# Compilers – Day 2

Let’s sling some code

## Lexical Analysis

Lexical analysis is the process of turning your raw input, in this case source code, into meaningful symbols. This is the first major component of your compiler and the simplest. At a high level, the component needs to be able to take in a string of source code and process it into a stream of tokens.

An example of this is:

func fib(n) {

if (n > 2) return fib(n - 1) + fib(n - 2);

return 1;

}

Becomes:

Func, Ident, LParen, Ident, RParen, LBracket, If, LParen, Ident, GreaterThan, Int, RParen, Return, Ident, LParen, Ident, Minus, Int, RParen, Plus, Ident, LParen, Ident, Minus, Int, RParen, SemiColon, Return, Int, SemiColon, RBracket

The reason we do this is so that subsequent components are not worried about whether some whitespace matters, how to parse a string, or what the exact text of a keyword is.

Now in the above example we just show the associated symbol for each token, in my case enumeration values, but some of them have more data associated than the symbol. For example the Ident tokens also have a string with them for the identifier name, the Int tokens have the integer value, etc. Additionally, if you plan on being able to pinpoint errors during later compilation stages, your tokens will need more data associated with them to help generate meaningful errors such as:

Lexical Analysis Failure: LexErr::UnknownSymbol

3:11> a = true ? "yes" : "no";

3:11> ^

Whether you plan to support rich error generation is up to you.

## Week 2 Homework

Your homework this week is to start writing code as well as to continue working on your grammar based on the feedback you received during check-in this week.

For the code portion, you will be writing your lexical analyzer (or tokenizer or lexer). This is required to be a standalone component that takes in a string or a file and exposes at least two methods:

* Current() – returns the token that the lexical analyzer most recently processed
* Advance() – tells the tokenizer to process until it has the next token

This lexical analyzer must be able to tokenize your test code and any additional test code you need to exercise the nuances of your grammar. Additionally, it is required to:

* Handle whatever comment style you have settled on
* All built in types (numbers, strings, booleans, etc.) must be processed here
* Bad syntax such non-terminated strings or ill-formed numbers need to generate an appropriate error
* Have an associated test suite that runs source code through it and verifies that it succeeds or fails as expected, as well as print out all symbol data processed for validation in a format of one symbol per line

## Homework Check-in

I’m going to walk around and give people individual feedback on their grammars and discuss any common issues I see between them.