Data Structures

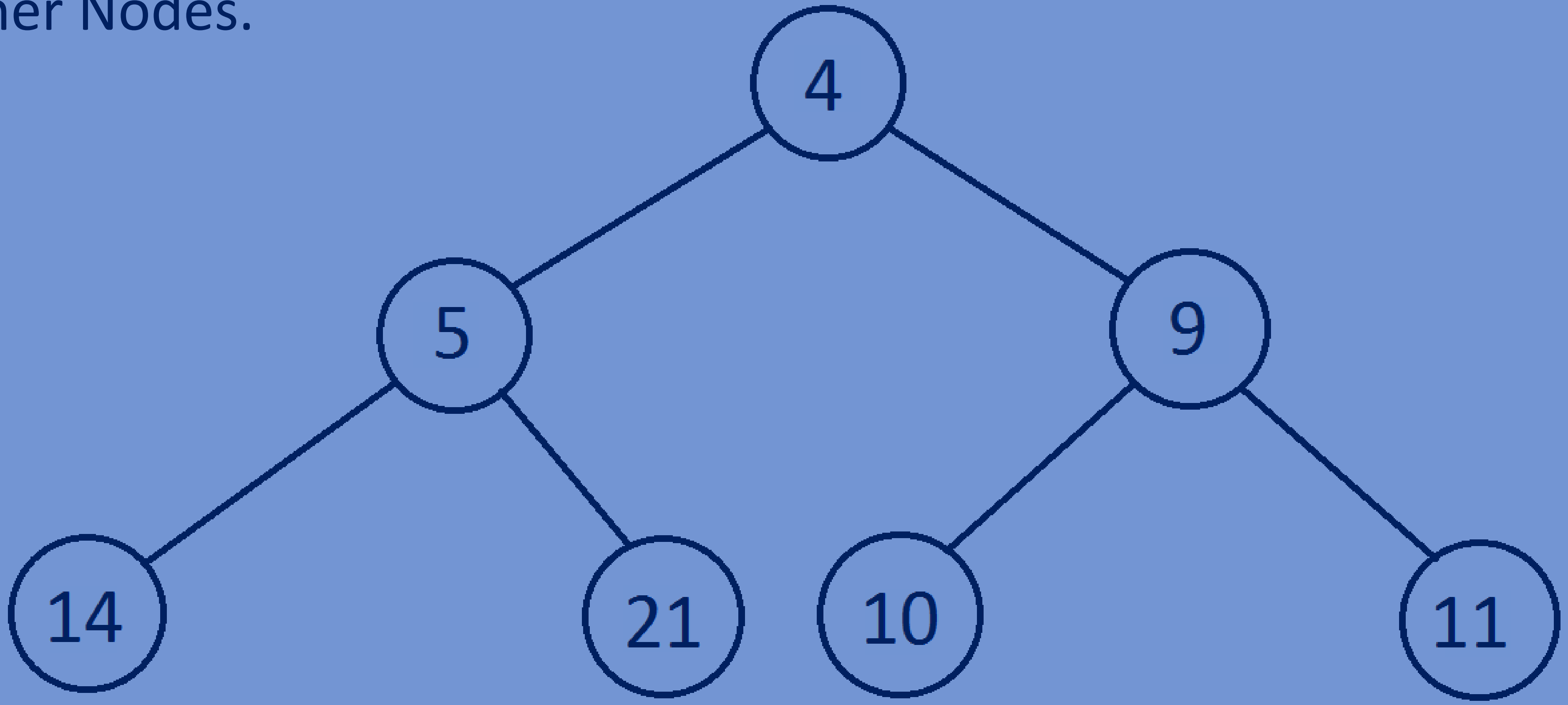
Implementation of Min Heap

Min binary heap:

- In a min heap the data of the root node must be less than or equal to its children

