МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

**"Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского"**

**Институт информационных технологий, математики и механики**

**ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ**

**Аналитические преобразования полиномов**

**от нескольких переменных**

**Выполнил:** студент группы 381806-1

Кизаев Дмитрий Дмитриевич

**Проверил:** кандидат технических наук доцент кафедры МОСТ ИИТММ

Сысоев Александр Владимирович

Нижний Новгород

2020

**Оглавление**

[Введение 3](#_Toc59544240)

[Постановка задачи 4](#_Toc59544241)

[Руководство пользователя 5](#_Toc59544242)

[Руководство программиста 7](#_Toc59544243)

[4.1 Структура программы 7](#_Toc59544244)

[4.2 Описание алгоритма 7](#_Toc59544245)

[4.2.1 Класс TList 7](#_Toc59544246)

[4.2.2 Класс TPolynom 8](#_Toc59544247)

[4.3 Описание функций 9](#_Toc59544248)

[4.3.1 Функции класса TNode 9](#_Toc59544249)

[4.1.1 Функции класса TList 10](#_Toc59544250)

[4.1.1 Функции класса TMonom 11](#_Toc59544251)

[4.1.1 Функции класса TPolynom 13](#_Toc59544252)

[Заключение 18](#_Toc59544253)

[Приложение 19](#_Toc59544254)

Введение

В наши дни компьютеры широко используются для аналитической обработки данных. Академическое применение компьютерных вычислений может включать в себя компьютерное доказательство теорем, логический вывод, анализ текстовой информации и многое другое. Отдельное место в этом списке занимает задача обработки полиномов, задаваемых в общей аналитической форме. Полиномы являются хорошо изученной областью математики (алгебра полиномов), которая широко используется в приложениях (аппроксимация экспериментальных данных, построение функциональных зависимостей и т.п.).

Лабораторная работа направлена на изучение методов компьютерной обработки полиномов. С этой целью в лабораторной работе изучаются различные варианты структуры хранения и разрабатывается программа для обработки полиномов. Основной учебной целью работы является практическое освоение методов организации структур хранения данных с помощью списков. В ходе выполнения лабораторной работы разрабатывается общая форма представления линейных списков, разрабатываются программы работы со списками, которые могут быть использованы и в других областях приложений.

Постановка задачи

Разработать программу, позволяющую проводить преобразования над полиномами от нескольких переменных, где каждый моном имеет следующий вид: K\*xa ybzс.

Программа должна реализовывать следующие функции:

* Ввод, хранение и удаление (обработка) полинома
* Копирование полиномов
* Операции над полиномами
  + Сложение полиномов
  + Вычитание полиномов
  + Умножение полиномов
* Операции над полиномами и константами
  + Аналогично списку выше
* Вывод результатов операций на экран

Входные данные:

* Строка с полиномом

Выходные данные:

* Итог всех преобразований над полиномом

При выполнении лабораторной работы можно использовать следующие основные предположения:

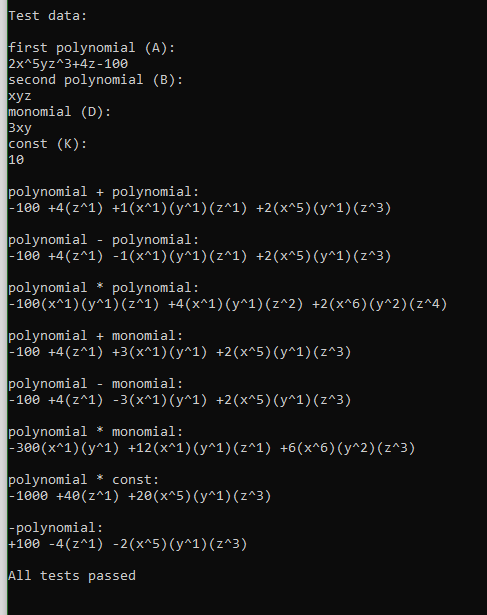
* Разработка структуры хранения должна быть ориентирована на представление полиномов от трех неизвестных.
* Степени переменных полиномов не могут превышать значения 9, т.е. 0 ≤ i, j, k ≤ 9. 57
* Число мономов в полиномах существенно меньше максимально возможного количества (тем самым, в структуре хранения должны находиться только мономы с ненулевыми коэффициентами).

Руководство пользователя

После запуска программа на экран выводится запрос на выбор режима (Рисунок 1).



В тестовом режиме реализованные операции выполняются для предварительно заданных монома, полинома и константы. Результаты выводятся на экран:



В режиме ручного ввода требуется выбрать операцию, а затем ввести операнды:

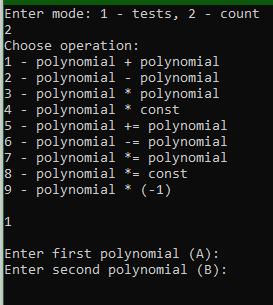


Рисунок . Меню действий

После ввода программа выведет результат, и можно будет повторить выполнение для других операций и операндов.

Руководство программиста

* 1. Структура программы

Программа состоит из 8 файлов:

* *Main.cpp,* который содержит функцию *main*
* *TList.h*, который содержит класс TList
* *TNode.h* который содержит класс TNode
* *TMonom.h*,который содержит специализацию класса TNode<unsigned int, double>
* *TMonom.cpp*, который содержит реализацию функций специализации класса TNode<unsigned int, double>
* *TPolynom.h*,который содержит класс TPolynom
* *TPolynom.cpp*, который содержит реализацию функций специализации класса TPolynom
  1. Описание алгоритма
     1. Класс TList

В программе используется такая структура данных, как связный список. Связный список — базовая динамическая структура данных, состоящая из узлов (в нашей программе мономов), каждый из которых содержит поле с данными, ключ и одну ссылку («связку») на следующий узел списка. Поле указателя последнего узла содержит нулевое значение.

Характеристиками связного списка являются:

* Указатель на первый элемент списка
* Указатель на предыдущий элемент списка
* Указатель на текущий элемент списка
* Указатель на следующий элемент списка

Операции над списком:

