

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
1 Постановка задачи .....	4
2 Руководство пользователя.....	5
2.1 Приложение для демонстрации работы таблиц.....	5
3 Руководство программиста.....	7
3.1 Описание алгоритмов.....	7
3.1.1 Просматриваемая Таблица.....	7
3.1.2 Отсортированная таблица .....	8
3.1.3 Хеш-таблица открытого перемешивания .....	11
3.2 Описание программной реализации .....	15
3.2.1 Описание класса TabRecord .....	15
3.2.2 Описание класса Table .....	16
3.2.3 Описание класса ScanTable .....	17
3.2.4 Описание класса SortedTable .....	19
3.2.5 Описание класса HashTable.....	20
Заключение .....	23
Литература .....	24
Приложения .....	25
Приложение А. Реализация класса TabRecord.....	25
Приложение Б. Реализация класса Table.....	25
Приложение В. Реализация класса ScanTable.....	27
Приложение Г. Реализация класса SortedTable.....	29
Приложение Е. Реализация класса HashTable.....	32
Приложение Ж. Программа демонстрации таблиц полиномов .....	35

## **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.

# 1 Постановка задачи

## **Цель:**

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

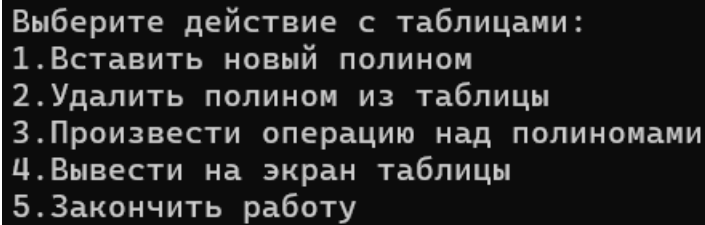
## **Задачи:**

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.

## 2 Руководство пользователя

### 2.1 Приложение для демонстрации работы таблиц

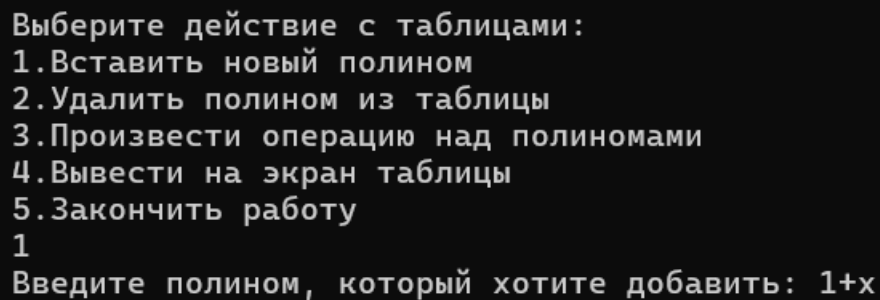
1. Запустите приложение sample\_table.exe. В результате появится окно, показанное ниже (Рис. 1)



```
Выберите действие с таблицами:  
1.Вставить новый полином  
2.Удалить полином из таблицы  
3.Произвести операцию над полиномами  
4.Вывести на экран таблицы  
5.Закончить работу
```

Рис. 1. Начальный вывод программы

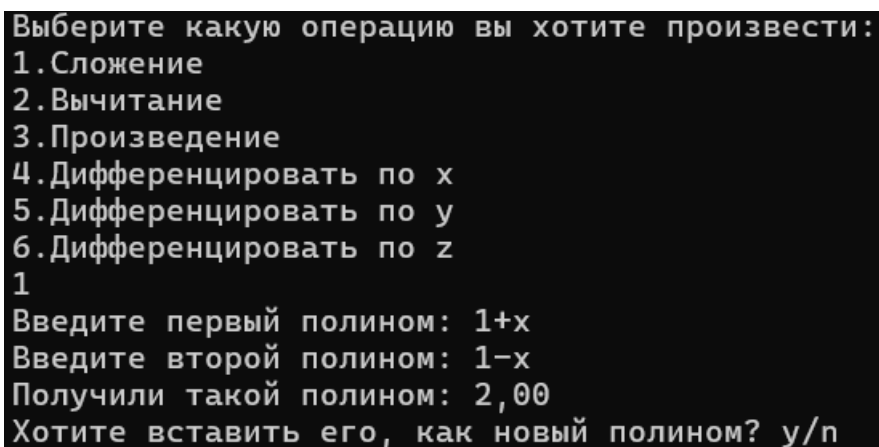
2. Сначала следует вставить новые полиномы в таблицу, поскольку она пустая. Делать это, следуя инструкциям на экране ().



```
Выберите действие с таблицами:  
1.Вставить новый полином  
2.Удалить полином из таблицы  
3.Произвести операцию над полиномами  
4.Вывести на экран таблицы  
5.Закончить работу  
1  
Введите полином, который хотите добавить: 1+x
```

