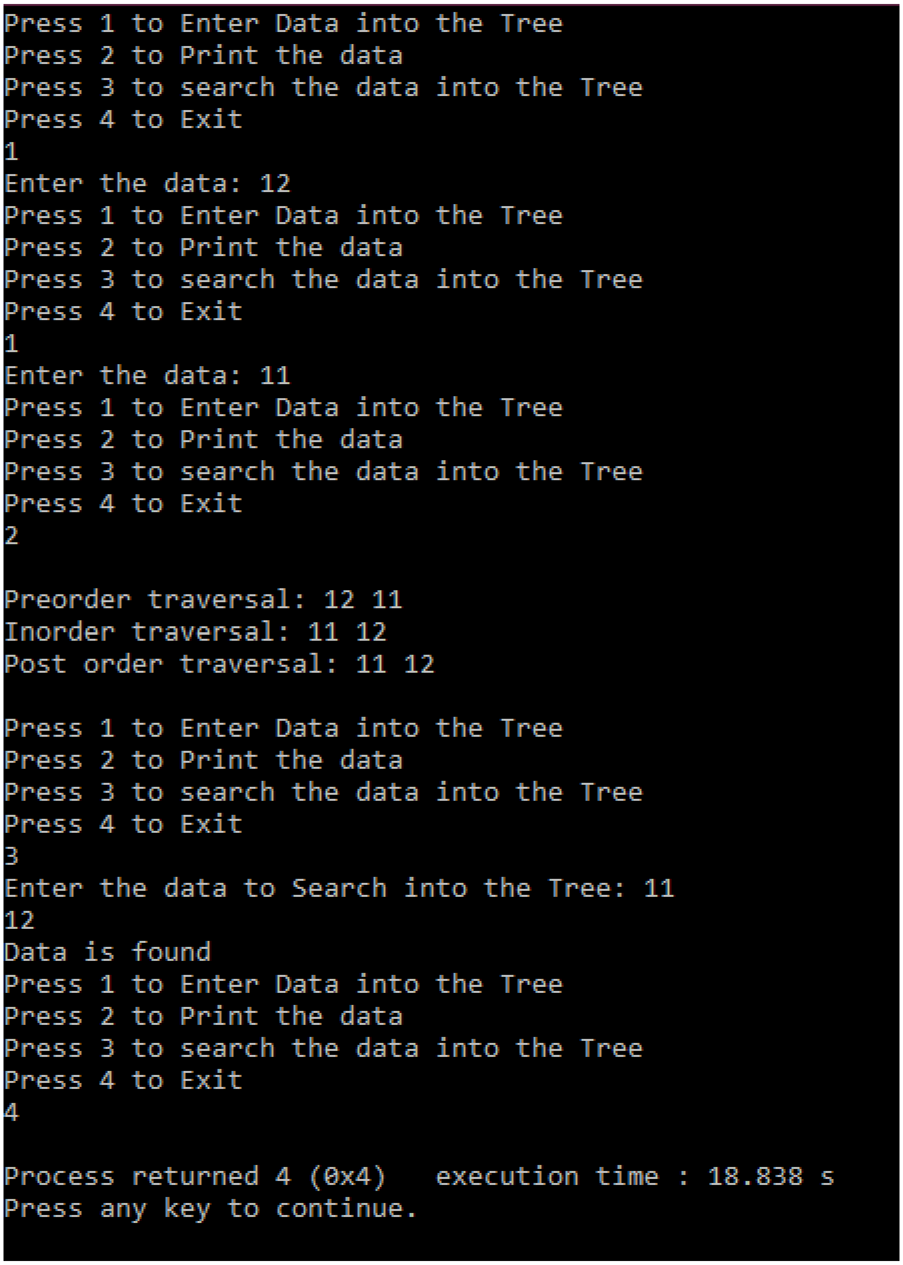
Data Structure Lab Assignment

19. Write a program in C to implement Tree Traversals.

```
#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 *tempNode = (struct node*) malloc(sizeof(struct node));
  struct node *current;
  struct node *parent;

  tempNode->data = data;
  tempNode->leftChild = NULL;
  tempNode->rightChild = NULL;


  if(root == NULL) {
    root = tempNode;
  } else {
    current = root;
    parent = NULL;
```

```
      while(1) {
         parent = current;


         if(data < parent->data) {
            current = current->leftChild;


            if(current == NULL) {
               parent->leftChild = tempNode;
               return;
            }
         }
         else {
            current = current->rightChild;


            if(current == NULL) {
               parent->rightChild = tempNode;
               return;
            }
         }
      }
   }
}

struct node* search(int data) {
   struct node *current = root;

   while(current->data != data) {
      if(current != NULL)
         printf("%d ",current->data);
      if(current->data > data) {
         current = current->leftChild;
      }

      else {
         current = current->rightChild;
      }


      if(current == NULL) {
         return current;
      }
   }

   return current;
}

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,data,choice,ser,val;

  do
  {
     printf("Press 1 to Enter Data into the Tree\nPress 2 to Print the data\nPress 3 to search the data into the Tree\nPress 4 to Exit\n");
     scanf("%d",&choice);
     switch(choice)
     {
     case 1:
      printf("Enter the data: ");
      scanf("%d",&data);
      insert(data);
      break;

     case 2:
      printf("\nPreorder traversal: ");
      pre_order_traversal(root);

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

      printf("\nPost order traversal: ");
      post_order_traversal(root);
      printf("\n\n");
      break;
     case 3:
      printf("Enter the data to Search into the Tree: ");
      scanf("%d",&ser);
      val = search(ser);
      if(val!=NULL)
         printf("\nData is found\n");
      else
         printf("\nData is not found\n");
      break;

     case 4:
      exit(4);
     default:
      printf("invalid Input\n");

     }

  }
  while(choice!=4);


}
```

OUTPUT



```
Press 1 to Enter Data into the Tree
Press 2 to Print the data
Press 3 to search the data into the Tree
Press 4 to Exit
1
Enter the data: 12
Press 1 to Enter Data into the Tree
Press 2 to Print the data
Press 3 to search the data into the Tree
Press 4 to Exit
1
Enter the data: 11
Press 1 to Enter Data into the Tree
Press 2 to Print the data
Press 3 to search the data into the Tree
Press 4 to Exit
2

Preorder traversal: 12 11
Inorder traversal: 11 12
Post order traversal: 11 12

Press 1 to Enter Data into the Tree
Press 2 to Print the data
Press 3 to search the data into the Tree
Press 4 to Exit
3
Enter the data to Search into the Tree: 11
12
Data is found
Press 1 to Enter Data into the Tree
Press 2 to Print the data
Press 3 to search the data into the Tree
Press 4 to Exit
4

Process returned 4 (0x4)   execution time : 18.838 s
Press any key to continue.
```