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) группы3822Б1ФИ1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Лысов И.М.

Подпись

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

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

Подпись

Нижний Новгород  
2024

Оглавление

[Введение 3](#_Toc167182127)

[1 Постановка задачи 4](#_Toc167182128)

[2 Руководство пользователя 5](#_Toc167182129)

[2.1 Приложение для демонстрации работы программы 5](#_Toc167182130)

[3 Руководство программиста 7](#_Toc167182131)

[3.1 Описание алгоритмов 7](#_Toc167182132)

[3.1.1 Линейный односвязный список 7](#_Toc167182133)

[3.1.2.Кольцевой список с головой 9](#_Toc167182134)

[3.1.3.Полином 11](#_Toc167182135)

[3.2 Описание программной реализации 14](#_Toc167182136)

[3.2.1 Схема наследования классов 14](#_Toc167182137)

[3.2.2 Описание класса TNode 14](#_Toc167182138)

[3.2.3 Описание класса TList 15](#_Toc167182139)

[3.2.4 Описание класса THeadRingList 18](#_Toc167182140)

[3.2.5 Описание класса TMonom 20](#_Toc167182141)

[3.2.6 Описание класса TPolynom 22](#_Toc167182142)

[Заключение 26](#_Toc167182143)

[Литература 27](#_Toc167182144)

[Приложения 28](#_Toc167182145)

[Приложение А. Реализация класса TList 28](#_Toc167182146)

[Приложение Б. Реализация классаTHeadRingList 32](#_Toc167182147)

[Приложение В. Реализация класса TMonom 33](#_Toc167182148)

[Приложение Г. Реализация класса TPolynom 34](#_Toc167182149)

# Введение

Полиномы — это математические выражения, которые представляют собой сумму или разность членов, каждый из которых состоит из переменной (или переменных), возведенной в некоторую степень и умноженной на коэффициент. Они широко применяются в различных областях математики и ее приложениях, таких как алгебра, анализ, теория чисел, физика и инженерия.

Полиномы играют ключевую роль в алгебре, где они используются для решения уравнений и изучения свойств функций. Они также находят применение в анализе данных, где они используются для моделирования и предсказания различных явлений. Благодаря своей простоте и универсальности, полиномы являются важным инструментом, как для теоретических исследований, так и для практических применений.

Одним из основных преимуществ полиномов является то, что они обладают простой структурой, что упрощает их анализ и обработку. Кроме того, производные и интегралы полиномов могут быть легко вычислены, что делает их незаменимым инструментом при решении задач дифференциального и интегрального исчислений. Таким образом, полиномы являются важным инструментом, как для теоретических исследований, так и для практического применения в различных областях науки и техники.

# 1 Постановка задачи

**Цель** – реализовать работу с полиномами с помощью кольцевого линейного односвязного списка THeadRingList.

**Задачи:**

1. Исследовать тематическую литературу.
2. Реализовать линейный односвязный список list.
3. Реализовать кольцевой линейный односвязный список THeadRingList.
4. Реализовать класс TPolynom, реализующий основные операции над полиномами.
5. Проверить работу созданных классов с помощью тестирования, чтобы убедиться в их правильности и эффективности.
6. Сделать выводы о проделанной работе.

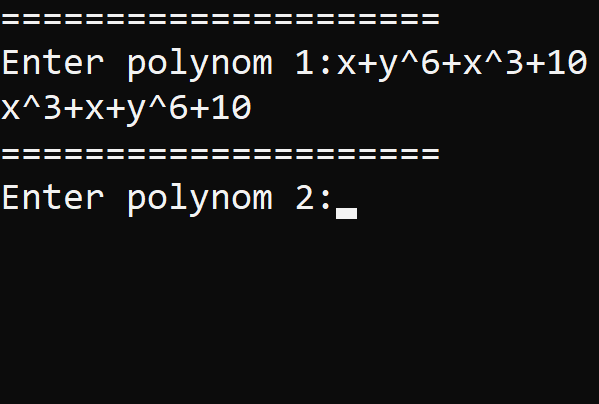
# 2 Руководство пользователя

## 2.1 Приложение для демонстрации работы программы

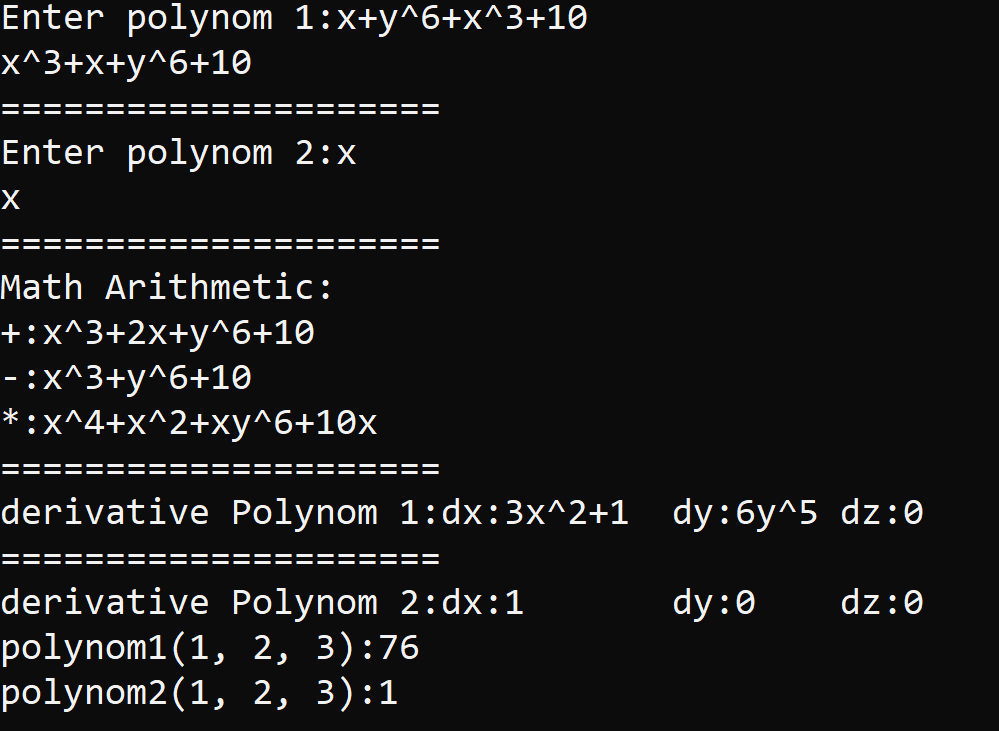
1. Запустить приложение sample\_polynom.exe. После этого запуститься программа, показывающая работу программы. После этого необходимо ввести первый полином, с которым будут применяться операции (рис. 1).



