Readme for tokenizer project by trf40, tfp15.

BRIEF OVERVIEW: tokenize() takes a single char * input and iterates through the input to break it into tokens, which are then saved into a linked list. The pointer for the head of the list is returned by tokenize(), making the function portable for other applications with the removal of the main() function. There are two global variables, startLoc and currentLoc, that hold locations for iterating and reading down the string input. The idea is that startLoc holds the starting location for the current token, and currentLoc holds the current/ending location, depending on where the code is in the logic. This allows me to calloc appropriate amounts of space when creating a node, and use strncpy() to copy over only the correct number of characters, before startLoc and currentLoc are updated for the next token.

tokenize() checks for any white space characters, alphabetical characters, punctuation (which means an operator), reads ahead to check for hex integers, octal, and decimal integers. Each sub-function has its own functions it sends the input to, depending on if the logic needs to change what type of token the current token is.

All three extra credit opportunities have been taken care of. Every C keyword is recognized, and the code is able to check if it's even possible for a token to be a keyword by length, so as to reduce the number of comparisons. The two types of comments are recognized; comments with // will skip all characters until a '\n' character is found, and /* comments continue until either the end of the string, or an ending */ is found. Last, the isPunctuation() function checks for single or double quotes in order to detect strings. If a quote is found, the code will iterate down until it finds a quote that matches the original quote (if the first quote was a double quote, single quotes will not end the string), and then the entire string is made into one string token regardless of content. The code also checks to make sure the matching quote mark is not escaped; if it is, then the string will continue past that. If, however, the string has no matching quote mark by the end of the entire input string, the entire input is an invalid input and the tokenizer function returns NULL for the head, marking the function as having failed.

The code is incredibly modular, with functions being split up very specifically by function and intention.