8

SYMBOL TABLE

8.1 INTRODUCTION

Complier requires information about the different names appearing in source program to compile successfully. These names may contain varying amount of information. Some name may contain information upto 100 bytes and some might require storage of 1000 bytes. Hence we require a data structure that can store such varying amount of information effectively and efficiently. We have defined a new data structure for this. We call it a **Symbol table.**

8.1.1 Definition

→ **Symbol table** is data structure that facilitates effective and efficient way of storing information about various names appearing in source program.

8.1.2 Characteristics of Symbol Table

(1) Each entry in symbol table is a pair of name, information.

Name	Information
A	int
B	Real

Fig. 8.1 : Symbol table

(2) Information stored in symbol table is used by different stages of compilation differently.

(3) Possible uses of information
   (1) Semantic interpretation of variables
   (2) Code generation
   (3) Aids in error handling
   (4) Helps in code optimization

Hence symbol stable play vital role in each phase of compilation.

(4) **Primary issues involved in symbol table.**

(a) **Formation of entries :** Deciding what type of format is to be used to store information.

(b) **Method of Accessing :** How the information stored in symbol table will be accessed by different phases. Method of Access may be sequential, or random.

(c) **Storage issues :** This issue address the problem of storage requirements for amount of information that is to be used for particular variable.

(d) **Scope management :** Same name can be used in different blocks and scope. So the possible manner by which scope management is done is also an important issue.

8.2 CONTENTS OF SYMBOL TABLE

Data is stored in the symbol table in form of name, information pair as shown in fig. 8.2.

Name	Information
X	REAL
Y	INT
Z	CHAR

Fig. 8.2 : Symbol Table

Operations which can be applied or Symbol Table :-

— Symbol table must allow **searching** particular name in table
— It allow **editing** information for a particular name
— It must allow **deleting** particular enteries
— Allow **adding** more information about name

8.2.1 Contents in symbol table

Following are the list of items that can be stored in symbol table

(1) Variable names
(2) Constants
(3) Procedure names
(4) Function names
(5) Literal constants and strings
(6) Complier generated temporaries
(7) Labels.

Complier may use single symbol table to store all information or may use seperate symbol tables. Consider a statement

```
int   a = 5, b ;
      b = a + 3 ;
```

These two statements can be stored in single symbol table as

Name	Type/Information
a	Var
b	Var
5	Const
3	Const

But if complier support different symbol table for different type of information then these two statements can be represented as.

Name	Information
a	Var
b	Var

(*a*) Symbol table for variables

Name	Information
5	const
3	const

(*b*) Symbol table for constants

Similarly we can have seperate symbol table for seperate data type. Consider statements

```
int     a, b = 7 ;
float   c, d ;
```

Name	Information
a	Var,Int
b	Var,Int

(*a*) Symbol table for int

Name	Information
c	Var, Real
d	Var, Real

(*b*) Symbol table for real

Name	Information
7	const

(*c*) Symbol table for const

The different information that can be stored in symbol table is listed below.

(1) Strings of character representing names

(2) If the same name is used in different block, then each name must be paired up with block name as well.

(3) Attributes of name, which include
   (*a*) type
   (*b*) current value

(4) Attribute type to indicate whether attribute name specified is
   (*a*) Label
   (*b*) Formal parameter
   (*c*) Actual parameter
   (*d*) Array name
   (*e*) Function name, etc

(5) Parameters such as dimensions of array including lower bound, upper bound along each dimension.

(6) Offset describing storage position to be allocated to name.

8.2.2 How to Store Names in Symbol Table ?

There are numerous ways of Representing names in symbol table. The most common of them is **Array representation** of symbol table.

Array Representation is cateogorised into various subcateogories :

(1) Fixed length array Representation

(2) Variable length array Representation

(3) Two array Representation

(4) Array Name Representation

(*a*) Fixed number of Dimension (*b*) No Limit on Dimensions.

8.2.2.1 Fixed Length Array Representation

(1) It is one of the simplest method of representing names in **contiguous array records.**

(2) Name field of the symbol table is of fixed size.

(3) Similarly size is also fixed for amount of information that can be stored in for a particular name.

Consider IBM 370 Representation

(1) Length of identifier = 08 characters

(2) Amount of information = 16 characters

Suppose each block can only hold 4 characters, hence we require 2 blocks to store a Identifier and 4 blocks to store its Information

