```
#ifndef ENSIIE_RATIONAL_H
#define ENSIIE_RATIONAL_H

#include <ostream>

namespace ensiie {

class Rat
{
  private: // data members
    int num_;
    int den_;

  public: // structors
    Rat(int num, int den = 1);

  public: // getters
    int get_num() const { return num_; }
    int get_den() const { return den_; }

  public: // setters
    void set_num(int num) { num_ = num; }
    void set_den(int den);
    void set(int num, int den);

  public: // operators
    Rat & operator+=(const Rat & r);
    Rat & operator+=(int v);
    Rat & operator-=(const Rat & r);
    Rat & operator-=(int v);
    Rat & operator*=(const Rat & r);
    Rat & operator*=(int v);
    Rat & operator/=(const Rat & r);
    Rat & operator/=(int v);

    friend Rat operator+(const Rat & r1, const Rat & r2);
    friend Rat operator-(const Rat & r1, const Rat & r2);
    friend Rat operator*(const Rat & r1, const Rat & r2);
    friend Rat operator/(const Rat & r1, const Rat & r2);
    friend bool operator==(const Rat & r1, const Rat & r2);
    friend bool operator==(const Rat & r1, int v);
    friend bool operator!=(const Rat & r1, const Rat & r2);
    friend bool operator<=(const Rat & r1, const Rat & r2);
    friend bool operator<(const Rat & r1, const Rat & r2);
    friend bool operator>=(const Rat & r1, const Rat & r2);
    friend bool operator>(const Rat & r1, const Rat & r2);

    friend std::ostream & operator<<(std::ostream & st, const Rat &r);

  private:
    int pgcd(int a, int b);
};

} // namespace ensiie

#endif // ENSIIE_RATIONAL_H
```

```
#include "rational.h"

namespace ensiie {

Rat::Rat(int num, int den)
{
    set(num, den);
}

void Rat::set_den(int den)
{
    if (den == 0)
        throw "Nul denominator";

    den_ = den;
}

void Rat::set(int num, int den)
{
    if (den == 0)
        throw "Nul denominator";

    if (num == 0)
        den = 1;

    if (den < 0)
    {
        num = -num;
        den = -den;
    }

    int d = pgcd(num, den);

    num_ = num / d;
    den_ = den / d;
}

Rat & Rat::operator+=(const Rat & r)
{
    set(num_ * r.den_ + den_ * r.num_, den_ * r.den_);

    return *this;
}

Rat & Rat::operator+=(int v)
{
    *this += Rat(v);

    return *this;
}

Rat & Rat::operator-=(const Rat & r)
{
    set(num_ * r.den_ - den_ * r.num_, den_ * r.den_);

    return *this;
}

Rat & Rat::operator-=(int v)
{
    *this -= Rat(v);

    return *this;
}

Rat & Rat::operator*=(const Rat & r)
{
    set(num_ * r.num_, den_ * r.den_);
    return *this;
}

Rat & Rat::operator*=(int v)
{
    *this *= Rat(v);

    return *this;
}

Rat & Rat::operator/=(const Rat & r)
{
    if (r == 0)
        throw "division by zero";

    set(num_ * r.den_, den_ * r.num_);

    return *this;
}

Rat & Rat::operator/=(int v)
{
    *this /= Rat(v);

    return *this;
}

Rat operator+(const Rat & r1, const Rat & r2)
{
    Rat r = r1;

    return r += r2;
}

Rat operator-(const Rat & r1, const Rat & r2)
{
    Rat r = r1;

    return r -= r2;
}

Rat operator*(const Rat & r1, const Rat & r2)
{
    Rat r = r1;

    return r *= r2;
}

Rat operator/(const Rat & r1, const Rat & r2)
{
    Rat r = r1;

    return r /= r2;
}

bool operator==(const Rat & r1, const Rat & r2)
{
    return r1.num_ * r2.den_ == r1.den_ * r2.num_;
}

bool operator==(const Rat & r, int v)
{
    return r == Rat(v);
}

bool operator!=(const Rat & r1, const Rat & r2)
{
    return !(r1 == r2);
}

bool operator<=(const Rat & r1, const Rat & r2)
{
    return r1.num_ * r2.den_ <= r1.den_ * r2.num_;
}

bool operator<(const Rat & r1, const Rat & r2)
{
    return r1.num_ * r2.den_ < r1.den_ * r2.num_;
}

bool operator>=(const Rat & r1, const Rat & r2)
{
    return r1.num_ * r2.den_ >= r1.den_ * r2.num_;
}

bool operator>(const Rat & r1, const Rat & r2)
{
    return r1.num_ * r2.den_ > r1.den_ * r2.num_;
}

std::ostream & operator<<(std::ostream & st, const Rat &r)
{
    st << r.num_;
    if (r.den_ != 1)
        st << "/" << r.den_;

    return st;
}
```

std::cout = operator << (std::cout, c);

```
int Rat::pgcd(int a, int b)
{
    // b always non nul
    if (a == 0)
        return 1;

    if (a < 0) a = -a;
    if (b < 0) b = -b;
    if (a < b)
    {
        int tmp = a;
        a = b;
        b = tmp;
    }

    int r = a;
    while (r != 0)
    {
        r = a % b;
        a = b;
        b = r;
    }

    return a;
}

} // namespace ensiie
```

S & operator (int i){
T t;
t[i] = val ⇔ S val =
t.operator[](i) = val; (1)
class T {
    S * data_; }

(1) ⇔ (2)

