

**Compiler Project**

**Author: Alexander Giovannelli**

**Project Start Date: 9. January 2018**

**Report Submission Date: 26. January 2018**

**Goal**

The goal of this assignment was to design and build a simple descent (LL (1)) compiler by hand.

**Software**

With this goal in mind, along with prior knowledge regarding programming languages and design processes, I selected to develop the compiler using the Java programing language in the Eclipse IDE. The program utilized only the JDK System Library (Java SE 8 [1.8.0\_131]) and its corresponding .jar files and no other external libraries.

**Structure**

Based on the provided course development outline, each part of the compiler program was implemented. On program instantiation, the Main class will initialize the classes in the following order: ErrorLogger, SymbolTable, Scan, and Parse. The ErrorLogger is an independent class that requires no interfacing with the remaining classes and is instantiated first as a result. The SymbolTable class requires the passage of the initialized ErrorLogger class, in order to log failed storage of symbols during the parsing process. The Scan class requires the passage of the path, provided by the user as an argument to the Main class. Finally, the parser utilizes exposed functions provided by the passed Scan class to iteratively analyze each token, storing tokens as necessary into the passed SymbolTable class during the process, as well as logging any encountered errors throughout execution using the provided ErrorLogger instantiation. Nested within the Parse class is an initialization and functional calls to the incomplete Translator class, which generates all declarations types from the Parse class into LLVM as a .ll file.

Objects and enumerations used by the previously discussed classes are as follows: Symbol, Token, and TokenType. The TokenType enumeration is mainly utilized to classify a portion of the Token object, upon the Scan class reading in values from the source code and tokenizing them. This Token object is then used inside of the Parse class in order to perform operations such as type checking and Symbol Table insertions and modifications, using the Symbol object respectively.

**Build Process**

**Highlights**

An important characteristic to note in regard to the implementation of the compiler program is the addition of all type checking processes into the Parse class, rather than the separation of concerns generating a new Type Check class object. This was decided on design review, where type checking was deemed to tie directly into the parsing process.

On project submission, incomplete code generation exists during the Parse class execution. It presently provides LLVM assembly language equivalents to all declaration statement types (i.e. variables and procedures), as well as allocating and storing data from passed parameters. Statement functionality remains incomplete and is left out as a result.