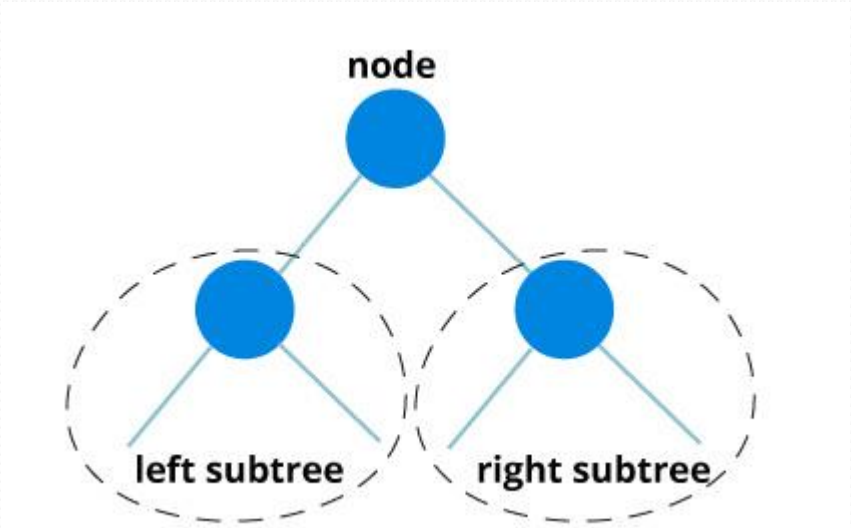
Tree Traversal

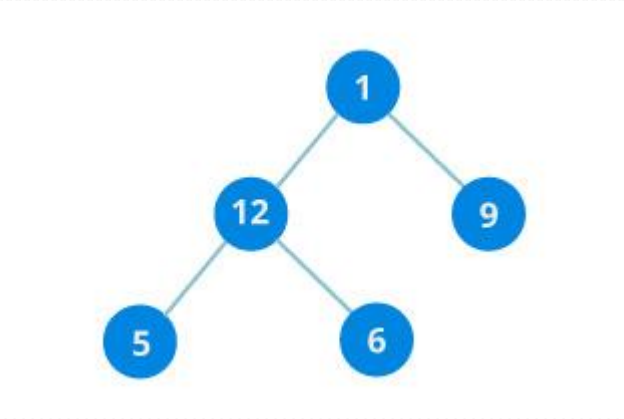
- Traversing a tree means visiting every node in the tree.
- You might for instance want to add all the values in the tree or find the largest one.
- For all these operations, you will need to visit each node of the tree.

Linear data structures like arrays, stacks, queues and linked list have only one way to read the data.

- But a hierarchical data structure like a tree can be traversed in different ways.

- The struct node pointed to by left and right might have other left and right children so we should think of them as sub-trees instead of sub-nodes.
- According to this structure, every tree is a combination of
  - ❑ A node carrying data
  - ❑ Two subtrees

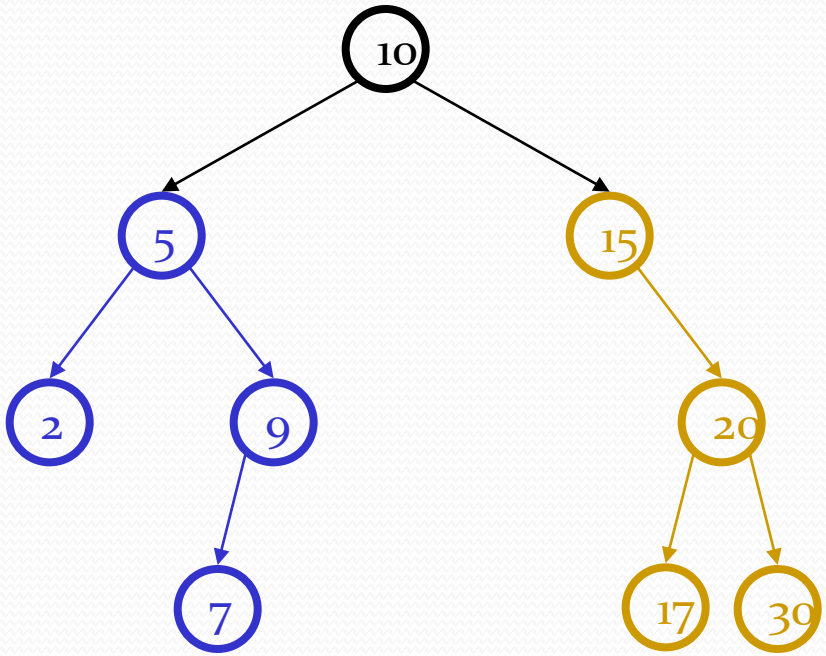




Inorder traversal

- visit left subtree
- visit node
- visit right subtree

- This method means traverse and print from the smallest to the largest values.
- We first need to print the roots left subtree, then we print the value of the root node finally print the value in the root's right subtree.



In order listing:

2→5→7→9→10→15→17→20→30

Inorder function print

```
void inorder(struct node* root)
{
    if(root == NULL) return;
    inorder(root->left);
    cout<<  root->data;
    inorder(root->right);
}
```

- The function **inorder**() takes the root of the binary tree as argument and prints the elements of the tree in **inorder**. It is a recursive function.

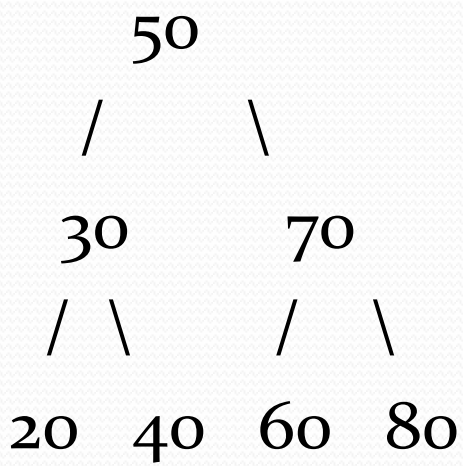
The complete code for Binary Search Tree insertion and searching in C programming language is posted below:

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   int data;
   struct node* left;
   struct node* right;
};

struct node* createNode(value){
   struct node* newNode new  node ;
   newNode->data = value;
   newNode->left = NULL;
   newNode->right = NULL;

   return newNode;
}

struct node* insert(struct node* root, int data)
{
   if (root == NULL) return createNode(data);

   if (data < root->data)
      root->left  = insert(root->left, data);
   else if (data > root->data)
      root->right = insert(root->right, data);

   return root;
}
```

```
void inorder(struct node* root)
{
   if(root == NULL) return;
   inorder(root->left);
   cout<<  root->data;
   inorder(root->right);
}



int main()
{
   struct node *root = NULL;
   root = insert(root, 8);
   insert(root, 3);
   insert(root, 1);
   insert(root, 6);
   insert(root, 7);
   insert(root, 10);
   insert(root, 14);
   insert(root, 4);

   inorder(root);
}
```

Homework:

Let us create following BST, then print its elements using Inorder procedure:



Postorder print function

Postorder method travers and print each node information after its left and right subtree:

1. visit all the nodes in the left subtree
2. visit the root node
3. visit all the nodes in the right subtree

```
Void postorder ()
{

  if (root ! = NULL)
   {
      Postorder (root-> left);          //Visit left subtree
      Postorder (root-> right);         //Visit right subtree
      cout<<root->data<<endl;           //Print data
   }
}
```