COS128: General Computer Science - Honors	**Spring 1996**

Problem Set # 2

Handed Out: Feb. 14, 1996 *Finish by: Feb. 21, 1996*

In this assignment, you will implement a "rational number package" – a set of functions that will use a precise representation of rational numbers and do standard arithmetic operations. Rational numbers are numbers that can be represented as the ratio of two integer, i.e. any number p/q where p and q are integers is a rational number. In C, we are forced to represent non-integer rational numbers using type `float` (or `double`). As you know, this representation can introduce imprecision. In our rational number package, we will use a pair of integers to represent a rational number. This can be done in C using an array. Suppose we want to declare a rational variable X. We can use

```
int X[2];
```

Since we will be using this type over an over again, we can use a *type definition* in C to give the type a name and then declare variables using that type name:

```
typedef int rational[2];
```

Now we can declare X as

```
rational X;
```

For the remainder of this description, we will refer to a rational number as a *pair* (p, q), which represents p/q. Certainly there are many representations for each rational number. For example, 2/3 can be represented as (2,3), (4,6), (18,27), or, in fact, $(2x, 3x)$ for any integer x. Therefore, we want to only use the *reduced* representation for a rational number. We will call a representation (p, q) *reduced* if:

1. q is positive;
2. The greatest common divisor of p and q is 1.

Therefore, to implement a rational number package, we first need a function to produce a reduced pair (p, q) from an arbitrary pair of integers (s, t). To do this, we need to calculate the greatest common divisor (abbreviated "gcd") of s and t. A famous algorithm for doing this is Euclid's algorithm [1]. This algorithm is based on the observation that, for positive s and t with $s \leq t$, the greatest common divisor of s and t is the same as the greatest common divisor of s and $t - s$. This observation yields the following function to compute the greatest common divisor:

[1] See Sedgewick, *Algorithms in C*, pg. 8.

```
int gcd (int s, int t)
/* returns greatest common divisor of s and t -- always positive */
/* modified from Sedgewick, Algorithms in C, pg 8. */

{
   int temp;
   s = abs(s);
   t = abs(t);
   if (s > t)               /*if s>t, switch s and t*/
       { temp = s;
         s = t;
         t = temp;
        }
   while (s > 0)
      {
         t = t-s;                /* subtract smaller from larger */
         if (s > t)               /*if s>t, switch s and t*/
              { temp = s;
                s = t;
                t = temp;
              }
       }
   return t;
}
```

where `abs(s)` is a standard library function in `stdlib` (so `#include <stdlib.h>`) that returns the absolute value of its argument.

When t is much larger than s, Euclid's algorithm as presented can iterate many times before finding the (relatively small) gcd. A similar algorithm can be written based on the observation that for positive s and t with $s \leq t$, the greatest common divisor of s and t is the same as the greatest common divisor of s and $t\%s$ (recall $t\%s$ is the remainder when t is divided by s). Both the original observation that gave Euclid's algorithm and this observation can be verified by writing t as $t = i \cdot s + r$, where r is the value of $t\%s$ and always has magnitude less than the magnitude of s, and i is the value of t/s.

The assignment:

Implement the following functions. Your functions should work correctly for negative rational numbers as well as positive rational numbers. (Hint: be sure not to make *any* assumptions about the arguments the function is passed except that they are the right type.)

```
void reduce (rational X);
```

replaces the pair (s, t) in X by the reduced pair (p, q). You should implement the faster gcd algorithm described above.

```
void add (rational X, rational Y, rational Result);
```

adds the rational numbers represented by X and Y and puts the *reduced* result in rational variable `Result`.

```
void minus (rational X, rational Result);
```

puts the *reduced* representation of the negative of the rational number represented by X in rational variable `Result`.

```
void multiply(rational X, rational Y, rational Result);
```

multiplies the rational numbers represented by X and Y and puts the *reduced* result in rational variable `Result`.

```
void divide(rational X, rational Y, rational Result);
```

multiplies the rational numbers represented by X and Y and puts the *reduced* result in rational variable `Result`.

Part 1, 70% Test your implementation by writing a main procedure that takes a rational number X entered from the keyboard and prints the pair that represents the rational result of evaluating $X^2/(X-1)$. Your main procedure should prompt the user to enter a numerator from the terminal and then prompt the user to enter a denominator from the terminal. Test your program on many different values of X.

Part 2, 30% Now we will revisit the midpoint method of estimating the area under a curve that you implemented in Assignment 1. Re-implement this algorithm for function x^2 using your rational numbers package. Test your new area estimator on interval $l = -1$ and $r = 1$ for $n = 1$ to $n = 100$. For each n, print out the pair representing the rational number that is the resulting area estimate. Also print out the difference, as a rational number, between this estimate and the actual area (2/3). (Make sure your printout clearly identifies each value.) If you have done a correct implementation, you should get reasonable estimates for the area for small values of n. However, at some point as you increase n, you will start getting strange results. (For a straightforward implementation that I wrote it happened at around $n = 40$, but where "it" happens depends on your exact implementation.) What is going wrong? (Hint: use lots of `printf`s to follow the calcuation for an n that gives strange results.)

Submission: Put the two programs that you write in two separate files: (1) `rat_poly.c` with the program for Part 1; and (2) `rat_area.c` with the program for Part 2. To electronically submit the two programs, type

```
/u/cs128/bin/submit 2 rat_poly.c rat_area.c
```

Programs must be electronically submitted by **10pm on Wednesday, February 21, 1996.** Hardcopy must be turned in during lecture the next morning. Your hardcopy should include your explanation of what is going wrong in part 2. Remember to use the official cover sheet for the hardcopy you submit.

Extra credit (10%): Modify your program so that the type of problem encountered in Part 2 does not just happen without any signal to the caller of the offending rational number function. It is up to you to decide what is the right thing to do and implement it. (If what you decide is the right thing is too difficult to implement in a reasonable amount of time, describe it. But be aware that some simple feedback can be given the user without requiring complex implementation.)

Challenge for the bored: Implement a rational number calculator that can take constants and operations entered by the user from the terminal and print results to the terminal.