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:

**«Таблицы»**

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Чижов М.А./

Подпись

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

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

Подпись

Нижний Новгород  
2024

**Содержание**

[Введение 3](#_Toc167730062)

[1. Постановка задачи 5](#_Toc167730063)

[2. Руководство пользователя 6](#_Toc167730064)

[2.1 Приложение для демонстрации работы полинома 6](#_Toc167730065)

[3. Руководство программиста 8](#_Toc167730066)

[3.1 Описание алгоритмов 8](#_Toc167730067)

[3.1.1 Просматриваемая таблица 8](#_Toc167730068)

[3.1.1 Сортированная таблица 10](#_Toc167730069)

[3.1.1 Хэш-таблица 12](#_Toc167730070)

[3.1 Описание программной реализации 14](#_Toc167730071)

[3.2.1. Описание класс TabRecord 14](#_Toc167730072)

[3.2.2. Описание класса Table 15](#_Toc167730073)

[3.2.3. Описание класса TScanTable 17](#_Toc167730074)

[3.2.4. Описание класса TSortTable 19](#_Toc167730075)

[3.2.5. Описание класса THashTable 21](#_Toc167730076)

[3.2.6. Описание класса TArrayHashTable 21](#_Toc167730077)

[Заключение 25](#_Toc167730078)

[Литература 26](#_Toc167730079)

[Приложения 27](#_Toc167730080)

[Приложение А. Реализация класса TabRecord 27](#_Toc167730081)

[Приложение Б. Реализация класса Table 27](#_Toc167730082)

[Приложение В. Реализация класса TScanTable 28](#_Toc167730083)

[Приложение Г. Реализация класса TSortTable 29](#_Toc167730084)

[Приложение Д. Реализация класса THashTable 31](#_Toc167730085)

[Приложение Е. Реализация класса TarrayHashTable 31](#_Toc167730086)

# Введение

В рамках лабораторной работы предстоит разработать три различные структуры данных, предназначенные для эффективной организации и операций с множеством записей.

Первой структурой данных, котор будет разработана, является просматриваемая таблица, которая обеспечит возможность последовательного прохода по всем элементам с целью поиска заданных значений или выполнения других операций.

Вторая разрабатываемая структура данных представляет собой сортированную таблицу, способную автоматически упорядочивать данные с целью ускорения операций поиска и извлечения информации из таблицы.

И, наконец, третьей разрабатываемой структурой данных будет хэш-таблица, позволяющая эффективно хранить и быстро извлекать данные с использованием хэширования для разрешения коллизий.

Целью данной лабораторной работы является изучение принципов работы и особенностей реализации указанных структур данных, а также оценка их эффективности при решении различных задач.

1. Постановка задачи

Цель – реализовать структуры данных: просматриваемая таблица, сортированная таблица, хэш-таблица.

Задачи:

1. Реализовать класс для работы с просматриваемыми таблицами.

2. Реализовать класс для работы с отсортированной таблицей.

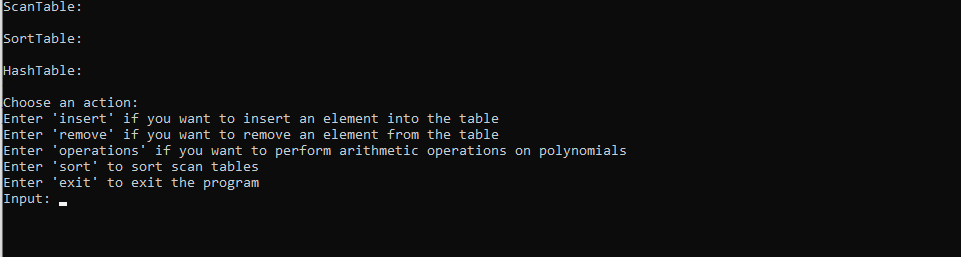
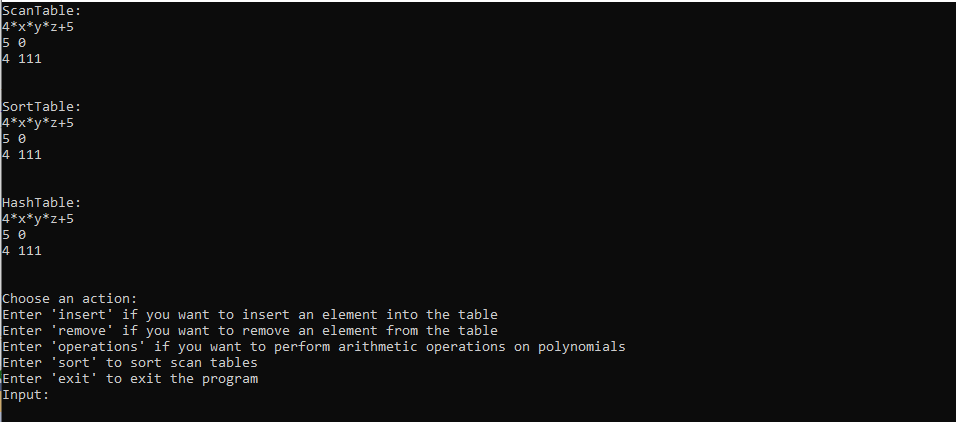
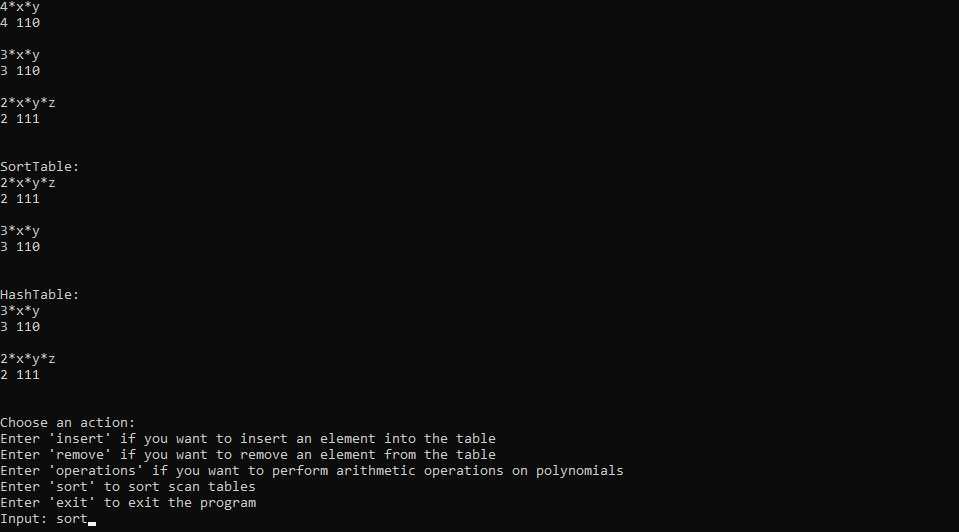
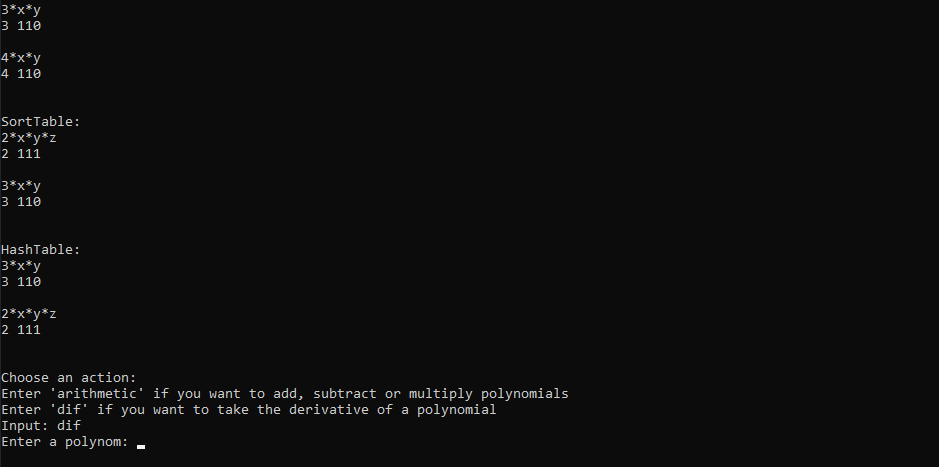
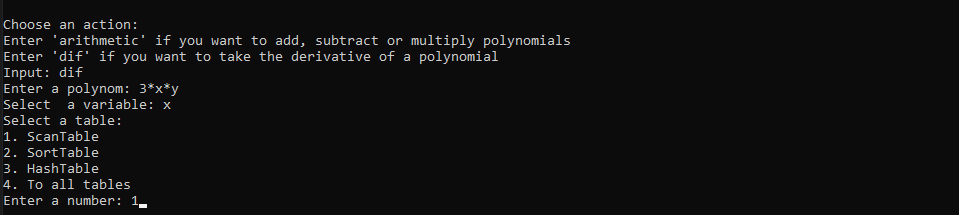
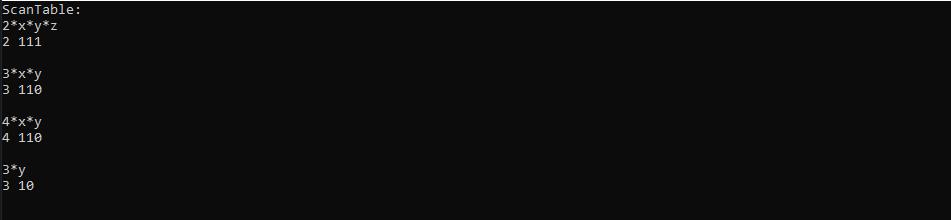
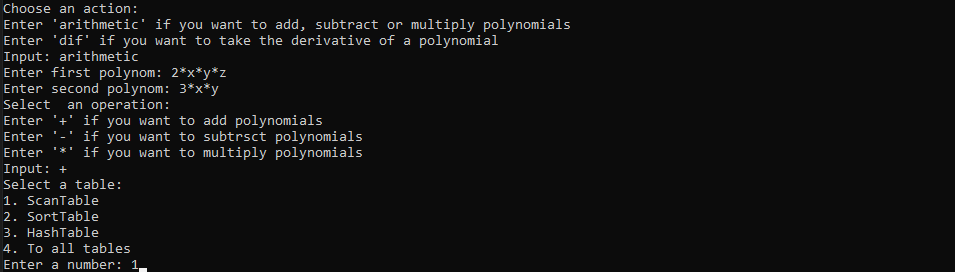
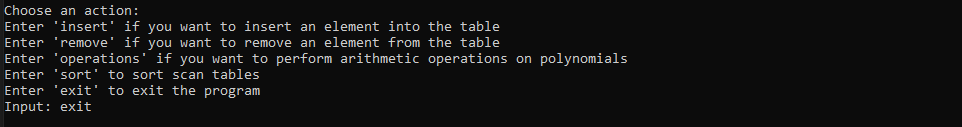
3. Реализовать класс для работы с хэш-таблицей.

4. Написать следующие операции для работы с таблицами: добавление, удаление, поиск записей.

5. Добавить вспомогательную операция сортировки просматриваемой таблицы.

1. Руководство пользователя

## Приложение для демонстрации работы таблиц

1. Запустите приложение с названием sample\_tables.exe. В результате появится окно, показанное ниже (рис. 1).
2. Основное окно программы
3. Введите команду insert, чтобы перейти к вставке полинома. После этого введите полином и выберите таблицу, в которую вы хотите вставить. (рис. 2).
4. Вставка полинома.
5. Предположим, что вставка произошла во все таблицы. В результате появится следующее окно с полином (рис. 3).
6. Полином, вставленный во все таблицы.
7. Для удаления полинома введите команду remove. После этого введите полином который вы хотите удалить и выберите таблицу. Предположим, что это будет просматриваемая таблица. После этого полином будет удален из первой таблицы (рис. 4).
8. Удаление полинома из таблицы 1.
9. Для сортировки просматриваемой таблицы введите команду sort (рис. 5).
10. Сортировка просматриваемой таблицы.
11. После этого на экран выведется результат сортировки просматриваемой таблицы (рис. 6).
12. Результат сортировки.
13. Для выполнения операций над полиномами введите команду operations. Перед вами появится окно, в котором вы должны ввести dif, если вы хотите взять производную, или arithmetic, если вы хотите выполнить сложение, вычитание или умножение полиномов (рис. 7).
14. Выбор операции.
15. Предположим, ввели команду для взятия производной. После этого необходимо ввести полином и выбрать переменную, по которой будет дифференцирования. Дифференцировать можно по x, y или z. После выбора переменной необходимо выбрать таблицу, в которую будет записан результат, предположим, что это первая таблица (рис. 8).
16. Дифференцирование полинома.
17. После этого результат дифференцирования запишется в первую таблицу (рис. 9).
18. Результат дифференцирования.
19. Для арифметических операций введите команду arithmetic. После этого введите два полинома. Выберите операцию, которую вы хотите выполнить. После этого выберите таблицу, в которую вы хотите записать результат (рис. 10).
20. Выбор арифметической операции.
21. После этого появится окно с выполненной операцией (рис. 11).
22. Результат сложения полиномов.
23. Введите команду exit, чтобы выйти из программы (рис. 12).
24. Выход из программы.
25. Руководство программиста
    1. Описание алгоритмов
       1. Просматриваемая таблица

