#pragma once

#include<iostream>

template <typename T>class fraction;

template <typename T>std::ostream& operator<<(std::ostream& os, const fraction<T>& fraction);

template <typename T>std::istream& operator>>(std::istream& is, fraction<T>& fraction);

template <typename T>class fraction {

private:

T numerator;

T denominator;

public:

//CONSTRUCTORS\DESTRUCTORS

fraction() :fraction(0, 1) {};

fraction(T num):fraction(num,1){}

fraction(T num, T denom)

{

denominator = denom;

numerator = num;

reduce();

}

//DATA ACCESS

T getNumerator() { return numerator; }

T getDenominator() { return denominator; }

//ARITHMETIC OPERATORS

fraction operator+(const fraction& other)const;

fraction operator\*(const fraction& other)const;

fraction operator\*(const T& other)const;

fraction operator/(const fraction & other)const;

fraction operator-(const fraction& other)const;

fraction operator-()const;

// I/O OPERATIONS

friend std::ostream& operator<<<>(std::ostream& os, const fraction<T>& fraction);

friend std::istream& operator>><>(std::istream& is, fraction<T>& fraction);

//COMPARISON OPERATORS WITH ZERO

fraction operator=(const fraction other);

fraction operator=(const T other);

//COMPARISON OPERATORS

bool operator==(const fraction& other)const;

//it only works for comparison with 1 and 0

bool operator==(const T other) const;

bool operator!=(const fraction& other) const;

bool operator>(const fraction& other)const;

bool operator<(const fraction& other)const;

//SPECIAL METHODS

fraction<T>& reduce();

T findGCD(T& a, T& b)const;

};

#include "fraction.cpp"