

# Why did I score zero?

## *A guide to (some) common mistakes*

### New Zealand Olympiad in Informatics

There are many traps that might cause your submission to score zero. Don't panic – it might be something very simple! Below are the most common reasons we've seen.

#### Before you begin:

We assume that you've already tested your program with the sample input, and that your program seems to work on those. If you have not, **make sure to always test with the sample before submitting!** It's much easier to debug your program on your computer first than to submit and figure out what's gone wrong later.

If you're confused about what the verdict your submission got means (e.g. Wrong Answer, Fatal Signal, Runtime Error), see [this document first](#).

## 1 Input/Output format

Every submission you make to our judging server is judged by a computer. This means that you will be taking in input and writing output in a very specific format defined in the problem statement. If your program does not handle input/output in **exactly the same format** described in the problem statement, then the computer judging your code will not understand it. It is unlikely you will get any marks if this happens.

One common trap is the use of input prompts, such as passing a string argument to Python's `input()` function. For more information on this, [see this document](#). Remember that you are writing your program for a computer, not a human, to use!

Another common trap is printing strings (such as a sentence) with very subtle differences to the expected output format. This can include:

- Missing or extra full stops:

Actual Output	Size: 16 Bytes	Expected Output	Size: 15 Bytes
1 Greetings NZIC.		1 Greetings NZIC	

- Extra whitespace (note the extra space between 'Greetings' and 'NZIC'):

Actual Output	Size: 16 Bytes	Expected Output	Size: 15 Bytes
1 Greetings NZIC		1 Greetings NZIC	

- Misspellings:

Actual Output	Size: 14 Bytes	Expected Output	Size: 15 Bytes
1 Grettins NZIC		1 Greetings NZIC	

These can all lead to situations where it *looks* like your code is giving the correct answers, but in fact is giving a slightly different (and incorrect) answer. Check the output your code gives for the sample inputs, compare it to the expected output, and make sure they're actually the same.

## 2 Submitting the wrong language

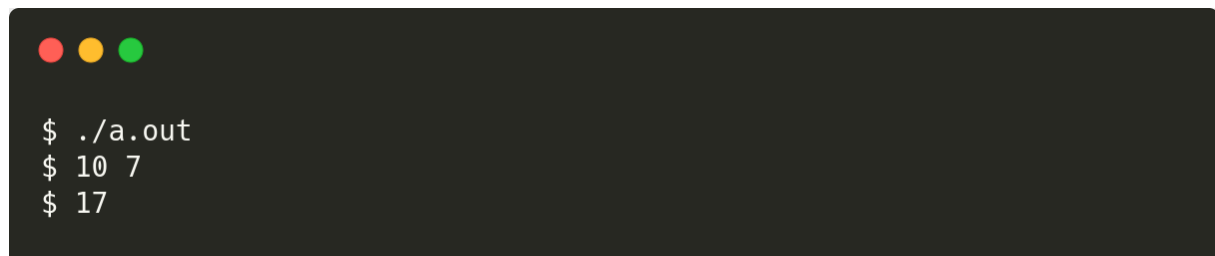
This one's pretty straight-forward – make sure you selected the language your program is in. Also make sure that your program can run in the *version* of the language you selected. We try to support relatively up-to-date version of languages, but we don't guarantee that we have the latest version of every language. It might be that your program uses new features that were released after the version we have.

## 3 Language-specific traps

There may be details specific to your language that are causing issues. For example, Java submissions must have a class named `Main` that contains a static method `main`. Check out [this problem set](#) and open the problem for your language of choice for more specific instructions.

## 4 Undefined Behaviour

Though rare, it is possible that you've come across a case where your code works for the sample input on your computer, but fails on the same sample input when you submit to the server. For example, the output on your computer is:



```
$ ./a.out
$ 10 7
$ 17
```

But the output on the server is:

Actual Output	Size: 11 Bytes	Expected Output	Size: 3 Bytes
1 1923012148		1 17	

This is most likely to happen if you program in C or C++, which both leave many opportunities to cause **undefined behaviour**. Undefined behaviour occurs when your program has behaviour that is unpredictable – that can vary between different computers, compilers, or executions of the program. Thus, while your program may happen to work well on your machine, it fails on our judging server.

This could have occurred in a variety of ways – most commonly, using variables before they are initialized, or accessing out-of-bounds memory (such as indexing past the end of an array). Check that all your variables are initialized – including values in arrays – and check all the places you try to index an array.

For example, note the following C++ code, which takes in two number as input and outputs their sum:

```

#include <iostream>

int main() {
    int a, b, c;
    cin >> a >> b;

    int sum = a + b + c;
    cout << sum << endl;
}

```

Notice that the variables `a`, `b` and `c` are not initialized - that is, we declare them without setting an initial value. `a` and `b` are later assigned values by `cin`, but `c` is never assigned a value. So what's the value of `c`?

It depends - this is undefined behaviour! Depending on the specifics of the computer, operating system, and even what other programs are running at the same time, it will vary. If you get 'lucky' and `c` happens to be 0 on your computer, then the output will be correct. But if, on the judging server, `c` happens to be a random value such as 1923012131, then you will get garbage output.

In general, it is good practice to always specify an initial value for your variables (e.g. `int a=0, b=0, c=0`; Failing that, you should always make sure that every variable you use always gets assigned a value before you use it.

Different output on the server compared to your machine, especially if the output seems random/garbage, is a telltale sign of undefined behaviour. Another possible sign of undefined behaviour in C and C++ is a fatal signal:

Sample #1.2		0.003 seconds	412.0 kB	Fatal Signal	
Input	Size: 10 Bytes	Actual Output	Size: 0 Bytes	Expected Output	Size: 2 Bytes
1	1 1 0			1 1	
2	5				
3	5				

This is an even more extreme version of undefined behaviour, where the same code 'works' on one computer but actually *crashes* on another! Most commonly, this is caused by accessing an array with an out-of-bounds index.

## 5 Bugs or an incorrect approach

Unfortunately, it could be that your solution has the correct approach but contains some bugs, or that your approach to the problem is incorrect. Try to come up with your own test cases and see if your program passes those. Think of potential edge cases - what are the most extreme examples of test cases you can think of? Make sure you read the problem very carefully, in case there was some requirement you missed.

## 6 We made a mistake

In some rare cases, the problem statement or test data might contain errors. If you're completely stuck on a problem and you don't think it's the fault of your program, you

can contact us at [nzic@nzo.org.nz](mailto:nzic@nzo.org.nz).