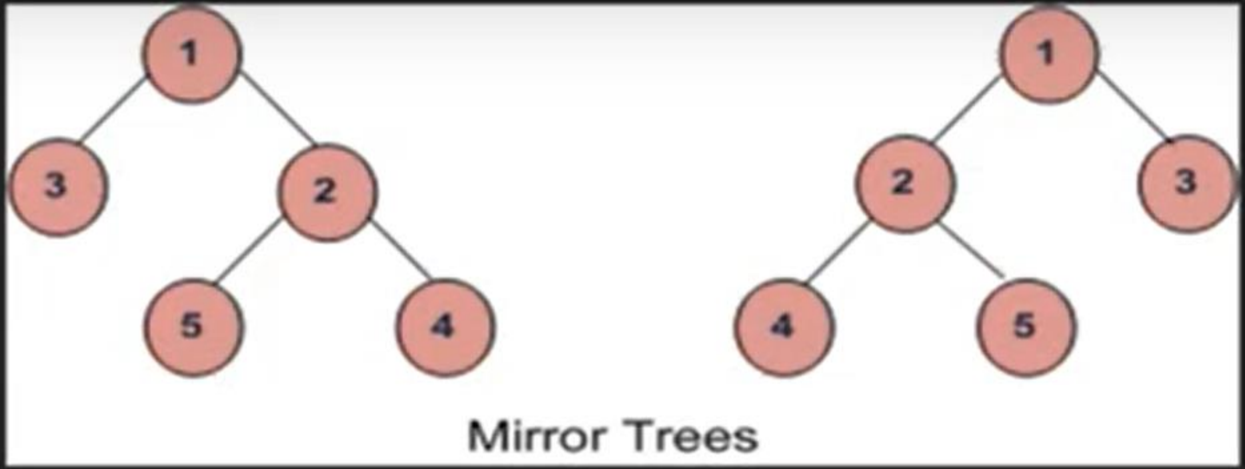
Binary Tree practice problems

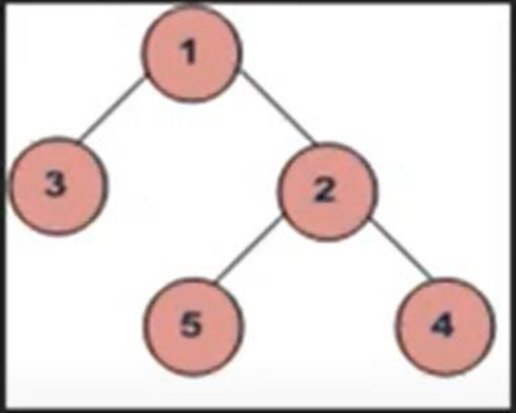


Mirror Trees

Mirror of a Tree: Mirror of a Binary Tree T is another Binary Tree M(T) with left and right children of all non-leaf nodes interchanged.

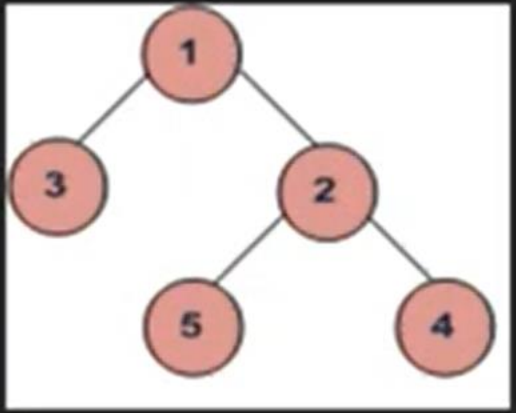
(1) Call Mirror for left-subtree i.e., Mirror(left-subtree)
(2) Call Mirror for right-subtree i.e., Mirror(right-subtree)
(3) Swap left and right subtrees.

```
temp = left-subtree
left-subtree = right-subtree
right-subtree = temp
```



```c
void mirror(struct node* node)
{
  if (node==NULL)
    return;
  else
  {
    struct node* temp;

    /* do the subtrees */
    mirror(node->left);
    mirror(node->right);

    /* swap the pointers in this node */
    temp        = node->left;
    node->left  = node->right;
    node->right = temp;
  }
}
```

(1) Call Mirror for left-subtree i.e., Mirror(left-subtree)
(2) Call Mirror for right-subtree i.e., Mirror(right-subtree)
(3) Swap left and right subtrees.
    temp = left-subtree
    left-subtree = right-subtree
    right-subtree = temp

