2023/2024 Southern California Regional ICPC Rehearsal/"Warm-Up"
October 14, 2023; 11:30 AM PDT

Zoom Webinar ID will be posted on the socalcontest.org web site.
If you have any questions about the rehearsal, or need additional rehearsal login IDs, please send a message to systems@socalcontest.org.

The following steps are designed to be executed within the programming contest environment. For the contest rehearsal, a virtual appliance exists that very nearly matches the actual contest environment. Download the virtual appliance from the socalcontest.org website.

Warm-Up Problem 1

This problem should be completed first. Do all the steps before attempting problems 2 and 3.

The purpose of this problem is to familiarize all contestants with many parts of the environment. All contestants should submit this input correctly and run all commands listed before moving on to Warm-up Problems 2 and 3.

Open the Firefox browser. To connect to DOMjudge, connect to:

https://rehearsal.socalcontest.org/domjudge/

There is a shortcut for the "Terminal" for the command prompt at the bottom of the screen. IDEs can be reached from the command prompt or from the Applications menu under Development.

Step 1: Type in the problem code:

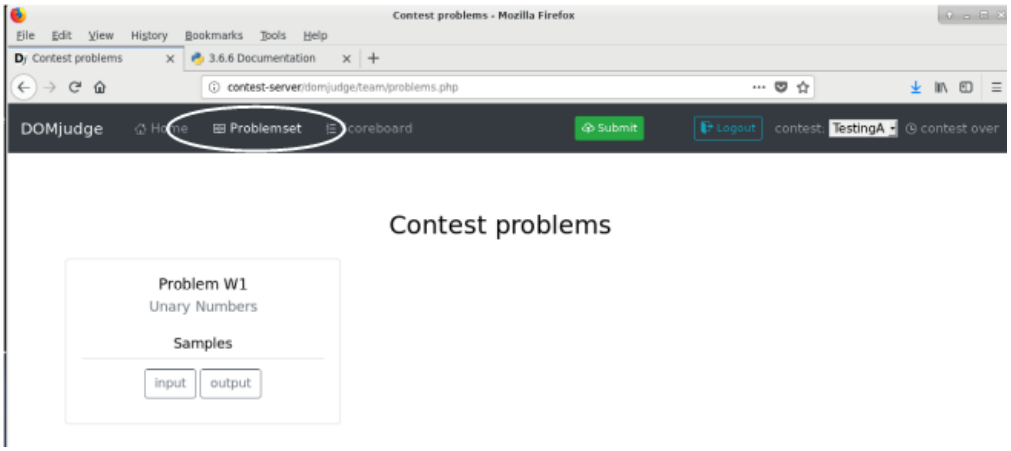
Select any one of the problem solutions and type it in (code follows) and save the file.

Step 2: Compile the code using from the command prompt:

compile *source_file*

Note that the compile command is not necessary for Python 3.

Step 3: Get any supplementary materials and sample input and output from DOMjudge:



Step 4: Test the code:

Use the test data provided, along with any other data you choose.

NOTE: During the contest your code will be judged against data you never see (the "judge's data"). The sample data provided are not exhaustive – it is your responsibility to design a thorough test plan.

Your program can be run after compilation with the following (compilation not needed for Python 3):

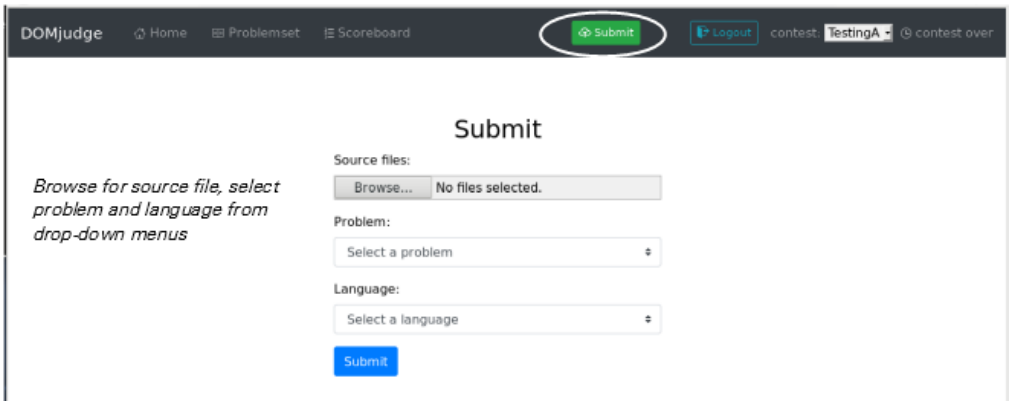
C, C++, input from keyboard `$ ./a.out`
C, C++, input from file *data.in* `$ ./a.out < data.in`
C, C++, input from *data.in*, output to *results.out* `$ ./a.out < data.in > results.out`

