

{

scant.Insert(k, p);

sortt.Insert(k, p);

hasht.Insert(k, p);

}

catch (...)

{

cout << "Repeated key" << endl << endl;

}

system("pause");

break;

}

case '3':

{

system("cls");

cout << "Delete polynom" << endl << "Enter the key: ";

cin >> k;

try

{

scant.Delete(k);

sortt.Delete(k);

hasht.Delete(k);

}

catch (...)

{

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

}

system("pause");

break;

}

case '4':

{

system("cls");

cout << "Find polynom" << endl<< "Enter the key: ";

cin >> k;

try

{

cout << \*(scant.Search(k)) << endl;

cout << \*(sortt.Search(k)) << endl;

cout << \*(hasht.Search(k)) << endl;

}

catch (...)

{

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

}

system("pause");

break;

}

case '5':

{

system("cls");

cout << "Enter the key of the first polynom: ";

cin >> k;

cout << "Enter the key of the second polynom: ";

cin >> k1;

try

{

res = \*(sortt.Search(k)) + \*(sortt.Search(k1));

cout << endl << \*(sortt.Search(k)) << '+' << \*(sortt.Search(k1)) << '=' << res << endl;

cout << endl << "Save polynom?" << endl << " 1) Yes" << endl << " 2) No" << endl << "Your choice: " << endl;

cin >> yn;

}

catch (...)

{

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

}

system("pause");

break;

}

case '6':

{

system("cls");

cout << "Enter the key of the first polynom: ";

cin >> k;

cout << "Enter the key of the second polynom: ";

cin >> k1;

try

{

res = \*(sortt.Search(k)) - \*(sortt.Search(k1));

cout << \*(sortt.Search(k)) << '-' << \*(sortt.Search(k1)) << '=' << res << endl;

cout << endl << "Save polynom?" << endl << " 1) Yes" << endl << " 2) No" << endl << "Your choice: " << endl;

cin >> yn;

}

catch (...)

{

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

}

system("pause");

break;

}

case '7':

{

system("cls");

cout << "Enter the key of the first polynom: ";

cin >> k;

cout << "Enter the key of the second polynom: ";

cin >> k1;

try

{

res = \*(sortt.Search(k)) \* \*(sortt.Search(k1));

cout << endl << \*(sortt.Search(k)) << '\*' << \*(sortt.Search(k1)) << '=' << res << endl;

cout << endl << "Save polynom?" << endl << " 1) Yes" << endl << " 2) No" << endl << "Your choice: " << endl;

cin >> yn;

}

catch (...)

{

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

}

system("pause");

break;

}

case '8': break;

}

} while (pmenu != '8');

}