NMU PROGRAMMING CONTEST #20
Saturday 30 March 2019

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. The team may solve the problems with the C, C++, C#, Java, Python, or Kotlin languages. 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 C, C++, or C#, 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, a DOS executable file compatible with Windows 10, the system on which the problems will be judged. If a problem is solved with Java or Kotlin, the team is to generate a single file called **prob*.jar** , an executable Java file, which will also be run in the Windows 10 environment. Compilers for C, C++, C#, Java, and Kotlin will be provided. The Java provided will be the most recent version at the time of the imaging of the laptops for the contest. If a problem is solved with Python, the team is to generate a single file called **prob*.py** . Python 2 and Python 3 are both acceptable. It should be understood that the problems were defined with Java in mind, and that the official solutions will be in Java. 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, optical, or solid state) 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 access or 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. The sequence "\r\n" will, however, emulate an end of line character perfectly.) 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. Shortly before the beginning of the contest, each team will receive its laptop and their first packet containing a copy of these rules, a laptop instruction guide, and one flash drive. The teams are encouraged to set up their laptops, and test the compiler. If there is a problem with the laptop, either at this time or during the contest itself, the team should contact a runner immediately; the runner will send someone from the tech team to solve the problem. Just prior to the beginning of the competition, each team will send one representative to a designated location to receive their second 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, submission slips, and some scratch paper. When a solution has been developed for a particular problem, the team will flag a runner who will accept from them a flash drive on which the team will have stored either a **prob*.exe** or **prob*.jar** file. The team will also give the runner a submission slip stating the school name, the team name, the room number, and the problem number. The runner will give the team a blank (replacement) flash drive and will deliver the paper and flash drive to a judge, who will mark the time of receipt. No other files should be placed on the flash drive; in particular, the source code itself should not appear on the drive. 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 informed in writing (via runner) whether it has successfully generated the correct output file, and in the event that the output file is incorrect, no information beyond the fact that it is incorrect will necessarily be revealed to the team, though the judge may at his/her discretion give a vague hint (e.g. "it infinite loops"). 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 10 seconds to execute will be considered an incorrect solution. The official input files will not be revealed to the teams or to any member thereof until after the competition has terminated. The team will be allowed to submit as many solutions as they like to a problem (with a penalty, see below).

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. A team may give a runner a flash drive with a file to be printed; the runner will give the team a replacement flash drive and will return with the printout.

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, we hope, 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 team 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, as 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 or independent 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 circumstance.

14. A grad student is defined as being a registered student already holding a bachelor's degree or equivalent. A grad team is defined as a team containing at least one grad student. Grad teams are allowed to compete and are entitled to any award they may win; however, for the purposes of school ranking, grad teams will have their **count** and **total** numbers divided by two (and rounded down) before computing the top three teams from that school.

REMINDERS

1. When a program fails, the team will be only be told that it failed; no specific reason will necessarily 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. Reading input from the keyboard or specifying input from a Graphical User Interface will cause the program to fail.

5. Screen and graphical output will be absolutely ignored.

6. You will be penalized for programs that fail.

7. <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 necessarily 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. 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.

4. 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.

5. 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,...);
.
.
.
fclose (in);
fclose (out);
```

C++:

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

C#:

```
StreamReader in = new StreamReader ("prob1.in");
StreamWriter out = new StreamWriter ("prob1.out");
.
.
.
... = in.ReadLine();
out.WriteLine (...);
.
.
.
in.Close();
out.Close();
```

Java:

```
Scanner in = new Scanner (new File ("prob1.in"));
PrintWriter out = new PrintWriter ("prob1.out");
.
.
.
... = in.nextLine();
out.println (...);
.
.
.
in.close();
out.close();
```

Python:

```
In = open ('prob1.in','r')
Out = open ('prob1.out','w')
.
.
.
In.close()
Out.close()
```

Kotlin:

```
val in = Scanner (File ("prob1.in"))
val out = PrintWriter ("prob1.out")
.
.
.
... = in.nextLine()
out.println (...)
.
.
.
in.close()
out.close()
```

Sample Questions

My Web Page always has old problems and solutions from this and from other similar competitions. Check it out: http://philos.nmu.edu/index.html

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 . C files should end in .c . C++ files should end in .cpp . C# files should end in .cs . Python files should end in .py ; Kotlin files should end in .kt .

3. Use the Tools menu to compile and run C, C++, C#, Java, Python, and Kotlin. If you are using Java or Kotlin, compile the file containing the main method. The compiler will correctly compact all the generated classes (including ones in different source code files) into a single executable Jar file.

4. When you hand in a solution in C, C++, or C#, put the executable (.exe file) on your flash drive. Nothing else. Don't waste my time with the source code. If you are using Java or Kotlin, put the .jar file on your flash drive. If you are using Python, put the .py file on your flash drive.

5. That's it!!!!