Symbol Tables

Use of Symbol Table

Record identifiers and collect information about various attributes.

These attributes provide information about

- its type and its scope, which informs the amount of storage to be allocated
- number and types of arguments to procedures
- return types of procedures.
- parameter passing methods for procedures
- Dimensions and bounds for arrays.

Symbol Table Use

- In languages with global scope, the lexer can enter identifier into a symbol table.
- In block-structured languages with scope, the parser enters identifier names into the symbol table and the semantic analyzer enters corresponding attributes

Symbol Table Use

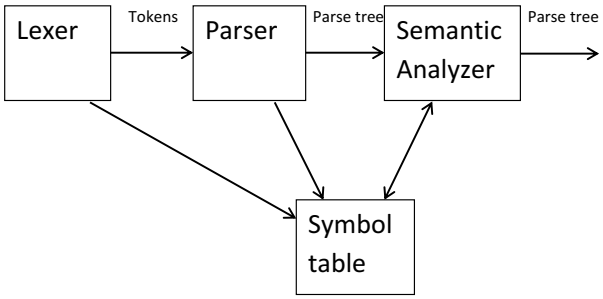
Symbol table information is used by the analysis and synthesis phases

To verify that used identifiers have been defined (declared)

To verify that expressions and assignments are semantically correct – type checking

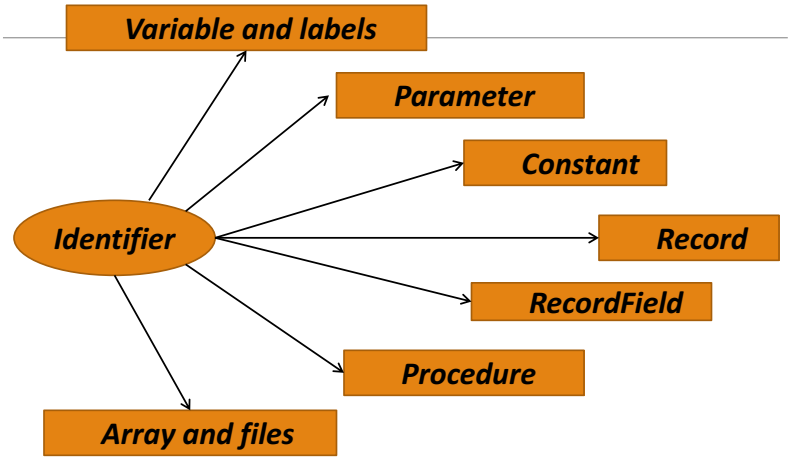
To generate intermediate or target code

Symbol Table Use



Notice that the semantic analysis is typically rolled into the parser and is not necessary a separate component.

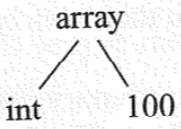
Some Roles of Identifiers



Information Collected

```
int a[100];
```

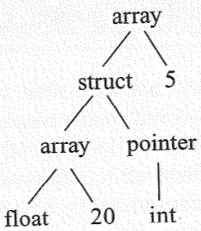
This can be represented by a tree as:



Information Collected

```
struct my {
  float f[20];
  int *pi;
} b[5];
```

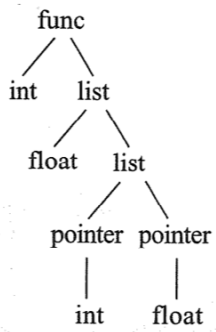
This array of `struct` definition can be represented by a tree as:



Information Collected

```
int func1(float a, int *b, float *c);
```

We can represent this by a tree such as:



Implementation of Symbol Tables

There are many data structures that can be used.

In the end of the day, a hash-table works best.

The key issue to be addressed is how to efficiently manage scope.

Implementation of Symbol Tables

Consider the following fragment.

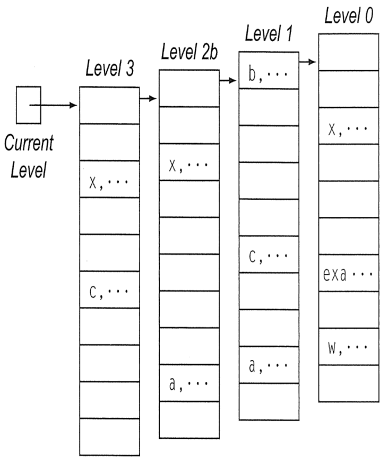
```
static int w;      /* level 0 */
int x;

void example(int a, int b) {
  int c;           /* level 1 */
  {
    int b, z;      /* level 2a */
      ...
  }
  {
    int a, x;      /* level 2b */
      ...
      {
        int c, x; /* level 3 */
        b = a + b + c + w;
      }
  }
}
```

Level	Names
0	w, x, example
1	a, b, c
2a	b, z
2b	a, x
3	c, x

Implementation of Symbol Tables

Nested tables:



Implementation of Symbol Tables

Nested tables require linear search on the order of the depth of the table.

We may wish to use a single table instead.

Most recent entries are at the front.

Two link fields:
- One to next entry.
- One for each identifier within the same scope.

Notice that a hash function may hash more than one identifier into each bucket.

You have two options:
1. A list of lists.
   - The top level list contains the different identifiers
   - The nested lists contain identifiers of the same name but of different scope.
2. A top level list with identifiers of different names intermixed.

Operations on Symbol Tables

In addition to the expected operations:
- `insert(<table>, <item>)`
- `lookUp(<table>, <id>)`