Министерство науки и высшего образования Российской Федерации

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

Институт Информационных технологий, математики и механики

Отчет по учебной практике

**Полином от нескольких переменных**

Выполнил:

студент гр. 381806 – 1

Пронькин Д.А.

Проверил:

к.т.н., доцент каф. МОСТ, ИИТММ

Кустикова В.Д.

Нижний Новгород

2020 г.

Содержание

[Введение 3](#_Toc35790880)

[Постановка задачи 4](#_Toc35790881)

[Руководство пользователя 5](#_Toc35790882)

[Руководство программиста 6](#_Toc35790883)

[Описание структуры программы 6](#_Toc35790884)

[Описание структур данных 6](#_Toc35790885)

[Класс TNode 6](#_Toc35790886)

[Класс TList 7](#_Toc35790887)

[Класс Monom 10](#_Toc35790888)

[Класс Polynom 12](#_Toc35790889)

[Описание алгоритмов 16](#_Toc35790890)

[Список 16](#_Toc35790891)

[Полином 18](#_Toc35790892)

[Заключение 21](#_Toc35790893)

[Литература 22](#_Toc35790894)

[Приложения 23](#_Toc35790895)

[Приложение 1. Основная функция 23](#_Toc35790896)

[Приложение 2. Класс TNode 23](#_Toc35790897)

[Приложение 3. Класс TList 24](#_Toc35790898)

[Приложение 4. Класс Monom 29](#_Toc35790899)

[Приложение 5. Класс Polynom 32](#_Toc35790900)

# Введение

Целью данной лабораторной работы является практическое освоение методов организации структур хранения данных с помощью списков. В ходе выполнения лабораторной работы разрабатывается общая форма представления линейных списков, разрабатываются программы работы со списками, которые могут быть использованы и в других областях приложений.

# Постановка задачи

Задача: разработать алгоритм и составить программу для работы с полиномами от нескольких переменных.

Входные данные: 2 полинома от нескольких переменных в виде строк с форматированием вида -4\*x^4\*y^4\*z^4+2\*x\*y^2\*z^3+3\*x^3\*y^2\*z.

Выходные данные: полином от нескольких переменных в виде строки с форматированием вида -3\*x^3\*y^2\*z-4\*x^4\*y^4\*z^4+3\*x^2\*y\*z^3.

# Руководство пользователя

После запуска программы откроется консольный интерфейс с предложением ввести первый полином.

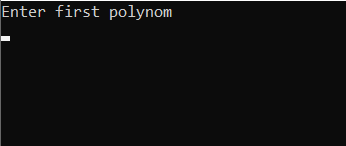


Рис. 1 Стартовый экран

После введения первого полинома нажмите клавишу Enter. Если введенный вами полином корректен, то программа предложит ввести второй полином.

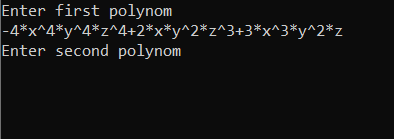


Рис. 2 Программа ожидает ввод второго полинома

После введения второго полинома нажмите клавишу Enter. Если введенный вами полином корректен, то программа выполнит сложение, вычитание и умножение введенных полиномов и выведет результат на экран.

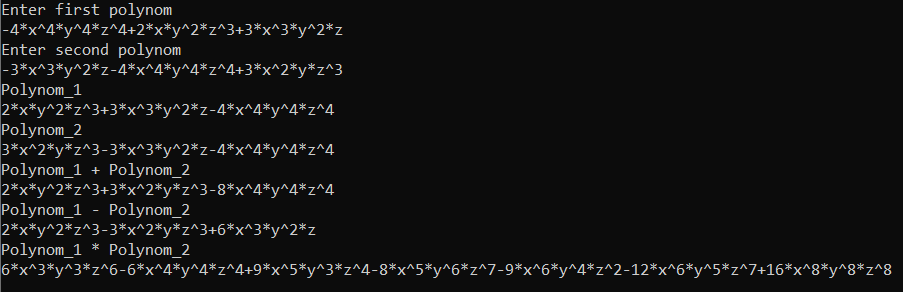


Рис. 3 Выведен результат

Если вы введете некорректный полином, то программа выведет сообщение об ошибке и завершит работу.

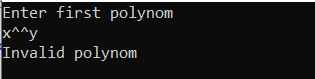


Рис. 4 Введен некорректный полином

# Руководство программиста

## Описание структуры программы

TNode.h – модуль с шаблонным классом, содержащий описание и реализацию функций звена списка.

TList.h – модуль с шаблонным классом, содержащий описание и реализацию функций списка.

Monom.h – модуль, содержащий описание специализированного класс TNode<double, unsigned int>, который выполняет роль монома.

Polynom.h – модуль, содержащий описание полинома.

## Описание структур данных

### Класс TNode

TNode – класс, реализующий звено списка.

**Объявление:**

template<class TData, class TKey>

class TNode {

public:

TData pData;

TKey key;

TNode\* pNext;

public:

TNode(const TNode&);

TNode(TKey, TData, TNode\* \_node = 0);

bool operator!=(const TNode&) const;

friend ostream& operator << (ostream& out, TNode<TData, TKey>& \_node) {

out << \_node.key << " " << \*\_node.pData << endl;

return out;

}

};

**Поля класса:**

pData – данные;

key – ключ;

pNext – указатель на следующее звено.

**Описание методов класса:**

TNode(TKey, TData, TNode\* \_node = 0)

Назначение: конструктор, инициализация объекта

Входные параметры: элементы типа TKey, TData и указатель на звено

Возвращаемое значение: нет

TNode(const TNode&)

Назначение: конструктор копирования, создание копии текущего стека

Входные параметры: константная ссылка на копируемый объект

Возвращаемое значение: нет

~TNode()

Назначение: деструктор, высвобождение памяти и удаление объекта класс

Входные параметры: нет

Возвращаемое значение: нет

bool operator!=(const TNode&) const

Назначение: оператор неравенства двух звеньев

Входные параметры: константная ссылка на объект типа TNode

Возвращаемое значение: логическая переменная

friend ostream& operator << (ostream& out, TNode<TData, TKey>& \_node)

Назначение: вывод TNode

Входные параметры: поток вывода, константная ссылка на объект типа TNode.

