Compiler’s project

The aim of the project is to write a compiler for Decaf. Decaf is a imperative language similar to C . The project involved writing a parser for parsing the source code, using flex and bison, and creating AST and for generating LLVM IR.

Key points and steps:
● Lexical Analyzer using Flex: A lexical analyzer breaks down a program into pairs of tokens and token types .
● We made use of regex for identifying patterns of literals, identifiers, operators, etc.
Regex is a sequence of characters that define a search pattern, mainly for use in pattern matching with strings, or string matching.
● These token pairs were forwarded to the bison parser.
● Next we built parser using Bison: Parsing is the process of matching grammar symbols to elements in the input data, according to the rules of the grammar. The parser obtains a sequence of tokens from the lexical analyzer, and recognizes it's structure in the form of a parse tree.
● We set up the grammar rules as per a standard decaf program, it receives tokens from the flex file which are in same order as they are encountered.
● Bison creates a parse tree according to the input received by it, and if any error is encountered it exits and throws a parse error.
● Apart from the tokens received from flex, some additional tokens are also specified, and also data typesof tokens are distinguished through union tag, which helps us in sending certain specified tokens (int, bool, etc) through flex itself.
● In the process of setting grammar rules for bison we encountered several shift/reduce conflicts which we handled by giving priority to certain rules through leftassoc, nonassoctags.

- Then we created a class hierarchy for different type of non terminals of grammer.
- For terminals enum classes are used.
- Inheritance is done for non-terminals like ASTExpression, ASTLiteral, ASTStatement.
- Nodes are created in bison file.
- Visitor design pattern is used for implementing visitors like print visitor and code generator.