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:

**«АНАЛИТИЧЕСКИЕ ПРЕОБРАЗОВАНИЯ ПОЛИНОМОВ ОТ НЕСКОЛЬКИХ ПЕРЕМЕННЫХ»**

**Выполнил(а):** студент группы 3822Б1ФИ1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Суворов Д.И. /

Подпись

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

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

Подпись

Нижний Новгород  
2024

**Содержание**

[Введение 3](#_Toc167750767)

[1 Руководство пользователя 5](#_Toc167750768)

[1.1 Приложение для демонстрации работы полиномов 5](#_Toc167750769)

[2 Руководство программиста 7](#_Toc167750770)

[2.1 Описание программной алгоритмов 7](#_Toc167750771)

[2.1.1 Линейный односвязный список 7](#_Toc167750772)

[2.1.1 Кольцевой список с головой 9](#_Toc167750773)

[2.1.2 Полином 10](#_Toc167750774)

[2.2 Описание программной реализации 14](#_Toc167750775)

[2.2.1 Схема наследования классов 14](#_Toc167750776)

[2.2.2 Описание класса TNode 14](#_Toc167750777)

[2.2.3 Описание класса TList 15](#_Toc167750778)

[2.2.4 Описание класса THeadRingList 18](#_Toc167750779)

[2.2.5 Описание классаTMonom 19](#_Toc167750780)

[2.2.6 Описание класса TPolynom 20](#_Toc167750781)

[Заключение 24](#_Toc167750782)

[Список литературы 25](#_Toc167750783)

[Приложения 26](#_Toc167750784)

[Приложение А. Реализация класса TList 26](#_Toc167750785)

[Приложение Б. Реализация классаTHeadRingList 29](#_Toc167750786)

[Приложение В. Реализация класса TMonom 31](#_Toc167750787)

[Приложение Г. Реализация класса TPolynom 31](#_Toc167750788)

# Введение

Полином – алгебраическое выражение, состоящее из суммы мономов, каждый из которых является произведением константы (коэффициента) и n-ого количества неизвестных в произвольных степенях. Общий вид полинома можно записать так:

В рамках этой лабораторной работы мы будем рассматривать полиному от трёх неизвестных, а степень каждой неизвестной не будет превышать 9.

Полиномы активно используются в математике и её приложениях. Они используются для моделирования физических процессов, в инженерных расчетах, при решении дифференциальных уравнений и в теории чисел. Также они нашли широкое применение в программировании. Так, полиному используются в компьютерной графике полиномиальные функции применяются для построения кривых и поверхностей, для разработки алгоритмов численного анализа, обработки сигналов, криптографии и сжатия данных. Полиномы являются основой для методов интерполяции, аппроксимации функций и решения систем линейных уравнений. В данной лабораторной работе мы изучим основные операции над полиномами, такие как сложение, вычитание, умножение и деление, а также рассмотрим алгоритмы их реализации на языке программирования.

# **Постановка задачи**

Цель – реализовать структуры данных для хранения полиномов.

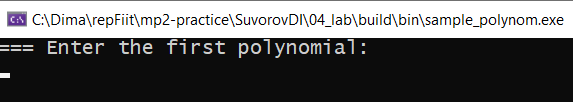
Задачи:

1. Внимательно изучить предметную область.
2. Реализовать класс THeadRingList.
3. Реализовать класс TPolynom.
4. Провести тестирование разработанных классов для проверки их корректной работы.
5. Сделать выводы о проделанной работе.

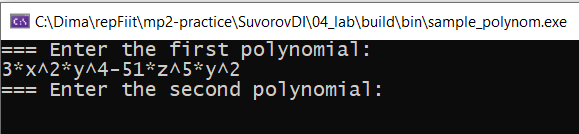
# Руководство пользователя

## Приложение для демонстрации работы полиномов

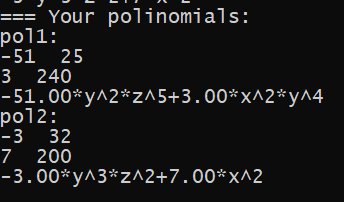
1. Запустите приложение с названием sample\_polynom.exe. В результате появится окно, показанное ниже, где вам нужно будет ввести первый полином (Символ умножения: «\*», степени: «^») (рис. 1).



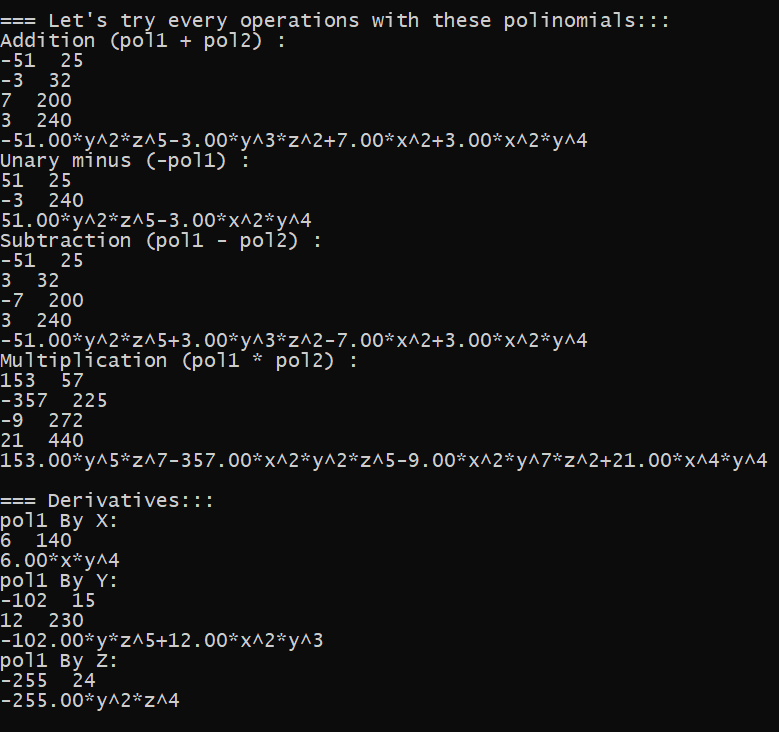
1. Основное окно программы
2. После ввода строкового представления первого полинома, по тем же правилам необходимо ввести второй полином (рис. 2).



