Lec-33 Symbol Table in Compiler Design | Attributes of Symbol Table in Hindi

Press Esc to exit full screen

Symbol Table

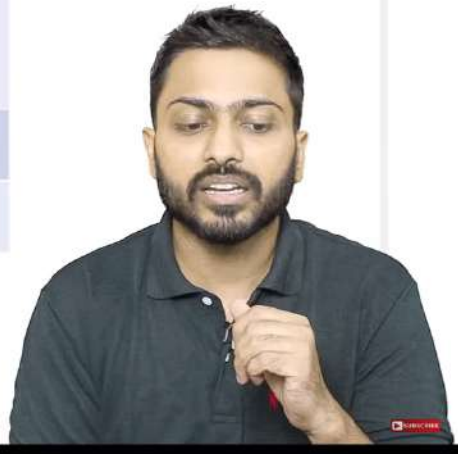
Name	Type	Size	Dimension	Line of Declaration	Line of Usage	Address
x	int	4	0	→ 4	6, 10	—
a	char	5	1	3	.....	—

```
1  main()
2  {
3  char a[5];
   int x;
   .
   .
   .x
   :
   x
```

*(a+i)

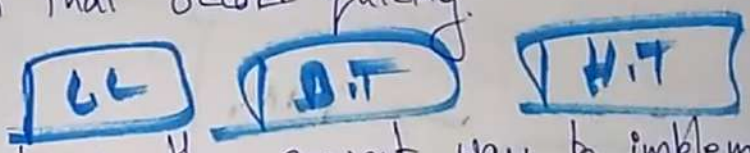
7:45 / 8:09 • Attributes of Symbol table ›

Data Structure	Insertion	Lookup Time
Unordered Array	O(1)	O(n)
Ordered Array	O(n)	O(log n)
Unordered Linked List	O(1)	O(n)
Ordered Linked Link	O(n)	O(n)
Search Tree	O(log n)	O(log n)
Hash Table	O(1)	O(1)



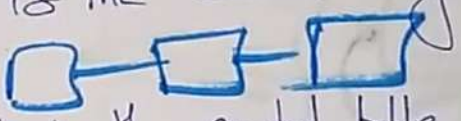
Symbol table Structure | Compiler Design | Lec-44 | Bhanu Priya

⇒ The data structure should be designed to allow the compiler to find the record for each name quickly and to store or retrieve data from that record quickly.



① Linked list :-

→ A linear list of records is the easiest way to implement symbol table.



→ The new names are added to the symbol table in the order they arrive.

→ Whenever a new name is to be added first it checks searched linear to check if the name already present in table or not.

Symbol table Structure | Compiler Design | Lec-44 | Bhanu Priya

Time Complexity - O(n)

Advantage - less space, additions are simple

Disadvantage - higher access time

② Binary trees

⇒ Efficient approach for symbol table organization.

→ We add two links left & right in each record in the search tree.

→ when ever a name is to be added, first search in the tree, if it does not exists then a record for new name is created & added at proper position.

→ This has alphabetical accessibility.

2:46 / 5:56

Symbol table Structure | Compiler Design | Lec-44 | Bhanu Priya

③ Hash table

⇒ In hashing scheme two tables are maintained.
- a hash table and a symbol table

⇒ A hash table is an array with index range: 0 to tablesize-1. These entries are pointer pointing to names of symbol table.

⇒ To search for a new name we use hash function that will result in any integer between 0 to table size-1.

⇒ Insertion & lookup can be made very fast - O(1).

