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

Отчет

по учебной практике

**Арифметические операции с полиномами.**

Выполнила:

студентка гр. 381703-1

Мочалова А.Р.

Проверил:

Ассистент кафедры МОСТ ИИТММ

Волокитин В.Д.

г. Нижний Новгород

2018г

Оглавление

[Введение 3](#_Toc515272243)

[Постановка учебно-практической задачи 4](#_Toc515272244)

[Руководство пользователя 5](#_Toc515272245)

[Руководство программиста 7](#_Toc515272246)

[Заключение 10](#_Toc515272247)

[Приложения 11](#_Toc515272249)

# Введение.

В математической практике везде используются многочлены – полиномы. Их можно увидеть в учебниках по математике 1 класса или в справочнике по математическому анализу для студентов, в различных физических сборниках и даже в экономических практиках и учебниках биологии. С этими многочленами постоянно требуется выполнять преобразования, самые элементарные из них, такие как сложение полиномов, их разность, произведение и умножение на константу мы и реализуем в этой лабораторной работе.

# Постановка учебно-практической задачи

**Цель работы:**

Разработать программу, выполняющую арифметические операции с полиномами трех переменных (x, y и z): сложение, вычитание, умножение на константу, умножение двух полиномов. Считается, что полином составлен из мономов от трех переменных с ограничением на степень каждой переменой от 0 до 99. Коэффициенты полинома - вещественные числа. Кроме того, необходимо разработать пользовательское консольное приложение.

**Особенности реализации:**

* В качестве структуры хранения полинома использовать список мономов с ненулевыми коэффициентами (односвязный или двухсвязный). Односвязный список удобнее реализовать с фиктивной головой. Элементы списка хранить упорядоченными.
* Степень полинома хранить в "свернутом" виде, т. е. степень должна быть представлена как трехзначное число, где число сотен – это степень при переменной “x”, число десятков - степень при переменной “y”, число единиц - степень при переменной “z”.
* Сложение полиномов осуществлять алгоритмом слияния упорядоченных массивов.
* Вычитание полиномов допускается выполнять через сложение с умножением на константу (C = A - B = A + (-1)\*B)
* При умножении и сложение (вычитание) необходимо следить, чтобы в итоговом полиноме были приведены подобные слагаемые и не хранилось мономов с нулевым коэффициентом.
* Если при умножении полиномов полученные степени переменных больше 9, выводить сообщение об ошибке.
* Считывание полинома у пользователя допускается в любом удобном для вас виде. Необходимо предоставить пользователю правила ввода данных
* Следует учесть, что пользователь может вводить полином, не упорядочив в нем мономы.

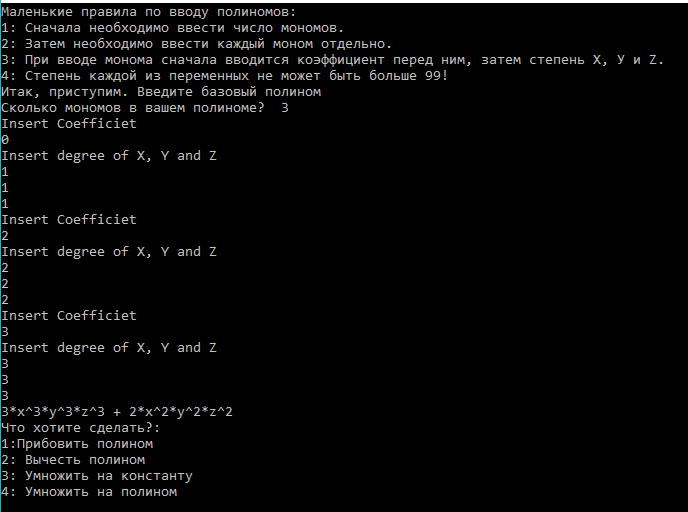
# Руководство пользователя

После запуска программы на экране появится диалоговое окно. В нем программа ознакомляет пользователя с правилом введения данных, а именно:

* Сначала необходимо ввести количество мономов в полиноме
* Затем необходимо ввести каждый моном отдельно
* Вводится коэффициент перед мономом, а затем степени Х, У и Z соответственно
* Степень каждой из переменных не должна быть больше 99

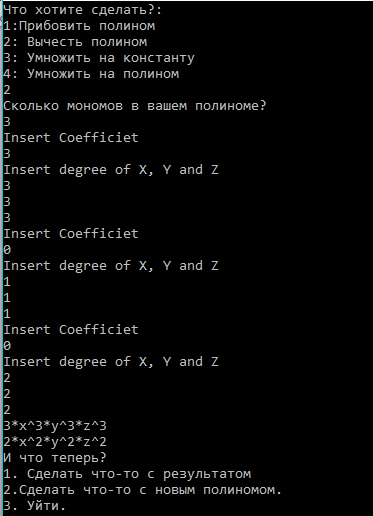
Затем пользователю предлагается ввести данные. При вводе некорректных данных программа закрывается и выдает исключение с соответствующим названием (название содержит информацию о виде ошибки).

После ввода выражения на экране появится введенный полином. Программа выводит полином, в котором мономы расположены порядке убывания степеней, вне зависимости оттого, в каком порядке их ввел пользователь. Также программа не учитывает мономы с нулевыми коэффициентами при вводе выражения.



Затем пользователю будет предложена возможность выбрать операцию. При выборе 1,2 и 4 пункта у пользователя запрашивается новое выражение. При выборе 3 пункта у пользователя запрашивается константа. При вводе иного числа программа выходит в финальное меню, где пользователь может покинуть программу или начать работу с выражением заново.

После выполнения выбранной операции программа выведет на экран результат, учитывая все коэффициенты, которые в ходе вычислений стали равны нулю, и возможное изменение порядка полинома. Если в ходе выполнения умножения полиномов, степень хотя бы одного монома стала выше 99, то программа выдает исключение с соответствующим названием. Затем пользователь может сделать что-то с результатом, начать работать с новыми данными или закончить работу, выбрав пункт 1, 2 или 3 соответственно. При вводе иных чисел программа автоматически прощается с пользователем и закрывается.



# Руководство программиста

Решение поставленной задачи осуществляется с помощью следующих элементов:

1. Структура Node

В данной структуре элементы списка. В ней есть следующие поля:

* data – хранит в себе значение элемента шаблонного типа
* next – хранит в себе адрес следующего элемента.

2.Шаблонный класс List

List – класс, который хранит в себе адрес начала и конца списка. Элементами этого списка являются элементы типа Node, которые связанны между собой, а именно каждый элемент хранит адрес следующего элемента. Все операции с элементами списка основываются на операциях с адресами элементов.

3.Структура Mon

Данная структура отвечает за хранение мономов, из которых состоит полином. Содержит в себе следующие поля:

* Value – хранит в себе коэффициент перед мономом
* degree – хранит в себе степень полинома в виде XXYYZZ

Для данной структуры также реализованы перегрузки следующих операций:

* >(<) – большим мономом считается тот, чья степень выше.
* == - мономы считаются равными, если равны их степени и коэффициенты перед ними.
* != - мономы не равны, если не выполняется хотя бы одно из условий предыдущего пункта.
* \* - умножение мономов выполняется через умножение коэффициентов и сложение степеней. Также проводится проверка на соответствие полученной степени допустимому значению.
* + - сложение выполняется только для мономов, равных степеней и реализуется путем складывания коэффициентов.

3. Класс Polinoms

Данный класс содержит себе список мономов и в нем реализуются следующие операции:

* Добавление монома в полином: данная операция вставляет элементы с учетом их порядка, т.е. сразу формирует отсортированный массив.
* Считывание данных пользователя: данная операция создает моном из данных, введенных пользователем, при этом проводя проверку корректности введенных данных, при необходимости останавливая программу.
* Перегрузка операции +: сложение двух полиномов выполняется через их слияние, а именно создаются два указателя, которые одновременно перемещаются по кадому из слогаемых полиномов, считывая минимальный элемент из двух. В случае равенства степеней элементов в результат записывается их сумма. После того, как один из указателей дойдет до конца соответствующего ему списка, если в другом полиноме еще остались мономы, они переносятся в результат.
* Перегрузка оператора -: вычитание полиномов производится через сложение с умножением на константу (C = A - B = A + (-1)\*B)
* Перегрузка операции \*(для 2-х полиномов): производится последовательное перемножение элементов, заем полученный результат проверяют на наличие подобных слагаемых, и , если такие находятся, то они складываются.
* Перегрузка операции \*(для константы): производится последовательное умножение каждого из мономов на константу.

