Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЕНИЯ

Кафедра интеллектуальных информационных технологий

**Отчёт**

**по лабораторной работе №1**

По дисциплине: Проектирование программного обеспечения в интеллектуальных системах

Выполнил: Пунинский И.Д., гр. 121702

Проверил: Никифоров С. А.

Минск, 2022

**Цель:** Описать класс многочлена от одной переменной, задаваемых степенью многочлена и массивом коэффициентов. Класс должен реализовывать следующие возможности:

* получение значений коэффициентов (оператор []);
* вычисление значения многочлена для заданного аргумента (оператор ());
* сложение двух многочленов (операторы +, +=);
* вычитание двух многочленов (операторы -, -=);
* произведение двух многочленов (операторы \*, \*=);
* деление двух многочленов (операторы /, /=);

**Описание структуры класса.**

Класс MC содержит переменную типа int содержащую степень полинома и «вектор» типа double – коэффициенты членов полинома.

Структура класса MС:

class MC

{

int deg = 0;

vector <double> coeff;

public:

MC();

MC(int n, double\* arr);

MC(vector <double> vect);

void out();

int outdeg();

const double operator() (const double num);

const double operator[] (const double num);

const MC operator+= (MC& rv);

const MC operator-= (MC& rv);

const MC operator\*= (MC& rv);

const MC operator/= (MC& rv);

const MC operator=(const MC& rv);

MC operator +(const MC& c1);

MC operator -(const MC& c1);

MC operator \*(const MC& rv);

MC operator /(const MC& c1);

};

У класса есть 3 конструктора. Первый (по умолчанию) создаёт «пустой» объект класса, второй создаёт объект класса из массива коэффициентов и степени полинома, третий создаёт объект класса из вектора коэффициентов и автоматически заполняет степень полинома. Деструктор по умолчанию (сгенерирован автоматически, неявно), т.к. в классе не присутствует динамическое выделение памяти, а используются STL-контейнеры.

**Описание и реализация методов класса:**

/\*\*

\* @brief Return degee and coefficients into console

\*/

void MC::out()

{

cout << this->deg << " | ";

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

{

cout << this->coeff[i] << " ";

}

cout << endl;

}

/\*\*

\* @brief Return plynom's degree

\* @return degree

\*/

int MC::outdeg()

{

return (this->deg);

}

/\*\*

\* @brief calculate polynom whith num

\* @param[in] num Inputed x

\* overload of () operator

\* @return buff Output number

\*/

const double MC::operator() (const double num)

{

double buff = 0;

for (int i = 0; i < deg + 1; i++)

{

buff = buff + (coeff[i] \* pow(num, i));

}

return buff;

}

/\*\*

\* @brief return polynom's coefficient by using it's number

\* @param[in] num Inputed number of coefficient

\* overload of [] operator

\* @return coeff[num] Output element of vector of coefficients

\*/

const double MC::operator[] (const double num)

{

if (num > this->deg) { return 0; }

return(coeff[num]);

}

/\*\*

\* @brief caculate sum of two polynom's

\* @param[in] rv Input of second polynom

\* overload of += operator

\* @return \*this Output this plolynom (sum of this polynom and another polynom)

\*/

const MC MC::operator+= (MC& rv)

{

if(this->deg >= rv.deg)

{

for (int i = 0; i < rv.deg + 1; i++)

{

this->coeff[i] = this->coeff[i] + rv.coeff[i];

}

}

else

{

for (int i = 0; i <= rv.deg; i++)

{

if( i <= this->deg )

{

this->coeff[i] = this->coeff[i] + rv.coeff[i];

}

else

{

this->coeff.push\_back(rv.coeff[i]);

}

}

this->deg = rv.deg;

}

return(\*this);

}

/\*\*

\* @brief caculate dif of two polynom's

\* @param[in] rv Input of second polynom

\* overload of -= operator

\* @return \*this Output this plolynom (dif of this polynom and another polynom)

\*/

const MC MC::operator-= (MC& rv)

{

if (this->deg >= rv.deg)

{

for (int i = 0; i < rv.deg + 1; i++)

{

this->coeff[i] = this->coeff[i] - rv.coeff[i];

}

}

else

{

for (int i = 0; i <= rv.deg; i++)

{

if (i <= this->deg)

{

this->coeff[i] = this->coeff[i] - rv.coeff[i];

}

else

{

this->coeff.push\_back(rv.coeff[i]\*(-1.0));

}

}

this->deg = rv.deg;

}

return(\*this);

}

/\*\*

\* @brief caculate prod of two polynom's

\* @param[in] rv Input of second polynom

\* overload of \*= operator

\* @return \*this Output this plolynom (prod of this polynom and another polynom)

\*/

const MC MC::operator\*= (MC& rv)

{

if (this->deg >= rv.deg)

{

vector <double> buff;

for (int i = 0; i < rv.deg + 1; i++)

{

for (int j = 0; j < this->deg + 1; j++)

{

buff.resize(this->deg + rv.deg + 1);

buff[j + i] = this->coeff[j] \* rv.coeff[i];

}

}

this->coeff = buff;

this->deg = buff.size() - 1;

}

else

{

vector <double> buff;

for (int i = 0; i < this->deg + 1; i++)

{

for (int j = 0; j < rv.deg + 1; j++)

{

buff.resize(this->deg + rv.deg + 1);

buff[j + i] = this->coeff[i] \* rv.coeff[j];

}

}

this->coeff = buff;

this->deg = buff.size() - 1;

}

return(\*this);

