Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное   
учреждение высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе №5**

**«Аналитические преобразования полиномов от нескольких переменных»**

**Выполнил**:студент группы 381703-2

Колегов И.А.

**Проверил**:

Доцент кафедры МОСТ

к.т.н. Сысоев А. В.

Нижний Новгород

2018

Оглавление

[Введение 3](#_Toc533162368)

[Постановка задачи 3](#_Toc533162369)

[Руководство пользователя 3](#_Toc533162370)

[Руководство программиста 4](#_Toc533162371)

[Заключение 5](#_Toc533162372)

[Приложение 5](#_Toc533162373)

# Введение

Лабораторная работа направлена на изучение методов компьютерной обработки

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

# Постановка задачи

В рамках лабораторной работы ставится задача создания программных средств,

поддерживающих эффективное представление полиномов и выполнение следующих

операций над ними:

1. ввод полинома
2. организация хранения полинома
3. удаление введенного ранее полинома;
4. копирование полинома;
5. сложение двух полиномов;
6. вычисление значения полинома при заданных значениях переменных;
7. вывод.

Состав реализуемых операций над полиномами может быть расширен при постановке

задания лабораторной работы. Предполагается, что в качестве структуры хранения будут использоваться списки. В качестве дополнительной цели в лабораторной работе ставится также задача разработки некоторого общего представления списков и операций по их обработке. В числе операций над списками должны быть реализованы следующие действия:

1. поддержка понятия текущего звена;
2. вставка звеньев в начало, после текущей позиции и в конец списков;
3. удаление звеньев в начале и в текущей позиции списков;
4. организация последовательного доступа к звеньям списка (итератор).

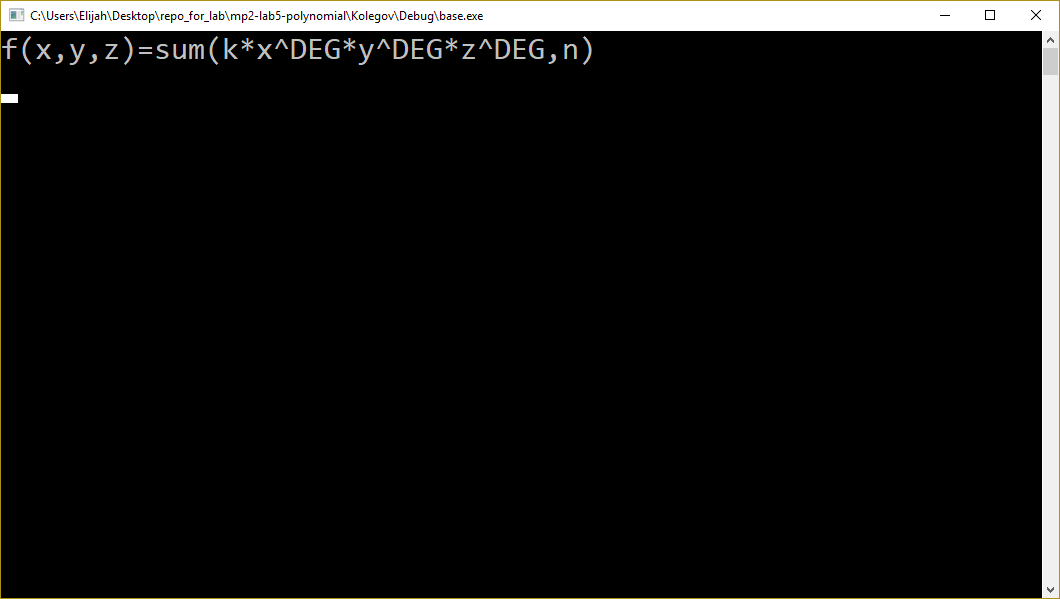
В ходе выполнения лабораторной работы должно быть выполнено сопоставление