Возвращаемое значение: поток вывода

### Класс TList

TList– класс, реализующий список

**Объявление:**

template<class TData, class TKey, class TListNode>

class TList {

protected:

TListNode\* pFirst;

TListNode\* pPrev;

TListNode\* pCurr;

TListNode\* pNext;

public:

TList();

TList(TListNode\*);

TList(const TList&);

~TList();

bool IsEnded() const;

void Next();

void Reset();

TListNode\* Search(TKey);

void Back(TKey, TData);

void Push(TKey, TData);

void InsertAfter(TKey, TData, TKey);

void InsertBefore(TKey, TData, TKey);

void Remove(TKey);

TListNode\* ReturnCurrent() const;

friend ostream& operator << (ostream& out, TList<TData, TKey, TListNode>& \_list) {

if (\_list.pFirst == 0) cout << "List is empty" << endl;

TListNode\* fpCurr = \_list.pCurr;

TListNode\* fpPrev = \_list.pPrev;

TListNode\* fpNext = \_list.pNext;

\_list.Reset();

while (!\_list.IsEnded()) {

out << \_list.pCurr->key << " " << \*\_list.pCurr->pData << endl;

\_list.Next();

}

\_list.pCurr = fpCurr;

\_list.pPrev = fpPrev;

\_list.pNext = fpNext;

return out;

}

};

**Поля класса:**

pFirst – указатель на первый элемент списка;

pPrev – указатель на предыдущий элемент списка.

pCurr – указатель на текущий элемент списка;

pNext – указатель на следующий элемент списка;

**Описание методов класса:**

TList()

Назначение: конструктор по умолчанию.

Входные параметры: нет.

Возвращаемое значение: нет.

TList(TListNode\*)

Назначение: конструктор копирования

Входные параметры: указатель на первое звено списка

Возвращаемое значение: нет

TList(const TList&)

Назначение: конструктор копирования

Входные параметры: константная ссылка на копируемый список

Возвращаемое значение: нет

~TList()

Назначение: деструктор, высвобождение памяти и удаление объекта класс

Входные параметры: нет

Возвращаемое значение: нет

bool IsEnded() const

Назначение: проверка на окончание списка

Входные параметры: нет

Возвращаемое значение: логическая переменная

void Next()

Назначение: переход к следующему элементу

Входные параметры: нет

Возвращаемое значение: нет

void Reset()

Назначение: переход к первому элементу списка

Входные параметры: нет

Возвращаемое значение: нет

TListNode\* Search(TKey)

Назначение: поиска элемента с заданным ключом

Входные параметры: ключ элемента, который необходимо найти

Возвращаемое значение: указатель на элемент списка

void Back(TKey, TData);

Назначение: вставка элемента в конец списка

Входные параметры: ключ и данные вставляемого элемента

Возвращаемое значение: нет

void Push(TKey, TData)

Назначение: вставка элемента в начало списка

Входные параметры: ключ и данные вставляемого элемента

Возвращаемое значение: нет

void InsertAfter(TKey, TData, TKey);

Назначение: вставка элемента в список после заданного элемента

Входные параметры: ключ и данные вставляемого элемента, ключ элемента после которого необходимо выполнить вставку

Возвращаемое значение: нет

void InsertBefore(TKey, TData, TKey);

Назначение: вставка элемента в список до заданного элемента

Входные параметры: ключ и данные вставляемого элемента, ключ элемента до которого необходимо выполнить вставку

Возвращаемое значение: нет

void Remove(TKey)

Назначение: удаление элемента с заданным ключом

Входные параметры: ключ элемента, который необходимо удалить

Возвращаемое значение: нет

TListNode\* ReturnCurrent() const

Назначение: возвращает указатель на текущий элемент

Входные параметры: нет

Возвращаемое значение: указатель на элемент списка

friend ostream& operator << (ostream& out, TList<TData, TKey, TListNode>& \_list)

Назначение: вывод TList

Входные параметры: поток вывода, константная ссылка на объект типа TList.

Возвращаемое значение: поток вывода

### Класс Monom

Monom – класс реализующий моном.

**Объявление:**

template<>

class TNode<double, unsigned int> {

public:

unsigned int key;

double pData;

TNode\* pNext;

TNode(const TNode&);

TNode(unsigned int, double, TNode\* \_monom = 0);

TNode(const string&);

~TNode();

TNode operator\*(double);

TNode operator+(const TNode&);

TNode operator-(const TNode&);

TNode operator\*(const TNode&);

TNode operator-() const;

TNode& operator=(const TNode&);

bool operator>(const TNode&) const;

bool operator<(const TNode&) const;

bool operator==(const TNode&) const;

bool operator!=(const TNode&) const;

};

typedef TNode<double, unsigned int> Monom;

**Поля класса:**

key – степень

pData – коэффициент

pNext – указатель на следующий моном

**Описание методов класса:**

TNode(const TNode&)

Назначение: конструктор копирования, создание копии текущего стека

Входные параметры: константная ссылка на копируемый объект

Возвращаемое значение: нет

TNode(unsigned int, double, TNode\* \_monom = 0);

Назначение: конструктор, инициализация объекта

Входные параметры: степень, коэффициент и указатель на следующий моном

Возвращаемое значение: нет

TNode(const string&)

Назначение: конструктор, инициализация объекта

Входные параметры: константная ссылка на строку

Возвращаемое значение: нет

~TNode();

Назначение: деструктор, высвобождение памяти и удаление объекта класс

Входные параметры: нет

Возвращаемое значение: нет

TNode operator\*(double)

Назначение: оператор умножения монома на число

Входные параметры: число на которое необходимо умножить моном

Возвращаемое значение: результат умножения

TNode operator+(const TNode&)

Назначение: оператор сложения двух мономов

Входные параметры: константная ссылка на моном

Возвращаемое значение: результат сложения

TNode operator-(const TNode&)

Назначение: оператор вычитания двух мономов

Входные параметры: константная ссылка на моном

Возвращаемое значение: результат вычитания

TNode operator\*(const TNode&);

Назначение: оператор умножения двух мономов

Входные параметры: константная ссылка на моном

Возвращаемое значение: результат умножения

TNode operator-() const;

Назначение: унарный минус

Входные параметры: нет

Возвращаемое значение: моном с противоположным знаком

TNode& operator=(const TNode&)

Назначение: оператор присваивания

Входные параметры: константная ссылка на моном

Возвращаемое значение: ссылка на моном

bool operator>(const TNode&) const

Назначение: оператор больше

Входные параметры: константная ссылка на моном

Возвращаемое значение: логическая переменная

bool operator<(const TNode&) const;

Назначение: оператор меньше

Входные параметры: константная ссылка на моном

Возвращаемое значение: логическая переменная

bool operator==(const TNode&) const

Назначение: оператор равенства

Входные параметры: константная ссылка на моном

Возвращаемое значение: логическая переменная

bool operator!=(const TNode&) const;

Назначение: оператор неравенства

Входные параметры: константная ссылка на моном

Возвращаемое значение: логическая переменная

### Класс Polynom

Polynom – класс, реализующий полином.