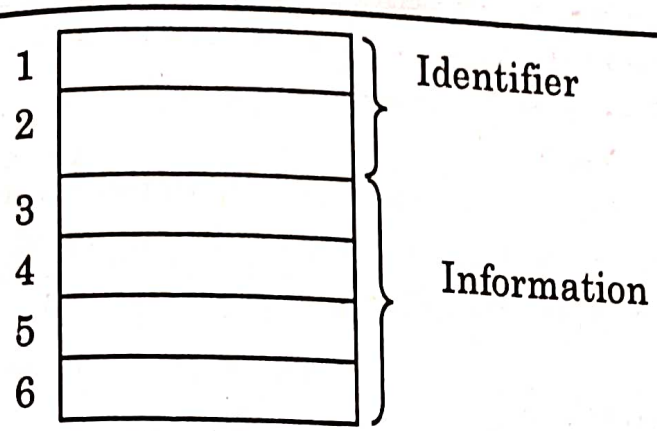
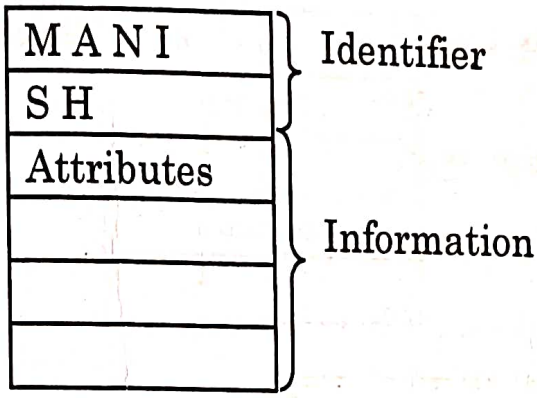
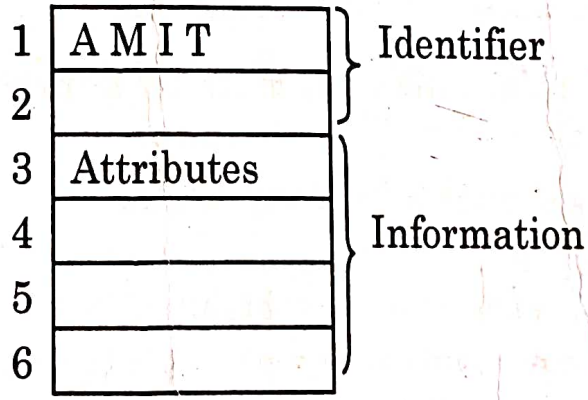


Fig. 8.3 : Fixed Length Array Representation.

Let us consider name **MANISH**



It is clearly visible that space allocated to name is wasted as only one and half block is occupied, however 2 block is allocated for name. Now consider another example



Here there is wastage of 1 complete block. We summed up this using following diagram. Shaded boxes in the symbol table denotes wastage of memory.

	Name								Information
1	M	A	N	I	S	H			
2	A	M	I	T					

Advantage

(1) Easy to understand

(2) Easy to implement and access

Disadvantages

(1) Wastage of large amount of memory
(2) Sequential access
(3) Insertion and deletion of records is comparatively very slow.
(4) Requires large processing and memory Requirement

8.2.2.2 Variable Length Array Representation

Instead of storing names directly in symbol table, we will save all names in separate array of character and will only store start index and length of name. This will overcome the disadvantage of fixed length name. Now we may have variable length exceeding 8 characters as it was fixed in previous method. Similarly we can represent previous example as

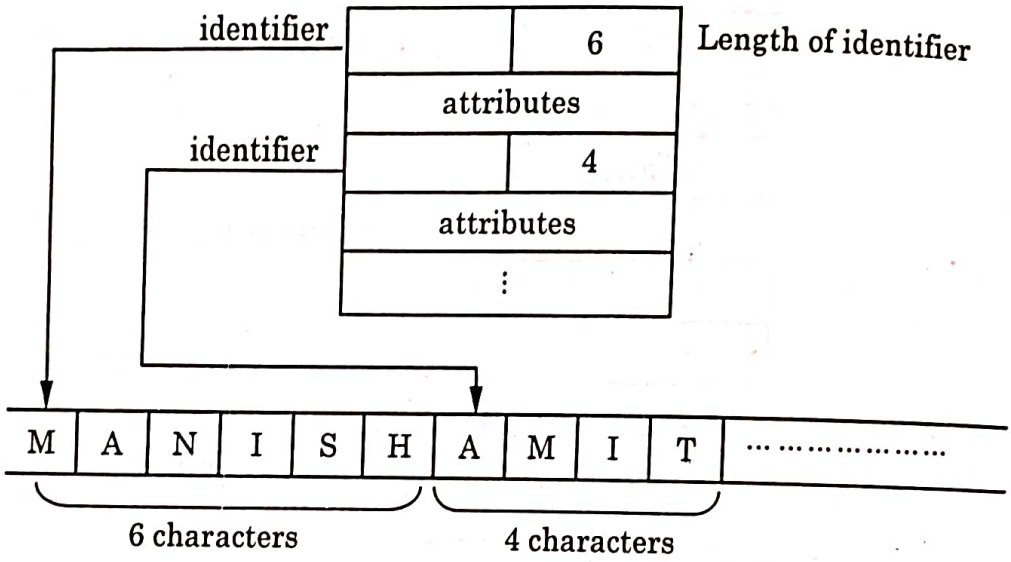


Fig. 8.4 : Variable Length Array Representation

Advantages

(1) Variable names can now be of any length.
(2) Ease of access
(3) No wastage of memory in symbol table
(4) Symbol table now more organised
(5) facilitates quicker editing

Disadvantages

(1) Problem of indirection maintenance
(2) Requirement of extra array record for storing variable name

8.2.2.3 Two Array Representation of Symbol Table

In symbol table we may store different type of information and names. Information associated with different names also varies. Hence it is not advisory to allocate same space to each type of name appearing in symbol table.

So, we can use two arrays to store name & information

Characteristics of Two array Representation.

(1) Instead of storing records in 1 array we will be using 2 arrays.

(2) One array is used to store identifiers and another array is used to store its attributes.

(3) Two arrays are connected with a rule.

If we suppose, An Identifier takes 1 word of memory & Information corresponding to Identifiers or names takes 4 words. If we store identifier at position $2i$ and the corresponding information of identifier at $4i$.

