Names and Entities

Entity: An object in the program, such as a class, method, field, variable etc.

Name: String of characters used to refer to some entity in the program.

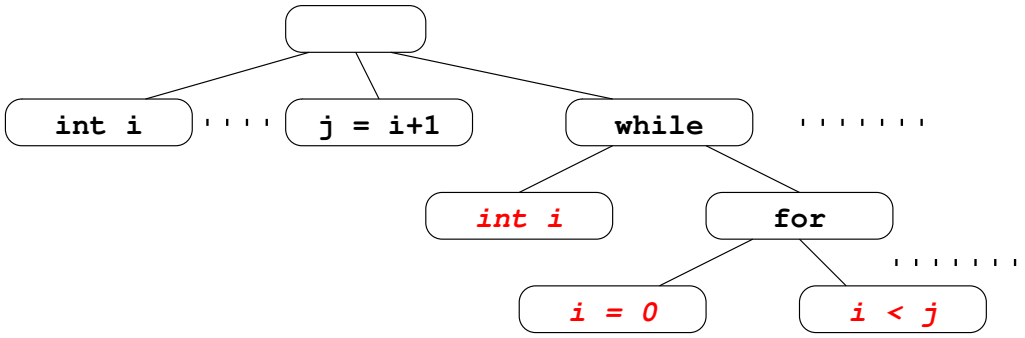
Examples:

```
class List {
  List next;
  int length() {
    if (this == null)
      return 0;
    else
      return 1 + next.length();
  }
}
```

```
int i;
:
j = i+1;
while (!  found) {
  int i;
  for(i=0; i<j; j++)
  :
```


Scopes

Scope of an entity is the fragment of the program where it may be legally manipulated.



Scope rules

- In Pascal, two entities can have the same name only if they are in different scopes.
- In Java, two entities in the same scope can have the same name, provided:
  1. they are of different *kinds*: classes, methods, fields, or
  2. they are methods with different parameter types.

Associating Entities with Names

Given some (unknown) entity with name `x` and kind `k`, which entity does this represent?

1. Determine all entities that can potentially correspond to `x`:
   Filter away entities not named `x`.
2. Among all entities named `x`, choose those of kind `k`.
3. Among all entities named `x` of kind `k`, choose those in the current scope.
4. Among all entities named `x` in kind `k` in the current scope, choose the one with the appropriate parameter types.

Step 4 is applied only if `k` indicates a method.

Names

- For a character string `x`, give it a unique identifier.
- Every occurrence of string `x` is associated with the same identifier.
- **Name Table**: Associates unique identifiers with names.

Implementation: Associates unique pointer with each name.
Functions provided:

- Create new name table
- Check table for presence of string
- Insert string in table if not already present
- *Delete string from table.*

Data Structure: Dictionary, such as Hash Table.

Differentiating Entities

Since two different entities may have same name, we have to

1. Maintain a separate name table for each scope in the program.
   Give a name, determine which entity it represents:

   Search name table of current scope; upon failure search name table of immediately enclosing scope, and so on until the name is found (success) or the root of the scope tree is reached.

2. Maintain an *entity* table that associates, with each name, all entities with that name in the currently active scope(s).
   Given a name, to determine the corresponding entity involves a table look-up.

Scope stacks

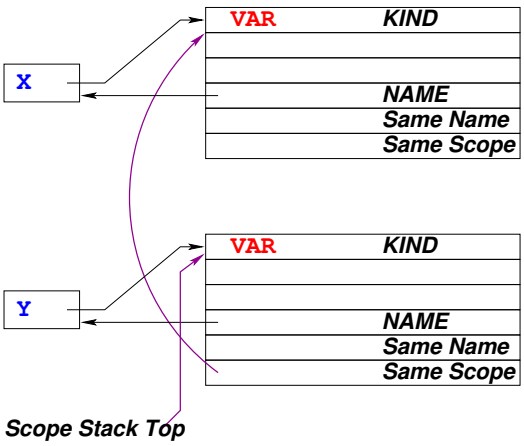
- Entities with same name are chained together, as a LIFO list.
  Top most (first) element of the list is the entity visible from the current scope.
- Entities declared in same scope are chained together;
  By keeping the pointers to scope threads themselves on a stack, we can "return" to old scope at the end of a block.

Entity Table

1. *find_entity*: Given a name and *kind*, find a matcing entity in the nearest enclosing scope.
   Return a pointer to the entity as well as a flag indicating whether or not the entity was found in the current scope itself.
2. *create_entity*: Given a name and kind, create a new entity in the current scope.
3. *enter_scope*: "Create" a new scope.
4. *leave_scope*: "return" from current scope to enclosing scope at end of block.

