1. Generate 3-address intermediate code for EctoScript. Please turnin cscheckin electronic copy of your whole project in a .tar but turnin paper copy only for those modules involved in the code generation.
    1. Create abstract data type(s) for lists of 3-address instructions. Define integers for each opcode. Implement operations to create a new list with one instruction, and list concatenation.
    2. Define a data type for "memory address". A memory address is a <region,offset> pair, where region is one of GLOBAL, CLASS, PARAMETER, or LOCAL. Offsets start with 0 in each region. Need global inserted / printed
    3. Write out "declarations" (pseudocode instructions) for classes and methods. Globals (functions and and variables) should be placed inside a "class foo", where foo was the base name of the source file.
    4. For each variable in each symbol table, assign it a memory address. Compute offsets assuming everything requires 4 bytes.
    5. Compute a synthesized attribute "location" for every expression. Allocate a "temporary variable" out of the LOCAL region for each value computed by an operator or method invocation. Huh? Each expression = each gen_tac expression, visitor design pattern equivalent? *temp_var = tac 'result'
    6. Compute a synthesized attribute "code" that builds a link list of 3-address instructions, and a memory address for each expression. See same ?'s as 5
    7. Output from this phase should consist of a file containing intermediate code instructions. If the input was foo.g0, the output file should be named foo.gic ("godiva intermediate code").
2. Notes
    1. your executable should still be named "ec"
    2. your program should accept and process an arbitrary number of source filenames on the command line
    3. take in files with .as extensions and write out corresponding intermediate code in files with a .ic (intermediate code) extension.
    4. write out the name of the file to standard out when you open it
    5. do NOT write out the tree, or other debugging information, by default; you may add command line options to print that info if you

want.

3. error messages should be written to *standard error* not stdout
4. if ANY file has a lexical error, your process exit status should return -1, for a syntax error -2, for a semantic error -3, and for no errors, return 0.