МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

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**

**ЛАБОРАТОРНАЯ РАБОТА**

на тему:

**«Реализация и использование различных структур таблиц для хранения и обработки полиномов»**

**Выполнил(а):** студент(ка) группы 3822Б1ФИ2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Коробейников А.П./

Подпись

**Проверил:** к.т.н, доцент каф. ВВиСП

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Кустикова В.Д./

Подпись

Нижний Новгород  
2024

**Содержание**

[Введение 3](#_Toc2213)

[1 Постановка задачи 4](#_Toc32738)

[2 Руководство пользователя 5](#_Toc2523)

[2.1 Приложение для демонстрации работы таблиц 5](#_Toc26449)

[3 Руководство программиста 7](#_Toc15741)

[3.1 Описание алгоритмов 7](#_Toc3391)

[3.1.1 Просматриваемая Таблица 7](#_Toc3421)

[3.1.2 Отсортированная таблица 8](#_Toc19523)

[3.1.3 Хеш-таблица открытого перемешивания 11](#_Toc27003)

[3.2 Описание программной реализации 15](#_Toc32008)

[3.2.1 Описание класса TabRecord 15](#_Toc22004)

[3.2.2 Описание класса Table 16](#_Toc20704)

[3.2.3 Описание класса ScanTable 18](#_Toc19901)

[3.2.4 Описание класса SortedTable 19](#_Toc21048)

[3.2.5 Описание класса HashTable 20](#_Toc18376)

[Заключение 23](#_Toc16769)

[Литература 24](#_Toc23242)

[Приложения 25](#_Toc31366)

[Приложение А. Реализация класса TabRecord 25](#_Toc7101)

[Приложение Б. Реализация класса Table 25](#_Toc11592)

[Приложение В. Реализация класса ScanTable 27](#_Toc30152)

[Приложение Г. Реализация класса SortedTable 29](#_Toc20177)

[Приложение Е. Реализация класса HashTable 32](#_Toc28230)

[Приложение Ж. Программа демонстрации таблиц полиномов 35](#_Toc2472)

# Введение

Лабораторная работа направлена на изучение новой структуры данных － таблица. В ходе работы мы ознакомимся с тремя видами таблица: просматриваемая таблица, упорядоченная таблица и хеш таблица. Мы подробно рассмотрим каждую из них, поймём какие достоинства и недостатки имеет каждая из них.

# Постановка задачи

**Цель:**

Разработка и реализация библиотеки, которая будет содержать в себе новую для нас структуру данных － таблицу.

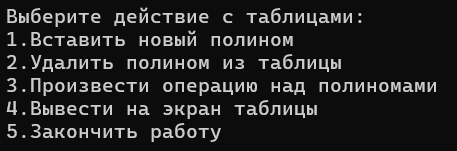
**Задачи:**

1. Изучить всю теорию о структуре данных － таблице. Узнать что это, определиться с терминологией.
2. Написать необходимые для хранения различных видов таблиц классы.
3. Написать тесты, для этих классов, чтобы убедиться в корректности работы.
4. Написать программу, которая будет подробно демонстрировать работу каждого вида таблицы.

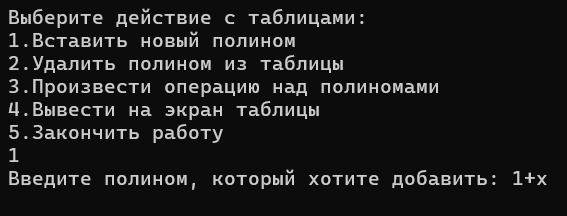
# Руководство пользователя

## Приложение для демонстрации работы таблиц

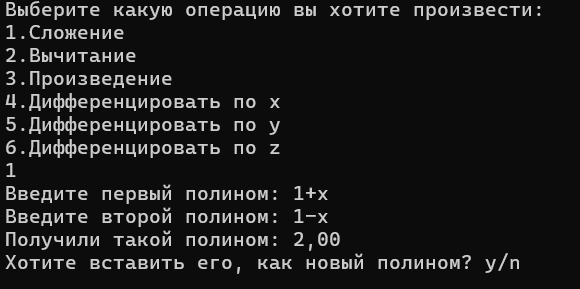
1. Запустите приложение sample\_table.exe. В результате появится окно, показанное ниже (рис. 1).



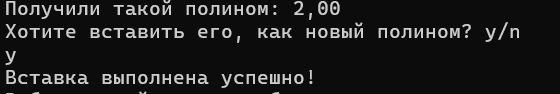
1. Начальный вывод программы
2. Сначала следует вставить новые полиномы в таблицу, поскольку она пустая. Делать это, следуя инструкциям на экране (рис. 2).



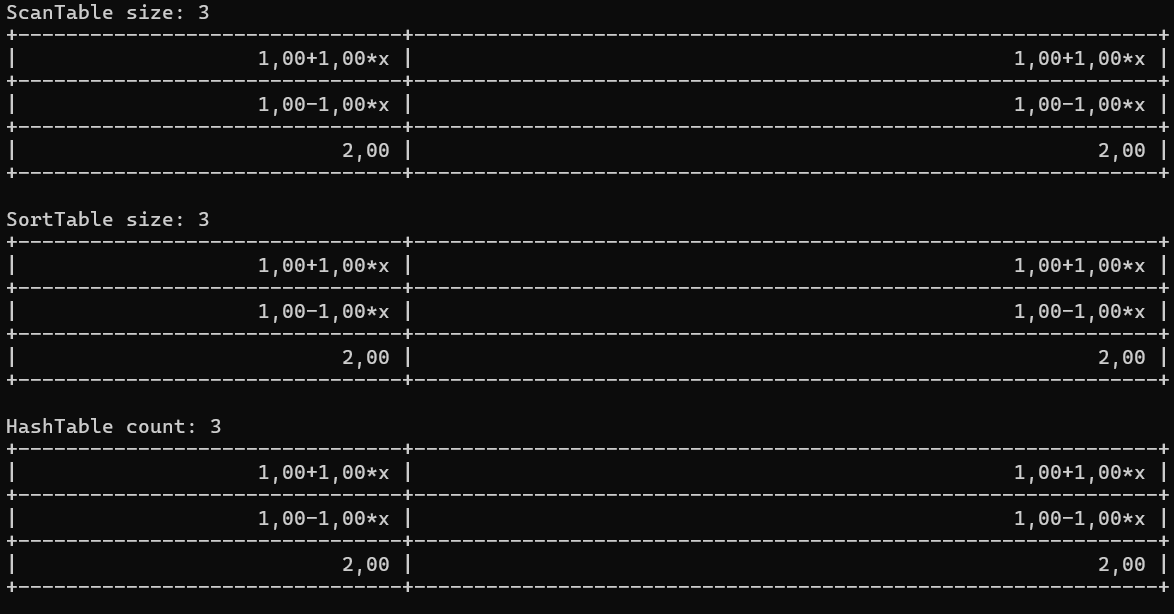
1. Вставка полинома
2. После того, как вставили новые полиномы, можно удалять их и производить над ними операции, следуя инструкция на экране. Приведём пример сложение двух полиномов. Нужно вводить полиномы, которые имеются у нас в таблице. Пример показан ниже (рис. 3).



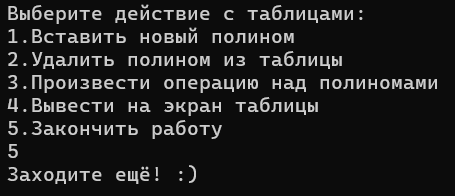
1. Инструкция к сложению элементов
2. После того, как вы произведёте операцию, вы можете выбрать, добавлять полученный результат в таблицу или нет (рис. 4).



1. Добавление результата в таблицу
2. После того, как вы совершите нужные вам операции. Вы можете посмотреть на результат, показав таблицы на экране (рис. 5).



1. Вывод таблиц
2. Также вы в любую секунду можете завершить работу (рис. 6).



1. Завершение работы

# Руководство программиста

## Описание алгоритмов

### Просматриваемая таблица

Просматриваемые таблицы － это таблицы, которые хранят записи в неупорядоченном виде и не имеют к записям прямого доступа по ключу.

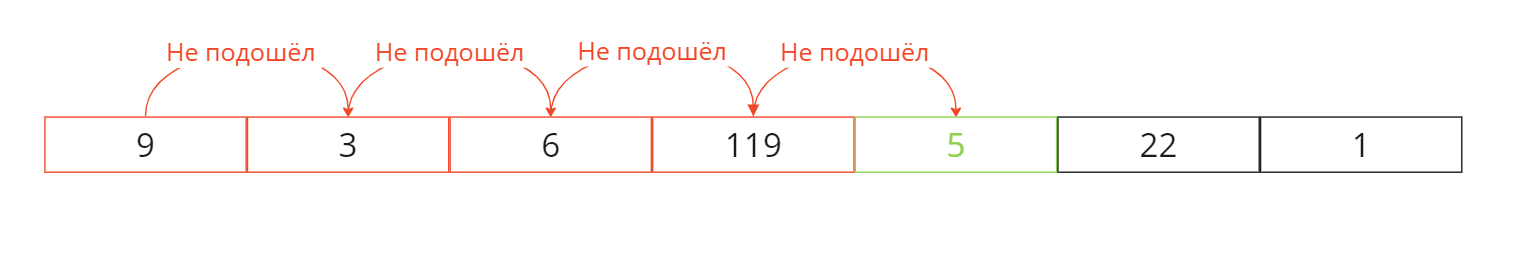
Работа с этими таблицами основана на простом принципе последовательного перебора всех записей таблицы для выполнения операций поиска, вставки и удаления.

