ACM@KU, Intramural Programming Competition: Rules, Resources, and Examples

The Association for Computing Machinery at the University of Kansas

April 23, 2016

Contents

1   Introduction

The ACM@KU Intramural Programming Competition is an ACM-style programming competition designed to test your knowledge of programming as a tool for general problem solving. Languages of the competition will include but not be necessarily limited to the following:

<div align="center">

C

C++

Clojure

Haskell

Java 8

JavaScript (via Node.js)

Lua

PHP

Python (v. 2.7 and 3.4)

Ruby

</div>

2   Contact

If you have questions regarding the competition, please feel free to contact:

Aleksander Eskilson
IPC Chair
aeskilson@ku.edu

3   Rules

You will be given a docket describing a total of 10 programming challenges, of varying difficulty, and arranged in mixed order. You will be allotted five hours in a single block of time, to write solution programs for these challenges. An automated system called DOMJudge provides a website interface, http://bit.ly/1jVYOw9 or its full url, http://ec2-54-88-68-156.compute-1.amazonaws.com/public/, where you can submit your solutions for judging, receive feedback, and check a running scoreboard and clock. Before the actual day of the competition, a test competition will be available here. The login username: test, password: pass, will allow you to browse the interface and even submit a solution to the problems of the test contest. Here, you can test submitting your programs for automated judging.

All problems will be described in a uniform fashion: An introduction to the problem will set up the scenario and describe what challenge you must write a program to solve. It will then describe how the input your program must read will be structured, and it will prescribe how the program's output must be formatted in order to be read correctly. An example of input and output will be provided for you to check your program against.

Note: when your program is actually judged, different input data will be used, but the formatting will be the same as described in the problem. Pay very special attention to the section describing how to properly write your programs to accept input from DOMJudge and how to structure our output.

Each correct problem is worth 1 point, so whichever competitor completes the most problems out of 10 will be designated the winner. In the case of ties, time (as marked in seconds, running from the beginning of the competition) will be used as a tiebreaker. So if more than 1 competitor completes all 10 problems, the first to have done so is the winner. You may submit your solutions for judging as many times as you need to, but you should be aware that each time you submit a solution that is judged to be incorrect, you will receive a time penalty that will add to your total time, so it is best to work hard to ensure your program is correct before submitting for judging. However, even if your submission turns out to be wrong, you should feel free to rework it until you get it right or decide to move on to another problem.

You must compete by yourself. You will be allowed use of the Internet and any books on your programming language may be used during the competition. You may choose to either use the Linux or Windows machines of the EECS Labs, or you may bring your own laptop. For students who do not have KU EECS accounts and who do not bring their own laptops, guest accounts will be provided.

Note: DOMJudge runs on an Ubuntu 14.04 LTS distro. All solutions then are compiled against the freely available compiler or interpreter for that distro. If you use special packages beyond your language's standard library or you use modified compilers or interpreters on your own machine, we cannot guarantee it will run precisely the same on the judging machine.

4   DOMJudge

4.1   Summary

Here we include a short summary of the DOMJudge system interface, the webportal by which you will submit your solutions for automated judging. The single constraint of this system is that your programs **must** be written to accept their input from `stdin` and write their output to `stdout` (also sometimes referred to as *writing to the console*). We talk a little more about what this means in the examples portion of this section. What follows is an introduction to the interface.

4.2   Interface

A web URL will be provided to you from which you will gain access to the DOMJudge submission interface. From your team page, <http://example.com/domjudge/team>, you will select **Select file...** in the left column and select the file of your source code to submit. By default, the problem is selected from the base of the filename and the language from the your file's file extension.

Viewing scores, submissions, and sending and reading clarification requests is done through the web team interface, <http://example.com/domjudge/team>.

The left column of the team interface shows your teams's row in the scoreboard: your position and the which problems you attempted and which you solved. Via th menu, you can view the public scoreboard page with scores of all the teams. The score column lists the number of problems solved and the total penalty time. Each cell in the problem column lists the number of submissions, and if th problem was solved as well as the time of the first correct submission for that problem since the contest start. Submissions can have the following resuts:

CORRECT: The submission passed all tests: you solved this problem!

COMPILER-ERROR: There was an error when compiling your program. On the submission details page you can inspect the exact error (this option might be disabled).

TIMELIMIT: Your program took longer than the maximum allowed time for this problem. Therefore it has been aborted. This might indicate that your program hangs in a loop or that your solution is not efficient enough.

