**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 |
| Поляков Данила |  | Правдина А.Д. |
| Подпись и дата: 17.05.2022 |  | Подпись и дата: 17.05.2022 |

Москва, 2022 г.

**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):
  + Без параметров для представления 0x0
  + С одним параметром, например 3, для представления 3x0
  + С двумя параметрами, например 3 и 2, для представления 3x2
* Перегруженный **operator +**, который получает 2 терма как параметры и возвращает терм-результат.

### Перегруженную операцию istream>> для ввода терма в виде, определенном выше в разделе «Ввод полинома».

* Перегруженную операцию **ostream <<** для печати терма в виде:  
  **3x0** как **3**, **3x1** как **3x**, **1x3** как **x^3**, **-3x2**  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
  + Без параметров для представления полинома 0
  + С одним целым параметром, например 3, для представления полинома 3
  + С одним параметром-термом, например **Term(3,2)**, для представления полинома **3x2**
* 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 << -** для печати полинома **5x5 - 4x3 + 1x2 + 0x1 - 7x0**

в виде: **5x^5 - 4x^3 + x^2 - 7**

**Текст программы**

**Листинг кода программы:**

**Term.hpp**

*// Copyright 2022 qqq <polyakovdd@student.bmstu.ru>  
  
  
#ifndef* CODE\_TERM\_HPP  
*#define* CODE\_TERM\_HPP  
  
*#include* <sstream>  
*#include* <iostream>  
  
*class* Polynomal;  
  
*class* Term {  
*protected*:  
 *int* \_coefficient = 0;  
 size\_t \_power = 0;  
  
 *friend class* Polynomal;  
  
*public*:  
 Term() = *default*;  
  
 Term(*int* coeff) : \_coefficient(coeff) {};  
  
 Term(*int* coeff, size\_t power) : \_coefficient(coeff), \_power(power) {};  
  
 Term &*operator*+=(*const* Term &term);  
  
 *friend* std::ostream &*operator*<<(std::ostream &out, *const* Term &term);  
  
 *friend* std::istream &*operator*>>(std::istream &in, Term &term);  
  
 *inline int* getCoeff() { *return* \_coefficient; }  
};  
  
  
Term *operator*+(Term s1, Term s2);  
  
  
*#endif //CODE\_TERM\_HPP*

**Polynomal.hpp**

*// Copyright 2022 qqq <polyakovdd@student.bmstu.ru>  
  
  
#ifndef* CODE\_POLYNOMAL\_HPP  
*#define* CODE\_POLYNOMAL\_HPP  
  
  
*#include* "MyVector.hpp"  
*#include* "Term.hpp"  
  
*class* Polynomal {  
 MyVector<Term> \_terms = {};  
 size\_t \_maxpower = 0;  
  
*public*:  
 Polynomal() = *default*;  
  
 Polynomal(*int* c, size\_t power = 0) : \_terms(Term(c, power)) {};  
  
 Polynomal(Term term) : \_maxpower(term.\_power) {  
 *for* (size\_t i = 0; i < term.\_power; ++i) {  
 \_terms.add\_element(Term(0, i));  
 }  
 \_terms.add\_element(term);  
 };  
  
 Polynomal(size\_t power) : \_maxpower(power) {  
 *for* (size\_t i = 0; i < power+1; ++i) {  
 \_terms.add\_element(Term(0, i));  
 }  
 };  
  
 Polynomal(*const* Polynomal &poly) : \_maxpower(poly.\_maxpower), \_terms(poly.\_terms) {};  
  
 Polynomal &*operator*=(*const* Polynomal &poly) = *default*;  
  
 Polynomal &*operator*+=(*const* Polynomal &);  
 Polynomal &*operator*+=(*const* Term &);  
  
 Polynomal &*operator*\*=(*const* Polynomal &);  
  
 *friend* std::ostream &*operator*<<(std::ostream &out, *const* Polynomal &polynomal);  
  
 *friend* std::istream &*operator*>>(std::istream &in, Polynomal &polynomal);  
};  
  