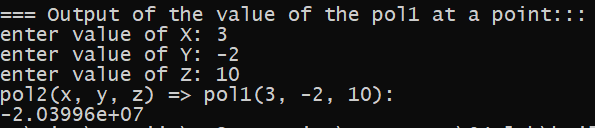
1. Результат работы программы
2. После ввода обоих полиномов на экран будут выведены введенные вами полиномы в виде перечисления коэффициентов и свертки каждого монома и строковое представление введенного полинома. (рис. 3).



1. Ввод второго полинома
2. Далее вам будет представлены результаты всех операций над двумя этими полиномами (рис. 4).



1. Результат работы программы
2. Далее вам будет предложено ввести значения неизвестных x, y, z для первого полинома и после ввода значений будет выведен результат вычисления значения первого полинома в точке с заданными координатами (рис. 5).



1. Результат работы программы

# Руководство программиста

## Описание программной алгоритмов

### Линейный односвязный список

Линейный односвязный список — динамическая структура данных, состоящая из узлов, содержащих данные и указатели на следующий и/или предыдущий узел списка.

Данный класс поддерживает следующие операции:

* **Операция добавления в начало**

1. Создаем узел, который будет являться головой списка.
2. Устанавливаем этот узел в качестве первого элемента списка.

Пример:

C

B

C

B

A

* **Операция добавления в конец**

1. Создаем новый узел, который будет указывать на конец списка.
2. Если список пустой (то есть нет других узлов), то новый узел становится началом списка.
3. Если список не пустой, начинаем с первого узла и движемся по списку до тех пор, пока не достигнем последнего узла.
4. Когда мы найдем текущий последний узел списка, делаем его указатель указывающим на новый узел, чтобы добавить новый узел в конец списка.

Пример:

B

A

C

B

A

**\**

* **Операция поиска**

1. Начать с первого узла списка и проверить текущий узел на наличие искомого элемента.
2. Если текущий узел содержит искомый элемент, завершить операцию успешно.
3. Если искомый элемент не найден в текущем узле, перейти к следующему узлу в списке (переход к следующему элементу).
4. Если достигнут конец списка (следующий узел равен nullptr), искомый элемент не найден в списке.
5. Возвращаем элемент или генерируем исключение.

* **Операция удаления элемента**

1. Используется функция поиска, чтобы найти элемент для удаления.
2. Если элемент найден, удаляем его из списка и обновляем указатели элементов.
3. Если элемент не найден при прохождении списка, сгенерируйте исключение или обработайте ситуацию отсутствия искомого элемента в списке.

Пример:

B

A

A

* **Операция очищения**

1. Последовательное удаление узлов.
2. Обновление указателей на начало и конец списка.

* **Операция вставки перед элементом**

1. Поиск узла с указанным значением.
2. Если найденный узел является первым узлом списка, вызывается метод вставки в начало.
3. Создание нового узла.
4. Если элемент не найден генерируем исключение.

Пример вставки перед 3:

C

A

C

B

A

* **Операция вставки после элемента**

1. Поиск узла с указанным значением.
2. Если найденный узел является последним узлом списка, вызывается метод вставки в конец.
3. Создание нового узла.
4. Если элемент не найден генерируем исключение.

Пример вставки после 1:

C

A

C

B

A

* **Операция получения текущего элемента**

Метод возвращает указатель на текущий узел списка. Этот узел используется внутри класса для отслеживания текущей позиции при выполнении операций со списком.

* **Операция получения размера листа**

Метод возвращает количество элементов в списке. Он проходит по всем узлам списка, увеличивая счетчик, пока не достигнет конца списка.

* **Операция вставки в сортированный список**

1. Проверка списка на пустоту или меньше ли элемент первого элемента. Если да, то вызывается метод вставки в начало и выход из функции.
2. Иначе проходимся по списку и ищем первый элемент, который больше переданного. Если такой нашелся, вставляем перед ним наш элемент.
3. Если список закончился, вызывается метод вставки в конец и выход из функции.
4. Иначе создается новый узел с значением.

Пример:

C

A

C

B

A

### Кольцевой список с головой

Кольцевой список с головой, или циклический список с головой, представляет собой структуру данных в виде связного списка, в котором последний элемент указывает на первый, образуя замкнутую цепочку связей. Такой подход обеспечивает удобство в работе с элементами списка и позволяет производить циклические операции без необходимости проверки на достижение конца списка.

Данный класс поддерживает все операции обыкновенного линейного односвязного листа, однако некоторые операции переопределяются:

* **Операция вставки в начало**

1. Вызывает метод вставки в начало из базового класса для вставки элемента.
2. Обновляется указатель на следующий элемент последнего узла на указатель на голову списка, а указатель на следующий элемент головы списка на первый узел.

Пример:

Head

3

2

3

2

1

Head

* **Операция удаления элемента**

1. Начинаем с первого узла и проверяем равно ли значение удаляемого элемента текущему узлу списка.
2. Если в списке один узел, удаляем его и устанавливаем все указатели в nullptr.
3. Если указанный элемент первый узел в списке, то удаляем его и сдвигаем указатель на голову и первый узел на следующий.
4. Иначе вызываем функцию удаления из базового класса.

Пример:

3

2

1

Head

2

Head

3

### Полином

В качестве структуры хранения для полинома будет использован кольцевой список с головой, который получается в результате преобразования входной строки в набор мономов. Полиномы состоят из мономов, которые имеют коэффициент и степень каждой из трёх неизвестных: x, y, x. Базовый набор операций для полинома:

* **Операция суммирования полиномов**
  1. Создаем новый пустой отсортированный кольцевой список для хранения суммированных мономов.
  2. Перебираем мономы из двух исходных списков, добавляя их в новый список с учетом сортировки.
  3. После завершения операции суммирования удаляем из нового списка все мономы с коэффициентом, равным нулю.