RUN-ERROR: There was an error during the execution of your program. This can have a lot of different causes like division by zero, incorrectly addressing memory (e.g. by indexing arrays out of bounds), trying to use more memory than the limit, etc. Also check that your program exits with exit code 0!

NO-OUTPUT: Your program did not generate any output. Check that you write to standard out.

WRONG-ANSWER: The output of your program was incorrect. This can happen simply because your solution is not correct, but remember that your output must comply exactly with the specifications of the jury.

PRESENTATION-ERROR: The output of your program has differences in presentation with the correct results (for example in the amount of whitespace). This will, like WRONG-ANSWER, count as an incorrect submission. This result is optional and might be disabled.

TOO-LATE: Bummer, you submitted after the contest ended! Your submission is stored but will not be processed anymore.

You are allowed to communicate with the jury, we omniscient few who oversee the entirety of the competition namelessly, and behind the inscrutability of teletype terminals, through the use of clarifications. These can be found in the right column of your team page. Both clarification replies from the jury and requests sent by your are displayed there.

There is also a button to submit a new clarification request to the jury. This request is only readable for the jury and they will respond as soon as possible. If the jury deems an answer relevant to everyone, it will be viewable to everyone.
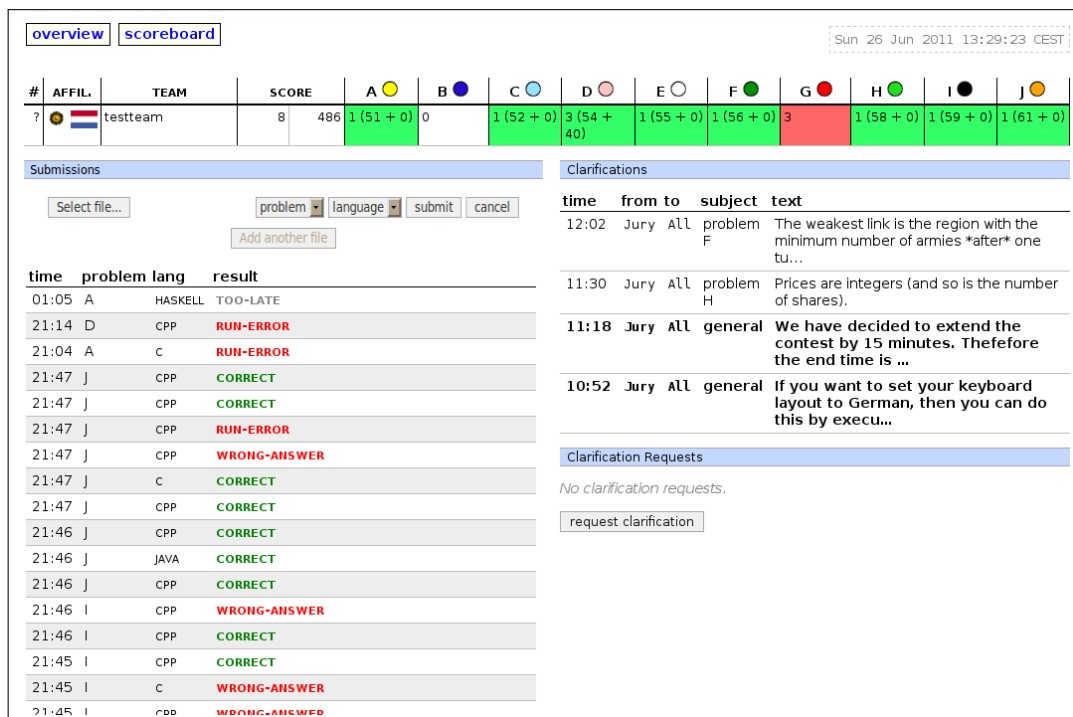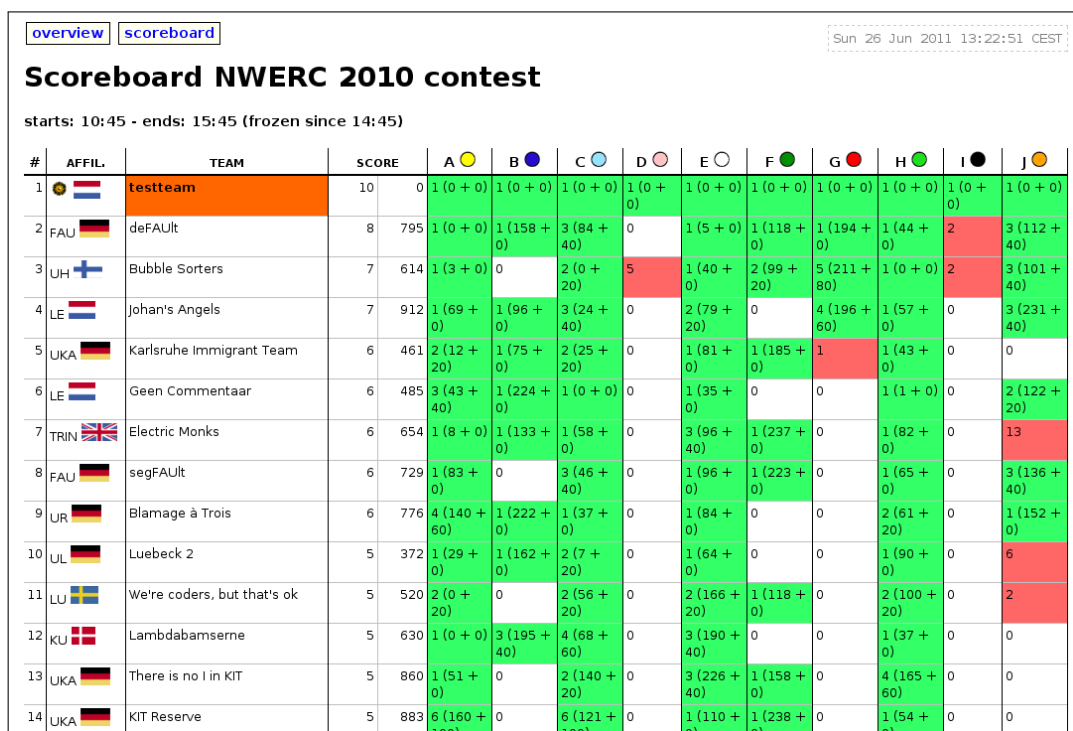
Figure 1: the web team interface overview page