1. Запуск программы и ввод первого полинома
2. После нажатия клавиши Enter программа выведет первый полином, затем необходимо будет ввести второй полином (рис. 2).



1. Вывод первого полинома и ввод второго полинома
2. После ввода и нажатия клавиши Enter, программа выведет второй полином и операции над полиномами. На экране будут выведены операции: сложение полиномов, вычитание полиномов, умножение полиномов, производные по каждой переменной (x,y,z) для каждого из полиномов, подсчет значения полинома в конкретной точке (рис. 3).



1. Результат работы программы

# 3 Руководство программиста

# 3.1 Описание алгоритмов

### 3.1.1 Линейный односвязный список

Линейный односвязный список представляет собой структуру данных, в которой элементы хранятся последовательно в памяти и связаны друг с другом указателями. Каждый элемент списка, называемый узлом, содержит данные и указатель на следующий элемент. Односвязные списки обеспечивают эффективное добавление и удаление элементов в начале или середине списка, но доступ к элементам осуществляется только последовательно, начиная с головы списка.

Одним из главных преимуществ односвязных списков является их динамическая природа, которая позволяет эффективно использовать память, поскольку размер списка может изменяться в процессе выполнения программы. Однако, из-за необходимости последовательного обхода списка для доступа к элементам, поиск элемента по значению может занять значительное время.

* Операция добавления элемента в начало списка.

1. Создаем узел, который будет являться новым первым элементом списка.
2. Устанавливаем указатель нового узла на текущий первый элемент списка.
3. Устанавливаем этот новый узел в качестве первого элемента списка.

Пример:

1

2

Пп1

3

1

2

* **Операция добавления элемента в конец списка**

1. Создание нового узла, который будет добавлен в конец списка.
   1. В случае пустого списка, новый узел становится началом списка.
   2. Если список не пустой, проходим по всем узлам, начиная с первого, до последнего узла.
2. После нахождения текущего последнего узла списка, его указатель перенаправляется на новый узел, чтобы добавить его в конец списка.

Пример:

1

3

3

1

4

* **Операция поиска элемента в односвязном списке**

1. Для поиска элемента в односвязном списке начинаем с первого узла и проверяем каждый узел на наличие искомого элемента.
   1. Если элемент найден в текущем узле, завершаем операцию успешно.
   2. Если элемент не найден, переходим к следующему узлу и повторяем процесс.
   3. Если достигнут конец списка, значит искомый элемент отсутствует.
2. Возвращаем найденный элемент или генерируем исключение, если элемент не найден в списке.

Пример:

Поиск элемента 3. В результате будет возвращен указатель на элемент 3.

2

1

3

Пример:

Поиск элемента 5. В результате будет возвращен указатель на nullptr.

4

1

3

* **Операция удаления элемента из списка.**

1. Для удаления элемента из списка сначала используется функция поиска, чтобы найти нужный элемент.
2. Если элемент найден, он удаляется из списка, и обновляются указатели на элементы.
3. Если элемент не обнаружен при прохождении списка, может быть сгенерировано исключение или обработана ситуация отсутствия искомого элемента в списке.

Пример:

2

1

3

2

3

* **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 списка**

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

* **Операция вставки в отсортированный список**

1. Проверка списка на пустоту или на то, меньше ли элемент первого элемента списка. Если это так, вызывается метод вставки в начало, и выполнение функции прекращается.
2. В противном случае производится обход списка в поиске нужного элемента.
3. Если обход списка завершился, вызывается метод вставки в конец, и выполнение функции прекращается.
4. В противном случае создается новый узел с указанным значением и вставляется в соответствующую позицию.

1

3

1

2

3

### Кольцевой список с головой

Кольцевые списки с головой представляют собой структуру данных, в которой последний элемент соединяется с первым, образуя замкнутую цепочку. При этом существует специальный узел - голова, который не содержит данных и служит для упрощения операций с элементами списка.

Голова списка всегда указывает на первый элемент, а указатель последнего элемента указывает на голову. В кольцевом списке реализуются операции вставки в начало, удаления элемента, очистки.

* **Операция вставки в начало** 
  1. Вызывается метод вставки в начало из базового класса для добавления нового элемента.
  2. Обновляется указатель на следующий элемент последнего узла, чтобы он указывал на голову списка, а указатель на следующий элемент головы списка - на первый узел.

 22

2

3

3

2

1

* **Операция удаления элемента**

1. Удаление элемента начинается с проверки, равен ли удаляемый элемент текущему узлу списка.
2. Если в списке находится только один узел, он удаляется, и все указатели устанавливаются в nullptr.
3. Если удаляемый элемент является первым узлом в списке, он удаляется, а указатель на голову и первый узел смещаются на следующий.
4. В противном случае вызывается функция удаления из базового класса.