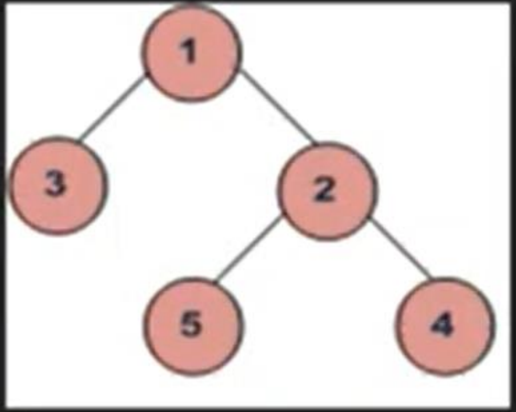


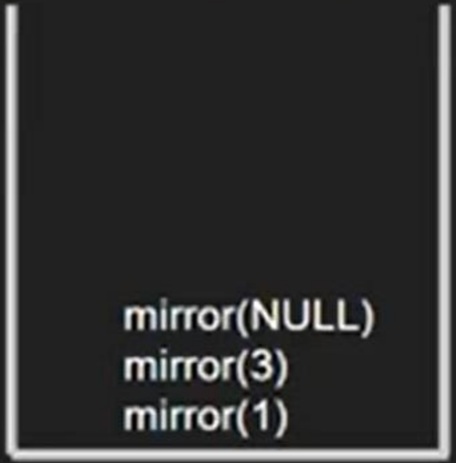
mirror(3)
mirror(1)

```
void mirror(struct node* node)
{
  if (node==NULL)
    return;
  else
  {
    struct node* temp;

    /* do the subtrees */
    mirror(node->left);
    mirror(node->right);

    /* swap the pointers in this node */
    temp        = node->left;
    node->left  = node->right;
    node->right = temp;
  }
}
```

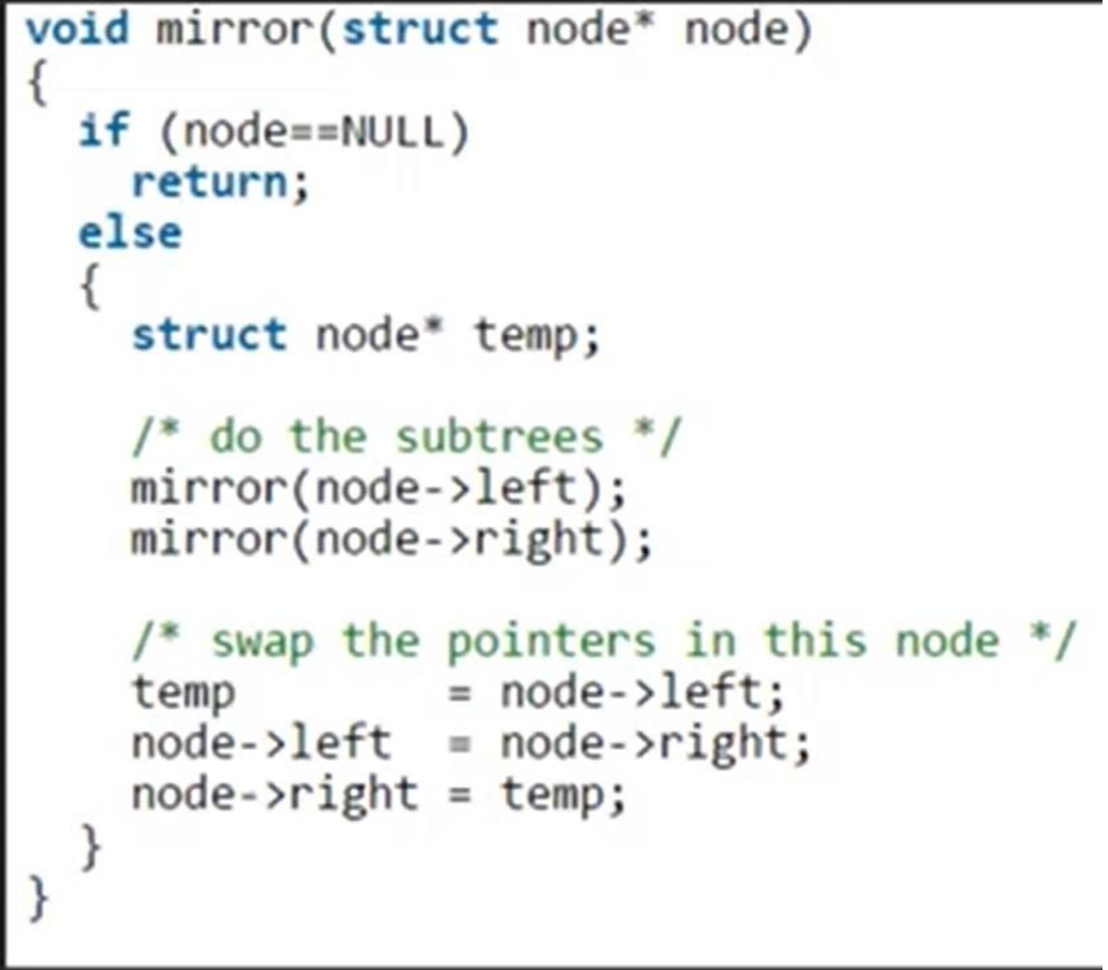
Node = 3

(1) Call Mirror for left-subtree i.e., Mirror(left-subtree)
(2) Call Mirror for right-subtree i.e., Mirror(right-subtree)
(3) Swap left and right subtrees.
    temp = left-subtree
    left-subtree = right-subtree
    right-subtree = temp





