Министерство образования и науки РФ

Нижегородский государственный университет имени Н.И. Лобачевского (Национальный исследовательский университет)

Институт Информационных Технологий Математики и Механики

Отчёт по лабораторной работе №1

Тема: «Работа с многочленами»

Выполнил: студент группы 0823-3

Нечесанов Владимир Александрович

Проверила: Шестакова Наталья Валерьевна.

Нижний Новгород

2016

# Оглавление

[Оглавление 1](#_Toc279967395)

[Введение 3](#_Toc279967396)

[Постановка задачи 3](#_Toc279967397)

[Руководство пользователя 3](#_Toc279967398)

[Руководство программиста 4](#_Toc279967399)

[Описание алгоритмов 4](#_Toc279967400)

[Структура данных 4](#_Toc279967401)

[Структура программного комплекса 10](#_Toc279967402)

[Заключение 11](#_Toc279967403)

[Список используемой литературы 12](#_Toc279967404)

[Приложение 13](#_Toc279967405)

# Введение

Многочлен (или полином) от *n* переменных — есть конечная формальная сумма вида

\sum_I c_I x_1^{i_1}x_2^{i_2}\cdots x_n^{i_n}

,

где *I* = *(i1, i2, ..., in)* есть набор из целых неотрицательных чисел (называется мультииндекс), *cI* — число (называемое «коэффициент многочлена»), зависящее только от мультииндекса *I*.

В частности, многочлен от одной переменной есть конечная формальная сумма вида

c_0 + c_1x^1 + \dots + c_mx^m, где *cI* - фиксированные коэффициенты, а x – переменная.

# Постановка задачи

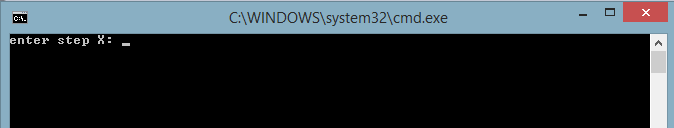
Требуется написать программу, которая позволяет работать с полиномами от трех переменных.

В качестве структуры хранения для полинома необходимо взять односвязный циклический список.

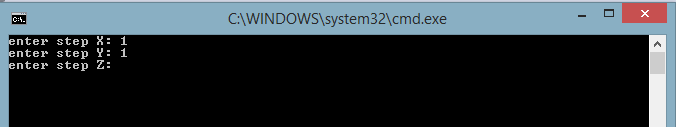
При добавлении монома, проверяется его равенство степеней с другими мономами, если они равны, то коэффициенты складываются.

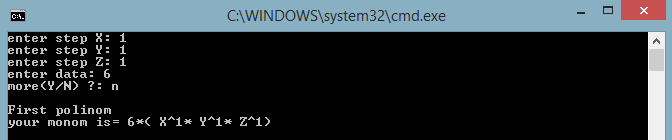
# Руководство пользователя

Для запуска программы откройте файл Polinom2.0



Пользователю предоставится возможность ввести три степени, для трех соответствующих переменных





# Руководство программиста

**Описание алгоритмов**

В качестве структуры хранения выбран односвязный список.

Принципиальным преимуществом перед массивом является структурная гибкость: порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера, а порядок обхода списка всегда явно задаётся его внутренними связями. У кольцевого списка указатель *next* указывает на *нулевой* (звено- заголовок) элемент списка.

**Структура данных**

**Структура данных** есть модель данных в виде математической структуры.

**Математическая структура:**

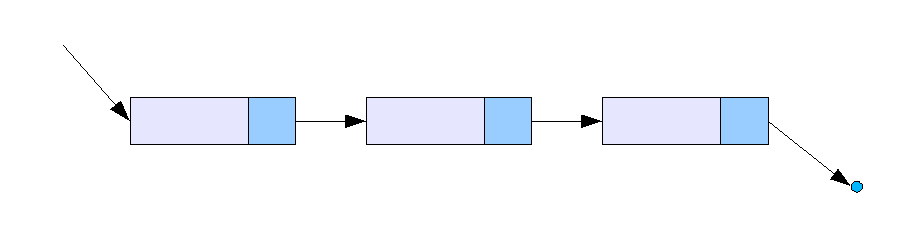
S=(M1,...,Mk;p1,...,ps)

Математическая структура есть одно или несколько множеств **M1,..,Mk**, элементы которых находятся в некоторых отношениях **p1,...,ps**.

