МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

Высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Отчет по Лабораторной работе**

**Операции над полиномами**

**Выполнил:** студент группы 381808-2

Алёхин Денис Андреевич

Нижний Новгород

2019

Содержание

[1. Введение 3](#_Toc24754091)

[2. Постановка задачи 4](#_Toc24754092)

[3. Руководство пользователя 5](#_Toc24754093)

[4. Руководство программиста 6](#_Toc24754094)

[4.1 Описание структуры программы 6](#_Toc24754095)

[4.2 Описание структур данных 7](#_Toc24754096)

[4.3 Описание алгоритмов 8](#_Toc24754097)

[5. Заключение 9](#_Toc24754098)

[Приложение: Фрагменты исходного кода 10](#_Toc24754099)

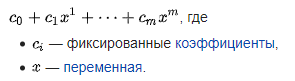
[ReversePolishNotation.h 10](#_Toc24754100)

[ReversePolishNotation.cpp 10](#_Toc24754101)

# Введение

Многочлен (или полином) от {\displaystyle n}n переменных {\displaystyle x\_{1},x\_{2},...x\_{n}}— это сумма одночленов или, строго, — конечная формальная сумма вида .

В частности, многочлен от одной переменной есть конечная формальная сумма вида



# Постановка задачи

Цель работы – разработка программы, позволяющей выполнять операции сложения, вычитания и умножения над полином.

Выполнение работы предполагает решение следующих задач:

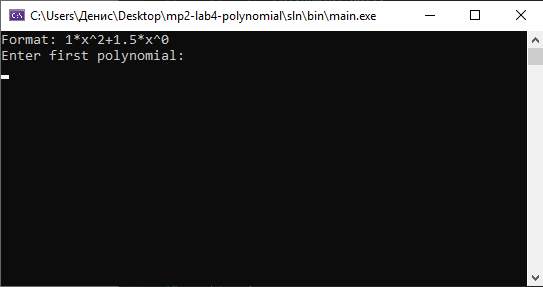
1. Разработка интерфейса класса List и, в частности, Node.
2. Реализация методов класса List.
3. Разработка интерфейса класса Polynomial.
4. Реализация методов класса Polynomial.
5. Реализация main.
6. Публикация исходных кодов в личном репозитории на GitHub.

# Руководство пользователя

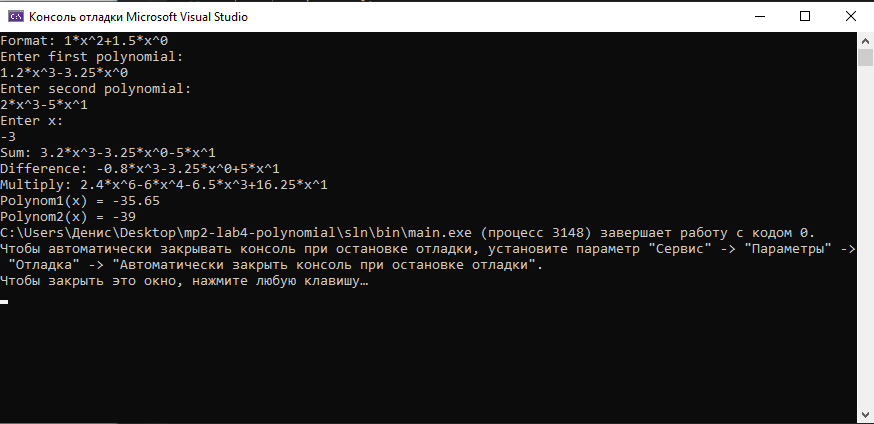
В ходе выполнения работы был получена библиотека Polynomial.

Пример использования данной библиотеки:

1. Программа просит ввести два полинома и значение x в заданном формате:



1. Программа подсчитывает результат сложения, вычитания, умножения двух полиномов, а так же их значения от x:



# Руководство программиста

## 4.1 Описание структуры программы

* docs — инструкции по выполнению лабораторной работы, полезные документы.
* gtest — библиотека Google Test.
* include — директория для размещения заголовочных файлов.
* samples — директория для размещения тестового приложения.
* sln — директория с файлами решений и проектов для VS 2008 и VS 2010, вложенные директории vc9 и vc10 соответственно.
* src — директория для размещения исходных кодов (cpp-файлы).
* test — директория с модульными тестами и основным приложением, инициализирующим запуск тестов.
* README.md — информация о проекте, которую вы сейчас читаете.
* Служебные файлы
  + .gitignore — перечень расширений файлов, игнорируемых Git при добавлении файлов в репозиторий.
  + CMakeLists.txt — корневой файл для сборки проекта с помощью CMake. Может быть использован для генерации проекта в среде разработки, отличной от Microsoft Visual Studio.