* Проверка на нахождение последнего элемента списка.
  1. Возвращает true, если указатель на текущий элемент имеет нулевое значение, если это не так, возвращает false.
  2. Переход к следующему элементу списка.
  3. Проверяем не находимся ли мы в конце списка. Если это так, бросаем исключение.
  4. Перебрасываем указатель текущего элемента на следующий элемент.
* Возвращение к первому элементу списка.
  1. Присваиваем указателю на текущий элемент списка указатель на первый элемент списка.
* Поиск элемента с заданным ключом.
  1. Выполняем обход по списку до тех пор, пока не встретится искомого звено.
  2. Если звено найдено – выходим из цикла и возвращаем найденное звено
  3. Если звено не найдено – возвращаем nullptr.
* Вставка элемента в конец списка.
  1. Если список пустой, то указатель на начало списка ссылаем на новый элемент
  2. Если список не пустой, то доходим до конца списка
  3. указатель на следующее звено последнего звена ссылаем на новый элемент, у нового звена следующим делаем nullptr.
* Вставка элемента в начало списка.
  1. Указатель на следующий элемент нового звена ссылаем на первое звено.
  2. Если список пустой, то ссылаем на nullptr.
  3. Указатель на начало списка ставим на новый элемент.
* Вставка элемента после заданного элемента.
  1. Если список пустой, бросаем исключение.
  2. Находим в списке заданное звено.
  3. Его указатель на следующий ссылаем на новое звено.
  4. Указатель на следующий у нового звена ссылаем на следующий после заданного.
  5. Если заданное звено списка не было найдено, бросается исключение.
* Вставка элемента перед заданным элементом.
  1. Если список пустой, бросаем исключение.
  2. Находим в списке заданное звено.
  3. Его предыдущему звену ссылаем указатель на следующий на новое звено
  4. Указатель на следующий нового звена ссылаем на заданное звено
  5. Если заданное звено списка не было найдено, бросается исключение.
* Удаление элемента с заданным ключом
  1. Проверяем список на пустоту. Если он пуст, бросаем исключение.
  2. Ищем элемент, который нужно удалить. Если его нет в списке, бросаем исключение. Искомый элемент становится текущем.
  3. Если текущий совпадает с первым элементом, удаляем первый элемент. Указатель на первый элемент переходит к следующему.
  4. Если не равны, меняем указатель на следующий элемент у предыдущего элемента на следующий элемент текущего. Удаляем текущий.
     1. Класс TPolynom

Структура данных полином основана на односвязном списке, где каждый узел списка является мономом. Ключ несет в себе значение – степени монома, где цифра в сотнях обозначает степень переменной Х, цифра в десятках степень Y, цифра в единицах – степень Z. При этом значение ключа не должно превышать 999. Данные являются значением коэффициента при мономе.

Характеристики полинома:

* Указатель на связный список.

Операции над полиномом:

* Присваивание полиномов.
  1. Если указатель на текущий полином равен указателю на переданный, то возвращаем текущий полином.
  2. Если переданный полином не пустой, то удаляем все его мономы.
  3. Создаем новый список мономов с помощью конструктора копирования.
  4. Возвращаем текущий полином.
* Сложение двух полиномов.
  1. Создаем результирующий полином.
  2. Операция сложения двух полиномов проходит на основе операции сложения полинома с мономом
  3. Сложение будет проводится до тех пор, пока не закончится переданный в функцию полином.
* Вычитание двух полиномов.
  1. Сложение двух полиномов, с использованием над одним из них операцию унарного минуса.
* Умножение полиномов.
  1. Реализовано с помощью перегрузки операции умножения у мономов.
  2. Проходим по одному полиному, последовательно перемножая текущий моном на мономы второго полинома.
* Унарный минус.
  1. Все коэффициентов мономов умножаются на -1.
* Сложение полинома с мономом
  1. Создаем результирующий полином на базе текущего.
  2. Если у результирующего полинома указатель на первый моном листа равен nullptr, вставляем переданный в функцию моном в конец листа
  3. Возвращаем текущий полином.
* Вычитание монома из полинома
  1. Данная операция происходит на основе операции сложения полинома с мономом, у которого заранее изменен знак на противоположный.
* Умножение полинома на моном
  1. Эта операция основана на перегрузке операции умножения мономов.
  2. Создается результирующий полином, все мономы которого умножаются на переданный моном (если лист в полиноме не пуст).
  3. Возвращаем результирующий полином.
  4. Описание функций
     1. Функции класса TNode

TNode()

* Назначение

Создание объекта класса TNode

TNode(const TNode& tmp)

* Назначение

Конструктор копирования класса TNode

* Входные параметры

tmp – объект класса, копия которого будет создаваться

TNode(Tkey \_key, TData \_data, TNode\* next = 0)

* Назначение

Создание объекта класса TNode с параметрами ключа, данных и указателем на следующий элемент

* Входные параметры

\_key – ключ

\_data – данные

next – указатель на следующий элемент

~TNode()

* Назначение

Удаление объекта класса TNode

* + 1. Функции класса TList

TList()

* Назначение

Создание объекта класса TList

TList(const TList<Tkey, TData>& list)

* Назначение

Конструктор копирования класса TList

* Входные параметры

list – объект класса, копия которого будет создаваться

TList(TNode<Tkey, TData>\* tmp)

* Назначение

Конструктор преобразования типов класса TList

* Входные параметры

tmp – объект класса TNode, на основе которого будет создан объект класса TList

~TList()

* Назначение

Удаление объекта класса Stack

bool is\_ended()const

* Назначение

Проверка списка на окончание

* Входные параметры

Отсутствуют

* Выходные параметры

Значение типа bool

void go\_next()

* Назначение

Смещение навигации в списке вперед

* Входные параметры

Отсутствуют

* Выходные параметры

Отсутствуют

void reset();

* Назначение

Смещение навигации в списке на начало списка

* Входные параметры

Отсутствуют

* Выходные параметры

Отсутствуют

TNode<Tkey, TData>\* Search(Tkey key)

* Назначение

Поиск элемента списка по заданному ключу

* Входные параметры

key – ключ, поиск которого будет выполняться

* Выходные параметры

Указатель на элемент списка с искомым ключом

void insert\_to\_end(const TNode<Tkey, TData>& tmp)

* Назначение

Вставка нового элемента в конец списка

* Входные параметры

tmp – вставляемый элемент

* Выходные параметры

Отсутствуют

void insert\_to\_start(const TNode<Tkey, TData>& tmp)

* Назначение

Вставка нового элемента в начало списка

* Входные параметры

tmp – вставляемый элемент

* Выходные параметры

Отсутствуют

void insert\_after(const TNode<Tkey, TData>& tmp, Tkey key)

* Назначение

Вставка нового элемента после элемента с заданным ключом в списке

* Входные параметры

tmp – вставляемый элемент

key – ключ элемента, после которого будет осуществлена вставка

* Выходные параметры

Отсутствуют

void InsertBefore(const TNode<Tkey, TData>& tmp, Tkey key)

* Назначение

Вставка нового элемента перед элементом с заданным ключом в списке

* Входные параметры

tmp – вставляемый элемент

key – ключ элемента, перед которым будет осуществлена вставка

* Выходные параметры

Отсутствуют

void Remove(Tkey key);

* Назначение

Удаление элемента с заданным ключом

* Входные параметры

key – ключ удаляемого элемента

* Выходные параметры

Отсутствуют

* + 1. Функции класса TMonom

TMonom()

* Назначение

Создание объекта класса TMonom

TMonom(const TMonom& tmp)

* Назначение

Конструктор копирования класса TMonom

* Входные параметры

tmp – объект класса, копия которого будет создаваться

TMonom(Tkey \_key, TData \_data, TMonom\* next = 0)

* Назначение

Создание объекта класса TMonom с параметрами ключа, данных и указателем на следующий элемент