Left sub tree of 3

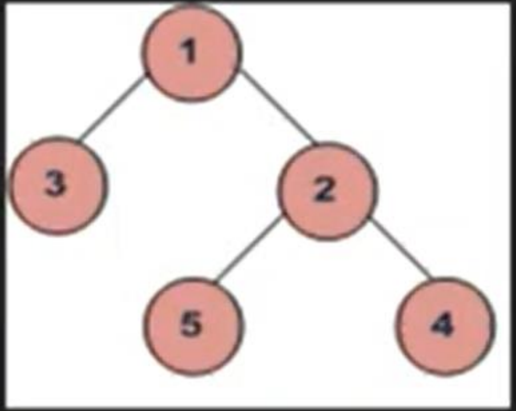


```
void mirror(struct node* node)
{
  if (node==NULL)
    return;
  else
  {
    struct node* temp;

    /* do the subtrees */
    mirror(node->left);
    mirror(node->right);

    /* swap the pointers in this node */
    temp        = node->left;
    node->left  = node->right;
    node->right = temp;
  }
}
```

Node = NULL

(1) Call Mirror for left-subtree i.e., Mirror(left-subtree)
(2) Call Mirror for right-subtree i.e., Mirror(right-subtree)
(3) Swap left and right subtrees.

```
temp = left-subtree
left-subtree = right-subtree
right-subtree = temp
```



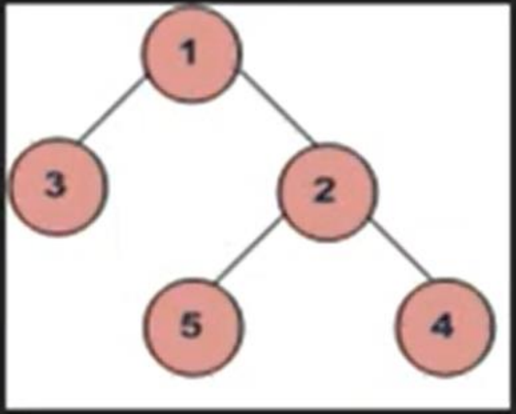
```
void mirror(struct node* node)
{
  if (node==NULL)
    return;
  else
  {
    struct node* temp;

    /* do the subtrees */
    mirror(node->left);
    mirror(node->right);

    /* swap the pointers in this node */
    temp         = node->left;
    node->left   = node->right;
    node->right  = temp;
  }
}
```

Node = NULL

```
mirror(NULL)
mirror(3)
mirror(1)
```

Right child of 3 is null

(1) Call Mirror for left-subtree i.e., Mirror(left-subtree)
(2) Call Mirror for right-subtree i.e., Mirror(right-subtree)
(3) Swap left and right subtrees.
    temp = left-subtree
    left-subtree = right-subtree
    right-subtree = temp



```
void mirror(struct node* node)
{
  if (node==NULL)
    return;
  else
  {
    struct node* temp;

    /* do the subtrees */
    mirror(node->left);
    mirror(node->right);

    /* swap the pointers in this node */
    temp        = node->left;
    node->left  = node->right;
    node->right = temp;
  }
}
```

Node = NULL

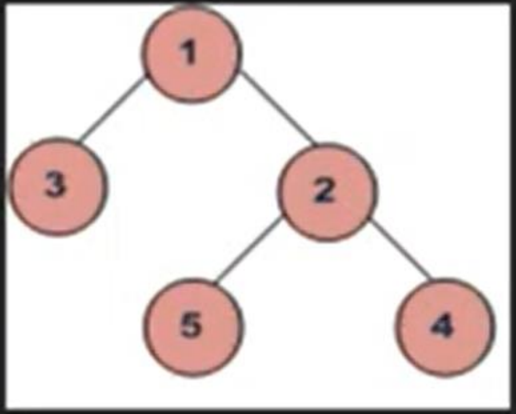
```
mirror(3)
mirror(1)
```

Left and right child of 3 is null
so return back to 3

(1) Call Mirror for left-subtree i.e., Mirror(left-subtree)
(2) Call Mirror for right-subtree i.e., Mirror(right-subtree)
(3) Swap left and right subtrees.

```
temp = left-subtree
left-subtree = right-subtree
right-subtree = temp
```

```
void mirror(struct node* node)
{
  if (node==NULL)
    return;
  else
  {
    struct node* temp;

    /* do the subtrees */
    mirror(node->left);
    mirror(node->right);

    /* swap the pointers in this node */
    temp        = node->left;
    node->left  = node->right;
    node->right = temp;
  }
}
```



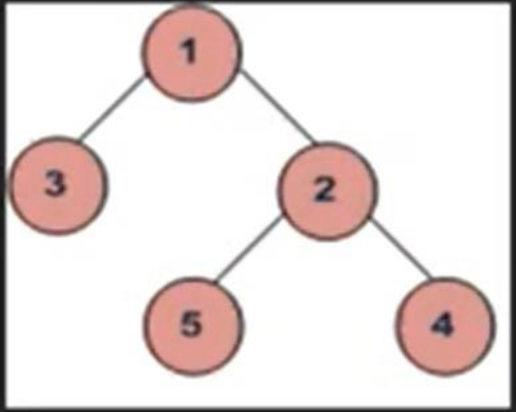
```
mirror(2)
mirror(1)
```

Node = 2

Right subtree of 1 ie-2

(1) Call Mirror for left-subtree i.e., Mirror(left-subtree)
(2) Call Mirror for right-subtree i.e., Mirror(right-subtree)
(3) Swap left and right subtrees.
    temp = left-subtree
    left-subtree = right-subtree
    right-subtree = temp

