Compile C to Wasm and display an Image in the Browser

Requirements:

1. Compile C to Webassembly
2. Javascript
3. Bootstraping

Compile C to Webassembly:

The command

emcc -O3 -s WASM=1 -s EXTRA_EXPORTED_RUNTIME_METHODS='["cwrap"]' -I libwebp webp.c libwebp/src/{dec,dsp,demux,enc,mux,utils}/*.c -s ALLOW_MEMORY_GROWTH=1 -s ASSERTIONS=1

is used to compile the c file into a wasm file and a javascript file. The flags are descriped as it followes

-O3: Various High Stage optimizations of Javascript,File Size and LLVM compiler infrastructure

-s: JavaScript code generation option passed into the Emscripten compiler.

- WASM = 1: compile into WASM Format

- EXTRA_EXPORTED_RUNTIME_METHODS = 1: If you try to access Runtime.* methods from outside the compiled code, then you must export that function and use it on the Module object.
  If you try to access Runtime.* methods from outside the compiled code, then you must export that function and use it on the Module object.

- ALLOW_MEMORY_GROWTH = 1: This variable specifies the geometric overgrowth rate of the heap at resize
- ASSERTIONS = 1: Whether we should add runtime assertions, for example to check that each allocation to the stack does not exceed its size, whether all allocations (stack and static) are of positive size, etc. It also enables StackOverflow Checks

-I: To tell the compiler where it can find libwebp's header files and also pass it all the C files of libwebp that it needs.

Libwebp: WebP codec: library to encode and decode images in WebP format. This package contains the library that can be used in other programs to add WebP support, as well as the command line tools 'cwebp' and 'dwebp'.

Javascript / Bootstrapping:

To include, interpret the wasm file and display the image, javascript is used. After compilation, a javascript file got created which has to be included in the html file.

Get an image from JavaScript into Wasm

at the encoding API of libwebp, it expects an array of bytes in RGB, RGBA, BGR or BGRA., the Canvas API has `getImageData()`, that gives us an Uint8ClampedArray containing the image data in RGBA:

```
async function loadImage(src) {
  // Load image
  const imgBlob = await fetch(src).then(resp => resp.blob());
  const img = await createImageBitmap(imgBlob);
  // Make canvas same size as image
  const canvas = document.createElement('canvas');
  canvas.width = img.width;
  canvas.height = img.height;
  // Draw image onto canvas
  const ctx = canvas.getContext('2d');
  ctx.drawImage(img, 0, 0);
  return ctx.getImageData(0, 0, img.width, img.height);
}
```

Now it's "only" a matter of copying the data from JavaScript into Wasm. For that, we need to expose two additional functions. One that allocates memory for the image inside Wasm and one that frees it up again:

```
EMSCRIPTEN_KEEPALIVE
uint8_t* create_buffer(int width, int height) {
  return malloc(width * height * 4 * sizeof(uint8_t));
}

EMSCRIPTEN_KEEPALIVE
void destroy_buffer(uint8_t* p) {
  free(p);
}
```

create_buffer() allocates a buffer for the RGBA image hence 4 bytes per pixel. The pointer returned by malloc() is the address of the first memory cell of that buffer. When the pointer is returned to JavaScript, it is treated as just a number. After exposing the function to JavaScript using **cwrap**, we can use that number to find the start of our buffer and copy the image data.

```
const api = {
  version: Module.cwrap('version', 'number', []),
  create_buffer: Module.cwrap('create_buffer', 'number', ['number', 'number']),
  destroy_buffer: Module.cwrap('destroy_buffer', '', ['number']),
};
```

```
const image = await loadImage('/image.jpg');
const p = api.create_buffer(image.width, image.height);
Module.HEAP8.set(image.data, p);
// ... call encoder ...
api.destroy_buffer(p);
```

Encode the image:

The image is now available in Wasm. Now the WebP encoder has to be called. Looking at the WebP documentation, WebPEncodeRGBA seems to fit. The function takes a pointer to the input image and its dimensions, as well as a quality option between 0 and 100. It also allocates an output buffer, that we need to free using WebPFree() once we are done with the WebP image. The result of the encoding operation is an output buffer and its length. Because functions in C can't have arrays as return types (unless we allocate memory dynamically), We resorted to a static global array.

```
int result[2];
EMSCRIPTEN_KEEPALIVE
void encode(uint8_t* img_in, int width, int height, float quality) {
  uint8_t* img_out;
  size_t size;

  size = WebPEncodeRGBA(img_in, width, height, width * 4, quality, &img_out);

  result[0] = (int)img_out;
  result[1] = size;
}

EMSCRIPTEN_KEEPALIVE
void free_result(uint8_t* result) {
  WebPFree(result);
}

EMSCRIPTEN_KEEPALIVE
int get_result_pointer() {
  return result[0];
}

EMSCRIPTEN_KEEPALIVE
int get_result_size() {
  return result[1];
}
```

