Lab Report 1 -- 9/17/2021

To set up your Emscripten environment first navigate to the [Emscripten Website](). On the site you will find a cmd line command to clone the core Emscripten SDK.

In the cmd line navigate to the desired location and enter the following commands:

```
# Get the emsdk repo
git clone https://github.com/emscripten-core/emsdk.git

# Enter that directory
cd emsdk
```

```
# Fetch the latest version of the emsdk (not needed the first time you clone)
git pull

# Download and install the latest SDK tools.
./emsdk install latest

# Make the "latest" SDK "active" for the current user. (writes .emscripten file)
./emsdk activate latest

# Activate PATH and other environment variables in the current terminal
source ./emsdk_env.sh
```

This will clone the repo for Emscripten onto your machine, update it with the new releases and set the latest tools to active.

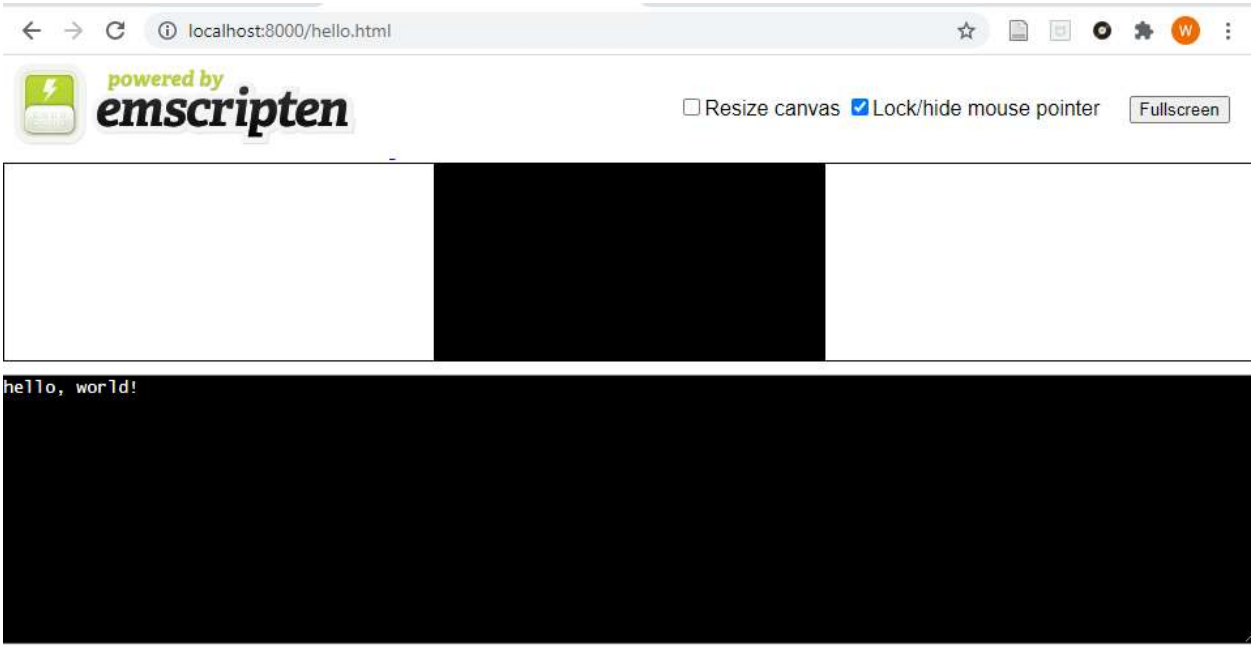
After Emscripten was downloaded onto my machine, I checked the version and tested the initial hello world program in order to confirm that the compiler was up and running correctly. The results of this are shown below:

```
PS C:\Users\wyatt\onedrive\desktop\School\Fall 2021\Advanced App\emsdk\upstream\emscripten> ./emcc -v
emcc (Emscripten gcc/clang-like replacement + linker emulating GNU ld) 2.0.29 (28ca7fb7ce895b21013212e4644a5794a15a76f9)

clang version 14.0.0 (https://github.com/llvm/llvm-project 8e284be04f2cd43a821289133a759afa2844f935)
Target: wasm32-unknown-emscripten
Thread model: posix
InstalledDir: C:/Users/wyatt/OneDrive/Desktop/School/Fall 2021/Advanced App/emsdk/upstream/bin
PS C:\Users\wyatt\onedrive\desktop\School\Fall 2021\Advanced App\emsdk\upstream\emscripten> ./emcc tests/hello_world.c
PS C:\Users\wyatt\onedrive\desktop\School\Fall 2021\Advanced App\emsdk\upstream\emscripten> node a.out.js
hello, world!
PS C:\Users\wyatt\onedrive\desktop\School\Fall 2021\Advanced App\emsdk\upstream\emscripten>
```

