**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Программирование на основе классов и шаблонов»

Отчет по лабораторной работе №6

«Полиномы»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-23Б |  | преподаватель каф. ИУ5 |
| Лачина Екатерина Андреевна |  | Аладин Д.В. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2024 г.

1. **Задание**

Написать программу ввода и оперирования полиномами, состоящими из термов. Для этого разработать классы Term и Polynomial, описание которых представлено ниже.

В работе запрещено использование [контейнеров STL](https://iu5edu.ru/wiki/cpp2/docs/lectures/stl/) (vector, list и др.).

**Ввод полинома**[**​**](https://iu5edu.ru/wiki/cpp2/docs/labs/lab6/#%D0%B2%D0%B2%D0%BE%D0%B4-%D0%BF%D0%BE%D0%BB%D0%B8%D0%BD%D0%BE%D0%BC%D0%B0)

* Термы полинома могут вводиться в любом порядке.
* Во вводимом терме может присутствовать отрицательные коэффициенты, например, -1.
* Терм (член полинома одного порядка) может складываться с другим термом. Например: 3x^2 – x^2, -3x^2 + x^2.
* Пробелы при вводе могут появляться где угодно.

Обратите внимание, что по заданию вы должны уметь работать со строками такого вида:

3x^2 - x^2 - 3x^2 + x ^2 + 5x^5 - 4x^3 +x^2 - 7

Результат разбора такой строки - **полином**. Этот разбор вы можете сделать в перегрузке ввода у полинома. В рамках данной перегрузки строку можно **сначала разобрать на термы**, потом **термы сложить, и получить полином**. При сложении вы можете вызывать перегрузку сложения полинома с термом.

Для разбора на термы можно использовать самостоятельно разработанный вектор из [предыдущей работы](https://iu5edu.ru/wiki/cpp2/docs/labs/lab5/) или динамический массив.

В работе **запрещено осуществлять парсинг конкретного терма в рамках полинома**. Парсинг конкретного терма необходимо делегировать классу Term. Иными словами, полином должен только детектировать термы в строке.

**Класс Term[​](https://iu5edu.ru/wiki/cpp2/docs/labs/lab6/" \l "%D0%BA%D0%BB%D0%B0%D1%81%D1%81-term" \o "Прямая ссылка на класс-term)**

* Целые члены-данные для коэффициента и показателя степени.
* Три конструктора (можно обойтись одним):
  + Без параметров для представления 0𝑥00*x*0.
  + С одним параметром, например 3, для представления 3𝑥03*x*0.
  + С двумя параметрами, например 3 и 2, для представления 3𝑥23*x*2.
* Перегруженный operator+, который получает 2 терма как параметры и возвращает терм-результат.
* Перегруженную операцию istream>> для поддержки вводна полинома в виде, определенном выше в разделе "Ввод полинома".
* Перегруженную операцию ostream<< для печати терма в виде:
  + 3𝑥03*x*0 как 3,
  + 3𝑥13*x*1 как 3x,
  + 1𝑥31*x*3 как x^3,
  + −3𝑥2−3*x*2 как -3x^2.
* Дружественный класс Polynomial.

**Класс Polynomial[​](https://iu5edu.ru/wiki/cpp2/docs/labs/lab6/" \l "%D0%BA%D0%BB%D0%B0%D1%81%D1%81-polynomial" \o "Прямая ссылка на класс-polynomial)**

* Члены-данные poly (динамический массив или сортированный список), и целое degree (степень)
* Три конструктора:
  + Без параметров для представления полинома 00.
  + С одним целым параметром, например 3, для представления полинома 33.
  + С одним параметром-термом, например Term(3, 2), для представления полинома 3𝑥23*x*2.
* Конструктор копирования и операторы присваивания =, +=, \*=.
* Скрытый член order\_ для указания способа хранения термов:
  + по возрастанию степени;
  + по убыванию степени.
* Друзья класса: operator\*, operator+ , каждый из которых получает 2 полинома как параметры и возвращает полином-результат.
* Перегруженную операцию istream>> для ввода полинома в виде, определенном выше в разделе "Ввод полинома".
* Друг класса ostream<< для печати полинома 5𝑥5−4𝑥3+1𝑥2+0𝑥1−7𝑥05*x*5−4*x*3+1*x*2+0*x*1−7*x*0 в виде: 5x^5 - 4x^3 +x^2 - 7.