Просматриваемая таблица – это структура данных, которая представляет собой одномерный массив элементов, размер которого заранее известен. Каждый элемент таблицы имеет структуру, состоящую из двух полей: key (ключ) и value (значение). Просматриваемая таблица может быть статической или динамической. Статическая таблица имеет фиксированный размер, который задается при ее создании. Динамическая таблица может изменять свой размер в процессе работы программы. Основные операции, которые могут быть выполнены с просматриваемой таблицей, включают поиск элемента по ключу, включение нового элемента в таблицу и удаление элемента из таблицы.

**Операция добавления записи**

Данный метод предназначен для добавления нового записи в таблицу. Он принимает два параметра: ключ и указатель на данные.

Внутри метода проверяется, не заполнена ли таблица до максимального размера. Если таблица заполнена, то генерируется исключение с сообщением "Error: scan table is full".

Если таблица не заполнена, то создается новая запись с переданными ключом и данными, и она добавляется в массив записей таблицы. Индекс новой записи в массиве равен текущему значению счетчика количества записей в таблице, которой увеличивается на единицу после добавления записи.

Пример:

|  |  |
| --- | --- |
| 1 | One |
| 3 | Three |
|  |  |
|  |  |

Добавим запись с ключом 2 и данными “Two”.

|  |  |
| --- | --- |
| 1 | One |
| 3 | Three |
| 2 | Two |
|  |  |

**Операция поиска**

Данный метод предназначен для поиска записи с определенным ключом в таблице. Внутри метода происходит итерация по всем записям в таблице. Для каждой записи проверяется, соответствует ли ее ключ заданному ключу. Если ключи совпадают, то запись возвращается как результат поиска, а текущий индекс устанавливается равным текущему индексу записи в массиве. Если запись с указанным ключом не найдена, то метод возвращает nullptr.

Пример:

|  |  |
| --- | --- |
| 1 | One |
| 3 | Three |
| 4 | Four |
| 2 | Two |

Найдем запись с ключом 2. Функция вернёт:

|  |  |
| --- | --- |
| 2 | Two |

Текущий индекс станет равен 2.

Найдем запись с ключом 7. Функция вернет nullptr.

**Операция удаления записи**

Данный метод предназначен для удаления записи с определенным ключом из таблицы. Внутри метода сначала проверяется, не пуста ли таблица. Если таблица пуста, то генерируется исключение с сообщением "Error: scan table is empty". Затем проверяется, содержится ли запись с указанным ключом в таблице, с помощью метода поиска. Если запись не найдена, то генерируется исключение с сообщением "Error: the element to be deleted was not found".

Если запись с указанным ключом найдена, то она удаляется из таблицы. Для этого сначала удаляется динамически выделенный объект записи, хранящийся по текущему индексу в массиве записей таблицы. Затем все записи с индексами, начиная с текущего и заканчивая последней, сдвигаются на один индекс влево. Наконец, уменьшается значение счетчика количества записей в таблице на единицу.

Пример:

|  |  |
| --- | --- |
| 1 | One |
| 3 | Three |
| 4 | Four |
| 2 | Two |

Удалим запись с ключом 4.

|  |  |
| --- | --- |
| 1 | One |
| 3 | Three |
| 2 | Two |
|  |  |

* + 1. Сортированная таблица

Сортированная таблица в программировании – это структура данных, которая представляет собой таблицу, элементы которой упорядочены по определенному ключу. Это означает, что каждый элемент в таблице имеет уникальный ключ, по которому он отсортирован относительно других элементов. Сортировка таблицы может быть выполнена по одному или нескольким ключам. Это позволяет эффективно выполнять поиск и фильтрацию данных в таблице. Сортировка таблицы может быть статической или динамической. В статической сортированной таблице порядок элементов задан при ее создании и не изменяется. В динамической сортированной таблице порядок элементов может изменяться в процессе работы программы. Сортированная таблица может быть реализована с помощью различных структур данных, таких как массивы, списки, деревья и других. Выбор подходящей структуры данных зависит от требований к скорости и объему данных.

**Операция поиска**

Данный метод предназначен для поиска записи с определенным ключом в сортированной таблице при помощи алгоритма бинарного поиска. В каждой итерации цикла вычисляется середина текущего диапазона и сравнивается ключ записи с заданным ключом. Если ключи совпадают, то запись возвращается как результат поиска, и текущий индекс устанавливается равным индексу записи в массиве.

Если ключ записи больше заданного ключа, то текущий диапазон индексов уменьшается вправо, иначе уменьшается влево.

Если запись с указанным ключом не найдена, то метод возвращает nullptr. В этом случае текущий индекс устанавливается равным индексу последней просмотренной записи в таблице.

Пример:

|  |  |
| --- | --- |
| 1 | One |
| 3 | Three |
| 4 | Four |
| 2 | Two |

Найдем запись с ключом 4. Функция вернёт:

|  |  |
| --- | --- |
| 4 | Four |

Текущий индекс станет равен 2.

Найдем запись с ключом 7. Функция вернет nullptr, а текущий индекс будет равен 2.

**Операция добавления записи**

Данный метод предназначен для вставки записи с заданным ключом и указателем на данные в сортированную таблицу. Внутри метода сначала проверяется, не заполнена ли таблица. Если таблица заполнена, то генерируется исключение с сообщением "Error: sorted table is full". В противном случае вызывается метод поиска, чтобы получить текущую позицию в таблице. Затем происходит сдвиг всех записей с индексами больше текущей позиции на один индекс вправо, освобождая место для вставки новой записи. Создается новая запись и устанавливается в освободившееся место в массиве. Наконец, счетчик количества записей в таблице увеличивается на единицу.