Где M1,...,Mk - базисные множества структуры.

Каждое отношение p1 есть двоичная функция, аргументами которой являются элементы базисного множества.

**Связный список** — базовая динамическая структура данных в информатике, состоящая из узлов, каждый из которых содержит как собственно данные, так и одну или две ссылки («связки») на следующий и/или предыдущий узел списка. Принципиальным преимуществом перед массивом является структурная гибкость: порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера, а порядок обхода списка всегда явно задаётся его внутренними связями.



Для реализации алгоритмов было использовано 2 класса:

* Класс Monom;
* Класс Polinom;

Описание классов:

### Класс Monom:

#pragma once

#pragma once

class Monom

{

int Data;

Monom \*pNext;

int StepX, StepY, StepZ;

public:

int GetData();

Monom\* GetNext();

int GetStepX();

int GetStepY();

int GetStepZ();

Monom(Monom &);

void SetNext(Monom \*p);

Monom(int D, Monom \*P, int SX, int SY,int SZ);

~Monom();

friend class Polinom;

};

#include "Monom.h"

Monom::Monom(int D, Monom \*P, int SX,int SY,int SZ)

{

Data = D;

pNext = P;

StepX = SX;

StepY = SY;

StepZ = SZ;

}

Monom::Monom(Monom &D)

{

Data = D.Data;

StepX = D.StepX;

StepY = D.StepY;

StepZ = D.StepZ;

pNext = 0;

}

Monom\* Monom::GetNext()

{

return pNext;

}

int Monom::GetStepX()

{

return StepX;

}

int Monom::GetStepY()

{

return StepY;

}

int Monom::GetStepZ()

{

return StepZ;

}

int Monom::GetData()

{

return Data;

}

void Monom::SetNext(Monom \*p)

{

pNext = p;

}

Monom::~Monom()

{

}

1. Класс Polinom

#pragma once

#include"Monom.h"

class Polinom

{

Monom \*pFirst;

public:

int isEmpty();

int isFull();

Polinom(int C);

Polinom addition(const Polinom &);

Polinom Minus(const Polinom &);

void Input();

void Add(Monom \*);

Polinom(Monom \*);

Polinom(Polinom &);

Polinom operator=(Polinom &);

void out(char\*);

Polinom();

~Polinom();

};

#include "Polinom.h"

#include<iostream>

using namespace std;

Polinom::Polinom()

{

pFirst = 0;

}

Polinom::Polinom(int C)

{

}

Polinom::Polinom(Polinom &A)

{

pFirst = 0;

for (Monom \*R = A.pFirst; R != 0; R = R->pNext)

{

this->Add(R);

}

}

Polinom Polinom:: operator=(Polinom &A)

{

Monom \*tmp = pFirst;

while (tmp != 0)

{

Monom \*prev = tmp;

tmp = tmp->pNext;

delete prev;

}

pFirst = 0;

for (Monom \*R = A.pFirst; R != 0; R = R->pNext)

{

this->Add(R);

}

return \*this;

}

Polinom::Polinom(Monom \*P)

{

pFirst = P;

}

void Polinom::Add(Monom\*A)

{

Monom \*Q = new Monom(\*A);//создали новый моном для добавления

if (pFirst == 0)

{

pFirst = Q;

}

else

{

Monom \*tmp = pFirst;

