The next phase to implement in your compiler is intermediate code generation. The code will be three-address code represented as quadruples (very much as described in the Dragon book).

Intermediate Code

The three-address code language consists of items of the following forms:

- \( x := y \ \text{binop} \ z \) where binop is one of: \( +, -, *, /, \text{and}, \text{or} \)
- \( x := \text{unop} \ y \) where unop is one of: \( -, \text{not} \)
- \( x := y \)
- \( L: \) where \( L \) is a label
- \( \text{goto} \ L \)
- \( \text{if} \ x \ \text{relop} \ y \ \text{goto} \ L \) where relop is one of \( <, \leq, =, \geq, >, \text{<>} \)
- \( \text{param} \ x \) precedes call operation, passing a parameter. NOTE: these operations should be issued in left-to-right order. The next phase of your compiler, the assembly-code generator, may decide to generate "push" operations in reverse order, but that is machine dependent.
- \( \text{call} \ p \) procedure call, no return value
- \( x := \text{funcall} \ f \) function call, \( x \) is assigned the value returned by \( f \)
- \( \text{return} \) procedure return
- \( \text{funreturn} \ x \) function return, returning \( x \)
- \( x := y[i] \) In this case, \( y[i] \) refers to the \( i \)th element of array \( y \), and is independent of the size of the elements. Assembly-code generation, the next phase, will convert this to a size-dependent array reference.
- \( x[i] := y \)
- \( x := y.b \) This is a reference to the \( b \) field of record \( y \). Note that the \( b \) field of \( y \) may itself be a record, in which case \( x.c \) might appear subsequently in the code. Similarly, if \( y.b \) is an array, \( x[i] \) can appear subsequently.
- \( x.z := y \)

Note that each of the variables \( x, y, \) and \( z \), can correspond to either variables in the source program (in particular, local variables of a procedure, formal parameters of a procedure, or global variables in the program), or temporary variables created by the intermediate code generator.
Representation of quadruples

Quadruples can be represented by a record (or C++/Java object) with four fields: \texttt{op}, \texttt{target}, \texttt{arg1}, and \texttt{arg2}. The target, arg1, and arg2 fields might refer to variables in the source program or to temporary variables created by the intermediate code generator. If the former, then those fields should point to the node in the AST containing the type information, etc. that your parser and type checker created. Otherwise, as your intermediate code generator generates a temporary variable, it should create a node with the necessary type information, which the field in the quadruple will point to.

Each procedure in the program can be represented by an array of quadruples. This array should be contained in a record or object that also points to the node giving type and formal parameter information about the procedure.

Output of intermediate code generator

In order to help debug the intermediate code generator, it should be able to print out the array of quadruples representing each procedure in the source program.