3

2

1

 22

2

3

* **Операция очистки кольцевого списка**

1. Начинаем с вызова метода Clear() из базового класса TList<T>, который отвечает за очистку списка.
2. После очистки основного списка обновляем указатель pNext последнего узла, чтобы он указывал на голову списка.
3. Таким образом, последний узел становится связанным с первым, образуя замкнутую структуру кольцевого списка.

### 3.1.3. Полином

Полином, как структура данных, составляется из кольцевого списка мономов, которые формируются при преобразовании входной строки. Мономы в полиноме определяются их коэффициентами и степенями.

Этот класс поддерживает следующие операции:

* Сложение полиномов:

Первый полином: x3 + y2 + z + 2.

Второй полином: x + y2 + z + 3.

Создается пустой кольцевой список list.

Пока не достигнут конец одного из полиномов берутся текущие мономы из обоих полиномов. Если мономы равны, их коэффициенты складываются. Результат суммы добавляется в список list.

Оставшиеся мономы из обоих полиномов добавляются в список list.

Создается новый полином result с мономами из списка list.

Имя нового полинома приводится к строке

Новый полином result возвращается в качестве результата операции сложения.

Пример:

5|0

2|2

2|20

1|100

1|300

Результат сложения:

3|0

1|1

1|20

1|100

2|0

1|1

1|20

1|300

* **Операция разности полиномов**

Операция состоит из двух этапов:

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

Первый полином: x3 + y2 + z + 2.

Второй полином: x + y2 + z + 3.

Пример:

Результат вычитания:

11

1|300

1|20

1|1

2|0

1|100

1|20

1|1

3|0

11

1|300

1|100

-1|0

* Операция умножения полиномов

Для того чтобы умножить полиномы необходимо:

Создается пустой кольцевой список list.

Пока не достигнут конец одного из полиномов берутся текущие мономы из обоих полиномов и перемножаются. Если мономы равны, их степени складываются. Результат суммы добавляется в список list.

Создается новый полином result с мономами из списка list.

Имя нового полинома приводится к строке

Новый полином result возвращается в качестве результата операции сложения.

Первый полином: x + y.

Второй полином: x + z.

Пример:

Результат умножения полиномов

1|100

1|10

1|100

1|1

1|200

1|101

1|110

1|11

* Операция дифференцирования полиномов:

Для того чтобы получить производную от полинома необходимо:

1. Создать пустой полином, который будет хранить результат производной операции.
2. Перебирать каждый моном в исходном полиноме.
3. Для каждого монома уменьшить его степень по указанной переменной и умножить коэффициент на старую степень.
4. Добавить полученные мономы в новый полином.
5. По завершении итерации по всем мономам вернуть новый полином, который представляет собой производную исходного по указанной переменной. Если результат не содержит ненулевых мономов, добавить нулевой моном.

Полином: x + y4+2.

Возьмем производную по y.

Было:

Стало:

1|100

1|40

2|0

4|30

## Описание программной реализации

### 3.2.1 Схема наследования классов

TNode

TList

TPolynom

TMonom

THeadRingList

ЕДшфе

TList

* Стрелками показаны отношения наследования, где базовый класс указывается перед производным классом.
* Ромбовидными стрелками обозначены отношения ассоциации, где класс-владелец связан с классом-компонентом.

### 3.2.2 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\_) {};

bool operator<(const TNode& other) const;

bool operator>(const TNode& other) const;

};

Поля:

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

Методы:

* TNode();

Назначение: конструктор по умолчанию.

Входные параметры отсутствуют:

Выходные параметры: отсутствуют.

* TNode(constT&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.

* bool operator<(const TNode& other) const

Назначение: сравнение двух узлов.

Входные параметры: звено other.

Выходные параметры: true, если текущее звено меньше другого звена, иначе false.

* bool operator>(const TNode& other) const

Назначение: сравнение двух узлов.

Входные параметры: звено other.

Выходные параметры: true, если текущее звено ,больше другого звена, иначе false.

### Описание класса TList

template <typename T>

class TList {

protected:

TNode<T>\* pFirst;

TNode<T>\* pLast;

TNode<T>\* pCurrent;

TNode<T>\* pStop;

TNode<T>\* pPrev;

public:

TList();

TList(const TList<T>& list);

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

virtual ~TList();

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

virtual void insert\_first(const T& data);

void insert\_last(const T& data);//

void insert\_before(const T& who, const T& where);

void insert\_after(const T& who, const T& where);

virtual void remove(const T& data);

int GetSize() const;

virtual void Clear();

void reset();

void next();

bool IsEnded() const;

T& GetCurrent() const;

bool IsEmpty() const;

bool IsFull() const;

void insert\_sort(const T& data);

virtual void Sort();

const TList<T>& operator=(const TList<T>& other);

};

Поля:

pFirst– указатель на первый элемент.

pStop – указатель на конец списка.

pCurrent – указатель на текущий элемент.

pPrev – указатель на предыдущий элемент.

pLast – указатель на последний элемент.

Методы:

* TList();

Назначение: создание списка без объектов.

Входные параметры: отсутствуют.

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.

* **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.

* **virtual ~TList();**

Назначение: освобождение памяти списка при удалении объекта.

Входные параметры: отсутствуют.

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, если элемент не найден.

* **virtual void insert\_first(const T& data);**

Назначение: вставляет новый узел с данными в начало списка.

Входные параметры: data – данные для нового узла.

Выходные параметры: отсутствуют.

* **virtual void insert\_last(const T& data);**

Назначение: вставляет новый узел с данными в конец списка.