**1. Поиск:**

* Для поиска записи в просматриваемой таблице начинается с первого элемента.
* Каждая запись таблицы последовательно сравнивается с ключом поиска до тех пор, пока не будет найдена запись с совпадающим ключом или пока не будут просмотрены все записи таблицы.
* Если элемент с заданным ключом найден, возвращается указатель на этот элемент; в противном случае возвращается значение nullptr, указывающее на отсутствие элемента в таблице.

Пример:

Поиск записи с ключом 5 (рис. 7)



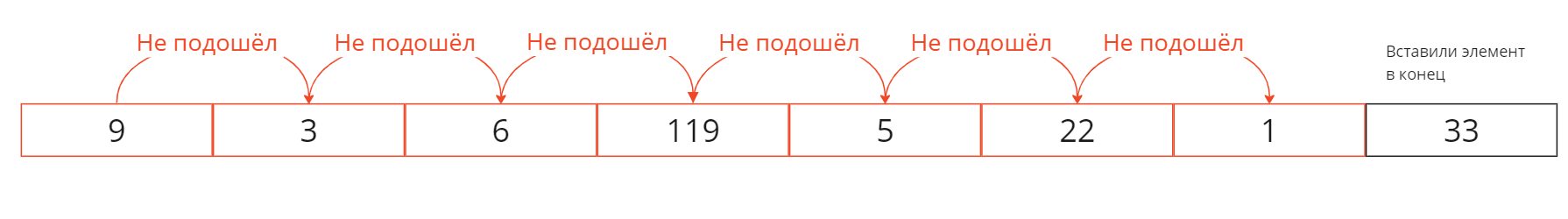
1. Поиск в просматриваемой таблице

**2. Вставка:**

* Вставка записи в просматриваемую таблицу также начинается с первой записи.
* Каждая запись таблицы последовательно сравнивается с ключом вставляемой записи до тех пор, пока не будет найдена запись с ключом, равным или большим ключу вставляемой записи, или пока не будет достигнут конец таблицы.
* Если запись с таким ключом уже существует в таблице, вставка не производится; в противном случае вставляемая запись вставляется в таблицу на место текущей записи или в конец таблицы, если такой элемент не найден.

Пример:

Вставка записи с ключом 33 (рис. 8).



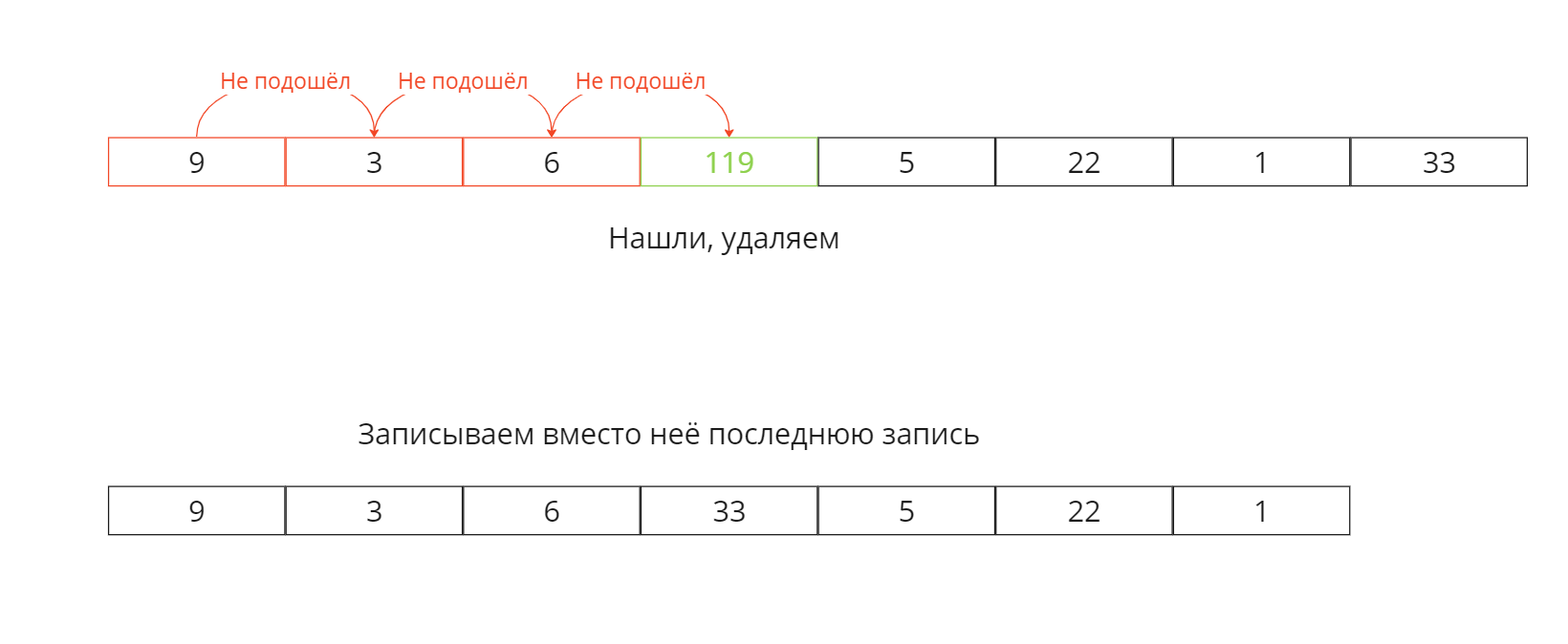
1. Вставка элемента в просматриваемую таблицу

**3. Удаление:**

* Удаление записи из таблицы линейного поиска также начинается с первого элемента.
* Каждая запись таблицы последовательно сравнивается с ключом удаляемой записи до тех пор, пока не будет найдена запись с совпадающим ключом или пока не будет достигнут конец таблицы.
* Если запись с заданным ключом найдена, она удаляется из таблицы, а на её место записывается последняя запись.

Пример:

Удаление записи с ключом 119 (рис. 9).



1. Удаление элемента в просматриваемой таблице

### Отсортированная таблица

Просматриваемые таблицы － это таблицы, которые всё также не имеют к записям прямого доступа по ключу, однако данные в них упорядочены, что позволяет нам увеличить скорость работы с ними.

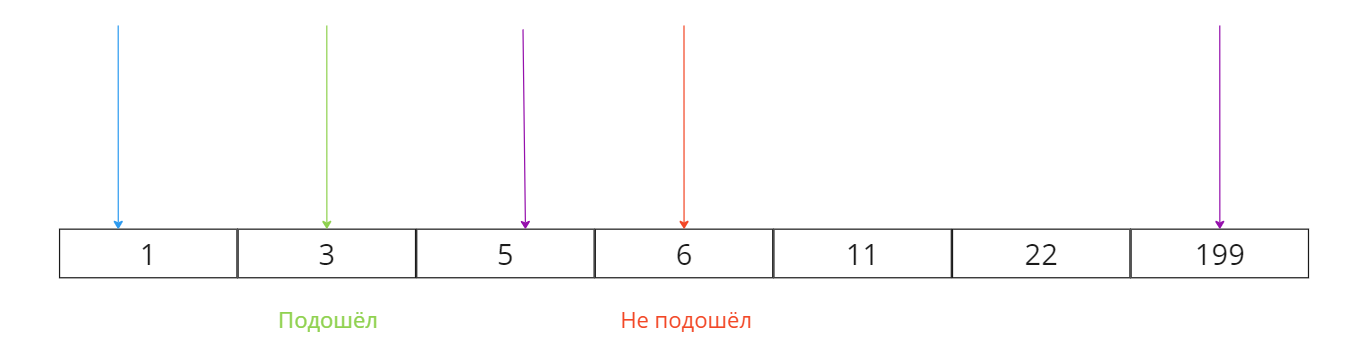
Работа с отсортированными таблицами применяет бинарный поиск для эффективного выполнения операций поиска, вставки и удаления элементов в отсортированной последовательности.

**1. Поиск:**

* Для поиска записи в отсортированной таблице с бинарным поиском сначала определяется середина отсортированной последовательности.
* Если ключ поиска совпадает с ключом в середине последовательности, запись найдена.
* Если ключ поиска меньше ключа в середине, поиск продолжается в левой половине последовательности; в противном случае поиск продолжается в правой половине.
* Процесс повторяется, пока не будет найдена запись с заданным ключом или пока не будет определено, что такой записи нет в таблице.

Пример:

Поиск записи с ключом 3 (рис. 10). Голубые стрелки - это стрелки левых границ поиска, фиолетовые - правых. Красные стрелки указывают на не подошедшие нам серединные записи, зелёная стрелка указывает на искомую запись.



