**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное автономное образовательное**

**учреждение высшего образования**

**«Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ**

**Тема**

**Полиномы**

Исполнитель:

студентка 2 курса

Антонова Л.А

Проверил:

аспирант кафедры МОСТ

Лебедев С.А.

**Нижний Новгород, 2017**

Содержание

[1. Введение 3](#_Toc470428035)

[2. Постановка задачи 4](#_Toc470428036)

[3.Рукводство пользователя 5](#_Toc470428037)

[4. Руководство программиста 8](#_Toc470428038)

[4.1 Описание структур данных 9](#_Toc470428039)

[4.2 Описание алгоритмов 11](#_Toc470428040)

[5. Заключение 13](#_Toc470428041)

[6. Литература 14](#_Toc470428042)

[7. Приложение 15](#_Toc470428043)

# Введение

# Мономом называется выражение вида , где i1, i2, … in – степени каждой переменной.

# Терм включает в себя коэффициент и моном, содержащий одну или несколько переменных, каждая из которых должна иметь степень.

# Полиномом называется конечная сумма термов

# В число возможных вычислительных процедур над полиномами входят действия по вычислению значений переменых, а также большинство математических операций (сложение, вычитание, умножение, вычисление частных производных, интегрирование и т.д)

### Полиномы, как математический объект, хорошо изучены в математике. Вычисление и привидение полиномов к каноническому виду очень важно для корректной работы многих алгоритмов. Поэтому для корректного вычисления и применения операций над полиномами необходимы компьютерные технологии. Это значительно увеличивает скорость обработки полиномов.

В лабораторной работе будет реализовано хранение и обработка полиномов от трех переменных. В качестве структуры хранения для данной алгоритмической системы будем использовать односвязный список.

# **Постановка задачи**

Цель данной работы — разработать структуру данных Односвязный список и с его помощью реализовать хранение полиномов от трех переменных. Для удобства, ограничим максимально возможную степень каждого полинома девятью. Таким образом полином будет представлен в виде Списка мономов, где моном – структура состоящая из двух полей: коэффициент и степень. Так как степень каждой переменной мы ограничили, её можно представить в виде одного трехзначного числа, где сотни будут соответствовать степени переменной Х, десятки – Y, единицы – Z;

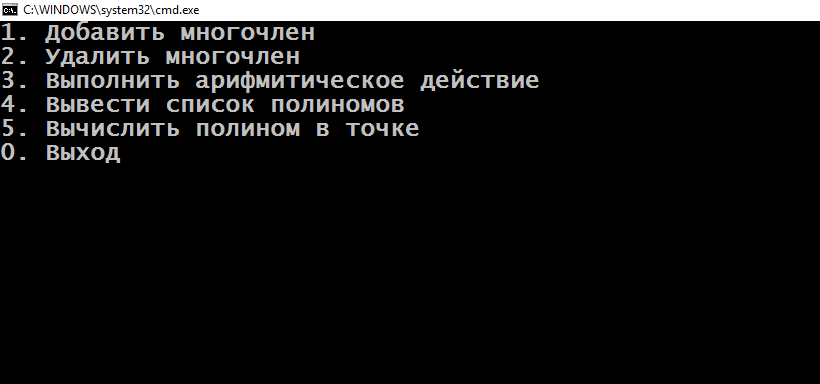
Выполнение работы предполагает решение следующих задач:

1. Разработка шаблонного класса `TList`.
2. Разработка интерфейса класса `Polynom` для работы с .полиномом
3. Разработка и реализация тестов для классов ` TList ` и ` Polynom ` на базе Google Test.
4. Реализация диалога с пользователем через консоль.

Допустимые операции над полиномами: сложение, вычитание, умножение, вычисление значения полинома в заданной точке.

