Source-to-LLVM

and running LLVM in the browser

Bobbie Soedirgo (T1)

Why LLVM?

- Industry grade compiler toolchain
- Modular, easier to build a frontend for relative to e.g. gcc
- Can target many architectures
- De facto choice if your language wants to target Wasm
  - Alternative: cranelift
- WAVM: compile Wasm to LLVM?!

Architecture

- `js-slang`: `program.js` (Source) → `program.json`
- `sourcec`: `program.json` (ESTree) → `program.ll`
- `llvm-wasm`: `program.ll` (LLVM IR) → `program.wasm`

Source-to-LLVM (`sourcec`)

Source §1

program ::= *import-directive* ... *statement* ...

import-directive ::= **import** { *import-names* } **from** *string* ;

import-names ::= ϵ | *name* (, *name*) ...

statement ::= **const** *name* = *expression* ;
| **function** *name* (*parameters*) *block*
| **return** *expression* ;
| *if-statement*
| *block*
| *expression* ;

parameters ::= ϵ | *name* (, *name*) ...

if-statement ::= **if** (*expression*) *block*
else (*block* | *if-statement*)

block ::= { *statement* ... }

expression ::= *number*
| **true** | **false**
| *string*
| *name*
| *expression binary-operator expression*
| *unary-operator expression*
| *expression* (*expressions*)
| (*name* | (*parameters*)) => *expression*
| (*name* | (*parameters*)) => *block*
| *expression* ? *expression* : *expression*
| (*expression*)

binary-operator ::= + | - | * | / | % | === | !==
| > | < | >= | <= | && | ||

unary-operator ::= ! | -

expressions ::= ϵ | *expression* (, *expression*) ...

What's missing?

```
program ::= import-directive ... statement ...
import-directive ::= import { import-names } from string ;
import-names ::= ϵ | name ( , name ) ...
statement ::= const name = expression ;
            | function name ( parameters ) block
            | return expression ;
            | if-statement
            | block
            | expression ;
parameters ::= ϵ | name ( , name ) ...
if-statement ::= if ( expression ) block
               else ( block | if-statement )
block ::= { statement ... }
```

```
expression ::= number
             | true | false
             | string
             | name
             | expression binary-operator expression
             | unary-operator expression
             | expression ( expressions )
             | ( name | ( parameters ) ) => expression
             | ( name | ( parameters ) ) => block
             | expression ? expression : expression
             | ( expression )
binary-operator ::= + | - | * | / | % | === | !==
                  | > | < | >= | <= | && | ||
unary-operator ::= ! | -
expressions ::= ϵ | expression ( , expression ) ...
```

