**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 №5

«Реализация класса для оперирования полиномами»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-22 |  | преподаватель каф. ИУ5 |
| Барышников Михаил |  | Козлов А.Д. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2020 г.

**Задача**

Разработать 2 класса Term и Polinom для ввода, хранения полиномов и проведения простых математических операций над ними (+, \*).

Класс Term содержит члены-данные коэффициента и показателя степени, три конструктора:

* Без параметров (0x0)
* С одним параметром - коэффициентом (3x0)
* С двумя параметрами, соответственно коэффициентом и показателем степени (3x2)

Также перегруженный operator+ для сложения термов с одинаковыми показателями степени и дружественная функция вывода ostream<<.

Классы Term и Polinom – дружественные. Второй класс по заданию должен был содержать 2 члена данных, первый из которых массив из 6 элементов, а второй degree (степень), от обоих я решил отказаться в пользу возможности проводить операции над большим объемом данных, т.е. над «большими» полиномами и в вместо статического массива я использовал класс написанный ранее my\_vector, реализованный без использования сторонних библиотек. Этот класс также содержит 3 конструктора:

* Без параметров, для представления полинома 0
* С одним целы параметром, для представления полинома с соответствующим количеством термов
* С одним параметром термом

Скрытый член order, было решено сделать статическим. Он используется для определения способа хранения термов в массиве полиномов по возрастанию или убыванию. Также друзья класса функции operator+, operator\*, каждая из которых получает на вход 2 полинома и выдает результат. А также +=, \*=, = и конструктор копий.

Друзья класса операторы ввода и вывода istream>> и ostream<< для полиномов.

Полином в вида **5x5 - 4x3 + 1x2 + 0x1 - 7x0** представлен для печати как **5x^5 - 4x^3 + x^2 - 7.**

**Текст программы**

* Term\_class.h

#pragma once

#include <iostream>

#include "Polinom\_class.h"

class Polinom;

class Term

{

int power;

int koef;

public:

Term();

Term(int);

Term(int, int);

Term operator+(const Term&); //

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

friend class Polinom;

friend std::ostream& operator<<(std::ostream&, const Polinom&);

friend Polinom operator\*(const Polinom&, const Polinom&);

};

* Polinom\_class.h

#pragma once

#include <string>

#include "Term\_class.h"

#include "../../../../../MY\_PROJECTS/VisualStudio/My\_Vector/My\_Vector/my\_vector.h"

class Term;

class Polinom

{

my\_vector<Term> poly;

static bool order;

public:

Polinom();

Polinom(int);

Polinom(const Term&);

Polinom(const Polinom&); //

friend Polinom operator+(const Polinom&, const Polinom&); // friend

friend Polinom operator\*(const Polinom&, const Polinom&); // friend

Polinom operator=(const Polinom&); //

Polinom operator+=(const Polinom&); //

Polinom operator\*=(const Polinom&); //

friend std::istream& operator>>(std::istream&, Polinom&);

friend std::ostream& operator<<(std::ostream&, const Polinom&);

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

static void set\_order(bool); // если передать значение true, то по возрастанию, иначе по убыванию (поумолчанию true)

};

* Term\_functions.cpp

#include "Term\_class.h"

Term::Term()

{

power = 0;

koef = 0;

}

Term::Term(int k)

{

power = 0;

koef = k;

}

Term::Term(int k, int p)

{

power = p;

koef = k;

}

Term Term::operator+(const Term &t)

{

if (t.power != power)

throw std::exception("The powers of Terms is not equal!");

return Term(t.koef + koef, power);

}

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

{

stream << (t.koef < 0 ? "- " : "");

if (t.koef != -1 && t.koef != 1 || t.power == 0)

stream << (t.koef < 0 ? -t.koef : t.koef);

if (t.koef != 0)

stream << (t.power > 0 ? "x" : "") << (t.power > 1 && t.koef ? "^" + std::to\_string(t.power) : "");

return stream;