Рис. 2. Вставка полинома

3. 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 на Рис. 3.



```
Выберите какую операцию вы хотите произвести:  
1.Сложение  
2.Вычитание  
3.Произведение  
4.Дифференцировать по x  
5.Дифференцировать по y  
6.Дифференцировать по z  
1  
Введите первый полином: 1+x  
Введите второй полином: 1-x  
Получили такой полином: 2,00  
Хотите вставить его, как новый полином? y/n
```

Рис. 3. Интрукция к сложению элементов

4. После того, как вы произведёте операцию, вы можете выбрать, добавлять полученный результат в таблицу или нет (Рис. 4).

```
Получили такой полином: 2,00
Хотите вставить его, как новый полином? y/n
y
Вставка выполнена успешно!
```

Рис. 4. Добавление результата в таблицу

5. После того, как вы совершите нужные вам операции. Вы можете посмотреть на результат, показав таблицы на экране (Рис. 5).

```
ScanTable size: 3
+-----+-----+
| 1,00+1,00*x | 1,00+1,00*x |
+-----+-----+
| 1,00-1,00*x | 1,00-1,00*x |
+-----+-----+
| 2,00 | 2,00 |
+-----+-----+

SortTable size: 3
+-----+-----+
| 1,00+1,00*x | 1,00+1,00*x |
+-----+-----+
| 1,00-1,00*x | 1,00-1,00*x |
+-----+-----+
| 2,00 | 2,00 |
+-----+-----+

HashTable count: 3
+-----+-----+
| 1,00+1,00*x | 1,00+1,00*x |
+-----+-----+
| 1,00-1,00*x | 1,00-1,00*x |
+-----+-----+
| 2,00 | 2,00 |
+-----+-----+
```

Рис. 5. Вывод таблиц

6. Также вы в любую секунду можете завершить работу (Рис. 6).

```
Выберите действие с таблицами:
1. Вставить новый полином
2. Удалить полином из таблицы
3. Произвести операцию над полиномами
4. Вывести на экран таблицы
5. Закончить работу
5
Заходите ещё! :)
```

Рис. 6. Завершение работы

## 3 Руководство программиста

### 3.1 Описание алгоритмов

#### 3.1.1 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)



Рис. 7. 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).

