CS61B Lecture #2

• Please make sure you have obtained an account, run ‘register’, and finished the survey today.

• In the future (next week), the password required for surveys and such will be your account password—the one you log in with.

• Discussion section room change:

Public Service Announcements:

• Want to find out why over 300 Berkeley students volunteer to tutor with OASES each year? Find out at their info session on 1/26, at 7:30 in 2050 VLSB. Or contact Kenneth at trinhk@gmail.com

• Upcoming events from the CSUA (details at http://csua.berkeley.edu):
  - 1/23: Intro to UNIX
  - 1/25: Mentoring session
  - 1/26: Intro to Emacs
Thinking Recursively

Understand and check \texttt{isDivisible(13,2)} by \textit{tracing one level}.

\begin{verbatim}
/** True iff X is divisible by
 * some number \( \geq K \) and < \( \text{X} \),
 * given \( K > 1 \). */
boolean isDivisible (int x, int k) {
    if (k >= x)
        return false;
    else if (x % k == 0)
        return true;
    else
        return isDivisible (x, k+1);
}
\end{verbatim}

\textbf{Lesson: Comments aid understanding. Make them count!}

\begin{itemize}
\item Call assigns \( x=13, \ k=2 \)
\item Body has form ‘if (\( k \geq x \)) \( S_1 \)
                    else \( S_2 \)’.
\item Since \( 2 < 13 \), we evaluate the first else.
\item Check if \( 13 \mod 2 = 0 \); it’s not.
\item Left with \( \text{isDivisible(13,3)} \).
\item Rather than tracing it, instead use the comment:
\item Since 13 is \textit{not} divisible by any integer in the range 3..12 (and 3 > 1), \text{isDivisible(13,3)} must be \textit{false}, and we’re done!
\item Sounds like that last step begs the question. Why doesn’t it?
\end{itemize}
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 (k >= x)
    return false;
else if (x % k == 0)
    return true;
else
    return isDivisible (x, k+1);
```

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

```plaintext
int k1 = k;
while (k1 < x) {
    if (x % k1 == 0)
        return true;
    k1 += 1;
}
return false;
```

```plaintext
for (int k1 = k; k1 < x; k1 += 1) {
    if (x % k1 == 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) {
        /*{ Sort items L to U-1 of A. }*/;
    }
}

Last modified: Fri Jan 20 14:26:32 2006
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 = indexOfLargest (A, L, U);
        /*{ Sort items L to U-1 of A. }*/;
    }
}

Last modified: Fri Jan 20 14:26:32 2006
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 = indexOfLargest (A, L, U);
        sort (A, L, U-1);      // Sort items L to U-1 of A
    }
}
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 = indexOfLargest (A, L, U);
        sort (A, L, U-1); // Sort items L to U-1 of A
    }
}
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 = indexOfLargest (A, L, U);
        sort (A, L, U-1); // Sort items L to U-1 of A
    }
}

Iterative version:

    while (L < U) {
        int k = indexOfLargest (A, L, U);
        U -= 1;
    }

And we’re done! Well, OK, not quite.
Really Find Largest

/** Value \(k\), \(I_0 \leq k \leq I_1\), such that \(V[k]\) is largest element among
* \(V[I_0]\), \ldots \(V[I_1]\). Requires \(I_0 \leq I_1\). */
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;
        // or if (V[i0].compareTo (V[k]) > 0) return i0; else return k;
    }
}

Iterative:

    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;
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 ();
}

/* Looking ahead: There’s a brand-new syntax for the for
 * loop here (as of J2SE 5): */
for (String s : A)
    System.out.print (s + " ");
/* Use it if you like, but let’s not stress over it yet! */