A Real Tokenizer is a program that takes an input stream of characters and outputs individual tokens of varying types based on the first char of each token. The tokenizer is modeled after a finite state machine, and it is always in a single state at any given time. After calling the function associated with a particular state, that function returns a state value to indicate the next function to call whether it is a repeat call or a call to a new function.

The tokenizer has distinct states for each type of token. Whitespace is ignored, and nothing is outputted. Escape characters do not have a specific state, but they are outputted as errors before scanning the next character. Alphanumeric, decimal, octal, hexadecimal, and operator tokens each have one function that is repeated called to account for variable token size and another function that is called at the end of a token to print that particular type of token out. Floating-point tokens have a third function that is called within the first floating-point function when an exponent is detected within the floating-point token. By creating states for each type of token, I was able to create modular code.

The qualifications for tokens vary depending on the type of token. Generally, tokens are terminated when a whitespace or a character not acceptable to that token is found immediately after. This means that input that potentially has malformed tokens will likely output several smaller tokens of varying types. Alphanumeric tokens need only start with a alphabetic letter, then they can have any number of digits or alphabetic characters after them. Decimal tokens start with any digit from 1-9 and have any number of digits afterwards. In the case of 0, especially in ambiguity with octal numbers, a lone 0 is considered a decimal constant 0. Octal tokens start with a 0 and have any number of octal digits after it. Hexadecimal tokens start with a 0x or 0X and have any number of hexadecimal digits after it. If there are no hexadecimal digits after 0x or 0X, the token is considered malformed and is printed as such. Floating-point tokens can be printed with or without an exponent. If the floating-point token has an exponent with incorrect structure, such as a lack of digits after the ‘e’ or a positive or negative sign without following digits, then the token is considered malformed and is printed as such. Operator tokens are checked against a table of valid C operators, and larger operators are considered first before printing smaller operators.

I did not use any data structures in my program apart from the tokenizer struct that I was provided. When a token was returned from TKGetNextToken(), I immediately printed out the character string and then freed the dynamically allocated memory of the string. This minimized the amount of memory used since I did not need the token once the output was printed.