## Описание структур данных

Класс List состоит из объектов структуры Node, которые хранят данные и ссылку на следующий элемент

template <class T>

class List

{

public:

List(const List& copyList);

List() : begin(nullptr), size(0) {}

void push\_back(T element);

void pop\_back();

void pop\_front();

List& operator=(const List&);

T getElement(int index) const;

int getSize() { return size; }

void changeElement(int index, T element);

Node<T>\* getBegin() { return begin; }

~List();

private:

Node<T>\* begin;

int size;

Node<T>\* getEnd();

};

Класс Polynomial состоит из списка мономов. Моном хранит степень и коэффициент.

class Monom

{

public:

Monom(const Monom& copyMonom);

Monom(std::string Coeff, std::string Power);

Monom(double Coeff, int Power);

friend bool operator==(const Monom&, const Monom&);

Monom operator+(const Monom& right);

Monom operator-(const Monom& right);

Monom operator\*(const Monom& right);

double getCoeff() const { return coeff; }

int getPower() const { return power; }

void setCoeff(double Coeff) { coeff = Coeff; }

void setPower(int Power) { power = Power; }

private:

double coeff;

int power;

};

class Polynomial

{

public:

Polynomial(const Polynomial& copyPol);

Polynomial() : size(0) {}

friend std::istream& operator>> (std::istream& in, Polynomial& obj);

friend std::ostream& operator<< (std::ostream& out, const Polynomial& obj);

Polynomial& operator=(const Polynomial&);

Polynomial operator+(const Polynomial& right);

Polynomial operator-(const Polynomial& right);

Polynomial operator\*(const Polynomial& right);

double calculate(int x);

int getSize() const { return size; }

void setPol(std::string pol);

Monom returnMonom(int index) const;

private:

List<Monom> data;

int size;

void addMonom(Monom);}

## Описание алгоритмов

В классе **List** реализованы следующие базовые методы:

push\_back(T element)

pop\_back()

pop\_front()

а так же

getSize()

getElement(int index)

changeElement(int index, T element)

Класс Polynomial считывает полином с помощью перегруженной операции ввода, подсчитывает сумму, разность и умножение с помощью соответствующих перегруженных операторов и выводит результат с помощью operator<<.

# Заключение

В ходе лабораторной работы был получен класс на основе которого была создана библиотека, позволяющая проводить операции над полиномамию.

# Приложение: Фрагменты исходного кода

## List.h

#pragma once

template <class T>

struct Node

{

Node(const Node& copyNode);

Node(T Data = 0);

T data;

Node\* next;

};

template <class T>

class List

{

public:

List(const List& copyList);

List() : begin(nullptr), size(0) {}

void push\_back(T element);

void pop\_back();

void pop\_front();

List& operator=(const List&);

T getElement(int index) const;

int getSize() { return size; }

void changeElement(int index, T element);

Node<T>\* getBegin() { return begin; }

~List();

private:

Node<T>\* begin;

int size;

Node<T>\* getEnd();

};

template <class T>

Node<T>::Node(const Node& copyNode) :

data(copyNode.data), next(copyNode.next) {}

template <class T>

Node<T>::Node(T Data) : data(Data), next(nullptr) {}

template <class T>

List<T>::List<T>(const List& copyList)

{

if (copyList.begin == nullptr)

{

begin = nullptr;

size = 0;

}

else

{

int i = 0;

size = copyList.size;

Node<T>\* copyTemp = copyList.begin;

begin = new Node<T>(copyTemp->data);

Node<T>\* temp = begin;

while (i != size - 1)

{

copyTemp = copyTemp->next;

temp->next = new Node<T>(copyTemp->data);

temp = temp->next;

i++;

}

}

}

template <class T>

Node<T>\* List<T>::getEnd()

{

if (begin == nullptr)

throw("List is empty");

Node<T>\* temp = begin;

while (temp->next != nullptr)

temp = temp->next;

return temp;

}

template<class T>

void List<T>::push\_back(T element)

