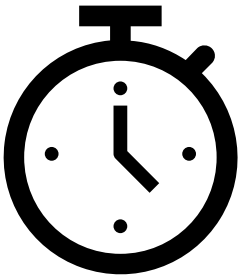
Semantic Analysis & Symbol Table

Zakia Zinat Choudhury
Lecturer
Department of Computer Science & Engineering
University of Rajshahi



1

Example

The SDD a simple declaration D consisting of a basic type T followed by a list L of identifiers.

T can be int or float.

For each identifier on the list, the type is entered into the symbol-table entry for the identifier.

We assume that entering the type for one identifier does not affect the symbol-table entry for any other identifier.

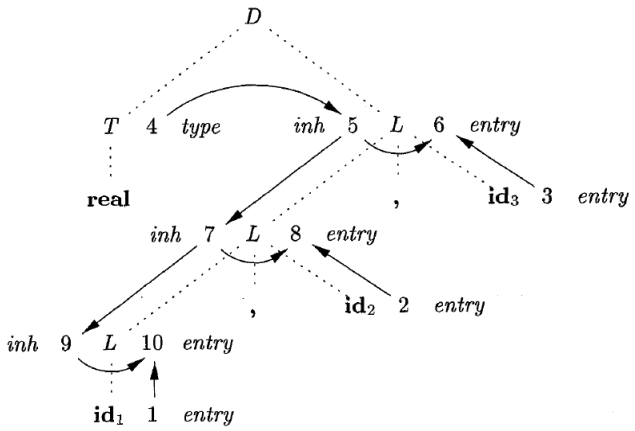
Thus, entries can be updated in any order. This SDD does not check whether an identifier is declared more than once; it can be modified to do so.

	PRODUCTION	SEMANTIC RULES
1)	$D \rightarrow T\ L$	$L.inh = T.type$
2)	$T \rightarrow \mathbf{int}$	$T.type = \text{integer}$
3)	$T \rightarrow \mathbf{float}$	$T.type = \text{float}$
4)	$L \rightarrow L_1\ ,\ \mathbf{id}$	$L_1.inh = L.inh$
		$addType(\mathbf{id}.entry, L.inh)$
5)	$L \rightarrow \mathbf{id}$	$addType(\mathbf{id}.entry, L.inh)$

2

Example

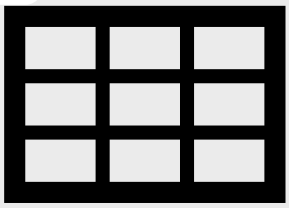
Dependency graph for a declaration **float** $\mathbf{id}_1$, $\mathbf{id}_2$, $\mathbf{id}_3$



3

Symbol Table

- The data structure that is created and maintained by the compilers for information storing regarding the occurrence of various entities like names of variables, functions, objects, classes is known as a **Symbol table**.
- The analysis and synthesis of the compiler uses the Symbol table.
- This information is collected incrementally and used by various phases of the compiler.



4

Components of Symbol Table

The symbol table you will build will have two main components

The *Name Table*

The name table is used to uniquely identify different names that occur in the program.

The *Entity Table*

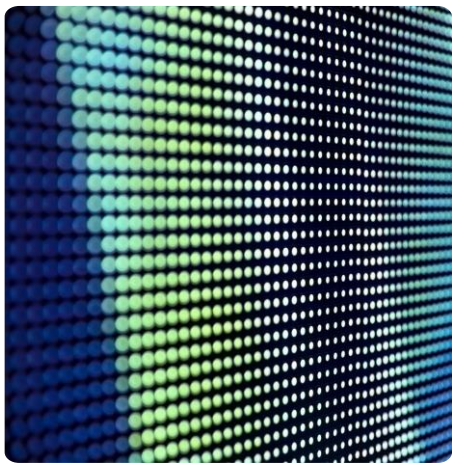
Each entity in the program is represented by a unique entry in the entity table.

5

Purpose of Symbol Table

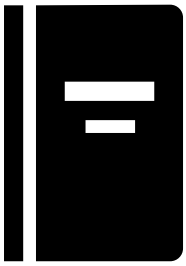
The symbol table used for following purposes

- It is used to store the name of all entities in a structured form at one place.
- It is used to verify if a variable has been declared.
- It is used to determine the scope of a name.
- It is used to implement type checking by verifying assignments and expressions in the source code are semantically correct.