Polynomal *operator*+(Polynomal, Polynomal);  
  
Polynomal *operator*\*(Polynomal, Polynomal);  
  
  
*#endif //CODE\_POLYNOMAL\_HPP*

**Term.hpp**

*// Copyright 2022 qqq <polyakovdd@student.bmstu.ru>  
  
  
#include* "Term.hpp"  
*#include* "Polynomal.hpp"  
  
  
std::ostream &*operator*<<(std::ostream &out, *const* Term &term) {  
 *if* (term.\_coefficient == 0) {  
 *return* out;  
 } *else* {  
 *if* (term.\_power == 0) {  
 out << term.\_coefficient;  
 } *else if* (term.\_power == 1) {  
 *if* (term.\_coefficient != 1 && term.\_coefficient != -1) {  
 out << term.\_coefficient << 'x';  
 } *else if* (term.\_coefficient == -1) { out << "-x"; }  
 *else* {  
 out << 'x';  
 }  
 } *else* {  
 *if* (term.\_coefficient != 1 && term.\_coefficient != -1) {  
 out << term.\_coefficient << 'x' << '^' << term.\_power;  
 } *else if* (term.\_coefficient == -1) {  
 out << '-' << 'x' << '^' << term.\_power;  
 } *else* {  
 out << 'x' << '^' << term.\_power;  
 }  
 }  
 }  
 *return* out;  
}  
  
  
std::istream &*operator*>>(std::istream &in, Term &term) {  
 in >> term.\_coefficient;  
 *if* (in.fail()) {  
 in.clear();  
 *//in.ignore();* in.unget();  
 *if* (in.peek() == '-') {  
 term.\_coefficient = -1;  
 in.get();  
 } *else* {  
 term.\_coefficient = 1;  
 in.get();  
 }  
 *if* (in.peek() == 'x') {  
 in.get();  
 *if* (in.peek() == '^') {  
 in.get();  
 in >> term.\_power;  
 *return* in;  
 } *else* {  
 term.\_power = 1;  
 *return* in;  
 }  
 } *else* {  
 term.\_power = 0;  
 *return* in;  
 }  
 } *else* {  
 *if* (in.peek() == 'x') {  
 in.get();  
 *if* (in.peek() == '^') {  
 in.get();  
 in >> term.\_power;  
 *return* in;  
 } *else* {  
 term.\_power = 1;  
 *return* in;  
 }  
 } *else* {  
 term.\_power = 0;  
 *return* in;  
 }  
 }  
 *return* in;  
}  
  
Term &Term::*operator*+=(*const* Term &term) {  
 \_coefficient += term.\_coefficient;  
 *return* \**this*;  
}  
  
Term *operator*+(Term s1, Term s2) {  
 Term result = s1;  
 result += s2;  
 *return* result;  
}

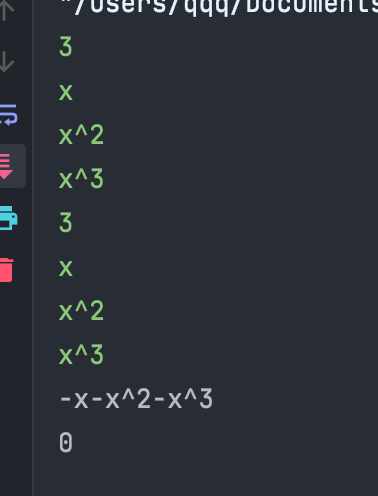
**Polynomal.cpp**

*// Copyright 2022 qqq <polyakovdd@student.bmstu.ru>  
  
#include* "Polynomal.hpp"  
  
std::ostream &*operator*<<(std::ostream &out, *const* Polynomal &polynomal) {  
 size\_t q = 0;  
 *bool* isNull = *true*;  
  