* Входные параметры

\_key – ключ

\_data – данные

next – указатель на следующий элемент

~TMonom()

* Назначение

Удаление объекта класса TMonom

TMonom& operator=(const TMonom& tmp)

* Назначение

Оператор присваивания мономов

* Входные параметры

tmp – элемент класса, значение которого будет присваиваться

* Выходные параметры

Ссылка на новый объект класса TMonom, значение которого совпадает с tmp

bool operator==(const TMonom& tmp)const

* Назначение

Оператор сравнения мономов

* Входные параметры

tmp – элемент, с которым будет сравниваться моном

* Выходные параметры

Значение типа bool

TMonom operator+(const TMonom& tmp)

* Назначение

Оператор сложения мономов

* Входные параметры

tmp – моном, с которым будет складываться исходный моном

* Выходные параметры

Результат сложения мономов

TMonom operator-(const TMonom& tmp)

* Назначение

Оператор вычитания мономов

* Входные параметры

tmp – моном, который будет вычитаться из исходного монома

* Выходные параметры

Результат вычитания мономов

TMonom operator\*(const TMonom& tmp)

* Назначение

Оператор умножения мономов

* Входные параметры

tmp – моном, на который будет умножаться исходный моном

* Выходные параметры

Результат умножения мономов

TMonom operator\*(double c)

* Назначение

Оператор умножения монома на константу

* Входные параметры

с – константа, на которую будет умножаться исходный моном

* Выходные параметры

Результат умножения монома на константу

TMonom& operator+=(const TMonom& tmp)

* Назначение

Оператор сложения мономов

* Входные параметры

tmp – моном, с которым будет складываться исходный моном

* Выходные параметры

Исходный моном, который сложили с tmp

TMonom& operator-=(const TMonom& tmp)

* Назначение

Оператор вычитания мономов

* Входные параметры

tmp – моном, который будет вычитаться из исходного монома

* Выходные параметры

Исходный моном, из которого вычли tmp

TMonom& operator\*=(const TMonom& tmp)

* Назначение

Оператор умножения мономов

* Входные параметры

tmp – моном, на который будет умножаться исходный моном

* Выходные параметры

Исходный моном, который умножили на tmp

TMonom& operator\*=(double c)

* Назначение

Оператор умножения монома на константу

* Входные параметры

с – константа, на которую будет умножаться исходный моном

* Выходные параметры

Исходный моном, который умножили на константу

TMonom operator-() const

* Назначение

Оператор унарного минуса

* Входные параметры

Отсутствуют

* Выходные параметры

Моном, над которым произвели операцию унарный минус

* + 1. Функции класса TPolynom

TPolynom()

* Назначение

Создание объекта класса TPolynom

TPolynom(const TPolynom& tmp)

* Назначение

Конструктор копирования класса TPolynom

* Входные параметры

tmp – объект класса, копия которого будет создаваться

TPolynom(const TList<unsigned int, double>\* list)

* Назначение

Конструктор преобразования типов класса TPolynom

* Входные параметры

list –объект класса TList, на основе которого будет создан объект класса TPolynom

TPolynom(const string& st)

* Назначение

Конструктор преобразования типов класса TPolynom

* Входные параметры

st –строка, на основе которой будет создан объект класса TPolynom

~TPolynom()

* Назначение

Удаление объекта класса TPolynom

void insert\_into(TMonom node)

* Назначение

Вставка нового монома в нужное место в полиноме, используется в конструкторах

* Входные параметры

node – вставляемый моном

* Выходные параметры

Отсутствуют

bool IsDouble(string a);

* Назначение

Проверка строки на содержание числовых значений, используется в конструкторе

* Входные параметры

a – проверяемая строка

* Выходные параметры

Значение типа bool

void CheckLim(const TList<unsigned int, double>\* tmp)const

* Назначение

Проверка листа из мономов на содержание степеней превышающих значение 999

* Входные параметры

tmp – проверяемый лист

* Выходные параметры

Значение типа bool

TPolynom operator+(const TPolynom& tmp)

* Назначение

Оператор сложения полиномов

* Входные параметры

tmp – полином, с которым складывают исходный полином

* Выходные параметры

Результат сложения полиномов

TPolynom operator-(const TPolynom& tmp)

* Назначение

Оператор вычитания полиномов

* Входные параметры

tmp – полином, который вычитают из исходного полинома

* Выходные параметры

Результат разности полиномов

TPolynom operator\*(const TPolynom& tmp)

* Назначение

Оператор умножения полиномов

* Входные параметры

tmp – полином, на который умножают исходный полином

* Выходные параметры

Результат умножения полиномов

TPolynom operator+(const TMonom& tmp)

* Назначение

Оператор сложения полинома и монома

* Входные параметры

tmp – моном, с которым складывают исходный полином

* Выходные параметры

Результат сложения полинома и монома

TPolynom operator-(const TMonom& tmp)

* Назначение

Оператор вычитания монома из полинома

* Входные параметры

tmp – моном, который вычитают из исходного полинома

* Выходные параметры

Результат разности полинома и монома

TPolynom operator\*(const TMonom& tmp)

* Назначение

Оператор умножения полинома на моном

* Входные параметры

tmp – моном, на который умножают исходный полином

* Выходные параметры

Результат умножения полинома на моном

TPolynom operator\*(double c)

* Назначение

Оператор умножения полинома на константу

* Входные параметры

c – константа, на которую умножают исходный полином

* Выходные параметры

Результат умножения полинома на константу

TPolynom& operator+=(const TPolynom& tmp)

* Назначение

Оператор сложения полиномов

* Входные параметры

tmp – полином, с которым складывают исходный полином

* Выходные параметры

Исходный полином, который сложили с tmp

TPolynom& operator-=(const TPolynom& tmp)

* Назначение

Оператор вычитания полиномов

* Входные параметры

tmp – полином, который вычитают из исходного полинома

* Выходные параметры

Исходный полином, из которого вычли tmp

TPolynom& operator\*=(const TPolynom& tmp)

* Назначение

Оператор умножения полиномов

* Входные параметры

tmp – полином, на который умножают исходный полином

* Выходные параметры

Исходный полином, который умножили на tmp

TPolynom& operator+=(const TMonom& tmp)

* Назначение

Оператор сложения полинома и монома

* Входные параметры

tmp – моном, с которым складывают исходный полином

* Выходные параметры

Исходный полином, который сложили с мономом

TPolynom& operator-=(const TMonom& tmp)

* Назначение

Оператор вычитания монома из полинома

* Входные параметры

tmp – моном, который вычитают из исходного полинома

* Выходные параметры
* Исходный полином, из которого вычли моном

TPolynom& operator\*=(const TMonom& tmp)

* Назначение

Оператор умножения полинома на моном

* Входные параметры

tmp – моном, на который умножают исходный полином

* Выходные параметры
* Исходный полином, который умножили на моном

TPolynom& operator\*=(double c)