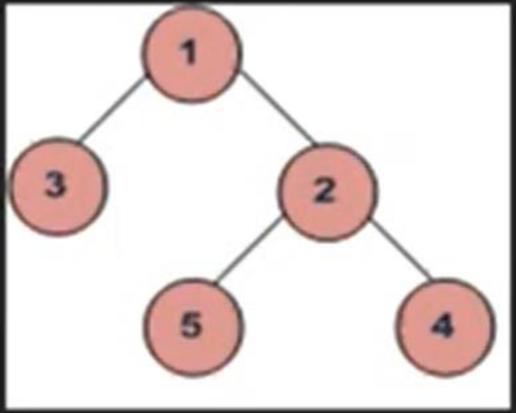
```c
void mirror(struct node* node)
{
  if (node==NULL)
    return;
  else
  {
    struct node* temp;

    /* do the subtrees */
    mirror(node->left);
    mirror(node->right);

    /* swap the pointers in this node */
    temp        = node->left;
    node->left  = node->right;
    node->right = temp;
  }
}
```



mirror(2)
mirror(1)

Node = 2

(1) Call Mirror for left-subtree i.e., Mirror(left-subtree)
(2) Call Mirror for right-subtree i.e., Mirror(right-subtree)
(3) Swap left and right subtrees.
temp = left-subtree
left-subtree = right-subtree
right-subtree = temp



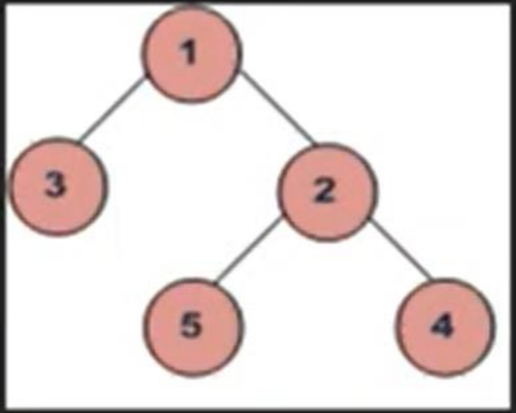
```
mirror(5)
mirror(2)
mirror(1)
```

```
void mirror(struct node* node)
{
  if (node==NULL)
    return;
  else
  {
    struct node* temp;

    /* do the subtrees */
    mirror(node->left);
    mirror(node->right);

    /* swap the pointers in this node */
    temp         = node->left;
    node->left  = node->right;
    node->right = temp;
  }
}
```

Node = 5

(1) Call Mirror for left-subtree i.e., Mirror(left-subtree)
(2) Call Mirror for right-subtree i.e., Mirror(right-subtree)
(3) Swap left and right subtrees.

```
temp = left-subtree
left-subtree = right-subtree
right-subtree = temp
```



```
void mirror(struct node* node)
{
  if (node==NULL)
    return;
  else
  {
    struct node* temp;

    /* do the subtrees */
    mirror(node->left);
    mirror(node->right);

    /* swap the pointers in this node */
    temp         = node->left;
    node->left  = node->right;
    node->right = temp;
  }
}
```

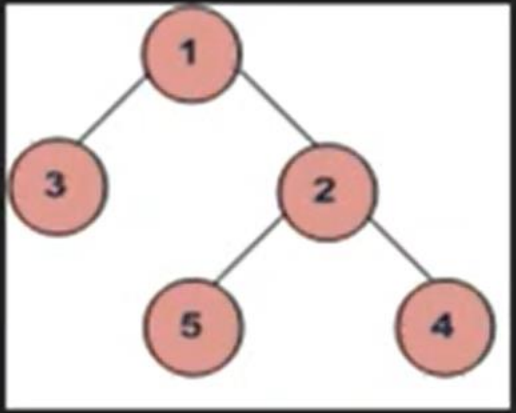
mirror(NULL)
mirror(5)
mirror(2)
mirror(1)

Node = NULL

Left subtree of 5 is null
so return to 5

(1) Call Mirror for left-subtree i.e., Mirror(left-subtree)
(2) Call Mirror for right-subtree i.e., Mirror(right-subtree)
(3) Swap left and right subtrees.

```
temp = left-subtree
left-subtree = right-subtree
right-subtree = temp
```



```
void mirror(struct node* node)
{
  if (node==NULL)
    return;
  else
  {
    struct node* temp;

    /* do the subtrees */
    mirror(node->left);
    mirror(node->right);

    /* swap the pointers in this node */
    temp        = node->left;
    node->left  = node->right;
    node->right = temp;
  }
}
```