}

/\*\*

\* @brief caculate div of two polynom's

\* @param[in] rv Input of second polynom

\* overload of /= operator

\* @return \*this Output this plolynom (div of this polynom and another polynom)

\*/

const MC MC::operator/= (MC& rv)

{

if (this->deg >= rv.deg)

{

for (int i = 0; i < rv.deg + 1; i++)

{

this->coeff[i] = this->coeff[i] / rv.coeff[i];

}

}

else

{

for (int i = 0; i <= rv.deg; i++)

{

if (i <= this->deg)

{

this->coeff[i] = this->coeff[i] / rv.coeff[i];

}

}

}

return(\*this);

}

/\*\*

\* @brief assignment another polynom to this polynom

\* @param[in] rv Input of second polynom

\* overload of = operator

\* @return \*this Output this modefied plolynom

\*/

const MC MC::operator=( const MC& rv)

{

if (this == &rv) {

return \*this;

}

if (this->deg > rv.deg)

{

for (int i = 0; i <= this->deg; i++)

{

if (i <= rv.deg)

{

this->coeff[i] = rv.coeff[i];

}

else { this->coeff.pop\_back(); }

}

this->deg = rv.deg;

}

else

{

if (this->deg > 0)

{

for (int i = 0; i < rv.deg + 1; i++)

{

if (i < this->deg + 1)

{

this->coeff[i] = rv.coeff[i];

}

else

{

this->coeff.push\_back(rv.coeff[i]);

}

}

this->deg = rv.deg;

}

else

{

for (int i = 0; i < rv.deg + 1; i++)

{

this->coeff.push\_back(rv.coeff[i]);

}

this->deg = rv.deg;

}

}

return \*this;

}

/\*\*

\* @brief caculate sum of two polynom's

\* @param[in] c1 Input of second polynom

\* overload of + operator

\* @return tmp Output temporery plolynom (polynom's sum)

\*/

MC MC::operator +(const MC& c1)

{

MC tmp;

if (c1.deg >= this->deg)

{

tmp.deg = c1.deg;

for (int i = 0; i <= tmp.deg; i++)

{

if (i <= this->deg)

{

tmp.coeff.push\_back(c1.coeff[i] + this->coeff[i]);

}

else

{

tmp.coeff.push\_back(c1.coeff[i]);

}

}

}

else

{

tmp.deg = this->deg;

for (int i = 0; i <= tmp.deg; i++)

{

if (i <= c1.deg)

{

tmp.coeff.push\_back(c1.coeff[i] + this->coeff[i]);

}

else

{

tmp.coeff.push\_back(this->coeff[i]);

}

}

}

return tmp;

}

/\*\*

\* @brief caculate dif of two polynom's

\* @param[in] c1 Input of second polynom

\* overload of - operator

\* @return tmp Output temporery plolynom (polynom's dif)

\*/

MC MC::operator -(const MC& c1)

{

MC tmp;

if (c1.deg >= this->deg)

{

tmp.deg = c1.deg;

for (int i = 0; i <= tmp.deg; i++)

{

if (i <= this->deg)

{

tmp.coeff.push\_back(this->coeff[i] - c1.coeff[i]);

}

else

{

tmp.coeff.push\_back(c1.coeff[i] \* (-1.0));

}

}

}

else

{

tmp.deg = this->deg;

for (int i = 0; i <= tmp.deg; i++)

{

if (i <= c1.deg)

{

tmp.coeff.push\_back(this->coeff[i] - c1.coeff[i]);

}

else

{

tmp.coeff.push\_back(this->coeff[i]);

}

}

}

return tmp;

}

/\*\*

\* @brief caculate prod of two polynom's

\* @param[in] rv Input of second polynom

\* overload of \* operator

\* @return tmp Output temporery plolynom (polynom's prod)

\*/

MC MC::operator \*(const MC& rv)

{

MC buff;

if (this->deg >= rv.deg)

{

for (int i = 0; i < rv.deg + 1; i++)

{

for (int j = 0; j < this->deg + 1; j++)

{

buff.coeff.resize(this->deg + rv.deg + 1);

buff.coeff[j + i] = this->coeff[j] \* rv.coeff[i];

}

}

}

else

{

for (int i = 0; i < this->deg + 1; i++)

{

for (int j = 0; j < rv.deg + 1; j++)

{

buff.coeff.resize(this->deg + rv.deg + 1);

buff.coeff[j + i] = this->coeff[i] \* rv.coeff[j];

}

}

}

buff.deg = buff.coeff.size() - 1;

return(buff);

}

/\*\*

\* @brief caculate div of two polynom's

\* @param[in] c1 Input of second polynom

\* overload of / operator

\* @return tmp Output temporery plolynom (polynom's div)

\*/

MC MC::operator /(const MC& c1)

{

MC tmp;

if (c1.deg >= this->deg)

{

tmp.deg = c1.deg;

for (int i = 0; i <= tmp.deg; i++)

{

if (i <= this->deg)

{

tmp.coeff.push\_back(this->coeff[i] / c1.coeff[i]);

}

else

{

tmp.coeff.push\_back(1.0 / c1.coeff[i]);

}

}

}

else

{

tmp.deg = this->deg;

for (int i = 0; i <= tmp.deg; i++)

{

if (i <= c1.deg)

{

tmp.coeff.push\_back(this->coeff[i] / c1.coeff[i]);

}

else

{

tmp.coeff.push\_back(this->coeff[i]);

}

}

}

return tmp;

Краткое описание методов сгенерировано с помощью Doxygen и описано в комментариях MC.cpp.