Figure 2: the scoreboard webpage

4.3 Testing

The DOMJudge system is fully automated. Once input/output files have been prepared to verify your program, no other human element is necessary in scoring besides clarifications on the part of the jury. But this is how it works:

With the web interface you can submit the source code of your solution to the jury. Note that you must send your source code, not a compiled version of the program. Your solution is then sent over the network to a judging computer that will automatically compile or interpret your solution and pipe the proper input file to the program. The output of your program will then be compared against the validated output file. If your output matches the expected output, your program is correct. If not, you will receive some information about what went wrong before you go back to the drawing board.

Note: because DOMJudge provides testing inputs as files from `stdin`, you should not write your programs to expect input from a human source, such as through the keyboard.

On your local machine, you could test each of your solutions either in a Bash terminal environment, or by using the `stdin` environment variable of your preferred IDE. To test your solutions against sample inputs, copy the input to a text file like `input.txt`. Then invoke your interpreter on your solution or run your compiled executable, and use the caret operator, `<`, to pipe the file into the program's `stdin`. A terminal command like one of the following should work:

```
python my-solution.py < input.txt
```

```
./my-c-executable < input.txt
```

Your program should then print the results to the terminal for you, that is, to `stdout`.

5 Input/Output Examples

Here we define how you must write your program to read from `stdin` and write to `stdout` for the languages supported by this competition. We do this through annotation of two example programs. The purpose of this section is to expose you to the common input/output functions of your language that will allow your program to accep intput from DOMJudge and write output back to it.

5.1 Hello World

Description: Write a program to send a friendly greeting!

Input: Write a program to read a list of names headed by an integer from `stdin`. The first line will be an integer representing the number of lines to read. Each subsequent line will be a single name.

Output: On a new line for each, print out to `stdout` the name you read as part of the string "Hello <name>!".

Sample:

input	outout
3 World Alan Ada	Hello world! Hello Alan! Hello Ada!

