МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования   
**«Национальный исследовательский   
Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

Направление подготовки «Прикладная математика и информатика»

**ОТЧЕТ**

по учебной практике

**Арифметические операции с полиномами**

**Выполнил:** студентка группы 381603-1

Жилина В.В.

**Проверила:** к.т.н.,доцент каф. МОСТ института ИТММ

Кустикова В.Д.

Нижний Новгород  
2018

Содержание

Введение 3

1 Постановка задачи 4

2 Руководство пользователя 5

3 Руководство программиста 6

3.1 Структура программы 6

3.2 Структуры данных 6

3.2.1 Шаблон класса циклический список с «головой» 6

3.2.2 Класс полиномов 6

3.3 Программная реализация 8

3.3.1 Класс Node 8

3.3.2 Класс Monom 8

3.3.3 Класс List 8

3.3.4 Класс Polinom 9

Заключение 10

Литература 11

Приложение 12

Приложение А. Программная реализация списка с головой 12

Приложение Б. Программная реализация монома и звена 14

Приложение В. Программная реализация полинома 15

Приложение Г. Основная программа 19

# Введение

Данная лабораторная работа посвящена разработке системы для арифметических действий над полиномами и их хранение, с использованием структуры циклического списка с головой. Реализация подобных математических операций упрощает работу со сложными вычислениями, повышает их точность. Один из методов обработки полиномов на ЭВМ представлен в данной программе. Её элементы могут быть использованы при разработке других программных комплексов и приложений.

# Постановка задачи

Цель: Разработать программу, выполняющую арифметические операции над полиномами от трех переменных (x, y, z). Полином – сумма мономов, степень каждого из которых от 0 до 9.Допустимые операции известны: сложение, вычитание, умножение двух полиномов и умножение полинома на константу. Программа должна выполнять предварительную проверку корректности выражения и вывести результат операции на экран.

Контрольный пример:

Введенное пользователем выражение(рис.1) и полученный результат выражения введенного пользователем(рис.2):

|  |  |
| --- | --- |
|  |  |
| Рис.1. Введенное выражение | Рис.2. Полученный результат |

# Руководство пользователя

Для начала работы необходимо запустить файл [sample.exe](../build/lab1_polinom/Lab1_polinom.sln). На экране появится меню(рис.3), при выборе пункта 1-4, появится надпись с просьбой ввести выражение (рис 4.), если оно введено верно появится результат. При выборе пункта 5-выход из программы.

|  |  |
| --- | --- |
|  |  |
| Рис.3. Меню программы | Рис.3. Просьба о вводе полинома и константы |

# Руководство программиста

## Структура программы

Программа состоит из 4 проектов:

1. gtest

Содержит в себе [gtest.h](../gtest/gtest.h) и [gtest-all.cpp](../gtest/gtest-all.cc). Файлы содержат google тесты, которые необходимы для проверки корректности работы реализованных классов.

1. polinom\_lib

Содержит в себе:

[Node.h](file:///C:\Users\user\Desktop\mp2-practice-2\ZhilinaL\Lab1\include\Node.h)-заголовочный файл, содержащий объявление шаблонного класса Node и его реализацию.

[Monom.h](file:///C:\Users\user\Desktop\mp2-practice-2\ZhilinaL\Lab1\include\Monom.h)- заголовочный файл, содержащий объявление шаблонного класса monom.

[Polinom.h](file:///C:\Users\user\Desktop\mp2-practice-2\ZhilinaL\Lab1\include\Polinom.h)- заголовочный файл, содержащий объявление шаблонного класса Polinom

[List.h](file:///C:\Users\user\Desktop\mp2-practice-2\ZhilinaL\Lab1\include\List.h)-заголовочный файл, содержащий объявление шаблонного класса list и его реализацию.

[Monom.cpp](file:///C:\Users\user\Desktop\mp2-practice-2\ZhilinaL\Lab1\src\Monom.cpp)-содержит реализацию класса monom

[Polinom.cpp](file:///C:\Users\user\Desktop\mp2-practice-2\ZhilinaL\Lab1\src\Polinom.cpp)-содержит реализацию класса Polinom.