разработанных средств работы со списками с возможностями работы со списками в

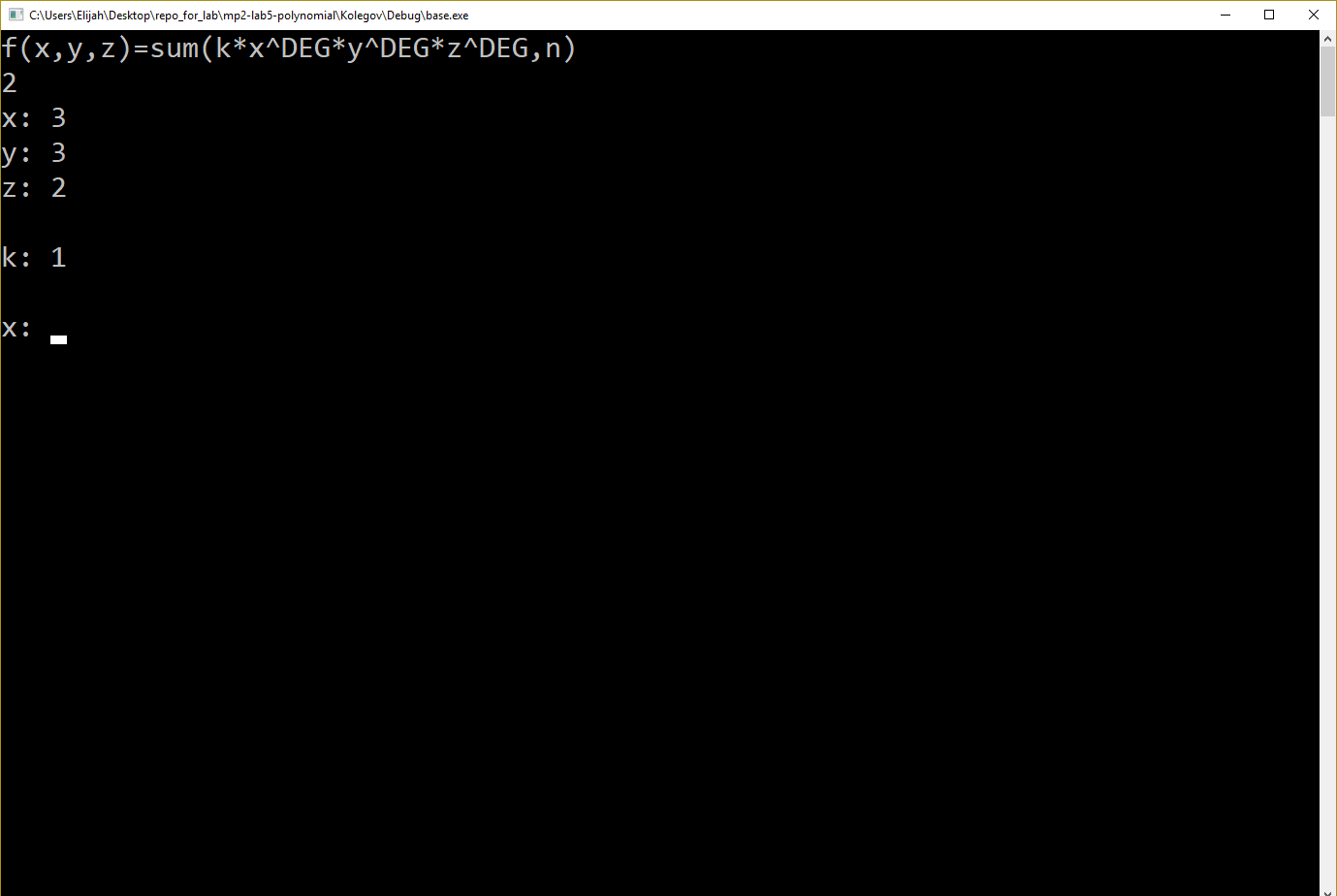
библиотеке STL.

# Руководство пользователя

Для работы с программой пользователю предлагается использовать консольный интерфейс.



Первым параметром указывается количество мономов, состоящих из 3х переменных (например: ). Затем вводятся степени мономов и коэффициент перед ними.



Затем, тем же самым образом пользователю предлагается ввести второй полином. После чего будет предложено ввести конкретные значения для переменных. По окончанию ввода программой будет выведен результат их сложения.

Для корректной работы программы на входные данные накладываются некоторые ограничения. Степень мономов должна идти в убывающем порядке. Значение степени не может превышать 9.

# Руководство программиста

Поставленная задача реализована в 2х классах: TList и Polynom. Первый класс реализует алгоритм односвязного списка, и все необходимые служебные операции для него. Второй класс реализует алгоритм хранения полинома в памяти, а так же возможность их сложения.

Основные поля и методы классов:

TList

Список полей:

1. Node<T>\* pFirst – указатель на первый элемент списка
2. Node<T>\* pLast – указатель на последний элемент списка
3. int size – размер списка

Методы:

1. private iterator IterateElement(int index) – возвращает итератор, который содержит элемент с переданным индексом.
2. T& operator[](int index) – возвращает элемент, по его номеру
3. void Add(T element) – добавляет элемент в начало списка
4. void Insert(T element, int index) – добавляет элемент в список, под переданным индексом

Так же внутри класса TList реализован класс iterator, который является наследником от одноименного стандартного класса. Главным образом в нем реализованы 4 метода: конструктор, оператор сложение, метод для получения величины, хранящейся в узле и оператор сравнения на равенство.

Polynom

Список полей

1. TList<Monom> monoms – список мономов, через который осуществляется хранение полинома

Список методов:

1. void AddMonom(int, double) – добавляет моном в полином, первым аргументом передается целочисленное значение, равное степени монома, записанного в строчку, например моном будет записан как 221. Вторым аргументом передается коэффициент монома.
2. Polynom operator+(Polynom& p) – реализует сложение полиномов методом слияния.
3. double Calculate() – через консольное окно узнает значение переменных, после чего вызывает double Calculate(int x, int y, int z)
4. double Calculate(int x, int y, int z) – считает значение полинома в точке (x,y,z);
5. void GetPolynomFromConsole() – запрашивает данные у пользователя через консоль и сохраняет их в список.

