BINARY TREE

AIM:

To create a binary tree using linked list anf perform the operations.

ALGORITHM:

· Initialize the root pointer of the binary tree to NULL.

· Display the menu for create, insert, search, delete, and exit operations.

· Read the user's choice from the keyboard.

· If the create option is selected, read the data and create a new node as the root of the tree.

· If the insert option is selected, read the data and insert the value into the binary tree based on binary search tree property.

· If the search option is selected, read the data and search for the value in the binary tree.

· Display whether the given value is found or not found in the tree.

· If the delete option is selected, read the data and delete the specified value from the binary tree.

· Rearrange the tree nodes after deletion to maintain the binary search tree property.

· Repeat the above steps until the user selects the exit option.

PROGRAM:

```c
/*
* Program to demonstrate Binary and its operations
* Author   : MUTHUGANESH S
* Date     : 20/01/2026
* Filename : BinaryTree.c
* retval   : void
*/

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

// Define the structure for a binary tree node
typedef struct Tree{
    int Data;
    struct Tree *Left;
    struct Tree *Right;
} Tree;

// Function to create a new tree node
Tree* createNode(int Data) {
    Tree* NewNode = (Tree*)malloc(sizeof(Tree));

    if(NewNode == NULL) {
        printf("Memory allocation failed\n");
```

```
        exit(1);
    }

    NewNode->Data = Data;
    NewNode->Left = NULL;
    NewNode->Right = NULL;
    return NewNode;
}

// Function to insert a new value into the binary tree
void Insert(Tree* root, int Data) {
    if (root == NULL) {
        root = createNode(Data);
        return;
    }

    if (Data < root->Data) {
        Insert(root->Left, Data);
    }
    else {
        Insert(root->Right, Data);
    }
}

// Function to search for a value in the binary tree
Tree* Search(Tree* root, int Data) {
    if (root == NULL || root->Data == Data) {
        return root;
    }

    if (Data < root->Data) {
        return Search(root->Left, Data);
    }
    else {
        return Search(root->Right, Data);
    }
}

// Function to delete a value from the binary tree
void DeleteValue(Tree* root, int Data) {
    if (root == NULL) {
        return;
    }

    if (Data < root->Data) {
        DeleteValue(root->Left, Data);
    }
    else if (Data > root->Data) {
        DeleteValue(root->Right, Data);
    }
    else {
        if (root->Left == NULL && root->Right == NULL) {
            free(root);
            root = NULL;
```

```
        }
        else if (root->Left == NULL) {
            Tree* temp = root;
            root = root->Right;
            free(temp);
        }
        else if (root->Right == NULL) {
            Tree* temp = root;
            root = root->Left;
            free(temp);
        }
        else {
            Tree* temp = root->Right;
            while (temp->Left != NULL) {
                temp = temp->Left;
            }
            root->Data = temp->Data;
            DeleteValue(root->Right, temp->Data);
        }
    }
}

// Main function to demonstrate the binary tree operations
int main(void){
    printf("Menu\n1.Create Tree\n2. Insert\n3. Search\n4. Delete\n5. Exit\n");
    int Choice, Data;
    Tree *Root = NULL;
    printf("Enter your choice: ");
    scanf("%d",&Choice);
    while(Choice!=5){
        switch (Choice)
        {
        case 1:
            printf("Enter root data: ");
            scanf("%d",&Data);
            Root = createNode(Data);
            break;

        case 2:
            printf("Enter data to insert: ");
            scanf("%d",&Data);
            Insert(Root,Data);
            break;

        case 3:{
            printf("Enter data to search: ");
            scanf("%d",&Data);
            Tree* Node = Search(Root, Data);
            if(Node == NULL){
                printf("Data not found\n");
            }
            else{
                printf("Data found\n");
            }
```

```
            break;
        }
        case 4:{
            printf("Enter data to delete: ");
            scanf("%d",&Data);
            DeleteValue(Root,Data);
            break;
        }
        default:{
            printf("Invalid Choice\n");
            break;
        }
        }
        printf("Enter your choice: ");
        scanf("%d",&Choice);
    }
    return 0;
}
```

OUTPUT: