

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

```
1 class Fraction{
2     long long numerator; //分子
3     long long denominator;//分母
4 }
```

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

```
1 static LL gcd(LL A, LL B) {
2     if (A < B) return gcd(B, A);
3     if (B == 0) return A;
4     return gcd(B, A % B);
5 }
6 static LL lcm(LL A, LL B) {
7     return 1LL * A * B / gcd(A, B);
8 }
9 static bool isInteger(double A) {
10     return ceil(A) == floor(A);
11 }
12 void simplify() {
13     if (numerator == 0) return;
14     int GCD = gcd(numerator, denominator);
15     numerator /= GCD;
```

```

16     denominator /= GCD;
17 }
18 void setNumerator(LL numerator) {
19     this->numerator = numerator;
20 }
21 void setDenominator(LL denominator) {
22     this->denominator = denominator;
23 }
24 LL getNumerator() {
25     return this->numerator;
26 }
27 LL getDenominator() {
28     return this->denominator;
29 }

```

2.3 构造函数

```

1 //缺省构造函数
2 Fraction() {
3     numerator = 0;
4     denominator = 1;
5 }
6 //有两个整数作为参数的构造函数
7 Fraction(LL numerator, LL denominator) {
8     this->numerator = numerator;
9     this->denominator = denominator;
10    this->Simplify();
11 }
12 //拷贝构造函数
13 Fraction(const Fraction &A) {
14     this->numerator = A.numerator;
15     this->denominator = A.denominator;
16     this->Simplify();
17 }

```

2.4 算术运算符重载

```

1 const Fraction operator+(const Fraction& A)const {
2     Fraction ans;
3     ans.denominator = lcm(A.denominator, this->denominator);
4     ans.numerator = ans.denominator / this->denominator * this->numerator;
5     ans.numerator += ans.denominator / A.denominator * A.numerator;
6     ans.Simplify();
7     return ans;
8 }
9 const Fraction operator-(const Fraction& A)const {
10    Fraction ans;
11    ans.denominator = lcm(A.denominator, this->denominator);
12    ans.numerator = ans.denominator / this->denominator * this->numerator;
13    ans.numerator -= ans.denominator / A.denominator * A.numerator;
14    ans.Simplify();
15    return ans;
16 }
17 const Fraction operator*(const Fraction& A)const {
18    Fraction ans;

```

```

19     ans.denominator = this->denominator * A.denominator;
20     ans.numerator = this->numerator * A.numerator;
21     ans.Simplify();
22     return ans;
23 }
24 const Fraction operator/(const Fraction& A)const {
25     Fraction ans;
26     ans.denominator = this->denominator * A.numerator;
27     ans.numerator = this->numerator * A.denominator;
28     ans.Simplify();
29     return ans;
30 }

```

2.5 关系运算符重载

```

1  bool operator<(const Fraction& A)const {
2      int LCM = lcm(this->denominator, A.denominator);
3      int this_numrator = LCM / this->denominator * this->numerator;
4      int A_numerator = LCM / A.denominator * A.numerator;
5      return this_numrator < A_numerator;
6  }
7  bool operator>(const Fraction& A)const {
8      return A < *this;
9  }
10 bool operator<=(const Fraction& A)const {
11     return !(*this > A);
12 }
13 bool operator>=(const Fraction& A)const {
14     return !(*this < A);
15 }
16 bool operator==(const Fraction& A)const {
17     return this->numerator == A.numerator && this->denominator ==
A.denominator;
18 }
19 bool operator!=(const Fraction& A)const {
20     return !(*this == A);
21 }

```

2.6 类型转化为双精度

```

1  operator double() const {
2      return (double)this->numerator / (double)this->denominator;
3  }

```

2.7 转化为字符串

```

1  string toString() {
2      char s[100];
3      sprintf(s, "%d/%d", this->numerator, this->denominator);
4      string ans(s);
5      return ans;
6  }

```

2.8 从一个有限精度的浮点数字符串转化为Fraction

```

1 void conversion(string value) {
2     double val = 0;
3     sscanf(value.c_str(), "%lf", &val);
4     denominator = 1;
5     while (!isInteger(val * denominator))denominator *= 10;
6     numerator = val * denominator;
7 }

```

2.9 从输入流中读取

```

1 static istream& operator>>(istream& is, Fraction& A) {
2     LL numerator = 0, denominator = 1;
3     is >> numerator >> denominator;
4     A.setNumerator(numerator);
5     A.setDenominator(denominator);
6     A.Simplify();
7     return is;
8 }

```

2.10 向输出流中输出

```

1 static ostream& operator<<(ostream& os, Fraction& A) {
2     return os << A.getNumerator() << '/' << A.getDenominator();
3 }

```

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

```

