МИНИСТЕРСТВО НАУКИ И ВЫСШЕГООБРАЗОВАНИЯ

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»**

**Выполнила:** студентка группы 3822Б1ФИ2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Фролова Е.А.

Подпись

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

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

Подпись

Нижний Новгород  
2024

**Содержание**

[Введение 3](#_Toc168568830)

[1 Постановка задачи 4](#_Toc168568831)

[2 Руководство пользователя 5](#_Toc168568832)

[2.1 Приложение для демонстрации работы таблиц, содержащих полиномы 5](#_Toc168568833)

[3 Руководство программиста 12](#_Toc168568834)

[3.1 Описание алгоритмов 12](#_Toc168568835)

[3.1.1 Просматриваемая таблица 12](#_Toc168568836)

[3.1.2 Отсортированные таблицы 14](#_Toc168568837)

[3.1.3 Хэш-таблица 16](#_Toc168568838)

[3.2 Описание программной реализации 18](#_Toc168568839)

[3.2.1 Схема наследования классов 18](#_Toc168568840)

[3.2.2 Описание класса TabRecord 19](#_Toc168568841)

[3.2.3 Описание структуры Table 20](#_Toc168568842)

[3.2.4 Описание класса TScanTable 23](#_Toc168568843)

[3.2.5 Описание TSortedTable 24](#_Toc168568844)

[3.2.6 Описание THashTasble 26](#_Toc168568845)

[3.2.7 Описание TArrayHashTable 26](#_Toc168568846)

[Заключение 29](#_Toc168568847)

[Литература 30](#_Toc168568848)

[Приложения 31](#_Toc168568849)

[Приложение А. Реализация класса TTabRecord 31](#_Toc168568850)

[Приложение Б. Реализация класса Table 32](#_Toc168568851)

[}Приложение В. Реализация класса TScanTable 34](#_Toc168568852)

[Приложение Г. Реализация класса TSortedTable 36](#_Toc168568853)

[Приложение Е Реализация классе THashTable 39](#_Toc168568854)

[Приложение Ж Реализация классе TArrayHashTable 40](#_Toc168568855)

# Введение

В предыдущей лабораторной работе мы реализовали структуру данных для представления и обработки полиномов, реализовав основные операции над ними, такие как сложение, вычитание, умножение и дифференцирование.

В данной лабораторной работе мы исследуем различные структуры таблиц для эффективного хранения и обработки полиномов.

В данной лабораторной работе мы сосредоточимся на реализации и использовании различных структур таблиц. Реализуем такие таблицы: просматриваемая таблица, отсортированная таблица, хэш-таблица. Изучим преимущества и недостатки каждой структуры данных в контексте работы с полиномами.

# Постановка задачи

Цель – реализация различных структур таблиц для хранения и обработки полиномов:

1. Создание хеш-таблицы и реализация функций функций:

* Функция хеширования, которая преобразует ключ элемента в индекс хеш-таблицы.
* Функция разрешения коллизий.
* Функция вставки элемента, которая добавляет новый элемент в хеш-таблицу.
* Функция поиска элемента, которая находит элемент по его ключу в хеш-таблице.
* Функция удаления элемента, которая удаляет элемент из хеш-таблицы.

1. Создание просматриваемой таблицы и реализация функций:

* Функция вставки элемента, которая добавляет новый элемент в просматриваемые таблицу.
* Функция поиска элемента, которая находит элемент по его ключу в просматриваемой таблице.
* Функция удаления элемента, которая удаляет элемент из просматриваемой таблицы.

1. Создание отсортированной таблицы и реализация:

* Функция вставки элемента, которая добавляет новый элемент в отсортированную таблицу с учетом сортировки.
* Функция поиска элемента, которая находит элемент по его ключу в отсортированной таблице с использованием бинарного поиска.
* Функция удаления элемента, которая удаляет элемент из отсортированной таблицы с учетом сортировки.

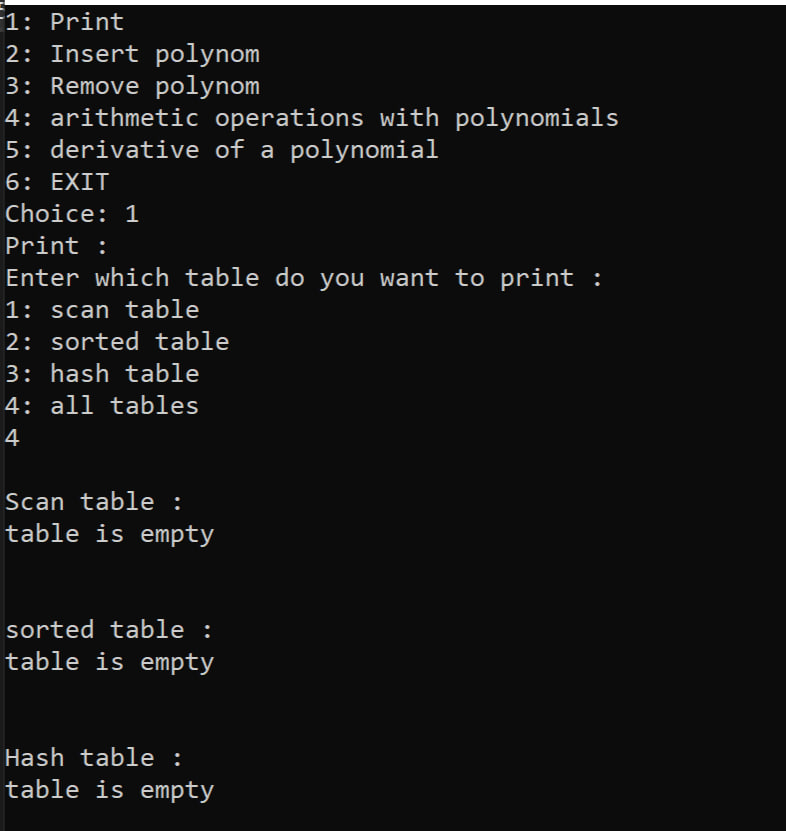
Добавление этих функций поможет нам полностью реализовать каждый тип таблицы и использовать их для конкретных операций вставки, поиска и удаления элементов.

# Руководство пользователя

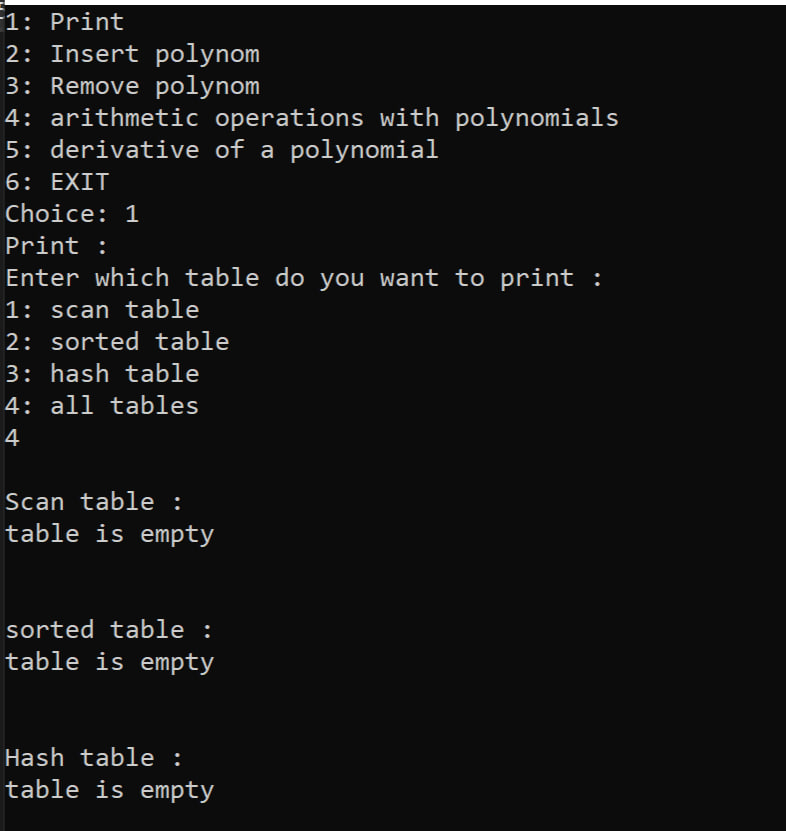
## Приложение для демонстрации работы таблиц, содержащих полиномы

1. Запускаем приложение с названием sample\_table.exe. В результате появится окно, показанное ниже (рис. 1). Здесь пользователю представлено меню что можно сделать в таблицах:

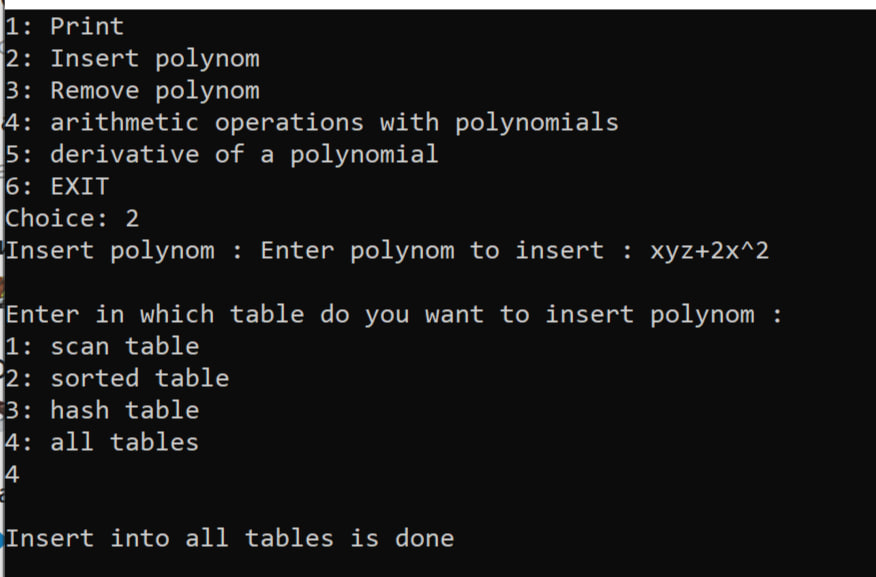
* Вывести таблицу.
* Внести полином.
* Удалить полином из таблицы.
* Произвести арифметические операции с полиномами в таблицах.
* Взять производные от полиномов в таблицах.



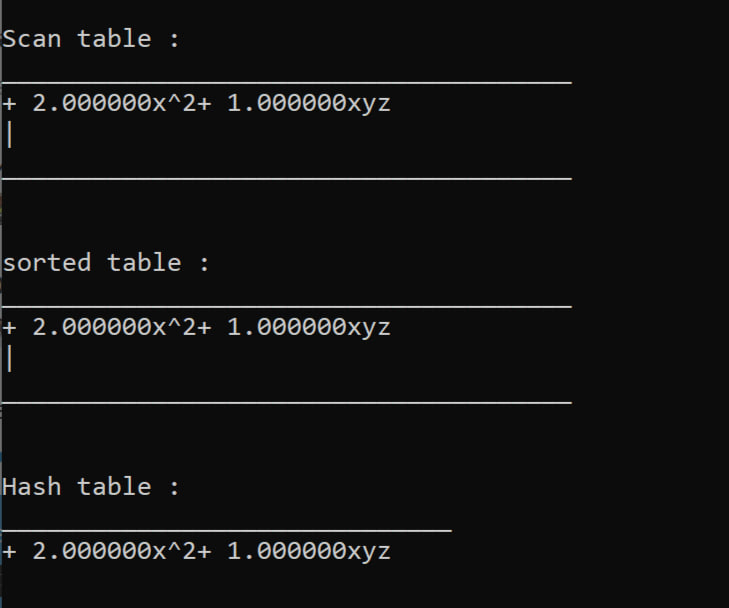
1. Основное окно программы.
2. Выведем таблицы, выбрав первый пункт, далее программа предложит нам вывести определенные таблицы или все таблицы сразу. Выбираем последнее (рис. 2).



