Brief Description

The code takes an input from argv[1] and scans through character by character. If the character is a number, the function parse_digit is called. If the character is a letter, the function parse_word is called. If the character is a quote, the function parse_quote is called. Anything else calls parse_operator. If a token is found, our print_token function is called to print the token and the iterator jumps to the end of the token and continues to scan. If a single line comment is found, the program is exited. If the start of a multiline comment is found, a variable called "commenting" is set to 1 and the program continues to scan for the end of a multiline comment where "commenting" is set to 0.

- parse_digit finds the end of a token and determines whether the characters create a token of type decimal integer, hexadecimal integer, octal integer, or float.
- parse_word finds the end of the word token and then checks if it is a keyword in our hash table using the function searchHash()
- parse_quote searches for the closing quote
- parse_operator finds the end of a token and determines the name of the operator

Error Handling

If there is no input, "No arguments" is printed and the program is exited.
If there is an unmatched quote, end of a multiline comment without a start, or unspecified operator, "invalid token" is printed.
When mallocing for the hash table, if there is a memory allocation error, "Memory allocation error!" is printed and the program is exited.

Efficiency

The components of our algorithm that contribute to the runtime are scanning through the string, printing tokens, and comparing tokens. Each of these steps are done in O(n) time so the worst case runtime is O(n) .
The only space we use is for the hash table which has O(1) space as it only contains the 31 keywords.

Great Features

We only require O(1) space for the hash table and nothing more since we implemented a struct which stores the end position of a token and the token name. This allowed us to print the token without having to use any space.

We decided to implement the hash table for keywords because it would be much faster to search for a keyword than doing 31 strcmp in if statements. We did so by creating 35 buckets since there are 31 keywords and chained using linked lists. To determine the bucket, we summed up the total characters in a string.