OO scope resolution

- object oriented languages typically support standard lexical scoping rules, but also incorporate scoping based on the class heirarchy
- within a method, a reference to a variable could refer to one that exists in one of the current lexical scopes, a field that is defined in the current class, or a field that is defined in one of the ancestor classes
- the scope resolution mechanism must identify and access the correct item

Object records

- just as activation records encapsulated the information associated with a function call, object records will encapsulate the information associated with an individual object
- OR must include a means to access the fields associated with that object (including those inherited from ancestor classes)
- OR must also include a means to access the methods associated with that class (again, including inherited, and typically distinguishing between overridden and inherited)

Multiple inheritance

- if a language allows inheritance from multiple different parent classes, then name clashes are possible (e.g. both parents define a method named print)
- need to support the language clash-resolution rules
- might be a default ordering (e.g. In event of clash resolve in the order inherited, e.g. “public A, B, C” pick A first)
- might mandate programmer explicitly resolve clashes (e.g. A::print(), B::print(), etc)

And yet more tables...

- linked list of symbol tables (from current to global scope)
- linked list of activation records (current functions)
- table of object records for all in-scope objects
- linked list of tables for class heirarchy, each table with pointers to the methods defined/overridden in the class, plus a pointer to the parent class table
- might actually be a tree or DAG if multiple inheritance permitted (need pointers to tables of each parent)

Method access

- compiler needs to insert code to ensure correct method is invoked at each call
- might use an <ancestor,offset> format: follow the chain of table pointers to the correct ancestor, then use offset to access correct method in that ancestor's table
- closed class structure: correct method can be identified at compile time
- open class structure: run-time identification of method needed (e.g. dynamic dispatch)

Possible approach

- one table for each class
  - pointer to each method defined/overridden by this class
  - pointer to each parent class table
- one table for each object
  - direct pointer to each of the methods it uses by default
  - entries for each field (including inherited)
  - pointer to the table for its class