# Руководство пользователя

Для работы с программной запустите «polynom.exe». На экране появится консоль:



Для того чтобы начать работу с полиномом сначала необходимо его добавить. Для этого введите «1», затем нужный многочлен в формате: С1<моном 1>+C2<моном 2>+…Cn<мноном n>, где

* C1, … Сn – коэффициенты, целые или десятичные числа.
* Каждый моном представляет собой последовательность X<степень X>Y<степень Y>Z<степень Z>.
* Если степень переменной 0, то переменную можно опустить
* Если степень переменной 1, то степень можно опустить
* Переменные вводятся в произвольном порядке

Пример:



При введении полинома необходимо помнить о том, что степень переменной в данной реализации ограничена и не может превышать девяти. При нарушении этого правила выведется сообщение об ошибке.



# Максимальное количество полиномов, которое может хранить программа – 9. Таким образом, если у вас в списке большое количество многочленов, необходимо воспользоваться функцией «Удалить многочлен»

# Для удаления многочлена из списка введите «2». Вам будет предложено ввести номер полинома который вы хотите удалить. После введения этого номера многочлен удалится из списка.

# Для того чтобы убедиться что полином удален вы можете вывести список полиномов командой «4».

# Пример:

# 

# 

# 

# Как вы могли заметить, полином с номером 3 исчез из списка и количество полиномов уменьшилось с 5 до 4.

# Для выполнения арифметических операций над полиномами введите «3». Для этого необходимо чтобы в памяти находился хотя бы один многочлен и чтобы количество полиномов в списке не превышало максимального. Вам будет предложено выбрать из трех операций «+», «-«, «\*». Введите номер операции чтобы продолжить. Затем вам будет предложено выбрать полиномы из списка в формате <номер полинома><операция><номер полинома>. Результат выведется на экран и автоматически добавится в список.

# 

# 

# Так же в программе есть возможность вычислить значение полинома в точке. Для этого введите команду «5» (в памяти должен находиться хотя бы один полином), выберите полином, а затем введите значения переменных в формате <значение X> <значение Y> <значение Z>.

# 

# Для того чтобы выйти из программы введите «0».

# Руководство программиста

## Описание структур данных

Программа состоит из двух классов **template class TList** и **Polynom**. **TList** в данном проекте предназначен для хранения мономов, предоставляет интерфейс для доступа к элементам без использования указателей [**class Polynom**](#_class_Polynom) реализует арифметические операции, вычисление в точке, ввод/вывод для полинома.

**Структура для хранения узлов списка**

**template <class T> struct Node**

T data;

Node\* pNext;

**template class List**

Поля:

Node<T> \*pHead; указатель на первый элемент списка:

Node<T> \*pCurrent; указатель на текущий элемент списка:

Node<T> \*preCurrent; указатель на элемент списка перед текущим:

int length; длина списка:

Методы:

TList() конструктор по умолчанию:

TList(const TList &l) конструктор копирования:

bool isEmpty() const проверка на пустоту:

~TList() деструктор:

Node<T>\* GetHead() получить первый элемент:

Node<T>\* GetCurrent() получить текущий элемент:

int GetLength() получить длину:

void PushBack(T \_data) вставить в конец списка:

void PushFront(T \_data) вставить в начало:

void SetCurrent() установить текущий элемент на начало:

bool MoveCurrent() передвинуть текущий:

void PushAfterCurrent(T \_data) вставка после текущего:

void PushBeforeCurrent(T \_data) вставка перед текущим:

Перегруженные операторы

TList<T>& operator = (const TList<T> &l)

friend bool operator ==(const TList<T> &l1, const TList<T> &l2

friend bool operator !=(const TList<T> &l1, const TList<T> &l2)

**struct Monom**

Поля:

int degree степень:

double coef коэффициент:

Перегруженные операторы:

bool operator==(Monom& m)

bool operator!=(Monom& m)

**class Polynom**

Поля:

TList<Monom> list лист мономов, образуюший полином:

Методы:

void CreatePolynom(const char\* str); создать полином из строки:

Polynom() конструктор по умолчанию:

Polynom(string str) { CreatePolynom(str.c\_str()); } конструктор с параметром-строкой:

Polynom(const Polynom& p); конструктор копирования:

bool isEmpty() проверка на пустоту:

void addEmpty(); добавить пустой:

void AddMonom(Monom m); добавить моном:

void SetCurrentMonom() установить курсор на начало:

bool NextMonom() передвинуть курсор:

double GetNextCoef() получить следующий коэффициент:

void InsertAfterCurrentMonom(Monom m) вставить перед текущего:

void InsertBeforeCurrentMonom(Monom m) вставить после текущего:

void InsertBack(Monom m) вставить в конец:

Monom GetCurrentMonom() получить текущий моном:

double Calculate(double X, double Y, double Z); посчитать значение в точке:

Переггрузка операторов

Polynom& operator=(const Polynom &p);

Polynom operator+(Polynom& p);

Polynom operator-(const Polynom& p);

Polynom operator\*(double c);

Polynom operator\*(Monom m);

Polynom operator\*(Polynom p);

bool operator==(const Polynom &p) const

bool operator!=(const Polynom &p) const

friend ostream& operator<<(ostream& os, Polynom &pol)

## **Описание алгоритмов**

**Сложение/вычитание полиномов**

В обоих полиномах устанавливается указатель на первый моном.

Пока в одном из полиномов не закончатся мономы выполняется цикл.

Если степень монома в полиноме номер 1 больше степени текущего монома в полиноме 2, то в результирующий полином добавляется моном из первого полинома и в первом полиноме указатель сдвигается на следующий моном. Если степени мономов равны, то создаётся моном tmp с такой же степенью и коэффициентом равным сумме коэффициентов мономов и добавляется к результату. Оба указателя сдвигаются. Если степень монома в полиноме 1 меньше степени монома в полиноме 2, то добавляется моном 2 и сдвигается указатель во втором полиноме.

Далее, если в каком либо из полиномов остались не задействованные мономы (указатель можно сдвинуть дальше), то они добавляются в результат.

**Умножение полинома на моном**

В мономе на который требуется выполнить умножение, вычисляются степени для X, Y, Z. В полиноме устанавливается указатель на первый моном.

Пока все мономы не будут умножены выполнятся цикл.

Вычисляются отдельно степени X, Y, Z для текущего монома в полиноме. Создается моном tmp с коэффициентом равным произведению мономов. Далее вычисляются суммы степеней, если одна из них больше 9, выдается ошибка. Иначе вычисляется трехзначное число, которое записывается как степень tmp. Tmp добавляется к результату.

**Умножение полиномов.**

Последовательно умножается полином на мономы другого полинома. Результат складывается.

**Вычисление полинома в точке**

Получаем отдельно степень каждой переменной. С помощью рекурсивной функции возведения в степень получаем результат.

**Возведение в степень**

Если степень равна 0, возвращаем 1. Если степень кратна 2, то возвращается (вызывается) функция возведения в степень для a в квадрате и степени меньшей в 2 раза. Иначе возвращаем a\*power(a, x-1)

# **Заключение**

# В программе реализована структура данных для хранения полиномов с использованием односвязного списка. Реализован простейший диалог с пользователем, позволяющий любому человеку работать с данной программой. К приложению были написаны тесты, подтверждающие корректность его работы.

# 6. **Литература**

1. В. П. Гергель Методические материалы к курсу «Методы программирования»/В.П. Гергель – Н.Новгород: изд-во Типография ННГУ, 2015. - 96 с.

