

Symbol Table Management

WHO CREATES SYMBOL TABLE?

- ▶ Identifiers and attributes are entered by the analysis phases when processing a definition (declaration) of an identifier.
- ▶ In simple languages with only global variables and implicit declarations:
 - ▶ The scanner can enter an identifier into a symbol table if it is not already there.
- ▶ In block-structured languages with scopes and explicit declarations:
 - ▶ The parser and/or semantic analyzer enter identifiers and corresponding attributes.

USE OF SYMBOL TABLE

- ▶ 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.

Assumptions

For this work, assume we:

- ▶ use static scoping.
- ▶ require that all names be declared before they are used.
- ▶ do not allow multiple declarations of a name in the same scope.
- ▶ do allow the same name to be declared in multiple nested scopes.
 - ▶ but only once per scope.

What Operations do we need ?

Using the previous given assumptions, we will need:

- ▶ look up a name in the current scope only.
 - ▶ to check if it is multiply declared.
- ▶ Look up a name in the current and enclosing scopes
 - ▶ to check for a use of an undeclared name, and
 - ▶ to link a use with the corresponding Symbol-table entry.
- ▶ insert a new name into the symbol table with its attributes.
- ▶ Do what must be done when a new scope is entered.
- ▶ Do What must be done when when a scope is exited.

SYMBOL TABLE DATA STRUCTURES

- ▶ Issues to consider : Operations required
 - ▶ **Insert** - Add symbol to symbol table
 - ▶ **Look Up** - Find symbol in the symbol table (and get its attributes)
- ▶ Insertion is done only once.
- ▶ Look Up is done many times.
- ▶ Need Fast Look Up.
- ▶ The data structure should be designed to allow the compiler to find the record for each name quickly and to store or retrieve data from that record quickly.


LINKED LIST

- ▶ A linear list of records is the easiest way to implement symbol table.
- ▶ The new names are added to the symbol table in the order they arrive.
- ▶ Whenever a new name is to be added it is first searched linearly or sequentially to check if the name is already present in the table or not and if not , it is added accordingly.
- ▶ Time complexity - $O(n)$.
- ▶ Advantage - less space, additions are simple.
- ▶ Disadvantages - higher access time.

Example:

```
int main()
{
    int f;
    {
        int a, b, c;
        int d;
        char v, g;
    }
    int j;
    {
        {
            char v;
            {
                int k;
            }
        }
    }
}
```


UNSORTED LIST



SL No	Name	Class	Datatype	Scope	Line No
1	<i>f</i>	identifier	int	1	3
2	<i>a</i>	identifier	int	2	4
3	<i>b</i>	identifier	int	2	4
4	<i>c</i>	identifier	int	2	4
5	<i>d</i>	identifier	int	2	5
6	<i>v</i>	identifier	char	2	6
7	<i>g</i>	identifier	char	2	6
8	<i>j</i>	identifier	int	1	9
9	<i>v</i>	identifier	char	3	12
10	<i>k</i>	identifier	int	4	14

Hash Table

- ▶ Table of k pointers numbered from *zero* to $k - 1$ that points to the symbol table and a record within the symbol table.
- ▶ To enter a name in to the symbol table we found out the hash value of the name by applying a suitable hash function.
- ▶ The hash function maps the name into an integer between *zero* and $k - 1$ and using this value as an index in the hash table.

$$H_{Key} = \underline{ASCII_{id} \bmod 11}$$

a	b	c	d	e	f	g	h	i	j
97	98	99	100	101	102	103	104	105	106