Java, input from keyboard `$ java classfile`
Java, input from file data.in `$ java classfile < data.in`
Java, input from data.in, output to results.out `$ java classfile < data.in > results.out`

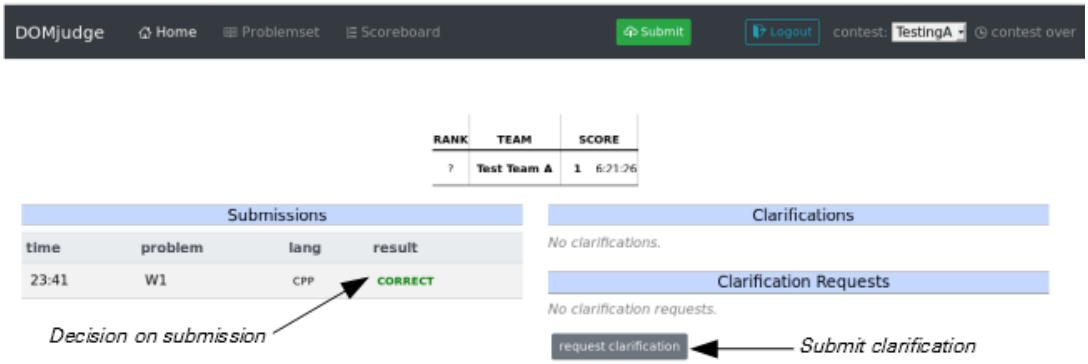
Python3, input from keyboard `$ python3 sourcefile.py3`
Python3, input from file data.in `$ python3 sourcefile.py3 < data.in`
Python3, input from data.in, output to results.out
`$ python3 sourcefile.py3 < data.in > results.out`

Kotlin, input from keyboard `$ kotlin ClassfileKt`
Kotlin, input from file data.in `$ kotlin ClassfileKt < data.in`
Kotlin, input from data.in, output to results.out
`$ kotlin ClassfileKt < data.in > results.out`

Step 5: Submit the code via the DOMJudge interface, first select the source file, then click submit:

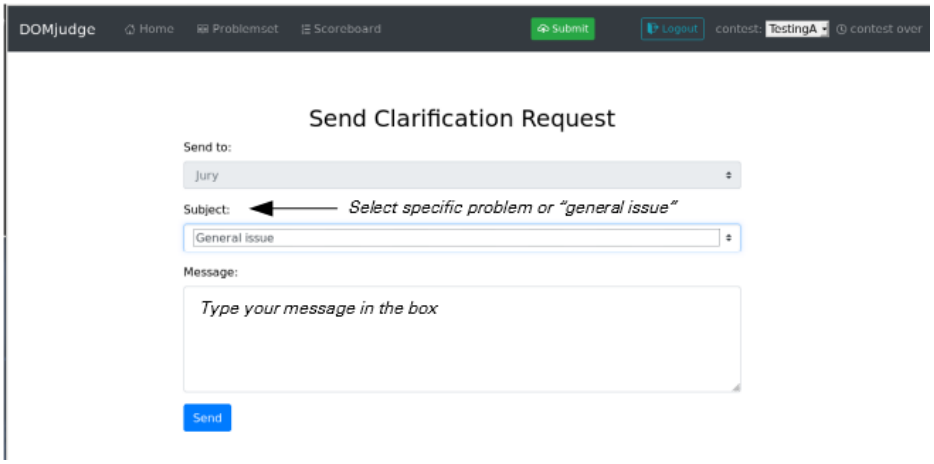


Step 6: See the results from your submission:



Step 7: Request a Clarification by clicking the *request clarification* button on the main page …

… and then completing the form. Use the problem number as the subject for questions on a specific problem.



