Emscripten

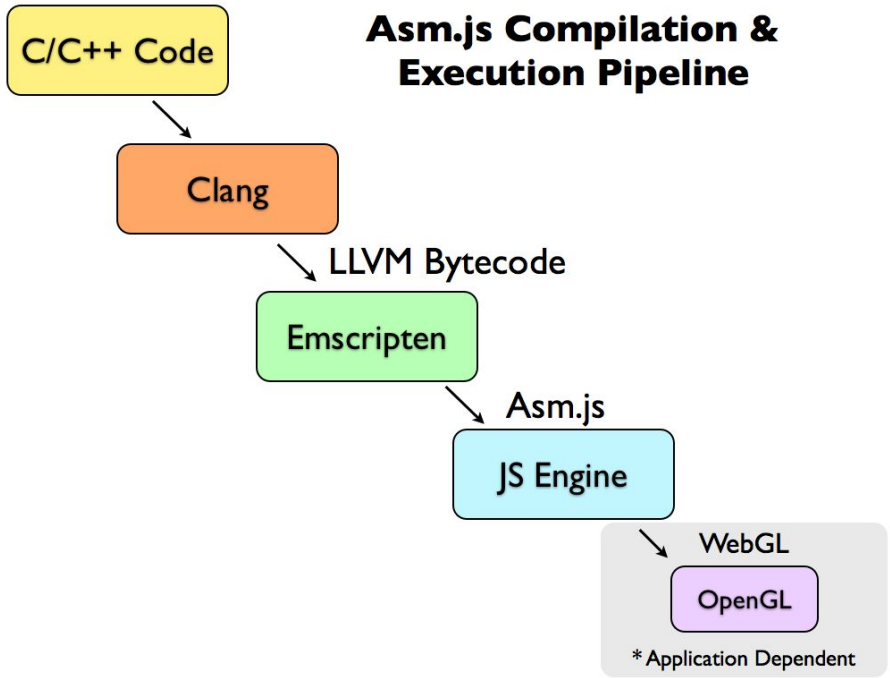
LLVM to JavaScript Compiler

Renata Gjoreska

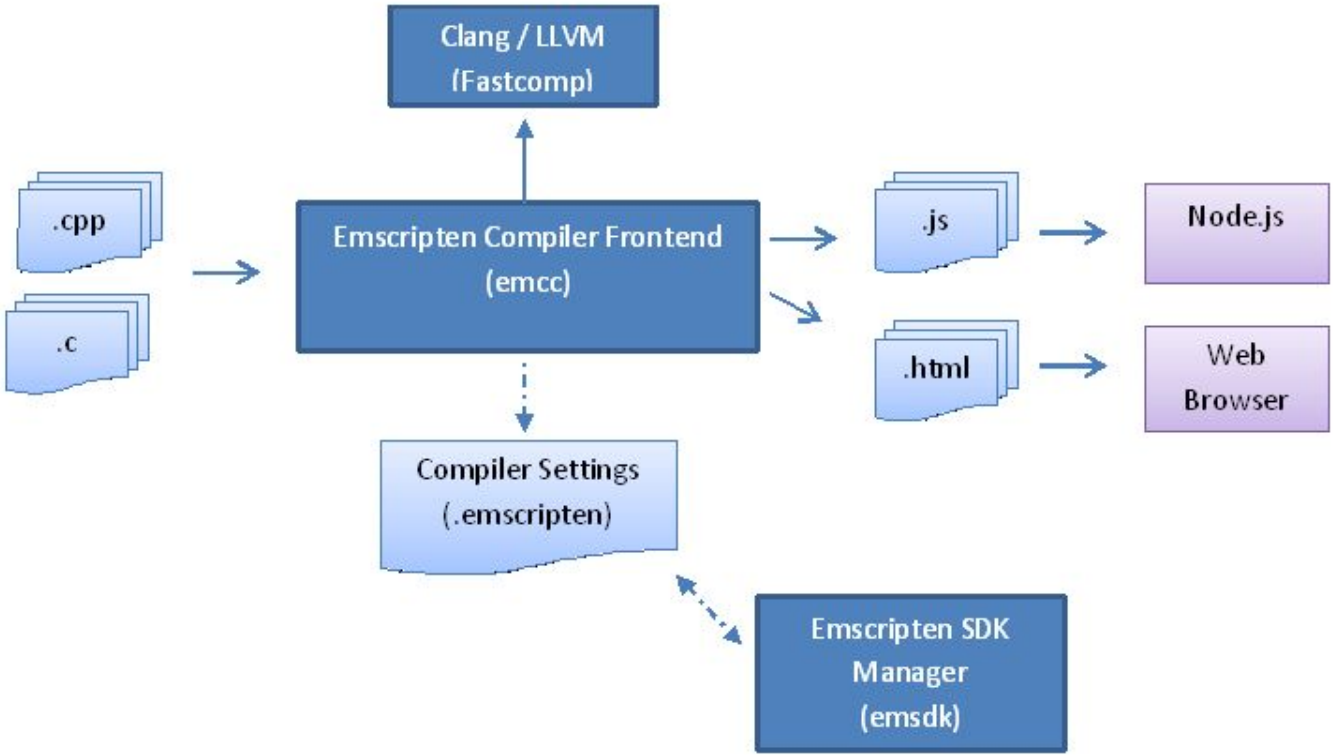
About Emscripten

- Compiles any code that can be translated into LLVM bitcode into JavaScript
- Compiles C and C++ code into JavaScript
- Compiles the C/C++ runtimes of other languages into JavaScript (Lua, Python)

- Support for C standard library, C++ standard library, STL, C++ exceptions, SDL, OpenGL, QT etc.
- Example projects: Unreal Engine 4 and Unity engine



Emscripten Toolchain



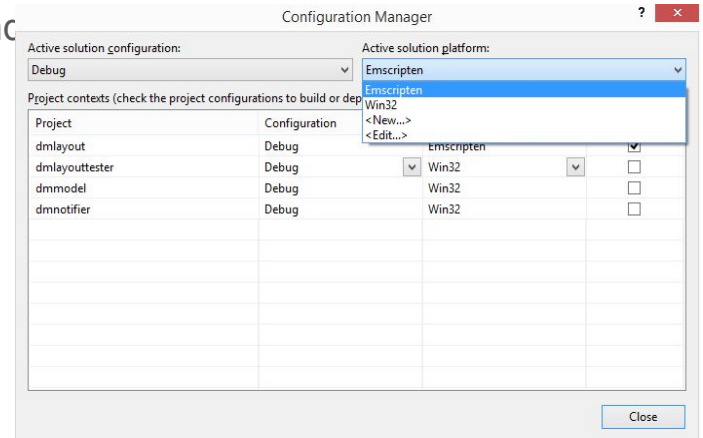
Asm.js

- Asm.js is a subset of JavaScript that is heavily restricted in what it can do and how it can operate.
- Asm.js is just JavaScript - no special browser plugin or feature needed in order to make it work.
- However a browser that is able to detect and optimize Asm.js code will run faster (Firefox)
- Ahead-of-time compilation

Getting started

- The Emscripten SDK provides the whole Emscripten toolchain in a single install package
- You can build Emscripten from source (if you are contributing to Emscripten)
- Download and install instructions:
  - https://kripken.github.io/emscripten-site/docs/getting_started/downloads.html

- On Windows platform:
  - Emscripten is integrated in Visual Studio 2010 using vs-tool plugin, which is automatically deployed by Windows NSIS Installer SDK if VS 2010 is presented on the system at install time
  - To add Emscripten as build option:
    i. Open *Configuration Manager* dialog for the solution
    ii. In the *Active Solution Platform* dropdown choose *New*.
    iii. In the *New* dialog, choose *Emscripten* as the platform name and *Copy settings from Empty*
  - Fastcomp (Clang+LLVM) and Emscripten compiler settings can be edited in exactly same way as other VS settings. On *right-click* on the project and contain two nodes: Clang C/C++ and EMCC Linker.



