Отчёт по лабораторной работе

**Арифметические операции с полиномами**

**Выполнила:**

студент ИИТММ гр. 382003-1

Полякова А.В.

Нижний Новгород

2022 г.

Содержание

[Введение 3](#_Toc532661930)

[Постановка задачи 4](#_Toc532661931)

[Руководство пользователя 5](#_Toc532661932)

[Руководство программиста 6](#_Toc532661933)

[Описание структуры программы 6](#_Toc532661934)

[Описание структур данных 6](#_Toc532661935)

[Описание алгоритмов 7](#_Toc532661936)

[Заключение 9](#_Toc532661937)

[Приложения](#_Toc532661939) 10

[Приложение 1 (Class Node) 1](#_Toc532661940)0

[Приложение 2 (Class List) 1](#_Toc532661941)0

[Приложение 3 (Class Monomial) 1](#_Toc532661942)1

[Приложение 4 (Class Polinomial) 1](#_Toc532661943)2

[Приложение 5 (Приведение подобных слагаемых) 1](#_Toc532661944)3

[Приложение 6 (Сложение многочленов) 1](#_Toc532661945)4

[Приложение 7 (Умножение многочленов)](#_Toc532661946) [1](#_Toc532661946)5

[Приложение 8 (Умножение многочлена на число) 1](#_Toc532661945)5

Введение

Во многих разделах математики и её приложениях возникает необходимость работать с многочленами. Это объекты, являющиеся суммой одночленов – произведений коэффициента и переменных с целыми неотрицательными степенями. С ними удобно выполнять арифметические операции (сложение, вычитание, умножение), их легко вычислять, в отличие, например, от показательной, логарифмической или тригонометрических функций. Это объясняет, почему они часто применяются в решении различных вычислительных задач.

Существует множество способов представлять многочлены. Использование конкретного варианта зависит от рода решаемой задачи. В работе приводится реализация класса многочленов с использованием списка, удобная для выполнения арифметических операций.

Постановка задачи

Необходимо 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 9 (опционально можно расширить данное ограничение). Коэффициенты полинома - вещественные числа.

Руководство пользователя

* Пользователь вводит многочлен по следующим правилам:

1. Коэффициенты и степени мономов вводятся последовательно и разделены знаком «;». Например, <коэфф1>;<степень1>;<коэфф2>;<степень2>. Часть <коэфф1>;<степень1> соответствует первому моному, <коэфф2>;<степень2> - второму моному.
2. Коэффициент является вещественным числом, его формат должен быть таким же, как у вещественных чисел в C++.
3. Степень монома вводится как 3 цифры: первая задаёт степень , вторая – степень , третья – степень .

Пустой ввод означает, что многочлен нулевой.

* Далее предлагается выбрать операцию: сложение, вычитание, умножение на многочлен или умножение на число.
* При выборе одной из первых трёх операций потребуется ввести ещё один многочлен, при выборе умножения на число – это самое число.
* После ввода необходимых данных на экран выведется результат операции.
* Далее пользователь может продолжить работу или выйти, введя «exit».

Руководство программиста

## Описание структуры программы

**Библиотеки, использующиеся в программе:**

1. iostream (Для ввода и вывода информации)
2. vector (Для временного хранения слагаемых многочлена)
3. sstream (Для удобной работы со строками)

**Алгоритм программы реализован следующим образом:**

1. **Пользователь вводит многочлен по правилам, описанным при запуске программы.**
2. **Многочлен проверяется соответствие формату ввода, при наличии ошибок или многочлен приравнивается 0, или выводится сообщение об ошибке.**
3. **Строка преобразуется в вектор коэффициентов и степеней, по ним строится список мономов.**
4. **Пользователь выбирает операцию. В зависимости от операции также нужно будет ввести другой многочлен (при сложении, вычитании или умножении) или число (при умножении на константу). Многочлены создаются так же, как в шаге 3.**
5. **Операция выполняется, результат выводится в консоль.**
6. **Пользователь снова может ввести многочлен (возврат к шагу 1) или выйти, введя «exit».**

***Описание структур данных***

В программе используются классы Node – узел списка (приложение 1), List – односвязный список (приложение 2), Monomial – моном (приложение 3), Polynomial – полином (приложение 4).

