

CS61B Lecture #2

- **Register!** Be sure to *get an account* and register (as in the first lab) by Friday, even if you are on the waiting list or seeking concurrent enrollment. Otherwise, you will be dropped!
- On waiting list only because of scheduling? Relax; we'll handle it.
- **Readers** available at Vick Copy, as indicated in the first handout (*not* Copy Central).
- **Lab 0** may be submitted any time before Friday midnight.
- **Unix Help Session** on Thursday at 6:30pm in 306 Soda, courtesy of the CSUA (Computer Science Undergraduate Association). Topics: basic Unix commands, X-windows, mail, and remote access. CSUA membership forms will be available.

Readings for Friday: *PIJ* 1.1-1.10, Chapter 4.

Observations about Labs

- Ask more questions, and do it earlier!
 - No penalty for stupid questions.
 - Most errors and glitches are minor; don't spend hours on them!
 - If you don't make progress for >10 minutes, time to get help.
- OK to collaborate on labs and ordinary homework to any extent.
- Important to use the tools (Emacs, make, gjdb, prcs).
 - Are examples of features any good PDE will have
 - Once learned, *really* cut down on tedium
 - Read the *Tools Documentation*.

Testing for primes

(Held over from last time)

```
/** True iff X is prime. */
static boolean isPrime (int x) {
    return x == 2 || /* || is "or", && is "and", ! is "not", */
        (x > 1 && /* (int) x is "truncate x to integer" */
         ! isDivisible (x, 2, 1 + (int) Math.sqrt(x)));
}

/** True iff X is divisible by any integer in range L .. U. */
static boolean isDivisible (int x, int L, int U) {
    if (L > U)
        return false;
    else
        /* % is "remainder" */
        return x % L == 0 || isDivisible (x, L+1, U);
}
```

Tracing the Code

Understand `isDivisible` by *tracing one level*. E.g., `isDivisible (13, 2, 4)`

- Call assigns `x=13, L=2, U=4`
- Body has form `'if (L>U) St else Sf'`.
 - Since $2 \leq 4$, return `13 % 2 == 0 || isDivisible(13,3,4)`
 - `13 % 2` is 1, so reduces to `isDivisible(13,3,4)`.
 - Now use the *comment*: since 13 is *not* divisible by any integer in the range 3..4, must be **false**.
 - We're done!
- Lesson: Comments aid understanding. Make them count!

Iteration

- `isDivisible` is *tail recursive*, and so creates an *iterative process*.
- Traditional "Algol family" production languages have special syntax for iteration.

```
static boolean isDivisible (int x, int L, int U) {
    while (L <= U) {
        if (x % L == 0)
            return true;
        L = L+1;    // or 'L += 1' or 'L++' (yuch).
    }
    return false;
}
```

Iteration: 'for' Loops

Let's do that loop again, slightly changed:

```
int y = L;           // Initialize
while (y <= U) {    // Test
    if (x % y == 0)
        return true;
    y += 1;         // Increment
}
return false;
```

That's a very common pattern, so there's a syntax for it:

```
for (int y = L; y <= U; y += 1)
    if (x % y == 0)
        return true;
return false;
```

More Iteration: Sort an Array

Problem. Print out the command-line arguments in order:

```
% java sort the quick brown fox jumped over the lazy dog
brown dog fox jumped lazy over quick the the
```

Plan.

```
class sort {
    public static void main (String[] words) {
        sort (words, 0, words.length-1);
        print (words);
    }

    /** Sort items A[L..U], with all others unchanged. */
    static void sort (String[] A, int L, int U) { /* TOMORROW */ }

    /** Print A on one line, separated by blanks. */
    static void print (String[] A) { /* TOMORROW */ }
}
```

Selection Sort

```
/** Sort items A[L..U], with all others unchanged. */
static void sort (String[] A, int L, int U) {
    if (L < U) {
        int k;
        k = Value s.t. A[k] is largest in A[L], ..., A[U];
        swap A[k] with A[U]
        Sort items L to U-1 of A.
    }
}
```

So, how do we do that last step?

Sorting the rest

```
/** Sort items A[L..U], with all others unchanged. */
static void sort (String[] A, int L, int U) {
    if (L < U) {
        int k;
        k = Value s.t. A[k] is largest in A[L], ..., A[U];
        swap A[k] with A[U]
        sort (A, L, U-1);
    }
}
```

- Why do we know that works?
- How do we swap?

Swapping

```
/** Sort items A[L..U], with all others unchanged. */
static void sort (String[] A, int L, int U) {
    if (L < U) {
        int k;
        k = Value s.t. A[k] is largest in A[L], ..., A[U];
        String tmp = A[k];
        A[k] = A[U]; A[U] = tmp;
        sort (A, L, U-1);
    }
}
```

- Why do we need tmp?
- Now the rest is easy:

Find Largest

```
/** Sort items A[L..U], with all others unchanged. */
static void sort (String[] A, int L, int U) {
    if (L < U) {
        int k;
        k = indexOfLargest (A, L, U);
        String tmp = A[k];
        A[k] = A[U]; A[U] = tmp;
        sort (A, L, U-1);
    }
}
```

Iterative Version of 'sort'

```
/** Sort items A[L..U], with all others unchanged. */
static void sort (String[] A, int L, int U) {
    while (L < U) {
        int k;
        k = indexOfLargest (A, L, U);
        String tmp = A[k];
        A[k] = A[U]; A[U] = tmp;
        U -= 1;
    }
}
```

- OK, so we aren't quite done yet.

Really Find Largest

```
/** Index, I0<=k<=I1, such that V[k] is largest element among
 * V[I0], ... V[I1]. Requires I0<=I1. */
static int indexOfLargest (String[] V, int i0, int i1) {
    if (i0 >= i1)
        return i1;
    else /* if (i0 < i1) */ {
        int k = indexOfLargest (V, i0+1, i1);
        return (V[i0].compareTo (V[k]) > 0) ? i0 : k;
    }
}
```

If you prefer, last return is equivalent to

```
if (V[i0].compareTo (V[k]) > 0)
    return i0;
else
    return k;
```

Last modified: Wed Aug 28 02:32:23 2002

CS61B: Lecture #2 13

Or the Iterative Version

```
/** Index, I0<=k<=I1, such that V[k] is largest element among
 * V[I0], ... V[I1]. Requires I0<=I1. */
static int indexOfLargest (String[] V, int i0, int i1) {
    int i, k;
    k = i1; // Deepest iteration
    for (i = i1-1; i >= i0; i -= 1)
        k = (V[i].compareTo (V[k]) > 0) ? i : k;
    return k;
}
```

Last modified: Wed Aug 28 02:32:23 2002

CS61B: Lecture #2 14

Finally, Printing

```
/** Print A on one line, separated by blanks. */
static void print (String[] A) {
    for (int i = 0; i < A.length; i += 1)
        System.out.print (A[i] + " ");
    System.out.println ();
}
```

Last modified: Wed Aug 28 02:32:23 2002

CS61B: Lecture #2 15