

## (i) Insertion, deletion, and traversal in BST.

```
#include<iostream>

using namespace std;
struct node
{
    int key;
    struct node *left, *right;
};

struct node *newNode(int item)
{
    struct node *temp = (struct node *)malloc(sizeof(struct node));
    temp->key = item;
    temp->left = temp->right = NULL;
    return temp;
}

void inorder(struct node *root)
{
    if (root != NULL)
    {
        inorder(root->left);
        cout<< root->key<<endl;
        inorder(root->right);
    }
}

struct node* insert(struct node* node, int key)
{
    /* If the tree is empty, return a new node */
    if (node == NULL) return newNode(key);
    /* Otherwise, recur down the tree */
    if (key < node->key)
        node->left = insert(node->left, key);
    else
        node->right = insert(node->right, key);

    return node;
}
```

```
}

struct node * minValueNode(struct node* node)
{
    struct node* current = node;

    while (current && current->left != NULL)
        current = current->left;

    return current;
}

struct node* deleteNode(struct node* root, int key)
{
    // base case
    if (root == NULL) return root;

    if (key < root->key)
        root->left = deleteNode(root->left, key);
    else if (key > root->key)
        root->right = deleteNode(root->right, key);

    else
    {
        if (root->left == NULL)
        {
            struct node *temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL)
        {
            struct node *temp = root->left;
            free(root);
            return temp;
        }

        // node with two children: Get the inorder successor (smallest
        // in the right subtree)
        struct node* temp = minValueNode(root->right);
        // Copy the inorder successor's content to this node
        root->key = temp->key;
        // Delete the inorder successor
        root->right = deleteNode(root->right, temp->key);
    }
}
```

```
    }  
    return root;  
}  
  
int main()  
{  
  
    struct node *root = NULL;  
  
    int ch, val, val2;  
    cout << "1)Insert\n";  
    cout << "2)Delete\n";  
    cout << "3)Display\n";  
    cout << "4)Exit\n";  
    do  
    {  
  
        cout << "Enter choice : " << endl;  
        cin >> ch;  
        switch (ch)  
        {  
            case 1:  
                cout << "Input for insertion: " << endl;  
                cin >> val;  
                root = insert(root, val);  
                break;  
            case 2:  
                cout << "element for deletion: " << endl;  
                cin >> val2;  
                deleteNode(root, val2);  
                break;  
            case 3:  
                cout<<"Inorder traversal of the modified tree \n";  
                inorder(root);  
  
                break;  
            case 4:  
                cout << "Exit\n";  
                break;  
            default:  
                cout << "Incorrect!\n";  
        }  
    }
```

```
    } while (ch != 4);  
    return 0;  
}
```

Output:

```
/DS/ds lab/assgn08/" && g++ bst.cpp -o bst && "/Users/amzamani/Des  
ktop/sem3/DS/ds lab/assgn08/"bst  
1)Insert  
2)Delete  
3)Display  
4)Exit  
Enter choice :  
1  
Input for insertion:  
50  
Enter choice :  
1  
Input for insertion:  
30  
Enter choice :  
1  
Input for insertion:  
20  
Enter choice :  
1  
Input for insertion:  
40  
Enter choice :  
3  
Inorder traversal of the modified tree  
20  
30  
40  
50  
Enter choice :  
2  
element for deletion:  
20  
Enter choice :  
3  
Inorder traversal of the modified tree  
30  
40  
50
```