}

* Polinom\_functions.cpp

#include "Polinom\_class.h"

bool Polinom::order = true;

Polinom::Polinom()

{}

Polinom::Polinom(int k)

{

for (int i = 0; i != k; ++i)

poly.push\_back(Term());

}

Polinom::Polinom(const Term& tm)

{

poly.push\_back(tm);

}

Polinom::Polinom(const Polinom &p)

{

poly = p.poly;

for (int i = 0; i != poly.size(); ++i)

for (int j = i + 1; j != poly.size(); ++j)

if (poly[i].power == poly[j].power)

{

poly[i] = poly[i] + poly[j];

poly.erase(j, j);

--j;

}

for (int i = 0; i != poly.size(); ++i)

for (int j = 0; j != poly.size() - i - 1; ++j)

if (order ? (poly[j].power > poly[j + 1].power) : (poly[j].power < poly[j + 1].power))

{

Term a = poly[j];

poly[j] = poly[j + 1];

poly[j + 1] = a;

}

}

Polinom Polinom::operator=(const Polinom &p)

{

poly = p.poly;

for (int i = 0; i != poly.size(); ++i)

for (int j = i + 1; j != poly.size(); ++j)

if (poly[i].power == poly[j].power)

{

poly[i] = poly[i] + poly[j];

poly.erase(j, j);

--j;

}

/\*for (int i = 0; i != poly.size(); ++i)

for (int j = 0; j != poly.size() - 1 - i; ++j)

if (poly[j].power > poly[j + 1].power)

{

Term a = poly[j];

poly[j] = poly[j + 1];

poly[j + 1] = a;

}\*/

return \*this;

}

Polinom Polinom::operator+=(const Polinom &p)

{

return \*this = \*this + p;

}

Polinom Polinom::operator\*=(const Polinom &p)

{

return \*this = \*this + p;

}

void Polinom::set\_order(bool q)

{

order = q;

}

Polinom operator\*(const Polinom& p1, const Polinom& p2)

{

Polinom a;

for (int i = 0; i != p1.poly.size(); ++i)

for (int j = 0; j != p2.poly.size(); ++j)

a.poly.push\_back(Term(p1.poly[i].koef \* p2.poly[j].koef, p1.poly[i].power + p2.poly[j].power));

return a;

}

Polinom operator+(const Polinom& p1, const Polinom& p2)

{

Polinom a;

for (int i = 0; i != p1.poly.size(); ++i)

a.poly.push\_back(p1.poly[i]);

for (int i = 0; i != p2.poly.size(); ++i)

a.poly.push\_back(p2.poly[i]);

return a;

}

std::istream& operator>>(std::istream& stream, Polinom& p)

{

int pos;

p.poly.erase();

std::string str;

std::getline(stream, str);

while ((pos = str.find\_first\_of(" ()")) != std::string::npos)

str.erase(pos, 1);

char zn = '+';

while (str != "")

{

int tmp;

if ((tmp = str.find\_first\_of("+-")) == std::string::npos)

tmp = str.length();

if (!tmp)

{

zn = str[0] == zn ? '+' : '-';

str.erase(0, 1);

}

else

{

int tmp\_sym;

p.poly.push\_back((tmp\_sym = str.find\_first\_of("x")) < tmp && tmp\_sym != std::string::npos ? (str.find\_first\_of("^") < tmp && str.find\_first\_of("^") != std::string::npos ? Term((zn == '-' ? -1 : 1) \* (tmp\_sym == 0 ? 1 : atoi(str.substr(0, tmp\_sym).c\_str())), atoi(str.substr(tmp\_sym + 2, tmp - tmp\_sym - 2).c\_str())) : Term((zn == '-' ? -1 : 1) \* (tmp\_sym == 0 ? 1 : atoi(str.substr(0, tmp\_sym).c\_str())), 1)) : Term((zn == '-' ? -1 : 1) \* atoi(str.substr(0, tmp).c\_str())));

zn = '+';

str.erase(0, tmp);

