**DAY 8 – BINARY TREES USING LINKED LIST**

10.Write a menu driven C program to implement binary trees using linked

lists and perform the following operations

a. Insert a new node.

b. Delete a specified node

c. Search a specified node

**PROGRAM**

#include<stdio.h>

#include<stdlib.h>

struct node

{

    int data;

    struct node \*left, \*right;

};

void displaypreorder(struct node \**ptr*)

{

    if(*ptr* != NULL)

    {

        printf("%d\t", *ptr* -> data);

        displaypreorder(*ptr* -> left);

        displaypreorder(*ptr* -> right);

    }

}

struct node \* buildtree(struct node \**ptr*, int *data*)

{

    int ch, leftdata, rightdata;

    struct node \*leftptr = NULL, \*rightptr = NULL;

    if(*ptr* != NULL)

*ptr* -> data = *data*;

    printf("Does %d have left child? - 1. Yes, 2. No: ", *ptr* -> data);

    scanf("%d", &ch);

    if(ch == 1)

    {

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

*ptr* -> left = leftptr;

        printf("Enter data of left child: ");

        scanf("%d", &leftdata);

        buildtree(leftptr, leftdata);

    }

    else

*ptr* -> left = NULL;

    printf("Does %d have right child? - 1. Yes, 2. No: ", *ptr* -> data);

    scanf("%d", &ch);

    if(ch == 1)

    {

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

*ptr* -> right = rightptr;

        printf("Enter data of right child: ");

        scanf("%d", &rightdata);

        buildtree(rightptr, rightdata);

    }

    else

*ptr* -> right = NULL;

        return *ptr*;

}

struct node \* searchnode(struct node \**ptr*, int *key*)

{

    if(*ptr* == NULL || *ptr* -> data == *key*)

    return *ptr*;

    else

    {

        if(searchnode(*ptr* -> left, *key*) == NULL)

        searchnode(*ptr* -> right, *key*);

    }

}

struct node \* insert(struct node \**root*, int *key*)

{

    int ch, data;

    struct node \*ptr;

    ptr = searchnode(*root*, *key*);

    if(ptr == NULL)

    {

        printf("Parent node not found.");

        return *root*;

    }

    if(ptr -> left == NULL || ptr -> right == NULL)

    {

        printf("Insert as - 1. Left Child, 2. Right Child: ");

        scanf("%d", &ch);

        if(ch == 1)

        {

            if(ptr -> left == NULL)

            {

                printf("Enter data of new node: ");

                scanf("%d", &data);

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

                newnode -> data = data;

                newnode -> left = newnode -> right = NULL;

                ptr -> left = newnode;

            }

            else

                printf("Left child is not empty.");

        }

        else

        {

            if(ptr -> right == NULL)

            {

                printf("Enter data of new node: ");

                scanf("%d", &data);

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

                newnode -> data = data;

                newnode -> left = newnode -> right = NULL;

                ptr -> right = newnode;

            }

        else

            printf("Right child is not empty.");

        }

    }

    else

        printf("Left and right children are not empty.");

    return *root*;

}

struct node \* searchparent(struct node \**ptr*, int *data*)

{

    if(*ptr* == NULL)

        return *ptr*;

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

    {

        if(*ptr* -> left -> data == *data*)

        return *ptr*;

    else

        searchparent(*ptr* -> left, *data*);

    }

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

{

    if(*ptr* -> right -> data == *data*)

        return *ptr*;

    else

searchparent(*ptr* -> right, *data*);

}

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

{

    if(*ptr* -> right -> data == *data* || *ptr* -> left -> data == *data*)

        return *ptr*;

else

{

    if(searchparent(*ptr* -> left, *data*) == NULL)

    searchparent(*ptr* -> right, *data*);

}

}

}

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

{

    struct node \*ptr = NULL, \*parent = NULL;

    if(*root* == NULL)

    {

        printf("\nTree is empty.");

        return *root*;

    }

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

    {

        free(*root*);

*root* = NULL;

    }

    else

    {

        ptr = searchnode(*root*, *data*);

        if(ptr == NULL)

            printf("Node not found.");

        else

        {

            if(ptr -> left == NULL && ptr -> right == NULL)

            {

                parent = searchparent(*root*, *data*);

                if(parent -> left == ptr)

                    parent -> left = NULL;

                else

                    parent -> right = NULL;

                free(ptr);

            }

            else

                printf("\nNode is not a leaf node.");

        }

    }

    return *root*;

}

void main()

{

    int ch, data;

    printf("Enter data of root node: ");

    scanf("%d", &data);

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

    root = buildtree(root, data);

    printf("\nBinary Tree:\t");

    displaypreorder(root);

    do

    {

        printf("\n\tMENU");

        printf("\n1. Insert\t2. Delete\n3. Search\t4. Exit");

        printf("\nEnter choice: ");

        scanf("%d", &ch);

        switch(ch)

