Федеральное государственное автономное образовательное учреждение

высшего образования

«Национальный исследовательский Нижегородский государственный университет

им. Н.И. Лобачевского» (ННГУ)

Институт Информационных технологий, математики и механики

Отчёт по лабораторной работе

**«Таблицы»**

**Выполнил**:

студент института ИТММ   
гр. 381603-1

Болотов Д.И.

**Проверил**:

Кандидат тех. наук, старший преподаватель каф. МОСТ   
института ИТММ

Кустикова В.Б.

Нижний Новгород

2018 г.

**Содержание**

[Введение 4](#_Toc515193385)

[1 Постановка задачи 5](#_Toc515193386)

[2 Руководство пользователя 6](#_Toc515193387)

[3 Руководство программиста 7](#_Toc515193388)

[3.1 Структура программы 7](#_Toc515193389)

[3.2 Структуры данных 7](#_Toc515193390)

[3.3 Алгоритмы 8](#_Toc515193391)

[3.4 Программная реализация 13](#_Toc515193392)

[3.4.1 Структура TMonom 13](#_Toc515193393)

[3.4.2 Класс TLink 14](#_Toc515193394)

[3.4.3 Класс TRingList 14](#_Toc515193395)

[3.4.4 Класс TPolinom 15](#_Toc515193396)

[3.4.5 Структура unit 15](#_Toc515193397)

[3.4.6 Класс table 16](#_Toc515193398)

[3.4.7 Класс ScanTable 16](#_Toc515193399)

[3.4.8 Класс SortedTable 17](#_Toc515193400)

[3.4.9 Класс HashTable 17](#_Toc515193401)

[Заключение 18](#_Toc515193402)

[Литература 19](#_Toc515193403)

[Приложение 20](#_Toc515193404)

[Приложение А. Программная реализация монома 20](#_Toc515193405)

[Приложение Б. Программная реализация звена 24](#_Toc515193406)

[Приложение В. Программная реализация списка 25](#_Toc515193407)

[Приложение Г. Программная реализация полинома 29](#_Toc515193408)

[Приложение Д. Программная реализация таблицы 35](#_Toc515193409)

[Приложение Д. Программная реализация просматриваемой таблицы 37](#_Toc515193410)

[Приложение Е. Программная реализация сортированной таблицы 39](#_Toc515193411)

[Приложение Ж. Программная реализация хэш-таблицы 41](#_Toc515193412)

[Приложение З. Реализация основного приложения 44](#_Toc515193413)

# Введение

Данная лабораторная посвящена теме “Таблицы”. Целью работы является разработка консольного приложения, выполняющего следующие действия: вставка, удаление и поиск в таблице. Каждая ячейка таблицы хранит данные и ключ. Ключом является строка, данными – полином. Для реализации полинома использовался циклический односвязный список.

Данная программа позволяет выполнять быструю работу с таблицами. Тем самым приложение экономит время и средства, которые человек затратил бы на эту работу. Программа может использоваться и в других более сложных проектах.

# Постановка задачи

Необходимо разработать программу, выполняющую операции в таблицах. Таблицы могут быть трёх видов: просматриваемые, сортированные (по ключу) и хэш-таблицы. Операции: вставка, удаление и поиск. Ключом является строка. В данных хранится полином от введенной строки. Полином может состоять из трех переменных (x, y, z). Степень переменных полинома принимает значения от 0 до 9. Коэффициенты могут быть только вещественными числами. После выполнения операций, программа должна вывести пользователю результат.

# Руководство пользователя

При запуске программы, пользователю будет предложено выбрать размер таблицы.

Далее пользователю будет предложено выбрать операцию, которую он хочет произвести:

1. Вставка
2. Удаление
3. Поиск

Выбрав одну из операций, пользователю необходимо будет ввести строку.

Далее программа выводит результат выполнения операции. Пользователь может увидеть одновременно все три таблицы. Если попытались удалить или найти элемент, которого нет в таблице, будет выведена ошибка. Пользователю предлагается повторить работу в программе. При отказе, программа прекращает свою работу.

# Руководство программиста

## Структура программы

Программа состоит из четырех проектов.

*gtest*

Содержит в себе gtest.h и gtest-all.cpp. Файлы содержат google тесты, которые необходимы для проверки корректности реализованных классов.

*table\_lib*

Содержит в себе:

1. TLink.h – реализация шаблонного класса звена.
2. TMonom.h – объявление полей и методов, которые отвечают за степень и коэффициент звена
3. TPolinom.h – объявление полей и методов класса полином
4. TRingList.h – реализация шаблонного класса список
5. Table.h – реализация абстрактного класса таблиц.
6. ScanTable.h – реализация класса просматриваемые таблицы.
7. SortedTable.h – реализация класса сортированные таблицы.
8. HashTable.h – реализация класса хэш-таблицы.
9. TMonom.cpp – реализация полей и методов класса Monom
10. TPolinom.cpp – реализация полей и методов класса полином

*table\_test*

Содержит в себе test\_main.cpp, test\_polinom.cpp, test\_ringlist.cpp, test\_table.cpp. Реализация тестов для проверки корректности реализованных классов.

*sample*

Содержит main.cpp – реализация меню консольного приложения.

## Структуры данных

В программе реализована структура TMonom (моном), класс TLink (звено), класс, TRingList (список), класс TPolinom (полином), Table (абстрактная таблица), ScanTable (просматриваемая таблица), SortedTable (сортировання таблица), HashTable (хэш-таблица). Полином состоит из зацикленного списка с головой. Список состоит из звеньев. Полями списка является звено – голова и звено – текущий элемент в списке. ­Звенья хранят в себе указатель на звено и моном. Моном состоит из числа двойной точности – коэффициент и беззнакового целого числа – степень монома. Просматриваемая, сортированная и хэш–таблицы являются наследниками абстрактного класса Table, они наследуют все поля и методы абстрактного класса. У хэш-таблиц дополнительно имеется указатель на массив, который отвечает за состояние записи в таблице (запись занята, удалена или пуста). Класс таблицы состоят из указателя на массив указателей, максимального размера таблицы, текущего размера таблицы и индекса текущей записи. В качестве массива указателей используется структура unit. Структура состоит из указателя на полином и строки.