If you don't have a specific Clarification regarding this rehearsal or one of the assigned problems, please request a Clarification to familiarize yourself with the process. Submit a question like "What was the color of that white horse?"

One objective of this rehearsal period is verifying that all components of the contest management system are working and configured correctly. The officials are reviewing everything. Submission response times are likely to be longer than they will be at the actual contest for similar submissions.

Step 8: (Optional) Find out how much time is left in the contest and look at the scoreboard:

Time left – Look at your start page from DOMjudge where it is displayed in the upper right-hand corner

Score –Click the *Scoreboard* button on the ribbon (see Step 3) to see the current scores; your team's score is displayed on the start page.

Step 9: See the on-line language/library documentation:

C++ library – file:///usr/share/doc/libstdc++-docs/html/index.html

Java API – file:///opt/docs/java/api/index.html

Python 3 – file:///opt/docs/python/index.html

Kotlin – file:///usr/share/doc/kotlin/kotlin-reference.pdf

Warm-up Problem from 2002/2003
Unary Numbers

What could be simpler than binary numbers? Unary numbers! A Unary number n, $n > 0$, is coded as $n - 1$ one bits followed by a zero bit. Thus the code for 5 is 11110. Here are some unary numbers.

decimal	*unary*
1	0
2	10
3	110
4	1110
5	11110
6	111110
7	1111110

Input consists of decimal numbers, one per line, with no leading or trailing whitespace. Each number will be in the range 1–76. Input is terminated by end-of-file.

For each number, produce a single line of output consisting of the input decimal number, with no leading zeroes or spaces, a single space, and the unary equivalent with no leading or trailing spaces.

Sample Input

```
76
37
5
28
14
8
1
```

Output for the Sample Input

```
76 11111111111111111111111111111111111111111111111111111111111111111111111111110
37 1111111111111111111111111111111110
5 11110
28 11111111111111111111111110
14 11111111111110
8 11111110
1 0
```

unary.cpp (or unary.cc)

```
#include <iostream>
using namespace std;

int main ()
{
   int n;

   while(cin >> n) {    // cin >> n is false when no values remain
      cout << n;
      cout << ' ';
      while (n > 1) {
         cout << '1';
         n--;
      }
      cout << '0' << endl;    // endl causes a newline, ASCII 0x0A, to be emitted
   }
   return 0;            // indicate normal program termination
}
```

unary.java:

```
import java.io.*;

class unary {                                //main class needs to match filename
   public static void main (String [] args) throws IOException
   {
      int            n;
      String         s;
      BufferedReader stdin;

      stdin=new BufferedReader(new InputStreamReader(System.in));
               // wrap BufferedReader around InputStreamReader around System.in

      while ( (s=stdin.readLine()) != null) {
               // BufferedReader.readLine returns null at end-of-file
         n=Integer.parseInt(s);
         System.out.print(n + " ");
         for (int i=n - 1; i > 0; i--) {
            System.out.print("1");
         }
         System.out.println("0");              // println() writes an ASCII 0x0A
      }
      System.exit(0);                       // indicate normal program termination
   }
}
```

---

unary.py3:

```
import sys

for line in sys.stdin:
    line=line.replace('\n','')  # remove end-of-line present in strings read from input
    n=int(line)
    print(n, end=' ')           # print number in decimal followed by one space
    for i in range(n - 1):
        print('1', end='')      # print '1' without any trailing characters
    print('0')                  # print '0' followed by newline

exit(0)                         # indicate normal program termination
```

unary.c:

```
#include <stdio.h>

int main()
{
   int  i;
   int  n;
   char s[4];  /* make room for up to two decimal digits, end-of-line (newline),
                  and a zero-byte to terminate the string */

   while(fgets(s,sizeof(s),stdin) != NULL) {
        /* read an entire line into s.  fgets() returns NULL at end-of-file */
      sscanf(s,"%d",&n);                      /* extract n from the input line */
      fprintf(stdout,"%d ",n);
      for (i=n - 1; i > 0; i--) {
         fputc('1',stdout);
      }
      fputs("0\n",stdout);
               /* the newline character, \n, emits an ASCII 0x0A */
   }
   return 0;                /* indicate normal program termination */
}
```

