Symbol tables and Structure of a Symbol Table : UNIT III

Symbol table: also called identifier or name table is a data structure which assists the compiler in the translation process : inchecking the semantic correctness and in proper code generation. All the names appearing in the source program are entered into the symbol table during lexical and syntactic analysis, and the table is searched everytime a name is encountered in the source text. Each entry in the symbol table is a pair (name, information) where information includes the string of characters, type, location in memory and other attributes. Symbol tables are required for :- (Uses

i) quick insertion of identifier and related information
ii) quick retrieval of identifier and iii) compile-time effici

To achieve compile time efficiency the compiler makes use of symbol tables, hence symbol table interaction consumes a major portion of the processor time during compilation. So efficient symbol table handling methods and data structures are required for both nested and non nested languages.

The structure of a symbol table is towardly given as :

Names	Attributes

contents

Where names stored in the symbol table are;—
variable names, constants, procedure names, function names, litral constants and strings, compiler generated temporaries and labels in the source language.

The information associated with the names or attributes are:- data type, declaring procedures, offset in storage, if structure or record then pointer to the structure table, for parameters whether parameter passing is by value or reference, number and type of arguments passed to the function, base address, dimensions, source and sourceline declared, line referenced, etc. link field for listing in alphabetical order.

Data Structures of Symbol tables: Following data structures are used for symbol tables for non-Block Structured languages: Arrays (linear list), lists (Self Organizing list), Tree-structured representation & Hashing, [Stack Symbol tables, Stack Implemented Tree-structured Symbol tables, Stack Implemented Hash-Structured Symbol tables are the symbol table organizations for Block-Structured languages.

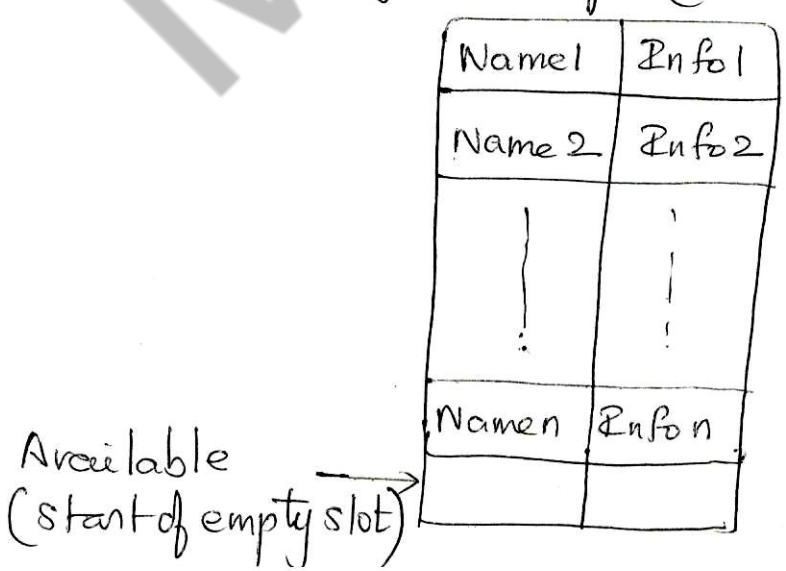
Operations on Symbol tables: The two most common operations are insertion and lookup (retrieval).

Symbol table Organizations for Non-Block-Structured Languages: (2)

A language in which each separately compiled unit is a single module that has no sub modules is called a non-block-structured language. All variables declared in the module are known throughout the module.

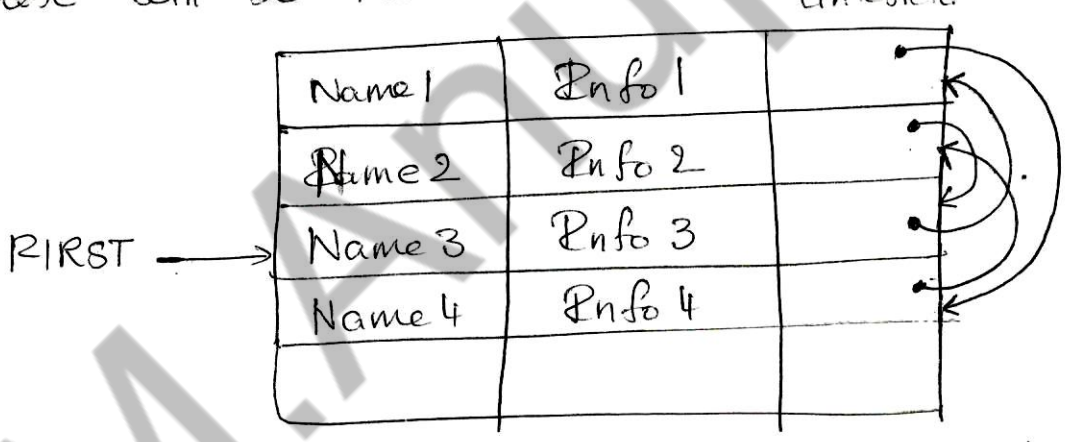
The primary measure used to determine the complexity of a symbol table operation is the average length of search i.e the avg no. of comparisons required to retrieve a record in a particular symbol table organization.

