The Unit tests I created combine pretty much all of the token types to ensure that when a complex line of code is put into the stream, it is all tokenized correctly. Specifically, the first one involves a combination of different token types such as multiple keyworks, literals is and identifiers. This test is important because it ensures that the lexer can successfully distinguish between different elements and assign the respective token correctly. The second involves a struct definition and string literals which is essential to test because it tests how the lexer handles nested structures with string literals which can be tricky because they require special handling. The third involves complex arithmetic and logical expressions and combines various operators with parenthesis to add complexity which is important because it is essential that the lexer can differentiate operator tokens as well as read parenthesis which will be important later. These tests are very comprehensive and simulate real code that would be lexed. The negative tests involve unrecognized symbols such as @ and $ which ensures the error handling of the lexer works which is critical to the functionality because its important that invalid input is recognized so that incorrect tokens can be avoided. Some challenges I faced were incrementing the line because I found it di icult to read a newline without incrementing before actually going to the next line, I addressed this simply by -1 the line in the token constructor before returning. Another challenge was reading multicharacter tokens at first, but it was easy to figure out by using a while loop that reads until a token is recognized.