

# Lab One

## 1. Emscripten Development Environment Setup

MacBook Pro

VSCode g++ compiler

### Preparation

1. Install XCode
2. Install git
3. Upgrade python from 3.7.9 to 3.9.7

### Install Emscripten SDK(in terminal)

1. `git clone https://github.com/emscripten-core/emsdk.git`
2. `cd emsdk`
3. `./emsdk install latest`
4. `./emsdk activate latest`
5. `source ./emsdk env.sh`

```
kehanzhang@KEHANS-MacBook-Pro Projects % git clone https://github.com/emscripten-core/emsdk.git
Cloning into 'emsdk'...
remote: Enumerating objects: 3385, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 3385 (delta 0), reused 0 (delta 0), pack-reused 3384
Receiving objects: 100% (3385/3385), 1.95 MiB | 1.84 MiB/s, done.
Resolving deltas: 100% (2220/2220), done.
kehanzhang@KEHANS-MacBook-Pro Projects % cd emsdk
kehanzhang@KEHANS-MacBook-Pro emsdk % ./emsdk install latest
Resolving SDK alias 'latest' to '3.1.20'
Resolving SDK version '3.1.20' to 'sdk-releases-upstream-d92c8639f406582d70a5dde27855f74ecf602f45-64bit'
Installing SDK 'sdk-releases-upstream-d92c8639f406582d70a5dde27855f74ecf602f45-64bit'..
Installing tool 'node-14.18.2-64bit'..
Downloading: /Users/kehanzhang/Desktop/Workspace/Projects/emsdk/zips/node-v14.18.2-darwin-x64.tar.gz from https://storage.googleapis.com/webassembly/emscripten-releases-builds/deps/node-v14.18.2-darwin-x64.tar.gz, 32076686 Bytes
Unpacking: /Users/kehanzhang/Desktop/Workspace/Projects/emsdk/zips/node-v14.18.2-darwin-x64.tar.gz to '/Users/kehanzhang/Desktop/Workspace/Projects/emsdk/node/14.18.2_64bit'
Done installing tool 'node-14.18.2-64bit'..
Installing tool 'python-3.9.2-64bit'..
Downloading: /Users/kehanzhang/Desktop/Workspace/Projects/emsdk/zips/python-3.9.2-3-macos-x86_64.tar.gz from https://storage.googleapis.com/webassembly/emscripten-releases-builds/deps/python-3.9.2-3-macos-x86_64.tar.gz, 31899321 Bytes
Unpacking: /Users/kehanzhang/Desktop/Workspace/Projects/emsdk/zips/python-3.9.2-3-macos-x86_64.tar.gz to '/Users/kehanzhang/Desktop/Workspace/Projects/emsdk/python/3.9.2_64bit'
Done installing tool 'python-3.9.2-64bit'..
Installing tool 'releases-upstream-d92c8639f406582d70a5dde27855f74ecf602f45-64bit'..
Downloading: /Users/kehanzhang/Desktop/Workspace/Projects/emsdk/zips/d92c8639f406582d70a5dde27855f74ecf602f45-wasm-binaries.tbz2 from https://storage.googleapis.com/webassembly/emscripten-releases-builds/mac/d92c8639f406582d70a5dde27855f74ecf602f45-wasm-binaries.tbz2, 349627729 Bytes
Unpacking: /Users/kehanzhang/Desktop/Workspace/Projects/emsdk/zips/d92c8639f406582d70a5dde27855f74ecf602f45-wasm-binaries.tbz2 to '/Users/kehanzhang/Desktop/Workspace/Projects/emsdk/upstream'