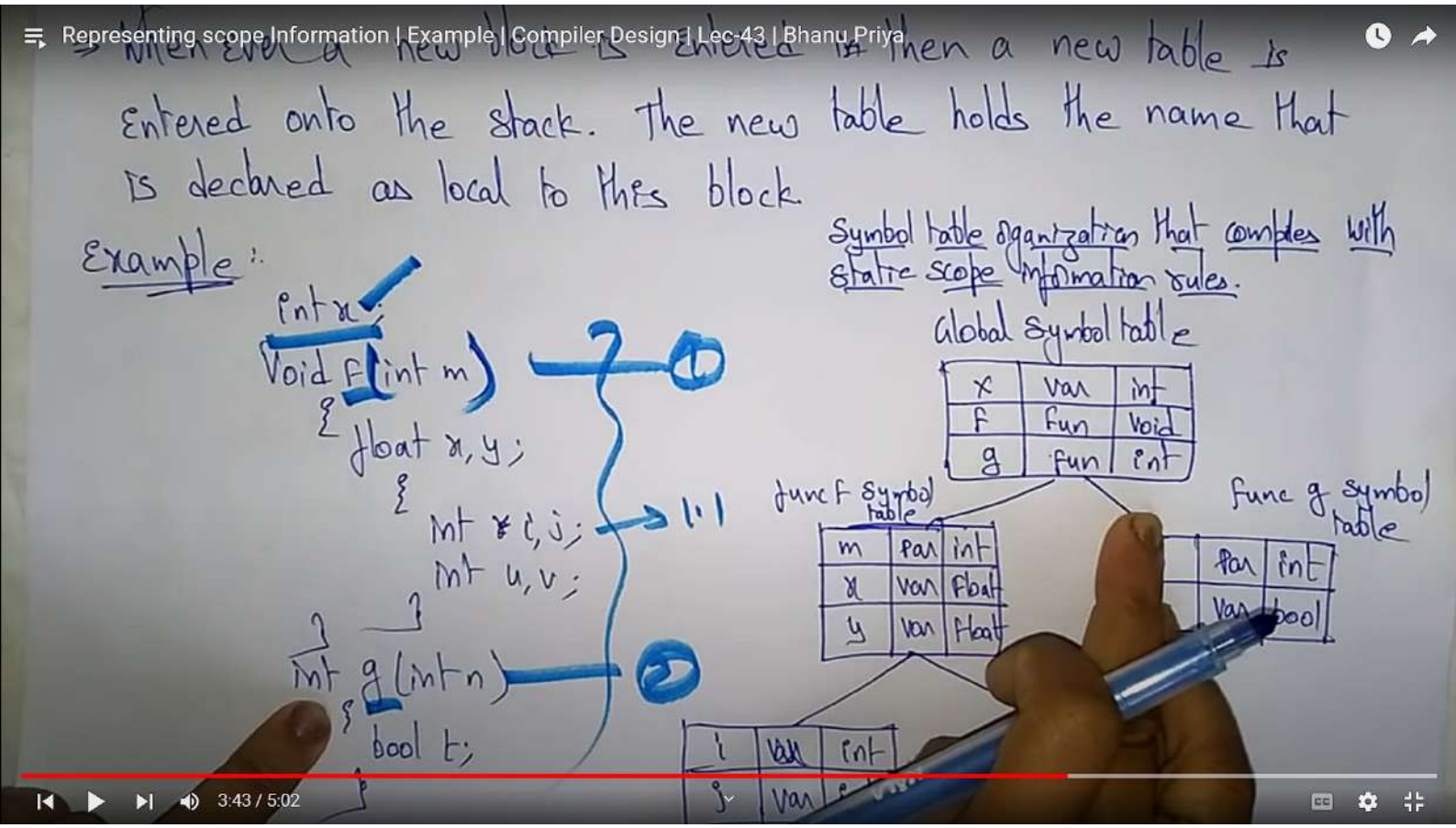
4:58 / 5:56

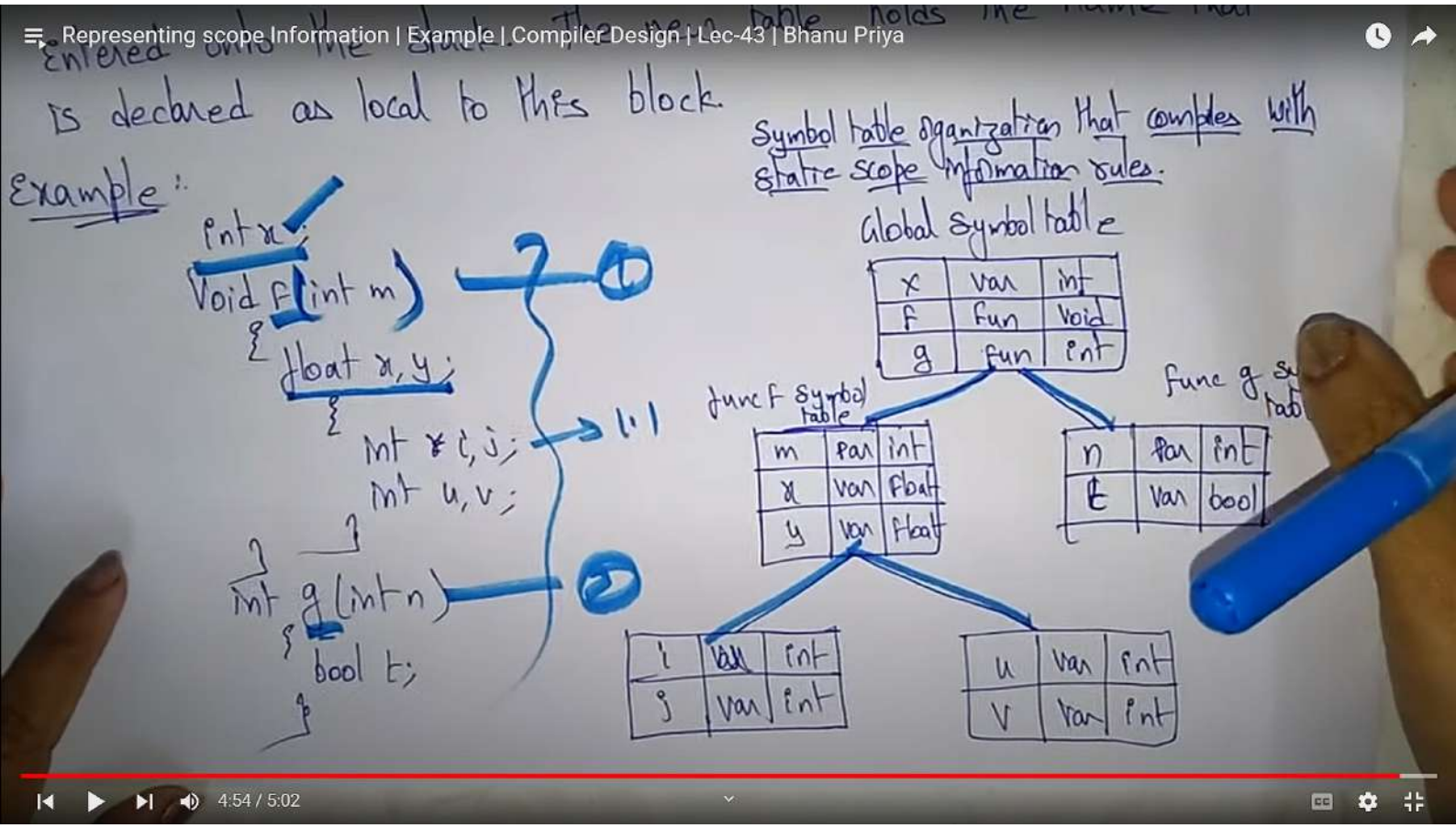
Representing scope information

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

⇒ The rules in a block-structured language are as follows:

1. If a name declared within block B, then it is will be valid only within B.
2. If B_1 is nested within B_2 then the name that is valid for B_2 is also valid for B_1 unless the name's identifier is redeclared in B_1.
3. The scope rules need a more complicated organization of symbol table than a list of associations between names and attribut


Representing scope Information | Example | Compiler Design | Lec-43 | Bhanu Priya
Then a new table is
Entered onto the stack. The new table holds the name that
is declared as local to this block.
Example :
Symbol table organization that complies with
static scope information rules.
Global Symbol table
x var int
f fun void
g fun int
int x;
Void f(int m)
{
float x, y;
{
int i, j;
int u, v;
}
int g(int n)
{
bool t;
}
Func f symbol table
m par int
x var float
y var float
Func g symbol table
par int
var bool
i
j
3:43 / 5:02


Representing scope Information | Example | Compiler Design | Lec-43 | Bhanu Priya
is declared as local to this block.
Example :
int x;
Void f(int m)
{
float x, y;
{
int i, j;
int u, v;
}
int g(int n)
{
bool t;
}
Symbol table organization that complies with static scope information rules.
Global Symbol table
x var int
f fun void
g fun int
func f Symbol table
m par int
x var float
y var float
i var int
j var int
u var int
v var int
func g
n par int
t var bool
4:54 / 5:02