## Алгоритмы

Конструктор полинома по строке:

Определяется длина строки, и запускается цикл. Цикл будет выполняться, пока не кончится строка. Внутри цикла реализован следующий алгоритм:

1. Если мы встретили на i–ой позиции число, или переменную, или ‘-‘, то заходим внутрь инструкции if, иначе увеличивается переменная цикла i (снова происходит проверка i–ой позиции).
2. В инструкции if создается новая переменная j.
3. Запускается цикл while. Если на i + 1 + j позиции находится ‘+’, ‘-‘ или конец строки, то увеличиваем переменную j. Иначе, выходим из цикла.
4. Следующим шагом в новую строку мы копируем исходную строку с i–ой позиции j + 1 символ. Таким образом, мы выделяем каждое слагаемое в строке.
5. Передаем подстроку в конструктор монома.
6. В список вставляем полученный моном.
7. Переменную цикла увеличиваем на i + j.
8. После завершения цикла вызывается функция приведения подобных слагаемых.

Сложение полиномов:

Создается новый полином. В списках у полиномов, которые являются первым и вторым слагаемым соответственно, указатель на текущий элемент переводиться на следующее за головой звено.

1. Запускается цикл while, условием которого является то, что оба списка не кончились. Если один из списков закончился, переходим к пункту 3.
2. Внутри цикла while сравниваем текущие звенья в списках. В новый список нового полинома вставляется моном, имеющее большее значение степени. Указатель передвигается на следующее звено. При равенстве степеней вставляется сумма двух моном. Указатель передвигается у обоих списков.
3. Происходит проверка. Если закончился первый список, то в новый список вставляется все оставшиеся звенья из второго списка. Аналогично в противоположном случае.
4. Возвращаем полученный новый полином.

Умножение полиномов:

Создается новый полином (1). У списка первого множителя указатель текущего элемента устанавливается на следующее звено после головы:

1. Запускается цикл while, условием которого является то, что список первого множителя не кончился. Если условие не выполняется, переходим к пункту 10.
2. Внутри цикла while у списка второго множителя указатель на текущий элемент устанавливается на следующее звено после головы. Запускается еще один цикл while, условием которого является то, что список второго множителя не кончился. Если условие не выполняется, переходим к пункту 9.
3. Создается моном
4. Коэффициентом нового монома является произведение коэффициентов двух мономов текущих звеньев.
5. Если степень не равно нулю, то создается новая переменная, отвечающая за ступень нового монома. Иначе, переходим к пункту 8. Она равно сумме степеней двух мономов текущих звеньев. Если эта переменная лежит в пределе от 0 до 999, то в степень монома сохраняем значение этой переменной. Иначе, выдается ошибка.
6. Создается новый полином. В него вставляется созданный моном.
7. К полиному (1) прибавляем только что созданный полином, и результат суммы записывается в (1).
8. Передвигаем указатель у списка второго множителя.
9. Передвигаем указатель у списка первого множителя.
10. Возвращаем значение (1).

Умножение полинома на константу:

Создается новый полином. Указатель у списка первого множителя устанавливается на следующий элемент после головы.

1. Запускается цикл while, условием которого является то, что список первого множителя не кончился. Иначе, переходим к пункту 4.
2. У монома текущего звена коэффициент умножаем на константу.
3. Вставляем в новый полином моном текущего звена списка первого множителя. Передвигаем указатель.
4. Возвращаем полученный полином.

Умножение константы на полином реализовано аналогично.

Вычитание полиномов:

Создается новый полином. Полином, являющийся вычитаемым, умножается на константу (-1). В созданный полином сохраняем сумма двух данных полиномов. Возвращаем полученный полином.

Функция вставка в список:

1. Если список пуст, создаем новое звено, которое указывает на голову списка. Назначаем текущим элементом новое звено. Голова списка указывает на текущий элемент.
2. Если список пуст. Текущий элемент устанавливаем на голову.
3. Создаем цикл while, условием которого является то, что если данные звена, на которое указывает текущий элемент, больше, чем данные, которые нужно вставить, то передвигаем указатель на следующее звено после текущего. Как только выйдем из цикла, создается новое звено, которое указывает на следующий элемент после текущего. Назначаем новое звено следующим элементом после текущего.

Конструктор монома по строке:

Создается 3 переменные a, b, c, отвечающие за степень переменной x, y, z соответственно. Создается переменная cf равная единице. Создается переменная abc. Определяется длина строки.

1. Запускается цикл for, условием которого является то, что строка не кончилась.
2. Если на i–ой позиции оказалось число. В коэффициент сохраняем число с помощью функции atof. Создаем переменную j. Создается цикл while. Пока на i + 1 + j позиции находится число или не конец строки, то увеличиваем переменную j. Передвигаем индекс i на i + j. Переходим к следующему пункту.
3. Если на i–ой позиции оказался ‘-‘. Если на следующей позиции оказалась переменная, то коэффициент назначаем равным (-1). Если на следующей позиции оказалось число, то коэффициент назначаем равным этому числу с помощью функции atof. Создается переменная j. Создается цикл while. Пока на i + 1 + j позиции находится число или не конец строки, то увеличиваем переменную j. Передвигаем индекс i на i + j. Переходим к следующему пункту.
4. Если на i–ой позиции оказалась переменная ‘x’. Если на следующей позиции находится знак степени ‘^’. Переходим к следующему пункту. Иначе переходим к пункту 6.
5. В переменную a сохраняем число, которое начинается с позиции i + 2. Создается переменная j. Пока на i + 2 + j позиции не встретился знак ‘\*’ или не конец строки, увеличиваем переменную j. В переменную i сохраняем значение i + j + 2. Переходим к пункту 8.
6. В переменную a сохраняем 1.
7. Аналогично выполняется для переменной ‘y’ и ‘z’.
8. В обобщенную степень сохраняем сумму произведений.

Перепаковка просматриваемых таблиц:

Создается новый указатель на массив указателей, у которого максимальный размер в два раза больше, чем был у таблицы, вызвавшей функция перепаковки. Копируются все элемента массива до текущего размера таблицы. Остальные элементы инициализируются NULL. Приравниваем указатели на массивы указателей.

Вставка в сортированные таблицы:

Проверяем, если таблицы заполнена, то вызываем функцию перепакови. Далее происходит поиск строки, которую пользователь хочет вставить. Если результатом поиска является пустой указатель, то в элемент массива, индексом которого является текущий размер массива, вставляем строку и полином по строке. Увеличиваем текущий размер таблицы и текущий индекс. Если такой ключ уже присутствует в таблице, то пользователь получит сообщение об ошибке.

Удаление в просматриваемой таблице:

Выполняется поиск строки, которую пользователь хочет удалить. Если результатом поиска является пустой указатель, пользователь получает сообщение об ошибке. Иначе, поиск выдает нам индекс записи, которую хотим удалить. На это место сохраняем последнюю запись в таблице. Уменьшаем текущий размер таблицы.

Поиск в просматриваемой таблице:

Запускается цикл while, условием которого является то, что мы не вышли за текущий размер таблицы и ключ элемента массива не равна введенной строке. После выхода из цикла происходит проверка. Если наша переменная поиска меньше, чем текущий размер таблицы, то к индексу приравниваем данную переменную. Возвращаем найденную запись. Если мы вышли за текущий размер таблицы, возвращается нулевой указатель.

Перепаковка в сортированных таблицах:

Аналогично просматриваемым таблицам.

Вставка в сортированную таблицу:

Проверяем, если таблицы заполнена, то вызываем функцию перепакови. Далее происходит поиск строки, которую пользователь хочет вставить. Если результатом поиска является пустой указатель, то происходит поиск строки м помощью цикла for. Проверка происходит с конца списка. Пока не найдется запись в массиве, у которой ключ будет больше введенной строки, мы передвигаем элементы. Как только выйдем из цикла, создаем новый элемент от введенной строки и полинома по строке. Увеличиваем текущий размер таблицы. Если поиск возвращает не нулевой указатель, выводится сообщение об ошибке.

Удаление в сортированной таблице:

Алгоритм используемый при удалении аналогичен вставке в сортированную таблицу.

Поиск в сортированной таблице:

Реализован бинарный поиск. Создается целочисленная переменная поиска равная нулю, переменная левой границы, правой границы и центральной позиции поиска. Центральная позиция равна сумме левой и правой границ, деленных пополам. Если ключ от элемента, расположенного на центральной позиции, больше, то двигаем правую границу. Аналогично двигаем левую границу, если строка меньше, чем ключ от элемента массива расположенного на центральной позиции. Если ключ равен введенной строке, увеличиваем переменную целочисленную переменную поиска, выходим из цикла и возвращаем указатель на элемент массива, индекс равен переменной центральной позиции. Иначе, возвращаем нулевой указатель.

Функция хэширования:

Создается целочисленная переменная. Создается цикл for. В переменную записывается сумма самой переменной и код текущего символа в строке. Результатом возвращается остаток от деления целочисленной переменой на максимальный размер таблицы.

Перепаковка в хэш­-таблице.

Создается новый указатель на массив указателей, у которого максимальный размер в два раза больше, чем был у таблицы, вызвавшей функция перепаковки. Весь массив инициализируется NULL. Создается новый указатель на массив состояния записей. Все они на данный момент является свободными. Далее запускается цикл for с нуля, условием которого является то, что мы не вышли за текущий размер таблицы. Если у текущего элемента запись занята или удалена, то мы вычисляемый хэш-функцию для текущего элемента и вставляем его в новый массив, при этом изменяем новый массив состояния записей.

Вставка в хэш-таблицы:

Если хэш-таблица заполнена больше, чем на заданное число, то выполняется перепаковка. Происходит поиск строки, которую хотим вставить. Если результатом поиска является не нулевой указатель, то выдаем сообщение об ошибке. Иначе высчитываем по хэш-функции позицию, в которую необходимо вставить строку. Если позиция свободная, создаем новую запись на этом от введенной строки и полинома по строке. Увеличиваем текущий размер таблицы, в таблице состояний элементов на данной позиции устанавливаем значение, отвечающее за то, что запись занята. Если позиция занята или удалена, выполняем поиск свободной. Найдя свободную позицию, создаем новую запись.

Удаление в хэш-таблице:

Если поиск возвращает нулевой указатель, выдаем сообщение об ошибке. Иначе, устанавливаем, что запись в таблице, которую удаляем, равна NULL. Изменяем строку состояния записей, уменьшаем текущий размер таблицы.

Поиск в хэш-таблице:

Высчитываем позицию по функции хэширвоания от введенной строки. Если в массиве состояния записей эта позиция равна нулю, возвращаем нулевой указатель. Иначе, делаем следующую проверку. Если запись не удалена и введенный ключ не равен строке, хранящейся в данной записи, то заходим внутрь цикла. Запоминаем первый вычисленный индекс по хэш-функции (он необходим для проверки, что не зациклился поиск). Высчитываем новую позицию. Запускается цикл while, условие которого является то, что мы не наткнулись на свободную запись, и не зациклился наш поиск. Внутри цикла происходит проверка. Если запись не удалена и наша введенная строка равна строке, хранящейся в данной записи, то мы устанавливаем индекс, равный этой записи, и возвращаем эту запись. Иначе, вычитываем новый индекс. Если мы вышли из цикла while, дальше происходит проверка. Если запись не занята или зациклился поиск, то возвращаем нулевой указатель.

## Программная реализация

### Структура TMonom

Поля:

double cf – коэффициент монома

unsigned int abc – степень монома

Методы:

TMonom() – конструктор по умолчанию

TMonom(const string s) – конструктор по строке

TMonom(double num, unsigned int num2) – конструктор по параметрам

~TMonom() – деструктор

TMonom operator + (const TMonom& monom) – сумма мономов

TMonom operator – (const TMonom& monom) – разность мономов

TMonom operator = (const TMonom& monom) – оператор копирования

bool operator == (const TMonom& monom) const – сравнение мономов

bool operator != (const TMonom& monom) const – сравнение мономов

bool operator > (const TMonom& monom) const – сравнение мономов

bool operator < (const TMonom& monom) const – сравнение мономов

### Класс TLink

Поля:

T data – данные

TLink\* pNext – указатель на следующее звено

Методы:

TLink() – конструктор по умолчанию

TLink(const TLink& unit) – конструктор копирования

TLink(T d, TLink\* unit = NULL) – конструктор по параметрам

~TLink() – деструктор

TLink& operator = (const TLink& unit) – оператор копирования

bool operator == (const TLink& unit) const – оператор сравнения

bool operator != (const TLink& unit) const – оператор сравнения

### Класс TRingList

Поля:

TLink <T> \*head – голова списка

TLink <T> \*current – текущий элемент списка

Методы:

TRingList() – конструктор по умолчанию

TRingList(const TRingList <T> & TRingList2) – конструктор копирования

~TRingList() – деструктор

TRingList<T>& operator = (const TRingList<T>& TRingList2) – оператор копирования

void Clean() – очистка списка

void Insert(const T& data) – упорядоченная вставка в список

void Reset() – текущий элемент указывает следующее звено после головы

void Next() – двигаем указатель текущего элемента на следующий

bool IsEnded() const – проверка на конце списка

TLink<T>\* GetLink() – вернуть текущее звено

bool operator == (const TRingList<T>& TRingList2) – оператор сравнения

### Класс TPolinom

Поля:

TRingList<TMonom> monoms – список

Методы:

TPolinom() – конструктор по умолчанию

TPolinom(const string s) – конструктор по строке

TPolinom(const TPolinom &poli2) – конструткор копирования

~TPolinom() – деструктор

TPolinom operator + (const TPolinom& poli2) – оператор суммы

TPolinom operator - (const TPolinom& poli2) – оператор разности

TPolinom operator \* (const TPolinom& poli2) – оператор произведения

TPolinom operator \* (double cf2) – оператор произведения на константу

friend TPolinom operator \* (double cf2, const TPolinom& poli2) – оператор произведения константы и полинома

TPolinom& operator = (const Tpolinom& tmp) – оператор копирвоания

bool operator == (const TPolinom& poli2) – оператор сравнения

void PrivPod() – функция приведения подобных звеньев

friend ostream& operator <<(ostream& ostr, const TPolinom& poli2) – перегрузка вывода полинома

### Структура unit

Поля:

TPolinom – данные

String key – ключ

Методы:

Unit() – конструктор по умолчанию

Unit(const string s, const TPolinom \*poli2) – конструктор копирования

Unit& operator = (const unit&tmp) – оператор копирования

### Класс table

Поля:

unit \*\*str – указатель на массив указателей

int max\_size – максимальный размер

int size – текущий размер

int index – индекс текущий записи

virtual void Repacking() = 0 – перепаковка

Методы:

table(int nsize = 10) – конструктор по умолчанию

virtual ~table()– деструктор

virtual void Insert(const string k, const TPolinom \*poli2) = 0 – вставка

virtual unit\* Search(const string k) = 0 – поиск

virtual void Delete(const string k) = 0 – удаление

virtual void Reset()– индекс передвигается в начало

virtual bool IsFull() const – проверка на полноту

virtual void Next()– передвинуть индекс

virtual unit\* GetCurrent() const – вернуть текущую запись

virtual int GetSize() const – вернуть текущий размер

virtual int GetMaxSize() const – вернуть максимальный размер

virtual int GetIndex() const – вернуть текущий индекс

virtual void PrintTable()– печать таблицы

### Класс ScanTable

Поля:

void Repacking()override – перепаковка

Методы:

ScanTable(int nsize = 10): table(nsize) – конструктор по умолчанию

~table()– деструктор

ScanTable(const ScanTable &T) – конструктор копирования

void Insert(const string k, const TPolinom \*poli2)override – вставка

unit\* Search(const string k)override – поиск

void Delete(const string k)override – удаление

### Класс SortedTable

Поля:

virtual oid Repacking()override – перепаковка

Методы:

SortedTable(int nsize = 10): table(nsize) – конструктор по умолчанию

~table()– деструктор

SortedTable(const ScanTable &T) – конструктор преобразования

void Insert(const string k, const TPolinom \*poli2)override – вставка

unit\* Search(const string k)override – поиск

void Delete(const string k)override – удаление

### Класс HashTable

Поля:

virtual void Repacking()override – перепаковка

int hash(string key) const – хэш-функция

int \*State – массив состояния записей

Методы:

SortedTable(int nsize = 10): table(nsize) – конструктор по умолчанию

SortedTable(const ScanTable &T) – конструктор преобразования

void Insert(const string k, const TPolinom \*poli2)override – вставка

unit\* Search(const string k)override – поиск

void Delete(const string k)override – удаление

# Заключение

В данной программе было реализовано выполнения операция в таблице. Можно выполнять вставку, удаление и поиск записей в таблице. Таблицы трёх видов: просматриваемы, сортированные (по ключу) и хэш-таблицы.

# Литература

Рабочие материалы к учебному курсу «Методы программирования». *Гергель В.П*. 2002 г.

# Приложение

## Приложение А. Программная реализация монома

TMonom.h

#pragma once

#include <iostream>

#include <stdio.h>

#include <cstring>

#include <string>

#include <cstdlib>

using namespace std;

struct TMonom

{

double cf;

unsigned int abc;

TMonom();

TMonom(const string s);

TMonom(double num, unsigned int num2);

~TMonom() {}

TMonom operator + (const TMonom& monom);

TMonom operator - (const TMonom& monom);

TMonom operator = (const TMonom& monom);

bool operator == (const TMonom& monom) const;

bool operator != (const TMonom& monom) const;

bool operator > (const TMonom& monom) const;

bool operator < (const TMonom& monom) const;

};

TMonom.cpp

#include "TMonom.h"

TMonom::TMonom()

{

cf = NULL;

abc = NULL;

}

TMonom::TMonom(const string s)

{

int a = 0;

int b = 0;

int c = 0;

cf = 1;

abc = 0;

int k = s.length();

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

{

if (isdigit(s[i]))

{

this->cf = atof(&s[i]);

int j = 0;

while ((s[i + 1 + j] != '\0') && (isdigit(s[i + 1 + j])))

{

j++;

}

i = i + j;

}

if (s[i] == '-')

{

if (isalpha(s[i + 1]))

{

this->cf = -1;

}

if (isdigit(s[i + 1]))

{

cf = atof(&s[i]);

this->cf = cf;

}

int j = 0;

while ((s[i + 1 + j] != '\0') && (isdigit(s[i + 1 + j])))

{

j++;

}

i = i + j;

}

if (s[i] == 'x')

{

if (s[i + 1] == '^')

{

a = atof(&s[i + 2]);

int j = 0;

while ((s[i + 2 + j] != '\*') && (s[i + 2 + j] != '\0') && (s[i + 2 + j] == '+') && (s[i + 2 + j] == '-'))

{

j++;

}

i = i + j + 2;

}

else

{

a = 1;

}

}

if (s[i] == 'y')

{

if (s[i + 1] == '^')

{

b = atof(&s[i + 2]);

int j = 0;

while ((s[i + 2 + j] != '\*') && (s[i + 2 + j] != '\0') && (s[i + 2 + j] == '+') && (s[i + 2 + j] == '-'))

{

j++;

}

i = i + j + 2;

}

else

{

b = 1;

}

}

if (s[i] == 'z')

{

if (s[i + 1] == '^')

{

c = atof(&s[i + 2]);

int j = 0;

while ((s[i + 2 + j] != '\*') && (s[i + 2 + j] != '\0') && (s[i + 2 + j] == '+') && (s[i + 2 + j] == '-'))

{

j++;

}

i = i + j + 2;

