Name: Deeptej Dhauskar | Roll no: 211105017

Lab Session #9 (In threaded Binary trees)

_

Code:

```
#include <stdio.h>
#include <stdlib.h>
typedef enum
{
    false,
    true
} boolean;

struct node
{
    int info;
    boolean lthread, rthread;
    struct node *left, *right;
};

struct node *in_succ(struct node *ptr)
{
    if (ptr->rthread == true)
        return ptr->right;
    else
    {
        ptr = ptr->right;
        while (ptr->lthread == false)
            ptr = ptr->left;
        return ptr;
    }
}

struct node *in_pred(struct node *ptr)
{
    if (ptr->lthread == true)
        return ptr->left;
    else
    {
        ptr = ptr->left;
        while (ptr->rthread == false)
            ptr = ptr->right;
        return ptr;
    }
}

void inorder(struct node *root)
{
    printf("The inorder traversal of
the tree is: ");
    if (root == NULL)
    {
        printf("<empty>\n");
        return;
    }
    struct node *ptr = root;
    while (ptr->lthread == false)
        ptr = ptr->left; // leftmost
node
    while (ptr != NULL)
    {
        printf("%3d", ptr->info);
        ptr = in_succ(ptr);
    }
    printf("\n");
}

struct node *insert(struct node *root,
int item)
{
    int found = 0;
    struct node *temp, *par = NULL,
*ptr = root;
    while (ptr != NULL)
    {
        if (ptr->info == item)
        {
            found = 1;
            break;
        }
        par = ptr;
```

```
        if (item < ptr->info)
        {
            if (ptr->lthread == false)
                ptr = ptr->left;
            else
                break;
        }
        else
        {
            if (ptr->rthread == false)
                ptr = ptr->right;
            else
                break;
        }
    }
    if (found)
        printf("Duplicate Element\n");
    else
    {
        if ((temp = (struct node
*)malloc(sizeof(struct node))) ==
NULL)
        {
            printf("<memory
overflow>\n");
            return root;
        }
        temp->info = item;
        temp->lthread = temp->rthread
= true;
        if (par == NULL)
        {
            root = temp;
            temp->left = temp->right =
NULL;
        }
        else if (item < par->info)
        {
            temp->left = par->left,
temp->right = par;
            par->lthread = false,
par->left = temp;
        }
        else
        {
            temp->right = par->right,
temp->left = par;
            par->rthread = false,
par->right = temp;
        }
    }
    return root;
}

struct node *case_a(struct node *root,
struct node *par, struct node *ptr)
{
    // node to be deleted has no kids
    struct node *temp = ptr;
    if (par == NULL)
        root = NULL;
    else if (ptr == par->left)
    {
        par->lthread = true;
        par->left = ptr->left;
    }
    else
    {
        par->rthread == true;
        par->right = ptr->right;
    }

    free(temp);
    return root;
}

struct node *case_b(struct node *root,
struct node *par, struct node *ptr)
{
    // ptr has one child;
    struct node *child;
    if (ptr->lthread == false)
        child = ptr->left;
    else
        child = ptr->right;
    struct node *temp = ptr;
```

```
    if (par == NULL)
        root = child;
    else if (par->left == ptr)
        par->left = child;
    else
        par->right = child;
    struct node *p, *s;
    p = in_pred(ptr), s =
in_succ(ptr);
    if (ptr->lthread == false)
        p->right = s;
    else
    {
        if (ptr->rthread == false)
            s->left = p;
    }
    free(temp);
    return root;
}

struct node *case_c(struct node *root,
struct node *par, struct node *ptr)
{
    struct node *parsucc = ptr, *succ
= ptr->right;
    while (succ->lthread == false)
    {
        parsucc = succ;
        succ = succ->left;
    }
    ptr->info = succ->info;
    if (succ->lthread == true &&
succ->rthread == true)
        root = case_a(root, parsucc,
succ);
    else
        root = case_b(root, parsucc,
succ);
    return root;
}

struct node *delete (struct node
*root, int item)
{
    struct node *ptr = root, *temp,
*par = NULL;
    int found = 0;
    while (ptr != NULL)
    {
        if (ptr->info == item)
        {
            found = 1;
            break;
        }
        else
        {
            par = ptr;
            if (item < ptr->info)
            {
                if (ptr->lthread ==
false)
                    ptr = ptr->left;
                else
                    break;
            }
            else
            {
                if (ptr->rthread ==
false)
                    ptr = ptr->right;
                else
                    break;
            }
        }
    }
    if (found)
    {
        if (ptr->lthread == false &&
ptr->rthread == false)
            root = case_c(root, par,
ptr);
        else if (ptr->rthread ==
false)
            root = case_b(root, par,
ptr);
```

```
        else if (ptr->lthread ==
false)
            root = case_b(root, par,
ptr);
        else
            root = case_a(root, par,
ptr);
    }
    else
        printf("%d not present in the
tree\n", item);
    return root;
}

void *search(struct node *root, int
skey)
{
    struct node *ptr = root;
    int found = 0;
    while (ptr != NULL)
    {
        if (ptr->info == skey)
        {
            found = 1;
            break;
        }
        else if (skey < ptr->info)
        {
            if (ptr->lthread == false)
                ptr = ptr->left;
            else
                break;
        }
        else
        {
            if (ptr->rthread == false)
                ptr = ptr->right;
            else
                break;
        }
    }
    if (found)
        printf("Found\n");
    else
        printf("Not Found\n");
}

int main(int argc, char const *argv[])
{
    struct node *root = NULL;
    int arr[11] = {19, 84, 64, 66, 58,
83, 26, 23, 69, 88, 70};
    printf("The tree contains the
following nodes\n");
    for (int i = 0; i < 11; i++)
    {
        root = insert(root, arr[i]);
        printf("%d, ", arr[i]);
    }
    int choice, ikey, dkey, skey;
    do
    {
        printf("Select an option\n");
        printf("1. Insert a new
node\n2. Delete an existing node\n3.
Search for a node\n4. Inorder
Traversal\n-1. Exit\n");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("Enter number: ");
            scanf("%d", &ikey);
            root = insert(root, ikey);
            break;
        case 2:
            printf("Enter the term to
be deleted: ");
            scanf("%d", &dkey);
            root = delete (root,
dkey);
            break;
        case 3:
            printf("Enter an element
to be searched: ");
            scanf("%d", &skey);
```

```
            search(root, skey);
            break;
        case 4:
            inorder(root);
            break;
        default:
            printf("Invalid
option!\nTry again\n");
```

Output:

```
C:\Users\Deeptej\Desktop\Data-Structur
es-Lab\Lab Session #9> &
.\"lab7_inthread.exe"
The tree contains the following nodes
19, 84, 64, 66, 58, 83, 26, 23, 69,
88, 70, Select an option
1. Insert a new node
2. Delete an existing node
3. Search for a node
4. Inorder Traversal
-1. Exit
2
Enter the term to be deleted: 58
Select an option
1. Insert a new node
2. Delete an existing node
3. Search for a node
4. Inorder Traversal
-1. Exit
2
Enter the term to be deleted: 19
Select an option
1. Insert a new node
2. Delete an existing node
3. Search for a node
4. Inorder Traversal
-1. Exit
4
The inorder traversal of the tree is:
23 26 64 66 69 70 83 84 88
Select an option
1. Insert a new node
```

```
                break;
            }
        } while (choice != -1);

        return 0;
}
```

```
2. Delete an existing node
3. Search for a node
4. Inorder Traversal
-1. Exit
```

–