**Объявление:**

class Polynom {

private:

TList<double, unsigned int, Monom>\* monoms;

void Parse(const string&);

public:

Polynom();

Polynom(const Polynom&);

Polynom(const string&);

~Polynom();

Polynom operator+(const Monom&);

Polynom operator-(const Monom&);

Polynom operator\*(const Monom&);

Polynom& operator+=(const Monom&);

Polynom& operator-=(const Monom&);

Polynom& operator\*=(const Monom&);

Polynom operator+(const Polynom&);

Polynom operator-(const Polynom&);

Polynom operator\*(const Polynom&);

Polynom& operator+=(const Polynom&);

Polynom& operator-=(const Polynom&);

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

Polynom operator-() const;

Polynom& operator=(const Polynom&);

bool operator==(const Polynom&) const;

friend ostream& operator<<(ostream&, const Polynom&);

friend istream& operator>>(istream&, Polynom&);

};

**Поля класса:**

monoms – список мономов

**Описание методов класса:**

Polynom();

Назначение: конструктор по умолчанию

Входные параметры: нет

Возвращаемое значение: нет

Polynom(const Polynom&)

Назначение: конструктор копирования, создание копии текущего стека

Входные параметры: константная ссылка на копируемый объект

Возвращаемое значение: нет

Polynom(const string&)

Назначение: конструктор, инициализация объекта

Входные параметры: константная ссылка на строку

Возвращаемое значение: нет

~Polynom()

Назначение: деструктор, высвобождение памяти и удаление объекта класс

Входные параметры: нет

Возвращаемое значение: нет

Polynom operator+(const Monom&)

Назначение: оператор сложения полинома и монома

Входные параметры: константная ссылка на моном

Возвращаемое значение: результат сложения

Polynom operator-(const Monom&)

Назначение: оператор вычитания монома из полинома

Входные параметры: константная ссылка на моном

Возвращаемое значение: результат вычитания

Polynom operator\*(const Monom&)

Назначение: оператор умножения полинома и монома

Входные параметры: константная ссылка на моном

Возвращаемое значение: результат умножения

Polynom& operator+=(const Monom&);

Назначение: оператор сложения полинома и монома, совмещённый с присваиванием

Входные параметры: константная ссылка на моном

Возвращаемое значение: ссылка на объект класса Polynom

Polynom& operator-=(const Monom&)

Назначение: оператор вычитания монома из полинома, совмещённый с присваиванием

Входные параметры: константная ссылка на моном

Возвращаемое значение: ссылка на объект класса Polynom

Polynom& operator\*=(const Monom&)

Назначение: оператор умножения полинома и монома, совмещённый с присваиванием

Входные параметры: константная ссылка на моном

Возвращаемое значение: ссылка на объект класса Polynom

Polynom operator+(const Polynom&)

Назначение: оператор сложения полиномов

Входные параметры: константная ссылка на полином

Возвращаемое значение: результат сложения

Polynom operator-(const Polynom&)

Назначение: оператор вычитания полиномов

Входные параметры: константная ссылка на полином

Возвращаемое значение: результат вычитания

Polynom operator\*(const Polynom&)

Назначение: оператор умножения полиномов

Входные параметры: константная ссылка на полином

Возвращаемое значение: результат умножения

Polynom& operator+=(const Polynom&)

Назначение: оператор сложения полиномов, совмещённый с присваиванием

Входные параметры: константная ссылка на полином

Возвращаемое значение: ссылка на объект класса Polynom

Polynom& operator-=(const Polynom&)

Назначение: оператор вычитания полиномов, совмещённый с присваиванием

Входные параметры: константная ссылка на полином

Возвращаемое значение: ссылка на объект класса Polynom

Polynom& operator\*=(const Polynom&)

Назначение: оператор умножения полиномов, совмещённый с присваиванием

Входные параметры: константная ссылка на полином

Возвращаемое значение: ссылка на объект класса Polynom

Polynom operator-() const;

Назначение: унарный минус

Входные параметры: нет

Возвращаемое значение: объект класса Polynom

Polynom& operator=(const Polynom&)

Назначение: оператор присваивания

Входные параметры: константная ссылка на полином

Возвращаемое значение: ссылка на полином

bool operator==(const Polynom&) const

Назначение: оператор равенства

Входные параметры: константная ссылка на полином

Возвращаемое значение: логическая переменная

friend ostream& operator<<(ostream&, const Polynom&)

Назначение: вывод полинома

Входные параметры: поток вывода, константная ссылка на объект типа Polynom.

Возвращаемое значение: поток вывода

friend istream& operator>>(istream&, Polynom&);

Назначение: ввод полинома

Входные параметры: поток ввода, константная ссылка на объект тип Polynom

Возвращаемое значение: поток ввода

## Описание алгоритмов

### Список

Для работы со списками предлагается реализовать следующие операции:

методы проверка на пустоту и проверки на полноту списка;

методы навигации по списку (итератор);

методы вставки в начало, в конец, после звена с заданным ключи и перед звеном с заданным ключом;

методы удаления звена с заданным ключом и поиск звена с заданным ключом.

#### Вставка в начало

Рис. 1 Вставка в начало

1. Создать новое звено списка temp.
2. Установить указатель на следующее звено у temp равным pFirst.
3. Поменять указатель начала списка на temp.

#### Вставка в конец

Рис. 2 Вставка в конец

1. Установить указатель temp на последнее звено списка.
2. Создать новое звено node.
3. Установить указатель на следующее звено у temp равным node.
4. Установить указатель на следующее звено у node равным NULL.

#### Вставка после заданного элемента

Рис. 3 Вставка после заданного элемента

1. Найти звено с заданным ключом и записать его в temp.
2. Создать новое звено node.
3. Установить указатель на следующее звено у node равным следующим у temp.
4. Установить указатель на следующее звено у temp равным node.

#### Вставка до заданного элемента

Рис. 4 Вставка до заданного элемента

1. Найти звено, которое стоит перед звеном с заданным ключом и записать его в temp.
2. Создать новое звено node.
3. Установить указатель на следующее звено у node равным следующим у temp.
4. Установить указатель на следующее звено у temp равным node.

