Task 3: WASM at the Edge

------------------------

For this task, you will cross-compile your WAT compiler and WASM VM (tasks 2 and 1) from C++ to WASM

so you can compile and run WebAssembly, in WebAssembly, both in a browser and on the edge (in a Cloudflare edge worker).

The goal is to have a user be able to provide a WAT string (e.g., via a <textarea> on a web page),

and then have that string compiled and executed, the result shown to the user in the same web page.

For this task, you get a lot of example code/detailed explanations for the basics,

but you'll still have to figure out some things along the way to make it work properly.

MUST have features:

- ~~Browser-based execution of task 1 and 2 via emscripten cross compilation~~

- ~~Cloudflare worker-based execution of task 1 and 2 via emscripten cross compilation~~

- ~~Basic user interface (minimal: pass WAT string via URL request parameter)~~

- ~~Basic output from WASM execution (minimal: single integer output, console logs in wrangler/browser logs)~~

Nice to have/expected for good score:

- ~~Proper user interface (e.g., <textarea> where user can provide larger scripts)~~

- similar to https://webassembly.github.io/wabt/demo/wat2wasm/

- ~~Proper output (e.g., full WASM bytestream output as well as execution output values)~~

- similar to https://webassembly.github.io/wabt/demo/wat2wasm/

- ~~Ideally, this task 3 version of your compiler+VM supports all the features the "local" compilations of task 1 and 2 also support~~

- There can be some exceptions, see also below

Optional:

- Find an open source C++/RUST/... project, cross-compile it to WASM as well, and run it at the CloudFlare edge

- This can even be one that already has instructions on how to compile to wasm!

- Stress test the Free cloudflare workers plan

- CF workers is limited to 10ms execution time and a small amount of memory

- You can try to stress tests their systems by executing WASM code that takes a long time, has an infinite loop, that attempts to allocate lots of memory, etc.

- The goal is to see how far you can go before they detect problems + how they handle errors gracefully