A solution for the problem in C:

```c
#include <stdio.h>

int main() {
    int ntests;
    char name[100];

    scanf("%d\n", &ntests);

    int i;
    for (i = 0; i < ntests; i++) {
        scanf("%s\n", name);
        printf("Hello %s!\n", name);
    }

    return 0;
}
```

Here, we note that C uses the `scanf` function to read from `stdin` and `printf` to write to `stdout`. It declares an integer variable `ntests`, and `scanf` reads in a value from the console and assigns that to `ntests` (we note that `scanf` really uses a *reference* to `ntests`, that's what the *and* sign means, but that's special C stuff). `scanf` is later used with a character array that will hold our names. Our character array's size is 100, which is just arbitrarily large so we can hold any name we might read. We then use the `printf` function to print out our strings to `stdout` as we loop through them. We have to always `return 0` or else our program will **error**!

A solution in C++:

```cpp
#include <iostream>
#include <string>

using namespace std;

int main() {
    int ntests;
    string name;

    cin >> ntests;
    for (int i = 0; i < ntests; i++) {
        cin >> name;
        cout << "Hello " << name << "!" << endl;
    }

    return 0;
}
```

C++ uses `cin >>` to push `stdin` lines into a variable, and uses `cout >>` to print out lines to `stdout`. C++ also has real strings, not just character arrays. Other than that, the logic of this solution is the same as the solution in C.

A solution in Java:

```java
import java.io.*;

class Main {
    public static BufferedReader in;

    public static void main(String[] args) throws IOException {
        in = new BufferedReader(new InputStreamReader(System.in));

        int nTests = Integer.parseInt(in.readLine());

        for (int i = 0; i < nTests; i++) {
            String name = in.readLine();
            System.out.println("Hello " + name + "!");
        }
    }
}
```

For Java, we first note our class **needs** to be named *Main*. Java uses a `BufferedReader` object that it imports from `java.io.*`, which it uses to gather input from `System.in`. Using the command `System.out.println(...)` will let you write to `stdout`.

A solution in Python 2.7:

```python
import fileinput
import sys

num = sys.stdin.readline()

for line in fileinput.input():
    print "Hello " + line.strip() + "!"
```

For Python 2.7, we import `fileinput` and `sys` to get our input from `stdin`. Python has some looping mechanisms that will allow us to cleverly not need the integer representing the number of lines, the command on line 4 reads it in, but does nothing with it. The special Python `for` loop on line 6 will loop over all the lines from `stdin` and store each one temporarily in the variable `line`. It's worth noting that if we read in the lines this way, we need to strip off the hidden `newline` character that separates each line of input from the string we've read in. `strip()` does that for us. Then we just print it out to `stdout` with the aptly named `print` function.

A solution in Python 3:

```python
import fileinput
import sys

num = sys.stdin.readline()

for line in fileinput.input():
    print("Hello " + line.strip() + "!")
```

Python comes in two common versions. Python 2.7 and Python 3 are just different enough that it matters. All we notice here is that the Python 3 `print` command wraps its contents in parenthesis. There are some other differences too, so make sure you know which version of Python you're using.

Note: for both versions of Python, it's common to include *interpreter directives* at the top of the file, that may look like `#!/usr/bin/python`. You must **not** include a line like this in your submission. DOMJudge will be mad at you. If this doesn't look familiar to you anyway, just forget about it and go on with your day.

A solution in Ruby:

```ruby
1  $stdin.each_line.each_with_index do |line, i|
2      if i == 0
3          next
4      end
5      puts "Hello #{line.strip}!"
6  end
```

Ruby uses a special variable `$stdin` to represent console input. The method `each_line` grabs all the lines from `stdin` at one time, and the method `each_with_index` allow us to keep track of our position in the loop. Since the first line is just a silly integer, we can use the index of 0 to skip it, then we can do what we'd like with subsequent lines. In our case, we use the function `puts` to write to `stdout`. You'd likely replicate this looping pattern if you decide to use Ruby.

A solution in PHP:

```php
1  <?php
2
3  fgets(STDIN);
4
5  while($line = fgets(STDIN)) {
6      $name = rtrim($line);
7      echo "Hello $name!\n";
8  }
9
10 ?>
```

PHP can simply iterate over `stdin` by looping as long as `fgets(STDIN)` is not null. Since PHP does not really need to allocate arrays of fixed size, we can in this case ignore the first line of input, which is just an integer indicating the number of subsequent lines, so we call `fgets` and do not assign the returned value.