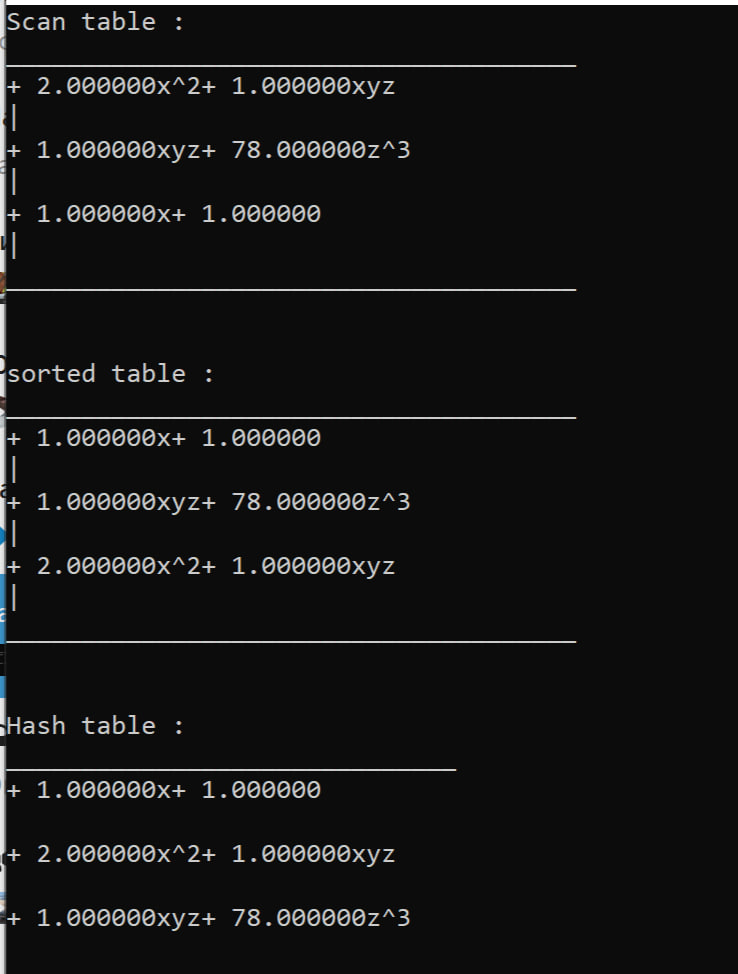
1. Выведены таблицы.
2. Внесем полином во все таблицы. Выбираем в меню второй пункт(Insert polynom), далее программа предложит ввести полином и покажет варианты таблиц в которых может находиться нывый полином. Внесем опять во все таблицы (рис. 3) и проверим результат (рис. 4).



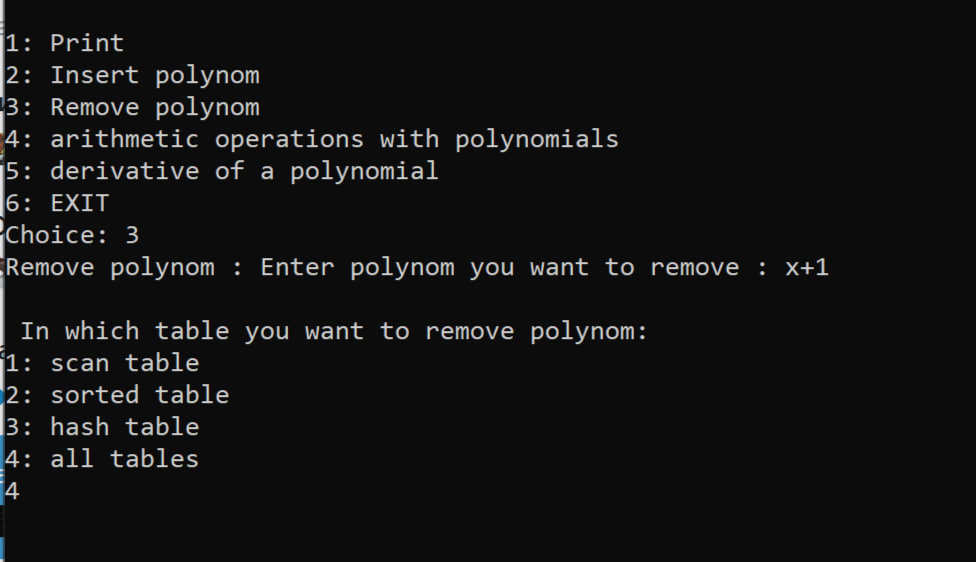
1. Новый полином записан в таблицы.



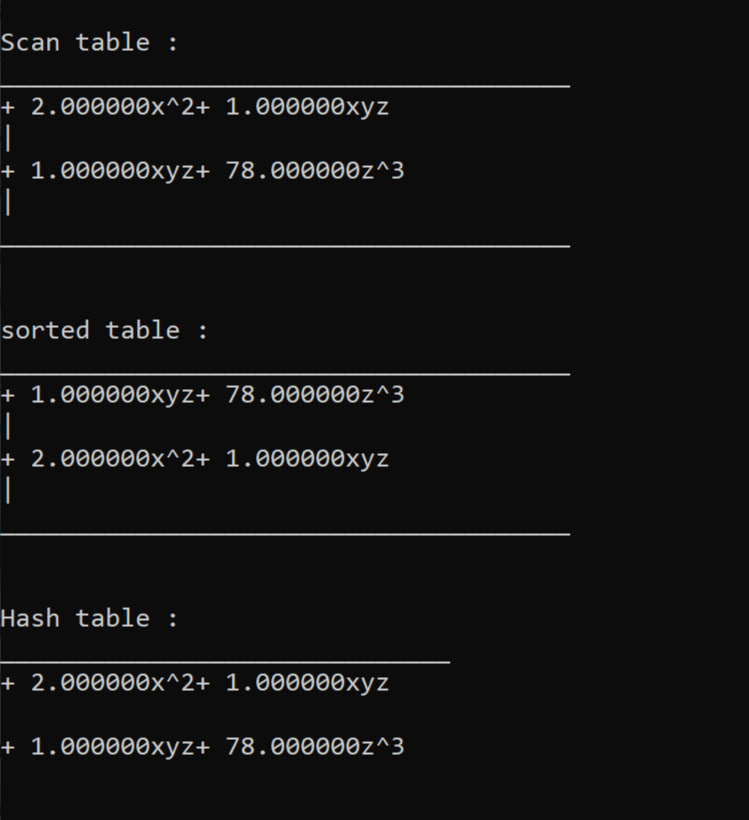
1. Полином в таблицах.
2. Таким же методом внесем ещё два полинома и выведем таблички (рис. 5).



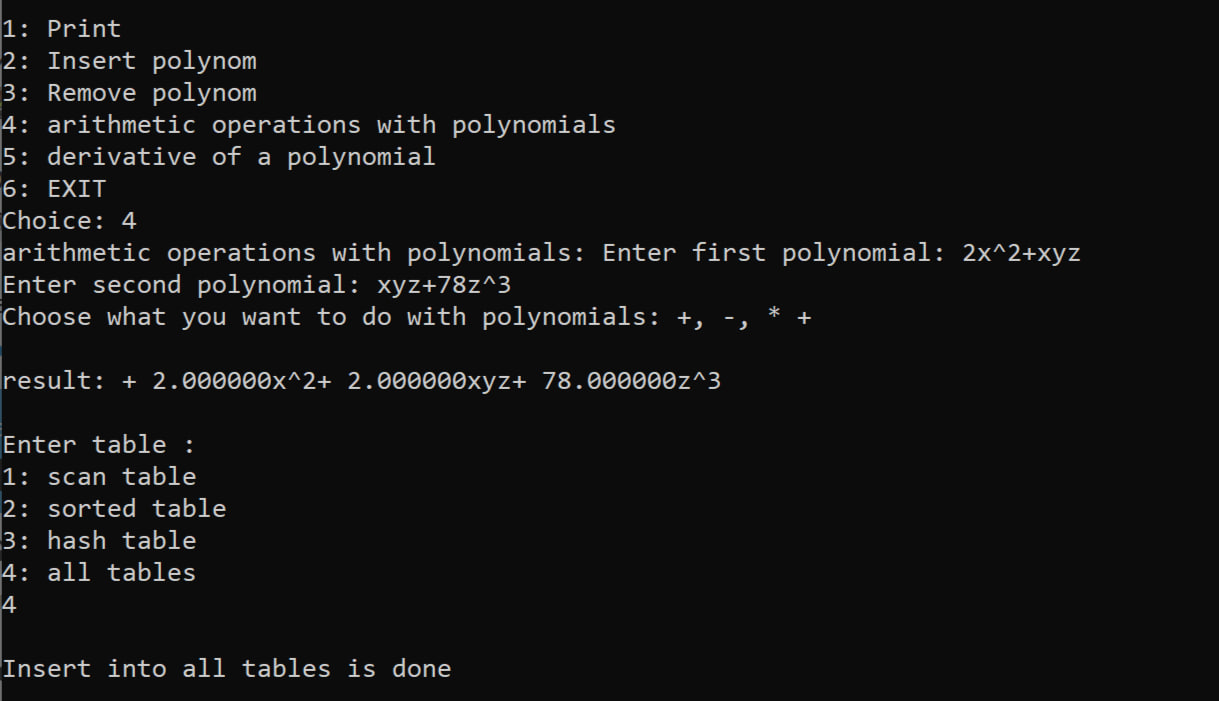
1. Все три полинома находятся в таблицах.
2. Удалим один из полиномов существующих в таблицах. Для этого в меню выбираем третий пункт (Remove polynom). Далее опять выберем чтобы полином был удален из всех таблиц (рис. 6).



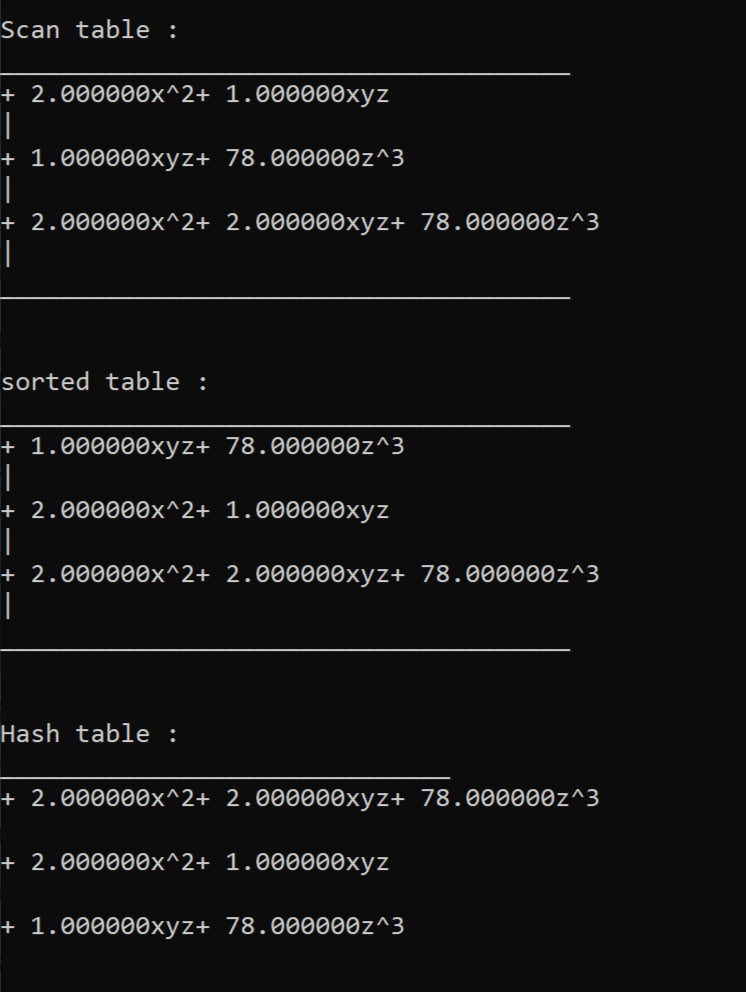
1. Удаление полинома.
2. Выведем кк сейчас выглядят таблицы (рис. 7).