1. Unordered Symbol tables: It is the simplest method in which attribute entries are added to the table in the order in which the variables are declared. This should be used only if the expected size of the table is small since insertion and lookup is directly proportional to the table size. Eg: Arrays (linear list)



2. Ordered Symbol tables: The table is usually lexicographically ordered on variable names. The implementation is using linked list. A link field is added to each record. The two most popular search techniques applied are linear search and binary search. A pointer 'first' is maintained to point to first record of the symbol table.

Eg: List (Self Organizing list) or linked list. When the name is referenced or created it is moved to the front of the list. The most frequently referred names will tend to be in the front of the list hence access time to these will be the least.



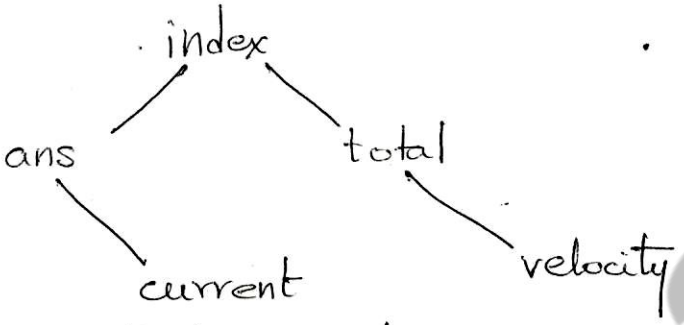
The reference to these names are Name3, Name1, Name4, Name2

3. Tree – Structured Symbol tables: When the scope of information is presented in hierarchial manner then it forms a tree structure representation. To overcome the drawback of sequential representation being slow in searching the desired identifier, a binary tree

is built using lexicographic ordering of tokens. (3)

The typical data structure is. | leftchild | Name | Info | rightchild |

Eg: int index, total, velocity, current, ans.



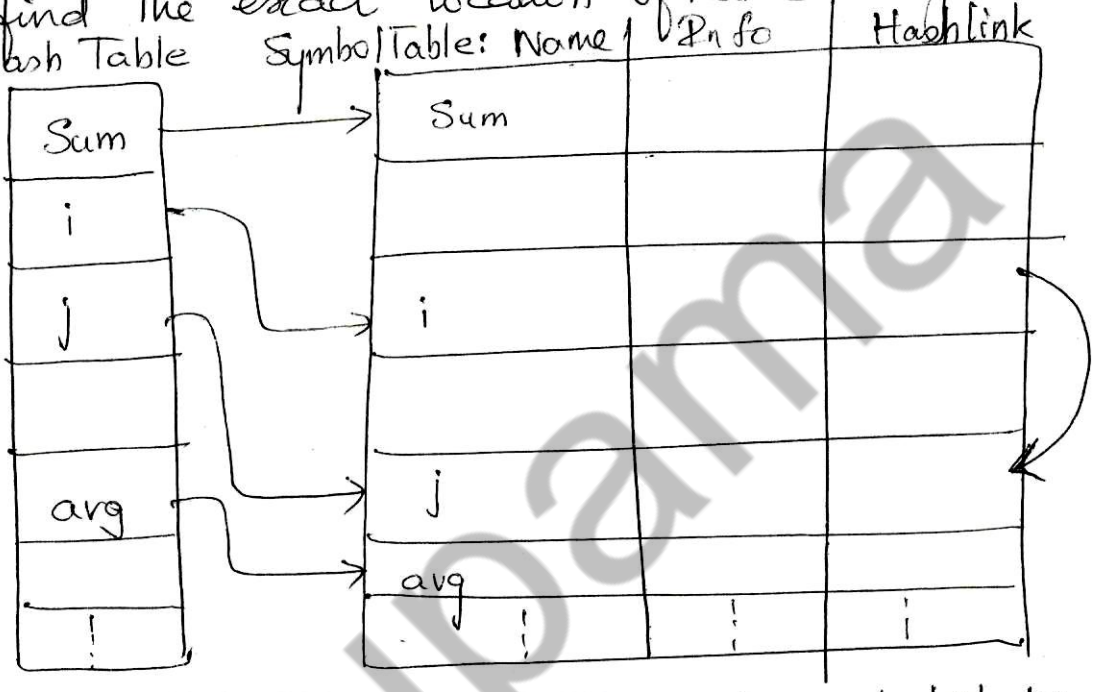
The major problem with binary tree is since insertion always takes place at the leaves, the search tree becomes unbalanced. Hence an ideal search tree would be an AVL tree, and for a very large number of variables, a special m-ary type of tree structure called a trie is used in conjunction with a binary tree.

