|  |  |
| --- | --- |
| C:\Documents and Settings\admin\Рабочий стол\МИРЭА_ЭМБЛЕМА_приказ.JPG | |
| МИНОБРАНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  «**МОСКОВСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»**  **МИРЭА** | |
| Институт информационных технологий  Кафедра математического обеспечения и стандартизации информационных технологий | |
| **Отчет по лабораторной работе №7**  **Тема работы «Создание класса. Перегрузка операций.»** | |
| Выполнил: |  |
| Студент группы ИКБО-07-17  Отчет принял:  Преподаватель кафедры МОСИТ | Акжигитов Р. Р.  Миронов Антон Николаевич |
| Москва  2018 | |

Оглавление

[Задание 1 3](#_Toc515001260)

[Условия задания 3](#_Toc515001261)

[Декомпозиция 3](#_Toc515001262)

[Определения классов 3](#_Toc515001263)

[Перегрузка 4](#_Toc515001264)

[Тестирование 5](#_Toc515001265)

[Кодирование алгоритма программы 5](#_Toc515001266)

[Вывод 8](#_Toc515001267)

# Задание 1

# Условия задания

Линейный многочлен N – ой степени (узел содержит коэффициент, степень). Определить степень многочлена. Вычислить значение многочлена при заданном х.

+ Сложить два многочлена

= Присвоить

# Декомпозиция

1. Написать класс узел, который будет содержать коэффициент при слагаемом и степень неизвестного.
2. Реализовать геттеры и сеттеры получения степени и коэффициента, так как они понадобятся при сложении многочленов.
3. Реализовать класс многочлен, который будет хранить массив (вектор) узлов
4. Реализовать функции добавления узла: поиск по узлам многочлена и проверка по одинаковым степеням, если степени одинаковы, то коэффициенты необходимо сложить (при сложении)
5. Перегрузить сложение, как цикл по узлам одного из многочленов. Создаем пустой многочлен. Прибавляем один многочлен, далее прибавляем второй, да так, чтобы не было наложений, а все одинаковые по степеням узлы складывались по коэффициентам.
6. Написать конструкторы и деструкторы, конструкторы копирования.

# Определения классов

class Node {

public:

Node() { k = 1; power = 0; };

Node(double k, int power) { this->k = k; this->power = power; }

int getPower() { return power; }

double getK() { return k; }

void setK(double k) { this->k = k; }

private:

double k;

int power;

};

class Poly {

public:

Poly() {};

Poly(std::vector<Node> data) { this->data = data; }

Poly(Poly& p) {

std::cout << "\tcopy constructor" << std::endl;

for (auto i : p.data)

this->data.push\_back(i);

}

Poly(const Poly& p) {

std::cout << "\tconst copy constructor" << std::endl;

for (auto i : p.data)

this->data.push\_back(i);

}

Poly(Node node) { this->data.push\_back(node); }

~Poly() { std::vector<Node> name; data.swap(name); std::cout << data.size() << " " << "destruct\n"; }

std::vector<Node> getData() { return data; }

void Add(Node node) {

bool flag = false;

for (auto& i : this->data) {

if (node.getPower() == i.getPower()) {

flag = true;

i.setK(i.getK() + node.getK());

}

}

if (!flag)

this->data.push\_back(node);

}

int Power() {

std::vector<int> powers;

for (auto i : this->data) powers.push\_back(i.getPower());

if (powers.size() > 0) {

int max = powers[0];

for (int i = 1; i < powers.size(); i++)

max = max < powers[i] ? powers[i] : max;

return max;

}

return 0;

}

double calculate(double x) {

double total = 0;

for (auto i : this->data)

total += i.getK() \* pow(x, i.getPower());

return total;

}

Poly operator+(Poly x) {

Poly p;

for (auto& i : this->data)

p.Add(i);

for (auto& i : x.getData())

p.Add(i);

return p;

}

Poly& operator=(Poly& x) {

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

this->data = x.data;

return \*this;

}

friend std::ostream& operator<<(std::ostream& os, Poly& p);

private:

std::vector<Node> data;

};

# Перегрузка

Poly operator+(Poly x) {

Poly p;

for (auto& i : this->data)

p.Add(i);

for (auto& i : x.getData())

p.Add(i);

return p;

}

Poly& operator=(Poly& x) {

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

this->data = x.data;

return \*this;

}

friend std::ostream& operator<<(std::ostream& os, Poly& p);

std::ostream& operator<<(std::ostream& os, Poly& p) {

for (int i = 0; i < p.data.size(); i++)

if (i != p.data.size() - 1)

if (p.data[i].getPower() == 0)

os << p.data[i].getK() << " + ";

else

os << p.data[i].getK() << "x^" << p.data[i].getPower() << " + ";

else

if (p.data[i].getPower() == 0)

os << p.data[i].getK() << std::endl;

else

os << p.data[i].getK() << "x^" << p.data[i].getPower() << std::endl;

//os << p.data[i].getK() << (p.data[i].getPower() == 0 ? ("x^" << p.data[i].getPower()) : "") << (i == p.data.size() - 1 ? "\n" : " + ");

return os;

}

# Тестирование

int main() {

Poly p(std::vector<Node>{Node(-1, 5), Node(2, 3), Node(7, 2), Node(17, 0)});

Poly n(std::vector<Node>{Node(5, 3), Node(-3, 2), Node(3, 0)});

std::cout << "f(x) = " << p;

std::cout << "f(2) = " << p.calculate(2) << "\n\n";

std::cout << "g(x) = " << n;

std::cout << "g(3) = " << n.calculate(3) << "\n\n";

