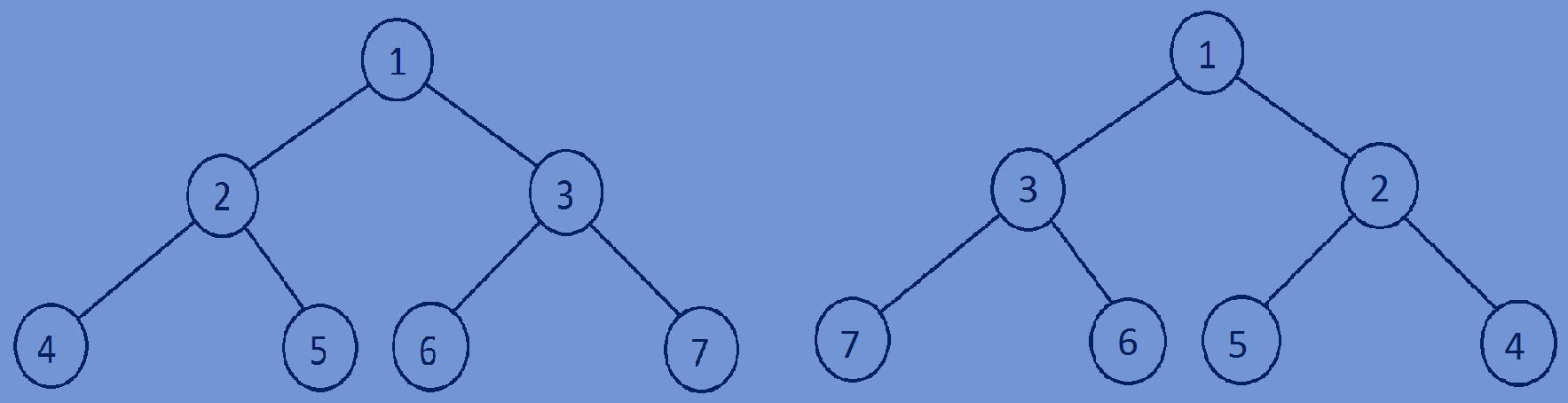
**Data Structures**

Finding Mirror Tree of a given Binary Tree

**Mirror of a binary tree:**

* A mirror tree of a binary tree is a binary tree in which left and right child of every

Node is interchanged except the leaf nodes.



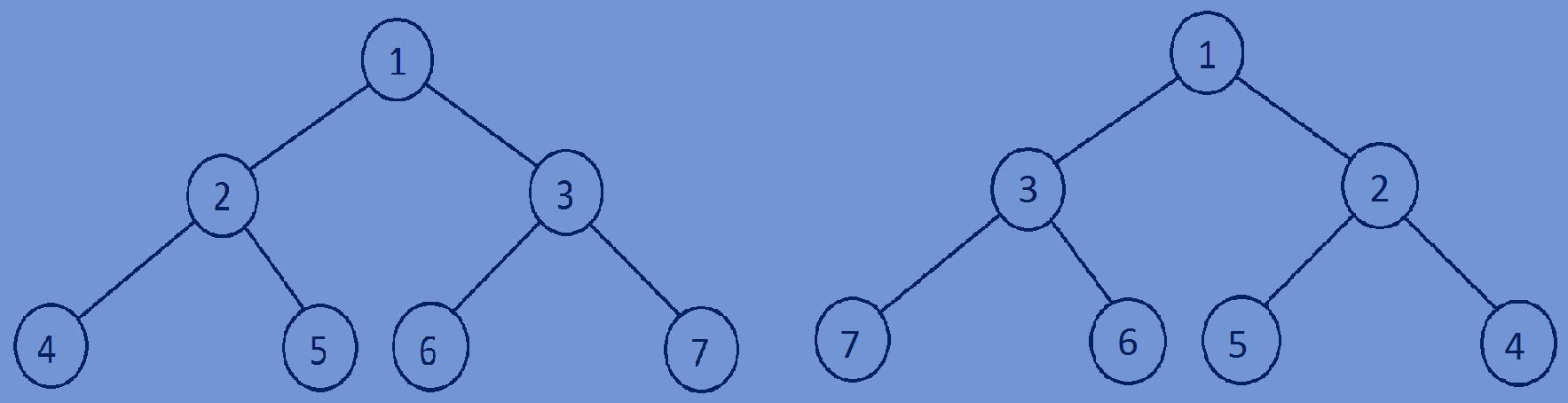
Given binary tree Mirror binary tree

**Finding the mirror of a given Binary Tree:**

* Write a recursive function to convert the given binary tree into its mirror tree.
* Traverse through each node of a binary tree and while traversing, at each