Done installing tool 'releases-upstream-d92c8639f406582d70a5dde27855f74ecf602f45-64bit'..
Done installing SDK 'sdk-releases-upstream-d92c8639f406582d70a5dde27855f74ecf602f45-64bit'..
kehanzhang@KEHANS-MacBook-Pro emsdk % ./emsdk activate latest
Resolving SDK alias 'latest' to '3.1.20'
Resolving SDK version '3.1.20' to 'sdk-releases-upstream-d92c8639f406582d70a5dde27855f74ecf602f45-64bit'
Setting the following tools as active:
node-14.18.2-64bit
python-3.9.2-64bit
releases-upstream-d92c8639f406582d70a5dde27855f74ecf602f45-64bit

Next steps:
- To conveniently access emsdk tools from the command line,
  consider adding the following directories to your PATH:
    /Users/kehanzhang/Desktop/Workspace/Projects/emsdk
    /Users/kehanzhang/Desktop/Workspace/Projects/emsdk/node/14.18.2_64bit/bin
    /Users/kehanzhang/Desktop/Workspace/Projects/emsdk/upstream/emscripten
- This can be done for the current shell by running:
  source "/Users/kehanzhang/Desktop/Workspace/Projects/emsdk/emsdk_env.sh"
- Configure emsdk in your shell startup scripts by running:
  echo "source "/Users/kehanzhang/Desktop/Workspace/Projects/emsdk/emsdk_env.sh"" >> $HOME/.zprofile
kehanzhang@KEHANS-MacBook-Pro emsdk % source ./emsdk_env.sh
Setting up EMSDK environment (suppress these messages with EMSDK_QUIET=1)
Adding directories to PATH:
PATH += /Users/kehanzhang/Desktop/Workspace/Projects/emsdk
PATH += /Users/kehanzhang/Desktop/Workspace/Projects/emsdk/upstream/emscripten
PATH += /Users/kehanzhang/Desktop/Workspace/Projects/emsdk/node/14.18.2_64bit/bin

Setting environment variables:
PATH = /Users/kehanzhang/Desktop/Workspace/Projects/emsdk:/Users/kehanzhang/Desktop/Workspace/Projects/emsdk/upstream/emscripten:/Users/kehanzhang/Desktop/Workspace/Projects/emsdk/node/14.18.2_64bit/bin:/Users/kehanzhang/.pyenv/versions/3.9.7/bin:/usr/local/Cellar/pyenv/2.3.4/libexec:/usr/local/Cellar/pyenv/2.3.4/plugins/python-build/bin:/Users/kehanzhang/.pyenv/shims:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Library/Apple/usr/bin
EMSDK = /Users/kehanzhang/Desktop/Workspace/Projects/emsdk
EM_CONFIG = /Users/kehanzhang/Desktop/Workspace/Projects/emsdk/.emscripten
EMSDK_NODE = /Users/kehanzhang/Desktop/Workspace/Projects/emsdk/node/14.18.2_64bit/bin/node
EMSDK_PYTHON = /Users/kehanzhang/Desktop/Workspace/Projects/emsdk/python/3.9.2_64bit/bin/python3
SSL_CERT_FILE = /Users/kehanzhang/Desktop/Workspace/Projects/emsdk/python/3.9.2_64bit/lib/python3.9/site-packages/certifi/cacert.pem
kehanzhang@KEHANS-MacBook-Pro emsdk %
```

