Lecture 11/Lab 4 IST 230

— Some Basic Ideas you learned in grade school

— Given 2 positive integers x, y

1) Greatest Common Divisor (gcd) is the largest positive integer that divides both x and y.

Ex: $\gcd(15, 6) = 3$; $\gcd(28, 14) = 14$

2) Least Common Multiple (lcm) is the smallest positive integer which is divisible by both x and y.

Ex: $\mathrm{lcm}(15, 6) = 30$ since $30/15 = 0$ (Rem 0) $30/6 = 5$ (Rem 0) and No smaller value works

Try It!

Ex: $\mathrm{lcm}(28, 14) = 28$
Since $28/28 = 1$ and $28/14 = 2$
(certainly, no integer < 28 Divides 28)

Fact: Usually harder to compute lcm than gcd.

Fact: For any positive integers m, n
$\mathrm{lcm}(x, y) * \gcd(x, y) = x * y$
(i.e., if you know gcd, can compute lcm easily).

<u>Euclid</u>'s <u>Algorithm</u> to Compute $gcd(x, y)$:

→ Without loss of generality, Assume $x > y$

→ Define the sequence of <u>division</u> <u>algorithm</u> iterations

$$(1) \quad x = q_1 y + r_1 \longrightarrow \frac{x}{y} = q_1 \text{ Remainder } r_1$$

<u>Quotient</u> <u>Remainder</u>

$$(2) \quad y = q_2 r_1 + r_2 \longrightarrow \frac{y}{r_1} = q_2 \text{ Remainder } r_2$$

$$(3) \quad r_1 = q_3 r_2 + r_3 \longrightarrow \frac{r_1}{r_2} = q_3 \text{ Remainder } r_3$$

$$(4) \quad r_2 = q_4 r_3 + r_4 \longrightarrow \frac{r_2}{r_3} = q_4 \text{ Remainder } r_4$$

$$\vdots$$

$$(N-1) \quad r_{N-2} = q_{N-1} r_{N-1} + r_N \longrightarrow \frac{r_{N-2}}{r_{N-3}} = q_{N-1} \text{ rem } r_N$$

$$(N) \quad r_{N-1} = q_N r_N$$

where r_N is the <u>last</u> <u>non-zero</u> <u>remainder</u>.

<u>Then</u>, $gcd(x, y) = r_N$.

Euclid's Algorithm in Words:

(1) $\dfrac{\text{Dividend } (x)}{\text{Divisor } (y)} \longrightarrow$ Quotient(q), remainder(r)

(2) If Remainder $= 0$, STOP $\left(\begin{array}{l} \gcd(x, y) = \\ \text{last nonzero remainder} \end{array} \right)$

(3) Toss Quotient

(4) Move Divisor to Dividend

(5) Move Remainder to Divisor

(6) Start at (1) again.

Ex: $\gcd(87, 18)$

(1) $\dfrac{87}{18} = 4$ Rem 15

(2) $\dfrac{18}{15} = 1$ Rem 3

(3) $\dfrac{15}{3} = 5$ Rem 0

The last non-zero remainder was 3

Hence, $\gcd(87, 18) = 3$.

Questions

- Does Euclid's Alg always work
  → yes
- How fast does it converge to gcd (x, y)
  → Every other iteration, the value of dividend reduced by a factor of at least 2.
  (Converges Exponentially fast
  → That's fast)

Euclid's Algorithm can be broken down into Several Steps - Essential to create computer algorithm

1. Declare Variables & initialize

Dividend x } Need to initialize these
Divisor y } (Input Data)

Quotient q } These are computed values
Remainder r } (Output Data)

2. Divide x by y, Toss Quotient, giving Remainder.
→ Modulo Division operation
(will Discuss Monday)

In C, C++, Java, represented by

$r = x \% y;$ { In Math-ese, written $x \bmod y$ }

3. Test to see if $r > 0$
(If it is not, we're done → write gcd(x, y) as the last value of x computed

4. Move Divisor to Dividend

$x = y;$

5. Move Remainder to Divisor

$y = r;$

6. Go back to Step 2.

A simple Java Applet to do Euclid's Algorithm

→ Go to ..../ist230/labs/lab4/GCD_Simple directory, Download

GCD.java
GCD.html

into C:\Temp\Temporary Internet Files
(Don't seem to have much choice here unless you have a floppy)

→ Now create a new directory

C:\Temp\GCD

and copy these 2 files into this directory.

→ Start up any text editor/Java IDE you like

(I like JPad, in the Programming subdirectory of Start Menu/Programs; You can compile and test your applet using the Sun JDK within JPad - It's very nice and Simple)

Note that the program GCD.java is not completed:

→ The entire program is embedded in a single class GCD (which extends the Applet class)

The methods

init(), action(), takestep()

deal strictly with initializing the applet and x, y values, and provide a rudimentary graphic interface

→ We don't touch these.

The method

gcd()

Contains all the computations

→ Note the variables named

x_1, y_1, r_1

(Why don't we need q_1?)

:

OUTPUT is an Integer

x₁, y₁ are Integers

```
int gcd (int x1, int y1) {
  int r1 // Declare remainder as integer
  while ( ? > 0 ) {
  — you fill in code here —
  }
  return x1 // Send back last
  }
}
```

while loop

The while loop causes the code you fill in here to continue to execute until ? > 0