NMU PROGRAMMING CONTEST #4
Saturday 29 March 2003

1. A team consists of at most three members. Each team will have a single computer at its disposal to develop solutions to the assigned problems. The team may share use of the computer in any way it sees fit to do so. The team may solve the problems individually or collectively at its discretion. However, a team member may not communicate—personally, electronically, or otherwise—with anyone other than a judge (via runner) or another member of the same team during the contest.

2. The judges will be happy to answer questions of clarification during the contest and are willing to assist with technical difficulties, but they cannot provide any hints or information relating to how the problems should be solved.

3. There are 6 problem specifications and 5 hours in which to solve them. The problems need not be of equivalent difficulty with each other and are given in no particular order. They may be solved in whatever order the team sees fit. The team may solve the problems with any compiled language or with Java. The team is not constrained to using the same language for each problem; each problem is solved completely independently of the others. If a problem is solved with a compiled language, the team is to generate a single file called **prob*.exe** , where * (defined here and used hereafter) is the digit between 1 and 6 corresponding to the problem being solved. This file should be either a Windows or DOS executable file compatible with Windows ME, the system on which the problems will be judged. If a problem is solved with Java, the team is to generate one or more files ending with **.class** , corresponding to the classes used to solve the problem. The main class must be called **prob*.class** . The class files must be compatible with Java 2 SDK v. 1.3.1, the Java interpreter being used to test these solutions. Compilers for C, C++, and Java will be provided. It should be understood, though, that the problems were defined with C in mind, and that the official solutions will be in C. The team may use any printed reference materials during the contest but is disallowed from using the Internet and from using media (such as magnetic or optical) specifically designed for computer reading.

4. The executed program will open an existing file, **prob*.in** , for reading, and will create and open a file, **prob*.out**, for writing. The program must use these files and no others. The use of other files will cause the program to fail during judicial testing. Any output sent to the screen will be ignored by the judges; only the output sent to **prob*.out** will be evaluated. The judge will not enter information to the program via the keyboard, so any attempt to read from the keyboard will cause the program to hang during judicial testing, thus disqualifying it (see below). If Java is used to solve a problem, it must be run as an application, rather than an applet; applets cannot modify files. Any graphical output will be ignored; any request for GUI input will be ignored, thus disqualifying the program, as above.

5. **prob*.in** and **prob*.out** are text files. They will be constrained to containing only the 94 visible ASCII characters, the space character (no program will be required to parse or print a **<TAB>** character) and the end-of-line character, which varies between systems and will be denoted by **<EOLN>**. In the Windows/DOS environment, **<EOLN>** is actually represented by two consecutive ASCII characters: 13,10. Most programming languages developed for the Windows environment handle this two-byte representation naturally, with no extra effort required on the part of the programmer; in particular, the “\n” representation for the end-of-line character in C/C++ in Windows does indeed refer to the required two-byte sequence. (However, Java has a peculiarity: the println command will generate the correct two-byte sequence, but use of “\n” will not generate the correct sequence. A Java program that uses “\n” will fail during judicial testing.) All files are terminated with the end-of-file marker, denoted by **<EOF>**, which also varies between systems. In the Windows/DOS environment, **<EOF>** is handled automatically by the operating system, and no special care need be taken to ensure that it is written properly. The team may assume that the judicial input file will be formatted correctly—it is not necessary to check for syntax errors. The output file must be formatted *exactly* to specification. For each valid input file, there will be *one and only one* correct output file. A program that yields an incorrectly formatted output file will be judged to be incorrect, even if the bulk of the program performs correctly. Each problem specification will contain a sample input and output file, but this sample need not be the same input file used for judicial testing.

6. Just prior to the beginning of the competition, each team will send one representative to a designated location to receive their packet. All teams will be given their packets simultaneously, and the 5-hour countdown will begin at