A solution in JavaScript

```javascript
 1  var fs = require('fs');
 2  var BUFSIZE=256;
 3  var buf = new Buffer(BUFSIZE);
 4  var bytesRead;
 5
 6  var stdin = '';
 7
 8  while (true) {
 9      bytesRead = 0;
10      try {
11          bytesRead = fs.readSync(process.stdin.fd, buf, 0, BUFSIZE);
12      } catch (e) {
13          if (e.code === 'EAGAIN') {
14              console.error('ERROR: interactive stdin input not supported.');
15              process.exit(1);
16          } else if (e.code === 'EOF') {
17              break;
18          }
19          throw e;
20      }
21      if (bytesRead === 0) {
22          break;
23      }
24      stdin += buf.toString(null, 0, bytesRead);
25  }
26
27  // Insert code here. STDIN is captured in "stdin" variable
28  var sayHello = function(name) {
29      console.log("Hello " + name + "!");
30  }
31
32  var i = 0;
33  stdin.split(/\n/).forEach(function(item) {
34      if (i > 0 && i < stdin.split(/\n/).length - 1) {
35          sayHello(item);
36      }
37      i += 1;
38  });
```

JavaScript is special. By default Node.js performs reading from `stdin` asynchronously, but this is in conflict with how DOMJudge runs. Lines 1 through 25 allow us to read from `stdin` synchronously. This input is put in the variable named `stdin` declared on line 6. This code can be replicated into all your solutions. One thing to note however, is that when we separate the `stdin` input variable by newlines, we end up with an array that has the first integer of input, and a trailing empty string, so when we iterate over each line with `forEach`, we keep track of our position with a variable `i`. This way, we only print what we need to. By using the code at the beginning of the file, and keeping in mind the trailing empty string of newline separated input, JavaScript can be just as speedy for coding as any other language allowed by the competition.

5.2 Fraction Simplification

Description: Write a program to simplify fractions.

Input: Write your program to read pairs of integers from `stdin`. On each line of input, there will be a pair of space separated integers. The first will represent the `numerator` of the fraction, the second will represent the `denominator`. The end of input will be signaled by a `0`. For a pair of integers, you will never be given a numerator or denominator with a value of `0`. The only `0` given as input will be the one that signals the end of input.

Output: Write to `stdout` the simplified fraction, with the numerator separated from the denominator by a *forward slash*, `/`. Each simplified fraction should be separated from the next by a newline. If the simplified fraction would be a single integer, write it without a denominator.

Sample:

input	outout
18 12 28 49 99 11 13 5 0	3/2 4/7 9 13/5

This example, though mathematically intuitive, is really quite a bit more complicated, reflecting the higher level of difficulty for the competition. Still, it makes a very instructive example of how to handle input for DOMJudge.

The general strategy for this program will be to loop over input until we encounter our *sentinel value* of `0`, calculate the Greatest Common Divisor of the two integers in the fraction, divide both by that GCD, and then print out the fraction formatted correctly. The method we choose to compute the GCD is straightforward: check all the integers between 1 and the smaller number of the fraction. Whichever is the largest that divides both the numerator and the denominator without a remainder is the GCD. A more clever method of computing GCD's exists called the Euclidean Algorithm. You can check it out if you feel adventurous, but we opt for a simpler one here for clarity, and hey, you should use whatever makes sense to you fastest when competing.

A solution for the problem in C:

```c
#include <stdio.h>

int main() {
    int numerator;
    int denominator;
    int gcd;

    while(1) {
        scanf("%d ", &numerator);
        if (numerator == 0) {
            break;
        } else {
            scanf("%d\n", &denominator);
            if (numerator % denominator == 0) {
                printf("%d\n", numerator / denominator);
            } else {

                gcd = find_gcd(numerator, denominator);

                if (gcd == 1) {
                    printf("%d/%d\n", numerator, denominator);
                } else {
                    int simple_num = numerator / gcd;
                    int simple_den = denominator / gcd;
                    printf("%d/%d\n", simple_num, simple_den);
                }
            }
        }
    }

    return 0;
}

int find_gcd(int num, int den) {
    int smallest;
    if (num >= den) {
        smallest = den;
    } else {
        smallest = num;
    }

    int i;
    int gcd;
    for (i = 1; i <= smallest; i++) {
        if (num % i == 0 && den % i == 0) {
            gcd = i;
        }
    }
    return gcd;
}
```

