```
//              Course:  CS2400-60 Computer Science 2
//                Name:  Abdalkarim, Marina
//          Assignment:  Programming Assignment P10.1
//       Date assigned: 11/19/18
//            Date due: 12/6/18
//     Date handed in: 12/4/18
//              Remark: The program tests all functions and separate files.
// Code for rational.h
#include <iostream>
#ifndef rational_H
#define rational_H
#include <fstream>
#include <cstdlib>
#include <iomanip>
#include <cassert>
#include <string>
using namespace std;
class rational
{
	friend istream& operator>>(istream& in, rational &robj);
	// Postcondition: the two integer values entered by the user are assigned to num and
	// denom of robj
	friend ostream& operator<<(ostream& out, const rational &robj);
	// Postcondition: displays the contents of robj passed to the function in the following
	// format: a/b where b must be positive; e.g., 1/2,  -5/9 (not 5/-9), 1/4 (not 2/8, etc.)
public:
	rational();
	// default constructor
	rational(int, int);
	// second constructor
	void set(int n, int d);
	// mutator
	// Postcondition: calling rational object is set to n/d
	rational operator+(const rational &r2) const;
	// Postcondition: sum of calling rational object and r2 is returned
	rational operator-(const rational &r2) const;
	// Postcondition: (calling rational object - r2) is returned
	rational operator*(const rational &r2) const;
	// Postcondition: product of calling rational object and r2 is returned
```

```
        rational operator/(const rational &r2) const;
        // Postcondition: (calling rational object / r2) is returned
        bool operator<(const rational&r2) const;
        // try to use the overloaded '-' operator
        // Postcondition: returns true if the calling object is less than r2; returns false otherwise
        bool operator==(const rational&r2) const;
        // try to use the overloaded '-' operator
        // Postcondition: returns true if the calling object is equal to r2; returns false otherwise
        int getNum();
        // Another accessor function; gaining access to the value of the "num" data member of the
        // calling rational object
        // Postcondition: returns the name of the calling object
        int getDen();
        // Another accessor function; gaining access to the value of the "den" data member of the
        // calling rational object
        // Postcondition: returns the name of the calling object
        void setNum(int a);
        // Another accessor function; gaining access to the value of the "num" data member of the
        // calling rational object
        // Postcondition: initializes the calling objects
        void setDen(int a);
        // Another accessor function; gaining access to the value of the "den" data member of the
        // calling rational object
        // Postcondition: initializes the calling object
        int helper();
        // Another accessor function; gaining access to the gcd function
        // Postcondition: returns the greatest common divisor
private:
        int GCD() const;
        // Functions  kept in private section are known as the “helper” or “auxiliary” functions;
