CS61B, Fall 2008 **HW #1** **P. N. Hilfinger**

Homework Exercises. (Due Wed., 11 September, at midnight)

We have skeleton files for your solutions. Assuming you have set up `svnwork` and `staff` directories as indicated in the first lab, you can use this procedure on the Unix machines:

```
# Fetch any new stuff into your working copy of staff
% svn update ~/staff
% cd ~/svnwork
% svn copy ~/staff/hw1 hw1
% svn commit -m "Get initial HW1 skeleton"
```

Now you have a `trunk/hw1` directory in the repository, with a copy in your Subversion working directory (`~/svnwork`). Modify the files there, commit them (frequently as you are working), and then hand them in with

```
% cd ~/svnwork
% svn commit -m "Make sure HW1 work is committed"
% svn copy hw1 svn+ssh://cs61b-ta@quasar.cs.berkeley.edu/cs61b-yu/tags/hw1-1 \
>     -m 'Submit HW1'
```

Place your answers to all these problems in the file named `Progs.java`. You will also find a program, `HW1Test.java` in the directory, which you should fill in so that `java HW1Test` tests your functions.

1. Complete the following Java functions so that they perform as indicated in their comments. You may add any additional auxiliary functions and definitions you want. Definitions: A *sociable pair* consists of two different positive integers such that the sum of all distinct divisors of either one of them is equal to the other. For example, 220 is evenly divisible by 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, and 110, which add to 284. 284 is divisible by 1, 2, 4, 71, and 142, which add up to 220.

a.
```
/** The sum of all integers, k, such that 1 <= k < N and
 *  N is evenly divisible by k. */
static int factorSum (int N) {
    /* *Fill in here* */
}
```

b.
```
/** Print the set of all sociable pairs whose members are all
 *  between 1 and N>=0 (inclusive) on the standard output (one pair per
 *  line, smallest member of each pair first, with no repetitions). */
static void printSociablePairs (int N) {
     /* *Fill in here* */
}
```

2. Complete the following Java functions so that they perform as indicated in their comments. You may add any additional auxiliary functions and definitions you want.

a.
```
/** A list consisting of the elements of A followed by the
 *  the elements of B.  May modify items of A.
 *  Don't use 'new'. */
static IntList dcatenate(IntList A, IntList B) {
    /* *Fill in here* */
}
```

continued

b.
```
/** The sublist consisting of LEN items from list L,
 *  beginning with item #START (where the first item is #0).
 *  Does not modify the original list elements.
 *  It is an error if the desired items don't exist. */
static IntList sublist (IntList L, int start, int len) {
    /* *Fill in here* */
}
```

c.
```
/** The sublist consisting of LEN items from list L,
 *  beginning with item #START (where the first item is #0).
 *  May modify the original list elements. Don't use 'new'.
 *  It is an error if the desired items don't exist. */
static IntList dsublist (IntList L, int start, int len) {
    /* *Fill in here* */
}
```