Create a Fraction class which conforms to the following requirements. The class name should be Fraction with the declaration in Fraction.h and the definitions in Fraction.cpp. The following constructors, getters, setters, functions and operator overloading should be provided within your class. The class should pass all the tests provided in FractionDriver.cpp without modifying driver code! **YOU** should update the functions appropriately in the resulting .cpp AND .h files (meaning I have intentionally left out things like “friend”, “const”, passing by reference from functions and parameters, etc.)! There are comments to include a few (like getters) as inline functions. Likewise, there is a requirement that the ! operator call the invert function (this call can be done inline as well). Select operators can be defined in terms of its pair – and those definitions should be left as-is. Comments for functionality of some functions are provided. However, pre and post conditions are not always included below (but it may be wise to add those comments in to help with code clarity).

Note the location of multFracs – it is neither a member function, nor a friend function of the class (meaning you cannot directly access the member variables of the lhs and rhs parameters).

The following is meant to help guide construction of your Fraction.cpp and Fraction.h files… (This code should help start your Fraction.h file…)

///////////CONSTRUCTORS///////////

//default constructor setting fraction to 0/1

//constructor setting fraction to num/1

//constructor setting fraction to num/den or num/1 if den is zero

//copy constructor where num = source.num, den = source.den

///////////FUNCTIONS//////////////

///////////MEMBER/////////////////

//Description: Fraction's numerator and denominator are set to the users' input

// (unless they enter a denominator of zero!)

void readin();

//Fraction is print to cout in (N/D) format (numerator / denominator)

void print();

//Post: A new fraction is returned where

// the new numerator is the calling object's denominator

// and the new denominator is the calling object's numerator

Fraction reciprocal();

//Post: Calling object's numerator and denominator are individually multiplied by m

void unreduce(int m);

//Post: fraction value is returned as a float

inline float toDecimal() //inline definition

///////////////// ACCESSOR FUNCTIONS //////////////////

int getNumer() //inlined definition

int getDenom() //inlined definition

static double getZeroTol();

///////////////// MUTATOR FUNCTIONS ////////////////////

void setNumer(int n);

//Description: TRUE is returned if update was successful.

bool setDenom(int d);

static void setZeroTol(double d);

///////////OPERATORS//////////////

///////////MEMBER/////////////////

Fraction & operator= (Fraction rhs);

Fraction & operator\*= (Fraction rhs);

void operator! () //inline; call the **invert** function on the calling object

///////////NON MEMBER/////////////////

ostream & operator<< (ostream & out, Fraction source);

Fraction operator\* (Fraction lhs, Fraction rhs);

bool operator== (Fraction lhs, Fraction rhs);

//no not change the red lines!

bool operator!= (const Fraction & lhs, const Fraction & rhs) {return !(lhs==rhs);}

bool operator> (Fraction lhs, Fraction rhs);

bool operator<= (const Fraction & lhs, const Fraction & rhs) {return !(lhs > rhs);}

bool operator< (Fraction lhs, Fraction rhs);

bool operator>= (const Fraction & lhs, const Fraction & rhs) {return !(lhs < rhs);}

///////////FRIEND/////////////////

//Pre: Numerator cannot be zero

//Post: The fraction values are swapped - where

// the new numerator is the original denominator

// and the new denominator is the original numerator

friend void invert(Fraction & f);

private:

int m\_Numerator; //fraction's numerator

int m\_Denominator; //fraction's denominator

static double zero\_tolerance; //initial value should be set to 0.0001

};// END of class declaration

//Pre: lhs and rhs are valid fractions (neither den = 0)

//Post: returned fraction's numerator is the lhs numerator \* rhs numerator

// returned fraction's denominator is the lhs denominator \* rhs denominator

Fraction multFracs(const Fraction & lhs, const Fraction & rhs);