

Environment Set Up Instruction

Dong Chen

April 15, 2021

This is an instruction to set up the environment in each language to run benchmark tests. The instruction includes set up for assembly script, go, rust, and p0. This is a supplement document for the project of class 4tb3/6tb3.

1 AssemblyScript

This is the instruction to set up AssemblyScript in your local machines.

1.1 Step 1: Install AssemblyScript

In order to run AssemblyScript, you have to set up the environment. The following command will install the Assembly Script which is a subset of TypeScript.

```
npm install -g assemblyscript
```

1.2 Step 2: Create a TypeScript file

Then, create a file with .ts suffix. This a ts file is a TypeScript file. In short, it is a language like Javascript but it has data types. The following is the file calculate Fibonacci sequence in TypeScript.

```
export function fib(n: i32): i32 {  
  var a = 0  
  if (n === 0) return 0  
  if (n === 1) return 1  
  
  a = fib(n-1) + fib(n-2)  
  return a  
}
```

1.3 Step 3: Transform TypeScript to WebAssembly

After creating the TypeScript file, use the following command to transform TypeScript to WebAssembly. `asc` is the command for AssemblyScript.

```
asc fib.ts -o fib.wasm
```

A `wasm` file will be created, and it has equivalent methods to its source file.

1.4 Step 4: Create a html file

In order to run a `wasm` file, we can use Javascript via local browsers. The following html file needs to be created, it can load the `wasm` file. Therefore we can use the browser console to test the `wasm` file.

```
<html>
<body>
  <script>
    fetch('fib.wasm').then(response =>
      | response.arrayBuffer()
    ).then(bytes =>
      | WebAssembly.instantiate(bytes, {imports: {}})
    ).then(results => {
      | window.fib = results.instance.exports.fib;
      | });
  </script>
</body>
</html>
```

1.5 Step 5: Install http-server

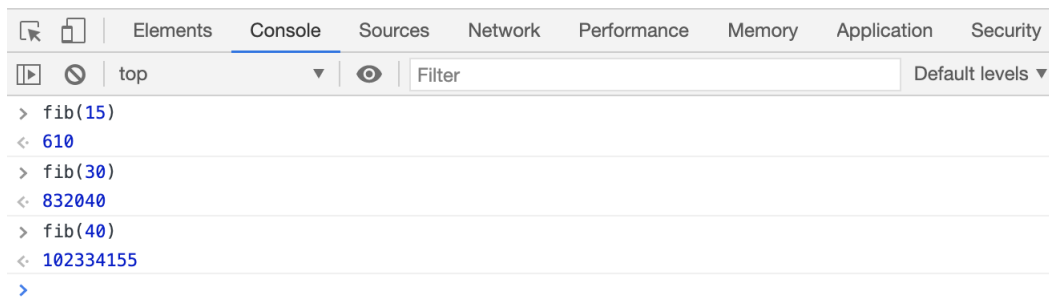
In order to mimics the host server environment, we have to install `http-server` package. The following command will install `http-server`

```
npm install -g http-server
```

After installation, run `http-server` by typing `http-server` in the command line and load the html file.

1.6 Step 6: Run Webassembly in Browsers

In the browser console, type `fib()` to test the outcome. The example shows in the following.



2 Go Language

This is the instruction to set up the Go language in your local machines.

2.1 Step 1: Install Go language

There are various ways to install Go. You can reference to Golang Download and install

The author use brew to install go by using the following command in the command terminal.

```
brew install go
```

2.2 Step 2: Create a Go file

Then, create a go file. The following is a go file with the Fibonacci sequence. In the following go file, the author exposed the fib file, so users can access the fib method in the browser console. Additionally, the go file will read the input which come from users and calculate the Fibonacci sequence.

```

1  // go
2  package main
3
4  import (
5  |   "syscall/js"
6  )
7
8  func main() {
9  |   c := make(chan bool)
10 |   //1. Exposing go functions/values in javascript variables.
11 |   js.Global().Set("fib", js.FuncOf(fib))
12 |   //2. This channel will prevent the go program to exit
13 |   <-c
14 }
15
16 func fib(this js.Value, inputs []js.Value) interface{} {
17 |   fibnum := fibinternal(inputs[0].Int())
18 |   return fibnum
19 }
20
21 func fibinternal(n int) int {
22 |   if n == 0 {
23 |       return 0
24 |   }
25 |   if n == 1 {
26 |       return 1
27 |   }
28 |   sum := fibinternal(n-1) + fibinternal(n-2)
29 |   return sum
30 }
31

```

2.3 Step 3: Compile Go to WebAssembly

To compile a go file, first, initialize the go mod in the current folder. It will create a .mod file. Then, run the following command to start the transformation.

```
GOOS=js GOARCH=wasm go build -o fib.wasm fib.go
```

A wasm file will be created, and it has equivalent methods to its source file.

2.4 Step 4: Copy wasm_exec.js and wasm_exec.html

These are two files already provide by Go language, we can reuse them to run webassembly in the local browser. These two files locates at (go env GOROOT)/misc/wasm/. Use the command “go env GOROOT” to get the exact root path, and concatenate

it with the rest of path.