# 7. Приложение

**Фрагменты исходного кода программы.**

**Файл TList.h**

template <class T>

struct Node

{

T data;

Node\* pNext;

};

template <class T>

class TList

{

private:

Node<T> \*pHead;

Node<T> \*pCurrent;

Node<T> \*preCurrent;

int length;

public:

TList()

{

pHead = NULL;

length = 0;

}

TList(const TList &l)

{

Node<T> \*p = l.pHead;

while (p != 0)

{

PushBack(p->data);

p = p->pNext;

}

}

bool isEmpty() const { return (pHead == 0); }

~TList()

{

Node <T> \*p;

while (!isEmpty())

{

p = pHead;

pHead = pHead->pNext;

delete p;

}

}

Node<T>\* GetHead() { return pHead; }

Node<T>\* GetCurrent() { return pCurrent; }

int GetLength() { return length; }

void PushBack(T \_data)

{

Node<T> \*p = pHead;

Node<T> \*newNode = new Node<T>;

newNode->data = \_data;

newNode->pNext = NULL;

// если список пуст

if (p == NULL) {

pHead = newNode;

}

else {

// иначе

while (p->pNext != NULL) {

p = p->pNext;

}

p->pNext = newNode;

}

length++;

}

void PushFront(T \_data)

{

Node<T> \*newNode = new Node<T>;

newNode->data = \_data;

newNode->pNext = pHead;

pHead = newNode;

length++;;

}

void SetCurrent()

{

pCurrent = pHead;

preCurrent = 0;

}

bool MoveCurrent()

{

if (pCurrent == 0) return false;

preCurrent = pCurrent;

if (pCurrent != 0)

pCurrent = pCurrent->pNext;

return (pCurrent != 0);

}

void PushAfterCurrent(T \_data)

{

Node<T> \*newNode = new Node<T>;

newNode->data = \_data;

if (pCurrent != 0)

{

newNode->pNext = pCurrent->pNext;

pCurrent->pNext = newNode;

}

else

pCurrent = newNode;

length++;

}

void PushBeforeCurrent(T \_data)

{

Node<T> \*newNode = new Node<T>;

newNode->data = \_data;

newNode->pNext = pCurrent;

if (preCurrent != 0)

preCurrent->pNext = newNode;

else

pHead = newNode;

length++;

}

TList<T>& operator = (const TList<T> &l)

{

if (this == &l)

return \*this;

Node <T> \*tmp;

while (!isEmpty())

{

tmp = pHead;

pHead = pHead->pNext;

delete tmp;

}

tmp = l.pHead;

while (tmp != 0)

{

PushBack(tmp->data);

tmp = tmp->pNext;

}

return \*this;

}

friend bool operator ==(const TList<T> &l1, const TList<T> &l2)

{

Node<T>\* p1 = l1.pHead, \*p2 = l2.pHead;

while (p1 && p2)

{

if (p1->data != p2->data)

return false;

p1 = p1->pNext;

p2 = p2->pNext;

}

return (p1 == 0 && p2 == 0);

}

friend bool operator !=(const TList<T> &l1, const TList<T> &l2)

{

return(!(l1 == l2));

}

};

**Файл TPolynom.h**

struct Monom

{

int degree; //степень

double coef; //коэффициент

bool operator==(Monom& m)

{

return(degree == m.degree && coef == m.coef);

}

bool operator!=(Monom& m)

{

return !(\*this == m);

}

};

class Polynom

{

public:

TList<Monom> list; //полином

void CreatePolynom(const char\* str);

Polynom() {}

Polynom(string str) { CreatePolynom(str.c\_str()); }

Polynom(const Polynom& p);

bool isEmpty() { return list.isEmpty(); }

void addEmpty();

void AddMonom(Monom m);

void SetCurrentMonom() { list.SetCurrent(); }

bool NextMonom() { return list.MoveCurrent(); }

double GetNextCoef()

{

if (list.GetCurrent()->pNext == 0)

return 0;

else

return list.GetCurrent()->pNext->data.coef;

}

void InsertAfterCurrentMonom(Monom m) { list.PushAfterCurrent(m); }

void InsertBeforeCurrentMonom(Monom m) { list.PushBeforeCurrent(m); }

void InsertBack(Monom m) { list.PushBack(m); }

Monom GetCurrentMonom() { return list.GetCurrent()->data; }

Polynom& operator=(const Polynom &p);

Polynom operator+(Polynom& p);

Polynom operator-(const Polynom& p);

