2021/2022 Southern California Regional ICPC Rehearsal/"Warm-Up"
February 19, 2022; 10:00 AM PST

Zoom Webinar ID: 835 3076 9177
Passcode: 235711
https://us02web.zoom.us/j/83530769177?pwd=amVnZU13em9uR2ZEdDNZME15amYvZz09

Warm-Up Problem 1—*Save this document for reference during the contest!*

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.

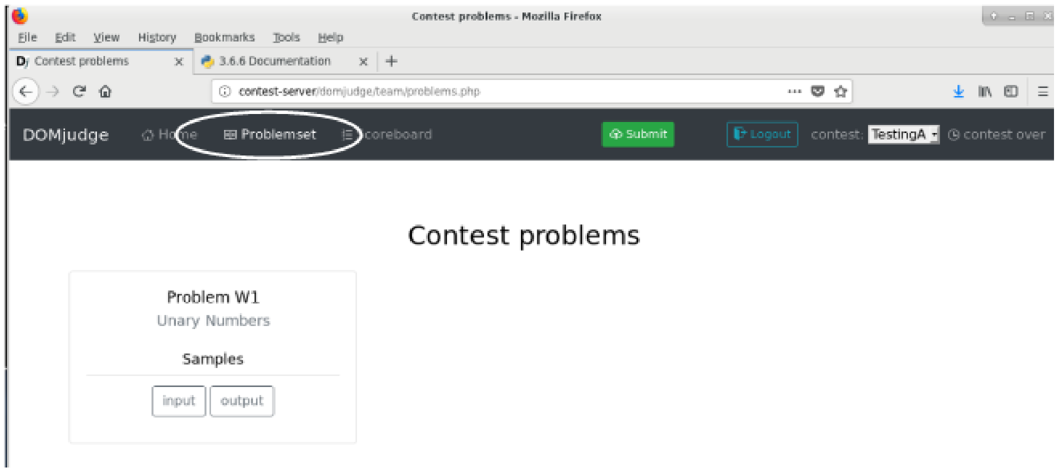
Step 1: Type in the problem code:
Select any one of the problem solutions and type it in (code follows) and save the file.

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:

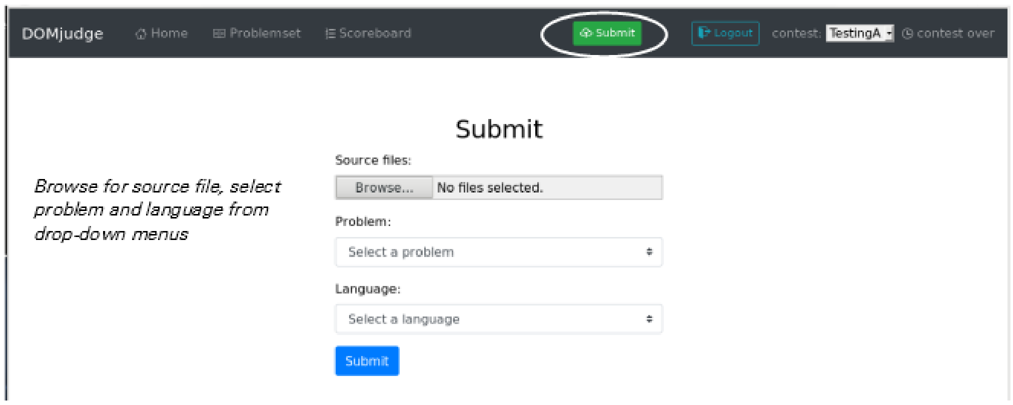


Step 4: Test the code:

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 is not exhaustive – it is your responsibility to design a thorough test plan.

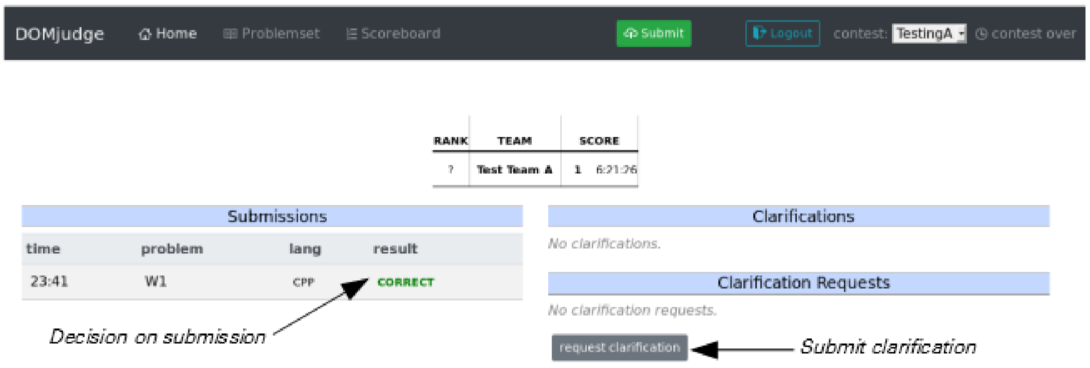
Step 5: Submit the code via the DOMJudge interface, first select the source file, then click submit:



No files selected.

