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:

**«Таблицы»**

**Выполнил:** студентка группы 3822Б1ФИ2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Резанцева А. А./

Подпись

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

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

Подпись

Нижний Новгород  
2024

**Содержание**

[Введение 3](#_Toc167713063)

[1 Постановка задачи 4](#_Toc167713064)

[2 Руководство пользователя 5](#_Toc167713065)

[2.1 Приложение для демонстрации работы таблиц 5](#_Toc167713066)

[3 Руководство программиста 13](#_Toc167713067)

[3.1 Описание алгоритмов 13](#_Toc167713068)

[3.1.1 Неупорядоченная таблица 13](#_Toc167713069)

[3.1.2 Упорядоченная таблица 15](#_Toc167713070)

[3.1.3 Хэш таблица 17](#_Toc167713071)

[3.1.4 Описание класса TTabRecord 19](#_Toc167713072)

[3.1.5 Описание класса TTable 20](#_Toc167713073)

[3.1.6 Описание класса TScanTable 23](#_Toc167713074)

[3.1.7 Описание класса SortedTable 24](#_Toc167713075)

[3.1.8 Описание класса THashTable 25](#_Toc167713076)

[3.1.9 Описание класса TArrayHashTable 26](#_Toc167713077)

[Заключение 30](#_Toc167713078)

[Литература 31](#_Toc167713079)

[Приложения 32](#_Toc167713080)

[Приложение А. Реализация класса TTabRecord 32](#_Toc167713081)

[Приложение Б. Реализация класса TTable 32](#_Toc167713082)

[Приложение В. Реализация класса TScanTable 34](#_Toc167713083)

[Приложение Г. Реализация класса SortedTable 36](#_Toc167713084)

[Приложение Д. Реализация класса THashTable 38](#_Toc167713085)

[Приложение Е. Реализация класса TArrayHashTable 38](#_Toc167713086)

[Приложение Ж. Sample\_table 41](#_Toc167713087)

# Введение

Лабораторная работа направлена на изучение и реализацию трех типов таблиц: упорядоченной, неупорядоченной и хэш таблицы.

В таблицах будут содержаться полиномы, реализованные на базе предыдущей лабораторной работы. Таблицы должны обеспечить возможность работы над полиномами, например, сложение, умножение, вычитание, взятие производной внутри самих таблиц.

# Постановка задачи

**Цель:**

Цель работы – научиться работать с записями таблицы. Роль записей в таблицах будут играть полиномы на линейных односвязных списках, которые были реализованы в предыдущей лабораторной работе. Ключ к записи – это строковое представление полинома.

**Задачи:**

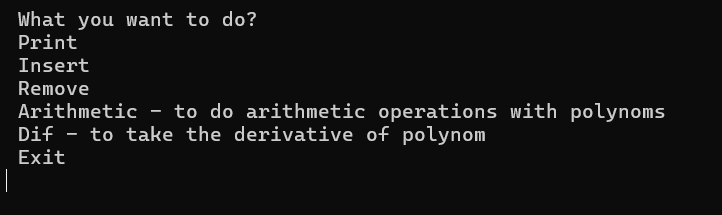
1. Изучение основных принципов работы с таблицами.
2. Создание упорядоченной, неупорядоченной и хэш таблиц.
3. Предоставить возможность хранить, добавлять, удалять в таблицах и еще сортировать в упорядоченной.
4. Написание программы на С++.
5. Тестирование программы на различных входных данных.

Результатом выполнения лабораторной работы станет полнофункциональная реализация алгоритмов работы с таблицами, которые могут быть использованы для различных задач.

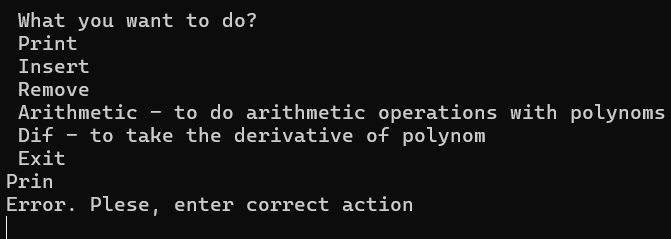
# Руководство пользователя

## Приложение для демонстрации работы таблиц

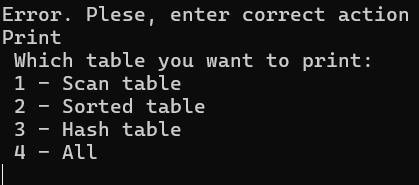
1. Запустите приложение с названием sample\_table.exe. В результате появится окно, показанное ниже (рис. 1). Пользователю будет предложен список возможных действий, и программа будет ожидать ввод перечисленных команд.



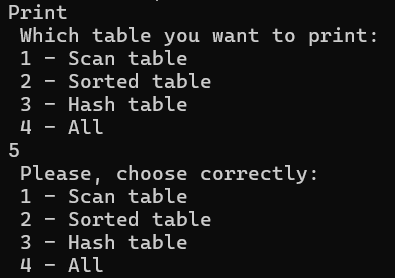
1. Основное окно программы
2. При вводе неверной команды, программа попросит ввести корректно как показано ниже (рис. 2).



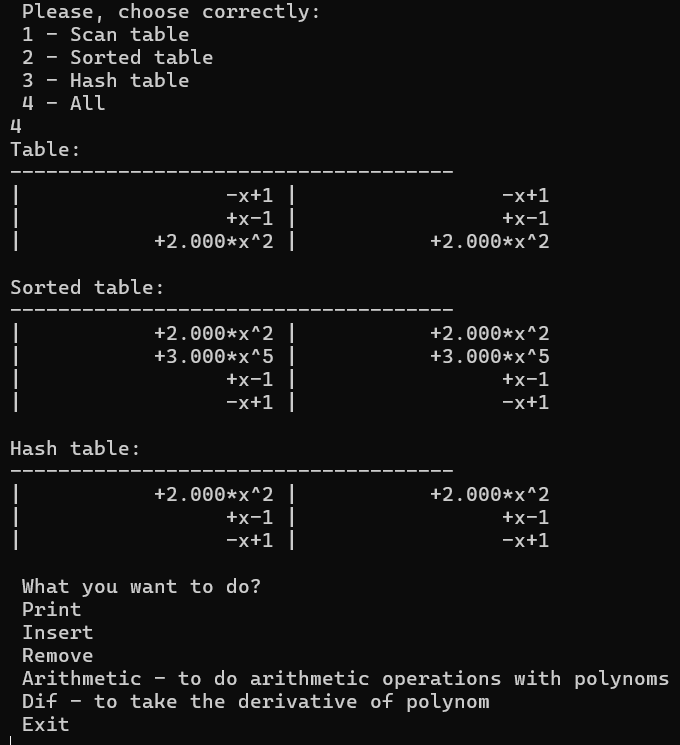
1. Проверка на корретный ввод
2. После правильного ввода Print, пользователю будет предложено вывести таблицы следующим образом (рис. 3):



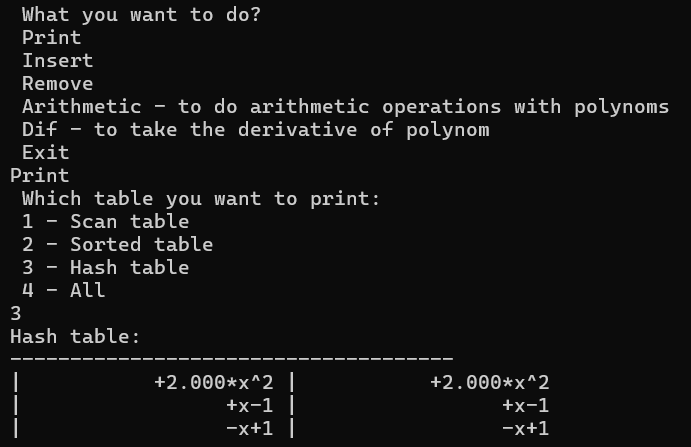
1. Выбор вывода таблицы 1- просматриваемая, 2 – упорядоченная, 3 – хэш таблица, 4 – все
2. Если пользователь ввел другое число, то программа попросит ввести корретное как показано ниже (рис. 4).



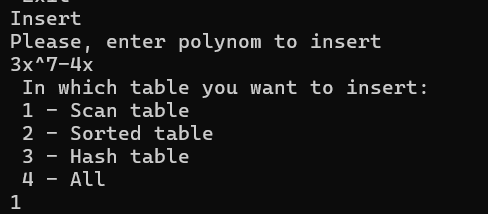
1. Проверка на корректный выбор таблицы
2. После ввода одного из ожидаемых значений будет выведена соответствующая таблица или все как представлено ниже (рис. 5).