Storage organisation | Runtime memory | Compiler Design | Lec- 47 | Bhanu Priya

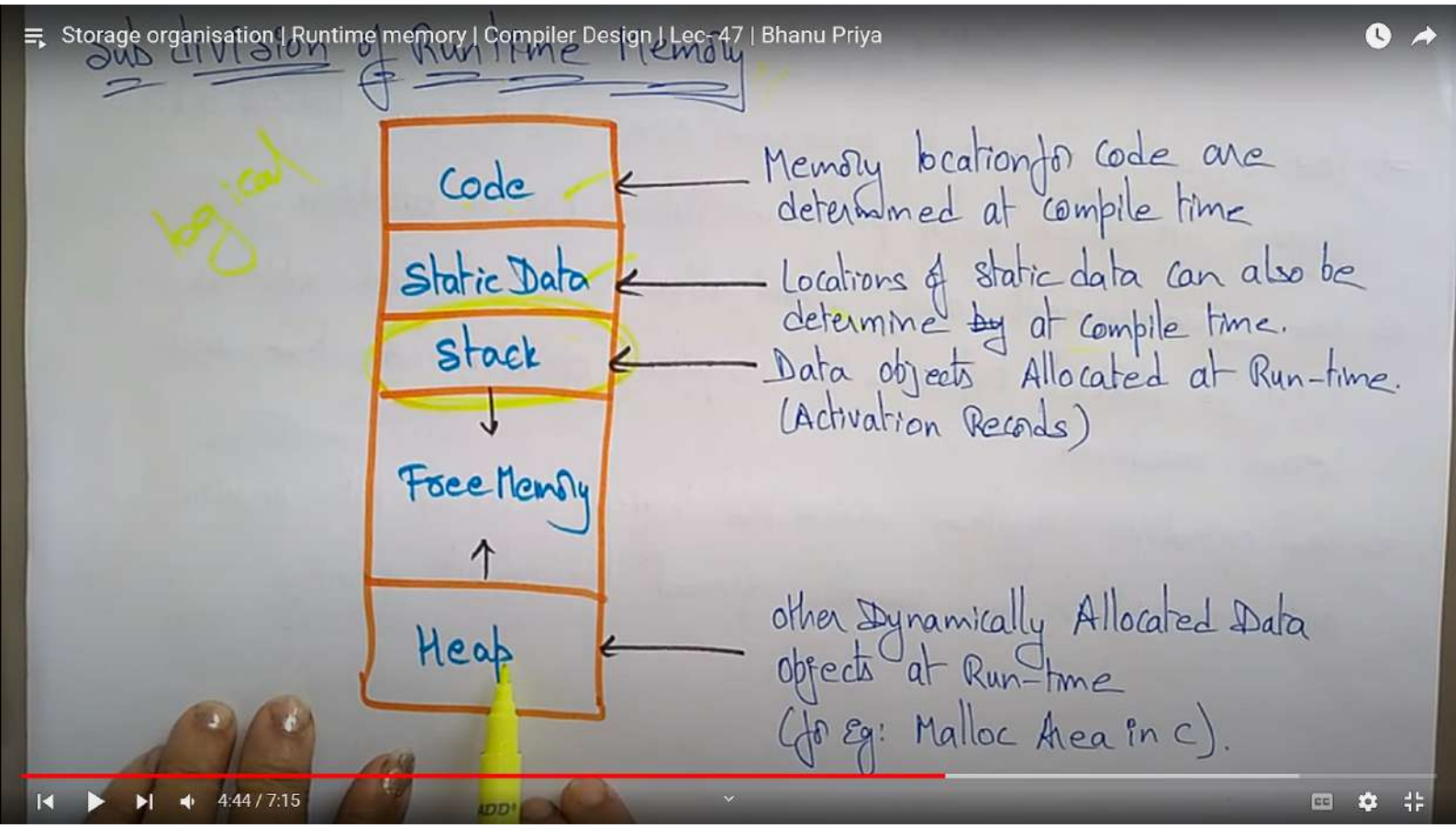
Storage Organisation

⇒ The executing target program runs in its own logical address space in which each program value has a location.

⇒ The management and organization of this logical address space is shared between the compiler, operating system and target machine.

⇒ The operating system maps the logical address into physical addresses, which are usually spread throughout memory.

2:20 / 7:15


Storage organisation | Runtime memory | Compiler Design | Lec-47 | Bhanu Priya
Sub division of Runtime Memory
logical
Code
Static Data
Stack
Free Memory
Heap
Memory location for code are determined at compile time
Locations of static data can also be determine by at compile time.
Data objects Allocated at Run-time. (Activation Records)
other Dynamically Allocated Data objects at Run-time (for Eg: Malloc Area in C).
4:44 / 7:15

Storage organisation | Runtime memory | Compiler Design | Lec- 47 | Bhanu Priya

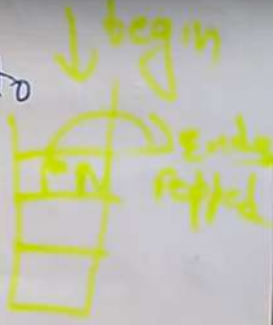
- Runtime storage comes into blocks, where a byte is used to show the smallest unit of addressable memory. Using the four bytes a machine word can form.
- object of multibyte is stored in consecutive bytes and gives the first byte address
- Runtime storage can be subdivide to hold the different components of an executing program:
  1. Generated executable code
  2. static data objects
  3. Dynamic data object - heap
  4. Automatic data objects stack

7:08 / 7:15

Activation record | Compiler Design | Lec-48 | Bhanu Priya

Activation Record

⇒ Control stack is a runtime stack which is used to keep track of the live procedure activations,
i.e., It is used to find out the procedures whose execution have not been completed.



⇒ When the activation begins then the procedure name will push on to the stack and when returns (activation ends) then it will popped.