        // they help the public member functions
        // to carry out some subtasks; e.g., if a rational number internally stored as 2/8 should be
        // changed to 1/4 before it is displayed!
        // Postcondition: returns the "greatest common divisor" between the numerator and
        // denominator of the calling rational object
        int num;        // num: numerator
        int den;        // den: denominator
};
int fillArrayFromDiskFile(rational arr[]);
```

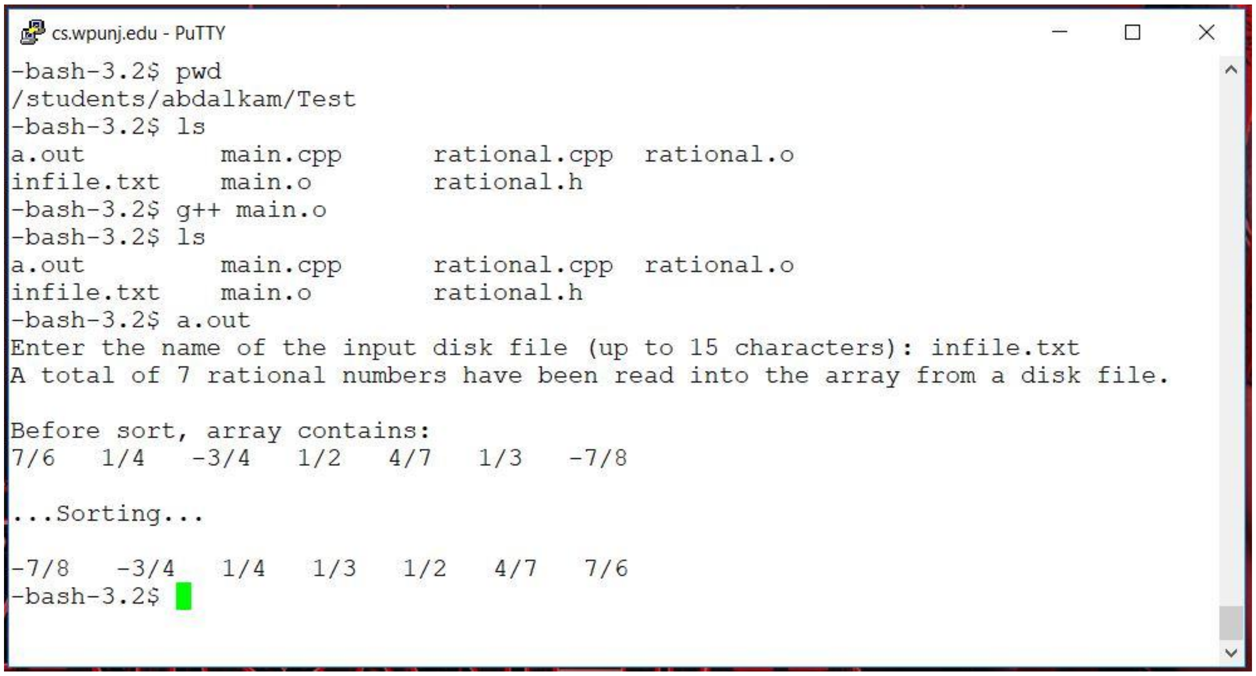
```
//  Precondition: array r[] is assumed to have enough capacity to store all rational numbers in the
// disk file
// Postcondition: returns the actual # of rational numbers read from the disk file
void displayArray(rational arr[], int n);
// Postcondition: display n rational numbers
void sort(rational arr[], int n);
//  Precondition: rational array declared in the calling function and the # of elements to be sorted
// must be passed to the function
// Postcondition: n rational number is the array are sorted in ascending order
void swap(rational &x, rational &y);
// Postcondition: contents of memory locations referenced by r1 and r2 are swapped
// Code for rational.cpp
#include "rational.h"
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <iomanip>
#include <cassert>
#include <string>
using namespace std;
void rational::set(int a, int b)
{
	num = a;
	den = b;
}
rational::rational()
{
	num = 0;
	den = 0;
}
rational::rational(int n, int d)
{
	num = n;
	den = d;
}
rational rational::operator+(const rational &r2) const
{
	rational sum;
	sum.num = (num * r2.den) + (r2.num * den);
	sum.den = den * r2.den;
	return sum;
}
istream& operator>>(istream& in, rational &robj)
{
	in >> robj.num >> robj.den;
	if (robj.den < 1)
	{
		robj.num = robj.num * (-1);
		robj.den = robj.den * (-1);
	}
	return in;
}
ostream& operator<<(ostream& out, const rational &robj)
{
	out << robj.num << "/" << robj.den;
	return out;
}
rational rational::operator-(const rational &r2) const
{
	rational diff;
	diff.num = (num * r2.den) - (r2.num * den);
	diff.den = den * r2.den;
	return diff;
}
rational rational::operator*(const rational &r2) const
{
	rational mul;
	mul.num = num * r2.num;
	mul.den = den * r2.den;
	return mul;
}
rational rational::operator/(const rational &r2) const
{
	rational div, temp;
	temp.num = r2.den;
	temp.den = r2.num;
	div.num = num * temp.num;
	div.den = den * temp.den;
	return div;
}
bool rational::operator<(const rational&r2) const
{
	rational diff;
	diff.num = (num * r2.den) - (r2.num * den);
	diff.den = den * r2.den;
	if (diff.num < 0)
		return true;
	else
		return false;
```

```
}
bool rational::operator==(const rational &r2) const
{
	rational diff;
	diff.num = (num * r2.den) - (r2.num * den);
	diff.den = den * r2.den;
	if (diff.num == 0)
		return true;
	else
		return false;
}
int fillArrayFromDiskFile(rational arr[])
{
	cout << "Enter the name of the input disk file (up to 15 characters): infile.txt" << endl;
	cout << "A total of 7 rational numbers have been read into the array from a disk file." <<
	endl << endl;
