Московский государственный технический

университет им. Н.Э. Баумана

Проверил:

Преподаватель каф. ИУ5

Подпись и дата:

Факультет «Информатика и система управления»

Кафедра ИУ5 «Система обработки информации и управления»

Выполнил:

Студент группы ИУ5-22Б

Яншин Т. И.

Подпись и дата:

Курс «Программирование на основе классов и шаблонов»

Отчёт по лабораторной работе №6

Полиномы

## **Задание**

Задание

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

ОСТОРОЖНО

В работе запрещено использование контейнеров STL (vector, list и др.).

В данной работе можно использовать самостоятельно разработанный вектор из предыдущей работы. Перед этим настоятельно рекомендуется изучить статью о линковке шаблонного класса и его специализации.

Ввод полинома

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.

К СВЕДЕНИЮ

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

Пробелы расставлены в случайном порядке.

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 или динамический массив.

ОСТОРОЖНО

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

Целые члены-данные для коэффициента и показателя степени.

Три конструктора (можно обойтись одним):

Без параметров для представления

С одним параметром, например 3, для представления

С двумя параметрами, например 3 и 2, для представления

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

Члены-данные poly (динамический массив или сортированный список), и целое degree (степень)

Три конструктора:

Без параметров для представления полинома

С одним целым параметром, например 3, для представления полинома

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

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

## **Текст программы**

Внешний CMakeLists.txt

cmake\_minimum\_required(VERSION 3.23)

set(project "lab6")

project(**${project}**)

set(CMAKE\_CXX\_STANDARD 17)

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

main.cpp)

add\_subdirectory("polinom")

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

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

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

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

target\_link\_libraries(**${project}** polinom)

Внутренний polinom /CMakeLists.txt

cmake\_minimum\_required(VERSION 3.8)

set(project "polinom")

project(**${project}**)

set(CMAKE\_CXX\_STANDARD 17)

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

polinom.cpp)

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

polinom.h

term.h

MyVector.h)

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

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

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

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

STATIC

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

Main.cpp

#include "polinom/polinom.h"

#include <iostream>

#include <cstring>

// Включите необходимые заголовки и используйте правильные пространства имен

int main()

{

std::setlocale(LC\_ALL, "Russian");

char inter[200];

std::cout << "Введите 'd' - для демонстрационного режима или 'i' - для интерактивного: ";

std::cin >> inter;

// Игнорируем оставшийся символ новой строки в буфере

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

if (inter[0] == 'd')

{

MyVector<Term> terms;

char str[] = "3x^2 - 4x + 5 + 3x^2";

std::cout << str << "\n";

// Разбор строки и заполнение вектора термами

Term::parseTerms(str, terms);

Polynomial poly(terms);

std::cout << poly;

}

else

{

std::cout << "Введите терм: ";

Term term;

std::cin >> term;

std::cout << term;

std::cout << "\n";

std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');

std::cout << "Введите полином: ";

Polynomial poly;

std::cin >> poly;

std::cout << poly;

}

return 0;

}

MyVector.h

#include <iostream>

#include <cmath>

const int MAX\_SIZE = 5;

template <typename T>

class MyVector

{

protected:

size\_t max\_size;

size\_t size;

T\* pdata;

public:

MyVector(T el = NULL , int max\_size = MAX\_SIZE);

MyVector(MyVector& vecToCopy);

~MyVector();

void add\_element(T el);

void clear();

bool delete\_element(int index);

size\_t find(T el);

void resize();

void sort();

size\_t get\_size() { return size; }

size\_t get\_max\_size() { return max\_size; }

T\* begin() { return pdata; }

T\* end() { return pdata + size; }

bool empty() const { return size == 0; }

MyVector& operator=(MyVector& v);

char\* operator[](int i);

T& operator[](size\_t index);

template<typename T>

friend std::ostream& operator<<(std::ostream& out, const MyVector<T>& v);

};

size\_t CalculateNewSize(int size, int max\_size)

