Binary search tree and Traversals – Dr.Saleena

```
#include <stdio.h>
#include <stdlib.h>

struct node {
   int data;

   struct node *leftChild;
   struct node *rightChild;
};

struct node *root = NULL;

void insert(int data)
{
   struct node *newnode = (struct node*) malloc(sizeof(struct node));
   struct node *current;
   struct node *parent;

   newnode->data = data;
   newnode->leftChild = NULL;
   newnode->rightChild = NULL;
   //if tree is empty
   if(root == NULL) {
      root = newnode;
   } else
   {
      current = root;
      parent = NULL;
      while(1)
                {
         parent = current;
        //go to to left of the tree
         if(data < parent->data)
                        {
            current = current->leftChild;
            //insert to the left
            if(current == NULL)
                                    {
               parent->leftChild = newnode;
               return;
            }
         }  //go to right of the tree
         else
                        {
            current = current->rightChild;
            //insert to the right
            if(current == NULL)
                                    {
               parent->rightChild = newnode;
               return;
            }
         } //end of else
      } //end of while
   } // if tree not empty
} // End of Insert Function

void pre_order_traversal(struct node* root)
{
  if(root != NULL)
  {
    printf("%d ",root->data);
    pre_order_traversal(root->leftChild);
    pre_order_traversal(root->rightChild);
  }
}

void inorder_traversal(struct node* root)
{
  if(root != NULL)
  {
    inorder_traversal(root->leftChild);
    printf("%d ",root->data);
    inorder_traversal(root->rightChild);
  }
}

void post_order_traversal(struct node* root)
{
  if(root != NULL)
  {
    post_order_traversal(root->leftChild);
    post_order_traversal(root->rightChild);
    printf("%d ", root->data);
  }
}

int main()
{
  int i;
  int array[7] = { 27, 14, 35, 10, 19, 31, 42 };

  for(i = 0; i < 7; i++)
    insert(array[i]);

  printf("\nPreorder traversal: ");
  pre_order_traversal(root);

  printf("\nInorder traversal: ");
  inorder_traversal(root);

  printf("\nPost order traversal: ");
  post_order_traversal(root);

  return 0;
}
```