Пример:

Первый полином: x^3+y^2+y+x

Второй полином: x^3+y+x

Результат: 2x^3+y^2+2y+2x

Мономы первого полинома:

1|10

1|20

1|100

1|300

0|-1

Мономы второго полинома:

1|10

1|100

1|300

0|-1

Результат:

2|10

1|20

2|100

2|300

0|-1

* **Операция разности полиномов**

Операция состоит из двух этапов:

1. Умножение коэффициентов вычитаемого полинома на (-1).
2. Применение операции суммы к данным полиномам.

Пример:

Первый полином: x^3+y^2+y+x

Второй полином: x^3+y+z

Результат:x+y^2+-z

Мономы первого полинома:

1|10

1|20

1|100

1|300

0|-1

Мономы второго полинома:

1|10

1|100

1|300

0|-1

Результат:

-1|10

1|2-

1|100

0|-1

* **Операция произведения полиномов**
  1. Создаем новый пустой список для хранения умноженных мономов.
  2. Итерируем по мономам первого полинома.
  3. Внутри этого цикла итерируем также по мономам второго полинома.
  4. Умножаем каждую пару мономов и добавляем ненулевые результаты в новый список.
  5. Если в результате умножения нет ненулевых мономов, добавляем нулевой моном.
  6. После завершения операции умножения возвращаем новый полином.

Пример:

Первый полином: x^2+y

Второй полином:x^3+x

Результат:x^5+x^3+x^3\*y+x\*y

Мономы первого полинома:

1|200

0|-1

1|10

Мономы второго полинома:

0|-1

1|100

1|300

Результат:

0|-1

1|500

1|310

1|300

1|110

* **Операция дифференцирования полиномов**
  1. Создаем новый пустой список для хранения производных мономов.
  2. Итерируем по мономам в исходном полиноме.
  3. Для каждого монома уменьшаем его степень по указанной переменной и умножаем коэффициент на новую степень.
  4. Полученные мономы добавляем в новый список.
  5. После завершения операции по всем мономам, возвращаем новый полином, который является производной исходного по указанной переменной. Если в результате нет ненулевых мономов, добавляется нулевой моном.

Пример:

Полином: z^3+z^2+y+x

Результат после дифференцирования функции по z:3z^2+2z

Мономы полинома до операции:

0|-1

1|100

1|3

1|2

1|10

Результат:

3|2

0|-1

2|1

## Описание программной реализации

### Схема наследования классов

**TNode**

**TList**

**THeadRingList**

**TMonom**

**TPolynom**

1. Схема наследования классов

На рис. 6 показаны отношения между классами:

Класс TNode описывает узлы как линейного односвязного списка, так и кольцевого списка с головой.

Класс TList описывает линейный односвязный список.

Класс THeadRingList описывает кольцевой список с головой.

Класс TMonom описывает моном внутри полинома.

Класс TPolynom описывает полином, как кольцевой список с головой, содержащий мономы.

### Описание класса TNode

template <typename T>

struct TNode {

T data;

TNode<T>\* pNext;

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

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

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

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

};

Поля:

* data – данные, хранящиеся в звене.
* pNext – указатель на следующее звено.

Методы:

* TNode();

Назначение: конструктор по умолчанию.

Входные параметры отсутствуют:

Выходные параметры: отсутствуют.

* TNode(const T& data);

Назначение: конструктор с параметром.

Входные параметры: data– данные для хранения в звене.

Выходные параметры: отсутствуют.

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

Назначение: конструктор с параметром.

Входные параметры: \_pNext – указатель на следующее звено.

Выходные параметры: отсутствуют.

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

Назначение: конструктор с параметрами.

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

Выходные параметры: отсутствуют.

### Описание класса TList

template <typename T>

class TList {

protected:

TNode<T>\* pFirst;

TNode<T>\* pCurr;

TNode<T>\* pStop;

TNode<T>\* pLast;

TNode<T>\* pStop;

TNode<T>\* pPrev;

public:

TList();

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

TList(const TList<T>& list);

virtual ~TList();

virtual void InsertFirst(const T& data);

virtual void Remove(const T& data);

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

void InsertLast(const T& data);

void InsertAfter(const T& data, const T&beforedata);

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

TNode<T>\* GetCurr() const;

TNode<T>\* GetPrev() const;

void Clear();

bool IsEmpty() const;

bool IsFull() const;

bool IsEnded()const;

void Next();

void Reset();

};

Поля:

* pFirst– указатель на первый элемент.
* pStop – указатель на конец списка.
* pCurr – указатель на текущий элемент.
* pLast – указатель на последний элемент.
* pPrev – указатель на предыдущий элемент.
* pStop – указатель, обозначающий конец списка.

Методы:

* TList();

Назначение: создание пустого списка.

Входные параметры: отсутствуют.

Выходные параметры: новый объект класса TList.

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

Назначение: создание списка с заданным начальным узлом

Входные параметры: \_pFirst – указатель на первый узел списка.

Выходные параметры: новый объект класса TList.

* TList(const TList<T>& list);

Назначение: создание копии существующего списка.

Входные параметры: list – существующий список для копирования.

Выходные параметры: новый объект класса TList, являющийся копией списка list.

* virtual ~TList();

Назначение: освобождение памяти списка при удалении объекта.

Входные параметры: отсутствуют.

Выходные параметры: освобожденная память объекта класса TList.

* virtual void InsertFirst(const T& data);

Назначение: вставляет новый узел с данными в начало списка.

Входные параметры: data – данные для нового узла.

Выходные параметры: отсутствуют.

* virtual void Remove(constT&data);

Назначение: удаляет узел с определенными данными из списка.

Входные параметры: data – данные узла для удаления.

Выходные параметры: отсутствуют.

* void InsertBefore(const T& data, const T&nextdata);

Назначение: вставляет новый узел с данными перед узлом с определенными данными.

Входные параметры: data – данные для нового узла, nextdata – данные узла, перед которым будет вставлен новый узел.

Выходные параметры: отсутствуют.

* void InsertLast(constT&data);

Назначение: вставляет новый узел с данными в конец списка.

Входные параметры: data – данные для нового узла.

Выходные параметры: отсутствуют.

* void InsertAfter(const T& data, const T&beforedata);

Назначение: вставляет новый узел с данными после узла с определенными данными.

