

NAME: SWETHA.J
REGISTER NO.:230701357

Ex-9: Implementation Binary Search Tree

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

// Definition of the binary tree node structure
struct tree {
    int data;
    struct tree *left;
    struct tree *right;
}*root=NULL;

// Function declarations
void insert();
void deleteNode(struct tree *, int);
struct tree *inorder_succ(struct tree *);
void inorder(struct tree *);
void search();

int main() {
    int ans = 1, key;
    struct tree *ptr = NULL;
    int choice;

    do {
        printf("Enter your choice:\n1. Insert\n2. Delete\n3.
Display\n4. Search\n");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                insert();
                break;
            case 2:
                printf("\nEnter the value to be deleted\n");
                scanf("%d", &key);
                ptr = root;
                deleteNode(ptr, key);
                break;
            case 3:
                ptr = root;
                inorder(ptr);
                break;
            case 4:
                search();
                break;
        }
        printf("\nWant to continue?\nPress 1.YES \t 0.NO\n");
        scanf("%d", &ans);
    } while (ans == 1);

    return 0;
}
```

```

}

void insert() {
    int Flag = 0, key;
    struct tree *parent = NULL, *ptr = root;

    printf("Enter the value to be inserted\n");
    scanf("%d", &key);

    while (ptr != NULL && Flag == 0) {
        if (key < ptr->data) {
            parent = ptr;
            ptr = ptr->left;
        } else if (key > ptr->data) {
            parent = ptr;
            ptr = ptr->right;
        } else if (key == ptr->data) {
            Flag = 1;
        }
    }

    // Creating new node using malloc and setting the data and links of
the new node
    struct tree *newnode = malloc(sizeof(struct tree));
    newnode->left = newnode->right = NULL;
    newnode->data = key;

    if (parent == NULL) {
        root = newnode;
    } else {
        if (key < parent->data)
            parent->left = newnode;
        else
            parent->right = newnode;
    }
}

void inorder(struct tree *ptr) {
    if (ptr != NULL) {
        inorder(ptr->left);
        printf("%d -> ", ptr->data);
        inorder(ptr->right);
    }
}

void search() {
    int Flag = 0, key;
    struct tree *parent = NULL, *ptr = root;

    printf("Enter the key to be searched\n");
    scanf("%d", &key);

    while (ptr != NULL && Flag == 0) {
        if (key < ptr->data) {
            parent = ptr;
            ptr = ptr->left;
        } else if (key > ptr->data) {
            parent = ptr;

```

```

        ptr = ptr->right;
    } else if (key == ptr->data) {
        Flag = 1;
        printf("%d found\n", ptr->data);
    }
}

if (Flag == 0)
    printf("Required Key not found\n");
}

void deleteNode(struct tree *ptr, int key) {
    struct tree *parent = NULL;
    int Flag = 0;

    while (ptr != NULL && Flag == 0) {
        if (key < ptr->data) {
            parent = ptr;
            ptr = ptr->left;
        } else if (key > ptr->data) {
            parent = ptr;
            ptr = ptr->right;
        } else if (key == ptr->data) {
            Flag = 1;
        }
    }

    if (Flag == 0) {
        printf("Required Key does not exist\n");
        return;
    }

    // If the node to be deleted is a leaf node
    if (ptr->left == NULL && ptr->right == NULL) {
        if (parent == NULL) {
            root = NULL;
        } else if (key < parent->data) {
            parent->left = NULL;
        } else {
            parent->right = NULL;
        }
        free(ptr);
    }

    // If the node to be deleted has one child
    else if (ptr->left == NULL || ptr->right == NULL) {
        if (parent == NULL) {
            if (ptr->right == NULL)
                root = ptr->left;
            else
                root = ptr->right;
        } else if (key < parent->data) {
            if (ptr->left != NULL)
                parent->left = ptr->left;
            else
                parent->left = ptr->right;
        } else {
            if (ptr->left != NULL)
                parent->right = ptr->left;

```

```

        else
            parent->right = ptr->right;
    }
    free(ptr);
}
// If the node to be deleted has two children
else {
    struct tree *new_ptr;
    new_ptr = inorder_succ(ptr->right);
    int save = new_ptr->data;
    deleteNode(ptr, new_ptr->data);
    ptr->data = save;
}
}

struct tree *inorder_succ(struct tree *pt) {
    while (pt->left != NULL) {
        pt = pt->left;
    }
    return pt;
}

```