Running the node a.out.js test that the code was correctly transcribed into javascript. To test the WASM I opened a local host, and ran the output html file I generated. Steps to this are shown below:

```
PS C:\Users\wyatt\onedrive\desktop\School\Fall 2021\Advanced App\emsdk\upstream\emscripten> ./emcc tests/hello_world.c -
o hello.html
PS C:\Users\wyatt\onedrive\desktop\School\Fall 2021\Advanced App\emsdk\upstream\emscripten> python -m http.server
Serving HTTP on :: port 8000 (http://[::]:8000/) ...
::1 - - [17/Sep/2021 19:21:48] "GET /hello.html HTTP/1.1" 200 -
::1 - - [17/Sep/2021 19:21:48] "GET /hello.js HTTP/1.1" 200 -
::1 - - [17/Sep/2021 19:21:48] "GET /hello.wasm HTTP/1.1" 200 -
```



This confirms that emscripten also correctly transcribed the hello_world.c program into functionable WASM code.

Timing Analysis:

	A	B	C
1	JS No OPP (Microsec)	Wasm No Opp (Microsec)	
2	93665	30500	
3	100390	32100	
4	113485	29700	
5	87343	31500	
6	93062	56600	
7	86145	30100	
8	78071	31599	
9	90175	29900	
10	99065	30600	
11	104442	29800	
12	89065	31900	
13	88876	31100	
14	87456	30799	
15	94334	32600	
16	73570	29500	
17	107311	38799	
18	95613	32199	
19	97970	34100	
20	78361	33900	
21	101599	31900	
22	99169	31599	
23		Native	WASM
24	Significance Level	0.05	0.05
25	Mean	93293.66667	32895
26	Standard Deviation	9650.066239	5667.150456
27	Sample Size	20	20
28	Confidence Value	4229.250288	2483.692558
29	Confidence Interval	93292.66 +- 4229.25	32895.00 +- 2483.69

I tested running the transcripted native and wasm code 20 times per code base. The sheet above shows the results of these trials. From the trials we can see that the WASM code preforms on average 2.85x faster than the native code. This was as expected as WASM code takes lesss time to fetch, parse, and compile than native JS code.

Optimizations:

```
PS C:\Users\wyatt\onedrive\desktop\School\Fall 2021\Advanced App\emsdk\upstream\emscripten> ./em++ Sprint3/main.cpp -O2 -o output2.html
```

JS OPP 2 (Microsec)	Wasm Opp 2(microsec)	
90395	18000	
89180	5799	
100157	6400	
78223	6899	
90382	7200	
91671	7000	
83668	6600	
74202	7100	
73762	5700	
86482	6899	

	Native Opp 2	WASM Opp 2
Significance Level	0.05	0.05
Mean	85812.2	7759.7
Standard Deviation	7974.786992	3448.932619
Sample Size	20	20
Confidence Value	3495.040277	1511.533591
Confidence Interval	85812.20 +- 3495.04	7759.70 +- 1511.53

Using the Second Optimization feature impacted the wasm code significantly more than the native code. After optimization the wasm code now runs 11.06x faster than the native code. The optimized wasm code runs 4.24x faster than the unoptimized wasm code.

The optimized native code runs 1.08x faster than the unoptimized native code. The Wasm code is better optimized because it is closer to machine code than java script and has already gone through optimization on the server side.