---

unary_seed.c:

```
#include <stdio.h>

int main()
{
   int  i;
   int  n;
   char s[4];  /* make room for up to two decimal digits, end-of-line (newline),
                    and a zero-byte to terminate the string */

/*
 * Attempt to read an entire line into s.  The read (fgets) preceding the while loop is the seed
read.
 */
   fgets(s,sizeof(s),stdin);               /* attempt to read an entire line into s */
   while( !feof(stdin) ) {                    /* while not end-of-file ... */
      sscanf(s,"%d",&n);                      /* extract n from the input line */
      fprintf(stdout,"%d ",n);
      for (i=n - 1; i > 0; i--) {
         fputc('1',stdout);
      }
      fputs("0\n",stdout);   /* the newline character, \n, emits an ASCII 0x0A */
      fgets(s,sizeof(s),stdin);          /* attempt to read an entire line into s */
   }
   return 0;                             /* indicate normal program termination */
}
```

unary.kt:

```
import kotlin.system.exitProcess

fun main() {
  var s: String?
  s=readLine()            // seed read
  while (s != null) {
    var n=s.toInt()
    print(n); print(" ")
    while (n > 1) {
      print("1");
      n -= 1;
    }
    println("0")
    s=readLine()
  }
  exitProcess(0)          // indicate normal program termination
}
```

2023/2024 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Warm-Up Problem 2
Xylophone

A *xylophone* is a musical instrument made of wooden bars, each of which makes a specific pitch when struck with a mallet. The wooden bars must have contiguous integer lengths from the longest to the shortest, without duplicates. In other words, every bar except for the rightmost one must have a length exactly 1 longer than the one immediately to its right. For example, a xylophone may have bars of lengths [7, 6, 5, 4] or [10, 9, 8], but not [7, 5, 4] nor [3, 3, 2, 1].

You already have 3 wooden bars of different lengths, and want to create a xylophone using all of them. You may not cut the bars or alter them in any way, but you may buy additional wooden bars as necessary. The cost of buying a wooden bar is equal to its length. Find the minimum cost to build a xylophone.

The input contains a single line with three space-separated integers, denoting the lengths of the wooden bars you already have. Each integer is between 1 and 5000, inclusive. You are guaranteed that the three integers are distinct.

Your program must output a single integer that represents the minimum cost to make a xylophone using all three given wooden bars.

Sample Input

```
10 3 7
```

Output for the Sample Input

```
32
```

2023/2024 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Warm-Up Problem 3
Swamp County Toll Roads

Swamp County has just completed its first express toll roads. As is now the norm, there are no attended toll booths—all tolls are collected electronically. Rather than issue electronic tags or transponders to road users, toll collection is done based on the automated reading of license plates as vehicles pass through a toll plaza.

For a variety of reasons, sometimes the license plate images are of low quality. When this happens it is possible for license plates to be mis-read, particularly since some characters closely resemble others. The county has studied cases where this has happened and has come up with estimates of the likelihood that a given character will read correctly.

The county would like your team to write a program that will, given the error estimates for individual characters as stated above, take a list of license plates as read and determine the probability that the entire license plate was read correctly.

Input to your program will begin with a list of accuracy estimates. Each estimate consists of the character, a single space, and its estimate of a correct read, in the range $0 < p < 1$. For example, the first line of the sample input indicates there is an 88% chance that an "I" read from a license plate is actually an "I". Any characters not in the list are taken as 100% ($p = 1$). You may assume that the reading of one character has no effect on the reading of another character (they are independent). This list ends with an empty line.

The remaining lines of input contain license plates as read, one per line, starting in the first column. Each license plate contains a string of one to eight upper-case letters or digits. There are no more than fifty license plates in the input. The list of license plates ends with end-of-file.

For each license plate, your program is to print a line with the the probability that the license plate was read correctly as a value between zero and one. The value is to be rounded to and printed with three digits after the decimal point. No leading or trailing white space is to appear on an output line.

Sample Input

```
I 0.88
1 0.99
0 0.87
O 0.95
Q 0.87

PROGRAM
ICPC2018
2JKB843
1JKL893
SNU8J5
```

Warm-Up Problem 3
Swamp County Toll Roads (continued)

Output for the Sample Input

```
0.870
0.758
1.000
0.990
1.000
```