### Test Installation and Hello World Project

1. Test installation of Emscripten: `emcc -v`
2. create `hello_world.c` file
3. `emcc test/hello_world.c`

generated two files: `a.out.js` and `a.out.wasm`

```
kehanzhang@KEHANS-MacBook-Pro emsdk % emcc test/hello_world.c
kehanzhang@KEHANS-MacBook-Pro emsdk % node a.out.js
hello, world!
```

#### 4. `emcc test/hello_world.c -o hello.html`

generated three files: hello.html, hello.js and hello.wasm



#### 5. Start a local server: `python -m http.server`

```
kehanzhang@KEHANS-MacBook-Pro ~ % python -m http.server
Serving HTTP on :: port 8000 (http://[::]:8000/) ...
::1 - - [10/Sep/2022 11:35:58] "GET / HTTP/1.1" 200 -
::1 - - [10/Sep/2022 11:35:58] code 404, message File not found
::1 - - [10/Sep/2022 11:35:58] "GET /favicon.ico HTTP/1.1" 404 -
::1 - - [10/Sep/2022 11:37:04] "GET /Desktop/ HTTP/1.1" 200 -
::1 - - [10/Sep/2022 11:37:09] "GET /Desktop/Workspace/ HTTP/1.1" 200 -
::1 - - [10/Sep/2022 11:37:11] "GET /Desktop/Workspace/Projects/ HTTP/1.1" 200 -
::1 - - [10/Sep/2022 11:37:16] "GET /Desktop/Workspace/Projects/lab-1-emsripten-Kehan-Zhang/ HTTP/1.1" 200 -
::1 - - [10/Sep/2022 11:37:18] "GET /Desktop/Workspace/Projects/CS7320-AI/ HTTP/1.1" 200 -
::1 - - [10/Sep/2022 11:37:20] "GET /Desktop/Workspace/Projects/emskd/ HTTP/1.1" 200 -
::1 - - [10/Sep/2022 11:37:27] "GET /Desktop/Workspace/Projects/emskd/hello.html HTTP/1.1" 200 -
::1 - - [10/Sep/2022 11:37:27] "GET /Desktop/Workspace/Projects/emskd/hello.js HTTP/1.1" 200 -
::1 - - [10/Sep/2022 11:37:27] "GET /Desktop/Workspace/Projects/emskd/hello.wasm HTTP/1.1" 200 -
```

#### 6. Find and open hello.html in "http://localhost:8000/"



## 2. Identify Code Base and Transpile

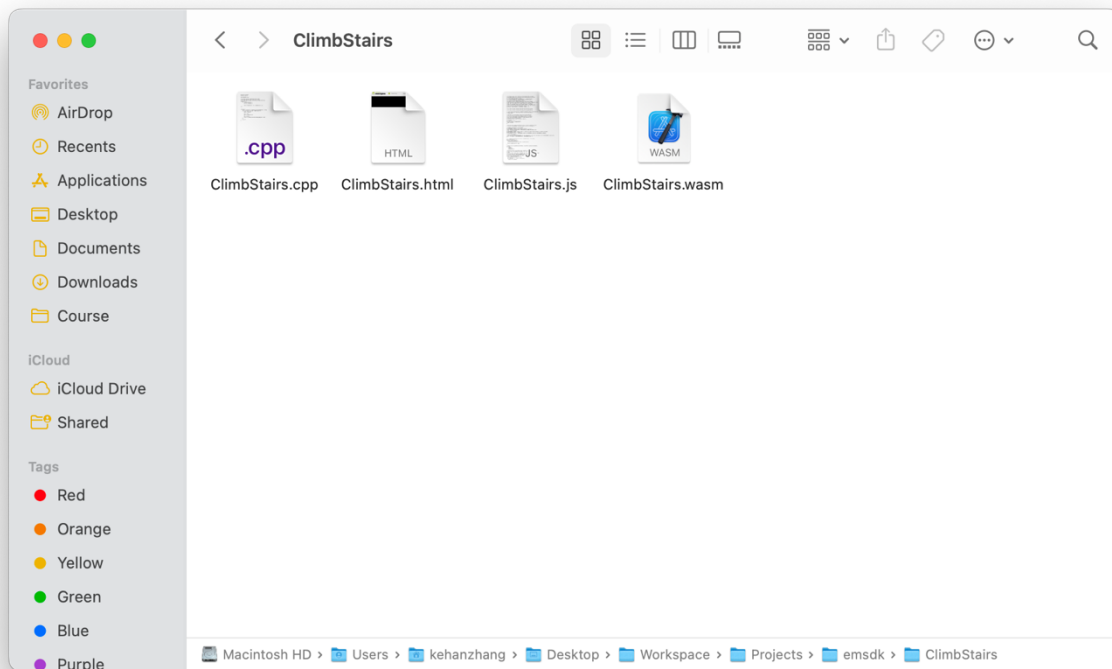
I selected an algorithm to solve Climbing-Stairs problem in a recursive way.

```
1  #include <iostream>
2  #include <time.h>
3
4  using namespace std;
5
6  // It takes n steps to reach the top of a stair.
7  // Each time you can either climb 1 or 2 steps.
8  // Calculate distinct ways you can take to climb to the top.
9  int climbStairs(int n) {
10     if (n == 1) return 1;
11     if (n == 2) return 2;
12
13     return climbStairs(n - 1) + climbStairs(n - 2);
14 }
15
16
17 int main() {
18     int n = 30;
19     cout << climbStairs(n);
20
21     return 0;
22 }
```