# Заключение

В данной лабораторной работе были изучены следующие аспекты:

1. Концепция односвязного списка и его реализация.
2. Разработка класса полиномов и его реализация.
3. Изучения принципа сложения массивов слиянием.
4. Реализован диалог с пользователем.

В ходе работы была разработана среда для работы с полиномами и вычисления простых операций с ними, которая учитывает корректность введенного выражения и полученного результата.

Приложения

Файл list.h:

Приложение 1. Разработка структуры Node, класса List и его методов

struct Node

{

T data;

Node<T> \*next;

};

template <class T>

class List {

public:

Node<T> \*head;

Node<T> \*tail;

List();

void insert\_front(T x);

void insert\_at\_end (T x);

void insert\_after(Node<T> \*x, T y);

void del (T x);

bool operator == (const List<T> &a) const;

};

template <class T>

void List<T>::insert\_front(T x)

{

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

a->data = x;

if (head == nullptr)

{

a->next = nullptr;

tail = a;

}

else

{

a->next = head;

}

head = a;

}

template <class T>

void List<T>::insert\_at\_end(T x)

{

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

Node<T> \*tmp = head;

a->data = x;

if (head == nullptr)

{

a->next = nullptr;

head = a;

}

else

{

while (tmp->next != nullptr)

{

tmp = tmp->next;

}

a->next = nullptr;

tmp->next = a;

}

tail = a;

}

template <class T>

void List<T>::insert\_after(Node<T>\* x, T y)

{

Node<T> \*tmp = head;

while (tmp != nullptr)

{

if (tmp->data == x->data)

{

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

a->data = y;

a->next = tmp->next;

tmp->next= a;

return;

}

tmp = tmp -> next;

}

}

template <class T>

void List<T>::del(T x)

{

if (head == nullptr)

{

return;

}

Node<T> \*a = head;

while (a->next != nullptr)

{

if (head->data == x)

{

head = head->next;

}

else if (a->next->data == x)

{

a->next = a->next->next;

if (a -> next == nullptr)

break;

}

a = a->next;

}

if (a->data == x)

del(x);

}

template <class T>

bool List<T>::operator == (const List<T> &a) const

{

Node <T> \*x = head, \*y = a.head;

while( (x != nullptr) && (y != nullptr))

{

if (x->data == y->data)

{

x = x->next;

y = y->next;

}

else return 0;

}

if ((x != nullptr) || (y != nullptr))

return 0;

else return 1;

}

Файл polinoms.h:

Приложение 2: Структура Mon и ее перегрузки

struct Mon

{

double Value;

int degree;

};

bool operator > (Mon a, Mon b)

{

if (a.degree > b.degree)

return 1;

else return 0;

};

bool operator < (Mon a, Mon b)

{

if (a > b)

return 0;

else return 1;

};

bool operator == (Mon a, Mon b)

{

if ((b.degree == a.degree) && (b.Value == a.Value))

return 1;

else return 0;

};

bool operator != (Mon a, Mon b)

{

if ((a.degree != b.degree) && (a.Value != b.Value))

return 1;

else return 0;

};

Mon operator \* (Mon a, Mon b)

{

Mon res;

res.Value = a.Value \* b.Value;

res.degree = a.degree + b.degree;

int d1,d2;

d1 = a.degree;

d2 = b.degree;

int t = d1%10+d2%10;

if (t < 100)

{

d1=d1/100;

d2=d2/100;

t = d1%100+d2%100;

if (t < 100)

{

d1=d1/100;

d2=d2/100;

t = d1+d2;

if(t < 100)