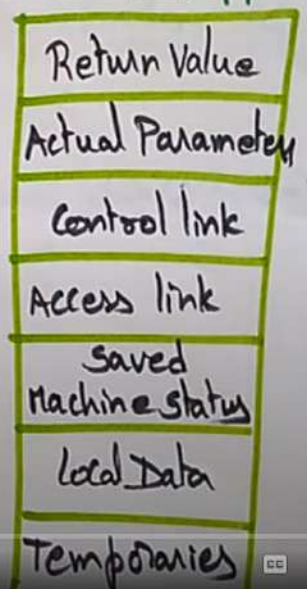
⇒ Activation record is used to manage the information needed by a single execution of a procedure.

2:26 / 6:44

Activation record | Compiler Design | Lec-48 | Bhanu Priya

⇒ An Activation record is pushed into the stack when a procedure is called, and is popped when the control returns to the caller function.

calling popped.



✱ The Contents of Activation records:

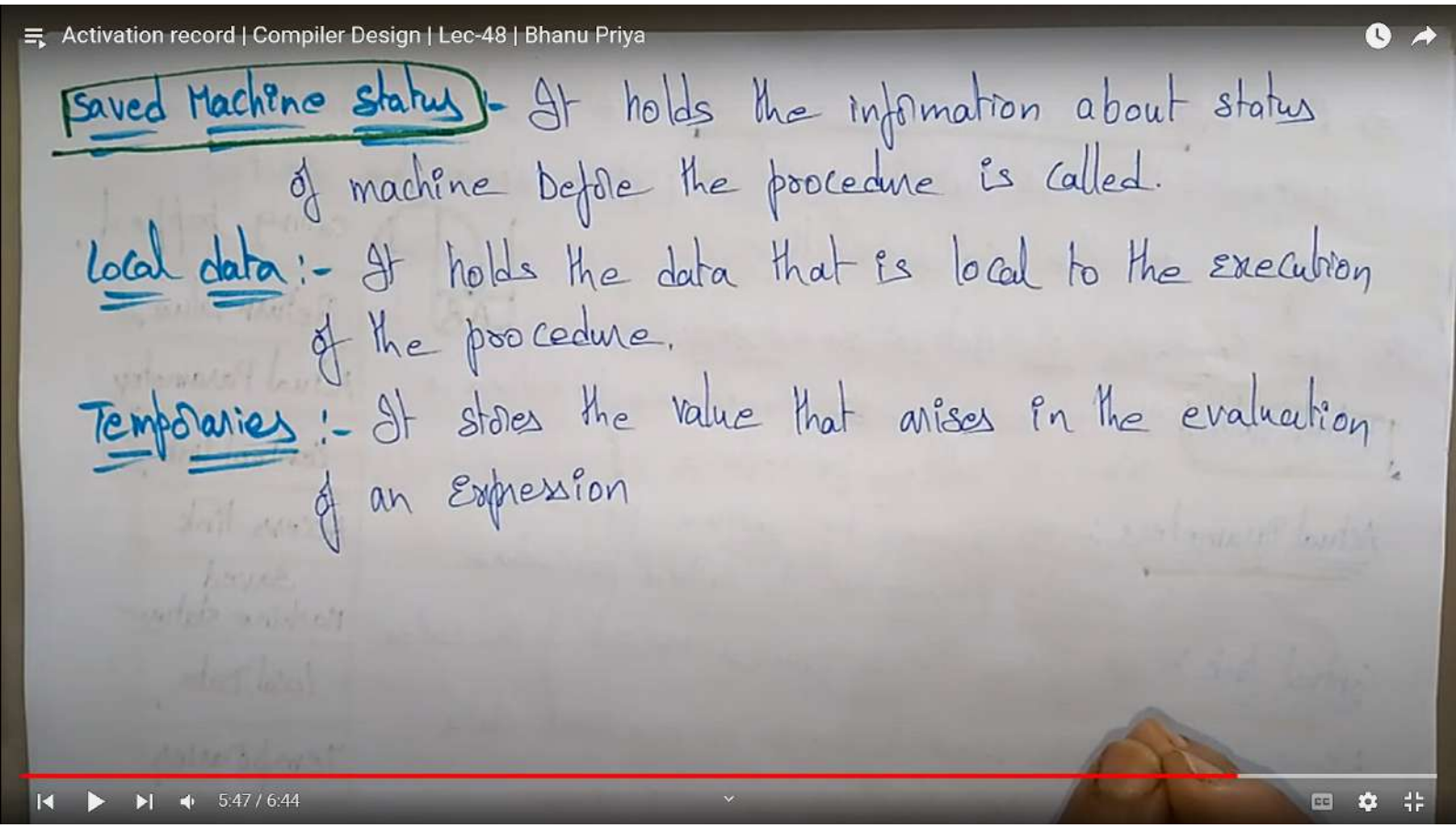
Return Value:- It is used by called procedure to return a value to calling procedure.

