**Московский государственный технический**

**университет им. Н.Э. Баумана.**

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Программирование на основе классов и шаблонов»

Отчет по лабораторной работе №6

«Написать программу ввода и оперирования полиномами, состоящими из термов. Использовать классы Term and Polynomial»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-21 |  | преподаватель каф. ИУ5 |
| Агличеев Михаил |  | Селиверстова А.В. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2020 г.

1. Постановка задачи.

Ввод полинома

 Термы полинома могут вводиться в любом порядке.

 Во вводимом терме может присутствовать коэффициент -1.

 Терм (член полинома одного порядка) может складываться с другим термом

(например, допустим ввод 3x^2+(–x^2), -3x^2 + x^2)

 Пробелы при вводе могут появляться где угодно.

класс Term

 Целые члены-данные для коэффициента и показателя степени

 Три конструктора (можно обойтись одним):

o Без параметров для представления 0x 0

o С одним параметром, например 3, для представления 3x 0

o С двумя параметрами, например 3 и 2, для представления 3x 2

 Перегруженный operator +, который получает 2 терма как параметры и возвращает

терм-результат.

 Перегруженную операцию istream&gt;&gt; для ввода терма в виде, определенном выше в

разделе «Ввод полинома».

 Перегруженную операцию ostream &lt;&lt; для печати терма в виде:

3x 0 как 3, 3x 1 как 3x, 1x 3 как x^3, -3x 2 как -3x^2

 Дружественный класс Polynomial

Опишите и протестируйте этот класс до создания класса Polynomial. Представьте main() для

демонстрации работы этого класса независимо от Polynomial.

класс Polynomial

 Члены-данные poly (массив из 6 термов или сортированный список), и целое degree

(степень)

 Три конструктора

o Без параметров для представления полинома 0

o С одним целым параметром, например 3, для представления полинома 3

o С одним параметром-термом, например Term(3,2), для представления

полинома 3x 2

 Конструктор копирования и операторы присваивания =, +=, \*=

 Скрытый член Order для указания способа хранения термов: по возрастанию или

убыванию степени.

 Друзья класса: operator \*, operator +, каждый из которых получает 2 полинома как

параметры и возвращает полином-результат.

 Друг класса ostream &lt;&lt; - для печати полинома 5x 5 - 4x 3 + 1x 2 + 0x 1 - 7x 0

в виде: 5x^5 - 4x^3 + x^2 - 7

1. Разработка Интерфейса класса

Поля класса Term:

Private: int multiplier – множитель терма (Например, для терма “3x^5”, это 3)

Size\_t power – Показатель терма (Для аналогичного терма, это 5)

Методы Класса Term:

Public: Term(int multiplier = 0, int power = 0) – конструктор терма из множителя и показателя, просто множителя, по умолчанию

Term(string& line) – Конструктор, преобразующий строку с **1 термом**

~Term() = default – Так как память нигде не выделяется, используется деструктор по умолчанию

friend Term operator+(const Term& first, const Term& second) – оператор сложения термов

friend istream& operator>>(istream& stream, Term& term) – оператор вывода терма в поток

friend ostream& operator<<(ostream& stream, const Term& term) – оператор ввода терма из потока

Поля класса Polynomial:

Private: vector<Term> poly – вектор (динамический массив), содержащий в себе термы – составляющие полинома

size\_t degree – показатель полинома – степень наибольшего из его термов

bool order – переменная, указывающая, отсортирован ли poly по возрастанию (true) или убыванию (false)

Методы Класса Polynomial:

Public: Polynomial(int multiplier = 0) – конструктор создания полинома, представляющего собой некое число, конструктор по умолчанию

Polynomial(Term& term) – конструктор создания полинома из 1 терма

Polynomial(const Polynomial& other) – конструктор копирования

void operator=(const Polynomial& other) – оператор приравнивания

void operator+=(const Polynomial& other) – оператор суммы полиномов

void operator\*=(const Polynomial& other) – оператор умножения полиномов

friend Polynomial operator+(const Polynomial& first, const Polynomial& second) – оператор суммы полиномов

friend Polynomial operator\*(const Polynomial& first, const Polynomial& second) – оператор умножения полиномов

friend istream& operator>>(istream& stream, Polynomial& polynomial) – оператор ввода полинома из потока

friend ostream& operator<<(ostream& stream, const Polynomial& polynomial) – оператор вывода полинома в поток

1. Текст программы

Файл term.h:

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Polynomial;

class Term {

private:

int multiplier;

size\_t power;

public:

Term(int multiplier = 0, int power = 0);

Term(string& line);

~Term() = default;

friend Term operator+(const Term& first, const Term& second);

friend istream& operator>>(istream& stream, Term& term);

friend ostream& operator<<(ostream& stream, const Term& term);

friend class Polynomial;

friend istream& operator>>(istream& stream, Polynomial& polynomial);

friend ostream& operator<<(ostream& stream, const Polynomial& polynomial);

};

Файл term.cpp:

#include "Term.h"

#include <string>

using namespace std;

Term::Term(int multiplier, int power) {

this->multiplier = multiplier;

this->power = power;

}

Term::Term(string& line) {

this->multiplier = 0; // Default multiplier and power values.

this->power = 0;

size\_t multiplier\_end = 0;

if (line.length()) {

if (line[0] == '-') {

++multiplier\_end; // This fixes bug occuring when reading negative terms.

