```
//             Course:  CS2400-60 Computer Science 2
//               Name:  Abdalkarim, Marina
//      Assignment:  Programming Assignment P7.1
//    Date assigned: 10/28/18
//           Date due: 11/27/18
//  Date handed in: 11/27/18
//             Remark: The program tests all functions.
#include <iostream>
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
        // den 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 helper();
        // Another accessor function; gaining access to the gcd function
        // Postcondition: returns the greatest common divisor
        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
```

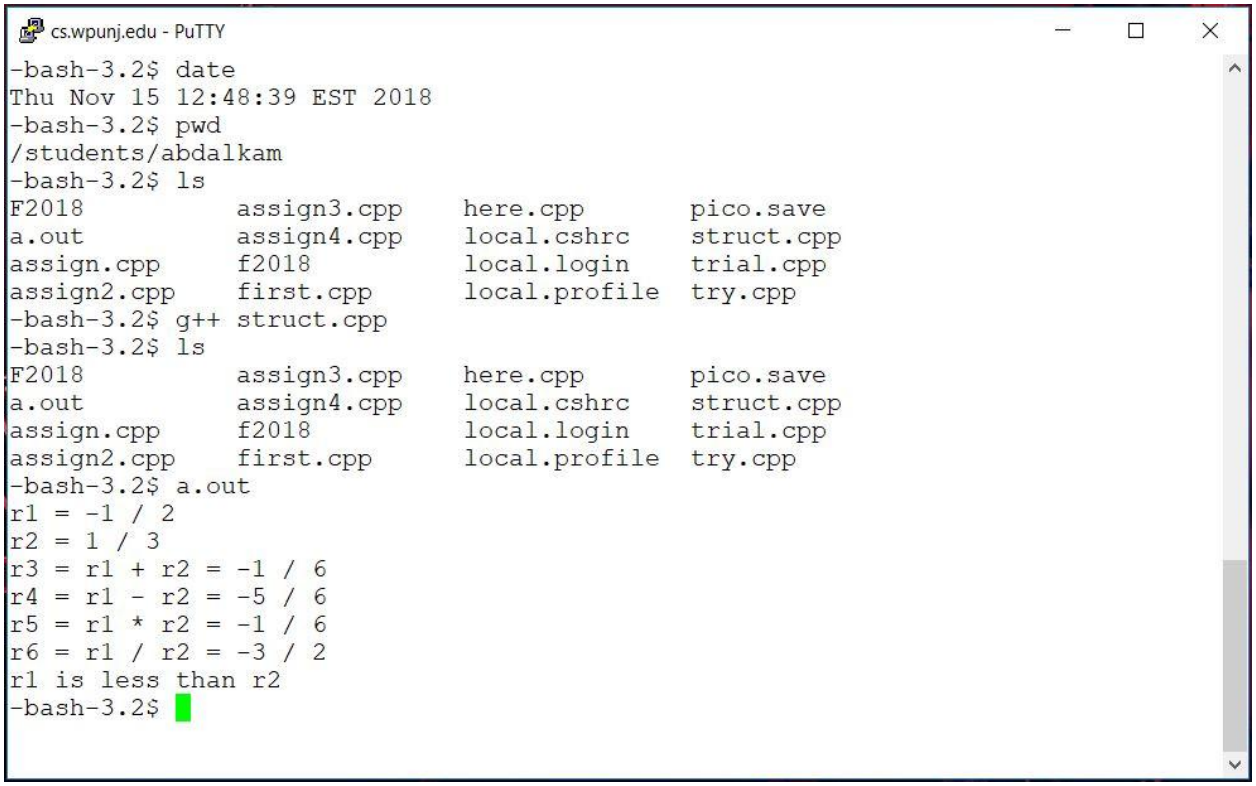
```
        // Postcondition: returns true if the calling object is less than r2; returns false otherwise
        bool operator==(const rational&r2) const;
        // try to use the overloaded '-' operator
        // Postcondition: returns true if the calling object is equal to r2; returns false otherwise
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 main()
{
        rational x, y;
        x.set(-1, 2);
        y.set(1, 3);
        x.helper();
        y.helper();
        cout << "r1 = ";
        cout << x << endl;
        cout << "r2 = ";
        cout << y << endl;
        cout << "r3 = r1 + r2 = ";
        cout << x + y << endl;
        cout << "r4 = r1 - r2 = ";
        cout << x - y << endl;
        cout << "r5 = r1 * r2 = ";
        cout << x * y << endl;
        cout << "r6 = r1 / r2 = ";
        cout << x / y << endl;
        if (x < y)                  cout << "r1 is less than r2" << endl;
        else if (x == y)

                cout << "r1 is equal to r2" << endl;
        else
```

```
                cout << "r1 is greater than r2" << endl;
        return 0;
}
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 << endl;
        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;
}
rational rational::helper()
{
	int gcd;
	gcd = GCD();
	rational a;
	a.num = num / gcd;
	a.den = den / gcd;
	return a;
}
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;
}
```




```
//             Course:  CS2400-60 Computer Science 2
//               Name:  Abdalkarim, Marina
//      Assignment:  Programming Assignment P7.2
//    Date assigned: 10/28/18
//          Date due: 11/27/18
//  Date handed in: 11/27/18
//            Remark: The program tests all functions.
