2.1 Symbol Table

A compiler uses a symbol table to keep track of scope and binding information about names (which is the key of the symbol table search). The symbol table is search every time a name is encountered in the source text. Changes to the symbol table occur if a new name or new information about an existing name discovered. The information stored in symbol table is collected by the analysis phases of the compiler and used by the synthesis phases to generate the target code.

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.

2.2 Symbol Table Entries

Each entry in the symbol table is for declaration of a name. the format of entries does not have to be uniform, because the information saved about a name depends on the usage of the name, each entry can have implemented as a record consisting of a sequence of consecutive words of memory, to keep symbol-table records uniform. It may be convenient for some of the information about a name to be kept outside the table entry with only a pointer to this information stored in the record.

2.3 Structure of Symbol Table Record

Each record represents a row containing a list of attribute value that are associated with a particular variable. These attributes depend on the nature of the programming language for which the compiler is written. The following are the typical attributes for a symbol table: -

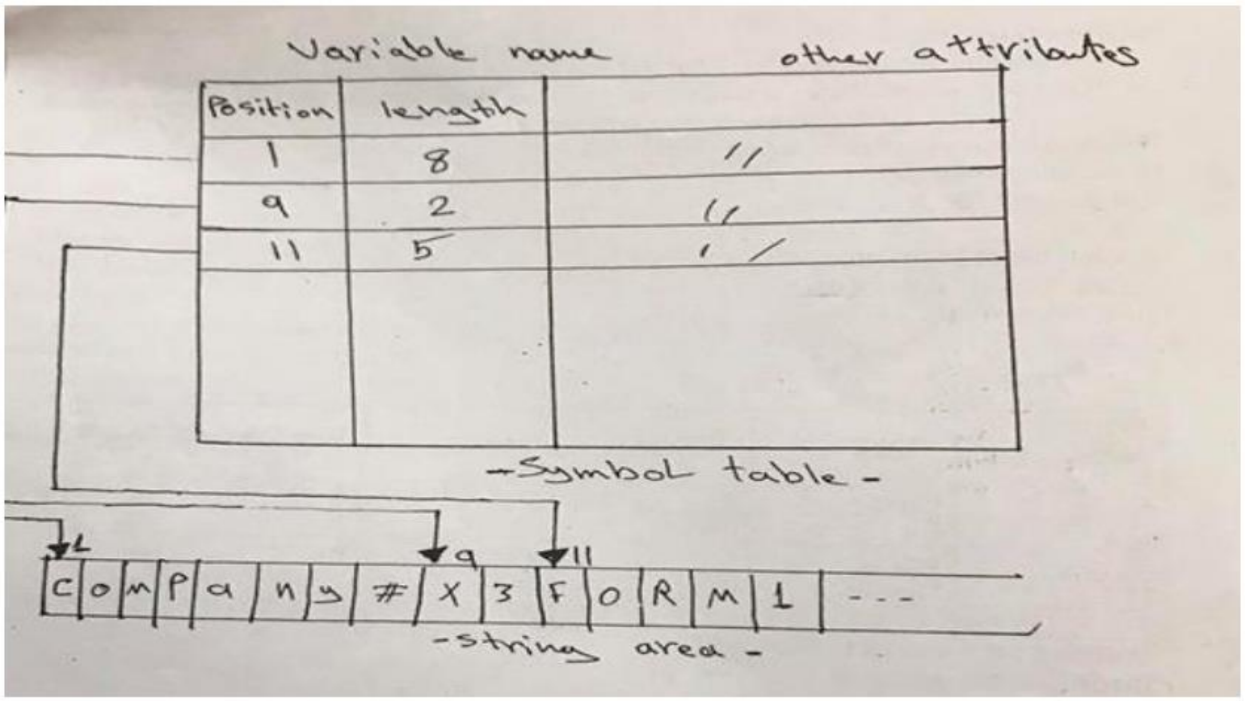
Variable Name	Address	Type	Dimension	Line Declared	Line Reference
A	0	Int	0	2	9,10,11,13
B	2	Int	0	2	13,14
K	4	Char	1	3	11
N	9	Char	0	4	12
G	10	Char	2	5	
C	40	Int	1	6	14