{

if (begin == nullptr)

{

begin = new Node<T>(element);

size++;

}

else

{

Node<T>\* temp = getEnd();

temp->next = new Node<T>(element);

size++;

}

}

template<class T>

void List<T>::pop\_back()

{

if (begin == nullptr)

throw ("List is empty");

else

{

Node<T>\* temp = begin;

if (temp->next != nullptr)

while (temp->next->next != nullptr)

temp = temp->next;

delete temp->next;

temp->next = nullptr;

size--;

if (size == 0)

begin = nullptr;

}

}

template<class T>

void List<T>::pop\_front()

{

if (begin == nullptr)

throw ("List is empty");

else

{

Node<T>\* temp = begin;

begin = temp->next;

delete temp;

size--;

}

}

template<class T>

List<T>& List<T>::operator=(const List<T>& copyList)

{

if (this == &copyList)

return \*this;

if (begin != nullptr)

delete this;

size = copyList.size;

if (copyList.begin == nullptr)

{

begin = nullptr;

return \*this;

}

int i = 0;

Node<T>\* copyTemp = copyList.begin;

begin = new Node<T>(copyTemp->data);

Node<T>\* temp = begin;

while (i != size - 1)

{

copyTemp = copyTemp->next;

temp->next = new Node<T>(copyTemp->data);

temp = temp->next;

i++;

}

return \*this;

}

template<class T>

T List<T>::getElement(int index) const

{

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

throw("Error");

Node<T>\* temp = begin;

int i = 0;

while (i != index)

{

temp = temp->next;

i++;

}

return temp->data;

}

template<class T>

void List<T>::changeElement(int index, T element)

{

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

throw("Error");

Node<T> \* temp = begin;

int i = 0;

while (i != index)

{

temp = temp->next;

i++;

}

temp->data = element;

}

template<class T>

List<T>::~List()

{

while (begin != nullptr)

pop\_front();

}

## Polynomial.h

#pragma once

#include "List.h"

#include <iostream>

#include <string>

class Monom

{

public:

Monom(const Monom& copyMonom);

Monom(std::string Coeff, std::string Power);

Monom(double Coeff, int Power);

friend bool operator==(const Monom&, const Monom&);

Monom operator+(const Monom& right);

Monom operator-(const Monom& right);

Monom operator\*(const Monom& right);

double getCoeff() const { return coeff; }

int getPower() const { return power; }

void setCoeff(double Coeff) { coeff = Coeff; }

void setPower(int Power) { power = Power; }

private:

double coeff;

int power;

};

class Polynomial

{

public:

Polynomial(const Polynomial& copyPol);

Polynomial() : size(0) {}

friend std::istream& operator>> (std::istream& in, Polynomial& obj);

friend std::ostream& operator<< (std::ostream& out, const Polynomial& obj);

Polynomial& operator=(const Polynomial&);

Polynomial operator+(const Polynomial& right);

Polynomial operator-(const Polynomial& right);

Polynomial operator\*(const Polynomial& right);

double calculate(int x);

int getSize() const { return size; }

void setPol(std::string pol);

Monom returnMonom(int index) const;

private:

List<Monom> data;

int size;

void addMonom(Monom);

};

## Polynomial.cpp

#include "Polynomial.h"

#include <string>

#include <cmath>

Monom::Monom(const Monom& copyMonom) :

coeff(copyMonom.coeff), power(copyMonom.power) {}

Monom::Monom(std::string Coeff, std::string Power)

{

coeff = stod(Coeff);

power = stoi(Power);

}

Monom::Monom(double Coeff, int Power)

{

coeff = Coeff;

power = Power;

}

Monom Monom::operator+(const Monom& right)

{

Monom temp = \*this;

temp.setCoeff(coeff + right.coeff);

return temp;

}

Monom Monom::operator-(const Monom& right)

{

Monom temp = \*this;

temp.setCoeff(coeff - right.coeff);

return temp;

}

Monom Monom::operator\*(const Monom& right)

{

Monom temp = \*this;

temp.setCoeff(coeff \* right.coeff);

temp.setPower(power + right.power);

return temp;

}

bool operator==(const Monom& leftMon, const Monom& rightMon)

{

if (leftMon.coeff == rightMon.coeff && leftMon.power == rightMon.power)

return true;

return false;