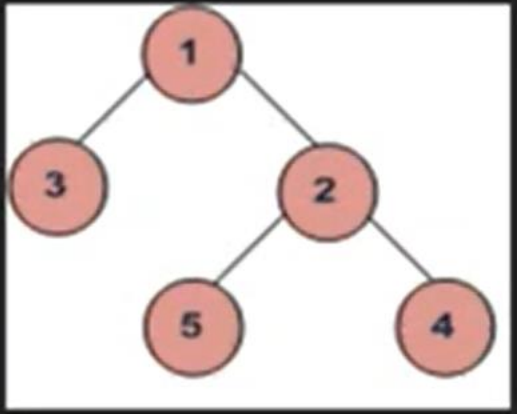
mirror(NULL)
mirror(5)
mirror(2)
mirror(1)

Node = NULL

Right subtree of 5 is null so return to 5

(1) Call Mirror for left-subtree i.e., Mirror(left-subtree)
(2) Call Mirror for right-subtree i.e., Mirror(right-subtree)
(3) Swap left and right subtrees.
    temp = left-subtree
    left-subtree = right-subtree
    right-subtree = temp

```
void mirror(struct node* node)
{
  if (node==NULL)
    return;
  else
  {
    struct node* temp;

    /* do the subtrees */
    mirror(node->left);
    mirror(node->right);

    /* swap the pointers in this node */
    temp          = node->left;
    node->left  = node->right;
    node->right = temp;
  }
}
```



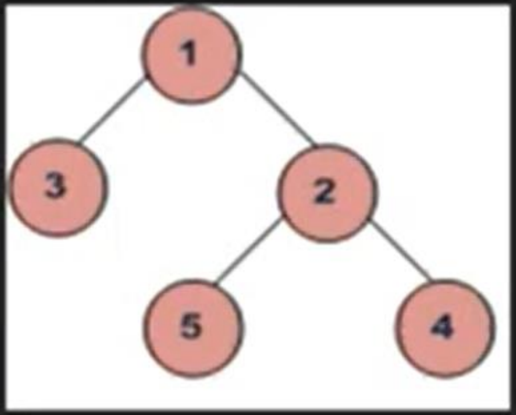
mirror(5)
mirror(2)
mirror(1)

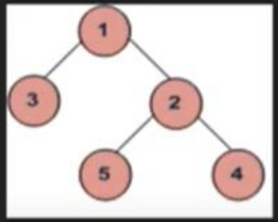
Node = NULL

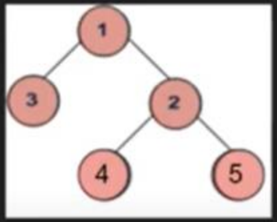
As both children are null nothing will be swapped

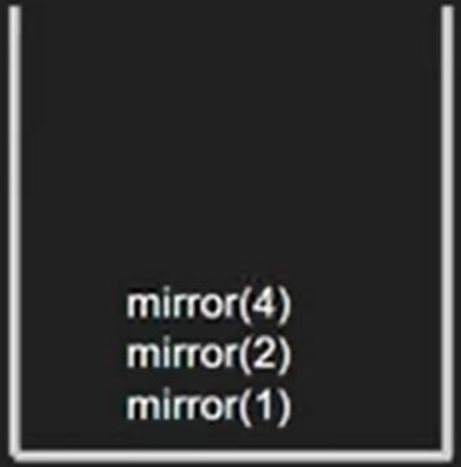
(1) Call Mirror for left-subtree i.e., Mirror(left-subtree)
(2) Call Mirror for right-subtree i.e., Mirror(right-subtree)
(3) Swap left and right subtrees.
    temp = left-subtree
    left-subtree = right-subtree
    right-subtree = temp

```
void mirror(struct node* node)
{
  if (node==NULL)
    return;
  else
  {
    struct node* temp;

    /* do the subtrees */
    mirror(node->left);
    mirror(node->right);

    /* swap the pointers in this node */
    temp        = node->left;
    node->left  = node->right;
    node->right = temp;
  }
}
```