Входные параметры: data – данные для нового узла, beforedata – данные узла, после которого будет вставлен новый узел.

Выходные параметры: отсутствуют.

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

Назначение: поиск узла с указанным значением.

Входные параметры: data – искомое значение.

Выходные параметры: указатель на узел с заданным значением, либо nullptr.

* TNode<T>\* GetCurr();

Назначение: возвращает указатель на текущий узел.

Входные параметры: отсутствуют.

Выходные параметры: указатель на текущий узел.

* TNode<T>\* GetPrev();

Назначение: возвращает указатель на предыдущий узел.

Входные параметры: отсутствуют.

Выходные параметры: указатель на предыдущий узел.

* void Clear();

Назначение: очищает список, освобождает выделенную память.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

* bool IsEmpty() const;

Назначение: проверяет, пуст ли список.

Входные параметры: отсутствуют.

Выходные параметры: true – если список пуст, false – в противном случае.

* bool IsFull() const;

Назначение: проверяет, полон ли список.

Входные параметры: отсутствуют.

Выходные параметры: true – если список полон, false – в противном случае.

* bool IsEnded() const;

Назначение: проверяет, достигли ли конца списка.

Входные параметры: отсутствуют.

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

* void Next();

Назначение: переход к следующему узлу.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

* virtual void Reset();

Назначение: установка текущего узла как первого.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

### Описание класса THeadRingList

template <typename T>

class THeadRingList : public TList<T> {

private:

TNode<T>\* pHead;

public:

THeadRingList();

THeadRingList(const THeadRingList&ringL);

~THeadRingList();

const THeadRingList<TData>& operator=(const THeadRingList<TData>& l);

void RemoveFirst();

void InsertFirst(const TData& data);

void InsertLast(const TData& data);

bool IsEnded() const;

};

Поля:

* pHead – указатель на головной элемент.

Методы:

* THeadRingList();

Назначение: конструктор без параметров, создает пустой кольцевой список.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

* THeadRingList(const TRingList<T>& rlist);

Назначение: конструктор копирования, создает копию существующего кольцевого списка.

Входные параметры: rlist – ссылка на существующий кольцевой список.

Выходные параметры: отсутствуют.

* const THeadRingList<TData>& operator=(const THeadRingList<TData>& l);

Назначение: оператор присваивания, копирования значений из принимаемого объекта в текущий.

Входные параметры: l – ссылка на существующий кольцевой список.

Выходные параметры: отсутствуют.

* virtual ~THeadRingList();

Назначение: виртуальный деструктор, освобождает выделенную память при уничтожении объектов производных классов.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

* void InsertFirst(const T& data);

Назначение: вставляет новый узел с данными в начало списка.

Входные параметры: data – данные для нового узла.

Выходные параметры: отсутствуют.

* void InsertLast(const TData& data);

Назначение: вставляет новый узел с данными в конец списка.

Входные параметры: data – данные для нового узла.

Выходные параметры: отсутствуют.

* void RemoveFirst();

Назначение: Удаляет первый элемент в кольцевом списке.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

* bool IsEnded() const;

Назначение: проверяет, достигли ли конца списка.

Входные параметры: отсутствуют.

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

### Описание классаTMonom

class TMonom {

public:

double coeff;

int degree;

TMonom();

TMonom(const TMonom&monom);

TMonom(double \_coeff, int \_degree);

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

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

bool operator <(const TMonom& data)const;

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

bool operator >(const TMonom& data)const;

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

};

Поля:

* coeff– коэффициент монома.
* degree– степень монома.

Методы:

* Tmonom();

Назначение: конструктор по умолчанию, инициализирует объект TMonom с коэффициентом и степенью равными нулю.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

* TMonom(const TMonom&monom);

Назначение: конструктор копирования, создает копию существующего TMonom.

Входные параметры: monom – ссылка на существующий объект TMonom.

Выходные параметры: отсутствуют.

* TMonom(double \_coeff, int \_degree);

Назначение: создает объект TMonom с указанным коэффициентом и степенью.

Входные параметры: \_coeff – коэффициент, \_degree - степень.

Выходные параметры: отсутствуют.

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

Назначение: перегруженный оператор "равно". Проверяет равенство коэффициента и степени двух объектов TMonom.

Входные параметры: data – ссылка на объект TMonom для сравнения.

Выходные параметры: true, если объекты равны, иначе false.

* bool operator != (const TMonom& data) const;

Назначение: перегруженный оператор "не равно". Проверяет неравенство коэффициента и степени двух объектов TMonom.

Входные параметры: data – ссылка на объект TMonom для сравнения.

Выходные параметры: true, если объекты не равны, иначе false.

* bool operator < (const TMonom& data) const;

Назначение: перегруженный оператор "меньше". Сравнивает два объекта TMonom по убыванию степени.

Входные параметры: data – ссылка на объект TMonom для сравнения.

Выходные параметры: true, если текущий объект меньше, иначе false.

* bool operator <= (const TMonom& data) const;

Назначение: перегруженный оператор "меньше или равно". Сравнивает два объекта TMonom по убыванию степени.

Входные параметры: data – ссылка на объект TMonom для сравнения.

Выходные параметры: true, если текущий объект меньше, иначе false.

* bool operator > (const TMonom& data) const;

Назначение: перегруженный оператор "больше".

Входные параметры: data – ссылка на объект TMonom для сравнения.

Выходные параметры: true, если текущий объект больше, иначе false.

* bool operator >= (const TMonom& data) const;

Назначение: перегруженный оператор "больше или равно".

Входные параметры: data – ссылка на объект TMonom для сравнения.

Выходные параметры: true, если текущий объект больше или равен, иначе false.

### Описание класса TPolynom

class TPolynom {

private:

string name;

THeadRingList<TMonom>\* monoms;

void tokinize\_polynom(const std::string& name);

bool check\_each\_char\_is\_correct(const std::string& name);

void InsertToSort(const TMonom& monom);

public:

TPolynom();

TPolynom(const string& \_name);

TPolynom(const THeadRingList<TMonom>\* m);

TPolynom(const TPolynom& p);

~TPolynom();

TPolynom operator +(const TPolynom& p);

TPolynom operator -(const TPolynom& p);

TPolynom operator-() const;

TPolynom operator \*(const TPolynom& p);

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

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

TPolynom dx() const;

TPolynom dy() const;

TPolynom dz() const;

string ToString()const;

bool operator ==(const TPolynom&p) const;

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

};

Поля:

* name – строка полинома.
* monoms – кольцевой линейный односвязный список.