1. **Листинг**

**Папка Laba2sem6**

**CMakeLists.txt**

cmake\_minimum\_required(VERSION 3.23)

set(project "LabaSix2")

project(**${project}**)

set(CMAKE\_CXX\_STANDARD 17)

set(**${project}**\_SOURCES

polynominal.cpp term.cpp)

set(**${project}**\_HEADERS

polynominal.h term.h)

set(**${project}**\_SOURCE\_LIST

${**${project}**\_SOURCES}

${**${project}**\_HEADERS})

add\_library(**${project}**

STATIC

${**${project}**\_SOURCE\_LIST})

**term.h**

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <cstring>

char\* trimSpaces(char\* str);

class Term {

private:

int degree;

int coefficient;

public:

friend class Polynomial;

Term();

Term(int coef, int deg);

Term(const Term& other); //copy

Term(char\* str);

void parse(char\* buf);

friend std::ostream& operator<<(std::ostream& out, const Term& t);

friend std::istream& operator>>(std::istream& in, Term& t);

friend Term operator+(Term& t1, Term& t2);

friend Term operator\*(Term& t1, Term& t2);

Term& operator=(const Term& t);

int getDegree() const { return degree; }

int getCoefficient() const { return coefficient; }

};

**term.cpp**

#include "term.h"

char\* trimSpaces(char\* str) {

char buffer[256];

int i = 0, j = 0;

while (str[i] != '\0') {

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

buffer[j++] = str[i];

}

i++;

}

buffer[j] = '\0';

delete[] str;

str = new char[strlen(buffer) + 1];

std::strcpy(str, buffer);

return str;

}

Term::Term() : coefficient(0), degree(0) {}

Term::Term(int coef, int deg) : coefficient(coef), degree(deg) {}

Term::Term(const Term& other) : coefficient(other.coefficient), degree(other.degree) {}

Term::Term(char\* str) {

parse(str);

}

void Term::parse(char\* buf) {

char buffer[256];

int index = 0, counter = 0;

coefficient = 0;

degree = 0;

if (buf[0] == '-' && buf[1] == 'x') {

coefficient = -1;

counter = 1;

}

else if (buf[0] == 'x') {

coefficient = 1;

counter = 0;

}

else if (buf[0] == '+' && buf[1] == 'x') {

coefficient = 1;

counter = 1;

}

else if (buf[0] == '0' && buf[1] == 'x' && buf[1] != '^') {

coefficient = 0;

counter = 1;

}

else if (buf[0] == '0' && buf[1] == 'x' && buf[1] == '^' && buf[2] == '0') {

coefficient = 0;

counter = 0;

}

else if (buf[0] == '-' && buf[1] == '0' && buf[1] == 'x' && buf[2] == '^' && buf[3] == '0') {

coefficient = 0;

counter = 0;

}

else {

for (int i = 0; i < 256; ++i) {

if (buf[i] == 'x' || buf[i] == '\0') {

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

buffer[j] = buf[j];

buffer[i] = '\0';

coefficient = atoi(buffer);

if (buf[i] == '\0') {

counter = -1;

degree = 0;

}

else {

counter = i;

}

break;

}

}

}

if (counter != -1) {

if (buf[counter + 1] == '\0') {

degree = 1;

}

else {

counter += 2;

for (int i = counter; i < 256; ++i) {

if (buf[i] == '\0') {

for (int j = counter; j < i; ++j)

buffer[j - counter] = buf[j];

buffer[i - counter] = '\0';

degree = atoi(buffer);

break;

}

}

}

}

}

Term operator+(Term& t1, Term& t2) {

if (t1.degree == t2.degree)

return { t1.coefficient + t2.coefficient, t1.degree };

else {

std::cout << "Ошибка, сумма термов разной степени" << std::endl;

return { 0, 0 };