Poly s = p + n;

std::cout << "f(x) + g(x) = " << s;

std::cout << "f(2) + g(2) = " << s.calculate(2) << "\n\n";

s = p;

std::cout << "s = f(x) = " << s;

Poly forCopy(std::vector<Node>{Node(2, 2), Node(1, 1)});

std::cout << "obj" << std::endl;

Poly obj(forCopy);

std::cout << "obj1" << std::endl;

Poly obj1 = obj;

std::cout << "obj2" << std::endl;

Poly obj2 = Poly(Poly(obj1));

system("pause");

return 0;

}

# Кодирование алгоритма программы

#include <iostream>

#include <vector>

#include <algorithm>

#include "math.h"

class Node {

public:

Node() { k = 1; power = 0; };

Node(double k, int power) { this->k = k; this->power = power; }

int getPower() { return power; }

double getK() { return k; }

void setK(double k) { this->k = k; }

private:

double k;

int power;

};

class Poly {

public:

Poly() {};

Poly(std::vector<Node> data) { this->data = data; }

Poly(Poly& p) {

std::cout << "\tcopy constructor" << std::endl;

for (auto i : p.data)

this->data.push\_back(i);

}

Poly(const Poly& p) {

std::cout << "\tconst copy constructor" << std::endl;

for (auto i : p.data)

this->data.push\_back(i);

}

Poly(Node node) { this->data.push\_back(node); }

~Poly() { std::vector<Node> name; data.swap(name); std::cout << data.size() << " " << "destruct\n"; }

std::vector<Node> getData() { return data; }

void Add(Node node) {

bool flag = false;

for (auto& i : this->data) {

if (node.getPower() == i.getPower()) {

flag = true;

i.setK(i.getK() + node.getK());

}

}

if (!flag)

this->data.push\_back(node);

}

//void Substract(Node node) { data.erase(std::remove(data.begin(), data.end(), node), data.end()); }

int Power() {

std::vector<int> powers;

for (auto i : this->data) powers.push\_back(i.getPower());

if (powers.size() > 0) {

int max = powers[0];

for (int i = 1; i < powers.size(); i++)

max = max < powers[i] ? powers[i] : max;

return max;

}

return 0;

}

double calculate(double x) {

double total = 0;

for (auto i : this->data)

total += i.getK() \* pow(x, i.getPower());

return total;

}

Poly operator+(Poly x) {

Poly p;

for (auto& i : this->data)

p.Add(i);

for (auto& i : x.getData())

p.Add(i);

return p;

}

Poly& operator=(Poly& x) {

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

this->data = x.data;

return \*this;

}

friend std::ostream& operator<<(std::ostream& os, Poly& p);

private:

std::vector<Node> data;

};

std::ostream& operator<<(std::ostream& os, Poly& p) {

for (int i = 0; i < p.data.size(); i++)

if (i != p.data.size() - 1)

if (p.data[i].getPower() == 0)

os << p.data[i].getK() << " + ";

else

os << p.data[i].getK() << "x^" << p.data[i].getPower() << " + ";

else

if (p.data[i].getPower() == 0)

os << p.data[i].getK() << std::endl;

else

os << p.data[i].getK() << "x^" << p.data[i].getPower() << std::endl;

//os << p.data[i].getK() << (p.data[i].getPower() == 0 ? ("x^" << p.data[i].getPower()) : "") << (i == p.data.size() - 1 ? "\n" : " + ");

return os;

}

int main() {

Poly p(std::vector<Node>{Node(-1, 5), Node(2, 3), Node(7, 2), Node(17, 0)});

Poly n(std::vector<Node>{Node(5, 3), Node(-3, 2), Node(3, 0)});

std::cout << "f(x) = " << p;

std::cout << "f(2) = " << p.calculate(2) << "\n\n";

std::cout << "g(x) = " << n;

std::cout << "g(3) = " << n.calculate(3) << "\n\n";

Poly s = p + n;

std::cout << "f(x) + g(x) = " << s;

std::cout << "f(2) + g(2) = " << s.calculate(2) << "\n\n";

s = p;

std::cout << "s = f(x) = " << s;

Poly forCopy(std::vector<Node>{Node(2, 2), Node(1, 1)});

std::cout << "obj" << std::endl;

Poly obj(forCopy);

std::cout << "obj1" << std::endl;

Poly obj1 = obj;

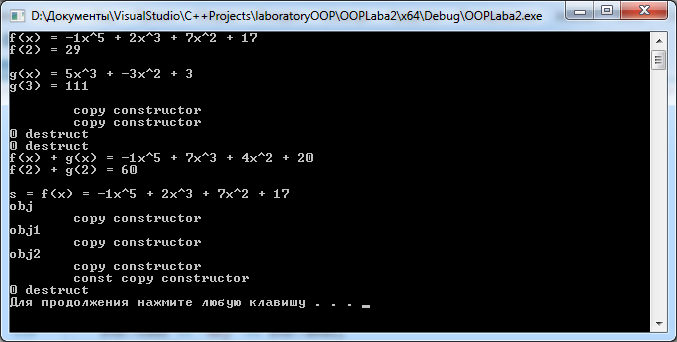
std::cout << "obj2" << std::endl;

Poly obj2 = Poly(Poly(obj1));

system("pause");

return 0;

}



# Вывод

В данной лабораторной работе я научился перегружать операции (унарные, бинарные, побитовые, логические), узнал про дружественные функции, прочитал про исключения перегрузок. Выполнив работу, я на практике научился ими пользоваться.