```
#include <iostream>
#include <cstdarg>
#include <cassert>

#include "polynomial.h"

Polynomial::Polynomial(int order) {
        this->order = order;
        this->modulus = 1;
        this->coefficients = new int[order+1];
        for (int i = 0; i <= order; i++)
                this->coefficients[i] = 0;
}

Polynomial::Polynomial(int order, int modulus, ...) {
        va_list ap;
        this->order = order;
        this->modulus = modulus;
        this->coefficients = new int[order+1];
        va_start(ap, modulus);
        for (int i = 0; i <= order; i++) {
                int c = va_arg(ap, int);
                this->coefficients[order-i] = ((c%modulus)+modulus)%modulus;
        }
        va_end(ap);
}

Polynomial::Polynomial(const Polynomial& p) {
        this->order = p.order;
        this->modulus = p.modulus;
        this->coefficients = new int[p.order+1];
        for (int i = 0; i <= order; i++)
                this->coefficients[i] = p.coefficients[i];
}

Polynomial::~Polynomial() {
        delete coefficients;
}

int Polynomial::highestOrder() const {
        int max = -1;
        for (int i = 0; i <= this->order; i++)
                if (this->coefficients[i]) max = i;
        return max;
}

const Polynomial& Polynomial::operator += (const Polynomial& r) {
        /* Can't add polynomials under different moduli */
        assert(this->modulus == r.modulus);

        int order = this->order;
        Polynomial *n;
        if (r.order > order) {
                order = r.order;
                n = new Polynomial(order);
        } else
                n = this;

        n->modulus = this->modulus;
        int m = n->modulus;
        for (int i = 0; i <= order; i++) {
                if (i > this->order)
                        n->coefficients[i] = r.coefficients[i];
                else if (i > r.order)
                        n->coefficients[i] = this->coefficients[i];
```



```
                else
                        n->coefficients[i] = this->coefficients[i]+r.coefficient
s[i];
                n->coefficients[i] %= m;
                n->coefficients[i] += m;
                n->coefficients[i] %= m;
        }
        if (n != this) {
                this->order = n->order;
                delete this->coefficients;
                this->coefficients = n->coefficients;
        }
        return *this;
}

const Polynomial& Polynomial::operator *= (const Polynomial& r) {
        /* Can't multiply polynomials under different moduli */
        assert(this->modulus == r.modulus);

        int order = this->order+r.order;

        Polynomial *n = new Polynomial(order);
        n->modulus = this->modulus;
        int m = n->modulus;
        for (int i = 0; i <= this->order; i++) {
                for (int j = 0; j <= r.order; j++) {
                        n->coefficients[i+j] += this->coefficients[i]
                                                * r.coefficients[j];
                        n->coefficients[i+j] %= m;
                        n->coefficients[i+j] += m;
                        n->coefficients[i+j] %= m;
                }
        }
        this->order = n->order;
        delete this->coefficients;
        this->coefficients = n->coefficients;
        return *this;
}

const Polynomial& Polynomial::operator *= (int x) {
        for (int i = 0; i <= this->order; i++) {
                this->coefficients[i] = this->coefficients[i]*x;
                this->coefficients[i] %= this->modulus;
                this->coefficients[i] += this->modulus;
                this->coefficients[i] %= this->modulus;
        }
        return *this;
}

const Polynomial& Polynomial::operator ^= (int pow) {
        if (pow == 0) {
                this->order = 0;
                delete this->coefficients;
                this->coefficients = new int[1];
                this->coefficients[0] = 1;
                return *this;
        }
        Polynomial p(*this);
        for (int i = 1; i < pow; i++) {
                *this *= p;
        }
        return *this;
}

std::ostream& operator << (std::ostream &os, const Polynomial& p) {
```



```
        char x = 'x';
        bool first = true;
        for (int i = p.order; i >= 0; i--) {
                if (p.coefficients[i] == 0) continue;
                if (!first) {
                        os << "+";
                } else {
                        first = false;
                }
                if (p.coefficients[i] > 1 || i == 0) {
                        os << p.coefficients[i];
                }
                if (i != 0) os << x;