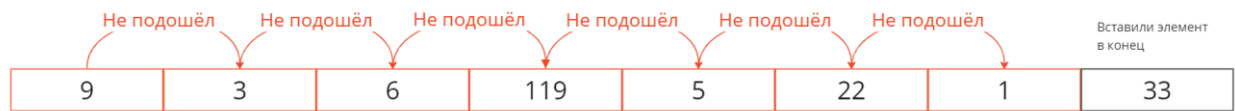


Рис. 8. Вставка элемента в просматриваемую таблицу

### 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).



Нашли, удаляем

Записываем вместо неё последнюю запись

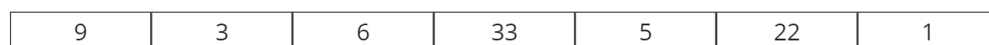


Рис. 9. Удаление элемента в просматриваемой таблице

### 3.1.2 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.

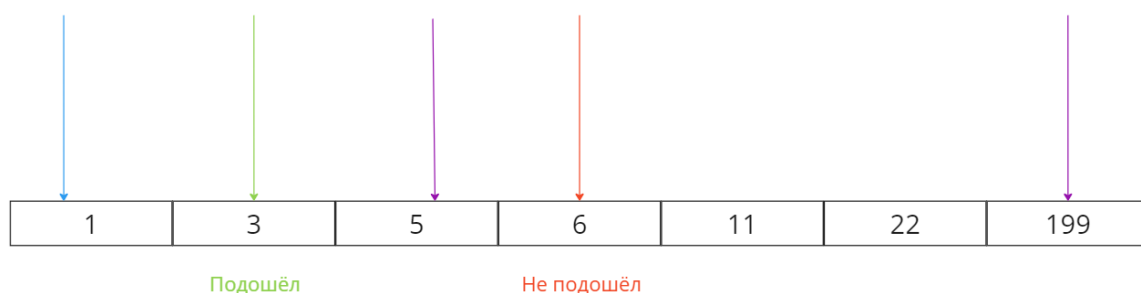


Рис. 10. Поиск в отсортированной таблице

## 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.

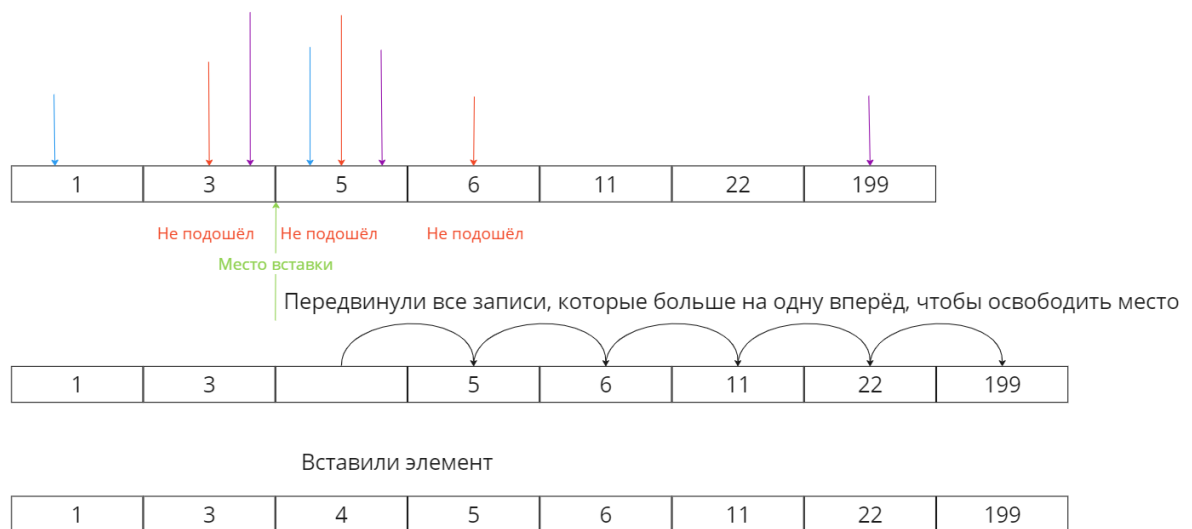


Рис. 11. Вставка в отсортированную таблицу

### 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.



Рис. 12. Удаление из отсортированной таблицы

### 3.1.3 Хеш-таблица открытого перемешивания

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

Алгоритм хеш-таблицы с открытым перемешиванием (или просто открытой адресацией) предполагает разрешение коллизий путем 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.