* Назначение

Оператор умножения полинома на константу

* Входные параметры

c – константа, на которую умножают исходный полином

* Выходные параметры
* Исходный полином, который умножили на константу

TPolynom& operator=(const TPolynom&)

* Назначение

Оператор присваивания полиномов

* Входные параметры

tmp – элемент класса, значение которого будет присваиваться

* Выходные параметры

Ссылка на новый объект класса TPolynom, значение которого совпадает с tmp

bool operator==(const TPolynom&)const

* Назначение

Оператор сравнения полиномов

* Входные параметры

tmp – элемент, с которым будет сравниваться полином

* Выходные параметры

Значение типа bool

TPolynom operator-() const

* Назначение

Оператор унарного минуса

* Входные параметры

Отсутствуют

* Выходные параметры

Полином, взятый с обратным знаком

Заключение

В ходе практической работы была разработана и реализована программа, поддерживающую аналитические преобразования над полиномами от 3 переменных. В ходе работы были реализованы: шаблонный класс списка, шаблонный класс узла списка, их специализации, необходимые для структуры данных Полином. Также были успешно проведены тесты по работе с программой.

Приложение

* ***Main.cpp***

#include <iostream>

#include "TPolynom.h"

#include <string>

using namespace std;

void main()

{

int mode = 0;

cout << "Enter mode: 1 - tests, 2 - count" << endl;

cin >> mode;

if (mode == 1)

{

cout << endl << "Test data:" << endl << endl;

string string\_1 = "2x^5yz^3+4z-100", string\_2 = "xyz", string\_3 = "3xy";

double k = 10;

cout << "first polynomial (A):" << endl;

cout << string\_1 << endl;

cout << "second polynomial (B):" << endl;

cout << string\_2 << endl;

cout << "monomial (D):" << endl;

cout << string\_3 << endl;

cout << "const (K):" << endl;

cout << k << endl;

TPolynom a(string\_1);

TPolynom b(string\_2);

TMonom d(string\_3);

TPolynom dd(string\_3);

try

{

TPolynom test1 = a + b;

cout << endl << "polynomial + polynomial:" << endl;

cout << test1 << endl;

}

catch (char\* error\_msg)

{

cout << error\_msg << endl;

};

try

{

TPolynom test2 = a - b;

cout << endl << "polynomial - polynomial:" << endl;

cout << test2 << endl;

}

catch (char\* error\_msg)

{

cout << error\_msg << endl;

};

try

{

TPolynom test3 = a \* b;

cout << endl << "polynomial \* polynomial:" << endl;

cout << test3 << endl;

}

catch (char\* error\_msg)

{

cout << error\_msg << endl;

};

try

{

TPolynom test4 = a + dd;

cout << endl << "polynomial + monomial:" << endl;

cout << test4 << endl;

}

catch (char\* error\_msg)

{

cout << error\_msg << endl;

};

try

{

TPolynom test5 = a - dd;

cout << endl << "polynomial - monomial:" << endl;

cout << test5 << endl;

}

catch (char\* error\_msg)

{

cout << error\_msg << endl;

};

try

{

TPolynom test6 = a \* dd;

cout << endl << "polynomial \* monomial:" << endl;

cout << test6 << endl;

}

catch (char\* error\_msg)

{

cout << error\_msg << endl;

};

try

{

TPolynom test7 = a \* k;

cout << endl << "polynomial \* const:" << endl;

cout << test7 << endl;

}

catch (char\* error\_msg)

{

cout << error\_msg << endl;

};

try

{

TPolynom test8 = -a;

cout << endl << "-polynomial:" << endl;

cout << test8 << endl;

}

catch (char\* error\_msg)

{

cout << error\_msg << endl;

};

cout << endl << "All tests passed" << endl;

string go\_on;

cin >> go\_on;

cout << endl;

}

else if (mode == 2)

while (true)

{

int oper;

try

{

cout << "Choose operation:" << endl <<

"1 - polynomial + polynomial" << endl <<

"2 - polynomial - polynomial" << endl <<

"3 - polynomial \* polynomial" << endl <<

"4 - polynomial \* const" << endl <<

"5 - polynomial += polynomial" << endl <<

"6 - polynomial -= polynomial" << endl <<

"7 - polynomial \*= polynomial" << endl <<

"8 - polynomial \*= const" << endl <<

"9 - polynomial \* (-1)" << endl <<

endl;

cin >> oper;

while (!((oper >= 1) && (oper <= 15)))

cin >> oper;

cout << endl;

switch (oper)

{

case 1:

try

{

string string\_1, string\_2;

cout << "Enter first polynomial (A):" << endl;

cin >> string\_1;

cout << "Enter second polynomial (B):" << endl;

cin >> string\_2;

TPolynom a(string\_1);

TPolynom b(string\_2);

TPolynom c = a + b;

cout << endl << "A + B:" << endl;

cout << c << endl;

}

catch (char\* error\_msg)

{

cout << error\_msg << endl;

}

break;

case 2:

try

{

string string\_1, string\_2;

cout << "Enter first polynomial (A):" << endl;

cin >> string\_1;

cout << "Enter second polynomial (B):" << endl;

cin >> string\_2;

TPolynom a(string\_1);

TPolynom b(string\_2);

TPolynom c = a - b;

cout << endl << "A - B:" << endl;

cout << c << endl;

}

catch (char\* error\_msg)

{

cout << error\_msg << endl;

}

break;

case 3:

try

{

string string\_1, string\_2;

cout << "Enter first polynomial (A):" << endl;

cin >> string\_1;

cout << "Enter second polynomial (B):" << endl;

cin >> string\_2;

TPolynom a(string\_1);

TPolynom b(string\_2);

TPolynom c = a \* b;

cout << endl << "A \* B:" << endl;

cout << c << endl;

}

catch (char\* error\_msg)

{

cout << error\_msg << endl;

}

break;

case 4:

try

{

string string\_1;

double b;

cout << "Enter polynomial (A):" << endl;

cin >> string\_1;

cout << "Enter const (B):" << endl;

cin >> b;

TPolynom a(string\_1);

TPolynom c = a \* b;

cout << endl << "A \* B:" << endl;

cout << c << endl;

}

catch (char\* error\_msg)

{

cout << error\_msg << endl;

}

break;

case 5:

try

{

string string\_1, string\_2;

cout << "Enter first polynomial (A):" << endl;

cin >> string\_1;

cout << "Enter second polynomial (B):" << endl;

cin >> string\_2;

TPolynom a(string\_1);

TPolynom b(string\_2);

a += b;

cout << endl << "A += B:" << endl;

cout << a << endl;

}

catch (char\* error\_msg)

{

cout << error\_msg << endl;

}

break;

case 6:

try

{

string string\_1, string\_2;

cout << "Enter first polynomial (A):" << endl;

cin >> string\_1;

cout << "Enter second polynomial (B):" << endl;

cin >> string\_2;

TPolynom a(string\_1);

TPolynom b(string\_2);

a -= b;

cout << endl << "A -= B:" << endl;

cout << a << endl;