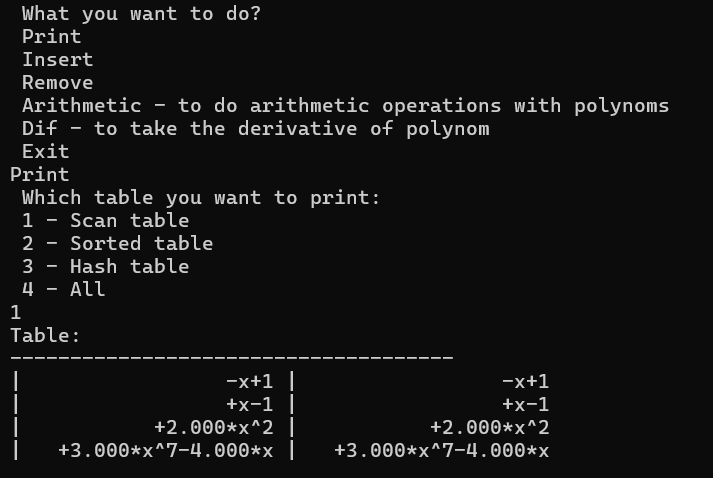
1. Вывод таблиц
2. Вывод одной из таблиц представлен ниже (рис. 6):



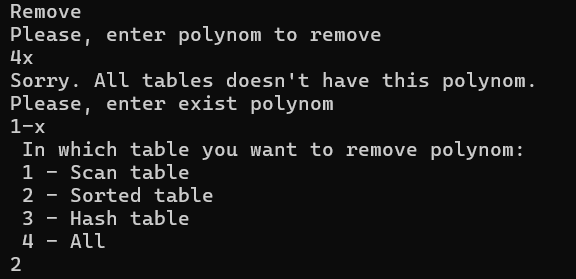
1. Вывод 3 - хэш таблицы
2. После каждой операции пользователю снова будет предложен список возможных действий над таблицами в виде основного окна программы, пока он не введет Exit для завершения программы (рис. 1).
3. Рассмотрим следующую возможность - Insert. Вставка полинома в таблицу. Пользователь введет Insert, далее ему предложат ввести полином и выбрать в какую таблицу вставить. Здесь также есть обработка неверного ввода при выборе таблицы (рис. 7):



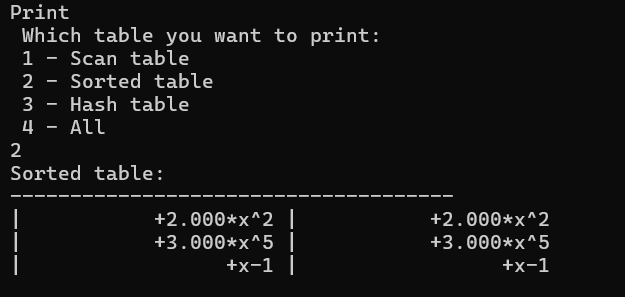
1. Вставка записи в таблицу 1 – в просматриваемую, 2 – в упорядоченную, 3 – в хэш таблицу, 4 – во все.
2. Проверим, добавилась ли пользовательская запись в просматриваемую таблицу (рис. 8).



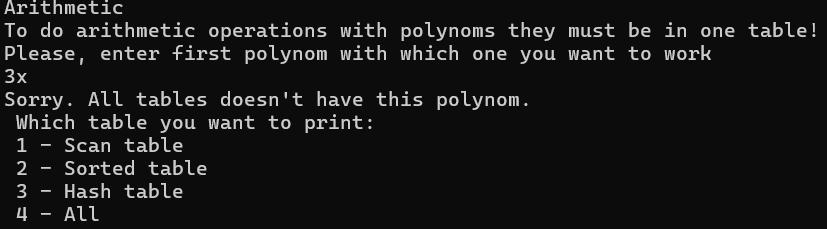
1. Проверка вставки с помощью Print
2. Рассмотрим следующую возможность – Remove. Удаление записи о полиноме в таблицах. После ввода пользователем команды Remove, ему будет предложено ввести полином, который он хочет удалить. В случае, если он не содержится не в одной из таблиц, программа попросит ввести другой, существующий, далее спросит из какой таблицы нужно удалить (рис. 9).



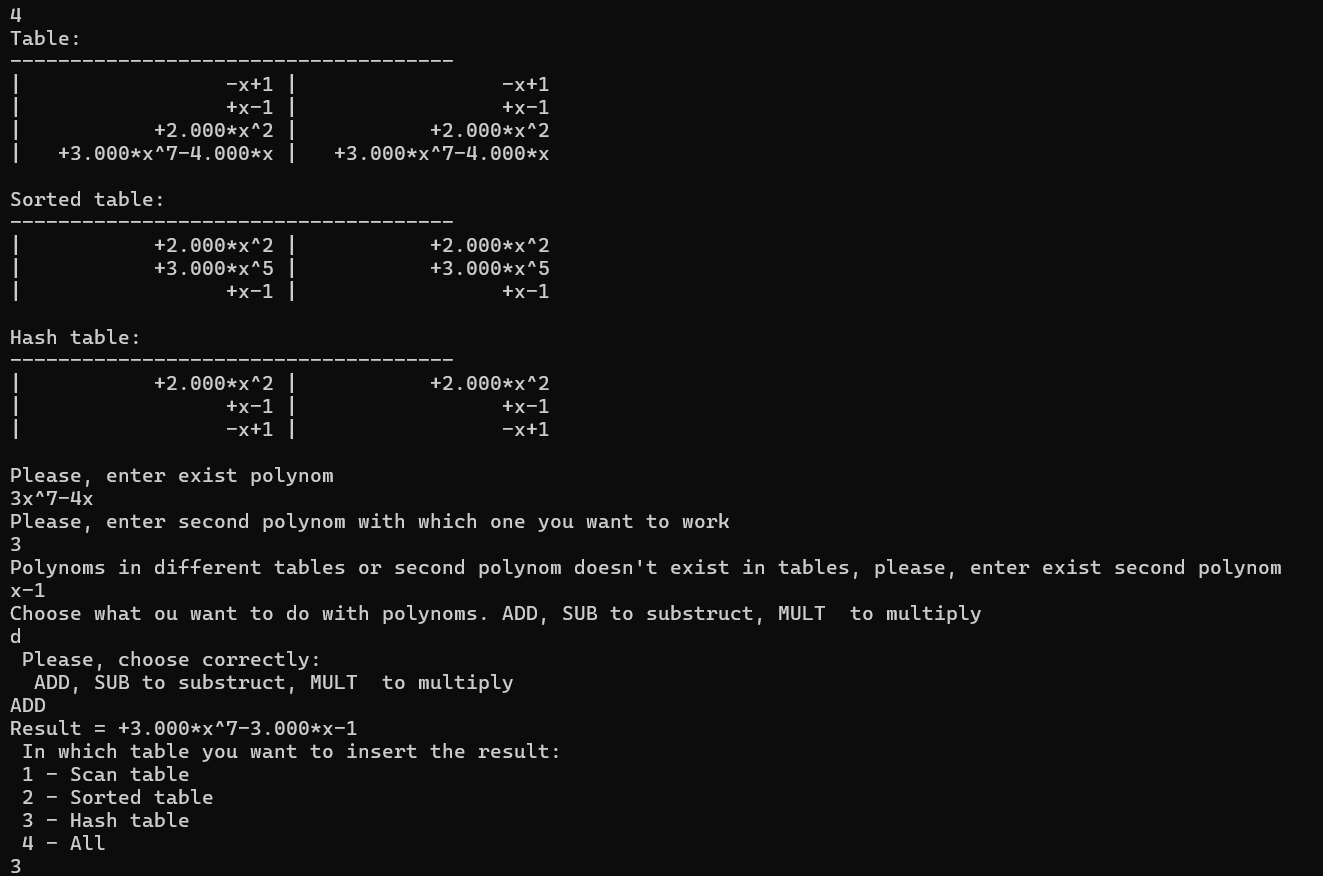
1. Пример удаления записи о полиноме из таблицы 2 – упорядоченной
2. Проверим результат удаления записи из упорядоченной таблицы (рис. 10):