#### Удаление звена

Рис. 5 удаление элемента

1. Записать указатель на звено, стоящее перед удаляемым, в temp.
2. Записать указатель на удаляемое звено в node.
3. Записать указатель на звено, стоящее после удаляемого, в next.
4. Установить указатель на следующее звено у temp равным next.
5. Удалить node.

### Полином

Для организации быстрого доступа может быть использовано упорядоченное хранение мономов. Для задания порядка следования можно принять лексикографическое упорядочивание по степеням переменных, при котором мономы упорядочиваются по степеням первой переменной, потом по второй переменной, и только затем по третьей переменной. В общем виде это правило можно записать как соотношение: моном предшествует моному тогда и только тогда, если

(A1 > A2) ˅ (A1 = A2) & (B1 > B2) ˅ (A1 = A2) & (B1 = B2) & (C1 > C2).

Проверка лексикографического порядка занимает сравнительно много времени. Ее можно существенно упростить при помощи свернутой степени (индекса) монома, образуемой с использованием позиционной системы счисления: для монома со степенями (A, B, C) ставится в соответствие величина

ABC = A \* 100 + B \* 10 + C.

Данное соответствие является взаимно-однозначным. Обратное соответствие определяется при помощи выражений

A = [ABC / 100], B = [ABC % 100 / 10], C = ABC % 10.

Кроме того, введенное соответствие порождает порядок, полностью совпадающий с лексикографическим порядком

> ‎‎↔ ABC1 > ABC2.

Выполненное рассуждение позволяет определить, что наиболее эффективным способом организации структуры хранения полиномов являются линейный (односвязный) список. Тем самым, в рамках лабораторной работы появляется подзадача – разработка структуры хранения в виде линейных списков. Данная разработка должна быть выполнена в некоторой общей постановке с тем, чтобы разработанные программы работы со списками могли быть далее использованы и в других ситуациях, в которых необходимы списковые структуры хранения.

Для работы с полиномами предлагается реализовать следующие операции:

1. конструкторы инициализации и копирования;
2. метод присваивания;
3. метод сложения полиномов, вычитания полиномов, умножения полиномов.

**Операции:**

Алгоритмы рассматриваются на примере A операция B.

**Сложение двух полиномов (на листе):**

1. Создается пустой результирующий полином;
2. Берется текущий моном M1 из полинома A и текущий моном M2 из полинома B, если:
   1. M1 > M2, то в конец результирующего полинома добавляется моном M2 и передвигаемся вперед по полиному B;
   2. M1 < M2, то в конец результирующего полинома добавляется моном M1 и передвигаемся вперед по полиному A;
   3. степени M1 и M2 равны, то в конец результирующего полинома добавляется сумма M1 и M2 и передвигаемся вперед по полиному A и B;
3. Повторяем алгоритм пока один из полиномов не закончится;
4. Добавляем в конец результирующего полинома оставшиеся мономы из не пустого полинома.

**Вычитание двух мономов (на листе):**

1. Создается пустой результирующий полином;
2. Берется текущий моном M1 из полинома A и текущий моном M2 из полинома B, если:
   1. M1 > M2, то в конец результирующего полинома добавляется моном M2 с противоположным знаком и передвигаемся вперед по полиному B;
   2. M1 < M2, то в конец результирующего полинома добавляется моном M1 и передвигаемся вперед по полиному A;
   3. степени M1 и M2 равны, то в конец результирующего полинома добавляется разность M1 и M2 и передвигаемся вперед по полиному A и B;
3. Повторяем алгоритм пока один из мономов не закончится;
4. Если:
   1. Не пуст полином A, то добавляем в конец результирующего полинома оставшиеся мономы;
   2. Не пуст полином B, то добавляем в конец результирующего полинома оставшиеся мономы со знаком минус.

**Умножение двух полиномов (на листе):**

1. Создается пустой результирующий полином;
2. Встаем в начало полинома B;
3. Умножаем полином A на текущий моном полинома B;
4. Прибавляем к результирующему полиному результат умножения
5. Передвигаемся вперед по полиному B;
6. Повторяем пока полином B не закончится.

# Заключение

В ходе выполнения лабораторной работы, была изучена структура данных список, её принципы работы и возможные операции. Рассмотрен и реализован на практике класс объекта полином.

# Литература

# Приложения

## Приложение 1. Основная функция

void main() {

try {

Polynom polynom\_1;

Polynom polynom\_2;

cout << "Enter first polynom" << endl;

cin >> polynom\_1;

cout << "Enter second polynom" << endl;

cin >> polynom\_2;

cout << "Polynom\_1" << endl;

cout << polynom\_1 << endl;

cout << "Polynom\_2" << endl;

cout << polynom\_2 << endl;

Polynom polynom\_o;

polynom\_o = polynom\_1 + polynom\_2;

cout << "Polynom\_1 + Polynom\_2" << endl;

cout << polynom\_o << endl;

polynom\_o = polynom\_1 - polynom\_2;

cout << "Polynom\_1 - Polynom\_2" << endl;

cout << polynom\_o << endl;

polynom\_o = polynom\_1 \* polynom\_2;

cout << "Polynom\_1 \* Polynom\_2" << endl;

cout << polynom\_o << endl;

}

catch (const exception & e) {

cout << e.what() << endl;

}

}

## Приложение 2. Класс TNode

template<class TData, class TKey>

class TNode {

public:

TData pData;

TKey key;

TNode\* pNext;

public:

TNode(const TNode&);

TNode(TKey, TData, TNode\* \_node = 0);

bool operator!=(const TNode&) const;

friend ostream& operator << (ostream& out, TNode<TData, TKey>& \_node) {

out << \_node.key << " " << \*\_node.pData << endl;

return out;

}

};

template<class TData, class TKey>

TNode<TData, TKey>::TNode(const TNode<TData, TKey>& copy) {

pData = copy.pData;

key = copy.key;

pNext = copy.pNext;

}

template<class TData, class TKey>

TNode<TData, TKey>::TNode(TKey \_key, TData \_data, TNode\* \_pNext) {

pData = \_data;

key = \_key;

pNext = \_pNext;

}

template<class TData, class TKey>

bool TNode<TData, TKey>::operator!=(const TNode& \_node) const{

if (pData == \_node.pData && key == \_node.key) return false;

return true;

}

## Приложение 3. Класс TList

using namespace std;

template<class TData, class TKey, class TListNode>

class TList {

protected:

TListNode\* pFirst;

TListNode\* pPrev;

TListNode\* pCurr;

TListNode\* pNext;

public:

TList();

TList(TListNode\*);

TList(const TList&);

~TList();

bool IsEnded() const;

void Next();

void Reset();

TListNode\* Search(TKey);

void Back(TKey, TData);

void Push(TKey, TData);

void InsertAfter(TKey, TData, TKey);

void InsertBefore(TKey, TData, TKey);

void Remove(TKey);

TListNode\* ReturnCurrent() const;

friend ostream& operator << (ostream& out, TList<TData, TKey, TListNode>& \_list) {

if (\_list.pFirst == 0) cout << "List is empty" << endl;

TListNode\* fpCurr = \_list.pCurr;

TListNode\* fpPrev = \_list.pPrev;

TListNode\* fpNext = \_list.pNext;

\_list.Reset();

while (!\_list.IsEnded()) {

out << \_list.pCurr->key << " " << \*\_list.pCurr->pData << endl;

\_list.Next();

}

\_list.pCurr = fpCurr;

\_list.pPrev = fpPrev;

\_list.pNext = fpNext;

return out;

}

};

template<class TData, class TKey, class TListNode>

TList<TData, TKey, TListNode>::TList() {

pFirst = 0;

pPrev = 0;

pCurr = 0;

pNext = 0;

}

template<class TData, class TKey, class TListNode>

TList<TData, TKey, TListNode>::TList(TListNode\* list) {

if (list == 0) throw "Empty link";

pFirst = new TListNode(\*list);

TListNode\* node = list;

TListNode\* tmp = pFirst;

while (tmp->pNext != 0) {

node->pNext = TListNode(\*tmp->pNext);

node = node->pNext;

tmp = tmp->pNext;

}

pPrev = 0;

pCurr = pFirst;

pNext = pCurr->pNext;

}

template<class TData, class TKey, class TListNode>

TList<TData, TKey, TListNode>::TList(const TList<TData, TKey, TListNode>& list) {

if (list.pFirst == 0) {

pFirst = 0;

pPrev = 0;

pCurr = 0;

pNext = 0;

return;

}

pFirst = new TListNode(\*list.pFirst);

TListNode\* node = pFirst;

TListNode\* tmp = list.pFirst;

while (tmp->pNext != 0) {

node->pNext = new TListNode(\*(tmp->pNext));

node = node->pNext;

tmp = tmp->pNext;

}

pCurr = pFirst;

pPrev = 0;

pNext = pCurr->pNext;

}

template<class TData, class TKey, class TListNode>

TList<TData, TKey, TListNode>::~TList() {

TListNode\* del = pFirst;

while (del != 0) {

TListNode\* tmp = del->pNext;

delete del;

del = tmp;

}

}

template<class TData, class TKey, class TListNode>

void TList<TData, TKey, TListNode>::Reset() {

pPrev = 0;

pCurr = pFirst;

if (pFirst != 0)

pNext = pFirst->pNext;

else

pNext = 0;

}

template<class TData, class TKey, class TListNode>

void TList<TData, TKey, TListNode>::Next() {

if (IsEnded()) throw "List is ended";

pPrev = pCurr;

pCurr = pNext;

if (pNext != 0)

pNext = pNext->pNext;

else pNext = 0;

}

template<class TData, class TKey, class TListNode>

bool TList<TData, TKey, TListNode>::IsEnded() const {

return (pCurr == 0);

}

template<class TData, class TKey, class TListNode>

TListNode\* TList<TData, TKey, TListNode>::Search(TKey key) {

if (pFirst == 0) return 0;

TListNode\* fpCurr = pCurr;

TListNode\* fpPrev = pPrev;

TListNode\* fpNext = pNext;

Reset();

while ((!IsEnded()) && (pCurr->key != key))

Next();

if (pCurr == 0) {

pPrev = fpPrev;

pCurr = fpCurr;

pNext = fpNext;

return 0;

}

TListNode\* cpCurr = pCurr;

pPrev = fpPrev;

pCurr = fpCurr;

pNext = fpNext;

return cpCurr;

}

template<class TData, class TKey, class TListNode>

void TList<TData, TKey, TListNode>::Push(TKey \_key, TData \_data) {

if (pFirst == 0) {

pFirst = new TListNode(\_key, \_data);

pCurr = pFirst;

return;

}

pFirst = new TListNode(\_key, \_data, pCurr);

if (pCurr == pFirst->pNext) pPrev = pFirst;

}

template<class TData, class TKey, class TListNode>

void TList<TData, TKey, TListNode>::Back(TKey \_key, TData \_data) {

if (pFirst == 0) {

pFirst = new TListNode(\_key, \_data);

pCurr = pFirst;

return;

}

TListNode\* fpCurr = pCurr;

TListNode\* fpPrev = pPrev;

TListNode\* fpNext = pNext;

while (!IsEnded())

Next();

pPrev->pNext = new TListNode(\_key, \_data);

if (fpCurr == pPrev) {

pNext = pPrev->pNext;

pPrev = fpPrev;

pCurr = fpCurr;

}

else if (fpCurr == 0) {

pPrev = pPrev->pNext;

pCurr = fpCurr;

pNext = fpNext;

}

}

template<class TData, class TKey, class TListNode>

void TList<TData, TKey, TListNode>::InsertAfter(TKey ikey, TData data, TKey akey) {

if (pFirst == 0) {

pFirst = new TListNode(ikey, data);

pCurr = pFirst;

return;

}

TListNode\* fpCurr = pCurr;

TListNode\* fpPrev = pPrev;

TListNode\* fpNext = pNext;

Reset();

while ((!IsEnded()) && (pCurr->key != akey))

Next();

if (pCurr == 0) throw "Item not listed";

TListNode\* ins = new TListNode(ikey, data, pNext);

pCurr->pNext = ins;

if (pCurr == fpCurr) {

pNext = pCurr->pNext;

pPrev = fpPrev;

}

else if (pNext == fpCurr) {

pPrev = pCurr->pNext;

pCurr = fpCurr;

pNext = fpNext;

}

else {

pPrev = fpPrev;

pCurr = fpCurr;

pNext = fpNext;

}

}

template<class TData, class TKey, class TListNode>

void TList<TData, TKey, TListNode>::InsertBefore(TKey ikey, TData data, TKey bkey) {

if (pFirst == 0) {

pFirst = new TListNode(ikey, data);

pCurr = pFirst;

return;