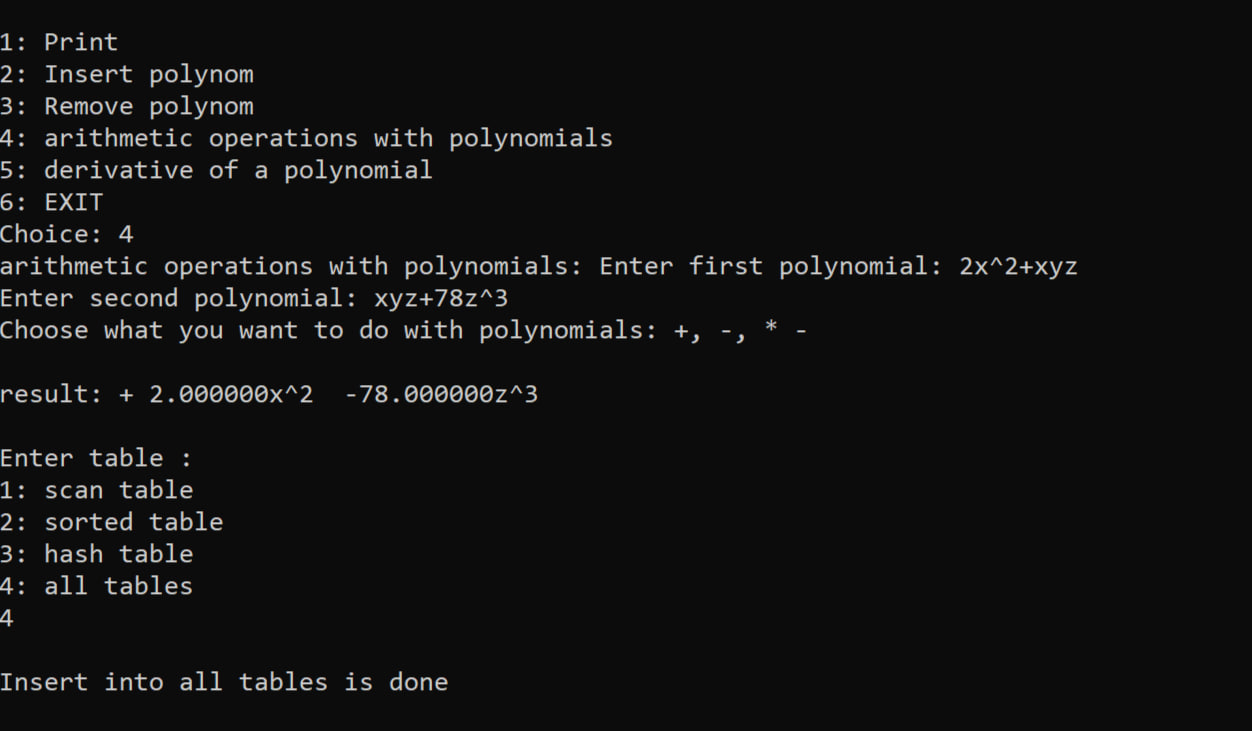
1. Удален полином.
2. Произведем арифметические операции с полиномами, находящимися в таблицах. Для этого в предложенном меню выбираем четвертый пункт и вводим два полинома с которыми хотим выполнить операцию (Полиномы должны существовать в любой из таблиц!). Далее выбираем действие которое хотим произвести с полиномами. Программа выведет нам результат операции и предложит поместить его в определенные таблицы, так же выберем все таблицы (рис. 8).



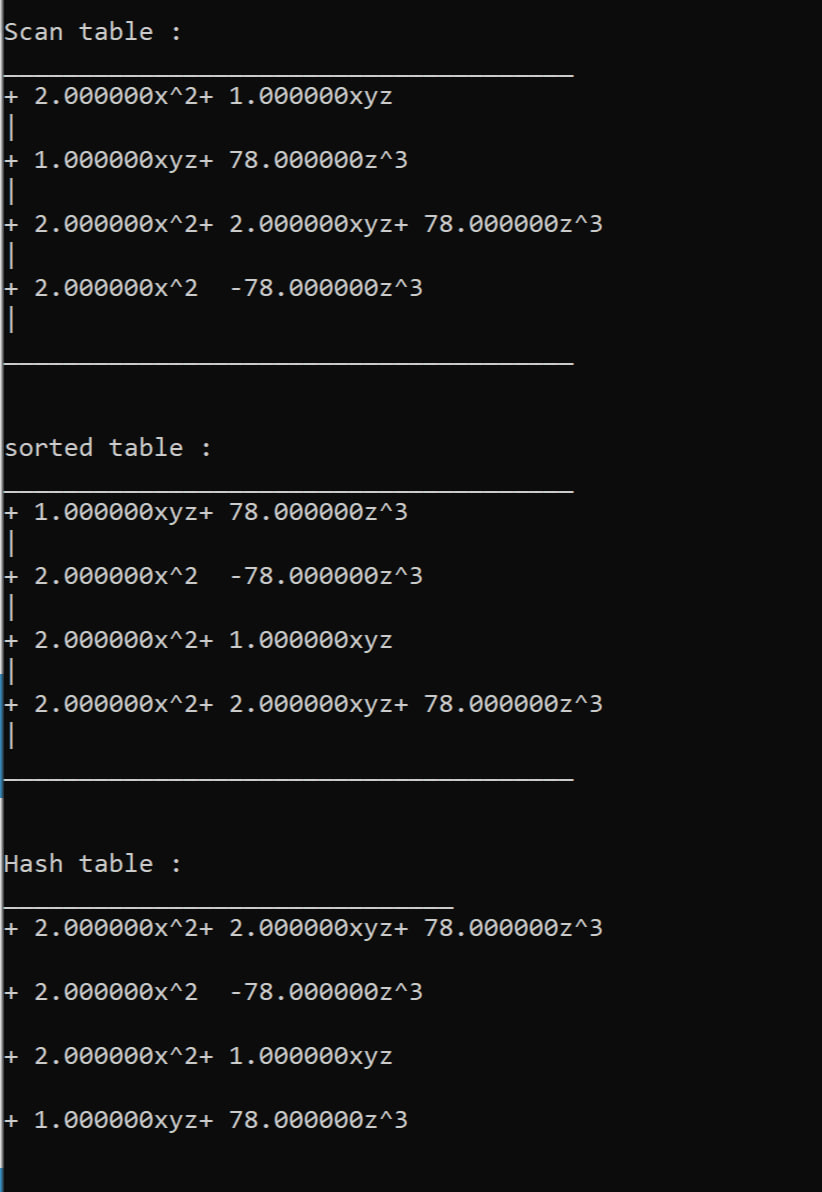
1. Арифметические операции с полиномами.
2. Выведем таблицы (рис. 9).



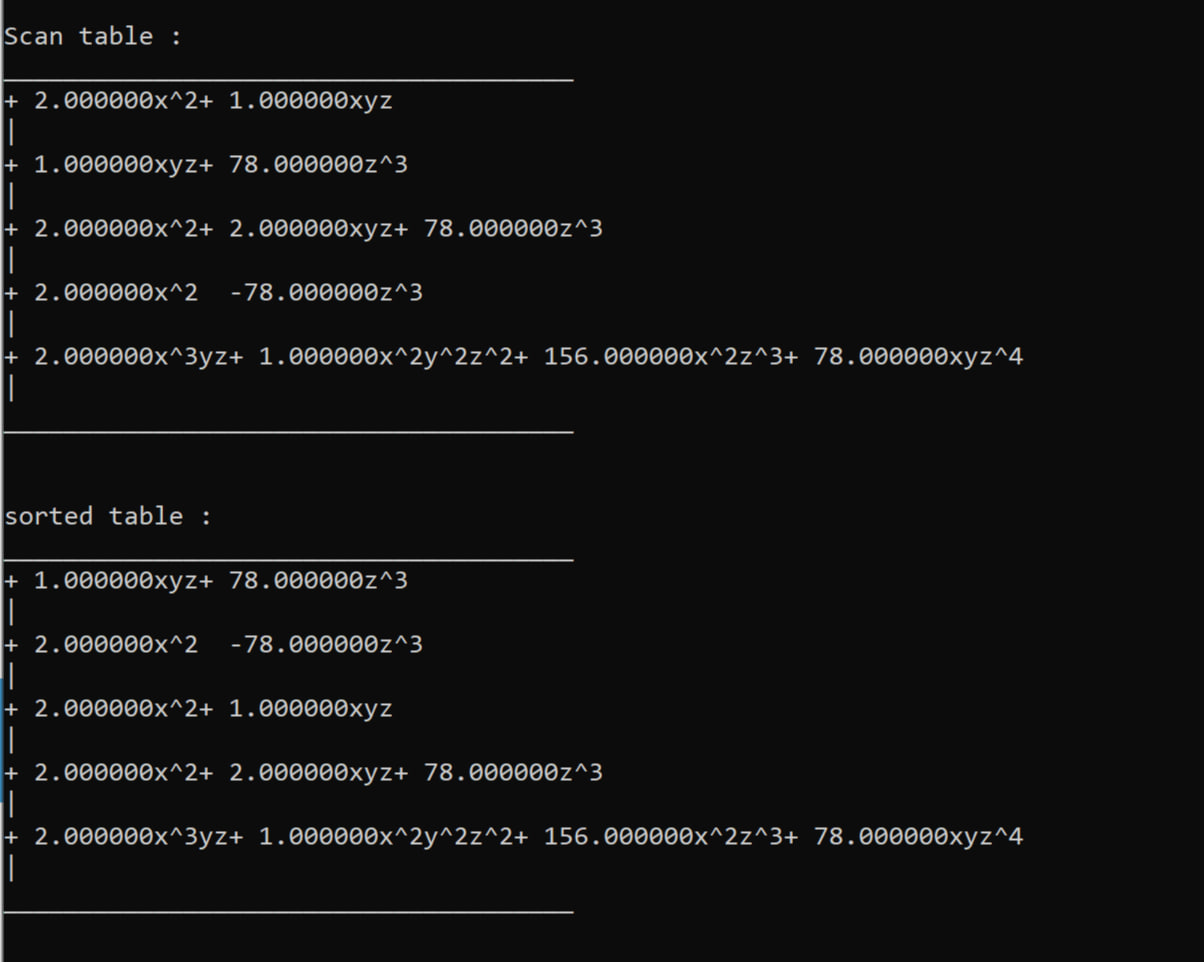
1. Результат сложения полиномов внесен в таблицы.
2. Так же ля операции вычитания выбираем четвертый пункт, вводим полиномы и выбираем действие, поместим результат операции во все таблицы (рис. 10). Выведем таблицы (рис. 11).



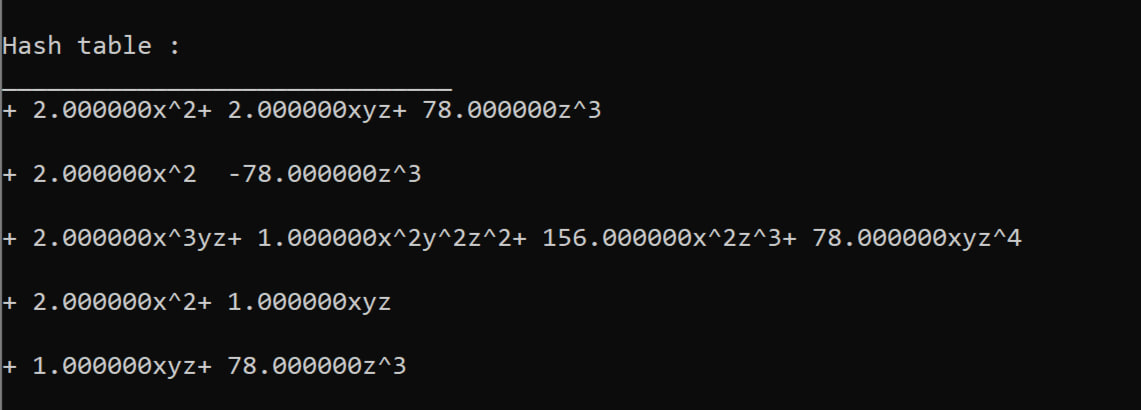
1. Вычитание полиномов.



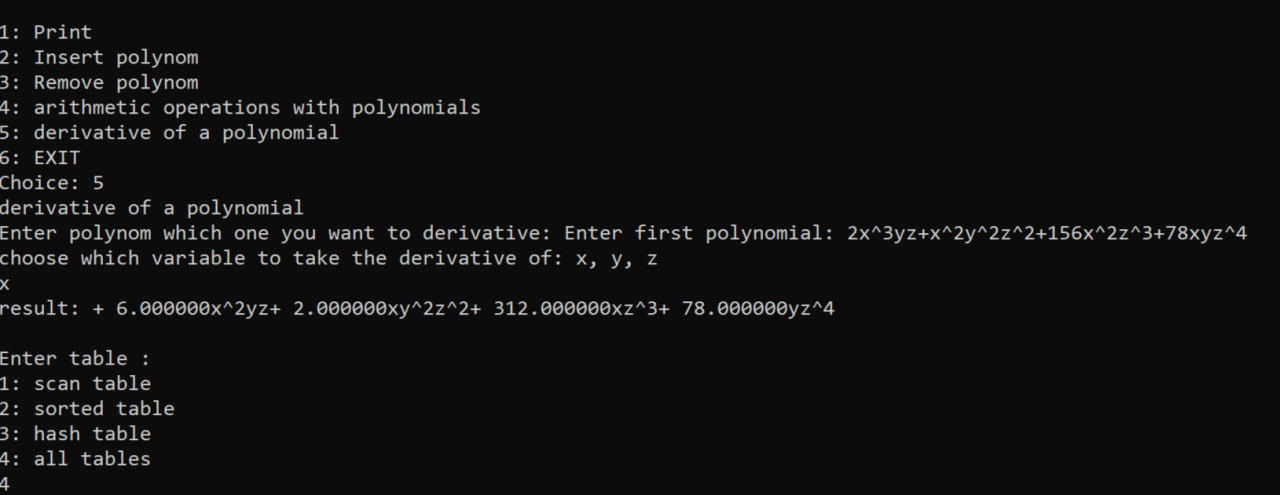
1. Результат арифметической операции в таблице.
2. Точно так же перемножим введеные полиномы и выведем результат (рис. 12) (рис. 13).