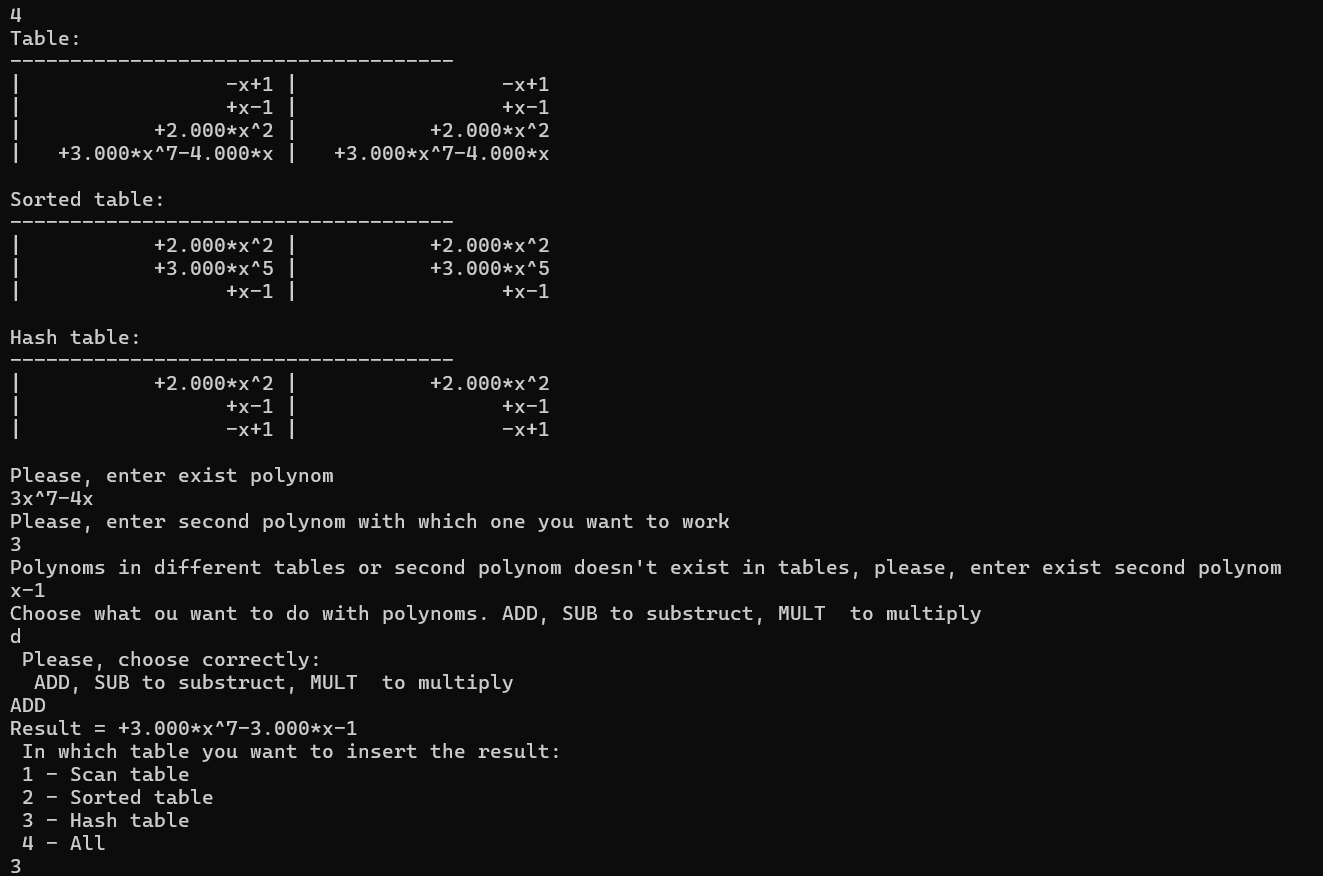
1. Проверка удаления записи из упорядоченной таблицы
2. Рассмотрим следующую возможность – Arithmetic. Пользователю будет выведено предупреждение, что для выполнения арифметических операций над полиномами они должны быть в одной таблице. Далее его попросят ввести первый полином, если его нет ни в одной из таблиц, то пользователю предложат вывести таблицы, чтобы ввести существующий полином (рис. 11).



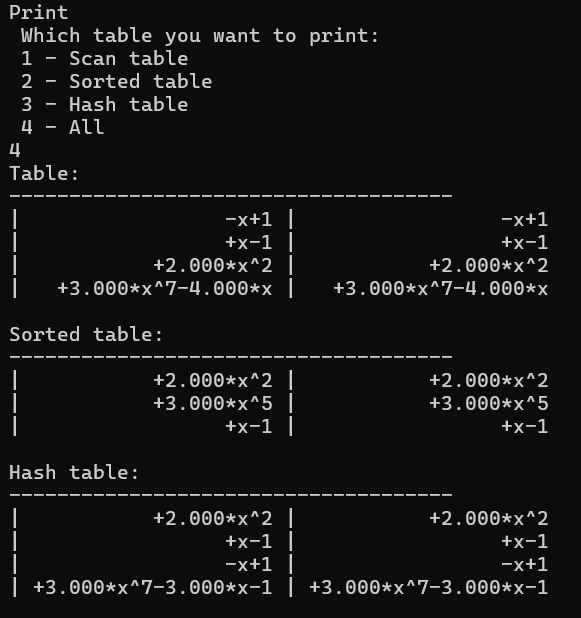
1. Проверка наличия первого полинома в таблицах
2. Далле выведется соответствующая выбору пользователя таблица и пользователь введет первый полином. После этого также будет проверка на второй полином, если пользователь ввел не существующий или если они лежат не в одной таблице, то программа попросит ввести корректный второй полином (рис. 12).



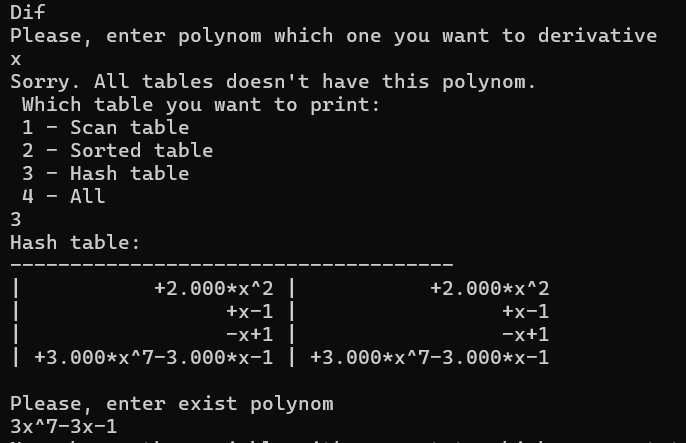
1. Проверка, что оба полинома лежат в одной таблице
2. Далее, при выполнении условия, что оба полинома лежат в одной таблице, будет предложено 3 возможных действия над ними: ADD – сложить, SUB – вычесть, MULT – умножить полиномы друг на друга. Здесь также есть проверка на корректный ввод действия. В данном примере выбрали операцию ADD, пользователю выведен результат, и программа предоставляет возможность сохранить результат в таблицы по выбору пользователя (рис. 13).



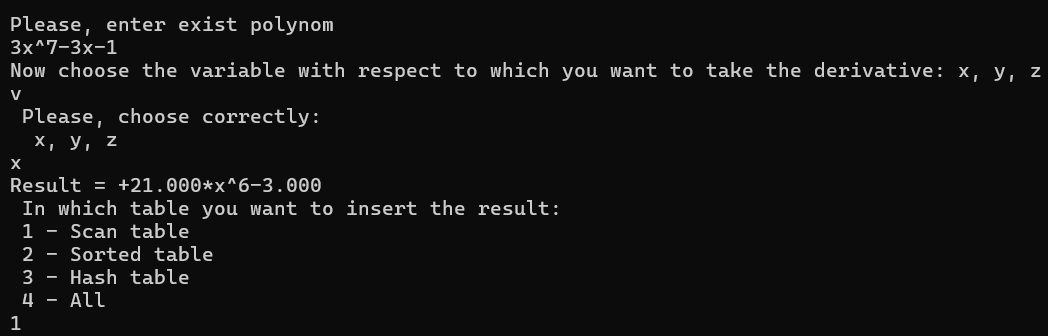
1. Пример использования функции ADD
2. Проверим, что наш результат сохранился в Hash таблицу (рис. 14)



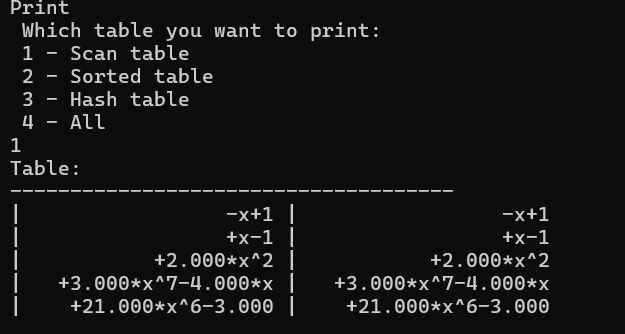
1. Проверка вставки результата в нужную таблицу
2. Рассмотрим следующую возможность – Dif - взятие производной по одной из трех переменных. Пользователю будет предложено ввести полином, от которого он хочет взять производную. Здесь также есть проверка наличия полинома в таблицах. Если никакая из таблиц не содержит введенный полином, пользователю предложат вывести таблицы и ввести существующий (рис. 15).



1. Проверка существования записи полинома для взятия производной
2. Далее пользователя спросят, по какой из трех переменной он хочет взять производную. Здесь также есть проверка на корректный ввод, после которого будет выведен результат, который предложат вставить в таблицы по выбору пользователя (рис. 16)



1. Взятие производной и вставка в просматриваемою таблицу
2. Проверим, что наш результат сохранился в просматриваемой таблице (рис. 17).



1. Проверка добавления результата взятия производной
2. Теперь завершим программу с помощью команды Exit (рис. 18).



1. Завершение программы

# Руководство программиста

## Описание алгоритмов

### Неупорядоченная таблица

Неупорядоченная таблица представлена массивом указателей на записи, где каждая запись представляет собой пару ключ-значение. Неупорядоченная таблица поддерживает операции поиска, операции и удаления, проверки на пустоту и полноту. Структура данных хранит элементы в первых n ячейках массива записей, имеет указатель на последний не занятый элемент (в случае полноты он будет выходить за пределы массива).

**Операция добавления**

Операция добавления элемента реализуется при помощи указателя на первый свободный элемент. В свободную ячейку вставляется переданная запись, количество записей в таблице увеличивается. Если этот элемент уже был в таблице, запись перезаписывается.

