МИНИСТЕРСТВО НАУКИ И ВЫСШЕГООБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**ЛАБОРАТОРНАЯ РАБОТА**

на тему:

**«Полиномы»**

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Савченко М.П./

Подпись

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

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

Подпись

Нижний Новгород  
2023

**Содержание**

# Введение

Полиномы являются важным математическим объектом, широко применяемым в различных областях науки и техники. Они используются для аппроксимации функций, решения уравнений, моделирования и анализа данных. В данной лабораторной работе мы рассматриваем полиномы от трех переменных , которые представляют собой алгебраические выражения, составленные из суммы и произведения степенных функций переменных. Для удобства манипуляции и вычислений с такими полиномами необходимо реализовать соответствующую структуру данных.

В данном отчете представлена реализация класса, описывающего полиномы от трех переменных на языке программирования C++ с использованием связных списков. Класс позволяет удобно создавать, хранить, модифицировать и вычислять значения полиномов. Использование связных списков обеспечивает гибкую структуру хранения и управления полиномами любой степени и сложности.

# Постановка задачи

**Цель лабораторной работы:**

Целью данной лабораторной работы является разработка и реализация класса, описывающего полиномы от трех переменных (x, y, z) на основе односвязных списков на языке программирования C++. Разработанный класс должен предоставлять удобные средства для работы с полиномиальными выражениями, включая их создание, модификацию и выполнение операций над ними.

**Задачи лабораторной работы:**

1. Спроектировать и описать структуру звеньев односвязного списка, которая будет использоваться для представления мономов.
2. Разработать класс односвязного списка, который будет содержать методы для добавления, удаления и обхода элементов списка.
3. Создать класс монома, который будет представлять отдельные члены полинома и содержать информацию о степени переменных и их коэффициенте.
4. Реализовать операции для класса монома, такие как сложение, вычитание и умножение, взятие производной по переменной.
5. Разработать класс полинома, который будет содержать методы для создания, модификации и вычисления полиномиальных выражений.
6. Обеспечить возможность выполнения арифметических операций над полиномами, таких как сложение, вычитание и умножение.
7. Провести тестирование разработанных классов на различных входных данных, чтобы убедиться в их корректности и эффективности.
8. Описать процесс разработки, принятые решения и особенности реализации в отчете по лабораторной работе.

# Руководство пользователя

## Приложение для демонстрации работы связного списка

1. Запустите приложение с названием \*.exe. В результате появится окно, показанное ниже (рис. 1).



1. Основное окно программы

## Приложение для демонстрации работы полиномов

# Руководство программиста

## Описание алгоритмов

### Связный список

### Полиномы

## Описание программной реализации

### Описание класса TNode

template <typename T>

struct TNode {

T key;

TNode<T>\* pNext;

TNode() : key(), pNext(nullptr) {};

TNode(const T& data) : key(data), pNext(nullptr) {};

TNode(TNode<T>\* \_pNext) : key(), pNext(\_pNext) {};

TNode(const T& data, TNode<T>\* \_pNext) : key(data), pNext(\_pNext) {};

};

Назначение: представление звена списка.

**Поля:**

key – данные, хранящиеся в звене.

pNext – указатель на следующее звено.

**Конструкторы:**

TNode();

Назночение: конструктор по умолчанию.

TNode(const T& data);

Назночение: конструктор с параметром.

Входные параметры: data – данные для хранения в звене.

TNode(TNode<T>\* \_pNext);

Назночение: конструктор с параметром.

Входные параметры: \_pNext – указатель на следующее звено.

TNode(const T& data, TNode<T>\* \_pNext);

Назночение: конструктор с параметрами.

Входные параметры: data – данные для хранения в звене, \_pNext – указатель на следующее звено.

### Описание класса TList

template <typename T>

class TList {

protected:

TNode<T>\* pFirst;

TNode<T>\* pLast;

TNode<T>\* pCurr;

TNode<T>\* pStop;

public:

TList();

TList(TNode<T>\* \_pFirst);

TList(const TList<T>& obj);

virtual ~TList();

virtual TNode<T>\* Search(const T& data);

virtual void InsertFirst(const T& data);

virtual void InsertLast(const T& data);

virtual void InsertBefore(const T& data, const T& before);

virtual void InsertAfter(const T& data, const T& after);

virtual void InsertBefore(const T& data);

virtual void InsertAfter(const T& data);

virtual void Remove(const T& data);

virtual void Clear();

virtual size\_t GetSize() const;

virtual bool IsFull() const;

virtual bool IsEmpty() const;

virtual void Reset();

virtual TNode<T>\* GetCurrent() const;

virtual void Next(const int count = 1);

virtual bool IsEnded() const;

virtual TNode<T>\* first() const;

virtual TNode<T>\* last() const;

virtual TNode<T>\* stop() const;

};

Назначение: стуртура данных «Односвязный список».

**Поля:**

pFirst – указатель на первый элемент.

pLast – указатель на последний элемент.

pCurr – указатель на текущий элемент.

pStop – указатель на конец списка.

**Конструкторы:**

TList();

Назночение: конструктор по умолчанию.

TList(TNode<T>\* \_pFirst);

Назночение: конструктор преобразования типа.

Входные параметры: \_pFirst – указатель на первое звено.

TList(const TList<T>& obj);

Назночение: конструктор копирования.

Входные параметры: obj – копируемый односвязный список.

virtual ~TList();

Назночение: деструктор.

**Методы:**

virtual TNode<T>\* Search(const T& data);

Назночение: поиск звена в списке по ключю.

Входные параметры: data – искомый ключ.

Выходные параметры: указатель на найденное звено в списке.

virtual void InsertFirst(const T& data);

Назночение: вставка в начало списка.

Входные параметры: data – вставляемые данные.

virtual void InsertLast(const T& data);

Назночение: вставка в конец списка.

Входные параметры: data – вставляемые данные.

virtual void InsertBefore(const T& data, const T& before);

Назночение: вставка перед заданным элементом.

Входные параметры: data – вставляемые данные, before – элемент, перед которым должна произойти вставка.

virtual void InsertAfter(const T& data, const T& after);

Назночение: вставка после заданного элемента.

Входные параметры: data – вставляемые данные, before – элемент, после которого должна произойти вставка.

virtual void InsertBefore(const T& data);

Назночение: вставка перед текущем элементом.