A solution for the problem in C++:

```cpp
#include <iostream>
#include <string>

using namespace std;

int find_gcd(int num, int den) {
    int smallest;
    if (num >= den) {
        smallest = den;
    } else {
        smallest = num;
    }

    int gcd;
    for (int i = 1; i <= smallest; i++) {
        if (num % i == 0 && den % i == 0) {
            gcd = i;
        }
    }
    return gcd;
}

int main() {
    int numerator;
    int denominator;
    int gcd;

    while(true) {
        cin >> numerator;
        if (numerator == 0) {
            break;
        } else {
            cin >> denominator;
            if (numerator % denominator == 0) {
                cout << numerator / denominator << endl;
            } else {

                gcd = find_gcd(numerator, denominator);

                if (gcd == 1) {
                    cout << numerator << "/" << denominator << endl;
                } else {
                    int simple_num = numerator / gcd;
                    int simple_den = denominator / gcd;
                    cout << simple_num << "/" << simple_den << endl;
                }
            }
        }
    }

    return 0;
}
```

A solution for the problem in Java:

```java
import java.io.*;
import java.util.Scanner;

class Main {
    public static void main(String[] args) throws IOException {
        Scanner in = new Scanner(System.in);
        int numerator;
        int denominator;
        int gcd;

        while(true) {
            numerator = in.nextInt();
            if (numerator == 0) {
                break;
            } else {
                denominator = in.nextInt();
                if (numerator % denominator == 0) {
                    System.out.println(numerator / denominator);
                } else {

                    gcd = findGCD(numerator, denominator);

                    if (gcd == 1) {
                        System.out.println(numerator + "/" + denominator);
                    } else {
                        int simple_num = numerator / gcd;
                        int simple_den = denominator / gcd;
                        System.out.println(simple_num + "/" + simple_den);
                    }
                }
            }
        }
    }

    public static int findGCD(int num, int den) {
        int smallest;
        if (num >= den) {
            smallest = den;
        } else {
            smallest = num;
        }

        int gcd = 1;
        for (int i = 1; i <= smallest; i++) {
            if (num % i == 0 && den % i == 0) {
                gcd = i;
            }
        }
        return gcd;
    }
}
```

A solution for the problem in Python 2.7:

```python
import fileinput
import sys

for line in fileinput.input():
    if line.strip() == "0":
        break
    else:
        num, den = line.split(' ')
        numerator, denominator = int(num), int(den)
        if numerator % denominator == 0:
            print numerator / denominator
        else:
            smallest = numerator
            if numerator >= denominator:
                smallest = denominator

            gcd = 1
            for i in range(1, smallest):
                if numerator % i == 0 and denominator % i == 0:
                    gcd = i

            if gcd == 1:
                print str(numerator) + "/" + str(denominator)
            else:
                simple_num = numerator / gcd
                simple_den = denominator / gcd
                print str(simple_num) + "/" + str(simple_den)
```

A solution for the problem in Python 3:

```python
import fileinput
import sys

for line in fileinput.input():
    if line.strip() == "0":
        break
    else:
        num, den = line.split(' ');
        numerator, denominator = int(num), int(den)
        if numerator % denominator == 0:
            print(numerator // denominator)
        else:
            smallest = numerator
            if numerator >= denominator:
                smallest = denominator

            gcd = 1
            for i in range(1, smallest):
                if numerator % i == 0 and denominator % i == 0:
                    gcd = i

            if gcd == 1:
                print("{}/{}".format(numerator, denominator))
            else:
                simple_num = numerator // gcd
                simple_den = denominator // gcd
                print("{}/{}".format(simple_num, simple_den))
```

A solution for the problem in Ruby:

```ruby
$stdin.each_line do |line|
    if line.strip == "0"
        break
    end
    num, den = line.split(' ')
    numerator, denominator = num.to_i, den.to_i
    if numerator % denominator == 0
        puts "#{numerator / denominator}"
    else
        smallest = numerator
        if numerator >= denominator
            smallest = denominator
        end

        gcd = 1
        (1..smallest).each do |i|
            if numerator % i == 0 and denominator % i == 0
                gcd = i
            end
        end

        if gcd == 1
            puts "#{numerator}/#{denominator}"
        else
            simple_num = numerator / gcd
            simple_den = denominator / gcd
            puts "#{simple_num}/#{simple_den}"
        end
    end
end
```

6   Going Forward

Now that you know how DOMJudge expects you to use Input/Output, feel free to log into the guest account of our test DOMJudge instance and review the practice problems there. You can also get a feel for the environment you'll use to judge submit your programs. We also keep a running list of repos for our previous years' problems: https://github.com/acmatku. These folders are named ku-acm-competition-****, and they have dockets, input/output files, and sample solutions.