Пример:

|  |  |  |
| --- | --- | --- |
| 2 | 4 |  |
| one | two |  |

Операция добавления элемента five с ключом 3:

|  |  |  |
| --- | --- | --- |
| 2 | 4 | 3 |
| one | two | five |

**Операция удаления**

Операция удаления элемента реализуется при помощи перебора всех элементов, пока не будет найден искомый. Удаляем найденный элемент, уменьшая количество элементов в таблице. Если элемент не был последним в таблице, все записи, у которых позиция больше, чем у искомого, будут сдвинуты влево.

Пример:

|  |  |  |
| --- | --- | --- |
| 2 | 4 |  |
| one | two |  |

Операция удаления элемента с ключом 2:

|  |  |  |
| --- | --- | --- |
| 4 |  |  |
| two |  |  |

**Операция поиска**

Операция поиска элемента реализуется при помощи перебора всех элементов, пока не будет найден искомый. Если искомый элемент не был найден, то функция вернёт нулевой указатель, иначе вернет указатель на элемент.

Пример:

|  |  |  |
| --- | --- | --- |
| 2 | 4 |  |
| one | two |  |

Операция поиска элемента с ключом 2:

|  |
| --- |
| 2 |
| one |

**Операция проверки на пустоту**

Операция проверки на пустоту реализована при помощи переменной, отвечающей за текущее количество записей в таблице. Если она равна нулю, то таблица пуста, иначе в ней имеются записи.

Пример:

|  |  |  |
| --- | --- | --- |
| 2 | 4 |  |
| one | two |  |

Операция проверки на пустоту:

Результат: false

**Операция проверки на полноту**

Операция проверки на полноту реализована при помощи переменной, отвечающей за текущее количество записей в таблице. Если она равна максимальному количеству записей в таблице, то таблица полна.

Пример:

|  |  |  |
| --- | --- | --- |
| 2 | 4 | 1 |
| one | two | five |

Операция проверки на полноту:

Результат: true

### Упорядоченная таблица

Упорядоченная таблица представлена массивом указателей на записи, где каждая запись представляет собой пару ключ-значение. Упорядоченная таблица поддерживает операции поиска, операции и удаления, проверки на пустоту и полноту. Структура данных хранит элементы в первых n ячейках массива записей в отсортированном по ключам по не убыванию. Поддержка элементов в отсортированном порядке позволяет быстрее искать элемент в таблице при помощи алгоритма «Бинарный поиск».

**Функция хэширования**

Алгоритм возвращает хэш-код ключа, используя стандартную хэш-функцию. Затем полученный хэш-код делится на максимальное число элементов таблицы, что позволяет определить индекс ячейки хэш-таблицы, где будет храниться информация с данным ключом. Этот процесс обеспечивает равномерное распределение данных по таблице и быстрый доступ к записям.

**Операция добавления**

Операция добавления элемента реализуется при помощи поиска алгоритма «Бинарный поиск», который находит позицию для искомого элемента и добавляет туда переданную запись, увеличивая количество элементов в таблице. Все элементы, имеющие индекс больше, сдвигаются вправо. Если этот элемент уже был в таблице, запись перезаписывается.

Пример:

|  |  |  |
| --- | --- | --- |
| 2 | 4 |  |
| one | two |  |

Операция добавления элемента two с ключом 3:

|  |  |  |
| --- | --- | --- |
| 2 | 3 | 4 |
| one | two | five |

**Операция удаления**

Операция удаления элемента реализуется при помощи алгоритма «Бинарный поиск», который находит элемент за оптимальное время. Элемент удаляется и уменьшается текущее количество записей в таблице. Если элемент не был последним в таблице, все записи, у которых позиция больше, чем у искомого, будут сдвинуты влево.

Пример:

|  |  |  |
| --- | --- | --- |
| 2 | 4 |  |
| one | two |  |

Операция удаления элемента с ключом 2:

|  |  |  |
| --- | --- | --- |
| 4 |  |  |
| two |  |  |

**Операция поиска**

Операция поиска элемента в упорядоченной таблице реализуется с помощью «Бинарного поиска». Алгоритм «Бинарного поиска» в отсортированных таблицах использует стратегию разделения интервала пополам, что позволяет эффективно находить элементы. Сравнивая ключ с целевым значением и последовательно сужая интервал поиска, алгоритм быстро находит искомый элемент и возвращает указатель на него или возвращает нулевой указатель, если элемент не найден.

Пример:

|  |  |  |
| --- | --- | --- |
| 2 | 4 |  |
| one | two |  |

Операция поиска элемента с ключом 2:

|  |
| --- |
| 2 |
| one |

**Операция проверки на пустоту**

Операция проверки на пустоту реализована при помощи переменной, отвечающей за текущее количество записей в таблице. Если она равна нулю, то таблица пуста, иначе в ней имеются записи.

Пример:

|  |  |  |
| --- | --- | --- |
| 2 | 4 |  |
| one | two |  |

Операция проверки на пустоту:

Результат: false

**Операция проверки на полноту**

Операция проверки на полноту реализована при помощи переменной, отвечающей за текущее количество записей в таблице. Если она равна максимальному количеству записей в таблице, то таблица полна.

Пример:

|  |  |  |
| --- | --- | --- |
| 2 | 4 | 1 |
| one | two | five |

Операция проверки на полноту:

Результат: true

### Хэш таблица

Хэш таблица представлена массивом указателей на записи, где каждая запись представляет собой пару ключ-значение и наличием указателя на фиктивный элемент, который будет означать, что текущий элемент был удалён. Этот указатель нужен для обработки коллизий с помощью подхода «открытое перемешивание». В таблице реализована функция хэширования, которая должна ускорить поиск элемента. Структура данных хранит элементы хаотичном порядке, согласно хэш функции.

**Операция добавления**

Операция добавления элемента реализуется при помощи хэш функции, которая считает позицию для искомого элемента. Если текущая ячейка занята каким-то другим элементом, то проверяются все последующие элементы таблицы, используя линейный сдвиг, пока не найдем пустую ячейку, либо не вернёмся обратно. Если структура хранения не полна, то добавляем новый элемент в пустую ячейку. Если этот элемент уже был в таблице, запись перезаписывается.

Пример:

|  |  |  |
| --- | --- | --- |
| 2 |  | 4 |
| one |  | two |

Операция добавления элемента five с ключом 3:

|  |  |  |
| --- | --- | --- |
| 2 | 3 | 4 |
| one | five | two |

**Операция удаления**

Операция удаления элемента реализуется при помощи алгоритма «Бинарный поиск», который находит элемент за оптимальное время. Если запись найдена, она удаляется, освобождая память, и количество записей в таблице уменьшается.

Пример:

|  |  |  |
| --- | --- | --- |
| 2 |  | 4 |
| one |  | two |

Операция удаления элемента с ключом 4:

|  |  |  |
| --- | --- | --- |
| 2 |  |  |
| one |  |  |

**Операция поиска**

Операция поиска элемента реализуется при помощи хэш функции, которая считает позицию для искомого элемента. Если текущая ячейка занята каким-то другим элементом, то проверяются все последующие элементы таблицы, используя линейный сдвиг, пока либо не упрёмся в ячейку, в которой никто не был, либо не вернёмся обратно. Если искомый элемент не был найден, возвращаем пустой указатель, иначе – указатель на запись.

Пример:

|  |  |  |
| --- | --- | --- |
| 2 |  | 4 |
| one |  | two |

Операция поиска элемента с ключом 2:

|  |
| --- |
| 2 |
| one |

**Операция проверки на пустоту**

Операция проверки на пустоту реализована при помощи переменной, отвечающей за текущее количество записей в таблице. Если она равна нулю, то таблица пуста, иначе в ней имеются записи.

Пример:

|  |  |  |
| --- | --- | --- |
| 2 | 4 |  |
| one | two |  |

Операция проверки на пустоту:

Результат: false

**Операция проверки на полноту**

Операция проверки на полноту реализована при помощи переменной, отвечающей за текущее количество записей в таблице. Если она равна максимальному количеству записей в таблице, то таблица полна.

Пример:

|  |  |  |
| --- | --- | --- |
| 2 | 4 | 1 |
| one | two | five |

Операция проверки на полноту:

Результат: true

### Описание класса TTabRecord

template <typename TKey, typename TData>

class TTabRecord {

protected:

TKey key;

TData\* data;

public:

TTabRecord();

~TTabRecord();

TTabRecord(const TKey& k, const TData& d);

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

TKey GetKey() const;

TData\* GetData() const;

};

Назначение: представление записи в таблицы.

Поля:

**data**– указатель на массив типа **TData**.

key – ключ к записи.

Методы:

TTabRecord();

Назначение: конструктор по умолчанию.

Входные параметры: отсутствуют.

Выходные параметры: новая запись с инициализированными значениями.

~TTabRecord();

Назначение: деструктор.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

TTabRecord(const TKey& k, const TData& d);

Назначение: инициализация значения **data** записи и ключа k к этой записи.

Входные параметры: **d** – значение **data**, **k** – значение **key.**

Выходные параметры: новая запись с инициализированным значением **data** и **key**.

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

Назначение: конструктор копирования.

Входные параметры: record–запись, на основе которой создаем копию.