Входные параметры: data – вставляемые данные.

virtual void InsertAfter(const T& data);

Назночение: вставка после текущего элемента.

Входные параметры: data – вставляемые данные.

virtual void Remove(const T& data);

Назночение: удаление элемента из списка.

Входные параметры: data – удаляеммые данные.

virtual void Clear();

Назночение: удаление всех звеньев списка.

virtual size\_t GetSize() const;

Назночение: получение количества звеньев в списке.

Выходные параметры: количества звеньев в списка.

virtual bool IsFull() const;

Назночение: проверка на заполненность памяти.

Выходные параметры: true или false.

virtual bool IsEmpty() const;

Назночение: проверка, пуст ли список.

Выходные параметры: true или false.

virtual void Reset();

Назночение: переход в начало списка.

virtual TNode<T>\* GetCurrent() const;

Назночение: получение текущего звена списка.

Выходные параметры: ссылка на текущее звено списка.

virtual void Next(const int count = 1);

Назночение: переход на следующее звено списка.

Входные параметры: count – количество переходов.

virtual bool IsEnded() const;

Назночение: проверка, является ли текущее звено концом списка.

Выходные параметры: true или false.

virtual TNode<T>\* first() const;

Назночение: получение указателя на первое звено списка.

Выходные параметры: указатель на первое звено списка.

virtual TNode<T>\* last() const;

Назночение: получение указателя на последнее звено списка.

Выходные параметры: указатель на последнее звено списка.

virtual TNode<T>\* stop() const;

Назночение: получение указателя на конец списка.

Выходные параметры: указатель на конец списка.

### Описание класса TRingList

template <typename T>

class TRingList : public TList<T> {

private:

TNode<T>\* pHead;

public:

TRingList();

TRingList(TNode<T>\* \_pFirst);

TRingList(const TRingList<T>& rlist);

virtual ~TRingList();

void InsertFirst(const T& data);

};

Назначение: стуртура данных «Односвязный список закольцованный».

**Поля:**

pHead – фиктивное звено для закальцовывания списка.

**Конструкторы:**

TRingList();

Назночение: конструктор по умолчанию.

TRingList(TNode<T>\* \_pFirst);

Назночение: конструктор преобразования типа.

Входные параметры: \_pFirst – указатель на первое звено.

TRingList(const TRingList<T>& rlist);

Назночение: конструктор копирования.

Входные параметры: rlist – копируемый список.

virtual ~TRingList();

Назночение: деструктор.

**Методы:**

void InsertFirst(const T& data);

Назночение: вставка в начало списка.

Входные параметры: data – вставляемые данные.

### Описание класса TMonom

class TMonom{

private:

double coeff;

int degree;

public:

TMonom();

TMonom(const TMonom& monom);

TMonom(double \_coeff, int \_degree);

TMonom(double \_coeff, int degX, int degY, int degZ);

double get\_coeff() const;

int get\_degree() const;

int get\_degX() const;

int get\_degY() const;

int get\_degZ() const;

void set\_coeff(double \_coeff);

void set\_degree(int \_degree);

void set\_degree(int degX, int degY, int degZ);

double value(double x, double y, double z);

virtual bool operator<(const TMonom& monom) const;

virtual bool operator>(const TMonom& monom) const;

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

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

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

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

const TMonom& operator=(const TMonom& data);

TMonom operator\*(const TMonom& monom) const;

TMonom dif\_x() const;

TMonom dif\_y() const;

TMonom dif\_z() const;

string get\_string() const;

friend ostream& operator<<(ostream& out, const TMonom& monom);

};

Назначение: представление монома.

**Поля:**

coeff – коэффициент монома.

degree – свертка степеней монома.

**Конструкторы:**

TMonom();

Назночение: конструктор по умолчанию.

TMonom(const TMonom& monom);

Назночение: конструктор копирования.

Входные параметры: monom – копируемый моном.

TMonom(double \_coeff, int \_degree);

Назночение: конструктор с параметрами.

Входные параметры: \_coeff – коэффициент монома, \_degree – свертка степеней монома.

TMonom(double \_coeff, int degX, int degY, int degZ);

Назночение: конструктор с параметрами.

Входные параметры: \_coeff – коэффициент монома, degX,degY,degZ – степени соответствующих переменных.

**Методы:**

double get\_coeff() const;

Назночение: получение значения коэффициента монома.

Выходные параметры: значение коэффициента монома.

int get\_degree() const;

Назночение: получение значения свертки степеней монома.

Выходные параметры: значение свертки степеней монома

int get\_degX() const;

Назночение: получение значения степени переменной X.

Выходные параметры: значение степени переменной X.

int get\_degY() const;

Назночение: получение значения степени переменной Y.

Выходные параметры: значение степени переменной Y.

int get\_degZ() const;

Назночение: получение значения степени переменной Z.

Выходные параметры: значение степени переменной Z.

void set\_coeff(double \_coeff);

Назночение: задание коэффициента монома.

Входные параметры: \_coeff - коэффициент монома.

void set\_degree(int \_degree);

Назночение: задание свертки степеней монома.

Входные параметры: \_degree – свертка степени монома.

void set\_degree(int degX, int degY, int degZ);

Назночение: задание свертки степеней монома.

Входные параметры: degX,degY,degZ – степени соответствующих переменных.

double value(double x, double y, double z);

Назночение: получение значения монома в точке.

Входные параметры: x,y,z – координаты точки.

Выходные параметры: значение монома в точке

virtual bool operator<(const TMonom& monom) const;

Назночение: операция сравнения “<”.

Входные параметры: monom – сравниваемый моном.

Выходные параметры: true или false.

virtual bool operator>(const TMonom& monom) const;

Назночение: операция сравнения “>”.

Входные параметры: monom – сравниваемый моном.

Выходные параметры: true или false.

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

Назночение: операция сравнения “<=”.

Входные параметры: monom – сравниваемый моном.

Выходные параметры: true или false.

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

Назночение: операция сравнения “>=”.

Входные параметры: monom – сравниваемый моном.

Выходные параметры: true или false.

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

Назночение: операция сравнения “==”.

Входные параметры: monom – сравниваемый моном.

Выходные параметры: true или false.