}

catch (char\* error\_msg)

{

cout << error\_msg << endl;

}

break;

case 7:

try

{

string string\_1, string\_2;

cout << "Enter first polynomial (A):" << endl;

cin >> string\_1;

cout << "Enter second polynomial (B):" << endl;

cin >> string\_2;

TPolynom a(string\_1);

TPolynom b(string\_2);

a \*= b;

cout << endl << "A \*= B:" << endl;

cout << a << endl;

}

catch (char\* error\_msg)

{

cout << error\_msg << endl;

}

break;

case 8:

try

{

string string\_1;

double b;

cout << "Enter polynomial (A):" << endl;

cin >> string\_1;

cout << "Enter const (B):" << endl;

cin >> b;

TPolynom a(string\_1);

a \*= b;

cout << endl << "A \*= B:" << endl;

cout << a << endl;

}

catch (char\* error\_msg)

{

cout << error\_msg << endl;

}

break;

case 9:

try

{

string string\_1;

cout << "Enter polynomial (A):" << endl;

cin >> string\_1;

TPolynom a(string\_1);

TPolynom c = -a;

cout << endl << "-A:" << endl;

cout << c << endl;

}

catch (char\* error\_msg)

{

cout << error\_msg << endl;

}

break;

default:

break;

}

}

catch (char\* error\_msg)

{

cout << error\_msg;

}

cout << endl;

string go\_on;

cin >> go\_on;

cout << endl;

}

}

* ***TList.h***

#pragma once

#include "TNode.h"

#include <iostream>

using namespace std;

template <class Tkey, class TData>

class TList

{

public:

TNode<Tkey, TData>\* p\_first;

TNode<Tkey, TData>\* p\_previous;

TNode<Tkey, TData>\* p\_current;

TNode<Tkey, TData>\* p\_next;

TList();

TList(const TList<Tkey, TData>& list);

TList(TNode<Tkey, TData>\* tmp);

~TList();

bool is\_ended()const;

void go\_next();

void reset();

TNode<Tkey, TData>\* Search(Tkey key);

void insert\_to\_end(const TNode<Tkey, TData>&);

void insert\_to\_start(const TNode<Tkey, TData>&);

void insert\_after(const TNode<Tkey, TData>&, Tkey key);

void insert\_before(const TNode<Tkey, TData>&, Tkey key);

void remove(Tkey key);

friend ostream& operator<<(ostream& out, TList<Tkey, TData>& list)

{

list.reset();

while (!(list.is\_ended()))

{

out << list->p\_current << endl;

list.go\_next();

}

return out;

}

};

template <class Tkey, class TData>

TList<Tkey, TData>::TList()

{

p\_first = nullptr;

p\_previous = nullptr;

p\_current = nullptr;

p\_next = nullptr;

}

template <class Tkey, class TData>

TList<Tkey, TData>::TList(const TList<Tkey, TData>& list)

{

if (list.p\_first == nullptr)

{

p\_first = nullptr;

p\_previous = nullptr;

p\_current = nullptr;

p\_next = nullptr;

return;

}

p\_first = new TNode<Tkey, TData>(\*(list.p\_first));

p\_current = p\_first;

TNode<Tkey, TData>\* node = list.p\_first;

node = node->p\_next;

while (node != nullptr)

{

p\_current->p\_next = new TNode<Tkey, TData>(\*node);

p\_current = p\_current->p\_next;

node = node->p\_next;

}

reset();

}

template <class Tkey, class TData>

TList<Tkey, TData>::TList(TNode<Tkey, TData>\* list)

{

p\_first = new TNode<Tkey, TData>(\*list);

p\_current = p\_first;

TNode<Tkey, TData>\* node = list;

node = node->p\_next;

while (node != nullptr)

{

p\_current->p\_next = new TNode<Tkey, TData>(\*node);

p\_current = p\_current->p\_next;

node = node->p\_next;

}

reset();

}

template <class Tkey, class TData>

TList<Tkey, TData>::~TList()

{

reset();

while (!is\_ended())

{

go\_next();

delete p\_previous;

}

delete p\_current;

p\_first = nullptr;

p\_current = nullptr;

p\_previous = nullptr;

p\_next = nullptr;

}

template <class Tkey, class TData>

bool TList<Tkey, TData>::is\_ended()const

{

return (p\_current == nullptr);

}

template <class Tkey, class TData>

void TList<Tkey, TData>::go\_next()

{

if (is\_ended())

return;

p\_previous = p\_current;

p\_current = p\_current->p\_next;

if (p\_next != nullptr)

p\_next = p\_next->p\_next;

}

template <class Tkey, class TData>

void TList<Tkey, TData>::reset()

{

p\_previous = nullptr;

p\_current = p\_first;

if (p\_first != nullptr)

{

p\_next = p\_current->p\_next;

}

else

{

p\_next = nullptr;

}

}

template <class Tkey, class TData>

TNode<Tkey, TData>\* TList<Tkey, TData>::Search(Tkey \_key)

{

reset();

while ((!is\_ended()) && (p\_current->key != \_key))

go\_next();

if (is\_ended())

{

reset();

return nullptr;

}

return p\_current;

}

template <class Tkey, class TData>

void TList<Tkey, TData>::insert\_to\_end(const TNode<Tkey, TData>& tmp)

{

if (p\_first == nullptr)

{

p\_first = new TNode<Tkey, TData>(tmp);

p\_current = p\_first;

}

else

{

TNode<Tkey, TData>\* pprev = p\_previous;

TNode<Tkey, TData>\* pcurr = p\_current;

TNode<Tkey, TData>\* pnext = p\_next;

while (!is\_ended())

go\_next();

p\_current = new TNode<Tkey, TData>(tmp);

p\_previous->p\_next = p\_current;

p\_previous = pprev;

p\_current = pcurr;

p\_next = pnext;

}

}

template <class Tkey, class TData>

void TList<Tkey, TData>::insert\_to\_start(const TNode<Tkey, TData>& tmp)

{

TNode<Tkey, TData>\* tmp1 = new TNode<Tkey, TData>(tmp.key, tmp.data, p\_first);

if (p\_current == p\_first)

{

p\_previous = tmp1;

}

p\_first = tmp1;

}

template <class Tkey, class TData>

void TList<Tkey, TData>::insert\_after(const TNode<Tkey, TData>& s, Tkey n)

{

if (p\_first == nullptr)

throw "er";

TNode<Tkey, TData>\* pprev = p\_previous;

TNode<Tkey, TData>\* pcurr = p\_current;

TNode<Tkey, TData>\* pnext = p\_next;

if (Search(n) == nullptr)

{

p\_previous = pprev;

p\_current = pcurr;

p\_next = pnext;

return;

}

TNode<Tkey, TData>\* tmp = new TNode<Tkey, TData>(s.key, s.data, p\_next);

p\_current->p\_next = tmp;

p\_previous = pprev;

p\_current = pcurr;

p\_next = pnext;