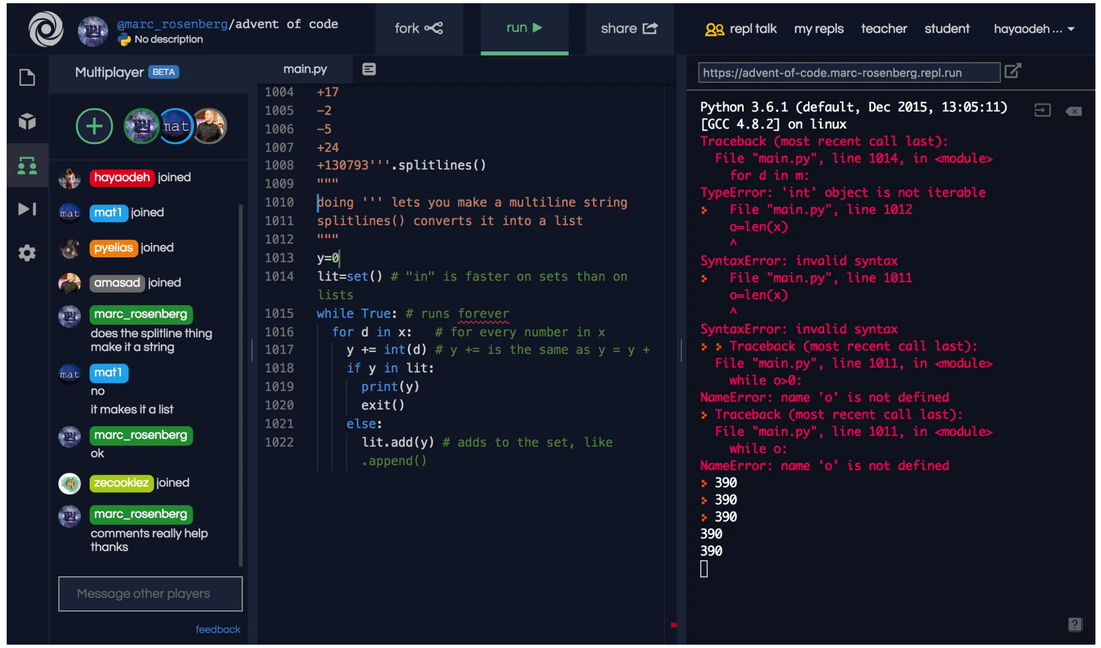
How it works

- Types:
  - `undefined`, `boolean`, `number`, `function`
  - Boxed values
- Environment:
  - Chain of frames, gets passed when entering a new scope
- Conditionals:
  - Phi values
- Functions:
  - Hoisted to top level (including closures)
  - Takes a function pointer and frame pointer as argument

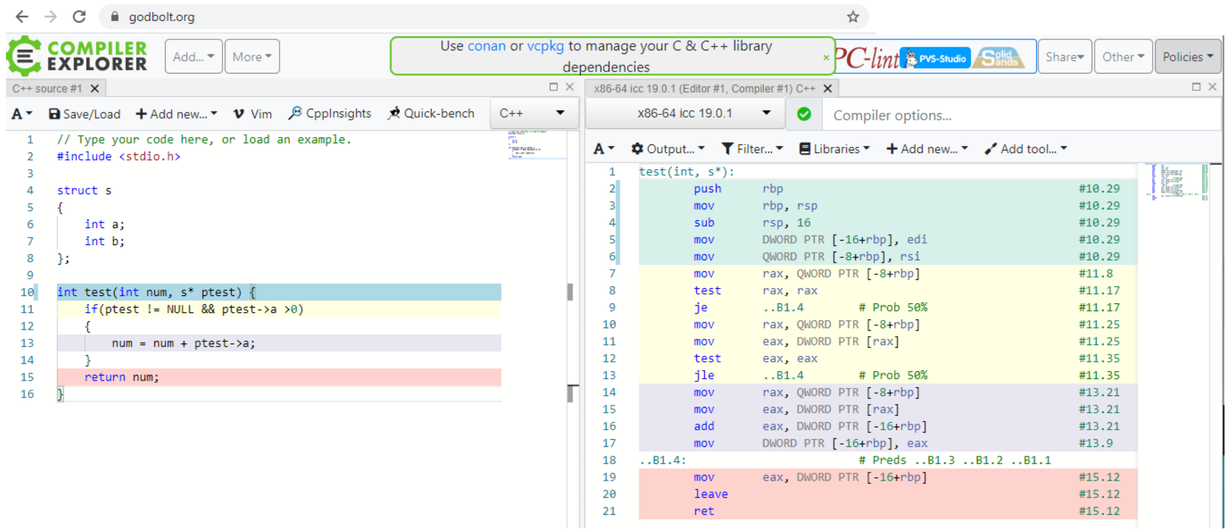
LLVM in the browser (`llvm-wasm`)

Why run in the browser?

Convenience & reproducibility



Replit



Godbolt

Why run in the browser?

