Definition → Symbol table is an Important data structure created and maintained by Compiler in order to store Information about various Entities like, Variables, function names, names, strings, Constants & Labels.

* It store Information about scope & binding Information about names.
* It is built in lexical & syntax analysis Phase.
* The Information is Collected by the analysis Phases of the Compiler & used by Synthesis Phases of Compiler for Code generation.

It is used by various phases of Compiler as follows:-

Lexical Analysis → Creates new table entries.

Syntax Analysis → add Information about attribute, type, scope, & use in the table.

Semantic Analysis → to check the expressions semantically Correct (Type checking)

Intermediate Code generation – table helps in adding temporary Information for Information Code.

⑤ Code optimization → uses symbol table for machine dependent optimization.

⑥ Code generation :- uses address information of identifier present in the table for code generation.

Information used by compiler from symbol table

① Data type & name

② Declaring Procedure

③ Pointer to Structure / Record

④ Parameter Passing ~~by key~~ value / by Reference

⑤ No and types of arguments passed to function.

⑥ Base address.

Symbal Table

Name	TYPE	Size	Dimension	Line of Declaration	Line usage	Address
x	Int	4	0	4	6-10	
a	char	5	1	3		

```
main()
{
Char a[5];
Int x;
```

Data Structure	Inserted	Lookup time
Unordered Array	O(1)	O(n)
ordered Array	O(n)	O(log n)
Unordered Linked List	O(1)	O(n)
ordered Linked List	O(n)	O(n)
Search tree	O(log n)	O(log n)
Hash table	O(1)	O(1)

Hash - Table

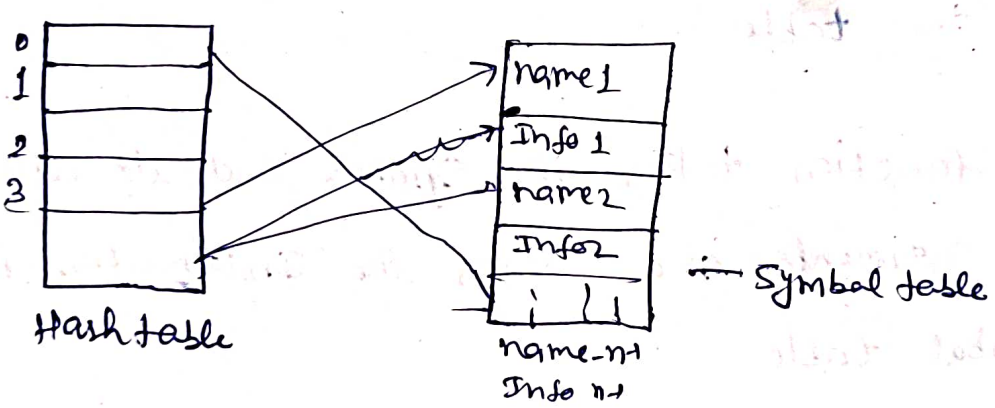
① A hash table is a table of k pointers from 0 to k-1 that points to the record in a symbol table.

② Before entering a name into the symbol table. we find out the hash value of the name by applying hash function.

③ This hash value specify the position of name in the symbol table having value 0 to k-1.

Position of name = h(name) = value (0 to k-1)

④



⑤ The advantage of using hashing is that it is very quick in searching.

⑥ The disadvantage of hashing is that it is complicated to Implement.

Operations

A symbol table either, linear or hash should provide following operations

Insert() * This operation is used to add information in the symbol table about unique names occuring in the source code.

* this operation is used frequently by analysis phase.

* where token are identified and names are stored in the table.

* this function takes the symbol and its attribute as arguments and stores the information in the symbol table.

Ex→ Int a

Insert (a, int);

lookup (←)

* This operators is used to search name in the symbol table to determine.

Ex→ lookup (symbol)

* if symbol exist in the table
* if it is declare before it is being used
* if the name is used in the scope
* if the symbol is initialize or symbol declare multiple times

Data structure for Symbol table

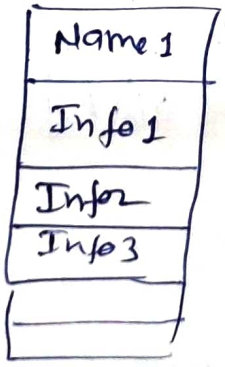
Linear list

① easiest way to Implement Symbol table.

② New names are added to the table in the order that they arrive.

③ whenever a new name is to be added. The table is First searched linearly to check whether or nat the name is already present in the table.

④ if the name is nat present ~~in~~ then the record for new name is created & added to the end of list.



advantage

⑤ Takes less space
addition is easy

⑥ higher accessing time

⑦ Retrieval of Information is also done linearly.