}

template <class Tkey, class TData>

void TList<Tkey, TData>::insert\_before(const TNode<Tkey, TData>& s, Tkey n)

{

if (p\_first == nullptr)

throw "er";

TNode<Tkey, TData>\* pprev = p\_previous;

TNode<Tkey, TData>\* pcurr = p\_current;

TNode<Tkey, TData>\* pnext = p\_next;

if (Search(n) == nullptr)

{

p\_previous = pprev;

p\_current = pcurr;

p\_next = pnext;

return;

}

if (p\_first == p\_current)

{

TNode<Tkey, TData>\* tmp = new TNode<Tkey, TData>(s.key, s.data, p\_first);

if (pcurr == p\_first)

{

p\_previous = tmp;

}

else

{

p\_previous = pprev;

p\_current = pcurr;

p\_next = pnext;

}

p\_first = tmp;

return;

}

TNode<Tkey, TData>\* tmp = new TNode<Tkey, TData>(s.key, s.data, p\_current);

p\_previous->p\_next = tmp;

p\_previous = pprev;

p\_current = pcurr;

p\_next = pnext;

}

template <class Tkey, class TData>

void TList<Tkey, TData>::remove(Tkey s)

{

if (p\_first == nullptr)

throw "er";

TNode<Tkey, TData>\* pprev = p\_previous;

TNode<Tkey, TData>\* pcurr = p\_current;

TNode<Tkey, TData>\* pnext = p\_next;

if (Search(s) == 0)

{

p\_previous = pprev;

p\_current = pcurr;

p\_next = pnext;

return;

}

if (p\_first == p\_current)

{

if (pcurr == p\_first)

{

if (p\_next != nullptr)

if (p\_next == 0)

{

p\_next = p\_next->p\_next;

}

p\_current = p\_next;

p\_previous->p\_next = p\_current;

}

delete p\_first;

p\_first = p\_next;

return;

}

if (pcurr == p\_current)

{

delete p\_current;

p\_current = pnext;

p\_previous->p\_next = p\_current;

if (p\_next == 0)

{

p\_next = p\_next->p\_next;

}

return;

}

p\_previous->p\_next = p\_current->p\_next;

delete p\_current;

p\_current = p\_next;

if (p\_next == 0)

{

p\_next = p\_next->p\_next;

}

p\_previous = pprev;

p\_current = pcurr;

p\_next = pnext;

}

* ***TNode.h***

#pragma once

#include <iostream>

using namespace std;

template <class Tkey, class TData>

class TNode

{

public:

Tkey key;

TData data;

TNode\* p\_next;

TNode();

TNode(const TNode&);

TNode(Tkey, TData, TNode\* next = 0);

~TNode();

friend ostream& operator<<(ostream& out, TNode<Tkey, TData>& node)

{

out << node.key << " - " << node.data;

return out;

}

};

template <class Tkey, class TData>

TNode<Tkey, TData>::TNode()

{

p\_next = nullptr;

}

template <class Tkey, class TData>

TNode<Tkey, TData>::TNode(const TNode<Tkey, TData>& tmp)

{

key = tmp.key;

data = tmp.data;

p\_next = nullptr;

}

template <class Tkey, class TData>

TNode<Tkey, TData>::TNode(Tkey \_key, TData \_data, TNode\* next)

{

key = \_key;

data = \_data;

p\_next = next;

}

template <class Tkey, class TData>

TNode<Tkey, TData>::~TNode()

{

key = nullptr;

data = nullptr;

}

* ***TMonom.h***

#pragma once

#include "TList.h"

#include <iostream>

using namespace std;

template <>

class TNode<unsigned int, double>

{

public:

unsigned int key;

double data;

TNode\* p\_next;

TNode();

TNode(const string tmp);

TNode(const TNode<unsigned int, double>&);

TNode(unsigned int, double, TNode<unsigned int, double>\* next = 0);

~TNode();

TNode<unsigned int, double>& operator=(const TNode<unsigned int, double>&);

bool operator==(const TNode<unsigned int, double>&)const;

TNode<unsigned int, double> operator+(const TNode<unsigned int, double>&);

TNode<unsigned int, double> operator-(const TNode<unsigned int, double>&);

TNode<unsigned int, double> operator\*(const TNode<unsigned int, double>&);

TNode<unsigned int, double> operator\*(double);

TNode<unsigned int, double>& operator+=(const TNode<unsigned int, double>&);

TNode<unsigned int, double>& operator-=(const TNode<unsigned int, double>&);

TNode<unsigned int, double>& operator\*=(const TNode<unsigned int, double>&);

TNode<unsigned int, double>& operator\*=(double);

TNode<unsigned int, double> operator-() const;

friend ostream& operator<<(ostream& out, TNode<unsigned int, double>& node)

{

if (node.data > 0)

out << "+";

out << node.data;

if (node.key / 100 != 0)

out << "(x^" << node.key / 100 << ")";

if ((node.key / 10) % 10 != 0)

out << "(y^" << (node.key / 10) % 10 << ")";

if (node.key % 10 != 0)

out << "(z^" << node.key % 10 << ")";

out << " ";

return out;

}

};

* ***TMonom.cpp***

#include "TMonom.h"

#include <iostream>

#include <string>

using namespace std;

typedef TNode<unsigned int, double> TMonom;

TNode<unsigned int, double>::TNode()

{

key = 0;

data = 0;

p\_next = nullptr;

}

TNode<unsigned int, double>::TNode(const string input\_str)

{

int i = 0;

char sign = '+';

int power = 0;

char current = 'n';

if (input\_str[0] == '-')

{

sign = '-';

i++;

}

string coef = "";

int num = 0;

while (i < input\_str.length())

{

switch (input\_str[i])

{

case 'x':

power += 100;

if (current == 'n')

{

if (coef == "")

num = 1;

else

num = stoi(coef);

coef = "";

}

current = 'x';

break;

case 'y':

power += 10;

if (current == 'n')

{

if (coef == "")

num = 1;

else

num = stoi(coef);

coef = "";

}

else if (current == 'x')

{

if (coef == "")

num = 1;

else if (coef == "^")

num = stoi(coef);

power += 100 \* (num - 1);

coef = "1";

}

current = 'y';

break;

case 'z':

power += 1;

if (current == 'n')

{

if (coef == "")

num = 1;

else

num = stoi(coef);

coef = "1";

}

else if (current == 'x')

{

if (coef == "")

num = 1;

else

num = stoi(coef);

power += 100 \* (num - 1);

coef = "";

}

else if (current == 'y')

{

if (coef == "")

num = 1;

else

num = stoi(coef);

power += 10 \* (num - 1);

coef = "";

}

current = 'z';

break;

case '^':

break;

default:

coef += input\_str[i];

break;

}

i++;

}

if (coef == "")

num = 1;

else

num = stoi(coef);

if (current == 'x')

power += 100 \* (num - 1);

else if (current == 'y')

power += 10 \* (num - 1);

else if (current == 'z')

power += num - 1;

key = power;

data = num;

p\_next = nullptr;