As, information takes 4 words. Therefore, information will be stored starting from $4i$ & ending at $4i + 3$.

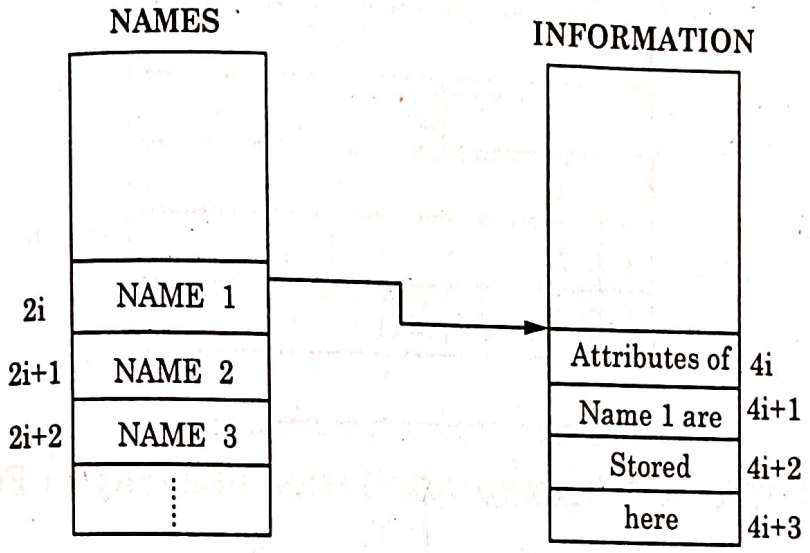


Fig. 8.5 : Two Array Symbol Table Scheme.

So, to Access the value of particular Name & its corresponding information, we just have to put value of i.

E.g. If we put $i = 1$

$\therefore$ Name at location $2i = 2 * 1 = 2$ will be accessed.

Information at location = $4i = 4$ will be accessed.

$\therefore$ Depending on various values of i, we can find different information corresponding to different names.

Advantages

(1) Faster access to information

(2) Low wastage of memory

Disadvantages

(1) Requirement of additional array

(2) Complexity of symbol table increases

(3) Memory requirement increases

8.2.2.4 Array Names Representation

This Representation is used to represent array name and its dimensions and other attributes in symbol table.

Array names can be Represented in symbol table in two different manner :–

(1) Fixed number of dimensions

(2) Variable number (No Limit) of dimensions.

(1) Fixed number of dimensions: Consider an example of FORTRAN, where number of dimension is limited to 3 for an array Fig. 8.5 show array representation in symbol table

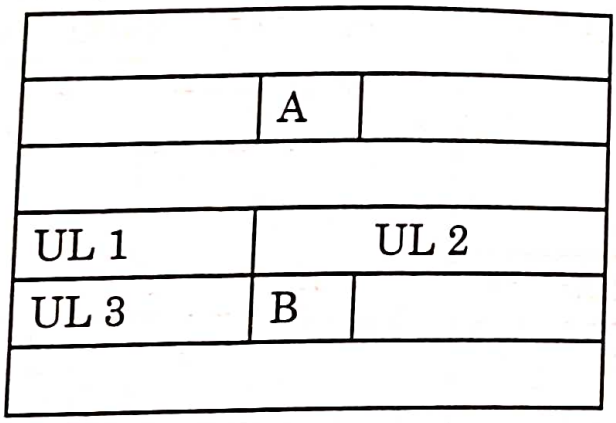


Fig. 8.5 : Symbol table representation of Array in FORTRAN

(*a*) The word containing A represent dimension of array. A is 2 bit long

Bit 1	Bit 2	Description of Array
0	0	Not an array
0	1	1 Dimensional
1	0	2 dimensional
1	1	3 dimensional array

(*b*) UL1, UL2, UL3 represents upper limit for each of the dimension of Array. The Upper Limits of array can be formal parameter.

For this possibility we have provided 3 bits for B to define whether any of three possible limits are formal parameter

0	1	0

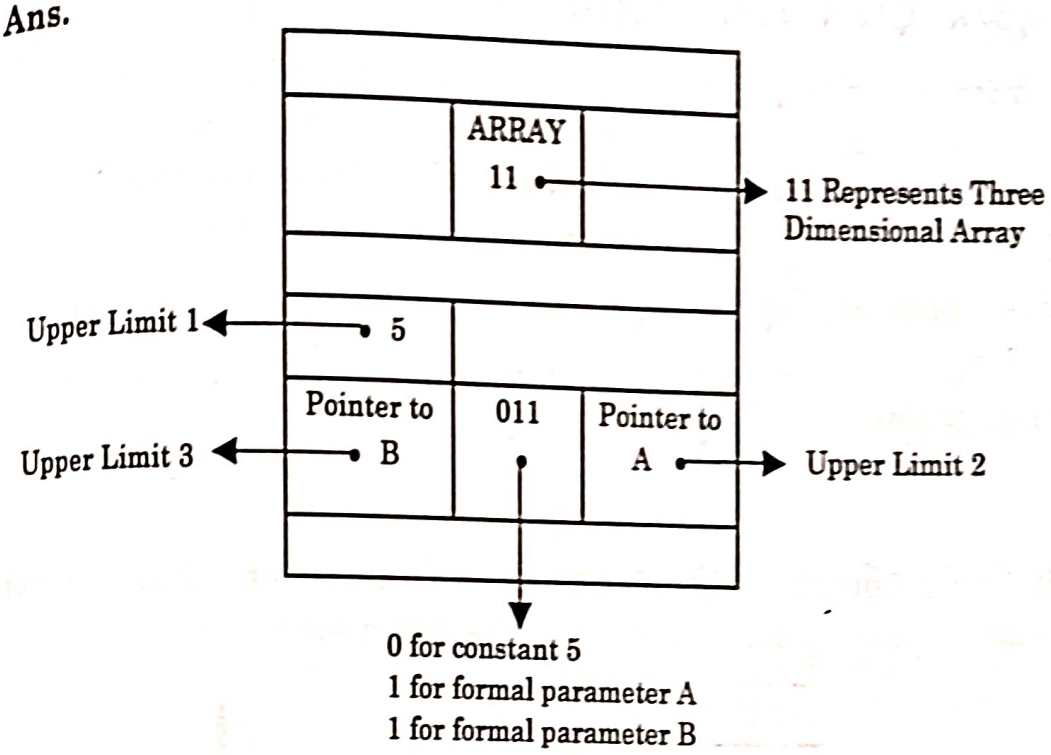
Here *i*th bit is 1 that means UL*i* is formal parameter.