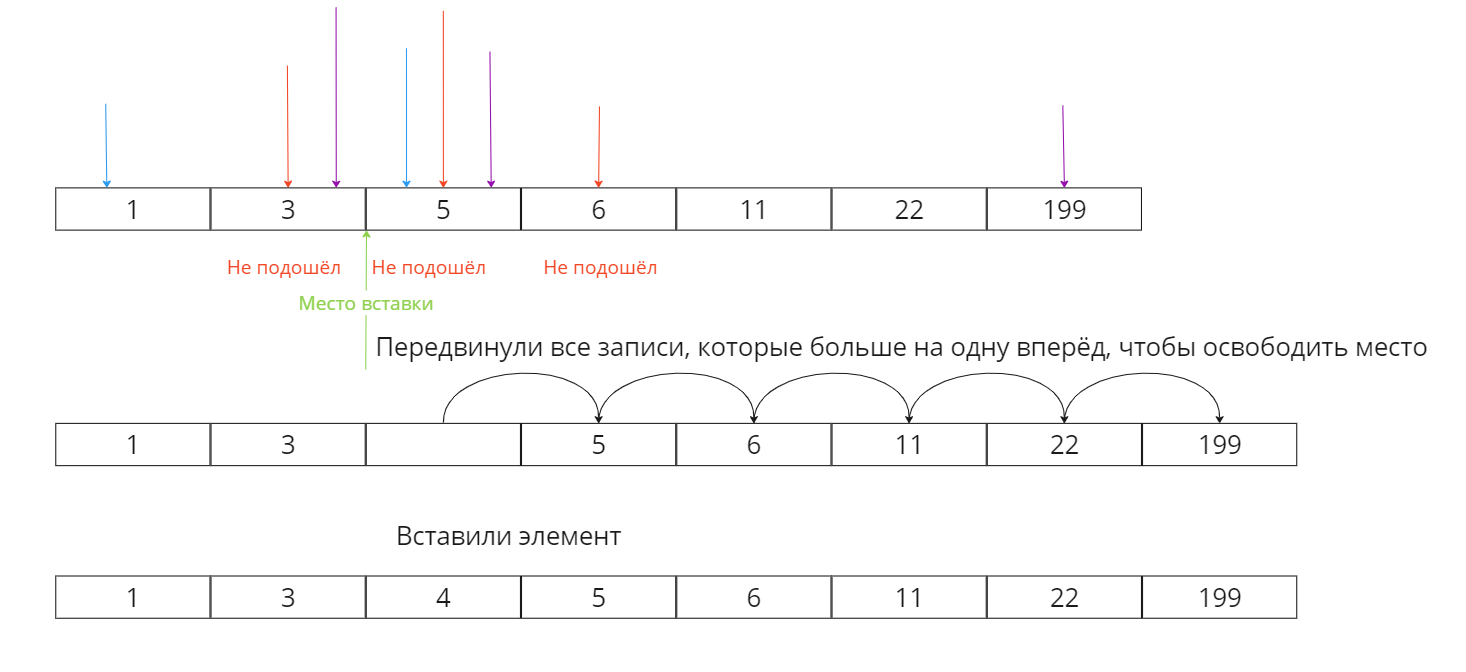
1. Поиск в отсортированной таблице

**2. Вставка:**

* Для вставки новой записи в отсортированную таблицу сначала определяется место для вставки с помощью бинарного поиска.
* Затем все записи, начиная с найденной позиции и до конца таблицы, сдвигаются на одну позицию вправо, чтобы освободить место для новой записи.
* Новая запись вставляется на найденное место.

Пример:

Вставка записи с ключом 4 (рис. 11). Обозначения те же. Теперь для большей ясности при изменении позиции стрелки увеличиваются.



1. Вставка в отсортированную таблицу

**3. Удаление:**

* Для удаления записи из отсортированной таблицы сначала запись находится с помощью бинарного поиска.
* После нахождения записи его удаляют, а все последующие записи сдвигаются на одну позицию влево, чтобы заполнить пустую позицию.
* Таким образом, отсортированность таблицы сохраняется после удаления записи.

Пример: Удаление записи с ключом 3 (рис. 12). Обозначения те же.



1. Удаление из отсортированной таблицы

### Хеш-таблица

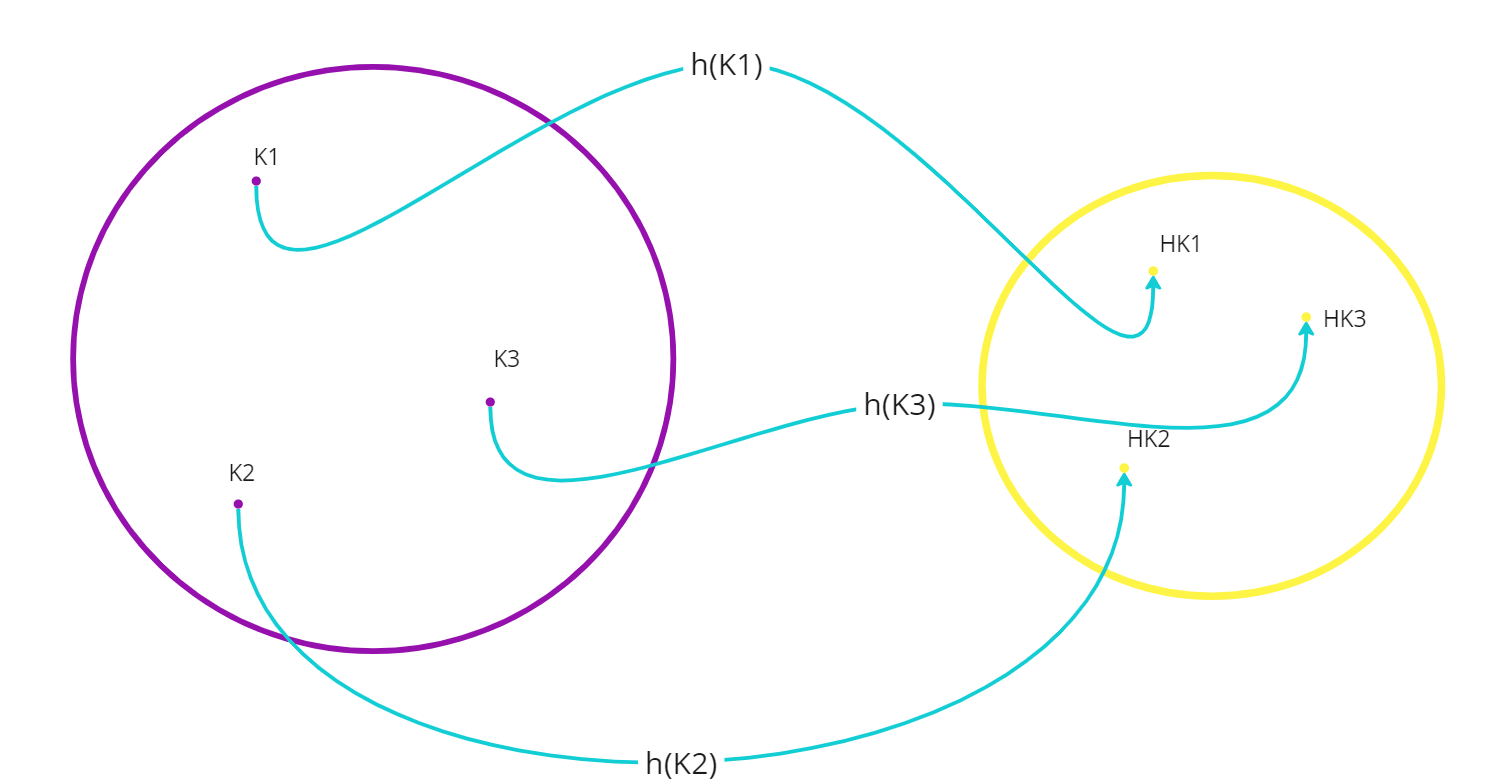
Хеш таблицы － это таблицы которые хранят записи в неупорядоченном виде, однако имеют к записям доступа по ключу через хеш функцию.

Алгоритм работы хеш-таблицы с открытым перемешиванием (или просто открытой адресацией) предполагает разрешение коллизий посредствам поиска свободной ячейки в таблице для вставки элемента.

**1. Хеширование:**

* Для взаимодействия записи с таблицей сначала вычисляется хеш-значение ключа записи с помощью хеш-функции.
* Хеш-функция преобразует ключ в индекс таблицы, в котором элемент должен быть размещен.

Для наглядности на рис. 13 представлены множество ключей и множество хеш-значений ключей, которые получаются с помощью хеш-функции. На рисунке множество хеш-значений специально нарисовано меньше, чем множество ключей, рассмотрим следующий пункт.



