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

Митрохин Никита Павлович.

Проверила: Шестакова Наталья Валерьевна.

Нижний Новгород

2016

# Оглавление

[Оглавление 1](#_Toc279967395)

[Введение 3](#_Toc279967396)

[Постановка задачи 4](#_Toc279967397)

[Руководство пользователя 5](#_Toc279967398)

[Руководство программиста 6](#_Toc279967399)

[Описание алгоритмов 7](#_Toc279967400)

[Структура данных 7](#_Toc279967401)

[Структура программного комплекса 9](#_Toc279967402)

[Заключение 9](#_Toc279967403)

[Список используемой литературы 12](#_Toc279967404)

[Приложение 13](#_Toc279967405)

[Rational.h 13](#_Toc279967406)

[Rational.cpp 13](#_Toc279967407)

[Link.h 15](#_Toc279967408)

[Link.cpp 16](#_Toc279967409)

[List.h 16](#_Toc279967410)

[List.cpp 17](#_Toc279967411)

[Polynomial.h 18](#_Toc279967412)

[Polynomial.cpp 19](#_Toc279967413)

[main.cpp 20](#_Toc279967414)

# Введение

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



,

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

, где *cI* - фиксированные коэффициенты, а x – переменная.



Коэффициенты многочлена обычно берутся из определённого коммутативного кольца *R* (чаще всего поля, например, поля вещественных или комплексных чисел). В этом случае, относительно операций сложения и умножения многочлены образуют кольцо (более того ассоциативно-коммутативную алгебру над кольцом R без делителей нуля) которое обозначается *R[x1, x2, ..., xn].*

Изучение полиномиальных уравнений и их решений составляло едва ли не главный объект «классической алгебры». С изучением многочленов связан целый ряд преобразований в математике: введение в рассмотрение нуля, отрицательных, а затем и комплексных чисел, а также появление теории групп как раздела математики и выделение классов специальных функций в анализе.

Техническая простота вычислений, связанных с многочленами, по сравнению с более сложными классами функций, а также тот факт, что множество многочленов плотно в пространстве непрерывных функций на компактных подмножествах евклидова пространства, способствовали развитию методов разложения в ряды и полиномиальной интерполяции в математическом анализе.

Многочлены также играют ключевую роль в алгебраической геометрии, объектом которой являются множества, определённые как решения систем многочленов. Особые свойства преобразования коэффициентов при умножении многочленов используются в алгебраической геометрии, алгебре, теории узлов и других разделах математики для кодирования, или выражения многочленами свойств различных объектов.

# Постановка задачи

Требуется разработать программный комплекс, реализующий работу с полиномами от нескольких переменных. В рассматриваемом случае для определенности будем работать с полиномами от трех переменных.

В качестве структуры хранения для полинома необходимо взять односвязный циклический список, поскольку добавление и удаление мономов из полинома – частая работа с динамической памятью. Мономы будут являться звеньями списка, хранящие коэффициент монома, свернутую степень, указатель на следующее звено-моном.

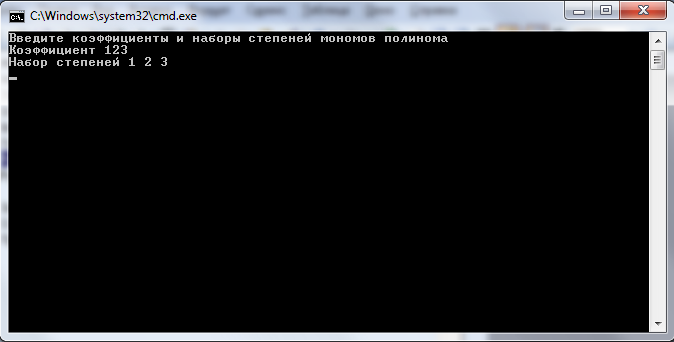
Добавление мономов в полином происходит поэтапно. На каждой итерации необходимо проверять, существует ли уже в полиноме моном с таким набором степеней Если существует, то необходимо сложить коэффициенты мономов, предварительно проверив их сумму на равенство, ведь если сумма равна 0, то нет смысла хранить данный моном в списке. То есть в таком случае удаляем звено и никаких новых добавлений мономов на данной итерации происходить не будет.