Polynom operator\*(double c);

Polynom operator\*(Monom m);

Polynom operator\*(Polynom p);

bool operator==(const Polynom &p) const { return (list == p.list); }

bool operator!=(const Polynom &p) const { return (list != p.list); }

double Calculate(double X, double Y, double Z);

friend ostream& operator<<(ostream& os, Polynom &pol)

{

int degreeX, degreeY, degreeZ, degree;

double coef, nextCoef;

pol.SetCurrentMonom();

if (pol.GetCurrentMonom().coef == 0 || pol.isEmpty())

{

os << "0";

return os;

}

do

{

coef=pol.GetCurrentMonom().coef;

degree = pol.GetCurrentMonom().degree;

degreeX = degree / 100;

degreeY = (degree - degreeX \* 100) / 10;

degreeZ = degree - degreeX\*100-degreeY \* 10;

if (coef == 1)

os << "";

else

if (coef == -1)

os << "-";

else

os << coef;

if (degreeX > 0)

{

if (degreeX > 1)

os << "X^" << degreeX;

else

os << "X";

}

if (degreeY > 0)

{

if (degreeY > 1)

os << "Y^" << degreeY;

else

os << "Y";

}

if (degreeZ > 0)

{

if (degreeZ > 1)

os << "Z^" << degreeZ;

else

os << "Z";

}

nextCoef = pol.GetNextCoef();

if (nextCoef > 0) os << "+";

if (nextCoef == 0) return os;

} while (pol.NextMonom());

}

};

**Файл TPolynom.cpp**

double power(double a, int x)

{

if (x == 0) return 1;

if (x < 0) return power(1 / a, -x);

if (x % 2) return a \* power(a, x - 1);

return power(a \* a, x / 2);

}

Polynom::Polynom(const Polynom &p)

{

list = p.list;

}

void Polynom::addEmpty()

{

Monom m;

m.degree = 0;

m.coef = 0;

list.PushFront(m);

}

void Polynom::AddMonom( Monom m)

{

if (m.coef == 0)

return;

SetCurrentMonom();

if (list.GetHead()==0)

{

InsertBack(m);

return;

}

do

{

if (GetCurrentMonom().degree < m.degree)

{

InsertBeforeCurrentMonom(m);

return;

}

} while (NextMonom());

InsertBack(m);

}

void Polynom::CreatePolynom(const char\* str)

{

int i = 0;

char \*stringVal = new char[10];

int k = 0;

int j = 0;

double monomCoef = 1;

int monomDegree = 0;

char symbolPolinom;

while (str[i] != '\0') {

monomCoef = 1;

monomDegree = 0;

// вычисляем коэф очередного монома

for (; isdigit(str[i]) != 0 || str[i] == '-'||str[i]==','; i++)

{

stringVal[j++] = str[i];

}

stringVal[j] = '\0';

if (stringVal[0] == '-' && stringVal[1] == '\0')

{

stringVal[1] = '1';

stringVal[2] = '\0';

}

if (atof(stringVal) == 0)

monomCoef = 1;

else

monomCoef = atof(stringVal);

j = 0;

int degree;

stringVal[j] = '\0';

// вычисляем степени монома

while (str[i] != '+' && str[i] != '-' && str[i] != '\0')

{

symbolPolinom = str[i++];

for (; isdigit(str[i]) != 0; i++)

{

stringVal[j++] = str[i];

}

stringVal[j] = '\0';

j = 0;

if (atoi(stringVal) == 0)

degree = 1;

else

degree = atoi(stringVal);

if (degree < 0 || degree>9)

{

cout << "Введена неверная степень" << endl;

return;

}

if (symbolPolinom == 'X' || symbolPolinom == 'x')

monomDegree += 100 \* degree;

else if (symbolPolinom == 'Y' || symbolPolinom == 'y')

monomDegree += 10 \* degree;

else if (symbolPolinom == 'Z' || symbolPolinom == 'z')

monomDegree += degree;

j = 0;

stringVal[j] = '\0';

