Semantic Analysis and Intermediate Representation for the basicL Language

Conor Smyth, 12452382. All work is my own.

My implementation is an expansion of the first assignment with the grammar slightly modified to allow for the changes. The parser scans through the code doing lexical and syntax analysis and then creates an abstract syntax tree to allow the SemanticCheckVisitor to do some semantic checks on the code.

The SemanticCheckVisitor uses the abstract syntax tree and generates a symbol table used to keep track of variables which have been declared, tracking scope, name, type and argument count for functions.

The semantic checks that occur are as follows:

- The identifier that is declared has not already been declared within the same scope
- The called function has the correct number of parameters
- The value you are assigning to a constant is the correct value
- The function called exists

There is no intermediate code representation implemented.

Syntax Tree & Symbol Table

The syntax tree is structured where it breaks the code down into the relevant nodes for required to parse the tree.
The symbol table is a list that holds all variables in the file and only adds a variable if it has no issues.

Classes

- The STC class holds information about the variables in the language.
- The DataType class is an enum file for the data types in the language.
- The ParserUtilities class holds functions that are used by the various files in the codebase.
- The SemanticCheckerVisitor is the file that does the semantic checks on the abstract syntax tree.

Execution Instructions

You compile the compiler and generate files as follow:

- jjtree BasicLAnalyser.jjt ; Generates .jj file
- javacc BasicLAnalyser.jj ; Generates java classes
- javac *.java ; Compiles all the java files
- java BasicLAnalyser fileToAnalyse ; runs the analyser on the file