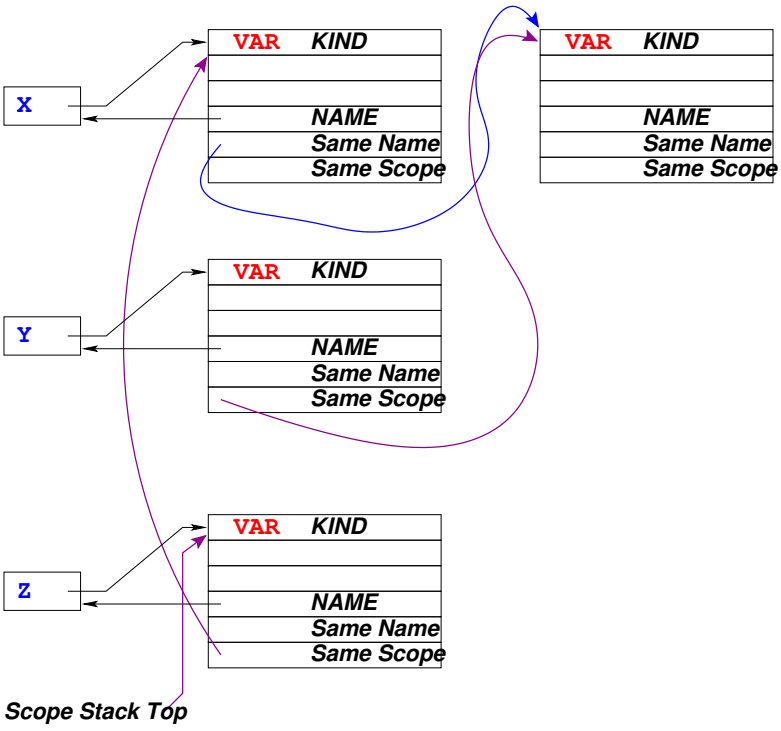
Entity Table

```
int X, Y;
{
  int X, Z;
```



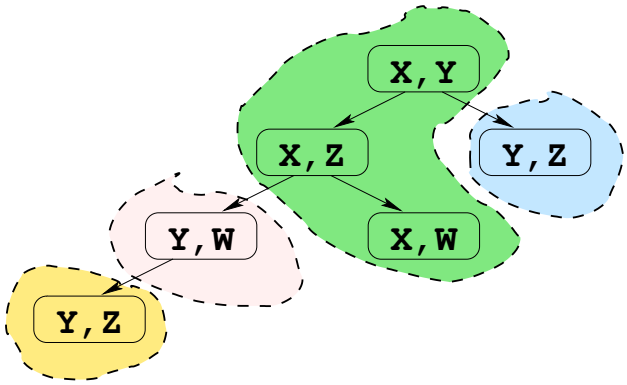
Entity Table

```
int X, Y;
{
  int X, Z;
```