}

// создаем и записывам моном в лист

Monom m;

m.coef = monomCoef;

m.degree = monomDegree;

AddMonom(m);

if (str[i] != '-' && str[i] != '\0')

i++;

}

}

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

{

if (this == &p)

return \*this;

list = p.list;

return \*this;

}

Polynom Polynom::operator+(Polynom& p)

{

Polynom result;

SetCurrentMonom();

p.SetCurrentMonom();

bool check1 = true, check2 = true;

Monom m1, m2;

while (check1 && check2)

{

m1 = GetCurrentMonom();

m2 = p.GetCurrentMonom();

if (m1.degree < m2.degree)

{

result.AddMonom(m2);

check2 = p.NextMonom();

}

else if (m1.degree == m2.degree)

{

Monom tmp;

tmp.coef = m1.coef + m2.coef;

tmp.degree = m1.degree;

result.AddMonom(tmp);

check2 = p.NextMonom();

check1 = NextMonom();

}

else

{

result.AddMonom(m1);

check1 = NextMonom();

}

}

while (check1)

{

m1 = GetCurrentMonom();

result.AddMonom(m1);

check1 = NextMonom();

}

while (check2)

{

m2 = p.GetCurrentMonom();

result.AddMonom(m2);

check2 = p.NextMonom();

}

if (result.isEmpty()) result.addEmpty();

return result;

}

Polynom Polynom::operator-(const Polynom& p)

{

Polynom result;

if (\*this != p) {

Polynom tmp(p);

tmp = tmp\*(-1);

result = tmp + \*this;

}

else

{

result.addEmpty();

}

return result;

}

Polynom Polynom::operator\*(double c)

{

Polynom result;

if (c != 0)

{

Monom tmp;

SetCurrentMonom();

do

{

tmp.coef = GetCurrentMonom().coef\*c;

tmp.degree = GetCurrentMonom().degree;

result.AddMonom(tmp);

} while (NextMonom());

}

else result.addEmpty();

return result;

}

Polynom Polynom::operator\*(Monom m)

{

Polynom result;

Monom tmp;

int degree, degreeX, degreeY, degreeZ, degreeXm, degreeYm, degreeZm;

SetCurrentMonom();

degreeXm = m.degree / 100;

degreeYm = (m.degree - degreeXm \* 100) / 10;

degreeZm = m.degree - degreeXm \* 100 - degreeYm \* 10;

do

{

degree = GetCurrentMonom().degree;

tmp.coef = GetCurrentMonom().coef\*m.coef;

degreeX =degree / 100;

degreeY = (degree - degreeX \* 100) / 10;

degreeZ = degree - degreeX \* 100 - degreeY \* 10;

degreeX += degreeXm;

degreeY += degreeYm;

degreeZ += degreeZm;

tmp.degree = degreeX \* 100 + degreeY \* 10 + degreeZ;

if (degreeX < 10 && degreeY < 10 && degreeZ < 10)

{

result.AddMonom(tmp);

}

else

{

cout << "Умножение невозможно" << endl;

result.addEmpty();

return result;

}

} while (NextMonom());

return result;

}

Polynom Polynom::operator\*(Polynom p)

{

Polynom result;

p.SetCurrentMonom();

Polynom tmp;

bool check = false;

do

{

tmp= (\*this)\*p.GetCurrentMonom();

if (result.isEmpty()) result = tmp;

else result = result + tmp;

if (result.list.GetHead()->data.coef == 0)

{

return result;

}

}

while (p.NextMonom());

return result;

}

double Polynom::Calculate(double X, double Y, double Z)

{

double result=0,tmp=0;

Monom m;

int degreeX, degreeY, degreeZ;

SetCurrentMonom();

do

{

m = GetCurrentMonom();

degreeX = m.degree / 100;

degreeY = (m.degree - degreeX \* 100) / 10;

degreeZ = m.degree - degreeX \* 100 - degreeY \* 10;

tmp = power(X, degreeX)\*power(Y, degreeY)\*power(Z, degreeZ)\*m.coef;

result += tmp;

