Compilers Progress Report

Joshua Stephenson-Losey, Keefer Sands, Sean Jungst, Brendan Tracey

Scanner

1. General Description of a Scanner

A Scanner (or Lexer) takes a sequence of characters from an input and converts them into a value and a token. The value is simply the printed characters in the object, and the token is separated into different types that will determine its functionality later down the line. It is important to note that the scanner does not care about correct syntax and order. It's only job is to properly convert everything passed to it is set to the right value and has the correct token type. The parser will handle all of that once everything has been broken down into tokens. The purpose of this is so that the variable names can be properly stored and so that the functionality of the code can be analyzed and consistent.

2. Scanner Implementation/Methods

We used java as our language of choice, seeing as everyone in the group has done most of our projects in Java. It is supported by ANTLR and has pretty clear pros and cons. For resources, we spent a lot of time using 'The Definitive ANTLR 4 reference' online for how to use the ANTLR engine to generate the proper lexer wew needed. First we started by defining all the possible characters as fragments so that it would be easier to define the tokens. For our main categories of fragments we chose, Digit, Lowercase,

Uppercase, Letter and Char. We chose these as they cover everything all possible types while also being easy enough to reference for our token types. We used the provided tokens, Keyword, Identifier, Int literal, Float Literal, String Literal, Comment, Operator, Whitespace, Newline. Using the fragment definitions we defined all the possible combinations that would qualify for each token. In the end all the tokens had an airtight list of what input would produce what token. Our output list was generated properly and successfully converted everything into its corresponding token.

3. Challenges and Obstacles

Getting ANTLR set up on our machines for coordination proved more challenging and time consuming then we anticipated, but once the Classpath and folder storage location had been found, it worked reasonably. Once the ANTLR engine was working properly and grammars could be generated, it took us a bit to convert everything to the correct input output format. Once these criteria had been met, we moved on to make the conversion algorithm. When we first began we found it extremely tedious to list every single individual character for each token possible input so we used fragments to cut down on repetitive typing for token definitions. This definitely helped get rid of redundancies in our code and significantly cut down on time. We split the fragments into Digit, Uppercase, Lowercase, Letter, and Char. By using the built in range functions we were able to reduce it to about 100 characters total. Using these fragments we were able to quickly reference large ranges of characters and repeatedly reference them from both other fragment definitions and the token definition. Other than that once we had configured everything to get consistent outputs, it was pretty straightforward.