Web Asm Challenge README

Hello world

Go to https://asm.web.ctfcompetition.com/?debug=1 and put this code:

```
.data

.code
&main:
prt int 1 string hello, world.
```

Make sure there is an empty line between .data and .code.

What this does is:

1. Define an empty data section
2. Define a code section
3. Define a label called "main" in the code section
4. Print to the File Descriptor #1 the string hello, world.

When you click on test, you will see that your javascript console will say:

```
Starting tests..
hello, world.
hello, world.
hello, world.
```

This is because it ran your code three times, and the file descriptor #1 points to console.log.

Let's try something else, change the code to:

```
.data

.code
&main:
prt int 3 string hello, world.
```

What should happen now is that you will get a JavaScript exception. That is because there is no filedescriptor "3".

Now let's try something else.

```
.data
$input mem 1

.code
&main:
get $input int 0
prt int 1 $input
```

What this does is use the "get" instruction to get data from the File Descriptor 0, then it moves that data into the variable called $input, and then it prints $input into the File Descriptor 1.

If you run this code, you will get:

```
Starting tests..
hello, world
```

You might be wondering where the "hello, world" came from! The answer is that when you click on Test, a series of test cases are ran against your code. You can see the full list of test cases here: https://asm.web.ctfcompetition.com/js/testcases.data.js

As you can see the first test case for the hello_world challenge has the input "hello, world". The other two test cases are empty, which is why the test printed 2 empty lines too.

If you change the challenge to Fibonacci, for example, the code will show you:

```
1
3
2
5
```

One thing that might be interesting, is that your code is ran simultaneously 4 times, which is why the print order wasn't in order.

Now that you understand the environment, let's solve our first challenge!

Hello, world.

Let's go back to https://asm.web.ctfcompetition.com/?debug=1 and this time, leave the code and don't modify it, and just click Test, you should see an alert that says "Your code is correct!" and then another one that says "Well done, now make your code smaller".

This is because we already implemented the solution to the first challenge, but it is a little bit too long. Let's make it shorter!

Find the line that says:

```
$reserved mem 256
```

What that line is doing, is reserving 256 variables at the offset of $reserved.

That is a waste of space! We only need one variable. Let's change that to say:

```
$reserved mem 1
```

What this will do, is reserve just one variable in memory. Click test and you will see an alert "Your code is correct!" followed by another one that says:

> Your answers:>hello, world!,>!,>!!!!!!

This means that you have code that is small enough, so you passed this challenge!

You can submit your code by clicking the Submit button, and you should see two messages saying the same thing the alert() says.

Pow

We are approaching the end of this README, the last thing missing is functions.

In this language, you call functions by referencing the label in the code section. To show you, we'll solve the second challenge called **pow**. See the annotated code below.

Instruction	**Description**
`.data`	(data section)
`$max mem 1`	Declare variable $max
`$result mem 1`	Declare variable $result
`$number mem 1`	Declare variable $number
`.code`	(code section)
`&main:`	Declare &main label
`get $number int 0`	Get from FD 0 a value and put it in $number
`get $max int 0`	Get from FD 0 a value and put it in $max
`mov $result &pow`	Move to $result the return value after calling &pow
`ret $result int 0`	Return $result with no errors
`&pow:`	Declare &pow label
`sub $max int 1`	Subtract 1 from $max
`jez &end $max`	If $max is equal to 0, jump to &end

`mul $number &pow`	Multiply $number by the return value of &pow
`&end:`	Declare &end label
`ret $number int 0`	Return $number with no errors

As you can see, this multiplies the number received from the input with itself up to $max times. In JavaScript this code could be written as:

```
var $max, $result, $number;
function main() {
  $number = prompt();
  $max = prompt();
  $result = pow();
  return $result;
}

function pow() {
    $max -= 1;
    if (!($max == 0)) {
        $number *= pow();
    }
    return $number;
}
```

Now that you know how to call functions you know everything you need to know to solve the challenges Fibonacci and Primes.

Good luck!

Annex

An advanced feature of the language is the use of direct memory access. While it should be rarely needed, here is how it works:

```
.data
$foo string hello world

.code
&main:
prt int 1 &deref

&deref:
ret int 1 int 0
```

The code above will print "hello world", and the reason for that is that in the &deref function we are returning the value at memory position #1, and since $foo is the first variable declared in the program, it returns what is in there.

You can also write to memory referenced by address, for example:

```
.data
$foo string hello

.code
&main:
prt int 1 $foo
mov int 1 string world
prt int 1 $foo
```

What this code will do is print the $foo variable (declared as "hello"), and then it will modify memory in position 1 to say "world", then it will print $foo again.

You can also reference the code section. Here's an example of that:

```
.data

.code
&main:
mov int 1 &deref
ret int 0 int 0

&unreachableCode:
prt int 1 string unreachable code?
ret int 0 int 0

&deref:
ret int 3 int 0
```

In this code &deref points to memory 3, and since we don't have anything in the data section, it points to the third instruction, which is the line that prints "unreachable code".