	ifstream fin;
	ifstream fileName("infile.txt");
	fin.open("infile.txt");
	if (fin.fail())
	{
		cout << "Input file opening failed.\n";
		exit(1);
	}
	for (int i = 0; i < 7; i++)
		fin >> arr[i];
	fin.close();
	return 0;
}
void displayArray(rational arr[], int n)
{
	for (int i = 0; i < n; i++)
	{
		int gcd = arr[i].helper();
		int c, d;
		c = arr[i].getNum() / gcd;
		d = arr[i].getDen() / gcd;
		if (d < 0)
		{
```

```
                    c = c * (-1);
                    d = d * (-1);
               }
               cout << c << "/" << d << "   ";
          }
}
void sort(rational arr[], int n)
{
          for (int pass = 1; pass < n; pass++)
          {
               for (int c = 0; c < n - pass; c++)
               {
                    double a, b, d, e;
                    a = arr[c].getNum();
                    b = arr[c].getDen();
                    d = arr[c + 1].getNum();
                    e = arr[c + 1].getDen();
                    double trial1, trial2;
                    trial1 = a / b;
                    trial2 = d / e;
                    if (trial1 > trial2)
                         swap(arr[c], arr[c + 1]);
               }
          }
}
void swap(rational &r1, rational &r2)
{
          rational temp;
          temp = r2;
          r2 = r1;
          r1 = temp;
}
int rational::getNum()
{
          return num;
}
int rational::getDen()
{
          return den;
```

```
}
void rational::setNum(int a)
{
        num = a;
}
void rational::setDen(int a)
{
        den = a;
}
int rational::helper()
{
        int gcd;
        gcd = GCD();
        return gcd;
}
int rational::GCD() const
{
        int a = 0, b = 0;
        int remainder = num % den;
        while (remainder != 0)
        {
                a = den;
                b = remainder;
                remainder = a % b;
        }
        return b;
}
```

```
// Code for main.cpp
#include "rational.h"
#include "rational.cpp"
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <iomanip>
#include <cassert>
#include <string>
using namespace std;
int main()
{
        const int SIZE = 7;
        rational s[SIZE];
        int fill, small = 0;
        fillArrayFromDiskFile(s);
        cout << "Before sort, array contains: " << endl;
        displayArray(s, SIZE);
        cout << endl;
        sort(s, SIZE);
        cout << endl;
        cout << "...Sorting..." << endl << endl;
        displayArray(s, SIZE);
        cout << endl;
        return 0;
}
```

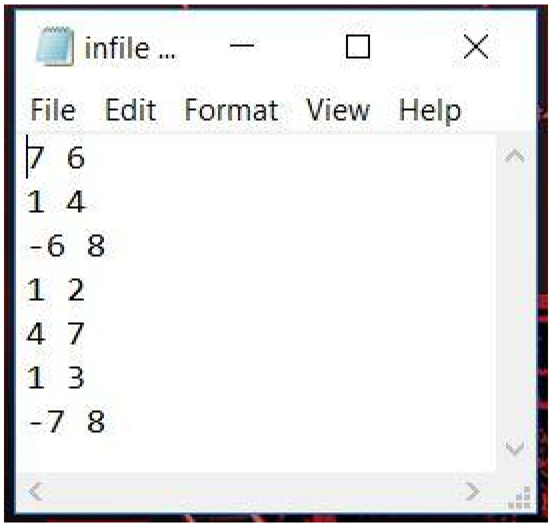


```
-bash-3.2$ pwd
/students/abdalkam/Test
-bash-3.2$ ls
a.out          main.cpp      rational.cpp  rational.o
infile.txt     main.o        rational.h
-bash-3.2$ g++ main.o
-bash-3.2$ ls
a.out          main.cpp      rational.cpp  rational.o
infile.txt     main.o        rational.h
-bash-3.2$ a.out
Enter the name of the input disk file (up to 15 characters): infile.txt
A total of 7 rational numbers have been read into the array from a disk file.

Before sort, array contains:
7/6   1/4   -3/4   1/2   4/7   1/3   -7/8

...Sorting...

-7/8   -3/4   1/4   1/3   1/2   4/7   7/6
-bash-3.2$
```



```
7 6
1 4
-6 8
1 2
4 7
1 3
-7 8
```