1. polinom\_test

Содержит в себе [Test\_main.cpp](../test/Test_main.cpp), [test\_polinoms.cpp](../test/test_polinoms.cpp), [test\_list.cpp](../test/test_list.cpp). Реализация тестов для проверки корректности работы реализованных классов.

1. sample

Содержит [Main.cpp](file:///C:\Users\user\Desktop\mp2-practice-2\ZhilinaL\Lab1\samples\Main.cpp)– реализация пользовательского интерфейса.

## Структуры данных

### Шаблон класса циклический список с «головой»

В качестве структуры хранения полинома используется циклический список с головой. Звенья списка – мономы, однозначно определяемые через коэффициент и обобщенную степень. Голова – фиктивное звено, не содержащее монома. Элементы списка хранятся в порядке возрастания обобщённых степеней.

Для его реализации был написан шаблонный класс Node-звено списка. Сам список представляет собой структуру с двумя полями:

-указатель на голову звена

-указатель на текущее звено.

Описание алгоритмов:

Упорядоченная вставка

На вход передаем данные, для хранения в новом звене списка.

Алгоритм:

1.Устанавливаем указатель actual на голову списка.

2.Содаем указатель el, выделяем память под новый узел.

3. До тех пор пока данные следующего после actual узла меьше входных и список не кончился смещаем actual.

4.Вставляем новый узел el после найденного actual.

### Класс полиномов

Класс полиномов является полноценным решением данной лабораторной работы. С помощью него реализовано хранение полиномов и операции над ними. У данного класса одно поле:

-циклический список мономов

Описание алгоритмов:

1. Сложение полиномов

Реализован как перегрузка оператора “+”. На вход передаем два полинома. На выход возвращаем результат их сложения.

Алгоритм:

1. Создаем полином res для записи результатов сложения.

1. Сдвигаем указатели обоих списков и списка-результата на звенья, следующие за головой.

2.Начинаем цикл, пока оба полинома не закончатся выполняем следующее:

2.1.Сравниваем звенья списков (обобщенные степени).

2.2.Если звено одного полинома меньше, то вставляем в конец списка-результата и сдвигаем указатели списка, из которого взяли элемент и списка-результата на одно звено вперёд.

2.3. Если звенья равны, создаём моном, коэффициент которого равен сумме коэффициентов мономов, лежащих в этих звеньях, приводя подобные, и записываем его в конец списка-результата, сдвигаем указатели обоих списков и списка-результата на одно звено вперёд.

3.После того, как один из списков кончился, записываем остаток второго списка в список-результат.

4.Возвращаем res.

1. Разность полиномов

Реализован как перегрузка оператора “-”. Реализуется с помощью операции сложения полиномов. Вычитаемый полином умножается на (-1) и складывается и уменьшаемым полиномом.

1. Умножение на константу

Реализован как перегрузка оператора “\*”. На вход передаем полином и константу. На выход возвращаем результат умножения.

Алгоритм:

1. Проверяем равна ли константа 0.

2. Если не равна, то создаем полином-результат res, равный исходному.

3.Сдвигаем указатель res на звено, следующее за головой.

4.Пока полином res не закончится умножаем коэффициент монома, лежащего в текущем звене, на переданную константу и сдвигаем указатель на следующее звено.

5.Возвращаем res.

1. Умножение полиномов

Реализован как перегрузка оператора “\*”. На вход передаем два полинома. На выход возвращаем результат их умножения.

Алгоритм:

1. Сдвигаем указатели обоих списков на звенья, следующие за головой.

2.Пока в первом списке не дошли до конца (3-8 пункты):

3.Создаём полином-копию второго полинома temp и сдвигаем в нём (рассматривая его как список) указатель на звено, следующее за головой, пока в нем не дошли до конца.

4.Если мономы в текущих звеньях первого списка и списка-копии второго удовлетворяют условию: сумма степеней “x”, “y” или “z” одного и другого меньше 10, то складываем их обобщенные степени, перемножаем коэффициенты, результат записываем в текущее звено списка-копии второго полинома.

5.Иначе выводим сообщение об ошибке " Too big degree ".

6.Сдвигаем указатель в temp на следующее звено.

7.Записываем в res сумму самого res и temp.

8.Сдвигаем указатель в первом списке на следующее звено.

9.Возвращаем res.