}

}

for (multiplier\_end; multiplier\_end < line.length(); ++multiplier\_end) {

if (!isdigit(line[multiplier\_end])) {

break;

}

}

if (!multiplier\_end) {

if (line[0] == 'x') {

this->multiplier = 1;

if (line == "x") {

this->power = 1;

return;

}

}

else {

cerr << "Error: bad format" << endl;

return;

}

}

else if (multiplier\_end == line.length()) {

this->multiplier = stoi(line);

return;

}

else if (multiplier\_end > line.length() - 2) {

if (multiplier\_end == line.length() - 1 && line[multiplier\_end] == 'x') {

string multiplier(line, 0, multiplier\_end);

this->multiplier = stoi(multiplier);

this->power = 1;

return;

}

cerr << "Error: bad format" << endl;

return;

}

else if (line[multiplier\_end] != 'x' || line[multiplier\_end + 1] != '^') {

cerr << "Error: bad format" << endl;

return;

}

else {

string multiplier(line, 0, multiplier\_end);

this->multiplier = stoi(multiplier);

}

size\_t power\_ind = multiplier\_end + 2;

for (power\_ind; power\_ind < line.length(); ++power\_ind) {

if (!isdigit(line[power\_ind])) {

cerr << "Error: bad format" << endl;

return;

}

}

string power(line, multiplier\_end + 2, string::npos);

this->power = stoi(power);

}

Term operator+(const Term& first, const Term& second) {

Term result;

if (first.power != second.power) {

return result;

}

result.multiplier = first.multiplier + second.multiplier;

result.power = first.power;

return result;

}

istream& operator>>(istream& stream, Term& term) {

string line;

stream >> line;

term = Term(line);

return stream;

}

ostream& operator<<(ostream& stream, const Term& term) {

if (!term.power) {

stream << term.multiplier;

return stream;

}

switch (term.multiplier) {

case 1:

break;

case 0:

stream << 0;

return stream;

case -1:

stream << '-';

break;

default:

stream << term.multiplier;

break;

}

stream << 'x';

if (term.power != 1) {

stream << '^' << term.power;

}

return stream;

}

Файл polynomial.h:

#pragma once

#include <iostream>

#include <vector>

#include "Term.h"

using namespace std;

class Polynomial {

private:

vector<Term> poly;

size\_t degree;

bool order; // True if ascending order, false otherwise

public:

Polynomial(int multiplier = 0);

Polynomial(Term& term);

Polynomial(const Polynomial& other);

void operator=(const Polynomial& other);

void operator+=(const Polynomial& other);

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

friend Polynomial operator+(const Polynomial& first, const Polynomial& second);

friend Polynomial operator\*(const Polynomial& first, const Polynomial& second);

friend istream& operator>>(istream& stream, Polynomial& polynomial);

friend ostream& operator<<(ostream& stream, const Polynomial& polynomial);

};

Файл polynomial.cpp:

#include <iostream>

#include <vector>

#include <algorithm>

#include <string>

#include "Polynomial.h"

Polynomial::Polynomial(int multiplier) {

poly.resize(1);

poly[0].multiplier = multiplier;

degree = 0;

order = true;

}

Polynomial::Polynomial(Term& term) {

poly.resize(term.power + 1);

poly[term.power] = term;

for (size\_t i = 1; i <= term.power; ++i) {

poly[i].power = i;

}

degree = term.power;

order = true;

}

Polynomial::Polynomial(const Polynomial& other) {

poly = other.poly; // That copies poly, but resulting poly is entirely separate.

degree = other.degree;

order = other.order;

}

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

poly = other.poly; // That copies poly, but resulting poly is entirely separate.

degree = other.degree;

order = other.order;

}

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

size\_t new\_degree = max(degree, other.degree);

vector<Term> new\_poly(new\_degree + 1);

for (size\_t i = 0; i <= new\_degree; ++i) {

new\_poly[i].power = i;

}

if (other.order) {

for (size\_t i = 0; i <= other.degree; ++i) {

new\_poly[i] = other.poly[i];

}

}

else {

for (size\_t i = 0; i <= other.degree; ++i) {

new\_poly[i] = other.poly[other.degree - i];

}

}

if (order) {

for (size\_t i = 0; i <= degree; ++i) {

new\_poly[i].multiplier += poly[i].multiplier;

}

}

else {

for (size\_t i = 0; i <= degree; ++i) {

new\_poly[i].multiplier += poly[degree - i].multiplier;

}

reverse(new\_poly.begin(), new\_poly.end());

}

poly = new\_poly;

degree = new\_degree;