## Test Code Base Compiling with emcc

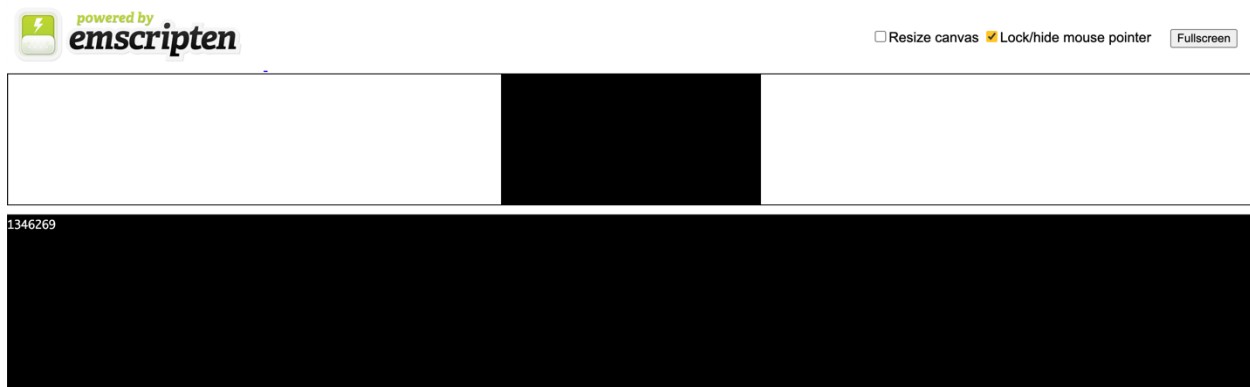
1. `emcc ClimbStairs.cpp -o ClimbStairs.html -s EXIT_RUNTIME`

The command above will generate three files: ClimbStairs.html, ClimbStairs.js and ClimbStairs.wasm



2. Open the html file in "http://localhost:8000/"

The program will output number of ways to climb a stair with 30 steps. Result will be displayed on the webpage.

