DATA STRUCTURES

- Each scope will have its own static symbol table which describes all the identifiers that are defined in the scope. Each identifier has an entry in the table:
- All entries need: index into identifier table, type, and an indication of the complexity of the object that the identifier will representing.
  - Complex structure (e.g. arrays, structs) will need more information.
  - Functions also need fields for list of parameters, scope (the function's symbol table), and body to evaluate
- The parser keeps a scope stack which stores active scopes during parsing.
  - Each entry in the scope stack is a references to the symbol table of an active scope (i.e. a scope that is in the process of being parsed)
  - Bottom entry is static scope.

PARSING

- When a function declaration is being parsed:
  - an entry for that variable is created in the symbol table topmost in the scope stack (because the function is local to current scope)
  - A new symbol table for that function is created and a reference to it stored in the scope field of the function's entry. This symbol table is the function's scope.
  - A reference to this scope is also pushed on the scope stack
  - Parameter declarations are parsed:
    - For each new parameter, an entry is added to the function's scope.
    - A list of parameters is kept in the symbol table entry for the function being parsed.
  - The function's body in parsed: see other bullets for explanations of how parser deals with identifiers it encounters.
  - When the body has been parsed, add reference to body AST in entry for function in symbol table. Note that ASTs for declarations of locally defined identifiers will have been removed from the AST of the function's body because they have been translated into symbol table actions and are not needed anymore.
- When a variable declaration is parsed, an entry for that variable is created in the symbol table topmost in the scope stack, i.e. the scope being currently parsed.

- For each identifier encountered not in a declaration, search scope stack top to bottom for first reference to that identifer. Add that reference (scope + index of entry in scope) to identifier entry in AST.

EVALUATION OF FUNCTION CALLS

- Note that because most languages allow recursive functions, at run time there can be multiple instances of the same function parameters and local variable: one set each time the function is called.
- Need activation stack: each record keeps
  - Reference to active scope
  - Symbol table for the variable (non-function) entries of that scope's symbol table.

scope	value
	value

- Static scope symbol table always on activation stack
- Evaluating fn_call

```
public Object visit(ASTfn_call node, Object data)
    // Evaluate list of parameters → list of values
    // If not done already by semantic analysis, check their
       correctness
    // Push activation record on activation stack,
    // Copy parameter values into it
    // Try: Evaluate function body – this may change the values of
       some of the entries in the activation record, as well as
       other entries for the activation records of some other
       functions, including static scope.
    // Catch: return value passed in exception needs to be stored
       in a local variable.
    // Finally: Pop activation record off activation stack
    // Finally: Return return value
```

Alternate Implementation

- Replace global activation stack by multiple activation stacks, one for each symbol table entry.
  - Upon entry, push values of parameters on parameter activation stacks, and push a placeholder on stacks of local variables
  - Before exit, pop top item off all the stacks of all the variables in the scope's symbol table.