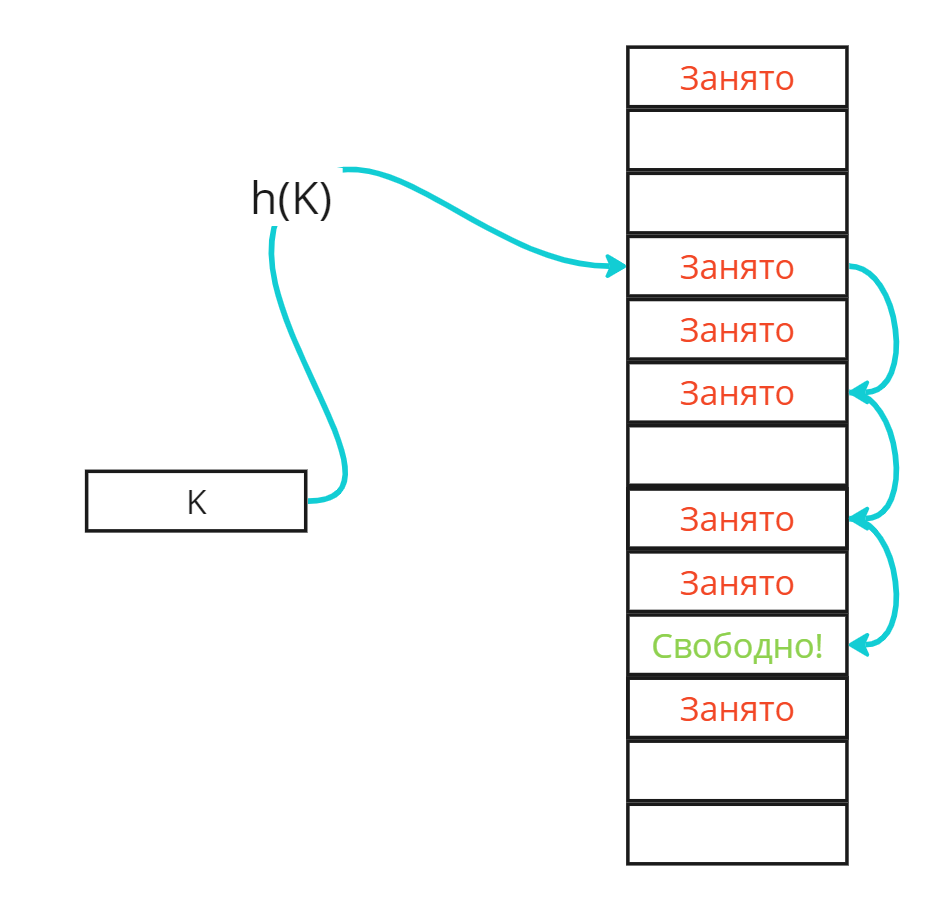
1. Значение ключе и хеш-значений ключей

**2. Разрешение коллизий:**

* Если запись пытается вставиться в ячейку, которая уже занята другим элементом (коллизия), применяется метод открытого перемешивания для нахождения свободной ячейки.
* Один из наиболее распространенных методов открытого перемешивания － это метод линейного пробирования, при котором выполняется поиск следующей свободной ячейки посредствам последовательного пробирования по индексам таблицы.
* Другие методы включают квадратичное пробирование и двойное хеширование, каждый из которых имеет свои преимущества и недостатки.

Пример:

На рис. 14 представлен один из самых примитивных способов открытого перемешивания. Мы получили хеш-значение по ключу, но по этому значению ячейка в таблице занята, тогда мы ищем следующую прибавляя к полученному хэш-значению некоторый шаг, пока не дойдём до свободной ячейки.



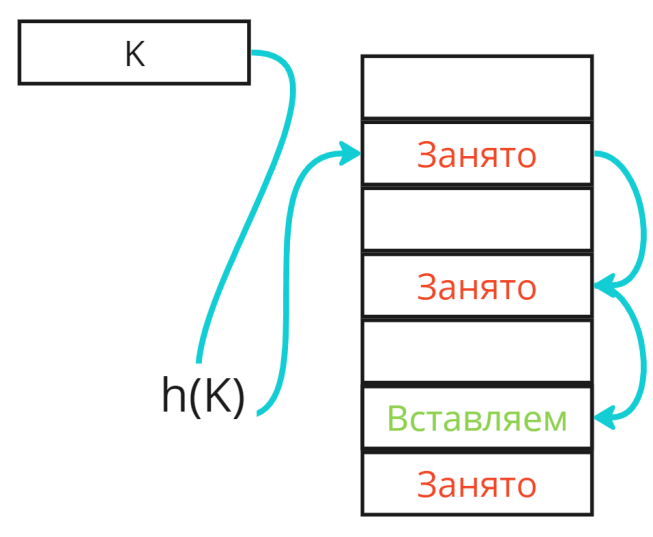
1. Разрешение коллизии

**3. Вставка:**

* После вычисления хеш-значения элемента и разрешения возможной коллизии, элемент вставляется в найденную свободную ячейку.

Пример:

С помощью метода открытого перемешивания нашли свободную ячейку и вставили туда нашу запись (рис. 15).



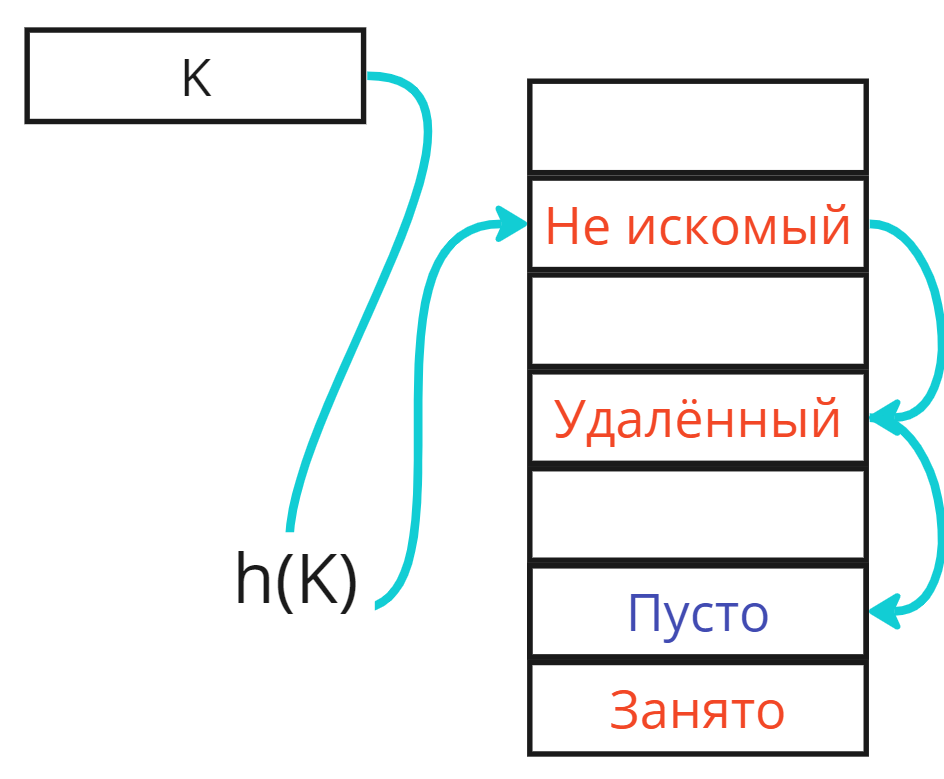
1. Вставка в хеш-таблицу

**4. Поиск:**

* Для поиска записи сначала вычисляется хеш-значение ключа.
* Затем происходит последовательный поиск по ячейкам таблицы с использованием той же хеш-функции до тех пор, пока не будет найден запись с соответствующим ключом или пока не будет обнаружена пустая ячейка, что означает отсутствие записи в таблице.

Пример:

Получили хеш-значение по ключу. Начинаем поиск, дошли до пустой ячейки значит запись в таблице отсутствует (рис. 16).



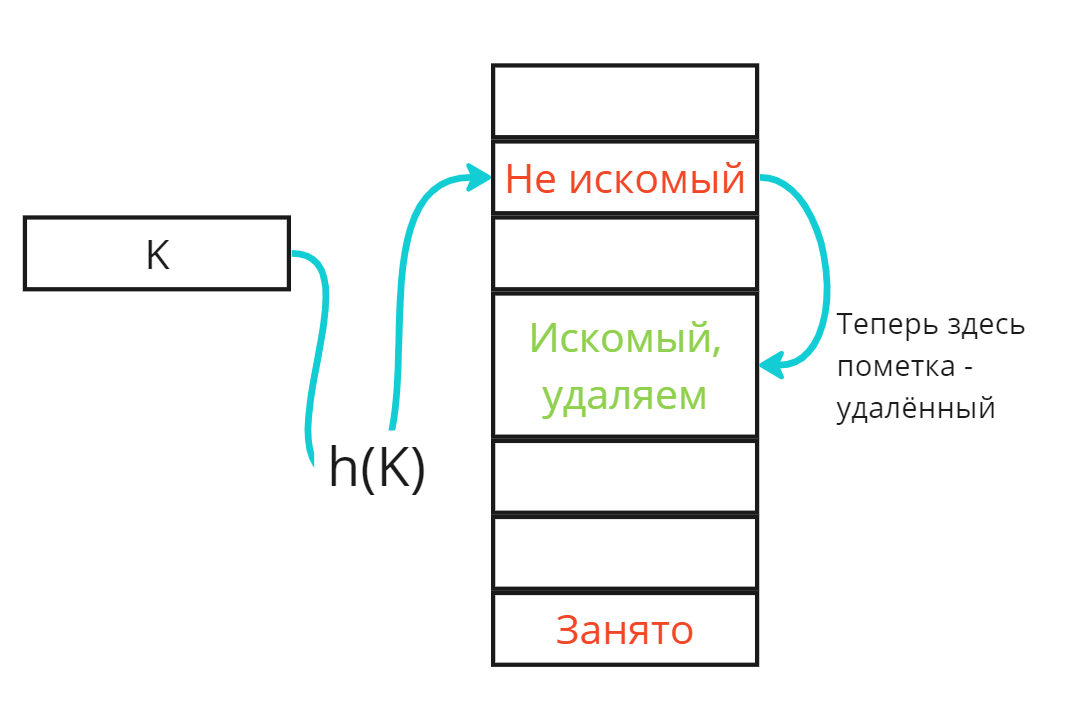
1. Поиск в хеш-таблице

**5. Удаление:**

* Для удаления записи сначала она находится в таблице с помощью поиска по ключу.
* После нахождения записи она удаляется из таблицы путём пометки соответствующей ячейки как свободной или удаления записи, в зависимости от конкретной реализации.

