Introduction

Welcome

Hello, and weclome to Competitive Programming. In order to test your program, the online judge will send it input and read output from it to see if it matches the expected result. You will need to be sure your program can handle input and output accurately and efficiently.

Objectives

Your objectives are to be able to correctly write input routines for three kinds of test inputs; to use `scanf` and `printf` properly for various types of variables, and to know how to write code for interactive tests.

Input Routines

The Kinds of Inputs

When an online judge runs your program, the majority of them will test your program by sending text to it via the standard input. In other words, it will simulate keyboard entry. In return, your program will communicate its results by printing to standard output using regular `printf` statements. We usually think of standard output as being the screen, but the judge captures this output in order to check it.

Sometimes the input is complicated enough that the judge will run your progr once for each test case. This makes things simple.

But you will see that many problems expect your code to be able to run multiple test cases in one execution. Your program also needs to be able to stop by itself when the test cases are over, or else you'll get a Time Limit Exceeded error.

Scenario 1 — Specific Number of Test Cases

To talk about this let's assume the problem wants you to add two numbers together and print the result. You don't know how many pairs of numbers you will get though.

The easiest scenario is when the input indicates the number of test cases up front. This is almost always an integer in the first line. You can give it a name such as `t` or `n` if you want, but be careful that none of the other input variables use the same name. I like to use a capitalized `N` or a word like `cases`. You only need to type it four times, so it's okay if the variable name is a little longer. You just want to be sure to decrement it each time you run the loop and to be sure *not* to modify it accidentally in some other part of your code.

If you forget to decrement, it will run all the cases, and then keep going. The call to `scanf` will fail, most likely leaving the inputs the same. Your code will keep outputting the same result over and over, causing either a wrong answer or time limit exceeded.

Scenario 2 — End of Tests Marker

Another possibility is that you get an end of input marker. This is common when each test case itself needs a parameter for a number of inputs. Usually the problem uses a zero or minus one for the end of input marker.

Let's suppose we use two minus ones; here is one way to write it. We can test the input in the body of the while loop and `break` out if we hit the end of tests marker.

End of Tests Marker, pt 2

You can also do the scan inside the test case. The `scanf` call will return the number of variables it read, so if it succeeds it will be counted as true.

Explicit EOF

Sometimes you are given no warning that the test suite is done; the input simply stops. If the input ends, `scanf` returns an `EOF` character, and you can simply watch for that.

Using `scanf` and `printf`.

Why use them?

You might be wondering why we don't just use `cin` and `cout`. For many problems, that will work just fine. However, you need to know that `cin` and `cout` are slower, and I have solved problems both on Code Forces and UVa that get Time Limit Exceeded if you use them, but were accepted when using `scanf` and `printf`. This can happen when the time limit for the solution is very short and there is a lot of input.

Another reason to prefer `printf` and `scanf` is they give you a lot of control over the format, both of the input and the output.

Here is a table of a few common input formaters. You will want to study the C++ reference page for `scanf` and `printf` to see what else is available, particularly the options for floating point.

Regular like expressions

Many people don't know that `scanf` gives you something similar to regular expressions for input.

If there is a non-space literal, `scanf` will read that literal exactly. Here is an example of reading two integers but separated by a comma and wrapped by parenthesis.

If there is a space before a non-space or a formatter, then `scanf` will discard any leading whitespace. The second example allows for spaces before the open parenthesis the close parenthesis, and before the integers, but *not* before the comma.

Finally, `scanf` allows ranges in regular expression like notation. Suppose you wanted to read a binary strings followed by a space and then a string of vowels. You can do that with the third example.

Interactive Problems

Interactive Problems

ICPC is beginning to test a new class of problems: interactive problems. The idea is that the judge gives input to your program in response to your program's output. At the time I made this video, such a problem had not yet shown up in a real problem set, but they have used them in practice sets, which indicates that they are testing them out.

Interactive problems aren't really more difficult to write than normal ones, but they could more for more interesting kinds of problems, such as games. The one thing you really have to remember is to flush the output every time you print, or perhaps before any read operation. This is because most systems buffer input and output, and it's possible to have your output stuck in a buffer and never make it to the judge, resulting again in Time Limit Exceeded.

Next
Transcript — Template →