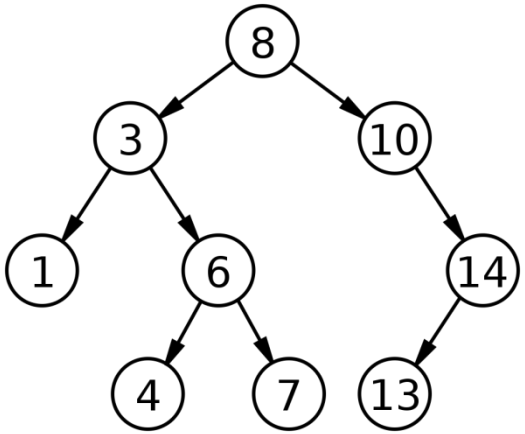
Binary Trees: Search & Insert

COP 3502 – Computer Science I

Binary Search Tree



- We already got idea about a BST in our last slide.
- Now we will learn how to build a BST
  - How to insert an item in a way that it will insert the item properly.

Binary Search Tree Creation

- Insertion into a Binary Search Tree
- Do you remember in linked list, before inserting a node what we have to do?
- We have to actually **create** the node that we want to insert
  - malloc space for the node
  - And save appropriate data value(s) into it
  - Here's our struct from last time:

```
struct tree_node {
        int data;
        struct tree_node *left;
        struct tree_node *right;
}
```

Create node function

```
struct tree_node* create_node(int val) {

        // Allocate space for the node
        struct tree_node* temp;
        temp = (struct tree_node*)malloc(sizeof(struct tree_node));

        // Initialize the fields
        temp->data = val;
        temp->left = NULL;
        temp->right = NULL;

         // Return a pointer to the created node.
        return temp;
}
```

Binary Search Tree: Insertion

- We have the node, it is **time to insert**!
- **BST**s must maintain their ordering property
  - Smaller items to the left of any given root
  - And greater items to the right of that root

- So when we insert, we MUST follow these rules
- You simply start at the root and either
  1) Go right if the new value is greater than the root
  2) Go left if the new value is less than the root
- Keep doing this till you come to an empty position
- An example will make this clear…

Binary Search Tree: Insertion

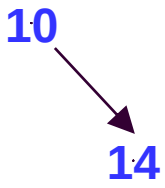
- Let's assume we insert the following data values, in their order of appearance into an initially empty BST:
  - **10,** 14, 6, 2, 5, 15, and 17

10

- **Step 1**:
  - Create a new node with value 10
  - Insert node into tree
  - The tree is currently empty
  - New node becomes the root

Binary Search Tree: Insertion

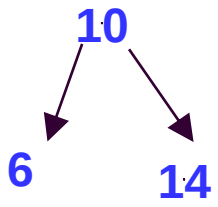
- **10, 14**, 6, 2, 5, 15, and 17
- **Step 2**:
  - Create a new node with value 14
  - This node belongs in the right subtree of node 10
    - Since 14 > 10
  - The right subtree of node 10 is empty
    - So node 14 becomes the right child of node 10



Binary Search Tree: Insertion

- Insertion into a Binary Search Tree
  - **10, 14, 6**, 2, 5, 15, and 17

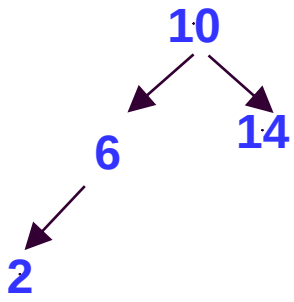
  - **Step 3**:
    - Create a new node with value 6
    - This node belongs in the left subtree of node 10
      - Since 6 < 10
    - The left subtree of node 10 is empty
      - So node 6 becomes the left child of node 10



Binary Search Tree: Insertion

- Insertion into a Binary Search Tree
  - **10, 14, 6, 2**, 5, 15, and 17

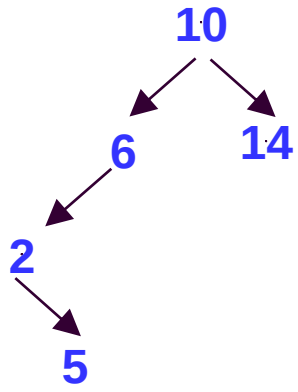
  - **Step 4**:
    - Create a new node with value 2
    - This node belongs in the left subtree of node 10
      - Since 2 < 10
    - The root of the left subtree is 6
    - The new node belongs in the left subtree of node 6
      - Since 2 < 6
    - So node 2 becomes the left child of node 6