Выходные параметры: отсутствуют.

TKey GetKey() const;

Назначение: получение значения ключа.

Входные параметры: отсутствуют.

Выходные параметры: ключ.

TData\* GetData() const;

Назначение: получение значения данных записи.

Входные параметры: отсутствуют.

Выходные параметры: данные.

### Описание класса TTable

template <typename TKey, typename TData>

class TTable

{

protected:

int size;

int maxSize;

int currPos;

public:

TTable()};

TTable(int \_maxSize);

virtual void Insert(const TKey& k, const TData& d) = 0;

virtual void Remove(const TKey& k) = 0;

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

virtual TKey GetKey() const = 0;

virtual TData\* GetData() const = 0;

int GetSize() const;

virtual bool IsTabEnded() const;

virtual bool Reset();

virtual bool Next();

bool IsFull() const;

bool IsEmpty() const;

};

Назначение: абстрактный класс для представления различных видов таблиц

Поля:

size – количество элементов в таблице.

maxSize – максимальное число элементов в таблицы.

currPos – индекс текущего элемента.

Методы:

TTable()

Назначение: конструктор по умолчанию.

Входные параметры отсутствуют:

Выходные параметры: отсутствуют.

TTable(int \_maxSize);

Назначение: конструктор с параметрами.

Входные параметры: \_maxSize – максимальное количество элементов в таблице.

Выходные параметры: отсутствуют.

virtual void Insert(const TKey& k, const TData& d) = 0;

Назначение: вставка записи в таблицу.

Входные параметры: k – ключ, d – запись, которую хотим вставить.

Выходные параметры: отсутствуют.

virtual void Remove(const TKey& k) = 0;

Назначение: удаление записи из таблицы.

Входные параметры: k – ключ.

Выходные параметры: отсутствуют.

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

Назначение: поиск записи в таблице по ключу.

Входные параметры: k – ключ, который ищем.

Выходные параметры: указатель на запись. Если запись не была найдена, вернётся нулевой указатель.

virtual TKey GetKey() const = 0;

Назначение: получение значения ключа.

Входные параметры: отсутствуют.

Выходные параметры: ключ.

virtual TData\* GetData() const = 0;

Назначение: получение значения записи.

Входные параметры: отсутствуют.

Выходные параметры: запись.

int GetSize() const;

Назначение: получение текущего количества записей.

Входные параметры: отсутствуют.

Выходные параметры: количество записей size.

virtual bool isFull() const;

Назначение: проверка таблицы на полноту.

Входные параметры отсутствуют:

Выходные параметры: true – если таблица полна, false – в противном случае.

virtual bool isEmpty() const;

Назначение: проверка на пустоту.

Входные параметры: отсутствуют.

Выходные параметры: true – если таблица пуста, false – в противном случае.

virtual bool Next();

Назначение: переход к следующей записи.

Входные параметры: отсутствуют.

Выходные параметры: false, если индекс вышел за пределы таблицы, иначе true.

virtual bool IsTabEnded() const;

Назначение: проверяет, дошли ли мы до конца таблицы, то есть проверяет currPos >= maxSize

Входные параметры: отсутствуют.

Выходные параметры: true – если дошли (больше), false – в противном случае.

virtual bool Reset();

Назначение: ставит индекс текущего элемента на начало.

Входные параметры: отсутствуют.

Выходные параметры: false, если индекс вышел за пределы таблицы, иначе true.

### Описание класса TScanTable

template<typename TKey, typename TData>

class TScanTable : public TTable<TKey,TData>

{

protected:

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

public:

TScanTable(int \_maxSize);

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

~TScanTable();

void Insert(const TKey& k, const TData& d) override;

virtual void Remove(const TKey& k) override;

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

virtual TKey GetKey() const override;

virtual TData\* GetData() const override;

friend std::ostream& operator<<(std::ostream& out, const TScanTable<TKey, TData>& stab);

}

Назначение: представление просматриваемой неупорядоченной таблицы

Поля:

records – массив указателей на записи.

Методы:

TScanTable(int \_maxSize);

Назначение: конструктор с параметром.

Входные параметры: \_maxSize – максимальное количество элементов в таблице.

Выходные параметры: отсутствуют.

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

Назначение: конструктор копирования

Входные параметры: obj – ссылка на существующую таблицу, на основе которой будет создан новый.

Выходные параметры: отсутствуют.

~TScanTable();

Назначение: деструктор.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

virtual void Insert(const TKey& k, const TData& d) override;

Назначение: вставка записи в таблицу.

Входные параметры: k – ключ, d – запись, которую хотим вставить.

Выходные параметры: отсутствуют.

virtual void Remove(const TKey& k) override;

Назначение: удаление записи из таблицы.

Входные параметры: k – ключ.

Выходные параметры: отсутствуют.

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

Назначение: поиск записи в таблице по ключу.

Входные параметры: k – ключ, который ищем.

Выходные параметры: указатель на запись. Если запись не была найдена, вернётся нулевой указатель.

virtual TKey GetKey() const override;

Назначение: получение значения ключа.

Входные параметры: отсутствуют.

Выходные параметры: ключ.

virtual TData\* GetData() const override;

Назначение: получение значения записи.

Входные параметры: отсутствуют.

Выходные параметры: запись.

Friend std::ostream& operator<<(std::ostream& out, const TScanTable<TKey, TData>& stab);

Назначение: оператор вывода.

Входные параметры: **out** – ссылка на объект типа **ostream**, который представляет

выходной поток, stab – ссылка на объект типа TScanTable<TKey, TData> который будет выводиться.

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

### **Описание класса** SortedTable

template <typename TKey, typename TData>

class SortedTable : public TScanTable<TKey, TData>

{

private:

void Sort();

void QuickSort(TTabRecord<TKey, TData>\* a, int n);

public:

SortedTable(int \_maxSize);

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

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

virtual void Insert(const TKey& k, const TData& d) override;

virtual void Remove(const TKey& k) override;

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

};

Назначение: представление упорядоченной таблицы.

Поля: отсутствуют.

Методы:

SortedTable(int \_maxSize);

Назначение: конструктор с параметром.

Входные параметры: \_maxSize – максимальное количество элементов в таблице.

Выходные параметры: отсутствуют.

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

Назначение: конструктор с параметрами, копирующий данные неупорядоченной таблицы.

Входные параметры: st – таблица, которую копируем.

Выходные параметры: отсутствуют.

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

Назначение: конструктор копирования.

Входные параметры: **srt** – ссылка на существующий объект SortedTable<TKey, TData>, на основе которого создаем новую упорядоченную таблицу.

Выходные параметры: отсутствуют.

virtual void Insert(const TKey& k, const TData& d) override;

Назначение: вставка записи в таблицу.

Входные параметры: k – ключ, d – запись, которую хотим вставить.

Выходные параметры: отсутствуют.

virtual void Remove(const TKey& k) override;

Назначение: удаление записи из таблицы.

Входные параметры: k – ключ.

Выходные параметры: отсутствуют.

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

Назначение: поиск записи в таблице по ключу.

Входные параметры: k – ключ, который ищем.

Выходные параметры: указатель на запись. Если запись не была найдена, вернётся нулевой указатель.

### Описание класса THashTable

template <typename TKey, typename TData>

class THashTable : public TTable<TKey, TData>

{

protected:

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

public:

THashTable(size\_t \_size): TTable<TKey,TData>(\_size);

};

Назначение: абстрактный базовый класс для хэш таблиц

Поля: отсутствуют.

Методы:

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

Назначение: функция хэширования.

Входные параметры: k – ключ, по которому ищем элемент.

Выходные параметры: хэш, первый возможный индекс элемента в таблице.

THashTable(size\_t \_size): TTable<TKey,TData>(\_size) ;

Назначение: конструктор с параметрами.

Входные параметры: \_size – максимальный размер таблицы.

Выходные параметры: отсутствуют.

### Описание класса TArrayHashTable

template <typename TKey, typename TData>

class TArrayHashTable : public THashTable<TKey, TData>

{

protected:

size\_t hash\_step;

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

size\_t GetNextPos(size\_t index);

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

TTabRecord<TKey, TData>\* pMark;

int freePosIndex;

public:

TArrayHashTable(size\_t maxSize, size\_t step);

TArrayHashTable(const TArrayHashTable& obj);

virtual ~TArrayHashTable();

virtual void Insert(const TKey& k, const TData& d) override;

virtual void Remove(const TKey& k) override;

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

virtual bool Reset() override;

virtual bool Next() override;

virtual TKey GetKey() const override;

virtual TData\* GetData() const override;

friend std::ostream& operator<<(std::ostream& out, const TArrayHashTable<TKey, TData>& ht);

}

Назначение: представление хэш таблиц с помощью метода открытого перемешивания

Поля:

hash\_step - линейный шаг в методе «открытое перемешивание».

records – массив указателей на записи.

pMark - фиктивная запись, которая означает, что элемент в ячейке был удалён.

freePosIndex – индекс свободной позиции.

Методы:

size\_t GetNextPos(size\_t index);

Назначение: получение следующей позиции с учетом заданной.

Входные параметры: index – текущий индекс в хэш-таблице, для которого нужно рассчитать следующую позицию.