}

}

Term operator\*(Term& t1, Term& t2) {

Term result;

result.coefficient = t1.coefficient \* t2.coefficient;

result.degree = t1.degree + t2.degree;

return result;

}

Term& Term::operator=(const Term& t) {

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

this->coefficient = t.coefficient;

this->degree = t.degree;

return \*this;

}

std::ostream& operator<<(std::ostream& out, const Term& t) {

if (t.coefficient == 0) {

return out;

}

else {

char sign = '+';

if (t.coefficient < 0)

sign = '-';

int absCoef = abs(t.coefficient);

if (absCoef == 1 && t.degree == 1)

out << sign << "x ";

else if (t.degree == 0)

out << sign << absCoef << ' ';

else if (t.degree == 1)

out << sign << absCoef << "x ";

else if (absCoef == 1)

out << sign << "x^" << t.degree << ' ';

else

out << sign << absCoef << "x^" << t.degree << ' ';

return out;

}

}

std::istream& operator>>(std::istream& in, Term& t) {

char\* buf = new char[256];

in.getline(buf, 256);

buf = trimSpaces(buf);

t.parse(buf);

delete[] buf;

return in;

}

**polynominal.cpp**

#include "polynominal.h"

#include <cstring>

Polynomial::Polynomial(Term term, int num, int cap, int sortOrder) {

capacity = cap;

count = num;

terms = new Term[capacity];

terms[0] = term;

sortingOrder = sortOrder;

}

Polynomial::Polynomial() {

Term term;

count = 0;

capacity = 1;

sortingOrder = 1;

terms = new Term[capacity];

}

Polynomial::~Polynomial() {

delete[] terms;

}

Polynomial::Polynomial(const Polynomial& other) {

sortingOrder = other.sortingOrder;

capacity = other.capacity;

count = other.count;

terms = new Term[capacity];

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

terms[i] = other.terms[i];

}

void Polynomial::addTerm(Term term) {

if (count < capacity) {

terms[count] = term;

count++;

}

else {

Term\* newTerms = new Term[count + 1];

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

newTerms[i] = terms[i];

delete[] terms;

newTerms[count] = term;

terms = newTerms;

count++;

capacity++;

}

}

void Polynomial::addTerm(int coef, int exp) {

Term term(coef, exp);

if (count < capacity) {

terms[count] = term;

count++;

}

else {

Term\* newTerms = new Term[count + 1];

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

newTerms[i] = terms[i];

delete[] terms;

newTerms[count] = term;

terms = newTerms;

count++;

capacity++;

}

}

void Polynomial::removeTerm(int index) {

for (int i = index; i < count - 1; ++i)

terms[i] = terms[i + 1];

--count;

}

void Polynomial::sort() {

for (int i = 0; i < count; ++i) {

for (int j = i + 1; j < count; ++j) {

if ((terms[i].getDegree()) \* sortingOrder < (terms[j].getDegree()) \* sortingOrder) {

Term temp(terms[i]);

terms[i] = terms[j];

terms[j] = temp;

}

}

}

addTerm(0, -1);

int i = 0;

int j = 0;

int k = 0;

while (terms[i].getDegree() != -1) {

if (terms[i + 1].getDegree() != terms[i].getDegree()) {

i++;

j = i;

k = 0;

}

else {

while (terms[j].getDegree() == terms[i].getDegree()) {

j++;

k++;

}

for (int h = i + 1; h < j; ++h) {

terms[i].coefficient += terms[h].coefficient;

}

for (int h = 1; h < k; ++h)

removeTerm(i + 1);

i++;

j = i;

k = 0;

}

}

removeTerm(count - 1);

}

Polynomial& Polynomial::add(Polynomial& other) {

Term\* newTerms = new Term[count + other.count];

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

newTerms[i] = terms[i];

for (int i = count; i < count + other.count; ++i)

newTerms[i] = other.terms[i - count];

count = count + other.count;

capacity = count;

terms = newTerms;

sort();

return \*this;

}

Polynomial& Polynomial::multiply(Polynomial& other) {

Term\* newTerms = new Term[count \* other.count];

int k = 0;

for (int i = 0; i < other.count; ++i) {

for (int j = 0; j < count; ++j) {

Term temp(terms[j] \* other.terms[i]);

newTerms[k] = temp;

k++;