Browse for source file, select problem and language from drop-down menus

Step 6: See the results from your submission:



RANK	TEAM	SCORE	
?	Test Team A	1	6:21:26

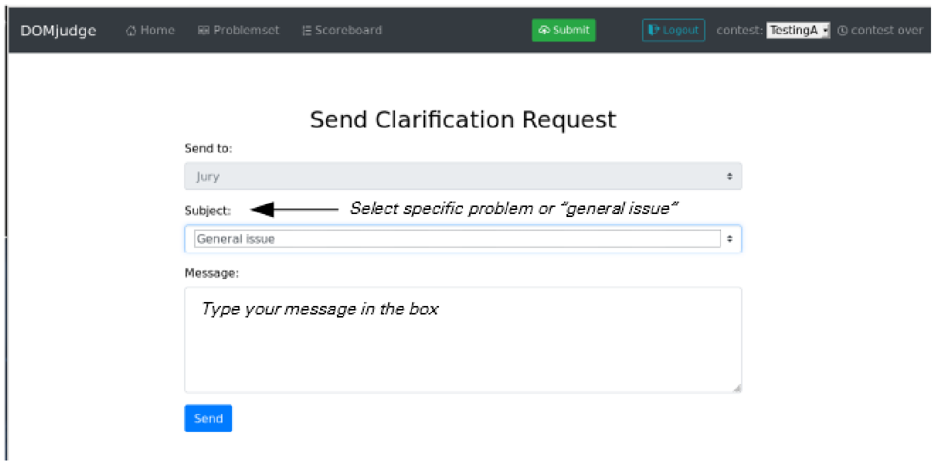
time	problem	lang	result
23:41	W1	CPP	CORRECT

No clarifications.

No clarification requests.

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.



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: (To be available in the final contest VM image) See the on-line language/library documentation:

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

Java API – *file:///usr/share/doc/java-docs/api/index.html*

Python 3 – *file:///usr/share/doc/python-3.10.2-docs-html/index.html*

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 1111111111111111111111111111111111111111111111111111111111111111111111111110
37 1111111111111111111111111111111111110
5 11110
28 1111111111111111111111111110
14 11111111111110
8 11111110
1 0
```

unary.cpp (or unary.cc)

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

int main ()
{
   int n;

   cin >> n;              // seed read
   while(!cin.eof()) {    // eof() valid only after attempted read
      cout << n;
      cout << ' ';
      while (n > 1) {
         cout << '1';
         n--;
      }
      cout << "0\n";      // the newline character, \n, emits ASCII 0x0A
      cin >> n;
   }
   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 */
}
```

Warm-Up Problem 2
“Russian Peasant” Multiplication

“Russian Peasant” Multiplication was used for centuries by people who had minimal formal education—those who learned addition and subtraction, but not the multiplication tables. The method uses iterative addition along with doubling and halving of the operands.

To use this method, write the operands at the top of two columns. The remaining column values are created by repeatedly halving the first operand, truncating any fractional part, while doubling the second one, until the value in the first column reaches one. The product is the sum of the values in the second column that correspond to odd values in the first. This is shown as a running total in a third column.

You are to write a program to demonstrate the mechanics of this method. Input is one to sixteen lines, each containing a pair of integers with values between 1 and 32,767 inclusive. The values in each pair will be separated from each other by a single space. Input is terminated by the end-of-file.

For each pair, print a series of lines, where each line shows a step in the “Russian Peasant” multiplication. Values on each line are to be separated from each other by a single space. End each multiplication with a line containing an equal sign, a single space, and the final product. No leading or trailing whitespace is to appear on a printed line.

Sample Input

```
75 232
20 1357
```

Output for the Sample Input

```
75 232 232
37 464 696
18 928 696
9 1856 2552
4 3712 2552
2 7424 2552
1 14848 17400
= 17400
20 1357 0
10 2714 0
5 5428 5428
2 10856 5428
1 21712 27140
= 27140
```

2021/2022 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 only an integer denoting the 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
```