Node swap the left and right subtrees.

temp=tree->left;  
tree->left=tree->right;  
tree->right=temp;



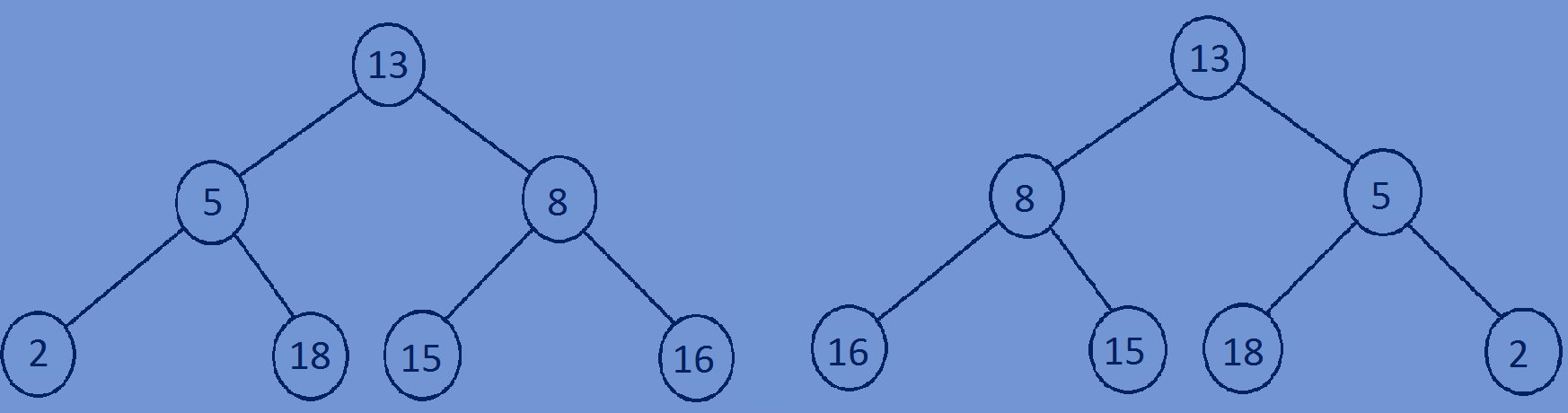
**Function for converting a given binary tree into its mirror tree:**

//finding mirror of a binary tree  
**void** MirrorTree(Btree \*tree){  
 **if**(tree==NULL){  
 **return**;  
 }  
 MirrorTree(tree->left);  
 MirrorTree(tree->right);  
 Btree \*temp;  
 temp=tree->left;  
 tree->left=tree->right;  
 tree->right=temp;  
}

**Whole program:**

#include<stdio.h>  
#include<stdlib.h>  
#include<time.h>  
//creating a node  
**typedef struct** Btree{  
 **int** data;  
 **struct** Btree \*left;  
 **struct** Btree \*right;  
}Btree;  
//creating new nodes  
Btree \*createnewnode(){  
 Btree \*newnode=(Btree\*)malloc(**sizeof**(Btree));  
 //generates random no between 1 and 20  
 newnode->data=rand()%20+1;  
 newnode->left=NULL;  
 newnode->right=NULL;  
 **return** newnode;  
}  
//finding mirror of a tree  
**void** MirrorTree(Btree \*tree){  
 **if**(tree==NULL){  
 **return**;  
 }  
 MirrorTree(tree->left);  
 MirrorTree(tree->right);  
 Btree \*temp;  
 temp=tree->left;  
 tree->left=tree->right;  
 tree->right=temp;  
}  
//print all the nodes of a tree in preorder fashion  
**void** preorder(Btree \*root){  
 **if**(root){  
 printf("%d ",root->data);  
 preorder(root->left);  
 preorder(root->right);  
 }  
}  
//main function  
**int** main(){  
 Btree \*tree=NULL;  
 //making time as seed   
 srand((**unsigned**)(time(NULL)));  
 tree=createnewnode();  
 tree->left=createnewnode();  
 tree->right=createnewnode();  
 tree->left->left=createnewnode();  
 tree->left->right=createnewnode();  
 tree->right->left=createnewnode();  
 tree->right->right=createnewnode();  
 preorder(tree);printf("\n");  
 MirrorTree(tree);  
 preorder(tree);  
 **return** 0;  
}

13 5 2 18 8 15 16

13 8 16 15 5 18 2