#include <iostream>
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:
        int helper();
        // Another accessor function; gaining access to the gcd function
```

```
        // Postcondition: returns the greatest common divisor
        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
        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
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;
        int den;
};
int fillArray(rational r[], int size);
//  Precondition: address & physical size of the rational array declared in the calling function
// must be passed to the function
// Postcondition: returns the actual # of rational numbers entered by user which must be less than
// or equal to size
void displayArray(rational r[], int n);
// Postcondition: display n rational numbers
void selectionSort(rational r[], int n);
```

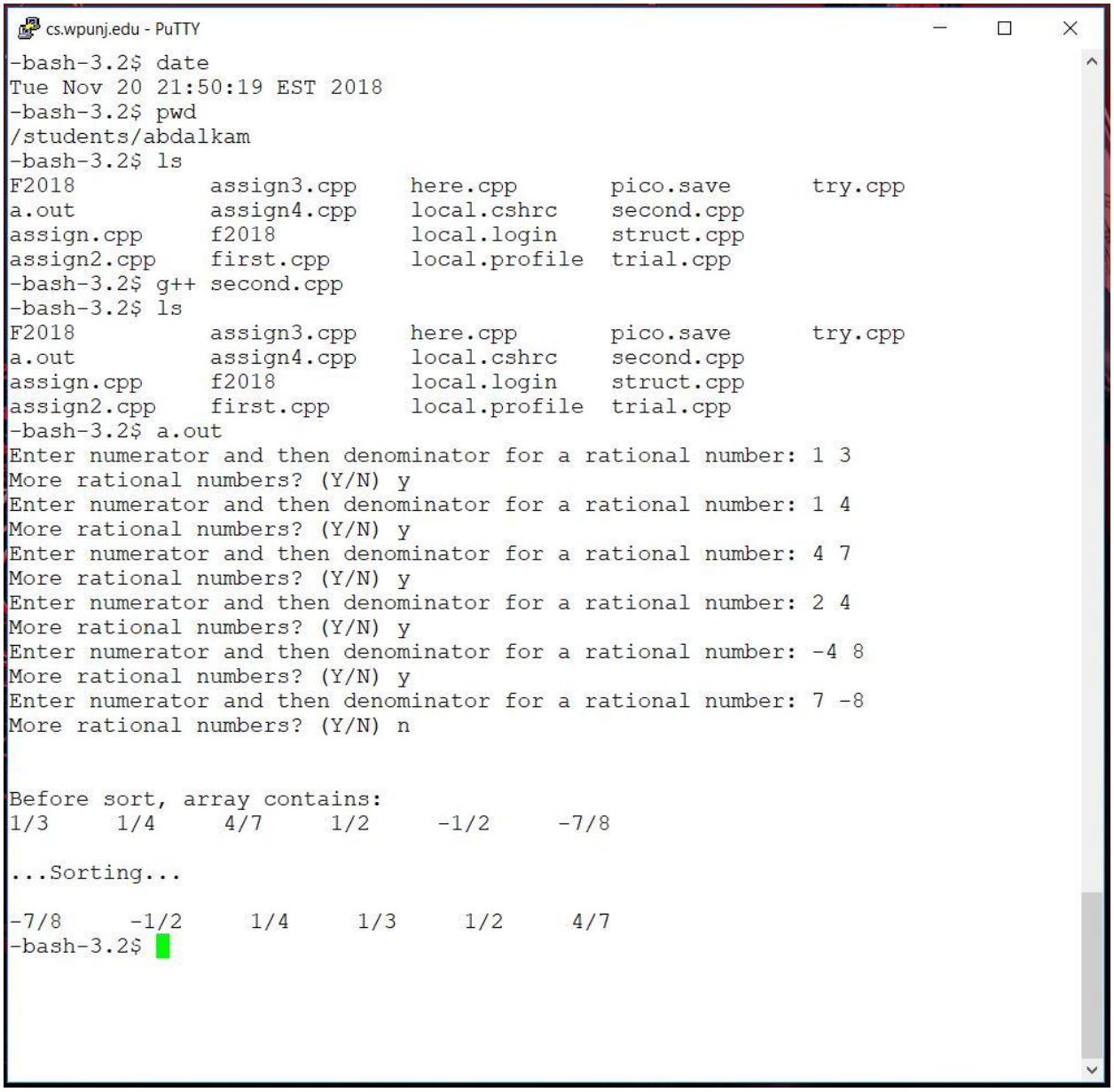
```
//  Precondition: rational array declared in the calling function and the # of elements to be sorted
// must be passed to the function
// Postcondition: n rational number is the array are sorted in ascending order
int findSmallestRationalNumber(rational r[], int first, int n);
//  Precondition: accepts address, subscript value of the first element of the unsorted sub-list, and
// the # of array elements n
// Postcondition: returns subscript value of the smallest rational number in the unsorted sub-list
// of the array
void swap(rational &r1, rational &r2);
// Postcondition: contents of memory locations referenced by r1 and r2 are swapped
int main()
{
	const int SIZE = 6;
	rational s[SIZE];
	int fill, small = 0;
	findSmallestRationalNumber(s, small, SIZE);
	fill = fillArray(s, SIZE);
	cout << endl << endl;
	cout << "Before sort, array contains: " << endl;
	displayArray(s, SIZE);
	cout << endl;
	selectionSort(s, SIZE);
	cout << endl;
	cout << "...Sorting..." << endl << endl;
	displayArray(s, SIZE);
	cout << endl;
	return 0;
}
istream& operator>>(istream& in, rational &robj)
{
	in >> robj.num >> robj.den;
	return in;
}
ostream& operator<<(ostream& out, const rational &robj)
{
	out << robj.num << "/" << robj.den;
	return out;
}
int fillArray(rational r[], int size)
```

```
{
	char ask = 'y';
	int more = 1;
	int i = 0;
	do
	{
		cout << "Enter numerator and then denominator for a rational number: ";
		cin >> r[i];
		cout << "More rational numbers? (Y/N) ";
		cin >> ask;
		i++;
		more++;
	} while (ask == 'y' || ask == 'Y');
	return more;
}
void displayArray(rational r[], int n)
{
	for (int i = 0; i < n; i++)
	{
		int gcd = r[i].helper();
		int c, d;
		c = r[i].getNum() / gcd;
		d = r[i].getDen() / gcd;
		if (d < 0)
		{
			c = c * (-1);
			d = d * (-1);
		}
		cout << c << "/" << d << "    ";
	}
}
void selectionSort(rational r[], int n)
{
	for (int pass = 1; pass < n; pass++)
	{
		for (int c = 0; c < n - pass; c++)