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

Назночение: операция сравнения “!=”.

Входные параметры: monom – сравниваемый моном.

Выходные параметры: true или false.

const TMonom& operator=(const TMonom& data);

Назночение: оператор присваивания монома.

Входные параметры: data – присваемый моном.

Выходные параметры: ссылка на себя.

TMonom operator\*(const TMonom& monom) const;

Назночение: оператор умножения мономов.

Входные параметры: monom – моном, на который происходит умножение.

Выходные параметры: результат умножения мономов.

TMonom dif\_x() const;

Назночение: взятие производной монома по X.

Выходные параметры: производная монома по X.

TMonom dif\_y() const;

Назночение: взятие производной монома по Y.

Выходные параметры: производная монома по Y.

TMonom dif\_z() const;

Назночение: взятие производной монома по Z.

Выходные параметры: производная монома по Z.

string get\_string() const;

Назночение: преобразование монома в строку.

Выходные параметры: сформированная строка монома.

friend ostream& operator<<(ostream& out, const TMonom& monom);

Назночение: вывод строки монома.

Входные параметры: out – поток вывода, monom – выводящийся моном.

Выходные параметры: поток вывода.

### Описание класса TPolynom

class TPolynom {

private:

TRingList<TMonom>\* monoms;

static map<string, int> priority;

void Parse(string& name);

void ToMonoms(vector<string>& lexems);

void AddMonom(const TMonom& m);

public:

TPolynom();

TPolynom(const string& \_name);

TPolynom(const TRingList<TMonom>& ringlist);

TPolynom(const TPolynom& polynom);

~TPolynom();

const TPolynom& operator=(const TPolynom& polynom);

const TPolynom operator+(const TPolynom& polynom) const;

const TPolynom operator-() const;

const TPolynom operator-(const TPolynom& polynom) const;

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

double operator()(double x, double y, double z) const;

TPolynom dif\_x() const;

TPolynom dif\_y() const;

TPolynom dif\_z() const;

string get\_string() const;

friend ostream& operator<<(ostream& out, const TPolynom& polynom);

private:

bool IsOperator(const string& isopr) const;

bool IsConst(const string& isopd) const;

bool IsVariable(const string& str) const;

int FindOperator(const string& name, int pos = 0) const;

void ConvertInfix(string& name);

void CorrectnessCheck(const string& name) const;

};

Назначение: представление полинома.

**Поля:**

monoms – указатель на односвязный кольцевой список мономов.

priority – список арифметических операций с их приоритетами.

**Конструкторы:**

TPolynom();

Назночение: конструктор по умолчанию

TPolynom(const string& \_name);

Назночение: конструктор с параметром.

Входные параметры: \_name – строка полинома.

TPolynom(const TRingList<TMonom>& ringlist);

Назночение: конструктор преобразования типа

Входные параметры: ringlist - односвязный кольцевой список мономов.

TPolynom(const TPolynom& polynom);

Назночение: конструктор копирования.

Входные параметры: polynom – копируемый полином.

~TPolynom();

Назночение: деструктор.

**Методы:**

const TPolynom& operator=(const TPolynom& polynom);

Назночение: оператор присваивания полинома.

Входные параметры: polynom – присваиваемый полином.

Выходные параметры: ссылка на себя.

const TPolynom operator+(const TPolynom& polynom) const;

Назночение: оператор сложения полиномов.

Входные параметры: polynom – прибавляемый полином.

Выходные параметры: результат сложения полиномов.

const TPolynom operator-() const;

Назночение: оператор унарного минуса.

Выходные параметры: результат унарного минуса.

const TPolynom operator-(const TPolynom& polynom) const;

Назночение: оператор вычитания полиномов.

Входные параметры: polynom – вычитаемый полином.

Выходные параметры: результат вычитания полиномов

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

Назночение: оператор произведения полиномов.

Входные параметры: polynom – полином, на который умножают.

Выходные параметры: результат произведения полиномов

double operator()(double x, double y, double z) const;

Назночение: получение значения полинома в точке.

Входные параметры: x,y,z – координаты точки.

Выходные параметры: значение полинома в точке

TPolynom dif\_x() const;

Назночение: взятие производной полинома по X.

Выходные параметры: производная полинома по X.

TPolynom dif\_y() const;

Назночение: взятие производной полинома по Y.

Выходные параметры: производная полинома по Y.

TPolynom dif\_z() const;

Назночение: взятие производной полинома по Z.

Выходные параметры: производная полинома по Z.

string get\_string() const;

Назночение: преобразование полинома в строку.

Выходные параметры: сформированная строка полинома.

friend ostream& operator<<(ostream& out, const TPolynom& polynom);

Назночение: вывод строки полинома.

Входные параметры: out – поток вывода, monom – выводящийся полином.

Выходные параметры: поток вывода.

**Служебные методы:**

void Parse(string& name);

Назночение: анализ строки инфиксной формы арифметического выражения.

Входные параметры: name – строка инфиксной формы арифметического выражения.

void ToMonoms(vector<string>& lexems);

Назночение: анализ массива лексем инфиксной формы арифметического выражения для выделения отдельных мономов.

Входные параметры: lexems - лексемы инфиксной формы арифметического выражения.

void AddMonom(const TMonom& m);

Назночение: упорядоченное добавление монома в полином.

Входные параметры: m – добавляемый моном.

bool IsOperator(const string& isopr) const;

Назначение: проверка строки, является ли она доступным оператором.

Входные параметры: isopr – проверяемая строка.

Выходные параметры: true или false.

bool IsConst(const string& isopd) const;

Назночение: проверка строки, является ли она константой, но не операндом.

Входные параметры: isopd – проверяемая строка.

Выходные параметры: true или false.

bool IsVariable(const string& str) const;

Назночение: проверка строки, является ли она доступной переменной.

Входные параметры: str – проверяемая строка.

Выходные параметры: true или false.

int FindOperator(const string& name, int pos = 0) const;

Назначение: поиск первого встречного оператора.

Входные параметры: name – строка, в которой происходит поиск, pos – начальная позиция поиска.

Выходные параметры: индекс первого встречного оператора.

void ConvertInfix(string& name);

Назночение: корректирование строки инфиксной формы арифметического выражения (избавление от пробелов; добавление отсутствующих операторов умножения на скобки; избавление от сокращенной записи действительных констант).

