Contest Rules

Duke Internet Programming Contest

November 18, 1993

1 Concept

The idea of this contest is to enable many people to participate in a programming contest. We hope it will be fun for all involved, as well as boosting interest in the programming teams involved in the ACM programming contests. This contest is very informal, and we hope that the contestants will compete in this spirit. The success of the contest depends on the integrity of the contestants.

You should also have received a copy of the guidelines describing the details of how to participate in the contest.

This document consists of 4 pages.

2 Teams

Teams consist of *up to* four people. There are three classes of teams:

I Novice programmers.

II Expert programmers participating as an individual.

III Expert programmers with more than one person on a team.

When you register your team at the start of the contest, you decide which category in which you wish to participate. The novice category will contain some extra problems that should be easier to solve, and the expert categories will contain extra problems which are more challenging. There will be a set of problems that are common to both categories, as well.

3 Contest specifics

1. We will provide a set of problems to be solved. There will be 3–10 problems in the set. These will be provided to a local contest administrator.

2. The contest lasts three hours. The judges have the power to extend the contest if necessary.

3. Each team may use only one terminal (or workstation). You may use any type of computer you wish, but you must use our software to submit your solutions.

4. We will judge programs using the compilers or interpreters as listed in the Table 1. A † indicates that the language is interpreted rather than compiled. See Section 6 for additional information on the specific languages.

Language	Compiler	Version	Libraries
C	gcc	2.5.2	standard C, -lm
GNU C++	gcc	2.5.2	standard C, -lg++ -lm
Sun C++	CC (cfront)	2.1	default and -lm
Objective-C	gcc	2.5.2	standard C, -lobjc -lm
Pascal	Sun Pascal	2.1	default
FORTRAN	Sun FORTRAN	1.4	default
Scheme (Revised^4)	scm†	4d0	slib 1d5
(Literate) Haskell	LML hbc	0.999.5	default
Perl	perl†	4.036	default
Icon	icont	8.10	IPL
Emacs Lisp	GNU Emacs†	19.19	default
Standard ML	SML/NJ	0.93	smlnj-lib 0.2
Matlab	matlab†	4.0a	default
Common Lisp	CMU Lisp	17c	minimal
Ada	Meridian OpenAda	4.1.3	default

Table 1: List of languages.

5. You may use any editor, debugger, and windowing system. If your program does not compile or run on our SPARCstation running SunOS 4.1.3, that is unfortunate for you.

6. No arguments will be passed to the program at run time.

7. You may work on the problems in any order, and submit them in any order.

8. You may use any books and other materials you wish, except electronic copies of pre-existing code. You may not use any code that is on-line. This rule means you must type in all code you hand in. You may not use any program that generates code in the language which you submit the program for judging. For example, if you are using Haskell as your programming language for a given problem, you may not use any program that generates Haskell and submit the generated code.

9. You may test problems on your computer as often as you wish.

10. To receive credit for a problem solution, you must turn in the source code for judging. The mechanism is described below. The response will be one of the following:

    - Correct
    - Incorrect Output
    - Compiler Error
    - Runtime Error
    - Runtime Limit Exceeded (2 minutes wall-clock time on the judges' machine; this should not be an issue if your program is coded correctly).
    - Submission Botch (something went wrong with how you submitted the program to us).

It is possible that you receive a "Correct" response for a submission that may later be ruled not to count. For example, if you are an expert team and submit a novice problem, the judging system will score the problem like any other, but the score tabulating program will reject it.

11. If a contestant believes that a problem is stated ambiguously and requires clarification, the contestant may submit a question to the judges (mechanism described below). The question and its answer may be provided to all contestants if it seems relevant. A response to the question is guaranteed, although the judges may refuse to answer any question.

12. No discussion is permitted with people outside your team.

13. All input is read from `stdin`, and all judged output is to be written to `stdout`. Output to `stderr` will be ignored.

14. You are responsible for backing up your own code and for making your own printed listings.

15. Use only system include files (e.g., `stdio.h`), as necessary. You may also use library functions as provided for the language you are using. The program should be self-contained: do not submit multiple-file solutions.

16. After submitting a judged run, you must wait for the response before *re*submitting the same problem. A severe penalty applies if you break this rule. You may, however, submit solutions to other problems while waiting for a response from the judges.

17. Judges may disqualify any team not competing in the spirit of the contest. Examples of unfair practices are stealing files, impeding others, degrading performance, unfair use of software tools, adjusting system clocks, etc.

18. The decision of the judges is final.

19. The judges are not responsible for network or system failures or delays.

4 Mechanisms

The mechanism for submitting judged runs and clarifications uses electronic mail. A set of programs are installed on your system that take the name of your program file, problem number, and team number, and sends the program to the judges. Your response is returned through e-mail, without code. A similar program sends a clarification request. The exact programs and how to use them is described in an accompanying document. If you did not get a copy of it, ask your site coordinator for one.

Clarifications and periodic scores are distributed by e-mail.

5 Scoring

- No prizes are awarded.
- The team that has had the most problems judged correct at the end of the contest will be declared the winner. There are separate rankings for each category, as well as overall. If more than one team has solved the same number of problems, the winner is the one with the fewest penalty points.

1. Penalty points are assessed for solved problems only.
   (a) One penalty point is assessed for each minute from the beginning of the contest until the problem was solved.
   (b) The time of solution is defined to be the time of submission, according to your local computer time.
   (c) Twenty penalty points are assessed for each judged run.
   (d) Fifty penalty points are assessed for breaking rule 16.
2. No penalty points are assessed for any problem which was not ultimately solved.
3. The contest coordinators reserve the right to adjust penalty points to allow for unforeseen circumstances.

- The score is posted every half hour through e-mail. The scores may be withheld near the end of the contest to keep the final results a surprise.
- The final results will be mailed out (we hope) within a half-hour after the contest ends.

6 Language Notes

Here we describe the details of how each language is going to be used during the contest, as well as any notes that may be relevant to that language. Compiled languages are handled similarly to this:

```
gcc foo.c -lm
a.out <input.data >output
```

and then the `output` file is tested for correctness. The exact link libraries specified are listed in Table 1. Interpreted languages (labelled with a † in the table) are handled similarly to this:

```
scm foo.scm <input.file >output
```

Note that even though `icont` is an interpreter, it is treated like a compiler since it creates an executable.

Emacs Lisp is interpreted with the command

```
emacs -batch -l foo.el <input.file >output
```

Note that the Emacs 19 `(message)` function writes to `stderr`, so use `(print)` or `(princ)` and direct output to `stdout`. In Emacs 18, `(message)` wrote to `stdout`.

Programs written using Standard ML **must** adhere to the following conventions. The entire smlnj-lib version 0.2 is pre-dumped into the sml compiler image; all the functions defined therein are available for your use. The final statement of your submitted program must be the following:

```
exportFn("a.out",main);
```

where `main` may be substituted for the entry point into your program. This will create an executable file called `a.out` which can then be tested. Your entire file will be compiled like this:

```
smll < foo.sml
```

where `smll` is the sml compiler with the library already dumped into it.

Matlab programmers should read input from the file `matlab.in` and write to the file `matlab.out` instead of `stdin` and `stdout` respectively.

People wishing to use the CMU Common Lisp environment provided here should contact Alejandro Caro `<acaro@whopper.lcs.mit.edu>` for the specific details of the environment.

Ada programs must have the entry point named `main`.