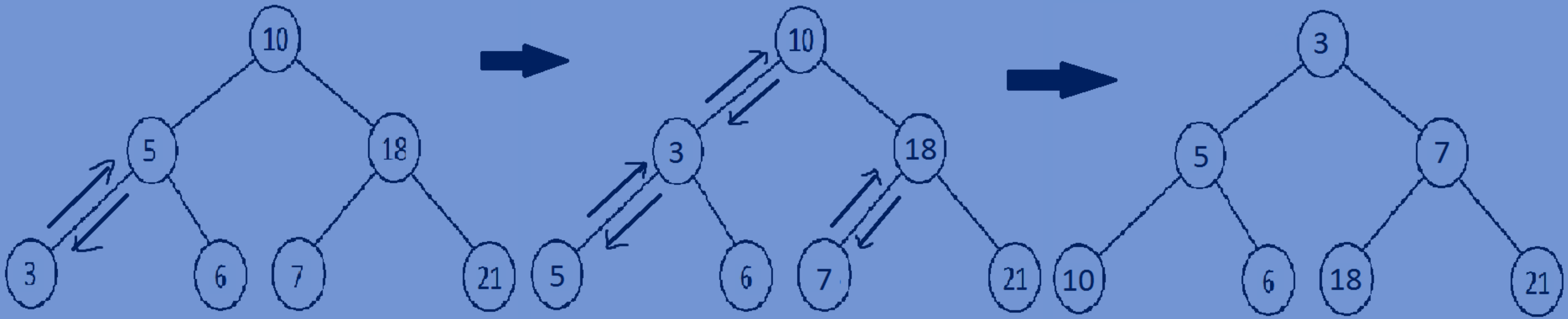
Nodes data and this property holds for all the nodes of the min binary heap.

- In case of min heap the root node will have the smallest element compared to all other Nodes.



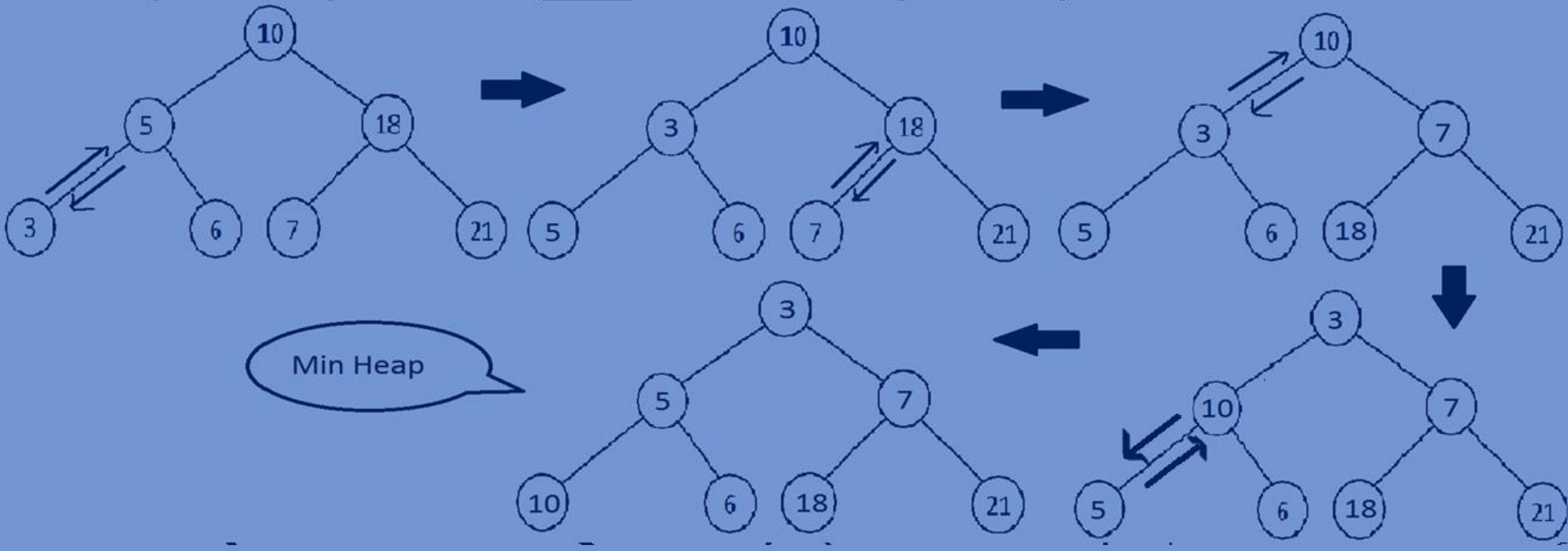
Implementation of Min binary heap:

- Given a complete binary tree convert the complete binary tree into a min binary heap.
- For a given node, heapify the left subtree,similarly heapify the right subtree And after that if necessary bubble down the current node such that the tree Rooted with the current node satisfies min heap property.



Implementation of Min binary heap using a complete binary tree:

- First traverse through each node of a complete binary tree in preorder fashion and while traversing compare the Data in the left child of the node with data of the node and if data in left child is less than data in the parent node then swap the Data and again call the recursive function with left subtree,if the data in right Child is less than data in the parent node swap the data and again call the recursive function with the right subtree.

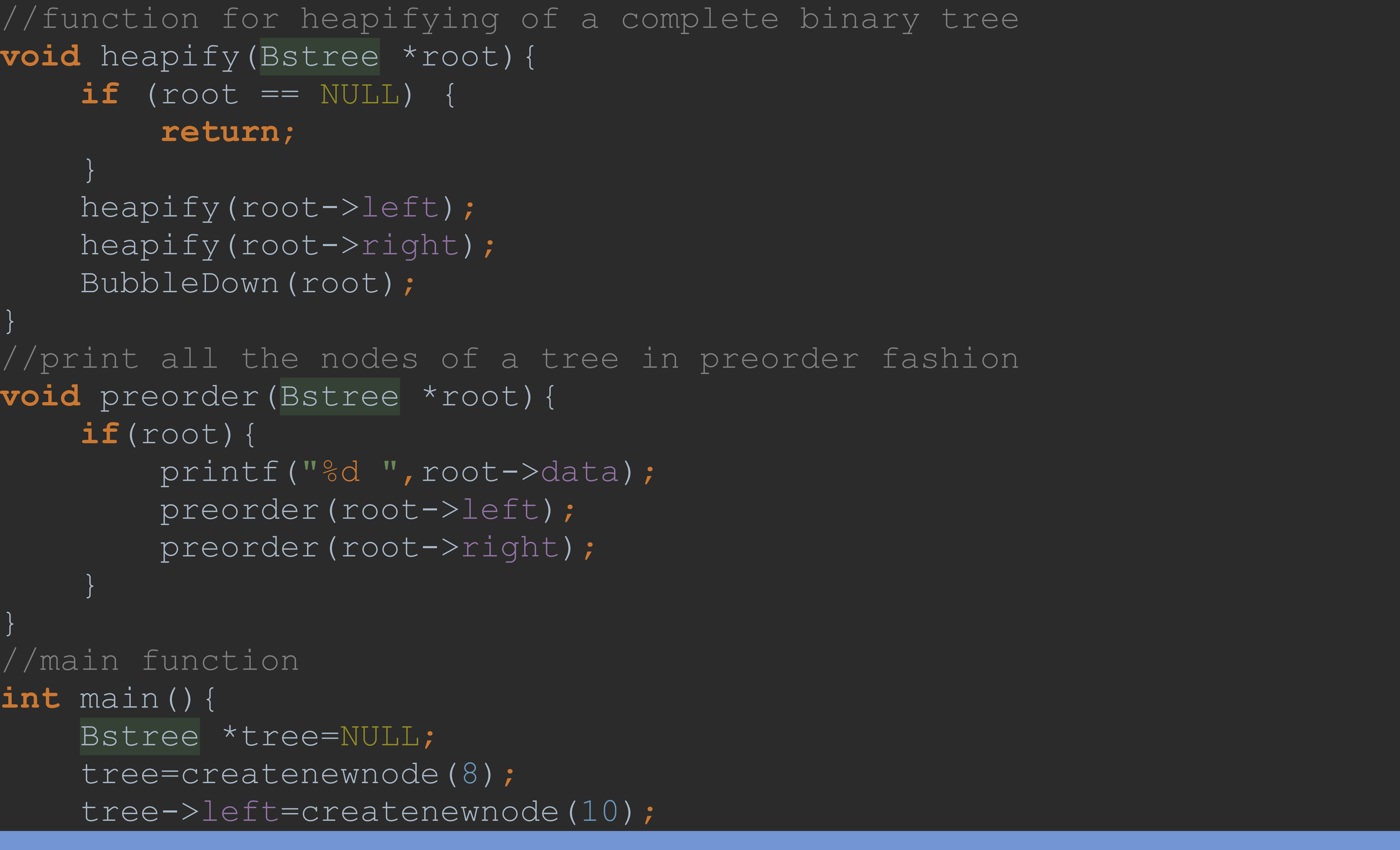


Function for heapifying a complete binary tree:

```
//function for heapifying of a complete binary tree
void heapify(Bstree *root){
    if (root == NULL) {
        return;
    }
    heapify(root->left);
    heapify(root->right);
    BubbleDown(root);
}
//recursive function for bubbling the nodes based on comparison.
void BubbleDown(Bstree *node){
    Bstree *smallestNode;
    if(node==NULL){
        return;
    }
    if(node->left!=NULL && node->left->data < node->data){
        smallestNode=node->left;
        swapData(node,smallestNode);
        BubbleDown(smallestNode);
    }
    if(node->right!=NULL && node->right->data < node->data){
        smallestNode=node->right;
        swapData(node,smallestNode);
        BubbleDown(smallestNode);
    }
}
//swaping data of two nodes
void swapData(Bstree *temp1,Bstree *temp2){
    int temp3;
    temp3=temp1->data;
    temp1->data=temp2->data;
    temp2->data=temp3;
}
```

Whole program:

```
#include<stdio.h>
#include<stdlib.h>