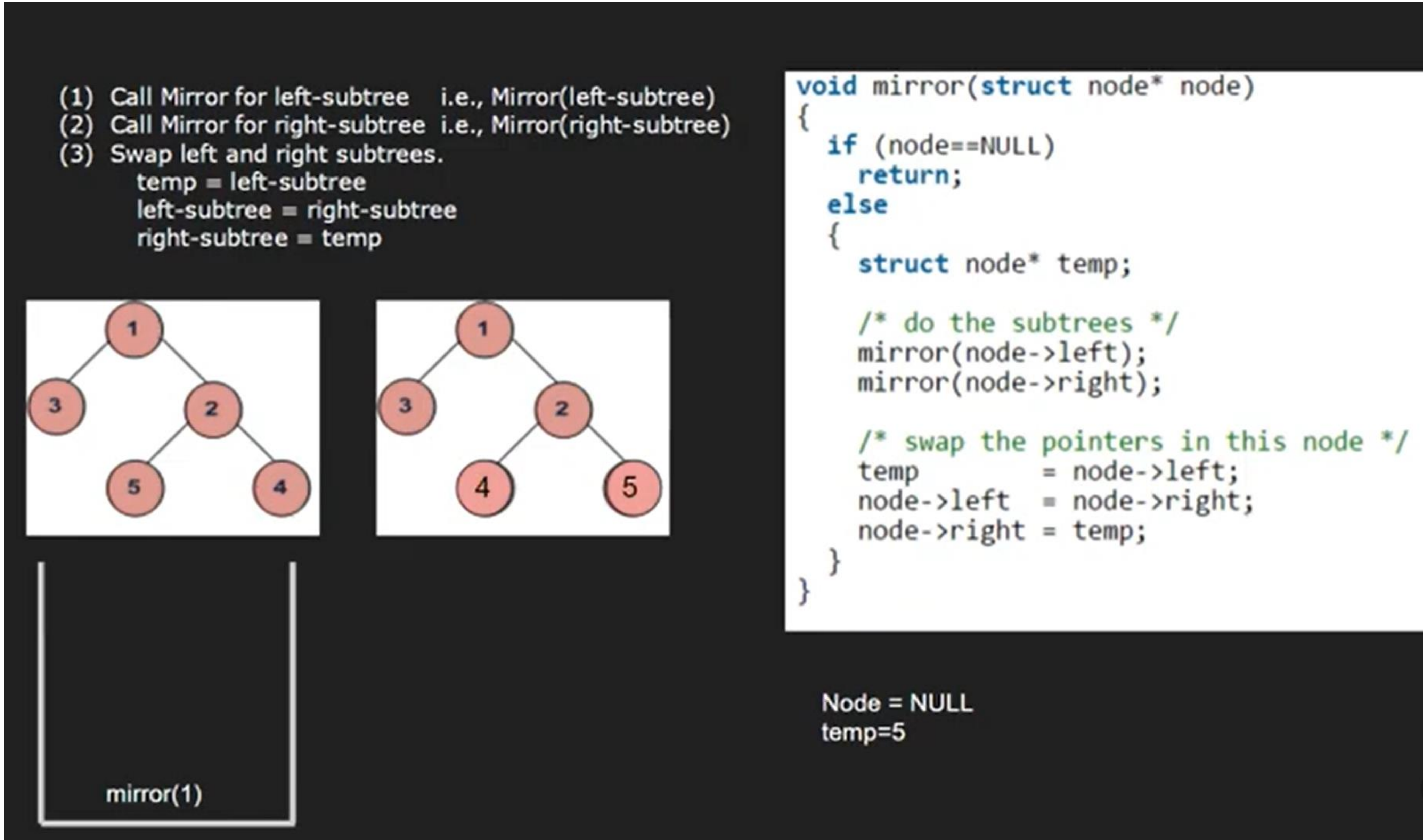




Node = 4

Node 4 has no children so nothing swapped. After completing both the children now go to 2 and swap 4 and 5

After execution of node 2 return to 1, its both children are traversed so swap them



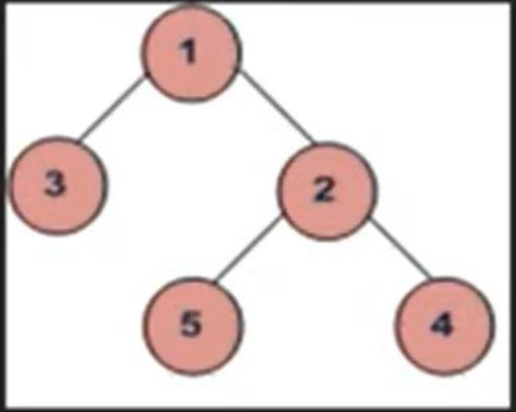
(1) Call Mirror for left-subtree   i.e., Mirror(left-subtree)
(2) Call Mirror for right-subtree i.e., Mirror(right-subtree)
(3) Swap left and right subtrees.
    temp = left-subtree
    left-subtree = right-subtree
    right-subtree = temp

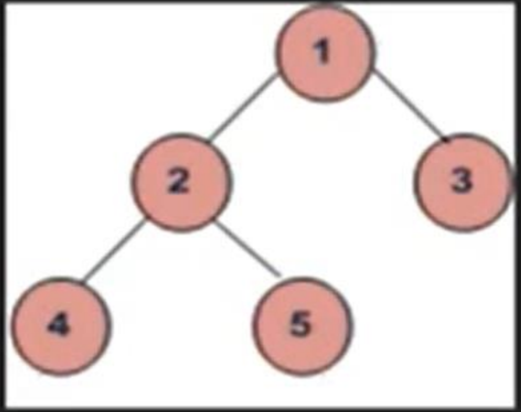
```
void mirror(struct node* node)
{
  if (node==NULL)
    return;
  else
  {
    struct node* temp;

    /* do the subtrees */
    mirror(node->left);
    mirror(node->right);

    /* swap the pointers in this node */
    temp        = node->left;
    node->left  = node->right;
    node->right = temp;
  }
}
```