In above case 2nd bit is 1 and hence UL2 is passed as formal parameter and UL2 in symbol table now will contain a pointer to Symbol table Record containing value of UL2.

Example 1: *Consider following subroutine in FORTRAN.*

SUBROUTINE S (ARRAY, A, B)
INTEGER A, B
REAL ARRAY (5, A, B)
Construct Symbol Table Record for ARRAY in this Subroutine.

Ans.



∴ UL1 = 5

UL2 = Pointer to record for name A

UL3 = Pointer to record for name B.

(2) Linked array representation of array with no limit on dimension :

(1) Symbol table will store number of dimension of an array.

(2) It contains a pointer to an array containing lower and upper limit of first dimension, this array is connected to another array which contain lower and upper limit for second dimension and so on.

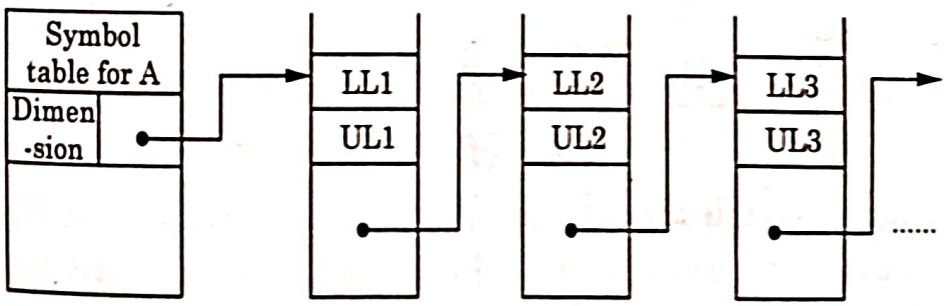


Fig. 8.6 : Storage of Array with no limit on dimensions

Advantages

(1) Random access of symbol table is facilitated.

(2) Ease of accessibility is increased.

(3) Easy to maintain information.

Disadvantages

(1) Problems of pointer indirection and dangling references.

(2) extra requirement of seperate array for each dimension.

8.3 DATA STRUCTURE FOR SYMBOL TABLE

Information in symbol table can be stored in computer system in various different data structures. Each data structure has its own advantages and disadvantages. **Most common data structures used for symbol table :-**

(*a*) Lists

(*b*) Self Organizing Lists

(*c*) Trees

(*d*) Hash tables

8.3.1 List

(1) It is conceptually simplest and easy to implement data structure for symbol table in linear list of records as shown below :

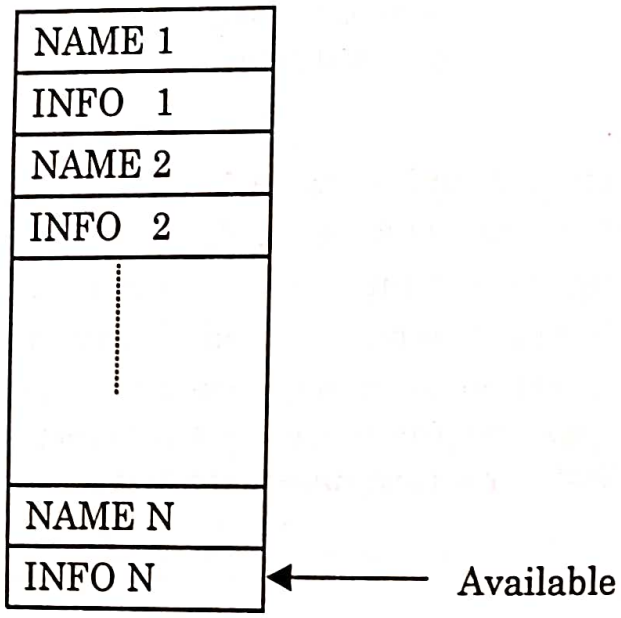


Fig. 8.7 : List

(2) We can use single array to store the name and its associated information.

(3) New names are added to end of list on first come first serve basis.

(4) Method of access is sequential.

Some fact about list data structure :

(a) Work necessary to insert new name is proportional to n, where symbol table contain n names.

(b) cost of searching a name in list is proportional to $n/2$.