{} else throw "Too Large degree Y in multiply";

}

else throw "Too Large degree X in multiply";

}

else throw "Too Large degree Z in multiply";

return res;

};

Mon operator + (Mon a, Mon b)

{

Mon res;

if (a.degree == b.degree)

{

res.degree = a.degree;

res.Value = a.Value + b.Value;

}

else throw "its not monoms";

return res;

}

Приложение 3: Класс Polinoms и методы установки монома в полином

class Polinoms

{

public:

List <Mon> Polinom;

void Insert (Mon tmp)

{

if (tmp.Value != 0)

{

Node <Mon> \*a = Polinom.head;

if (a == nullptr)

Polinom.insert\_front(tmp);

else

{

if ( a == Polinom.head)

{

if (a->data.degree < tmp.degree)

Polinom.insert\_front(tmp);

else

if (a->data.degree == tmp.degree)

{

a->data.Value +=tmp.Value;

}

else goto e;

}

else

e: while (a != nullptr)

{

if (a->next == nullptr)

{

Polinom.insert\_at\_end(tmp);

break;

}

else if ( a->next->data.degree < tmp.degree)

{

Polinom.insert\_after(a, tmp);

break;

}

else

if (a->next->data.degree == tmp.degree)

{

a->next->data.Value +=tmp.Value;

break;

}

else a = a->next;

}

}

}

};

void Set(double k, double x , double y, double z)

{

Mon n;

n.Value = k;

if ( x > 99)

throw "Too large degree of X!";

else if ( (x != (int)x) || ( x < 0))

throw "Wrong degree X";

else

n.degree = (int)x\*10000;

if ( y > 99)

throw "Too large degree of Y!";

else if ( (y != (int)y) || ( y < 0))

throw "Wrong degree Y";

else

n.degree += (int)y\*100;

if ( z > 99)

throw "Too large degree of Z!";

else if ( (z != (int)z) || ( z < 0))

throw "Wrong degree Z";

else

n.degree +=(int)z;

Insert(n);

};

}

Приложение 4: Перегрузки операций для класса Polinoms

bool operator == (const Polinoms a) const

{

if (Polinom == a.Polinom)

return 1;

else return 0;

}

Polinoms operator + (Polinoms a)

{

Polinoms res;

Node <Mon> \*x = Polinom.head;

Node <Mon> \*y = a.Polinom.head;

while((x != nullptr) && (y != nullptr))

{

if(x->data > y->data)

{

res.Polinom.insert\_at\_end (x->data);

x = x->next;

}

else if(y->data > x->data)

{

res.Polinom.insert\_at\_end (y->data);

y = y->next;

}

else

{

res.Polinom.insert\_at\_end(y->data + x->data);

x = x->next;

y = y->next;

}

}

if((x == nullptr) && (y != nullptr))

{

while (y != nullptr)

{

res.Polinom.insert\_at\_end(y->data);

y = y->next;

}

}

else if((y == nullptr) && (x != nullptr))

{

while (x != nullptr)

{

res.Polinom.insert\_at\_end(x->data);

x = x->next;

}

}

x= res.Polinom.head;

while (x != nullptr)

{

if (x->data.Value == 0)

{

res.Polinom.del(x->data);

}

x=x->next;

}

return res;

};

Polinoms operator - (Polinoms a)

{

return (\*this + a\*(-1));

};

Polinoms operator \* (Polinoms a)

{

Polinoms res;

Node <Mon> \*p1 = Polinom.head;

Node <Mon> \*p2;

while (p1!= nullptr)

{

p2 = a.Polinom.head;

while(p2!= nullptr)

{

res.Insert (p1->data \* p2->data);

p2 = p2->next;

}

p1 = p1->next;

}

return res;

};

Polinoms operator \* (double a)

{

Polinoms res;

Node <Mon> \*p;

p = Polinom.head;

if (a == 0)

Polinom.head = nullptr;

else

while ( p!= nullptr)

{

p->data.Value = p->data.Value \*a;

res.Insert(p->data);

p=p->next;

}

return res;

};