Assignment No: - C-17

Aim: To study Symbol Table and various operations on it.

Title: - Symbol Table implementation.

Problem Statement: The symbol table is generated by compiler. From this perspective, the symbol table is a set of name-attribute pairs. In a symbol table for a compiler, the name is an identifier, and the attributes might include an initial value and a list of lines that use the identifier. Perform the following operations on symbol table:
(1) Determine if a particular name is in the table
(2) Retrieve the attributes of that name
(3) Modify the attributes of that name
(4) Insert a new name and its attributes
(5) Delete a name and its attributes

Theory:

During type checking, a compiler checks whether the use of names (such as variables, functions, type names) is consistent with their definition in the program. Consequently, it is necessary to remember declarations so that we can detect inconsistencies and misuses during type checking. This is the task of a symbol table. Note that a symbol table is a compile-time data structure. It's not used during run time by statically typed languages. Formally, a symbol table maps names into declarations (called attributes), such as mapping the **variable name x to its type int.** More specifically, a symbol table stores:

- For each type name, its type definition.
- For each variable name, its type. If the variable is an array, it also stores dimension information. It may also store storage class, offset in activation record etc.
- For each constant name, its type and value.
- For each function and procedure, its formal parameter list and its output type. Each formal parameter must have name, type, type of passing (by-reference or by-value), etc.

DATA STRUCTURE FOR SYMBOL TABLE:

List:

The simplest and easiest to implement data structure for symbol table is a linear list of records. We use single array or collection of several arrays for this purpose to store name and their associated information. Now names are added to end of array. End of array always marks by a point known as space. When we insert any name in this list then searching is done in whole array from 'space' to beginning of array. If word is not found in array then we create an entry at 'space' and increment 'space' by one or value of data type

Variable	Information(type	Space (byte)
a	Integer	2
b	Float	4
c	Character	1
d	Long	4

SPACE

Figure: Symbol table as list

Hash Table:

A hash table, or a hash map, is a data structure that associates keys with values 'Open hashing' is a key that is applied to hash table. In hashing –open, there is a property that no limit on number of entries that can be made in table.

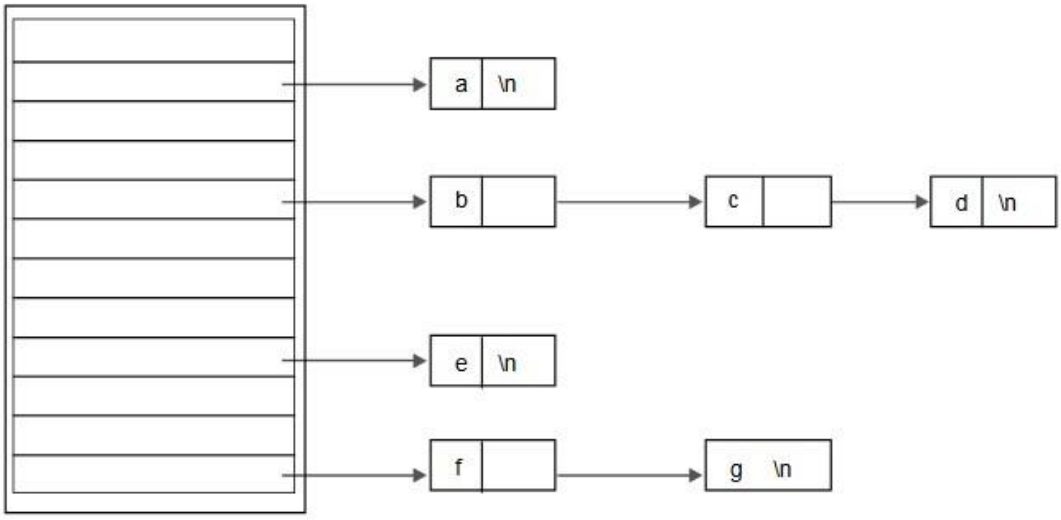


Figure: Symbol table as hash table (\n represent NULL)

Search Tree:

Another approach to organize symbol table is that we add two link fields i.e. left and right child, we use these field as binary search tree. All names are created as child of root node that always follow the property of binary tree i.e. name <name I and Name j <name. These two statements show that all smaller name than Name i must be left child of name otherwise right child of name j. For inserting any name it always follow binary search tree insert algorithm.

Example : Create list, search tree and hash table for given program for given program int

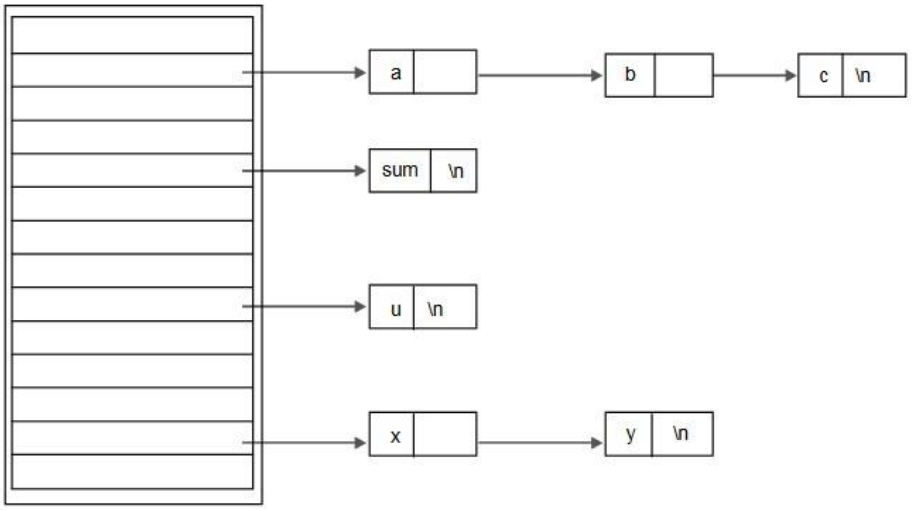
```
a,b,c;
int sum (int x, int y)
{ a = x+y
return (a)
} main ()
{ int u,
u=sum (5,6);
}
```

i) List

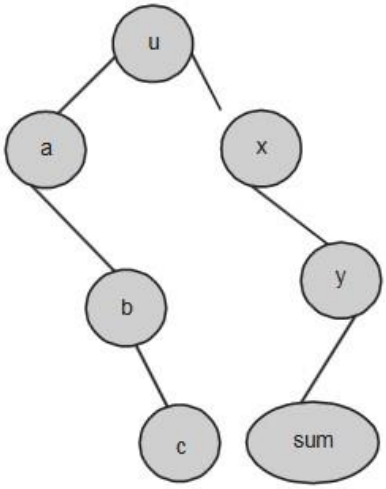
Variable	Information	Space(byte)
u	Integer	2 byte
a	Integer	2 byte
b	Integer	2 byte
c	Integer	2 byte
x	Integer	2 byte
y	Integer	2 byte
sum	Integer	2 byte

Space

ii) Hash Table



(iii) Search Tree



Algorithm:

Algorithms are similar to BST Dictionary implementation for Insert, Delete, update and Search operations. So write the algorithms by referring the Symbol table assignment.

Flowchart:

Conclusion:

FAQ:

1. Is the sequence ⟨23, 17, 14, 6, 13, 10, 1, 5, 7, 12⟩ a heap?
2. Where in a heap might the smallest or largest element reside?
3. What are the minimum and maximum numbers of elements in a heap of height *h*?
4. What are the time and space complexities of heap?