**DAY 9 – BINARY TREES USING ARRAYS**

11.Write a menu driven C program to implement binary trees using arrays

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<math.h>

int a[100];

int size, h, loc = 0;

void display()

{

    int i;

    for(i = 1; i <= size; i++)

    {

        if(a[i] != -1)

        printf("%d ", a[i]);

        else

            printf("- ");

    }

}

void buildtree(int *i*, int *item*)

{

    int newL, newR, ch;

    if(*i* != 0)

    {

        a[*i*] = *item*;

        printf("Does %d have left child? - 1. Yes, 2. No: ", a[*i*]);

        scanf("%d", &ch);

        if(ch == 1)

        {

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

            scanf("%d", &newL);

            buildtree(2 \* *i*, newL);

        }

        else

            buildtree(0, 0);

        printf("Does %d have right child? - 1. Yes, 2. No: ", a[*i*]);

        scanf("%d", &ch);

        if(ch == 1)

        {

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

            scanf("%d", &newR);

            buildtree(2 \* *i* + 1, newR);

        }

        else

            buildtree(0, 0);

    }

}

int search(int *i*, int *key*)

{

    if(a[*i*] == -1 || a[*i*] == *key*)

    {

        if(a[*i*] == *key*)

        loc = *i*;

        return *i*;

    }

    else

    {

        if(a[search(2 \* *i*, *key*)] == -1)

        search(2 \* *i* + 1, *key*);

    }

}

void insert(int *key*)

{

    int ch, item;

    search(1, *key*);

    if(a[loc] != *key*)

    {

        printf("Parent node not found.");

        return;

    }

    if(a[2 \* loc] == -1 || a[2 \* loc + 1] == -1)

    {

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

        scanf("%d", &ch);

        if(ch == 1)

        {

            if(a[2 \* loc] == -1)

            {

                printf("Enter data to insert: ");

                scanf("%d", &item);

                if(2 \* loc > size)

                {

                    h++;

                    size = pow(2, h + 1) - 1;

                }

            a[2 \* loc] = item;

            }

        else

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

        }

        else

        {

            if(a[2 \* loc + 1] == -1)

            {

                printf("Enter data to insert: ");

                scanf("%d", &item);

                if(2 \* loc + 1 > size)

                {

                    h++;

                    size = pow(2, h + 1) - 1;

                }

                a[2 \* loc + 1] = item;

            }

            else

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

        }

    }

    else

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

}

void delete(int *key*)

{

    int k, flag = 0;

    search(1, *key*);

    if(a[loc] == *key*)

    {

        if(a[2 \* loc] == -1 && a[2 \* loc + 1] == -1)

        {

            a[loc] = -1;

            for(k = pow(2, h); k <= size; k++)

            {

                if(a[k] != -1)

                flag = 1;

            }

            if(flag == 0)

            {

                h--;

                size = pow(2, h + 1) - 1;

            }

        }

        else

            printf("%d is not a leaf node.", *key*);

    }

    else

        printf("Node does not exist.");

}

int main()

{

    int i, ch, data;

    for(i=0;i<100;i++)

    a[i]=-1;

    printf("Enter height of the tree: ");

    scanf("%d", &h);

    size = pow(2, h + 1) - 1;

    printf("Enter root: ");

    scanf("%d", &data);

    buildtree(1, data);

    printf("Binary Tree: ");

    display();

    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*);

                    insert(data);

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

                    display();

                    break;

            case 2: if(h >= 0)

                    {

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

                        scanf("%d", &data);

                        delete(data);

                        if(h >= 0)

                        {

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

                            display();

                        }

                        else

                            printf("Tree is empty.");

                    }

                    else

                        printf("Tree is empty.");

                        break;

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

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

                    search(1, data);

                    if(a[loc] == data)

                        printf("\nNode found.");

                    else

                        printf("\nNode not found.");

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

                    display();

