**DAY 10 – BINARY SEARCH TREE**

12.. Write a menu driven C program to implement a binary search tree using

linked list and perform the following operations on it

a. Insertion.

b. Deletion.

c. Traversals.

d. Search for a specified node

**PROGRAM**

#include<stdio.h>

#include<stdlib.h>

struct node

{

    int data;

    struct node \*left, \*right;

}\*root=NULL;

void preorder(struct node \**ptr*)

{

    if(*ptr* != NULL)

    {

    printf("%d ", *ptr* -> data);

    preorder(*ptr* -> left);

    preorder(*ptr* -> right);

    }

}

void inorder(struct node \**ptr*)

{

    if(*ptr* != NULL)

    {

        inorder(*ptr* -> left);

        printf("%d ", *ptr* -> data);

        inorder(*ptr* -> right);

    }

}

void postorder(struct node \**ptr*)

{

    if(*ptr* != NULL)

    {

        postorder(*ptr* -> left);

        postorder(*ptr* -> right);

        printf("%d ", *ptr* -> data);

    }

}

void insert(int *val*)

{

    struct node \*newnode, \*nodeptr, \*parentptr;

    newnode = (struct node \*)malloc(sizeof(struct node));

    newnode -> data = *val*;

    newnode -> left = NULL;

    newnode -> right = NULL;

    if(root == NULL)

    root = newnode;

    else

    {

        parentptr = NULL;

        nodeptr = root;

        while(nodeptr != NULL)

        {

            parentptr = nodeptr;

            if(*val* < nodeptr -> data)

            nodeptr = nodeptr -> left;

            else

            nodeptr = nodeptr -> right;

        }

        if(*val* < parentptr -> data)

            parentptr -> left = newnode;

        else

            parentptr -> right = newnode;

    }

}

struct node \* inordersuccessor(struct node \**ptr*)

{

    while(*ptr* != NULL && *ptr* -> left != NULL)

*ptr* = *ptr* -> left;

    return *ptr*;

}

struct node \* delete(struct node \**root*, int *val*)

{

    struct node \*temp;

    if(*root* == NULL)

    printf("Node not found.");

    else if(*val* < *root* -> data)

*root* -> left = delete(*root* -> left, *val*);

    else if(*val* > *root* -> data)

*root* -> right = delete(*root* -> right, *val*);

    else

    {

        if(*root* -> left != NULL && *root* -> right != NULL)

        {

            temp = inordersuccessor(*root* -> right);

*root* -> data = temp -> data;

*root* -> right = delete(*root* -> right, temp -> data);

        }

        else

            {

            temp = *root*;

            if(*root* -> left == NULL && *root* -> right == NULL)

            {

                free(*root*);

                return NULL;

            }

            else if(*root* -> left != NULL)

*root* = *root* -> left;

            else

*root* = *root* -> right;

            free(temp);

        }

    }

    return *root*;

}

struct node \* search(struct node \**root*, int *val*)

{

    if(*root* == NULL || *root* -> data == *val*)

    return *root*;

    else if(*val* > *root* -> data)

    return search(*root* -> right, *val*);

    else

    return search(*root* -> left, *val*);

}

void main()

{

    int val, ch;

    printf("Building tree:\n");

    do

    {

        if(root == NULL)

        printf("\nEnter value of root (-1 to exit): ");

        else

        printf("Enter value of node (-1 to exit): ");

        scanf("%d", &val);

        if(val != -1)

        insert(val);

    }while(val != -1);

    do

    {

        printf("\n\t\t\tMENU");

        printf("\n1. Insert\t\t2. Delete\t\t3. Search");

        printf("\n4. Preorder traversal\t5. Inorder traversal\t6. Postorder traversal");

        printf("\nEnter choice: ");

        scanf("%d", &ch);

        switch(ch)

        {

            case 1: printf("Enter node to insert: ");

                    scanf("%d", &*val*);

                    insert(val);

                    break;

            case 2: if(root != NULL)

                    {

                        printf("Enter node to delete: ");

                        scanf("%d", &val);

                        root = delete(root, val);

                    }

                    else

                        printf("Tree is empty.");

                    break;

            case 3: printf("Enter node to search: ");

                    scanf("%d", &*val*);

                    if(search(root, val) != NULL)

                        printf("Node found.");

                    else

                        printf("Node not found.");

                    break;

            case 4: if(root != NULL)

                    {

                        printf("Preorder traversal: ");

                        preorder(root);

                    }

                    else

                        printf("Tree is empty.");

                    break;

            case 5: if(root != NULL)

                    {

                        printf("Inorder traversal: ");

                        inorder(root);

                    }

                    else

                        printf("Tree is empty.");

                    break;

            case 6: if(root != NULL)

                    {

                        printf("Postorder traversal: ");

                        postorder(root);

                    }

                    else

                        printf("Tree is empty.");

