Symbol Tables --

- Stand alone symbol tables
- Part of AST?

Name and Attribute "database" (Dictionary?)
- basis for a lot of semantic information
- "interface"
  - create (st)
  - enter (st, name, attributes)
  - STE = lookup (st, name)
  - destroy (st)

Implementations
- Linked list, array (unordered)
- ordered list
- binary search tree
- hash table
- C++ STL map ...
Block Structured Symbol Tables

Languages: Algol, Pascal, Modula-2, ATL/1

Program A;
  type Y = integer;

  Procedure P (X : Y);
  Var Y : Boolean;
  Begin
    ....
  End P;

begin
  ....
End A.

Scope 1: Y is type integer
Scope 2: Y is variable of type boolean
Procedure P’s declaration? Y a type or a variable?
(Most languages consider it to be a type here.)
Issues in block structured symbol tables

Scope: multiple, enclosed scopes. (Depth max?)

Current Scope, What causes scope

- procedure
- functions
- structs / records
- { .... } in C
- Explicit namespace constructs

Identifier Search Rules

```plaintext
scope <-- current scope
while scope is legal do
    x = lookup (scope, name)
    if (name is found) break;
scope <-- next scope out
```
Implementation for ATL

Global table: chaining declarations
One table (tree) for each level

-1: "Global"
0: Program
1: procedure level 1
2: procedure level 2

Maintain "current_level" variable

Interface:
Enter (level, id, name, attributes)
ste = Lookup (level, id)

New_Scope(), End_Scope() -- New elements

ste = Find_Id (ID)
   find ID using scope search rules
   using "current_level" as start (global variable?)
Extra symbol tables

Type rec = Record
    a, b : real;
    c, d : integer;
    rec : char;
end;

Symbol table as part of record description
Import/Export in Symbol Tables

Ada packages, C++ namespaces, Modula 2 modules, ATL/2, ...

- Import into current scope?
- How to get imported symbols?
- Separate compilation?
  - Representation
  - Importing/Exporting only part of the names?

Altered Search Rules
- With statement (pascal, modula-2)