{

if (size == max\_size || size \* 4 == max\_size)

{

return std::max(size \* 2, MAX\_SIZE);

}

return max\_size;

}

template <typename T>

T& MyVector<T>::operator[](size\_t index)

{

return pdata[index];

}

template <>

void MyVector<char\*>::clear()

{

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

{

delete[] pdata[i];

pdata[i] = nullptr;

}

size = 0;

delete[] pdata;

pdata = nullptr;

}

template <typename T>

void MyVector<T>::clear()

{

size = 0;

delete[] pdata;

pdata = nullptr;

}

// resize

template <typename T>

void MyVector<T>::resize()

{

size\_t new\_size = CalculateNewSize(this->size, this->max\_size);

if (new\_size == this->max\_size)

{

return;

}

T\* temp = new T[new\_size];

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

{

temp[i] = this->pdata[i];

}

delete[] this->pdata;

this->pdata = temp;

this->max\_size = new\_size;

}

template <>

void MyVector<char\*>::resize()

{

size\_t new\_size = CalculateNewSize(this->size, this->max\_size);

if (new\_size == this->max\_size)

{

return;

}

char\*\* temp = new char\* [new\_size];

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

{

temp[i] = new char[strlen(this->pdata[i]) + 1];

strcpy(temp[i], this->pdata[i]);

delete[] this->pdata[i];

}

delete[] this->pdata;

this->pdata = temp;

this->max\_size = new\_size;

}

// add el

template<typename T>

void MyVector<T>::add\_element(T el)

{

resize();

this->pdata[this->size] = el;

this->size++;

}

template<>

void MyVector<char\*>::add\_element(char\* el)

{

resize();

this->pdata[this->size] = new char[strlen(el) + 1];

strcpy(this->pdata[this->size], el);

this->size++;

}

// Конструктор

template <typename T>

MyVector<T>::MyVector(T el, int max\_size)

{

this->size = 0;

this->max\_size = max\_size;

this->pdata = new T[max\_size];

}

template <>

MyVector<char\*>::MyVector(char\* el, int max\_size)

{

this->size = 0;

this->max\_size = max\_size;

this->pdata = new char\* [max\_size];

if (el)

{

add\_element(el);

}

}

// Деструктор

template <typename T>

MyVector<T>::~MyVector()

{

delete[] this->pdata;

}

template <>

MyVector<char\*>::~MyVector()

{

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

{

delete[] pdata[i];

}

delete[] pdata;

}

// К Копирования

template <typename T>

MyVector<T>::MyVector(MyVector& vecToCopy)

{

this->max\_size = vecToCopy.max\_size;

this->size = vecToCopy.size;

this->pdata = new T[this->max\_size];

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

{

this->pdata[i] = vecToCopy.pdata[i];

}

}

template <>

MyVector<char\*>::MyVector(MyVector& vecToCopy)

{

this->max\_size = vecToCopy.max\_size;

this->size = vecToCopy.size;

this->pdata = new char\* [this->max\_size];

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

{

this->pdata[i] = new char[strlen(vecToCopy.pdata[i]) + 1];

strcpy(this->pdata[i], vecToCopy.pdata[i]);

}

}

// удаление

template<typename T>

bool MyVector<T>::delete\_element(int index)

{

if (index >= this->size || index < 0) {

return false;

}

for (size\_t i = index; i < this->size - 1; i++) {

this->pdata[i] = this->pdata[i + 1];

}

this->size--;

resize();

return true;

}

template<>

bool MyVector<char\*>::delete\_element(int index)

{

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

{

return false;

}

delete[] this->pdata[index];

for (size\_t i = index; i < this->size - 1; i++)

{

this->pdata[i] = this->pdata[i + 1];

}

this->size--;

resize();

return true;

}

// поиск

template<typename T>

size\_t MyVector<T>::find(T el)

{

for (size\_t i = 0; i < this->size; i++) {

if (el == pdata[i]) {

return i;

}

}

return -1;

