Michael Zipp
Jack Morgan
Maxwell McRae
CS 3240
Project Phase 1

Scanner Generator

To create our scanner generator, we first started with looking at how to implement Regex. Creating regular expressions from scratch can be a tricky business. Some of the harder parts were deciding how we would handle things like precedence and parentheses. We did some research and eventually found a suitable method to code our operators by.

The operators became trivial once we decided how we would code our project. Instead of trying to blend them into each and every state machine, there were ways to keep them separate from each other. By creating all our identifiers to lexical specifications first, we could then manipulate those small DFA's in order to create larger, more complex DFA's that recognized things like $IDENTIFIER or $FLOAT. Each operator could be "attached" to the DFA it was modifying to create the new complex DFA.

We decided to store our DFA's as custom-coded directed graphs. This was a general enough data type to allow modification by operators while still specific enough to have predictable behavior. We could easily follow different paths on the graphs with general functions that could adapt to different lexicons.

We had to make two assumptions over the course of our project. First, we assumed all input would be in ASCII characters. We used this assumption as a base from which to interpret different ranges of characters and symbols. By comparing the ASCII codes of relevant characters they can easily be mapped to a provided identifier. The second assumption we made was that there would be no dollar signs as inputs. Since dollar signs are used to signify identifiers in the lexical specifications, there would be no method of distinguishing extremely strangely named identifiers from the dollar sign as a character.

One of our biggest problems was correctly interpreting parentheses. Since parentheses come in pairs of two, they have to be matched before they can be interpreted. The subsection of an expression then had to have a DFA created for it separately which was then rejoined to the original expression from which it spawned. To do this, we created group objects that could be interpreted on their own. By treating things as groups they could be nested as well as modified on the whole.

In our source code its possible to see a few unused classes and structures. We had a few different ideas at the beginning about how we would name different classes and write certain functions. We left a lot of the nonfunctional code in the Eclipse project. We simply didn't reference it any more as it became obsolete.