EXPERIMENT –9 [Implementation of Binary Trees]

Dated: 26.10.2023

1. Write a program to insert an element, delete an element and search an element in the Binary Search Tree.

Source data:

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

struct node* createNode(int data){
   struct node *n;
   n = (struct node *) malloc(sizeof(struct node));
   n->data = data;
   n->left = NULL;
   n->right = NULL;
   return n;
}

void insert(struct node *root, int key){
  struct node *prev = NULL;
  while(root!=NULL){
     prev = root;
     if(key==root->data){
        printf("Cannot insert %d, already in BST", key);
```

```
            return;
        }
        else if(key<root->data){
            root = root->left;
        }
        else{
            root = root->right;
        }
    }
    struct node* new = createNode(key);
    if(key<prev->data){
        prev->left = new;
    }
    else{
        prev->right = new;
    }
}
void inOrder(struct  node* root){
    if(root!=NULL){
        inOrder(root->left);
        printf("%d ", root->data);
        inOrder(root->right);
    }
}
struct node *inOrderPredecessor(struct node* root){
    root = root->left;
    while (root->right!=NULL)
    {
        root = root->right;
```

```
    }
    return root;
}
struct node *deleteNode(struct node *root, int value){
    struct node* iPre;
    if (root == NULL){
        return NULL;
    }
    if (root->left==NULL&&root->right==NULL){
        free(root);
        return NULL;
    }
    if (value < root->data){
        root-> left = deleteNode(root->left,value);
    }
    else if (value > root->data){
        root-> right = deleteNode(root->right,value);
    }
    else{
        iPre = inOrderPredecessor(root);
        root->data = iPre->data;
        root->left = deleteNode(root->left, iPre->data);
    }
    return root;
}
struct node * search(struct node* root, int key){
    if(root==NULL){
        return NULL;
    }
```

```
    if(key==root->data){
        return root;
    }
    else if(key<root->data){
        return search(root->left, key);
    }
    else{
        return search(root->right, key);
    }
}
int main(){
    struct node *p = createNode(5);
    struct node *p1 = createNode(3);
    struct node *p2 = createNode(6);
    struct node *p3 = createNode(1);
    struct node *p4 = createNode(4);

    p->left = p1;
    p->right = p2;
    p1->left = p3;
    p1->right = p4;

    insert(p, 16);
    printf("%d\n", p->right->right->data);

    inOrder(p);
    printf("\n");
    deleteNode(p, 3);
    inOrder(p);
```

```
    struct node* n = search(p, 10);
    if(n!=NULL){
    printf("\nFound: %d", n->data);
    }
    else{
       printf("\nElement not found");
    }
    return 0;
}
```

Output:

16

1 3 4 5 6 16

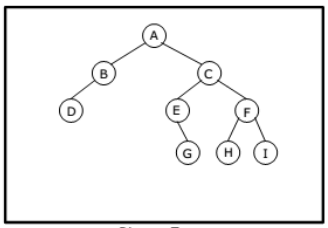
1 4 5 6 16

Element not found

2. Study and implement the Binary Tree and perform following three types of traversals in the given Binary Tree:

- Pre-Order Traversal
- Post Order Traversal
- In order Traversal

Example:



Binary Tree

- Preorder traversal yields: A, B, D, C, E, G, F, H, I
- Postorder traversal yields: D, B, G, E, H, I, F, C, A
- Inorder traversal yields: D, B, A, E, G, C, H, F, I
- Level order traversal yields: A, B, C, D, E, F, G, H, I

Pre, Post, Inorder and level order Traversing

Source code:

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

struct node {
   int data;
   struct node* left;
   struct node* right;
};
struct node* createNode(int data) {
   struct node* n = (struct node*)malloc(sizeof(struct node));
   n->data = data;
   n->left = NULL;
   n->right = NULL;
   return n;
}
void preOrder(struct node* root) {
   if (root != NULL) {
      printf("%c ", (char)root->data);
      preOrder(root->left);
      preOrder(root->right);
   }
}
void postOrder(struct node* root) {
   if (root != NULL) {
      postOrder(root->left);
      postOrder(root->right);
      printf("%c ", (char)root->data);
   }
```

```
}
void inOrder(struct node* root) {
   if (root != NULL) {
      inOrder(root->left);
      printf("%c ", (char)root->data);
      inOrder(root->right);
   }
}
int main() {
   struct node* p = createNode(97);  // ASCII for 'a'
   struct node* p1 = createNode(98); // ASCII for 'b'
   struct node* p2 = createNode(99); // ASCII for 'c'
   struct node* p3 = createNode(100); // ASCII for 'd'
   struct node* p4 = createNode(101); // ASCII for 'e'
   struct node* p5 = createNode(102); // ASCII for 'f'
   struct node* p6 = createNode(103); // ASCII for 'g'
   struct node* p7 = createNode(104); // ASCII for 'h'
   struct node* p8 = createNode(105); // ASCII for 'i'
   p->left = p1;
   p->right = p2;
   p1->left = p3;
   p2->left = p4;
   p2->right = p5;
   p4->right = p6;
   p5->left = p7;
   p5->right = p8;

   printf("Inorder Traversal: ");
   inOrder(p);
```

```
    printf("\n");
    printf("Preorder Traversal: ");
    preOrder(p);
    printf("\n");
    printf("Postorder Traversal: ");
    postOrder(p);
    printf("\n");

    return 0;
}
```

Output:

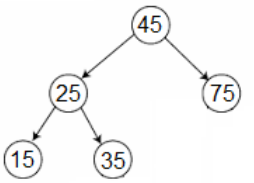
Inorder Traversal: d b a e g c h f i

Preorder Traversal: a b d c e g f h i

Postorder Traversal: d b g e h i f c a

3.Given a pre order traversal sequence of Binary Search Tree, construct the corresponding Binary Search Tree.

Binary search Tree - Preorder Traversal



Preorder traversal:

45, 25, 15, 35, 75

Source code:

```
#include <stdio.h>

#include <stdlib.h>

#include <limits.h>


struct TreeNode {
```

```
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};

struct TreeNode* createNode(int value) {
    struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    newNode->data = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

struct TreeNode* constructBST(int preOrder[], int* index, int min, int max, int size) {
    if (*index >= size) {
        return NULL;
    }
    int value = preOrder[*index];
    if (value < min || value > max) {
        return NULL;
    }

    struct TreeNode* root = createNode(value);
    (*index)++;
    root->left = constructBST(preOrder, index, min, value - 1, size);
```

```
    root->right = constructBST(preOrder, index, value, max, size);
    return root;
}

void inOrderTraversal(struct TreeNode* root) {
    if (root) {
        inOrderTraversal(root->left);
        printf("%d ", root->data);
        inOrderTraversal(root->right);
    }
}
int main() {
    int preOrder[] = {8, 5, 1, 7, 10, 12};
    int size = sizeof(preOrder) / sizeof(preOrder[0]);
    int index = 0;
    struct TreeNode* bstRoot = constructBST(preOrder, &index, INT_MIN, INT_MAX, size);
    printf("In-order Traversal of Constructed BST: ");
    inOrderTraversal(bstRoot);
    printf("\n");
    return 0;
}
```

Output:

In-order Traversal of Constructed BST: 1 5 7 8 10 12