}

TNode<unsigned int, double>::TNode(const TMonom& tmp)

{

key = tmp.key;

data = tmp.data;

p\_next = nullptr;

}

TNode<unsigned int, double>::TNode(unsigned int \_conv, double \_coeff, TNode<unsigned int, double> \* next)

{

key = \_conv;

data = \_coeff;

p\_next = next;

}

TNode<unsigned int, double>::~TNode()

{}

TMonom& TNode<unsigned int, double>::operator=(const TNode<unsigned int, double> & tmp)

{

if (\*this == tmp)

return \*this;

key = tmp.key;

data = tmp.data;

p\_next = nullptr;

}

bool TNode<unsigned int, double>::operator==(const TNode<unsigned int, double> & tmp)const

{

if ((tmp.key != key) || (tmp.data != data))

return false;

return true;

}

TMonom TNode<unsigned int, double>::operator+(const TNode<unsigned int, double> & tmp)

{

if (key != tmp.key)

throw "!diff conv";

TMonom rez(tmp);

rez.data += data;

return rez;

}

TMonom TNode<unsigned int, double>::operator-(const TNode<unsigned int, double> & tmp)

{

if (key != tmp.key)

throw "!diff conv";

TMonom rez(\*this);

rez.data -= tmp.data;

return rez;

}

TMonom TNode<unsigned int, double>::operator\*(const TNode<unsigned int, double> & tmp)

{

if (((tmp.key / 100) + (key / 100) > 9) || (((tmp.key / 10) % 10) + ((key / 10) % 10) > 9) ||

((tmp.key % 10) + (key % 10) > 9))

throw "!limit power";

TMonom rez;

rez.data = data \* tmp.data;

rez.key = key + tmp.key;

return rez;

}

TMonom TNode<unsigned int, double>::operator\*(double c)

{

TMonom rez(\*this);

rez.data \*= c;

return rez;

}

TMonom& TNode<unsigned int, double>::operator+=(const TNode<unsigned int, double> & tmp)

{

if (key != tmp.key)

throw "!diff conv";

data += tmp.data;

return \*this;

}

TMonom& TNode<unsigned int, double>::operator-=(const TNode<unsigned int, double> & tmp)

{

if (key != tmp.key)

throw "!diff conv";

data -= tmp.data;

return \*this;

}

TMonom& TNode<unsigned int, double>::operator\*=(const TNode<unsigned int, double> & tmp)

{

if (((tmp.key / 100 + key / 100) > 9) || ((tmp.key / 10 + key / 10) > 90)

|| ((tmp.key % 10 + key % 10) > 9))

throw "!limit power";

key += tmp.key;

data \*= tmp.data;

return \*this;

}

TMonom& TNode<unsigned int, double>::operator\*=(double c)

{

data \*= c;

return \*this;

}

TMonom TNode<unsigned int, double> ::operator-() const

{

TMonom rez(\*this);

rez \*= (-1);

return rez;

}

* ***TPolynom.h***

#pragma once

#include "TList.h"

#include "TMonom.h"

#include <iostream>

using namespace std;

typedef TNode<unsigned int, double> TMonom;

class TPolynom

{

private:

TList<unsigned int, double>\* polynomial;

void insert\_into(TMonom node);

bool is\_double(string a);

void check\_lim(const TList<unsigned int, double>\*)const;

public:

TPolynom();

TPolynom(const TList<unsigned int, double>\*);

TPolynom(const string&);

TPolynom(const TPolynom&);

~TPolynom();

TPolynom operator+(const TPolynom&);

TPolynom operator-(const TPolynom&);

TPolynom operator\*(const TPolynom&);

TPolynom operator+(const TMonom&);

TPolynom operator-(const TMonom&);

TPolynom operator\*(const TMonom&);

TPolynom operator\*(double);

TPolynom& operator+=(const TPolynom&);

TPolynom& operator-=(const TPolynom&);

TPolynom& operator\*=(const TPolynom&);

TPolynom& operator+=(const TMonom&);

TPolynom& operator-=(const TMonom&);

TPolynom& operator\*=(const TMonom&);

TPolynom& operator\*=(double);

TPolynom& operator=(const TPolynom&);

bool operator==(const TPolynom&)const;

TPolynom operator-() const;

friend ostream& operator<<(ostream& out, TPolynom& tmp)

{

TMonom\* saved\_prev = tmp.polynomial->p\_previous;

TMonom\* saved\_curr = tmp.polynomial->p\_current;

TMonom\* saved\_next = tmp.polynomial->p\_next;

tmp.polynomial->reset();

while (!tmp.polynomial->is\_ended())

{

out << \*(tmp.polynomial->p\_current);

tmp.polynomial->go\_next();

}

tmp.polynomial->p\_previous = saved\_prev;

tmp.polynomial->p\_current = saved\_curr;

tmp.polynomial->p\_next = saved\_next;

return out;

}

* ***TPolynom.cpp***

#include "TPolynom.h"

#include <iostream>

#include <string>

using namespace std;

TPolynom::TPolynom()

{

polynomial = new TList<unsigned int, double>();

}

TPolynom::TPolynom(const TList<unsigned int, double>\* tm)

{

TList<unsigned int, double>\* tmp = new TList<unsigned int, double>(\*tm);

polynomial = new TList<unsigned int, double>(tmp->p\_first);

check\_lim(tmp);

tmp->reset();

tmp->go\_next();

int conv = tmp->p\_first->key;

while (!tmp->is\_ended())

{

if (tmp->p\_current->key < conv)

{

polynomial->insert\_before(\*tmp->p\_current, conv);

}

else

{

polynomial->insert\_after(\*tmp->p\_current, conv);

}

conv = tmp->p\_current->key;

}

}

TPolynom::TPolynom(const string& st)

{

if (st == "")

throw "!empty line";

polynomial = new TList<unsigned int, double>();

string tmp = "";

string powerst = "";

char sign = '+';

char current = ' ';

unsigned int power = 0;

double num = 0;

int i = 0;

TMonom rez;

rez.p\_next = NULL;

while (i < st.length())

{

if (st[i] != ' ')

{

switch (st[i])

{

case 'x':

power += 100;

current = 'x';

break;

case 'y':

power += 10;

current = 'y';

break;

case 'z':

power += 1;

current = 'z';

break;

case '+':

if ((tmp == "") && (power != 0))

num = 1;

else if (tmp != "")

num = stoi(tmp);

if (sign == '-')

num \*= (-1);

rez.key = power;

rez.data = num;

insert\_into(rez);

power = 0;

num = 0;

tmp = "";

sign = '+';

break;

case '-':

if ((tmp == "") && (power != 0))

num = 1;

else if (tmp != "")

num = stoi(tmp);

if (sign == '-')

num \*= (-1);

rez.key = power;

rez.data = num;

insert\_into(rez);

power = 0;

num = 0;

tmp = "";

sign = '-';

break;

case '\*':

break;

case '^':

i++;

powerst += st[i];

switch (current)