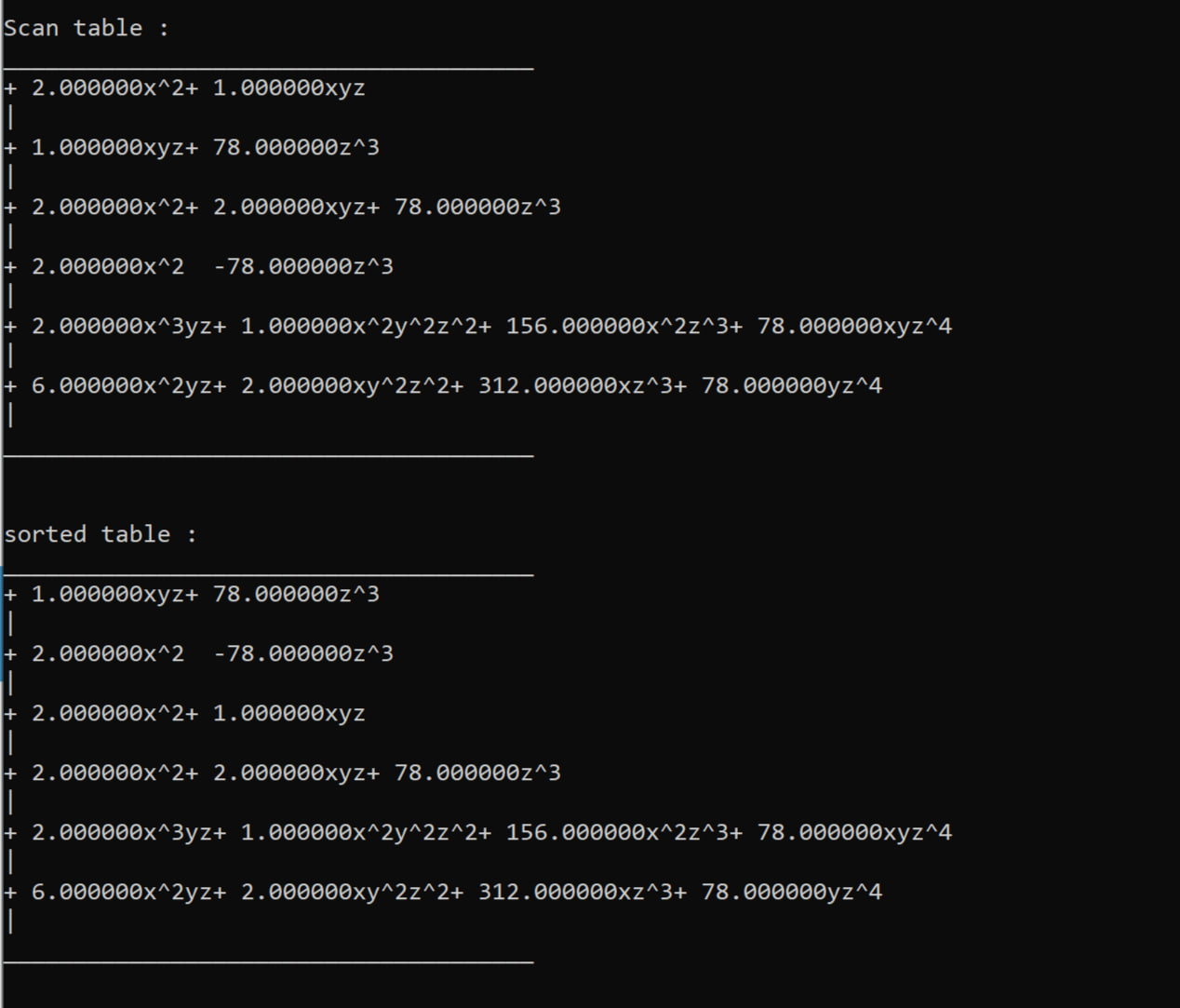
1. Основное окно программы.



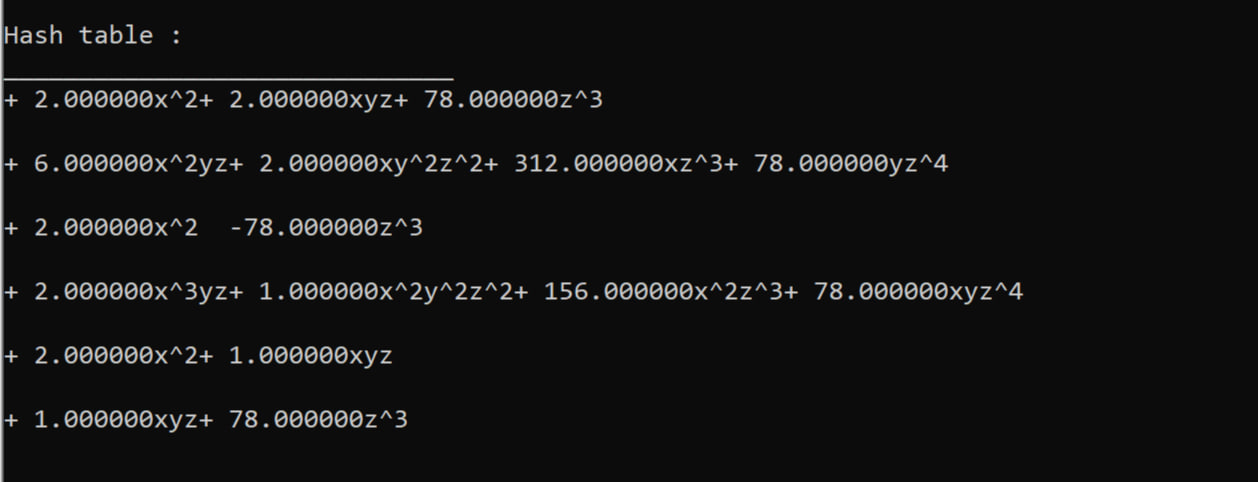
1. Основное окно программы.
2. Возьмем производную от полинома, существующего в таблице. Для этого в предложенном меню вибираем пятый пункт, вводим полином от которого хотим взять производную (Полином должен находиться в любой из таблиц!), далее выбираем переменную по которй будембрать производную. Помещаем результат во все таблицы (рис. 14).



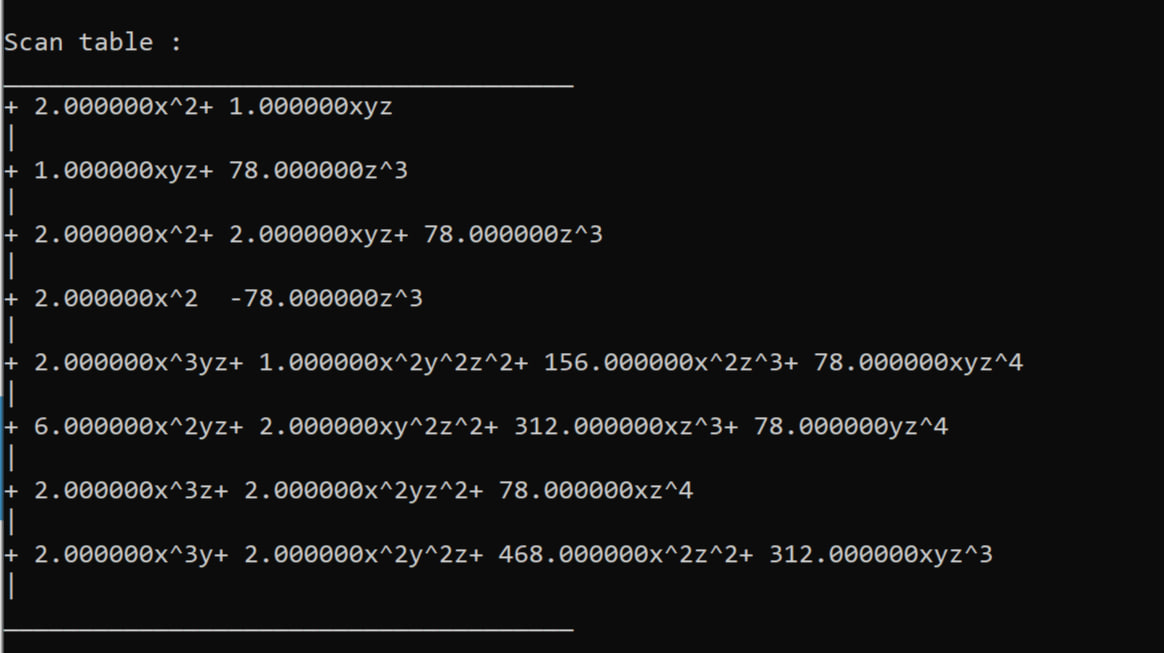
1. Производная от полинома.
2. Выведем таблицы (Рис. 15) (Рис. 16).



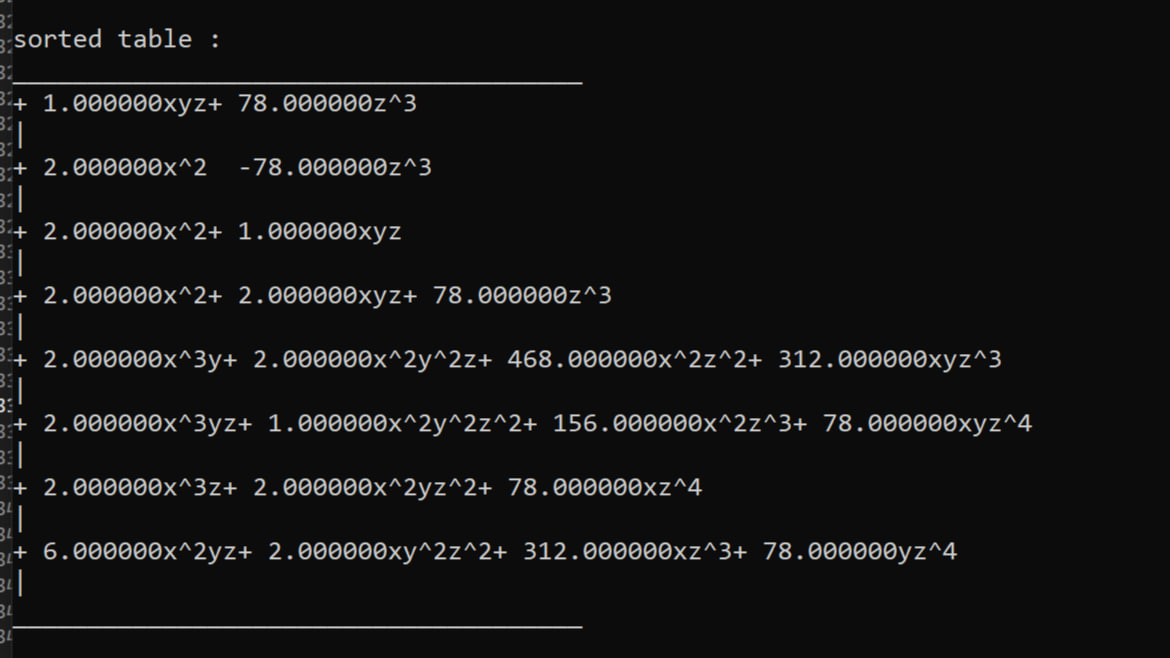
1. Результат взятия производной от полинома в таблицах.



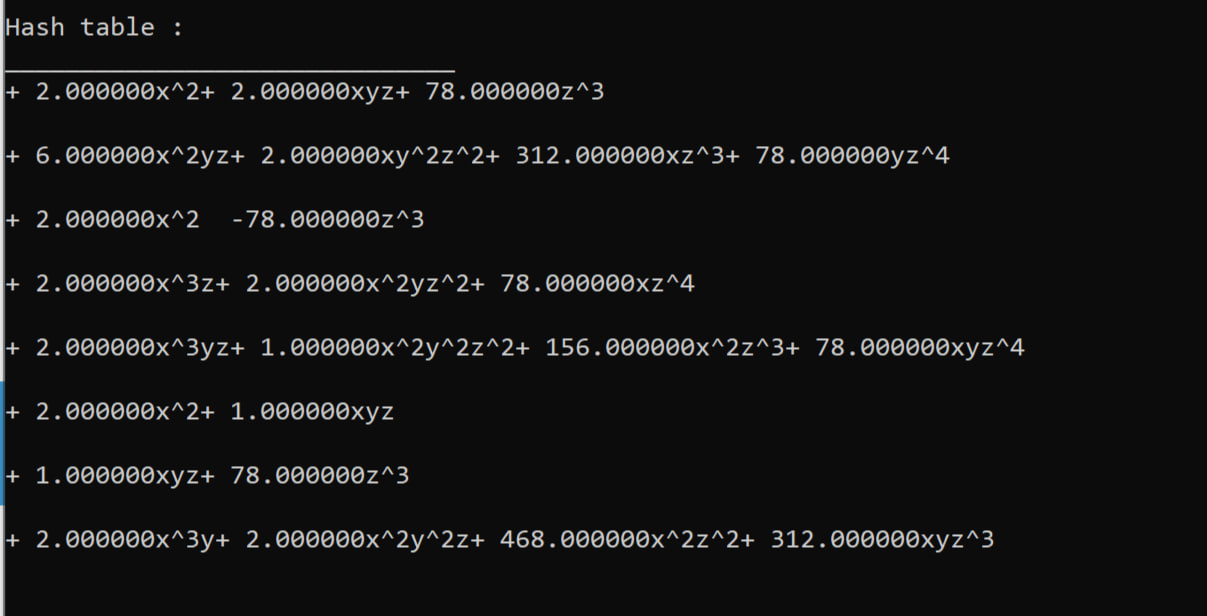
1. Производная.
2. Точно так же берем производную от полинома в таблице по другим переменным. Выведем результат (рис. 17) (рис. 18) (рис. 19).