Monom \*prev = 0;

while (tmp != 0 && tmp->StepX > Q->StepX && tmp->StepY > Q->StepY && tmp->StepZ > Q->StepZ)

{

prev = tmp;

tmp = tmp->pNext;

}

if (tmp == 0)//добавление в конец

prev->pNext = Q;

else

{

if (tmp->StepX == Q->StepX && tmp->StepY == Q->StepY && tmp->StepZ == Q->StepZ)//если степени равны, то складываем коэффициенты

{

tmp->Data = tmp->Data + Q->Data;

}

else

{

if (prev != 0)//добавление в середину

{

Q->pNext = tmp;

prev->pNext = Q;

}

else//добавление в голову

{

Q->pNext = pFirst;

pFirst = Q;

}

}

}

}

}

int Polinom::isEmpty()

{

return pFirst == 0;

}

int Polinom::isFull()

{

Monom\* tmp = new Monom(1, 0, 0, 0, 0);

int res = (tmp == 0);

if (tmp != 0)

{

delete tmp;

}

return res;

}

Polinom Polinom::addition(const Polinom &A)

{

Monom \*tmp;

Monom \*tmpA;

Polinom res;

tmp = this->pFirst;

tmpA = A.pFirst;

while (tmp != 0 && tmpA != 0)

{

if (tmp->StepX > tmpA->StepX )

{

res.Add(tmp);

tmp = tmp->pNext;

}

else

{

if (tmp->StepX == tmpA->StepX && tmp->StepY == tmpA->StepY && tmp->StepZ == tmpA->StepZ)

{

res.Add(new Monom(tmp->Data + tmpA->Data, 0, tmp->StepX, tmp->StepY, tmp->StepZ));

tmp = tmp->pNext;

tmpA = tmpA->pNext;

}

else

{

res.Add(tmpA);

tmpA = tmpA->pNext;

}

}

}

//дописываем остатки хвостов

while (tmp != 0)

{

res.Add(tmp);

tmp = tmp->pNext;

}

while (tmpA != 0)

{

res.Add(tmpA);

tmpA = tmpA->pNext;

}

return res;

}

Polinom Polinom::Minus(const Polinom &M)

{

Monom \*tmp;

Monom \*tmpA;

Polinom res;

tmp = this->pFirst;

tmpA = M.pFirst;

while (tmp != 0 && tmpA != 0)

{

if (tmp->StepX > tmpA->StepX)

{

res.Add(tmp);

tmp = tmp->pNext;

}

else

{

if (tmp->StepX == tmpA->StepX && tmp->StepY == tmpA->StepY && tmp->StepZ == tmpA->StepZ)

{

res.Add(new Monom(tmp->Data - tmpA->Data, 0, tmp->StepX, tmp->StepY, tmp->StepZ));

tmp = tmp->pNext;

tmpA = tmpA->pNext;

}

else

{

res.Add(tmpA);

tmpA = tmpA->pNext;

}

}

}

//дописываем остатки хвостов

while (tmp != 0)

{

res.Add(tmp);

tmp = tmp->pNext;

}

while (tmpA != 0)

{

res.Add(tmpA);

tmpA = tmpA->pNext;

}

return res;

}

void Polinom::Input()

{

int StepX,StepY,StepZ;

int Data;

char Y;

while (1)

{

cout << "enter step X: ";

cin >> StepX;

cout << "enter step Y: ";

cin >> StepY;

cout << "enter step Z: ";

cin >> StepZ;

cout << "enter data: ";

cin >> Data;

Monom M(Data,0,StepX,StepY,StepZ);

this->Add(&M);

cout << "more(Y/N) ?: ";

cin >> Y;

if (Y == 'N' || Y == 'n')

break;

}

cout << endl;

}

void Polinom::out(char\*Z)

{

cout << Z << endl;

for (Monom \*i = pFirst; i != 0; i = i->pNext)

