Time limit: Maximum 8 hours (strong candidates typically finish in 4-6 hours)
Submission: Public GitHub repository (or zip) with excellent README, tests, and clean code structure
External libraries: None allowed (except for your test runner if you want Jest/Vitest/etc.)

Objective

Build a complete mini JavaScript -> WebAssembly Text Format (.wat) compiler from scratch.

The output .wat file must be valid and executable by any standard WebAssembly runtime (wasmtime, wabt, wasm3, etc.). Binary .wasm output is not required. Text format is sufficient.

Supported Language Subset

Your compiler must fully support **only** the following JavaScript features (and nothing else):

- Integer arithmetic with the operators: + - * / % == != < > <= >= ! (unary negation)
- Variable declarations: let and const (block-scoped, no reassignment required for const)
- if / else statements
- while loops
- Function declarations and calls (parameters and return)
- Block statements { ... }
- Expressions as statements, and top-level expressions
- Nested function calls with arbitrary expressions as arguments
- No floats, strings, objects, arrays, classes, arrow functions, for-loops, switch, try/catch, etc.

All values are 32-bit signed integers (i32 in WebAssembly).

Required Test Programs

Your compiler must correctly compile and run **all** of these programs with no modifications:

1. Factorial (iterative)

```
function fact(n) {
  let result = 1;
  while (n > 0) {
    result = result * n;
    n = n - 1;
  }
  return result;
}

fact(5);  // must return 120
```

2. Euclidean GCD

```
function gcd(a, b) {
  while (b != 0) {
    let t = b;
    b = a % b;
    a = t;
  }
  return a;
}
gcd(48, 18);  // must return 6
```

3. Ackermann function (recursive - the real filter)

```
function ack(m, n) {
  if (m == 0) return n + 1;
  if (n == 0) return ack(m - 1, 1);
  return ack(m - 1, ack(m, n - 1));
}
ack(3, 4);  // must return 125
```

Runtime Requirements

- Provide a simple CLI: `node compiler.js input.js > output.wat`
- The generated module must export a function named `_start` that executes the entire program and returns the value of the final top-level expression.
- Alternatively, export each top-level function by name and provide a tiny loader script that instantiates the module and invokes the desired export.
- We will test with `wasmtime output.wat --invoke _start` (or your loader).

Evaluation Criteria (in order)

1. 100% correctness on all three programs above (especially Ackermann 3,4 - most submissions fail here)
2. Clean separation of concerns (lexer -> parser -> AST -> code generator)
3. Correct WebAssembly stack machine usage (no stack corruption, proper control flow with block, loop, br, br_if)
4. Proper handling of block scoping and local variable allocation
5. README with clear instructions and architectural overview
6. Comprehensive tests (at minimum the three programs above as golden tests)

Bonus (not required, but instantly identifies exceptional candidates)

- Tail-call elimination (so ack(3, 8) or higher runs without stack overflow)
- Basic constant folding / dead-code elimination
- Source-location comments in the generated .wat
- Distinction between let and const (immutability where possible)

Explicitly Out of Scope

- Any ES6+ features
- Floating-point numbers
- Strings, arrays, objects, DOM
- Error recovery (crash loudly on unsupported input)

This challenge is designed to find engineers who truly understand parsing, scope, control flow translation, and the WebAssembly stack machine.

Good luck. We're excited to see what you build.