Пример:

На рис. 17 при поиске, мы нашли нужный элемент, затем удалили его и оставили пометку, что запись была удалена.



1. Удаление в хеш-таблице

## Описание программной реализации

### Описание класса TabRecord

template <class TKey, class TData>

struct TabRecord {

TKey key;

TData\* data;

TabRecord();

TabRecord(const TKey& \_key, TData\* \_data);

TabRecord(const TabRecord<TKey, TData>& tr);

~TabRecord();

const TabRecord<TKey, TData>& operator=(const TabRecord<TKey, TData>& tr);

};

**Назначение:** представление записи таблицы.

**Поля:**

key – ключ записи.

data – указатель на данные таблицы.

**Конструкторы:**

**TabRecord();**

Назначение: конструктор по умолчанию.

**TabRecord(const TKey& \_key, TData\* \_data);**

Назначение: конструктор с параметром.

Входные параметры: \_key – ключ, \_data – указатель на данные.

**TabRecord(const TabRecord<TKey, TData>& tr);**

Назначение: конструктор копирования.

Входные параметры: tr – копируемый объект.

**~TabRecord();**

Назначение: деструктор.

**Методы:**

**const TabRecord<TKey, TData>& operator=(const TabRecord<TKey, TData>& tr);**

Назначение: оператор присваивания.

Входные параметры: tr – присваиваемы объект.

Выходные параметры: ссылка на себя.

### Описание класса Table

template <class TKey, class TData>

class Table {

protected:

int count;

int max\_size;

int curr\_pos;

public:

Table();

virtual void insert(const TKey& \_key, TData\* \_data) = 0;

virtual void remove(const TKey& \_key) = 0;

virtual TabRecord<TKey, TData>\* find(const TKey& \_key) = 0;

virtual TabRecord<TKey, TData>\* operator[](const TKey& \_key) = 0;

bool full() const noexcept;

bool empty() const noexcept;

bool ended() const noexcept;

virtual bool reset() noexcept;

virtual bool next() noexcept;

int get\_size() const noexcept;

int get\_max\_size() const noexcept;

};

**Назначение:** абстрактное представление таблицы.

**Поля:**

count – количество записей в таблице.

max\_size – максимальное число записей в таблице.

curr\_pos – индекс текущей записи в таблице.

**Конструкторы:**

**Table();**

Назначение: конструктор с параметром.

Входные параметры: отсутствуют.

**Методы:**

**virtual void insert(const TKey& \_key, TData\* \_data) = 0;**

Назначение: вставка записи в таблицу.

Входные параметры: \_key – ключ, \_data – указатель на данные.

**virtual void remove(const TKey& \_key) = 0;**

Назначение: удаление записи из таблицы по ключу.

Входные параметры: \_key – ключ.

**virtual TabRecord<TKey, TData>\* find(const TKey& \_key) = 0;**

Назначение: поиск записи в таблице по ключу.

Входные параметры: \_key – ключ.

Выходные параметры: указатель на запись.

**virtual TabRecord<TKey, TData>\* operator[](const TKey& \_key) = 0;**

Назначение: поиск записи в таблице по ключу.

Входные параметры: \_key – ключ.

Выходные параметры: указатель на запись.

**bool full() const noexcept;**

Назначение: проверка, заполнена ли таблица.

Выходные параметры: true или false.

**bool empty() const noexcept;**

Назначение: проверка, пуста ли таблица.

Выходные параметры: true или false.

**bool ended() const noexcept;**

Назначение: проверка, является ли текущая запись в таблице последней.

Выходные параметры: true или false.

**virtual bool reset() noexcept;**

Назначение: переход в начало таблицы.

Выходные параметры: true или false.

**virtual bool next() noexcept;**

Назначение: переход на следующую запись таблицы.

Выходные параметры: true или false.

**int get\_size() const noexcept;**

Назначение: получение числа записей в таблице.

Выходные параметры: количество записей в таблице.

**int get\_max\_size() const noexcept;**

Назначение: получение максимального числа записей в таблице.

Выходные параметры: максимальное количество записей в таблице.

### Описание класса ScanTable

template <class TKey, class TData>

class ScanTable : public Table<TKey, TData> {

protected:

TabRecord<TKey, TData>\*\* recs;

public:

ScanTable(int \_max\_size = 100);

ScanTable(const ScanTable<TKey, TData>& st);

virtual ~ScanTable();

virtual void insert(const TKey& \_key, TData\* \_data);

virtual void remove(const TKey& \_key);

virtual TabRecord<TKey, TData>\* find(const TKey& key);

TabRecord<TKey, TData>\* operator[](const TKey& \_key);

friend ostream& operator<<(ostream& os, const ScanTable& table);

};

**Назначение:** представление таблицы поиска.

**Поля:**

recs – память под указатели на записи.

**Конструкторы:**

**ScanTable(int \_max\_size = 100);**

Назначение: конструктор с параметром.

Входные параметры: \_max\_size – максимальный размер таблицы.

**ScanTable(const ScanTable<TKey, TData>& st);**

Назначение: конструктор копирования.

Входные параметры: st – копируемый объект.

**virtual ~ScanTable();**

Назначение: деструктор.

**Методы:**

**virtual void insert(const TKey& \_key, TData\* \_data);**

Назначение: вставка записи в таблицу.

Входные параметры: \_key – ключ, \_data – указатель на данные.

**virtual void remove(const TKey& \_key);**

Назначение: удаление записи из таблицы по ключу.

Входные параметры: \_key – ключ.

**virtual TabRecord<TKey, TData>\* find(const TKey& key);**

Назначение: поиск записи в таблице по ключу.

Входные параметры: \_key – ключ.

Выходные параметры: указатель на запись.

**TabRecord<TKey, TData>\* operator[](const TKey& \_key);**

Назначение: поиск записи в таблице по ключу.

Входные параметры: \_key – ключ.

Выходные параметры: указатель на запись.

**friend std::ostream& operator<<(std::ostream& os, const ScanTable& table);**

Назначение: вывод таблицы.

Входные параметры: os – поток вывода, table – печатаемая таблица.

Выходные параметры: поток вывода.

### Описание класса SortedTable

template <class TKey, class TData>

class SortedTable : public ScanTable<TKey, TData> {

private:

void sort();

public:

SortedTable(int \_max\_size = 100);

SortedTable(const ScanTable<TKey, TData>& st);

SortedTable(const SortedTable<TKey, TData>& srt);

TabRecord<TKey, TData>\* find(const TKey& key);

TabRecord<TKey, TData>\* operator[](const TKey& \_key);

void insert(const TKey& \_key, TData\* \_data);

void remove(const TKey& \_key);

friend ostream& operator<<(ostream& os, const SortedTable& table);

};

**Назначение:** представление отсортированной таблицы

**Конструкторы:**

**SortedTable(int \_max\_size = 100);**

Назначение: конструктор с параметром.

Входные параметры: \_max\_size – максимальный размер таблицы.

**SortedTable(const ScanTable<TKey, TData>& st);**

Назначение: конструктор преобразования типов.

Входные параметры: st – копируемый объект.

**SortedTable(const SortedTable<TKey, TData>& srt);**

Назначение: конструктор копирования.

Входные параметры: st – копируемый объект.

**Методы:**

**TabRecord<TKey, TData>\* find(const TKey& key);**

Назначение: поиск записи в таблице по ключу.

Входные параметры: \_key – ключ.

Выходные параметры: указатель на запись.

**TabRecord<TKey, TData>\* operator[](const TKey& \_key);**

Назначение: поиск записи в таблице по ключу.

Входные параметры: \_key – ключ.

Выходные параметры: указатель на запись.

**void insert(const TKey& \_key, TData\* \_data);**

Назначение: вставка записи в таблицу.

Входные параметры: \_key – ключ, \_data – указатель на данные.

**void remove(const TKey& \_key);**

Назначение: удаление записи из таблицы по ключу.

Входные параметры: \_key – ключ.

**friend ostream& operator<<(ostream& os, const SortedTable& table)**

Назначение: печать таблицы.

Входные параметры: os – поток вывода, table – печатаемая таблица.

