COMPILER THEORY PROJECT REPORT

Technical Manual and Highlights

Benjamin D. Ward

ECE6083: Compiler Theory/Compiler Laboratory

5 July 2015

This brief guide documents the design structure, setup instructions, and practical usage of the project compiler, along with highlights of special features. The compiler was written in Java using the Eclipse IDE and targets LLVM assembler. The basic concept is that a runnable JAR file — the compiler Java executable — is ran on top of the JVM, reads from a source file containing code written in the project language, and writes the equivalent LLVM assembler code to an output file. This generated LLVM output file can then be read by the Clang LLVM compiler to generate an executable for the target system.

The design structure is based upon these widely-known compiler module concepts: scanner, parser, semantic analyzer, and code-generator. The scanner module reads characters from the source file to construct tokens that are streamed into the parser module. The parser module reads these tokens and checks for syntactic correctness. It also groups together these tokens to create higher-level constructs containing type-checking details that are used to form the resulting AST (abstract-parse-tree) and allow the semantic analyzer module to simultaneously perform type-checking on this AST. In order for the semantic analyzer to type check both symbols and constants contained in expressions and type check arguments in procedure calls, it must keep track of defined variable and procedure definitions using a symbol table. Symbol table definitions are then added and removed based upon the current scope being analyzed which is determined by the parser. In the end, the code generator module walks the syntactically-correct type-checked AST to form its own higher-level constructs which are used to write code in the target language, in this case LLVM assembler, to an output file.