Homework #1

RELEASE DATE: 03/06/2015

DUE DATE: 03/24/2015, **16:20** in CSIE R102/R104 and on the online submission system

As directed below, you need to submit your code to the designated place on the course website.

Any form of cheating, lying or plagiarism will not be tolerated. Students can get zero scores and/or get negative scores and/or fail the class and/or be kicked out of school and/or receive other punishments for those kinds of misconducts.

Discussions on course materials and homework solutions are encouraged. But you should write the final solutions alone and understand them fully. Books, notes, and Internet resources can be consulted, but not copied from.

Since everyone needs to write the final solutions alone, there is absolutely no need to lend your homework solutions and/or source codes to your classmates at any time. In order to maximize the level of fairness in this class, lending and borrowing homework solutions are both regarded as dishonest behaviors and will be punished according to the honesty policy.

Both English and Traditional Chinese are allowed for writing any part of your homework (if the compiler recognizes Traditional Chinese, of course). We do not accept any other languages. As for coding, either C or C++ or a mixture of them is allowed.

This homework set comes with 200 points and 20 bonus points. In general, every homework set of ours would come with a full credit of 200 points.

1.1 More from the Class

(1) (10%) The five criteria of algorithm come from Professor Donald Knuth, who wrote a famous series of book *The Art of Computer Programming* that is a must-read in Computer Science. Write down a short paragraph ($\leq$ 15 sentences) to introduce either Professor Knuth or the book in a way that a usual high school student would understand.

(2) (10%) Prove that the following algorithm returns the length of given string:

```
STRLEN(character array str)
i ← 0
while str[i] is not 'end of string' do
  i ← i + 1
end while
return i
```

1.2 Searching for Greatest Common Divisor

The greatest common divisor $gcd(a, b)$ between two positive integers a and b is defined as the largest positive integer that divides both a and b without a remainder. Mathematically speaking, for any integer $k > gcd(a, b)$, $(a \bmod k) \neq 0$ or $(b \bmod k) \neq 0$. In the following problems, we translate several properties of the *gcd* function to corresponding algorithms. In particular, we will pay some respect to Euclid's algorithm, which is arguably one of the earliest algorithms ever.

(1) (10%) Prove that the following GCD-By-Reverse-Search algorithm correctly returns the *gcd* for positive integers a, b using the mathematical definition of *gcd*.

```
GCD-By-Reverse-Search(positive integer a, positive integer b)
for $i \leftarrow \min(a, b)$ to 1 do
  if $a \bmod i = 0$ and $b \bmod i = 0$ then
    return $i$
  end if
end for
```

(2) (10%) Assume that $a > b$. What is the minimum number of iterations that GCD-By-Reverse-Search needs (for a fixed a)? When does the situation happen? *(The situation is usually called the best case.)*

1.3 Filtering for Greatest Common Divisor

Filtering algorithms try to extract small factors within the *gcd* using the following property.

(1) (10%) If any positive integer ℓ divides both a and b, prove that $gcd(a, b) = \ell \cdot gcd(a/\ell, b/\ell)$ using only the mathematical definition of *gcd*.

(2) (10%) Dr. Filter uses the definition above to design the following algorithm for *gcd*. Note that GCD-By-Filter is not an algorithm yet because there is a $\square$ in the last line, which violates the *definiteness* property of an algorithm. What should $\square$ be? Briefly argue that your choice of $\square$ makes GCD-By-Filter correct for positive integers a, b.

```
GCD-By-Filter(integer a, integer b)
for $i \leftarrow 2$ to $\min(a, b)$ do
  if $a \bmod i = 0$ and $b \bmod i = 0$ then
    return $i *$ GCD-By-Filter($a/i$, $b/i$)
  end if
end for
return $\square$
```

(3) (10%) Dr. FastFilter wants to speed up GCD-By-Filter by adding a parameter s that limits the range of i so smaller integers that have been tested for division will not be tested again. Complete the algorithm GCD-By-Filter-Faster below by filling in the $\triangle$ part. Briefly argue that your choice of $\triangle$ makes GCD-By-Filter-Faster correct for positive integers a, b.

```
GCD-By-Filter-Faster(integer a, integer b)
return GCD-By-Filter-Faster-Internal(a, b, $\triangle$)

GCD-By-Filter-Faster-Internal(integer a, integer b, integer s)
for $i \leftarrow s$ to $\min(a, b)$ do
  if $a \bmod i = 0$ and $b \bmod i = 0$ then
    return $i *$ GCD-By-Filter-Faster-Internal($a/i$, $b/i$, $i$)
  end if
end for
return $\square$
```

1.4 Binary Algorithm for GCD

Well, as a CSIE student, you shall not be satisfied with the efficiency of the algorithms mentioned above. Let's study another faster algorithm. The algorithm, called Binary Algorithm for GCD. "Binary", you say. Yes, binary is a magic word in computer science. Unlike the previous two algorithms, the Binary Algorithm needs only subtraction and division by 2 to work.

```
GCD-By-Binary(positive integer a, positive integer b)
$n \leftarrow \min(a, b)$, $m \leftarrow \max(a, b)$, $ans \leftarrow 1$
while $n \neq 0$ and $m \neq 0$ do
  if $n$ is even and $m$ is even then
    $ans \leftarrow ans \times 2$
    $n \leftarrow n/2$
    $m \leftarrow m/2$
  else if $n$ is even and $m$ is odd then
    $n \leftarrow n/2$
  else if $n$ is odd and $m$ is even then
    $m \leftarrow m/2$
  end if
  if $n > m$ then
    swap($n$, $m$)
  end if
  $m \leftarrow (m - n)$
end while
return $n \times ans$
```

(1) (10%) Write down the values of n, m and ans in the end of each iteration when $a = 56$ and $b = 14$.