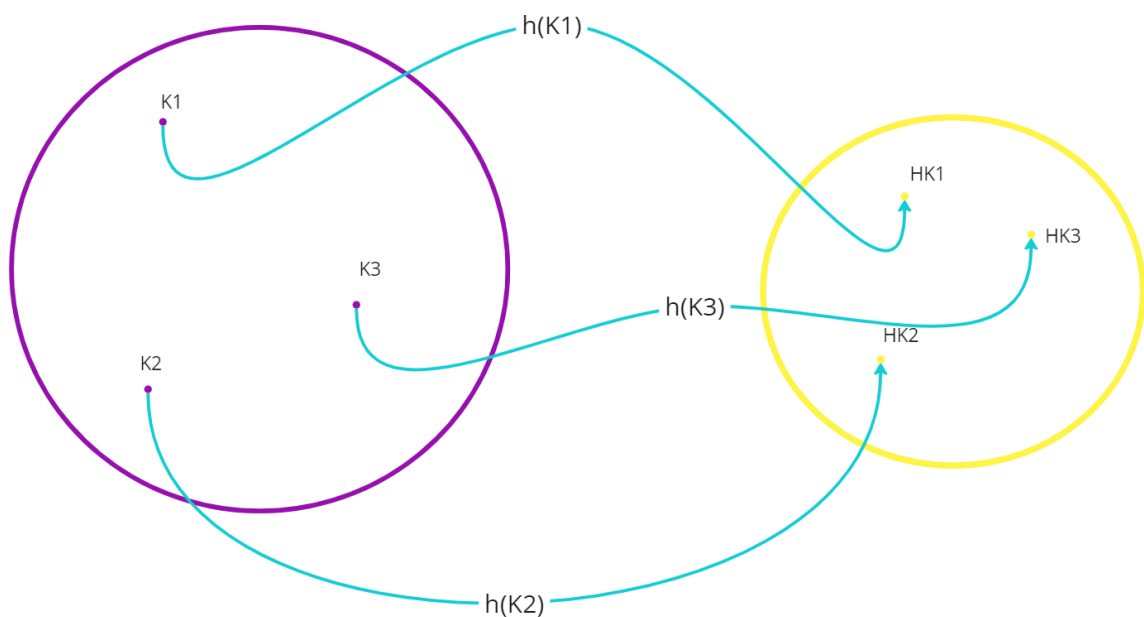


Рис. 13. Значение ключе и хеш-значений ключей

#### 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 ищется следующая свободная ячейка путем 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 дойдем до свободной ячейки.