4. Hash Symbol Tables: Hashing is an important technique used to search records of symbol table and search time is independent of the no. of records in the table. In this scheme two tables are maintained - a hash table and a symbol table.

The hash table consists of k entries from 0 to k-1. These entries are basically pointers to symbol table pointing to the names of symbol table. We use different hash functions like division method, midsquare hashing, folding to determine whether the name is in

symbol table. We use the function h such that h(' will result in any integer between 0 to k-1.' We searc any name by; position = h(name). We use this position to find the exact location of name in symbol table.

Eg:



The hash function should result in uniform distribution of names in symbol table, and minimum no. of collisions Various collision resolution techniques like open addressing chaining and rehashing are availble.

Hashing results in a quick search, but is complicated to implement.

Symbol Table Organization for Block-Structured Languages:

A block structured language is one in which a module can contain nested submodules and each submodule can have its own set of locally declared variables. A variable declared within a module A is accessible throughout that module unless the same variable name is redefined within a submodule of A. The redefinition of a variable holds throughout the scope of the submodule.

Block-structured language concepts:

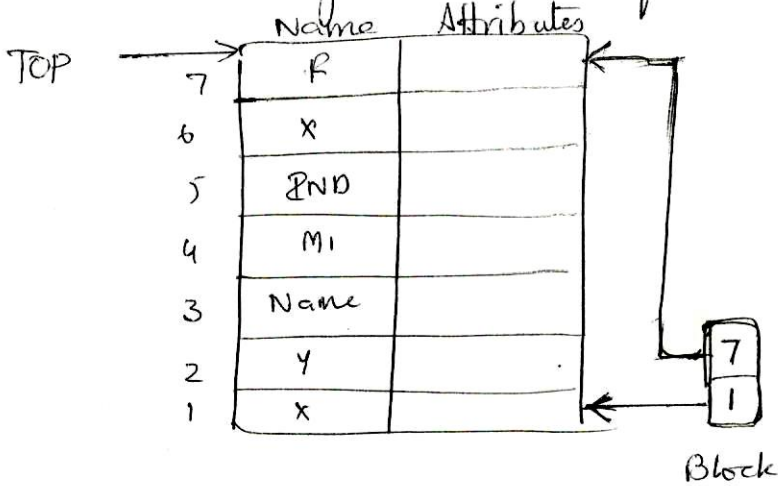
```
1 B Block
        int x, y, STRINGNAME;
        :
    2   M1: P Block (Int XYZ);
            INT X;
            :
            CALL M2(XYZ+1);
        END M1
    3   M2: P Block (Int J);
            :
            B Block;
            :
            END;
        END M2;
        :
        CALL M1(X/4);
        :
        :
END
```

A B Block is a Begin like block and a P Block procedure block. B Block execution is in sequential mann whereas P Block execution is invoked by a call. At execution time these blocks behave quite differently but during compilation both demand similar types of processing Another property of block structured languages is the duplicate use of names. So the search strategy must locate the latest instance of a variable name first.