Symbol Table Attributes:

1- Variable name must always reside in the symbol table, since it is the means by which a particular variable is defined for semantic analysis and code generation. A major problem in symbol table organization can be the variability in the length of identifier names. Two popular approaches to solve this problem:

a- We reserve a storage for maximum variable name length which provides quick table access.

b- We can place a string descriptor in the variable name field of the table. The descriptor contains position and length subfields. The position indicates the first character of the variable name in a general string area, and length subfield describes the number of characters in the variable name. A table access must always go through the descriptor to the string area for variable name matching, so this approach results in slow table access but saving of storage.



2- Object Code Address Must be associated with every variable in a program. This address dictates the relative location for values of a variable at run time. The object code is entered in the symbol when a variable is declared (or first encountered). This address is recalled from the table when the variable is referenced in the source program. Then, this address will be used by the instruction (load and store), that accesses the value of that variable.

3- Type: the type of a variable is used for both – An indication of the amount of memory that must be allocated to a variable of run time (e.g. int 2 byte, char 1 byte). It is the task of the semantic when handling declaration statement – for semantic checking (e.g. it is semantically inconsistent to multiply a string by a number).
4- Number of dimension and the number of parameters. Both are important for semantic checking. In array reference, the number of dimensions should agree with the number specified in the declaration of the array. Also, the number of parameters, in a procedure call must also agree with number used in the procedure heading or declaration.
5- Line declared Contains the line number in which the variable is declared in the source program.
6- Line referenced Contains all the line numbers that the variable is used in the source program.

*If the last two attributes are to be included in the table then the listing of the table is called **Cross-Reference Listing.**

EX/ Draw a diagram of a cross reference symbol table that would result when compiling the following program segment (with reporting error & warning message if any)

```
1 main()
2 {
3 int i,j[5];
4 char c, index[5][6], block[5];
5 float f;
6 i=0;
7 i=i+k;
8 f=f+I;
9 c='x';
10 block[4]=c;
11 }
```

Name	Object address	Type	Dime	Line Declared	Line Referenced
i	0	int	0	3	6,7,8
j	2	int	1	3	
c	12	char	0	4	9,10
index	13	char	2	4	
block	43	char	1	4	10
f	48	float	0	5	8
k					7

Error line7: k is undefined (semantic)
Warning line8: f used before initialization (semantic)
Warning line3: j is not used (semantic)
Warning line4: index is not used (semantic)

2.4 Operations Implemented on Symbol Table

The symbol table provides the following operations:

Insert ()

- Insert () operation is more frequently used in the analysis phase when the tokens are identified and names are stored in the table.
- The insert () operation is used to insert the information in the symbol table like the unique name occurring in the source code.
- In the source code, the attribute for a symbol is the information associated with that symbol. The information contains the state, value, type and scope about the symbol.
- The insert () function takes the symbol and its value in the form of argument.

For example:

int x;

Should be processed by the compiler as:

insert (x, int)

lookup()

In the symbol table, lookup() operation is used to search a name. It is used to determine:

- The existence of symbol in the table.
- The declaration of the symbol before it is used.
- Check whether the name is used in the scope.
- Initialization of the symbol.
- Checking whether the name is declared multiple times.

The basic format of lookup() function is as follows:

lookup (symbol)

This format is varying according to the programming language.