Actual Parameters:- It is used by calling procedure to supply parameters to called procedure.

Control link:- It points to activation record of the caller.

Access link:- It is used to refer to non local data held in other activation records.

3:28 / 6:44


Activation record | Compiler Design | Lec-48 | Bhanu Priya
Saved Machine status :- It holds the information about status of machine before the procedure is called.
Local data :- It holds the data that is local to the execution of the procedure.
Temporaries :- It stores the value that arises in the evaluation of an expression
5:47 / 6:44

Storage allocation | Static, Stack& Heap | Compiler Design | Lec- 49 | Bhanu Priya

Storage Allocation

The different ways to allocate Memory are :-

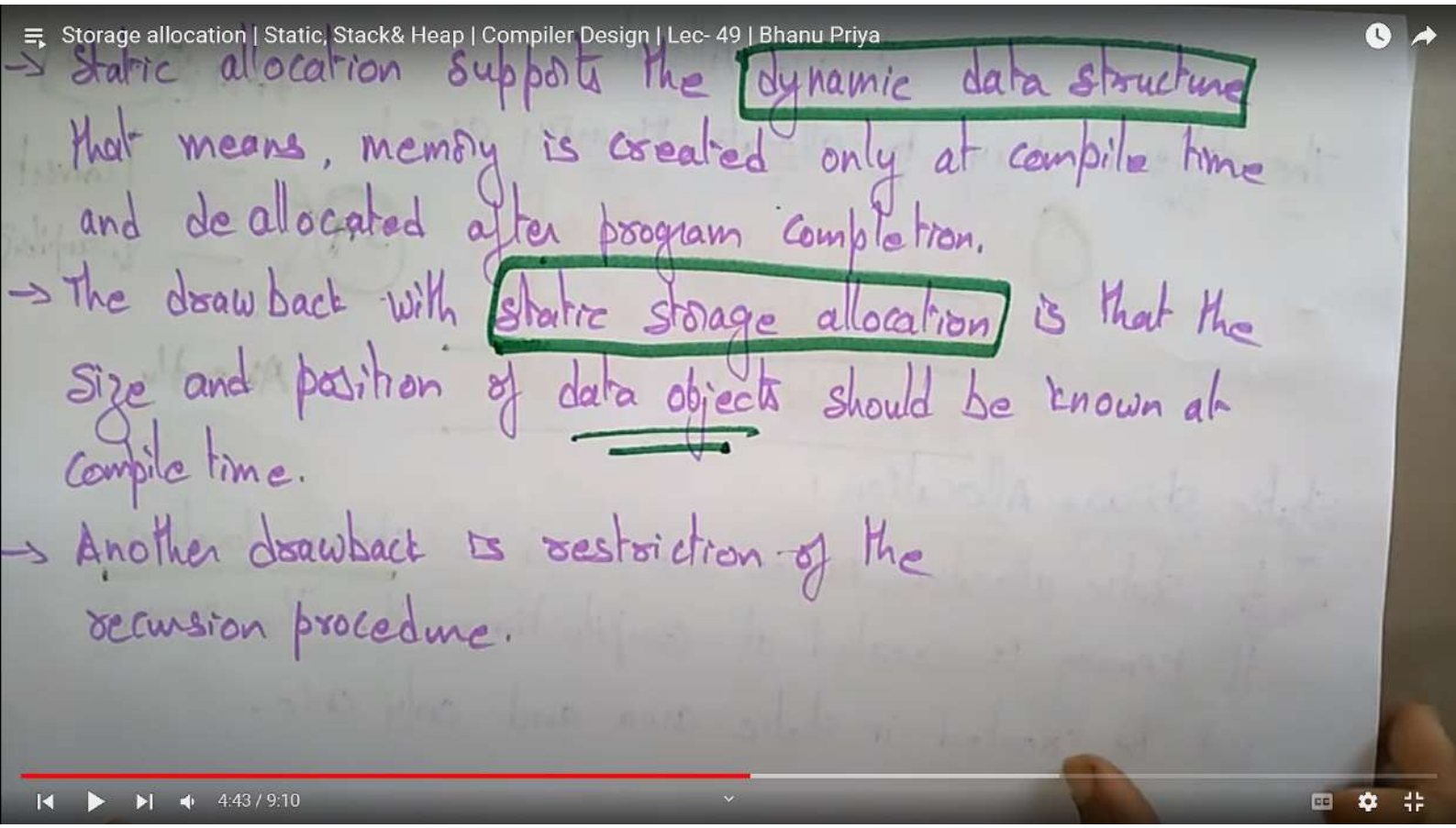
1. Static Storage Allocation
2. Stack Storage Allocation
3. Heap Storage Allocation

