Christopher & Sandeep
Tokenizer

For this project, we chose to implement a state machine through a switch statement and a state enum.
Firstly, we copy and then preprocess the input string. By preprocessing the input string we were able to turn all of the special whitespace characters: \t, \v, \f, \n, \r into a space character. This means that we have to go through the string at least twice in order to split the input string into tokens, but it also allows us to easily reconfigure the project to accept other characters as delimiters, making the project more portable.

When creating our tokenizer struct we thought about storing the token and type inside of the tokenizer struct, and just using realloc if they were unequal to null. However, we chose to lookup the type based on the STATE enum because we thought it would be easier to add a new entry into the enum State and stateTokenPrint should we ever need to expand the token types we consider.

We did not wish to return inputs such as 0 or .1 as bad tokens, so we made a design decision to set specific states for these two edge cases. We also used two arrays to handle C operators, one array for the token and a corresponding type array. This enables us to add additional C Operators in the future much more easily, greatly enhancing portability.