## Программная реализация

### Класс Node

Поля:

T data – данные в звене

Node\* next - указатель на следующее звено

Методы:

Node() { next = nullptr; } -конструктор по умолчанию-создает пустое звено

Node(T Data) { data = Data; next = nullptr; }-конструктор с параметром-данные звена

bool operator< (const Node& z) const { return (data < z.data); }-перегрузка операции меньше, сравнивает данные звеньев

bool operator> (const Node& z) const { return (data > z.data); }-перегрузка операции больше, сравнивает данные звеньев

### Класс Monom

Поля:

double coeff-коэффициент монома

unsigned int abc - степень монома, x^a,y^b,z^c

Методы:

monom(double c = 0, unsigned int a = 0) - конструктор с параметрами

monom& operator=(const monom& m)- перегрузка оператора присваивания

bool operator< (const monom& m) const -перегрузка оператора меньше, сравнивает общую степень и коэффициент

bool operator> (const monom& m) const -перегрузка оператора больше, сравнивает общую степень и коэффициент

Перегрузка операторов сравнения:

bool operator==(const monom& m) const

bool operator!=(const monom& m) const

### Класс List

Поля:

Node<T>\* head - указатель на голову списка-фиктивное звено

Node<T>\* cur - указатель на текущий элемент

Методы:

void Clean() - очистка списка

list() - конструктор по умолчанию

list(const list<T>& a) - конструктор копирования

~list() - деструктор

list<T>& operator=(const list<T> &a) - перегрузка оператора присваивания

void Insert(T elem) - вставка элемента в упорядоченный список

void InsertToTail(T elem) - вставка в конец списка

void InsertAfter(Node<T>\* N, T Data)- вставка после определенного звена

Перегрузка операторов сравнения:

bool operator==(const list<T>& sp) const

bool operator!=(const list<T>& sp) const

Методы навигации:

void Reset() - в начало списка

void Step() - шаг на одно звено вперед

Node<T>\* GetCur() const - получить текущий элемент

bool IsNotOver() const - проверка на окончание списка

### Класс Polinom

Поля:

list<monom> Plist - список из мономов

Методы:

list<monom> similar\_terms(list <monom> sp) - приведение подобных в полиноме

Polinom(const string pol = "") - разбор строки

Polinom(const Polinom& pol) - конструктор копирования

Polinom& operator=(const Polinom &pol)- перегрузка оператора присваивания

Polinom operator+(const Polinom& pol) const - перегрузка оператора сложения

Polinom operator\*(const double a) const - перегрузка оператора умножения с параметром-константа (умножение на константу слева)

friend Polinom operator\*(const double a, const Polinom& pol) - умножение на константу справа

Polinom operator-(const Polinom& pol) const - перегрузка оператора вычитания

Polinom operator\*(const Polinom& pol) const - перегрузка оператора умножения с параметром-полином (умножение полиномов)

Polinom operator-() const -унарный минус

friend ostream& operator<<(ostream &ostr, const Polinom &pol) - перегрузка оператора вывода

Polinom(list<monom> &list) : Plist(list) {} - конструктор по списку

Перегрузка операторов сравнения:

bool operator==(const Polinom& pol) const

bool operator!=(const Polinom& pol) const

# Заключение

В лабораторной работе была разработана система для арифметических действий над полиномами. Реализовано хранение полиномов на основе циклического односвязного списка с головой. Программа, как и требовалось, поддерживает операции над полиномами: сложение, вычитание, умножение на константу и перемножение двух полиномов.

# Литература

1. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ, Москва 2013
2. Гергель В.П. Рабочие материалы к учебному курсу «Методы программирования». 2002 г.