Методы:

* Tpolynom();

Назначение: конструктор по умолчанию, создает объект TPolynom с пустым списком мономов и пустым именем.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

* TPolynom(const std::string& name);

Назначение: создает объект TPolynom с указанным именем.

Входные параметры: name – строка, используемая в качестве имени полинома.

Выходные параметры: отсутствуют.

* TPolynom(const THeadRingList<TMonom>\* m);

Назначение: создает объект TPolynom на основе существующего кольцевого списка мономов.

Входные параметры: THeadRingList <TMonom>\* m – ссылка на кольцевой список мономов.

Выходные параметры: отсутствуют.

* TPolynom(const TPolynom& p);

Назначение: конструктор копирования, создает копию существующего объекта TPolynom.

Входные параметры: p – ссылка на существующий объект TPolynom.

Выходные параметры: отсутствуют.

* ~Tpolynom();

Назначение: деструктор, освобождает память при уничтожении объекта TPolynom.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

* void tokinize\_polynom(const std::string& name);

Назначение: разбирает строку, представляющую полином, и создает соответствующий список мономов.

Входные параметры: name – строка, представляющая полином.

Выходные параметры: отсутствуют.

* bool check\_each\_char\_is\_correct(const std::string& name);

Назначение: проверяет, что строка состоит только из символов x, y, z и цифр.

Входные параметры: name – ссылка проверяемую строку.

Выходные параметры: true, если строка состоит из допустимых символов. False – иначе.

* void InsertToSort(const TMonom& monom);

Назначение: вставка очередного монома в отсортированный список мономов.

Входные параметры: monom p – ссылка на объект TMonom, который необходимо вставить.

Выходные параметры: отсутствуют.

* TPolynom operator+(const TPolynom& p);

Назначение: перегруженный оператор сложения полиномов.

Входные параметры: p – ссылка на объект TPolynom.

Выходные параметры: объект TPolynom, который является результатом сложения полиномов.

* TPolynom operator-(const TPolynom& p);

Назначение: перегруженный оператор вычитания полиномов.

Входные параметры: p – ссылка на объект TPolynom.

Выходные параметры: объект TPolynom, который является результатом вычитания полиномов.

* TPolynom operator-();

Назначение: получение полинома с противоположными по знаку коэффициентами.

Входные параметры: отсутствуют.

Выходные параметры: объект TPolynom с противоположными знаками.

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

Назначение: перегруженный оператор умножения полиномов.

Входные параметры: p – ссылка на объект TPolynom.

Выходные параметры: объект TPolynom, который является результатом умножения полиномов.

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

Назначение: перегруженный оператор присваивания.

Входные параметры: p – ссылка на объект TPolynom, который присваивается текущему объекту.

Выходные параметры: копия объекта TPolynom после присваивания.

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

Назначение: вычисляет значение полинома для заданных значений переменных x, y и z.

Входные параметры: x, y, z.

Выходные параметры: значение полинома с заданными значениями переменных.

* TPolynom dx() const;

Назначение: возвращает производную по переменной «x» текущего полинома.

Входные параметры: отсутствуют.

Выходные параметры: полином – производная по «x».

* TPolynom dy() const;

Назначение: возвращает производную по переменной «y» текущего полинома.

Входные параметры: отсутствуют.

Выходные параметры: полином – производная по «y».

* TPolynom dz() const;

Назначение: возвращает производную по переменной «z» текущего полинома.

Входные параметры: отсутствуют.

Выходные параметры: полином – производная по «z».

* std::string ToString()const;

Назначение: возвращает строковое представление полинома.

Входные параметры: отсутствуют.

Выходные параметры: строковое представление полинома.

* friend std::ostream& operator<<(std::ostream& os, const TPolynom& p);

Назначение: оператор вывода для класса TPolynom.

Входные параметры:

os – ссылка на объект типа ostream, который представляет выходной поток.

p – ссылка на объект типа TPolynom который будет выводиться.

Выходные параметры: ссылка на объект типа ostream.

* bool operator == (const TPolynom&p) const;

Назначение: перегруженный оператор "равно". Проверяет равенство коэффициента и степени двух объектов TPolynom.

Входные параметры: p – ссылка на объект TPolynom для сравнения.

Выходные параметры: true, если объекты равны, иначе false.

# Заключение

В результате данной лабораторной работе удалось изучить и реализовать алгоритм обработки полиномов на основе односвязных списков. Была освоена работа с мономами и полиномами, а также разработали функционал для выполнения операций над ними: сложение, вычитание, умножение, и вычисление значений при заданных переменных. Это обеспечивает удобный и эффективный способ решения математических задач, требующих работы с полиномами.

# Список литературы

1. Кольцевой односвязный список [[Структуры данных в C# | Кольцевой односвязный список (metanit.com)](https://metanit.com/sharp/algoritm/2.7.php)].
2. Лекция «Списковые структуры хранения» Сысоев А. В. [<https://cloud.unn.ru/s/6g44ey6HFB4ncDy>].