}

}

p = Polinom(p);

return stream;

}

std::ostream& operator<<(std::ostream &stream, const Polinom &p)

{

for (int i = 0; i != p.poly.size(); ++i)

{

if (p.poly[i].koef != 0)

stream << (p.poly[i].koef < 0 ? "- " : (i && p.poly[i].koef >= 0 ? "+ " : "")) << Term((p.poly[i].koef < 0 ? -1 : 1) \* p.poly[i].koef, p.poly[i].power) << ' ';

}

return stream;

}

* my\_vector.h

#pragma once

#include <cstdlib>

const int size\_l = 50;

using namespace std;

template<class T>

class my\_vector

{

int length;

int length\_of\_used;

T \*ptr\_;

int add\_memory(T);

public:

my\_vector<T>();

my\_vector<T>(const my\_vector<T> &);

~my\_vector<T>();

void push\_back(T);

void pop\_back();

void erase();

my\_vector<T> operator=(const my\_vector<T>&);

void erase(int);

void erase(int, int);

//void set\_size(int);

int size() const;

T& operator[](int) const;

};

template<class T>

inline int my\_vector<T>::add\_memory(T b)

{

length += size\_l / 2;

T \*p = new T[length]; //

//T \*p = (T\*)malloc(sizeof(T) \* length);

if (p == NULL)

{

length -= size\_l / 2;

throw std::exception("Memory is full!");

}

for (int i = 0; i != length\_of\_used; ++i)

\*(p + i) = \*(ptr\_ + i);

delete[] ptr\_; //

//free(ptr\_);

ptr\_ = p;

push\_back(b);

return 0;

}

template<class T>

inline void my\_vector<T>::push\_back(T b)

{

length > length\_of\_used ? \*(ptr\_ + length\_of\_used++) = b : add\_memory(b);

}

template<class T>

inline void my\_vector<T>::pop\_back()

{

if (length\_of\_used)

\*(ptr\_ + --length\_of\_used) = 0;

}

template<class T>

inline void my\_vector<T>::erase()

{

while (length\_of\_used)

\*(ptr\_ + --length\_of\_used) = 0;

}

template<class T>

inline my\_vector<T> my\_vector<T>::operator=(const my\_vector<T>& b)

{

length = b.length;

length\_of\_used = b.length\_of\_used;

ptr\_ = new T[length]; //

//ptr\_ = (T\*)malloc(sizeof(T) \* length);

//if (ptr\_ == NULL) throw std::exception("Memory is full!");

for (int i = 0; i != length\_of\_used; ++i)

\*(ptr\_ + i) = \*(b.ptr\_ + i);

return \*this;

}

template<class T>

inline void my\_vector<T>::erase(int b)

{

if (b < 0 || b >= length\_of\_used)

throw std::exception("Error! This area of memory is not available!");

for (int i = b; i != length\_of\_used - 1; ++i)

\*(ptr\_ + i) = \*(ptr\_ + i + 1);

pop\_back();

}

template<class T>

inline void my\_vector<T>::erase(int b, int c)

{

if (b < 0 || b >= length\_of\_used || c >= length\_of\_used)

throw std::exception("Error! This area of memory is not available!");

if (b > c)

throw std::exception("Error! Range is not exist!");

int delta = c - b + 1;

for (int i = b; i != length\_of\_used - delta; ++i)

\*(ptr\_ + i) = \*(ptr\_ + i + delta);

delta = length\_of\_used - delta;

for (int i = length\_of\_used; i != delta; --i)

pop\_back();

}

template<class T>

inline int my\_vector<T>::size() const

{

return length\_of\_used;

}

template<class T>

inline T & my\_vector<T>::operator[](int b) const

{

return length\_of\_used > b ? \*(ptr\_ + b) : throw std::exception("Error! This area of memory is not available!");

}

template<class T>

inline my\_vector<T>::my\_vector()