Входные параметры: data – данные для нового узла.

Выходные параметры: отсутствуют.

* **void insert\_before(const T& who, const T&where);**

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

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

Выходные параметры: отсутствуют.

* **void insert\_after(const T& who, const T&where);**

Назначение: вставляет новый узел с данными до узла с определенными данными в списке.

Входные параметры: who – данные для нового узла, where – данные узла, до которого будет вставлен новый узел.

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.

* **int GetSize() const;**

Назначение: возвращает текущий размер списка.

Входные параметры: отсутствуют.

Выходные параметры: размер списка (целочисленное значение).

* **void clear();**

Назначение: очищает список, освобождает выделенную память.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

* **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.

* **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.

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

Назначение: возвращает указатель на текущий узел.

Входные параметры: отсутствуют.

Выходные параметры: указатель на текущий узел.

* **bool IsEmpty() const;**

Назначение: проверяет, пуст ли список.

Входные параметры: отсутствуют.

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

* **void insert\_sort(const T& data);**

Назначение: вставляет новый узел с данными в отсортированный список.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

* virtual void Sort();

Назначение: сортировка списка.

Входные данные: отсортированный или частично отсортированный список.

Выходные данные: список, отсортированный по возрастанию значений элементов.

* const TList<T>& operator=(const TList<T>& other);

Назначение: выполнение операции присваивания списков.

Входные данные: другой список, который нужно скопировать.

Выходные данные: ссылка на текущий список после присваивания.

### **Описание класса THeadRingList**

template <typename T>

class THeadRingList : public TList<T> {

private:

TNode<T>\* pHead;

public:

THeadRingList();

THeadRingList(const THeadRingList& ringL);

virtual ~THeadRingList();

void insert\_first(const T& data);

void remove(const T& data);

void Clear();

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

};

Поля:

pHead – указатель на головной элемент кольцевого списка.

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: новый объект класса THeadRingList, который является копией ringL.

* **virtual ~THeadRingList();**

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

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

* **void insert\_first(const T& data);**

Назначение: вставляет новый узел с данными в начало кольцевого списка.

Входные параметры: data – данные для нового узла.

Выходные параметры: отсутствуют.

* THeadRingList<T>::remove(const T& data)

Назначение: удаляет узел с заданными данными из кольцевого списка.

Входные данные: data — данные узла, который нужно удалить.

Выходные данные: отсутствуют.

* THeadRingList<T>::operator=(const THeadRingList<T>& l)

Назначение: оператор присваивания, копирующий содержимое одного кольцевого списка в другой.

Входные данные: l — кольцевой список, который нужно скопировать.

Выходные данные: ссылка на текущий объект класса THeadRingList.

* THeadRingList<T>::Clear()

Назначение: очищает кольцевой список, освобождая выделенную память.

Входные данные: отсутствуют.

Выходные данные: отсутствуют.

### Описание класса TMonom

class TMonom {

public:

int degree;

double coeff;

TMonom();

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;

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

TMonom operator+(const TMonom& monom)const;

double monom(double x, double y, double z) const {

return coeff \* pow(x, get\_x\_deg()) \* pow(y, get\_y\_deg())

\* pow(z, get\_z\_deg());}

int get\_x\_deg() const;

int get\_y\_deg() const;

int get\_z\_deg() 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 TMonom::operator==(const TMonom& data) const**

Назначение: проверяет равенство двух мономов по степени.

Входные данные: data — моном для сравнения.

Выходные данные: true, если степени мономов равны, иначе false.

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

Назначение: проверяет, меньше ли степень текущего монома, чем степень переданного монома.

Входные данные: data — моном для сравнения.

Выходные данные: true, если степень текущего монома меньше, иначе false.

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

Назначение: проверяет, меньше ли степень текущего монома, чем степень переданного монома.

Входные данные: data — моном для сравнения.

Выходные данные: true, если степень текущего монома меньше, иначе false.

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

Назначение: проверяет, меньше ли или равна степень текущего монома степени переданного монома.

Входные данные: data — моном для сравнения.

Выходные данные: true, если степень текущего монома меньше или равна, иначе false.

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

Назначение: проверяет неравенство двух мономов по степени.

Входные данные: data — моном для сравнения.

Выходные данные: true, если степени мономов не равны, иначе false.

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

Назначение: умножение двух мономов.

Входные данные: monom — моном, который нужно умножить на текущий моном.

Выходные данные: новый моном, который является результатом умножения.

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

Назначение: сложение двух мономов с одинаковой степенью.

Входные данные: monom — моном, который нужно сложить с текущим мономом.

Выходные данные: новый моном, который является результатом сложения.

* int TMonom::get\_x\_deg() const

Назначение: получение степени переменной x.

Входные данные: отсутствуют.

Выходные данные: степень переменной x (целое число).

* int TMonom::get\_y\_deg() const

Назначение: получение степени переменной y.

Входные данные: отсутствуют.

Выходные данные: степень переменной y (целое число).

* int TMonom::get\_z\_deg() const

Назначение: получение степени переменной z.

Входные данные: отсутствуют.

Выходные данные: степень переменной z (целое число).

### Описание класса TPolynom

class TPolynom {

private:

string name;

THeadRingList<TMonom> monoms;

void ParseMonoms(const string& \_name);

void delNULL();

void CombineSimilarMonoms();

public:

TPolynom();

TPolynom(const string& \_name);

TPolynom(const THeadRingList<TMonom>& list);

TPolynom operator+(const TPolynom& p);

TPolynom operator -(const TPolynom& polynom);

TPolynom operator-() const;

TPolynom operator \*(const TPolynom& polynom);

TPolynom dx() const;

TPolynom dy() const;

TPolynom dz() const;

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

string ToString()const;

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

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

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

friend istream& operator>>(std::istream& in, TPolynom& polynom);

};

Поля:

name – строка полинома.

monoms – кольцевой линейный односвязный список.