(c) To insert n names and m enquires, total work is

$cn\ (n + m)$ Where c is constant.

Advantages

(1) minimum space requirement

(2) Easy to understand & implement

Disadvantages

(1) sequential access

(2) amount of work done required to be done is high.

8.3.2 Self Organizing List

(1) This approach is used to decrease searching time in list organization.

(2) Special attribute link is added to information of name that allows dynamic features in list, this feature help in minimizing wastage of space and also promotes reusability to some extent.

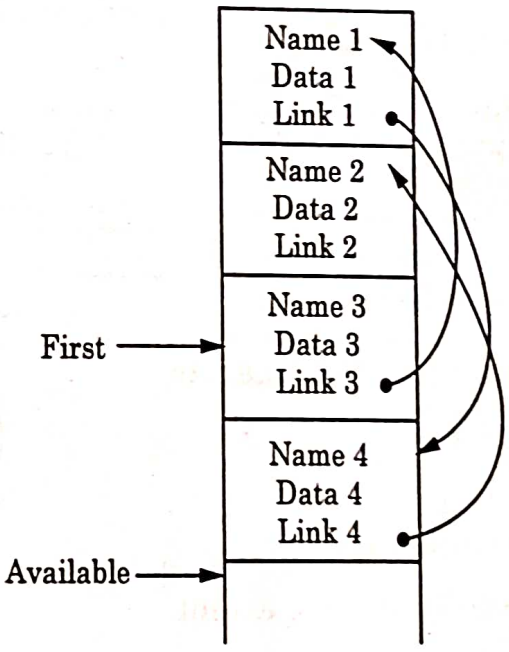


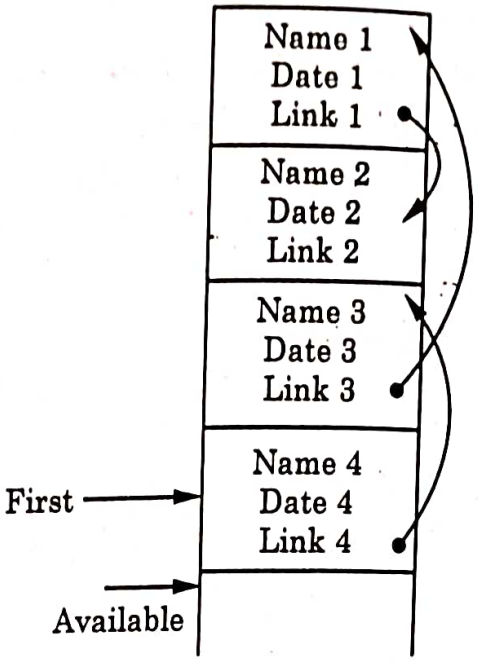
Fig. 8.8 : Self organizing List

- **First pointer** points to beginning of symbol table.
- If we have names in following order

  $3 \longrightarrow 1 \longrightarrow 4 \longrightarrow 2$
- If we need a particular record we need to move record to front of the list.

Suppose we need 4, so, new connections will be

$4 \longrightarrow 3 \longrightarrow 1 \longrightarrow 2$



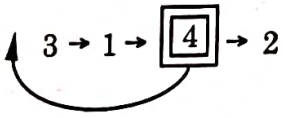
Mechanism

(*a*) Initial stage $3 \longrightarrow 1 \longrightarrow 4 \longrightarrow 2$

(*b*) 4th Name is required

$3 \longrightarrow 1 \longrightarrow \boxed{4} \longrightarrow 2$

(*c*) Move 4th to front



(*d*) $\boxed{4} \longrightarrow 3 \longrightarrow 1 \longrightarrow 2$

adjust connection (self organize)

(*e*) $4 \longrightarrow 3 \longrightarrow 1 \longrightarrow 2$.

Advantages

— Increases search efficiency.

— Promotes Reusability to some extent.

Disadvantages

— difficult to maintain so many links and connection.

— Extra space requirement is there.

8.3.3 Hash Tables

— Hashing is an important technique used to search the records of symbol table. This method is superior to list organization.

- In hashing scheme two tables are maintained
  (a) hash table
  (b) symbol table
- Hash table consist of K enteries from 0 to K – 1. These enteries are basically pointer to symbol table pointing to names of symbol table.
- To determine whether "Name" is in symbol table we, use a hash function 'h' such that h (Name) will result integer between 0 to K –1. We can search any name by
  position = h (Name)
- Using this position we can obtain the exact locations of name in symbol table.
- The hash function should result in distribution of name in symbol table.
- The hash function should be such that there will be minimum number of collision. Collision is a situation where hash function result in same location for storing names.
- Various collision resolution techniques are open addressing, chaning and rehashing.
- The **advantage** of hashing is quick search and **disadvantage** is that hashing is complicated to implement. Some extra space is required & obtaining scope of variable is difficult.