{

if (i->Data == 0)

cout << "your monom is= 0" << endl;

else

{

{

cout << "your monom is= " << i->Data << "\*" << "(" << " X^" << i->StepX << "\*" << " Y^" << i->StepY << "\*" << " Z^" << i->StepZ << ")";

cout << endl;

}

}

}

}

Polinom::~Polinom()

{

Monom \*tmp = pFirst;

while (tmp != 0)

{

Monom \*prev = tmp;

tmp = tmp->pNext;

delete prev;

}

}

1. Класс «Односвязный список»

class List

{

public:

PLink current;

List(void);

List(List&);

~List(void);

void AddLink(const int&, const Rational&);

void GoNext(void);

bool DelCurrentLink(void);

void ResetToFirst(void);

bool IndexOf(const int&);

const List& operator= (List);

int GetDegreeCollection(void) const;

Rational GetValue(void) const;

void SetValue(const Rational&);

void print(void);

};

1. Класс «Полином»

class Polynomial

{

List MonomList;

int\* GetDegree(void);

public:

void AddMonom(const int&, const Rational&);

Polynomial operator+ (Polynomial);

const Polynomial& operator= (Polynomial);

Polynomial(Polynomial&);

Polynomial(void);

friend ostream& operator<< (ostream& os, Polynomial Buffer);

};

**Структура программного комплекса**

Main.cpp;

Polinom.h

Polinom.cpp

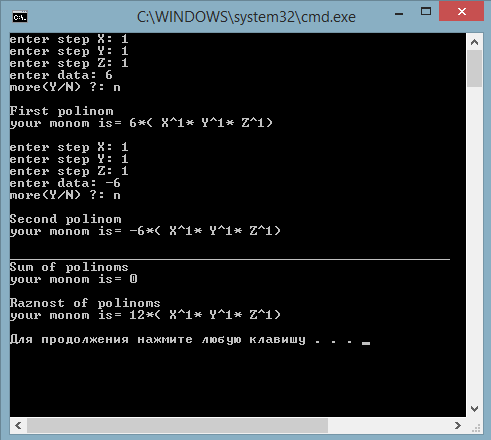
Monom.h

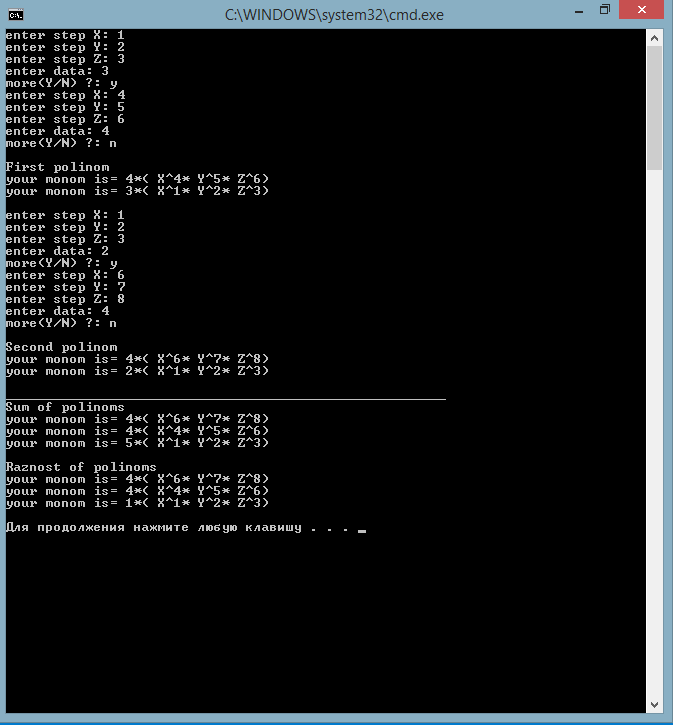
Monom.cpp

# Заключение

В результате выполнения данной лабораторной работы была создана программа, реализующая работу с полиномами от трёх переменных.