Выходные параметры: поток вывода.

**void sort();**

Назначение: сортировка записей.

Входные данные: отсутствуют.

Выходны еданные: отсутствуют.

### Описание класса HashTable

template <class TKey, class TData>

class HashTable : public Table<TKey, TData>

{

protected:

TabRecord<TKey, TData>\*\* recs;

TabRecord<TKey, TData>\* pMark = new TabRecord<TKey, TData>();

size\_t step;

virtual size\_t hash(const TKey& key) const;

public:

HashTable<TKey, TData>(int \_max\_size = 100);

HashTable<TKey, TData>(const HashTable& table);

virtual ~HashTable<TKey, TData>();

TabRecord<TKey, TData>\* find(const TKey& key);

virtual void insert(const TKey& \_key, TData\* \_data);

virtual void remove(const TKey& key);

TabRecord<TKey, TData>\* operator[](const TKey& \_key);

void next(int pos);

friend ostream& operator<<(ostream& os, const HashTable& table);

};

**Назначение:** абстрактное представление хеш-таблицы.

**Поля:**

recs – память под указатели на записи.

pMark – запись-метка.

step – шаг хеша.

**Конструкторы:**

**HashTable(int max\_size = 100);**

Назначение: конструктор с параметром.

Входные параметры: \_max\_size – максимальный размер таблицы.

HashTable<TKey, TData>(const HashTable& table);

Назначение: конструктор копирования.

Входные параметры: таблица, которую будем копировать.

virtual ~HashTable();

Назначение: деструктор.

**Методы:**

**virtual size\_t hash(const TKey& key) = 0;**

Назначение: хеш-функция.

Входные параметры: key – ключ.

Выходные параметры: хеш-значение ключа.

TabRecord<TKey, TData>\* find(const TKey& key);

Назначение: поиск записи в таблице по ключу.

Входные параметры: key – ключ.

Выходные параметры: указатель на запись.

TabRecord<TKey, TData>\* operator[](const TKey& \_key);

Назначение: поиск записи в таблице по ключу.

Входные параметры: \_key – ключ.

Выходные параметры: указатель на запись.

void insert(const TKey& \_key, TData\* \_data);

Назначение: вставка записи в таблицу.

Входные параметры: \_key – ключ, \_data – указатель на данные.

void remove(const TKey& \_key);

Назначение: удаление записи из таблицы по ключу.

Входные параметры: \_key – ключ.

void next(int pos);

Назначение: найти следующий пустой элемент.

Входные параметры: pos – позиция с которой ищем пустой элемент.

Выходные параметры: найденное хеш-значение.

friend ostream& operator<<(ostream& os, const ArrayHashTable& table);

Назначение: вывод таблицы.

Входные параметры: os – поток вывода, table – печатаемая таблица.

Выходные параметры: поток вывода.

# Заключение

В рамках данной лабораторной работы была разработана и реализована библиотека, которая содержит в себе структуру данных - таблицы. Были реализованы таблицы трёх видов и выявлены достоинства и недостатки каждой. Это значит, что в результате успешного выполнения этой лабораторной работы были получены полезные навыки разработки и тестирования новый для нас структур данных, которые могут быть применены в различных областях программирования, где требуется эффективная работа с данными.

# Литература

1. Лекция «Списковые структуры хранения» Сысоева А.В. [<https://cloud.unn.ru/s/x33MEa9on8HgNgw>]
2. Лекция «Списки в динамической памяти» Сысоева А.В. [<https://cloud.unn.ru/s/rCiKGSX33SSGPi4>]
3. Лекция «Полиномы» Сысоева А.В.[ <https://cloud.unn.ru/s/t6o9kp5g9bpf2yz>]
4. Лекция «Организация доступа по имени. Таблицы» Сысоева А.В. [<https://cloud.unn.ru/s/2Y92XyGc7r3XdBC>]
5. Лекция «Хеш-таблицы» Сысоева А.В. [<https://cloud.unn.ru/s/B5fr3gKAL2LoHyH>]

# Приложения

## Приложение А. Реализация класса TabRecord

**#**ifndef \_\_TABRECORD\_H\_\_

#define \_\_TABRECORD\_H\_\_

template <typename TKey, typename TData>

struct TabRecord {

TKey key;

TData\* data;

TabRecord();

TabRecord(const TKey& \_key, TData\* \_data);

TabRecord(const TabRecord<TKey, TData>& tr);

~TabRecord();

const TabRecord<TKey, TData>& operator=(const TabRecord<TKey, TData>& tr);

};

template <typename TKey, typename TData>

TabRecord<TKey, TData>::TabRecord() {

key = TKey();

data = new TData();

}

template <typename TKey, typename TData>

TabRecord<TKey, TData>::TabRecord(const TKey& \_key, TData\* \_data) {

key = \_key;

data = new TData(\*\_data);

}

template <typename TKey, typename TData>

TabRecord<TKey, TData>::TabRecord(const TabRecord<TKey, TData>& tr) {

key = tr.key;

data = new TData(\*tr.data);

}

template <typename TKey, typename TData>

TabRecord<TKey, TData>::~TabRecord() {

if (data) delete data;

}

template <typename TKey, typename TData>

const TabRecord<TKey, TData>& TabRecord<TKey, TData>::operator=(const TabRecord<TKey, TData>& tr) {

key = tr.key;

if (data) delete data;

data = new TData(\*tr.data);

return (\*this);

}

#endif // !TABRECORD\_H

## Приложение Б. Реализация класса Table

#ifndef \_\_TABLE\_H\_\_

#define \_\_TABLE\_H\_\_

#include "tabrecord.h"

#include <iostream>

using namespace std;

template <typename TKey, typename TData>

class Table{

protected:

int count;

int max\_size;

int curr\_pos;

public:

virtual void insert(const TKey& \_key, TData\* \_data) = 0;

virtual void remove(const TKey& \_key) = 0;

virtual TabRecord<TKey, TData>\* find(const TKey& \_key) = 0;

virtual TabRecord<TKey, TData>\* operator[](const TKey& \_key) = 0;

Table();

bool full() const noexcept;

bool empty() const noexcept;

bool ended() const noexcept;

virtual bool reset() noexcept;

virtual bool next() noexcept;

int get\_size() const noexcept;

int get\_max\_size() const noexcept;

};

template <typename TKey, typename TData>

Table<TKey, TData>::Table() {

count = 0;

max\_size = 100;

curr\_pos = -1;

}

template <typename TKey, typename TData>

bool Table<TKey, TData>::full() const noexcept {

return (count == max\_size);

}

template <typename TKey, typename TData>

bool Table<TKey, TData>::empty() const noexcept {

return (count == 0);

}

template <typename TKey, typename TData>

bool Table<TKey, TData>::ended() const noexcept {

return (curr\_pos >= max\_size);

}

template <typename TKey, typename TData>

bool Table<TKey, TData>::reset() noexcept {

if (!empty()) {

curr\_pos = 0;

return ended();

}

else {

curr\_pos = -1;

return ended();

}

}

template <typename TKey, typename TData>

bool Table<TKey, TData>::next() noexcept {

curr\_pos++;

return ended();

}

template <typename TKey, typename TData>

int Table<TKey, TData>::get\_size() const noexcept {

return count;

}

template <typename TKey, typename TData>

int Table<TKey, TData>::get\_max\_size() const noexcept {

return max\_size;

}

#endif

## Приложение В. Реализация класса ScanTable

#ifndef \_\_SCAN\_TABLE\_H\_\_

#define \_\_SCAN\_TABLE\_H\_\_

#include "table.h"

template <typename TKey, typename TData>

class ScanTable : public Table<TKey, TData> {

protected:

TabRecord<TKey, TData>\*\* recs;

public:

ScanTable(int \_max\_size = 100);

ScanTable(const ScanTable<TKey, TData>& st);

virtual ~ScanTable();

virtual void insert(const TKey& \_key, TData\* \_data);

virtual void remove(const TKey& \_key);

virtual TabRecord<TKey, TData>\* find(const TKey& key);

TabRecord<TKey, TData>\* operator[](const TKey& \_key);

friend ostream& operator<<(ostream& os, const ScanTable& table)

{

os << "ScanTable size: " << table.count << endl;

string str1 = string(32, '-'), str2 = string(62, '-');

os << '+' << str1 << '+' << str2 << '+' << endl;

for (int i = 0; i < table.count; ++i)

{

if (table.recs[i]) {

os << "| " << setw(30) << table.recs[i]->key << " | " << setw(60) << \*table.recs[i]->data << " |\n";

os << '+' << str1 << '+' << str2 << '+' << endl;

}

}

os << endl;

return os;

}

};

//конструкторы

template <typename TKey, typename TData>

ScanTable<TKey, TData>::ScanTable(int \_max\_size) {

if (\_max\_size <= 0) {

std::string exp = "ERROR: Table max\_size cant be less or equal than 0.";

throw exp;

}

max\_size = \_max\_size;

recs = new TabRecord<TKey, TData>\*[max\_size];

this->count = 0;

curr\_pos = -1;