## Описание алгоритмов

**Приведение подобных слагаемых (приложение 5):**

**Сортируем слагаемые в списке и, если он не пуст, проходимся по нему, начиная с головы, пока существует следующее слагаемое:**

1. **Сравниваем степени текущего и следующего слагаемых.**
2. **Если они совпадают, присваиваем текущему слагаемому сумму коэффициентов слагаемых и удаляем следующее слагаемое.**
3. **Переходим на следующее слагаемое.**

**Далее вновь проходимся по списку, чтобы удалить нулевые слагаемые за исключением случая, когда единственное слагаемое в списке – нулевое:**

1. **Устанавливаем текущий указатель на голову списка.**
2. **Если текущее слагаемое имеет коэффициент 0, удаляем его.**
3. **Переходим на следующее слагаемое.**

**Операция сложения многочленов (приложение 6):**

**В цикле, пока не достигнут конец одного из списков:**

1. **Если степень у монома первого многочлена больше, чем у второго, он добавляется в новый список, происходит переход к следующему слагаемому в первом многочлене.**
2. **Если степень у монома второго многочлена больше, чем у первого, он добавляется в новый список, происходит переход к следующему слагаемому во втором многочлене.**
3. **Если степени равны, добавляется слагаемое, равное сумме коэффициентов обоих слагаемых, в обоих многочленах происходит переход к следующему слагаемому.**

**Далее происходит добавление всех оставшихся слагаемых из списка, конец которого не был достигнут, и приведение подобных слагаемых.**

**Операция умножения многочленов (приложение 7):**

1. **Обходим первый список: на каждой итерации обходим второй список и добавляем в новый список моном, являющийся произведением текущих мономов.**
2. **Выполняем приведение подобных слагаемых.**

**Операция умножения многочлена на число (приложение 8):**

**Обходим список и умножаем коэффициенты всех слагаемых на число, если число не равно 0, иначе возвращаем нулевой многочлен.**

Заключение

В рамках данной лабораторной работы были реализованы структура данных список и класс многочленов, использующий список как структуру хранения мономов. Были реализованы операции над многочленами, основывающиеся на операциях над списком. Разработано пользовательское приложение для выполнения операций над многочленами. Корректность работы классов протестирована с помощью библиотеки Google Tests.

Приложения

## Приложение 1 (Class Node)

template<typename T>

struct Node

{

T data;

Node \*prev, \*next;

};

## Приложение 2 (Class List)

template<typename T>

class List

{

Node<T> \*head, \*tail; // Начальный и конечный элементы

public:

// Конструктор класса

List()

{

head = tail = nullptr;

}

// Получение начального элемента

Node<T>\* getHead()

{

return head;

}

// Получение конечного элемента

Node<T>\* getTail()

{

return tail;

}

// Добавление элемента item в список

void push\_back(const T& item)

{

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

tmp->next = nullptr;

tmp->data = item;

if (head != nullptr)

{

tmp->prev = tail;

tail->next = tmp;

tail = tmp;

}

else

{

tmp->prev = nullptr;

head = tail = tmp;

}

}

void clear()

{

head = tail = nullptr;

}

// Сортировка списка

void sort()

{

Node<T> \*i = head;

if (i != nullptr)

while (i->next != nullptr)

{

Node<T> \*j = i->next;

while (j != nullptr)

{

if (i->data < j->data)

{

T tmp = i->data;

i->data = j->data;

j->data = tmp;

}

j = j->next;

}

i = i->next;

}

}

};

## Приложение 3 (Class Monomial)

class Monomial

{

double coeff; // Коэффициент

int pow; // Показатели x, y, z

public:

// Конструктор по умолчанию

Monomial();

// Конструктор с параметрами

Monomial(double coeff, int pow);

// Получение коэффициента

double getCoeff() const;

// Получение степени

int getPow() const;

// Изменение коэффициента

void setCoeff(double coeff);

// Изменение степени

void setPow(int pow);

// Перегрузка операторов сравнения <, >, =

friend bool operator<(const Monomial& m1, const Monomial& m2);

friend bool operator>(const Monomial& m1, const Monomial& m2);

friend bool operator==(const Monomial& m1, const Monomial& m2);

// Перегрузка оператора потокового вывода

friend std::ostream& operator<<(std::ostream& os, const Monomial& m);

};

## Приложение 4 (класс Polinomial)

class Polynomial

{

// Список одночленов

List<Monomial>\* monomials;

// Упорядочивание многочлена

void arrange();

// Приведение подобных слагаемых

void combineLikeTerms();

public:

// Конструктор по умолчанию

Polynomial();

// Конструктор (параметр - список мономов)

Polynomial(const List<Monomial>& monomials);

// Конструктор (параметр - строка)

Polynomial(std::string s);

// Получение списка мономов

List<Monomial>\* getMonomials() const;

// Добавление одночлена

void addMonomial(const Monomial& m);

