CS 98/198, Fall 2003 Turning in Programs Electronically P. N. Hilfinger

To set up your account, execute

```
source ~ctest/bin/setup
```

in all shells that you are using. (This is for those of you using the C-shell. Others will have to examine this file and do the equivalent for their shells.)

Put each complete C solution into a file *N*`.c`, each complete C++ solution into a file *N*`.cc`, each complete Java program into a file *N*`.java`, and each complete Python program into a file *N*`.py`, where *N* is the number of the problem. Each program must reside entirely in a single file. In Java, the class containing the main program for problem *N* must be named `P`*N*. Each C/C++ file should start with the line

```
#include "contest.h"
```

and must contain no other `#include` directives, except as indicated below. Upon completion, each C/C++ program *must* terminate by calling `exit(0)` and each Java program must terminate with `System.exit(0)`. Python programs should also be sure to exit with a normal (0) return code.

When you have a solution to problem number *N* that you wish to submit, use the command

```
submit N
```

from the directory containing *N*`.c`, *N*`.cc`, *N*`.java`, or *N*`.py`. Before actually submitting your program, `submit` will first compile it and run it on one sample input file. No submission that is sent after the end of the contest will count. You should be aware that `submit` takes some time before it actually sends a program. In an emergency, you can use

```
submit -f N
```

which submits problem *N* without any checks.

All tests will use the compilation command

```
contest-gcc N
```

followed by one or more execution tests of the form (Bourne shell):

```
./N < test-input-file > test-output-file 2> junk-file
```

which sends normal output to *test-output-file* and error output to *junk-file.* The output from running each input file is then compared with a standard output file, or tested by a program in cases where the output is not unique. In this comparison, leading and trailing blanks are ignored and sequences of blanks are compressed to single blanks. Otherwise, the comparison is literal; be sure to follow the output formats *exactly.* It will do no good to argue about how trivially your program's output differs from what is expected; you'd be arguing with a program. Make sure that the last line of output ends with a newline.

Your program must not send any output to `stderr`; that is, the temporary file *junk-file* must be empty at the end of execution. Each test is subject to a time limit of about 45 seconds (unless otherwise stated). You will (eventually) be advised by mail whether your submissions pass.

The command `contest-gcc` *N*, where *N* is the number of a problem, is available to you for developing and testing your solutions. For C and C++ programs, it is equivalent to

```
gcc -Wall -o N -O -g -Iour-includes N.* -lstdc++ -lm
```

For Java programs, it is equivalent to

```
javac -g N N.java
```

followed by a command that creates an executable file called *N* that runs the command

```
java PN
```

when executed (so that it makes the execution of Java programs look the same as execution of C/C++ programs). For Python programs, it simply copies *N*`.py` into a file *N* with an appropriate "`#!`" line at the beginning (it's harmless if you already have one) and marks the file executable. The *our-includes* directory contains `contest.h` for C/C++, which also supplies the standard header files. The files in `~ctest/submission-tests/`*N*, where *N* is a problem number, contain the input files and standard output files that `submit` uses for its simple tests.

All input will be placed in `stdin` (regardless of the what the problem statements, generally taken from assorted contests, might say). You may assume that the input conforms to any restrictions in the problem statement; you need not check the input for correctness. Consequently, you are free to use `scanf` to read in numbers and strings and `gets` to read in lines.

Terminology. The term *free-form input* indicates that input numbers, words, or tokens are separated from each other by arbitrary whitespace characters. By standard C/UNIX convention, a whitespace character is a space, tab, return, newline, formfeed, or vertical tab character. A *word* or *token,* accordingly, is a sequence of non-whitespace characters delimited on each side by either whitespace or the beginning or end of the input file.