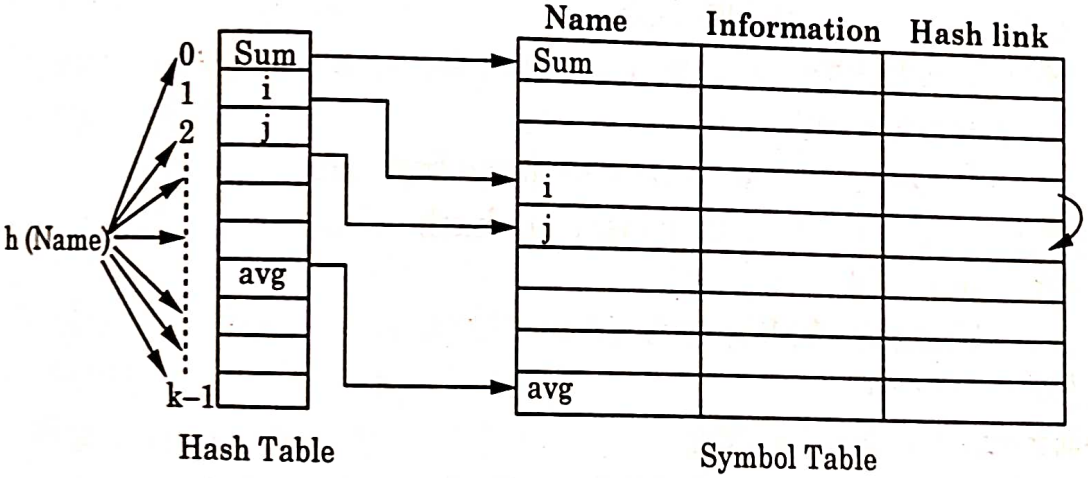


Fig. 8.9 : Hash Table

8.3.3.1 Types of hash function

(1) **Mid square method**

(a) Consider a key K
Let K = 3230

(b) Find the square of K
$K^2 = (3230)^2 = 10432900$

(c) Apply hash function

(1) hash function will eliminate few digits from both sides of K^2.

(2) Hence H(K) = 32

Similarly

K	7236	2435
K^2	52359696	5929225
H(K)	59	29

(d) deleting the number of digits from both sides depend on maximum number of keys required, deletion may or may not be same from both ends.

(2) Division method

Let key = K

no. of element = n

hash function = H(K)

H(K) = K mod m or H(K) = (K mod m) + 1

where m = prime number (largest)

Let us consider an example, we have to index 100 records and hence largest prime number close to 100 is 97, hence we will choose $m = 97$

Let K = 3230

H (3230) = 3230 mod 97 = 29

Hence hash function result in hash value = 29.

(3) Folding method

— We may use this method for large key values

— We can use two methods to get hash value

(1) Shift folding

(2) Boundary folding

(a) Shift Folding :

K = 324 987 626 191 532.

divide this key value into parts K_1 K_2 K_3 K_4 K_5 as shown below

$K_1 = 327$

$K_2 = 987$

$K_3 = 626$

$K_4 = 191$

$K_5 = 532$

2663

Hence

$H(K) = K_1 + K_2 + K_3 + K_4 + K_5 = 2663$

We can further reduce it to two digit by applying some other hash function.

(*b*) **Boundary folding :** is another folding method. Here some of sub keys are reversed to generate new simple hash values

$K = 327\ 987\ 626\ 191\ 532$

$K_1 = 327$

$K_2 = 789$

$K_3 = 626$

$K_4 = 191$

$K_5 = 253$ } Reversed

2168

$H(K) = K_1 + K_2^R + K_3 + K_4 + K_5^R$

$= 2168$

Here again we have to apply some hashing function to reduce it to two digit number.

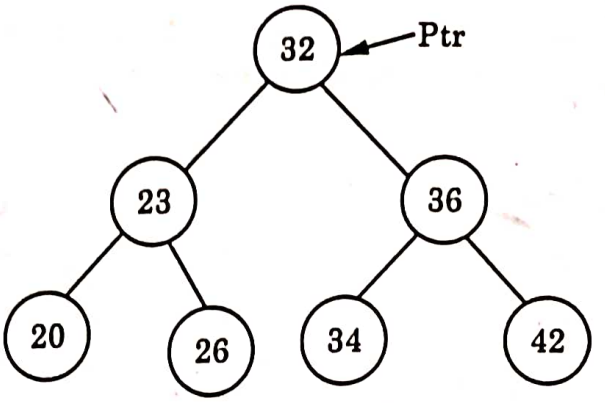
8.3.4 TREES

- Symbol table can also be stored in form of binary search tree.
- **Binary search tree have following properties :**
- each node can have atmost two children
  (*a*) left child (*b*) right child
- key value stored in parent node is more than key value stored in left of the children and less than right child.
- height of binary search tree is of order of log *n*.
- Maximum time required for searching and inserting is proportional to O (log *n*)

8.3.4.1 Binary Search Tree Algorithm

(1) Initially, Ptr will point to root of Tree.
(2) **While** Ptr ≠ NULL do
(3) **If** Name = Name (Ptr)
then return true
(4) **else if** Name < Name (Ptr) then
Ptr : = Left (Ptr)
(5) **else** Ptr : = Right (Ptr)
(6) end of Loop

E.g→ Consider a tree



Suppose we want to find 34 where Name (Ptr) = 32

Pass 1.

$34 \neq 32$

hence it will go to else loop

$34 > 32$

Now Ptr = Right (Ptr) = 36

Pass 2.

Again

$34 \neq 36$

hence it will go to else loop

$34 < 36$

$\therefore$ Ptr = left (ptr) = 34

Pass 3.

34 = = 34

hence it will return true

8.4 REPRESENTING SCOPE INFORMATION

Representing scope information is a concept in which scope of each variable name is preserved in symbol table so that we can use the same name in different blocks and different location.