{

case 'x':

power += (stoi(powerst) - 1) \* 100;

break;

case 'y':

power += (stoi(powerst) - 1) \* 10;

break;

case 'z':

power += stoi(powerst) - 1;

break;

}

powerst = st[i + 1];

if (is\_double(powerst))

throw "limit power";

else

powerst = "";

break;

default:

tmp += st[i];

break;

}

}

i++;

}

if (tmp == "")

num = 1;

else if (tmp != "")

num = stoi(tmp);

if (sign == '-')

num \*= (-1);

rez.key = power;

rez.data = num;

insert\_into(rez);

}

TPolynom::TPolynom(const TPolynom& tmp)

{

polynomial = new TList<unsigned int, double>(\*tmp.polynomial);

}

TPolynom::~TPolynom()

{

delete polynomial;

}

TPolynom TPolynom::operator+(const TPolynom& tm)

{

TPolynom tmp(tm);

tmp.polynomial->reset();

TPolynom rez(\*this);

rez.polynomial->reset();

while (tmp.polynomial->p\_current != NULL)

{

rez = rez + \*tmp.polynomial->p\_current;

tmp.polynomial->go\_next();

}

return rez;

}

TPolynom TPolynom::operator-(const TPolynom& tm)

{

TPolynom rez = (\*this) + (-tm);

return rez;

}

TPolynom TPolynom::operator\*(const TPolynom& tm)

{

TPolynom tmp(tm);

TPolynom tmp1(\*this);

tmp1.polynomial->reset();

tmp.polynomial->reset();

TPolynom rez;

while (!tmp1.polynomial->is\_ended())

{

rez += tmp \* \*tmp1.polynomial->p\_current;

tmp1.polynomial->go\_next();

}

return rez;

}

TPolynom TPolynom::operator+(const TMonom& tmp)

{

TPolynom rez(\*this);

rez.polynomial->reset();

if (rez.polynomial->p\_first == NULL)

{

rez.polynomial->insert\_to\_end(tmp);

return rez;

}

while ((rez.polynomial->p\_current != NULL) && (rez.polynomial->p\_current->key < tmp.key))

rez.polynomial->go\_next();

if (rez.polynomial->p\_current == NULL)

{

rez.polynomial->insert\_to\_end(tmp);

return rez;

}

if (rez.polynomial->p\_current->key == tmp.key)

{

if (rez.polynomial->p\_current->data + tmp.data == 0)

{

rez.polynomial->remove(rez.polynomial->p\_current->key);

}

else

{

\*rez.polynomial->p\_current += tmp;

}

}

else if (rez.polynomial->p\_current->key > tmp.key)

{

rez.polynomial->insert\_before(tmp, rez.polynomial->p\_current->key);

}

return rez;

}

TPolynom TPolynom::operator-(const TMonom& tmp)

{

TPolynom rez = \*this + (-tmp);

return rez;

}

TPolynom TPolynom::operator\*(const TMonom& tmp)

{

TPolynom rez;

if (tmp.data == 0)

return rez;

rez = \*this;

rez.polynomial->reset();

while (rez.polynomial->p\_current != NULL)

{

\*rez.polynomial->p\_current \*= tmp;

rez.polynomial->go\_next();

}

return rez;

}

TPolynom TPolynom::operator\*(double c)

{

TPolynom rez;

if (c == 0)

return rez;

rez = \*this;

rez.polynomial->reset();

while (!rez.polynomial->is\_ended())

{

\*rez.polynomial->p\_current \*= c;

rez.polynomial->go\_next();

}

return rez;

}

TPolynom& TPolynom::operator=(const TPolynom& tm)

{

if (\*this == tm)

return \*this;

delete polynomial;

polynomial = new TList<unsigned int, double>(\*tm.polynomial);

return \*this;

}

TPolynom& TPolynom::operator+=(const TPolynom& tm)

{

TPolynom tmp = (\*this) + tm;

\*this = tmp;

return \*this;

}

TPolynom& TPolynom::operator-=(const TPolynom& tm)

{

TPolynom tmp = (\*this) - tm;

\*this = tmp;

return \*this;

}

TPolynom& TPolynom::operator\*=(const TPolynom& tm)

{

TPolynom tmp = (\*this) \* tm;

\*this = tmp;

return \*this;

}

TPolynom& TPolynom::operator+=(const TMonom& tm)

{

TPolynom tmp = (\*this) \* tm;

\*this = tmp;

return \*this;

}

TPolynom& TPolynom::operator-=(const TMonom& tm)

{

TPolynom tmp = (\*this) - tm;

\*this = tmp;

return \*this;

}

TPolynom& TPolynom::operator\*=(const TMonom& tm)

{

TPolynom tmp = (\*this) \* tm;

\*this = tmp;

return \*this;

}

TPolynom& TPolynom::operator\*=(double c)

{

TPolynom tmp = (\*this) \* c;

\*this = tmp;

return \*this;

}

TPolynom TPolynom::operator-() const

{

TPolynom rez(\*this);

rez \*= (-1);

return rez;

}

bool TPolynom::operator==(const TPolynom& tmp)const

{

TPolynom a(\*this);

TPolynom b(tmp);

while ((!a.polynomial->is\_ended()) || (!b.polynomial->is\_ended()))

{

if (a.polynomial->p\_current != b.polynomial->p\_current)

return false;

a.polynomial->go\_next();

b.polynomial->go\_next();

}

if ((a.polynomial->is\_ended()) && (b.polynomial->is\_ended()))

return true;

return false;

}

bool TPolynom::is\_double(string a)

{

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

{

if ((a[i] != '0') && (a[i] != '1') && (a[i] != '2') && (a[i] != '3') && (a[i] != '4') &&

(a[i] != '5') && (a[i] != '6') && (a[i] != '7') && (a[i] != '8') && (a[i] != '9') && (a[i] != ' '))

return false;

}

return true;

}

void TPolynom::insert\_into(TMonom node)

{

if ((node.key == 0) && (node.data == 0))

return;

if (node.data == 0)

return;

if (polynomial->p\_first == NULL)

{

polynomial->insert\_to\_end(node);

return;

}

polynomial->reset();

while ((polynomial->p\_current->key < node.key) && (polynomial->p\_current->p\_next != NULL))

{

polynomial->go\_next();

}

if (polynomial->p\_current->key == node.key)

{

polynomial->p\_current->data = polynomial->p\_current->data + node.data;

return;

}

else if (polynomial->p\_current->key > node.key)

{

polynomial->insert\_before(node, polynomial->p\_current->key);

return;

}

polynomial->insert\_to\_end(node);

}

void TPolynom::check\_lim(const TList<unsigned int, double>\* tm)const

{

TList<unsigned int, double> tmp(\*tm);

tmp.reset();

while (!tmp.is\_ended())

{

if (tmp.p\_current->key > 999)

{

throw "!limit power";

}

tmp.go\_next();

}

}