Jasmin Kaur(jk1539), Jonathan Delin (jd1274)

Systems Programming

Professor Menendez

4 October 2020

Asst0

For this assignment, the approach we decided to implement was to iterate throughout the string and print out a token as each one is encountered. We created a copy of the string that was inputted, and passed it to each of our 3 main functions: opFunc, wordFunc, and numFunc. These functions were called if the character was a number (sent to numFunc), a letter (sent to wordFunc), or neither (sent to opFunc) after it was checked to see if the character was white-space.

In the opFunc for operators, we implemented if-else statements to check and see if the character was equal to any of the operators. The operator if-else statements were ordered in a way that would search greedily for the longest operator that it could find. If so, it would then print the operators one at a time until it reached a character that was not one of the specified operators. In the wordFunc for words, we created a temporary string called token. A character was added to that string only if it was a letter or a number. This loop would eventually end when it reached a character that was neither of those. Before printing the token string, it would check to see if the string only contains "sizeof". If so, it would print that as an operator (structure member) instead of a word. If it contains sizeof and other letters/numbers it would print as a word token.

Lastly, we utilized the numFunc for numbers which housed the most edge cases by far. For floating points, we had to take into account if the temporary string “token” contained a decimal or an exponent. If it contained an exponent, we had to make sure it did not contain two or more in a row (i.e 1.2e10e10). We also had to take into account that the exponents could be written as: e10, e+10, e-10, E10, E+10, and E-10. The exponent could also not contain another floating point or hexadecimal integer. For hexadecimal, we implemented switch statements to check and see if after the 0x came either a digit from 0-9, a letter from A-F, or a letter from a-f.

For octal, there were also many edge-cases. We had to make sure just a 0 would print as octal. We also had to check and see when a 0 was encountered, if the temporary string already contained numbers in it. If it did, then the 0 would just be added to that and the loop would continue and eventually print as a decimal integer. However, if the string was empty, then that meant the 0 would be the first number going into it, possibly making it octal. Yet, it could not be octal if a 8 or 9 came after the 0, so we had to add an if statement for that as well. If it passed those 3 cases, then a new loop would start that would continue adding characters to the temporary string until an 8 or 9 was encountered or a character that was not a number. If it reached a “.” then that meant it could possibly be a floating point instead of an octal integer (i.e 077.123). For this case, we simply checked to see if the current character was “.” and the one directly after it was a digit. If so, the “.” would be added to the string and the loop would continue adding digits until it reached a character that was not one. Then, it would print the string as a floating point instead of octal. However, if the current character was a “.” but the character directly after it was NOT a digit, then we would break from the loop, print the octal, and return the index the “.” was at. That way, it would then be sent to the operator function. Each function returns the index it stops at when the cases in them no longer apply. Thus, in the main function, when the loop to iterate throughout the string continues, the index for it now starts at the index the previous function (either opFunc, wordFunc, or numFunc) ended at.