}

}

delete[] terms;

count = count \* other.count;

capacity = count;

terms = newTerms;

sort();

return \*this;

}

Polynomial operator+(Polynomial& p1, Polynomial& p2) {

Polynomial result;

result.count = p1.count + p2.count;

result.capacity = result.count;

result.sortingOrder = (p1.sortingOrder == -1 && p2.sortingOrder == -1) ? -1 : 1;

delete[] result.terms;

result.terms = new Term[result.count];

for (int i = 0; i < p1.count; ++i) {

result.terms[i] = p1.terms[i];

}

for (int i = p1.count; i < result.count; ++i) {

result.terms[i] = p2.terms[i - p1.count];

}

result.sort();

return result;

}

Polynomial operator\*(Polynomial& p1, Polynomial& p2) {

Polynomial result;

delete[] result.terms;

result.terms = new Term[p1.count \* p2.count];

int k = 0;

for (int i = 0; i < p1.count; ++i) {

for (int j = 0; j < p2.count; ++j) {

Term temp(p1.terms[i] \* p2.terms[j]);

result.terms[k] = temp;

k++;

}

}

result.count = p1.count \* p2.count;

result.capacity = result.count;

result.sort();

return result;

}

std::istream& operator>>(std::istream& in, Polynomial& p) {

char\* buffer = new char[256];

char\* termBuffer = new char[256];

in.getline(buffer, 256);

buffer = trimSpaces(buffer);

int i = 0;

int j = 0;

while (buffer[i] != '\0') {

j = i;

do

j++;

while (buffer[j] != '-' && buffer[j] != '+' && buffer[j] != '\0');

for (int h = i; h <= j; ++h)

termBuffer[h - i] = buffer[h];

termBuffer[j - i] = '\0';

Term term(termBuffer);

p.addTerm(term);

i = j;

}

p.sortingOrder = 1;

delete[] buffer;

delete[] termBuffer;

return in;

}

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

for (int i = 0; i < p.count; ++i) {

out << p.terms[i];

}

out << '\n';

return out;

}

Polynomial& Polynomial::operator=(const Polynomial& other) {

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

sortingOrder = other.sortingOrder;

count = other.count;

capacity = count;

delete[] terms;

terms = new Term[count];

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

terms[i] = other.terms[i];

return \*this;

}

Polynomial& Polynomial::operator\*=(Polynomial& other) {

return multiply(other);

}

Polynomial& Polynomial::operator+=(Polynomial& other) {

return add(other);

}

**polynominal.h**

#pragma once

#include "term.h"

class Polynomial {

private:

Term\* terms;

int capacity, count;

int sortingOrder;

public:

Polynomial(Term term, int num, int cap, int sortOrder);

Polynomial();

~Polynomial();

Polynomial(const Polynomial& other); // copy

void addTerm(Term term);

void removeTerm(int index);

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

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

Polynomial& add(Polynomial& other);

void addTerm(int coef, int exp);

Polynomial& multiply(Polynomial& other);

void sort();

Polynomial& operator\*=(Polynomial& other);

Polynomial& operator=(const Polynomial& other);

Polynomial& operator+=(Polynomial& other);

friend Polynomial operator+(Polynomial& p1, Polynomial& p2);

friend Polynomial operator\*(Polynomial& p1, Polynomial& p2);

};

**main.cpp**

#include <iostream>

#include <cstring>

#include "LabaSix2/term.h"

#include "LabaSix2/polynominal.h"

int main() {

Polynomial x1, x2;

std::cin >> x1;

std::cin >> x2;

Polynomial x3;

x3 = x1 \* x2;

std::cout << "x1 \* x2 = " << x3 << '\n';

x3 = x1 + x2;

std::cout << "x1 + x2 = " << x3 << '\n';

Polynomial x4;

x4 = x2;

x4 \*= x1;

std::cout << x4 << std::endl;

Polynomial x5;

x5 = x2;

x5 += x1;

std::cout << x5 << std::endl;

return 0;}

1. **Тестирование**

   

  