}

Polynomial::Polynomial(const Polynomial& copyPol) :

size(copyPol.size), data(copyPol.data){}

Polynomial& Polynomial::operator=(const Polynomial& copyPol)

{

if (this == &copyPol)

return \*this;

size = copyPol.size;

data = copyPol.data;

return \*this;

}

Polynomial Polynomial::operator+(const Polynomial& right)

{

Polynomial temp = \*this;

bool\* isRightMonomUsed = new bool[right.size - 1];

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

isRightMonomUsed[i] = 0;

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

{

int power = temp.data.getElement(i).getPower();

for(int j = 0; j < right.size; j++)

if (right.data.getElement(j).getPower() == power)

{

temp.data.changeElement(i, temp.data.getElement(i) + right.data.getElement(j));

isRightMonomUsed[j] = 1;

break;

}

}

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

if (isRightMonomUsed[i] == 0)

temp.addMonom(right.returnMonom(i));

//delete[] isRightMonomUsed;

return temp;

}

Polynomial Polynomial::operator-(const Polynomial& right)

{

Polynomial temp = \*this;

bool\* isRightMonomUsed = new bool[right.size - 1];

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

isRightMonomUsed[i] = 0;

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

{

int power = temp.data.getElement(i).getPower();

for (int j = 0; j < right.size; j++)

if (right.data.getElement(j).getPower() == power)

{

temp.data.changeElement(i, temp.data.getElement(i) - right.data.getElement(j));

isRightMonomUsed[j] = 1;

break;

}

}

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

if (isRightMonomUsed[i] == 0)

{

Monom temp2 = right.returnMonom(i);

temp2.setCoeff(-temp2.getCoeff());

temp.addMonom(temp2);

}

//delete[] isRightMonomUsed;

return temp;

}

Polynomial Polynomial::operator\*(const Polynomial& right)

{

Polynomial temp;

for (int i = 0; i < this->size; i++)

for (int j = 0; j < right.size; j++)

temp.addMonom(this->returnMonom(i) \* right.returnMonom(j));

return temp;

}

double Polynomial::calculate(int x)

{

double result = 0;

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

result += data.getElement(i).getCoeff() \* pow(x, data.getElement(i).getPower());

return result;

}

void Polynomial::setPol(std::string pol)

{

int i = 0;

while (i < pol.size())

{

std::string coeff = "";

std::string power = "";

while (pol[i] != '\*')

coeff += pol[i++];

i += 3;

while ((pol[i] != '+' && pol[i] != '-') && i != pol.size())

power += pol[i++];

data.push\_back(Monom(coeff, power));

size++;

}

}

Monom Polynomial::returnMonom(int index) const

{

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

throw("Wrong index");

return Monom(data.getElement(index).getCoeff(), data.getElement(index).getPower());

}

void Polynomial::addMonom(Monom mon)

{

data.push\_back(mon);

size++;

}

std::istream& operator>>(std::istream& in, Polynomial& obj)

{

std::string pol;

in >> pol;

int i = 0;

while (i < pol.size())

{

std::string coeff = "";

std::string power = "";

while (pol[i] != '\*')

coeff += pol[i++];

i += 3;

while ((pol[i] != '+' && pol[i] != '-') && i != pol.size())

power += pol[i++];

obj.data.push\_back(Monom(coeff, power));

obj.size++;

}

return in;

}

std::ostream& operator<<(std::ostream& out, const Polynomial& obj)

{

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

{

out << obj.data.getElement(i).getCoeff() << "\*x^"

<< obj.data.getElement(i).getPower();

if (i != obj.size - 1 && obj.data.getElement(i + 1).getCoeff() >= 0)

out << "+";

}

return out;

}

## main.cpp

#include <iostream>

#include "Polynomial.h"

using namespace std;

int main()

{

try

{

cout << "Format: 1\*x^2+1.5\*x^0\n";

Polynomial pol1;

cout << "Enter first polynomial:\n";

cin >> pol1;

Polynomial pol2;

cout << "Enter second polynomial:\n";

cin >> pol2;

cout << "Enter x:\n";

double x;

cin >> x;

cout << "Sum: " << pol1 + pol2;

cout << "\nDifference: " << pol1 - pol2;

cout << "\nMultiply: " << pol1 \* pol2;

cout << "\nPolynom1(x) = " << pol1.calculate(x);

cout << "\nPolynom2(x) = " << pol2.calculate(x);

}

catch (...)

{

cout << "Error...";

}

}