2024/2025 Southern California Regional ICPC Rehearsal/"Warm-Up"
November 2, 2024; Attended from 2 to 4 PM Pacific Time

Zoom Link: https://us02web.zoom.us/j/85074773578

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:

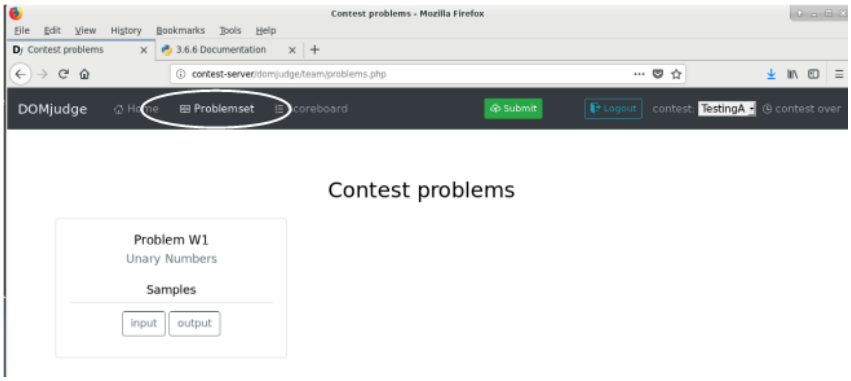
Select any one of the problem solutions and type it in (code follows) and save the file.

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):

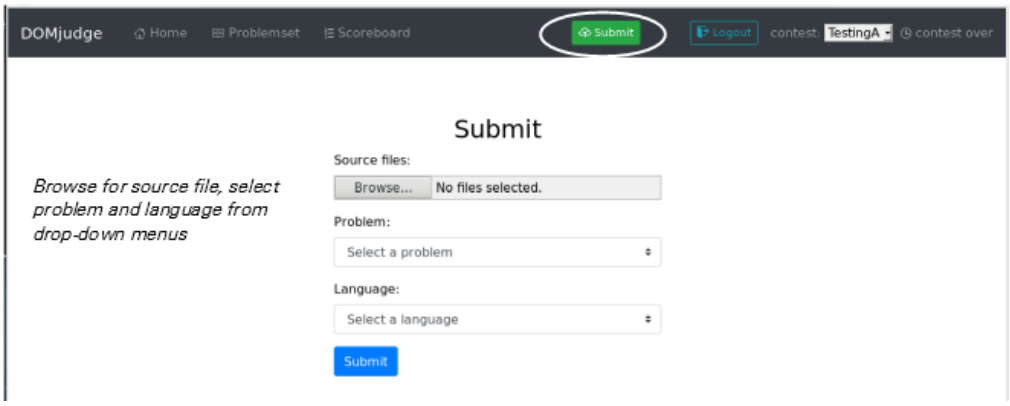
C, C++, input from keyboard `$ ./a.out`
C, C++, input from file *data.in* `$ ./a.out < data.in`
C, C++, input from *data.in*, output to *results.out* `$ ./a.out < data.in > results.out`

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`

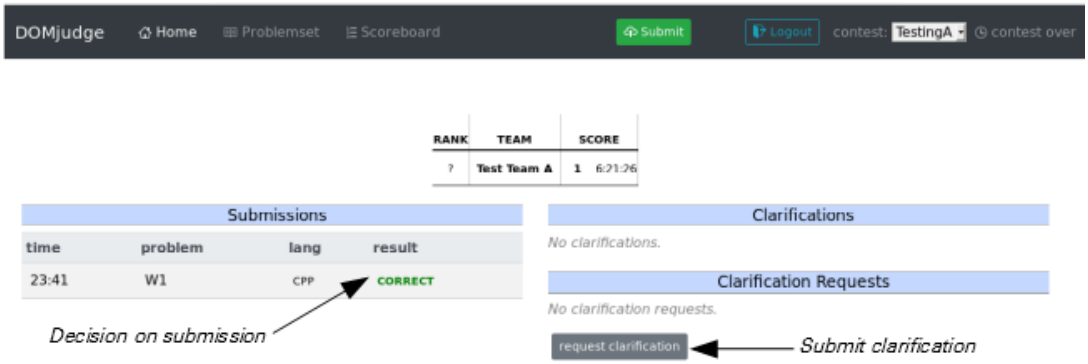
Kotlin, input from keyboard `$ kotlin ClassfileKt`
Kotlin, input from file data.in `$ kotlin ClassfileKt < data.in`
Kotlin, input from data.in, output to results.out
`$ kotlin ClassfileKt < data.in > results.out`