Выходные параметры: следующая позиция.

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

Назначение: функция хэширования.

Входные параметры: k– ключ, по которому ищем элемент.

Выходные параметры: хэш, первый возможный индекс элемента в таблице.

TArrayHashTable(size\_t maxSize, size\_t step);

Назначение: конструктор с параметрами.

Входные параметры: maxSize – максимальный размер таблицы, step - линейный шаг.

Выходные параметры: отсутствуют.

TArrayHashTable(const TArrayHashTable& obj);

Назначение: конструктор копирования.

Входные параметры: obj – ссылка на существующую таблицу, на основе которой будет создана новая.

Выходные параметры: отсутствуют.

virtual ~TArrayHashTable();

Назначение: деструктор.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

virtual void Insert(const TKey& k, const TData& d) override;

Назначение: вставка записи в таблицу.

Входные параметры: k – ключ, d – запись, которую хотим вставить.

Выходные параметры: отсутствуют.

virtual void Remove(const TKey& k) override;

Назначение: удаление записи из таблицы.

Входные параметры: k – ключ.

Выходные параметры: отсутствуют.

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

Назначение: поиск записи в таблице по ключу.

Входные параметры: k – ключ, который ищем.

Выходные параметры: указатель на запись. Если запись не была найдена, вернётся нулевой указатель.

virtual TKey GetKey() const override;

Назначение: получение значения ключа.

Входные параметры: отсутствуют.

Выходные параметры: ключ.

virtual TData\* GetData() const override;

Назначение: получение значения записи.

Входные параметры: отсутствуют.

Выходные параметры: запись.

virtual bool Next();

Назначение: переход к следующей записи.

Входные параметры: отсутствуют.

Выходные параметры: false, если индекс вышел за пределы таблицы, иначе true.

virtual bool Reset();

Назначение: ставит индекс текущего элемента на начало.

Входные параметры: отсутствуют.

Выходные параметры: false, если индекс вышел за пределы таблицы, иначе true.

friend std::ostream& operator<<(std::ostream& out, const TArrayHashTable<TKey, TData>& ht);

Назначение: оператор вывода.

Входные параметры: **out** – ссылка на объект типа **ostream**, который представляет

выходной поток, ht – ссылка на объект типа TArrayHashTable <TKey, TData> который будет выводиться.

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

# Заключение

В рамках данной лабораторной работы была разработана и реализована структура данных для работы с записями таблиц. Были созданы классы SortedTable, TScanTable, TArrayHashTable предоставляющие функционал для работы с упорядоченной, просматриваемой, хэш таблицы соответственно.

Таким образом, результатом выполнения лабораторной работы стала реализация структуры данных для работы с записями в таблицах, позволяющей удобно и эффективно выполнять различные операции над ними. В качестве записей выступали полиномы, реализованные в прошлой лабораторной работе. Программа позволяет выполнять арифметические операции над ними в таблицах, вставлять, удалять и выводить полиномы.

# Литература

1. Неупорядоченные таблицы [https://studfile.net/preview/7081338/page:6/].
2. Упорядоченные таблицы [https://studfile.net/preview/1047385/].
3. Хэш таблица [https://ru.wikipedia.org/wiki/Хеш-таблица]
4. Хэш таблица [https://neerc.ifmo.ru/wiki/index.php?title=Хеш-таблица]

# Приложения

## Приложение А. Реализация класса TTabRecord

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

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

template <typename TKey, typename TData>

class TTabRecord {

protected:

TKey key;

TData\* data;

public:

TTabRecord();

~TTabRecord() { };

TTabRecord(const TKey& k, const TData& d);

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

TKey GetKey() const;

TData\* GetData() const;

};

template <typename TKey, typename TData> TTabRecord<TKey, TData>::TTabRecord()

{

}

template <typename TKey, typename TData> TTabRecord<TKey, TData>::TTabRecord(const TKey& k, const TData& d)

{

key = k;

data = new TData(d);

}

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

{

key = record.key;

data = record.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;

}

#endif

## Приложение Б. Реализация класса TTable

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

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

#include "tabRecord.h"

#include <iostream>

template <typename TKey, typename TData>

class TTable

{

protected:

int size;

int maxSize;

int currPos;

public:

TTable() { size = 0; currPos = -1; };

TTable(int \_maxSize);

virtual void Insert(const TKey& k, const TData& d) = 0;

virtual void Remove(const TKey& k) = 0;

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

//Get

virtual TKey GetKey() const = 0;

virtual TData\* GetData() const = 0;

int GetSize() const;

//навигация

virtual bool IsTabEnded() const;

virtual bool Reset();

virtual bool Next();

bool IsFull() const;

bool IsEmpty() const;

};

template <typename TKey, typename TData> TTable<TKey, TData>::TTable(int \_maxSize)

{

maxSize = \_maxSize;

size = 0;

currPos = -1;

}

template <typename TKey, typename TData> int TTable<TKey, TData>::GetSize() const

{

return size;

}

template <typename TKey, typename TData> bool TTable<TKey, TData>::IsFull() const

{

return (size == maxSize);

}

template <typename TKey, typename TData> bool TTable<TKey, TData>::IsEmpty() const

{

return (size == 0);

}

template <typename TKey, typename TData> bool TTable<TKey, TData>::IsTabEnded() const

{

return (currPos >= maxSize);

}

template <typename TKey, typename TData> bool TTable<TKey, TData>::Reset()

{

if (!IsEmpty())

{

currPos = 0;

return true;

}

else

{

currPos = -1;

return false;

}

}

template <typename TKey, typename TData> bool TTable<TKey, TData>::Next()

{

if ( currPos < maxSize && !IsEmpty())

{

currPos++;

return true;

}

else

{

return false;

}

}

#endif

## Приложение В. Реализация класса TScanTable

#ifndef \_\_SCANTABLE\_H\_\_

#define \_\_SCANTABLE\_H\_\_

#include "table.h"

#include <iomanip>

template<typename TKey, typename TData>

class TScanTable : public TTable<TKey,TData>

{

protected:

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

public:

TScanTable(int \_maxSize);

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

~TScanTable();

void Insert(const TKey& k, const TData& d) override;

virtual void Remove(const TKey& k) override;

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

virtual TKey GetKey() const override;

virtual TData\* GetData() const override;

friend std::ostream& operator<<(std::ostream& out, const TScanTable<TKey, TData>& stab)

{

std::cout << "-------------------------------------" << std::endl;

for (int i = 0; i < stab.maxSize; i++) {

if (stab.records[i] != nullptr)

{

out << "| " << std::setw(20) << stab.records[i]->GetKey() << " | " << std::setw(20) << \*(stab.records[i]->GetData()) << std::endl;

}

}

return out;

}

};

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

{

records = new TTabRecord<TKey, TData>\*[\_maxSize]();

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

{

records[i] = nullptr;

}

size = 0;

currPos = 0;

maxSize = \_maxSize;

}

template<typename TKey, typename TData> TScanTable<TKey, TData>::TScanTable(const TScanTable<TKey, TData>& obj)

{

size = obj.size;

currPos = obj.currPos;

maxSize = obj.maxSize;

records = new TTabRecord<TKey, TData>\*[maxSize]();

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

{

if (obj.records[i])

records[i] = new TTabRecord<TKey, TData>(\*obj.records[i]);

else

records[i] = nullptr;

}

}

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

{

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

{

if (records[i])

{

delete records[i];

}

}

delete[] records;

}

template<typename TKey, typename TData> void TScanTable<TKey, TData>::Insert(const TKey& k, const TData& d)

{

if (IsFull())

{

throw "Table is full. Can't insert";

}

records[size++] = new TTabRecord<TKey,TData>(k, d);

}

template<typename TKey, typename TData> TTabRecord<TKey, TData>\* TScanTable<TKey, TData>::Find(const TKey& k)

{

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

{

if (records[i]->GetKey() == k) {

currPos = i;

return records[i];

}

}

return nullptr;

}

template<typename TKey, typename TData> void TScanTable<TKey, TData>::Remove(const TKey& k)

{

if ( this->IsEmpty()) {

throw "Nothing to remove. Table is empty";

}

if (this->Find(k) != nullptr) {

delete records[currPos];

for (int i = currPos; i < size; i++) {

records[i] = records[i + 1];

size--;

}

}

else

throw "Record not found";

}

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

{

if (currPos < size)

return records[currPos]->GetKey();

else

{

throw "Can't get key.Current position is out of range";

}

}

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

{

if (currPos < size)

return records[currPos]->GetData();

else

{

throw "Can't get data. Current position is out of range";

}

}

#endif

## Приложение Г. Реализация класса SortedTable

#ifndef \_\_SORTEDTABLE\_H\_\_

#define \_\_SORTEDTABLE\_H\_\_

#include "scanTable.h"

template <typename TKey, typename TData>

class SortedTable : public TScanTable<TKey, TData>

{

private:

void Sort();

void QuickSort(TTabRecord<TKey, TData>\* a, int n);

public:

SortedTable(int \_maxSize);

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

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

virtual void Insert(const TKey& k, const TData& d) override;

virtual void Remove(const TKey& k) override;