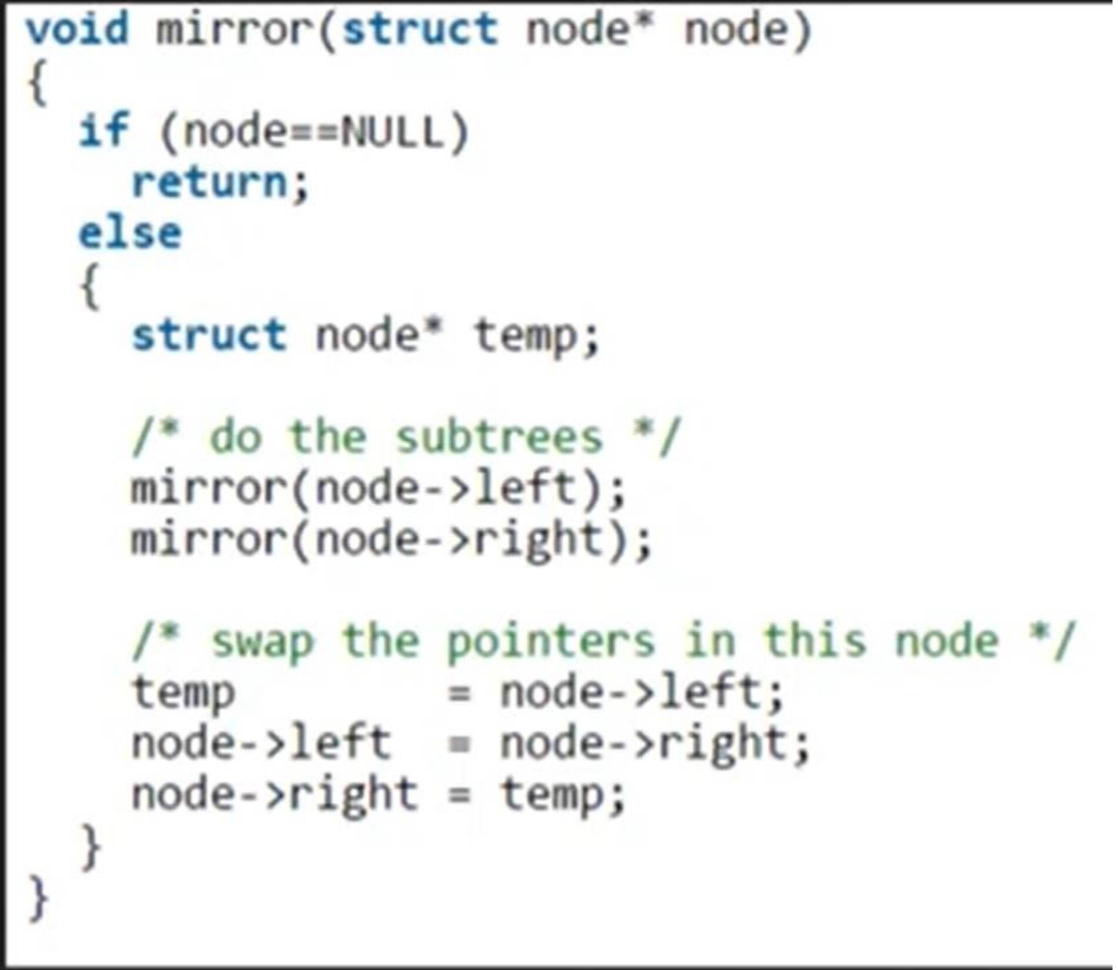
Node = NULL
temp=5

(1) Call Mirror for left-subtree i.e., Mirror(left-subtree)
(2) Call Mirror for right-subtree i.e., Mirror(right-subtree)
(3) Swap left and right subtrees.

```
    temp = left-subtree
    left-subtree = right-subtree
    right-subtree = temp
```