- On Mac:
  - Simplest way is to use Homebrew: brew install emscripten
  - Compile c++ files with: `./emcc <path>.cpp -o <name>.html`
  - For compiling project use CMake
  - cmake -DCMAKE_TOOLCHAIN_FILE=<EMSCRIPTEN_PATH>/cmake/Modules/Platform/Emscripten.cmake CMAKE_CXXFLAGS="-s -Wno-warn-absolute-paths" will generate appropriate Makefile

Emcc flags

- Generating output:
  - -o <target>: the target filename extension defines output type
  - -o <name>.js, -o <name>.html, -o <name>.bc, -o <name>.o
- Optimization: -O0, -O1, -O2, -Os, -Oz, -O3
- JavaScript code generation option into the Emscripten compiler:
  - -s OPTION = VALUE (ex. `-s RUNTIME_LINKED_LIBS="['liblib.so']", -s EXPORT_FUNCTION="['_testFunction']"`)
- How much debug information is kept when compiling from bitcode to JS:
  - -g<level>
  - -g0: make no effort to keep code debuggable
  - -g4: highest level of debuggability. Shows line number debug comments and generates **source maps** (source maps allow to view and debug C/C++ source code in your browser's debugger! Works in Firefox, Chrome and Safari)

Connecting C++ and JavaScript

- Various options for connecting “normal” JavaScript with compiled (asm.js) code - calling C/C++ compiled functions from JavaScript and vice versa:

- Calling compiled C functions from JavaScript: ccall() or cwrap()
- Calling compiled C++ classes from JavaScript: Embind or WebIDL-Binder
- Calling JavaScript functions from C/C++: emscripten_run_script() or EM_ASM() (faster)

Calling compiled C functions from JavaScript

```
extern "C" int sum(int x, int y)
{
    return x + y;
}
```

- ccall():

```
var result = Module.ccall('sum', // name of C function
'number', // return type
  [number', number'], // argument types
  [10, 5]); // arguments
```

- cwrap():

```
var int_sum = Module.cwrap('sum', 'number',  [number', number']);
int_sum(10, 5);
```

Important Tips:

- **Export all functions that you need to call from JavaScript**
  - *-s EXPORTED_FUNCTIONS="['_sum', '_subtract']"* exports sum and subtract functions
  - It is also important to use _ at the beginning of the function names in the EXPORTED_FUNCTIONS list
  - Functions are called from JavaScript without _ at the beginning of the name
- **Use Module.ccall() not ccall() by itself**
  - *Module* is a global JavaScript object that Emscripten-generated code call at various points in its execution
  - More about Module object: https://kripken.github.io/emscripten-site/docs/api_reference/module.html?highlight=module

Calling compiled C++ classes from JavaScript

- Embind is used to bind C++ compiled classes and functions to JavaScript, so that compiled code can be used in a natural way by “normal” JavaScript.
- Embind provides binding of: functions, classes, abstract classes,value types, enums, constants, smart pointers etc.

- Include emscripten/bind.h
- Set --bind compiler option

Calling compiled C++ classes from JavaScript

```cpp
class Test1
{
public:
    Test1(int number, std::string text)
        : m_number(number)
        , m_text(text)
    {};

    int getNumber() const { return m_number; }
    void setNumber(int val) { m_number = val; }

    std::string getText() const { return m_text; }
    void setText(std::string val) { m_text = val; }

    int squereNumber() { return m_number * m_number; }

    static std::string greeting() { return "Hello all from static method"; }

private:
    int m_number;
    std::string m_text;
};

// Binding code
EMSCRIPTEN_BINDINGS(planner) {
    class_<Test1>("Test1")
        .constructor<int, std::string>()
        .function("squereNumber", &Test1::squereNumber)
        .class_function("greeting", &Test1::greeting)
        .property("m_number", &Test1::getNumber, &Test1::setNumber)
        .property("m_text", &Test1::getText, &Test1::setText)
        ;
}
```

```
var instance = new Module.Example(10, "first example")
undefined
instance.squereNumber()
100
instance.m_number = 5
5
instance.squereNumber()
25
Module.Example.greeting()
"Hello all from static method"
```

