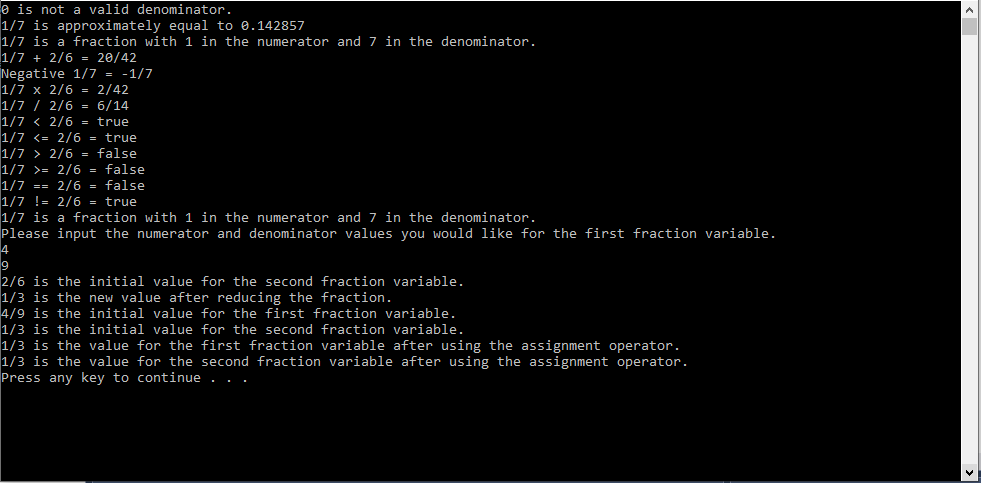
Name: William A. Brannon

Assignment: Lab Assignment Report #1

Date Due: Monday, January, 28, 2019

Class: Fundamentals of Programming II Section #1

**Program Output**



**Lab1.cpp**

// Lab1.cpp : fraction class, operator overloading

// By William A. Brannon on 01/23/2019

#include "stdafx.h"

#include <iostream>

#include <string>

using namespace std;

class Fraction

{

int num, den;

public:

Fraction() {

num = 0;

den = 1;

};

Fraction(int a, int b) {

num = a;

den = b;

};

Fraction(int a) {

num = a;

den = 1;

};

Fraction(const Fraction& a) {

num = a.num;

den = a.den;

};

static bool isValidDenom(int a) {

return a > 0;

};

double toDecimal() const {

return num / (double)den;

};

int numerator() const {

return num;

}

int denominator() const {

return den;

}

string toString() const {

return to\_string(num) + "/" + to\_string(den);

};

Fraction operator-() {

return Fraction(-num, den);

};

Fraction operator+(const Fraction &a)

{

return Fraction(num \* a.den +

den \* a.num,

den \* a.den);

}

Fraction operator\*(const Fraction &m)

{

return Fraction(this->num \* m.num,

this->den \* m.den);

}

Fraction operator/(const Fraction &d)

{

return Fraction(this->num \* d.den,

this->den \* d.num);

}

bool operator<(const Fraction &r) {

double thisVal = this->toDecimal();

double rVal = r.toDecimal();

return thisVal < rVal;

}

bool operator<=(const Fraction &r) {

double thisVal = this->toDecimal();

double rVal = r.toDecimal();

return thisVal <= rVal;

}

bool operator>(const Fraction &r) {

double thisVal = this->toDecimal();

double rVal = r.toDecimal();

return thisVal > rVal;

}

bool operator>=(const Fraction &r) {

double thisVal = this->toDecimal();

double rVal = r.toDecimal();

return thisVal >= rVal;

}

bool operator==(const Fraction &r) {

double thisVal = this->toDecimal();

double rVal = r.toDecimal();

return thisVal == rVal;

}

bool operator!=(const Fraction &r) {

double thisVal = this->toDecimal();

double rVal = r.toDecimal();

return thisVal != rVal;