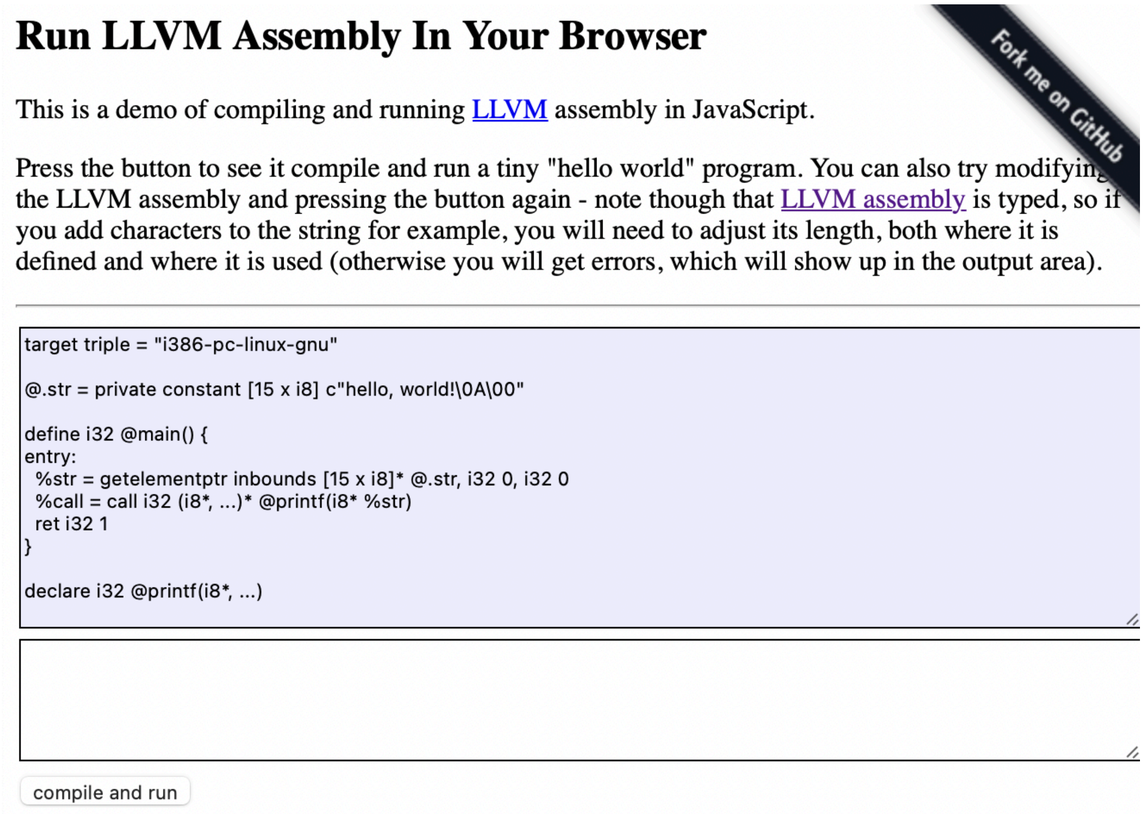
Internet connectivity

- Computer labs, connection issues
- Subpar internet
- Can even run on you phone!

Prior arts

`llvm.js`

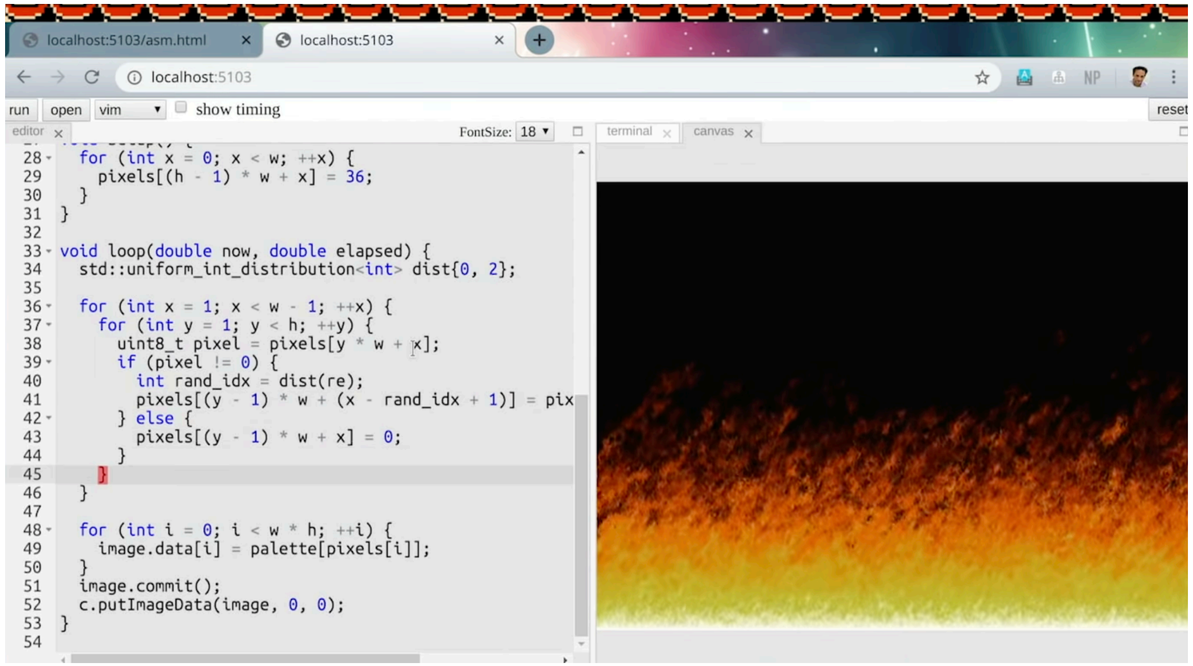
- Compiles LLVM IR to asm.js
- Uses an old (JS) version of Emscripten