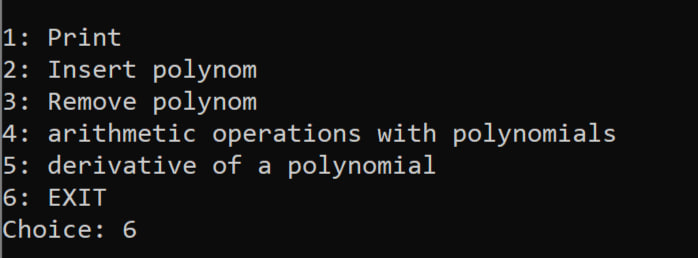
1. Просматриваемая таблица.



1. Отсортированная таблица.



1. Хэш-таблица.
2. Для завершения работы программы в предложенном меню выбираем шестой пункт (EXIT) (рис. 20).



1. Выход из программы.

# Руководство программиста

## Описание алгоритмов

### Просматриваемая таблица

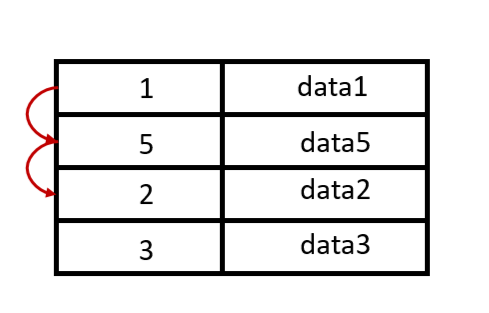
Просматриваемые таблицы - это таблица, порядок расположения записей в которой произволен по отношению к значениям ключей.

Рассмотри основные операции:

1. **Поиск.**

* Перебирать все записи в таблице (с начала и до конца).
* Для каждой записи проверяем ключ этой записи с ключом, который мы ищем.
* Если ключи совпадают, прерываем обход, так как нашли нужную запись.

Пример: ишем запись, имеющую ключ 2 (рис. 21).

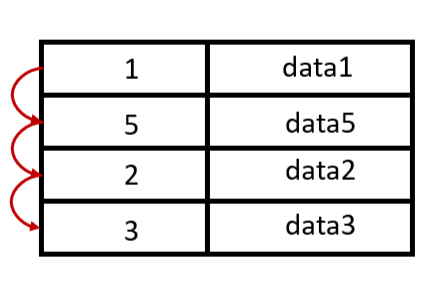


1. Операция поиска.
2. **Вставка.**

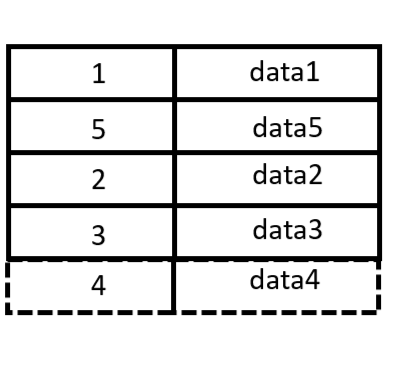
* Проверяем, не заполнена ли уже таблица.
* Если таблица заполнена, вставка невозможна.
* Иначе ищем запись с заданным ключом в таблице.
* Если запись найдена, запись так же не возможна потому что такой ключ уже есть.
* Если запись не найдена, то создаем новую запись с заданным ключом и данными.
* Увеличиваем счетчик количества записей в таблице.

Пример:

Вставим запись с ключом 4.



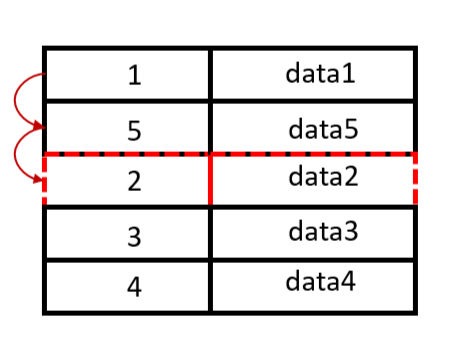
1. Выполнили обход поля с таким ключом нет.



1. Выполнили вставку в конец.
2. **Удаление.**

* Проверяем, не пуста ли таблица.
* Если таблица пуста, удалять нечего.
* Ищем запись с заданным ключом в таблице.
* Если запись найдена, удаляем ее и перемещаем последнюю запись вместо удаленной.
* Уменьшаем счетчик количества записей в таблице.
* Если запись не найдена, удалять нечего.

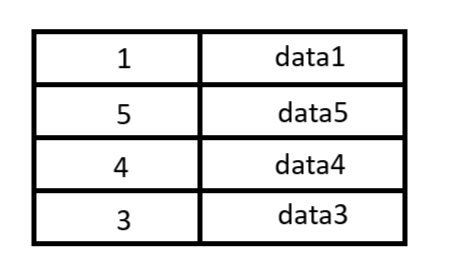
Пример: удалим запись с ключом 2.



1. Нашли запись с ключом 2, удаляем.



1. На место этой записи ставим последнбб в таблице.



1. Результат.

### Отсортированные таблицы

Отсортированная таблица - это таблица, записи в которой расположены в порядке «возрастания/убывания» ключей.

Особенности основных операций:

* Поиск: так как таблица отсортирована, можно использовать эффективные алгоритмы поиска, такие как бинарный поиск, для быстрого нахождения записи по ключу.
* Вставка: при вставке новой записи необходимо найти правильное место для вставки в отсортированной последовательности. Это может потребовать сдвига существующих записей для освобождения места.
* Удаление: при удалении записи также необходимо сохранить порядок сортировки. Обычно это включает поиск записи по ключу и удаление ее, а затем возможное пересортирование или сдвиг оставшихся записей.

1. **Поиск**

Бинарный поиск. Как он работает:

1. Находим середину текущего диапазона.
2. Сравниваем ключ искомого элемента с ключом элемента в середине.
3. Если ключи совпадают, мы нашли элемент и возвращаем его.
4. Если ключ искомого элемента больше, чем ключ в середине, мы ищем справа от середины.
5. Если ключ искомого элемента меньше, чем ключ в середине, мы ищем слева от середины.
6. Повторяем шаги 1-5, пока не найдем элемент или пока левая граница не станет больше правой.

Пример:

* Предположим, у нас есть таблица с ключами [3, 6, 8, 11, 15, 21, 25]. Мы ищем ключ 11.
* Начинаем с левой границы (3) и правой границы (25). Находим середину (8).
* 11 больше 8, поэтому сдвигаем левую границу на 1 позицию правее середины.
* Теперь рассматриваем правую половину: [11, 15, 21, 25].
* Находим новую середину (15).
* 11 меньше 15, поэтому сдвигаем правую границу на 1 позицию левее середины.
* Рассматриваем левую половину: [11].
* Найден элемент с ключом 11.

1. **Вставка**
2. Проверяем, не заполнена ли уже наша таблица. Если да, то вставку нельзя выполнить.
3. Ищем элемент с таким же ключом в таблице. Если находим, также втавка невозможна.
4. Если элемент уникален, создаем новую запись с ключом и данными.
5. Находим позицию, куда нужно вставить новый элемент в отсортированной таблице.
6. Сдвигаем все элементы, которые идут после позиции вставки, на одну позицию вправо.
7. Вставляем новый элемент на найденную позицию.
8. Увеличиваем счетчик элементов.

Пример:

Предположим, у нас есть отсортированная таблица с ключами [3, 6, 8, 11, 15] и соответствующими данными. Мы хотим добавить элемент с ключом 9 и данными "Данные9".

* Проверяем, не заполнена ли уже наша таблица.
* Ищем элемент с ключом 9. Не находим.
* Создаем новую запись с ключом 9 и данными "Данные9".
* Находим позицию для вставки нового элемента (между 8 и 11).
* Сдвигаем элементы после этой позиции на одну позицию вправо.
* Вставляем новый элемент на найденную позицию.
* Увеличиваем счетчик элементов.

1. **Удаление**
2. Проверяем, не пуста ли наша таблица. Если да, то удаление невозможно.
3. Ищем элемент с заданным ключом в таблице. Если не находим, удалять нечего.
4. Если элемент найден, удаляем его из таблицы.
5. Сдвигаем все элементы, которые идут после удаленного, на одну позицию влево.
6. Уменьшаем счетчик элементов.

Пример:

Предположим, у нас есть отсортированная таблица с ключами [3, 6, 8, 9, 11, 15] и соответствующими данными. Мы хотим удалить элемент с ключом 9.

* Проверяем, не пуста ли наша таблица.
* Ищем элемент с ключом 9. Находим.
* Удаляем элемент с ключом 9.
* Сдвигаем все элементы после удаленного (11 и 15) на одну позицию влево [3,6,8,11,15].
* Уменьшаем счетчик элементов.

### Хэш-таблица

Хэш-таблица - это структура данных, которая использует хэширование для хранения и быстрого поиска пар "ключ-значение". Основная идея заключается в том, что каждому ключу соответствует определенное значение (или набор значений), и хэш-таблица позволяет быстро находить значение по ключу.

1. **Хэширование**

Процесс хэширования в хэш-таблицах обычно включает в себя следующие шаги:

1. Ключ преобразуется в хэш (целое число) с использованием хэш-функции. Целью хорошей хэш-функции является равномерное распределение хэшей для различных ключей, чтобы минимизировать коллизии.
2. Полученный хэш используется как индекс массива (или другой структуры данных) для быстрого доступа к значению, связанному с этим ключом.
3. **Разрешение коллизий**

Коллизия в хэш-таблицах происходит, когда два или более ключа хэшируются в одно и то же значение хэш-функции. В идеальной ситуации каждый ключ должен иметь уникальное значение хэша, но из-за ограничений размера таблицы и конечного количества возможных хэшей, коллизии могут возникать.

Когда коллизия происходит, необходимо принять решение, как разрешить эту ситуацию. Существует несколько способов управления коллизиями:

Один из наиболее распространенных метод открытого перемешивания (Open Addressing): При использовании этого метода, если возникает коллизия, новый элемент помещается в другую доступную ячейку таблицы. Методом линейного пробирования ищется следующая свободная ячейка путем последовательного пробирования по индексам таблицы.