}

else

{

c = 1;

}

}

}

this->abc = a \* 100 + b \* 10 + c;

}

TMonom::TMonom(double num, unsigned int num2)

{

cf = num;

abc = num2;

}

TMonom TMonom::operator + (const TMonom& monom)

{

cf = cf + monom.cf;

return \*this;

}

TMonom TMonom::operator - (const TMonom& monom)

{

cf = cf - monom.cf;

return \*this;

}

TMonom TMonom::operator = (const TMonom& monom)

{

cf = monom.cf;

abc = monom.abc;

return \*this;

}

bool TMonom::operator == (const TMonom& monom) const

{

int k = 0;

if (abc != monom.abc)

k++;

if (k == 0)

return true;

else

return false;

}

bool TMonom::operator != (const TMonom& monom) const

{

int k = 0;

if (abc == monom.abc)

k++;

if (k == 0)

return true;

else

return false;

}

bool TMonom::operator > (const TMonom& monom) const

{

int k = 0;

if (abc > monom.abc)

k++;

if (k != 0)

return true;

else

return false;

}

bool TMonom::operator < (const TMonom& monom) const

{

int k = 0;

if (abc < monom.abc)

k++;

if (k != 0)

return true;

else

return false;

}

## Приложение Б. Программная реализация звена

TLink.h

#pragma once

#include "TMonom.h"

template <typename T>

class TLink

{

public:

T data;

TLink\* pNext;

TLink();

TLink(const TLink& unit);

TLink(T d, TLink\* unit= NULL);

~TLink() {}

TLink& operator = (const TLink& unit);

bool operator == (const TLink& unit) const;

bool operator != (const TLink& unit) const;

};

template <typename T>

TLink<T>::TLink()

{

pNext = NULL;

}

template <typename T>

TLink<T>::TLink(const TLink& unit)

{

data = unit.data;

pNext = unit.pNext;

}

template <typename T>

TLink<T>::TLink(T d, TLink\* unit)

{

data = d;

pNext = unit;

}

template <typename T>

TLink<T>& TLink<T>::operator=(const TLink& unit)

{

data = unit.data;

pNext = unit.pNext;

return \*this;

}

template <typename T>

bool TLink<T>::operator== (const TLink& unit) const

{

int k = 0;

if (data != unit.data)

k++;

if (pNext != unit.pNext)

k++;

if (k == 0)

return true;

else

return false;

}

template <typename T>

bool TLink<T>::operator!= (const TLink& unit) const

{

int k = 0;

if (data == unit.data)

k++;

if (pNext == unit.pNext)

k++;

if (k != 0)

return true;

else

return false;

}

## Приложение В. Программная реализация списка

TRingList.h

#pragma once

#include "TLink.h"

template <typename T>

class TRingList

{

private:

TLink <T> \*head;

TLink <T> \*current;

public:

TRingList();

TRingList(const TRingList <T> & TRingList2);

~TRingList();

TRingList<T>& operator = (const TRingList<T>& TRingList2);

void Clean();

void Insert(const T& data);

void Reset();

void Next();

bool IsEnded() const ;

TLink<T>\* GetLink();

bool operator == (const TRingList<T>& TRingList2) const;

};

template <typename T>

TRingList<T>::TRingList()

{

head = new TLink<T>();

head->pNext = head;

current = head;

}

template <typename T>

void TRingList<T>::Clean()

{

if (head->pNext != head)

{

TLink<T> \*tmp = head;

TLink<T> \*tmp2 = head->pNext;

while (tmp2 != head)

{

tmp = tmp2->pNext;

delete tmp2;

tmp2 = tmp;

}

}

}

template <typename T>

TRingList<T>::~TRingList()

{

this->Clean();

delete head;

}

template <typename T>

TRingList<T>& TRingList<T>::operator = (const TRingList<T>& TRingList2)

{

if (this != &TRingList2)

{

this->Clean();

if (TRingList2.head->pNext == TRingList2.head)

{

head = new TLink<T>();

head->pNext = head;

current = head;

}

else

{

head = new TLink<T>(TRingList2.head->data, TRingList2.head->pNext);

current = head;

TLink<T> \*tmp = TRingList2.head->pNext;

while (tmp != TRingList2.head)

{

current->pNext = new TLink<T>(tmp->data);

current = current->pNext;

tmp = tmp->pNext;

}

current->pNext = head;

}

}

return \*this;

}

template <typename T>

TRingList<T>::TRingList(const TRingList<T> & TRingList2)

{

if (TRingList2.head->pNext == TRingList2.head)

{

head = new TLink<T>();

head->pNext = head;

current = head;

}

else

{

head = new TLink<T>(TRingList2.head->data, TRingList2.head->pNext);

current = head;

TLink<T> \*tmp = TRingList2.head->pNext;

while (tmp != TRingList2.head)

{

current->pNext = new TLink<T>(tmp->data);

current = current->pNext;

tmp = tmp->pNext;

}

current->pNext = head;

}

}

template <typename T>

void TRingList<T>::Insert(const T& data)

{

if (head->pNext == head)

{

current = new TLink<T>(data, head);

head->pNext = current;

}

else

{

current = head;

while (current->pNext->data > data)

{

this->Next();

}

current->pNext = new TLink<T>(data, current->pNext);

}

}

template <typename T>

void TRingList<T>::Reset()

{

current = head->pNext;

}

template <typename T>

void TRingList<T>::Next()

{

current = current->pNext;

}

template <typename T>

bool TRingList<T>::IsEnded() const

{

if (current == head)

{

return true;

}

else

{

return false;

}

}

template <typename T>

TLink<T>\* TRingList<T>::GetLink()

{

return current;

}

template <typename T>

bool TRingList<T>::operator == (const TRingList<T>& TRingList2) const

{

TRingList<T> tmp(\*this);

TRingList<T> tmp2(TRingList2);

tmp.Reset();

tmp2.Reset();

int k = 0;

while ((tmp.IsEnded() == false) && (tmp2.IsEnded() == false))

{

if (tmp.GetLink()->data != tmp2.GetLink()->data)

{

k++;

}

tmp.Next();

tmp2.Next();

}

if ((tmp.IsEnded() == false) && (tmp2.IsEnded() == true))

{

k++;

tmp.Next();

}

if ((tmp.IsEnded() == true) && (tmp2.IsEnded() == false))

{

k++;

tmp2.Next();

}

if (k == 0)

{

return true;

}

else

{

return false;

}

}

## Приложение Г. Программная реализация полинома

TPolinom.h

#pragma once

#include "TRingList.h"

class TPolinom

{

private:

TRingList<TMonom> monoms;

public:

TPolinom() {}; // конструктор

TPolinom(const string s);

TPolinom(const TPolinom &poli2);

~TPolinom() {};

TPolinom operator + (const TPolinom& poli2);

TPolinom operator - (const TPolinom& poli2);

TPolinom operator \* (const TPolinom& poli2);

TPolinom operator \* (double cf2);

friend TPolinom operator \*(double cf2, const TPolinom& poli2);

TPolinom& operator = (const TPolinom& tmp);

bool operator == (const TPolinom& poli2) const;

void PrivPod();

friend ostream& operator <<(ostream& ostr, const TPolinom& poli2);

};

TPolinom.cpp

#include "TPolinom.h"

TPolinom& TPolinom::operator = (const TPolinom& tmp)

{

monoms = tmp.monoms;

return \*this;

}

TPolinom::TPolinom(const string s)

{

int k = s.length();

string c;

if (k != 0)

{

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

{

if ((isdigit(s[i])) || (isalpha(s[i])) || (s[i] == '-'))

{

int j = 0;

while ((s[i + 1 + j] != '+') && (s[i + 1 + j] != '-') && ((s[i + 1 + j] != '\0')))

{

j++;

}

c = s.substr(i, j + 1);

TMonom tmp(c);

monoms.Insert(tmp);

i = i + j;

}

}

}

this->PrivPod();

}

TPolinom::TPolinom(const TPolinom &poli2)

{

monoms = poli2.monoms;

}

TPolinom TPolinom::operator + (const TPolinom& poli2)

{

TPolinom C;

TPolinom tmp(poli2);

monoms.Reset();

tmp.monoms.Reset();

while ((monoms.IsEnded() == false) && (tmp.monoms.IsEnded() == false))

{

if (monoms.GetLink()->data > tmp.monoms.GetLink()->data)

{

C.monoms.Insert(monoms.GetLink()->data);

monoms.Next();

}

else if (monoms.GetLink()->data < tmp.monoms.GetLink()->data)

{

C.monoms.Insert(tmp.monoms.GetLink()->data);

tmp.monoms.Next();

}

else if (monoms.GetLink()->data == tmp.monoms.GetLink()->data)

{

if ((monoms.GetLink()->data.cf + tmp.monoms.GetLink()->data.cf) != 0)

{

C.monoms.Insert(monoms.GetLink()->data + tmp.monoms.GetLink()->data);

}

monoms.Next();

tmp.monoms.Next();

}

}

while (monoms.IsEnded() == false)

{

C.monoms.Insert(monoms.GetLink()->data);

monoms.Next();

}

while (tmp.monoms.IsEnded() == false)

{

C.monoms.Insert(tmp.monoms.GetLink()->data);

tmp.monoms.Next();

}

return C;

}

TPolinom TPolinom::operator - (const TPolinom& poli2)

{

TPolinom C;

TPolinom tmp(poli2);

tmp = tmp \* (-1);

C = \*this + tmp;

return C;

}

TPolinom TPolinom::operator \* (const TPolinom& poli2)

{

TPolinom C;

TPolinom tmp(poli2);

monoms.Reset();

while (monoms.IsEnded() == false)

{

tmp.monoms.Reset();

while (tmp.monoms.IsEnded() == false)

{

TMonom monom;

monom.cf = tmp.monoms.GetLink()->data.cf \* monoms.GetLink()->data.cf;

if (monom.cf != 0)

{

int ABC = tmp.monoms.GetLink()->data.abc + monoms.GetLink()->data.abc;

if ((ABC / 100 < 10) && (ABC / 10 % 10 < 10) && (ABC % 10 < 10))

monom.abc = ABC;

else

throw "Error";

TPolinom tmp2;

tmp2.monoms.Insert(monom);

C = C + tmp2;

}

tmp.monoms.Next();

}

monoms.Next();

}

return C;

}

TPolinom TPolinom::operator \* (double cf2)

{

TPolinom C;

monoms.Reset();

while (monoms.IsEnded() == false)

{

monoms.GetLink()->data.cf = cf2 \* monoms.GetLink()->data.cf;

C.monoms.Insert(monoms.GetLink()->data);

monoms.Next();

}

return C;

}

TPolinom operator \*(double cf2, const TPolinom& poli2)

{

TPolinom C;

TPolinom tmp(poli2);

tmp.monoms.Reset();

while (tmp.monoms.IsEnded() == false)

{

tmp.monoms.GetLink()->data.cf = cf2 \* tmp.monoms.GetLink()->data.cf;

C.monoms.Insert(tmp.monoms.GetLink()->data);

tmp.monoms.Next();

}

return C;

}

ostream& operator <<(ostream& ostr, const TPolinom& poli2)

{

TPolinom B(poli2);

B.monoms.Reset();

if (B.monoms.IsEnded() == false)

{

while (B.monoms.IsEnded() == false)

{

if (B.monoms.GetLink()->data.cf > 0)

{

if (B.monoms.GetLink()->data.abc != 0)

{

if (B.monoms.GetLink()->data.cf == 1)

{

ostr << "+";

}

else

{

ostr << "+" << B.monoms.GetLink()->data.cf;

}

}

else

{

ostr << "+" << B.monoms.GetLink()->data.cf;

}

}

if (B.monoms.GetLink()->data.cf < 0)

{

ostr << B.monoms.GetLink()->data.cf;

}

if (B.monoms.GetLink()->data.abc != 0)

{

int a = B.monoms.GetLink()->data.abc / 100;

if (a > 1)

{

ostr << "x^" << a;

}

if (a == 1)

{

ostr << "x";

}

a = B.monoms.GetLink()->data.abc / 10 % 10;

if (a > 1)

{

ostr << "y^" << a;

}

if (a == 1)

{

ostr << "y";

}

a = B.monoms.GetLink()->data.abc % 10;

if (a > 1)

{

ostr << "z^" << a;

}

if (a == 1)

{

ostr << "z";

}

}

B.monoms.Next();

}

}

else

{

ostr << "0";

}

return ostr;

}

bool TPolinom::operator == (const TPolinom& poli2) const

{

TPolinom A(\*this);

TPolinom B(poli2);

return A.monoms == B.monoms;

}

void TPolinom::PrivPod()

{

TPolinom A(\*this);

TPolinom res;

A.monoms.Reset();

while (A.monoms.IsEnded() == false)

{

res.monoms.Reset();

if (res.monoms.IsEnded() == false)

