19005082001
4th Sem / Comp. Engg.

Data Structures Using C

Program -

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

typedef struct BST {
    int data;
    struct BST *lchild, *rchild;
} node;

void insert (node *, node *);
void inorder (node *);
void preorder (node *);
void postorder (node *);
node * search (node *, int, node **);

void main() {
    int choice;
    char ans = 'N';
    int key;
    node * new_node, *root, * tmp, *parent;
    node * get_node ();
    root = NULL;
    clrscr ();
```

```
printf ("\n Program for Binary Search Tree");
do{
    printf ("\n 1. Create ");
    printf ("\n 2. Search ");
    printf ("\n 3. Recursive Traversals ");
    printf ("\n 4. Exit ");
    printf ("\n Enter your choice : ");
    scanf ("%d", &choice);

  switch (choice) {
  case 1 :
    do {
      new_node = get_node ();
      printf ("\n Enter The Element");
      scanf ("%d", & new_node -> data);

      if (root == NULL) /* Tree is not Created */
        root = new_node;
      else
        insert (root, new_node);

      printf ("\n Want To enter More Elements?
        (y/n)");
      ans = getch ();
    } while (ans == 'y');
    break;

case 2:
    printf ("\n Enter Elements to be searched:");
    scanf ("%d", & key);

    tmp = search (root, key, & parent);
```

```
        printf("\n Parent of node %d is %d", temp
          ->data, parent->data);
        break;

case 3:
    if (root == NULL)
        printf("Tree Is Not Created");
    else {
        printf("\n The Inorder display: ");
        inorder(root);
        printf("\n The Preorder display: ");
        preorder(root);
        printf("\n The Postorder display: ");
        postorder(root);
    }
    break;
    }
} while (choice != 4);
}
/*
  Get new Node
*/
node * get_node() {
    node * temp;
    temp = (node *) malloc (sizeof(node));
    temp -> lchild = NULL;
    temp -> rchild = NULL;
    return temp;
}
/*
  This function is for creating a binary search
tree */
```

```
void insert (node * root, node * new_node) {
  if (new_node -> data < root -> data) {
    if (root -> lchild == NULL)
      root -> lchild = new_node;
    else
      insert (root -> lchild, new_node);
  }
  if (new_node -> data > root -> data) {
    if (root -> rchild == NULL)
      root -> rchild = new_node;
    else
      insert (root -> rchild, new_node);
  }
}
/* this function is for searching the node
from binary Search Tree */
node * search (node* root, int key, node **parent)
{
  node* temp;
  temp = root;
  while (temp != NULL) {
    if (temp -> data == key) {
      printf ("\n The %d Element is present",
temp -> data);
      return temp;
    }
    *parent = temp;

    if (temp -> data > key)
      temp = temp -> lchild;
    else
      temp = temp -> rchild; }
```

```
    return NULL;
}
/*
 This function displays the tree in inorder
  fashion */
void inorder (node * temp) {
  if (temp != NULL) {
     inorder (temp -> lchild);
     printf ("%d", temp -> data);
     inorder (temp -> rchild);
   }
  }
/* This function displays the tree in
 preorder fashion */
  void preorder (node * temp) {
    if (temp != NULL) {
     printf ("%d", temp -> data);
     preorder (temp -> lchild);
     preorder (temp -> rchild);
    }
  }
/* This function displays the tree in
    postorder fashion */
void postorder (node * temp ) {
   if (temp != NULL) {
     postorder (temp -> lchild);
     postorder (temp -> rchild);
     printf ("%d", temp -> data);
   }
}
```