Все написанные классы были протестированы.





# Список используемой литературы

Б. Страуструп Язык программирования С++. Специальное издание. Пер. с англ. – М.: ООО «Бином-Пресс», 2008 г. – 1104 с.: ил.

# Приложение

Rational.h

#ifndef \_\_RATIONAL

#define \_\_RATIONAL

#include <iostream>

using namespace std;

class Rational

{

public:

Rational ();

Rational (const int&, const int&);

Rational (const Rational&);

const Rational& operator= (const Rational&);

Rational operator+ (const Rational&);

Rational operator- (const Rational&);

bool operator == (Rational&);

friend ostream& operator<< (ostream& ostr, const Rational&);

friend istream& operator>> (istream& istr, Rational& b);

int GetNum(void) const;

private:

int num;

unsigned int den;

void reduce();

unsigned int NOK(const Rational&);

};

#endif

Rational.cpp

#include "Rational.h"

int NOD(const int& a1, const int& b1)

{

int a = abs(a1), b = abs(b1);

while(a != 0 && b != 0){

if (a >= b)

a = a % b;

else

b = b % a;

}

return a + b;

}

Rational::Rational() : num(0), den(1)

{

}

Rational::Rational(const int& a, const int& b)

{

if (b < 0)

exit(1);

num = a;

if (b)

den = b;

else

exit(1);

reduce();

}

Rational::Rational(const Rational& Copy)

{

num = Copy.num;

den = Copy.den;

}

const Rational& Rational::operator= (const Rational& Copy)

{

num = Copy.num;

den = Copy.den;

return \*this;

}

Rational Rational::operator+ (const Rational& Ratio)

{

int Tmp = NOK(Ratio);

Rational t;

int c1 = Tmp / den, c2 = Tmp / Ratio.den;

t.num = num \* c1 + Ratio.num \* c2;

t.den = Tmp;

t.reduce();

return t;

}

Rational Rational::operator- (const Rational& Ratio)

{

int Tmp = NOK(Ratio);

Rational t;

int c1 = Tmp / den, c2 = Tmp / Ratio.den;

t.num = num \* c1 - Ratio.num \* c2;

t.den = Tmp;

t.reduce();

return t;

}

bool Rational::operator == (Rational& sr)

{

sr.reduce();

if (den == sr.den && num == sr.num)

return true;

else

return false;

}

unsigned int Rational::NOK(const Rational& Ratio)

{

return Ratio.den \* den / NOD(Ratio.den, den);

}

void Rational::reduce()

{

int nod = NOD(num,den);

num = num / nod;

den = den / nod;

}

ostream& operator<< (ostream& ostr, const Rational& b)

{

if (b.den == 1 || b.num == 0)

ostr << b.num;

else

ostr << b.num << '/' << b.den;

return ostr;

}

int Rational::GetNum(void) const

{

return num;

}

istream& operator>> (istream& istr, Rational& b){

char\* str = new char [1024];

cin.getline(str, 1024);

int i = 0;

unsigned int bt = 0;

bool flag = false;

l1:

int len = strlen(str)-1;

while (str[len]==' '){

str[len] = NULL;

goto l1;

}

if (strlen(str)==0)

exit(1);

while (str[i]==' ')

++i;

if (str[i]=='-'){

flag = true;

++i;

}

while (str[i]!='.' && str[i]!='/' && i<strlen(str)){

bt = (str[i] - '0') + bt\*10;

++i;

}

++i;

switch (str[i-1]){

case '.': {

int j = 0;

while (i<(int)strlen(str)){

bt = (str[i] - '0') + bt\*10;

++i, ++j;

}

b.den = int(pow(10.0,j));

break;

}

case '/': {

b.den = 0;

while (i<(int)strlen(str)){

b.den = (str[i]-'0') + b.den\*10;

++i;

}

break;

}

}

b.num = flag ? -int(bt) : bt;

b.reduce();

return istr;