Binary Search Tree: Insertion

- Insertion into a Binary Search Tree
  - **10, 14, 6, 2, 5**, 15, and 17

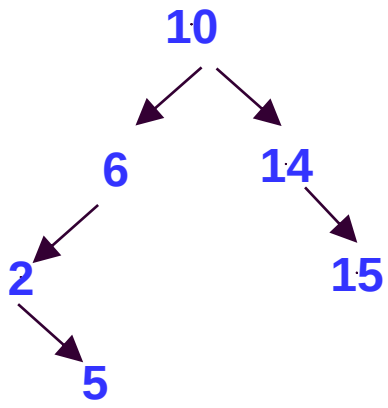
  - **Step 5**:
    - Create a new node with value 5
    - This node belongs in the left subtree of node 10
      - Since 5 < 10
    - The new node belongs in the left subtree of node 6
      - Since 5 < 6
    - And the new node belongs in the right subtree of node 2
      - Since 5 > 2



Binary Search Tree: Insertion

- Insertion into a Binary Search Tree
  - **10, 14, 6, 2, 5, 15**, and 17

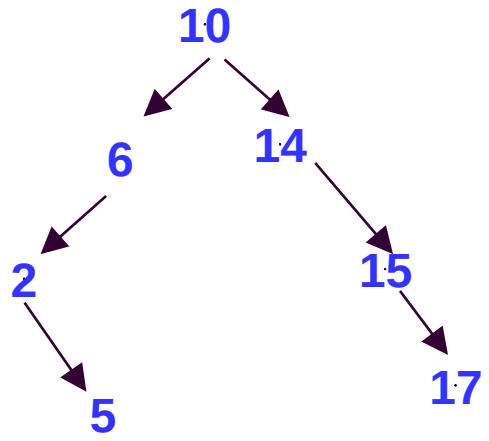
  - **Step 6**:
    - Create a new node with value 15
    - This node belongs in the right subtree of node 10
      - Since 15 > 10
    - The new node belongs in the right subtree of node 14
      - Since 15 > 14
    - The right subtree of node 14 is empty
    - So node 15 becomes right child of node 14



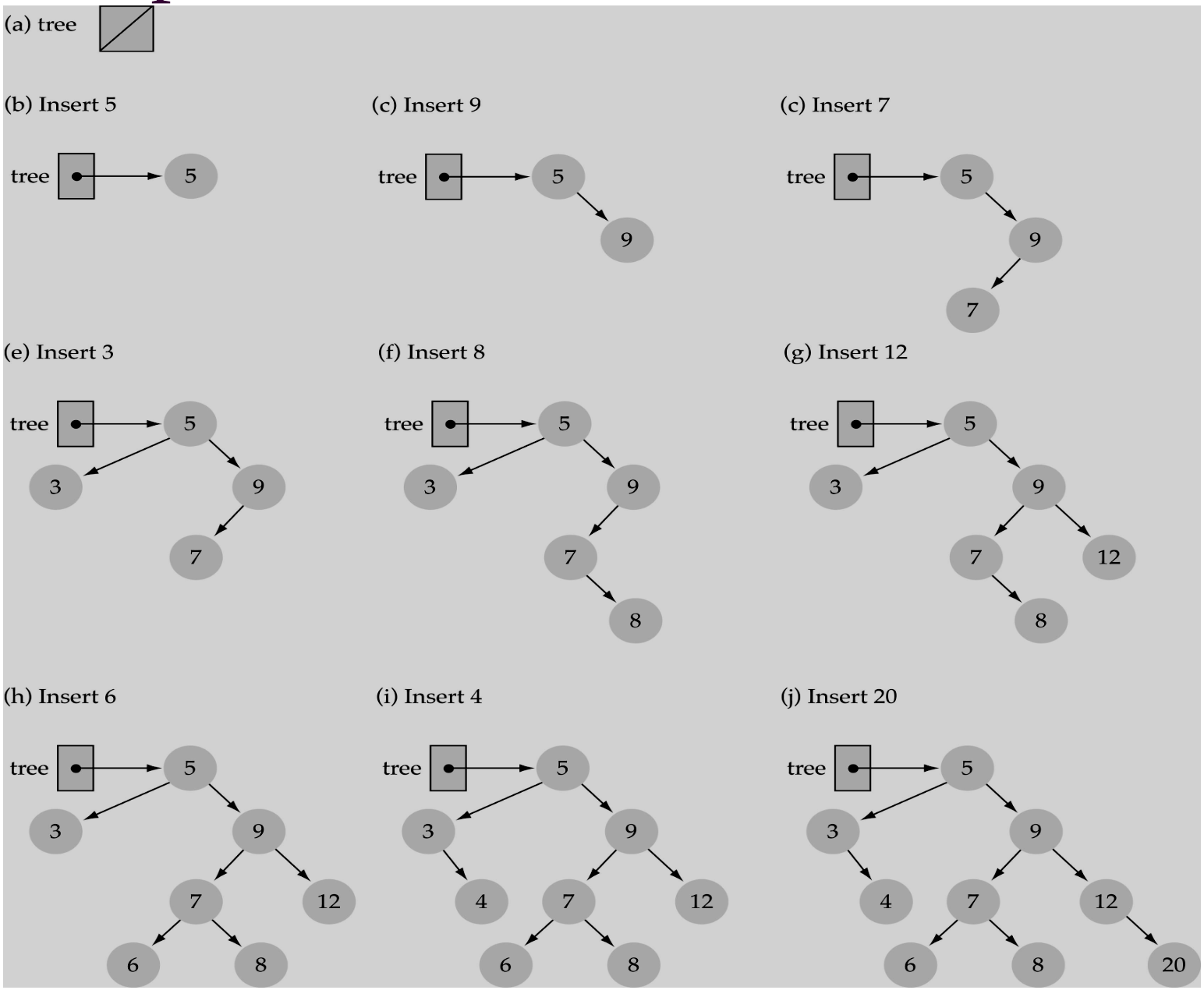
Binary Search Tree: Insertion

- Insertion into a Binary Search Tree
  - **10, 14, 6, 2, 5, 15**, and **17**

  - **Step 7**:
    - Create a new node with value 17
    - This node belongs in the right subtree of node 10
      - Since 17 > 10
    - The new node belongs in the right subtree of node 14
      - Since 17 > 14
    - And the new node belongs in the right subtree of node 15
      - Since 17 > 15

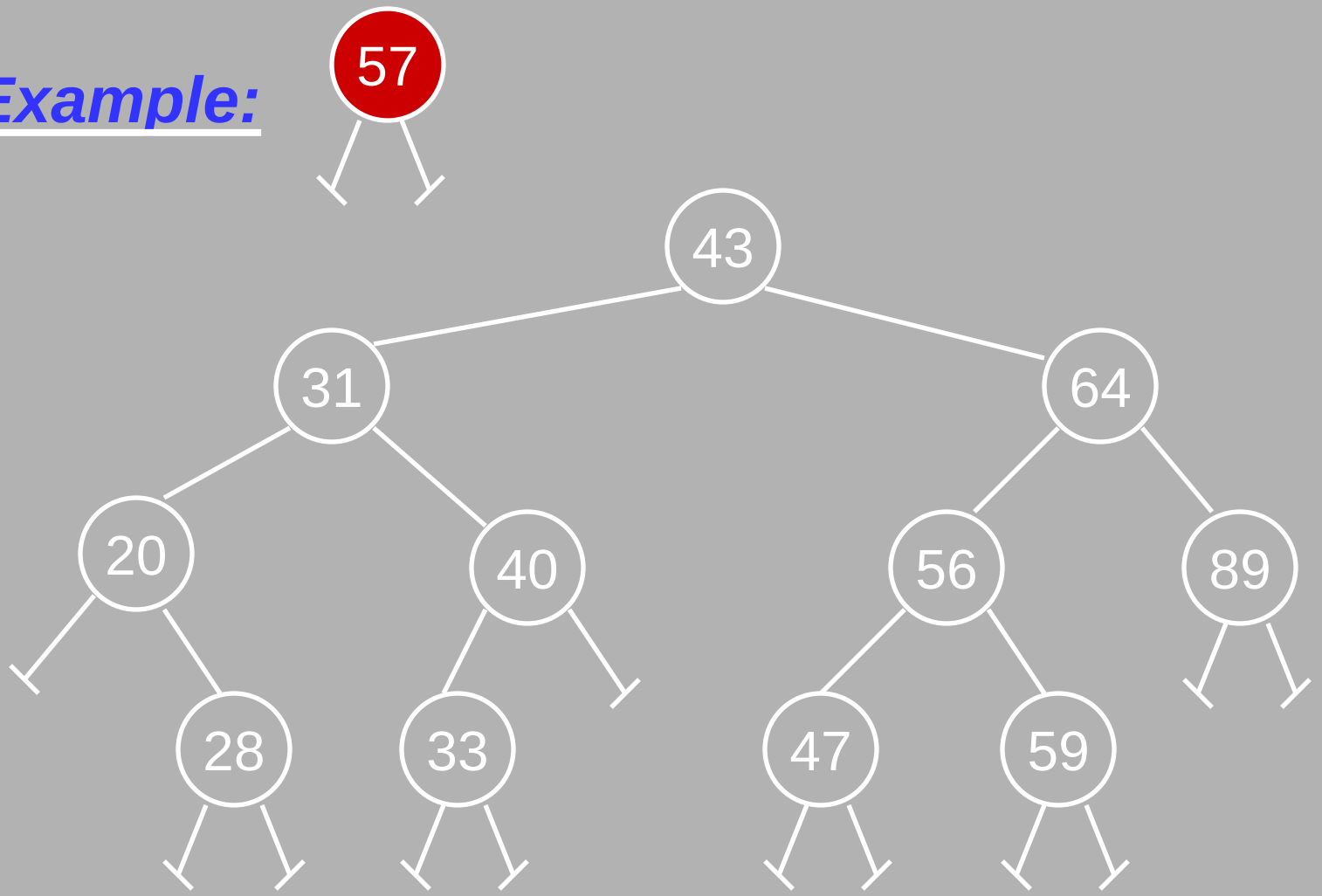


More Example



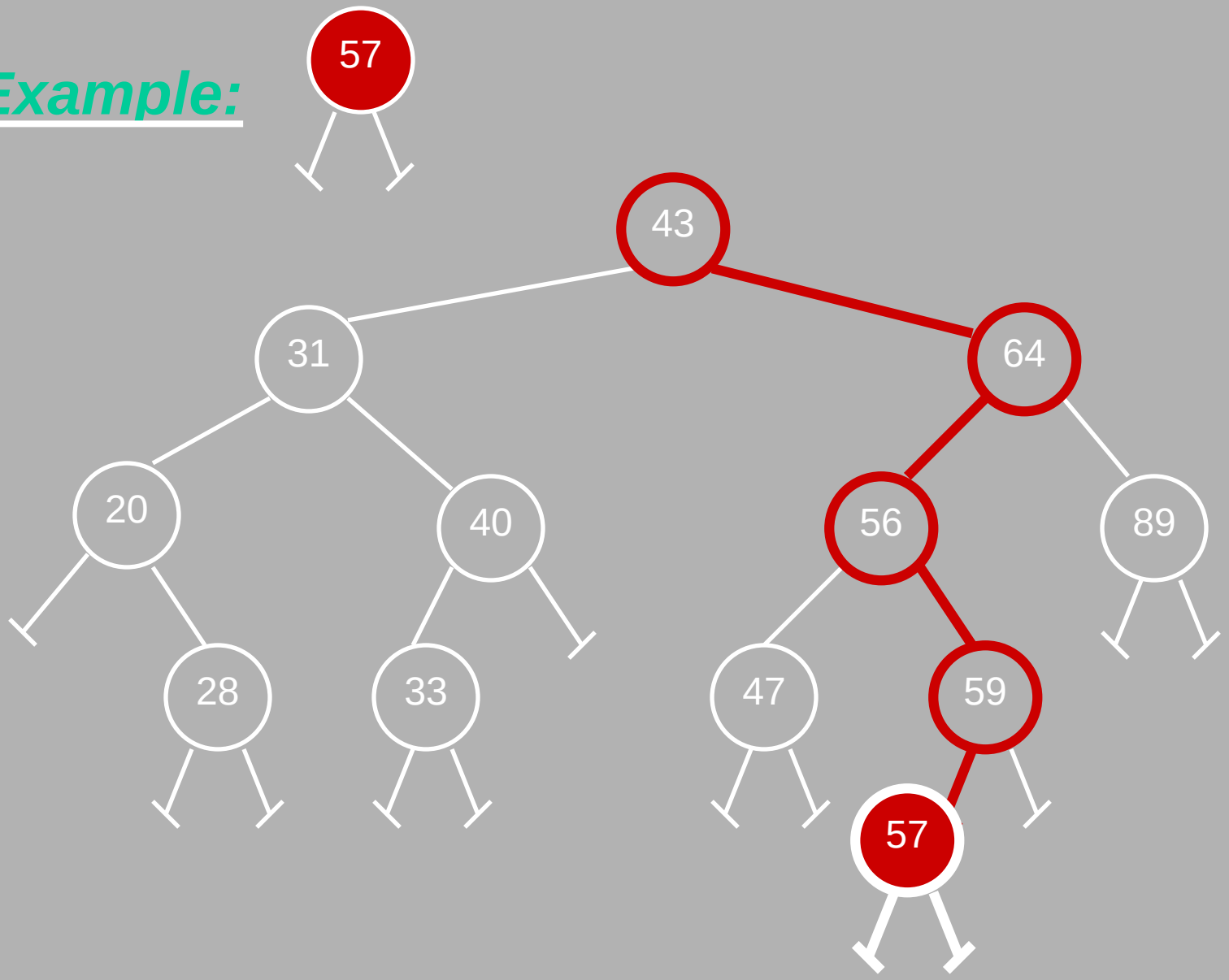
Insert (Where 57 will be inserted?)

Example:



Insert

Example:



How the main function will look like

```
struct tree_node *my_root=NULL, *temp_node;
while ( some exit condition) {
            printf(“Enter a value to insert: ");
            scanf("%d", &val);
            temp_node = create_node(val); // Create the node.
             // Insert the value.
            my_root = insert(my_root, temp_node);

   //other code

}
```

How the other code will look like?

- Create new node for the item.
- Find a parent node.
- Attach new node as a leaf.

- You can find the appropriate position using recursion or loop.

Insert: Recursive

- **parameters:** Here is how we called the function from the main function **insert(my_root, temp_node);**

- **If my_root is** NULL, the tree is empty
  - Just return temp_node as it will be assinged to root.

- Else, see which subtree the node should be inserted into
  - Compare the temp_node's data to the root's item
- Based on the comparison, recursively either insert into the right subtree or into the left subtree.

```
struct tree_node* insert(struct tree_node *root, struct tree_node *element) {
  // Inserting into an empty tree.
  if (root == NULL)
    return element;
  else {
        // element should be inserted to the right.
        if (element->data > root->data) {
                // There is a right subtree to insert the node.
                if (root->right != NULL)
                        root->right = insert(root->right, element);
                // Place the node directly to the right of root.
                else
                        root->right = element;
         }
         // element should be inserted to the left.
         else {
                // There is a left subtree to insert the node.
                if (root->left != NULL)
                        root->left = insert(root->left, element);
                // Place the node directly to the left of root.
                else
                        root->left = element;
         }
    // Return the root pointer of the updated tree.
    return root;
  }
}
```

Testing Binary Search Tree after creating it

How can you check whether you Binary search tree was properly created or not?

In order to test your binary search tree creation, print your tree in in-order and see whether the in-order produces the data in sorted order or not!

BST- Computing Sum of Nodes

- How would you compute some of nodes in a linked list?
  - Simply traversed the list and summed the values
- Similarly, we know how to traverse a tree and while traversing we can sum the data.
- Choose any tree traversal method we learned:
  - preorder, inorder, postorder

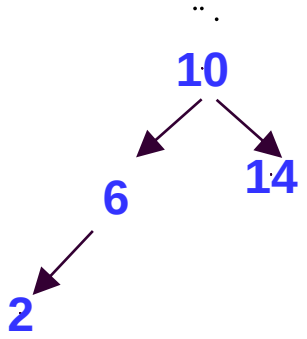
- Can we do it even easier way?
  - All we need is to add values at root , left and right and then return the answer.
  - See the function.

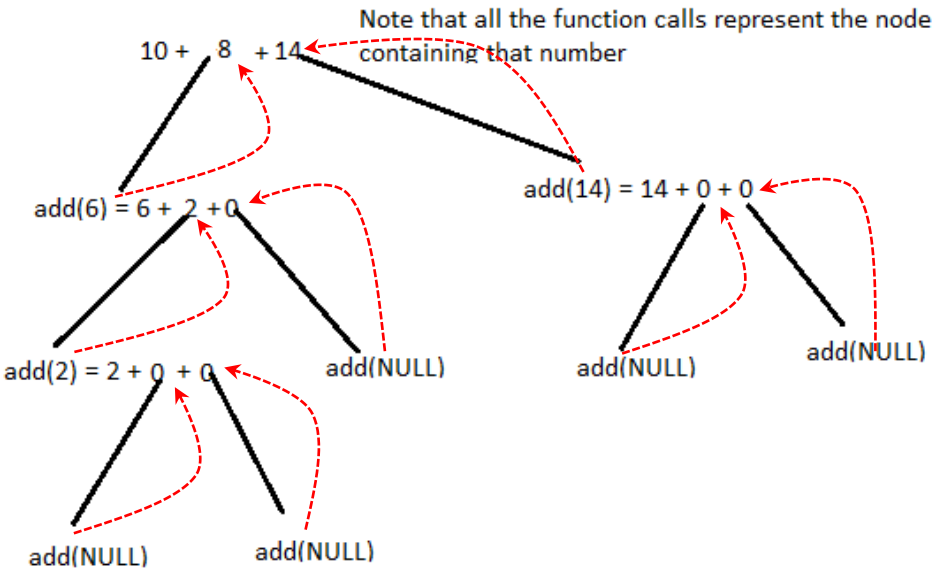
```
int add(struct tree_node *current_ptr) {

   if (current_ptr != NULL)
        return current_ptr->data+add(current_ptr->left)+
         add(current_ptr->right);
     else
        return 0;
}
```

BST- Computing Sum of Nodes

- Let us analyze the code how is it calculating the sum of node.
- As we have discussed in multiple examples during merge sort and recursion, drawing recursion tree can help you to understand a recursion.

```
int add(struct tree_node *current_ptr) {

   if (current_ptr != NULL)
        return current_ptr->data+add(current_ptr->left)+
         add(current_ptr->right);
     else
        return 0;
}
```





Searching in BST

- Searching should be easier as you already know about insertion
- Steps to search for the node with the data you are looking for:
  1) **IF** the root node is NULL, return false.

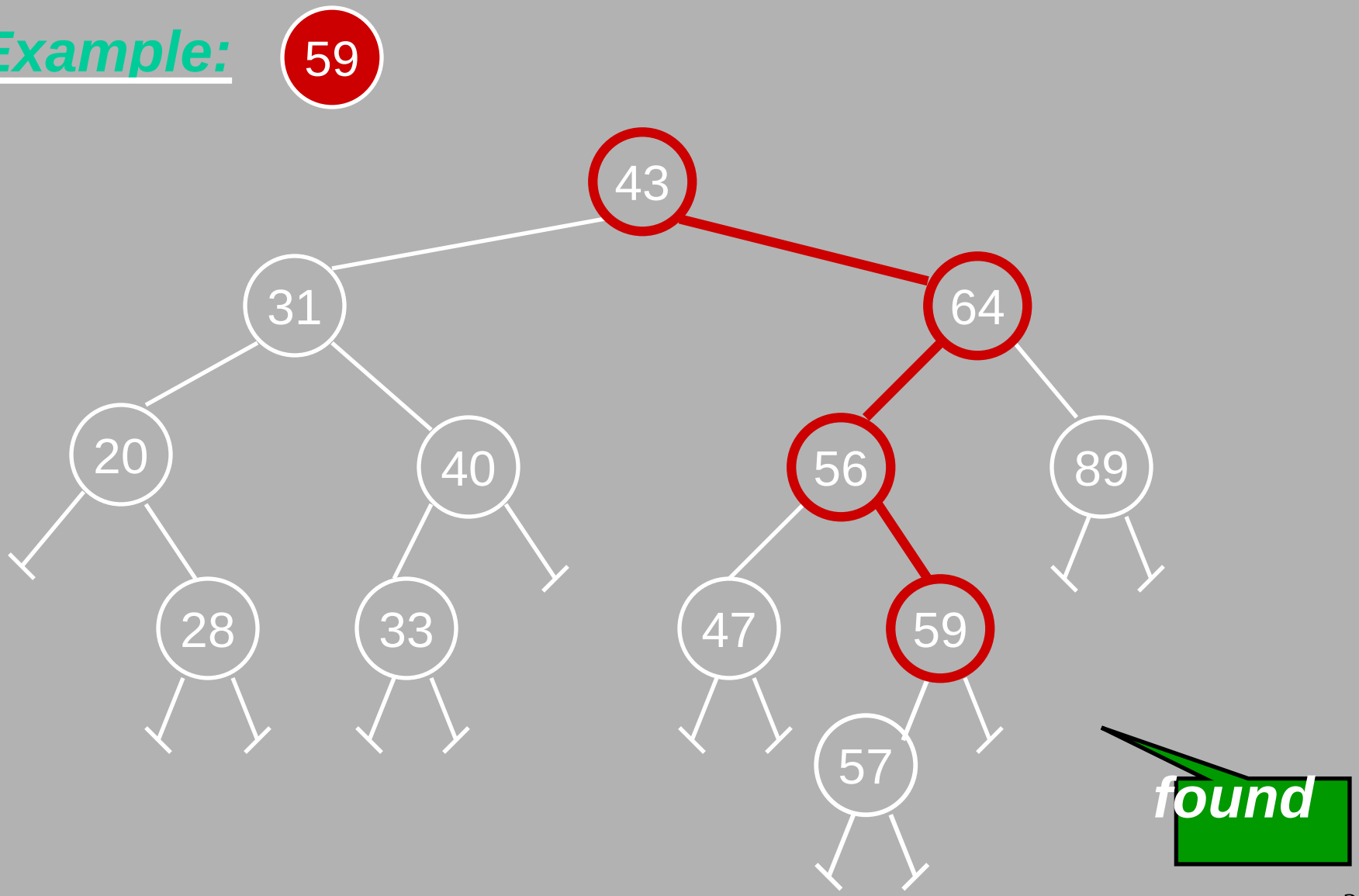
     ELSE
  2) Check the root node. If the value we are searching for is in the root, return 1 (representing “found”).
  3) If not, if the value is less than that stored in the root node, recursively search in the left subtree.
  4) Otherwise, recursively search in the right subtree.

Search: Checklist

- if target key is less than current node's key, search the left sub-tree.
- else, if target key is greater than current node's key, search the right sub-tree.
- returns:
  - if found, 1 to indicate it is found.
  - otherwise, 0 to indicate it is not found.