Representing scope information involves :–

(1) Lifetime of variable in a particular block

(2) Representing names in symbol table along with indicator of block in which it appear.

(3) Suppose we have a variable name 'a' in block A and same variable in block B. If we store 'a' in symbol table without block information it will only store the first instance of 'a' which it encounter, hence to overcome this problem names are stored in form of pair (variable name, block name)so that same name can be used in different blocks and procedures.

There are two ways of representing scope information

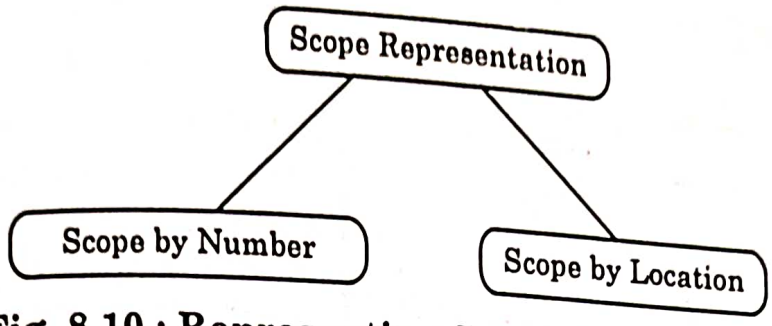


Fig. 8.10 : Representing Scope Information

(4) Scope Representation reflects visibility of variable name in source program.

8.4.1 Scope Representation by Number

It stores all values in one Symbol Table.

The same name can be declared many times as distinct name in different blocks or procedures.

So, each procedure or block can be given a unique number.

So, Symbol Table will not only contain just the name of identifier, but each entry will contain a pair (name, procedure–number).

So, we can recognize the particular identifier in a procedure by matching not only the name of the identifier but also to match procedure–number to which it belongs.

Each block or procedure will be assigned a number.

For example :

```
Procedure 1( )
{ var a, b;
Procedure 2 ( )
{
      var   a, x ;
}

Procedure 3 ( )
{
      var   c, x ;
}
}
```

Procedure No. 2

Procedure No. 3

Procedure No. 1

Following shows the configuration of Symbol Table. *i.e.* fill identifiers with their corresponding procedure or block no.

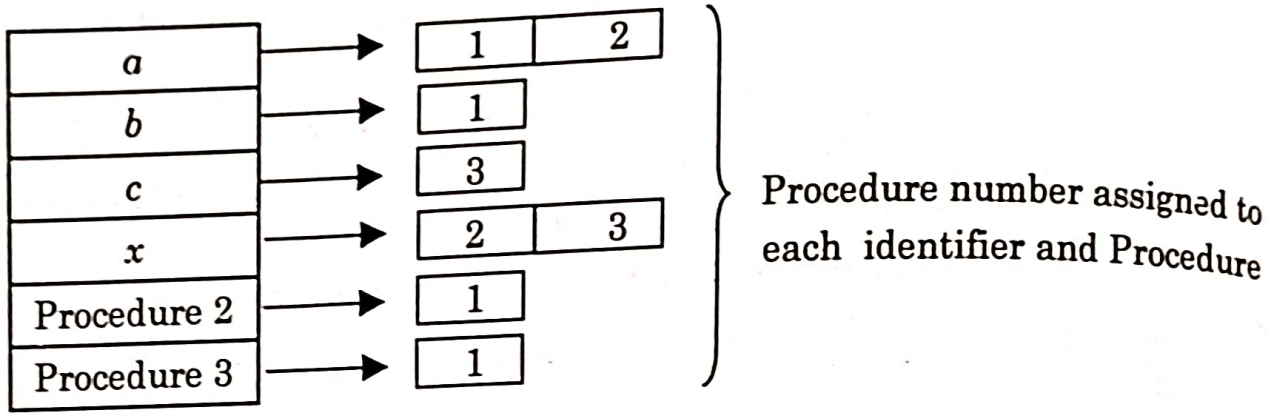


Fig. 8.11 : Scope Representation by Number

Here, the identifier or name will belong to most closely nested of active sub programs declaring that identifier or procedure.

8.4.2 Scope Representation By Blocks or Location

(1) It creates separate table for each Scope Block or procedure.

(2) Multiple identifier with same name can easily be inserted because same name variable will be shared in separate block.

(3) However, we cannot have two variables with same name in same block or procedure.

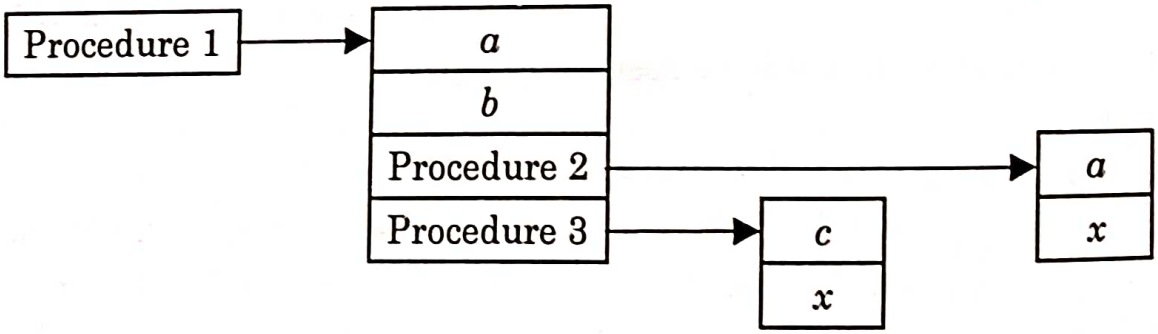


Fig. 8.12 : Scope Representation by Block

(4) Each procedure or subprogram will have a different table representation of its contents. When an identifier will be searched, proper table *i.e.* subprogram or procedure number and identifier name will be matched.