## (ii) Insertion, deletion, and traversal in threaded binary tree.

```
#include <stdio.h>
#include <stdlib.h>
enum marker
{
    CHILD,
    THREAD
};
struct tbstNode
{
    int data;
    struct tbstNode *link[2];
    int marker[2];
};
struct tbstNode *root = NULL;
struct tbstNode *createNode(int data)
{
    struct tbstNode *newNode;
    newNode = (struct tbstNode *)malloc(sizeof(struct tbstNode));
    newNode->data = data;
    newNode->link[0] = newNode->link[1] = NULL;
    newNode->marker[0] = newNode->marker[1] = THREAD;
    return newNode;
}
void insertion(int data)
{
    struct tbstNode *parent, *newNode, *temp;
    int path;
    if (!root)
    {
        root = createNode(data);
        return;
    }
    parent = root;
```

```
/* find the location to insert the new node */
while (1)
{
    if (data == parent->data)
    {
        printf("Duplicates Not Allowed\n");
        return;
    }
    path = (data > parent->data) ? 1 : 0;
    if (parent->marker[path] == THREAD)
        break;
    else
        parent = parent->link[path];
}
/*
* newnode's left points to predecessor and
* right to successor
*/
newNode = createNode(data);
newNode->link[path] = parent->link[path];
parent->marker[path] = CHILD;
newNode->link[!path] = parent;
parent->link[path] = newNode;
return;
}

void delete (int data)
{
    struct tbstNode *current, *parent, *temp;
    int path;
    parent = root;
    current = root;
    /* search the node to delete */
    while (1)
    {
        if (data == current->data)
            break;
        path = (data > current->data) ? 1 : 0;
        if (current->marker[path] == THREAD)
        {
            printf("Given data is not available!!\n");
            return;
        }
    }
}
```

```
    parent = current;
    current = current->link[path];
}
if (current->marker[1] == THREAD)
{
    if (current->marker[0] == CHILD)
    {
        /* node with single child */
        temp = current->link[0];
        while (temp->marker[1] == CHILD)
        {
            temp = temp->link[1];
        }
        temp->link[1] = current->link[1];
        if (current == root)
        {
            root = current->link[0];
        }
        else
        {
            parent->link[path] = current->link[0];
        }
    }
    else
    {
        /* deleting leaf node */
        if (current == root)
        {
            root = NULL;
        }
        else
        {
            parent->link[path] = current->link[path];
            parent->marker[path] = THREAD;
        }
    }
}
else
{
    temp = current->link[1];
    /*
* node with two child - whose right child has
```



```
* no left child
*/

if (temp->marker[0] == THREAD)
{
    temp->link[0] = current->link[0];
    temp->marker[0] = current->marker[0];
    if (temp->marker[0] == CHILD)
    {
        struct tbstNode *x = temp->link[0];
        while (x->marker[1] == CHILD)
        {
            x = x->link[1];
        }
        x->link[1] = temp;
    }
    if (current == root)
    {
        root = temp;
    }
    else
    {
        printf("path: %d data:%d\n", path, parent->data);
        parent->link[path] = temp;
    }
}
else
{
    /* node with two child */
    struct tbstNode *child;
    while (1)
    {
        child = temp->link[0];
        if (child->marker[0] == THREAD)
            break;
        temp = child;
    }
    if (child->marker[1] == CHILD)
        temp->link[0] = child->link[1];
    else
    {
        temp->link[0] = child;
        temp->marker[0] = THREAD;
    }
}
```

```
    }
    child->link[0] = current->link[0];
    /* update the links */
    if (current->marker[0] == CHILD)
    {
        struct tbstNode *x = current->link[0];
        while (x->marker[1] == CHILD)
            x = x->link[1];
        x->link[1] = child;
        child->marker[0] = CHILD;
    }
    child->link[1] = current->link[1];
    child->marker[1] = CHILD;
    if (current == root)
        root = child;
    else
        parent->link[path] = child;
}
}
/* deallocation */
free(current);
return;
}

void traversal()
{
    struct tbstNode *myNode;
    if (!root)
    {
        printf("Threaded Binary Tree Not Exists!!\n");
        return;
    }
    myNode = root;
    while (1)
    {
        while (myNode->marker[0] == CHILD)
        {
            myNode = myNode->link[0];
        }
        printf("%d ", myNode->data);
        myNode = myNode->link[1];
        if (myNode)
        {

```

```
        printf("%d ", myNode->data);
        myNode = myNode->link[1];
    }
    if (!myNode)
        break;
}
printf("\n");
return;
}

void search(int data)
{
    struct tbstNode *myNode;
    int path;
    if (!root)
    {
        printf("Tree Not Available!!\n");
        return;
    }
    myNode = root;
    while (1)
    {
        if (myNode->data == data)
        {
            printf("Given data present in TBST!!\n");
            return;
        }
        path = (data > myNode->data) ? 1 : 0;
        if (myNode->marker[path] == THREAD)
            break;
        else
            myNode = myNode->link[path];
    }
    printf("Given data is not present in TBST!!\n");
    return;
}

int main()
{
    int data, ch;
    while (1)
    {
        printf("1. Insertion\t2. Deletion\n");
        printf("3. Searching\t4. Traversal\n");
```

```
printf("5. Exit\nEnter your choice:");
scanf("%d", &ch);
switch (ch)
{
case 1:
    printf("Enter your input data:");
    scanf("%d", &data);
    insertion(data);
    break;
case 2:
    printf("Enter your input data:");
    scanf("%d", &data);
    delete (data);
    break;
case 3:
    printf("Enter your input data:");
    scanf("%d", &data);
    search(data);
    break;
case 4:
    traversal();
    break;
case 5:
    exit(0);
default:
    printf("You have entered wrong option!!\n");
    break;
}
printf("\n");
}
```

Output :

```
"/Users/amzamani/Desktop/sem3/DS/ds lab/assgn08/"'threadedbit
```

```
1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit
```

```
Enter your choice:1
```

```
Enter your input data:10
```

```
1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit
```

```
Enter your choice:1
```

```
Enter your input data:20
```

```
1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit
```

```
Enter your choice:1
```

```
Enter your input data:30
```

```
1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit
```

```
Enter your choice:4
```

```
10 20 30
```

```
1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit
```

```
Enter your choice:3
```

```
Enter your input data:20
```

```
Given data present in TBST!!
```

```
1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit
```

```
Enter your choice:2
```

```
Enter your input data:20
```

```
path: 1 data:10
```

```
1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit
```

```
Enter your choice:4
```

```
10 30
```

```
1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit
```