To design the lexer some core goals should be met. First, the lexer must be able to tokenize simple F# programs. Secondly, it should provide full functionality to the parser.

It will be helpful for the parser and lexer to have a well-communicated base design. This design should communicate features such as types. For example, should each type be a token of its specific type (e.g. int for “int”), a token of type with a string attribute (e.g. type with attribute “int” for “int”), or even a subtype of type in which it can adopt various types (e.g. type being an element of token and int being an element of type)? Good communication of the modules’ design will also allow a seamless transition of the parser’s test cases to lexer outputs.

Before the start of developing the lexer, it is important to plan the character combinations all tokens can take. This initial plan must also include precedence of which tokens match (e.g. “if” matching the start of an if statement before attempting to recognize it as a variable). The lexer will serve best implementing the largest matched case (e.g. <= not matching < and =).

As an initial plan, the lexer should first accommodate all constants such as numbers and strings. It would then be practical to implement expressional operators with urgency on arithmetic, Boolean, and bitwise operators. Next, to complement the expressions, let and variables should be implemented as to be able to test assign expressions. Caution will be used when implementing variables as to not stop variable names incorrectly matching with operators such as “let”. Next types will be implemented. After this lambda to functions should be implemented as to complement the run-time features later in the model. These functions would require the addition of brackets. If time allows it remaining features of F# such as matching, list functionality, and standard library functions like “List.Map”.

A good test bench should be designed for the lexer. Not only should this test bench accommodate for an incremental range of features as the design goes on, though, it would also benefit from covering multiple features used with each other.