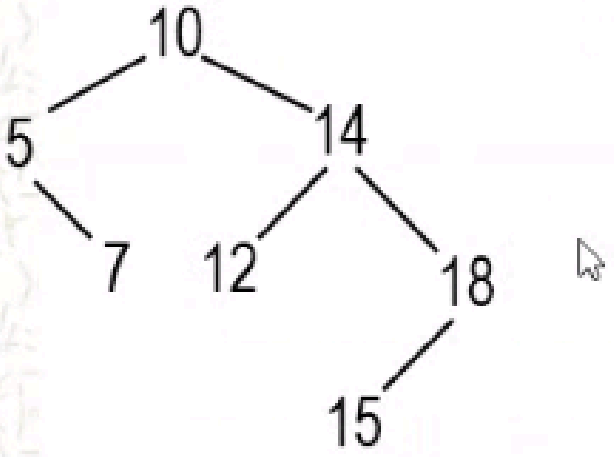
CIS 2101

Binary Search Tree (BST)

Binary Search Tree (BST)

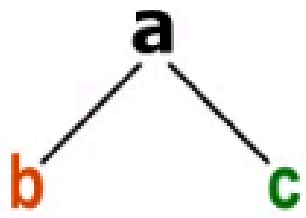
SET A = {10, 5, 14, 12, 7, 18, 15}



BST - Characteristics

BST property

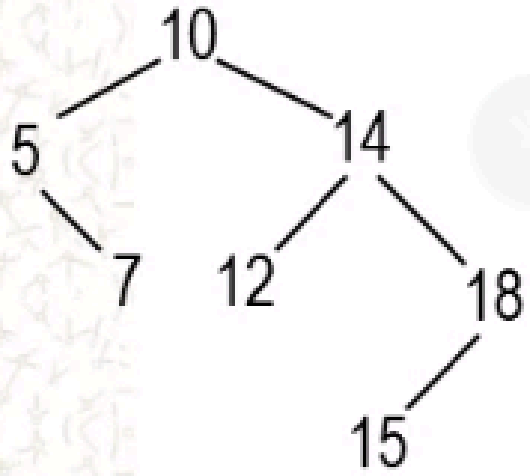
" All elements stored in the **left subtree** of any node **x**
are all **less than** the element stored at **x**, and
all elements stored in the **right subtree** of **x**
are **greater than** the element stored at **x**"



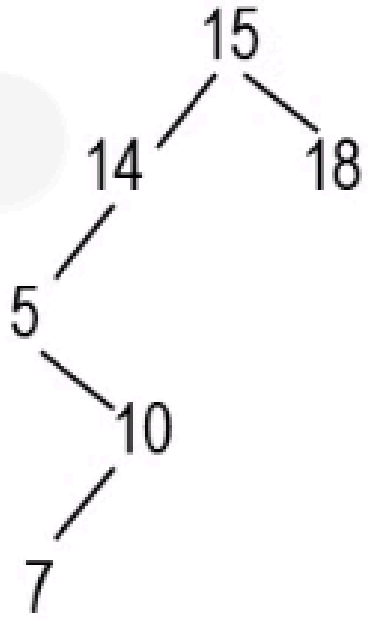
b < a

BST – Sets A&B

SET A = {10, 5, 14, 12, 7, 18, 15}



SET B = {15, 14, 5, 10, 18, 7, 12}



Exercise

1. Describe the BST that will be created if the elements to be inserted in an initially empty BST is sorted in ascending order ? Descending order ?
2. What is the running time of operation Member for the BST produced in #1 ?
3. Write a recursive and NON-recursive functions for insert and delete
4. What is an AVL tree ?
5. Differentiate BST from Binary Search.
6. Write the code of the 3 recursive tree traversals.

BST Operations and Application

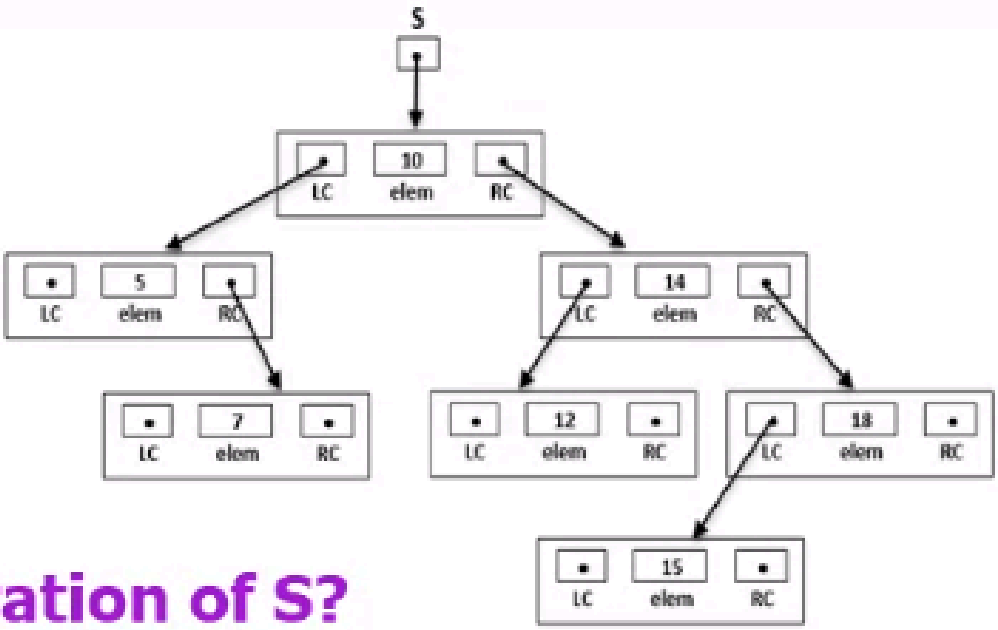
- Operations supported by BST have a running time of $O(\log_2 N)$
  1. Insert
  2. Delete
  3. Member
  4. Min
  5. Max
- **Application:**
  - BST can be use to represent the set of identifiers of a programming language like C

Pointer Implementation of BST

Definition & Declaration:

```
typedef struct cell{
   int elem;
   struct cell *LC;
   struct cell *RC;
}ctype, *BST;
BST S;
```

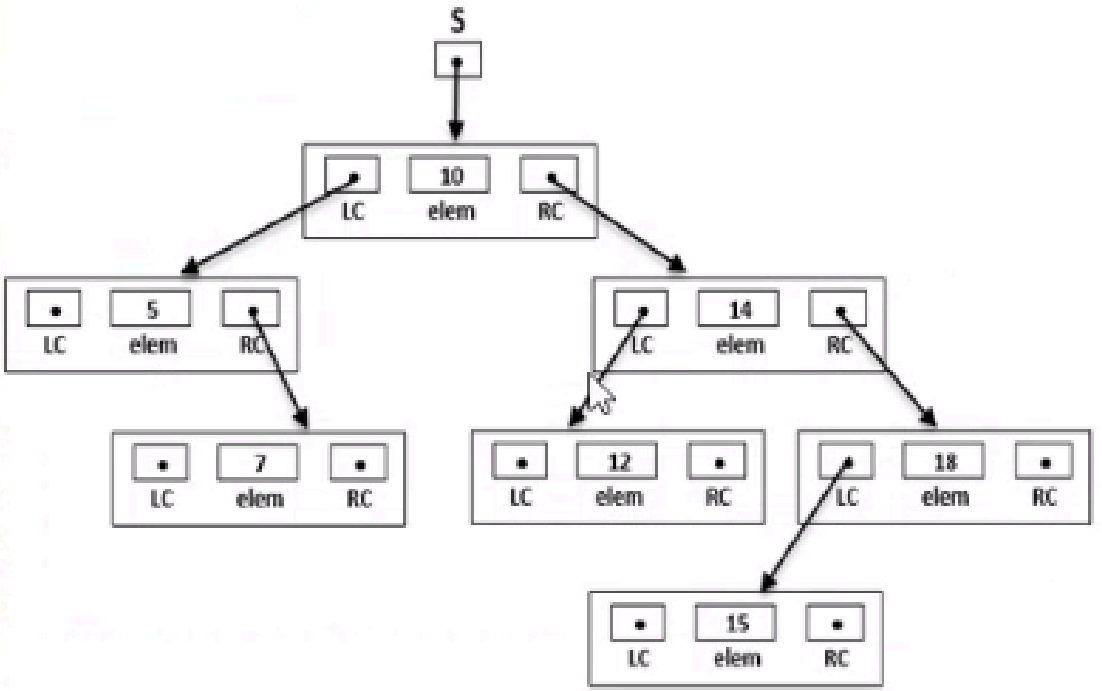
SET S = { 10, 5, 14, 12, 7, 18, 15 }



What is the declaration of S?

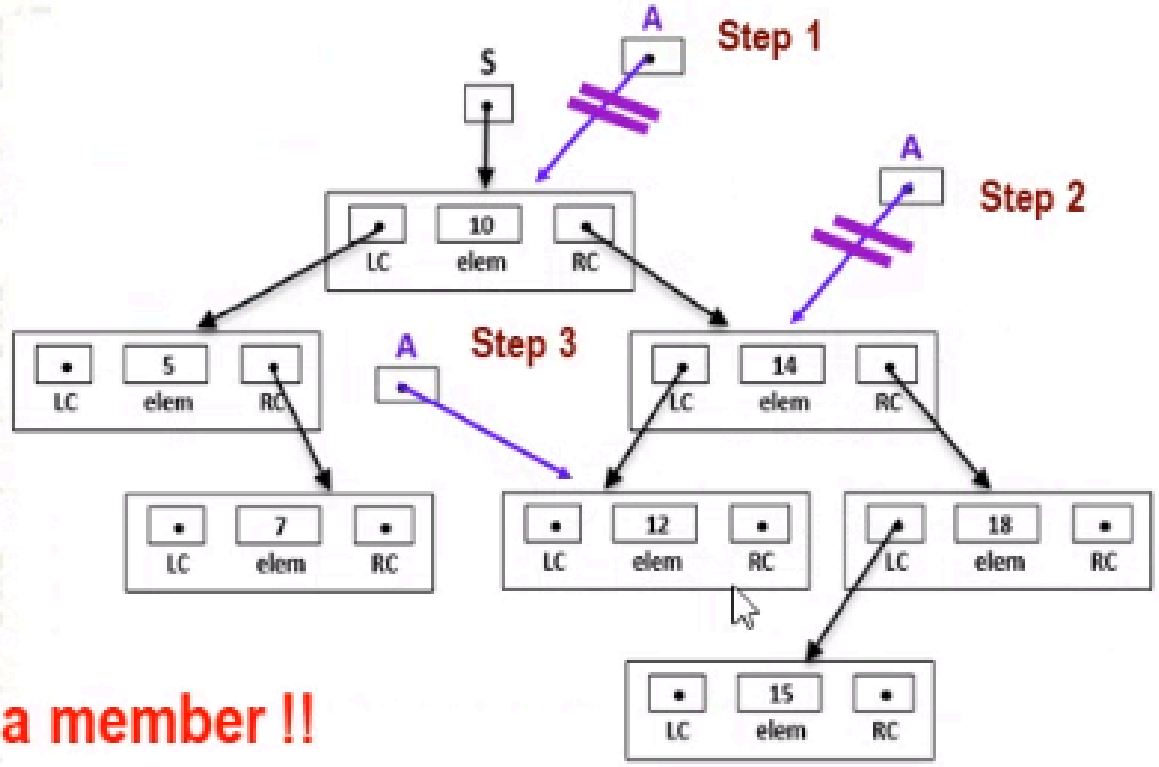
Operation: Member(12,S)

(Sample 1: Pointer to Node Approach)



Operation: Member(12,S)