Для удобства необходимо хранить мономы в списке используя следующий порядок: для любой пары соседних мономов *DegreeCollection(i) > DegreeCollection(i + 1)*,

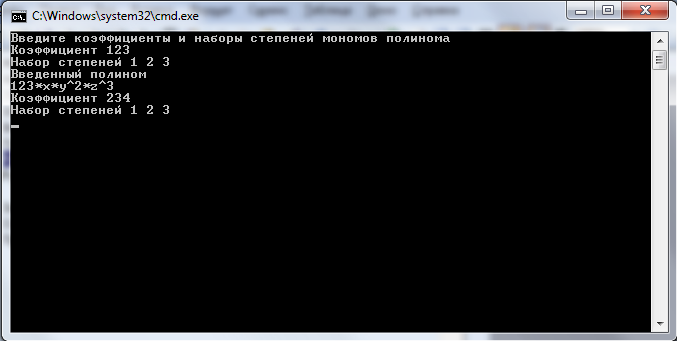
то есть в порядке убывания свернутых степеней.

# Руководство пользователя

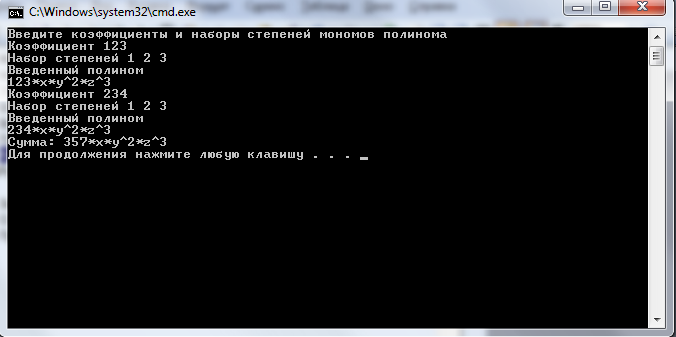
Для запуска программы откройте файл *Polynomial.exe*. Появится окно консоли с сообщением «Введите коэффициенты и наборы степеней мономов полинома». Необходимо нажать на клавишу «Enter». Выведется строка со значением «Коэффициент», где необходимо ввести коэффициент добавляемого монома и вновь нажать «Enter». Затем пользователю необходимо ввести набор степеней для монома и нажать «Enter», например:



Если необходимо еще ввести мономы, то нажмите «Enter», иначе клавишу c цифрой *1*. После чего выведется составленный из введенных мономов полином. Затем аналогичным образом нужно ввести второй полином.



Результатом работы программы является полученные в результате сложение двух введенных полиномов третий полином. Он выведен в графе «Сумма».



# 

# Руководство программиста

**Описание алгоритмов**

Основной структурой данных, используемой в данной лабораторной работе, является кольцевой односвязный список. Это структура данных, состоящая из звеньев, каждый из которых содержит как собственные данные, так и одну ссылку («связки») на следующее или предыдущее звено.

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.

Основные алгоритмические моменты методы сложения полиномов состоят в следующем (перегрузка оператора operator+): операция сложения сводится в последовательной обработке мономов складываемых полиномов:

* Все мономы первого полинома добавляются в отдельный полином *Tmp*;
* Затем последовательно добавляются мономы второго полинома по следующему принципу:
* Если в полиноме *Tmp* есть моном такой же степени, как и текущий рассматриваемый моном во втором полиноме, то, если значения коэффициентов в сумме дают *0*, то моном удаляется из *Tmp* и текущий указатель во втором полиноме перемещается на следующий моном;
* Если же в *Tmp* нет монома такой же степени, то ищется позиция для вставки монома в соответствии с порядком убывания степеней;

Так же стоит отметить способ хранения степеней монома в звене. Это значение высчитывается следующим образом: есть набор степеней *a1*, *a2*, *a3*. Это все переводится в единое число и хранится в звене:

*DegreeCollection = a1\*BASE^2 + a2 \* BASE + a3*,

где *BASE* – играет роль системы счисления (принимает значение *10^n*, *n* – натуральное число).

**Структура данных**

Для реализации алгоритмов было использовано 4 класса:

* Класс «Рациональные числа» (Rational);
* Класс «Звено» (Link);
* Класс «Односвязный список» (List);
* Класс «Полином» (Polynomial);

Описание классов:

### Класс «Рациональные числа»

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&);

};

1. Класс «Звено»

class Link;

typedef Link\* PLink;

class Link

{

public:

Rational Value;

int DegreeCollection;

PLink next;

Link(void);

Link(const int&, const Rational&);

};

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 – файл, реализующий интерфейс пользователя.

