## 9. Implementation of Binary Search Tree

**Program :**

```
#include <stdio.h>

#include <stdlib.h>

struct node {
  struct node * left;
  int element;
  struct node * right;
};
typedef struct node Node;
Node * Insert(Node * Tree, int e);
Node * Find(Node * Tree, int e);
Node * FindMin(Node * Tree);
void Display(Node * Tree);
Node * Delete(Node * Tree, int e);
Node * result;
int main() {
  Node * Tree = NULL;
  int n, i, e;
  printf("Enter number of nodes in the tree : ");
  scanf("%d", & n);
  printf("Enter the elements :\n");
  for (i = 1; i <= n; i++) {
    scanf("%d", & e);
    Tree = Insert(Tree, e);
  }
  printf("Tree elements in inorder :\n");
  Display(Tree);
  int choice;
  do {
    printf("\n1.Insert\n2.Delete\n3.Search\n4.Display\n5.Exit\n");
    printf("Enter your choice : ");
    scanf("%d", & choice);
    switch (choice) {
    case 1:
      printf("Enter the value to be inserted: ");
      scanf("%d", & e);
      Tree = Insert(Tree, e);
      break;
    case 2:
      printf("Enter the value to be deleted: ");
      scanf("%d", & e);
      Tree = Delete(Tree, e);
      break;
    case 3:
      printf("Enter the value to find: ");
      scanf("%d", & e);
      result = Find(Tree, e);
      if (result == NULL)
        printf("Element is not found...!");
      else
        printf("Element is found...!");
      printf("\n");
      break;
    case 4:
      printf("Tree elements in inorder :\n");
      Display(Tree);
      break;
    case 5:
      break;
    default:
      printf("Retry again....\n");
      break;
    }
  } while (choice != 5);
  return 0;
}
Node * Insert(Node * Tree, int e) {
  Node * NewNode = malloc(sizeof(Node));
  if (Tree == NULL) {
    NewNode -> element = e;
    NewNode -> left = NULL;
    NewNode -> right = NULL;
    Tree = NewNode;
  } else if (e < Tree -> element)
    Tree -> left = Insert(Tree -> left, e);
  else if (e > Tree -> element)
    Tree -> right = Insert(Tree -> right, e);
  return Tree;
}
void Display(Node * Tree) {
  if (Tree != NULL) {
    Display(Tree -> left);
    printf("%d\t", Tree -> element);
    Display(Tree -> right);
  }
}
Node * Delete(Node * Tree, int e) {
  Node * TempNode = malloc(sizeof(Node));
  if (e < Tree -> element) {
    Tree -> left = Delete(Tree -> left, e);
  } else if (e > Tree -> element) {
    Tree -> right = Delete(Tree -> right, e);
  } else if (Tree -> left && Tree -> right) {
    TempNode = FindMin(Tree -> right);
    Tree -> element = TempNode -> element;
    Tree -> right = Delete(Tree -> right, Tree -> element);
  } else {
    TempNode = Tree;
    if (Tree -> left == NULL)
      Tree = Tree -> right;
    else if (Tree -> right == NULL)
      Tree = Tree -> left;
    free(TempNode);
  }
  return Tree;
}
Node * FindMin(Node * Tree) {
  if (Tree != NULL) {
    if (Tree -> left == NULL)
      return Tree;
    else
      FindMin(Tree -> left);
  }
}
Node * Find(Node * Tree, int e) {
  if (Tree == NULL)
    return NULL;
  else if (e < Tree -> element)
    return Find(Tree -> left, e);
  else if (e > Tree -> element)
    return Find(Tree -> right, e);
  else
    return Tree;
}
```

**Output :**

```
Enter number of nodes in the tree : 5
Enter the elements :
2
34
32
15
65
Tree elements in inorder :
2     15    32    34    65
1.Insert
2.Delete
3.Search
4.Display
5.Exit
Enter your choice : 1
Enter the value to be inserted: 23

1.Insert
2.Delete
3.Search
4.Display
5.Exit
Enter your choice : 2
Enter the value to be deleted: 34

1.Insert
2.Delete
3.Search
4.Display
5.Exit
Enter your choice : 3
Enter the value to find: 15
Element is found...!

1.Insert
2.Delete
```

```
3.Search
4.Display
5.Exit
Enter your choice : 3
Enter the value to find: 33
Element is not found...!

1.Insert
2.Delete
3.Search
4.Display
5.Exit
Enter your choice : 4
Tree elements in inorder :
2    15    23   32    65
1.Insert
2.Delete
3.Search
4.Display
5.Exit
Enter your choice : 5
```