Search Tree

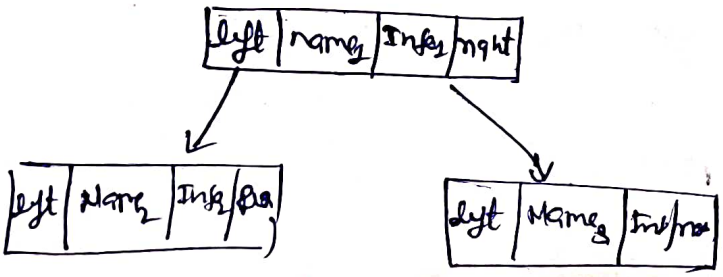
① It is an efficient approach to Implement symbol table.

② We add two links left & right in each record.

③ These links point to the record in the symbol table.

④ Whenever the name is to be added, first the name is searched in the tree.

⑤ If the name is not available then the record for the new name is created and added to the proper position in the search tree.



⑥ For large no of records this method is better as compare to linear list methods.

Run time Storage Strategies

Static allocation

In this allocation scheme, the compilation data is bound to a fixed location in the memory and it does not change when the program executes.

* As the memory ~~Locations~~ Requirement and storage locations are known in advance
* if memory is created at compile time then the memory will be created in static area and only once.
* Static allocation supports the dynamic data structure that means memory is created only at compile time and deallocated after program completion.
* The drawback with static storage allocation is that the size and position of data object should be known at compile time.
* Another drawback is restriction of the Recursion Procedure

In the source program every name possesses a region of validity called the scope of that name.

The Rules in a block structure language

① if a name declared within block B then it will be valid only within B.

② if B1 block is nested within B2 then the name that is valid for block B2 is also valid for B1

* These scope Rules need a more complicated organization of Symbol table than a list of associations between names & attributes.
* Tables are organized into stack and each table contains the list of names and associated attributes.
* whenever a new block is entered then a new table is entered on to the stack.
* when the declaration is compiled then the table is searched for name.
* if the name is not found in the table then the new name is inserted.
* when the name's reference is translated then each table is searched, starting from the each table on the stack.

```
Int {value}
Vaid one ()
{
    int a;
    Int b;
    {
        Int c
        Int d
    }
    Int e
    {
        Int f
        Int g
    }
}
```

Symbol table

Value	Var	Type
one	B+6	Vaid
Two	B+C	Vaid

a	Var	Int
b	Var	Int
e	Var	Int

c	Var	Int
d	Var	Int

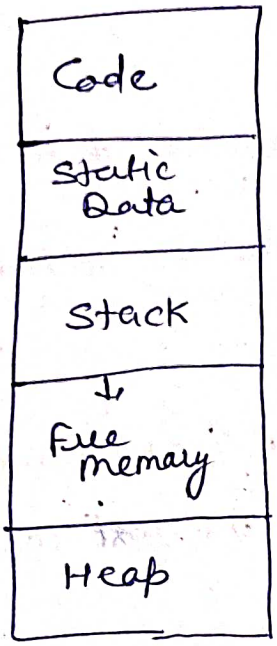
f	Var	Int
g	Var	Int

x	Var	Int
y	Var	Int
r	Var	Int

p	Var	Int
q	Var	Int

```
vaid two ()
{
    Int x;
    Int y;
    {
        Int p;
        Int q;
    }
    Intr
}
```

Sub division of Run-time memory



When Program is running on that time How the memory is utilized.

Code section — where Program is Loaded for execution and also called as text section.

Static data — * global varible, Storage class are Static in C-Language
* Local varible Preserves its value during repeated call.

Heap → dynamic allocation of memory

* At the Execution time of Input Source Program, data objects are an Important factor that occupies some memory.

* Memory size of a source Program calculated from various Information like length, dimensions and machine address

* Each Executed Program would have space for Instruction and data so that memory allocation is not performed in first Pass of Compiler.

  There is a need to manage memory when a Program is running.

* This memory management must connect to the data objects of Program.

* Programs request for memory blocks and release memory blocks.

* Passing Parameters to functions needs attention.

Activation Record

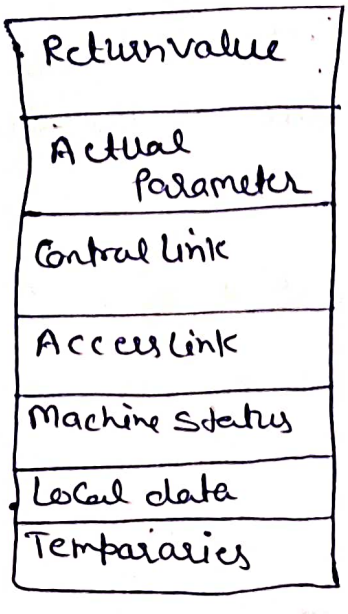
* The Information needed by a single execution or a single activation of a procedure is managed using a contiguous block of storage called Activation Record.