Методы:

* TPolynom::TPolynom()

Назначение: конструктор по умолчанию для создания пустого полинома.

Входные данные: отсутствуют.

Выходные данные: новый объект класса TPolynom с пустым списком мономов и именем "0".

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

Назначение: конструктор, создающий полином на основе строки.

Входные данные: \_name — строка, представляющая полином.

Выходные данные: новый объект класса TPolynom, инициализированный на основе строки.

* TPolynom::TPolynom(const THeadRingList<TMonom>& list)

Назначение: конструктор, создающий полином на основе переданного списка мономов.

Входные данные: list — список мономов, представляющий полином.

Выходные данные: новый объект класса TPolynom, инициализированный на основе списка мономов.

* TPolynom::ToString()

Назначение: преобразует полином в строковое представление.

Входные данные: отсутствуют.

Выходные данные: строка, представляющая полином.

* TPolynom::ParseMonoms(const string& \_name)

Назначение: разбирает строковое представление полинома и заполняет список мономов.

Входные данные: \_name — строка, представляющая полином.

Выходные данные: отсутствуют.

* TPolynom TPolynom::operator+(const TPolynom& polynom\_)

Назначение: выполняет сложение двух полиномов и возвращает новый полином, представляющий их сумму.

Входные данные: polynom\_: полином, который нужно добавить к текущему полиному.

Выходные данные: новый объект класса TPolynom, представляющий сумму полиномов.

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

Назначение: выполняет вычитание полиномов.

Входные данные: polynom: полином, который нужно вычесть из текущего полинома.

Выходные данные: новый объект класса TPolynom, представляющий разность полиномов.

* TPolynom TPolynom::operator-() const

Назначение: возвращает новый полином, представляющий отрицание текущего полинома.

Входные данные: отсутствуют.

Выходные данные: новый объект класса TPolynom, представляющий отрицание текущего полинома.

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

Назначение: выполняет умножение двух полиномов и возвращает новый полином, представляющий их произведение.

Входные данные: polynom\_ полином, который нужно умножить на текущий полином.

Выходные данные: новый объект класса TPolynom, представляющий произведение полиномов.

* TPolynom TPolynom::dx() const

Назначение: вычисляет производную полинома по переменной x.

Входные данные: отсутствуют.

Выходные данные: новый объект класса TPolynom, представляющий производную текущего полинома по переменной x.

* TPolynom TPolynom::dy() const

Назначение: вычисляет производную полинома по переменной y.

Входные данные: отсутствуют.

Выходные данные: новый объект класса TPolynom, представляющий производную текущего полинома по переменной y.

* TPolynom TPolynom::dz() const

Назначение: вычисляет производную полинома по переменной z.

Входные данные: отсутствуют.

Выходные данные: новый объект класса TPolynom, представляющий производную текущего полинома по переменной z.

* bool TPolynom::operator==(const TPolynom& polynom) const

Назначение: сравнивает два полинома на равенство.

Входные данные: полином для сравнения.

Выходные данные: true, если полиномы равны, иначе false.

* bool TPolynom::operator!=(const TPolynom& polynom) const

Назначение: проверяет неравенство двух полиномов.

Входные данные: полином для сравнения.

Выходные данные: true, если полиномы не равны, иначе false.

* void TPolynom::delNULL()

Назначение: удаляет мономы с нулевым коэффициентом из полинома.

Входные данные: отсутствуют.

Выходные данные: отсутствуют.

* void TPolynom::CombineSimilarMonoms()

Назначение: объединяет мономы с одинаковыми степенями в полиноме.

Входные данные: отсутствуют.

Выходные данные: отсутствуют.

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

Назначение: вычисляет значение полинома в заданных точках 𝑥,𝑦,𝑧

Входные данные: значения переменных x,y,z.

Выходные данные: значение полинома в указанных точках.

* ostream& operator<<(ostream& out, const TPolynom& polynom)

Назначение: отправляет строковое представление полинома в выходной поток.

Входные данные: ссылка на выходной поток out и константная ссылка на полином polynom.

Выходные данные: ссылка на выходной поток.

* istream& operator>>(std::istream& in, TPolynom& polynom)

Назначение: читает строковое представление полинома из входного потока.

Входные данные: ссылка на входной поток in и ссылка на объект полинома polynom.

Выходные данные: ссылка на входной поток.

# **Заключение**

В ходе данной лабораторной работы были исследованы и реализованы методы обработки полиномов на основе связанных списков. Мы познакомились с принципами работы с мономами и полиномами, а также научились выполнять над ними различные операции, включая сложение, вычитание, умножение и вычисление значений при заданных переменных. Этот опыт дал понимание о том, как эффективно решать математические задачи, связанные с полиномами, с помощью кольцевых односвязных списков с головой. Кроме этого, списки, как обычные, так и кольцевые повсеместно используются в программировании, поэтому знание и умение ими пользоваться позволяет упростить решение задач.

# Литература

1. Динамические структуры типа «список» [[12.2. Динамические структуры типа «список» (studfile.net)](https://studfile.net/preview/364624/page:2/)]
2. Лекция «Списковые структуры хранения» Сысоев А. В. [ [https://cloud.unn.ru/s/6g44ey6HFB4ncDy](https://cloud.unn.ru/s/6g44ey6HFB4ncDy" \t "_blank)].
3. Кольцевой односвязный список [[Структуры данных в C# | Кольцевой односвязный список (metanit.com)](https://metanit.com/sharp/algoritm/2.7.php)].