# Приложения

# Приложение А. Реализация класса TList

template <typename TData>

TList<TData>::TList() {

pFirst = nullptr;

pLast = nullptr;

pCurr = nullptr;

pPrev = nullptr;

pStop = nullptr;

}

template <typename TData>

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

return pFirst == nullptr;

}

template <typename TData>

TList<TData>::TList(const TList& l) {

if (l.IsEmpty())

{

pFirst=nullptr;

pLast=nullptr;

pCurr=nullptr;

pPrev=nullptr;

pStop=nullptr;

return;

}

pFirst = new TNode<TData>(l.pFirst->data);

TNode<TData>\* tmp = pFirst;

TNode<TData>\* ltmp = l.pFirst->pNext;

while(ltmp != l.pStop)

{

tmp->pNext = new TNode<TData>(ltmp->data);

tmp = tmp->pNext;

ltmp = ltmp->pNext;

}

pLast = tmp;

pCurr = pFirst;

pPrev = nullptr;

pStop = nullptr;

}

template <typename TData>

TList<TData>::TList(TNode<TData>\* pNode) {

pFirst = pNode;

TNode<TData>\* tmp = pNode;

while (tmp->pNext != nullptr)

tmp = tmp->pNext;

pLast = tmp;

pCurr = pFirst;

pPrev = nullptr;

pStop = nullptr;

}

template <typename TData>

void TList<TData>::Clear() {

if (pFirst == nullptr)

return;

TNode<TData>\* curr = pFirst;

TNode<TData>\* next = pFirst->pNext;

while (next != pStop) {

delete curr;

curr = next;

next = curr->pNext;

}

delete curr;

pCurr = nullptr;

pFirst = nullptr;

pPrev = nullptr;

pLast = nullptr;

}

template <typename TData>

TList<TData>::~TList() {

Clear();

}

template <typename TData>

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

TNode<TData>\* tmp = new TNode<TData>();

if (tmp == nullptr)

return true;

delete tmp;

return false;

}

template <typename TData>

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

return pCurr == pStop;

}

template <typename TData>

TNode<TData>\* TList<TData>::Search(const TData& data) {

if (IsEmpty()) return nullptr;

TNode<TData>\* prev = nullptr;

TNode<TData>\* curr = pFirst;

while (curr != pStop && curr->data != data) {

prev = curr;

curr = curr->pNext;

}

if (curr == pStop) return nullptr;

pCurr = curr;

pPrev = prev;

return curr;

}

template <typename TData>

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

TNode<TData>\* new\_first = new TNode<TData>(data, pFirst);

pFirst = new\_first;

if (pLast == nullptr) {

pLast = pFirst;

}

pCurr = pFirst;

pPrev = nullptr;

}

template <typename TData>

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

if (IsEmpty()) {

InsertFirst(data);

return;

}

TNode<TData>\* new\_last = new TNode<TData>(data, pStop);

pLast->pNext = new\_last;

pPrev = pLast;

pLast = new\_last;

pCurr = new\_last;

}

template <typename TData>

void TList<TData>::InsertBefore(const TData& who, const TData& where) {

TNode<TData>\* pWhere = Search(where);

if (pWhere == nullptr) {

throw std::exception("no elements");

}

if (pWhere == pFirst) {

InsertFirst(who);

return;

}

TNode<TData>\* new\_node = new TNode<TData>(who, pWhere);

pPrev->pNext = new\_node;

pCurr = new\_node;

}

template <typename TData>

void TList<TData>::InsertAfter(const TData& who, const TData& where) {

TNode<TData>\* pWhere = Search(where);

if (pWhere == nullptr) {

throw std::exception("no elements");

}

if (pWhere == pLast) {

InsertLast(who);

return;

}

TNode<TData>\* new\_node = new TNode<TData>(who, pWhere->pNext);

pWhere->pNext = new\_node;

pCurr = new\_node;

pPrev = pWhere;

}

template <typename TData>

void TList<TData>::RemoveFirst() {

TNode<TData>\* first = pFirst;

pFirst = pFirst->pNext;

delete first;

}

template <typename TData>

void TList<TData>::Remove(const TData& where)

{

TNode<TData>\* pWhere = Search(where);

if (pWhere == nullptr) {

throw std::exception("no elements");

}

if (pWhere == pFirst && pWhere->pNext == pStop)

{

Clear();

return;

}

if (pWhere == pFirst)

{

RemoveFirst();

pCurr = pFirst;

return;

}

if (pWhere == pLast)

{

pPrev->pNext = pStop;

pLast = pPrev;

pCurr = pStop;

delete pWhere;

return;

}

pPrev->pNext = pWhere->pNext;

pCurr = pWhere->pNext;

delete pWhere;

}

template <typename TData>

void TList<TData>::Reset() {

pCurr = pFirst;

pPrev = nullptr;

}

template <typename TData>

void TList<TData>::Next() {

if (pCurr == pStop)

throw std::exception("end of the list");

pPrev = pCurr;

pCurr = pCurr->pNext;

}

template <typename TData>

TNode<TData>\* TList<TData>::GetCurr() const {

return pCurr;

}

template <typename TData>

TNode<TData>\* TList<TData>::GetPrev() const {

return pPrev;

}

## Приложение Б. Реализация классаTHeadRingList

template <typename TData>

THeadRingList<TData>::THeadRingList() : TList<TData>() {

pHead = new TNode<TData>();

pStop = pHead;

}

template <typename TData>

THeadRingList<TData>::THeadRingList(const THeadRingList<TData>& ringL) : TList<TData>(ringL) {

pHead = new TNode<TData>(ringL.pHead->data, pFirst);

if (!ringL.IsEmpty())

pLast->pNext = pHead;

pStop = pHead;

}

template <typename TData>

THeadRingList<TData>::~THeadRingList() {

delete pHead;

}

template <typename TData>

void THeadRingList<TData>::InsertFirst(const TData& data) {

TList<TData>::InsertFirst(data);

pHead->pNext = pFirst;

pStop = pHead;

pLast->pNext = pHead;

}

template <typename TData>

void THeadRingList<TData>::InsertLast(const TData& data) {

if (IsEmpty()) {

THeadRingList<TData>::InsertFirst(data);

return;

}

TList<TData>::InsertLast(data);

}

template <typename TData>

void THeadRingList<TData>::RemoveFirst() {

TNode<TData>\* first = pFirst;

pFirst = pFirst->pNext;

pHead->pNext = pFirst;

delete first;

}

template <typename TData>

const THeadRingList<TData>& THeadRingList<TData>::operator=(const THeadRingList<TData>& l) {

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

if (l.IsEmpty())

{

pFirst=nullptr;

pLast=nullptr;

pCurr=nullptr;

pPrev=nullptr;

pStop=nullptr;

pHead = new TNode<TData>();

pHead->pNext = pHead;

return \*(this);

}

Clear();

pHead = new TNode<TData>();

pHead->pNext = pFirst;

pFirst = new TNode<TData>(l.pFirst->data);