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

};

template<typename TKey, typename TData>

void SortedTable<TKey, TData>::QuickSort(TTabRecord<TKey, TData>\* a, int n)

{

int i = 0;

int j = n - 1;

TTabRecord<TKey, TData> pivot = a[n / 2];

do {

while (a[i].GetKey() < pivot.GetKey())

i++;

while (a[j].GetKey() > pivot.GetKey())

j--;

if (i <= j) {

TTabRecord<TKey, TData> tmp = a[i];

a[i] = a[j];

a[j] = tmp;

i++;

j--;

}

} while (i <= j);

if (j > 0)

QuickSort(a, j + 1);

if (i < n)

QuickSort(a + i, n - i);

}

template<typename TKey, typename TData> void

SortedTable<TKey, TData>::Sort()

{

QuickSort(records, size);

}

template <typename TKey, typename TData>

SortedTable<TKey, TData>::SortedTable(int max\_size) : TScanTable<TKey, TData>(max\_size) {}

template <typename TKey, typename TData>

SortedTable<TKey, TData>::SortedTable(const TScanTable<TKey, TData>& st) : TScanTable<TKey, TData>(st)

{

Sort();

}

template <typename TKey, typename TData>

SortedTable<TKey, TData>::SortedTable(const SortedTable<TKey, TData>& srt) : TScanTable<TKey, TData>(srt) { this->Sort(); }

template<typename TKey, typename TData> TTabRecord<TKey, TData>\* SortedTable<TKey, TData>::Find(const TKey& k)

{

int i1 = 0;

int i2 = size - 1;

while (i1 <= i2)

{

int mid = (i1 + i2) / 2;

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

{

currPos = mid;

return records[mid];

}

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

{

currPos = mid;

i1 = mid + 1;

}

else

i2 = mid - 1;

}

currPos = i1;

return nullptr;

}

template<typename TKey, typename TData>

void SortedTable<TKey, TData>::Insert(const TKey& k, const TData& d) {

if (IsFull())

throw "Table is full. Can't insert";

if (records[currPos] != nullptr && records[currPos]->GetKey() < k)

currPos++;

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

{

records[i + 1] = records[i];

}

records[currPos] = new TTabRecord<TKey, TData>(k, d);

size++;

}

template<typename TKey, typename TData>

void SortedTable<TKey, TData>::Remove(const TKey& k)

{

if (IsEmpty())

throw "Nothing to remove. Table is empty";

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

if (rec == nullptr)

throw "Record not found";

delete rec;

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

{

records[i] = records[i + 1];

}

size--;

}

#endif

## Приложение Д. Реализация класса THashTable

#ifndef \_\_HASHTABLE\_H\_\_

#define \_\_HASHTABLE\_H\_\_

#include "table.h"

template <typename TKey, typename TData>

class THashTable : public TTable<TKey, TData>

{

protected:

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

public:

THashTable(size\_t \_size): TTable<TKey,TData>(\_size) {};

};

#endif

## Приложение Е. Реализация класса TArrayHashTable

#ifndef \_\_ARRAYHASHTABLE\_H\_\_

#define \_\_ARRAYHASHTABLE\_H\_\_

#include "hashTable.h"

#include <iostream>

#include <iomanip>

template <typename TKey, typename TData>

class TArrayHashTable : public THashTable<TKey, TData>

{

protected:

size\_t hash\_step;

TTabRecord<TKey, TData>\*\* records; //массив указателей на записи

size\_t GetNextPos(size\_t index) { return (index + hash\_step) % maxSize; }

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

TTabRecord<TKey, TData>\* pMark; //пометка о том, что был удален

int freePosIndex; //index свободной позиции

public:

TArrayHashTable(size\_t maxSize, size\_t step);

TArrayHashTable(const TArrayHashTable& obj);

virtual ~TArrayHashTable();

virtual void Insert(const TKey& k, const TData& d) override;

virtual void Remove(const TKey& k) override;

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

virtual bool Reset() override;

virtual bool Next() override;

virtual TKey GetKey() const override;

virtual TData\* GetData() const override;

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

{

std::cout << "-------------------------------------" << std::endl;

for (int i = 0; i < ht.maxSize; i++) {

if (ht.records[i] != ht.pMark && ht.records[i] != nullptr)

{

out << "| " << std::setw(20) << ht.records[i]->GetKey() << " | " << std::setw(20) << \*(ht.records[i]->GetData()) << std::endl;

}

}

return out;

}

};

template <typename TKey, typename TData>

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

{

records = new TTabRecord<TKey, TData>\* [\_maxSize];

hash\_step = \_step;

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

freePosIndex = -1;

for (int i = 0; i < \_maxSize; i++) {

records[i] = nullptr;

}

}

template <typename TKey, typename TData>

TArrayHashTable<TKey, TData>::TArrayHashTable(const TArrayHashTable& obj):THashTable<TKey, TData>(obj.maxSize)

{

maxSize = obj.maxSize;

currPos = obj.currPos;

hash\_step = obj.hash\_step;

freePosIndex = obj.freePosIndex;

records = new TTabRecord<TKey, TData>\* [maxSize];

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

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

{

TTabRecord<TKey, TData>\* tmp = obj.records[i];

if (obj.records[i] == obj.pMark) {

records[i] = pMark;

}

else if (obj.records[i] == nullptr) {

records[i] = nullptr;

}

else

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

}

}

template <typename TKey, typename TData>

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

{

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

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

{

delete records[i];

}

}

if (records)

delete[] records;

if (pMark)

delete pMark;

}

template<typename TKey, typename TData>

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

{

std::hash<TKey> hasher;

return hasher(k) % maxSize;

}

template<typename TKey, typename TData>

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

{

currPos = 0;

while (!IsTabEnded())

{

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

break;

currPos++;

}

return IsTabEnded();

}

template<typename TKey, typename TData>

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

{

currPos++;

while (!IsTabEnded())

{

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

break;

currPos++;

}

return IsTabEnded();

}

template<typename TKey, typename TData>

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

{

if (currPos < maxSize)

{

return records[currPos]->GetKey();

}

else

throw "Error. Can't get key. Current position out of range";

}

template<typename TKey, typename TData>

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

{

if (currPos < maxSize)

{

return records[currPos]->GetData();

}

else

throw "Error. Can't get data. Current position out of range";

}

template<typename TKey, typename TData>

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

{

currPos = hash\_func(k);

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

{

if (records[currPos] == nullptr)

{

return nullptr;

}

else

if (records[currPos] == pMark && freePosIndex == -1)

{

freePosIndex = currPos;

}

else

if (records[currPos]->GetKey() == k)

{

return records[currPos];

}

currPos = GetNextPos(currPos);

}

return nullptr;

}

template<typename TKey, typename TData>

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

{

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

if (tmp == nullptr)

{

throw " Error. Record is empty";

}

delete tmp;

records[currPos] = pMark;

freePosIndex = -1;

size--;

}

template<typename TKey, typename TData>

void TArrayHashTable<TKey, TData>::Insert(const TKey& k, const TData& d)

{

if (IsFull())

{

throw "Error. Can't insert, table is full";

}

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

{

currPos = freePosIndex;

}

records[currPos] = new TTabRecord<TKey, TData>(k, d);

size++;

}

#endif

## Приложение Ж. Sample\_table

#include <iostream>

#include "table.h"

#include "scanTable.h"

#include "sortedTable.h"

#include "tpolynom.h"

#include "arrayHashTable.h"

TScanTable<std::string, TPolinom> scan\_table(101);

SortedTable<std::string, TPolinom> sorted\_table(101);

TArrayHashTable<std::string, TPolinom> hash\_table(101, 1);

int Correct(int type)

{

while (type != 1 && type != 2 && type != 3 && type != 4)

{

std::cout << " Please, choose correctly: \n 1 - Scan table \n 2 - Sorted table \n 3 - Hash table\n 4 - All\n";

std::cin >> type;

}

return type;

}

void Print()

{

int type;

std::cout << " Which table you want to print: \n 1 - Scan table \n 2 - Sorted table \n 3 - Hash table\n 4 - All\n";

std::cin >> type;

type = Correct(type);

switch (type)

{

case 1:

std::cout << "Table: " << std::endl;

std::cout << scan\_table << std::endl;

break;

case 2:

std::cout << "Sorted table: " << std::endl;

std::cout << sorted\_table << std::endl;

break;

case 3:

std::cout << "Hash table: " << std::endl;

std::cout << hash\_table << std::endl;

break;

case 4:

std::cout << "Table: " << std::endl;

std::cout << scan\_table << std::endl;

std::cout << "Sorted table: " << std::endl;

std::cout << sorted\_table << std::endl;

std::cout << "Hash table: " << std::endl;

std::cout << hash\_table << std::endl;

break;

}

}

void insertResult(TPolinom res, int type)

{

type = Correct(type);

switch (type)

{

case 1:

if (!scan\_table.IsFull()) {

scan\_table.Insert(res.getFormula(), res);

break;

}

else

{

std::cout << " Sorry. Table is full.Please, choose another table: \n 2 - Sorted table \n 3 - Hash table\n ";

std::cin >> type;

while (type != 2 && type != 3)

std::cin >> type;

insertResult(res, type);

break;

}

case 2:

if (!sorted\_table.IsFull())