that second. Within this packet will be three copies of the problem specifications and six floppy diskettes. Each diskette will be designated for one specific problem. Each diskette will contain a hidden identification file, which must not be removed or modified in any way. When a solution has been developed for a particular problem, the team will place either **prob*.exe** or a collection of **.class** files on the appropriate diskette and will deliver that diskette to a judge (via runner), who will mark the time of receipt. No other files should be placed on this diskette; in particular, the source code itself should not appear on the diskette. The judge will execute the submitted program against a preexisting **prob*.in** file. The generated **prob*.out** file will be compared against the one and only correct output file. The team will be told (via runner) whether it has successfully generated the correct output file, but in the event that the output file is incorrect, no information beyond the fact that it is incorrect will be revealed to the team. Any run-time error in the program will cause the solution to be judged incorrect, even if the correct output file is generated despite this. The sample solutions all run "instantly" on the official input file; any submitted program that takes longer than 5 seconds to execute will be considered an incorrect solution, but, again, the only information revealed to the team will be that the solution was incorrect. The official input files will not be revealed to the teams or to any member thereof until after the competition has terminated. If a submission is judged to be incorrect, the diskette will be returned to the team; the team will be allowed to submit as many solutions as they like to a problem (with a penalty, see below). If a submission is judged to be correct, the diskette will be retained by the judges. All diskettes are to be returned to the judges at the termination of the competition.

7. All communication between the judges and contestants will occur via intermediaries, known as runners. Any question or request must be made in writing and handed to a runner who will deliver the message to a judge. The judge's response will also be in writing. At no time during the competition are the judges and contestants to communicate face-to-face. The runners may also be used to obtain paper copies of programs in progress. Just give a runner a diskette with a file you would like printed.

8. All attempted solutions will be archived by the judges. There will be no known error in the problems by any of the judges prior to the competition, but if an error is discovered in either the problems or the judging during the course of the competition or during the course of a challenge afterward, the archived solutions will be reexamined and the scores retabulated based on these results. Aside from this retabulation, no further remedy will be offered. (In particular, if a team uses a large part of its resources to ferret out a non-existent bug in an improperly rejected solution, it will not be offered compensatory time.) However, as stated, care will be taken to prevent errors from occurring before the competition begins, so that, hopefully, this situation will not arise.

9. Any attempt by the executed program to copy or damage the preexisting input file in any fashion or to demean the integrity of the contest in any way will result in the immediate disqualification of the submitting team.

10. Scoring is determined by a pair of integers as follows: The **count** for each term is the number of problems solved. The **total** for each team is the total number of minutes required to solve each problem starting from the beginning of the contest. A penalty of 20 minutes is added to the **total** for each unsuccessful attempted solution of a problem, provided that a correct solution of that problem is eventually submitted. More precisely: at the beginning of the competition, **count** and **total** are initialized to zero. Upon the evaluation of a correct submission, **count** is incremented by one, **total** is incremented by the number of minutes between the start of the contest and the submission of the executable, and **total** is further incremented by 20 for each incorrect solution to the problem submitted prior to the correct one.

11. Team ranking is determined by each team's **count**, with the highest **count** being ranked first. Ties are broken by **total**, in favor of the team with the lowest **total**. Any subsequent ties remain unbroken. With this in mind, it is to the team's advantage to seek out and complete the easier problems first. This will result in a smaller **total**. However, keep in mind as well that difficulty is a matter of individual perception, and that the problems are given in no particular order.

12. Schools may submit as many teams as they care to. School ranking is determined by summing the **count** values and summing the **total** values of the three highest-ranking teams affiliated with that school. These aggregate values are used as above to rank the participating schools. Schools that submit fewer than three teams are thus disadvantaged in the school ranking; however, schools that submit more than three teams have no gross advantage, in that most schools can predict which of their teams will be the three highest-ranking. Note that ties only occur if

both **count** and **total** are identical, so if there is more than one team in third place, it does not matter which is added to the aggregate score.

13. Schools are not constrained to submitting a number of contestants that is divisible by three. Teams must not contain more than three members, but may contain fewer. Additionally, individuals unaffiliated with a team may attend as well and may form teams with other unaffiliated individuals from their own or other schools or from no school. Such hybrid teams are entitled to their team ranking and to any formal acknowledgement thereof, but their scores cannot be figured into the aggregate score for any school under any circumstances.

REMINDERS

1. When a program fails, the team will be only be told that it failed; no specific reason will be given. You may have made a simple careless error such as those described below.

2. Failure to format the output file *exactly* as specified will cause the program to fail.

3. Failure to name the submission, input, and output files *exactly* as specified will cause the program to fail.

4. Failure to submit the program on the correct diskette will cause the program to fail.

5. Reading input from the keyboard or specifying input from a Graphical User Interface will cause the program to fail.

6. Screen and graphical output will be absolutely ignored.

7. You will be penalized for programs that fail.

8. <EOLN> and <EOF> in the specifications refer to the end-of-line and end-of-file markers. They do not represent the literal strings "<EOLN>" and "<EOF>". Similarly, each space in the sample input and output files will be represented by ·.