Входные параметры: name – корректируемая строка.

void CorrectnessCheck(const string& name) const;

Назночение: проверка инфиксной формы арифметического выражения на наличие критических ошибок в ее записи.

Входные параметры: name – проверяемая строка.

# Заключение

В результате выполнения лабораторной работы был разработан и реализован класс, описывающий полиномы от трех переменных (x, y, z) на основе односвязных списков на языке программирования C++. Этот класс предоставляет удобные средства для работы с полиномиальными выражениями, включая создание, модификацию и выполнение операций над ними.

В процессе работы были успешно выполнены все поставленные задачи. Была спроектирована и описана структура звеньев односвязного списка, разработан класс односвязного списка с необходимыми методами, создан класс монома для представления отдельных членов полинома, а также реализован класс полинома с возможностью выполнения арифметических операций над ними.

Тестирование разработанных классов показало их корректность и эффективность при обработке различных входных данных.

Данная лабораторная работа позволила углубить понимание структур данных и алгоритмов, а также приобрести практические навыки в разработке классов и их применении для решения задач вычислительной математики.

В будущем возможно расширение функциональности разработанных классов, включая поддержку дополнительных операций над полиномами и оптимизацию алгоритмов для улучшения производительности.

# Литература

1. Лекция «Списковые структуры хранения» Сысоева А.В. https://cloud.unn.ru/s/x33MEa9on8HgNgw
2. Лекция «Списки в динамической памяти» Сысоева А.В. https://cloud.unn.ru/s/rCiKGSX33SSGPi4
3. Лекция «Полиномы» Сысоева А.В. https://cloud.unn.ru/s/t6o9kp5g9bpf2yz

# Приложения

## Приложение А. Реализация класса TNode

#include <iostream>

using namespace std;

template <typename T>

struct TNode {

T key;

TNode<T>\* pNext;

TNode() : key(), pNext(nullptr) {};

TNode(const T& data) : key(data), pNext(nullptr) {};

TNode(TNode<T>\* \_pNext) : key(), pNext(\_pNext) {};

TNode(const T& data, TNode<T>\* \_pNext) : key(data), pNext(\_pNext)

};

## Приложение Б. Реализация класса TList

#include "tnode.h"

template <typename T>

class TList {

protected:

TNode<T>\* pFirst;

TNode<T>\* pLast;

TNode<T>\* pCurr;

TNode<T>\* pStop;

public:

TList();

TList(TNode<T>\* \_pFirst);

TList(const TList<T>& obj);

virtual ~TList();

virtual TNode<T>\* Search(const T& data);

virtual void InsertFirst(const T& data);

virtual void InsertLast(const T& data);

virtual void InsertBefore(const T& data, const T& before);

virtual void InsertAfter(const T& data, const T& after);

virtual void InsertBefore(const T& data);

virtual void InsertAfter(const T& data);

virtual void Remove(const T& data);

virtual void Clear();

virtual size\_t GetSize() const;

virtual bool IsFull() const;

virtual bool IsEmpty() const;

virtual void Reset();

virtual TNode<T>\* GetCurrent() const;

virtual void Next(const int count = 1);

virtual bool IsEnded() const;

virtual TNode<T>\* first() const;

virtual TNode<T>\* last() const;

virtual TNode<T>\* stop() const;

};

template <typename T>

TList<T>::TList() {

pStop = nullptr;

pFirst = pStop;

pLast = pStop;

pCurr = pStop;

}

template <typename T>

TList<T>::TList(TNode<T>\* \_pFirst) {

pStop = nullptr;

pFirst = \_pFirst;

if (pFirst == pStop) {

pLast = pStop;

pCurr = pStop;

return;

}

TNode<T>\* tmp = pFirst;

while (tmp->pNext != pStop)

tmp = tmp->pNext;

pLast = tmp;

}

template <typename T>

TList<T>::TList(const TList<T>& obj) {

if (obj.pFirst == obj.pStop) {

pStop = nullptr;

pFirst = pStop;

pLast = pStop;

pCurr = pFirst;

return;

}

pFirst = new TNode<T>(obj.pFirst->key);

TNode<T>\* tmp = obj.pFirst;

TNode<T>\* pNode = pFirst;

while (tmp->pNext != obj.pStop) {

pNode->pNext = new TNode<T>(tmp->pNext->key);

pNode = pNode->pNext;

tmp = tmp->pNext;

}

pLast = pNode;

pStop = pLast->pNext;

pCurr = pFirst;

}

template <typename T>

TList<T>::~TList() {

Clear();

}

template <typename T>

TNode<T>\* TList<T>::Search(const T& data) { // if (obj.Search(data) == obj.stop) - NOT FOUND

TNode<T>\* tmp = pFirst;

while (tmp != pStop && tmp->key != data)

tmp = tmp->pNext;

return tmp;

}

template <typename T>

void TList<T>::InsertFirst(const T& data) {

TNode<T>\* tmp = new TNode<T>(data, pFirst);

if (pFirst == pStop) pLast = tmp;

pFirst = tmp;

if (pCurr == pStop) Reset();

}

template <typename T>

void TList<T>::InsertLast(const T& data) {

if (pFirst == pStop) {

InsertFirst(data);

return;

}

TNode<T>\* tmp = new TNode<T>(data, pStop);

pLast->pNext = tmp;

pLast = pLast->pNext;

if (pCurr == pStop) Reset();

}

template <typename T>

void TList<T>::InsertBefore(const T& data, const T& before) {

TNode<T>\* pBefore = nullptr;

TNode<T>\* pNode = pFirst;

while (pNode != pStop && pNode->key != before) {

pBefore = pNode;

pNode = pNode->pNext;

}

if (pNode == pStop) {

string exp = "Error: key not found";

throw exp;

}

if (pNode == pFirst) {

InsertFirst(data);

return;

}

TNode<T>\* tmp = new TNode<T>(data, pNode);

pBefore->pNext = tmp;

if (pCurr == pStop) Reset();

}

template <typename T>

void TList<T>::InsertAfter(const T& data, const T& after) {

TNode<T>\* pNode = Search(after);

if (pNode == pStop) {

string exp = "Error: key not found";

throw exp;

}

if (pNode == pLast) {

InsertLast(data);

return;

