```
#include <stdio.h>
#include <stdlib.h>
int lvl = -1;
int arr[16];
typedef struct BST
{
    int data;
    struct BST *left;
    struct BST *right;
} node;
int search_Seq(int key);
int get_right_child(int index);
void insertArray(int key, int item);
void deleteArray(int item);
void deleteArray(int item);
int get_right_child(int index);
int get_left_child(int index);
void preorderArray(int indx);
void postorderArray(int indx);
void inorderArray(int indx);
void arrayTree();
node *create();
void insert(node *, node *);
void preorder(node *);
void inorder(node *);
void postorder(node *);
node *Searchpar(node *, int);
void insert(node *root, node *temp);
node *Searchpar(node *root, int val);
int delete_tree(node *root, int val);
void preorder(node *root);
void inorder(node *root);
void postorder(node *root);
void listTree();
int main()
{
    int op;
choice:
    printf("Which binary tree representation do u want to
use?\n");
    printf("1 - Array (Binary Tree) \t\t 2 - Linked List
(Binary Search Tree)\n");
    scanf("%d", &op);
    if (op == 1)
```

```
        arrayTree();
    else if (op == 2)
        listTree();
    else
    {
        printf("Choose 1 or 2\n");
        goto choice;
    }
    return 0;
}
int search_Seq(int key)
{
    int i;
    for (i = 1; i <= 15; i++)
    {
        if (arr[i] == key)
            return i;
    }
    return 0;
}
void insertArray(int key, int item)
{
    int loc, op;
    loc = search_Seq(key);
    if (loc == 0)
        printf("No such key element found\n");
    else if ((arr[2 * loc] == 0) || (arr[(2 * loc) + 1] == 0))
    {
    choice:
        printf("Which child node do u want to enter data to ?
\n 1 - Left child \t\t 2 - Rightchild\n");
        scanf("%d", &op);
        if (op == 1)
        {
            if (arr[2 * loc] == 0)
            {
                arr[2 * loc] = item;
                lvl++;
            }
            else
                printf("Insertion not possible at left
child\n");
        }
        else if (op == 2)
```

```
        {
            if (arr[(2 * loc) + 1] == 0)
            {
                arr[(2 * loc) + 1] = item;
                lvl++;
            }
            else
                printf("Insertion not possible at right
child\n");
        }
        else
        {
            printf("Wrong choice\n");
            goto choice;
        }
    }
    else
        printf("Item cannot be inserted as leaf node\n");
}
void deleteArray(int item)
{
    int flag = 0;
    int loc = search_Seq(item);
    if (loc == 0)
        printf("No such key element found\n");
    else if ((arr[2 * loc] == 0) && (arr[(2 * loc) + 1] == 0))
    {
        flag = 1;
        arr[loc] = 0;
    }
    else
        printf("The node to be deleted is not leaf node\n");
    if (flag == 0)
        printf("No deletion\n");
    else
        printf("Deleted\n");
}
int get_right_child(int index)
{
    if (arr[index] != 0 && ((2 * index) + 1) <= 15)
        return (2 * index) + 1;
    return -1;
}
int get_left_child(int index)
```

```
{
    if (arr[index] != 0 && (2 * index) <= 15)
        return 2 * index;
    return -1;
}
void preorderArray(int indx)
{
    if (indx > 0 && arr[indx] != 0)
    {
        printf("%d\t", arr[indx]);
        preorderArray(get_left_child(indx));
        preorderArray(get_right_child(indx));
    }
}
void postorderArray(int indx)
{
    if (indx > 0 && arr[indx] != 0)
    {
        postorderArray(get_left_child(indx));
        postorderArray(get_right_child(indx));
        printf("%d\t", arr[indx]);
    }
}
void inorderArray(int indx)
{
    if (indx > 0 && arr[indx] != 0)
    {
        inorderArray(get_left_child(indx));
        printf("%d\t", arr[indx]);
        inorderArray(get_right_child(indx));
    }
}
void arrayTree()
{
    int op, item, key, i;
    char op2 = 'n';
    arr[0] = 0;
choice:
    printf("Which operation do u want to perform?\n");
    printf("1 - Insertion \t\t 2 - Deletion\t\t 3 -
Traversal\n");
    scanf("%d", &op);
    switch (op)
    {
```

```
    case 1:
        if (lvl == -1)
        {
            printf("Enter the item to insert as root node\n");
            scanf("%d", &item);
            arr[1] = item;
            lvl++;
        }
        else
        {
            printf("Enter the key after which it is to be
inserted\n");
            scanf("%d", &key);
            printf("Enter the item to be insert\n");
            scanf("%d", &item);