1. **Вставка**

Вставка элемента в хэш-таблицу включает несколько шагов:

1. Хэширование ключа: сначала ключ элемента хэшируется с использованием хэш-функции. Хэш-функция преобразует ключ в индекс ячейки внутри хэш-таблицы.
2. Определение места вставки: полученный хэш используется для определения места, куда следует вставить элемент в таблицу.
3. Разрешение коллизий: если на месте вставки уже находится другой элемент (коллизия), то используется метод разрешения коллизий, такой как метод цепочек или метод открытого перемешивания, чтобы разместить элемент в таблице.
4. Вставка элемента: после разрешения коллизии элемент вставляется на свое место в таблице.
5. **Поиск**

Алгоритм поиска в хэш-таблице с использованием линейного пробирования можно разбить на следующие этапы:

1. Вычисление хеша: алгоритм начинается с вычисления хеша для ключа элемента. Хеш-функция преобразует ключ в индекс таблицы, где элемент должен быть расположен.
2. Поиск элемента: алгоритм проверяет, есть ли элемент с таким хешем в таблице. Если элемент не найден, он переходит к следующему этапу.
3. Линейное пробирование: в случае отсутствия элемента с заданным хешем, алгоритм использует метод линейного пробирования, перемещаясь на следующую ячейку в таблице с учетом шага хеширования. Этот шаг повторяется до тех пор, пока не будет найден элемент с соответствующим ключом или пока не будет пройдена вся таблица.
4. Поиск свободных ячеек: если элемент не найден, но в таблице есть свободные ячейки, алгоритм возвращает информацию об их наличии.
5. Занятые ячейки: если все ячейки заняты, алгоритм возвращает информацию об отсутствии искомого элемента.
6. Обработка удаленных ячеек: в алгоритме предусмотрено специальное поведение при обнаружении "удаленной" ячейки (pMark), чтобы продолжить поиск следующего свободного места для возможной вставки нового элемента.
7. **Удаление**

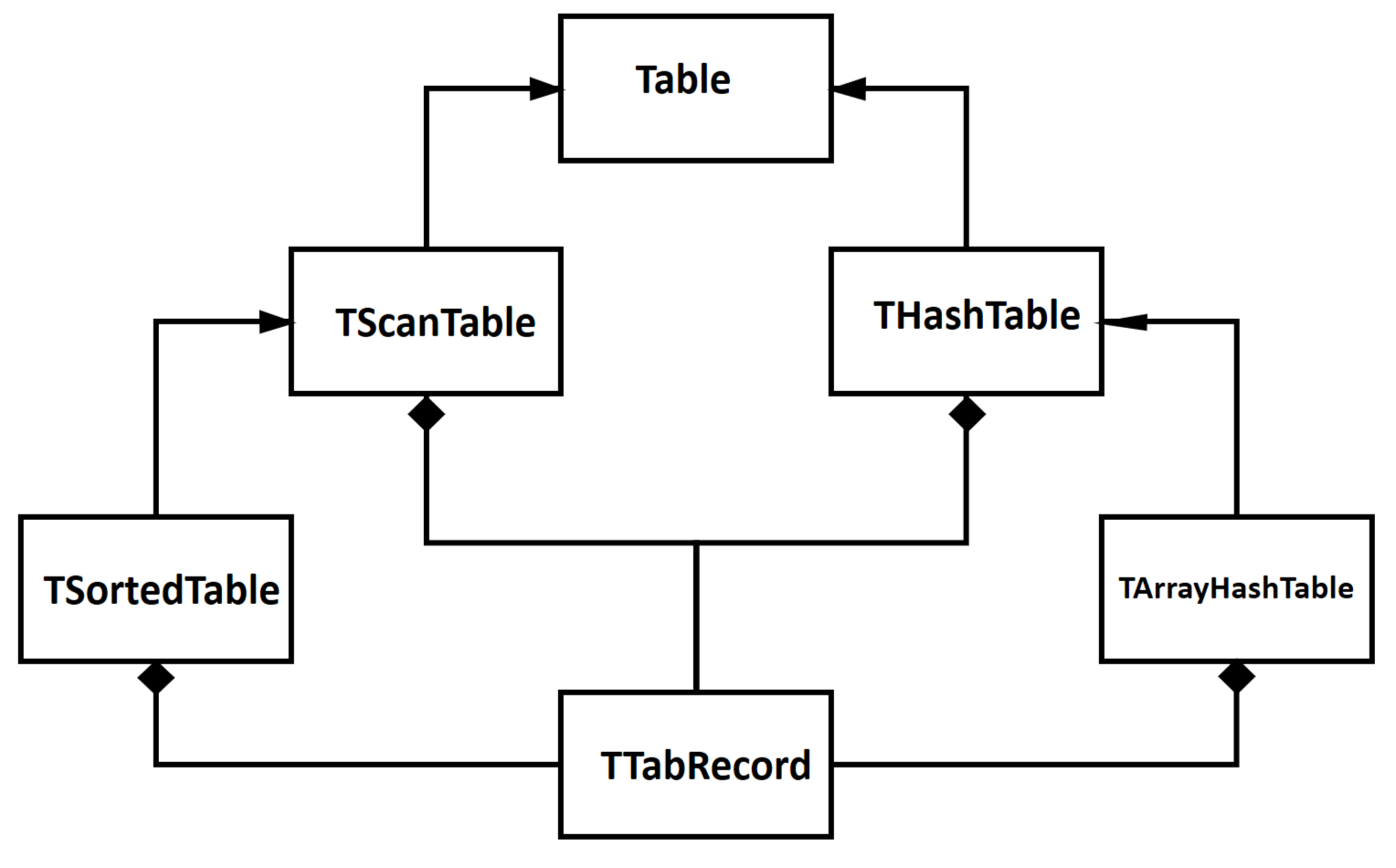
Алгоритм, описывающий, удаление в хэш-таблице:

1. Проверка наличия элементов: Сначала проверяется, содержит ли таблица элементы. Если таблица пуста, удалять нечего.
2. Поиск элемента: выполняется метод для поиска элемента по ключу. Если элемент не найден, удалять нечего.
3. Удаление элемента: найденный элемент удаляется из таблицы путем пометки соответствующей ячейки как свободной.

## Описание программной реализации

### Схема наследования классов

* Table:
* Базовый (абстрактный) класс для всех таблиц. Он содержит общие методы и свойства, которые должны быть реализованы в наследуемых классах.
* Методы и свойства: Вставка, удаление, поиск записей, общие свойства таблиц.
* TScanTable (наследуется от Table):
* Реализует линейный (последовательный) доступ к данным.
* Методы: Вставка, удаление, поиск с использованием линейного сканирования.
* TSortedTable (наследуется от TScanTable):
* Добавляет функциональность сортировки к последовательной таблице.
* Методы: Вставка с поддержанием порядка, бинарный поиск, удаление с поддержанием порядка.
* THashTable (наследуется от Table):
* Использует хэширование для организации данных.
* Методы: Хэш-функция, вставка с хэшированием, удаление, поиск по хэш-ключу.
* TArrayHashTable (наследуется от THashTable):
* Реализует хэш-таблицу с использованием массивов.
* Методы: Обработка коллизий, вставка и удаление с учетом хэширования, управление массивами.
* TTabRecord:
* Представляет запись (элемент данных), используемую в таблицах.
* Методы и свойства: Поля для ключа и данных, методы сравнения записей.
* TTabRecord используется всеми вышеупомянутыми классами для представления данных.



1. Схема наследования классов

### Описание класса TabRecord

template<typename TKey, typename TData>

class TTabRecord

{

public:

TKey key;

TData\* data;

public:

TTabRecord();

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

TTabRecord(const TTabRecord<TKey, TData>& record);

~TTabRecord();

TKey GetKey() const;

TData\* GetData() const;

const TTabRecord<TKey, TData>& operator=(const TTabRecord<TKey, TData>& TabRec);

};

Назначение: представление записи в таблице.

Поля:

**key** - поле типа **TKey**, представляющее ключ записи.

**data** - указатель на данные типа **TData**, связанные с ключом.

Методы:

TTabRecord()

Назначение: конструктор по умолчанию. Создает пустую запись.

TTabRecord(const TKey& \_key, const TData\* \_data)

Назначение:

конструктор с параметрами. Создает запись с заданным ключом и данными.

Входные данные:

**\_key** - ключ записи.

**\_data** - указатель на данные типа **TData**.

TTabRecord(const TTabRecord<TKey, TData>& record)

Назначение: конструктор копирования. Создает копию существующей записи.

Входные параметры: **record**- копируемый объект.

~TTabRecord()

Назначение: деструктор. Освобождает ресурсы, связанные с записью.

TKey GetKey() const

Назначение: метод, возвращающий ключ записи.

Входные параметры: отсутствуют.

Выходные параметры: ключ записи типа **TKey**.

TData\* GetData() const

Назначение: метод, возвращающий указатель на данные.

Входные параметры: отсутствуют.

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

const TTabRecord<TKey, TData>& operator=(const TTabRecord<TKey, TData>& TabRec)

Назначение: перегруженный оператор присваивания. Присваивает одну запись другой.

Входные параметры: запись типа **TTabRecord<TKey, TData>**, которую нужно присвоить.

Выходные параметры: ссылка на объект типа **TTabRecord<TKey, TData>**, который был присвоен.

### Описание структуры Table

class Table

{

protected:

int count;

int maxSize;

int currPos;

public:

Table(int max\_size = 100);

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

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

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

bool IsFull() const noexcept;

bool IsEmpty() const noexcept;

bool IsTabEnded() const noexcept;

virtual bool Reset() noexcept;

virtual bool Next() noexcept;

int GetCurrPos() const noexcept;

int GetMaxSize() const noexcept;

int GetCount() const noexcept;

};

Назначение: предназначен для представления абстрактного типа данных таблицы, который может хранить пары ключ-значение.

Поля:

**count** – переменная для обозначения количества элементов в таблице;

**maxSize** – переменная для представления размера таблицы.

**currPos** – переменная представляющая позицию текущего элемента.

Методы:

Table(int max\_size = 100)

Назначение: создает новую таблицу с заданным размером **max\_size**.

Входные параметры:

**max\_size** - размер таблицы.

Выходные параметры: нет

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

Назначение: вставляет новую пару ключ-данные в таблицу (абстрактный метод).

Входные параметры:

**\_key**: ключ новой пары ключ-данные.

**\_data**: данные, связанные с ключом.

Выходные параметры: нет.

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

Назначение: удаляет пару ключ-данные по указанному ключу (абстрактный метод).

Входные параметры:

**\_key**: ключ пары ключ-данные, которую нужно удалить.

Выходные параметры: нет

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

Назначение: ищет и возвращает пару ключ-данные по указанному ключу (абстрактный метод).

Входные параметры:

**\_key**: ключ пары ключ-данные, которую нужно найти.

Выходные параметры:

Указатель на пару ключ-данные, если она найдена, или **nullptr**, если не найдена.

bool IsFull() const noexcept;

Назначение: проверяет, достигла ли таблица максимального размера.

Входные параметры: нет.

Выходные параметры:

**true**, если таблица полная, и **false**, если нет.

bool IsEmpty() const noexcept;

Назначение: проверяет, пуста ли таблица.

Входные параметры: нет.

Выходные параметры:

**true**, если таблица пустая, и **false**, если нет.

bool IsTabEnded() const noexcept;

Назначение: проверяет, достигнута ли последняя запись в таблице.

Входные параметры: нет

Выходные параметры:

**true**, если таблица пуста или достигнут конец, и **false**, если нет.

virtual bool Reset() noexcept;

Назначение: сбрасывает текущую позицию в таблице на первую запись (абстрактный метод).

Входные параметры: нет

Выходные параметры:

**true**, если сброс выполнен успешно, и **false**, если таблица пуста.

virtual bool Next() noexcept;

Назначение: перемещает текущую позицию в таблице на следующую запись (абстрактный метод).

Входные параметры: нет

Выходные параметры:

**true**, если перемещение выполнено успешно, и **false**, если текущая позиция уже находится в конце таблицы.

int GetCurrPos() const noexcept;

Назначение: возвращает текущую позицию в таблице.

Входные параметры: нет

Выходные параметры:

текущая позиция в таблице, от 0 до **GetCount()** - 1.

**int GetMaxSize() const noexcept;**

Назначение: возвращает максимальный размер таблицы.

Входные параметры: нет

Выходные параметры:

максимальный размер таблицы.

**int GetCount() const noexcept;**

Назначение: возвращает количество записей в таблице.

Входные параметры: нет

Выходные параметры:

количество записей в таблице.

### Описание класса TScanTable

template<typename TKey, typename TData>

class TScanTable : public Table<TKey, TData>

{

protected:

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

public:

TScanTable(int \_maxSize = 100);

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

virtual ~TScanTable();

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

virtual void Remove(const TKey& \_key);

virtual TTabRecord<TKey, TData>\* Find(const TKey& \_key);

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

};

Назначение: представление таблицы.

Поля:

**recs** - массив указателей на записи типа **TTabRecord<TKey, TData>,** представляющих элементы таблицы.

Методы:

TScanTable(int \_maxSize = 100)

Назначение: конструктор с параметром. Создает таблицу с максимальным размером **\_maxSize**.

TScanTable(const TScanTable<TKey, TData>& ScanTable)

Назначение: конструктор копирования. Создает копию существующей таблицы.

~TScanTable()

Назначение: виртуальный деструктор. Освобождает ресурсы, связанные с таблицей.

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

Назначение: метод для вставки новой записи в таблицу.

Входные параметры: \_key - ключ типа **TKey, \_data** - указатель на данные типа **TData**.

Выходные параметры: отсутствуют.

void Remove(const TKey& \_key)

Назначение: метод для удаления записи из таблицы по ключу.

Входные параметры: **\_key** - ключ типа **TKey**, по которому нужно удалить запись.

Выходные параметры: отсутствуют.

TTabRecord<TKey, TData>\* Find(const TKey& \_key)

Назначение: метод для поиска записи в таблице по ключу.

Входные параметры: **\_key** - ключ типа **TKey**, который нужно найти.

Выходные параметры: указатель на запись типа **TTabRecord<TKey, TData>,** соответствующую найденному ключу.

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

Назначение: перегруженный оператор вывода в поток. Позволяет выводить содержимое таблицы в поток вывода.

