**Scopes**

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

```
int i
j = i+1
while
int i
for

i = 0
i < j
```

**Scope rules**

In Pascal, two entities in different scopes can have the same name.

In Decaf, 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,

---

**Compilation**

```
// Source Program

// Lexical Analysis

// Parsing

// Semantic Analysis
// (e.g., type checking)

// Intermediate code Generation

// Code Optimization

// Final code generation

// Target Program
```

---

**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:**

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

```java
int i;
:
:
:
:
:
```

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

1. Array of strings (character arrays), sequential search using strcmp.
   Wasted space, large search time.
2. Array of pointers to strings, sequential search using strcmp.
   Large search time.
Need a method to quickly narrow down search.

Hash Tables

Data structure that quickly prunes search space by looking at one or more symbols in the input.

Hash function: Compute an integer value (small) from the input string.

Hash table: Array of pointers to lists of strings.
i-th array element points to all strings s such that hash(s) = i.

Collision: When a hash table element points to more than one string.

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.

Unique 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.
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.

Hash Tables

An Implementation of Hash Tables

Algorithm \texttt{search(Hash Table, string)} { 
  /* Hash Table: Array of pointers to list of strings. */
  hashvalue = hash(string);
  search(string in list(Hash Table[hashvalue], string));
}

Algorithm \texttt{insert(Hash Table, string)} { 
  hashvalue = hash(string);
  insert(string into list(Hash Table[hashvalue], string));
}
Entity Table

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

1. find_entity: Given a name and kind, find a matching 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.