//creating a node
typedef  struct Bstree{
    int data;
    struct Bstree *left;
    struct Bstree *right;
}Bstree;
//creating new nodes
Bstree *createnewnode(int data){
    Bstree *newnode=(Bstree*)malloc(sizeof(Bstree));
    newnode->data=data;
    newnode->left=NULL;
    newnode->right=NULL;
    return newnode;
}
//swaping data of two nodes
void swapData(Bstree *temp1,Bstree *temp2){
    int temp3;
    temp3=temp1->data;
    temp1->data=temp2->data;
    temp2->data=temp3;
}
//recursive function for bubbling the nodes based on comparison.
void BubbleDown(Bstree *node){
    Bstree *smallestNode;
    if(node==NULL){
        return;
    }
    if(node->left!=NULL && node->left->data < node->data){
        smallestNode=node->left;
        swapData(node,smallestNode);
        BubbleDown(smallestNode);
    }
    if(node->right!=NULL && node->right->data < node->data){
        smallestNode=node->right;
        swapData(node,smallestNode);
        BubbleDown(smallestNode);
    }
}
```



```c
//function for heapifying of a complete binary tree
void heapify(Bstree *root){
    if (root == NULL) {
        return;
    }
    heapify(root->left);
    heapify(root->right);
    BubbleDown(root);
}
//print all the nodes of a tree in preorder fashion
void preorder(Bstree *root){
    if(root){
        printf("%d ",root->data);
        preorder(root->left);
        preorder(root->right);
    }
}
//main function
int main(){
    Bstree *tree=NULL;
    tree=createnewnode(8);
    tree->left=createnewnode(10);
    tree->right=createnewnode(9);
    tree->left->left=createnewnode(2);
    tree->left->right=createnewnode(7);
    tree->right->left=createnewnode(1);
    tree->right->right=createnewnode(2);
    preorder(tree);printf("\n");
    heapify(tree);
    preorder(tree);
    return 0;
}
```

Output:

```
8 10 2 7 9 1 3
1 7 10 8 2 9 3
```