Пример:

|  |  |
| --- | --- |
| 1 | One |
| 3 | Three |
|  |  |
|  |  |

Добавим запись с ключом 2 и данными “Two”.

|  |  |
| --- | --- |
| 1 | One |
| 2 | Two |
| 3 | Three |
|  |  |

**Операция удаления записи**

Данный метод предназначен для удаления записи с заданным ключом из сортированной таблицы. Внутри метода сначала проверяется, не пуста ли таблица. Если таблица пуста, то генерируется исключение с сообщением "Error: sorted table is empty". В противном случае вызывается метод поиска записи с заданным ключом в таблице.

Если запись с таким ключом не найдена, то генерируется исключение с сообщением "Error: the element to be deleted was not found". Затем удаляется найденная запись, а все записи с индексами больше, чем индекс текущей позиции, сдвигаются на один индекс влево, заполняя освободившееся пространство. Наконец, счетчик количества записей в таблице уменьшается на единицу.

Пример:

|  |  |
| --- | --- |
| 1 | One |
| 2 | Two |
| 3 | Three |
| 4 | Four |

Удалим запись с ключом 3.

|  |  |
| --- | --- |
| 1 | One |
| 2 | Two |
| 4 | Four |
|  |  |

* + 1. Хэш-таблица

Хэш-таблица в программировании – это структура данных, которая позволяет хранить пары ключ-значение и обеспечивает быстрый доступ к значениям по ключу. Хэш-таблица использует хэш-функцию для преобразования ключа в индекс, который указывает на место хранения значения в таблице. Это позволяет получить доступ к значению по ключу за константное время, независимо от размера таблицы. Хэш-таблицы широко используются в различных областях программирования, таких как базы данных, веб-разработка, игры и многое другое

**Операция поиска**

Этот метод предназначен для поиска записи с заданным ключом в хэш-таблице. Метод начинает поиск с текущей позиции, которая вычисляется с помощью хэш-функции. Затем метод перебирает все записи в хэш-таблице и проверяет каждую запись с индексом текущей позиции на nullptr. Если запись равна nullptr, то метод возвращает nullptr, указывая на отсутствие такой записи в хэш-таблице. Если запись не равна nullptr, то метод проверяет, является ли запись удаленной записью.

Если запись удалена, то метод обновляет индекс свободной позиции. Наконец, метод проверяет, соответствует ли ключ текущей записи заданному ключу.

Если ключи совпадают, то метод возвращает указатель на найденную запись. Если запись с заданным ключом не найдена, метод повторяет цикл до максимального размера хэш-таблицы. Если запись не найдена, метод возвращает nullptr.

Пример:

|  |  |
| --- | --- |
| One | 100 |
| Three | 300 |
| Two | 200 |
|  |  |

Найдем запись с ключом Two. Функция вернёт:

|  |  |
| --- | --- |
| Two | 200 |

Найдем запись с ключом Seven. Функция вернет nullptr.

**Операция добавления записи**

Данный метод предназначен для вставки записи с заданным ключом и указателем на данные в хэш-таблицу.

Сначала метод проверяет, не заполнена ли хэш-таблица. Если хэш-таблица заполнена, метод генерирует исключение с сообщением "Error: hash table is full".

Затем метод вызывает метод поиска записи с заданным ключом в хэш-таблице. Если запись с таким ключом уже существует и свободная позиция не равна -1, то метод обновляет текущую позицию на значение свободной позиции. Затем метод создает новую запись с заданным ключом и указателем на данные. Далее метод вставляет новую запись в хэш-таблицу по текущей позиции. Если запись с заданным ключом не существует, метод просто создает новую запись и вставляет ее в хэш-таблицу. Наконец, метод увеличивает счетчик количества записей в хэш-таблице.

Пример:

|  |  |
| --- | --- |
| One | 100 |
|  |  |
| Two | 200 |
|  |  |

Добавим запись с ключом Three и данными “300”

|  |  |
| --- | --- |
| One | 100 |
| Three | 300 |
| Two | 200 |
|  |  |

**Операция удаления записи**

Этот метод предназначен для удаления записи с заданным ключом из хэш-таблицы.

Сначала вызывается метод для поиска записи с заданным ключом в хэш-таблице. Если запись с таким ключом не существует, метод генерирует исключение с сообщением "Error: record not found".

Если запись найдена, то метод удаляет найденную запись. Затем метод заменяет удаленную запись специальным маркером удаленной записи. Далее метод обновляет индекс свободной позиции на текущую позицию. Наконец, метод уменьшает счетчик количества записей в хэш-таблице.

Пример:

|  |  |
| --- | --- |
| One | 100 |
| Three | 300 |
| Two | 200 |
|  |  |

Удалим запись с ключом Three.

|  |  |
| --- | --- |
| One | 100 |
|  |  |
| Two | 200 |
|  |  |

## Описание программной реализации

### Описание класс TabRecord

template <class TKey, class TData>

class TabRecord

{

TKey key;

TData\* data;

public:

TabRecord();

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

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

TKey GetKey() const;

TData\* GetData() const;

};

Назначение: представление узла.

Поля:

**key** – значение ключа.

**data** – указатель на данные.

Методы:

TabRecord();

Назначение: инициализация объекта класса **TabRecord**.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

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

Назначение: конструктор с параметрами для установки ключа и данных объекта..

Входные параметры:

**\_key** - константная ссылка на объект типа **TKey**, представляющий ключ.

**\_data** - указатель на объект типа **TData**, представляющий данные.

Выходные параметры: новый объект с инициализированным значениями.

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

Назначение: конструктор копирования для создания копии объекта **TabRecord**.

Входные параметры: **tr** - константная ссылка на объект типа **TabRecord** который будет скопирован.

Выходные параметры: скопированный объект типа **TabRecord**.

TKey GetKey() const;

Назначение: получение значения ключа объекта.

Входные параметры: отсутствуют.

Выходные параметры: возвращает значение ключа объекта типа **TKey**.

TData\* GetData() const;

Назначение: получение указателя на данные объекта.

Входные параметры: отсутствуют.

Выходные параметры: возвращает указатель на данные объекта типа **TData**.