Calling JavaScript from C/C++

- For calling JavaScript code from C/C++ use emscripten_runscript():

```
static std::string greeting()
{
    emscripten_run_script("alert('hi')");
    return "Hello all from static method";
}
```

- or use EM_ASM() (and similar macros) - faster way:

```
static void justCallJavaScript()
{
    int x = EM_ASM_INT({
        Module.print('I received: ' + $0);
        return $0 + 5;
    }, 100);

    printf("JavaScript called, now result is %d \n", x);
}
```

- Use:
  - EM_ASM(), if no arguments pass, nothing to return
  - EM_ASM_(), for passing arguments, nothing to return
  - EM_ASM_INT(), passing arguments, return type int (similar for other types)

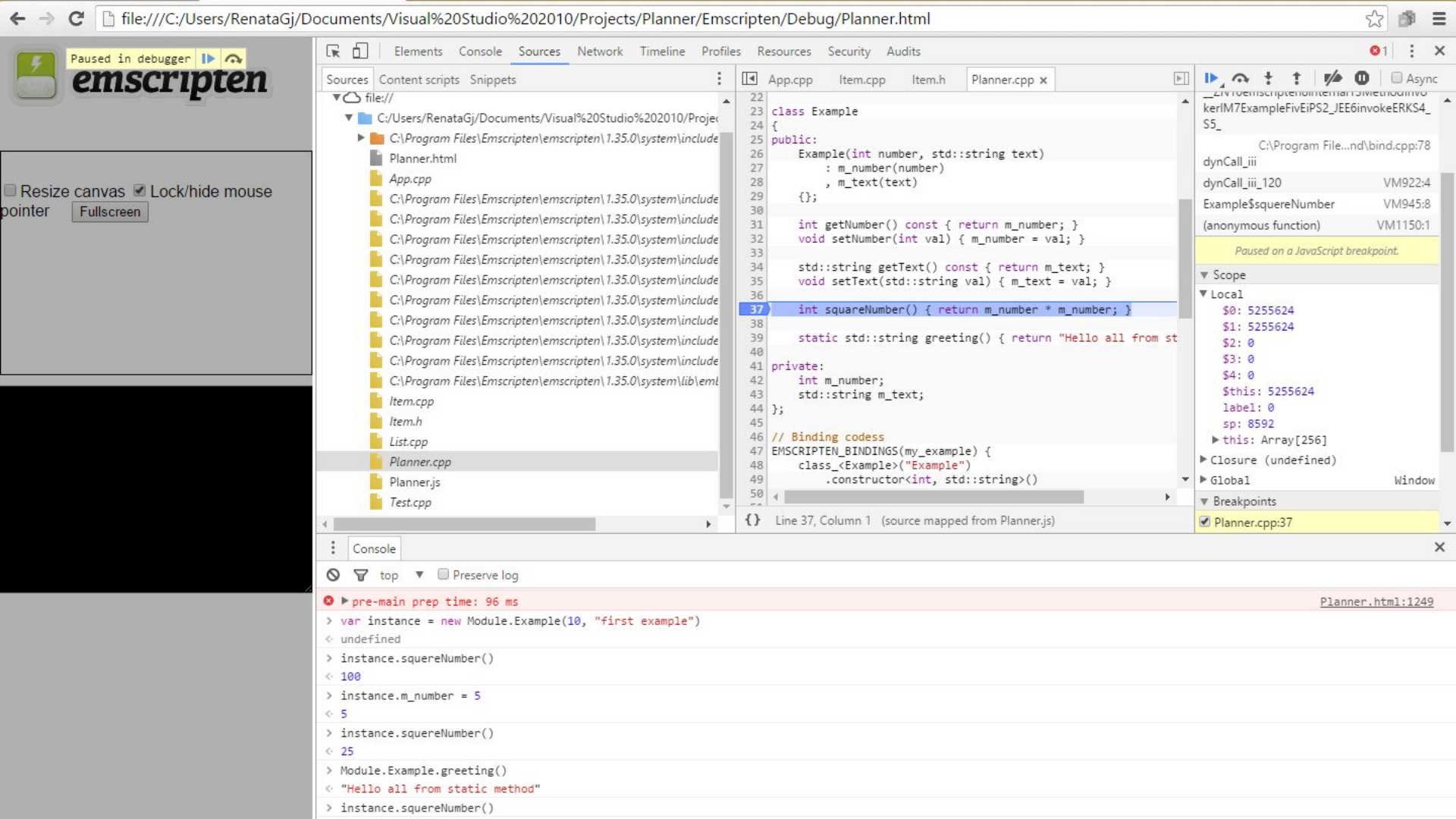
```
EM_ASM( alert('hello world!'); );

EM_ASM_({
    Module.print('I received: ' + $0);
}, 100);

int x = EM_ASM_INT({
    Module.print('I received: ' + $0);
    return $0 + 5;
}, 100);
```