}

template<>

size\_t MyVector<char\*>::find(char\* el)

{

for (size\_t i = 0; i < this->size; i++) {

if (el == pdata[i]) {

return i;

}

}

return -1;

}

// сортировка

template <typename T>

void MyVector<T>::sort()

{

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

for (size\_t j = 0; j < size - 1; j++) {

if (pdata[j] > pdata[j + 1]) {

std::swap(pdata[j], pdata[j + 1]);

}

}

}

}

template <>

void MyVector<char\*>::sort()

{

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

{

for (size\_t j = 0; j < size - 1; j++)

{

if (strcmp(pdata[j], pdata[j + 1])) //str

{

std::swap(pdata[j], pdata[j + 1]);

}

}

}

}

/\* Перегрузки операторов \*/

template<typename T>

MyVector<T>& MyVector<T>::operator=(MyVector& v)

{

if (this != &v) {

this->size = v.size;

this->max\_size = v.max\_size;

this->pdata = new T[this->max\_size];

for (size\_t i = 0; i < this->size; i++) {

this->pdata[i] = v.pdata[i];

}

}

return \*this;

}

template<typename T>

char\* MyVector<T>::operator[](int i)

{

return this->pdata[i];

}

template<typename T>

std::ostream& operator<<(std::ostream& out, const MyVector<T>& v)

{

for (size\_t i = 0; i < v.size; ++i)

{

out << v.pdata[i] << " ";

}

return out;

}

/\* Перегрузки операторов \*/

template<>

MyVector<char\*>& MyVector<char\*>::operator=(MyVector& v)

{

if (this != &v)

{

this->size = v.size;

this->max\_size = v.max\_size;

this->pdata = new char\* [this->max\_size];

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

{

this->pdata[i] = new char[strlen(v.pdata[i]) + 1]; // +1 for null terminator

strcpy(this->pdata[i], v.pdata[i]);

}

}

return \*this;

}

template<>

char\* MyVector<char\*>::operator[](int i)

{

return this->pdata[i];

}

template<>

std::ostream& operator<<(std::ostream& out, const MyVector<char\*>& v)

{

for (size\_t i = 0; i < v.size; ++i)

{

out << v.pdata[i] << " ";

}

return out;

}

MyVector.cpp

#include "MyVector.h"

Polinom.h

#pragma once

#include "term.h"

class Polynomial

{

private:

MyVector<Term> poly;

int degree;

bool order\_; // true - по возрастанию, false - по убыванию

public:

// Constructors

Polynomial(bool ascending\_order = true)

: order\_(ascending\_order)

{

poly.add\_element(Term(0, 0));

degree = 0;

}

Polynomial(MyVector<Term>& terms, bool ascending\_order = true)

: poly(terms), order\_(ascending\_order), degree(getDegree())

{

Sort();

}

Polynomial(int n, bool ascending\_order = true)

: order\_(ascending\_order)

{

poly.add\_element(Term(n, 0));

degree = 0;

}

Polynomial(Term term, bool ascending\_order = true)

: order\_(ascending\_order)

{

poly.add\_element(term);

degree = std::max(degree, term.GetY());

}

Polynomial(Polynomial& p)

: poly(p.poly), degree(p.degree), order\_(p.order\_)

{}

// Methods

void SetTerm(const Term& t, bool subtraction = false)

{

bool found = false;

for (auto& term : poly)

{

if (term.GetY() == t.GetY())

{

term.SetX(subtraction ? term.GetX() - t.GetX() : term.GetX() + t.GetX());

found = true;

break;

}

}

if (!found && !subtraction) poly.add\_element(t);

Sort();

}

bool HaveTerm(Term t)

{

for (auto& term : poly)

{

if (term.GetY() == t.GetY())

{

return true;

}

}

return false;

}

int getDegree()

{

int deg = 0;

for (auto& term : poly)

{

deg = std::max(deg, term.GetY());

}

return deg;

}

void Sort()

{

MyVector<Term> p;

degree = getDegree();

for (int i = (order\_ ? 0 : degree);