{

while ((res.monoms.IsEnded() == false) && (res.monoms.GetLink()->data.abc != A.monoms.GetLink()->data.abc))

{

res.monoms.Next();

}

if (res.monoms.IsEnded() == true)

{

res.monoms.Insert(A.monoms.GetLink()->data);

}

if (res.monoms.GetLink()->data.abc == A.monoms.GetLink()->data.abc)

{

res.monoms.GetLink()->data.cf = res.monoms.GetLink()->data.cf + A.monoms.GetLink()->data.cf;

}

}

else

{

res.monoms.Insert(A.monoms.GetLink()->data);

}

A.monoms.Next();

}

res.monoms.Reset();

TPolinom B;

while (res.monoms.IsEnded() == false)

{

// удаляем, когда cf = 0

if (res.monoms.GetLink()->data.cf != 0)

{

B.monoms.Insert(res.monoms.GetLink()->data);

}

res.monoms.Next();

}

\*this = B;

}

## Приложение Д. Программная реализация таблицы

Table.h

#pragma once

#include "TPolinom.h"

using namespace std;

struct unit

{

TPolinom \*data;

string key;

unit() { key = "", data = &TPolinom(); }

unit(const string s, const TPolinom \*poli2) { key = s; data = const\_cast<TPolinom\*>(poli2);}

unit& operator= (const unit &tmp) { data = tmp.data; key = tmp.key.substr(0, tmp.key.length()); return \*this; }

};

class table

{

protected:

unit \*\*str;

int max\_size;

int size;

int index;

virtual void Repacking() = 0;

public:

table(int nsize = 10);

virtual ~table() { delete[] str; }

virtual void Insert(const string k, const TPolinom \*poli2) = 0;

virtual unit\* Search(const string k) = 0;

virtual void Delete(const string k) = 0;

virtual void Reset();

virtual bool IsFull() const ;

virtual void Next();

virtual unit\* GetCurrent() const;

virtual int GetSize() const;

virtual int GetMaxSize() const;

virtual int GetIndex() const;

virtual void PrintTable();

};

table::table(int nsize)

{

max\_size = nsize;

size = 0;

index = -1;

str = new unit\*[nsize];

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

{

str[i] = NULL;

}

}

void table::Reset()

{

index = 0;

}

bool table::IsFull() const

{

if (max\_size == size)

{

return true;

}

else

{

return false;

}

}

void table::Next()

{

index = index + 1;

}

unit\* table::GetCurrent() const

{

return str[index];

}

int table::GetSize() const

{

return size;

}

int table::GetMaxSize() const

{

return max\_size;

}

int table::GetIndex() const

{

return index;

}

void table::PrintTable()

{

Reset();

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

{

if (str[i] != NULL)

{

cout << "Позиция :" << i << " Ключ: " << str[i]->key << endl;

}

else

{

cout << "Позиция :" << i << " Ключ: NULL" << endl;

}

this->Next();

}

}

## Приложение Д. Программная реализация просматриваемой таблицы

ScanTable.h

#pragma once

#include "Table.h"

class ScanTable : public table

{

private:

void Repacking() override;

public:

ScanTable(int nsize = 10) : table(nsize) { size = 0; index = -1; }

ScanTable(const ScanTable &T);

~ScanTable() {}

void Insert(const string k, const TPolinom \*poli2) override;

unit\* Search(const string k) override;

void Delete(const string k) override;

};

void ScanTable::Repacking()

{

max\_size = max\_size \* 2;

unit \*\*tmp = new unit\*[max\_size];

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

{

tmp[i] = str[i];

}

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

{

tmp[i] = NULL;

}

delete str;

str = tmp;

}

void ScanTable::Insert(const string k, const TPolinom \*poli2)

{

if (max\_size == size)

{

Repacking();

}

if (Search(k) != nullptr)

{

throw "В просматриваемой таблице уже присутствует данный ключ.";

}

else

{

str[size] = new unit(k, poli2);

size = size + 1;

index = index + 1;

}

}

unit\* ScanTable::Search(const string k)

{

int i = 0;

while ((i < size) && (str[i]->key != k))

{

i++;

}

if (i < size)

{

index = i;

return str[i];

}

else

{

return nullptr;

}

}

void ScanTable::Delete(const string k)

{

if (Search(k) == nullptr)

{

throw "В просматриваемых таблицах отсутствует элемент, который хотите удалить.";

}

else

{

str[index]->data = str[size - 1]->data;

str[index]->key = str[size - 1]->key;

str[size - 1] = NULL;

size = size - 1;

index = size - 1;

}

}

ScanTable::ScanTable(const ScanTable &T)

{

max\_size = T.max\_size;

size = T.size;

index = T.index;

str = T.str;

}

## Приложение Е. Программная реализация сортированной таблицы

SortedTable.h

#pragma once

#include "Table.h"

class SortedTable : public table

{

private:

void Repacking() override;

public:

SortedTable(int nsize = 10) : table(nsize) { size = 0; index = -1; }

SortedTable(const ScanTable T);

~SortedTable() {}

void Insert(const string k, const TPolinom \*poli2) override;

unit\* Search(const string k) override;

void Delete(const string k) override;

};

void SortedTable::Repacking()

{

max\_size = max\_size \* 2;

unit \*\*tmp = new unit\*[max\_size];

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

{

tmp[i] = str[i];

}

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

{

tmp[i] = NULL;

}

delete str;

str = tmp;

}

void SortedTable::Insert(const string k, const TPolinom \*poli2)

{

if (max\_size == size)

{

Repacking();

}

if (Search(k) != nullptr)

{

throw "В сортированной таблице уже присутствует данный ключ.";

}

else

{

int i;

for (i = size; (i > 0) && (str[i - 1]->key < k); i--)

{

str[i] = str[i - 1];

}

str[i] = new unit(k, poli2);

size = size + 1;

index = index + 1;

}

}

unit\* SortedTable::Search(const string k)

{

int left = 0;

int right = size - 1;

int middle = 0;

int a = 0;

while ((left <= right) && (a != 1))

{

middle = (left + right) / 2;

if (k > str[middle]->key)

{

right = middle - 1;

}

else if (k < str[middle]->key)

{

left = middle + 1;

}

else if (k == str[middle]->key)

{

a = 1;

}

}

if (a == 0)

{

return nullptr;

}

else

{

index = middle;

return str[middle];

}

}

void SortedTable::Delete(const string k)

{

if (Search(k) == nullptr)

{

throw "В сортированных таблицах отсутствует элемент, который хотите удалить.";

}

else

{

int i = 0;

while (str[i]->key > k)