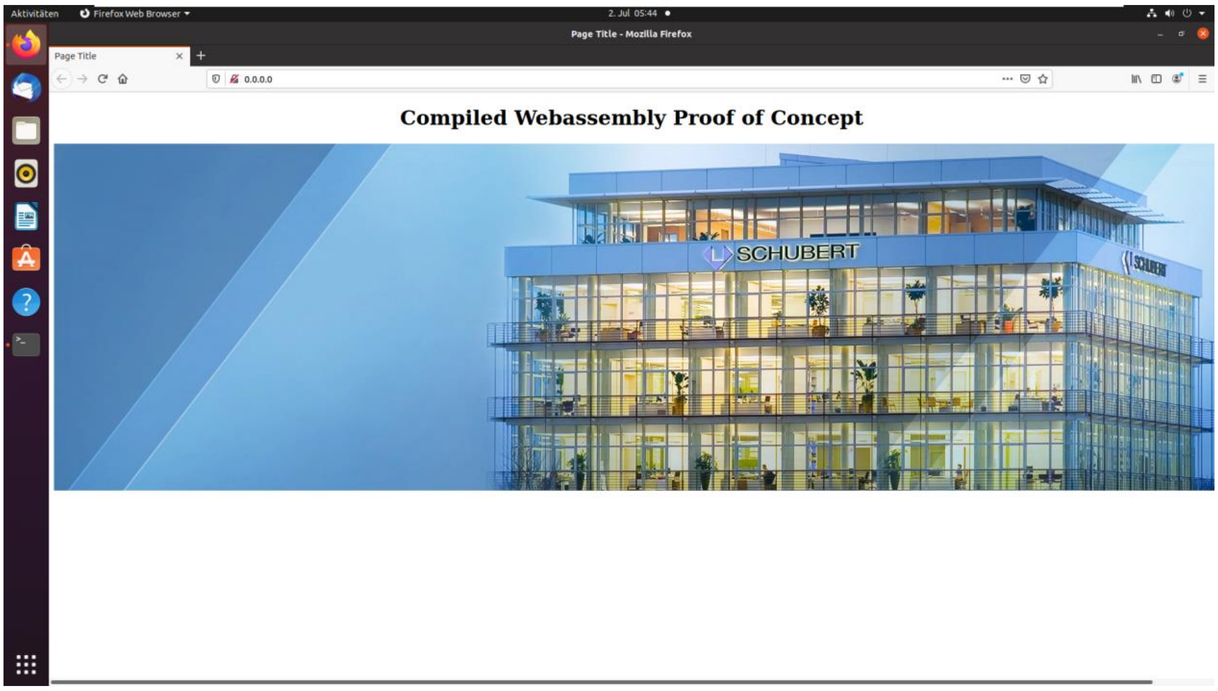
Now with everything in place, we can call the encoding function, grab the pointer and image size, put it in a JavaScript buffer, and release all the Wasm buffers we have allocated in the process.

```
api.encode(p, image.width, image.height, 100);
const resultPointer = api.get_result_pointer();
const resultSize = api.get_result_size();
const resultView = new Uint8Array(Module.HEAP8.buffer, resultPointer, resultSize);
const result = new Uint8Array(resultView);
api.free_result(resultPointer);
```