}

TNode<T>\* tmp = new TNode<T>(data, pNode->pNext);

pNode->pNext = tmp;

if (pCurr == pStop) Reset();

}

/////////////////////////////////

template <typename T>

void TList<T>::InsertBefore(const T& data) {

if (pCurr == pStop) {

string exp = "Error: pCurrent is not installed, use Reset() and Next()";

throw exp;

}

if (pCurr == pFirst) {

InsertFirst(data);

return;

}

TNode<T>\* before = pFirst;

while (before->pNext != pCurr)

before = before->pNext;

TNode<T>\* tmp = new TNode<T>(data, pCurr);

before->pNext = tmp;

}

template <typename T>

void TList<T>::InsertAfter(const T& data) {

if (pCurr == pStop) {

string exp = "Error: pCurrent is not installed, use Reset() and Next()";

throw exp;

}

if (IsEnded()) {

InsertLast(data);

return;

}

TNode<T>\* tmp = new TNode<T>(data, pCurr->pNext);

pCurr->pNext = tmp;

}

/////////////////////////////////

template <typename T>

void TList<T>::Remove(const T& data) {

if (pFirst == pStop) {

string exp = "Error: list is empty";

throw exp;

}

TNode<T>\* pNode = pFirst, \*pPrev = nullptr;

while (pNode->pNext != pStop && pNode->key != data) {

pPrev = pNode;

pNode = pNode->pNext;

}

if (pNode->pNext == pStop && pNode->key != data) {

string exp = "Error: key not found";

throw exp;

}

if (pCurr == pNode) Reset();

if (pPrev == nullptr) {

pFirst = pNode->pNext;

delete pNode;

return;

}

pPrev->pNext = pNode->pNext;

delete pNode;

}

template <typename T>

void TList<T>::Clear() {

TNode<T>\* tmp = pFirst;

while (pFirst != pStop) {

pFirst = tmp->pNext;

delete tmp;

tmp = pFirst;

}

pLast = pStop;

pCurr = pStop;

}

template <typename T>

size\_t TList<T>::GetSize() const {

size\_t size = 0;

TNode<T>\* tmp = pFirst;

while (tmp != pStop) {

tmp = tmp->pNext;

++size;

}

return size;

}

template <typename T>

bool TList<T>::IsFull() const {

TNode<T>\* tmp = new TNode<T>;

if (tmp != pStop) {

return false;

}

else {

return true;

}

}

template <typename T>

bool TList<T>::IsEmpty() const {

if (pFirst == pStop) return true;

else return false;

}

template <typename T>

void TList<T>::Reset() {

pCurr = pFirst;

}

template <typename T>

TNode<T>\* TList<T>::GetCurrent() const {

return pCurr;

}

template <typename T>

void TList<T>::Next(const int count) {

if (count <= 0) {

string exp = "Error: count can't be less than 0";

throw exp;

}

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

if (!IsEnded() || pCurr != pStop) pCurr = pCurr->pNext;

else Reset();

}

}

template <typename T>

bool TList<T>::IsEnded() const {

if (pCurr == pStop) return true;

//if (pCurr->pNext == pStop) return true;

else return false;

}

template <typename T>

TNode<T>\* TList<T>::first() const {

return pFirst;

}

template <typename T>

TNode<T>\* TList<T>::last() const {

return pLast;

}

template <typename T>

TNode<T>\* TList<T>::stop() const {

return pStop;

}

## Приложение В. Реализация класса TRingList

#include <iostream>

#include "tlist.h"

using namespace std;

template <typename T>

class TRingList : public TList<T> {

private:

TNode<T>\* pHead;

public:

TRingList();

TRingList(TNode<T>\* \_pFirst);

TRingList(const TRingList<T>& rlist);

virtual ~TRingList();

void InsertFirst(const T& data);

};

template <typename T>

TRingList<T>::TRingList() {

pHead = new TNode<T>();

pHead->pNext = pHead;

pStop = pHead;

pFirst = pHead; // nullptr

pLast = pHead;

pCurr = pHead;

} // !!! IF pFirst = nullptr (etc.), THEN WHOLE PROJECT BREAKS DOWN !!!

template <typename T>

TRingList<T>::TRingList(TNode<T>\* \_pFirst) : TList(\_pFirst) {

pHead = new TNode<T>();

pHead->pNext = pHead;

pStop = pHead;

if (pFirst == nullptr) {

pFirst = pHead; //nullptr

pLast = pHead;

pCurr = pHead;

}

else {

TNode<T>\* tmp = pFirst;

pHead->pNext = pFirst;

while (tmp->pNext != nullptr) {

tmp = tmp->pNext;

}

tmp->pNext = pHead;

pLast = tmp;

}

}

template <typename T>

TRingList<T>::TRingList(const TRingList<T>& rlist) {

pHead = new TNode<T>();

pHead->pNext = pHead;

pStop = pHead;

if (rlist.pFirst == rlist.pStop) { //nullptr

pFirst = pHead; // nullptr

pLast = pHead;

pCurr = pHead;

}

else {

pFirst = new TNode<T>(rlist.pFirst->key);

TNode<T>\* tmp = rlist.pFirst;

TNode<T>\* pNode = pFirst;

pHead->pNext = pFirst;

while (tmp->pNext != rlist.pStop) {

pNode->pNext = new TNode<T>(tmp->pNext->key);

pNode = pNode->pNext;

tmp = tmp->pNext;

}

pLast = pNode;

pLast->pNext = pStop;

pCurr = pFirst;

}

}

template <typename T>

TRingList<T>::~TRingList() {

if (pStop) delete pStop;

}

template <typename T>

void TRingList<T>::InsertFirst(const T& data) {

TNode<T>\* tmp = new TNode<T>(data, pFirst);

if (pFirst == pStop) pLast = tmp;

pFirst = tmp;

pHead->pNext = pFirst;

if (pCurr == pStop) Reset();

}

## Приложение Г. Реализация класса TMonom

#include <iostream>

#include <string>

using namespace std;

class TMonom{

private:

double coeff;

int degree;

public:

TMonom();

TMonom(const TMonom& monom);

TMonom(double \_coeff, int \_degree);