INSPIRATION AND DIFFERENCES

This competition is modeled on the ACM International Programming Contest. Here are some ways in which this competition *differs* from the one sponsored by the ACM.

1. The winner(s) of this competition do not compete in a higher competition following this one. This buck stops here.

2. In this contest, according to the official rules, no reason will be given whatsoever for a program failure, as the ACM has several (overlapping and confusing, in my opinion) categories in which errors can lie. However, in the past, the contest judge (who is also the author of these rules) hasn't had the heart to be brutally binary in all judgments. The judge reserves the right to give nonspecific information pertaining to a failure as long as it is done in a fair and impartial manner.

3. Any compiled language (as well as Java) is acceptable in this competition. The ACM restricts its competition to certain languages.

4. All problems in this competition are specified completely without ambiguity. There is one and only one correct output file for every valid input file. The ACM competition allows for a certain amount of ambiguity.

5. There is no designation in this competition between official and unofficial teams. All teams are welcome to participate. The only caveat is that hybrid teams from multiple schools or containing non-students as members may not contribute to any aggregate school score.

6. All registered students are eligible to compete for the school they attend. The ACM competition limits the participation of graduate students.

REDIRECTING INPUT AND OUTPUT

Many participants in these competitions wonder how to read program input from a file and how to write program output to a file, rather than using standard input and output. Here is how to do it in some languages; if you're using a different language, you'll have to look it up on your own.

C:

```
FILE *in, *out;
.
.
.
in = fopen ("prob1.in","r");
out = fopen ("prob1.out","w");
.
.
.
fscanf (in,...);
fprintf (out,...);
```

C++:

```
ifstream in ("prob1.in");
ofstream out ("prob1.out");
.
.
.
in >> ...;
out << ...;
```

Java:

```
BufferedReader in = new BufferedReader (new FileReader ("prob1.in"));
PrintStream out = new PrintStream (new FileOutputStream ("prob1.out"));
.
.
.
... = in.readLine();
out.println (...);
```

Pascal:

```
VAR
 IN, OUT : TEXT;

.
.
.
RESET (IN,"prob1.in");
REWRITE (OUT,"prob1.out");
.
.
.
```

```
READLN (IN,...);
WRITELN (OUT,...);
```

Sample Questions

My Web Page always has old problems and solutions from this and from other similar competitions. Check it out: http://euclid.nmu.edu/~apoe

Using TextPad

TextPad is the world's easiest editor/compiler. Just follow these simple guidelines and you should have no trouble submitting programs in the contest.

1. Every file you make is a text file. There are no “Project” files or any other nasty bloated thing you have to worry about.

2. Use the File menu to create new documents, to open existing documents or to save documents. Java files should end in .java . You can save C/C++ files with .c, .cpp, or .cc .

3. Use the Tools menu to compile and run Java and C/C++. If you are using Java, be sure that you Run Java Application. Applets are not acceptable for this contest.

4. When you hand in a solution, put the executable (.exe file) on your floppy. Nothing else. Don't waste my time with the source code. If you are using Java, put EVERY .class file on your disk. Even if you put multiple classes in the same source file, the Java compiler will make a separate .class file for each one.

5. That's it!!!!

Submitting Solutions Addendum

Northern Michigan University recently discontinued the use of floppy drives with its laptops. The laptops now all have burners. Rather than six individual floppies, each encoded with a problem, you have three blank CD-R's. These CD-R's are not encoded; you may submit any problem any on any CD-R. Your CD-R will be returned to you after your program is graded, irrespective of whether your program worked. When you pass in a CD-R, include a sheet of paper indicating who you are and which problem you are submitting. RETURN ALL CD-R'S AT CONTEST END!

To burn a file on a CD-R, use the Easy CD Creator, a link to which is on your Desktop:

1. Double click on the icon.
2. Move mouse to make a data CD.
3. Click on Data CD Project.
4. Drag the files you want to burn from the upper window to the lower window. When you're done, click Record. It has already been set up to burn a session while leaving the CD open. Do not change these settings. It will allow you to reuse the CD later.
5. If you are submitting a DOS executable, I only need to see the .exe file. If you are submitting JAVA class files, I need EVERY .class file generated by your program.