}

template <typename TKey, typename TData>

ScanTable<TKey, TData>::ScanTable(const ScanTable<TKey, TData>& st) {

max\_size = st.max\_size;

this->count = st.count;

curr\_pos = st.curr\_pos;

recs = new TabRecord<TKey, TData>\* [max\_size];

for (int i = 0; i < count; i++) {

TKey key = st.recs[i]->key;

TData\* data = st.recs[i]->data;

recs[i] = new TabRecord<TKey, TData>(key, data);

}

}

//деструктор

template <typename TKey, typename TData>

ScanTable<TKey, TData>::~ScanTable() {

if (recs != nullptr) {

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

if (recs[i] != nullptr) delete recs[i];

delete recs;

}

}

//3 метода и оператор

template <typename TKey, typename TData>

TabRecord<TKey, TData>\* ScanTable<TKey, TData>::find(const TKey& key) {

TabRecord<TKey, TData>\* res = nullptr;

for (int i = 0; i < count; i++) {

if (recs[i]->key == key) {

curr\_pos = i;

res = recs[i];

break;

}

}

return res;

}

template <typename TKey, typename TData>

void ScanTable<TKey, TData>::insert(const TKey& \_key, TData\* \_data) {

if (full()) {

std::string exp = "ERROR: Table is full.";

throw exp;

}

if (find(\_key) == nullptr) {

recs[count++] = new TabRecord<TKey, TData>(\_key, \_data);

}

}

template <typename TKey, typename TData>

void ScanTable<TKey, TData>::remove(const TKey& \_key) {

if (empty()) {

std::string exp = "ERROR: Table is empty.";

throw exp;

}

if (find(\_key) != nullptr) {

delete recs[curr\_pos];

if (count - 1 != curr\_pos) {

recs[curr\_pos] = new TabRecord<TKey, TData>(\*recs[count - 1]);

delete recs[count - 1];

}

else {

recs[curr\_pos] = nullptr;

}

/\*for (int i = curr\_pos; i < count; i++) {

recs[i] = recs[i + 1];

}\*/

count--;

}

else {

string exp = "ERROR: key not found.";

throw exp;

}

}

template <typename TKey, typename TData>

TabRecord<TKey, TData>\* ScanTable<TKey, TData>::operator[](const TKey& \_key) {

return find(\_key);

}

#endif

## Приложение Г. Реализация класса SortedTable

#ifndef \_\_SCAN\_TABLE\_H\_\_

#define \_\_SCAN\_TABLE\_H\_\_

#include "table.h"

template <typename TKey, typename TData>

class ScanTable : public Table<TKey, TData> {

protected:

TabRecord<TKey, TData>\*\* recs;

public:

ScanTable(int \_max\_size = 100);

ScanTable(const ScanTable<TKey, TData>& st);

virtual ~ScanTable();

virtual void insert(const TKey& \_key, TData\* \_data);

virtual void remove(const TKey& \_key);

virtual TabRecord<TKey, TData>\* find(const TKey& key);

TabRecord<TKey, TData>\* operator[](const TKey& \_key);

friend ostream& operator<<(ostream& os, const ScanTable& table)

{

os << "ScanTable size: " << table.count << endl;

string str1 = string(32, '-'), str2 = string(62, '-');

os << '+' << str1 << '+' << str2 << '+' << endl;

for (int i = 0; i < table.count; ++i)

{

if (table.recs[i]) {

os << "| " << setw(30) << table.recs[i]->key << " | " << setw(60) << \*table.recs[i]->data << " |\n";

os << '+' << str1 << '+' << str2 << '+' << endl;

}

}

os << endl;

return os;

}

};

//конструкторы

template <typename TKey, typename TData>

ScanTable<TKey, TData>::ScanTable(int \_max\_size) {

if (\_max\_size <= 0) {

std::string exp = "ERROR: Table max\_size cant be less or equal than 0.";

throw exp;

}

max\_size = \_max\_size;

recs = new TabRecord<TKey, TData>\*[max\_size];

this->count = 0;

curr\_pos = -1;

}

template <typename TKey, typename TData>

ScanTable<TKey, TData>::ScanTable(const ScanTable<TKey, TData>& st) {

max\_size = st.max\_size;

this->count = st.count;

curr\_pos = st.curr\_pos;

recs = new TabRecord<TKey, TData>\* [max\_size];

for (int i = 0; i < count; i++) {

recs[i] = new TabRecord<TKey, TData>(st.recs[i]->key, st.recs[i]->data);

}

}

//деструктор

template <typename TKey, typename TData>

ScanTable<TKey, TData>::~ScanTable() {

if (recs != nullptr) {

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

if (recs[i] != nullptr) delete recs[i];

delete recs;

}

}

//3 метода и оператор

template <typename TKey, typename TData>

TabRecord<TKey, TData>\* ScanTable<TKey, TData>::find(const TKey& key) {

TabRecord<TKey, TData>\* res = nullptr;

for (int i = 0; i < count; i++) {

if (recs[i]->key == key) {

curr\_pos = i;

res = recs[i];

break;

}

}

return res;

}

template <typename TKey, typename TData>

void ScanTable<TKey, TData>::insert(const TKey& \_key, TData\* \_data) {

if (full()) {

throw "ERROR: Table is full.";

}

if (find(\_key) != nullptr) {

throw "Key repeat, it's not good";

}

recs[count++] = new TabRecord<TKey, TData>(\_key, \_data);

}

template <typename TKey, typename TData>

void ScanTable<TKey, TData>::remove(const TKey& \_key) {

if (empty()) {

string exp = "ERROR: Table is empty.";

throw exp;

}

if (find(\_key) != nullptr) {

delete recs[curr\_pos];

if (count - 1 != curr\_pos) {

recs[curr\_pos] = new TabRecord<TKey, TData>(\*recs[count - 1]);

delete recs[count - 1];

}

else {

recs[curr\_pos] = nullptr;

}

count--;

}

else {

throw "ERROR: key not found.";

}

}

template <typename TKey, typename TData>

TabRecord<TKey, TData>\* ScanTable<TKey, TData>::operator[](const TKey& \_key) {

return find(\_key);

}

#endif

## Приложение Е. Реализация класса HashTable

#ifndef \_\_HASH\_TABLE\_H\_\_

#define \_\_HASH\_TABLE\_H\_\_

#include "table.h"

#include <iostream>

#include <iomanip>

template <class TKey, class TData>

class HashTable : public Table<TKey, TData>

{

protected:

TabRecord<TKey, TData>\*\* recs;

TabRecord<TKey, TData>\* pMark = new TabRecord<TKey, TData>();

size\_t step;

int free\_pos;

virtual size\_t hash(const TKey& key) const;

void next(int pos);

public:

HashTable<TKey, TData>(int \_max\_size = 100, size\_t \_step = 13);

HashTable<TKey, TData>(const HashTable& table);

virtual ~HashTable<TKey, TData>();

TabRecord<TKey, TData>\* find(const TKey& key);

virtual void insert(const TKey& \_key, TData\* \_data);

virtual void remove(const TKey& key);

TabRecord<TKey, TData>\* operator[](const TKey& \_key);

friend ostream& operator<<(ostream& os, const HashTable& table)

{

os << "HashTable count: " << table.count << endl;

string str1 = string(32, '-'), str2 = string(62, '-');

os << '+' << str1 << '+' << str2 << '+' << endl;

for (int i = 0; i < table.max\_size; ++i)

{

if (table.recs[i] && table.recs[i] != table.pMark) {

os << "| " << setw(30) << table.recs[i]->key << " | " << setw(60) << \*table.recs[i]->data << " | \n";

os << '+' << str1 << '+' << str2 << '+' << endl;

}

}

os <<endl;

return os;

}

};

template <class TKey, class TData>

size\_t HashTable<TKey, TData>::hash(const TKey& key) const

{

std::hash<TKey> hasher;

return hasher(key) % max\_size;

}

template <class TKey, class TData>

HashTable<TKey, TData>::HashTable(int \_max\_size, size\_t \_step)

{

if (\_max\_size <= 0)

{

throw "Size can't is negative";

}

max\_size = \_max\_size;

count = 0;

curr\_pos = 0;

free\_pos = 0;

int flag = 0;

for (int i = 2; i <= \_max\_size; i++)

{

if ((\_max\_size % i == 0) && (step % i == 0)) {

flag = 0;

break;

}

else if (i == \_max\_size) flag = 1;

}

if (flag == 1) {

throw "step is bad";

}

else step = \_step;

recs = new TabRecord<TKey, TData>\* [max\_size];

for (int i = 0; i < \_max\_size; ++i) recs[i] = nullptr;

}

template <class TKey, class TData>

HashTable<TKey, TData>::HashTable(const HashTable& table)

{

this->max\_size = table.max\_size;

this->step = table.step;

this->count = table.count;

free\_pos = table.free\_pos;

curr\_pos = table.curr\_pos;

pMark = new TabRecord<TKey, TData>();

recs = new TabRecord<TKey, TData>\* [max\_size];

for (int i = 0; i < max\_size; ++i)

{

if (table.recs[i])

{

recs[i] = new TabRecord<TKey, TData>(\*table.recs[i]);

}

else {

recs[i] = nullptr;

}

}

}

template <class TKey, class TData>

HashTable<TKey, TData>::~HashTable()

{

for (int i = 0; i < max\_size; ++i)