Link.h – заголовочный файл класса Link.

Link.cpp – файл реализации класса Link

Rational.h – заголовочный файл класса Rational.

Rational.cpp – файл реализации класса Rational.

List.h – заголовочный файл класса List.

List.cpp - файл реализации класса List.

Polynomial.h – заголовочный файл класса Polynomial.

Polynomial.cpp - файл реализации класса Polynomial.

# 

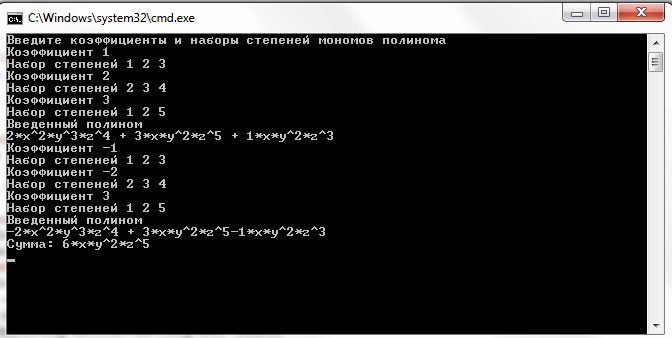
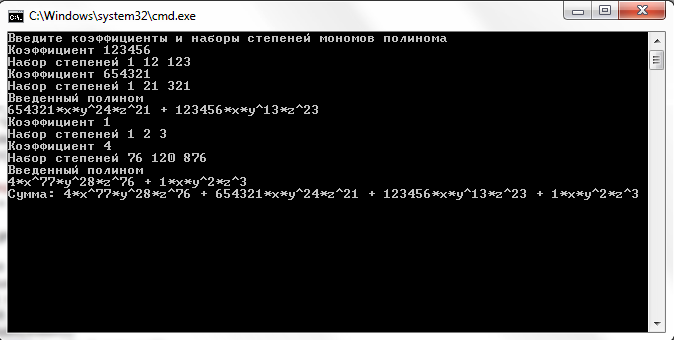
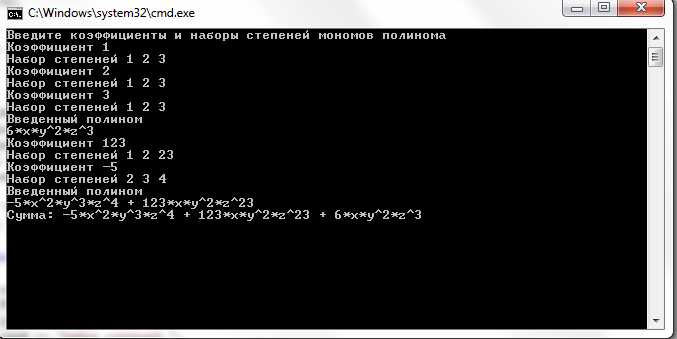
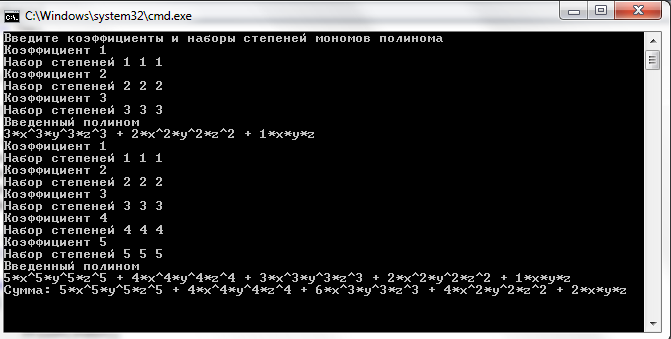
# Заключение

В результате выполнения данной лабораторной работы был разработан программный комплекс, реализующий работу с полиномами от трёх переменных.

Разработанные классы можно совершенствовать, добавляя туда необходимые разработчику функции, например, вычитание, деление с остатком, умножение и т.д.

Все написанные классы были протестированы.

Таким образом, задачи, поставленные в начале, были успешно выполнены, а результаты проверены.



# Список используемой литературы

Гергель В.П. «Рабочие материалы к учебному курсу «Методы программирования» ННГУ, 2002

Культин Н.Б. Основы программирования в Microsoft Visual C++ 2010. – СПб.: БХВ-Петербург, 2010. – 384 с.: ил.

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;

}