(order\_ ? i <= degree : i >= 0);

(order\_ ? ++i : --i))

{

int x = 0;

int y = i;

for (auto& term : poly)

{

if (term.GetY() == i)

{

x += term.GetX();

}

}

if (x != 0)

{

p.add\_element(Term(x, y));

}

}

this->poly = p;

degree = getDegree();

}

Polynomial& operator= (Polynomial& p)

{

if (this == &p)

return \*this;

poly = p.poly;

degree = p.degree;

order\_ = p.order\_;

return \*this;

}

void operator+= (Polynomial p)

{

for (auto& term : p.poly)

{

this->poly.add\_element(term);

}

Sort();

}

void operator-= (Polynomial p)

{

for (auto& term : p.poly)

{

term.SetX((-1) \* term.GetX());

this->poly.add\_element(term);

}

Sort();

}

friend Polynomial operator+ (Polynomial p1, Polynomial p2)

{

Polynomial p(p1.order\_);

for (auto& term : p1.poly)

{

p.poly.add\_element(term);

}

for (auto& term : p2.poly)

{

p.poly.add\_element(term);

}

p.Sort();

return p;

}

friend Polynomial operator- (Polynomial p1, Polynomial p2)

{

Polynomial p(p1.order\_);

for (auto& term : p1.poly)

{

p.poly.add\_element(term);

}

for (auto& term : p2.poly)

{

term.SetX((-1) \* term.GetX());

p.poly.add\_element(term);

}

p.Sort();

return p;

}

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

{

Polynomial p(p1.order\_);

for (auto& term1 : p1.poly)

{

for (auto& term2 : p2.poly)

{

p.poly.add\_element(term1 \* term2);

}

}

p.Sort();

return p;

}

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

{

char str[256];

in.getline(str, 256);

MyVector<Term> terms;

Term::parseTerms(str, terms);

Polynomial np(terms, p.order\_);

p = np;

return in;

}

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

{

if (p.poly.empty())

{

out << 0 << std::endl;

return;

}

bool isFirst = true;

for (size\_t i = p.poly.get\_size(); i--;)

{

int X = p.poly[i].GetX();

int Y = p.poly[i].GetY();

if (p.poly[i].IsZero()) continue;

if (!isFirst)

{

char symbol = X > 0 ? '+' : '-';

out << " " << symbol << " ";

}

else if (X < 0)

{

out << "-";

}

isFirst = false;

if (std::abs(X) != 1 || Y == 0)

{

out << std::abs(X);

}

if (Y == 1)

{

out << "x";

}

else if (Y != 0)

{

out << "x^" << Y;

}

}

out << std::endl;

}

};

term.h

#include <iostream>

#include "MyVector.h"

using namespace std;

class Term

{

private:

int x;

int y;

friend class Polynomial;

public:

static void parseTerms(char\* str, MyVector<Term>& terms)

{

char\* ptr = str; // Указатель на текущую позицию в строке

int sign = 1; // Знак следующего терма

int coefficient, exponent;

while (\*ptr != '\0') {

coefficient = 0;

exponent = 0;

bool isCoef = false;

while (\*ptr == ' ') ++ptr;

if (\*ptr == '-') {

sign = -1;

++ptr;

}

else if (\*ptr == '+') {

sign = 1;

++ptr;

}

else {

sign = 1;

}

while (\*ptr == ' ') ++ptr;

while (isdigit(\*ptr)) {

coefficient = coefficient \* 10 + (\*ptr - '0');

++ptr;

isCoef = true;

}

if (!isCoef && (\*ptr != 'x' && \*ptr != 'X')) {

coefficient = 1;

}

coefficient \*= sign;

while (\*ptr == ' ') ++ptr;

if (\*ptr == 'x' || \*ptr == 'X') {

if (!isCoef) coefficient = sign; // Если коэффициент не указан явно, то он равен знаку (+1 или -1)