### Описание класса Table

template <class TKey, class TData>

class Table

{

protected:

int Count;

int maxsize;

int currPos;

public:

Table(int \_maxsize);

virtual void Insert(TKey k, TData\* d) = 0;

virtual void Remove(TKey k) = 0;

virtual TabRecord<TKey, TData>\* Find(TKey k) = 0;

bool IsFull() const;

bool IsEmpty() const;

virtual bool IsTabEnded() const;

virtual bool Next();

virtual bool Reset();

int GetCount() const;

virtual TKey GetKey() const = 0;

virtual TData\* GetData() const = 0;

};

Назначение: представление таблицы.

Поля:

**Сount** – количество записей в таблице .

**maxsize** – максимальный размер таблицы.

**currPos** – текущая позиция.

Методы:

Table(int \_maxsize);

Назначение: конструктор класса, инициализирует таблицу с заданным максимальным размером.

Входные параметры: **\_maxsize** - максимальный размер таблицы.

Выходные параметры: отсутствуют.

virtual void Insert(TKey k, TData\* d) = 0;

Назначение: абстрактный метод для вставки элемента с ключом k и данными d в таблицу.

Входные параметры:

**k** – ключ элемента.

**d** – указатель на данные элемента.

Выходные параметры: отсутствуют.

virtual void Remove(TKey k) = 0;

Назначение: абстрактный метод для удаления элемента по ключу **k** из таблицы.

Входные параметры: **k** – ключ элемента, который требуется удалить.

Выходные параметры: отсутствуют.

virtual TabRecord<TKey, TData>\* Find(TKey k) = 0;

Назначение: абстрактный метод для поиска элемента по ключу **k** в таблице.

Входные параметры: **k** – ключ элемента, который требуется найти.

Выходные параметры: указатель на объект типа **TabRecord** для найденного элемента.

bool IsFull() const;

Назначение: проверяет, заполнена ли таблица полностью.

Входные параметры: отсутствуют.

Выходные параметры: возвращает true, если таблица полная, иначе false.

bool IsEmpty() const;

Назначение: проверяет, пуста ли таблица.

Входные параметры: отсутствуют.

Выходные параметры: возвращает true, если таблица пуста, иначе false.

virtual bool IsTabEnded() const;

Назначение: проверяет, завершено ли перебор всех элементов таблицы.

Входные параметры: отсутствуют.

Выходные параметры: возвращает true, если перебор завершен, иначе false.

virtual bool Next();

Назначение: переходит к следующему элементу в таблице.

Входные параметры: отсутствуют.

Выходные параметры: возвращает true, если переход выполнен успешно, иначе false.

virtual bool Reset();

Назначение: сбрасывает позицию текущего элемента в таблице.

Входные параметры: отсутствуют.

Выходные параметры: возвращает true, если сброс выполнен успешно, иначе false.

int GetCount() const;

Назначение: возвращает количество элементов в таблице.

Входные параметры: отсутствуют.

Выходные параметры: количество элементов в таблице.

virtual TKey GetKey() const = 0;

Назначение: возвращает ключ текущего элемента.

Входные параметры: отсутствуют.

Выходные параметры: ключ текущего элемента.

virtual TData\* GetData() const = 0;

Назначение: Возвращает данные текущего элемента.

Входные параметры: отсутствуют.

Выходные параметры: указатель на данные текущего элемента.

### Описание класса TScanTable

template <class TKey, class TData>

class TScanTable : public Table<TKey,TData>

{

protected:

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

public:

TScanTable(int \_maxsize);

TScanTable(const TScanTable& st);

~TScanTable();

void Insert(TKey k, TData\* data);

void Remove(TKey k);

TabRecord<TKey, TData>\* Find(TKey k);

TKey GetKey() const;

TData\* GetData() const;

friend ostream& operator<<(ostream& out, const TScanTable<TKey, TData>& st)

{

TScanTable<TKey, TData> t(st);

if(!t.Reset());

for (int i = 0; i < t.GetCount(); i++)

{

out << \*(t.GetData());

t.Next();

}

return out;

}

};

Назначение: представление просматриваемой таблицы.

Поля:

**recs** – массив указателей на объекты типа **TabRecord**.

Методы:

TScanTable(int \_maxsize);

Назначение: конструктор, инициализирует таблицу с заданным максимальным размером.

Входные параметры: **\_maxsize** – максимальный размер таблицы сканирования.

Выходные параметры: отсутствуют.

TScanTable(const TScanTable& st);

Назначение: конструктор копирования, создает копию таблицы на основе существующей.

Входные параметры: **st** – ссылка на существующий объект класса **TScanTable**, который требуется скопировать.

Выходные параметры: скопированный объект типа **TScanTable**.

~TScanTable();

Назначение: деструктор, освобождает ресурсы, занимаемые таблицей.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

void Insert(TKey k, TData\* data);

Назначение: вставляет элемент с ключом **k** и данными **data** в таблицу.

Входные параметры:

**k** – ключ элемента.

**data** – указатель на данные элемента.

Выходные параметры: отсутствуют.

void Remove(TKey k);

Назначение: удаляет элемент с ключом **k** из таблицы.

Входные параметры: **k** ключ элемента, который требуется удалить.

Выходные параметры: отсутствуют.

TabRecord<TKey, TData>\* Find(TKey k);

Назначение: ищет элемент по ключу **k** в просматриваемой таблице.

Входные параметры: **k** – ключ элемента, который требуется найти.

Выходные параметры: указатель на объект типа **TabRecord** для найденного элемента.

TKey GetKey() const;

Назначение: возвращает ключ текущего элемента в просматриваемой таблице.

Входные параметры: отсутствуют.

Выходные параметры: ключ текущего элемента.

TData\* GetData() const;

Назначение: возвращает данные текущего элемента в просматриваемой таблице.

Входные параметры: отсутствуют.

Выходные параметры: указатель на данные текущего элемента.

friend ostream& operator<<(ostream& out, const TScanTable<TKey, TData>& st)

Назначение: оператор вывода для класса **TRingList**.

Входные параметры:

**os** – ссылка на объект типа **ostream**, который представляет выходной поток.

**st** – ссылка на объект типа **TScanTable<TKey, TData>** который будет выводиться.

Выходные параметры: ссылка на объект типа **ostream**.

