Programming for the Web in your Programming Language

Author: Teo Samaržija

JavaScript

- JavaScript has its special place on the client-side of the web.
- However, you do not need to write everything (or even a significant part) in JavaScript, as many people claim.
- **These days, it is probably easier to make your programming language run on the web than to make it run natively on x86 (AMD and Intel) or ARM (most smartphones) processors.**

WebAssembly

- Originally a Mozilla’s standard for textual representation of JavaScript Bytecode.
- These days, it is both a textual and binary representation of JavaScript Bytecode and is supported by all popular browsers except Internet Explorer 11 (becoming obsolete).
- WebAssembly Binary Toolkit: feature-rich and easy-to-use assemblers and linkers that target WebAssembly.

Installing WebAssembly

- Hard way:
  - Use CMAKE to compile it from source. CMAKE is hard to install properly on many versions of Linux.
- Easier way:
  - Download the binary releases from GitHub, and hope they will work on your version on Linux.
- Easy way:
  - Install NodeJS and NPM. They work without problems on nearly all versions of Linux.
  - Install the version of WebAssembly Binary Toolkit compiled with Emscripten form the NPM repository (called `wabt`).

Why I used C++?

- Many successful compilers are written in C++.
- C++ is the middle ground between JavaScript and Rust in terms of “strictness”.
- No need to worry about deep versus shallow copying (so I thought).
- C++ has advanced significantly since C++98: lambda functions, regular expressions, string manipulation template functions in the standard library, multi-line strings...

Automated Testing

- Some parts of the compiler, such as the tokenizer and the parser, are very easy to test automatically, and you will save yourself a significant amount of time if you automatize it.
- My testing code is 385 lines.

Tokenizer

- Tokenizer is the part of the compiler that tells other parts of the compiler where one word in a programming language ends and where other begins.
- ~~MISCONCEPTION: You need to learn Lex or a similar tool.~~
- I used only C++ standard library, and it is 256 lines of code.

Parser

- Parser is the part of the compiler that tells other parts of the compiler which word is connected to each word grammatically.
- ~~MISCONCEPTION: You need to learn to use tools such as YACC or BISON.~~
- ~~MISCONCEPTION: You need to learn to use advanced object-oriented structures, such as builders and composites.~~
- I used only C++ standard library and it is 964 lines. It is a recursive algorithm using collections and iterators.

Semantic analysis

- Check for the malformed sentences that parser does not catch, but which will crash the compiler. Such also exist in natural languages: *More people have been to Russia than I have.*
- Annotate the types of expressions in the syntax tree, to make things easier for the compiler.
- My semantic analyzer is 292 lines of code.

Compilation Context

- You need to make data classes that will keep the information about the current state of compilations (information about currently defined variables, functions, structures…).
- My `compilingContext.cpp` file is 50 lines of code, my `AssemblyCode.cpp` file is 83 lines, and my `TreeRootNode.cpp` file, which initializes those data classes, is 824 lines of code.

Compiler

- In the strict sense, “compiler” is only the subprogram that translates the syntax tree into assembly code.
- ~~MISCONCEPTION: You need to use advanced object-oriented concepts, such as visitors.~~
- ~~MISCONCEPTION: You need to deal with the bytecode or machine code.~~ That is mostly the assembler’s job, not the compiler’s job. Of course, sometimes the C++ bit manipulation comes useful, because it is often easier to do something in the compiler than to properly delegate it to the assembler.
- ~~MISCONCEPTION: Frameworks such as LLVM make things significantly easier.~~ WebAssembly has about the same features that LLVM IR has. It supports the statements such as `if`, `else`, `loop`...
- My compiler is 1358 lines of code.

Pros of writing in your language

- Every programming language has quirks which people do not know about, and those quirks are a major source of bugs.
  - Using EMSCRIPTEN (which compiles C++ to JavaScript Bytecode) might even make one worse-off, because, although JavaScript has many quirks, the quirks of JavaScript tend to be well-known to web-developers.
- You have a better understanding of how the environment works.