Above Diagram shows the Symbol Table Representation representing Scope information Blockwise.

(5) This approach is more reliable and more easy to understand.

8.4.3 Representing Scope information in FORTRAN

- FORTRAN program consist of main program, subroutines and function.
- Each name has a scope consisting of one routine *i.e.* variable scope is limited to end of routine.
- We can generate object code for each routine upon reaching end of routine and hence the names stored in symbol table for variable belonging to that particular routine need to be eliminated.
- hence we need to store only names that are external to routine and also of common block in symbol table.
- Consider a hashing scheme as shown.

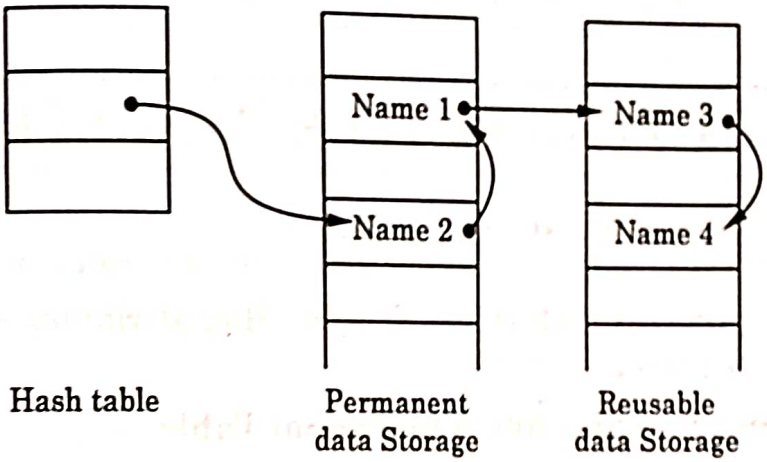


Fig. 8.13 : Hash table with Permanent and Resuable Storage

- In each chain, names that are external to current routine will appear first.
- We append new internal names to end and new external name to beginning.

It is clear from above diagram hash value has selected name 2 and hence 2 is part of current routine and Name 1 is external to it, thats why Name 1 is written in beginning and Name 2 is appended at end.

- When we reach end of routine and generate object code for it, we reset pointer to available storage in reusable table but not permanent one.
- All pointer that are from permanent table to reusable table are set to nill and hash table will directly point to reusable table.
- We can also reuse the storage space used for names by following way
  - We will use location in name link storage areas to represent name, such a location may represent different routines, since that area is reused.

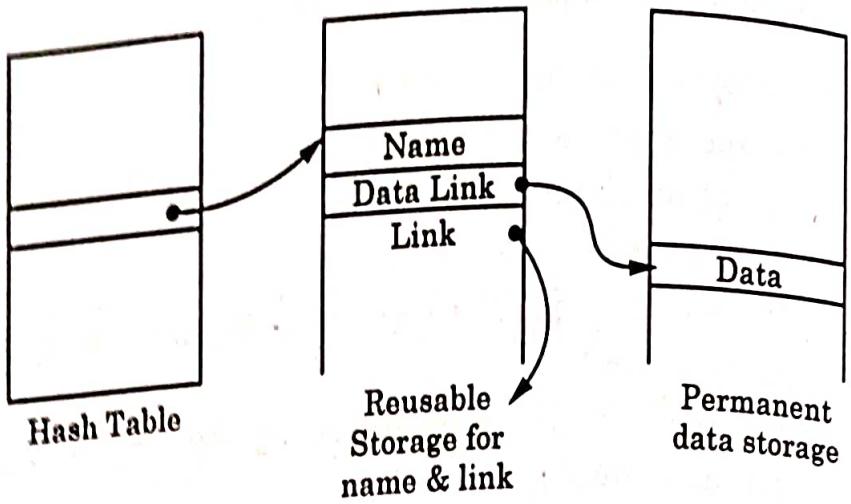


Fig. 8.14 : Reusing Storage Space

KEY POINTS TO REMEMBER

⇒ **Symbol Table :** It is a data structure which store various names or identifiers used in source program with their corresponding attributes such as their data type, scope or its address.

⇒ **Methods of representing name in symbol Table :–**

(1) Fixed Length Representation
(2) Variable Length Representation
(3) Two Array Representation
(4) Array Name Representation
   (*a*) Fixed No. of dimension (*b*) No Limit on Dimension.

⇒ **Data structures for Symbol Table**

(1) Lists (2) Self Organizing Lists
(3) Tress (4) Hash Table.

⇒ Each symbol table has the responsibility to store with each identifier, its scope *i.e.* its procedure or sup-program number to which it belongs.

REVIEW QUESTIONS

Q.1. Define Symbol Table.
Q.2. Discuss Two Array Symbol table scheme.
Q.3. What are applications of symbol table in compiler design ?
Q.4. Explain various data structure for Symbol Tables.
Q.5. How is the scope information represented in Symbol Table ?