# Приложение

## Приложение А. Программная реализация списка с головой

List.h

#pragma once

#include "Node.h"

using namespace std;

template<typename T>

class list

{

private:

Node<T>\* head;

Node<T>\* cur;

public:

void Clean(); list();

list(const list<T>& a);

~list();

list<T>& operator=(const list<T> &a);

void Insert(T elem);

void InsertToTail(T elem);

void InsertAfter(Node<T>\* N, T Data);

bool operator==(const list<T>& sp) const;

bool operator!=(const list<T>& sp) const { return !(\*this == sp); }

void Reset() { cur = head->next; }

void Step() { cur = cur->next; }

Node<T>\* GetCur() const { return cur; }

bool IsNotOver() const { return !(cur == head); }

};

template <typename T>

void list<T>::Clean()

{

Node<T>\* cur = head->next;

while (cur != head)

{

Node<T>\* temp = cur->next;

delete cur;

cur = temp;

}

head->next = head;

}

template <typename T>

list<T>::list()

{

head = new Node<T>;

head->next = head;

cur = head;

}

template <typename T>

list<T>::list(const list<T>& a)

{

head = new Node<T>;

Node<T>\* A = a.head;

Node<T>\* B = head;

if (A->next == a.head)

{

head->next = head;

return;

}

while (A->next != a.head)

{

A = A->next;

B->next = new Node<T>(A->data);

B = B->next;

}

B->next = head;

cur = head->next;

}

template <typename T>

list<T>::~list()

{

Clean();

delete head;

}

template <typename T>

list<T>& list<T>::operator=(const list<T>& a)

{

Clean();

Node<T>\* A = a.head;

Node<T>\* B = head;

while (A->next != a.head)

{

A = A->next;

B->next = new Node<T>(A->data);

B = B->next;

}

B->next = head;

cur = head;

return \*this;

}

template <typename T>

void list<T>::Insert(T elem)

{

Node<T>\* actual = head;

Node<T>\* el = new Node<T>(elem);

while ((actual->next != head) && (\*(actual->next) < \*el))

actual = actual->next;

Node<T>\* actual\_2 = actual->next;

actual->next = el;

actual->next->next = actual\_2;

}

template<typename T>

bool list<T>::operator==(const list<T>& sp) const

{

bool res = true;

if (this != &sp)

{

Node<T>\* a = head->next;

Node<T>\* b = sp.head->next;

while (a->data == b->data && a != head && b != sp.head)

{

a = a->next;

b = b->next;

}

if (a != head || b != sp.head)

res = false;

}

return res;

}

template<typename T>

void list<T> ::InsertToTail(T elem)

{

Reset();

while (cur->next != head)

Step();

Node<T>\* temp = cur->next;

cur->next = new Node<T>(elem);

cur->next->next = temp;

}

template <class T>

void list<T>::InsertAfter(Node<T>\* N, T Data)

{

Node<T>\* temp = N->next;

N->next = new Node<T>(Data);

N->next->next = temp;

}

## Приложение Б. Программная реализация монома и звена

Monom.h

#pragma once

class monom

{

public:

double coeff;

unsigned int abc;

monom(double c = 0, unsigned int a = 0);

monom& operator=(const monom& m);

bool operator< (const monom& m) const;

bool operator> (const monom& m) const;

bool operator==(const monom& m) const;

bool operator!=(const monom& m) const;

};

Monom.cpp

#include "Monom.h"

monom::monom(double c, unsigned int a)

{

coeff = c;

abc = a;

}

monom& monom:: operator=(const monom& m)

{

coeff = m.coeff;

abc = m.abc;

return \*this;

}

bool monom:: operator< (const monom& m) const

{

bool res = true;

if (abc >= m.abc)

res = false;

return res;

}

bool monom:: operator> (const monom& m) const

{

bool res = true;

if (abc <= m.abc)

res = false;

return res;

}

bool monom:: operator==(const monom& m) const

{

bool res = true;

if (abc != m.abc || coeff != m.coeff)

res = false;

return res;

}

bool monom:: operator!=(const monom& m) const

{

return !(\*this == m);

}

Node.h

#pragma once

using namespace std;

template<typename T>

class Node

{

public:

T data;

Node\* next;

Node() { next = nullptr;

Node(T Data) { data = Data; next = nullptr; }

bool operator< (const Node& z) const { return (data < z.data);

bool operator> (const Node& z) const { return (data > z.data);

};

## Приложение В. Программная реализация полинома

Polinom.h

#pragma once

#include "Monom.h"

#include "List.h"

#include <iostream>

#include <string>

using namespace std;

class Polinom

{

private:

list<monom> Plist;//список из мономов

list<monom> similar\_terms(list <monom> sp);

public:

Polinom(const string pol = "");

Polinom(const Polinom& pol);

Polinom& operator=(const Polinom &pol);