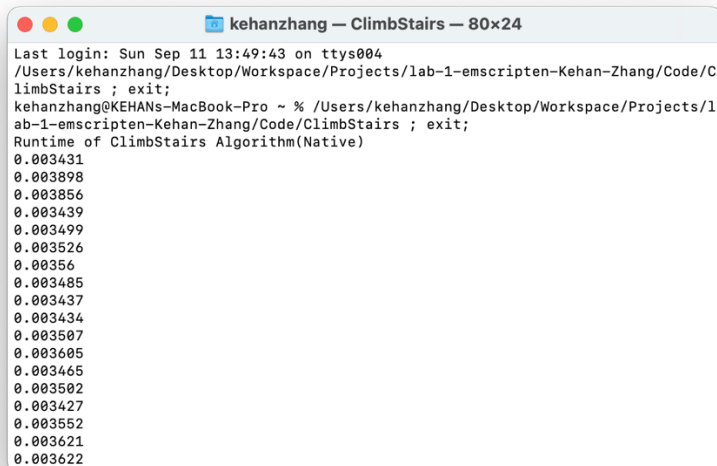


## Build Code Base to WebAssembly

### 1. Modify Code Base

I made climbStairs function extern, ran this algorithm 1000 times and outputted the runtime of each execution.

```
1  #include <iostream>
2  #include <time.h>
3
4  using namespace std;
5
6  // It takes n steps to reach the top of a stair.
7  // Each time you can either climb 1 or 2 steps.
8  // Calculate distinct ways you can take to climb to the top.
9  extern "C" {
10     int climbStairs(int n) {
11         if (n == 1) return 1;
12         if (n == 2) return 2;
13
14         return climbStairs(n - 1) + climbStairs(n - 2);
15     }
16 }
17
18
19 int main() {
20     cout << "Runtime of ClimbStairs Algorithm(Native)" << endl;
21     for (int count = 0; count < 1000; count++) {
22         clock_t start, end;
23         start = clock();
24
25         int n = 30;
26         climbStairs(n);
27
28         end = clock();
29         cout << (double)(end-start)/CLOCKS_PER_SEC << endl;
30     }
31
32     return 0;
33 }
```



```
kehanzhang — ClimbStairs — 80x24
Last login: Sun Sep 11 13:49:43 on ttys004
/Users/kehanzhang/Desktop/Workspace/Projects/lab-1-emscripten-Kehan-Zhang/Code/C
limbStairs ; exit;
kehanzhang@KEHANS-MacBook-Pro ~ % /Users/kehanzhang/Desktop/Workspace/Projects/l
ab-1-emscripten-Kehan-Zhang/Code/ClimbStairs ; exit;
Runtime of ClimbStairs Algorithm(Native)
0.003431
0.003898
0.003856
0.003439
0.003499
0.003526
0.00356
0.003485
0.003437
0.003434
0.003507
0.003605
0.003465
0.003502
0.003427
0.003552
0.003621
0.003622
```

## 2. Transpile Code Base

```
emcc ClimbStairs.cpp -o /Users/kehanzhang/Desktop/Workspace/Projects/lab-1-emscripten-Kehan-Zhang/Code/ClimbStairs.js -s EXPORTED_FUNCTIONS=["_climbStairs"] -s EXPORTED_RUNTIME_METHODS=["ccall"]
```

```
kehanzhang@KEHANS-MacBook-Pro ClimbStairs % emcc ClimbStairs.cpp -o /Users/kehanzhang/Desktop/Workspace/Projects/lab-1-emscripten-Kehan-Zhang/Code/ClimbStairs.js -s EXPORTED_FUNCTIONS=["_climbStairs"] -s EXPORTED_RUNTIME_METHODS=["ccall"]
system_libs:WARNING: main() is in the input files, but "_main" is not in EXPORTED_FUNCTIONS, which means it may be eliminated as dead code. Export it if you want main() to run.
```

This command generated two files: ClimbStairs.js and ClimbStairs.wasm



ClimbStairs.wasm



ClimbStairs.js

## 3. Created Time.js file to calculate the execution time of wasm based code base

```
var addModule = Module.onRuntimeInitialized = function() {  
  for (var i = 0; i < 1000; i++) {  
    var start = performance.now();  
    var result = Module.ccall('climbStairs', 'number', ['int'], [30]);  
    var end = performance.now();  
    var runtime = (end - start) / 1000;  
    console.log(runtime.toFixed(4));  
  }  
}
```

## 4. Create ClimbStairs.html file

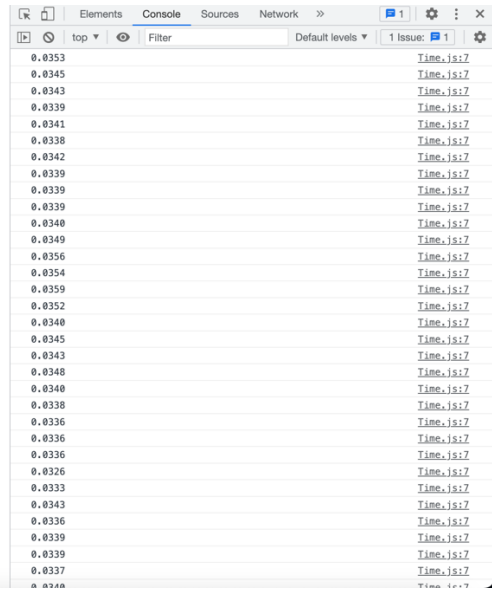
```
<!DOCTYPE html>  
<head>  
  <meta charset="utf-8"/>  
</head>  
<body>  
  Runtime of ClimbStairs Algorithm(WASM Based)  
  
  <script src="ClimbStairs.js"></script>  
  <script src="Time.js"></script>  
</body>  
</html>
```

5. In webpage controller settings, disabled cache and disabled group similar message. Runtime of 1000 execution is shown in console.

Disabling cache is for testing code easily after modification.