            insertArray(key, item);
        }
        printf("Continue operations ? (y/n)\n");
        fflush(stdin);
        scanf("%c", &op2);
        if (op2 == 'y')
            goto choice;
        break;
    case 2:
        printf("Enter the item to be deleted\n");
        scanf("%d", &item);
        deleteArray(item);
        printf("Continue operations ? (y/n)\n");
        fflush(stdin);
        scanf("%c", &op2);
        if (op2 == 'y')
            goto choice;
        break;
    case 3:
    traverse:
        printf("\nChoose a traversal :\n1 - Inorder\t2 -
Preorder\t3 - Postorder\n");
        scanf("%d", &op);
        if (op == 1)
            inorderArray(1);
        else if (op == 2)
            preorderArray(1);
        else if (op == 3)
            postorderArray(1);
```

```
        else
        {
            printf("Wrong choice\n");
            goto traverse;
        }
        printf("\nContinue operations ? (y/n)\n");
        fflush(stdin);
        scanf("%c", &op2);
        if (op2 == 'y')
            goto choice;
        break;
    default:
        printf("Choose a proper option\n");
        goto choice;
    }
}
void listTree()
{
    int option, val;
    char ch;
    node *root = NULL, *temp;
    do
    {
        temp = create();
        if (root == NULL)
            root = temp;
        else
            insert(root, temp);
        printf("\nDo you want to enter more(y/n)?");
        getchar();
        scanf("%c", &ch);
    } while (ch == 'y' | ch == 'Y');
    do
    {
        printf("1-preorder\t2:inorder\t3:postorder\t4:Deletion
\n ");
        scanf("%d", &option);
        if (option == 1)
            preorder(root);
        else if (option == 2)
            inorder(root);
        else if (option == 3)
            postorder(root);
        else
```

```
        {
            printf("Enter the element you want to delete:");
            scanf("%d", &val);
            delete_tree(root, val);
        }
        printf("\nPerform operations? (y/n)?");
        getchar();
        scanf("%c", &ch);
    } while (ch == 'y' | ch == 'Y');
}
node *create()
{
    node *temp;
    printf("Enter the node values for binary search tree:");
    temp = (node *)malloc(sizeof(node));
    scanf("%d", &temp->data);
    temp->left = temp->right = NULL;
    return temp;
}
void insert(node *root, node *temp)
{
    if (temp->data < root->data)
    {
        if (root->left != NULL)
            insert(root->left, temp);
        else
            root->left = temp;
    }
    if (temp->data > root->data)
    {
        if (root->right != NULL)
            insert(root->right, temp);
        else
            root->right = temp;
    }
}
node *Searchpar(node *root, int val)
{
    node *parent, *ptr1, *ptr2;
    parent = root;
    if (root->data != val)
    {
        ptr1 = root->left;
        ptr2 = root->right;
```

```
        if (ptr1 != NULL)
        {
            Searchpar(ptr1, val);
        }
        else
        {
            parent = NULL;
        }
        if (ptr2 != NULL)
        {
            Searchpar(ptr2, val);
        }
        else
        {
            parent = NULL;
        }
    }
    else
    {
        return parent;
    }
}
int delete_tree(node *root, int val)
{
    node *ptr, *par, *ptr1, *ptr2;
    int flag = 0;
    ptr = root;
    while (ptr != NULL && flag == 0)
    {
        if (val < ptr->data)
        {
            par = ptr;
            ptr = ptr->left;
        }
        else if (val > ptr->data)
        {
            par = ptr;
            ptr = ptr->right;
        }
        else if (ptr->data == val)
        {
            flag = 1;
        }
    }
```

```
    if (flag == 0)
    {
        printf("Item doesn't exist.");
    }
    if (ptr->left == NULL && ptr->right == NULL)
    {
        if (par->left == ptr)
        {
            par->left = NULL;
        }
        else
        {
            par->right = NULL;
        }
    }
}
void preorder(node *root)
{
    if (root != NULL)
    {
        printf("%d ", root->data);
        preorder(root->left);
        preorder(root->right);
    }
}
void inorder(node *root)
{
    if (root != NULL)
    {
        inorder(root->left);
        printf("%d ", root->data);
        inorder(root->right);
    }
}
void postorder(node *root)
{
    if (root != NULL)
    {
        postorder(root->left);
        postorder(root->right);
        printf("%d ", root->data);
    }
}
```