### Описание класса TSortTable

template <class TKey, class TData>

class TSortTable : public TScanTable<TKey, TData>

{

private:

void Sort(int left, int right);

public:

TSortTable(int \_maxsize);

TSortTable(const TScanTable\* st);

TSortTable(const TSortTable& rst);

void Insert(TKey k, TData\* data);

void Remove(TKey k);

TabRecord<TKey, TData>\* Find(TKey k);

friend ostream& operator<<(ostream& out, const TSortTable<TKey, TData>& st)

{

TSortTable<TKey, TData> t(st);

if (!t.Reset());

for (int i = 0; i < t.GetCount(); i++)

{

out << \*(t.GetData());

t.Next();

}

return out;

}

};

Назначение: представление сортированной таблицы.

Методы:

void Sort(int left, int right);

Назначение: производит сортировку элементов таблицы с использованием алгоритма быстрой сортировки (Quick Sort).

Входные параметры:

**left** – индекс левой границы сортируемой области.

**right** – индекс правой границы сортируемой области.

Выходные параметры: отсутствуют.

TSortTable(int \_maxsize);

Назначение: конструктор, инициализирует таблицу с заданным максимальным размером.

Входные параметры: **\_maxsize** – максимальный размер сортированной таблицы.

Выходные параметры: отсутствуют.

TSortTable(const TScanTable\* st);

Назначение: конструктор класса, создает сортированную таблицу на основе существующей просматриваемой таблице.

Входные параметры: **st** – указатель на существующий объект класса **TScanTable**, который требуется отсортировать.

Выходные параметры: сортированная таблица.

TSortTable(const TSortTable& rst);

Назначение: конструктор копирования, создает копию таблицы на основе существующей.

Входные параметры: **rst** – ссылка на существующий объект класса **TSortTable**, который требуется скопировать.

Выходные параметры: скопированный объект типа **TSortTable**.

void Insert(TKey k, TData\* data);

Назначение: вставляет элемент с ключом **k** и данными **data** в таблицу.

Входные параметры:

**k** – ключ элемента.

**data** – указатель на данные элемента.

Выходные параметры: отсутствуют.

void Remove(TKey k);

Назначение: удаляет элемент с ключом **k** из таблицы.

Входные параметры: **k** ключ элемента, который требуется удалить.

Выходные параметры: отсутствуют.

TabRecord<TKey, TData>\* Find(TKey k);

Назначение: ищет элемент по ключу **k** в сортированной таблице.

Входные параметры: **k** – ключ элемента, который требуется найти.

Выходные параметры: указатель на объект типа **TabRecord** для найденного элемента.

friend ostream& operator<<(ostream& out, const TSortTable<TKey, TData>& st)

Назначение: оператор вывода для класса **TSortTable**.

Входные параметры:

**os** – ссылка на объект типа **ostream**, который представляет выходной поток.

**st** – ссылка на объект типа **TSortTable** который будет выводиться.

Выходные параметры: ссылка на объект типа **ostream**.

* + 1. Описание класса THashTable

template<class TKey,class TData>

class THashTable : public Table<TKey, TData>

{

protected:

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

public:

THashTable(int size) : Table<TKey, TData>(size) {};

};

Назначение: представление хеш-таблицы.

Методы:

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

Назначение: Вычисляет хэш-код ключа.

Входные параметры: key – ключ для вычисления хэш-кода.

Выходные параметры: целочисленное значение, представляющее хэш-код ключа..

THashTable(int size) : Table<TKey, TData>(size) {};

Назначение: Создает экземпляр хеш-таблицы заданного размера.

Входные параметры: **size** – максимальный размер хэш-таблицы.

Выходные параметры: отсутствуют.

* + 1. Описание класса TArrayHashTable

template<class TKey, class TData>

class TArrayHashTable : public THashTable<TKey, TData>

{

protected:

size\_t hash\_step;

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

TabRecord<TKey, TData>\* pMark;

int freePos;

size\_t GetNextPos(size\_t ind) { return (ind + hash\_step) % maxsize; }

public:

TArrayHashTable(size\_t \_maxsize, size\_t \_step);

TArrayHashTable(const TArrayHashTable& ht);

virtual ~TArrayHashTable();

void Insert(TKey k, TData\* d);

void Remove(TKey k);

TabRecord<TKey, TData>\* Find(TKey k);

bool Reset();

bool Next();

TKey GetKey() const ;

TData\* GetData() const;

friend ostream& operator<<(ostream& out, const TArrayHashTable<TKey, TData>& h)

{

TArrayHashTable<TKey, TData> ht(h);

if (!ht.Reset())

{

while (!ht.IsTabEnded())

{

if (ht.recs[ht.currPos] != nullptr && ht.recs[ht.currPos] != ht.pMark)

out << \*(ht.GetData());

ht.Next();

}

}

return out;

};

};

Назначение: представление хеш-таблицы с повторным перемешиванием.

Поля:

**hash\_step** – шаг хэширования.

**recs** – массив указателей на записи.

**pMark** – отметка, что объект был удален.

**freePos** – это индекс свободной позиции в массиве.

Методы:

TArrayHashTable(size\_t \_maxsize, size\_t \_step);

Назначение: конструктор, инициализирует таблицу с заданным максимальным размером и шагом.

Входные параметры:

**\_maxsize** – максимальный размер хэш-таблицы.

**\_step** – размер шага.

Выходные параметры: отсутствуют.

TArrayHashTable(const TArrayHashTable& ht);

Назначение: конструктор копирования, создает копию таблицы на основе существующей.

Входные параметры: **ht** – ссылка на существующий объект класса **TArrayHashTable**, который требуется скопировать.

Выходные параметры: скопированный объект типа **TArrayHashTable**.

virtual ~TarrayHashTable();

Назначение: деструктор, освобождает ресурсы, занимаемые таблицей.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

void Insert(TKey k, TData\* d);

Назначение: вставляет элемент с ключом **k** и данными **data** в хэш-таблицу.

Входные параметры:

**k** – ключ элемента.

**data** – указатель на данные элемента.

Выходные параметры: отсутствуют.

void Remove(TKey k);

Назначение: удаляет элемент с ключом **k** из хэш-таблицы.

Входные параметры: **k** ключ элемента, который требуется удалить.

Выходные параметры: отсутствуют.

TabRecord<TKey, TData>\* Find(TKey k);

Назначение: ищет элемент по ключу **k** в хэш-таблице.

Входные параметры: **k** – ключ элемента, который требуется найти.