TMonom(double \_coeff, int degX, int degY, int degZ);

double get\_coeff() const;

int get\_degree() const;

int get\_degX() const;

int get\_degY() const;

int get\_degZ() const;

void set\_coeff(double \_coeff);

void set\_degree(int \_degree);

void set\_degree(int degX, int degY, int degZ);

double value(double x, double y, double z);

virtual bool operator<(const TMonom& monom) const;

virtual bool operator>(const TMonom& monom) const;

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

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

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

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

const TMonom& operator=(const TMonom& data);

TMonom operator\*(const TMonom& monom) const;

TMonom dif\_x() const;

TMonom dif\_y() const;

TMonom dif\_z() const;

string get\_string() const;

friend ostream& operator<<(ostream& out, const TMonom& monom);

};

TMonom::TMonom() {

coeff = 0;

degree = -1;

}

TMonom::TMonom(const TMonom& monom) {

coeff = monom.coeff;

degree = monom.degree;

}

TMonom::TMonom(double \_coeff, int \_degree) {

if (\_degree > 999 || \_degree < 0) {

string exp = "Error: degree must be in [0, 999]";

throw exp;

}

coeff = \_coeff;

degree = \_degree;

}

TMonom::TMonom(double \_coeff, int degX, int degY, int degZ) {

if (degX < 0 || degY < 0 || degZ < 0 || degX > 9 || degY > 9 || degZ > 9) {

string exp = "Error: x,y,z must be in [0, 9]";

throw exp;

}

coeff = \_coeff;

degree = 100 \* degX + 10 \* degY + degZ;

}

double TMonom::get\_coeff() const {

return coeff;

}

int TMonom::get\_degree() const {

return degree;

}

int TMonom::get\_degX() const {

int degX = degree / 100;

return degX;

}

int TMonom::get\_degY() const {

int degY = (degree / 10) % 10;

return degY;

}

int TMonom::get\_degZ() const {

int degZ = degree % 10;

return degZ;

}

void TMonom::set\_coeff(double \_coeff) {

coeff = \_coeff;

}

void TMonom::set\_degree(int \_degree) {

if (\_degree > 999 || \_degree < 0) {

string exp = "Error: degree must be in [0, 999]";

throw exp;

}

degree = \_degree;

}

void TMonom::set\_degree(int degX, int degY, int degZ) {

if (degX < 0 || degY < 0 || degZ < 0 || degX > 9 || degY > 9 || degZ > 9) {

string exp = "Error: degX,degY,degZ must be in [0, 9]";

throw exp;

}

degree = 100 \* degX + 10 \* degY + degZ;

}

double TMonom::value(double x, double y, double z) {

double resX = 1, resY = 1, resZ = 1;

int degZ = get\_degZ();

int degY = get\_degY();

int degX = get\_degX();

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

resZ \*= z;

}

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

resY \*= y;

}

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

resX \*= x;

}

return coeff \* resX \* resY \* resZ;

}

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

if (degree < monom.degree) return true;

return false;

}

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

if (degree > monom.degree) return true;

return false;

}

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

if (degree <= monom.degree) return true;

return false;

}

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

if (degree >= monom.degree) return true;

return false;

}

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

if (degree == monom.degree) return true;

return false;

}

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

return !(degree == monom.degree);

}

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

degree = monom.degree;

coeff = monom.coeff;

return (\*this);

}

TMonom TMonom::operator\*(const TMonom& monom) const {

int degZ1 = get\_degZ();

int degY1 = get\_degY();

int degX1 = get\_degX();

int degZ2 = monom.get\_degZ();

int degY2 = monom.get\_degY();

int degX2 = monom.get\_degX();

if (degZ1 + degZ2 > 9 || degY1 + degY2 > 9 || degX1 + degX2 > 9 ||

degZ1 + degZ2 < 0 || degY1 + degY2 < 0 || degX1 + degX2 < 0) {

string exp = "Error: res\_degrees must be in [0, 9]";

throw exp;

}

int res\_degree = degree + monom.degree;

double res\_coeff = coeff \* monom.coeff;

TMonom res(res\_coeff, res\_degree);

return res;

}

TMonom TMonom::dif\_x() const {

int degX = get\_degX();

if (degX == 0) {

TMonom res(0, 0);

return res;

}

else {

double \_coeff = coeff \* degX;

int \_degree = degree - 100;

TMonom res(\_coeff, \_degree);

return res;

}

}

TMonom TMonom::dif\_y() const {

int degY = get\_degY();

if (degY == 0) {

TMonom res(0, 0);

return res;

}

else {

double \_coeff = coeff \* degY;

int \_degree = degree - 10;

TMonom res(\_coeff, \_degree);

return res;

}

}

TMonom TMonom::dif\_z() const {

int degZ = get\_degZ();

if (degZ == 0) {

TMonom res(0, 0);

return res;

}

else {

double \_coeff = coeff \* degZ;

int \_degree = degree - 1;

TMonom res(\_coeff, \_degree);

return res;

}

}

string TMonom::get\_string() const {

int degZ = get\_degZ();

int degY = get\_degY();

int degX = get\_degX();

string res = "";

res += to\_string(coeff);

if (degX == 1) res += "\*x";

else if (degX > 1) res += "\*x^" + to\_string(degX);

if (degY == 1) res += "\*y";

else if (degY > 1) res += "\*y^" + to\_string(degY);

if (degZ == 1) res += "\*z";

else if (degZ > 1) res += "\*z^" + to\_string(degZ);

return res;

}

ostream& operator<<(ostream& out, const TMonom& monom) {

return out << monom.get\_string();

}

## Приложение Д. Реализация класса TPolynom

#include <iostream>

#include <string>

#include <map>

#include <vector>

#include "tringlist.h"

#include "tmonom.h"

using namespace std;

#define infix name

class TPolynom {

private:

TRingList<TMonom>\* monoms;

static map<string, int> priority;

void Parse(string& name);

void ToMonoms(vector<string>& lexems);

void AddMonom(const TMonom& m);

public:

TPolynom();

TPolynom(const string& \_name);

TPolynom(const TRingList<TMonom>& ringlist);

TPolynom(const TPolynom& polynom);

~TPolynom();

const TPolynom& operator=(const TPolynom& polynom);

const TPolynom operator+(const TPolynom& polynom) const;

const TPolynom operator-() const;

const TPolynom operator-(const TPolynom& polynom) const;