TNode<TData>\* tmp = pFirst;

TNode<TData>\* ltmp = l.pFirst->pNext;

while(ltmp != l.pStop)

{

tmp->pNext = new TNode<TData>(ltmp->data);

tmp = tmp->pNext;

ltmp = ltmp->pNext;

}

pLast = tmp;

pCurr = pFirst;

pPrev = nullptr;

pStop = nullptr;

return \*(this);

}

template <typename TData>

bool THeadRingList<TData>::IsEnded() const {

if (IsEmpty())

return true;

return pCurr == pStop;

}

## Приложение В. Реализация класса TMonom

TMonom::TMonom() {

coeff\_ = 0;

degree\_ = -1;

}

TMonom::TMonom(double coeff, int16\_t degree) {

if (degree > 999 || degree < 0) throw std::exception("BadDegree!\n");

coeff\_ = coeff;

degree\_ = degree;

}

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

return degree\_ < data.degree\_;

}

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

return degree\_ <= data.degree\_;

}

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

return degree\_ > data.degree\_;

}

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

return degree\_ >= data.degree\_;

}

// Для приведения подобных

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

return degree\_ == data.degree\_;

}

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

return degree\_ != data.degree\_;

}

## Приложение Г. Реализация класса TPolynom

TPolynom::TPolynom(const std::string& name) : monoms() {

this->name = name;

bool res = check\_each\_char\_is\_correct(name);

tokinize\_polynom(name);

}

TPolynom::TPolynom(const THeadRingList<TMonom>& l) {

if (l.IsEmpty()) {

name = "";

monoms = THeadRingList<TMonom>();

return;

}

THeadRingList<TMonom> tmp\_l(l);

tmp\_l.Reset();

while (!tmp\_l.IsEnded()) {

InsertToSort(tmp\_l.GetCurr()->data);

tmp\_l.Next();

}

if (monoms.IsEmpty()) {

TMonom zero(0, 0);

monoms.InsertFirst(zero);

}

name = ToString();

}

TPolynom::TPolynom(const TPolynom& p) : monoms(p.monoms), name(p.name) {}

bool TPolynom::check\_each\_char\_is\_correct(const std::string& name) {

std::string correct = "0123456789xyz\*^+-";

for (char ch : name) {

if (correct.find(ch) == std::string::npos) return false;

}

return true;

}

void TPolynom::InsertToSort(const TMonom& monom) {

monoms.Reset();

if (monom.coeff\_ == 0 && monom.degree\_ != 0) { return; }

if (monom.coeff\_ == 0 && monom.degree\_ == 0 && !monoms.IsEmpty()) { return; }

if (monoms.IsEmpty() || monoms.GetCurr()->data > monom) {

monoms.InsertFirst(monom);

return;

}

if (monoms.GetCurr()->data.coeff\_ == 0 && monoms.GetCurr()->data.degree\_ == 0) {

monoms.Remove(monoms.GetCurr()->data);

}

while (!monoms.IsEnded() && monoms.GetCurr()->data < monom) {

monoms.Next();

}

if (monoms.IsEnded()) {

monoms.InsertLast(monom);

return;

}

if (monoms.GetCurr()->data == monom) {

monoms.GetCurr()->data.coeff\_ = monoms.GetCurr()->data.coeff\_ + monom.coeff\_;

if (monoms.GetCurr()->data.coeff\_ == 0) {

monoms.Remove(monoms.GetCurr()->data);

}

return;

}

monoms.InsertBefore(monom, monoms.GetCurr()->data);

}

std::string TPolynom::ToString() const {

std::string str;

TPolynom new\_this(\*this);

if (new\_this.monoms.IsEmpty()) {

return "EMPTY!";

}

bool firstTerm = true;

new\_this.monoms.Reset();

if (new\_this.monoms.GetCurr()->data.coeff\_ == 0

&&

new\_this.monoms.GetCurr()->data.degree\_ == 0

) {

return "0";

}

while (!new\_this.monoms.IsEnded()) {

int deg = new\_this.monoms.GetCurr()->data.degree\_;

double coeff = new\_this.monoms.GetCurr()->data.coeff\_;

int x = deg / 100;

int y = (deg % 100) / 10;

int z = deg % 10;

if (coeff != 0) {

if (!firstTerm) {

str += ((coeff > 0) ? "+" : "-");

}

else {

if (coeff < 0) str += '-';

firstTerm = false;

}

if (abs(coeff) != 1 || deg == 0) {

char buf[15];

sprintf(buf, "%.2f", abs(coeff));

str += std::string(buf);

}

std::string mul\_symbol = ((abs(coeff) == 1) ? "" : "\*");

if (x != 0) {

str += (mul\_symbol + "x") + ((x != 1) ? "^" + std::to\_string(x) : "");

}

if (y != 0) {

mul\_symbol = (x == 0) ? mul\_symbol : "\*";

str += (mul\_symbol + "y") + ((y != 1) ? "^" + std::to\_string(y) : "");

}

if (z != 0) {

mul\_symbol = (x == 0 && y == 0) ? mul\_symbol : "\*";

str += (mul\_symbol + "z") + ((z != 1) ? "^" + std::to\_string(z) : "");

}

}

new\_this.monoms.Next();

}

return str;

}

TPolynom TPolynom::operator-() const {

TPolynom negativePol(\*this);

while (!negativePol.monoms.IsEnded()) {

negativePol.monoms.GetCurr()->data.coeff\_ = negativePol.monoms.GetCurr()->data.coeff\_ \* (-1);

negativePol.monoms.Next();

}

negativePol.name = negativePol.ToString();

return negativePol;

}

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

TPolynom res(p);

monoms.Reset();

while (!monoms.IsEnded()) {

res.InsertToSort(monoms.GetCurr()->data);

monoms.Next();

}

if (res.monoms.IsEmpty()) {

TMonom zero(0, 0);

res.monoms.InsertFirst(zero);

}

res.name = res.ToString();

return res;

}

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

TPolynom res = (\*this) + (-p);

res.name = res.ToString();

return res;

}

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

TPolynom res\_pol;

