What? → Why? → How?

A project so ridiculous and asinine it’ll make you say “tf?”

By whom: Adam McDaniel, Vicky Chakpuang, Logan Wrinkle

What Did We Make?

We wrote a collection of compilers based on complexity *so absurd and unnecessary* that it becomes comical.

Stage One: *How* does it work?

For more information on BF, visit
*https://esolangs.org/wiki/Brainf***

- This is Brainfuck, a language designed to be as *minimal* as possible. It is ***barely*** turing complete.
- Our overarching compiler takes our high level language, designed to be readable by humans, and writes an ***equivalent program in this***.

Operations of Our Stage 1 (*How*) Backend

Symbol	Description
+	Increment the value at the tape pointer.
-	Decrement the value at the tape pointer.
<	Move the tape pointer left.
>	Move the tape pointer right.
[	Begin a while loop with the value at the tape pointer as the condition.
]	End the while loop.

Symbol	Description
,	Read a character from STDIN into the tape at the pointer.
.	Print a character to STDOUT from the tape at the pointer.
*	Set the pointer equal to the value at the current cell.
&	Undo a * operation.
?	Allocate a given amount of memory dynamically and store the result in the current cell.
!	Free the pointer at the current cell.

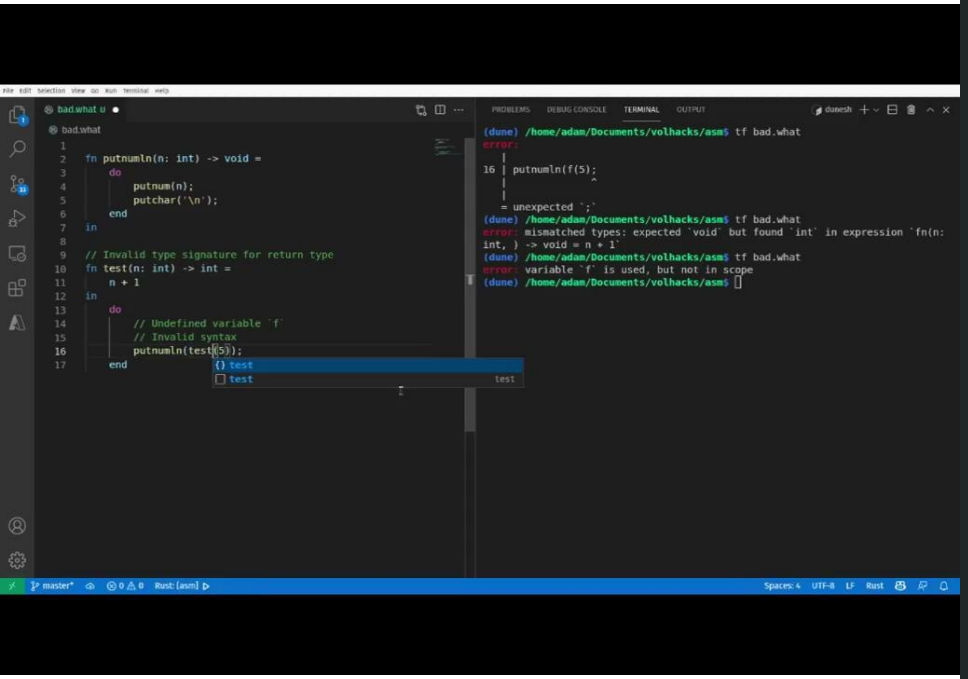
Stage Two: *Why*?

- To compile to Brainfuck effectively, we created an *intermediate representation* (named *Why*): a language in between the frontend and backend.
- This mediates the difficulty of compiling a high level language all at once.
- It implements lower level abstractions, like a stack and heap, for the higher level language to make use of.

Stage Three: *What*?

- Finally, we compile our high level language (named *What*) to our intermediate representation, which in turn is assembled to Brainfuck.
- We enforce a small type system with strict rules, we compile function calls and nested block scopes, and we even allow manual memory management with ***alloc*** and ***free***.

Stage Three: *What*?



- Our error checking is extremely nice for such a short window to implement the project!

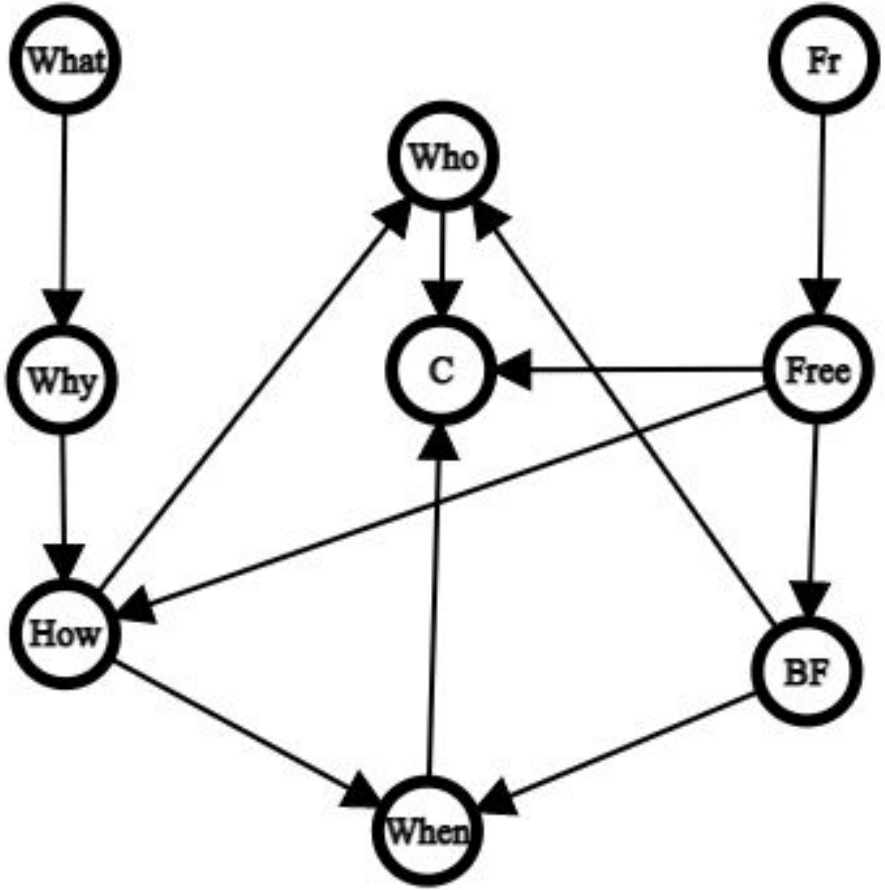
Compiler Collection Overview

1. In addition to our main compiler for *What* and *Why* to *How*, we also have a few other related compilers in the family.

2. *When* is a compiler from both Brainfuck and *How* to *C*, which allows us to quickly run our generated code.

3. *Who* is a ***self-hosting, bootstrapped compiler implemented in pure Brainfuck*** capable of compiling both Brainfuck and *How* code.

4. *Free* is a very stripped down, untyped language that can compile to pure Brainfuck, which we used to implement *Who*.



Why Did We Make This?

We were very impressed by the notion of turing completeness, and we wanted to explore its extremes.

What Did We Learn?

All programming languages are fundamentally equivalent: they are merely different implementations of the same thing.

Anything We'd Like the Judges to Consider?

Even though it may look small, keep in mind that this was an ***extremely*** difficult accomplishment: *a type-checking program that writes other programs*. This is a highly involved theoretical Computer Science problem, and we are very satisfied with our results.


Thank you! 👋