1. Symbol tables
    1. Build a symbol table data type that you can instantiate for each scope: packages, classes, methods.
    2. For each symbol, you should enter a symbol table entry, a struct consisting of enough information to support the semantic analysis. Typically this will include symbol name, reference to which scope (parent symbol table) the symbol is defined in, declared data type, and...
    3. any auxiliary flags, e.g. const. More?
2. Variable declarations
    1. You must emit an error if a variable is redeclared in the same scope.
    2. You must detect and report an error for undeclared variables. * EctoScript is allowed to require variable declarations to occur prior to their use in a given source file.
    3. You must emit an error if a const variable is assigned other than in its initializer.
3. Types
    1. EctoScript supports the standard built-in types, and classes. Built-in types include int, void, Boolean, String, Number, uint, Null, Object + user-defined types
4. Type Checking
    1. EctoScript supports only the "strict" subset, in which types are required and checked everywhere at compile-time. You may choose to support the "standard dialect", but if so you will have to implement runtime type checking. The "strict" mode entails:
        1. Check that function call signatures match
        2. Report expressions with incompatible types very basic types done
        3. +++ Report duplicate definition conflicts
        4. +++ Report unbound references
        5. +++ Unfound packages
        6. *** Report dynamic addition of properties on sealed objects
        7. *** Report writing to const variables
        8. *** Report code that deletes fixed properties
5. Functions - Functions don't have to be methods (inside classes) in ActionScript. Think: like C++. Method lookahead in classes
    1. Die if func return type mismatch
    2. Die if type defined for constructor
6. Classes - Enjoy the classes in your object-oriented language. Inheritance is not required and would be "extra credit". Declarations Instantiation
7. Packages - it is pretty hard to get away from packages in Java-based languages. But there are too many cans of worms here.
    1. You should implement user-defined packages that contain classes, functions, variables and constants. What we said in class was: an import is never really a whole package, it is always one class (one .as file) from within a "package" where a package is just a directory structure. Imports as package, should be class subtree? One class only, really?
    2. Make sure class name matches
8. Built-in Functions! - Without importing, we will implement the following "built-in" functions:
    1. `trace(s)` - which is equivalent to printf("%s\n" ,s) and
    2. `String read()` - which is somewhat similar to gets(), returning one line of text input.
9. Notes:
    1. your executable should be named "ec"
    2. your program should accept and process an arbitrary number of source filenames on the command line
    3. write out the name of the file to standard out when you open it cmd line and import

4. 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.
5. error messages should be written to *standard error* not stdout
6. 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.

Tests

- classnam.as
- constret.as
- fib.as
- missparam.as
- missreturn.as
- mustreturn.as
- param.as
- synerr.as
- typeck.as
- ufib.as
- undeclare.as
- vardecl.as