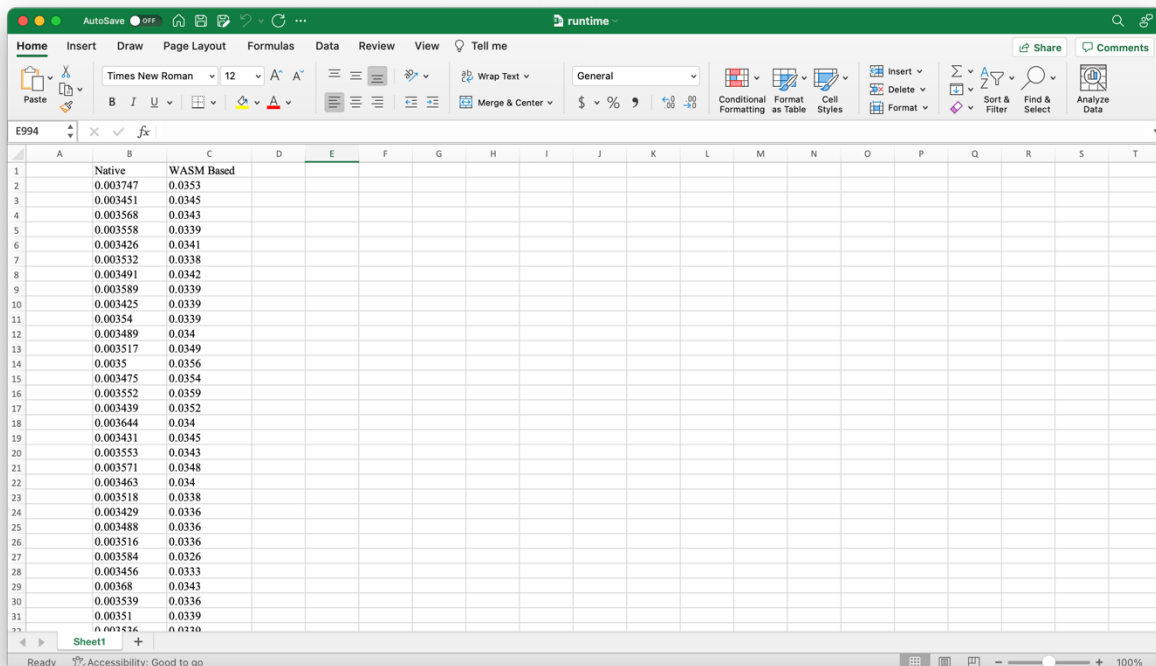
Disabling group similar message is for pasting runtime in console to excel

Runtime of ClimbStairs Algorithm(WASM Based)



### 3. Compare Execution Time between Native and WASM Based Code Bases

1. Paste 1000 execution runtime of native code base and wasm code base to one excel file, and perform basic statistic measure on them.



We can tell from the statistic results in the form below that execution time of native code base is in the range of 0.003411 seconds to 0.003894 seconds, and that of wasm based code base is in the range of 0.0315 seconds to 0.0481 seconds. What is more, the average execution time of native code base is 0.003465401 seconds, while the average execution time of wasm based code base is 0.033979 seconds, which is almost 10 times as that of native code base.

<b>mean</b>	<b>0.003465401</b>	<b>0.033979</b>
<b>max</b>	<b>0.003894</b>	<b>0.0481</b>
<b>min</b>	<b>0.003411</b>	<b>0.0315</b>

2. Use Excel to calculate 95% confidence intervals. We can see from the form below: 95% confidence interval of native code base is 0.003462508 to 0.003468294, 95% confidence code base of wasm based code base is 0.033213386 to 0.034026452. These results are of statistically significant since it shows that execution time of wasm based code base is 10 times as that of native code base. We can conclude that performance of native code base is much better than wasm based code base.