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

double operator()(double x, double y, double z) const;

TPolynom dif\_x() const;

TPolynom dif\_y() const;

TPolynom dif\_z() const;

string get\_string() const;

friend ostream& operator<<(ostream& out, const TPolynom& polynom);

private:

bool IsOperator(const string& isopr) const;

bool IsConst(const string& isopd) const;

bool IsVariable(const string& str) const;

int FindOperator(const string& name, int pos = 0) const;

void ConvertInfix(string& name);

void CorrectnessCheck(const string& name) const;

};

map<string, int> TPolynom::priority = {

{"^", 4},

{"\*", 3},

{"/", 3},

{"+", 2},

{"-", 2},

{"(", 1},

{")", 1}

};

bool TPolynom::IsOperator(const string& isopr) const { // opr : priority can't be char

bool flag = false;

for (const auto& opr : priority) {

if (isopr == opr.first) {

flag = true;

break;

}

}

return flag;

}

bool TPolynom::IsConst(const string& isopd) const {

bool flag = true;

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

if (isopd[i] < '0' || isopd[i] > '9') {

if (isopd[i] != '.')

flag = false;

break;

}

return flag;

}

bool TPolynom::IsVariable(const string& str) const {

return (str == "x" || str == "y" || str == "z");

}

// If found any operator, returns index. Else returns -1.

int TPolynom::FindOperator(const string& name, int pos) const {

if (pos < 0 || pos >= infix.size()) return -1;

int ind = -1;

for (int i = pos; i < infix.size(); i++) {

string isopr;

isopr += infix[i];

if (IsOperator(isopr)) {

ind = i;

break;

}

}

return ind;

}

void TPolynom::ConvertInfix(string& name) {

string nospaces;

for (int i = 0; i < infix.size(); i++) {

if (infix[i] != ' ')

nospaces += infix[i];

}

infix = nospaces;

string tmp;

if (infix[0] == '-') tmp += "0-";

else if (infix[0] == '.') tmp += "0.";

else if (infix[0] == '(') tmp += '(';

else tmp += infix[0];

for (int i = 1; i < infix.size() - 1; i++) {

char elem = infix[i];

if (elem == ' ') {

continue;

}

else if (elem == '-') {

if (infix[i - 1] == '(')

tmp += '0';

tmp += '-';

}

else if (elem == '(') {

if (infix[i - 1] == ')' || infix[i - 1] == '.' || (infix[i - 1] >= '0' && infix[i - 1] <= '9'))

tmp += '\*';

tmp += '(';

}

else if (elem == '.') {

if (infix[i - 1] < '0' || infix[i - 1] > '9') {

if (infix[i - 1] == ')')

tmp += '\*';

tmp += '0';

}

tmp += '.';

if (infix[i + 1] < '0' || infix[i + 1] > '9') {

tmp += '0';

if (infix[i + 1] == '(')

tmp += '\*';

}

}

else if (elem >= '0' && elem <= '9') {

if (infix[i - 1] == ')')

tmp += '\*';

tmp += elem;

}

else {

tmp += elem;

}

}

if (infix[infix.size() - 1] == '.') tmp += ".0";

else if (infix[infix.size() - 1] == ')') tmp += ')';

else tmp += infix[infix.size() - 1];

infix = tmp;

}

void TPolynom::CorrectnessCheck(const string& name) const {

int op\_bracket = 0;

int cl\_bracket = 0;

int dot = 0;

string exp = "Incorrect arithmetic expression";

if (infix[0] == '+' || infix[0] == '\*' || infix[0] == '/' ||

infix[0] == ')' || infix[0] == '^') throw exp;

if (infix[infix.size() - 1] == '+' || infix[infix.size() - 1] == '-' ||

infix[infix.size() - 1] == '\*' || infix[infix.size() - 1] == '/' ||

infix[infix.size() - 1] == '(' || infix[infix.size() - 1] == '^') throw exp;

if (infix[0] == '(') op\_bracket++;

else if (infix[0] == '.') dot++;

if (infix[infix.size() - 1] == ')') cl\_bracket++;

else if (infix[infix.size() - 1] == '.') dot++;

for (int i = 1; i < infix.size() - 1; i++) {

char elem = infix[i];

if (elem == '(')

op\_bracket++;

else if (elem == ')') {

if (infix[i - 1] == '(' || infix[i - 1] == '+' || infix[i - 1] == '-' ||

infix[i - 1] == '\*' || infix[i - 1] == '/' || infix[i - 1] == '^') throw exp;

cl\_bracket++;

}

else if (elem == '.')

dot++;

else if (elem == '+' || elem == '-' || elem == '\*' || elem == '/' || elem == '^') {

if (dot > 1) throw exp;

dot = 0;

if (infix[i - 1] == '+' || infix[i - 1] == '-' || infix[i - 1] == '\*' ||

infix[i - 1] == '/' || infix[i - 1] == '(' || infix[i - 1] == '^') throw exp;

}

}

if (op\_bracket != cl\_bracket) throw exp;

}

void TPolynom::Parse(string& name) {

ConvertInfix(name);

CorrectnessCheck(name);

vector<string> lexems;

int firstind, secondind;

string token;

firstind = FindOperator(name);

secondind = FindOperator(name, firstind + 1);

if (firstind == -1) {

lexems.push\_back(infix);

return;

}

if (firstind > 0) {

lexems.push\_back(infix.substr(0, firstind));

}

while (secondind != -1) {

string opr = infix.substr(firstind, 1);

string opd = infix.substr(firstind + 1, secondind - firstind - 1);

lexems.push\_back(opr);

if (opd != "")

lexems.push\_back(opd);

firstind = secondind;

secondind = FindOperator(name, firstind + 1);

}

lexems.push\_back(infix.substr(firstind, 1));

if (firstind != infix.size() - 1)

lexems.push\_back(infix.substr(firstind + 1));

ToMonoms(lexems);

}

// ======================================================= //

void TPolynom::ToMonoms(vector<string>& \_lexems) {

double coeff = 1;

int degX = 0, degY = 0, degZ = 0;

int next\_const\_sign = 1;

vector<string> lexems;

for (const string& token : \_lexems) {

if (token != "\*" && token != "^") {

lexems.push\_back(token);