// Перегрузка арифметических операторов

friend Polynomial operator+(const Polynomial& p1, const Polynomial& p2);

friend Polynomial operator-(const Polynomial& p1, const Polynomial& p2);

friend Polynomial operator\*(const Polynomial& p1, const Polynomial& p2);

friend Polynomial operator\*(const Polynomial& p, double c);

friend Polynomial operator\*(double c, const Polynomial& p);

// Перегрузка оператора потокового вывода

friend std::ostream& operator<<(std::ostream& os, const Polynomial& p);

// Текстовое представление многочлена

std::string to\_string();

};

## Приложение 5 (Приведение подобных слагаемых)

void Polynomial::combineLikeTerms()

{

arrange(); // Упорядочивание

Node<Monomial> \*a = monomials->getHead(); // Получение головы

if (a != nullptr) // Если она не пуста

while (a->next != nullptr) // Пока не пуст следующий

{

Node<Monomial> \*b = a->next;

if (a->data.getPow() == b->data.getPow()) // Если степени соседей равны

{

// Коэффициент 1го - сумма коэффициентов

a->data.setCoeff(a->data.getCoeff() + b->data.getCoeff());

// Удаление 2го (с проверкой, не является ли концом)

if (b == this->getMonomials()->getTail())

{

b = b->prev;

b->next = nullptr;

}

else

{

b->prev->next = b->next;

b->next->prev = b->prev;

}

}

a = b;

}

a = monomials->getHead();

// Удаление возникших нулей (если есть только ноль, то не удаляется)

while (a != nullptr)

{

if (abs(a->data.getCoeff()) < EPS)

{

// Если голова

if (a == this->getMonomials()->getHead())

{

if (a->next == nullptr)

break;

a = a->next;

a->prev = nullptr;

}

// Если хвост

else if (a == this->getMonomials()->getTail())

{

a = a->prev;

a->next = nullptr;

}

// Иначе

else

{

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

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

}

}

a = a->next;

}

}

## Приложение 6 (Сложение многочленов)

Polynomial operator+(const Polynomial& p1, const Polynomial& p2)

{

// Получение первых элементов многочленов

Node<Monomial> \*a = p1.getMonomials()->getHead(),

\*b = p2.getMonomials()->getHead();

Polynomial c;

// Пока оба не закончились

while (a != nullptr && b != nullptr)

{

if (a->data > b->data) // Если степень 1го монома больше

{

c.addMonomial(a->data); // Запись 1го монома

a = a->next; // Движение по 1му многочлену

}

else if (a->data < b->data) // Если степень 2го монома больше

{

c.addMonomial(b->data); // Запись 2го монома

b = b->next; // Движение по 2му многочлену

}

else // Если равны

{

double cCoeff = a->data.getCoeff() + b->data.getCoeff();

if (cCoeff != 0) // Если новый коэффициент не ноль

c.addMonomial(Monomial(cCoeff, a->data.getPow())); // Добавляем моном

a = a->next; // Движение по обоим многочленам

b = b->next;

}

}

while (a != nullptr) // Добавление непрочитанных мономов из 1го многочлена

{

c.addMonomial(a->data);

a = a->next;

}

while (b != nullptr) // Добавление непрочитанных мономов из 2го многочлена

{

c.addMonomial(b->data);

b = b->next;

}

c.combineLikeTerms(); // Приведение подобных

return c;

}

## Приложение 7 (Умножение многочленов)

Polynomial operator\*(const Polynomial& p1, const Polynomial& p2)

{

Node<Monomial> \*a = p1.getMonomials()->getHead();

Polynomial c;

while (a != nullptr) // Пока не прочитан 1й многочлен

{

Node<Monomial> \*b = p2.getMonomials()->getHead();

while (b != nullptr) // Пока не прочитан 1й многочлен

{

// Если переполнение, то исключение

if (!isMultiplicative(a->data.getPow(), b->data.getPow()))

throw std::overflow\_error("Произошло переполнение!");

// Перемножение коэффицентов и суммирование показателей

c.addMonomial(Monomial(a->data.getCoeff() \* b->data.getCoeff(),

a->data.getPow() + b->data.getPow()));

b = b->next;

}

a = a->next;

}

c.combineLikeTerms();

return c;

}

## Приложение 8 (Умножение многочлена на число)

Polynomial operator\*(const Polynomial& p, double c)

{

Node<Monomial> \*a = p.getMonomials()->getHead();

Polynomial b;

if (c != 0)

{

while (a != nullptr) // Пока не прочитан многочлен

{

b.addMonomial(Monomial(a->data.getCoeff() \* c,

a->data.getPow())); // Умножение коэффициента

a = a->next;

}

b.combineLikeTerms(); // Приведение подобных

return b;

}

else

return Polynomial();

}