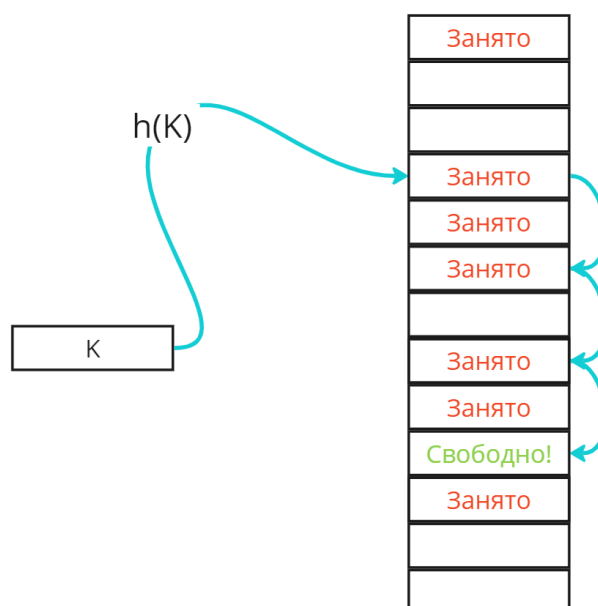


Рис. 14. Разрешение коллизии

### 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).

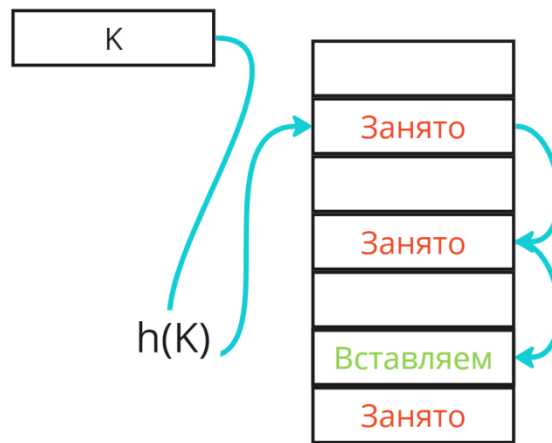


Рис. 15. Вставка в хеш-таблицу

#### 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).

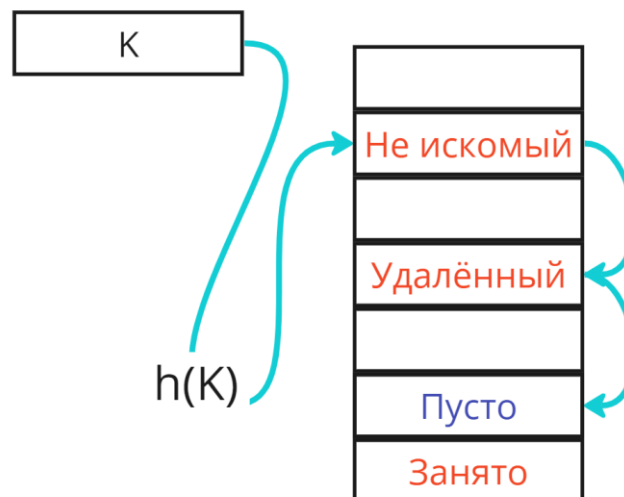


Рис. 16. Поиск в хеш-таблице

#### 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.

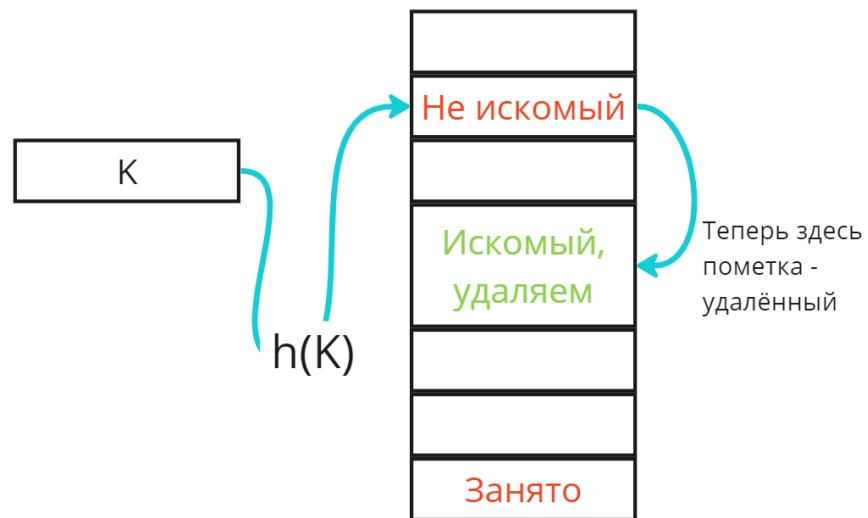


Рис. 17. Удаление в хеш-таблице

## 3.2 Описание программной реализации

### 3.2.1 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** – присваиваемый объект.

Выходные параметры: ссылка на себя.

### 3.2.2 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.

### 3.2.3 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.

### 3.2.4 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.

### 3.2.5 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 новых для нас структур данных, которые могут быть применены в различных областях программирования, где требуется эффективная работа с данными.

## Литература

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

#include "scantable.h"

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);
    ~SortedTable();

    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)
    {

        os << "SortTable 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 <class TKey, class TData>
SortedTable<TKey, TData>::SortedTable(int _max_size) : ScanTable(_max_size)
{}

template <class TKey, class TData>
SortedTable<TKey, TData>::SortedTable(const ScanTable<TKey, TData>* st) :
ScanTable(*st) {
    sort();
}

template <class TKey, class TData>
SortedTable<TKey, TData>::SortedTable(const SortedTable<TKey, TData>& srt) {
    count = srt.count;
    max_size = srt.max_size;
    curr_pos = srt.curr_pos;

    recs = new TabRecord<TKey, TData>* [max_size];
    for (int i = 0; i < count; i++) {
        recs[i] = new TabRecord<TKey, TData>(*srt.recs[i]);
    }
}