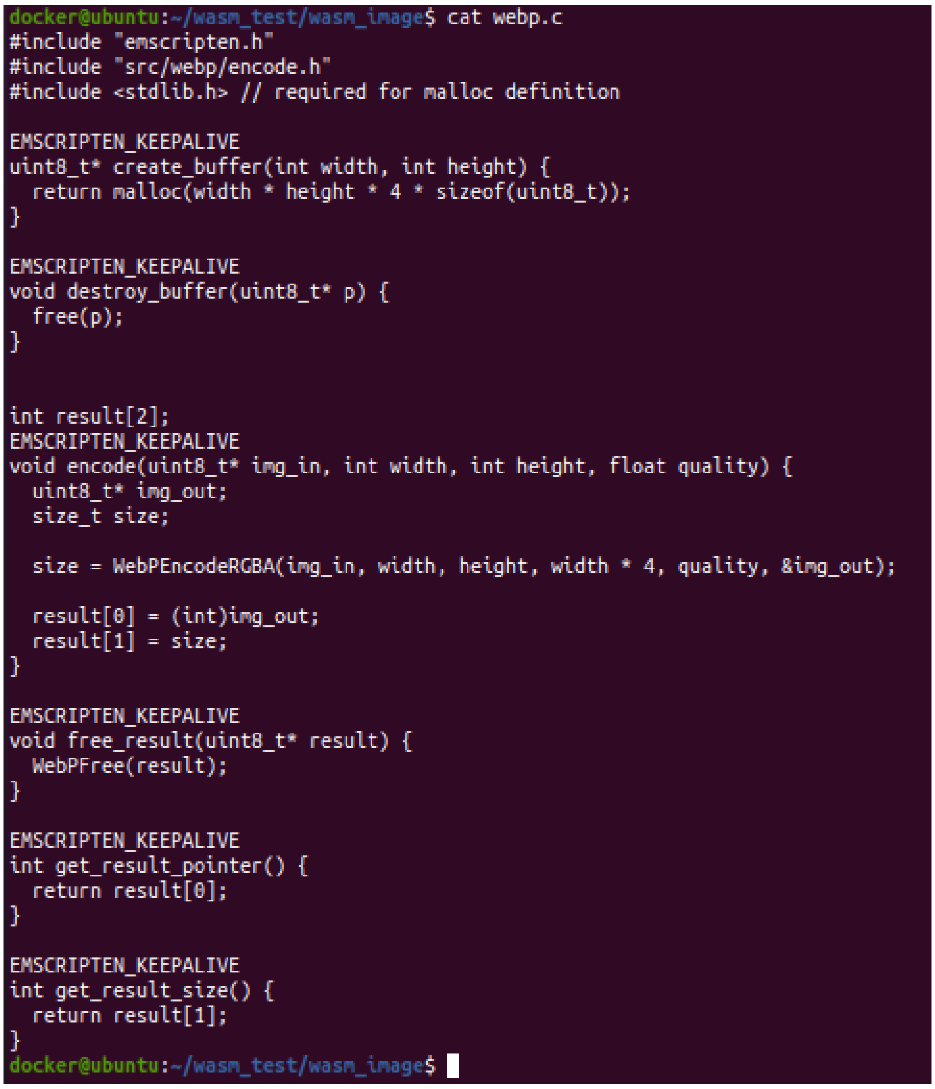
At the End we have to turn our result buffer into a blob using javascript.

```
const blob = new Blob([result], {type: 'image/webp'});
const blobURL = URL.createObjectURL(blob);
const img = document.createElement('img');
img.src = blobURL;
document.querySelector('.output').appendChild(img)
```

The Result looks like this:

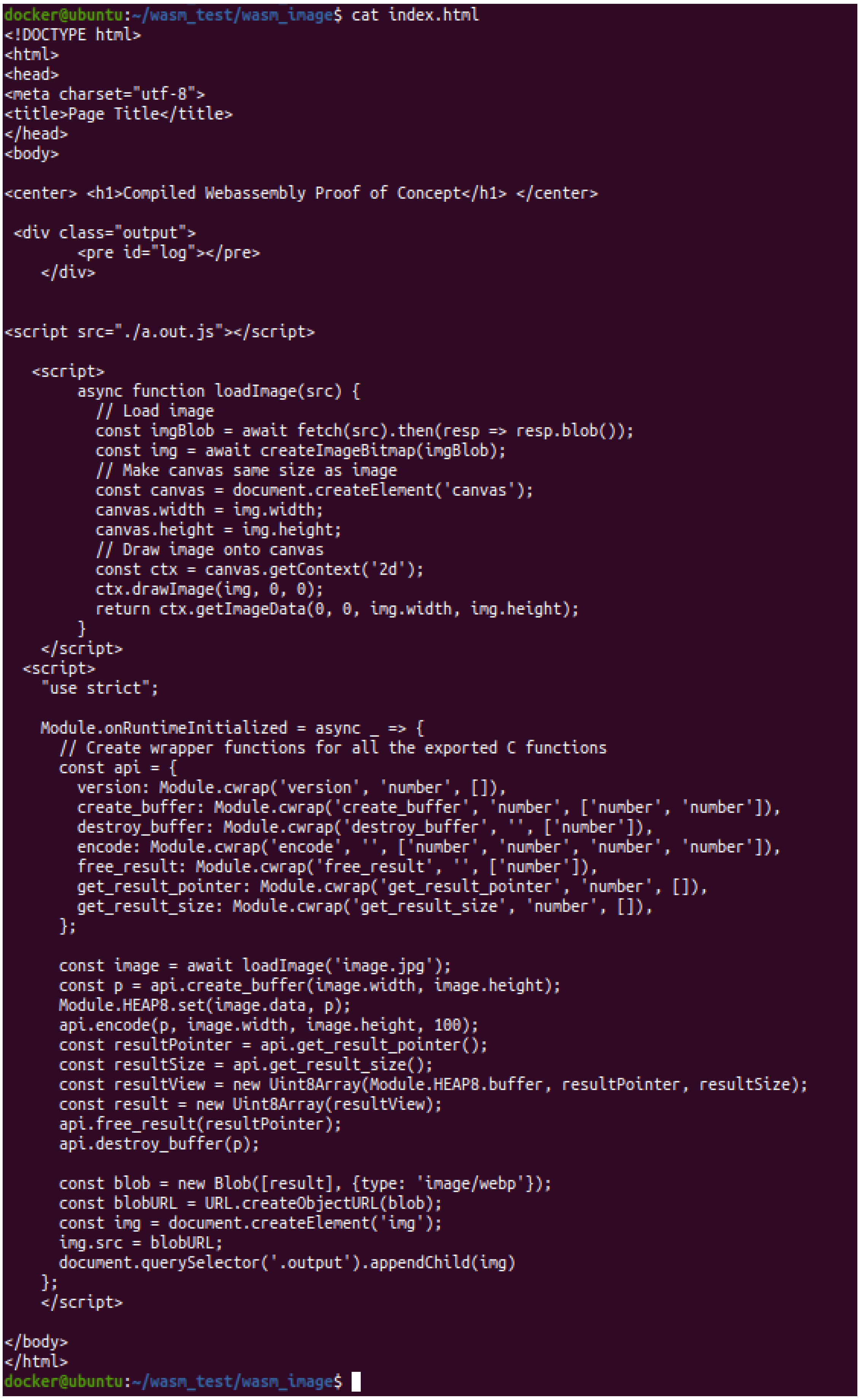


Webp.c Sourcecode:



```
docker@ubuntu:~/wasm_test/wasm_image$ cat webp.c
#include "emscripten.h"
#include "src/webp/encode.h"
#include <stdlib.h> // required for malloc definition

EMSCRIPTEN_KEEPALIVE
uint8_t* create_buffer(int width, int height) {
  return malloc(width * height * 4 * sizeof(uint8_t));
}

EMSCRIPTEN_KEEPALIVE
void destroy_buffer(uint8_t* p) {
  free(p);
}


int result[2];
EMSCRIPTEN_KEEPALIVE
void encode(uint8_t* img_in, int width, int height, float quality) {
  uint8_t* img_out;
  size_t size;

  size = WebPEncodeRGBA(img_in, width, height, width * 4, quality, &img_out);

  result[0] = (int)img_out;
  result[1] = size;
}

EMSCRIPTEN_KEEPALIVE
void free_result(uint8_t* result) {
  WebPFree(result);
}

EMSCRIPTEN_KEEPALIVE
int get_result_pointer() {
  return result[0];
}

EMSCRIPTEN_KEEPALIVE
int get_result_size() {
  return result[1];
}
docker@ubuntu:~/wasm_test/wasm_image$
```

Index.html Sourcecode:



```
docker@ubuntu:~/wasm_test/wasm_image$ cat index.html
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Page Title</title>
</head>
<body>

<center> <h1>Compiled Webassembly Proof of Concept</h1> </center>

 <div class="output">
        <pre id="log"></pre>
    </div>


<script src="./a.out.js"></script>

   <script>
        async function loadImage(src) {
          // Load image
          const imgBlob = await fetch(src).then(resp => resp.blob());
          const img = await createImageBitmap(imgBlob);
          // Make canvas same size as image
          const canvas = document.createElement('canvas');
          canvas.width = img.width;
          canvas.height = img.height;
          // Draw image onto canvas
          const ctx = canvas.getContext('2d');
          ctx.drawImage(img, 0, 0);
          return ctx.getImageData(0, 0, img.width, img.height);
        }
    </script>
  <script>
    "use strict";

    Module.onRuntimeInitialized = async _ => {
      // Create wrapper functions for all the exported C functions
      const api = {
        version: Module.cwrap('version', 'number', []),
        create_buffer: Module.cwrap('create_buffer', 'number', ['number', 'number']),
        destroy_buffer: Module.cwrap('destroy_buffer', '', ['number']),
        encode: Module.cwrap('encode', '', ['number', 'number', 'number', 'number']),
        free_result: Module.cwrap('free_result', '', ['number']),
        get_result_pointer: Module.cwrap('get_result_pointer', 'number', []),
        get_result_size: Module.cwrap('get_result_size', 'number', []),
      };

      const image = await loadImage('image.jpg');
      const p = api.create_buffer(image.width, image.height);
      Module.HEAP8.set(image.data, p);
      api.encode(p, image.width, image.height, 100);
      const resultPointer = api.get_result_pointer();
      const resultSize = api.get_result_size();
      const resultView = new Uint8Array(Module.HEAP8.buffer, resultPointer, resultSize);
      const result = new Uint8Array(resultView);
      api.free_result(resultPointer);
      api.destroy_buffer(p);

      const blob = new Blob([result], {type: 'image/webp'});
      const blobURL = URL.createObjectURL(blob);
      const img = document.createElement('img');
      img.src = blobURL;
      document.querySelector('.output').appendChild(img)
    };
    </script>

</body>
</html>
docker@ubuntu:~/wasm_test/wasm_image$
```