(2) (10%) Prove that GCD-By-Binary satisfies the *finiteness* property. That is, the **while** only runs for a finite number of iterations for any given pair of positive integers (a, b).

(3) (10%) Assume that $a > b$. Please prove that $gcd(a, b) = gcd(a - b, b)$. You can see that the Binary Algorithm uses this somewhere.

(4) (Bonus 10%) Assume that $a > b$. What is the maximum number of iterations that GCD-By-Binary needs (for a fixed a)? Provide the tightest upper bound on the maximum number that you can think of, and prove your answer.

1.5 Euclid's Algorithm for Greatest Common Divisor

Now, let's take a look at Euclid's algorithm. Euclid's algorithm, as you probably learned in high school, is as follows.

```
GCD-By-Euclid(positive integer a, positive integer b)
$n \leftarrow \min(a, b)$, $m \leftarrow \max(a, b)$
while $m \bmod n \neq 0$ do
  $tmp \leftarrow n$
  $n \leftarrow m \bmod n$
  $m \leftarrow tmp$
end while
return $n$
```

(1) (10%) Write down the values of m, n and tmp in the end of each iteration when $a = 56$ and $b = 14$.

(2) (10%) Assume that it takes T iterations of **while** to run GCD-By-Euclid(a, b) for some positive integers a and b. How many iterations does it take to run GCD-By-Euclid($3a$, $3b$)? Formally prove your result.

(3) (Bonus 10%) Assume that $a > b$. If applying GCD-By-Euclid(a, b) takes T iterations, prove that $a \geq F_{T+2}$ and $b \geq F_{T+1}$. (The Fibonacci series $F_1 = F_2 = 1, F_i = F_{i-1} + F_{i-2}$ for $i \geq 3$.)

1.6 Comparison of GCD

In all the implementations below, your code should be able to work correctly for any positive integers a and b.

(1) (30%) Implement GCD-By-Reverse-Search, GCD-By-Filter, GCD-By-Filter-Faster, GCD-By-Binary and GCD-By-Euclid in one single file `hw1_6.c` or `hw1_6.cpp`. The code should take two 4-byte integers a and b from the standard input per line until reaching $a = 0$. Then, the code should output the *gcd* of a and b computed by each algorithm and the **number of iterations** the algorithm takes.

Please feel free to copy/use/modify the GCD-By-Binary or GCD-By-Euclid programs for Problem 1.7 for this problem.

Sample Input

```
3 2
4 3
0
```

Sample Output *(Note that the AAA, BBB, ··· should be filled by the actual numbers.)*

```
Case (3, 2): GCD-By-Reverse-Search = 1, taking AAA iterations
Case (3, 2): GCD-By-Filter = 1, taking BBB iterations
Case (3, 2): GCD-By-Filter-Faster = 1, taking CCC iterations
Case (3, 2): GCD-By-Binary = 1, taking DDD iterations
Case (3, 2): GCD-By-Euclid = 1, taking EEE iterations
Case (4, 3): ...
```

(2) (10%) Consider $a = 11260$ and $b \in \{52000, 52001, 52002, \cdots, 54260\}$. Compute the average number of iterations (of **for** or **while**) that each of GCD-By-Reverse-Search, GCD-By-Filter, GCD-By-Filter-Faster, GCD-By-Binary and GCD-By-Euclid takes to compute *gcd*(a, b). Briefly state your findings.

1.7 GCD of Big Integers

Sometimes one `int` variable cannot represent the integer we want. Even if we use an `unsigned` 4-byte integer, the maximum possible value is only $2^{32} - 1$. So we may need a data *structure*, such as an integer *array* to represent larger values. We will call it `BigInteger`. For instance, you can use an integer array where each element represents one digit under some base, like representing 1126 under base 100 by

```
int digits[10]={26,11};
```

There are other choices, of course. Please implement a (non-negative) BigInteger data structure to accompany with the two algorithms below.

(1) (20%) Complete the GCD-By-Binary algorithm implemented in `gcd_by_binary.cpp` by adding the necessary data description to `biginteger_for_binary.h` and writing a new file `biginteger_for_binary.cpp`. Please feel free to add any other functions necessary, but you are **NOT** allowed to modify `gcd_by_binary.cpp`. Your code should allow calculating the *gcd* of two 256-decimal-digit integers A and B from the standard input.

(2) (20%) Complete the GCD-By-Euclid algorithm implemented in `gcd_by_euclid.cpp` by adding the necessary data description to `biginteger_for_euclid.h` and writing a new file `biginteger_for_euclid.cpp`. Please feel free to add any other functions necessary, but you are **NOT** allowed to modify `gcd_by_euclid.cpp`. Your code should allow calculating the *gcd* of two 256-decimal-digit integers A and B from the standard input.

Submission File

Please submit your written part of the homework on all problems together to R102/R104 before the deadline. For the coding part, the TAs will announce a submission website later. Please follow the guidelines of the announcement. Note that the TAs will use the `Makefile` provided on the course website to test your code. **Please make sure that your code can be compiled and run with the Makefile on CSIE R217 linux machines. Otherwise your program "fails" its most basic test and can result in ZERO!**