Prior arts

`wasm-clang`

- Compiles `clang` to Wasm/WASI
- Using hacked LLVM source
- https://youtu.be/5N4b-rU-OAA



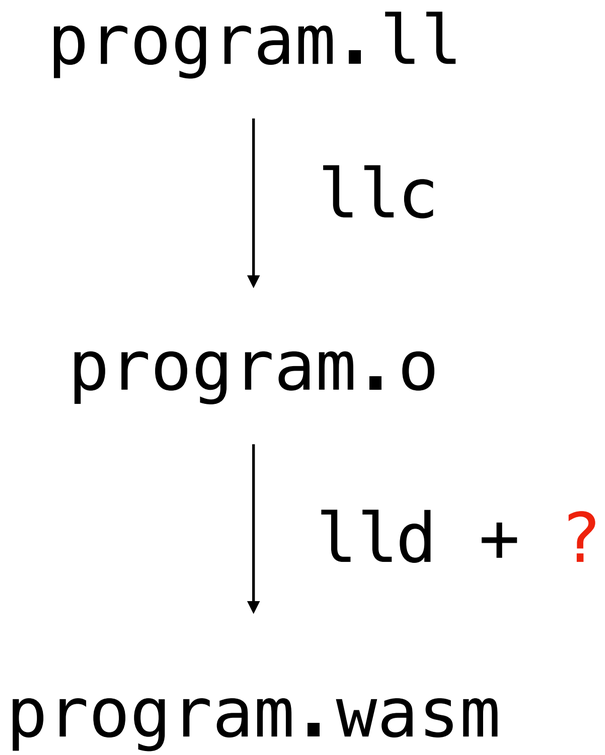
How it works

Which LLVM tools do we need?

- `llc`: LLVM static compiler
  - `program.ll → program.o`
- `lld`: LLVM linker
  - `program.o + ? → program.wasm`
- Compiled to Wasm + JS with Emscripten

How it works

Compilation



Running the Wasm module

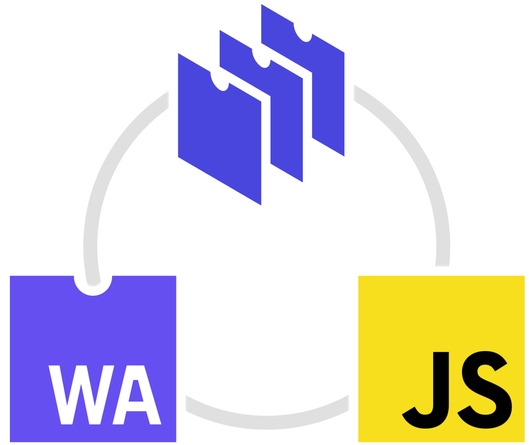
WASI

- WebAssembly System Interface
- Provides a syscall interface to common libc functions
  - `printf`
  - `malloc`
  - `write`
- Can run outside of browser (Wasmtime, Wasmer)

Running the Wasm module

`wasmer-js`

- WASI is not natively supported in browsers yet
- `wasmer-js`: WASI polyfill



Demo

Future extensions

Source-to-LLVM in the browser

- Emitting LLVM IR directly
  - Kinda painful
- Linking to LLVM support libraries
  - Need to know which libraries are needed
  - Need to integrate it with the LLVM C/C++ API/wrapper

Proper tail calls

- Hinges on the Wasm tail call proposal
- https://github.com/WebAssembly/tail-call

Garbage collection

- Hinges on the Wasm GC proposal
- https://github.com/WebAssembly/gc
- Alternative if not targeting Wasm:
  - Boehm-Demers-Weiser: https://github.com/ivmai/bdwgc