}

}

for (int i = 0; i < lexems.size(); i++) {

if (lexems[i] == "+" || lexems[i] == "-") {

TMonom monom(next\_const\_sign \* coeff, degX, degY, degZ);

AddMonom(monom);

if (lexems[i] == "+") next\_const\_sign = 1;

else next\_const\_sign = -1;

coeff = 1;

degX = 0;

degY = 0;

degZ = 0;

}

else {

if (lexems[i] == "x") {

if (i < lexems.size() - 1) {

if (!IsVariable(lexems[i + 1]) && !IsOperator(lexems[i + 1])) {

degX += stoi(lexems[i + 1]);

i++;

}

else {

degX += 1;

}

}

else {

degX += 1;

}

}

else if (lexems[i] == "y") {

if (i < lexems.size() - 1) {

if (!IsVariable(lexems[i + 1]) && !IsOperator(lexems[i + 1])) {

degY += stoi(lexems[i + 1]);

i++;

}

else {

degY += 1;

}

}

else {

degY += 1;

}

}

else if (lexems[i] == "z") {

if (i < lexems.size() - 1) {

if (!IsVariable(lexems[i + 1]) && !IsOperator(lexems[i + 1])) {

degZ += stoi(lexems[i + 1]);

i++;

}

else {

degZ += 1;

}

}

else {

degZ += 1;

}

}

else {

if (!IsConst(lexems[i])) {

string exp = "Error: not valid operand";

throw exp;

}

coeff \*= stod(lexems[i]);

}

}

}

TMonom monom(next\_const\_sign \* coeff, degX, degY, degZ);

AddMonom(monom);

}

void TPolynom::AddMonom(const TMonom& m) {

if (m.get\_coeff() == 0) return;

TNode<TMonom>\* tmp = monoms->first();

while (tmp != monoms->stop()) {

if (m == tmp->key) {

double coeff = m.get\_coeff() + tmp->key.get\_coeff();

if (coeff == 0.0f) {

monoms->Remove(tmp->key);

return;

}

else {

tmp->key.set\_coeff(coeff);

return;

}

}

else if (m < tmp->key) {

monoms->InsertBefore(m, tmp->key);

return;

}

else {

tmp = tmp->pNext;

}

}

monoms->InsertLast(m);

}

TPolynom::TPolynom() {

monoms = new TRingList<TMonom>;

}

TPolynom::TPolynom(const string& \_name) {

monoms = new TRingList<TMonom>;

string name = \_name;

Parse(name);

}

TPolynom::TPolynom(const TRingList<TMonom>& ringlist) {

monoms = new TRingList<TMonom>(ringlist);

}

TPolynom::TPolynom(const TPolynom& polynom) {

monoms = new TRingList<TMonom>(\*polynom.monoms);

}

TPolynom::~TPolynom() {

if (monoms) delete monoms;

}

const TPolynom& TPolynom::operator=(TPolynom& polynom) {

if (this == &polynom) return (\*this);

if (monoms) delete monoms;

monoms = new TRingList<TMonom>(\*polynom.monoms);

return (\*this);

}

const TPolynom TPolynom::operator+(const TPolynom& polynom) const {

TPolynom res(\*this);

polynom.monoms->Reset();

TNode<TMonom>\* tmp\_p = polynom.monoms->first();

while (tmp\_p != polynom.monoms->stop()) {

TMonom curr\_monom = tmp\_p->key;

res.AddMonom(curr\_monom);

tmp\_p = tmp\_p->pNext;

}

return res;

}

const TPolynom TPolynom::operator-() const {

TPolynom res(\*this);

TNode<TMonom>\* tmp = res.monoms->first();

while (tmp != res.monoms->stop()) {

double coeff = (-1) \* tmp->key.get\_coeff();

tmp->key.set\_coeff(coeff);

tmp = tmp->pNext;

}

return res;

}

const TPolynom TPolynom::operator-(const TPolynom& polynom) const {

TPolynom res(-polynom + (\*this));

return res;

}

const TPolynom TPolynom::operator\*(const TPolynom& polynom) const {

TPolynom res;

TNode<TMonom>\* tmp = monoms->first();

while (tmp != monoms->stop()) {

TMonom curr1 = tmp->key;

TNode<TMonom>\* tmp\_p = polynom.monoms->first();

while (tmp\_p != polynom.monoms->stop()) {

TMonom curr2 = tmp\_p->key;

res.AddMonom(curr1 \* curr2);

tmp\_p = tmp\_p->pNext;

}

tmp = tmp->pNext;

}

return res;

}

double TPolynom::operator()(double x, double y, double z) const {

double res = 0;

TNode<TMonom>\* tmp = monoms->first();

while (tmp != monoms->stop()) {

res += tmp->key.value(x, y, z);

tmp = tmp->pNext;

}

return res;

}

TPolynom TPolynom::dif\_x() const {

TPolynom res;

TNode<TMonom>\* tmp = monoms->first();

while (tmp != monoms->stop()) {

TMonom monom = tmp->key.dif\_x();

res.AddMonom(monom);

tmp = tmp->pNext;

}

return res;

}

TPolynom TPolynom::dif\_y() const {

TPolynom res;

TNode<TMonom>\* tmp = monoms->first();

while (tmp != monoms->stop()) {

TMonom monom = tmp->key.dif\_y();

res.AddMonom(monom);

tmp = tmp->pNext;

}

return res;

}

TPolynom TPolynom::dif\_z() const {

TPolynom res;

TNode<TMonom>\* tmp = monoms->first();

while (tmp != monoms->stop()) {

TMonom monom = tmp->key.dif\_z();

res.AddMonom(monom);

tmp = tmp->pNext;

}

return res;

}

string TPolynom::get\_string() const {

if (monoms->GetSize() == 0) return "0.000000";

string res = "";

TNode<TMonom>\* tmp = monoms->first();

if (tmp == monoms->stop()) {

return res;

}

else {

res += tmp->key.get\_string();

tmp = tmp->pNext;

while (tmp != monoms->stop()) {

string monom = tmp->key.get\_string();

if (monom[0] == '-') res += monom;

else res += "+" + monom;

tmp = tmp->pNext;

}

}

return res;

}

ostream& operator<<(ostream& out, const TPolynom& monom) {

return out << monom.get\_string();

}