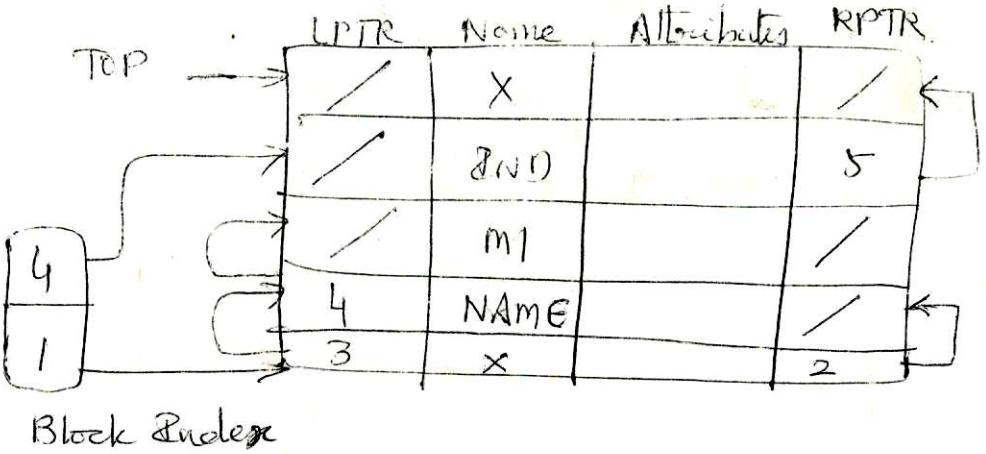
1. Stack Symbol tables: The records containing the variable's attributes are simply stacked as the declarations are encountered. Upon reaching the end of the block all the records for variables declared in the block are removed since these variables cannot be referenced outside the block.
Lookup operation involves linear search while insertion is very simple as we add to the top location in the stack
This organization is similar to the unordered table and should be used only in instances where the Symbol table remains small throughout the compilation of the program.

The set operation for a Stack symbol table generates

a new block index entry at the top of the block index structure. It contains the value of the current top of stack. The reset operation effectively removes all the records in the stack symbol table for the block just compiled. Eg:

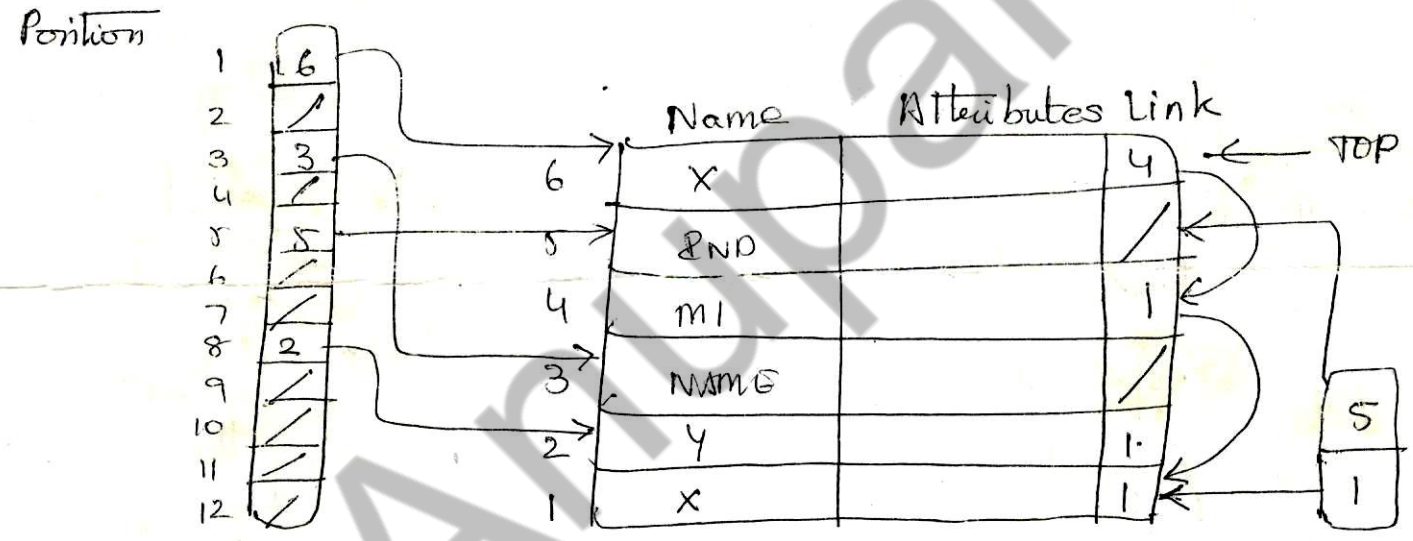


2. Stack Implemented Tree-Structured Symbol Tables: The time to perform table lookup and insertion operations on large tables can be reduced using a tree-structured organization. The problem of deleting records from the table on completion of block compilation is addressed by rebalancing the tree. Another type of organization is where a forest of trees is amenable to the problem of deleting records. Each block is allocated its own tree-structured table. When the compilation of a block is finished, the entire table associated with that block is released. The symbol table is maintained as a stack. Tree-structured symbol tables must be used for a reasonably large number of entries when memory space is at a premium.



3. Stack Implemented Hash-Structured Symbol Tables :-

Here we use a hashing technique which uses chaining to resolve collisions in the hash table.



Hash table Symbol table.

It is an important technique used to search records of symbol table and search time is independent of the no. of records in the table. Hashing results in a quick search though implementation is complicated.