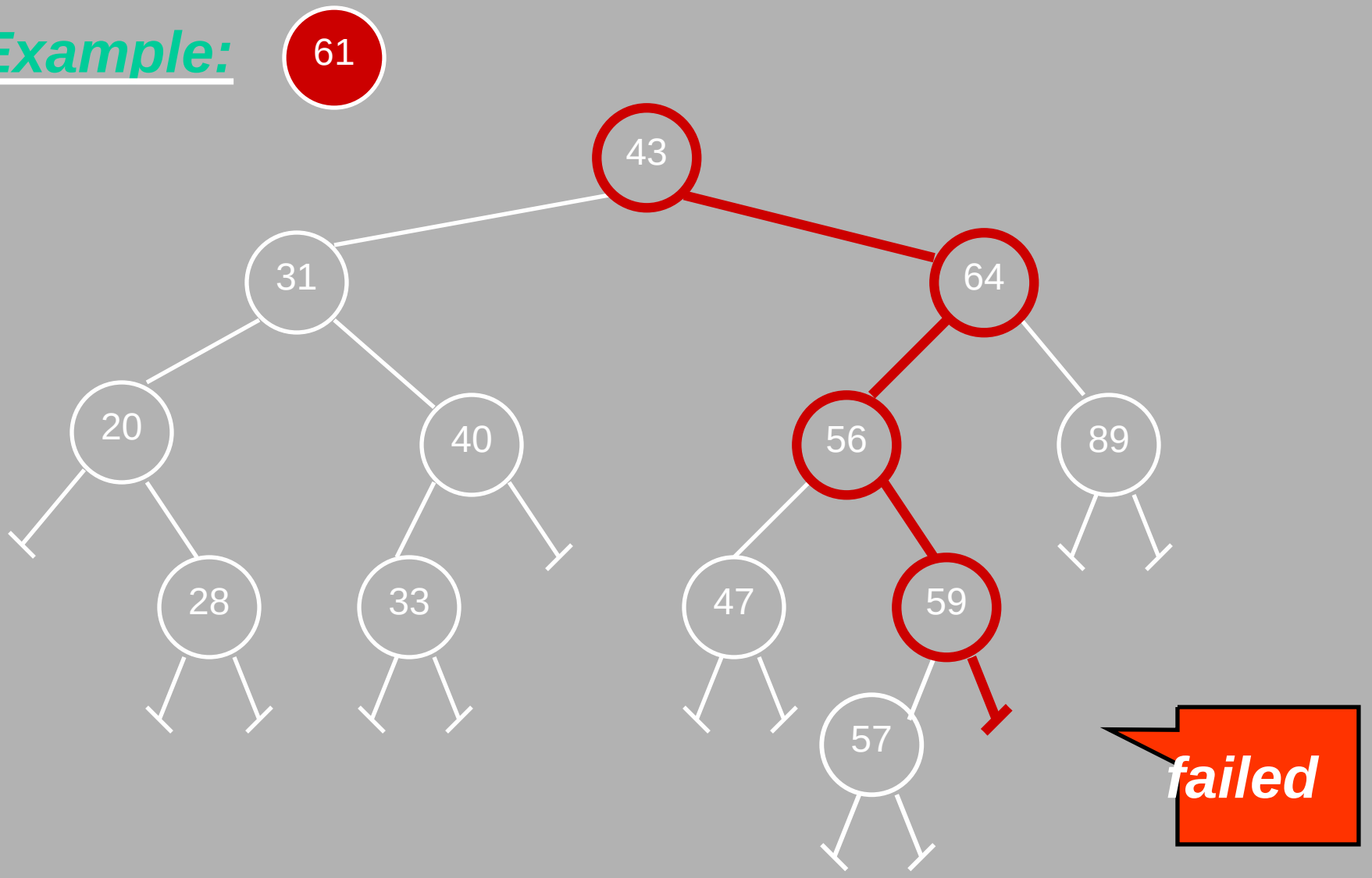
Search

Example:



Search

Example:



Searching in BST code

```
int find(struct tree_node *current_ptr, int val) {

  // Check if there are nodes in the tree.
  if (current_ptr != NULL) {

    // Found the value at the root.
    if (current_ptr->data == val)
      return 1;

    // Search to the left.
    if (val < current_ptr->data)
      return find(current_ptr->left, val);

    // Or...search to the right.
    else
      return find(current_ptr->right, val);

  }
  else
    return 0;
}
```

How would you call the find function?

```
printf("What value would you like to search for?\n");
    scanf("%d", &val);
    if (find(my_root, val))
      printf(" Found %d in the tree.\n", val);
    else
      printf(" Did not find %d in the tree.\n", val);
```

Searching in an arbitrary binary tree

- As a normal binary tree does not store data in a structured way like BST, we will need to start looking for all the items.
- You could simply perform one of the traversal methods, checking each node in the process
  - Unfortunately, this won't be O(log n) anymore
  - It degenerates to O(n) since we possibly check all nodes

Exercise

- Class Exercise:
  - Write a function that prints out all the values in a **binary tree** that are greater than or equal to a value passed to the function.
  - Here is the prototype:
  - `void PrintBig(struct tree_node *current_ptr, int value);`

Here is the full BST code with various functionalities and it is a must code to explore:
http://www.cs.ucf.edu/~dmarino/ucf/transparency/cop3502/samplepr ogs/bintree.c

References and acknowledgement

1.) Many slides and text were taken from Jonathan's notes with his permission.
https://www.cs.ucf.edu/courses/cop3502/spr2012/notes/COP3502_23_BinaryTrees2.pdf

2) More explanation and reading:
http://www.cs.ucf.edu/~dmarino/ucf/transparency/cop3502/lec/BinaryTrees-2.doc

3) Full BST code with various functionalities:
http://www.cs.ucf.edu/~dmarino/ucf/transparency/cop3502/sampleprogs/bintree.c