TPolynom tmp\_p(p);

monoms.Reset();

while (!monoms.IsEnded()) {

tmp\_p.monoms.Reset();

while (!tmp\_p.monoms.IsEnded()) {

TMonom mon1 = monoms.GetCurr()->data;

TMonom mon2 = tmp\_p.monoms.GetCurr()->data;

double newCoeff = mon1.coeff\_ \* mon2.coeff\_;

int16\_t newDegree = mon1.degree\_ + mon2.degree\_;

if (newDegree > 999) throw std::exception("invalid\_degree");

res\_pol.InsertToSort(TMonom(newCoeff, newDegree));

tmp\_p.monoms.Next();

}

monoms.Next();

}

if (res\_pol.monoms.IsEmpty()) {

TMonom zero(0, 0);

res\_pol.monoms.InsertFirst(zero);

}

res\_pol.name = res\_pol.ToString();

return res\_pol;

}

bool TPolynom::operator==(const TPolynom& p) const {

TPolynom new\_this(\*this);

TPolynom new\_p(p);

while (!new\_p.monoms.IsEnded() && !new\_this.monoms.IsEnded()) {

if (new\_p.monoms.GetCurr()->data.coeff\_ != new\_this.monoms.GetCurr()->data.coeff\_

|| new\_p.monoms.GetCurr()->data.degree\_ != new\_this.monoms.GetCurr()->data.degree\_) return false;

new\_p.monoms.Next();

new\_this.monoms.Next();

}

if (new\_p.monoms.IsEnded() && new\_this.monoms.IsEnded()) return true;

return false;

}

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

double result = 0;

TPolynom tmp\_this(\*this);

while (!tmp\_this.monoms.IsEnded()) {

double mn;

mn = tmp\_this.monoms.GetCurr()->data.coeff\_;

mn \*= pow(x, tmp\_this.monoms.GetCurr()->data.degree\_ / 100);

mn \*= pow(y, tmp\_this.monoms.GetCurr()->data.degree\_ / 10 % 10);

mn \*= pow(z, tmp\_this.monoms.GetCurr()->data.degree\_ % 10);

result += mn;

tmp\_this.monoms.Next();

}

return result;

}

TPolynom TPolynom::dx() const {

TPolynom dx\_pol;

TPolynom tmp\_this(\*this);

while (!tmp\_this.monoms.IsEnded()) {

if (tmp\_this.monoms.GetCurr()->data.degree\_ / 100 != 0) {

double newCoeff = tmp\_this.monoms.GetCurr()->data.coeff\_ \* (tmp\_this.monoms.GetCurr()->data.degree\_ / 100);

int16\_t newDegree = tmp\_this.monoms.GetCurr()->data.degree\_ - 100;

dx\_pol.InsertToSort(TMonom(newCoeff, newDegree));

}

tmp\_this.monoms.Next();

}

if (dx\_pol.monoms.IsEmpty()) {

dx\_pol = TPolynom("0");

}

return dx\_pol;

}

TPolynom TPolynom::dy() const {

TPolynom dy\_pol;

TPolynom tmp\_this(\*this);

while (!tmp\_this.monoms.IsEnded()) {

if (tmp\_this.monoms.GetCurr()->data.degree\_ / 10 % 10 != 0) {

double newCoeff = tmp\_this.monoms.GetCurr()->data.coeff\_ \* (tmp\_this.monoms.GetCurr()->data.degree\_ / 10 % 10);

int16\_t newDegree = tmp\_this.monoms.GetCurr()->data.degree\_ - 10;

dy\_pol.InsertToSort(TMonom(newCoeff, newDegree));

}

tmp\_this.monoms.Next();

}

if (dy\_pol.monoms.IsEmpty()) {

dy\_pol = TPolynom("0");

}

return dy\_pol;

}

TPolynom TPolynom::dz() const {

TPolynom dz\_pol;

TPolynom tmp\_this(\*this);

while (!tmp\_this.monoms.IsEnded()) {

if (tmp\_this.monoms.GetCurr()->data.degree\_ % 10 != 0) {

double newCoeff = tmp\_this.monoms.GetCurr()->data.coeff\_ \* (tmp\_this.monoms.GetCurr()->data.degree\_ % 10);

int16\_t newDegree = tmp\_this.monoms.GetCurr()->data.degree\_ - 1;

dz\_pol.InsertToSort(TMonom(newCoeff, newDegree));

}

tmp\_this.monoms.Next();

}

if (dz\_pol.monoms.IsEmpty()) {

dz\_pol = TPolynom("0");

}

return dz\_pol;

}

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

if (this == &p) {

return (\*this);

}

name = p.name;

monoms = p.monoms;

return \*(this);

}

void TPolynom::tokinize\_polynom(const std::string& name) {

std::string str = name;

while (!str.empty()) {

int degree = 0;

size\_t j = str.find\_first\_of("+-", 1);

std::string monom = str.substr(0, j);

if (monom[monom.length() - 1] == '^') {

throw std::exception("Negative degree");

}

str.erase(0, j);

std::string coefficent = monom.substr(0, monom.find\_first\_of("xyz"));

TMonom tmp;

tmp.coeff\_ = (coefficent == "" || coefficent == "+") ? 1 : (coefficent == "-") ? -1 : stod(coefficent);

monom.erase(0, monom.find\_first\_of("xyz"));

for (size\_t i = 0; i < monom.size(); ++i) {

if (isalpha(monom[i])) {

int exp = 1;

if (monom[i + 1] == '^') {

size\_t exp\_start = i + 2;

while (isdigit(monom[exp\_start])) {

exp\_start++;

}

exp = stoi(monom.substr(i + 2, exp\_start - i - 2));

}

switch (monom[i]) {

case 'x':

degree += exp \* 100;

break;

case 'y':

degree += exp \* 10;

break;

case 'z':

degree += exp \* 1;

break;

default:

throw ("exp");

break;

}

}

}

tmp.degree\_ = degree;

if (tmp.coeff\_ != 0) {

this->InsertToSort(tmp);

}

}

if (this->monoms.IsEmpty()) {

TMonom zero(0, 0);

this->monoms.InsertFirst(zero);

}

this->name = this->ToString();

}