        {

            case 1: printf("\nEnter parent node of new node: ");

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

                    root = insert(root, data);

                    printf("\nBinary Tree:\t");

                    displaypreorder(root);

                    break;

            case 2: printf("\nEnter node to delete: ");

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

                    root = delete(root, data);

                    if(root != NULL)

                    {

                        printf("\nBinary Tree:\t");

                        displaypreorder(root);

                    }

                    else

                        printf("\nTree is empty.");

                        break;

            case 3: printf("\nEnter node to search: ");

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

                    if(searchnode(root, data) != NULL)

                        printf("\nNode found.");

                    else

                        printf("\nNode not found.");

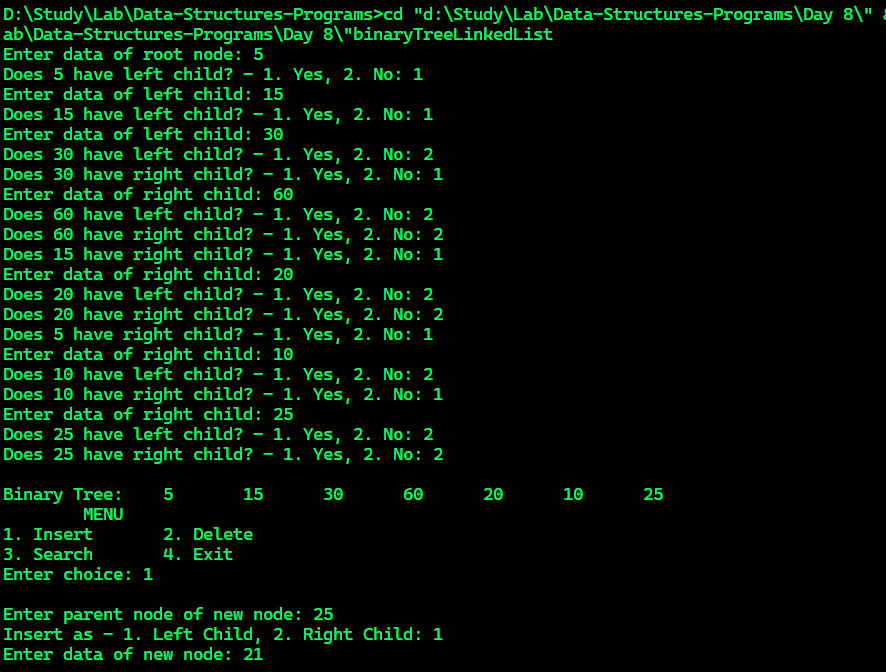
                    break;

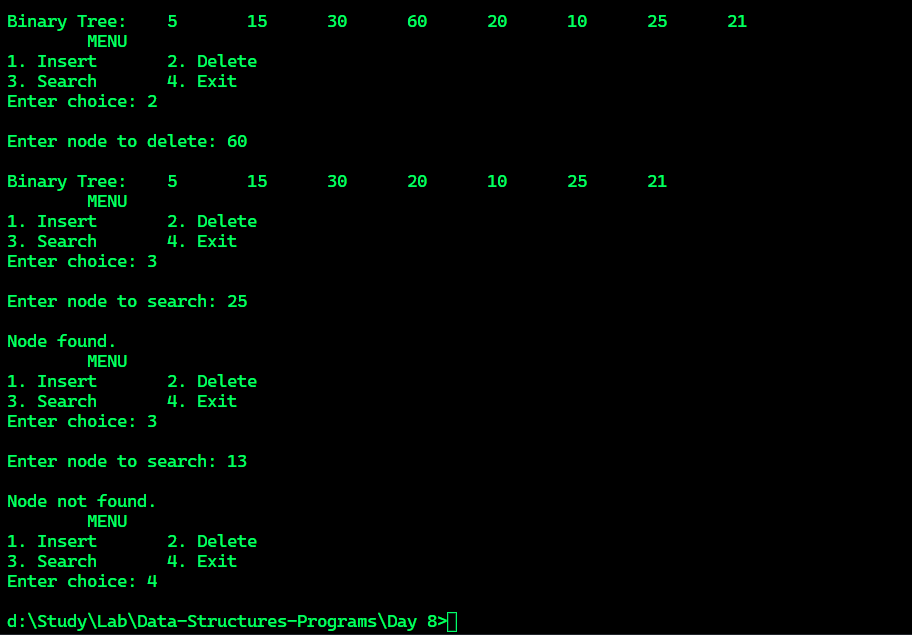
        }

    }while(ch >= 1 && ch <= 3);

}

**OUTPUT**

****

****