}

Link.h

#ifndef \_Link

#define \_Link

#include "Rational.h"

class Link;

typedef Link\* PLink;

class Link

{

public:

Rational Value;

int DegreeCollection;

PLink next;

Link(void);

Link(const int&, const Rational&);

};

#endif

Link.cpp

#include "Link.h"

#include <lmcons.h>

Link::Link(void)

{

next = NULL;

DegreeCollection = 0;

Value = Rational(0, 1);

}

Link::Link(const int& IntValue, const Rational& RationalValue)

{

DegreeCollection = IntValue;

Value = RationalValue;

next = NULL;

}

List.h

#ifndef \_List

#define \_List

#include "Link.h"

#include <iostream>

using namespace std;

class List

{

public:

PLink current;

public:

List(void);

List(List&);

~List(void);

void AddLink(const int&, const Rational&); // добавить звено после текущего;

void GoNext(void); // идем по списку вправо;

bool DelCurrentLink(void); // удаление текущего звена;

void ResetToFirst(void); // переместить в начало списка;

bool IndexOf(const int&); // найти звено по DegreeCollection;

const List& operator= (List);

int GetDegreeCollection(void) const;

Rational GetValue(void) const;

void SetValue(const Rational&);

void print(void);

};

#endif

List.cpp

#include "List.h"

List::List(void)

{

current = new Link;

current->DegreeCollection = -1;

current->next = current;

}

List::~List(void)

{

ResetToFirst();

while(DelCurrentLink())

;

}

List::List(List& CopyList)

{

current = new Link;

current->DegreeCollection = -1;

current->next = current;

CopyList.ResetToFirst();

while(CopyList.GetDegreeCollection() >= 0)

{

AddLink(CopyList.GetDegreeCollection(), CopyList.GetValue());

CopyList.GoNext();

}

}

void List::AddLink(const int& ValueInt, const Rational& ValueRational)

{

PLink p = new Link(ValueInt, ValueRational);

p->next = current->next;

current->next = p;

current = p;

}

void List::GoNext(void)

{

current = current->next;

}

int List::GetDegreeCollection(void) const

{

return current->DegreeCollection;

}

Rational List::GetValue(void) const

{

return current->Value;

}

bool List::DelCurrentLink(void)

{

if (current->next == current)

return false;

else

{

PLink Tmp;

Tmp = current;

ResetToFirst();

while (current->next != Tmp)

GoNext();

current->next = Tmp->next;

GoNext();

delete Tmp;

return true;

}

}

void List::ResetToFirst(void)

{

while(current->DegreeCollection >= 0)

GoNext();

GoNext();

}

void List::print(void)

{

ResetToFirst();

int i = 0;

while(current->DegreeCollection != -1)

{

cout << ++i << '.' << endl << "Value = " << current->Value << endl;

cout << "DegreeCollection = " << current->DegreeCollection << endl;

GoNext();

}

}

void List::SetValue(const Rational& RationalValue)

{

current->Value = RationalValue;

}

bool List::IndexOf(const int& BufferValue)

{

ResetToFirst();

while(current->DegreeCollection != -1)

{

if (current->DegreeCollection == BufferValue)

return true;

GoNext();

}

return false;

}

const List& List::operator= (List CopyList)

{

this->~List();

List();

CopyList.ResetToFirst();

while (CopyList.GetDegreeCollection() >= 0)

{

AddLink(CopyList.GetDegreeCollection(), CopyList.GetValue());

CopyList.GoNext();

}

return \*this;

}

Polynomial.h

#ifndef \_Polynomial

#define \_Polynomial

#include "List.h"

#include "Rational.h"

static int BASE = 100;

#define n 3

class Polynomial

{

List MonomList;

// список коэффициентов полинома;

int\* GetDegree(void);

// получаем степени из набора;

public:

void AddMonom(const int&, const Rational&); // добавить звено;