Так же была реализована структура Monom, в которой хранится степень монома, и его коэффициент.

# Заключение

В результате лабораторной работы были изучены методы компьютерной обработки полиномов, а так же такая структура хранения данных, как список.

# Приложение

**Polynom.h:**

#ifndef \_\_PROC\_H\_\_

#define \_\_PORC\_H\_\_

#include "list.h"

#include <iostream>

#include <string>

using namespace std;

struct Monom

{

int koef;

int degree;

Monom& operator+(Monom m)

{

Monom res;

res.koef = m.koef + koef;

res.degree = degree;

return res;

}

bool operator==(Monom m)

{

return koef == m.koef && degree == m.degree;

}

bool operator!=(Monom m)

{

return !(\*this == m);

}

};

class Polynom

{

TList<Monom> monoms;

public:

Polynom() {}

Polynom(Polynom& p) { monoms = p.monoms; }

void AddMonom(int, double);

Polynom operator+(Polynom& p);

double Calculate();

double Calculate(int x, int y, int z);

void GetPolynomFromConsole()

{

cout << "f(x,y,z)=sum(k\*x^DEG\*y^DEG\*z^DEG,n)\n";

int n;

cin >> n;

//Надеюсь, что оно здесь отсортировано

for(int i = 0; i < n; i++)

{

int deg = 1;

int digit;

cout << "x: ";

cin >> digit;

deg = 100\* digit;

cout << "y: ";

cin >> digit;

deg += 10 \* digit;

cout << "z: ";

cin >> digit;

deg += digit;

cout << '\n';

double k;

cout << "k: ";

cin >> k;

cout << '\n';

AddMonom(deg, k);

}

}

friend ostream& operator<< (ostream& os, Polynom& pol)

{

os << "f(x,y,z)=";

for (auto m : pol.monoms)

os << '+' << m.koef << "\*x^" << (m.degree / 100) << "\*y^" << (m.degree % 100) / 10 << "\*z^" << m.degree % 10;

}

bool operator==(Polynom& pol)

{

return monoms == pol.monoms;

}

};

#endif

**List.h:**

#ifndef \_\_LIST\_H\_\_

#define \_\_LIST\_H\_\_

using namespace std;

#include <iostream>

#include <algorithm>

template <class T>

struct Node {

Node<T>\* pNext;

T value;

};

template <class T>

class TList

{

public:

class iterator : public std::iterator<std::input\_iterator\_tag, T, T, Node<T>\*, T> {

Node<T>\* current;

public:

iterator(Node<T>\* point) : current(point) {}

//prefix

iterator& operator++() { current = current->pNext; return \*this; }

//postfix

iterator operator++(int) { ListIterator res = \*this; ++(\*this); return res; }

iterator operator+(int c)

{

for (int i = 0; i < c; i++)

operator++();

return \*this;

}

bool operator==(iterator li) const { return current == li.current; }

bool operator!=(iterator li) const { return !(\*this == li); }

reference operator\*() const { return current->value; }

pointer getNode() const { return current; }

};

private:

Node<T>\* pFirst;

Node<T>\* pLast;

int size;

iterator IterateElement(int index)

{

if (index < 0 || index >= size)

throw "out of bounds";

return (iterator(pFirst) + (index));

}

T GetReadOnly(int index)

{

T el = \*IterateElement(index);

return el;

}

void freeMem()

{

Node<T>\*\* ptrArray = new Node<T>\*[size];

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

ptrArray[i] = IterateElement(i).getNode();

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

delete ptrArray[i];

delete[] ptrArray;

}

public:

TList()

{

pLast = new Node<T>();

pFirst = pLast;

size = 0;

};

TList(TList<T>& list)

{

size = 0;

pLast = new Node<T>();

pFirst = pLast;

for (auto el : list)

Add(el);

}

T& operator[](int index)

{

T el = \*IterateElement(index);

return (T&)el;

}

void Add(T element)

{

Insert(element, size);

}

void Insert(T element, int index)

{

Node<T>\* n;

if (index == 0)

{

Node<T>\* newElement = new Node<T>();

newElement->pNext = pFirst;

newElement->value = element;

pFirst = newElement;

}

else

{

n = IterateElement(index - 1).getNode();

Node<T>\* newElement = new Node<T>();

newElement->pNext = n->pNext;

newElement->value = element;

n->pNext = newElement;

}

size++;

}

bool operator==(TList<T>& list)

{

if (size != list.size)

return false;

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

if (\*IterateElement(i) != list.GetReadOnly(i))

return false;

return true;

}

bool operator!=(TList<T>& list) const

{

return !(\*this == list);

}

TList& operator=(TList<T>& list)

{

freeMem();

pLast = new Node<T>();

pFirst = pLast;

size = 0;

for (auto el : list)

Add(el);

return \*this;

}

~TList()

{

freeMem();

}

int Size()

{

return size;

}

iterator begin() { return iterator(pFirst); }

iterator end() { return iterator(pLast); }

};

#endif