2.5 Data structure for symbol table

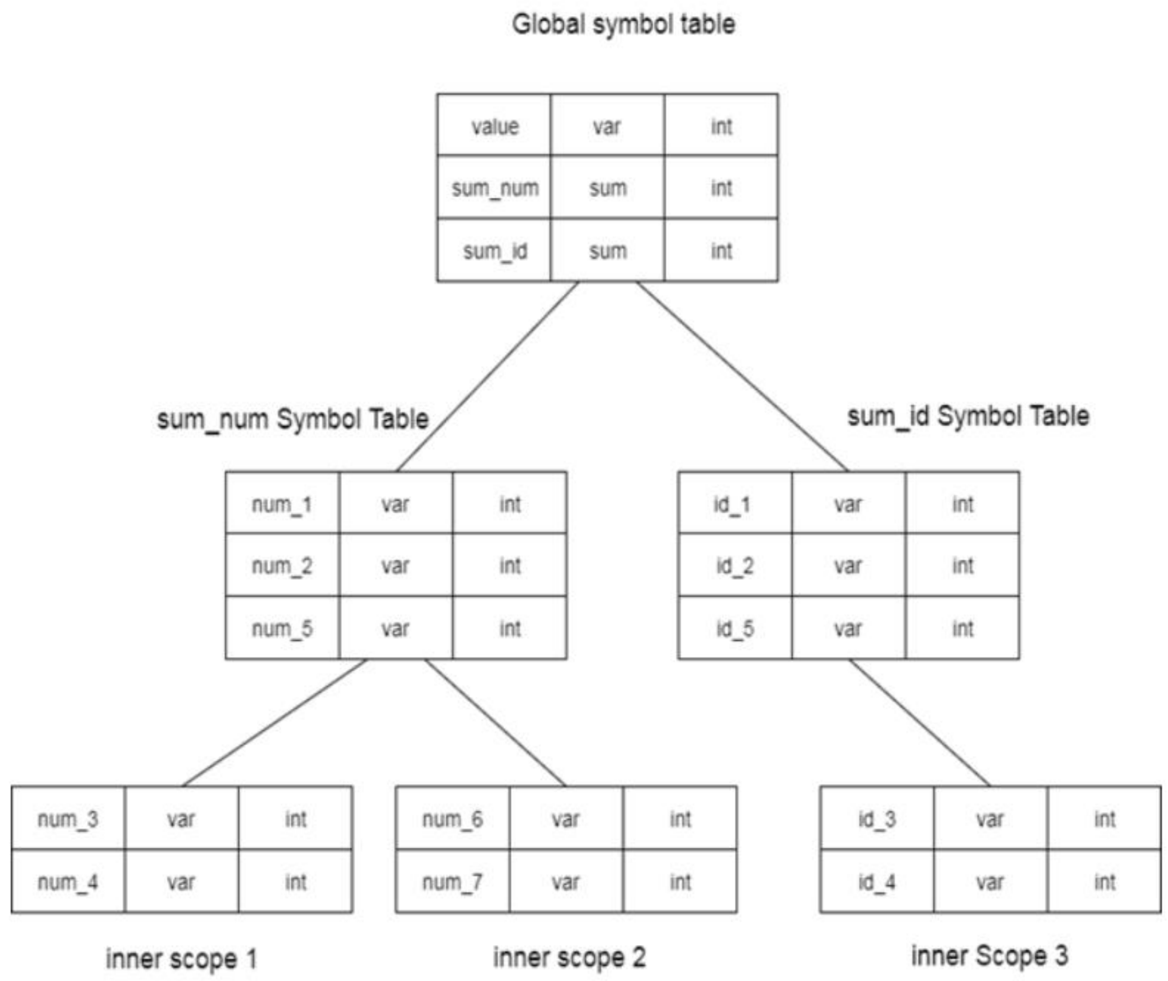
- A compiler contains two type of symbol table: global symbol table and scope symbol table.
- Global symbol table can be accessed by all the procedures and scope symbol table.

The scope of a name and symbol table is arranged in the hierarchy structure as shown below:

For example, if we take the following code:

```
int value=10;

void sum_num()
    {
      int num_1;
      int num_2;

          {
           int num_3;
           int num_4;
          }

      int num_5;

          {
           int_num 6;
           int_num 7;
          }
    }

Void sum_id
    {
      int id_1;
      int id_2;

          {
           int id_3;
           int id_4;
          }

      int num_5;
 }
```

The data structure of symbol tables is represaented as follow:



*The global symbol table contains one global variable and two procedure names. The name mentioned in the sum_num table **is not available** for sum_id and its child tables.

2.6 Symbol Table Organizations

There are different organizations from simple which are storage efficient, to more complex which provide fast table access. The primary fact used to determine the table complexity is the average length of search; which is the average number of compressions required to retrieve a symbol table record in a particular table organization.

