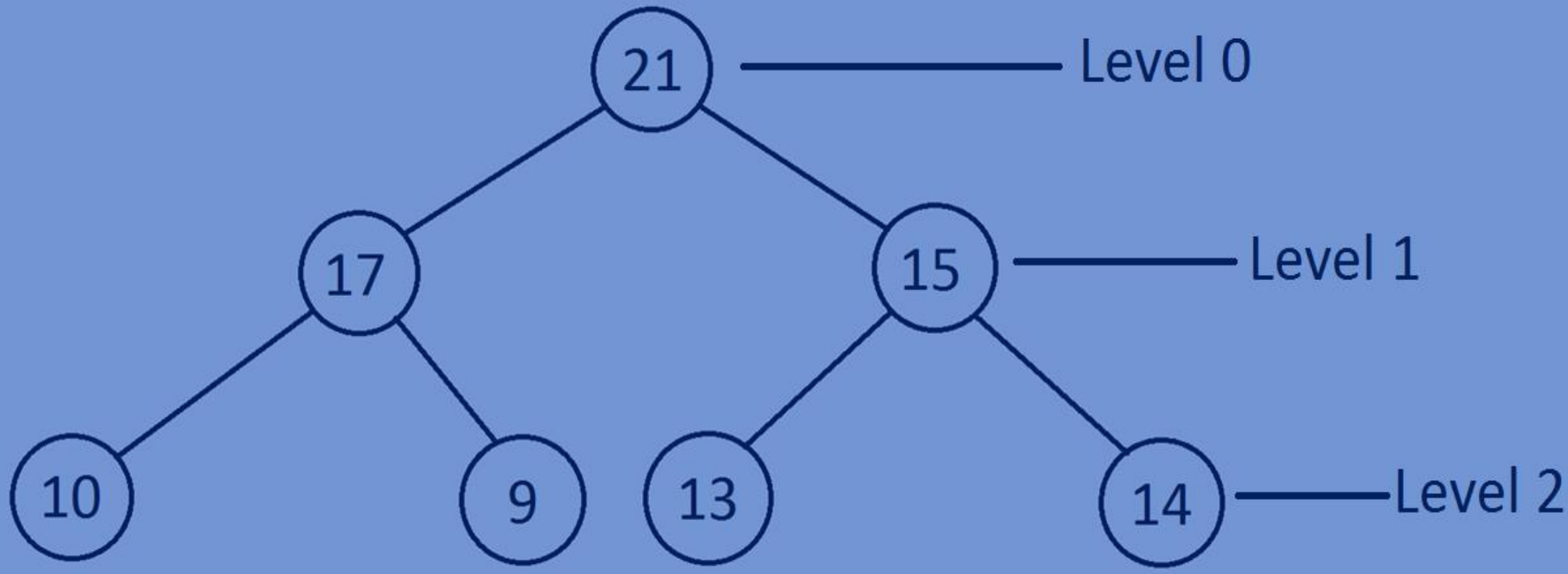
Data Structures

Printing the level of each node of a heap

Level of a node:

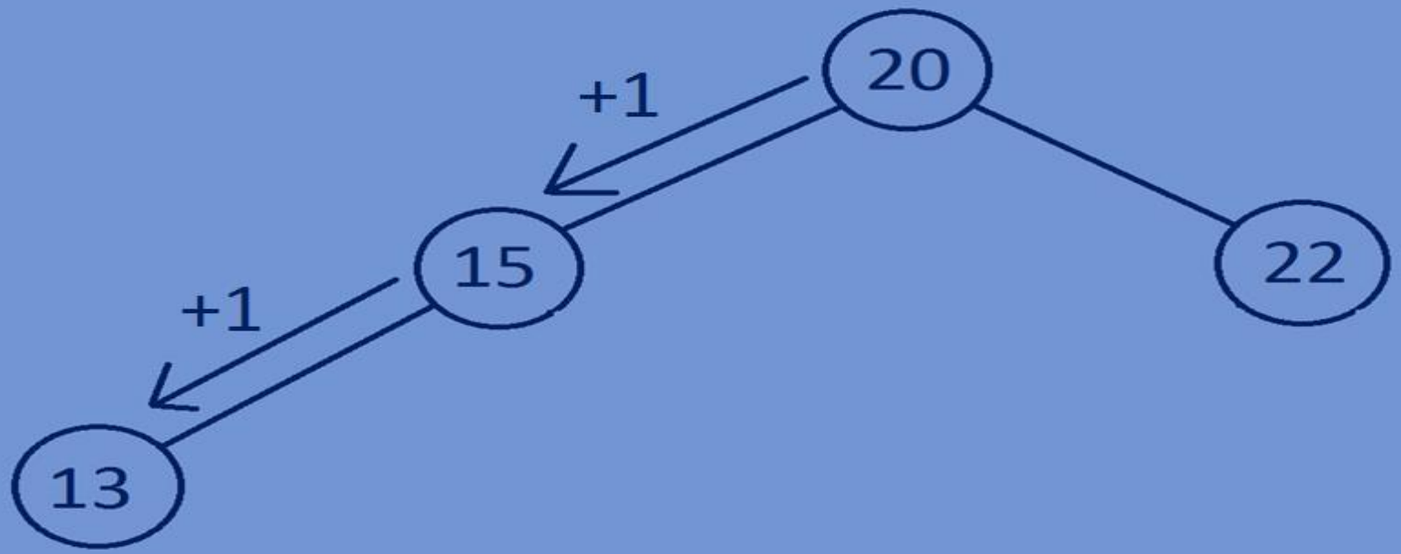
- the total no of edges between the root node and the current node is called as level of a node.
- Traverse through the tree and while traversing increment the level of the tree with one in each recursive call of the same function.



Printing level of each node:

- If root is equal to NULL then return -1;
- If root->data is equal to data return the level.
- Else call the function recursively incrementing the level and return the level.

```
int levelofnode=callevel(root->left,data,level+1);
if(levelofnode!=-1)
    return levelofnode;
return callevel(root->right,data,level+1);
```



Function for printing level of each node in a heap:

```
//recursive function for finding the level of a node
int callevel(Bstree *root,int data,int level){
    if(root==NULL){
        return -1;
    }
    if(root->data==data){
        return level;
    }
    int levelofnode=callevel(root->left,data,level+1);
    if(levelofnode!=-1){
        return levelofnode;
    }
    return callevel(root->right,data,level+1);
}
```

Whole program:

```c
#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;
}
//function for bubbling of data in root node
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);
    }
```

```
}
//function for heapifying a complete binary tree
void heapifyTree(Bstree *root){
    if (root == NULL) {
        return;
    }

    heapifyTree(root->left);
    heapifyTree(root->right);

    bubbleDown(root);
}
//recursive function for finding the level of a node
int callevel(Bstree *root,int data,int level){
    if(root==NULL){
        return -1;
    }
    if(root->data==data){
        return level;
    }
    int levelofnode=callevel(root->left,data,level+1);
    if(levelofnode!=-1){
```

```
        return levelofnode;
    }
    return callevel(root->right,data,level+1);
}
//print all the nodes of a tree in preorder fashion
void preorder(Bstree *root,Bstree *tree){
    if(root){
        printf("%d-%d  ",root->data,callevel(tree,root->data,0));
        preorder(root->left,tree);
        preorder(root->right,tree);
    }
}
//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(3);
```

```
    preorder(tree,tree);printf("\n");
    heapifyTree(tree);
    preorder(tree,tree);printf("\n");
    return 0;
}
```

Output:

8-0 10-1 2-2 7-2 9-1 1-2 3-2

1-0 7-1 10-2 8-2 2-1 9-2 3-2