* If a procedure is non recursive then there exist only one activation of procedure at any one time.

* Whereas if a procedure is recursive several activations of that procedure may be active at the same time.

* The size of almost all the fields of an activation record can be determined at compile time.

* The Information in Activation record is organised in a manner that enable easy access at execution time.



Temparaics — are used to store temparary variable

Local data — holds data that are local to an Execution of a Procedure.

Machine Status — field used to store Information about the state of the machine before the Procedure is called.

Access Link (Optional) — This field is used to refer to non local data field in other activation Record.

Control Link — This field Points to activation Record of the Caller.

Actual Parameters — This field is used by the Calling Procedure to supply Parameters to the Called Procedure.

Return Value — This field is used by the Called Procedure to return value to the Calling Procedure.

- A Pointer Called CEP [current Environment Pointer] is used to Point one of the fields of the activation record.

Activation Tree

A Program is a sequence of Instructions Combined into a number of Procedures. Instructions in a Procedure are Executed sequentially.

A Procedure has a start and an end delimeter and everything inside it is called the body of the Procedure.

The Execution of a Procedure is called its activation.

An Activation Record Contains all the neccesary Information required to call a Procedure.

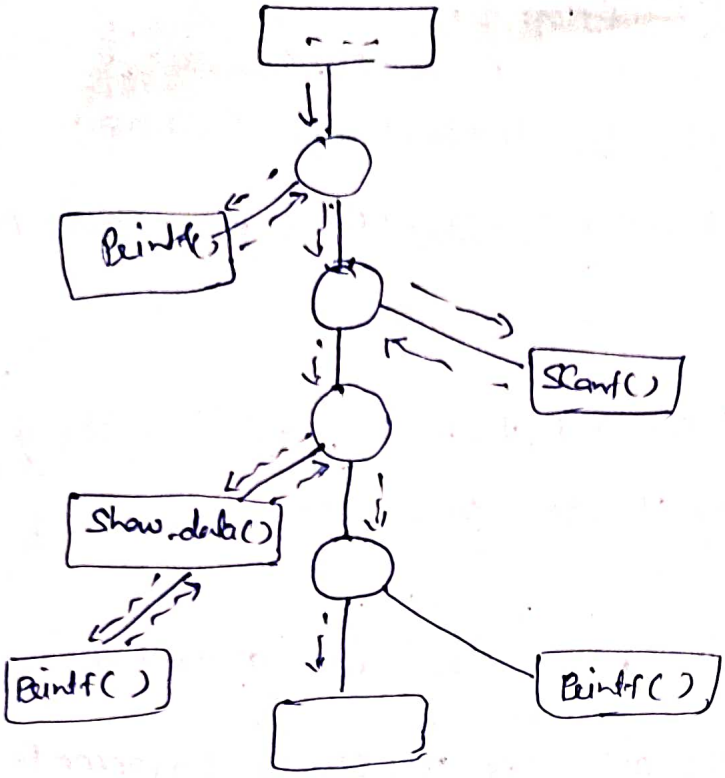
Whenever a Procedure Executed, its activation Record is stored on the stack. also known as Control stack.

The Program Control flow in a sequential flow in a sequential manner. when a Procedure is called its Control is transferred to the called Procedure. when a called Procedure is Executed, it it returns the control back to the ~~earlier~~ caller.

This type of control flow makes it easier to represent a series of activations in the form of tree known as activation-Tree.

```
Printf( Enter your name );
scanf("%s", username);
show_data(username);
Printf("Press any key to continue...");

int show_data(char *user)
{
  Printf("Your name is %s", username);
  return; }
```


Printf()
Scanf()
Show_data()
Printf()
Printf()

Need of Back Patching

① During the generation of 3-address code in single pass is that we may not know the address (label) where program control should go.

② A no of jumping statement is generated with the target label or address is tempararily left unspecified.

③ Backpatching – is the process of filling up unspecified information of labels using appropriate semantic actions during the code generation process.

Ex Using Backpatching, generate an Intermediate Code for the following expression.

A<B OR C<D AND P<Q

S1;

① if A<B goto (7) (true)
② goto (false)
③ if C<D go to 5 (true)
④ goto (false)
⑤ if P<Q goto (7) (true)
⑥ go to (false)
⑦ Next statement

AND has higher precedence than OR statement