- **Unordered Symbol Table:**

1- simple organization

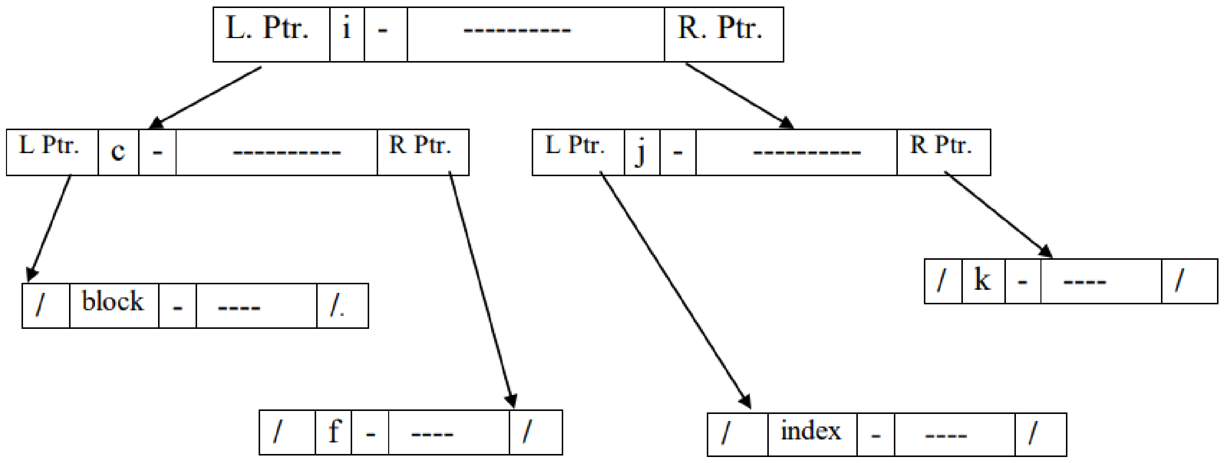
2- attribute entries are added to the table in the order in which the variables are declared.
3- Insertion operation requires no comparisons (other than checking for symbol table overflow).
4- Look up (retrieval) operation requires **average search length** of (n+1)/2 (for n records in the table)
5- can be used only if the expected size of the symbol table is small.

- **Ordered Symbol Table with Binary Search**

1- The table lexically ordered on the variable names.
2- An insertion operation must be accompanied by a lookup operation which determines where in the symbol table the variables attributes should be placed.
3- Table lookup can be implemented using Binary search strategy.

- **Tree Structured Symbol Tables**

Figure 2 shows new fields are present in a record structure, which are the left pointer and right pointer fields. Access to the tree is through the root node (top node of the tree). A search proceeds down the links of the tree until the node is found or a Null link field (/) is encountered, which mean unsuccessful search. Because of this insertion strategy the organization of the tree structure is dependent upon the order of insertion.



(Figure -2) Tree Structured Symbol Tables

Example: Construct un order and order symbol table for the following source code that write in C++ language, and show the errors and warnings message.

1 int sum (int f, int b, int a)

2 { int c,m,f1[10],x3;

3 Char cu[5][3]

4 f=f+b*(a+1)

5 b=b+1;

6 m=m+1;

7 return f;

8 }

1- **Un order Symbol Table.**

Variable name	Type	Dimension	Address	Line Declaration	Line Referencing
f	Int	0	0	1	4.7
b	Int	0	2	1	4,5
a	Int	0	4	1	4
c	Int	0	6	2	-
m	Int	0	8	2	6
f1	Int	1 (10 element)	10	2	-
x3	Int	0	30	2	-
cu	char	2 (15 element)	32	3	-
			47		

2- Order Symbol table

Variable name	Type	Dimension	Address	Line Declaration	Line Referencing
a	Int	0	0	1	4
b	Int	0	2	1	4,5
c	Int	0	4	2	-
cu	char	2 (15 element)	6	3	-
f	Int	0	21	1	4,7
f1	Int	1(10 element)	23	2	-
m	Int	0	43	2	6
x3	Int	0	45	2	-
			47		

Warning Message: The variables (c, f1, x3, cu) declares but not use.

Error Message: No error message.