Anirudh Tunoori
netid: at813
RUID:165003175
tokenizer.c

My program implements the basic skeleton along with a few additional methods. I designed a tokenizer struct with three elements an index, a size, and a char* that contains the input string that was copied from argv[1] from the main function. My destroy method simply frees the tokenizer struct and is called only once at the end of the main function to free the dynamic memory that was allocated in create. My get next token method breaks the “string” in the tokenizer struct by using any whitespace as delimiters. I dealt with the whitespace by using the is space function. The tokens are then returned to the main function one by one. As the main function receives the tokens it is passed to a function called identify (outside the skeleton). Identify passes the token into a stateTransitionAlgorithm which simply changes the state of an enum, id before returning the identity of the token back to the main function. The states found in the state transition algorithm correspond to the arcs and states of the finite state machine. There are a total of eight states along with one “mal” state in my implementation. The states change the status of the id (enum). Finally I dealt with the escape characters in the main function with a series of if statements. I stored the escape characters in a char variable and printed the appropriate hex value. Some challenges that I encountered during my work had to do with transitioning through the various state functions. While the implementation was not overly difficult, debugging and testing all of the states were time consuming. When working on the get Nexttoken function I faced a few difficulties with adding the null byte to the end of each token and allocating space for the token correctly. Finally, It took me while to realize that I accidentally dereferenced a pointer while it was still in use at a later stage in the program, the challenge lay in locating the source of error since it was not immediately obvious.

Special Features and Considerations:

The single quote and exclamation mark sometimes results in incorrect output ex: “!5a” or ” 8\’ ”
Since I considered white space to be the delimiters for tokens, my program treats 123p123 as one token that is identified as invalid due to the character ‘p’.