Administrivia

• Please make sure you have obtained a Unix account.

• If you decide not to take this course after all, please tell CalCentral ASAP, so that we can adjust the waiting list accordingly.

• HW #0 will be due next Friday at midnight. While you get credit for any submission, we strongly suggest that you give the problems a serious try.

• We strongly discourage taking this course P/NP (or S/U).
Lecture #2: Let's Write a Program: Prime Numbers

Problem: want java Primes $U$ to print prime numbers through $U$.
  You type: java Primes 101
  It types: 2 3 5 7 11 13 17 19 23 29
            31 37 41 43 47 53 59 61 67 71
            73 79 83 89 97 101

Definition: A prime number is an integer greater than 1 that has no
divisors smaller than itself other than 1. (Alternatively: $p > 1$ is prime iff $\gcd(p, x) = 1$ for all $0 < x < p$.)
public class Primes {
   /** Print all primes up to ARGVS[0] (interpreted as an
    * integer), 10 to a line. */
   public static void main(String[] args) {
       printPrimes(Integer.parseInt(args[0]));
   }

   /** Print all primes up to and including LIMIT, 10 to
    * a line. */
   private static void printPrimes(int limit) {
       /*{ For every integer, x, between 2 and LIMIT, print it if
            isPrime(x), 10 to a line. }*/
   }

   /** True iff X is prime. */
   private static boolean isPrime(int x) {
       return /*( X is prime )*/;
   }
}
Testing for Primes

private static boolean isPrime(int x) {
    return /*( X is prime )*/;
}


private static boolean isPrime(int x) {
    if (x <= 1)
        return false;
    else
        return !isDivisible(x, 2, x); // "!" means "not"
}
Testing for Primes

private static boolean isPrime(int x) {
    if (x <= 1)
        return false;
    else
        return !isDivisible(x, 2, x);  // "!" means "not"
}

/** True iff X is divisible by any positive number >= LOW >= 1
 * and < HIGH. */
private static boolean isDivisible(int x, int low, int high) {
    return /*( True iff x is divisible by k, low<=k<high. )*/;
}
private static boolean isPrime(int x) {
    if (x <= 1)
        return false;
    else
        return !isDivisible(x, 2, x);  // "!" means "not"
}

/** True iff X is divisible by any positive number >= LOW >= 1 
 * and < HIGH. */
private static boolean isDivisible(int x, int low, int high) {
    if (low >= high)  // a "guard"
        return false;
    else if (x % low == 0)  // "%" means "remainder"
        return true;
    else
        return isDivisible(x, low, high);
}
Thinking Recursively

Understand and check $\text{isDivisible}(13, 2)$ by *tracing one level.*

/** True iff $X$ is divisible by some number
  * $\geq \text{LOW} \geq 1$ and $< \text{HIGH}$. */
private static boolean isDivisible...
  if (low >= high)
    return false;
  else if (x % low == 0)
    return true;
  else
    return isDivisible(x, low + 1, high);
}

Lesson: Comments aid understanding. Make them *count!*

• Call assigns $x=13$, $\text{low}=2$, $\text{high}=13$
• Body has form
  
  if (low $\geq$ high) $S_1$ else $S_2$.

• Since $2 < 13$, we evaluate the (first) else.
• Check if $13 \mod 2 = 0$; it's not.
• Left with $\text{isDivisible}(13, 3, 13)$.
• Rather than tracing it, instead *use the comment:*

  • Since $13$ is *not* divisible by any integer in the range $3..12$, $\text{isDivisible}(13, 3, 13)$ must be *false*, and we're done!

• Sounds like that last step begs the question. Why doesn’t it?
Iteration

- **isDivisible** is *tail recursive*, and so creates an *iterative process*.
- Traditional “Algol family” production languages have special syntax for iteration. Four equivalent versions of **isDivisible**:

  ```plaintext
  if (low >= high)
      return false;
  else if (x % low == 0)
      return true;
  else
      return isDivisible(x, low+1, high);
  ```

  ```plaintext
  while (low < high) { // !(low >= high)
      if (x % low == 0)
          return true;
      low = low+1; // or low += 1, or (yuch) low++
  }
  return false;
  ```

  ```plaintext
  int k = low;
  while (k < high) {
      if (x % k == 0)
          return true;
      k += 1;
  }
  return false;
  ```

  ```plaintext
  for (int k = low; k < high; k += 1) {
      if (x % k == 0)
          return true;
  }
  return false;
  ```
Using Facts about Primes

• A couple of obvious facts:
  - \( k \leq \sqrt{N} \) iff \( N/k \geq \sqrt{N} \), for \( N, k > 0 \).
  - If \( k \) divides \( N \) then \( N/k \) divides \( N \).

• So how far do we really have to go to find a possible divisor for \( x \)?
Using Facts about Primes

• A couple of obvious facts:
  - \( k \leq \sqrt{N} \) iff \( N/k \geq \sqrt{N} \), for \( N, k > 0 \).
  - If \( k \) divides \( N \) then \( N/k \) divides \( N \).

• So how far do we really have to go to find a possible divisor for \( x \)?
  Only up to and including \( \sqrt{x} \).
Using Facts about Primes

- A couple of obvious facts:
  - \( k \leq \sqrt{N} \) iff \( N/k \geq \sqrt{N} \), for \( N, k > 0 \).
  - If \( k \) divides \( N \) then \( N/k \) divides \( N \).

- So how far do we really have to go to find a possible divisor for \( x \)?
  "Only up to and including \( \sqrt{x} \)."
- So, reimplement \texttt{isPrime}:

```java
private static boolean isPrime(int x) {
  if (x <= 1)
    return false;
  else
    return !isDivisible(x, 2, (int) Math.round(Math.sqrt(x)));
  // (int) ... here converts to an integer in the range
  // \(-2^31..2^31 - 1\) (type ‘int’) from one in the
  // range \(-2^63..2^63 - 1\) (type ‘long’).
}
```
Cautionary Aside: Floating Point

• In the last slide, we used

\[(\text{int}) \ \text{Math.round}(\text{Math.sqrt}(x))\];

intending that this would check all values of \(k\) up to and including the square root of \(x\).

• Since floating-point operations yield \textit{approximations} to the corresponding mathematical operations, you might ask the following about \texttt{Math.round(Math.sqrt(x))}:

  - Is it always at least \(\lfloor \sqrt{x} \rfloor\)? (\(\lfloor z \rfloor\) means “the largest integer \(\leq z\).”) If not, we might miss testing \(\sqrt{x}\) when \(x\) is a perfect square.

• As it happens, the answer is “yes” for IEEE floating-point square roots.

• Just an example of the sort of detail that must be checked in edge cases.
Final Task: printPrimes (Simplified)

/** Print all primes up to and including LIMIT. */
private static void printPrimes(int limit) {

}
Simplified printPrimes Solution

/** Print all primes up to and including LIMIT. */
private static void printPrimes(int limit) {
    for (int p = 2; p <= limit; p += 1) {
        if (isPrime(p)) {
            System.out.print(p + " ");
        }
    }
    System.out.println();
}
/** Print all primes up to and including LIMIT, 10 to * a line. */

private static void printPrimes(int limit) {
    int np;
    np = 0;
    for (int p = 2; p <= limit; p += 1) {
        if (isPrime(p)) {
            System.out.print(p + " ");
            np += 1;
            if (np % 10 == 0)
                System.out.println();
        }
    }
    if (np % 10 != 0)
        System.out.println();
}