The Lexer uses a File-Read Loop to advance through the file character by character. The Lexer also maintains a list of Active Contexts. A Context is basically a potential Token. A Context consists of a TokenType member and a StateType member, among other things which we'll see later. The TokenType member indicates the type of Token that the Context could potentially emit. The StateType member indicates where we are in the process of deciding whether or not to emit the Token. The Active Contexts represent all the potential Tokens that we're currently deciding whether to emit.

One iteration of the File-Read Loop goes through three stages of the Active Contexts: Initial Contexts, Middle Contexts, and Final Contexts. The Initial Contexts represent the Contexts as they are before anything else happens in the iteration. When the Lexer encounters the Current Character as it advances in the file, it first finds all the ways this character could be part of a Token. These ways are represented by a list of CharType Mappings which can be retrieved for any character. A CharType Mapping is basically a potential Context. A CharType Mapping consists of a TokenType member, a StateType member, and a Position in the Token where the character would appear.

For example, there are two tokens that contain the '{' character: { (TokenType.LCURLY) and {{ (TokenType.DBLLCURLY). So CharType.LCURLY maps to:

1. TokenType.LCURLY, Position 0, StateType.COMPLETE
2. TokenType.DBLLCURLY, Position 0, StateType.CONTINUE
3. TokenType.DBLLCURLY, Position 1, StateType.COMPLETE

A CharType Mapping can potentially be added as a new Context if its Position is 0 and its TokenType member *does not* already appear in the Active Contexts. This is known as a Potential Add. Or a CharType Mapping can potentially form a path for a Context that *does* already appear in the Active Contexts to change its Position and StateType, or just its Position. This is known as a Potential Path.

If we are encountering the '{' character as the first character in the file, then Initial Contexts will be empty. As such, only Potential Adds are possible, which means only CharType Mappings with Position 0 are valid candidates. So we can eliminate CharType Mapping 3 above.

This leaves us with CharType Mappings 1 and 2. Since a Context with StateType COMPLETE should trigger a Token to be emitted with the Context's TokenType, if we add CharType Mappings 1 and 2 as Contexts with their StateType members unchanged, CharType Mapping 1 will emit a Token when it might not be appropriate, e.g. when the next character is a '{' as well, so the Token should actually be DBLLCURLY.

Since the Lexer does not use lookaheads, we introduce StateType.SUSPEND as an alternative to COMPLETE. It implies that we are waiting for other possibilities of what TokenType a CharType could be part of to be eliminated (or not) by future CharTypes before we commit to (i.e. emit a Token with) the "suspended" Context's TokenType.

So given CharType Mappings 1 and 2, we need a way to decide when we translate CharType Mapping 1 into a Context, whether it should have StateType SUSPEND. In order to make this decision, we introduce the concept of overshadowing. Intuitively, CharType Mapping 2 "overshadows" CharType Mapping 1 because they have the same Position, but CharType Mapping 2's TokenType has a longer Completed Value String (i.e. a DBLLCURLY Token is longer than an LCURLY Token). I am currently trying to find a systematic way to implement this concept of overshadowing in code.