++ptr;

exponent = 1;

while (\*ptr == ' ') ++ptr;

if (\*ptr == '^') {

++ptr;

exponent = 0;

while (\*ptr == ' ') ++ptr;

while (isdigit(\*ptr)) {

exponent = exponent \* 10 + (\*ptr - '0');

++ptr;

}

}

}

else {

exponent = 0;

}

terms.add\_element(Term(coefficient, exponent));

while (\*ptr == ' ') ++ptr;

}

}

Term()

{

x = 0;

y = 0;

}

Term(int x, int y = 1)

{

this->x = x;

this->y = y;

}

Term(const Term& t)

{

this->x = t.x;

this->y = t.y;

}

Term operator= (Term t)

{

this->x = t.x;

this->y = t.y;

return \*this;

}

friend Term operator+ (Term t1, Term t2)

{

if (t1.GetY() != t2.GetY())

{

cout << "Ошибка" << endl;

exit(1);

}

return Term(t1.x + t2.x, t1.y);

}

friend Term operator- (Term t1, Term t2)

{

if (t1.GetY() != t2.GetY())

{

cout << "Ошибка" << endl;

exit(0);

}

return Term(t1.x - t2.x, t1.y);

}

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

{

return Term(t1.x \* t2.x, t1.y + t2.y);

}

friend Term operator\* (double x, Term t)

{

return Term(x \* t.x, t.y);

}

friend Term operator\* (Term t, double x)

{

return Term(x \* t.x, t.y);

}

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

{

if (t.x == 0)

out << 0;

else if (t.y == 0)

out << t.x;

else if (t.y == 1)

out << t.x << "x";

else

out << t.x << "x^" << t.y;

return out;

}

friend istream& operator>>(istream& in, Term& t) {

char input[100]; // Массив для хранения входной строки

in >> input; // Считываем строку из потока

// Указатель на текущую позицию в строке

const char\* ptr = input;

int sign = 1; // Знак следующего терма

int coefficient = 0, exponent = 0;

// Пропуск пробелов до начала терма

while (\*ptr == ' ') ++ptr;

// Проверка знака терма

if (\*ptr == '-') {

sign = -1;

++ptr;

}

else if (\*ptr == '+') {

sign = 1;

++ptr;

}

else {

sign = 1;

}

// Пропуск пробелов после знака

while (\*ptr == ' ') ++ptr;

// Чтение коэффициента

bool isCoef = false;

while (\*ptr >= '0' && \*ptr <= '9') {

coefficient = coefficient \* 10 + (\*ptr - '0');

++ptr;

isCoef = true;

}

// Случай, если коэффициент не указан

if (!isCoef && (\*ptr != 'x' && \*ptr != 'X'))

coefficient = 1;

coefficient \*= sign;

// Пропуск пробелов после коэффициента (если есть)

while (\*ptr == ' ') ++ptr;

if (\*ptr == 'x' || \*ptr == 'X') {

if (!isCoef) coefficient = sign; // Если коэффициент не указан явно, то он равен знаку (+1 или -1)

++ptr;

exponent = 1;

// Пропуск пробелов после 'x'

while (\*ptr == ' ') ++ptr;

// Чтение экспоненты, если она указана

if (\*ptr == '^') {

++ptr;

exponent = 0; // Сброс экспоненты перед чтением

while (\*ptr == ' ') ++ptr; // Пропуск пробелов после '^'

while (\*ptr >= '0' && \*ptr <= '9') {

exponent = exponent \* 10 + (\*ptr - '0');

++ptr;

}

}

}

else {

if (isCoef || coefficient == 0) exponent = 0;

}

t.x = coefficient;

t.y = exponent;

return in;

}

// Работа с приватными членами класса

int GetX() const { return this->x; }

int GetY() const { return this->y; }

void SetX(int value) { this->x = value; }

void SetY(int value) { this->y = value; }

bool IsZero() const { return this->x == 0; }

};

**Тестирование**

|  |
| --- |
|  |
|  |
|  |
|  |