# Приложения

## Приложение А. Реализация класса TList

template <typename T>

TList<T>::TList() {

pFirst = nullptr;

pLast = nullptr;

pCurrent = nullptr;

pPrev = nullptr;

pStop = nullptr;

}

template <typename T>

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

if (list.IsEmpty())

{

pFirst = nullptr;

pLast = nullptr;

pCurrent = nullptr;

pPrev = nullptr;

pStop = nullptr;

return;

}

pFirst = new TNode<T>(list.pFirst->data);

TNode<T>\* node\_tmp = pFirst;

TNode<T>\* list\_tmp = list.pFirst->pNext;

while (list\_tmp != list.pStop)

{

node\_tmp->pNext = new TNode<T>(list\_tmp->data);

node\_tmp = node\_tmp->pNext;

list\_tmp = list\_tmp->pNext;

}

pLast = node\_tmp;

pCurrent = pFirst;

pStop = pPrev = nullptr;

}

template <typename T>

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

pFirst = pNode;

TNode<T>\* tmp = pNode;

while (tmp->pNext != nullptr) {

tmp = tmp->pNext;

}

pLast = tmp;

pCurrent = pFirst;

pStop = pPrev = nullptr;

}

template <typename T>

TList<T>::~TList() {

Clear();

}

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;

pCurrent = nullptr;

pFirst = nullptr;

pPrev = nullptr;

pLast = nullptr;

}

template <typename T>

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

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

if (tmp == nullptr)

return true;

delete tmp;

return false;

}

template <typename T>

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

return (pFirst == nullptr);

}

template <typename T>

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

return (pCurrent == pStop);

}

template <typename T>

void TList<T>::insert\_first(const T& data) {

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

if (pLast == nullptr) {

pLast = pFirst;

}

pCurrent = pFirst;

pPrev = nullptr;

}

template <typename T>

void TList<T>::insert\_last(const T& data) {

if (IsEmpty()) {

insert\_first(data);

return;

}

pLast->pNext = new TNode<T>(data, pStop);

pLast = pLast->pNext;

pCurrent = pLast;

}

template <typename T>

TNode<T>\* TList<T>::search(const T& data) {

if (IsEmpty())

return nullptr;

TNode<T>\* prev = nullptr;

TNode<T>\* curr = pFirst;

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

prev = curr;

curr = curr->pNext;

}

if (curr == pStop)

return nullptr;

pCurrent = curr;

pPrev = prev;

return curr;

}

template<typename T>

int TList<T>::GetSize() const {

if (pFirst == nullptr)

return 0;

int size = 0;

for (TNode<T>\* node = pFirst; node != pStop; node = node->pNext) {

++size;

}

return size;

}

template <typename T>

void TList<T>::insert\_before(const T& who, const T& where) {

TNode<T>\* pWhere = search(where);

if (pWhere == nullptr) {

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

}

if (pWhere == pFirst) {

insert\_first(who);

return;

}

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

pPrev->pNext = new\_node;

pCurrent = new\_node;

}

template <typename T>

void TList<T>::insert\_after(const T& who, const T& where) {

TNode<T>\* pWhere = search(where);

if (pWhere == nullptr) {

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

}

if (pWhere == pLast) {

insert\_last(who);

return;

}

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

pWhere->pNext = new\_node;

pCurrent = new\_node;

pPrev = pWhere;

}

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)

{

TNode<TData>\* first = pFirst;

pFirst = pFirst->pNext;

delete first;

pCurrent = pFirst;

return;

}

if (pWhere == pLast)

{

pPrev->pNext = pStop;

pLast = pPrev;

pCurrent = pStop;

delete pWhere;

return;

}

pPrev->pNext = pWhere->pNext;

pCurrent = pWhere->pNext;

delete pWhere;

}

template <typename T>

void TList<T>::reset() {

pCurrent = pFirst;

pPrev = nullptr;

}

template <typename T>

void TList<T>::next() {

if (IsEnded()) {

throw std::range\_error("List is ended");

}

pPrev = pCurrent;

pCurrent = pCurrent->pNext;

}

template<typename T>

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

return pCurrent->data;

}

template <typename T>

void TList<T>::insert\_sort(const T& data) {

if (IsEmpty() || data < pFirst->data) {

insert\_first(data);

return;

}

TNode<T>\* tmp = pFirst;

while (tmp->pNext != pStop && tmp->pNext->data <= data) {

tmp = tmp->pNext;

}

insert\_after(data, tmp->data);

}

template <typename T>

void TList<T>::Sort() {

if (IsEmpty() || GetSize() == 1) {

return;

}

TNode<T>\* Current = pFirst;

TNode<T>\* Previous = nullptr;

bool sorted = false;

while (!sorted) {

sorted = true;

Current = pFirst;

Previous = nullptr;

while (Current->pNext != nullptr && Current->pNext != pStop) {

TNode<T>\* next = Current->pNext;

if (Current->data < next->data) { // Используем пользовательскую функцию сравнения

if (Previous == nullptr) {

pFirst = next;

}

else {

Previous->pNext = next;

}

Current->pNext = next->pNext;

next->pNext = Current;

Previous = next;

sorted = false;

}

else {

Previous = Current;

Current = Current->pNext;

}

}

}

pCurrent = pFirst;

}

template <typename T>

const TList<T>& TList<T>::operator=(const TList<T>& other)

{

if (this == &other) { return \*this; }

Clear();

TNode<T>\* otherCurr = other.pFirst;

while (otherCurr != other.pLast)

{

insert\_last(otherCurr->data);

otherCurr = otherCurr->pNext;

}

insert\_last(otherCurr->data);

return \*this;