}

friend ostream & operator<<(ostream& out, const Fraction& fr);

friend istream & operator>>(istream& in, Fraction& fr);

void reduceFraction() {

int gcd = 1;

int largest = (num > den) ? num : den;

for (int x = 2; x <= largest; ++x) {

if (num % x == 0 && den % x == 0) {

gcd = x;

}

}

num = num / gcd;

den = den / gcd;

}

Fraction operator=(const Fraction& a) {

this->num = a.num;

this->den = a.den;

return \*this;

}

};

ostream & operator<<(ostream& out, const Fraction& fr) {

if (fr.num == 0)

out << fr.num;

else if (fr.den == 1)

out << fr.num;

else

out << fr.num << '/' << fr.den;

return out;

};

istream & operator>>(istream& in, Fraction& fr) {

in >> fr.num >> fr.den;

return in;

};

int main()

{

//variables with arbitrary values for testing

Fraction newFrac(1, 7);

Fraction newFrac2(2, 6);

//Valid Denominator Test

if (Fraction::isValidDenom(0)) {

cout << "0 is a valid denominator." << endl;

}

else {

cout << "0 is not a valid denominator." << endl;

}

//To Decimal Test

cout << newFrac << " is approximately equal to " << newFrac.toDecimal() << endl;

//To String Test

cout << newFrac.toString() << " is a fraction with " << newFrac.numerator() <<" in the numerator and " << newFrac.denominator() << " in the denominator." << endl;

//Addition Test

cout << newFrac << " + " << newFrac2 << " = " << newFrac + newFrac2 << endl;

//Negative Test

cout << "Negative " << newFrac << " = " << -newFrac << endl;

//Multiplication Test

cout << newFrac << " x " << newFrac2 << " = " << newFrac \* newFrac2 << endl;

//Division Test

cout << newFrac << " / " << newFrac2 << " = " << newFrac / newFrac2 << endl;

//< Test

bool temp = newFrac < newFrac2;

string TempString;

if (temp) {

TempString = "true";

}

else {

TempString = "false";

}

cout << newFrac << " < " << newFrac2 << " = " << TempString << endl;

//<= Test

temp = newFrac <= newFrac2;

if (temp) {

TempString = "true";

}

else {

TempString = "false";

}

cout << newFrac << " <= " << newFrac2 << " = " << TempString << endl;

//> Test

temp = newFrac > newFrac2;

if (temp) {

TempString = "true";

}

else {

TempString = "false";

}

cout << newFrac << " > " << newFrac2 << " = " << TempString << endl;

//>= Test

temp = newFrac >= newFrac2;

if (temp) {

TempString = "true";

}

else {

TempString = "false";

}

cout << newFrac << " >= " << newFrac2 << " = " << TempString << endl;

//== Test

temp = newFrac == newFrac2;

if (temp) {

TempString = "true";

}

else {

TempString = "false";

}

cout << newFrac << " == " << newFrac2 << " = " << TempString << endl;

//!= Test

temp = newFrac != newFrac2;

if (temp) {

TempString = "true";

}

else {

TempString = "false";

}

cout << newFrac << " != " << newFrac2 << " = " << TempString << endl;

//stream insertion Test

cout << newFrac << " is a fraction with " << newFrac.numerator() << " in the numerator and " << newFrac.denominator() << " in the denominator." << endl;

//stream extraction Test

cout << "Please input the numerator and denominator values you would like for the first fraction variable." << endl;

cin >> newFrac;

//Fraction Reduction Test

cout << newFrac2 << " is the initial value for the second fraction variable." << endl;

newFrac2.reduceFraction();

cout << newFrac2 << " is the new value after reducing the fraction." << endl;

//Assignment Operator Test

cout << newFrac << " is the initial value for the first fraction variable." << endl;

cout << newFrac2 << " is the initial value for the second fraction variable." << endl;

newFrac = newFrac2;

cout << newFrac << " is the value for the first fraction variable after using the assignment operator." << endl;

cout << newFrac2 << " is the value for the second fraction variable after using the assignment operator." << endl;

system("pause");

return 0;

}