                if (i > 1) os << '^' << i;
        }
        if (first) os << "0";
        return os;
}

bool operator == (const Polynomial& l, const Polynomial& r) {
    if (l.modulus != r.modulus) return false;
        int order = l.order;
        if (r.order > order) order = r.order;
        for (int i = 0; i <= order; i++) {
                if (i > l.order)
        { if (r.coefficients[i]) return false; }
                else if (i > r.order)
        { if (l.coefficients[i]) return false; }
                else
            if (r.coefficients[i] != l.coefficients[i]) return false;
    }
    return true;
}

Polynomial& operator + (const Polynomial& l, const Polynomial& r) {
        int order = l.order;
        if (r.order > order) order = r.order;

        /* Can't add polynomials under different moduli */
        assert(l.modulus == r.modulus);

        Polynomial *n = new Polynomial(order);
        n->modulus = l.modulus;
        int m = n->modulus;
        for (int i = 0; i <= order; i++) {
                if (i > l.order)
                        n->coefficients[i] = r.coefficients[i];
                else if (i > r.order)
                        n->coefficients[i] = l.coefficients[i];
                else
                        n->coefficients[i] = l.coefficients[i] + r.coefficients[
i];
                n->coefficients[i] %= m;
                n->coefficients[i] += m;
                n->coefficients[i] %= m;
        }
        return *n;
}

Polynomial& operator - (const Polynomial& l) {
        Polynomial *n = new Polynomial(0, l.modulus, -1);
        return *n *l;
}

Polynomial& operator * (const Polynomial& l, const Polynomial& r) {
```



```
        int order = l.order+r.order;

        /* Can't multiply polynomials under different moduli */
        assert(l.modulus == r.modulus);

        Polynomial *n = new Polynomial(order);
        n->modulus = l.modulus;
        int m = n->modulus;
        for (int i = 0; i <= l.order; i++) {
                for (int j = 0; j <= r.order; j++) {
                        n->coefficients[i+j] += l.coefficients[i]*r.coefficients
[j];
                        n->coefficients[i+j] %= m;
                        n->coefficients[i+j] += m;
                        n->coefficients[i+j] %= m;
                }
        }
        return *n;
}

Polynomial& operator * (int l, const Polynomial& r) {
        int order = r.order;

        Polynomial *n = new Polynomial(r.order);
        n->modulus = r.modulus;
        int m = r.modulus;
        for (int i = 0; i <= r.order; i++) {
                n->coefficients[i] = r.coefficients[i]*l;
                n->coefficients[i] %= m;
                n->coefficients[i] += m;
                n->coefficients[i] %= m;
        }
        return *n;
}

Polynomial& operator * (const Polynomial& l, int r) {
        return r * l;
}

void dividePolynomial(const Polynomial& l, const Polynomial& r, Polynomial** ret
q, Polynomial** retm) {
        /* Can't modulo polynomials under different moduli */
        assert(l.modulus == r.modulus);

        Polynomial *divisor = new Polynomial(r); divisor->modulus = l.modulus;
        Polynomial *quotient = new Polynomial(l.highestOrder()); quotient->modul
us = l.modulus;
        Polynomial *remainder = new Polynomial(l); remainder->modulus = l.modulu
s;

        while (remainder->highestOrder() >= divisor->highestOrder()) {
                int n = remainder->coefficients[remainder->highestOrder()] / div
isor->coefficients[divisor->highestOrder()];
                int o = remainder->highestOrder() - divisor->highestOrder();
                quotient->coefficients[o] = n;
                Polynomial mult(o); mult.modulus = l.modulus; mult.coefficients[
o] = n;
                *remainder += -(*divisor * mult);
        }

        if (retq)
                *retq = quotient;
        else
                delete quotient;
        if (retm)
```

```
                *retm = remainder;
        else
                delete remainder;
}

Polynomial& operator / (const Polynomial& l, const Polynomial& r) {
        Polynomial *q;
        dividePolynomial(l, r, &q, NULL);
        return *q;
}

Polynomial& operator % (const Polynomial& l, const Polynomial& r) {
        Polynomial *m;
        dividePolynomial(l, r, NULL, &m);
        return *m;
}

Polynomial& operator ^ (const Polynomial& l, int pow) {
        if (pow == 0)
                return *new Polynomial(0, l.modulus, 1);
        Polynomial *res = new Polynomial(l);
        for (int i = 1; i < pow; i++) {
                *res *= l;
        }
        return *res;
}
```