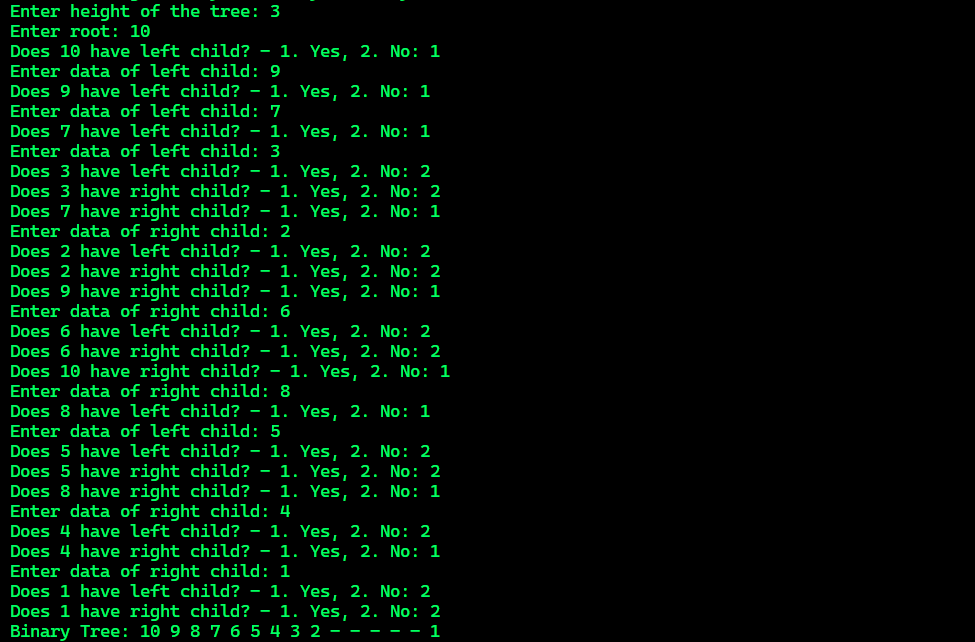
                    break;

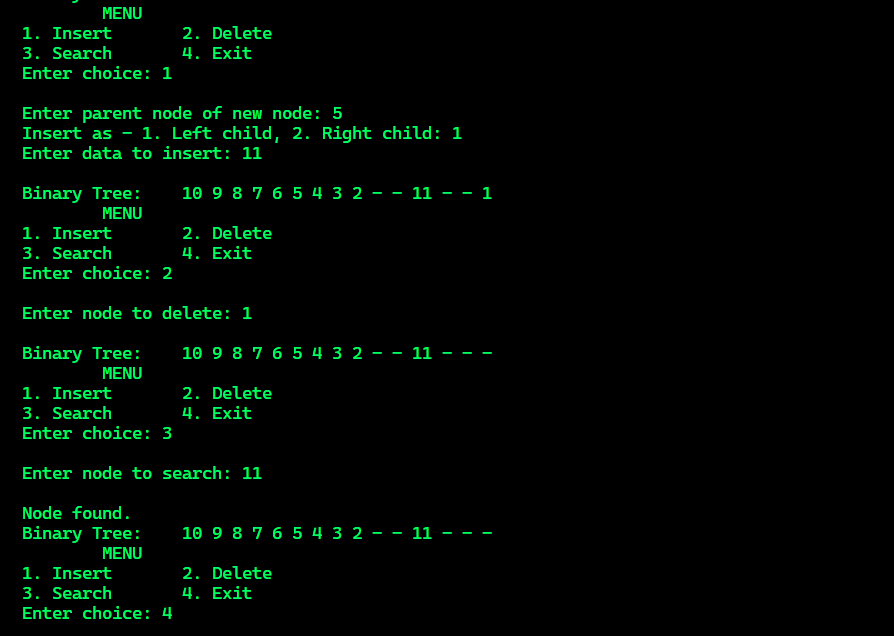
        }

    }while(ch >= 1 && ch <= 3);

}

**OUTPUT**

****

****