Quirks in programming languages

```
[teo.samarzija@teos-acer-laptop ~]$ node
Welcome to Node.js v12.16.3.
Type ".help" for more information.
> [1, 2, 11].sort()
[ 1, 11, 2 ]
>
```

```
function getStudent() {
    //Returns a null-pointer instead of a structure.
    return
        {
            name: "Teo",
            surname: "Samarzija"
        };
}
```

```
TreeNode(std::string newText, int newLine, int newColumn) {
  *this = TreeNode(); // For some weird reason, just "TreeNode()" won't do the
                      // trick.
  text = newText;
  lineNumber = newLine;
  columnNumber = newColumn;
}
```

```
        for (auto &pair : contextOfThatFunction
                              .localVariables) // The reference operator '&'
                                               // is needed because... C++.
          pair.second +=
              isPointerType(argument.text)
                  ? 4
                  : basicDataTypeSizes.count(argument.text)
                        ? basicDataTypeSizes.at(argument.text)
                        : context.structureSizes.at(
                              argument.text); // Push all the variables
                                              // further back on the stack.
```

Using default copy constructor corrupts a tree in C++

Asked 26 days ago Active 25 days ago Viewed 73 times

1

For example:

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;

struct TreeNode {
    vector<TreeNode> children;
    string name;
};

int main() {
    TreeNode leafNode;
    leafNode.name="c";
    TreeNode middleNode;
    middleNode.name="b";
    middleNode.children.push_back(leafNode);
    TreeNode rootNode;
    rootNode.name="a";
    rootNode.children.push_back(middleNode);
    rootNode=rootNode.children[0];
    cout <<rootNode.name <<endl;
}
```

Outputs `c`, in both CLANG and GCC. Of course, I'd expect it to output `b`. What's going on here?

Cons of writing in your language

- You need to write *everything* yourself – a major source of time-consuming bugs.
  - Communicating with useful JavaScript frameworks is not nearly as trivial as it is when writing in JavaScript.
- Making a language capable of useful string manipulation is not trivial.
- Programming tools tend to have poor support for non-C-like languages.

You need to write *everything*

```
137 Function sin(Decimal32 degrees) Which Returns Decimal32 Does {
138   If(degrees < 0) Then { Return - sin(-degrees); }
139   EndIf;
140   If degrees > 90 Then { Return cos(degrees - 90); }
141   EndIf;
142   If not(sineMemoisation[asm_f32("(f32.nearest (local.get 0))")] = 0)
143       Then { //"(local.get 0)" means, in WebAssembly, "Get the first argument of
144              // this function.", and the first argument of this function is
145              //"degrees". I've used inline assembly here because nothing else I
146              // write will output "f32.nearest" (called "round" in most
147              // programming languages) WebAssembly directive, and it's way more
148              // convenient to insert some inline assembly than to modify and
149              // recompile the compiler.
150     Return sineMemoisation[asm_f32("(f32.nearest (local.get 0))")];
151   }
152   EndIf;
153   /*
154    * Sine and cosine are defined in Mathematics 2 using the system of
155    * equations (Cauchy system):
156    *
157    * sin(0)=0
158    * cos(0)=1
159    * sin'(x)=cos(x)
160    * cos'(x)=-sin(x)
161    * ---------------
162    *
163    * Let's translate that as literally as possible to the programming
164    * language.
165    */
166   Decimal32 radians : = degrees / oneRadianInDegrees, tmpsin : = 0,
167                     tmpcos : = 1, epsilon : = radians / PRECISION, i : = 0;
168   While((epsilon > 0 and i < radians) or (epsilon < 0 and i > radians)) Loop {
169     tmpsin += epsilon * tmpcos;
170     tmpcos -= epsilon * tmpsin;
171     i += epsilon;
172   }
173   EndWhile;
174   Return sineMemoisation[asm_f32("(f32.nearest (local.get 0))")] : = tmpsin;
175 }
176 EndFunction;
```

```
205 // Now, let's implement some string manipulation functions. We can't
206 // call the methods of the JavaScript "String" class here.
207 Function strlen(CharacterPointer ptr) Which Returns Integer32 Does {
208   Return ValueAt(ptr) = 0 ? 0 : 1 + strlen(ptr + 1);
209 }
210 EndFunction;
```

I added the support for curly-brace-limited blocks into my language only to be able to use features of programming tools I usually use.

How to install my compiler?

- You need to also install WebAssembly Binary Toolkit to use it.
- You can download the releases on GitHub (my repository is called `AECforWebAssembly`). The Linux executable files there are statically linked, so they should run on wide range of Linuxes. There is also a 32-bit Windows executable, a FreeDOS executable and a WebAssembly executable for NodeJS.
- On most Linuxes, the source code of my compiler can be compiled with a one-liner.

Thank you for your attention!