}

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

vector<Term> new\_poly(degree + other.degree + 1);

for (size\_t i = 0; i <= degree + other.degree; ++i) {

new\_poly[i].power = i;

}

if (order) {

if (other.order) {

for (size\_t i = 0; i <= degree; ++i) {

for (size\_t j = 0; j <= other.degree; ++j) {

new\_poly[i + j].multiplier += poly[i].multiplier \* other.poly[j].multiplier;

}

}

}

else {

for (size\_t i = 0; i <= degree; ++i) {

for (size\_t j = 0; j <= other.degree; ++j) {

new\_poly[i + j].multiplier += poly[i].multiplier \* other.poly[other.degree - j].multiplier;

}

}

}

}

else {

if (other.order) {

for (size\_t i = 0; i <= degree; ++i) {

for (size\_t j = 0; j <= other.degree; ++j) {

new\_poly[i + j].multiplier += poly[degree - i].multiplier \* other.poly[j].multiplier;

}

}

}

else {

for (size\_t i = 0; i <= degree; ++i) {

for (size\_t j = 0; j <= other.degree; ++j) {

new\_poly[i + j].multiplier += poly[degree - i].multiplier \* other.poly[other.degree - j].multiplier;

}

}

}

reverse(new\_poly.begin(), new\_poly.end());

}

poly = new\_poly;

degree += other.degree;

}

Polynomial operator+(const Polynomial& first, const Polynomial& second) {

Polynomial result(first);

result += second;

return result;

}

Polynomial operator\*(const Polynomial& first, const Polynomial& second) {

Polynomial result(first);

result \*= second;

return result;

}

istream& operator>>(istream& stream, Polynomial& polynomial) {

string poly\_string;

getline(stream, poly\_string);

poly\_string.erase(remove(poly\_string.begin(), poly\_string.end(), ' '), poly\_string.end());

polynomial.poly.resize(1);

polynomial.poly[0].multiplier = 0;

polynomial.degree = 0;

polynomial.order = true;

while (!poly\_string.empty()) {

bool negative\_term = false;

if (poly\_string[0] == '-') {

negative\_term = true;

poly\_string.erase(0, 1);

}

size\_t plus\_pos = poly\_string.find('+');

size\_t minus\_pos = poly\_string.find('-');

Term term;

if (plus\_pos < minus\_pos) {

string term\_string = poly\_string.substr(0, plus\_pos);

poly\_string.erase(0, plus\_pos + 1);

if (term\_string[0] == '(') {

// Removing brackets.Backwards one is at the end, so we delete it first (it's faster).

term\_string.erase(term\_string.length() - 1);

term\_string.erase(0);

}

term = Term(term\_string);

}

else if (plus\_pos > minus\_pos) {

string term\_string = poly\_string.substr(0, minus\_pos);

poly\_string.erase(0, minus\_pos);

term = Term(term\_string);

}

else { // In that case, there are no pluses or minuses - we have our last term.

term = Term(poly\_string);

poly\_string = "";

}

if (negative\_term) {

term.multiplier \*= -1;

}

polynomial += Polynomial(term);

}

return stream;

}

ostream& operator<<(ostream& stream, const Polynomial& polynomial) {

bool first\_term = true;

vector<Term> new\_poly = polynomial.poly;

if (polynomial.order) {

reverse(new\_poly.begin(), new\_poly.end());

}

if (polynomial.degree) {

if (new\_poly[0].multiplier) {

if (new\_poly[0].multiplier > 0) {

stream << new\_poly[0] << ' ';

}

else {

stream << "- " << Term(-new\_poly[0].multiplier, new\_poly[0].power) << ' ';

}

}

for (size\_t i = 1; i < polynomial.degree; ++i) {

if (new\_poly[i].multiplier > 0) {

stream << "+ " << new\_poly[i] << ' ';

}

else if (new\_poly[i].multiplier < 0) {

stream << "- " << Term(-new\_poly[i].multiplier, new\_poly[i].power) << ' ';

}

}

}

else {

stream << '0';

}

return stream;

}

Файл main.cpp:

#include "Polynomial.h"

#include <iostream>

using namespace std;

int main() {

Polynomial first, second, third;

cout << "Input first polynomial" << endl;

cin >> first;

cout << "Input second polynomial" << endl;

cin >> second;

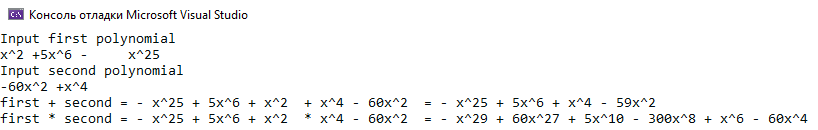
cout << "first + second = " << first << " + " << second << " = " << first + second << endl;

cout << "first \* second = " << first << " \* " << second << " = " << first \* second << endl;

return 0;

}

5. Анализ результатов



Вычисления проводятся программой верно, причём, как видно из примера, тут есть и начало строки с отрицательного терма, и положительные и отрицательные 2-е и последующие термы, множество пробельных символов, что позволяет заключить о верности работы программы.