作者：董佳昕

一、问题描述

编写一个**Fraction**类，实现以下功能

1. 缺省构造函数
2. 以两个整数作为参数的构造函数
3. 拷贝构造函数
4. 算术运算符重载：+、 -、 *、 /
5. 关系运算符重载： <、 >、 <=、 >=、 ==、 !=
6. 类型转化为double
7. 转化为字符串
8. 从流中读取/删除
9. 从一个有限精度的浮点数字符串转化为Fraction:1.414

二、实现思路

2.1 成员变量

两个long long类型的整数，分别表示分子、分母

```
class Fraction{
    long long numerator;  //分子
    long long denominator;//分母
}
```

2.2 辅助函数

1. 计算两个数的最大公约数
2. 计算两个数的最小公倍数
3. 判断浮点数是否为整数
4. 将分数化简
5. setter
6. getter

```
static LL gcd(LL A, LL B) {
    if (A < B)return gcd(B, A);
    if (B == 0)return A;
    return gcd(B, A % B);
}
static LL lcm(LL A, LL B) {
    return 1ll * A * B / gcd(A, B);
}
static bool isInteger(double A) {
    return ceil(A) == floor(A);
}
void Simplify() {
    if (numerator == 0)return;
    int GCD = gcd(numerator, denominator);
    numerator /= GCD;
```

```
    denominator /= GCD;
}
void setNumerator(LL numerator) {
    this->numerator = numerator;
}
void setDenominator(LL denominator) {
    this->denominator = denominator;
}
LL getNumerator() {
    return this->numerator;
}
LL getDenominator() {
    return this->denominator;
}
```

2.3 构造函数

```
//缺省构造函数
Fraction() {
    numerator = 0;
    denominator = 1;
}
//有两个整数作为参数的构造函数
Fraction(LL numerator, LL denominator) {
    this->numerator = numerator;
    this->denominator = denominator;
    this->Simplify();
}
//拷贝构造函数
Fraction(const Fraction &A) {
    this->numerator = A.numerator;
    this->denominator = A.denominator;
    this->Simplify();
}
```

2.4 算术运算符重载

```
const Fraction operator+(const Fraction& A)const {
    Fraction ans;
    ans.denominator = lcm(A.denominator, this->denominator);
    ans.numerator = ans.denominator / this->denominator * this->numerator;
    ans.numerator += ans.denominator / A.denominator * A.numerator;
    ans.Simplify();
    return ans;
}
const Fraction operator-(const Fraction& A)const {
    Fraction ans;
    ans.denominator = lcm(A.denominator, this->denominator);
    ans.numerator = ans.denominator / this->denominator * this->numerator;
    ans.numerator -= ans.denominator / A.denominator * A.numerator;
    ans.Simplify();
    return ans;
}
const Fraction operator*(const Fraction& A)const {
    Fraction ans;
```

```
    ans.denominator =  this->denominator * A.denominator;
    ans.numerator = this->numerator * A.numerator;
    ans.Simplify();
    return ans;
}
const Fraction operator/(const Fraction& A)const {
    Fraction ans;
    ans.denominator = this->denominator * A.numerator;
    ans.numerator = this->numerator * A.denominator;
    ans.Simplify();
    return ans;
}
```

2.5 关系运算符重载

```
bool operator<(const Fraction& A)const {
    int LCM = lcm(this->denominator, A.denominator);
    int this_numrator = LCM / this->denominator * this->numerator;
    int A_numerator = LCM / A.denominator * A.numerator;
    return this_numrator < A_numerator;
}
bool operator>(const Fraction& A)const {
    return A < *this;
}
bool operator<=(const Fraction& A)const {
    return !(*this > A);
}
bool operator>=(const Fraction& A)const {
    return !(*this < A);
}
bool operator==(const Fraction& A)const {
    return this->numerator == A.numerator && this->denominator ==
A.denominator;
}
bool operator!=(const Fraction& A)const {
    return !(*this == A);
}
```

2.6 类型转化为双精度

```
operator double() const {
    return (double)this->numerator / (double)this->denominator;
}
```

2.7 转化为字符串

```
string toString() {
    char s[100];
    sprintf(s, "%d/%d", this->numerator, this->denominator);
    string ans(s);
    return ans;
}
```

2.8 从一个有限精度的浮点数字符串转化为Fraction

```
void conversion(string value) {
    double val = 0;
    sscanf(value.c_str(), "%lf", &val);
    denominator = 1;
    while (!isInteger(val * denominator))denominator *= 10;
    numerator = val * denominator;
}
```

2.9 从输入流中读取

```
static istream& operator>>(istream& is, Fraction& A) {
    LL numerator = 0, denominator = 1;
    is >> numerator >> denominator;
    A.setNumerator(numerator);
    A.setDenominator(denominator);
    A.Simplify();
    return is;
}
```

2.10 向输出流中输出

```
static ostream& operator<<(ostream& os, Fraction& A) {
    return os << A.getNumerator() << '/' << A.getDenominator();
}
```

三、测试样例

```
default ctor(A):0/1

please input two integer
20 11
ctor takes two integers as parameters(A):20/11

copy ctor(B):20/11

arithmetical operators:
A + B : 40/11
A - B : 0/11
A * B : 400/121
A / B : 1/1

relational operators:
A <  B : false
A <= B : true
A == B : true
A != B : false
A >= B : true
A >  B : false

type cast to double:1.81818

to string(A):20/11

please input a finite decimal string like: 1.414
1.56
conversion from a finite decimal string(A):39/25
```