Polinom operator+(const Polinom& pol) const;

Polinom operator\*(const double a) const;

friend Polinom operator\*(const double a, const Polinom& pol) { return pol \* a;}

Polinom operator-(const Polinom& pol) const{return \*this + pol \* (-1.0); }

Polinom operator\*(const Polinom& pol) const;

Polinom operator-() const { return (-1.0)\*(\*this); }

friend ostream& operator<<(ostream &ostr, const Polinom &pol);

bool operator==(const Polinom& pol) const { return Plist == pol.Plist; }

bool operator!=(const Polinom& pol) const { return Plist != pol.Plist; }

Polinom(list<monom> &list) : Plist(list) {}

};

Polinpm.cpp

#include "Polinom.h"

using namespace std;

list<monom> Polinom::similar\_terms(list <monom> sp)

{

list<monom> res;

sp.Reset();

res.Reset();

monom tmp;

Node<monom>\* p = sp.GetCur();

sp.Step();

if (sp.IsNotOver())

{

sp.Reset();

while (sp.IsNotOver())

{

tmp.coeff = sp.GetCur()->data.coeff;

tmp.abc = sp.GetCur()->data.abc;

while ((sp.GetCur()->data.abc == sp.GetCur()->next->data.abc) && (sp.IsNotOver()))

{

tmp.coeff += sp.GetCur()->next->data.coeff;

sp.Step();

}

if (tmp.coeff)

{

res.Insert(tmp);

res.Step();

}

if (sp.GetCur())

sp.Step();

}

res.Reset();

if (res.GetCur()->data.abc == 0 && res.GetCur()->data.coeff == 0)

res.InsertAfter(res.GetCur(), NULL);

}

else

res = sp;

return res;

}

Polinom::Polinom(string pol)

{

list<monom> res;

while (pol.length())

{

string part;

monom temp;

int pos = 1;

while (pos < pol.length() && pol[pos] != '+' && pol[pos] != '-')

pos++;

part = pol.substr(0, pos); // substr

pol.erase(0, pos); // erase

pos = 0;

while (part[pos] != 'x' && part[pos] != 'y' && part[pos] != 'z' && pos < part.length())

pos++;

string c = part.substr(0, pos);

if (c == "+" || c.length() == 0)

temp.coeff = 1;

else

if (c == "-")

temp.coeff = -1;

else

temp.coeff = stod(c);

part.erase(0, pos);

part += ' ';

int a[3] = { 100,10,1 };

for (int i = 0; i < 3; i++)

{

pos = part.find((char)(120 + i));

if (pos > -1)

{

if (part[pos + 1] != '^')

part.insert(pos + 1, "^1");

temp.abc += a[i] \* stoi(part.substr(pos + 2, 1));

part.erase(pos, 3);

}

}

Plist.Insert(temp);

}

Plist = similar\_terms(Plist);

}

Polinom::Polinom(const Polinom& pol)

{

Plist = pol.Plist;

}

Polinom& Polinom:: operator=(const Polinom &pol)

{

Plist= pol.Plist;

return \*this;

}

Polinom Polinom::operator+(const Polinom& pol) const

{

Polinom res;

Polinom pthis = \*this;

Polinom p = pol;

pthis.Plist.Reset();

p.Plist.Reset();

res.Plist.Reset();

while (pthis.Plist.IsNotOver() && p.Plist.IsNotOver())

{

if (pthis.Plist.GetCur()->data > p.Plist.GetCur()->data)

{

res.Plist.InsertToTail(p.Plist.GetCur()->data);

p.Plist.Step();

res.Plist.Step();

}

else

if (pthis.Plist.GetCur()->data < p.Plist.GetCur()->data)

{

res.Plist.InsertToTail(pthis.Plist.GetCur()->data);

pthis.Plist.Step();

res.Plist.Step();

}

else

{

double new\_coeff = pthis.Plist.GetCur()->data.coeff + p.Plist.GetCur()->data.coeff;

if (new\_coeff)

{

monom temp(new\_coeff, pthis.Plist.GetCur()->data.abc);

res.Plist.InsertToTail(temp);

res.Plist.Step();

}

pthis.Plist.Step();

p.Plist.Step();

}

}

while (pthis.Plist.IsNotOver())

