INTRODUCTION TO WEBASSEMBLY

Introduction

What is WebAssembly? WebAssembly, or WASM, is essentially a type of code that allows languages to run on the web. It is the converted file of a programming language that you can use, hence it replaces the programming language file. The major browsers WASM works in is Firefox, Chrome, Safari and Edge and runs via JavaScript.

In this tutorial I will explain the basics of wasm and how to a basic program works with it, using C.

You can download the files from the link below:

https://github.com/mp-github-acc/WebAssemblyTutorial

Getting started

We need a browser, namely one of the previously mentioned that do support WASM.

If you don't have a way to host a local server, be sure to download one, such as XAMPP at https://www.apachefriends.org/index.html. Set it up and you will need to open the "index.html" from the GitHub link above (make sure the entire folder is moved with it). The code does have comments to explain the important parts so you can follow along there as well.

We are going to do the coding in https://webassembly.studio/ and then you will be able to download the files to run it in your browser using the local server.

Let's go!

Generating files

In WebAssembly Studio, you will have the option to create a new project. Create an Empty C Project. You will see that there is a couple of files that have been generated for you.

The “README.md” and “package.json” files are not relevant. The “build.ts” file is what will generate our wasm file for us. You will also see a “main.c”, which is what will be used to generate the wasm file. Then we have the “main.html” and “main.js” files.

You will see the html file does not have much, just a plain page. This is where the results will show, but we will get to that in a bit. The C file is short, just one “main” function that returns a value. The other code in this file is what allows us to access the function when it is converted into a wasm file, as a module. A module is essentially a component of a program.

Select the build and run option. Now we have our “main.wasm” file which is essentially the functionality of the C file in a different format. You will see a bit of code which looks familiar:

```
(func $main (export "main") (type $t1) (result i32)
  i32.const 42)
```

This is our main function from the C file but in a format a bit closer to machine level than human level.

Lastly you will see the bottom right of your screen is a block with a number in it. That is the result of the C function shown on the html via an iframe.

Here is a rundown of what happened:

-The build.ts file compiled the main.c file into main.wasm

-The main.html file is loaded, and since the main.js file is referenced, it loaded too

-The main.js made a request to load main.wasm file (line 1 in main.js)

-An instance of the wasm code was made (lines 2-4)

-A call is made to the main function which returns the value 42

(Line 5, “instance.exports.main()”)

-This value is set to show on main.html

(Line 5, “document.getElementById("container").textContent”)

Hopefully you have a better idea, now we can move on to do a bit more

Basic example

This link will demonstrate a basic example of what happens in the calculator file. It was made off the empty C project that WebAssembly Studio generates to show you how simple this calculator is to make. If the link fails, there is a copy available on the GitHub link as a zip: “wasm-project-zv6hqlz0jsi.zip”

https://webassembly.studio/?f=qpninby4qv

In the main.c, you can see there are a few functions for the basic operations, and each has a line “WASM_EXPORT” above. This allows us to access each function when we need it. The main.html isn’t important in this example, as we print the results under “Output”.

Let’s run through the main.js. the first 5 lines should look familiar. They are unchanged. We just fetch the wasm file and make an instance of the code that we can use. Then in lines 6-9, we simply call each function via “instance.exports”, reference the function and pass the parameters. Building and running this code in WebAssembly Studio or a local server will print to the console. You will see these values are the correct output as the result of their respective functions.

This example hopefully sheds some more light on how the basic calculator I made works. Only a few changes need to be made to make it function the same!

Note that if you open the wasm file in a text editor instead of WebAssembly Studio, you’ll see a bunch of random numbers and letters. This happens because WebAssembly Studio shows it in a more understandable way for a human user, but it looks like this, a bunch of numbers and letters. This is closer to machine language, as it is easier for your computer to interpret.

Making the calculator

Now we can run through my code for the calculator. I will explain how I did mine and how it works. If you haven't already, download the files from the GitHub link. Remember to open them in a local server. By the end, you can hopefully make one like mine!

There are quite a few files in the repository. The main ones to take note of are "index.html" "main.wasm" "script.js" and "style.css". You can open these in a text editor, as I have added comments to explain the important parts as well.

Open index.html in your browser. You can test out the calculations using the input fields and the results will show under the respective button after it is pressed. Opening the console in your browser will show you which functions execute and what the result was. Once you're satisfied, we can continue.

We don't need to do much to get the previous basic example to function the same as this calculator does.

Let's start with the html. The head tags contain some libraries, Bootstrap for layout, Font awesome for the '>' icons used in the buttons, jQuery which is necessary for bootstrap, a link tag to set the tab's icon in the browser, title for the tab and the CSS file for styling. The CSS only styles the page, doesn't do anything else so we can ignore that. The pages contents have the WebAssembly image with a heading and description.

Now the most important parts, the cards. You can see there are 4 cards, one for each of the basic operations. Note that each of these works the same, they only differ to work for their respective operations. We'll go over the addition. The input tags in the Addition card have unique ids for the first and second values, we will use these to grab the values to use with the function to calculate the result. The button calls the JavaScript function "addResult()" which is defined in the script.js file. Last important thing to note here is the span with the id "addResult". This is where we will show the result after it is calculated.

The script.js file is used when we call the function addResult(). Before this, we just declare some variables that we will use to get and set to/from the html. In this case, we have 'add1' and 'add2' which will be set to the values from the input tags in a bit, and the 'add' variable which we will use to store the result.

When addResult() is called by pressing the button on the page, it will go through this function addResult(). We first print to console to say that the add function is executing. The 'add1' and 'add2' variables are then set using jQuery to the values that were in the input field. The next few lines should be familiar again, we fetch the wasm file, and create an instance of it that we can use. We then set the 'add' variable to be the result of the add function, using 'instance.exports' again. We print this result to console and then set the 'addResult' in the html to then show the resulting value.

That's it! As I mentioned, the cards in the html are basically the same, just minor differences between them to work for their respective functions by changing some ids, and the same goes to the other JavaScript functions, they work the same!

Conclusion

Remember, there is no need for the C file anymore. Functionally, all that is needed to make this calculator is HTML, JavaScript, and the WASM file after it has been made.

Hope you enjoyed this tutorial and its explanations, and hopefully you understand a bit more about WebAssembly, what it can do and how it works.

References:

Logo:

https://www.logo.wine/logo/WebAssembly

Generate files:

https://webassembly.studio/

Information:

https://webassembly.org/

https://developer.mozilla.org/en-US/docs/WebAssembly