}

if (pFirst->key == bkey) {

Push(ikey, data);

return;

}

TListNode\* fpCurr = pCurr;

TListNode\* fpPrev = pPrev;

TListNode\* fpNext = pNext;

Reset();

while ((!IsEnded()) && (pCurr->key != bkey))

Next();

if (pCurr == 0) throw "Item not listed";

TListNode\* ins = new TListNode(ikey, data, pCurr);

pPrev->pNext = ins;

if (pCurr == fpCurr) {

pPrev = pPrev->pNext;

pNext = fpNext;

}

else if (pPrev == fpCurr) {

pNext = pPrev->pNext;

pPrev = fpPrev;

pCurr = fpCurr;

}

else {

pPrev = fpPrev;

pCurr = fpCurr;

pNext = fpNext;

}

}

template<class TData, class TKey, class TListNode>

void TList<TData, TKey, TListNode>::Remove(TKey dkey) {

if (pFirst == 0) throw "List is empty";

TListNode\* fpCurr = pCurr;

TListNode\* fpPrev = pPrev;

TListNode\* fpNext = pNext;

Reset();

while ((!IsEnded()) && (pCurr->key != dkey))

Next();

if (pCurr == 0) throw "Item not listed";

if (pCurr == pFirst) {

if (fpCurr == pFirst) {

delete pFirst;

pFirst = pNext;

pCurr = pFirst;

pNext = pCurr->pNext;

}

else

{

delete pFirst;

pFirst = pNext;

pCurr = fpCurr;

pPrev = fpPrev;

pNext = fpNext;

}

}

else if (pCurr == fpCurr) {

pPrev->pNext = pNext;

delete pCurr;

if (fpNext == 0) pNext = 0;

pNext = fpNext->pNext;

pPrev = fpPrev;

pCurr = fpNext;

}

else if (pPrev == fpCurr) {

pPrev->pNext = pNext;

delete pCurr;

pNext = pPrev->pNext;

pPrev = fpPrev;

pCurr = fpCurr;

}

else if (pNext == fpCurr) {

pPrev->pNext = pNext;

delete pCurr;

pCurr = fpCurr;

pNext = fpNext;

}

else {

pPrev->pNext = pNext;

delete pCurr;

pCurr = fpCurr;

pPrev = fpPrev;

pNext = fpNext;

}

}

template<class TData, class TKey, class TListNode>

TListNode\* TList<TData, TKey, TListNode>::ReturnCurrent() const{

return pCurr;

}

## Приложение 4. Класс Monom

template<>

class TNode<double, unsigned int> {

public:

unsigned int key;

double pData;

TNode\* pNext;

TNode(const TNode&);

TNode(unsigned int, double, TNode\* \_monom = 0);

TNode(const string&);

~TNode();

TNode operator\*(double);

TNode operator+(const TNode&);

TNode operator-(const TNode&);

TNode operator\*(const TNode&);

TNode operator-() const;

TNode& operator=(const TNode&);

bool operator>(const TNode&) const;

bool operator<(const TNode&) const;

bool operator==(const TNode&) const;

bool operator!=(const TNode&) const;

};

typedef TNode<double, unsigned int> Monom;

TNode<double, unsigned int>::TNode(const TNode<double, unsigned int>& \_monom) {

key = \_monom.key;

pData = \_monom.pData;

if (\_monom.pNext == NULL)

pNext = NULL;

pNext = \_monom.pNext;

}

TNode<double, unsigned int>::TNode(unsigned int \_key, double \_pData, TNode<double, unsigned int>\* \_monom) {

if (\_key < 0 || \_key > 999) throw exception("Incorrect monom");

key = \_key;

pData = \_pData;

if (\_monom == NULL)

pNext = NULL;

pNext = \_monom;

}

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

string sws;

string \_coef;

string \_degree;

string \_degreeX = "0";

string \_degreeY = "0";

string \_degreeZ = "0";

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

if (\_str[i] != '\*' && \_str[i] != '^') sws += \_str[i];

int i = 0;

if (sws[0] == 'x' || sws[0] == 'y' || sws[0] == 'z') \_coef = '1';

while (sws[i] != 'x' && sws[i] != 'z' && sws[i] != 'y' && i < sws.length()) {

\_coef += sws[i];

i++;

}

while (i < sws.length()) {

if (sws[i] == 'x') {

i++;

if (sws[i] == 'y' || sws[i] == 'z' || i == sws.length()) \_degreeX = '1';

else \_degreeX = sws[i];

}

if (sws[i] == 'y') {

i++;

if (sws[i] == 'x' || sws[i] == 'z' || i == sws.length()) \_degreeY = '1';

else \_degreeY = sws[i];

}

if (sws[i] == 'z') {

i++;

if (sws[i] == 'x' || sws[i] == 'y' || i == sws.length()) \_degreeZ = '1';

else \_degreeZ = sws[i];

}

i++;

}

\_degree = \_degree + \_degreeX + \_degreeY + \_degreeZ;

key = stoi(\_degree);

pData = stod(\_coef);

pNext = NULL;

}

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

TNode<double, unsigned int> TNode<double, unsigned int>::operator\*(double \_factor) {

TNode result(\*this);

result.pData \*= \_factor;

return result;

}

TNode<double, unsigned int> TNode<double, unsigned int>::operator+(const TNode<double, unsigned int>& \_monom) {

if (key != \_monom.key) return \*this;

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

result.pData += \_monom.pData;

return result;

}

TNode<double, unsigned int> TNode<double, unsigned int>::operator-(const TNode<double, unsigned int>& \_monom) {

if (key != \_monom.key) return \*this;

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

result.pData -= \_monom.pData;

return result;

}

TNode<double, unsigned int> TNode<double, unsigned int>::operator\*(const TNode<double, unsigned int>& \_monom) {

if (key + \_monom.key < 0 || key + \_monom.key > 999) throw "Incorrect degree";

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

result.pData \*= \_monom.pData;

result.key += \_monom.key;

return result;

}

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

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

result.pData \*= -1;

return result;

}

TNode<double, unsigned int>& TNode<double, unsigned int>::operator=(const TNode& \_monom) {

if (key == \_monom.key && pData == \_monom.pData) return \*this;

key = \_monom.key;

pData = \_monom.pData;

pNext = \_monom.pNext;

return \*this;

}

bool TNode<double, unsigned int>::operator<(const TNode& \_monom) const {

if (key < \_monom.key) return true;

return false;

}

bool TNode<double, unsigned int>::operator>(const TNode& \_monom) const {

if (key > \_monom.key) return true;

return false;