//деструктор
template <class TKey, class TData>
SortedTable<TKey, TData>::~~SortedTable() {
    //ScanTable<TKey,TData>::~~ScanTable();
}

template <class TKey, class TData>

```

```

TabRecord<TKey, TData>* SortedTable<TKey, TData>::find(const TKey& key) {
    int left = 0, right = count - 1;
    TabRecord<TKey, TData>* search = nullptr;

    while (left <= right) {
        int mid = (right + left) / 2;

        if (recs[mid]->key == key) {
            search = recs[mid];
            right = mid;
            left = mid + 1;
        }
        else if (recs[mid]->key < key) left = mid + 1;
        else right = mid - 1;
    }
    if (right != -1)
        if (search == nullptr)
            curr_pos = right + 1;
        else
            curr_pos = right;
    else
        curr_pos = 0;
    return search;
}

template <class TKey, class TData>
TabRecord<TKey, TData>* SortedTable<TKey, TData>::operator[] (const TKey&
_key) {
    return find(_key);
}

template <class TKey, class TData>
void SortedTable<TKey, TData>::insert(const TKey& _key, TData* _data) {
    if (full()) {
        std::string exp = "ERROR: Table is full.";
        throw exp;
    }

    //recs[count] = new TabRecord<TKey, TData>();
    //сделать проверку на find
    if (find(_key) == nullptr) {
        for (int i = count - 1; i >= curr_pos; i--) {
            recs[i + 1] = recs[i];
        }
        recs[curr_pos] = new TabRecord<TKey, TData>(_key, _data);

        count++;
    }
}

template <class TKey, class TData>
void SortedTable<TKey, TData>::remove(const TKey& _key) {
    if (empty()) {
        std::string exp = "ERROR: Table is empty.";
        throw exp;
    }

    TabRecord<TKey, TData>* rec = find(_key);
    if (rec == nullptr) {
        std::string exp = "ERROR: Key not found.";
        throw exp;
    }
    else {

```

```

        delete rec;
        for (int i = curr_pos; i < count - 1; i++) {
            recs[i] = recs[i + 1];
        }
        count--;
    }
}

template <class TKey, class TData>
void SortedTable<TKey, TData>::sort() {
    for (int i = 0; i < count; ++i)
    {
        for (int j = i + 1; j < count; ++j)
        {
            if (recs[i]->key > recs[j]->key)
            {
                TabRecord<TKey, TData>* t = recs[i];
                recs[i] = recs[j];
                recs[j] = t;
            }
        }
    }
}

#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;

    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)
    {
        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)
{
    if (_max_size <= 0)
    {
        throw "Size can't is negative";
    }
    max_size = _max_size;
    count = 0;
    curr_pos = 0;
    recs = new TabRecord<TKey, TData>* [max_size];
    for (int i = 0; i < _max_size; ++i) recs[i] = nullptr; //pMark

    step = (max_size == 13) ? 11 : 13;
}

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;
    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;
    curr_pos = hs;

    if (recs[hs] == nullptr)
    {
        return nullptr;
    }
    if (recs[hs]->key == key && recs[hs] != pMark)
    {
        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)
        {
            return nullptr;
        }
        t = (t + step) % max_size;
        ++c;
    }
    if (recs[curr_pos] != pMark && recs[curr_pos] != nullptr)
next(curr_pos);
    return nullptr;
}

template <class TKey, class TData>
void HashTable<TKey, TData>::insert(const TKey& key, TData* _data)
{
    if (ended())
    {
        throw string("Table is full\n");
    }
    TabRecord<TKey, TData>* exist = find(key);

    if (!exist)
    {
        recs[curr_pos] = new TabRecord<TKey, TData>(key, _data);
        count++;
    }
    else
    {
        exist->data = _data;
    }
}

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");
        }
        else
        {
            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 :( \nПопробуйте 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 :( \nПопробуйте ещё раз!\n";
}

```