Handling Declarations

- What does a declaration such as `int x;` do?
    - It introduces a new symbol in the current context.
    - It associates one or more attributes with this symbol: (*name*: `x`, *type*: `int`, *loc*: 3, *val*: 0, …).

Handling Declarations

- What does a declaration such as `int x;` do?
  - It introduces a new symbol in the current context.
  - It associates one or more attributes with this symbol: (*name*: `x`, *type*: `int`, *loc*: 3, *val*: 0, …).
- What action(s) does the parser need to take on encountering a declaration?
  - It needs to insert the attributes of the declared symbol into a dictionary, so that downstream uses of this symbol can look up the information they need from this dictionary.
  - There are no code generation actions associated with parsing a declaration.

Handling Declarations

- What does a declaration such as `int x;` do?
  - It introduces a new symbol in the current context.
  - It associates one or more attributes with this symbol: (*name*: `x`, *type*: `int`, *loc*: 3, *val*: 0, …).
- What action(s) does the parser need to take on encountering a declaration?
  - It needs to insert the attributes of the declared symbol into a dictionary, so that downstream uses of this symbol can look up the information they need from this dictionary.
  - There are no code generation actions associated with parsing a declaration.
- The symbol table is this dictionary of symbol attributes.
  - The primary attribute ("key") is the symbol name.
  - Needs to support insertion, deletion, and lookup operations.
  - Any data structure that supports these operations can be used: linked list, binary search tree, hash table.

Communicating Via The Symbol Table

- The symbol table entries are also used to communicate information among parts of the compiler.
  - Lexer: Installs the symbol name `x` into the symbol table, with an empty set of attributes.
  - Parser: Associates *type* attribute with `x`.
  - Intermediate code generator: Assigns *loc* attribute to `x`.
  - Back-end code generator: Uses *loc* attribute of `x` to generate machine code.
  - AST-based interpreter: Uses symbol record to execute interpreter actions.

- For LiveOak
  - Use a single global symbol table (LO-0).
  - Use one symbol table per method (LO-1, LO-2) or per class (LO-3).
  - Implement using `java.util.Hashtable` or a similar class.

Handling Scope

- Real programming languages have more extensive rules for determining visibility of names at different points of the source text.
    - Typically tree-structured.
    - E.g., C has file scope, parameter scope, function-local scope, and block scope.

```
/*  1 */ int x, y;
/*  2 */ void f(int x, int a) {
/*  3 */   int b = y;
/*  4 */   y = x+a*b;
/*  5 */   if (y < 5) {
/*  6 */     int a = 1;
/*  7 */     y = x+a*b;
/*  8 */   }
/*  9 */   else {
/* 10 */     int a = 2;
/* 11 */     y = x+a*b;
/* 12 */   }
/* 13 */ }
```

Q: How do we manage and organize the symbol table to correctly implement the scope rules of the language?

Option #1: One Symbol Table

- Assuming a hash table implementation with chaining-based collision resolution.

- Key Idea: Add an extra tag field to each hash table entry.
    - Tag identifies the scope in which the identifier was declared.

- Scope entry
    - Increment a global (or class-static) *scope counter*.
- Symbol insertion
    - Insert symbol record at head of chain, using *scope counter* value as tag.
- Symbol lookup
    - Return first true match of name. (Ignore tag.)
- Scope exit
    - Unlink (but don’t delete) all symbol records from hash table buckets whose tag matches the current *scope counter* value.

Option #2: A Stack of Symbol Tables

- Key Idea: Maintain an explicit stack of symbol tables.
- Scope entry
  - Push a new symbol table to TOS.
- Symbol insertion
  - Insert symbol into the symbol table at TOS.
- Symbol lookup
  - Search in symbol tables starting with TOS and going iteratively deeper into the stack until a match is found or the bottom of the stack is reached.
- Scope exit
  - Pop off symbol table at TOS.
  - Unlink (but don’t delete) symbol records from hash table buckets of this symbol table.