}

bool TNode<double, unsigned int>::operator==(const TNode& \_monom) const {

if (key == \_monom.key) return true;

return false;

}

bool TNode<double, unsigned int>::operator!=(const TNode& \_monom) const {

if (key != \_monom.key || pData != \_monom.pData) return true;

return false;

}

## Приложение 5. Класс Polynom

class Polynom {

private:

TList<double, unsigned int, Monom>\* monoms;

void Parse(const string&);

public:

Polynom();

Polynom(const Polynom&);

Polynom(const string&);

~Polynom();

Polynom operator+(const Monom&);

Polynom operator-(const Monom&);

Polynom operator\*(const Monom&);

Polynom& operator+=(const Monom&);

Polynom& operator-=(const Monom&);

Polynom& operator\*=(const Monom&);

Polynom operator+(const Polynom&);

Polynom operator-(const Polynom&);

Polynom operator\*(const Polynom&);

Polynom& operator+=(const Polynom&);

Polynom& operator-=(const Polynom&);

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

Polynom operator-() const;

Polynom& operator=(const Polynom&);

bool operator==(const Polynom&) const;

friend ostream& operator<<(ostream&, const Polynom&);

friend istream& operator>>(istream&, Polynom&);

};

void Polynom::Parse(const string& \_str) {

if (\_str[0] == '+') throw exception("Invalid polynom");

if (\_str[\_str.length() - 1] == '+' || \_str[\_str.length() - 1] == '-') throw exception("Invalid polynom");

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

if (\_str[i] == ' ') throw exception("Invalid polynom");

for (int i = 0; i < \_str.length() - 1; i++)

if (\_str[i] == '+' && \_str[i + 1] == '+' ||

\_str[i] == '+' && \_str[i + 1] == '-' ||

\_str[i] == '-' && \_str[i + 1] == '+' ||

\_str[i] == '-' && \_str[i + 1] == '-' ||

\_str[i] == '\*' && \_str[i + 1] == '\*' ||

\_str[i] == '\*' && \_str[i + 1] == '^' ||

\_str[i] == '^' && \_str[i + 1] == '\*' ||

\_str[i] == '^' && \_str[i + 1] == '^' ||

\_str[i] == '+' && \_str[i + 1] == '\*' ||

\_str[i] == '+' && \_str[i + 1] == '^' ||

\_str[i] == '\*' && \_str[i + 1] == '+' ||

\_str[i] == '^' && \_str[i + 1] == '+' ||

\_str[i] == '-' && \_str[i + 1] == '\*' ||

\_str[i] == '-' && \_str[i + 1] == '^' ||

\_str[i] == '\*' && \_str[i + 1] == '-' ||

\_str[i] == '^' && \_str[i + 1] == '-') throw exception("Invalid polynom");

for (int i = 0; i < \_str.length() - 1; i++)

if (\_str[i] == '^' &&

(\_str[i + 1] == 'x' || \_str[i + 1] == 'y' || \_str[i + 1] == 'z')) throw exception("Invalid polynom");

for (int i = 0; i < \_str.length() - 2; i++)

if (\_str[i] == '^' &&

(\_str[i + 2] != 'x' && \_str[i + 2] != 'y' && \_str[i + 2] != 'z' && \_str[i + 2] != '+' && \_str[i + 2] != '-' && \_str[i + 2] != '\*')) throw exception("Invalid polynom");

}

Polynom::Polynom() {

monoms = new TList<double, unsigned int, Monom>;

}

Polynom::Polynom(const Polynom& \_polynom) {

monoms = new TList<double, unsigned int, Monom>(\*\_polynom.monoms);

}

Polynom::Polynom(const string& str) {

Parse(str);

monoms = new TList<double, unsigned int, Monom>;

int i = 0;

while (i < str.length()) {

string \_monom;

if (str[i] == '-') {

\_monom = str[i];

i++;

}

if (str[i] == '+') i++;

while (str[i] != '+' && str[i] != '-' && i < str.length()) {

\_monom += str[i];

i++;

}

Monom monom(\_monom);

if (monom.pData != 0)

\*this += monom;

monoms->Reset();

}

}

Polynom::~Polynom() {

delete monoms;

}

Polynom Polynom::operator+(const Monom& \_monom) {

Polynom result(\*this);

result.monoms->Reset();

unsigned int \_bkey = 0;

while (!result.monoms->IsEnded() && \*result.monoms->ReturnCurrent() < \_monom && \*result.monoms->ReturnCurrent() != \_monom) result.monoms->Next();

if (result.monoms->IsEnded())

result.monoms->Back(\_monom.key, \_monom.pData);

else if (\*result.monoms->ReturnCurrent() > \_monom) {

\_bkey = result.monoms->ReturnCurrent()->key;

result.monoms->InsertBefore(\_monom.key, \_monom.pData, \_bkey);

}

else if (\*result.monoms->ReturnCurrent() == \_monom) {

\*result.monoms->ReturnCurrent() = \*result.monoms->ReturnCurrent() + \_monom;

if (result.monoms->ReturnCurrent()->pData == 0) result.monoms->Remove(\_monom.key);

}

result.monoms->Reset();

return result;

}

Polynom Polynom::operator-(const Monom& \_monom) {

return \*this + (-\_monom);

}

Polynom Polynom::operator\*(const Monom& \_monom) {

Polynom result(\*this);

result.monoms->Reset();

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

\*result.monoms->ReturnCurrent() = \*result.monoms->ReturnCurrent() \* \_monom;

result.monoms->Next();

}

result.monoms->Reset();

return result;

}

Polynom& Polynom::operator+=(const Monom& \_monom) {

Polynom copy(\*this);

\*this = copy + \_monom;

return \*this;

}

Polynom& Polynom::operator-=(const Monom& \_monom) {

Polynom copy(\*this);

\*this = copy - \_monom;

return \*this;

}

Polynom& Polynom::operator\*=(const Monom& \_monom) {

Polynom copy(\*this);

\*this = copy \* \_monom;

return \*this;

}

Polynom Polynom::operator+(const Polynom& \_polynom) {

Polynom result;

monoms->Reset();

\_polynom.monoms->Reset();

while (!monoms->IsEnded() && !\_polynom.monoms->IsEnded()) {

if (\*monoms->ReturnCurrent() < \*\_polynom.monoms->ReturnCurrent()) {

result.monoms->Back(monoms->ReturnCurrent()->key, monoms->ReturnCurrent()->pData);

result.monoms->Reset();

monoms->Next();