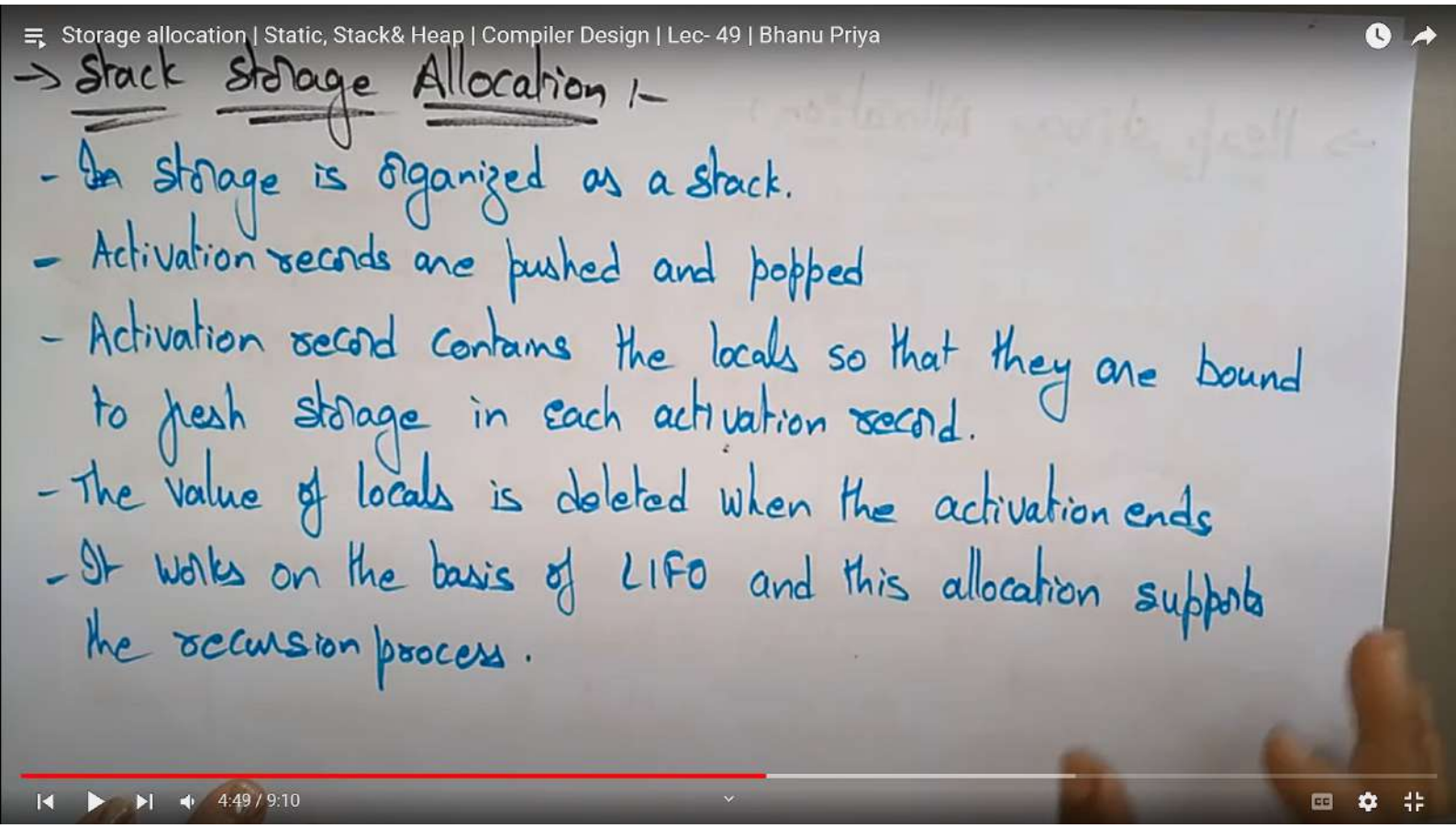
Static Storage Allocation :-

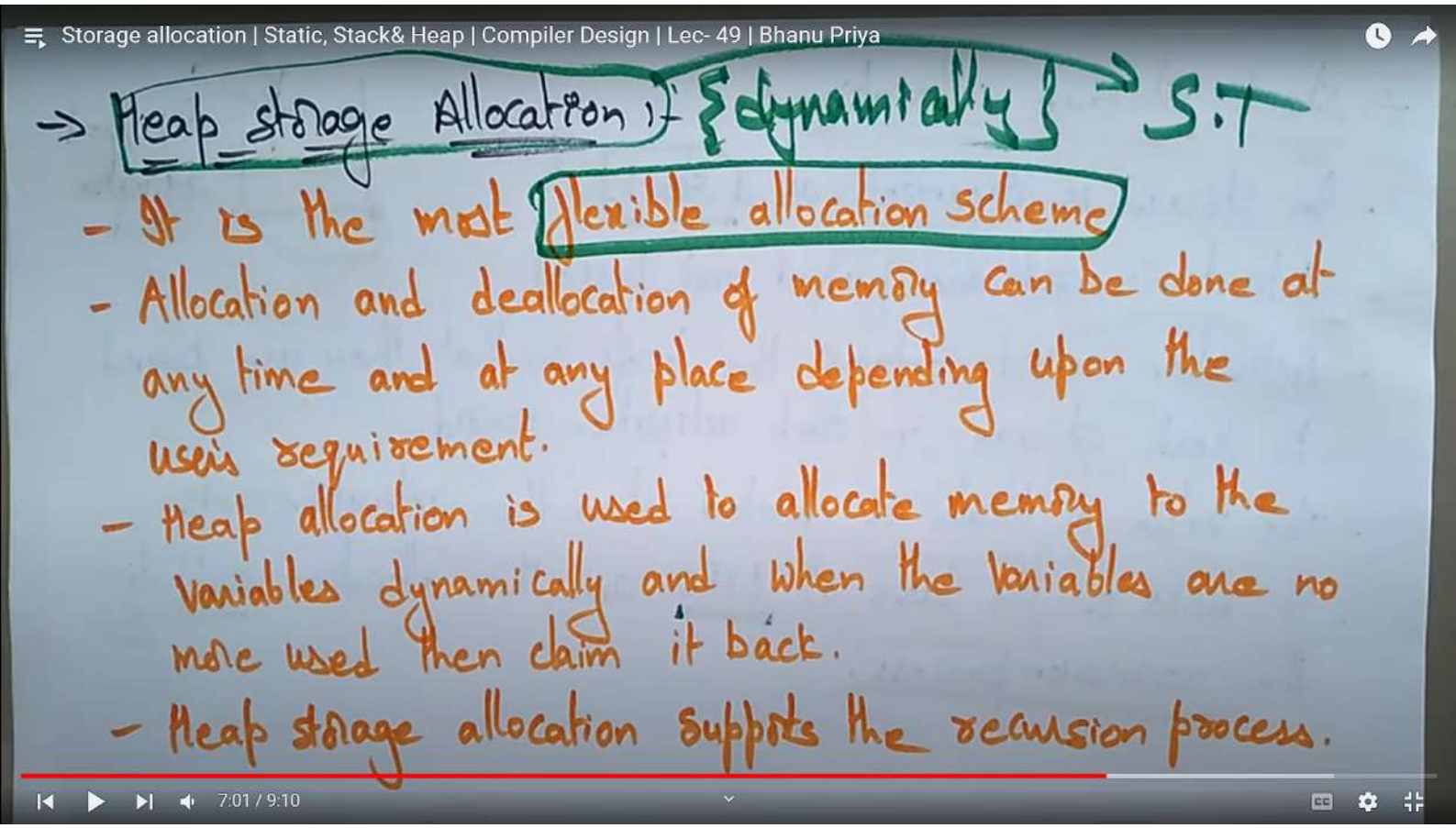
→ In static allocation, names are bound to storage locations.

→ If Memory is created at compile time then the memory will be created in static area and only once.

0:23 / 9:10


Storage allocation | Static, Stack& Heap | Compiler Design | Lec- 49 | Bhanu Priya
→ Static allocation supports the dynamic data structure that means, memory is created only at compile time and de allocated after program completion.
→ The drawback with static storage allocation is that the size and position of data objects should be known at compile time.
→ Another drawback is restriction of the recursion procedure.
4:43 / 9:10


Storage allocation | Static, Stack& Heap | Compiler Design | Lec- 49 | Bhanu Priya
→ Stack Storage Allocation :-
- Storage is organized as a stack.
- Activation records are pushed and popped
- Activation record contains the locals so that they are bound to fresh storage in each activation record.
- The value of locals is deleted when the activation ends
- It works on the basis of LIFO and this allocation supports the recursion process.
4:49 / 9:10


Storage allocation | Static, Stack& Heap | Compiler Design | Lec- 49 | Bhanu Priya
→ Heap storage Allocation : {dynamically} → S.T
- It is the most flexible allocation scheme
- Allocation and deallocation of memory can be done at any time and at any place depending upon the user's requirement.
- Heap allocation is used to allocate memory to the variables dynamically and when the variables are no more used then claim it back.
- Heap storage allocation supports the recursion process.
7:01 / 9:10