{

sorted\_table.Insert(res.getFormula(), res);

break;

}

else

{

std::cout << " Sorry. Sorted table is full.Please, choose another table: \n 1 - Scan table \n 3 - Hash table\n ";

std::cin >> type;

while (type != 1 && type != 3)

std::cin >> type;

insertResult(res, type);

break;

}

case 3:

if (!hash\_table.IsFull())

{

hash\_table.Insert(res.getFormula(), res);

break;

}

else

{

std::cout << " Sorry. Hash table is full.Please, choose another table: \n 1 - Scan table \n 2 - Sorted table\n ";

std::cin >> type;

while (type != 1 && type != 2)

std::cin >> type;

insertResult(res, type);

break;

}

case 4:

if (!sorted\_table.IsFull() && !scan\_table.IsFull() && !hash\_table.IsFull())

{

scan\_table.Insert(res.getFormula(), res);

sorted\_table.Insert(res.getFormula(), res);

hash\_table.Insert(res.getFormula(), res);

break;

}

else

{

std::cout << " Sorry. All tables are full." << std::endl;

break;

}

}

}

void Insert()

{

std::string s1;

std::cout << "Please, enter polynom to insert \n";

std::cin >> s1;

TPolinom p1(s1);

int type;

std::cout << " In which table you want to insert: \n 1 - Scan table \n 2 - Sorted table \n 3 - Hash table\n 4 - All\n";

std::cin >> type;

insertResult(p1, type);

}

void Remove()

{

std::string s1;

std::cout << "Please, enter polynom to remove \n";

std::cin >> s1;

TPolinom p1(s1);

while (scan\_table.Find(p1.getFormula()) == nullptr && sorted\_table.Find(p1.getFormula()) == nullptr && hash\_table.Find(p1.getFormula()) == nullptr)

{

bool flag;

flag = scan\_table.Find(p1.getFormula());

flag = sorted\_table.Find(p1.getFormula());

flag = hash\_table.Find(p1.getFormula());

std::cout << "Sorry. All tables doesn't have this polynom." << std::endl;

std::cout << "Please, enter exist polynom " << std::endl;

std::cin >> s1;

p1 = TPolinom(s1);

}

int type;

std::cout << " In which table you want to remove polynom: \n 1 - Scan table \n 2 - Sorted table \n 3 - Hash table\n 4 - All\n";

std::cin >> type;

type = Correct(type);

switch (type)

{

case 1:

if (scan\_table.Find(p1.getFormula()) != nullptr)

{

scan\_table.Remove(p1.getFormula());

}

else

{

std::cout << "Sorry. Scan table doesn't contain this polynom\n";

}

break;

case 2:

if (sorted\_table.Find(p1.getFormula()) != nullptr)

{

sorted\_table.Remove(p1.getFormula());

}

else

{

std::cout << "Sorry. Sorted table doesn't contain this polynom\n";

}

break;

case 3:

if (hash\_table.Find(p1.getFormula()) != nullptr)

{

hash\_table.Remove(p1.getFormula());

}

else

{

std::cout << "Sorry. Hash table doesn't contain this polynom\n";

}

break;

case 4:

if (hash\_table.Find(p1.getFormula()) != nullptr && scan\_table.Find(p1.getFormula()) != nullptr && sorted\_table.Find(p1.getFormula()) != nullptr)

{

scan\_table.Remove(p1.getFormula());

sorted\_table.Remove(p1.getFormula());

hash\_table.Remove(p1.getFormula());

}

else

{

std::cout << "Sorry. Not all tables have this polynom\n";

}

break;

scan\_table.Insert(p1.getFormula(), p1);

sorted\_table.Insert(p1.getFormula(), p1);

hash\_table.Insert(p1.getFormula(), p1);

break;

}

}

void Result(TPolinom result)

{

int type;

std::cout << "Result = " << result.getFormula() << std::endl;

std::cout << " In which table you want to insert the result: \n 1 - Scan table \n 2 - Sorted table \n 3 - Hash table\n 4 - All\n";

std::cin >> type;

insertResult(result, type);

}

//-------------In Table----------------------\\

bool inScanTable(TPolinom p1, TPolinom p2)

{

if (scan\_table.Find(p1.getFormula()) != nullptr && scan\_table.Find(p2.getFormula()) != nullptr)

return true;

return false;

}

bool inSortedTable(TPolinom p1, TPolinom p2)

{

if (sorted\_table.Find(p1.getFormula()) != nullptr && sorted\_table.Find(p2.getFormula()) != nullptr)

return true;

return false;

}

bool inHashTable(TPolinom p1, TPolinom p2)

{

if (hash\_table.Find(p1.getFormula()) != nullptr && hash\_table.Find(p2.getFormula()) != nullptr)

return true;

return false;

}

bool inOneTable(TPolinom p1, TPolinom p2)

{

if (inScanTable(p1, p2) || inSortedTable(p1, p2) || inHashTable(p1, p2))

return true;

return false;

}

void Dif()// DIF - to take the derivative

{

std::string s1, dif;

std::cout << "Please, enter polynom which one you want to derivative " << std::endl;

std::cin >> s1;

TPolinom p1(s1);

TPolinom result;

while (scan\_table.Find(p1.getFormula()) == nullptr && sorted\_table.Find(p1.getFormula()) == nullptr && hash\_table.Find(p1.getFormula()) == nullptr)

{

std::cout << "Sorry. All tables doesn't have this polynom." << std::endl;

Print();

std::cout << "Please, enter exist polynom " << std::endl;

std::cin >> s1;

p1 = TPolinom(s1);

}

std::cout << "Now choose the variable with respect to which you want to take the derivative: x, y, z" << std::endl;

std::cin >> dif;

while (dif != "x" && dif != "y" && dif != "z")

{

std::cout << " Please, choose correctly: \n x, y, z\n";

std::cin >> dif;

}

if (dif == "x")

{

result = p1.defX();

Result(result);

}

if (dif == "y")

{

result = p1.defY();

Result(result);

}

if (dif == "z")

{

result = p1.defZ();

Result(result);

}

}

void Arithmetic() //+, -, \*

{

std::string func, s1, s2;

std::cout << "To do arithmetic operations with polynoms they must be in one table!" << std::endl;

std::cout << "Please, enter first polynom with which one you want to work" << std::endl;

std::cin >> s1;

TPolinom p1(s1);

bool flag;

//проверка, есть ли первый полином в одной из таблиц

while (scan\_table.Find(p1.getFormula()) == nullptr && sorted\_table.Find(p1.getFormula()) == nullptr && hash\_table.Find(p1.getFormula()) == nullptr)

{

std::cout << "Sorry. All tables doesn't have this polynom." << std::endl;

Print();

std::cout << "Please, enter exist polynom " << std::endl;

std::cin >> s1;

p1 = TPolinom(s1);

}

std::cout << "Please, enter second polynom with which one you want to work" << std::endl;

std::cin >> s2;

TPolinom p2(s2);

TPolinom result;

while (!inOneTable(p1, p2))

{

std::cout << "Polynoms in different tables or second polynom doesn't exist in tables, please, enter exist second polynom" << std::endl;

std::cin >> s2;

TPolinom p2(s2);

}

std::cout << "Choose what ou want to do with polynoms. ADD, SUB to substruct, MULT to multiply" << std::endl;

std::cin >> func;

while (func != "ADD" && func != "SUB" && func != "MULT")

{

std::cout << " Please, choose correctly: \n ADD, SUB to substruct, MULT to multiply\n";

std::cin >> func;

}

if (func == "ADD")

{

result = p1 + p2;

Result(result);

}

if (func == "SUB")

{

result = p1 - p2;

Result(result);

}

if (func == "MULT")

{

result = p1 \* p2;

Result(result);

}

}

void user()

{

std::string func;

while (func != "Exit")

{

std::cout << " What you want to do? \n Print \n Insert\n Remove\n Arithmetic - to do arithmetic operations with polynoms \n Dif - to take the derivative of polynom \n Exit" << std::endl;

std::cin >> func;

while (func != "Print" && func != "Insert" && func != "Remove" && func != "Arithmetic" && func != "Dif" && func != "Exit")

{

std::cout << "Error. Plese, enter correct action" << std::endl;

std::cin >> func;

}

if (func == "Print")

Print();

if (func == "Insert")

Insert();

if (func == "Remove")

Remove();

if (func == "Arithmetic")

Arithmetic();

if (func == "Dif")

Dif();

}

}

int main()

{

TPolinom p("3\*x^5");

TPolinom t1("1-x");

TPolinom t2("x-1");

TPolinom t3("2\*x^2");

hash\_table.Insert(t1.getFormula(), t1);

hash\_table.Insert(t2.getFormula(), t2);

hash\_table.Insert(t3.getFormula(), t3);

scan\_table.Insert(t1.getFormula(), t1);

scan\_table.Insert(t2.getFormula(), t2);

scan\_table.Insert(t3.getFormula(), t3);

sorted\_table.Insert(t1.getFormula(), t1);

sorted\_table.Insert(t2.getFormula(), t2);

sorted\_table.Insert(t3.getFormula(), t3);

sorted\_table.Insert(p.getFormula(), p);

user();

return 0;

}