}

else if (\*monoms->ReturnCurrent() > \* \_polynom.monoms->ReturnCurrent()) {

result.monoms->Back(\_polynom.monoms->ReturnCurrent()->key, \_polynom.monoms->ReturnCurrent()->pData);

result.monoms->Reset();

\_polynom.monoms->Next();

}

else if (\*monoms->ReturnCurrent() == \*\_polynom.monoms->ReturnCurrent()) {

if (monoms->ReturnCurrent()->pData + \_polynom.monoms->ReturnCurrent()->pData != 0.0) {

result.monoms->Back(monoms->ReturnCurrent()->key, (monoms->ReturnCurrent()->pData + \_polynom.monoms->ReturnCurrent()->pData));

result.monoms->Reset();

}

monoms->Next();

\_polynom.monoms->Next();

}

}

while (!monoms->IsEnded()) {

result.monoms->Back(monoms->ReturnCurrent()->key, monoms->ReturnCurrent()->pData);

monoms->Next();

}

while (!\_polynom.monoms->IsEnded()) {

result.monoms->Back(\_polynom.monoms->ReturnCurrent()->key, \_polynom.monoms->ReturnCurrent()->pData);

\_polynom.monoms->Next();

}

result.monoms->Reset();

return result;

}

Polynom Polynom::operator-(const Polynom& \_polynom) {

return \*this + (-\_polynom);

}

Polynom Polynom::operator\*(const Polynom& \_polynom) {

Polynom result;

\_polynom.monoms->Reset();

while (!\_polynom.monoms->IsEnded()) {

result += (\*this \* \*\_polynom.monoms->ReturnCurrent());

\_polynom.monoms->Next();

}

return result;

}

Polynom& Polynom::operator+=(const Polynom& \_polynom) {

Polynom copy(\*this);

\*this = copy + \_polynom;

return \*this;

}

Polynom& Polynom::operator-=(const Polynom& \_polynom) {

Polynom copy(\*this);

\*this = copy - \_polynom;

return \*this;

}

Polynom& Polynom::operator\*=(const Polynom& \_polynom) {

Polynom copy(\*this);

\*this = copy \* \_polynom;

return \*this;

}

Polynom Polynom::operator-() const {

Polynom result(\*this);

result.monoms->Reset();

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

result.monoms->ReturnCurrent()->pData \*= -1;

result.monoms->Next();

}

return result;

}

Polynom& Polynom::operator=(const Polynom& \_polynom) {

if (\*this == \_polynom) return \*this;

delete monoms;

monoms = new TList<double, unsigned int, Monom>(\*(\_polynom.monoms));

monoms->Reset();

return \*this;

}

bool Polynom::operator==(const Polynom& \_polynom) const {

monoms->Reset();

\_polynom.monoms->Reset();

while (!monoms->IsEnded() && !\_polynom.monoms->IsEnded()) {

if (\*monoms->ReturnCurrent() != \*\_polynom.monoms->ReturnCurrent()) return false;

monoms->Next();

\_polynom.monoms->Next();

}

if (!monoms->IsEnded() || !\_polynom.monoms->IsEnded()) return false;

return true;

}

ostream& operator<<(ostream& out, const Polynom& \_polynom) {

\_polynom.monoms->Reset();

out << \_polynom.monoms->ReturnCurrent()->pData;

if (\_polynom.monoms->ReturnCurrent()->key / 100 == 1) out << "\*x";

if (\_polynom.monoms->ReturnCurrent()->key / 100 != 0 && \_polynom.monoms->ReturnCurrent()->key / 100 != 1) out << "\*x^" << \_polynom.monoms->ReturnCurrent()->key / 100;

if (\_polynom.monoms->ReturnCurrent()->key % 100 / 10 == 1) out << "\*y";

if (\_polynom.monoms->ReturnCurrent()->key % 100 / 10 != 0 && \_polynom.monoms->ReturnCurrent()->key % 100 / 10 != 1) out << "\*y^" << \_polynom.monoms->ReturnCurrent()->key % 100 / 10;

if (\_polynom.monoms->ReturnCurrent()->key % 10 == 1) out << "\*z";

if (\_polynom.monoms->ReturnCurrent()->key % 10 != 0 && \_polynom.monoms->ReturnCurrent()->key % 10 != 1) out << "\*z^" << \_polynom.monoms->ReturnCurrent()->key % 10;

\_polynom.monoms->Next();

while (!\_polynom.monoms->IsEnded()) {

if (\_polynom.monoms->ReturnCurrent()->pData > 0)

out << "+" << \_polynom.monoms->ReturnCurrent()->pData;

else

out << \_polynom.monoms->ReturnCurrent()->pData;

if (\_polynom.monoms->ReturnCurrent()->key / 100 == 1) out << "\*x";

if (\_polynom.monoms->ReturnCurrent()->key / 100 != 0 && \_polynom.monoms->ReturnCurrent()->key / 100 != 1) out << "\*x^" << \_polynom.monoms->ReturnCurrent()->key / 100;

if (\_polynom.monoms->ReturnCurrent()->key % 100 / 10 == 1) out << "\*y";

if (\_polynom.monoms->ReturnCurrent()->key % 100 / 10 != 0 && \_polynom.monoms->ReturnCurrent()->key % 100 / 10 != 1) out << "\*y^" << \_polynom.monoms->ReturnCurrent()->key % 100 / 10;

if (\_polynom.monoms->ReturnCurrent()->key % 10 == 1) out << "\*z";

if (\_polynom.monoms->ReturnCurrent()->key % 10 != 0 && \_polynom.monoms->ReturnCurrent()->key % 10 != 1) out << "\*z^" << \_polynom.monoms->ReturnCurrent()->key % 10;

\_polynom.monoms->Next();

}

return out;

}

istream& operator>>(istream& in, Polynom& \_polynom) {

delete \_polynom.monoms;

\_polynom.monoms = new TList<double, unsigned int, Monom>;

Polynom PolyParse;

string str;

getline(in, str);

PolyParse.Parse(str);

int i = 0;

while (i < str.length()) {

string \_monom;

if (str[i] == '-') {

\_monom = str[i];

i++;

}

if (str[i] == '+') i++;

while (str[i] != '+' && str[i] != '-' && i < str.length()) {

\_monom += str[i];

i++;

}

Monom monom(\_monom);

if (monom.pData != 0)

\_polynom += monom;

\_polynom.monoms->Reset();

}

return in;

}