6

Implementation



- If a compiler is to handle a small amount of data, then the symbol table can be implemented as an unordered list, which is easy to code, but it is only suitable for small tables only.
- A symbol table can be implemented in one of the following ways:
  - Linear (sorted or unsorted) list
  - Binary Search Tree
  - Hash table
- Among all, symbol tables are mostly implemented as hash tables, where the source code symbol itself is treated as a key for the hash function and the return value is the information about the symbol.

7

Operations

insert()

- ❑This operation is more frequently used by analysis phase, i.e., the first half of the compiler where tokens are identified, and names are stored in the table.
- ❑This operation is used to add information in the symbol table about unique names occurring in the source code.
- ❑The format or structure in which the names are stored depends upon the compiler in hand.
- ❑An attribute for a symbol in the source code is the information associated with that symbol.
- ❑This information contains the value, state, scope, and type about the symbol.
- ❑The insert() function takes the symbol and its attributes as arguments and stores the information in the symbol table.

For example:

```
int a;
```

should be processed by the compiler as:

```
insert(a, int);
```

8

Operations

lookup()

lookup() operation is used to search a name in the symbol table to determine:

- ❑if the symbol exists in the table.
- ❑if it is declared before it is being used.
- ❑if the name is used in the scope.
- ❑if the symbol is initialized.
- ❑if the symbol declared multiple times.

The format of lookup() function varies according to the programming language.

The basic format should match the following:
lookup(symbol)

This method returns 0 (zero) if the symbol does not exist in the symbol table.
If the symbol exists in the symbol table, it returns its attributes stored in the table.

9

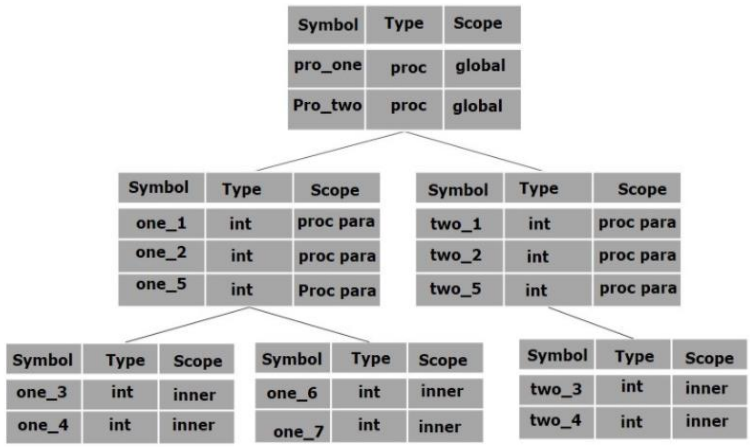
Scope Management

- A compiler maintains two types of symbol tables: a global symbol table which can be accessed by all the procedures and scope symbol tables that are created for each scope in the program.
- To determine the scope of a name, symbol tables are arranged in hierarchical structure as shown in the example below:

10

Example

```
. . .
int value=10;

void pro_one()
   {
   int one_1;
   int one_2;

      {              \
      int one_3;      |_  inner scope 1
      int one_4;      |
      }              /

   int one_5;

      {              \
      int one_6;      |_  inner scope 2
      int one_7;      |
      }              /
   }

void pro_two()
   {
   int two_1;
   int two_2;

      {              \
      int two_3;      |_  inner scope 3
      int two_4;      |
      }              /

   int two_5;
   }
. . .
```



11

Example

- The global symbol table contains names for one global variable (int value) and two procedure names, which should be available to all the child nodes shown above. The names mentioned in the pro_one symbol table (and all its child tables) are not available for pro_two symbols and its child tables.

- This symbol table data structure hierarchy is stored in the semantic analyzer and whenever a name needs to be searched in a symbol table, it is searched using the following algorithm:
  - ❑first a symbol will be searched in the current scope, i.e. current symbol table.
  - ❑if a name is found, then search is completed, else it will be searched in the parent symbol table until,
  - ❑either the name is found, or global symbol table has been searched for the name.

12