{

res.Plist.InsertToTail(pthis.Plist.GetCur()->data);

pthis.Plist.Step();

res.Plist.Step();

}

while (p.Plist.IsNotOver())

{

res.Plist.InsertToTail(p.Plist.GetCur()->data);

p.Plist.Step();

res.Plist.Step();

}

return res;

}

Polinom Polinom::operator\*(const double a) const

{

Polinom res;

if (a)

{

res = \*this;

res.Plist.Reset();

while (res.Plist.IsNotOver())

{

res.Plist.GetCur()->data.coeff \*= a;

res.Plist.Step();

}

}

return res;

}

Polinom Polinom::operator\*(const Polinom& pol) const

{

Polinom res;

Polinom pthis = \*this;

Polinom p = pol;

pthis.Plist.Reset();

p.Plist.Reset();

while (pthis.Plist.IsNotOver())

{

double pthis\_coeff = pthis.Plist.GetCur()->data.coeff;

int pthis\_abc = pthis.Plist.GetCur()->data.abc;

Polinom temp(pol);

temp.Plist.Reset();

while (temp.Plist.IsNotOver())

{

int temp\_abc = temp.Plist.GetCur()->data.abc;

if ((temp\_abc % 10 + pthis\_abc % 10) < 10 && (temp\_abc / 10 % 10 + pthis\_abc / 10 % 10) < 10 && (temp\_abc / 100 + pthis\_abc / 100) < 10)

{

temp.Plist.GetCur()->data.abc += pthis\_abc;

temp.Plist.GetCur()->data.coeff \*= pthis\_coeff;

}

else

throw "Too big degree";

temp.Plist.Step();

}

res = res + temp;

pthis.Plist.Step();

}

return res;

}

ostream& operator<<(ostream &ostr, const Polinom& pol)

{

Polinom p = pol;

p.Plist.Reset();

Polinom zero("");

if (pol == zero)

ostr << "0";

while (p.Plist.IsNotOver())

{

monom temp = p.Plist.GetCur()->data;

if (temp.coeff > 0)

{

ostr << "+";

if (temp.coeff == 1 && temp.abc == 0)

ostr << "1";

else

if (temp.coeff != 1)

ostr << temp.coeff;

}

else

ostr << temp.coeff;

int a = temp.abc / 100;

if (a>1)

ostr << "x^" << a;

else

if (a == 1)

ostr << "x";

a = temp.abc / 10 % 10;

if (a>1)

ostr << "y^" << a;

else

if (a == 1)

ostr << "y";

a = temp.abc % 10;

if (a>1)

ostr << "z^" << a;

else

if (a == 1)

ostr << "z";

p.Plist.Step();

}

return ostr;

}

## Приложение Г. Основная программа

int main()

{

char number;

string s1, s2;

Polinom p1, p2;

double con;

cout << "Hello, what do you wanna do?"<<endl;

cout << "1.Add 2 polinoms" << endl;

cout << "2.Subtract 2 polinoms" << endl;

cout << "3.Multiply 2 polinoms" << endl;

cout << "4.Multiply polinom by number" << endl;

cout << "5.Exit" << endl;

do

{

cout << "Do:" << endl;

cin >> number;

switch (number)

{

case '1':

{

cout << "Enter first polinom: ";

cin >> s1;

p1 = s1;

cout << "Enter second polinom: ";

cin >> s2;

p2 = s2;

cout << "Result of operation: " << p1 + p2 << endl << endl;

break;

}

case '2':

{

cout << "Enter first polinom: ";

cin >> s1;

p1 = s1;

cout << "Enter second polinom: ";

cin >> s2;

p2 = s2;

cout << "Result of operation: " << p1 - p2 << endl << endl;

break;

}

case '3':

{

cout << "Enter first polinom: ";

cin >> s1;

p1 = s1;

cout << "Enter second polinom: ";

cin >> s2;

p2 = s2;

cout << "Result of operation: " << p1 \* p2 << endl << endl;

break;

}

case '4':

{

cout << "Enter polinom: ";

cin >> s1;

p1 = s1;

cout << "Enter constant: ";

cin >> con;

cout << "Result of operation: " << con \* p1 << endl << endl;

break;

}

}

}

while (number != '5');

return 0;

}