**Vicharak Documentation**

**What Does This Do?**

This program is like a baby version of how compilers work. It reads a simple line of code like int x = 5;, checks if it makes sense (no typos or missing stuff), and then pretends to turn it into something a computer can understand.

It has three parts:

1. **Lexer**: Breaks the code into smaller pieces called tokens (like words in a sentence).
2. **Parser**: Makes sure the tokens are in the right order, like grammar in a sentence.
3. **Code Generator**: Pretends to create machine-level instructions from the parsed code.

**How It Works (Without Jargon)**

**Lexer (The Token Breaker)**

The lexer reads the code, skips any spaces, and figures out what each piece is. For example:

* int → It's a keyword.
* x → It's the name of a variable.
* = → It's an assignment symbol (like saying "x equals something").
* 5 → It's a number.
* ; → It means the line is finished.

It does this for the whole input line and gives back a list of all these "tokens."

**Parser (The Rule Checker)**

The parser looks at the tokens and checks if they’re in the right order. For example:

* Does it start with int?
* Does it have a proper variable name after that?
* Is there an = followed by a number?
* Does it end with a ;?

If something is missing or in the wrong place, it throws an error like "Expected a number after '='." If everything is good, it says, "Parsed successfully!"

**Code Generator (Fake Computer Translator)**

Once the parser confirms everything is correct, the code generator pretends to turn the line into assembly instructions. It’s not real assembly, just a demo. For example:

* Load 5 into a "register" (a computer's temporary storage).
* Store that value into memory for the variable x.

It’s just a way to show how high-level code becomes low-level instructions.

**What Can This Handle?**

* A single line of code: int x = 5;.
* That’s it—no loops, no functions, no crazy math. Just basic variable declarations and assignments.

**Why Is This Useful?**

* It’s a simple introduction to how compilers work.
* Shows the three steps: breaking things down, checking rules, and creating instructions.
* Gives you a base to build on if you want to learn more about compilers or programming languages.