Polynomial operator+ (Polynomial); // оператор +;

const Polynomial& operator= (Polynomial);

Polynomial(Polynomial&);

Polynomial(void);

friend ostream& operator<< (ostream& os, Polynomial Buffer);

};

#endif

Polynomial.cpp

#include "Polynomial.h"

Polynomial::Polynomial(Polynomial& CopyBuffer)

{

MonomList = CopyBuffer.MonomList;

}

Polynomial::Polynomial(void)

{

}

int\* Polynomial::GetDegree(void)

{

int \*degree = new int[n];

degree[0] = MonomList.GetDegreeCollection() / (BASE \* BASE);

degree[1] = MonomList.GetDegreeCollection() / BASE % BASE;

degree[2] = MonomList.GetDegreeCollection() % BASE;

return degree;

}

void Polynomial::AddMonom(const int& DegreeValue, const Rational& RationalValue)

{

if (MonomList.IndexOf(DegreeValue))

if ((MonomList.GetValue() + RationalValue) == Rational(0, 1))

MonomList.DelCurrentLink();

else

MonomList.SetValue(MonomList.GetValue() + RationalValue);

else

{

while(MonomList.current->DegreeCollection >= 0)

MonomList.GoNext();

while(DegreeValue < MonomList.current->next->DegreeCollection)

MonomList.GoNext();

MonomList.AddLink(DegreeValue, RationalValue);

}

}

Polynomial Polynomial::operator+ (Polynomial PolynomialBuffer)

{

Polynomial Tmp;

MonomList.ResetToFirst();

Tmp.MonomList = MonomList;

PolynomialBuffer.MonomList.ResetToFirst();

while(PolynomialBuffer.MonomList.GetDegreeCollection() >= 0)

{

Tmp.AddMonom(PolynomialBuffer.MonomList.GetDegreeCollection(), PolynomialBuffer.MonomList.GetValue());

PolynomialBuffer.MonomList.GoNext();

}

return Tmp;

}

const Polynomial& Polynomial::operator= (Polynomial CopyBuffer)

{

MonomList = CopyBuffer.MonomList;

return \*this;

}

ostream& operator<< (ostream& os, Polynomial Buffer)

{

Buffer.MonomList.ResetToFirst();

char symbol[3] = {'x', 'y', 'z'};

while(Buffer.MonomList.GetDegreeCollection() >= 0)

{

int\* degree = Buffer.GetDegree();

os << Buffer.MonomList.GetValue();

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

if (!degree[i])

continue;

else

if (degree[i] == 1)

os << '\*' << symbol[i];

else

os << '\*' << symbol[i] << '^' << degree[i];

if (Buffer.MonomList.current->next->DegreeCollection != -1 && Buffer.MonomList.current->next->Value.GetNum() >= 0)

os << " + ";

Buffer.MonomList.GoNext();

}

return os;

}

main.cpp

#include <conio.h>

#include "Rational.h"

#include "Polynomial.h"

int SetDegreeCollections(int\* degree)

{

int Tmp = degree[0];

for (int i = 1; i < n; ++i)

Tmp = degree[i] + (Tmp \*= BASE);

return Tmp;

}

#define size 2

int main()

{

BASE = 100;

setlocale(LC\_ALL, "Russian");

Polynomial \*A = new Polynomial[size];

Rational Index;

int degree[n];

cout << "Введите коэффициенты и наборы степеней мономов полинома" << endl;

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

{

while(\_getch() != '1')

{

cout << "Коэффициент ";

fflush(stdin);

cin >> Index;

cout << "Набор степеней ";

fflush(stdin);

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

cin >> degree[i];

A[j].AddMonom(SetDegreeCollections(degree), Index);

}

cout << "Введенный полином" << endl << A[j] <<endl;

}

cout << "Сумма: " << A[0] + A[1] << endl;

\_getch();

return 0;

}