UNIVERSITY OF CALIFORNIA
Department of Electrical Engineering
and Computer Sciences
Computer Science Division

CS61B **M. Brudno**
Summer 2000

Homework 1

Due: Tuesday, June 27, 2000

Create a directory to hold your answers to this homework set. To submit your homework, use the command '`submit hw1`' from within that directory. Where the problems specify file names to use for your solution, use exactly those names. Place the answers to all other homework in a file named '`hw1.txt`' within the directory you've created. Template files for some of the questions are available from `cs61b/hw/hw1/`

1. Convert the following Scheme fragments into Java statements or (static) functions with the same results, where possible. Where integers are involved, ignore the fact that Java's integers have an upper bound. Assume that undefined terms in the expressions are defined elsewhere. Where it is not possible to produce the same effect, say why.

a.
```
(define (exp x y)
   (if (= y 0) 1
       (* x (exp x (- y 1)))))

; Give both a recursive and a
; non-recursive solution.
```

b.
```
(cond ((< x 0) (h0 (- x)))
      ((< x 10) (h1 x))
      ((< x 1000) (h2 x))
      (else (h3 x)))
```

c.
```
(define (smaller n)
    (if (> n 0) (- n 1)
        #f))
```

d.
```
(cond ((or (eq? c #\u)
           (eq? c #\U))
       (up))
      ((or (eq? c #\d)
           (eq? c #\D))
       (down))
      ((eq? c #\=) (drop))
      ((eq? c #\+) (shoot))
      (else (report-error)))
  ; Use a switch statement.
```

2. Give an example when the `for` loop on the left will not equivalent to the `while` loop on the right:

```
for(init; test; update)
  {
    statements
  }
morestatements
```

```
init;
while( test )
  {
    statements
    update;
  }
 morestatements;
```

3. Complete the following Java functions so that they perform as indicated in their comments. Remember that some arrays have no elements. Put your answers to this problem (all five parts) in a file named `Arrays.java`. You may find the contents of the file `cs61b/hw/hw1/ArrayUtil.java` to be useful in testing your answers.

a.
```
/** The largest value in the array A. */
static int maximum(int[] A) {
    /* *Fill in here* */
}
```

b.
```
/** Returns the mathematical average (mean)  of the elements of
  * the array A. */
static double mean(int[] A) {
    /* *Fill in here* */
}
```

c.
```
/** A new array with 1 more element than A, consisting of the elements
 *  of A except that A[w]==b and all elements at indices > w are
 *  located one index higher than previously. */
static int[] insert(int[] A, int b, int w) {
    /* *Fill in here* */
}
```

d.
```
/** The number of pairs i,j such that i < j and A[i] > A[j] */
static int inversions(int[] A) {
    /* *Fill in here* */
}
```

e.
```
/** The transpose of A: all rows become columns and all columns
  * become rows. */
static int[][] transpose(int[][] A) {
    /* *Fill in here* */
}
```

4. Write the following Java functions and submit them in the file `Fibs.java`

a.
```
/** Returns the nth fibonacci number without allocating any arrays or
 *  making any recursive calls. For uniformity, fib(0)==0, fib(1)==1 */
static int getFib(int n) {
    /* *Fill in here* */
}
```

b.
```
/** Returns the sum of the first n fibonacci numbers. Feel free to
  * use the method defined above */
static int sigmaFib(int n) {
    /* *Fill in here* */
}
```

5. Complete the following Java functions so that they perform as indicated in their comments. Put your answers to this problem in a file named `Strings.java`.

a.
```
/** The number of words in string S.  A "word" is a sequence of
 *  non-whitespace characters that is bounded on both sides by
 *  by whitespace or an end of S.  Whitespace characters are
 *  ' ', '\t', '\n', and '\r'.  */
static int numWords(String S) {
    /* *Fill in here* */
}
```

b.
```
/** The string S with all sequences of adjacent blanks replaced
 *  by a single blank.  Thus, squeeze("This  is     a test")
 *  is "This is a test".
 */
static String squeeze(String S) {
    /* *Fill in here* */
}
```

6. Write a class, `LetterStream`, that represents a sequence consisting of all of the characters of some string. Assume the declaration

```
LetterStream stream = new LetterStream(someString);
```

Then

`stream.letter()` returns the next letter of `someString`, if any.

`stream.count()` returns the number of times `stream.letter()` appears in the remaining part of someStringN. Letters of different cases are considered different.

`stream.next()` changes `letter()` and `count()` to the next letter.

`stream.empty()` tests whether all the letters have been delivered. It becomes true when you call `next()`, but there is no next value for `letter()`, and it is always true when `someString` is the empty string.

`stream.reset()` returns the stream to initial condition, with `stream.letter()` being the first letter of `someString`.

For example,

```
LetterStream stream = new LetterStream(``Foo!'');
// stream.letter() is now 'F' and stream.count() is 0
stream.next ();
// stream.letter() is now 'o' and stream.count() is 1
stream.next ();
// stream.letter() is now 'o' and stream.count() is 0
stream.next ();
// stream.letter() is now '!' and stream.count() is 0
stream.next ();
// stream.empty() is now true.
```

The `letter`, `count`, and `next` methods should throw the run-time exception `IllegalStateException` if called when `empty()` is true. Put your `LetterStream` class in file `LetterStream.java`.

Create also a class `Letters` (in file `Letters.java` that contains a main program that uses the `LetterStream` class to print only the first occurence of each letter. For this question the strings are restricted to the 128 ASCII characters. *Hint:* The previous sentence was a hint.
Note: no template is provided for `Letters.java`.

Thus,

```
java Letters milkmilk
```

is to print

```
milk
```

and,

```
java Letters mississippi
```

is to print

```
misip
```