{

if (recs[i] == pMark) recs[i] = nullptr;

else if (recs[i] != nullptr) delete recs[i];

}

delete pMark;

}

template <class TKey, class TData>

TabRecord<TKey, TData>\* HashTable<TKey, TData>::find(const TKey& key)

{

int hs = hash(key), t = (hs + step) % max\_size, c = 1;

free\_pos = hs;

if (recs[hs] == nullptr)

{

free\_pos = hs;

return nullptr;

}

if (recs[hs]->key == key && recs[hs] != pMark)

{

curr\_pos = hs;

return recs[hs];

}

while (recs[t] != nullptr && t != hs && c < max\_size)

{

if (recs[t]->key == key)

{

curr\_pos = t;

return recs[t];

}

if (recs[t] == nullptr)

{

free\_pos = t;

return nullptr;

}

t = (t + step) % max\_size;

++c;

}

if (recs[free\_pos] != pMark && recs[free\_pos] != nullptr) {

next(free\_pos);

}

return nullptr;

}

template <class TKey, class TData>

void HashTable<TKey, TData>::insert(const TKey& key,TData\* \_data)

{

if (ended())

{

throw "Table is full\n";

}

TabRecord<TKey, TData>\* tmp = find(key);

if (!tmp)

{

recs[free\_pos] = new TabRecord<TKey, TData>(key, \_data);

count++;

}

else

{

tmp->data = \_data; // throw

}

}

template <class TKey, class TData>

void HashTable<TKey, TData>::remove(const TKey& key)

{

if (empty())

{

throw string("Table is empty\n");

}

TabRecord<TKey, TData>\* exist = find(key);

if (!exist)

{

throw string("Wrong key\n");

}

count--;

delete recs[curr\_pos]; recs[curr\_pos] = pMark;

}

template <typename TKey, typename TData>

TabRecord<TKey, TData>\* HashTable<TKey, TData>::operator[](const TKey& \_key) {

return find(\_key);

}

template <class TKey, class TData>

void HashTable<TKey, TData>::next(int pos)

{

if (count == max\_size) curr\_pos = 0;

int new\_pos = (pos + step % max\_size);

while (new\_pos != pos && (recs[new\_pos] != pMark && recs[new\_pos] != nullptr))

{

new\_pos = (new\_pos + step) % max\_size;

}

curr\_pos = new\_pos;

}

#endif

## Приложение Ж. Программа демонстрации таблиц полиномов

#include <iostream>

#include "sortedtable.h"

#include"HashTable.h"

#include"tpolynom.h"

#include<Windows.h>

void func(ScanTable<string, TPolynom>& scan\_t, SortedTable<string, TPolynom>& sort\_t, HashTable<string, TPolynom>& hash\_t);

void op(ScanTable<string, TPolynom>& scan\_t, SortedTable<string, TPolynom>& sort\_t, HashTable<string, TPolynom>& hash\_t, int n);

int main()

{

setlocale(LC\_ALL, "Russian");

ScanTable<string, TPolynom> scan\_t;

SortedTable<string, TPolynom> sort\_t;

HashTable<string, TPolynom> hash\_t;

try {

func(scan\_t, sort\_t, hash\_t);

}

catch (string exp){

cout << exp << endl;

return -1;

}

return 0;

}

void func(ScanTable<string, TPolynom>& scan\_t, SortedTable<string, TPolynom>& sort\_t, HashTable<string, TPolynom>& hash\_t) {

while (1) {

cout << "Выберите действие с таблицами: \n1.Вставить новый полином \n" <<

"2.Удалить полином из таблицы \n3.Произвести операцию над полиномами \n" <<

"4.Вывести на экран таблицы \n5.Закончить работу" << endl;

int action;

string tmp;

cin >> tmp;

if (tmp == "1" || tmp == "2" || tmp == "3" || tmp == "4" || tmp == "5") {

action = tmp[0] - '0';

}

else {

action = 6;

}

switch (action) {

case 1:

{

cout << "Введите полином, который хотите добавить: ";

try {

string polynom\_str = " ";

cin >> polynom\_str;

TPolynom\* polynom = new TPolynom(polynom\_str);

polynom\_str = polynom->GetString();

scan\_t.insert(polynom\_str, polynom);

sort\_t.insert(polynom\_str, polynom);

hash\_t.insert(polynom\_str, polynom);

cout << "Вставка выполнена успешно!\n";

}

catch (string exp) {

cout << exp << endl;

cout << "Упс, ошибка :( \nПопробуйте снова!";

}

break;

}

case 2:

{

cout << "Введите полином, который хотите удалить: ";

try {

string polynom\_str = " ";

cin >> polynom\_str;

scan\_t.remove(polynom\_str);

sort\_t.remove(polynom\_str);

hash\_t.remove(polynom\_str);

cout << "Удаление выполнено успешно! \n";

}

catch (string exp) {

cout << exp << endl;

cout << "Упс, ошибка :( \nПопробуйте снова!";

}

break;

}

case 3:

{

cout << "Выберите какую операцию вы хотите произвести: \n1.Сложение \n" <<

"2.Вычитание \n3.Произведение \n" <<

"4.Дифференцировать по x \n5.Дифференцировать по y\n" <<

"6.Дифференцировать по z" <<endl;

int n;

string tmp;

cin >> tmp;

if (tmp == "1" || tmp == "2" || tmp == "3" || tmp == "4" || tmp == "5" || tmp == "6") {

n = tmp[0] - '0';

}

else {

n = 7;

}

op(scan\_t, sort\_t, hash\_t, n);

break;

}

case 4:

{

try {

cout << scan\_t;

cout << sort\_t;

cout << hash\_t;

}

catch (string exp) {

cout << exp << endl;

cout << "Упс, ошибка :( \nПопробуйте снова!";

}

break;

}

case 5:

{

cout << "Заходите ещё! :)\n";

return;

break;

}

default:

cout << "Такого варианта выбора нет :( \nGопробуйте ещё раз!\n";

break;

}

}

}

void op(ScanTable<string, TPolynom>& scan\_t, SortedTable<string, TPolynom>& sort\_t, HashTable<string, TPolynom>& hash\_t, int n) {

if (n < 4 && n>0) {

cout << "Введите первый полином: ";

string polynom\_str1 = " ";

cin >> polynom\_str1;

TPolynom\* polynom1 = new TPolynom(polynom\_str1);

polynom\_str1 = polynom1->GetString();

if (scan\_t.find(polynom\_str1) == nullptr || sort\_t.find(polynom\_str1) == nullptr ||

hash\_t.find(polynom\_str1) == nullptr)

{

cout << "Упс, такого полинома в таблицах нет :(\n Попробуйте снова\n";

return;

}

cout << "Введите второй полином: ";

string polynom\_str2 = " ";

cin >> polynom\_str2;

TPolynom\* polynom2 = new TPolynom(polynom\_str2);

polynom\_str2 = polynom2->GetString();

if (scan\_t.find(polynom\_str2) == nullptr || sort\_t.find(polynom\_str2) == nullptr ||

hash\_t.find(polynom\_str2) == nullptr)

{

cout << "Упс, такого полинома в таблицах нет :(\n Попробуйте снова\n";

return;

}

TPolynom\* res = new TPolynom();

switch (n) {

case 1:

\*res = \*polynom1 + \*polynom2;

break;

case 2:

\*res = \*polynom1 - \*polynom2;

break;

case 3:

\*res = (\*polynom1) \* (\*polynom2);

break;

}

cout << "Получили такой полином: " << \*res << endl;

cout << "Хотите вставить его, как новый полином? y/n" << endl;

string ins\_bool;

cin >> ins\_bool;

if (ins\_bool == "y") {

string res\_str = res->GetString();

scan\_t.insert(res\_str, res);

sort\_t.insert(res\_str, res);

hash\_t.insert(res\_str, res);

cout << "Вставка выполнена успешно!\n";

}

return;

}

if (n < 7 && n >3) {

cout << "Введите полином, который хотите дифференцировать: ";

string polynom\_str = " ";

cin >> polynom\_str;

TPolynom\* polynom = new TPolynom(polynom\_str);

polynom\_str = polynom->GetString();

if (scan\_t.find(polynom\_str) == nullptr || sort\_t.find(polynom\_str) == nullptr ||

hash\_t.find(polynom\_str) == nullptr)

{

cout << "Упс, такого полинома в таблицах нет :(\n Попробуйте снова\n";

return;

}

TPolynom\* res = new TPolynom();

switch (n) {

case 4:

\*res = polynom->dif\_x();

break;

case 5:

\*res = polynom->dif\_y();

break;

case 6:

\*res = polynom->dif\_z();

break;

}

cout << "Получили такой полином: " << \*res << endl;

cout << "Хотите вставить его, как новый полином? y/n" << endl;

string ins\_bool;

cin >> ins\_bool;

if (ins\_bool == "y") {

string res\_str = res->GetString();

scan\_t.insert(res\_str, res);

sort\_t.insert(res\_str, res);

hash\_t.insert(res\_str, res);

cout << "Вставка выполнена успешно!\n";

}

return;

}

cout << "Такого варианта выбора нет :( \nGопробуйте ещё раз!\n";

}