

# 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 from 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 ]
> 
```

```
1 function getStudent() {
2     //Returns a null-pointer instead of a structure.
3     return
4     {
5         name: "Teo",
6         surname: "Samarzija"
7     };
8 }
```

```
188 TreeNode(std::string newText, int newLine, int newColumn) {
189     *this = TreeNode(); // For some weird reason, just "TreeNode()" won't do the
190                        // trick.
191     text = newText;
192     lineNumber = newLine;
193     columnNumber = newColumn;
194 }
```

```
381     for (auto &pair : contextOfThatFunction
382             .localVariables) // The reference operator '&'
383                             // is needed because... C++.
384         pair.second +=
385             isPointerType(argument.text)
386             ? 4
387             : basicDataTypeSizes.count(argument.text)
388             ? basicDataTypeSizes.at(argument.text)
389             : context.structureSizes.at(
390                 argument.text); // Push all the variables
391                                // 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

For example:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

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!