Выходные параметры: указатель на объект типа **TabRecord** для найденного элемента.

bool Reset();

Назначение: сбрасывает позицию текущего элемента в таблице.

Входные параметры: отсутствуют.

Выходные параметры: возвращает true, если сброс выполнен успешно, иначе false.

bool Next();

Назначение: переходит к следующему элементу в таблице.

Входные параметры: отсутствуют.

Выходные параметры: возвращает true, если переход выполнен успешно, иначе false.

TKey GetKey() const ;

Назначение: возвращает ключ текущего элемента в просматриваемой таблице.

Входные параметры: отсутствуют.

Выходные параметры: ключ текущего элемента.

TData\* GetData() const;

Назначение: возвращает данные текущего элемента в просматриваемой таблице.

Входные параметры: отсутствуют.

Выходные параметры: указатель на данные текущего элемента.

friend ostream& operator<<(ostream& out, const TArrayHashTable<TKey, TData>& h)

Назначение: оператор вывода для класса **TArrayHashTable**.

Входные параметры:

**os** – ссылка на объект типа **ostream**, который представляет выходной поток.

**h** – ссылка на объект типа **TArrayHashTable** который будет выводиться.

Выходные параметры: ссылка на объект типа **ostream**.

size\_t GetNextPos(size\_t ind) { return (ind + hash\_step) % maxsize; }

Назначение: Возвращает следующую позицию в таблице на основе заданного индекса.

Входные параметры: **ind** – текущий индекс в таблице.

Выходные параметры: новый индекс, представляющий следующую позицию в таблице.

# Заключение

В ходе выполнения лабораторной работы была реализована иерархия структур данных для работы с таблицами, включая структуры данных Table, TScanTable, TSortTable, THashTable и TArrayHashTable. Каждая из этих структур имеет свои методы и поля, позволяющие работать с таблицами в соответствии с их типом.

Класс Table является базовым классом для всех остальных классов и содержит общие методы и поля для работы с таблицами.

Класс TScanTable наследуется от Table и реализует простую таблицу, в которой элементы хранятся в массиве и могут быть доступны в любом порядке.

Класс TSortTable наследуется от TScanTable и реализует упорядоченную таблицу, в которой элементы хранятся в отсортированном порядке.

Класс THashTable является базовым классом для хэш-таблиц и содержит методы для работы с хеш-функцией.

Класс TArrayHashTable наследуется от THashTable и реализует хэш-таблицу с открытым адресованием, в которой элементы хранятся в массиве и могут быть доступны с помощью хэш-функции.

В ходе выполнения лабораторной работы были реализованы все необходимые методы для работы с таблицами, включая вставку, удаление, поиск и вывод элементов. Также были реализованы операторы ввода-вывода для классов TScanTable, TSortTable и TArrayHashTable.

# Литература

1. Лекция «Таблицы» Сысоева А.В. [https://cloud.unn.ru/s/RiYcLtYDCWJdpC3]
2. Лекция «Хэш-таблицы» Сысоева А.В. [https://cloud.unn.ru/s/RiYcLtYDCWJdpC3]

# Приложения

Приложение А. Реализация класса TabRecord

template <class TKey, class TData>

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

template <class TKey, class TData>

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

{

key = \_key;

data = new TData(\*\_data);

}

template <class TKey, class TData>

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

{

key = tr.key;

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

}

template <class TKey, class TData>

TKey TabRecord<TKey, TData>::GetKey() const

{

return key;

}

template <class TKey, class TData>

TData\* TabRecord<TKey, TData>::GetData() const

{

return data;

}

Приложение Б. Реализация класса Table

template <class TKey, class TData>

Table<TKey, TData>::Table(int \_maxsize)

{

Count = 0;

maxsize = \_maxsize;

currPos = -1;

}

template <class TKey, class TData>

bool Table<TKey, TData>::IsFull() const

{

return Count == maxsize;

}

template <class TKey, class TData>

bool Table<TKey, TData>::IsEmpty() const

{

return Count == 0;

}

template <class TKey, class TData>

bool Table<TKey, TData>::IsTabEnded() const

{

return currPos >= maxsize;

}

template <class TKey, class TData>

bool Table<TKey, TData>::Reset()

{

if (!IsEmpty())

currPos = 0;

else

currPos = -1;

return IsTabEnded();

}

template <class TKey, class TData>

bool Table<TKey, TData>::Next()

{

if (!IsTabEnded())

currPos++;

return IsTabEnded();

}

template <class TKey, class TData>

int Table<TKey, TData>::GetCount() const

{

return Count;

}

Приложение В. Реализация класса TScanTable

template <class TKey, class TData>

TScanTable<TKey, TData>::TScanTable(int \_maxsize) : Table<TKey, TData>(\_maxsize)

{

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

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

recs[i] = nullptr;

}

template <class TKey, class TData>

TScanTable<TKey, TData>::TScanTable(const TScanTable& st) : Table<TKey, TData>(st.maxsize)

{

Count = st.Count;

currPos = st.currPos;

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

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

{

if (st.recs[i] != nullptr)

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

else

recs[i] = nullptr;

}

}

template <class TKey, class TData>

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

{

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

{

if (recs[i] != nullptr)

delete recs[i];

}

delete[] recs;

}

template <class TKey, class TData>

void TScanTable<TKey, TData>::Insert(TKey k, TData\* data)

{

if (IsFull())

{

string msg = "Error: scan table is full";

throw msg;

}

recs[Count++] = new TabRecord<TKey,TData>(k, data);

}

template <class TKey, class TData>

void TScanTable<TKey, TData>::Remove(TKey k)

{

if (IsEmpty())

{

string msg = "Error: scan table is empty";

throw msg;

}

if (Find(k) != nullptr)

{

delete recs[currPos];

for (int i = currPos; i < this->Count; i++)

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

Count--;

}

else

{

string msg = "Error: the element to be deleted was not found";

throw msg;

}

}

template <class TKey, class TData>

TabRecord<TKey, TData>\* TScanTable<TKey, TData>::Find(TKey k)

{

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

{

if (recs[i]->GetKey() == k)

{

currPos = i;

return recs[i];

}

}

return nullptr;

}

template <class TKey, class TData>

TKey TScanTable<TKey, TData>::GetKey() const

{

if (currPos < Count)

return recs[currPos]->GetKey();

else

{

string msg = "Error: Current position is out of range";

throw msg;