S & operator [] (int i){
return data[i]; }
⊃⊂ t.operator[](i) : adresse de
t.data_[i]
équivalent à *(t.data_+i) = val (2)

```
#include <iostream>

#include "rational.h"

int main()
{
    try
    {
        ensiie::Rat r1(15, 3);
        std::cout << r1 << std::endl;
        ensiie::Rat r2(21, 15);
        std::cout << r2 << std::endl;
        std::cout << r1 + r2 << std::endl;
        ensiie::Rat r3(0, 15);
        std::cout << r3 << std::endl;
        std::cout << r1 / r3 << std::endl;
    }
    catch(const char *str)
    {
        std::cout << str << std::endl;
    }

    return 0;
}
```

```
#ifndef ENSIIE_VECTOR_H
#define ENSIIE_VECTOR_H

#include <ostream>

namespace ensiie {

class Vector
{
  private:
    double *data_;
    int size_;

  public: // structors
    Vector(int size);
    ~Vector();
    Vector(const Vector & v);

  public: // getters
    int get_size() const { return size_; }
    double operator[](int idx) const;

  public: // setters
    double & operator[](int idx);

  public: // operators

    friend Vector operator+(const Vector & v1, const Vector & v2);
    friend Vector operator-(const Vector & v1, const Vector & v2);
    friend double operator*(const Vector & v1, const Vector & v2);
    friend Vector operator*(const Vector & v, double val);
    friend Vector operator*(double val, const Vector & v);
    friend Vector operator/(const Vector & v1, double val);

    friend std::ostream & operator<<(std::ostream & st, const Vector &v);

  public:
    double norm(double p = 2) const;
    friend Vector cross(const Vector & v1, const Vector & v2);
};

} // namespace ensiie

#endif // ENSIIE_VECTOR_H
```

```
#include <cmath>

#include "vector.h"

namespace ensiie {

Vector::Vector(int size)
{
    if (size <= 0)
        throw "wrong vector size";

    size_ = size;
    data_ = new double[size_];
    if (!data_)
        throw "allocation of memory failed";
}

Vector::~Vector()
{
    delete [] data_;
}

Vector::Vector(const Vector & v)
{
    size_ = v.get_size();
    data_ = new double[size_];
    if (!data_)
        throw "allocation of memory failed";

    for (int i = 0; i < v.get_size(); i++)
        data_[i] = v[i];
}

double Vector::operator[](int idx) const
{
    if ((idx < 0) || (idx >= size_))
        throw "wrong index";

    return data_[idx];
}

double &Vector::operator[](int idx)
{
    if ((idx < 0) || (idx >= size_))
        throw "wrong index";

    return data_[idx];
}

Vector operator+(const Vector & v1, const Vector & v2)
{
    if (v1.get_size() != v2.get_size())
        throw "wrong size";

    Vector v(v1.get_size());

    for (int i = 0; i < v1.get_size(); i++)
        v[i] = v1[i] + v2[i];

    return v;
}

Vector operator-(const Vector & v1, const Vector & v2)
{
    if (v1.get_size() != v2.get_size())
        throw "wrong size";

    Vector v(v1.get_size());
    for (int i = 0; i < v1.get_size(); i++)
        v[i] = v1[i] - v2[i];

    return v;
}

double operator*(const Vector & v1, const Vector & v2)
{
    if (v1.get_size() != v2.get_size())
        throw "wrong size";

    double ip = 0;

    for (int i = 0; i < v1.get_size(); i++)
        ip += v1[i] * v2[i];

    return ip;
}

Vector operator*(const Vector & v, double val)
{
    Vector res(v.get_size());

    for (int i = 0; i < v.get_size(); i++)
        res[i] = v[i] * val;

    return res;
}

Vector operator*(double val, const Vector & v)
{
    return v * val;
}

Vector operator/(const Vector & v, double val)
{
    Vector res(v.get_size());

    for (int i = 0; i < v.get_size(); i++)
        res[i] = v[i] * val;

    return res;
}

std::ostream & operator<<(std::ostream & st, const Vector &v)
{
    st << "(";
    for (int i = 0; i < v.get_size(); i++)
    {
        st << v[i];
        if (i != (v.get_size() - 1))
            st << ",";
    }
    st << ")";

    return st;
}

double Vector::norm(double p) const
{
    if (p < 1)
        throw "wrong exponent";

    double n = 0;
    for (int i = 0; i < size_; i++)
        n += pow(fabs(operator[](i)), p);
    return pow(n, 1/p);
}

Vector cross(const Vector & v1, const Vector & v2)
{
    if ((v1.get_size() != 3) || (v2.get_size() != 3))
        throw "wrong size";

    Vector v(3);
    v[0] = v1[1] * v2[2] - v1[2] * v2[1];
    v[1] = v1[2] * v2[0] - v1[0] * v2[2];
    v[2] = v1[0] * v2[1] - v1[1] * v2[0];

    return v;
}

} // namespace ensiie
```

```
#include <iostream>

#include "vector.h"

int main()
{
    try
    {
        ensiie::Vector v1(3);
        v1[0] = 1;
        v1[1] = 3;
        v1[2] = 2;
        std::cout << v1 << std::endl;
        std::cout << v1.norm() << std::endl;
        ensiie::Vector v2(3);
        v2[0] = 3;
        v2[1] = 0;
        v2[2] = 5;
        std::cout << v2 << std::endl;
        ensiie::Vector v3 = cross(v1, v2);
        std::cout << v3 << std::endl;
        std::cout << v1 * v3 << std::endl;
    }
    catch(const char *str)
    {
        std::cout << str << std::endl;
    }

    return 0;
}
```