Debugging

- Emscripten cross-platform code can be debugged on either the native platform or using the web browser's toolset.

- For debugging of C++ in browser, source maps should be generated
- For generating source maps use -g4 compiler option
- After compilation .map file will be generated in the output directory
- Run the app in browser and start developer tools, all C++ files should appear in Sources tab

- More here: http://kripken.github.io/emscripten-site/docs/porting/Debugging.html#debugging-emcc-debug


file:///C:/Users/RenataGj/Documents/Visual%20Studio%202010/Projects/Planner/Emscripten/Debug/Planner.html
Paused in debugger
emscripten
Resize canvas
Lock/hide mouse pointer
Fullscreen
Elements
Console
Sources
Network
Timeline
Profiles
Resources
Security
Audits
Sources
Content scripts
Snippets
file://
C:/Users/RenataGj/Documents/Visual%20Studio%202010/Projec
C:\Program Files\Emscripten\emscripten\1.35.0\system\include
Planner.html
App.cpp
Item.cpp
Item.h
List.cpp
Planner.cpp
Planner.js
Test.cpp
App.cpp
Item.cpp
Item.h
Planner.cpp
Async
class Example
{
public:
Example(int number, std::string text)
: m_number(number)
, m_text(text)
{};
int getNumber() const { return m_number; }
void setNumber(int val) { m_number = val; }
std::string getText() const { return m_text; }
void setText(std::string val) { m_text = val; }
int squareNumber() { return m_number * m_number; }
static std::string greeting() { return "Hello all from st
private:
int m_number;
std::string m_text;
};
// Binding codess
EMSCRIPTEN_BINDINGS(my_example) {
class_<Example>("Example")
.constructor<int, std::string>()
Line 37, Column 1 (source mapped from Planner.js)
dynCall_iii
dynCall_iii_120 VM922:4
Example$squereNumber VM945:8
(anonymous function) VM1150:1
Paused on a JavaScript breakpoint.
Scope
Local
$0: 5255624
$1: 5255624
$2: 0
$3: 0
$4: 0
$this: 5255624
label: 0
sp: 8592
this: Array[256]
Closure (undefined)
Global Window
Breakpoints
Planner.cpp:37
Console
top
Preserve log
pre-main prep time: 96 ms
Planner.html:1249
var instance = new Module.Example(10, "first example")
undefined
instance.squereNumber()
100
instance.m_number = 5
5
instance.squereNumber()
25
Module.Example.greeting()
"Hello all from static method"
instance.squereNumber()

Demo code:

https://github.com/RenataGj/planner-emscripten-demo

Thank you! :)