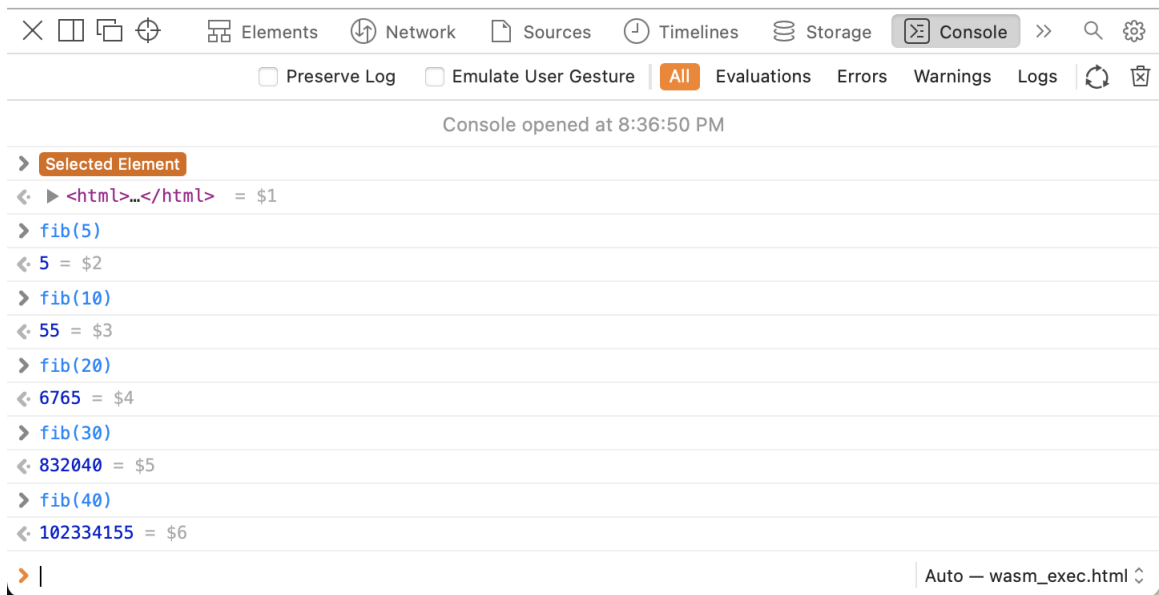
Edit the file name in html file to match the wasm name. The following is the html file for this project.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <script src="wasm_exec.js"></script>
    <script>
      if (WebAssembly) {
        // WebAssembly.instantiateStreaming is not currently available in Safari
        if (WebAssembly && !WebAssembly.instantiateStreaming) { // polyfill
          WebAssembly.instantiateStreaming = async (resp, importObject) => {
            const source = await (await resp).arrayBuffer();
            return await WebAssembly.instantiate(source, importObject);
          };
        }

        const go = new Go();
        WebAssembly.instantiateStreaming(fetch("fib.wasm"), go.importObject).then((result) => {
          go.run(result.instance);
        });
      } else {
        console.log("WebAssembly is not supported in your browser")
      }
    </script>
  </head>
  <body>
    <main id="wasm"></main>
  </body>
</html>
```

2.5 Step 5: Run Webassembly in Browsers

In this step, we use the same method in AssemblyScript, using http-server. In the browser console, type `fib()` to test the outcome. The example shows in the following.



3 Rust Language

This is the instruction to set up the Rust language in your local machines.

3.1 Step 1: Install Rust Language

There are various ways to install Go. You can reference to Rust Installation. The author uses the following installed Rust in the command terminal.

```
curl --proto '=https' --tlsv1.2 https://sh.rustup.rs -sSf | sh
```

3.2 Step 2: Create a Rust file

Then, create a rust file. The following is a rust file with the Fibonacci sequence.

```
fn main() {}

#[no_mangle]
pub extern "C" fn fib(n: u32) -> u64 {
    match n{
        0 => 0,
        1 => 1,
        _ => fib(n-1) + fib(n-2)
    }
}
```

3.3 Step 3: More Packages

There is one more package that need to be installed. Run the following to install the required target.

```
rustup target add wasm32-unknown-unknown --toolchain nightly
```

3.4 Step 4: Compile Go to WebAssembly

To compile a go file, run the following command to start the transformation.

```
rustc +nightly --target wasm32-unknown-unknown -O fib.rs
```

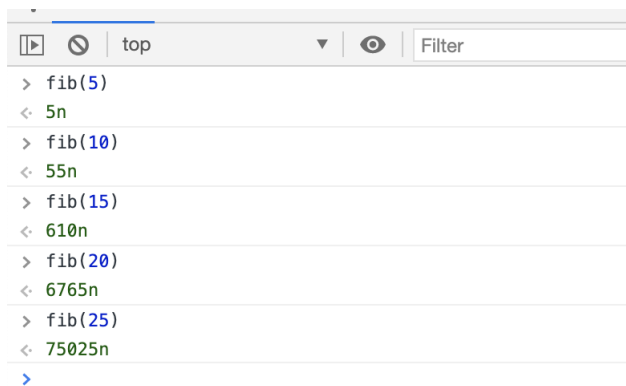
A wasm file will be created, and it has equivalent methods to its source file.

3.5 Step 5: Create a html file

A html needs to be created to load wasm locally. This is a similar implementation in AssemblyScript step 4.

3.6 Step 6: Run Webassembly in Browsers

In this step, we use the same method in AssemblyScript, using http-server. In the browser console, type `fib()` to test the outcome. The example shows in the following.



```
> fib(5)
< 5n
> fib(10)
< 55n
> fib(15)
< 610n
> fib(20)
< 6765n
> fib(25)
< 75025n
>
```

4 P0 Language

P0 language has been taught in the class. Please refer to chapter 5 on how to set up the P0 language.

5 Reference

- The simplest way to get started with WebAssembly
- Getting Started With WebAssembly and Go By Building an Image to ASCII Converter
- Rust for the Web