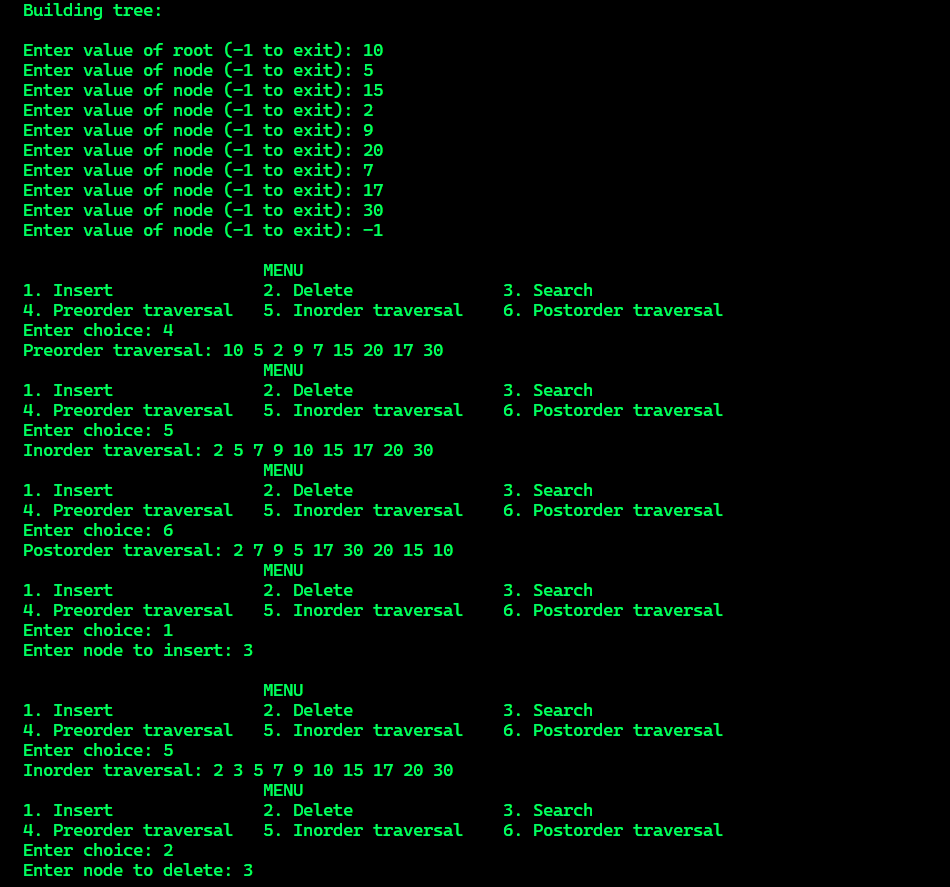
                    break;

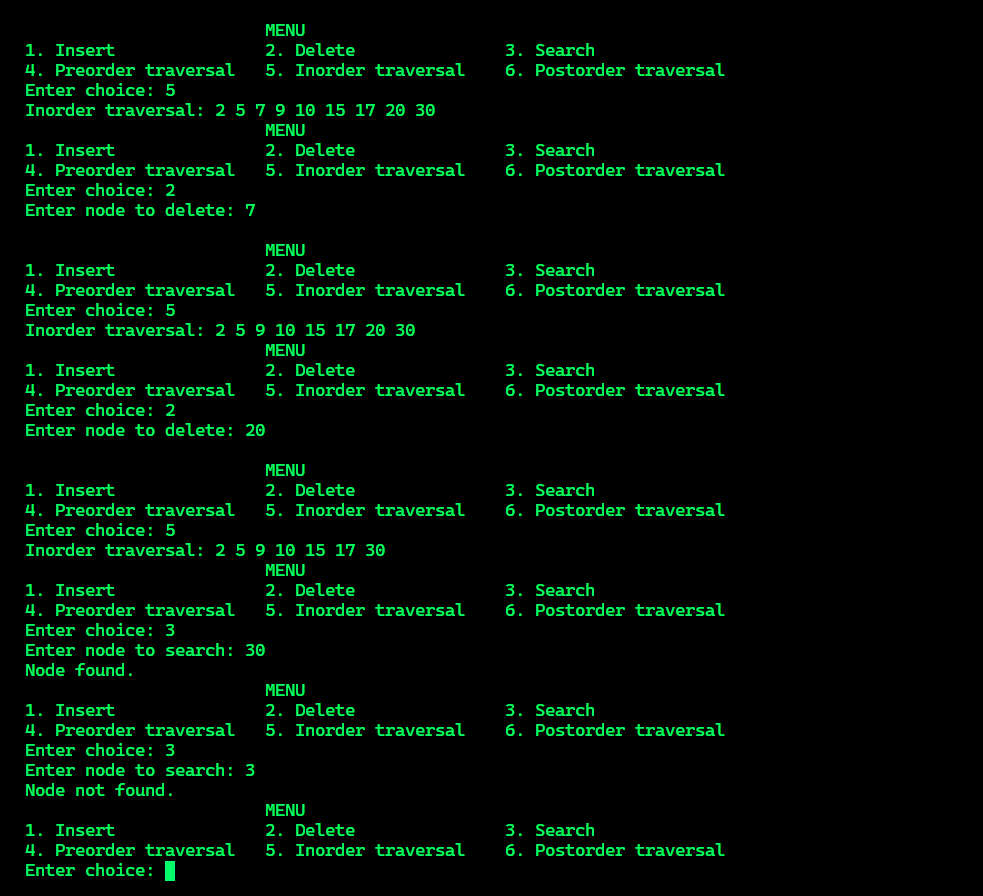
        }

    }while(ch >= 1 && ch <= 6);

}

**OUTPUT**

****

****