### Описание TSortedTable

template<typename TKey, typename TData>

class TSortedTable : public TScanTable<TKey, TData>

{

private:

void sort();

void quicksort(int left, int right);

int partition(int left, int right);

void swap(TTabRecord<TKey, TData>\*& a, TTabRecord<TKey, TData>\*& b);

public:

TSortedTable(int \_maxSize = 100);

TSortedTable(const TScanTable<TKey, TData>& ScanTable);

TSortedTable(const TSortedTable<TKey, TData>& SortTable);

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

void Remove(const TKey& \_key);

TTabRecord<TKey, TData>\* Find(const TKey& \_key);

};

Назначение: класс **TSortedTable** предназначен для хранения данных в виде таблицы с отсортированными элементами.

Методы:

TSortedTable(int \_maxSize = 100)

Назначение: инициализация объекта **TSortedTable**.

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

Выходные параметры: отсутствуют.

TSortedTable(const TScanTable<TKey, TData>& ScanTable)

Назначение: создание копии таблицы.

Входные параметры: **ScanTable** - ссылка на существующую таблицу.

Выходные параметры: отсутствуют.

TSortedTable(const TSortedTable<TKey, TData>& SortTable)

Назначение: создание копии объекта типа **TSortedTable**.

Входные параметры: **SortTable** - ссылка на существующий объект типа **TSortedTable**.

Выходные параметры: отсутствуют.

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

Назначение: вставка новой записи в таблицу с учетом сортировки.

Входные параметры: **\_key** - ключ типа **TKey, \_data** - указатель на данные типа **TData.**

Выходные параметры: отсутствуют.

void Remove(const TKey& \_key)

Назначение: удаление записи из таблицы с учетом сортировки.

Входные параметры: **\_key** - ключ типа **TKey**, по которому нужно удалить запись.

Выходные параметры: отсутствуют.

TTabRecord<TKey, TData>\* Find(const TKey& \_key)

Назначение: поиск записи в таблице с учетом сортировки.

Входные параметры: **\_key** - ключ типа **TKey**, который нужно найти.

Выходные параметры: указатель на запись типа **TTabRecord<TKey, TData>**, соответствующую найденному ключу.

**void sort()**

Назначение: метод для сортировки элементов таблицы.

**void quicksort(int left, int right)**

Назначение : метод для быстрой сортировки элементов.

**int partition(int left, int right)**

Назначение: метод для разделения элементов при быстрой сортировке.

**void swap(TTabRecord<TKey, TData>\*& a, TTabRecord<TKey, TData>\*& b)**

Назначение: метод для обмена значений двух указателей на записи.

### Описание THashTasble

template<typename TKey, typename TData>

class THashTable : public Table<TKey, TData>

{

protected:

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

public:

THashTable(size\_t size = 100) : Table<TKey, TData>(size) {};

};

Назначение: абстрактное представление хэш-таблицы.

Методы

THashTable(size\_t size = 100)

Назначение инициализация объекта **THashTable.**

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

Выходные параметры: отсутствуют.

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

Назначение: определение хэш-функции для ключа.

Входные параметры: **key** - ключ типа **TKey**.

Выходные параметры: значение типа **size\_t**, полученное в результате применения хэш-функции к ключу.

### Описание TArrayHashTable

template<typename TKey, typename TData>

class TArrayHashTable : public THashTable<TKey, TData>

{

protected:

size\_t hash\_step;

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

TTabRecord<TKey, TData>\* pMark;

int \_freepos\_ind;

size\_t hash\_func(const TKey& \_key);

void get\_next\_pos(size\_t ind);

void coprime(size\_t \_maxSize, size\_t \_hash\_step);

public:

TArrayHashTable(size\_t \_maxSize = 100, size\_t \_hashStep = 11);

TArrayHashTable(const TArrayHashTable<TKey, TData>& aht);

virtual ~TArrayHashTable();

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

void Remove(const TKey& \_key);

TTabRecord<TKey, TData>\* Find(const TKey& \_key);

friend std::ostream& operator<<(std::ostream& out, TArrayHashTable<TKey, TData>& aht)$

};

Назначение: класс **TArrayHashTable** предназначен для хранения данных в виде хэш-таблицы, используя массив для хранения элементов. Он предоставляет методы для добавления, удаления и поиска элементов в хэш-таблице.

Поля

**hash\_step** - размер шага для поиска свободной ячейки при коллизиях.

**recs** - массив указателей на записи (элементы таблицы).

**pMark** - маркер для пометки удаленных записей.

**\_freepos\_ind** - индекс первой свободной позиции в массиве.

Методы

TArrayHashTable(size\_t \_maxSize = 100, size\_t \_hashStep = 11)

Назначение: инициализация объекта **TArrayHashTable**.

Входные параметры: **\_maxSize** - максимальный размер хэш-таблицы, **\_hashStep** - размер шага для поиска свободной ячейки.

Выходные параметры: отсутствуют.

TArrayHashTable(const TArrayHashTable<TKey, TData>& aht)

Назначение: создание копии существующего объекта **TArrayHashTable**.

Входные параметры: **aht** - ссылка на объект **TArrayHashTable**, который нужно скопировать.

Выходные параметры: отсутствуют.

virtual ~TArrayHashTable()

Назначение: уничтожение объекта **TArrayHashTable**.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

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

Назначение: добавление нового элемента с ключом **\_key** и данными \_data в хэш-таблицу.

Входные параметры: **\_key** - ключ типа **TKey, \_data** - указатель на данные типа **TData**.

Выходные параметры: отсутствуют.

void Remove(const TKey& \_key)

Назначение: удаление элемента с ключом **\_key** из хэш-таблицы.

Входные параметры: **\_key** - ключ типа TKey.

Выходные параметры: отсутствуют.

TTabRecord<TKey, TData>\* Find(const TKey& \_key)

Назначение: поиск элемента по ключу \_**key** в хэш-таблице.

Входные параметры: **\_key** - ключ типа **TKey**.

Выходные параметры: указатель на запись типа **TTabRecord<TKey, TData>,** содержащую данные элемента, если элемент найден; в противном случае, возвращается нулевой указатель.

friend std::ostream& operator<<(std::ostream& out, TArrayHashTable<TKey, TData>& aht)

Назначение: перегрузка оператора вывода в поток для объектов класса **TArrayHashTable**.

Входные параметры: out - ссылка на поток вывода, aht - ссылка на объект **TArrayHashTable.**

Выходные параметры: ссылка на поток вывода.

size\_t hash\_func(const TKey& \_key)

Назначение: определение хэш-функции для ключа **\_key**.

Входные параметры: **\_key** - ключ типа **TKey**.

Выходные параметры: значение типа **size\_t**, полученное в результате применения хэш-функции к ключу.

void get\_next\_pos(size\_t ind)

Назначение: получение следующей позиции для размещения элемента при коллизиях.

Входные параметры: **ind** - индекс текущей позиции.

Выходные параметры: отсутствуют.

void coprime(size\_t \_maxSize, size\_t \_hash\_step)

Назначение: определение взаимно простого числа для использования в хэшировании.

Входные параметры: **\_maxSize** - максимальный ра

# Заключение

В данной лабораторной работе мы сосредоточились на реализации и использовании различных структур таблиц: просматриваемой таблицы, отсортированной таблицы и хэш-таблицы.

Изучение преимуществ и недостатков каждой структуры данных в контексте работы с полиномами позволило нам лучше понять, какая структура более эффективна для определенных операций. Например, просматриваемая таблица может быть полезна для операций, требующих последовательного доступа к элементам, в то время как хэш-таблица обеспечивает быстрый доступ к данным.

Можем подчеркнуть важность выбора подходящей структуры данных для оптимальной работы с полиномами, а также подчеркнуть значимость изучения и сравнения различных методов хранения информации для оптимизации процессов обработки данных.

# Литература

1. Лекция «Организация доступа по имени. Таблицы» Сысоев А.В [https://cloud.unn.ru/s/2Y92XyGc7r3XdBC].
2. Лекция «Хеш-таблицы» Сысоев А.В [https://cloud.unn.ru/s/B5fr3gKAL2LoHyH].

# Приложения

## Приложение А. Реализация класса TTabRecord

#ifndef \_TABRECORD\_H\_

#define \_TABRECORD\_H\_

#include "iostream"

using namespace std;

template<typename TKey, typename TData>

class TTabRecord

{

public:

TKey key;

TData\* data;

public:

TTabRecord();

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

TTabRecord(const TTabRecord<TKey, TData>& record);

~TTabRecord();

TKey GetKey() const;

TData\* GetData() const;

/\*void SetData(TData\* newData);\*/

const TTabRecord<TKey, TData>& operator=(const TTabRecord<TKey, TData>& TabRec);

};

template<typename TKey, typename TData>

TTabRecord<TKey, TData>::TTabRecord() : key(), data(nullptr) {};

template<typename TKey, typename TData>

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

{

key = \_key;

data = new TData(\*\_data);

}

template<typename TKey, typename TData>

TTabRecord<TKey, TData>::TTabRecord(const TTabRecord<TKey, TData>& record)

{

key = record.key;

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

}

template<typename TKey, typename TData>

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

{

if (data) delete data;

}

template<typename TKey, typename TData>

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

{

return key;

}

template<typename TKey, typename TData>

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

{

return data;

}

template<typename TKey, typename TData>

const TTabRecord<TKey, TData>& TTabRecord<TKey, TData>::operator=(const TTabRecord<TKey, TData>& TabRec)

{

key = TabRec.key;

if (data)

delete data;

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

return(\*this);

}

#endif // ! \_TABRECORD\_H\_

## Приложение Б. Реализация класса Table

#ifndef \_TABLE\_H\_

#define \_TABLE\_H\_

#include "TabRecord.h"

using namespace std;

template<typename TKey, typename TData>

class Table

{

protected:

int count;

int maxSize;

int currPos;

public:

Table(int max\_size = 101);

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

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

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

bool IsFull() const noexcept;

bool IsEmpty() const noexcept;

bool IsTabEnded() const noexcept;

virtual bool Reset() noexcept;

virtual bool Next() noexcept;

int GetCurrPos() const noexcept;

int GetMaxSize() const noexcept;

int GetCount() const noexcept;

};

template<typename TKey, typename TData>

Table<TKey, TData>::Table(int max\_size)

{

maxSize = max\_size;

count = 0;

currPos = -1;

}

template<typename TKey, typename TData>

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

{

return (count == maxSize);

}

template<typename TKey, typename TData>

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

{

return (count == 0);

}

template<typename TKey, typename TData>

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

{

return (currPos >= maxSize);//???

}

template<typename TKey, typename TData>

bool Table<TKey, TData>::Reset() noexcept//???

{

if (!IsEmpty())

currPos = 0;

else

currPos = -1;

return IsTabEnded();

}

template<typename TKey, typename TData>

bool Table<TKey, TData>::Next() noexcept//???

{

if (!IsTabEnded())

currPos++;

return IsTabEnded();

}

template<typename TKey, typename TData>

inline int Table<TKey, TData>::GetCurrPos() const noexcept

{

return currPos;

}

template<typename TKey, typename TData>

int Table<TKey, TData>::GetMaxSize() const noexcept

{

return maxSize;

}

template<typename TKey, typename TData>

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

{

return count;

}

#endif // \_TABLE\_H\_

## }Приложение В. Реализация класса TScanTable

#ifndef \_SCANTABLE\_H\_

#define \_SCANTABLE\_H\_

#include "Table.h"

#include <iostream>

template<typename TKey, typename TData>

class TScanTable : public Table<TKey, TData>

{

protected:

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

public:

TScanTable(int \_maxSize = 100);

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

virtual ~TScanTable();

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

virtual void Remove(const TKey& \_key);

virtual TTabRecord<TKey, TData>\* Find(const TKey& \_key);

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

{

if (st.IsEmpty())

{

out << "table is empty" << endl;

return out;

}

out << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

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

{

out /\*<< "|" << st.recs[i]->GetKey() << " | "\*/ << \*(st.recs[i]->GetData()) << "|" << endl;

}

out << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

return out;

}

};

template<typename TKey, typename TData>

TScanTable<TKey, TData>::TScanTable(int \_maxSize)

{

if (\_maxSize <= 0)

{

throw"Error";

}

this->maxSize = \_maxSize;

recs = new TTabRecord<TKey, TData>\*[this->maxSize];

for (int i = 0; i < this->maxSize; i++)

{

recs[i] = nullptr;

}

this->count = 0;

this->currPos = -1;

}

template<typename TKey, typename TData>

TScanTable<TKey, TData>::TScanTable(const TScanTable<TKey, TData>& ScanTable)

{

this->maxSize = ScanTable.maxSize;

this->count = ScanTable.count;

this->currPos = ScanTable.currPos;

recs = new TTabRecord<TKey, TData>\*[this->maxSize];

for (int i = 0; i < this->count; i++)

{

TKey \_key = ScanTable.recs[i]->key;

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

recs[i] = new TTabRecord<TKey, TData>(\_key, \_data);

