Scoping issues

- How do we implement dynamic scopes for functions, where call sequence identifies which scopes a function can access?
- How do we implement scopes for nested function definitions, where lexical structure identifies which scopes a function can access?
- In both cases, the function either needs access to stack frames beyond its own (to see their local variables) or we need an alternative way to store/access variables

Dynamic scopes: stack based

- One way to support access to dynamically scoped variables is to give each of them their own stack
- Whenever a new dynamically scoped variable of a specific name (e.g. X) is declared, its space is pushed on a stack that is just for variables named X
- When that variable's scope ends, it is popped off the stack
- Whenever a function refers to X, it uses whichever version of X is currently on top of stack
- We would need to maintain a list of stacks, somehow indexed by the name of the variables it supports

Dynamic scope: frame searches

- Another approach would be to give each stack frame a pointer to the frame from its caller
- Each frame may include a llist of variables defined in it, and their offsets within the frame
- Scan back through the call chain of frames, until you find the first (most recent) one that contains a variable of the name you’re looking for

Dynamic scope: environment lists

- Another possibility is to add extra (hidden) parameters with each function call, containing the names/values (or references) for each currently-active variable
- The callee simply uses the value (or reference) for the one it wants
- If the callee in turn calls another function, it passes an environment list that has been updated to reflect any declarations or changes made by the callee
- (this is the lisp approach)

Nested function definitions

- If one function declaration can be nested within another (so we have functions that are local to other functions) then the scoping rules need to address which of the parent function's variables are "in scope" for the child
- Similar to our dynamic scoping concerns, the child needs access to the stack frame for the parent, which in turn needs access to the stack frame for its parent, etc

  Solution could be handled much like our frame searching approach for dynamic scoping, but if the nesting is a lexical scope the compiler should be able to compute correct offsets for referenced variables