	<b>native</b>	<b>wasm based</b>
<b>mean</b>	<b>0.003465401</b>	<b>0.033979</b>
<b>max</b>	<b>0.003894</b>	<b>0.0481</b>
<b>min</b>	<b>0.003411</b>	<b>0.0315</b>
<b>standard_dev</b>	<b>4.66713E-05</b>	<b>0.000765614</b>
<b>95% confidence</b>	<b>2.89266E-06</b>	<b>4.74524E-05</b>
<b>mean - confidence</b>	<b>0.003462508</b>	<b>0.033213386</b>
<b>mean + confidence</b>	<b>0.003468294</b>	<b>0.034026452</b>

## 4. Improve Performance of WASM Code

I improved performance of wasm code by specifying optimization flags when running emcc. The levels include: -O0, -O1, -O2, -Os, -Oz, and -O3.

-O1: Build with lower optimization levels during development for a shorter compile/test iteration cycle

-O2: Get a well-optimized build.

-O3: Produce an ever better build than -O2, but at the cost of significantly longer compilation time and potentially larger code size.

-Os: Focuses on reducing code size while doing additional optimization.

(Reference: <https://emscripten.org/docs/optimizing/Optimizing-Code.html>)

I chose -O3 to improve performance of wasm code.

```
emcc -O3 ClimbStairs.cpp -o /Users/kehanzhang/Desktop/Workspace/Projects/lab-1-emscripten-Kehan-Zhang/Code/ClimbStairs.js -s EXPORTED_FUNCTIONS=["_climbStairs"] -s EXPORTED_RUNTIME_METHODS=["ccall"]
```

1. Paste 1000 execution runtime of improved wasm code base to excel file, and perform basic statistic measure on them.

	A	B	C	D
1		Native	WASM Based	WASM Improved
2		0.003747	0.0353	0.0046
3		0.003451	0.0345	0.004
4		0.003568	0.0343	0.0051
5		0.003558	0.0339	0.0041
6		0.003426	0.0341	0.0038
7		0.003532	0.0338	0.0038
8		0.003491	0.0342	0.0047
9		0.003589	0.0339	0.005
10		0.003425	0.0339	0.0037
11		0.00354	0.0339	0.0039
12		0.003489	0.034	0.0088
13		0.003517	0.0349	0.0045
14		0.0035	0.0356	0.0037
15		0.003475	0.0354	0.0038
16		0.003552	0.0359	0.0039
17		0.003439	0.0352	0.0038
18		0.003644	0.034	0.0039
19		0.003431	0.0345	0.004
20		0.003553	0.0343	0.0038
21		0.003571	0.0348	0.0037
22		0.003463	0.034	0.004
23		0.003518	0.0338	0.004
24		0.003429	0.0336	0.0043
25		0.003488	0.0336	0.0037
26		0.003516	0.0336	0.0037
27		0.003584	0.0326	0.0037
28		0.003456	0.0333	0.0037
29		0.00368	0.0343	0.004
30		0.003539	0.0336	0.0037
31		0.00351	0.0339	0.0038
32		0.003536	0.0339	0.0037
33		0.003478	0.0337	0.0037
34		0.003526	0.034	0.0037
35		0.003504	0.0338	0.0039
36		0.003483	0.0349	0.0038
37		0.003667	0.0341	0.0037

We can tell from the statistic results in the form below that execution time of improved wasm based code base is in the range of 0.0035 seconds to 0.0102 seconds. What is more, the average execution time of improved wasm code base is 0.0037356 seconds, which is a little bit longer than native code base.

	native	wasm based	wasm improved
mean	0.003465401	0.033979	0.0037356
max	0.003894	0.0481	0.0102
min	0.003411	0.0315	0.0035

2. Use Excel to calculate 95% confidence intervals. We can see from the form below: 95% confidence interval of improved wasm based code base is 0.003432938 to 0.003754359. These results are of statistically significant since it shows that execution time of improved wasm based code base is a little bit longer than native code base. We can conclude that performance of improved wasm code base is almost the same as native code base, and both of them are much better than wasm based code base.

	native	wasm based	wasm improved
mean	0.003465401	0.033979	0.0037356
max	0.003894	0.0481	0.0102
min	0.003411	0.0315	0.0035
standard_dev	4.66713E-05	0.000765614	0.000302662
95% confidence	2.89266E-06	4.74524E-05	1.87588E-05
mean - confidence	0.003462508	0.033213386	0.003432938
mean + confidence	0.003468294	0.034026452	0.003754359