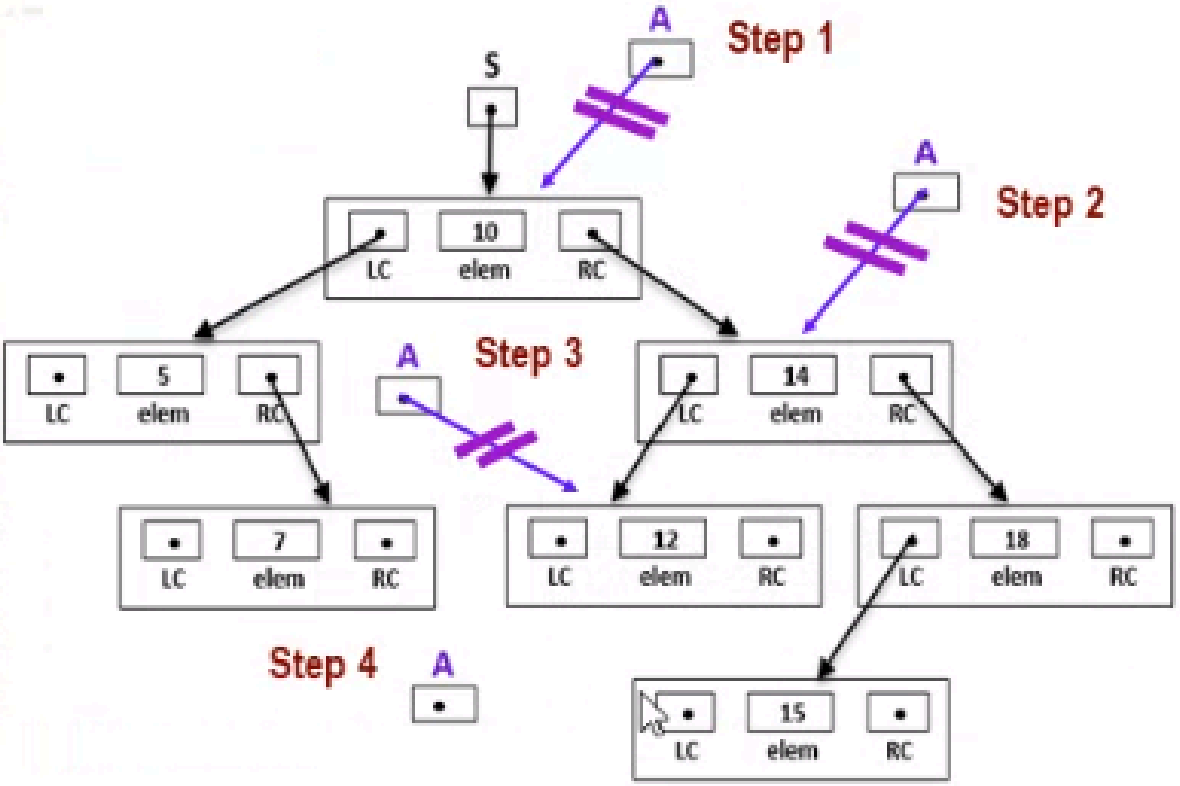
(Sample 1: Pointer to Node Approach)



12 is a member !!

Operation: Member(11,S)

(Sample 2: Pointer to Node Approach)



Recursive Member(S, x)

RECALL:

- What is a recursive function?
- Is recursion a good substitute for a loop?

Recursive Member(S, x)

RECALL:

- What is a recursive function?
- Is recursion a good substitute for a loop?
- Tips in making recursive function:
  - Make sure the function has at least 1 **exit point** or **base case** (a statement that will make the function terminate)
  - It is best to start the recursive function with exit point(s)

Non-recursive VS. Recursive

```
int Member(BST S, int x)
{
  while (S != NULL && S->elem != x) {
      if (x < S->elem) {
          S = S->LC;
      }else {
          S = S->RC;
      }
  }
  return(S == NULL) ? 0 : 1;
}
```

```
int Member(BST S, int x)
{
   if (S == NULL)
      return 0;
   else if (x == S->elem)
     return 1;
   else if (x < S->elem)
     return  Member(S->LC, x);
   else  /* x > A→elem */
     return  Member(S->RC,x);
```

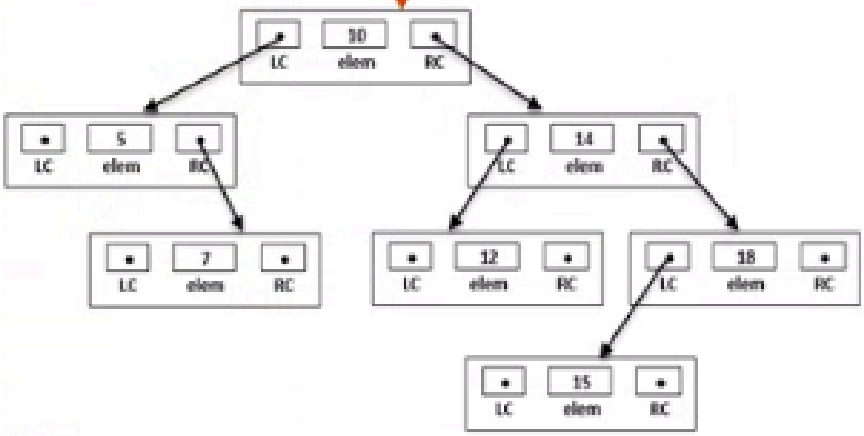
When will the loop terminate ? → Exit Points !!

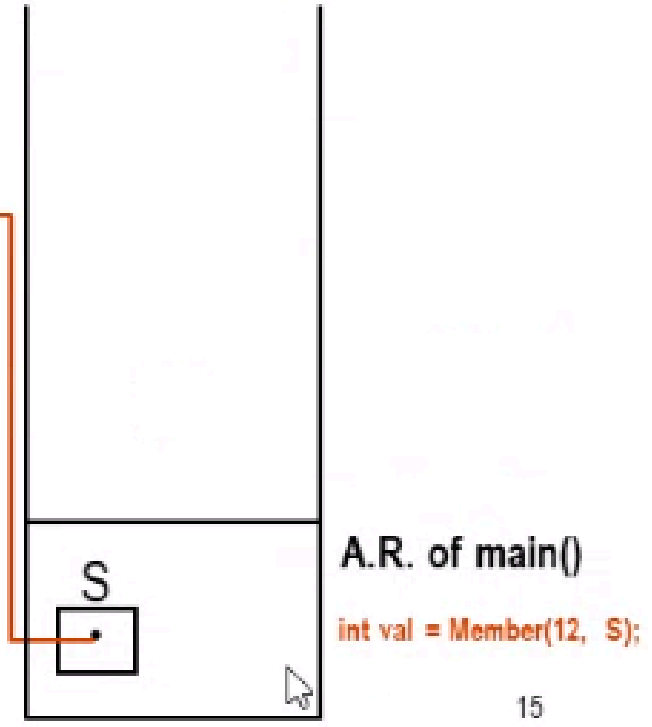
int val = Member (12,S) ?

(1st call)

Execution Stack

Heap Memory





Note: Assume function call is in main()

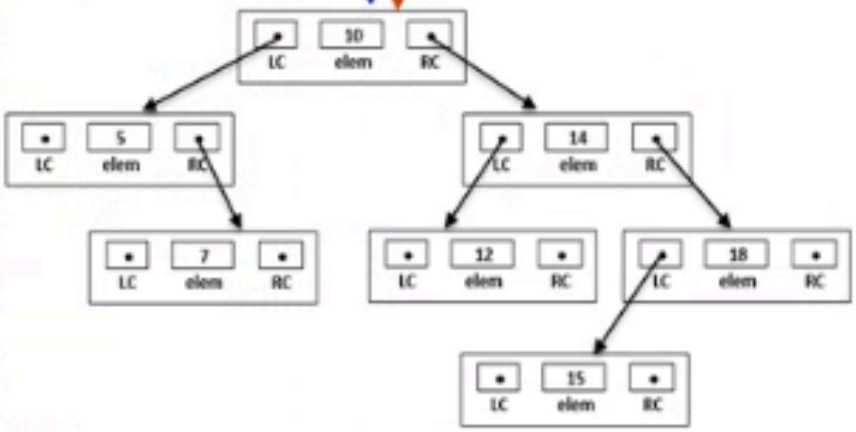
```
int Member(int x, SET A)
{
 if (A == NULL)
     return 0;
 else if (x == A→elem)
    return 1;
 else if (x < A→elem)
    return (Member(x,  A→LC));
 else  /* x > A→elem */
   return (Member(x,  A→RC));
}
```

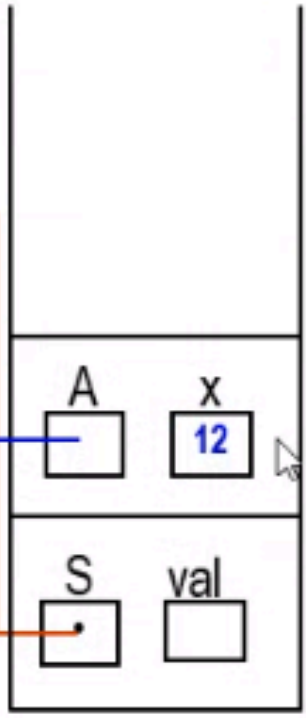
int val = Member (12,S) ?

(Executing 1st call)

Execution Stack

Heap Memory





A.R. of Member()

A.R. of main()

int val = Member(12, S);

Note: Assume function call is in main()

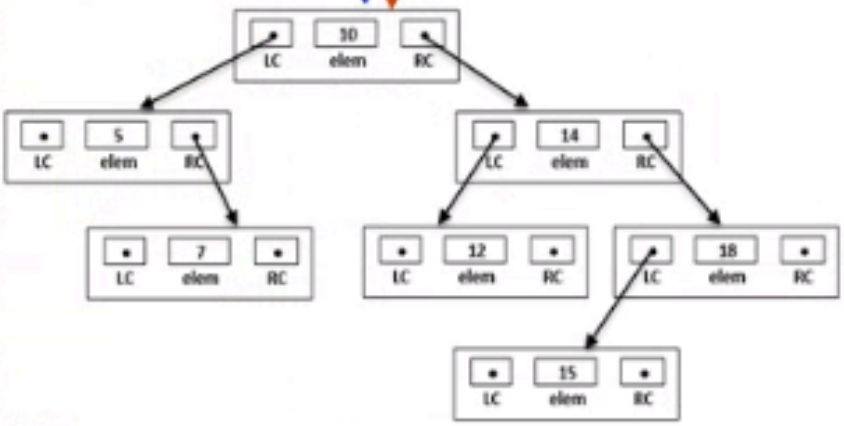
```
int Member(int x, SET A)
{
 if (A == NULL)
     return 0;
 else if (x == A→elem)
    return 1;
 else if (x < A→elem)
    return (Member(x,  A→LC));
 else  /* x > A→elem */
   return (Member(x,  A→RC));
}
```

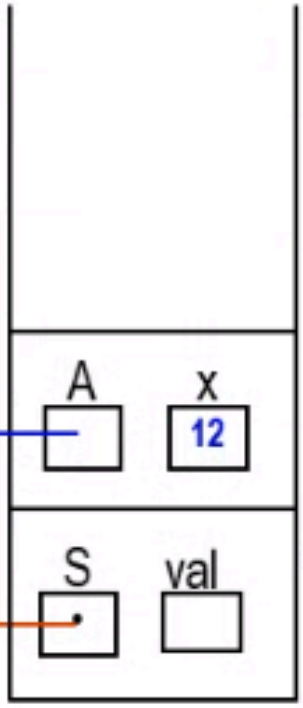
int val = Member (12,S) ?

(Executing 1st call)

Execution Stack

Heap Memory





A.R. of Member()

ret (Mem(x, A→RC));

A.R. of main()

int val = Member(12, S);

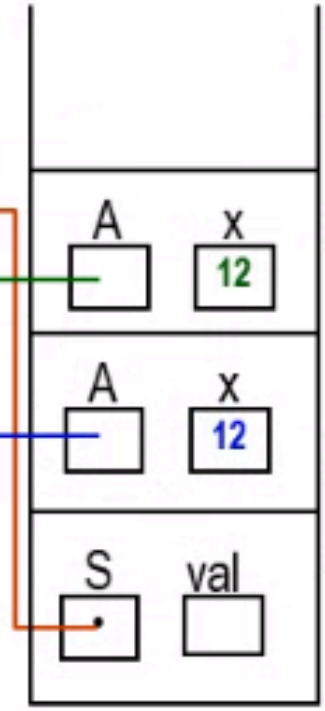
Note: Assume function call is in main()

```
int Member(int x, SET A)
{
 if (A == NULL)
     return 0;
 else if (x == A→elem)
    return 1;
 else if (x < A→elem)
    return (Member(x,  A→LC));
 else  /* x > A→elem */
   return (Member(x,  A→RC));
}
```

int val = Member (12,S) ?

(Executing 2nd call)

Execution Stack



A.R. of Member()

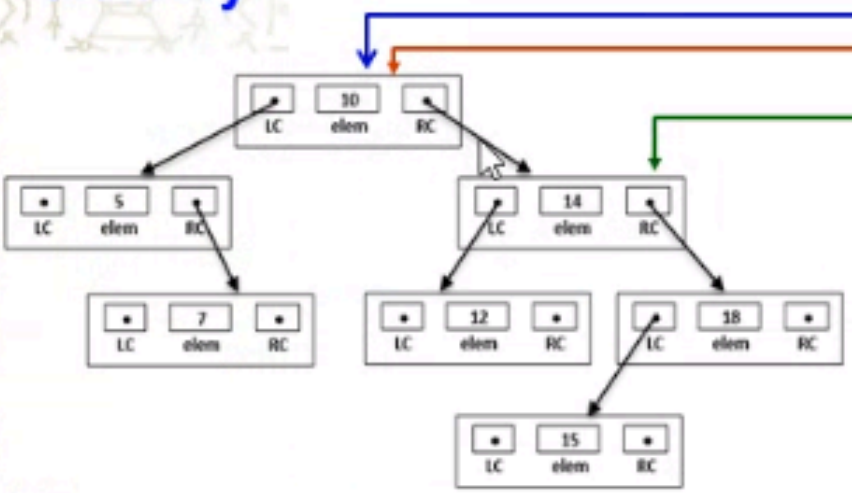
A.R. of Member()

ret (Mem(x, A→RC));

A.R. of main()

int val = Member(12, S);

Heap Memory



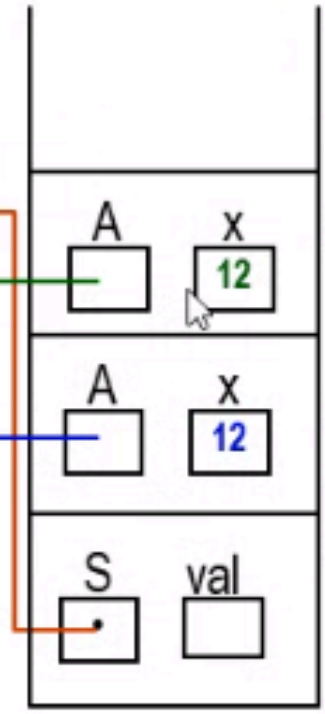
Note: Assume function call is in main()

```
int Member(int x, SET A)
{
 if (A == NULL)
     return 0;
 else if (x == A→elem)
    return 1;
 else if (x < A→elem)
    return (Member(x,  A→LC));
 else  /* x > A→elem */
    return (Member(x,  A→RC));
}
```

int val = Member (12,S) ?

(Executing 2nd call)

Execution Stack



A.R. of Member()

ret (Mem(x, A→LC));

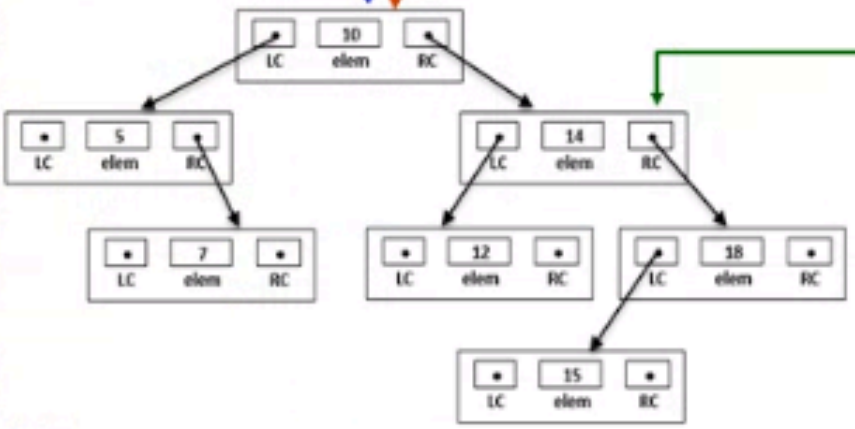
A.R. of Member()

ret (Mem(x, A→RC));

A.R. of main()

int val = Member(12, S);

Heap Memory



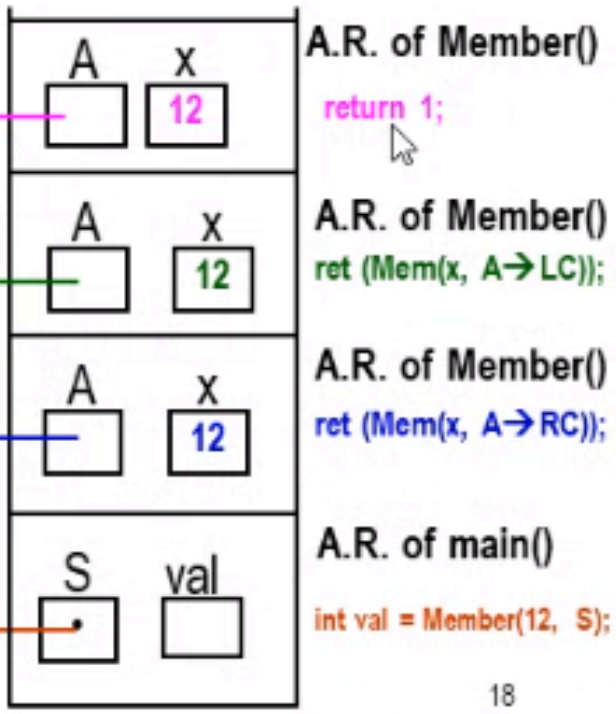
Note: Assume function call is in main()

```
int Member(int x, SET A)
{
 if (A == NULL)
     return 0;
 else if (x == A→elem)
    return 1;
 else if (x < A→elem)
    return (Member(x,  A→LC));
 else  /* x > A→elem */
   return (Member(x,  A→RC));
}
```

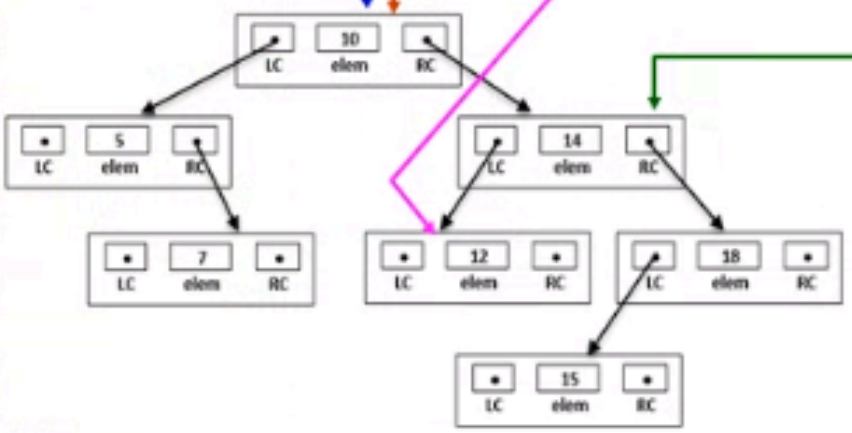
int val = Member (12,S) ?

(Executing 3rd call)

Execution Stack



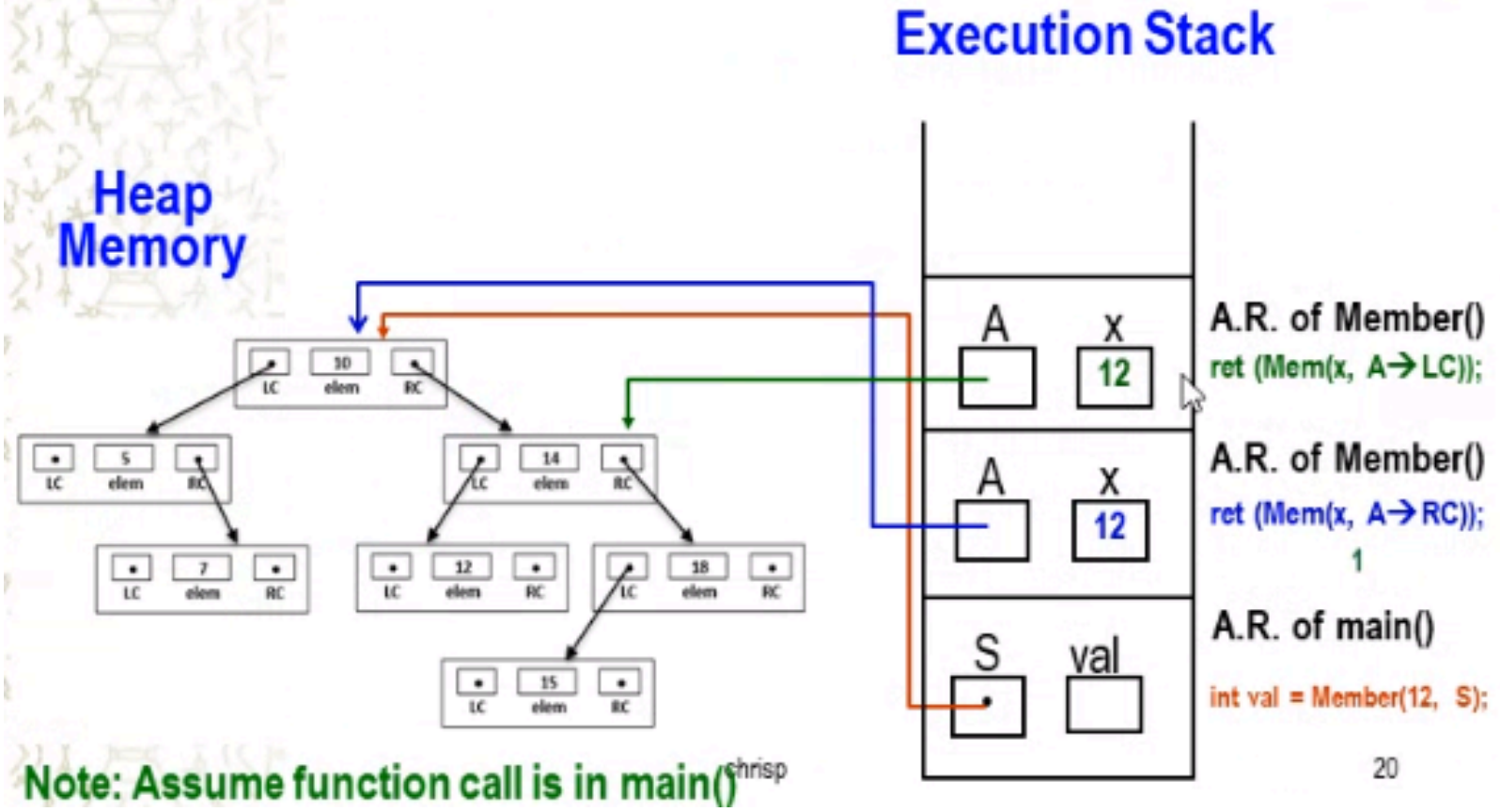
Heap Memory



Note: Assume function call is in main()

int val = Member (12,S) ?

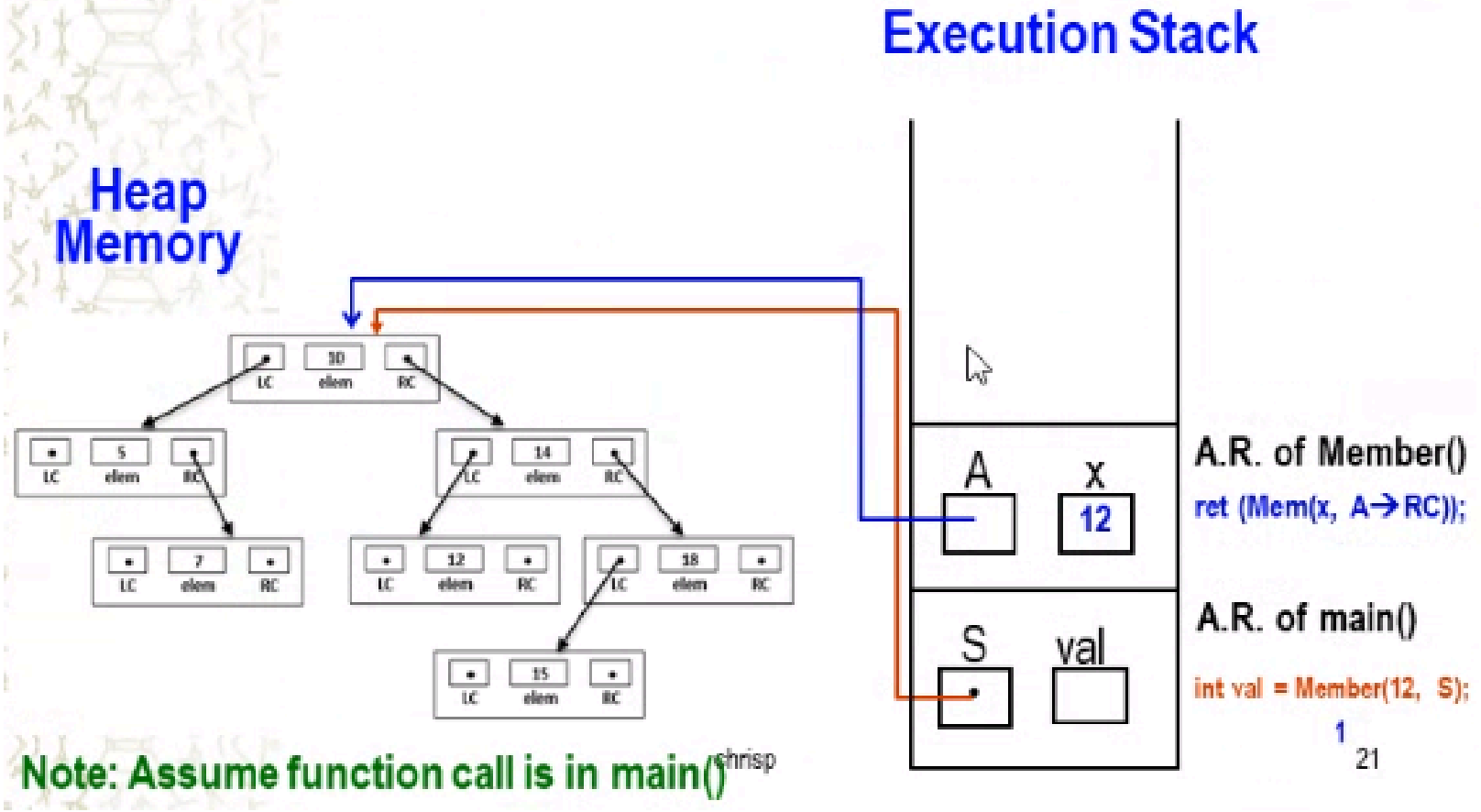
(Returning from 2nd call)



Note: Assume function call is in main()

int val = Member (12,S) ?

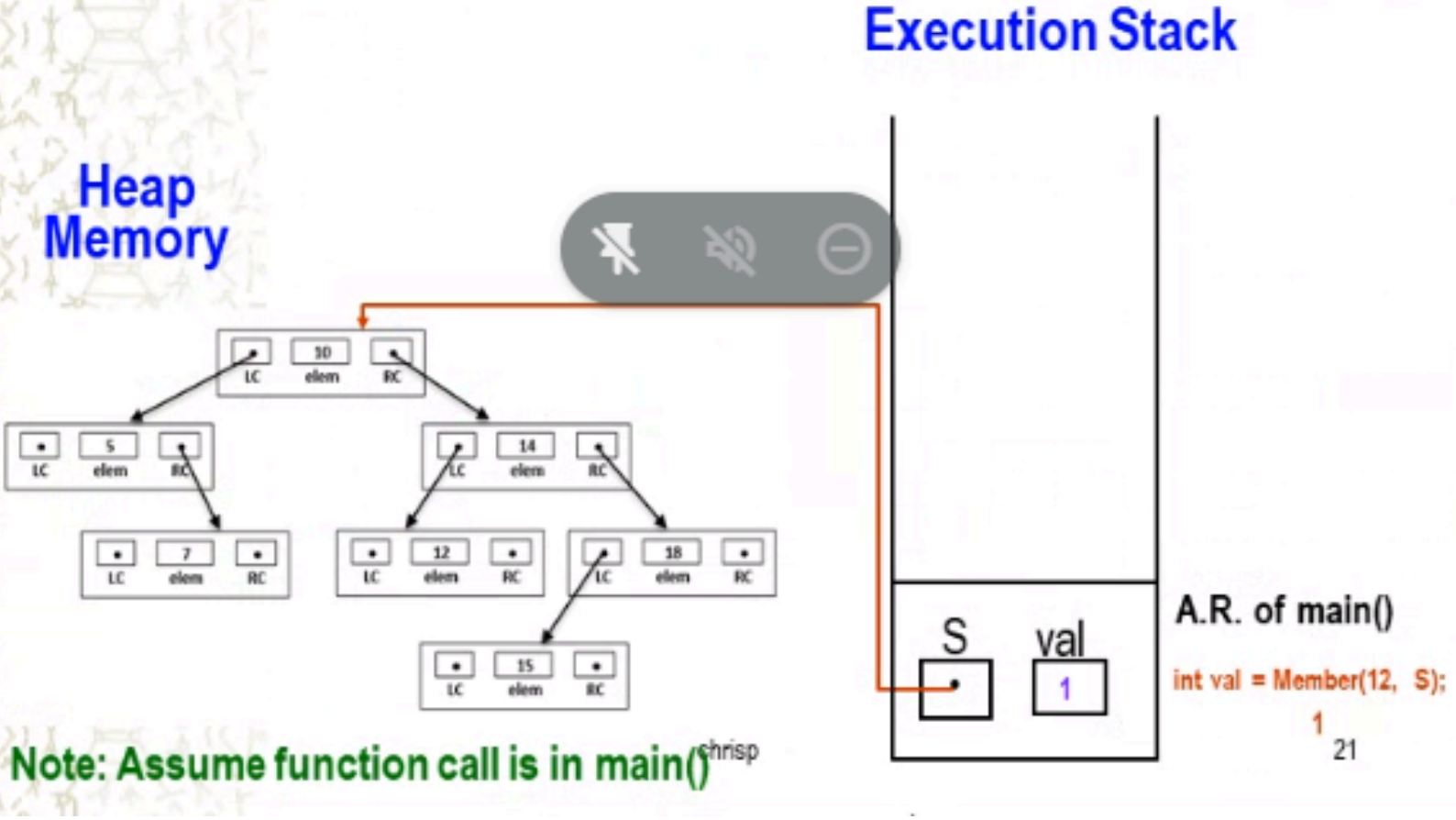
(Returning from 1st call)



Note: Assume function call is in main()

int val = Member (12,S) ?

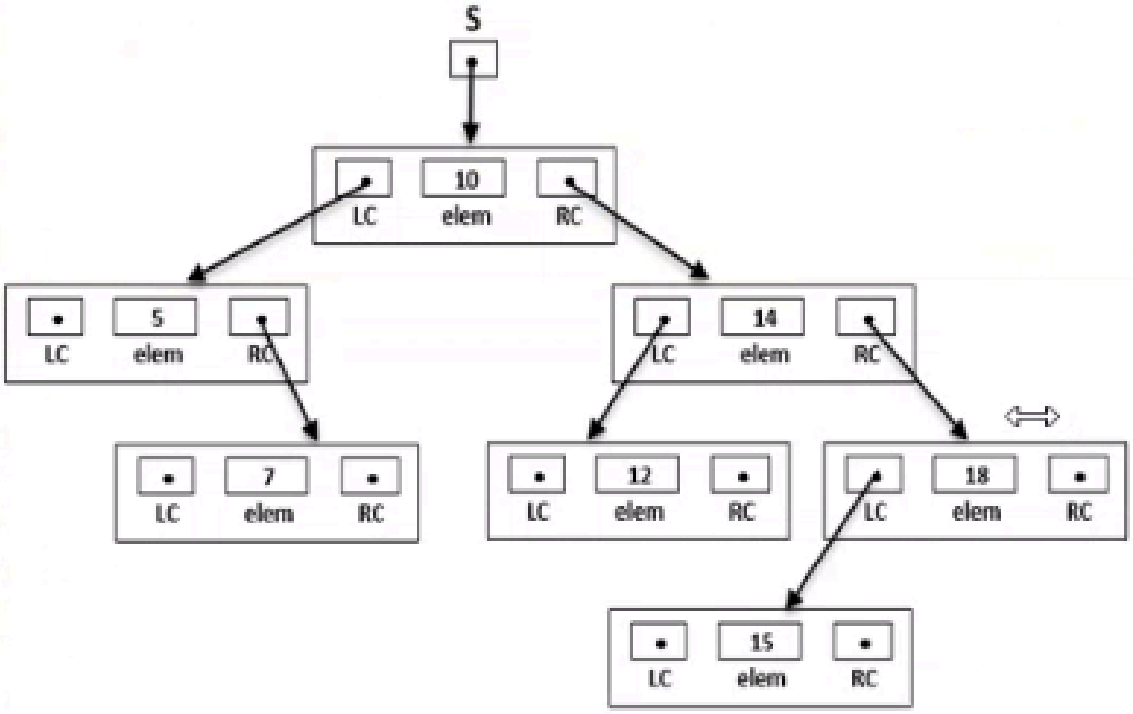
(Returning from 1st call)



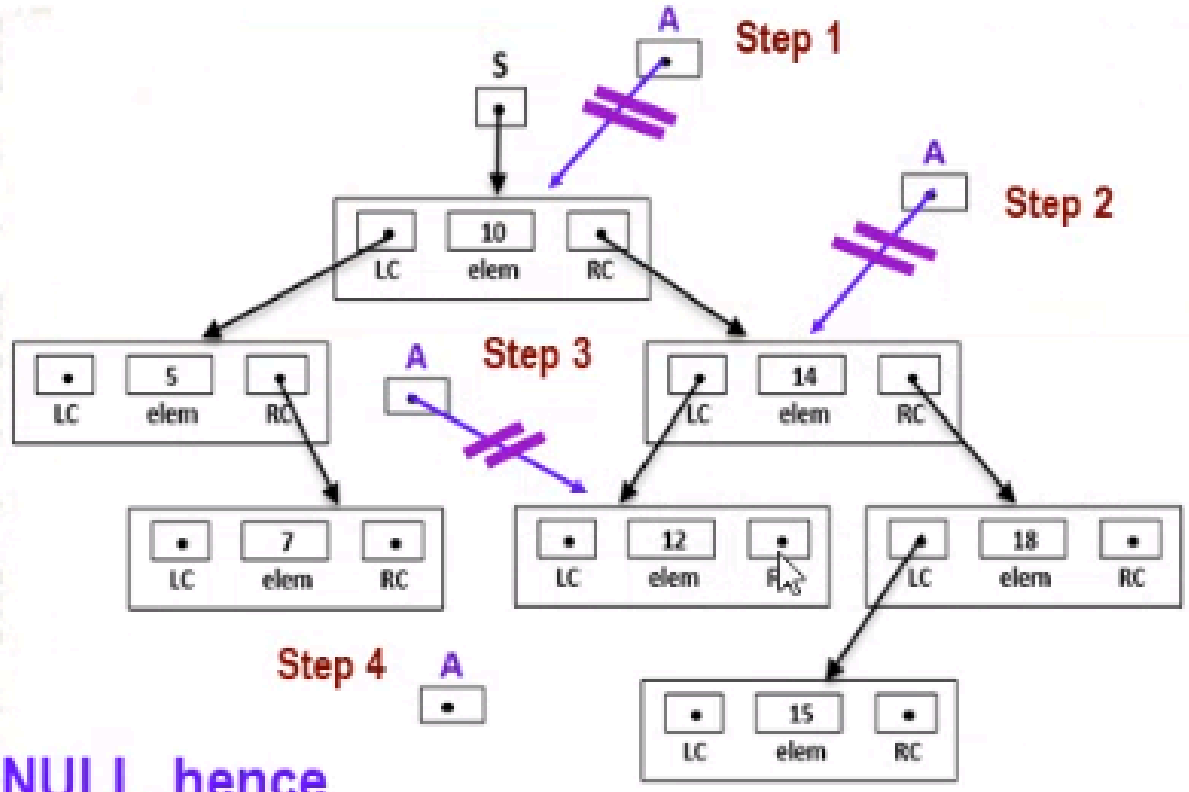
Note: Assume function call is in main()

Operation Insert

Operation: Insert 11



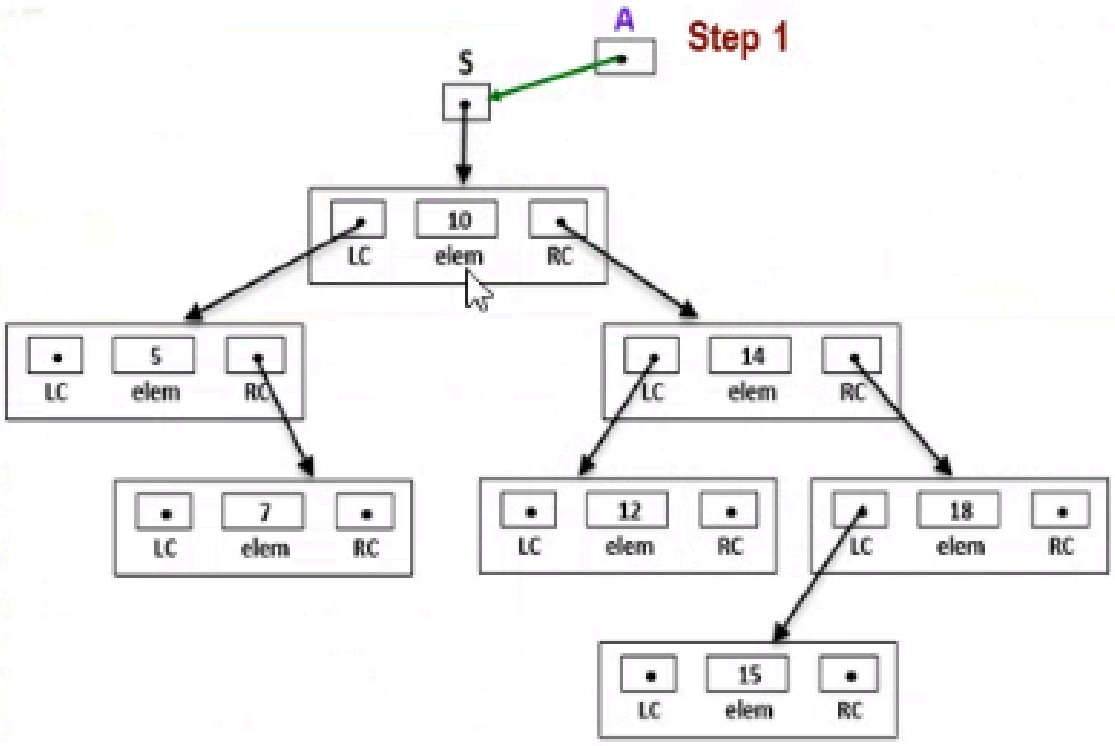
Operation: Insert 11



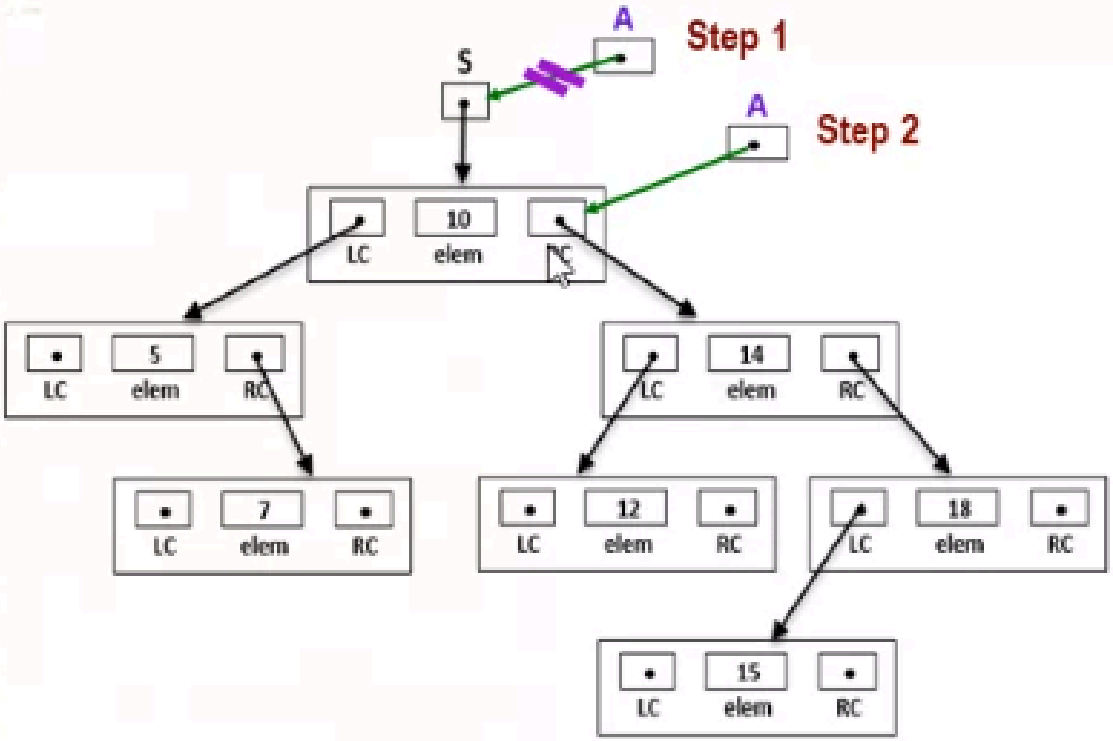
A is NULL, hence

A is no longer connected to the tree!!

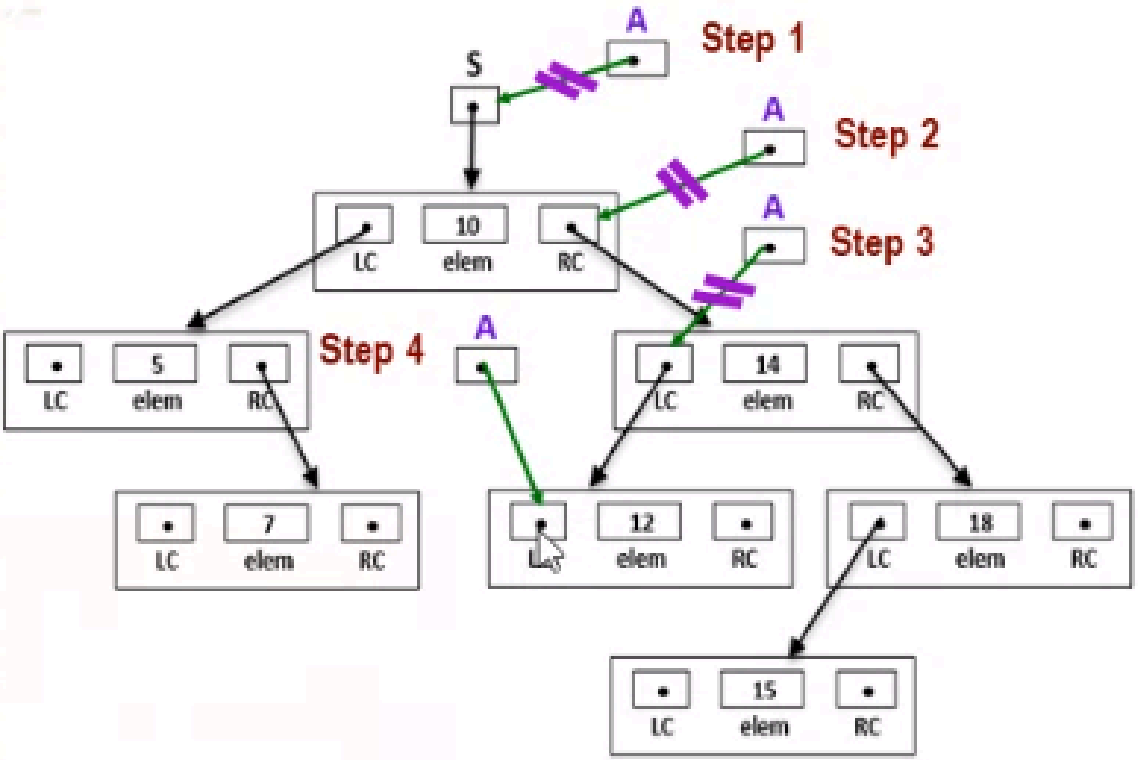
Operation: Insert 11
(A different Approach: PPN)



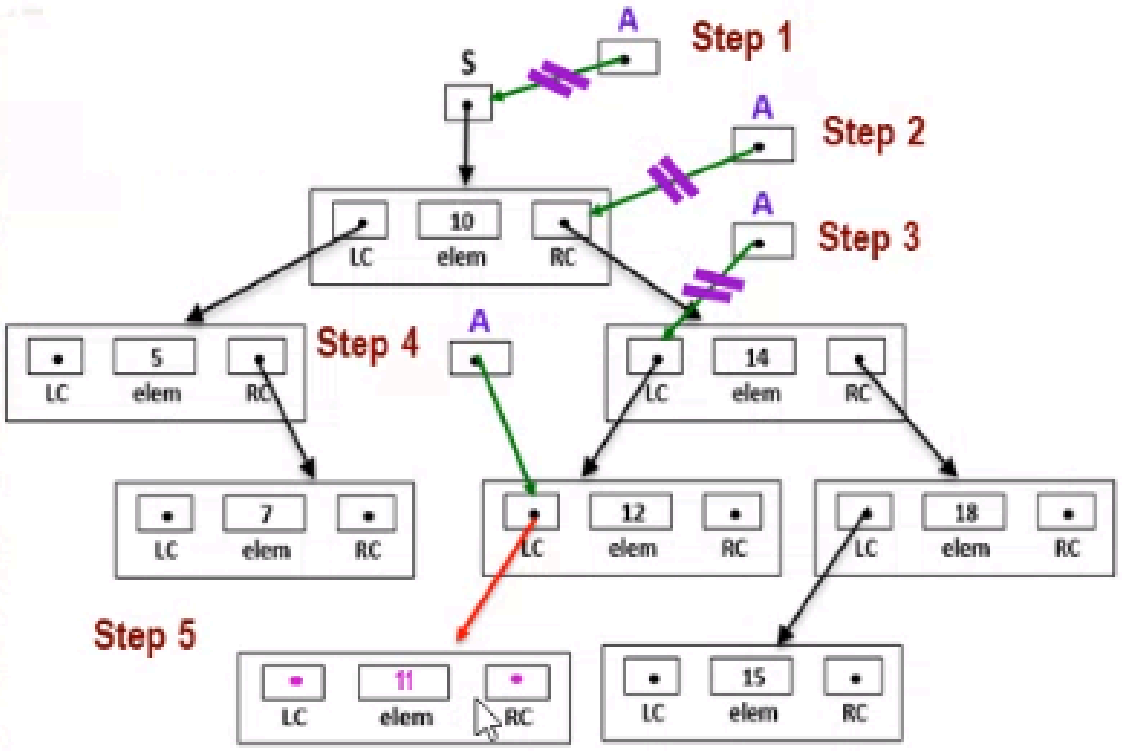
Operation: Insert 11
(A different Approach: PPN)



Operation: Insert 11
(A different Approach: PPN)

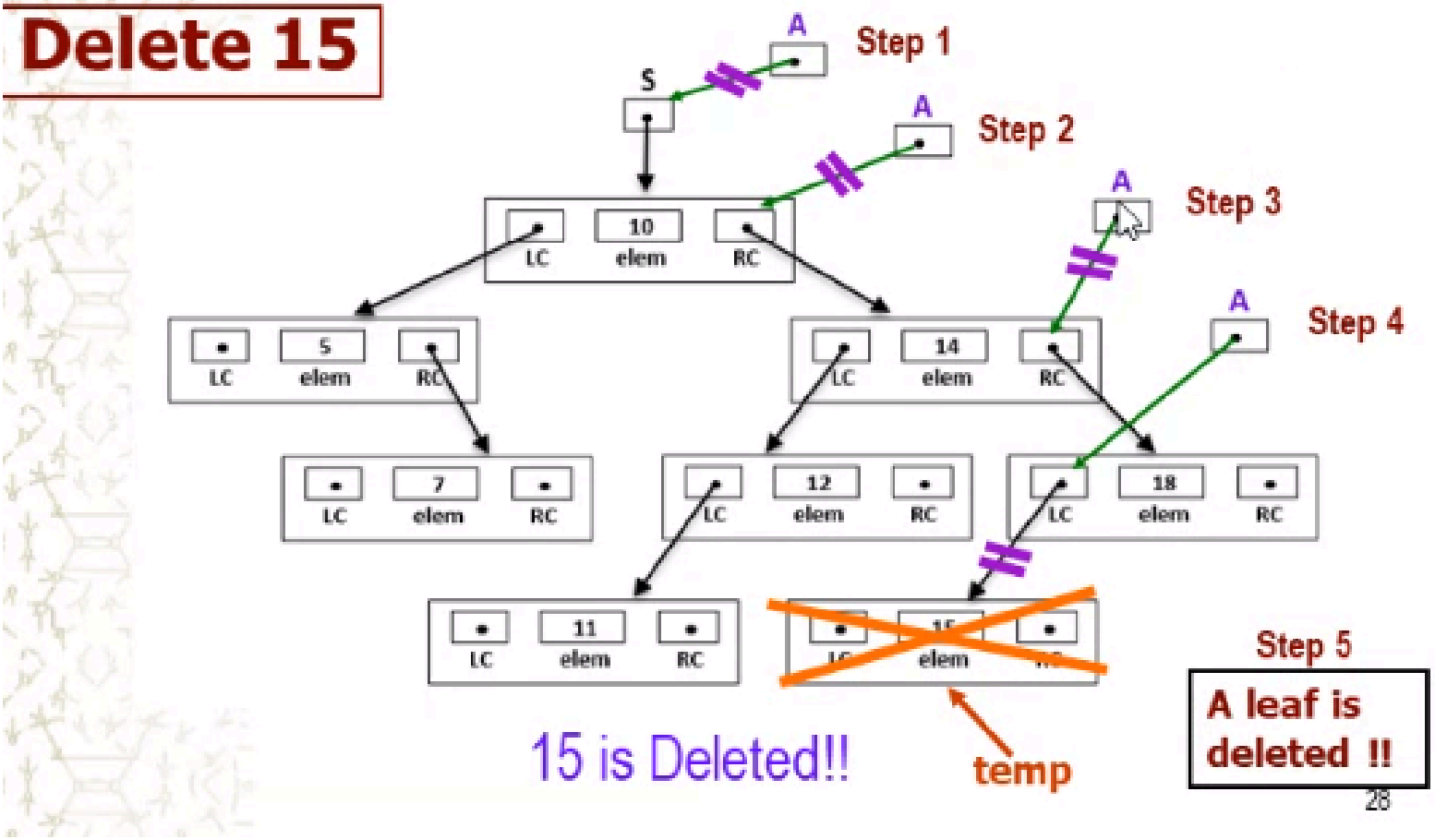


Operation: Insert 11
(A different Approach: PPN)



Operation Delete

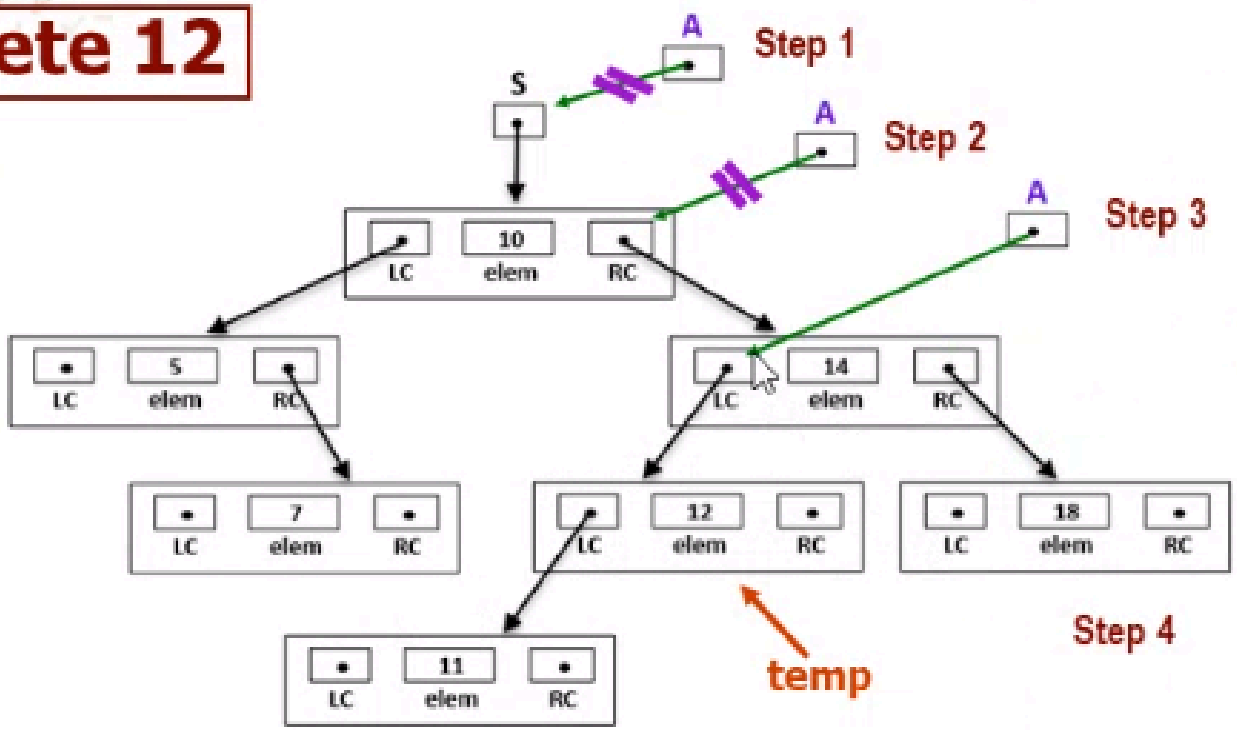
Operation: a) Delete a leaf



Operation: b) Delete an inode

With a LEFT child ONLY

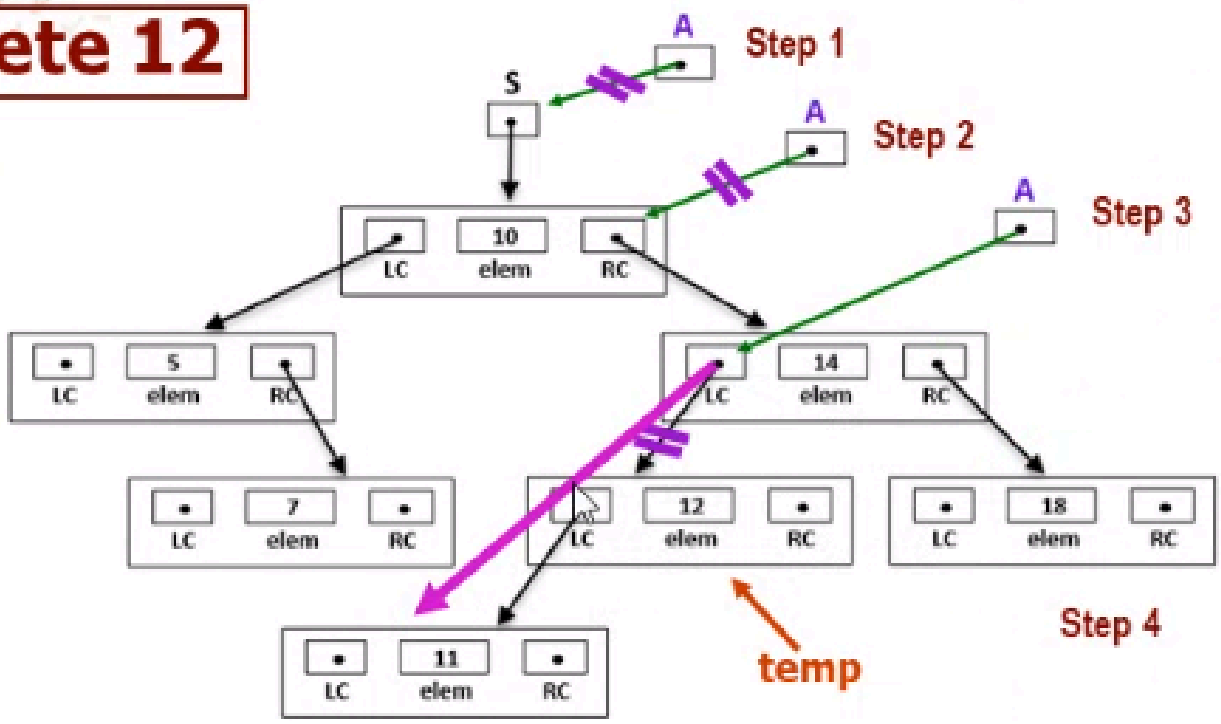
Delete 12



Operation: b) Delete an inode

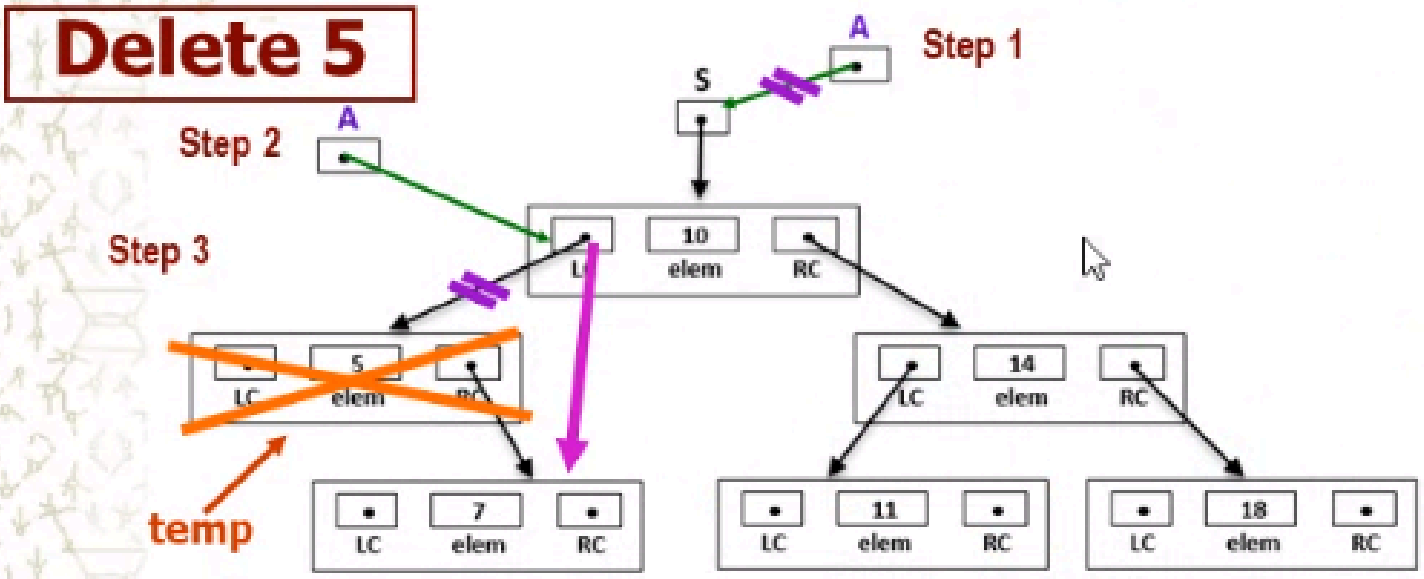
With a LEFT child ONLY

Delete 12



Operation: c) Delete an inode

With a RIGHT child ONLY



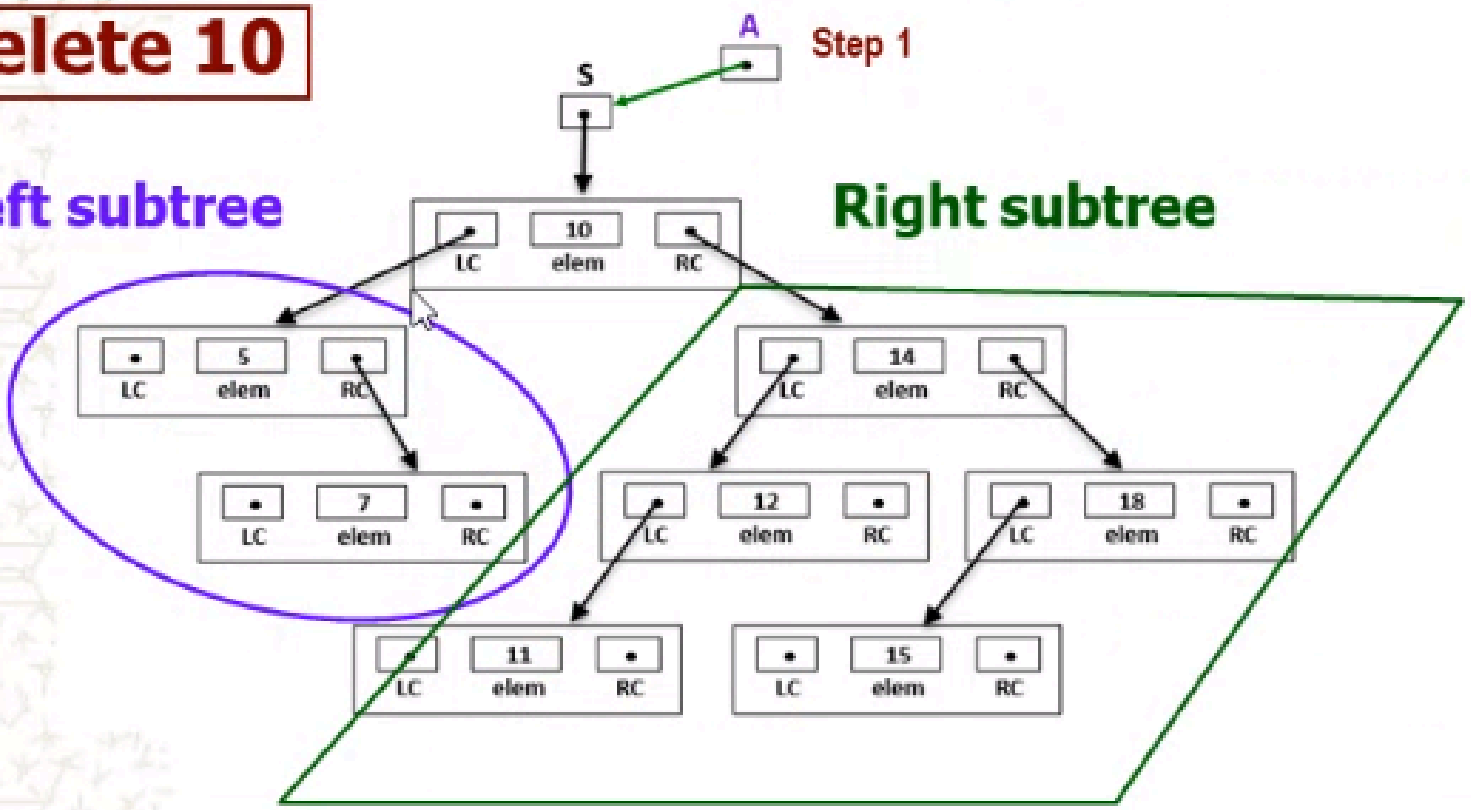
An interior node with ONLY a RIGHT child is deleted !!

5 is Deleted!!

Operation: d) Delete an inode

With RIGHT and LEFT children

Delete 10

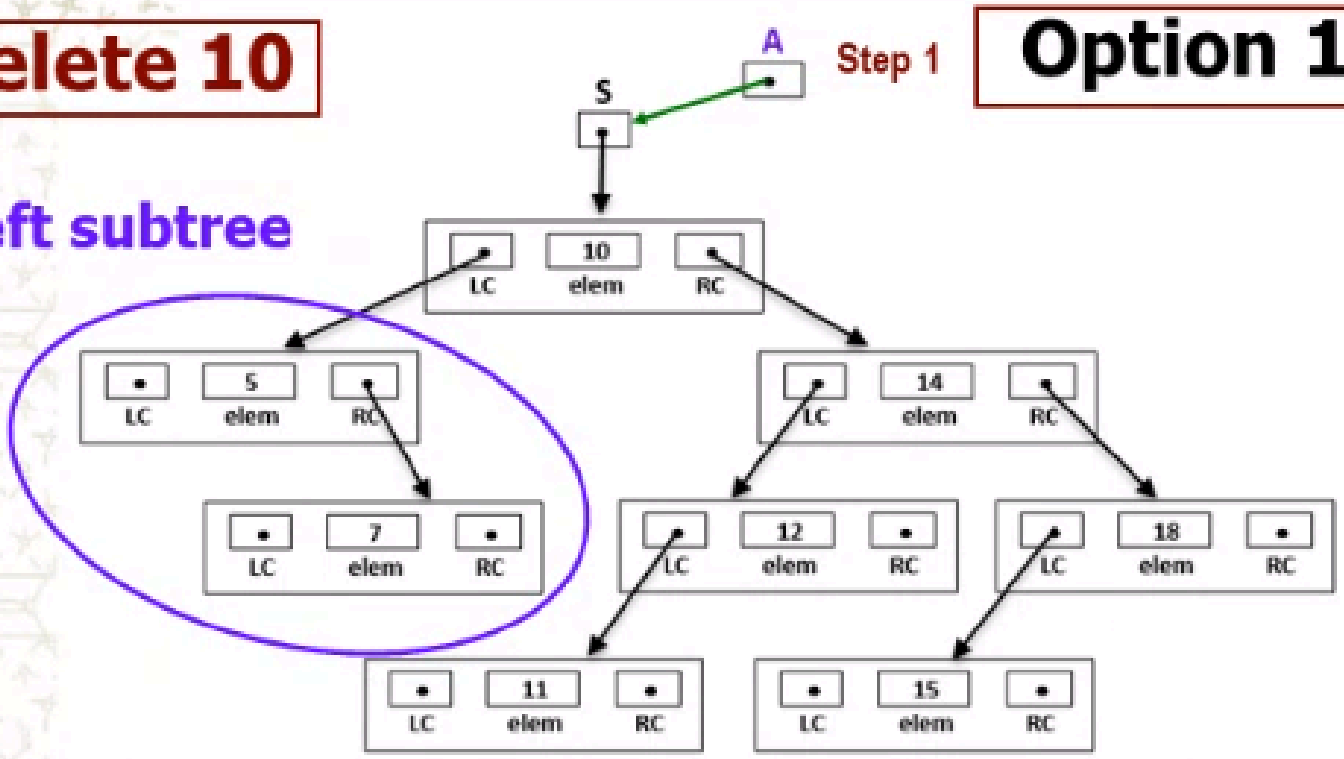


Operation: d) Delete an inode

Replace with node from Left subtree

Delete 10

Option 1



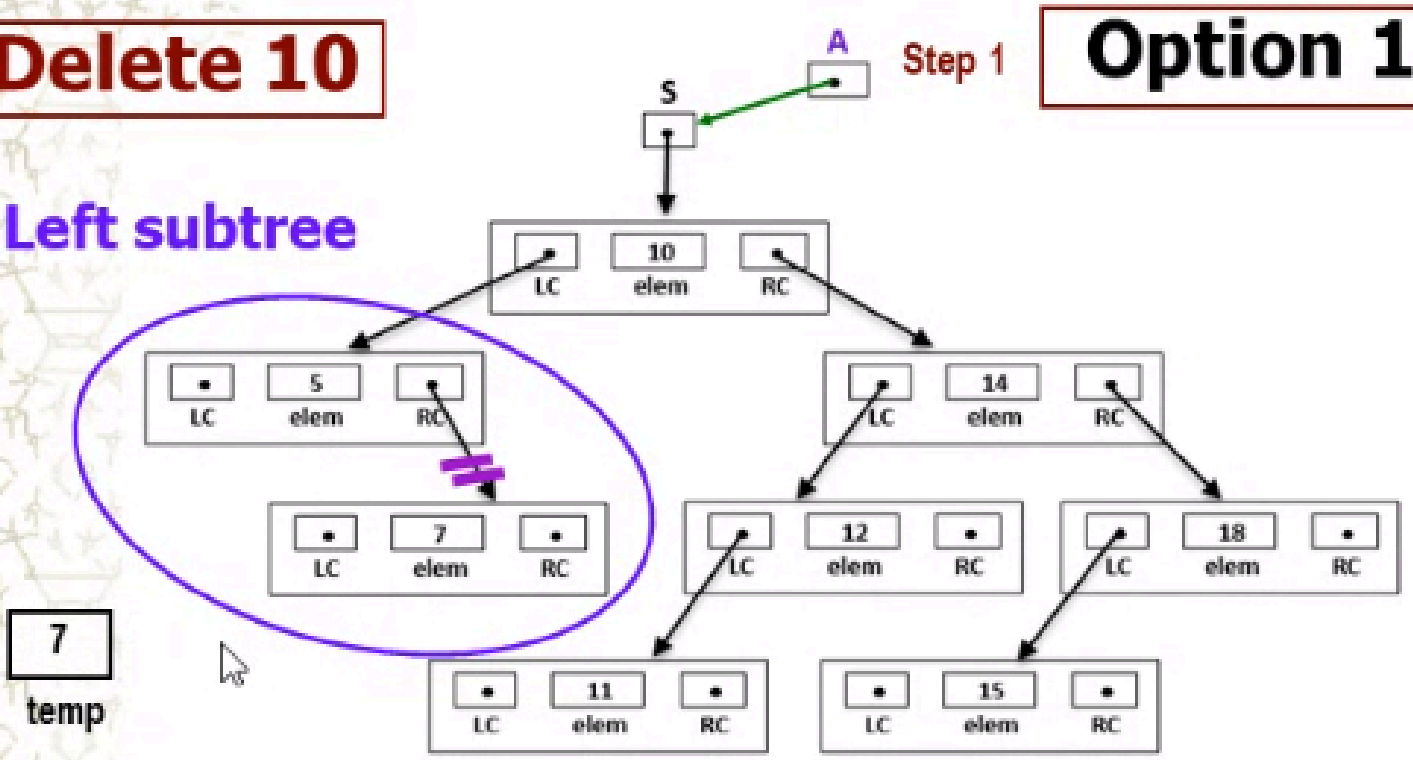
What is the immediate predecessor of 10 ? **7**

Operation: d) Delete an inode

Replace with node from Left subtree

Delete 10

Option 1



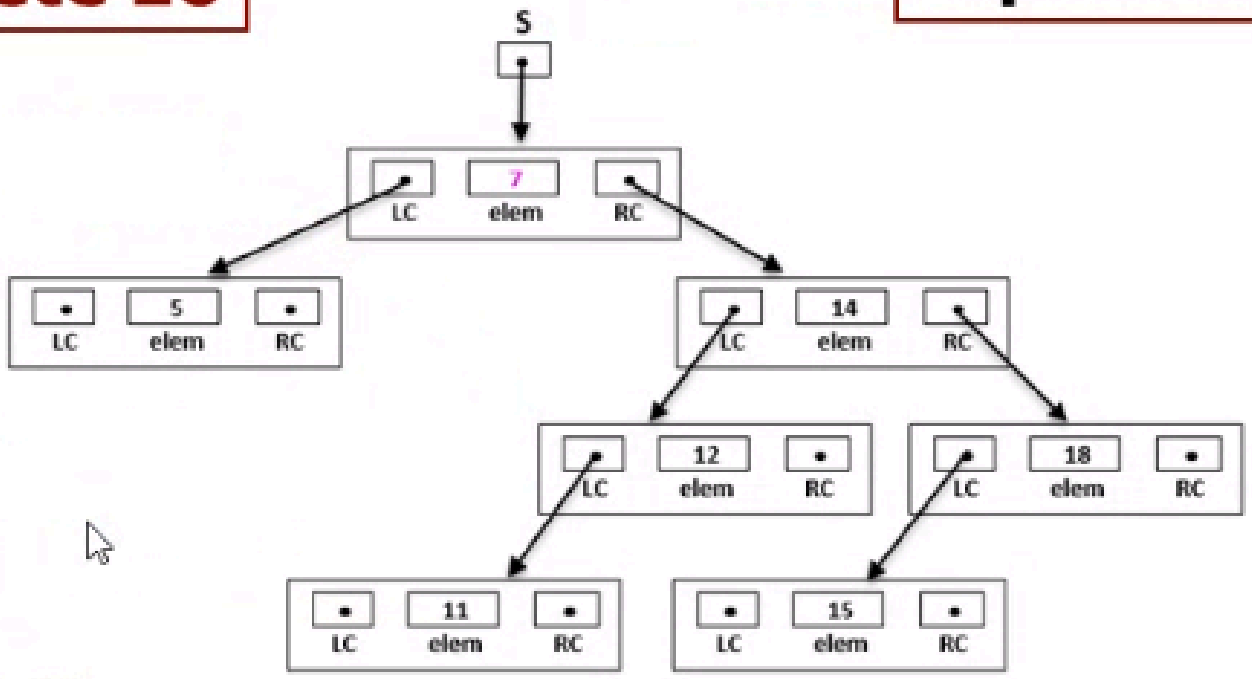
What is the immediate predecessor of 10 ? **7**

Operation: d) Delete an inode

Replace with node from Left subtree

Delete 10

Option 1

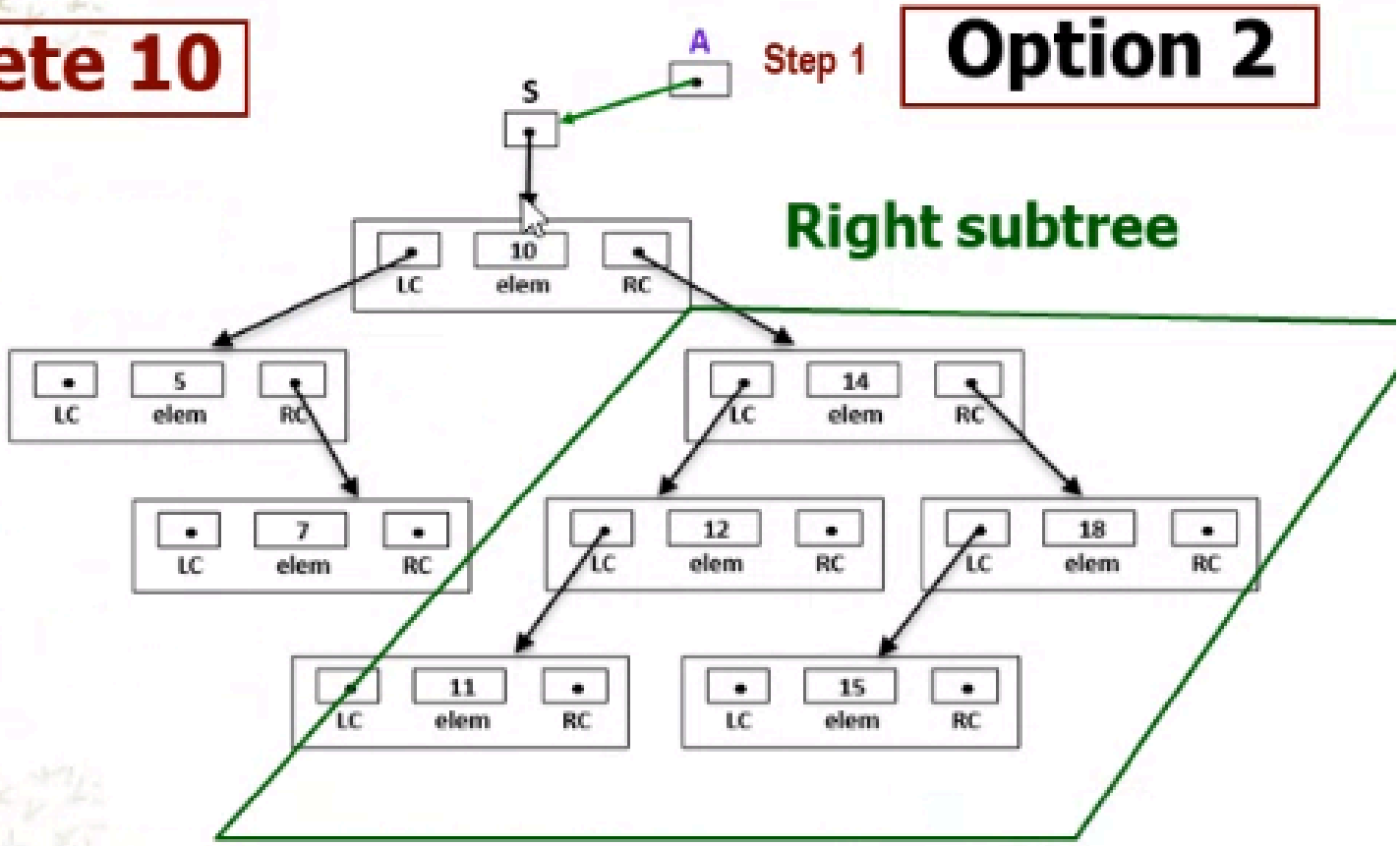


Operation: d) Delete an inode

Replace with node from Right subtree

Delete 10

Option 2



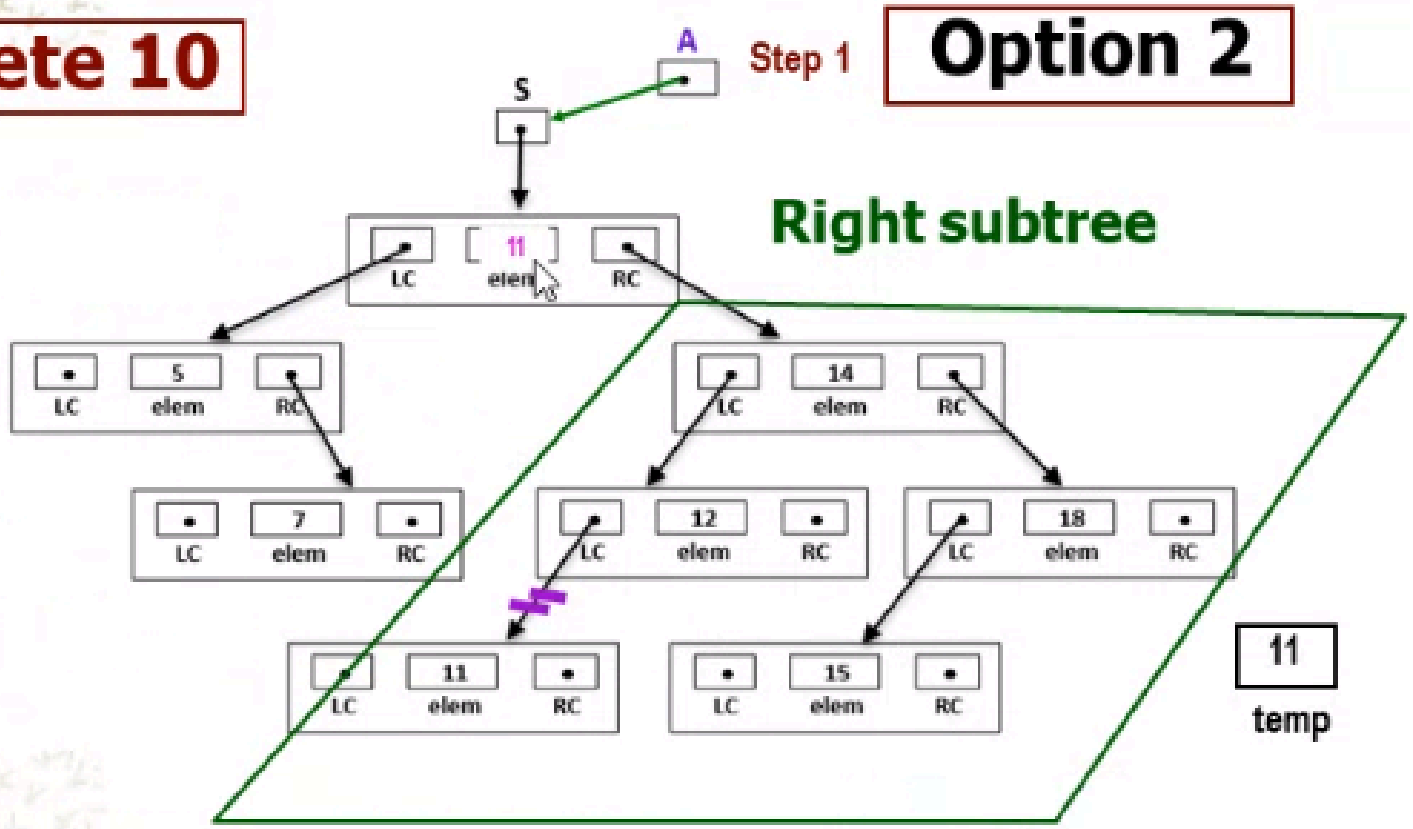
What is the immediate successor of 10 ? **11**

Operation: d) Delete an inode

Replace with node from Right subtree

Delete 10

Option 2



What is the immediate successor of 10 ? **11**

Operation: Delete

- Steps:
  1. Follow Membership test
  2. If element to be deleted is found and
     - the element is a **leaf** → **cut the link**
     - the element is an **interior node with ONLY 1 child** → **let the pointer to the element to be deleted point to the only child**
     - the element is an **interior node with 2 children** →
       - **replace the element with its immediate predecessor or immediate successor and delete the**
       - **Delete the immediate predecessor or successor**