}

}

template<typename TKey, typename TData>

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

{

if (recs != nullptr)

{

for (int i = 0; i < this->count; i++)

{

if (recs[i] != nullptr)

delete recs[i];

}

delete recs;

}

}

template<typename TKey, typename TData>

void TScanTable<TKey, TData>::Insert(const TKey& \_key, TData\* \_data)

{

if (this->IsFull())

{

throw"Error";

}

TTabRecord<TKey, TData>\* flag = Find(\_key);

if (flag!=nullptr)

{

throw"Error: record is exist";

}

else

{

recs[this->count] = new TTabRecord<TKey, TData>(\_key, \_data);

this->count++;

}

}

template<typename TKey, typename TData>

void TScanTable<TKey, TData>::Remove(const TKey& \_key)

{

if (this->IsEmpty())

{

throw"Error";

}

if (Find(\_key) != nullptr)

{

delete recs[this->currPos];

recs[this->currPos] = recs[(this->count) - 1];

this->count--;

}

else

{

throw"Error";

}

}

template<typename TKey, typename TData>

TTabRecord<TKey, TData>\* TScanTable<TKey, TData>::Find(const TKey& \_key)

{

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

for (int i = 0; i < this->count; i++)

{

if (recs[i]->GetKey() == \_key)

{

this->currPos = i;

res = recs[i];

break;

}

}

return res;

}

#endif // !\_SCANTABLE\_H\_

## Приложение Г. Реализация класса TSortedTable

#ifndef \_SORTEDTABLE\_H\_

#define \_SORTEDTABLE\_H\_

#include "ScanTable.h"

#include <iostream>

using namespace std;

template<typename TKey, typename TData>

class TSortedTable : public TScanTable<TKey, TData>

{

private:

void sort();

void quicksort(int left, int right);

int partition(int left, int right);

void swap(TTabRecord<TKey, TData>\*& a, TTabRecord<TKey, TData>\*& b);

public:

TSortedTable(int \_maxSize = 100);

TSortedTable(const TScanTable<TKey, TData>& ScanTable);

TSortedTable(const TSortedTable<TKey, TData>& SortTable);

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

void Remove(const TKey& \_key);

TTabRecord<TKey, TData>\* Find(const TKey& \_key);

};

template<typename TKey, typename TData>

void TSortedTable<TKey, TData>::sort()

{

quicksort(0, (this->count) - 1);

}

template<typename TKey, typename TData>

void TSortedTable<TKey, TData>::quicksort(int left, int right)

{

if (left < right)

{

int pivot = partition(left, right);

quicksort(left, pivot - 1);

quicksort(pivot + 1, right);

}

}

template<typename TKey, typename TData>

int TSortedTable<TKey, TData>::partition(int left, int right)

{

TKey pivot = this->recs[right]->GetKey();

int i = left - 1;

for (int j = left; j < right; j++)

{

if (this->recs[j]->GetKey() < pivot)

{

i++;

swap(this->recs[i], this->recs[j]);

}

}

swap(this->recs[i + 1], this->recs[right]);

return i + 1;

}

template<typename TKey, typename TData>

void TSortedTable<TKey, TData>::swap(TTabRecord<TKey, TData>\*& a, TTabRecord<TKey, TData>\*& b)

{

TTabRecord<TKey, TData>\* temp = a;

a = b;

b = temp;

}

template<typename TKey, typename TData>

TSortedTable<TKey, TData>::TSortedTable(int \_maxSize) : TScanTable<TKey, TData>(\_maxSize) {}

template<typename TKey, typename TData>

TSortedTable<TKey, TData>::TSortedTable(const TScanTable<TKey, TData>& ScanTable) : TScanTable<TKey, TData>(ScanTable)

{

sort();

}

template<typename TKey, typename TData>

TSortedTable<TKey, TData>::TSortedTable(const TSortedTable<TKey, TData>& SortTable)

{

this->maxSize = SortTable.maxSize;

this->count = SortTable.count;

this->currPos = SortTable.currPos;

this->recs = new TTabRecord<TKey, TData>\*[this->maxSize];

for (int i = 0; i < this->count; i++)

{

TKey \_key = TScanTable.recs[i]->key;

TData \_data = TScanTable.recs[i]->data;

this->recs[i] = new TTabRecord<TKey, TData>(\_key, \_data);

}

}

template<typename TKey, typename TData>

void TSortedTable<TKey, TData>::Insert(const TKey& \_key, TData\* \_data)

{

if (this->IsFull())

{

throw "Error";

}

TTabRecord<TKey, TData>\* flag = Find(\_key);

if (flag!=nullptr)

{

throw"Error";

}

else

{

TTabRecord<TKey, TData>\* newItem = new TTabRecord<TKey, TData>(\_key, \_data);

// Находим позицию для вставки нового элемента

int index = 0;

while (index < this->count && this->recs[index]->GetKey() < \_key)

{

index++;

}

// Сдвигаем элементы для освобождения места под новый элемент

for (int i = this->count; i > index; i--)

{

this->recs[i] = this->recs[i - 1];

}

// Вставляем новый элемент на найденную позицию

this->recs[index] = newItem;

this->count++; // Увеличиваем счетчик элементов

}

}

template<typename TKey, typename TData>

void TSortedTable<TKey, TData>::Remove(const TKey& \_key)

{

if (this->IsEmpty())

{

throw"Error";

}

TTabRecord<TKey, TData>\* rec = Find(\_key);

if (rec == nullptr)

{

throw"Error";

}

else

{

delete rec;

for (int i = this->currPos; i < this->count - 1; i++)

{

this->recs[i] = this->recs[i + 1];

}

this->count--;

}

}

template<typename TKey, typename TData>

TTabRecord<TKey, TData>\* TSortedTable<TKey, TData>::Find(const TKey& \_key)

{

int left = 0;

int right = (this->count) - 1; // индекс последнего элемента в таблице

while (left <= right)

{

int mid = left + (right - left) / 2; // находим середину текущего диапазона

if (this->recs[mid]->GetKey() == \_key)

{

right = mid;

left = mid + 1;

this->currPos = right;

return this->recs[mid]; // элемент найден

}

else if (this->recs[mid]->GetKey() < \_key)

{

left = mid + 1; // искать справа от середины

}

else

{

right = mid - 1; // искать слева от середины

}

}

return nullptr; // элемент с указанным ключом не найден

}

#endif // !\_SORTEDTABLE\_H\_

## Приложение Е Реализация классе THashTable

#ifndef \_HASHTABLE\_H\_

#define \_HASHTABLE\_H\_

#include"Table.h"

#include<iostream>

template<typename TKey, typename TData>

class THashTable : public Table<TKey, TData>

{

protected:

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

public:

THashTable(size\_t size = 100) : Table<TKey, TData>(size) {};

};

#endif // !\_HASHTABLE\_H\_

## Приложение Ж Реализация классе TArrayHashTable

#ifndef \_ARRAYHASHTABLE\_H\_

#define \_ARRAYHASHTABLE\_H\_

#include"Hashtable.h"

#include <iostream>

#include <functional>

using namespace std;

template<typename TKey, typename TData>

class TArrayHashTable : public THashTable<TKey, TData>

{

protected:

size\_t hash\_step;

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

TTabRecord<TKey, TData>\* pMark;

int \_freepos\_ind;

size\_t hash\_func(const TKey& \_key);

void get\_next\_pos(size\_t ind);

void coprime(size\_t \_maxSize, size\_t \_hash\_step);

public:

TArrayHashTable(size\_t \_maxSize = 100, size\_t \_hashStep = 11);

TArrayHashTable(const TArrayHashTable<TKey, TData>& aht);

virtual ~TArrayHashTable();

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

void Remove(const TKey& \_key);

TTabRecord<TKey, TData>\* Find(const TKey& \_key);

/\*bool Reset() noexcept;

bool Next() noexcept;\*/

friend std::ostream& operator<<(std::ostream& out, TArrayHashTable<TKey, TData>& aht)

{

if (aht.IsEmpty())

{

out << "table is empty" << endl;

return out;

}

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

for (int i = 0; i < aht.GetMaxSize(); i++)

{

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

{

out /\*<< "|" << aht.recs[i]->GetKey() << "|"\*/ << \*(aht.recs[i]->GetData()) << endl;

}

}

return out;

}

};

template<typename TKey, typename TData>

size\_t TArrayHashTable<TKey, TData>::hash\_func(const TKey& \_key)

{

std::hash<TKey> hasher;

return hasher(\_key) % (this->maxSize);

}

template<typename TKey, typename TData>

void TArrayHashTable<TKey, TData>::get\_next\_pos(size\_t ind)

{

if (this->IsFull()) \_freepos\_ind = 0;

int new\_ind = (ind + hash\_step % this->maxSize);

while (new\_ind != ind && (recs[new\_ind] != pMark && recs[new\_ind] != nullptr))

{

new\_ind = (new\_ind + hash\_step) % this->maxSize;

}

\_freepos\_ind = new\_ind;

}

template<typename TKey, typename TData>

void TArrayHashTable<TKey, TData>::coprime(size\_t \_maxSize, size\_t \_hash\_step)

{

if (\_maxSize <= 0)

{

throw "Error: \_maxSize should be greater than 0";

}

if (\_hash\_step <= 0)

{

throw "Error: \_hash\_step should be greater than 0";

}

size\_t a = \_maxSize;

size\_t b = \_hash\_step;

while (b != 0)

{

size\_t tmp = b;

b = a % b;

a = tmp;

}

if (a != 1)

{

throw "Error: \_maxSize and \_hash\_step are not coprime";

}

}

template<typename TKey, typename TData>

TArrayHashTable<TKey, TData>::TArrayHashTable(size\_t \_maxSize, size\_t \_hashStep) : THashTable<TKey, TData>(\_maxSize)

{

coprime(\_maxSize, \_hashStep);

recs = new TTabRecord<TKey, TData>\*[this->maxSize];

hash\_step = \_hashStep;

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

\_freepos\_ind = 0;

for (int i = 0; i < this->maxSize; i++)

{

recs[i] = nullptr;

}

}

template<typename TKey, typename TData>

TArrayHashTable<TKey, TData>::TArrayHashTable(const TArrayHashTable<TKey, TData>& aht)

{

this->maxSize = aht.maxSize;

this->count = aht.count;

hash\_step = aht.hash\_step;

this->currPos = aht.currPos;

\_freepos\_ind = aht.free\_pos\_ind;

recs = new TTabRecord<TKey, TData>\*[this->maxSize];

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

for (int i = 0; i < this->maxSize; i++)

{

TTabRecord<TKey, TData>\* tmp = aht.recs[i];

if (tmp == nullptr)

recs[i] = tmp;

else if (tmp == aht.pMark)

recs[i] = tmp;

else

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

}

}

template<typename TKey, typename TData>

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

{

for (int i = 0; i < this->maxSize; i++)

{

if (recs[i] != nullptr)

{

delete recs[i];

}

if (recs[i] == pMark)

{

recs[i] = nullptr;

}

}

if (recs) delete[] recs;

if (pMark) delete pMark;

}

template<typename TKey, typename TData>

void TArrayHashTable<TKey, TData>::Insert(const TKey& \_key, TData\* \_data)

{

if (this->IsFull())

{

throw"Error";

}

TTabRecord<TKey, TData>\* foundRecord = Find(\_key);

if (foundRecord == nullptr)

{

this->recs[\_freepos\_ind] = new TTabRecord<TKey, TData>(\_key, \_data);

this->count++;

}

else

{

throw"Error";

}

}

template<typename TKey, typename TData>

void TArrayHashTable<TKey, TData>::Remove(const TKey& \_key)

{

if (this->IsEmpty())

{

throw"Error";

}

TTabRecord<TKey, TData>\* tmp = Find(\_key);

if (tmp == nullptr)

{

throw"Error";

}

delete tmp;

recs[this->currPos] = pMark;

this->count--;

}

template<typename TKey, typename TData>

TTabRecord<TKey, TData>\* TArrayHashTable<TKey, TData>::Find(const TKey& \_key)

{

int hs = hash\_func(\_key), t = (hs + hash\_step) % this->maxSize, c = 1;

\_freepos\_ind = hs;

if (recs[hs] == nullptr)

{

\_freepos\_ind = hs;

return nullptr;

}

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

{

this->currPos = hs;

return recs[hs];

}

while (recs[t] != nullptr && t != hs && c < this->maxSize)

{

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

{

this->currPos = t;

return recs[t];

}

if (recs[t] == nullptr)

{

\_freepos\_ind = t;

return nullptr;

}

t = (t + hash\_step) % this->maxSize;

c++;

}

if (recs[\_freepos\_ind] != pMark && recs[\_freepos\_ind] != nullptr)

{

get\_next\_pos(\_freepos\_ind);

}

return nullptr;

}

#endif // !\_ARRAYHASHTABLE\_H\_