 *for* (size\_t i = 0; i < polynomal.\_maxpower + 1; ++i) {  
 *if* (polynomal.\_terms[i].getCoeff() != 0) {  
 q++;  
 isNull = *false*;  
 }  
 *if* (q > 1 && polynomal.\_terms[i].getCoeff() > 0) {  
 out << '+';  
 }  
 out << polynomal.\_terms[i];  
 }  
 *if* (isNull) { out << 0; }  
 *return* out;  
}  
  
std::istream &*operator*>>(std::istream &in, Polynomal &polynomal) {  
 size\_t count;  
 in >> count;  
 Term temp;  
 *for* (size\_t i = 0; i < count; ++i) {  
 in >> temp;  
 polynomal += temp;  
 }  
 *return* in;  
}  
  
Polynomal &Polynomal::*operator*+=(*const* Term &term) {  
 *if* (term.\_power > \_maxpower) {  
 \_maxpower++;  
 *while* (\_maxpower != term.\_power) {  
 *if* (\_maxpower >= \_terms.Maxsize()) {  
 \_terms.add\_element(Term(0, \_maxpower));  
 } *else* {  
 \_terms[\_maxpower] = Term(0, \_maxpower);  
 }  
 \_maxpower++;  
 }  
 \_terms[\_maxpower] = term;  
 } *else* {  
 \_terms[term.\_power] += term;  
 }  
 *return* \**this*;  
}  
  
Polynomal &Polynomal::*operator*+=(*const* Polynomal &polynomal) {  
 Polynomal result(\**this*);  
 *bool* isNeededToInsert = *true*;  
 *for* (size\_t i = 0; i < polynomal.\_maxpower + 1; ++i) {  
 *for* (size\_t j = 0; j < \_maxpower + 1; ++j) {  
 *if* (polynomal.\_terms[i].\_power == \_terms[j].\_power) {  
 result.\_terms[j] += polynomal.\_terms[i];  
 isNeededToInsert = *false*;  
 *break*;  
 }  
 }  
 *if* (isNeededToInsert) {  
 result.\_terms.add\_element(polynomal.\_terms[i]);  
 }  
 isNeededToInsert = *true*;  
 }  
  
 result.\_maxpower = result.\_terms.Size() - 1;  
 \**this* = result;  
  
 *return* \**this*;  
}  
  
Polynomal &Polynomal::*operator*\*=(*const* Polynomal &polynomal) {  
 MyVector<*int*> newCoeffs(0, polynomal.\_maxpower + \_maxpower);  
 *for* (size\_t i = 0; i < polynomal.\_maxpower + \_maxpower + 1; ++i) {  
 newCoeffs[i] = 0;  
 }  
 *for* (size\_t i = 0; i < \_maxpower + 1; ++i) {  
 *for* (size\_t j = 0; j < polynomal.\_maxpower + 1; ++j) {  
 newCoeffs[i + j] += polynomal.\_terms[j].getCoeff() \* \_terms[i].getCoeff();  
 }  
 }  
 Polynomal result(polynomal.\_maxpower + \_maxpower);  
 *for* (size\_t i = 0; i < polynomal.\_maxpower + \_maxpower + 1; ++i) {  
 result.\_terms[i].\_coefficient = newCoeffs[i];  
 }  
  
 \**this* = result;  
 *return* \**this*;  
  
}  
  
Polynomal *operator*+(Polynomal p1, Polynomal p2) {  
 Polynomal result = p1;  
 result += p2;  
 *return* result;  
}  
  
Polynomal *operator*\*(Polynomal p1, Polynomal p2) {  
 Polynomal result = p1;  
 result \*= p2;  
 *return* result;  
}

**main.cpp**

*#include* <iostream>  
*#include* "Polynomal.hpp"  
  
*int* main() {  
  
 Term a, b;  
 Polynomal ap, bp, ed(Term(-1, 0));  
  
 std::cin >> ap;  
 std::cin >> bp;  
 bp \*= ed;  
 std::cout << bp << '\n';  
 ap += bp;  
 std::cout << ap << '\n';  
 *return* 0;  
}

**Анализ результатов**

****