}

## Приложение Б. Реализация классаTHeadRingList

template <typename T>

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

pHead = new TNode<T>();

pStop = pHead;

}

template <typename T>

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

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

if (!ringL.IsEmpty())

pLast->pNext = pHead;

pStop = pHead;

}

template <typename T>

THeadRingList<T>::~THeadRingList() {

delete pHead;

}

template <typename T>

void THeadRingList<T>::insert\_first(const T& data) {

TList<T>::insert\_first(data);

pHead->pNext = pFirst;

pStop = pHead;

pLast->pNext = pHead;

}

template <typename T>

void THeadRingList<T>::remove(const T& data) {

if (!this->pFirst) {

return;

}

TList<T>::remove(data);

if (!this->pFirst) {

delete pHead;

pHead = nullptr;

}

else if (this->pFirst == pHead) {

pHead = this->pFirst;

}

pCurrent = this->pFirst;

}

template <typename T>

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

this->TList<T>::operator=(l);

this->pHead->pNext = this->pFirst;

return \*(this);

}

template <class T> void THeadRingList<T>::Clear() {

TList<T>::Clear();

pHead->pNext = pHead;

}

## Приложение В. Реализация класса TMonom

#include "monom.h"

#include <stdexcept>

TMonom::TMonom() : coeff(0.0), degree(-1) {};

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

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

throw std::exception("Degree must be from 0 to 999");

}

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 !(\*this == data);

}

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

if ((degree + monom.degree) <= 999 && (degree + monom.degree) >= 0) {

return TMonom(coeff \* monom.coeff, degree + monom.degree);

}

else {

throw std::exception("The degree must be between 0 and 999.");

}

}

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

return TMonom(coeff + monom.coeff, degree);

}

int TMonom::get\_x\_deg() const {

return degree / 100;

}

int TMonom::get\_y\_deg() const {

return (degree % 100) / 10;

}

int TMonom::get\_z\_deg() const {

return degree % 10;

}

## Приложение Г. Реализация класса TPolynom

#include "polynom.h"

#include <sstream>

TPolynom::TPolynom() : monoms(), name("0") {}

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

name = \_name;

ParseMonoms(\_name);

CombineSimilarMonoms();

}

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

if (!list.IsEmpty()) {

monoms = list;

delNULL();

CombineSimilarMonoms();

monoms.Sort();

}

else {

monoms.insert\_first(TMonom(0, 0));

}

name = ToString();

}

string TPolynom::ToString() const {

ostringstream stream;

if (monoms.IsEmpty()) {

stream << "0";

return stream.str();

}

THeadRingList<TMonom> sortedMonoms(monoms);

//sortedMonoms.Sort();

bool firstTerm = true;

// Проверка первого монома на равенство нулю

if (sortedMonoms.GetCurrent().coeff == 0 && sortedMonoms.GetCurrent().degree == 0) {

return "0";

}

while (!sortedMonoms.IsEnded()) {

const TMonom& monom = sortedMonoms.GetCurrent();

double coeff = monom.coeff;

if (coeff != 0) {

if (!firstTerm) {

stream << ((coeff > 0) ? "+" : "-");

}

else {

if (coeff < 0) {

stream << "-";

}

firstTerm = false;

}

const double kAbs = fabs(coeff);

if (monom.degree == 0 || kAbs != 1.0) {

stream << kAbs;

}

if (monom.degree != 0) {

const int x = monom.get\_x\_deg();

if (x != 0) stream << "x" + ((x != 1) ? "^" + std::to\_string(x) : "");

const int y = monom.get\_y\_deg();

if (y != 0) stream << "y" + ((y != 1) ? "^" + std::to\_string(y) : "");

const int z = monom.get\_z\_deg();

if (z != 0) stream << "z" + ((z != 1) ? "^" + std::to\_string(z) : "");

}

}

sortedMonoms.next();

}

return stream.str();

}

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

istringstream stream(\_name);

int sign = 1;

while (!stream.eof()) {

TMonom tmp(0, 0);

if (stream.peek() == '-') {

sign = -1;

stream.ignore(); // пропускаем минус

}

if (!(stream >> tmp.coeff)) {

tmp.coeff = 1.0;

stream.clear(); // снимает состояние ошибки

}

tmp.coeff \*= sign;

char c;

while (stream >> c) {

if (c == ' ' || c == '\*') {

continue;

}

if (c == '+' || c == '-') {

sign = (c == '-') ? -1 : 1;

break;

}

int degree\_component;

if (stream.peek() == '^') {

stream.ignore();

stream >> degree\_component;

} else {

degree\_component = 1;

}

const int components = 'z' - 'x';

int component\_index = c - 'x';

tmp.degree += degree\_component \* pow(10, (components - component\_index));

}

if (tmp.coeff != 0 || \_name == "0") {

monoms.insert\_sort(tmp);

}

}

name = ToString();

}

TPolynom TPolynom::operator+(const TPolynom& polynom\_)

{

THeadRingList<TMonom> list;

TPolynom cpolynom(polynom\_);

monoms.reset();

cpolynom.monoms.reset();

if (name == "0" || polynom\_.name == "0") { return (name == "0") ? polynom\_ : (\*this); }

while (!monoms.IsEnded() && !cpolynom.monoms.IsEnded())

{

TMonom monom1 = monoms.GetCurrent();

TMonom monom2 = cpolynom.monoms.GetCurrent();

if (monom1 == monom2)

{

double coef1 = monom1.coeff;

double coef2 = monom2.coeff;

double coef\_res = coef1 + coef2;

if (coef\_res != 0)

{

monom2.coeff = coef\_res;

list.insert\_last(monom2);

}

monoms.next();

cpolynom.monoms.next();

}

else if (monom2 < monom1)