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



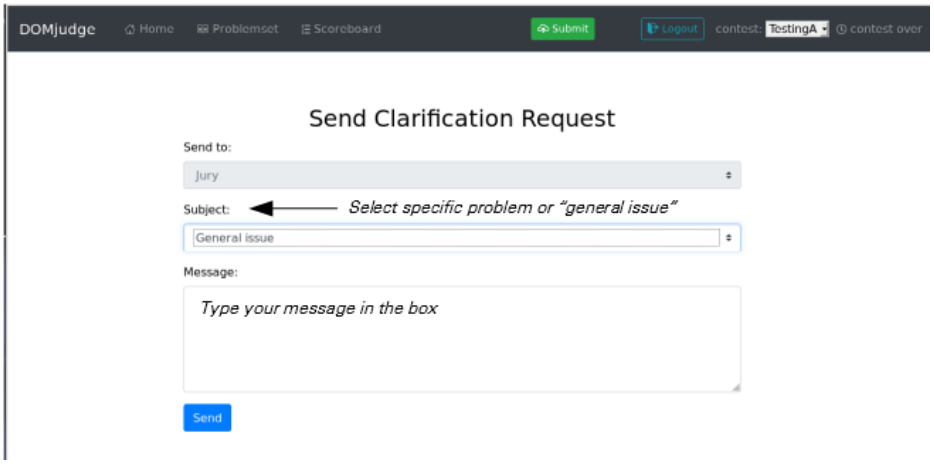
Kotlin submissions will also prompt for the entry point for your program. A default value of *ClassKt* (where the name of your class is substituted for *Class*) may be provided.

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 111111111110
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
}
```

2024/2025 SOUTHERN CALIFORNIA REGIONAL
INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Warm-Up Problem 2
Rock, Paper, Scissors

Rock, Paper, Scissors is a non-transitive game played by all ages. The following rules of the game are adapted from those published by the "official World RPS Society."

1.0. The Game is played where the players represent the three elements of Rock, Paper and Scissors with hand signals.
2.0. These hand signals are delivered simultaneously by the players.
3.0. The Outcome of play is determined by the following:
- Rock wins against Scissors
- Scissors wins against Paper
- Paper wins against Rock

If both players deliver the same hand signal, the outcome is a draw.

Your team is to write a program that will track the results of a series of Rock, Paper, Scissors games between two players.

Input to your program is a series of between 1 and 300 plays inclusive. Each play is represented on a single line. The line contains the first player's hand signal in the first column, followed by a single space and the second player's hand signal. Each hand signal is represented by the letter "R" for Rock, "P" for Paper, and "S" for Scissors. The list is terminated by the end-of-file.

Your program is to print one line containing the number of games won by the first player, the number of games won by the second player, and the number of drawn games in that order. The values are to appear without leading zeroes and are to be separated from each other by single spaces. Output is to begin in the first column and is not to contain any trailing whitespace.

Sample Input

```
R R
S P
R S
P P
P S
P R
```

Output for the Sample Input

```
3 1 2
```

2024/2025 SOUTHERN CALIFORNIA REGIONAL INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Warm-Up Problem 3
Permits in Kafkatown

Getting a business permit in Kafkatown requires a trip to City Hall. There you are given a permit form that must be signed by K city clerks whose names are printed at the bottom of the form.

Entering the clerks' room, you find a long line of people working their way down a narrow aisle along the clerks' desks. The aisle is so narrow that the line is forced to shuffle forward, single file, past each clerks' desk in turn. Once in the line you cannot leave, back up, or change positions with other people. The desks are numbered sequentially.

As you present your permit for a signature, you are told that no clerk will sign unless all of the signatures above his or her name on the permit form have already been filled in. To your dismay, the clerks' desks are not arranged in the same order as the names on your form.

How many times will you need to pass through the line until you can get your permit? Your team is to write a program to determine this.

For example, assume you need signatures from five clerks, at desks number 1, 23, 18, 13, and 99. You will have to go through the line three times: the first time to get signatures from clerks at desks 1 and 23, the second time to get a signature from the clerk at desk 18, and the third time to get signatures from clerks at desks 13 and 99.

The first line of input contains an integer K, the number of signatures you need to collect, in the range 1 to 100 inclusive. This is followed by K lines of input, each containing an integer in the range 1 to 100 inclusive, indicating the desk numbers of each of the clerks whose signature you need, in the order that they appear on your form. (Clerks whose signatures are not needed on your form are omitted from this list.) No desk number will appear more than once.

Your program is to print a single line containing the integer number of passes you will need to make through the line in order to collect all of the signatures that you need.

Sample Input

```
5
1
23
18
13
99
```

Output for the Sample Input

```
3
```