{

i++;

}

for (int j = i; j < size - 1; j++)

{

str[j] = str[j + 1];

}

str[size - 1] = NULL;

size = size - 1;

}

}

SortedTable::SortedTable(const ScanTable T)

{

ScanTable ST = T;

size = 0;

index = -1;

max\_size = ST.GetMaxSize();

ST.Reset();

for (int i = 0; i < ST.GetSize(); i++)

{

Insert(ST.GetCurrent()->key, ST.GetCurrent()->data);

ST.Next();

}

}

## Приложение Ж. Программная реализация хэш-таблицы

HashTable.h

#pragma once

#include "Table.h"

class HashTable : public table

{

private:

int hash(string key) const;

void Repacking() override;

int \*State;

public:

HashTable(int nsize = 10) : table(nsize) { size = 0; index = -1; State = new int[nsize]; for (int i = 0; i < max\_size; i++) { State[i] = 0; } };

void Insert(const string s, const TPolinom \*poli2) override;

void Delete(const string k) override;

unit\* Search(const string k) override;

};

int HashTable::hash(string key) const

{

int a = 7;

int h = 0;

for (int i = 0; i < key.length(); i++)

{

h = key[i] \* a + h;

}

return (h % max\_size);

}

void HashTable::Repacking()

{

max\_size = max\_size \* 2;

unit \*\*tmp = new unit\*[max\_size];

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

{

tmp[i] = NULL;

}

int \*State2;

State2 = new int[max\_size];

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

{

State2[i] = 0;

}

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

{

if (State[i] != 0)

{

int j = hash(str[i]->key);

tmp[j] = str[i];

State2[j] = 1;

}

}

str = tmp;

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

{

State[i] = State2[i];

}

}

void HashTable::Insert(const string s, const TPolinom \*poli2)

{

double a = size;

double b = max\_size;

if (a != 0)

{

if ((a / b) > 0.63)

{

Repacking();

}

}

if (Search(s) != nullptr)

{

throw "В хэш-таблице уже присутствует данный ключ.";

}

else

{

int place = hash(s);

if (State[place] == 0)

{

str[place] = new unit(s, poli2);

size = size + 1;

index = place;

State[place] = 1;

}

else

{

while (State[place] != 0)

{

place = (place + 11) % max\_size;

}

str[place] = new unit(s, poli2);

State[place] = 1;

size = size + 1;

index = place;

}

}

}

void HashTable::Delete(const string k)

{

if (Search(k) == nullptr)

{

throw "В хэш-таблицах отсутствует элемент, который хотите удалить.";

}

else

{

str[index] = NULL;

State[index] = -1;

size = size - 1;

}

}

unit\* HashTable::Search(const string k)

{

int place = hash(k);

int first = -10;

if (State[place] == 0)

{

return nullptr;

}

else if ((State[place] != -1) && (str[place]->key != k))

{

first = place;

place = (place + 11) % max\_size;

while ((State[place] != 0) && (place != first))

{

if ((State[place] != -1) && (str[place]->key == k))

{

index = place;

return str[place];

}

else

{

place = (place + 11) % max\_size;

}

}

}

if (State[place] != 1)

{

return nullptr;

}

else if (place == first)

{

return nullptr;

}

}

## Приложение З. Реализация основного приложения

#include "ScanTable.h"

#include "SortedTable.h"

#include "HashTable.h"

#include <iostream>

#include <stdio.h>

#include <cstring>

#include <string>

using namespace std;

void main()

{

setlocale(LC\_ALL, "Russian");

int c = 0;

int k;

int SIZE;

cout << "Пожалуйста, введите размер таблицы: ";

cin >> SIZE;

cout << endl;

ScanTable A(SIZE);

SortedTable B(SIZE);

HashTable C(SIZE);

while (c != 2)

{

system("cls");

cout << "Выберите операцию." << endl;

cout << "1. Вставка" << endl;

cout << "2. Удаление" << endl;

cout << "3. Поиск" << endl;

cout << "Ваш выбор: ";

cin >> k;

cout << endl;

switch (k)

{

case 1:

{

string c;

cout << "Введите строку: ";

cin.ignore();

getline(cin, c);

TPolinom F(c);

try

{

A.Insert(c, &F);

}

catch (const char \*ex)

{

cout << ex << endl;

}

try

{

B.Insert(c, &F);

}

catch (const char \*ex)

{

cout << ex << endl;

}

try

{

C.Insert(c, &F);

}

catch (const char \*ex)

{

cout << ex << endl;

}

cout << endl << "Просматриваемая таблица:" << endl;

A.PrintTable();

cout << endl << "Сортированная таблица: " << endl;

B.PrintTable();

cout << endl << "Хэш-таблица: " << endl;

C.PrintTable();

cout << endl;

break;

}

case 2:

{

string c;

cout << "Введите строку: ";

cin.ignore();

getline(cin, c);

TPolinom F(c);

try

{

A.Delete(c);

}

catch (const char \*ex)

{

cout << ex << endl;

}

try

{

B.Delete(c);

}

catch (const char \*ex)

{

cout << ex << endl;

}

try

{

C.Delete(c);

}

catch (const char \*ex)

{

cout << ex << endl;

}

cout << endl << "Просматриваемая таблица:" << endl;

A.PrintTable();

cout << endl << "Сортированная таблица: " << endl;

B.PrintTable();

cout << endl << "Хэш-таблица: " << endl;

C.PrintTable();

cout << endl;

break;

}

case 3:

{

string c;

cout << "Введите строку: ";

cin.ignore();

getline(cin, c);

TPolinom F(c);

cout << endl;

A.PrintTable();

cout << endl;

B.PrintTable();

cout << endl;

C.PrintTable();

cout << endl;

if (A.Search(c) != nullptr)

{

cout << endl << "Позиция в просматриваемой таблице: " << A.GetIndex() << endl;

}

else

{

cout << "Элемент не найден. " << endl;

}

if (B.Search(c) != nullptr)

{

cout << endl << "Позиция в сортированной таблице: " << B.GetIndex() << endl;

}

else

{

cout << "Элемент не найден. " << endl;

}

if (C.Search(c) != nullptr)

{

cout << endl << "Позиция в хэш-таблице: " << C.GetIndex() << endl;

}

else

{

cout << "Элемент не найден. " << endl;

}

break;

}

}

cout << endl << "Продолжить работу в программе? " << endl;

cout << "1. Да" << endl;

cout << "2. Нет" << endl;

cout << "Ваш выбор: ";

cin >> c;

}

cin.get();

}