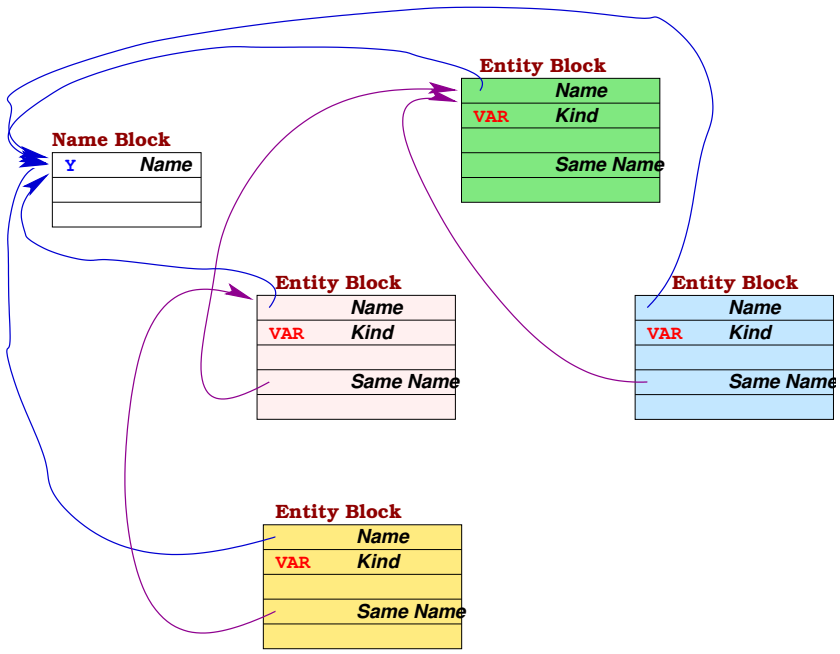


Scopes in Block Structured Languages

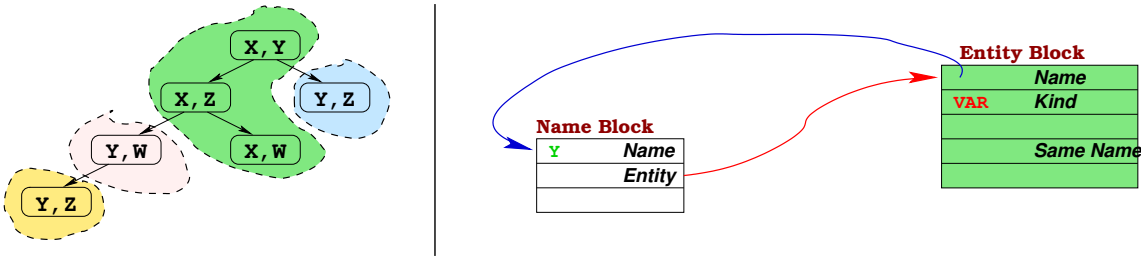
```
int X, Y;
{
  int X, Z;
  {
    int Y, W;
    {
      int Y, Z;
    }
  }
  {
    int X, W;
  }
}
{
  int Y, Z;
}
```



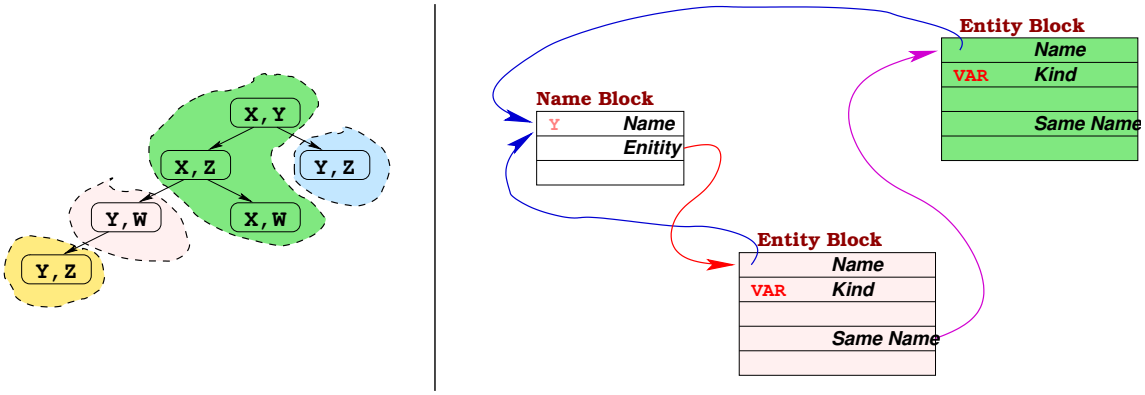
Scopes and Symbol Table



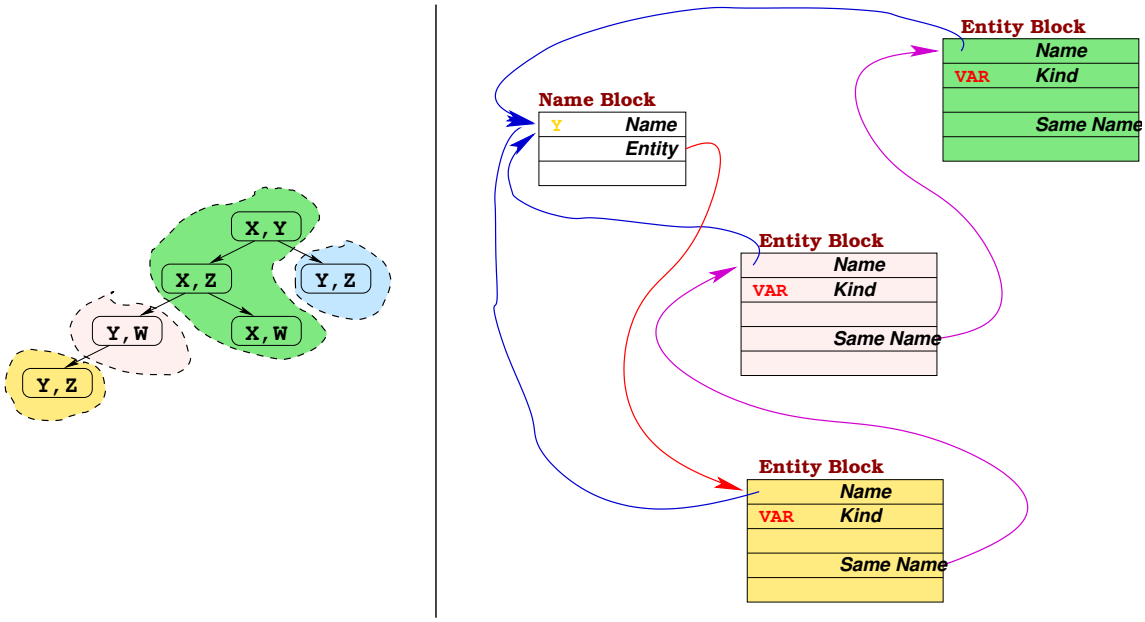
Scopes and Symbol Table



Scopes and Symbol Table



Scopes and Symbol Table



Scopes and Symbol Table