}

}

template <class TKey, class TData>

TData\* TScanTable<TKey, TData>::GetData() const

{

if (currPos < Count)

return recs[currPos]->GetData();

else

{

string msg = "Error: Current position is out of range";

throw msg;

}

}

Приложение Г. Реализация класса TSortTable

template <class TKey, class TData>

TSortTable<TKey, TData>::TSortTable(int \_maxsize) : TScanTable<TKey, TData>(\_maxsize) {}

template <class TKey, class TData>

TSortTable<TKey, TData>::TSortTable(const TScanTable<TKey, TData>\* st) : TScanTable<TKey, TData>(\*st)

{

Sort(0,Count-1);

}

template <class TKey, class TData>

TSortTable<TKey, TData>::TSortTable(const TSortTable& rst) : TScanTable<TKey, TData>(rst) {}

template <class TKey, class TData>

TabRecord<TKey, TData>\* TSortTable<TKey, TData>::Find(TKey k)

{

int i1 = 0;

int i2 = Count - 1;

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

while (i1 <= i2)

{

int mid = (i1 + i2) / 2;

if (recs[mid]->GetKey() == k)

{

Search = recs[mid];

i1 = mid + 1;

i2 = mid;

}

else if (k > recs[mid]->GetKey())

i1 = mid+1;

else

i2 = mid-1;

}

currPos = i2;

return Search;

}

template <class TKey, class TData>

void TSortTable<TKey, TData>::Insert(TKey k, TData\* data)

{

if (IsFull())

{

string msg = "Error: sorted table is full";

throw msg;

}

Find(k);

for (int i = Count - 1; i > currPos; i--)

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

recs[currPos+1] = new TabRecord<TKey, TData>(k, data);

Count++;

//this->Sort();

}

template <class TKey, class TData>

void TSortTable<TKey, TData>::Remove(TKey k)

{

if (IsEmpty())

{

string msg = "Error: sorted table is empty";

throw msg;

}

TabRecord<TKey, TData>\* s = Find(k);

if (s == nullptr)

{

string msg = "Error: the element to be deleted was not found";

throw msg;

}

delete s;

for (int i = currPos; i < Count; i++)

{

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

}

Count--;

}

template <class TKey, class TData>

void TSortTable<TKey, TData>::Sort(int left, int right)

{

int i = left;

int j = right;

int mid = (i + j) / 2;

do {

while (recs[i]->GetKey() < recs[mid]->GetKey())

i++;

while (recs[j]->GetKey() > recs[mid]->GetKey())

j--;

if (i <= j)

{

if(i != j)

{

TabRecord<TKey, TData>\* tmp = recs[i];

recs[i] = recs[j];

recs[j] = tmp;

}

i++;

j--;

}

} while (i <= j);

if (left < j)

Sort(left, j);

if (i < right)

Sort(i, right);

}

Приложение Д. Реализация класса THashTable

template<class TKey, class TData>

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

{

size\_t hash = 0;

for (char c : key)

hash += c;

return hash % maxsize;

}

Приложение Е. Реализация класса TarrayHashTable

template<class TKey, class TData>

TArrayHashTable<TKey, TData>::TArrayHashTable(size\_t \_maxsize, size\_t \_step) : THashTable<TKey, TData>(\_maxsize)

{

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

hash\_step = \_step;

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

freePos = -1;

for (int i = 0; i < \_maxsize; i++)

recs[i] = nullptr;

}

template<class TKey, class TData>

TArrayHashTable<TKey, TData>::TArrayHashTable(const TArrayHashTable& ht) : THashTable<TKey, TData>(ht.maxsize) //

{

maxsize = ht.maxsize;

currPos = ht.currPos;

hash\_step = ht.hash\_step;

freePos = ht.freePos;

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

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

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

{

TabRecord<TKey, TData>\* tmp = ht.recs[i];

if (ht.recs[i] == ht.pMark) {

recs[i] = pMark;

}

else if (ht.recs[i] == nullptr) {

recs[i] = nullptr;

}

else

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

}

}

template<class TKey, class TData>

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

{

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

{

if (recs[i] != nullptr && recs[i] != pMark)

delete recs[i];

}

if (recs)

delete[] recs;

if (pMark)

delete pMark;

}

template<class TKey, class TData>

TabRecord<TKey, TData>\* TArrayHashTable<TKey, TData>::Find(TKey k)

{

currPos = hash\_func(k);

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

{

if (recs[currPos] == nullptr)

return nullptr;

else if (recs[currPos] == pMark && freePos == -1)

freePos = currPos;

else if (recs[currPos]->GetKey() == k)

return recs[currPos];

currPos = GetNextPos(currPos);

}

return nullptr;

}

template<class TKey, class TData>

void TArrayHashTable<TKey, TData>::Insert(TKey k, TData\* d)

{

if (IsFull())

{

string msg = "Error: hash table is full";

throw msg;

}

if (Find(k) != nullptr && freePos != -1)

currPos = freePos;

recs[currPos] = new TabRecord<TKey, TData>(k, d);

Count++;

}

template<class TKey, class TData>

void TArrayHashTable<TKey, TData>::Remove(TKey k)

{

TabRecord<TKey, TData>\* tmp = Find(k);

if (tmp == nullptr)

{

string msg = "Error: hash table is empty";

throw msg;

}

delete tmp;

recs[currPos] = pMark;

freePos = -1;

Count--;

}

template<class TKey, class TData>

bool TArrayHashTable<TKey, TData>::Reset()

{

currPos = 0;

while (!IsTabEnded())

{

if (recs[currPos] != nullptr && recs[currPos] != pMark)

break;

currPos++;

}

return IsTabEnded();

}

template<class TKey, class TData>

bool TArrayHashTable<TKey, TData>::Next()

{

currPos++;

while (!IsTabEnded())

{

if (recs[currPos] != nullptr && recs[currPos] != pMark)

break;

currPos++;

}

return IsTabEnded();

}

template<class TKey, class TData>

TKey TArrayHashTable<TKey,TData>::GetKey() const

{

if (currPos < maxsize)

return recs[currPos]->GetKey();

else

{

string msg = "Error: Current position is out of range";

throw msg;

}

}

template<class TKey, class TData>

TData\* TArrayHashTable<TKey,TData>::GetData() const

{

if (currPos < maxsize)

return recs[currPos]->GetData();

else

{

string msg = "Error: Current position is out of range";

throw msg;

}

}