} while (NextMonom());

return result;

}

**Файл main.cpp**

void Print(Polynom\*p, int count)

{

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

{

cout << i << ". " << p[i] << endl;

}

}

int main()

{

setlocale(LC\_ALL, "Russian");

Polynom \*p = new Polynom[10];

int count = -1;

const int maxSize = 8;

string s;

int choise=-1,choise2, k;

string str;

string menu = { "1. Добавить многочлен \n2. Удалить многочлен\n3. Выполнить арифмитическое действие\n" };

menu += "4. Вывести список полиномов\n5. Вычислить полином в точке \n0. Выход\n";

while (choise != 0)

{

cout << menu;

cin >> choise;

if (choise == 1)

{

system("cls");

if (count <= maxSize)

{

cin >> str;

p[++count].CreatePolynom(str.c\_str());

}

else

cout << "Список полиномов полон,попробуйте удалить полином " << endl;

}

if (choise == 2)

{

system("cls");

if (count > 0) {

Print(p, count);

cin >> choise2;

if (choise >= 0 && choise2 <= count)

{

k = choise2;

for (k; k < count; k++)

{

p[k] = p[k + 1];

}

count--;

}

else cout << "Неверное значение" << endl;

}

else cout << "Нет полиномов для удаления" << endl;

}

if (choise == 3)

{

system("cls");

if (count < maxSize&&count>=0)

{

cout << "Выберите операцию" << endl;

cout << "1. +\n2. -\n3. \*\n";

char op;

cin >> op;

system("cls");

switch (op)

{

case '1':

cout << "Выберите полиномы из списка в формате /\*номер полинома\*/+/\*номер полинома\*/" << endl;

Print(p, count);

int n1, n2;

cin >> n1 >> op >> n2;

if (n1 >= 0 && n1 <= count&&n2 >= 0 && n2 <= count)

{

p[++count] = p[n1] + p[n2];

cout << "Результат:" << p[count] << endl;

}

else cout << "Введены неверные значения" << endl;

break;

case '2':

cout << "Выберите полиномы из списка в формате /\*номер полинома\*/-/\*номер полинома\*/" << endl;

Print(p, count);

cin >> n1 >> op >> n2;

if (n1 >= 0 && n1 <= count&&n2 >= 0 && n2 <= count)

{

p[++count] = p[n1] - p[n2];

cout << "Результат:" << p[count] << endl;

}

else cout << "Введены неверные значения" << endl;

break;

case '3':

cout << "Выберите полином из списка в формате /\*номер полинома\*/\*/\*номер полинома\*/" << endl;

Print(p, count);

double c;

cin >> n1 >> op >> n2;

if (n1 >= 0 && n1 <= count&&n2 >= 0 && n2 <= count)

{

p[++count] = p[n1] \* p[n2];

cout << "Результат:" << p[count] << endl;

}

else cout << "Введены неверные значения" << endl;

break;

default: cout << "Такой операции нет"; break;

}

}

else

if (count >= maxSize)

cout << "Недостаточно памяти для выполнения операции, попробуйте удалить полином" << endl;

else

cout << "Недостаточно полиномов для выполнения операций, попробуйте добавить полином" << endl;

}

if (choise == 4)

{

system("cls");

Print(p, count);

}

if (choise == 5)

{

if (count >= 0)

{

cout << "Выберите полином" << endl;

Print(p, count);

int n;

cin >> n;

if (n >= 0 && n <= count)

{

double X, Y, Z;

cout << "Введите точку в формате X Y Z " << endl;

cin >> X >> Y >> Z;

cout << "Результат: " << p[n].Calculate(X, Y, Z) << endl;

}

else cout << "Введено неверное значение" << endl;

}

else cout << "Недостаточно полиномов для выполнения операций, попробуйте добавить полином" << endl;

}

}

return 0;

}