{

length = size\_l;

length\_of\_used = 0;

ptr\_ = new T[length]; //

//ptr\_ = (T\*)malloc(sizeof(T) \* length);

if (ptr\_ == NULL)

{

length = 0;

throw std::exception("Memory is full!");

}

}

template<class T>

inline my\_vector<T>::my\_vector(const my\_vector<T>& b)

{

length = b.length;

length\_of\_used = b.length\_of\_used;

ptr\_ = new T[length]; //

//ptr\_ = (T\*)malloc(sizeof(T) \* length);

//if (ptr\_ == NULL) throw std::exception("Memory is full!");

for (int i = 0; i != length\_of\_used; ++i)

\*(ptr\_ + i) = \*(b.ptr\_ + i);

}

template<class T>

inline my\_vector<T>::~my\_vector()

{

delete[] ptr\_; //

//free(ptr\_);

}

* main.cpp

#include <iostream>

#include "Polinom\_class.h"

int main()

{

Polinom a, b;

char sym = 1;

bool order = true;

while (sym)

{

std::cin.ignore();

std::cout << "Enter first polinom:\n";

std::cin >> a;

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

std::cout << "Enter second polinom:\n";

std::cin >> b;

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

std::cout << "Enter operator (enter 0 to exit or enter \"d\" to change sorting):\n";

//std::cin.ignore();

std::cin >> sym;

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

switch (sym)

{

case '+':

std::cout << a + b << "\n";

break;

case '\*':

std::cout << a \* b << "\n";

break;

case 'd':

Polinom::set\_order(order = !order);

break;

}

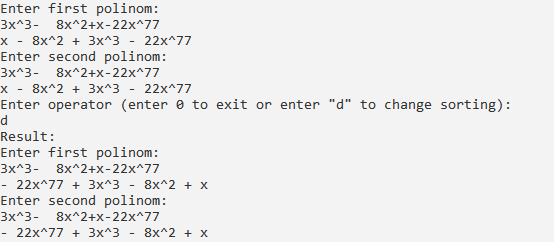
}

}

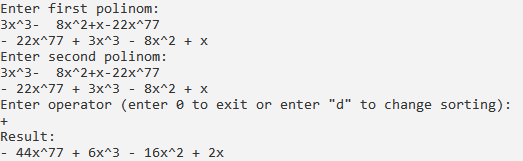
**Анализ результатов**

В данной программе в качестве «точки входа» выступает основная функция main(). В ней написан тестируемый код. Ниже представлены результаты выполнения. Для реализации методов данного класса был «с нуля» написан вспомогательный класс вектор (my\_vector), который можно использовать и отдельно от класса «Polinom».

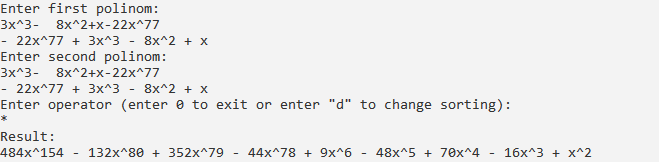
Тест №1: (изменения сортировки применяются со следующего вывода)



Тест №2:



Тест №3:



Тесты были проверены вручную. Программа выдала верные результаты. Также были проведены многочисленные тесты всех операторов, многие из которых не были прикреплены. Но основываясь на правильности приведенных тестов, можно сделать вывод, что остальные то же отработают правильно.

Примечание:

Символ «d» используется для тестирования сортировки термов в полиноме.

**Вывод**

В результате написания программы был получен опыт работы с классами (ООП). Были реализованы перегрузки основных операторов простых арифметических действий для термов и полиномов. В качестве доработки можно было бы переопределить остальные операторы. Еще из минусов – это возможность переполнения int значение элемента, но и это решаемо, т.к. можно подключить библиотеку для работы с длинной арифметикой на c++. В итоге был написан классы Polinom и Term, основанные на классе my\_vector, которые удобно использовать для работы с полиномами в том числе и для проведения операций над ними.