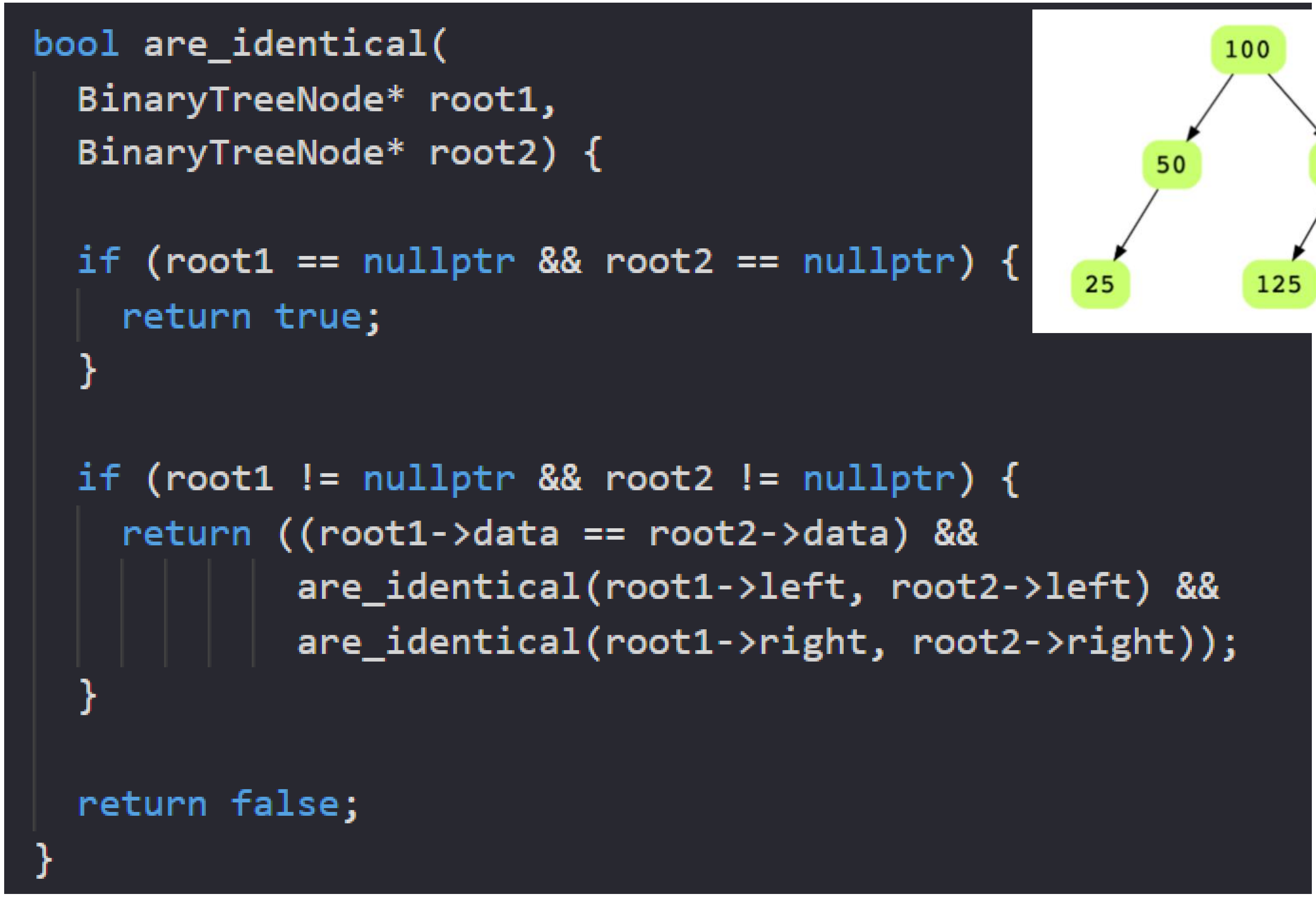




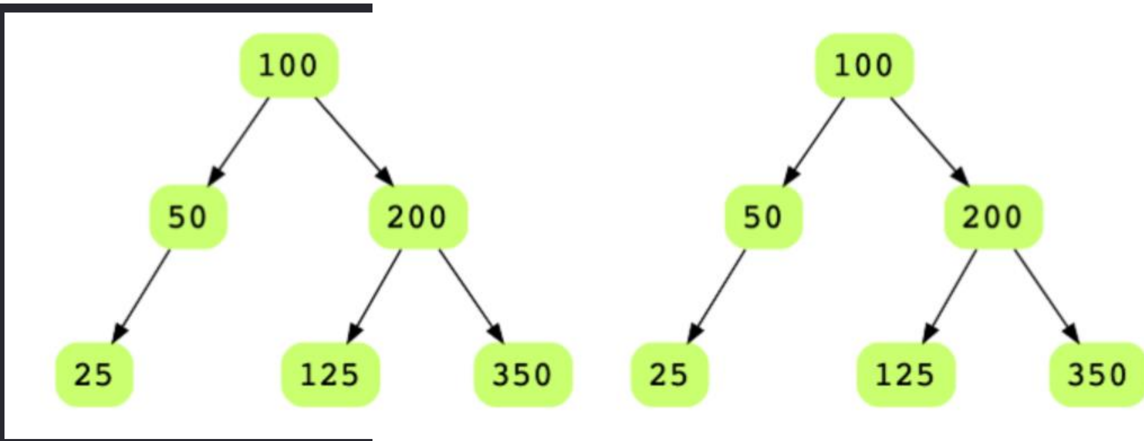
```
void mirror(struct node* node)
{
  if (node==NULL)
    return;
  else
  {
    struct node* temp;

    /* do the subtrees */
    mirror(node->left);
    mirror(node->right);

    /* swap the pointers in this node */
    temp        = node->left;
    node->left  = node->right;
    node->right = temp;
  }
}
```

Node = NULL
temp=3

If binary trees are identical



```
bool are_identical(
  BinaryTreeNode* root1,
  BinaryTreeNode* root2) {

  if (root1 == nullptr && root2 == nullptr) {
    return true;
  }

  if (root1 != nullptr && root2 != nullptr) {
    return ((root1->data == root2->data) &&
            are_identical(root1->left, root2->left) &&
            are_identical(root1->right, root2->right));
  }

  return false;
}
```



100
50
200
25
125
350
100
50
200
25
125
350