		{
			double a, b, d, e;
			a = r[c].getNum();
```

```
                        b = r[c].getDen();
                        d = r[c + 1].getNum();
                        e = r[c + 1].getDen();
                        double trial1, trial2;
                        trial1 = a / b;
                        trial2 = d / e;
                        if (trial1 > trial2)
                                swap(r[c], r[c + 1]);
                        int small;
                        small = findSmallestRationalNumber(r, c, n);
                        swap(r[0], r[small]);
                }
        }
}
int findSmallestRationalNumber(rational r[], int first, int n)
{
        rational small = r[first];
        for (int i = first + 1; i < n; i++)
        {
                double a, b, c, d, e, f;
                a = r[i].getNum();
                b = r[i].getDen();
                c = a / b;
                d = r[first].getNum();
                e = r[first].getDen();
                f = d / e;
                if (c < f)
                {
                        small = r[i];
                        first = i;
                }
                else
                {
                        small = r[first];
                        first = first;
                }
        }
        return first;
}
```

```
void swap(rational &r1, rational &r2)
{
	double temp1, temp2, a, b;
	temp1 = r1.getNum();
	a = r2.getNum();
	r1.setNum(a);
	r2.setNum(temp1);
	temp2 = r1.getDen();
	b = r2.getDen();
	r1.setDen(b);
	r2.setDen(temp2);
}
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 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;
}
```


cs.wpunj.edu - PuTTY
-bash-3.2$ date
Tue Nov 20 21:50:19 EST 2018
-bash-3.2$ pwd
/students/abdalkam
-bash-3.2$ ls
F2018 assign3.cpp here.cpp pico.save try.cpp
a.out assign4.cpp local.cshrc second.cpp
assign.cpp f2018 local.login struct.cpp
assign2.cpp first.cpp local.profile trial.cpp
-bash-3.2$ g++ second.cpp
-bash-3.2$ ls
F2018 assign3.cpp here.cpp pico.save try.cpp
a.out assign4.cpp local.cshrc second.cpp
assign.cpp f2018 local.login struct.cpp
assign2.cpp first.cpp local.profile trial.cpp
-bash-3.2$ a.out
Enter numerator and then denominator for a rational number: 1 3
More rational numbers? (Y/N) y
Enter numerator and then denominator for a rational number: 1 4
More rational numbers? (Y/N) y
Enter numerator and then denominator for a rational number: 4 7
More rational numbers? (Y/N) y
Enter numerator and then denominator for a rational number: 2 4
More rational numbers? (Y/N) y
Enter numerator and then denominator for a rational number: -4 8
More rational numbers? (Y/N) y
Enter numerator and then denominator for a rational number: 7 -8
More rational numbers? (Y/N) n
Before sort, array contains:
1/3 1/4 4/7 1/2 -1/2 -7/8
...Sorting...
-7/8 -1/2 1/4 1/3 1/2 4/7
-bash-3.2$