Report - Phase 2: Syntax Analysis

Yongun Seong 2017-19937

For the second phase of the compiler project, I implemented a top-down parser for the SnuPL/2 language.

My approach was to start at the simplest language constructs to build up to the larger, more complex structures. I first implemented the literal expressions, like identifiers, integers, booleans, and strings, and I implemented the top-level module last.

In my implementation, each parser method assumes that the scanner is at the correct position, and performs a series of `Peek()` and `Consume()`s to parse the token stream. If at any point the parser encounters an unexpected condition, it aborts the parse by calling `SetError()` with a descriptive message.

Most methods simply return its own AST node, though a few methods deviate from this pattern for various reasons. For instance, parsing a pre-declared identifier requires looking up the appropriate string in the symbol table to determine if it was indeed declared, or whether it is an erroneous use of an undeclared identifier.

During implementation, after a closer reading of the specification, I noticed that I had made a couple mistakes in my scanner implementation. First, it did not handle `longint` literals. Second, it mistakenly labeled all `||` operators as `&&`. I corrected both of these mitakes during this phase.

At the current stage, the accepts a larger set of programs than what is strictly allowed by the specification. This is because correctly parsing a SnuPL/2 program requires further work, such as semantic analysis and type checking. Thus, the parser takes a few shortcuts, to be filled in in later phases, returning a good-enough AST with the information that it has currently. For example, the parser does not make any attempt to validate the array size expression, and it incorrectly accepts non-numeric or non-constant expressions, instead pretending like all array specifications are open. Similarly, it also assumes that all expressions are integer-typed.