{

list.insert\_last(monom2);

cpolynom.monoms.next();

}

else

{

list.insert\_last(monom1);

monoms.next();

}

}

while (!monoms.IsEnded())

{

list.insert\_last(monoms.GetCurrent());

monoms.next();

}

while (!cpolynom.monoms.IsEnded())

{

list.insert\_last(cpolynom.monoms.GetCurrent());

cpolynom.monoms.next();

}

TPolynom result;

result.monoms = list;

result.name = result.ToString();

return result;

}

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

return (\*this) + (-polynom);

}

TPolynom TPolynom::operator-() const {

TPolynom result(\*this);

result.monoms.reset();

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

result.monoms.GetCurrent().coeff \*= -1;

result.monoms.next();

}

return result;

}

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

{

TPolynom cpolynom(polynom\_);

THeadRingList<TMonom> list;

monoms.reset();

cpolynom.monoms.reset();

while (!monoms.IsEnded())

{

const TMonom& m = monoms.GetCurrent();

cpolynom.monoms.reset();

while (!cpolynom.monoms.IsEnded())

{

const TMonom& m2 = cpolynom.monoms.GetCurrent();

double k = m.coeff;

double k2 = m2.coeff;

double k3 = k \* k2; // == 0

int d = m.degree;

int d2 = m2.degree;

int deg = d + d2; // > 999 можно типо выйти за границы

cpolynom.monoms.next();

if (k3 == 0 && deg != 0)continue;

TMonom mon(k3, deg);

list.insert\_last(mon);

}

monoms.next();

}

TPolynom result;

result.monoms = list;

result.name = result.ToString();

return result;

}

TPolynom TPolynom::dx() const {

TPolynom cp = \*this;

TPolynom result;

bool flag = false;

cp.monoms.reset();

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

const TMonom& monom = cp.monoms.GetCurrent();

if (monom.degree >= 100) {

flag = true;

int new\_degree = monom.degree - 100;

double new\_coeff = monom.coeff \* (monom.degree / 100);

result.monoms.insert\_last(TMonom(new\_coeff, new\_degree));

}

cp.monoms.next();

}

if(!flag) result.monoms.insert\_last(TMonom(0,0));

return result;

}

TPolynom TPolynom::dy() const {

TPolynom cp = \*this;

TPolynom result;

cp.monoms.reset();

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

const TMonom& monom = cp.monoms.GetCurrent();

int deg = cp.monoms.GetCurrent().degree;

int y = (deg % 100) / 10;

if (y >= 1) {

int new\_degree = monom.degree - 10;

double new\_coeff = monom.coeff \* (monom.get\_y\_deg() != 0 ? monom.get\_y\_deg() : 1);

result.monoms.insert\_last(TMonom(new\_coeff, new\_degree));

}

cp.monoms.next();

}

return result;

}

TPolynom TPolynom::dz() const {

TPolynom cp = \*this;

TPolynom result;

cp.monoms.reset();

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

const TMonom& monom = cp.monoms.GetCurrent();

int deg = cp.monoms.GetCurrent().degree;

int z = deg % 10;

if (z >= 1) {

int new\_degree = monom.degree - 1;

double new\_coeff = monom.coeff \* (monom.get\_y\_deg() != 0 ? monom.get\_z\_deg() : 1);

result.monoms.insert\_sort(TMonom(new\_coeff, new\_degree));

}

cp.monoms.next();

}

return result;

}

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

TPolynom cp = \*this;

TPolynom cp\_other = polynom;

cp.monoms.reset();

cp\_other.monoms.reset();

while (!cp.monoms.IsEnded() && !cp\_other.monoms.IsEnded()) {

if (cp.monoms.GetCurrent() != cp\_other.monoms.GetCurrent()) {

return false;

}

cp.monoms.next();

cp\_other.monoms.next();

}

return (cp.monoms.IsEnded() && cp\_other.monoms.IsEnded());

}

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

return !(\*this == polynom);

}

void TPolynom::delNULL() {

monoms.reset();

while (!monoms.IsEnded()) {

if (monoms.GetCurrent().coeff == 0.0) {

monoms.remove(monoms.GetCurrent());

}

else {

monoms.next();

}

}

}

void TPolynom::CombineSimilarMonoms() {

if (monoms.IsEmpty()) {

return;

}

THeadRingList<TMonom> combinedMonoms;

monoms.reset();

double coeffSum = 0.0;

int currentDegree = monoms.GetCurrent().degree;

while (!monoms.IsEnded()) {

if (monoms.GetCurrent().degree == currentDegree) {

coeffSum += monoms.GetCurrent().coeff;

}

else {

if (coeffSum != 0.0) {

TMonom combinedTerm(coeffSum, currentDegree);

combinedMonoms.insert\_last(combinedTerm);

coeffSum = 0.0;

}

currentDegree = monoms.GetCurrent().degree;

coeffSum = monoms.GetCurrent().coeff;

}

monoms.next();

}

if (coeffSum != 0.0) {

TMonom combinedTerm(coeffSum, currentDegree);

combinedMonoms.insert\_last(combinedTerm);

}

if (combinedMonoms.IsEmpty()) { combinedMonoms.insert\_first(TMonom(0, 0)); }

monoms = combinedMonoms;

}

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

TPolynom cp = \*this;

double result = 0.0;

cp.monoms.reset();

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

const TMonom& monom = cp.monoms.GetCurrent();

double monomValue = monom.monom(x, y, z);

result += monomValue;

cp.monoms.next();

}

return result;

}

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

cout << polynom.ToString();

return out;

}

istream& operator>>(std::istream& in, TPolynom& polynom) {

string input;

getline(in, input);

polynom = TPolynom(input);

return in;

}