Program

```c
#include<stdio.h>
#include<stdlib.h>
struct node {
    int key;
    struct node* left;
    struct node* right;
};
struct node *root=NULL,*temp=NULL;

struct node* create(int key){
    struct node* newnode = (struct node*)malloc(sizeof(struct node));
    newnode->key = key;
    newnode->left = newnode->right = NULL;
    if(root==NULL)
            root=newnode;
    return newnode;
}

void Preorder(struct node *root){
        if(root!=NULL){
                printf("%d ", root->key);
                Preorder(root->left);
                Preorder(root->right);
        }
}


void Inorder(struct node *root){
        if(root!=NULL){
                Inorder(root->left);
                printf("%d ", root->key);
                Inorder(root->right);
        }
}

void Postorder(struct node *root){
        if(root!=NULL){
                Postorder(root->left);
                Postorder(root->right);
                printf("%d ", root->key);
        }
}

struct node* findMin(struct node* root)
{
    struct node* temp=root;
    while (temp->left != NULL)
   {
        temp = temp->left;
    }
    return temp;
}


struct node* search(struct node *root,int item){
    if(root==NULL || root->key==item){
        printf("The node exists");
        return root;
    }
    else{
```

Experiment No: 27
Date:

BINARY TREE USING LINKED LIST

Aim:

To create a binary tree and perform operations on it using linked lists.

Algorithm:

1. Start.
2. Create a structure node (int key, struct node* left, struct node* right).
3. Declare pointers root and temp.
4. Define function struct node* create(int key).
5. Create newnode.

```
newnode->key = key
newnode->left = newnode->right = NULL
if(root==NULL)
  root=newnode
end if
```

6. Define function Preorder (struct node *root).

```
if(root!=NULL)
  print root->key
  Preorder(root->left)
  Preorder(root->right)
end if
```

7. Define function Inorder (struct node *root).

```
if(root!=NULL)
  Inorder(root->left)
  print root->key
  Inorder(root->right)
end if
```

8. Define function Postorder (struct node* root).

```
        if(item<root->key){
            return search(root->left,item);
        }
        else{
            return search(root->right,item);
        }

    }
}
struct node* insertion(struct node *root,int item){
    if(root==NULL){
        return create(item);
    }
    else if(item<root->key){
        root->left=insertion(root->left,item);
    }
    else{
        root->right=insertion(root->right,item);
    }
    return root;
}

struct node* Delete(struct node* root,int value){
    if(root==NULL)
        return root;
    else if(value<root->key)
    {
        root->left=Delete(root->left,value);
    }
    else if(value>root->key)
    {
        root->right= Delete(root->right,value);
    }
    else
    {
        if(root->left==NULL && root->right==NULL)
        {
          free(root);
          root=NULL;
          return root;
        }
        else if(root->left==NULL)
        {
            struct node* temp=root;
            root=root->right;
            free(temp);
            return root;
        }
        else if(root->right==NULL)
        {
            struct node* temp=root;
            root=root->left;
            free(temp);
            return root;
        }
        else
        {
            struct node* temp=findMin(root->right);
            root->key=temp->key;
            root->right=Delete(root->right,temp->key);
        }
    }
    return root;
```

```
if(root!=NULL)
  Postorder(root->left)
  Postorder(root->right)
  print root->key
end if
```

9. Define function struct node* findMin (struct node* root).

```
struct node* temp=root
Begin while loop: temp->left != NULL
  temp = temp->left
End while
```

10. Define function struct node* search (struct node* root,int item).

```
if(root==NULL or root->key==item){
  print The node exists
else
  if(item<root->key)
      return search(root->left,item)
  else
      return search(root->right,item)
  End if
End if
```

11. Define function struct node* search (struct node* root,int item).

```
if(root==NULL)
  return create(item)
if(item<root->key)
  root->left=insertion(root->left,item)
else
  root->right=insertion(root->right,item);
End if
```

12. Define function struct node* Delete (struct node* root,int value).

```
if(root==NULL)
  return root
if(value<root->key)
  root->left=Delete(root->left,value)
if(value>root->key)
  root->right= Delete(root->right,value)
else
  if(root->left==NULL and root->right==NULL)
    free(root)
    root=NULL
  if(root->left==NULL)
      struct node* temp=root
      root=root->right
      free(temp)
  if(root->right==NULL)
      struct node* temp=root;
      root=root->left;
      free(temp);
  else
```

```
}
int main()
{
        int ch,flag,item;
        do
        {
                printf("1 Create, 2 Preorder, 3 Inorder, 4 Postorder");
                printf("\n 5 Search, 6 Insertion, 7 Deletion");
                printf("\n Enter your selection: ");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:
                printf("Enter data: ");
                                scanf("%d",&item);
                                create(item);
                                break;
                        case 2:
                                Preorder(root);
                                break;
                        case 3:
                                Inorder(root);
                                break;
                        case 4:
                                Postorder(root);
                                break;
                        case 5:
                                printf("Enter data: ");
                                scanf("%d",&item);
                                        search(root,item);
                                break;
                        case 6:
                                    printf("Enter data: ");
                                scanf("%d",&item);
                                insertion(root,item);
                                break;
                        case 7:
                                printf("Enter data: ");
                                scanf("%d",&item);
                                Delete(root,item);
                                break;
                        default:
                                printf("Invalid entry");
                }
                printf("\n 1 to continue, 0 to exit: ");
                scanf("%d",&flag);
        }
        while(flag==1);
}
```

Output

```
1 Create, 2 Preorder, 3 Inorder, 4 Postorder
 5 Search, 6 Insertion, 7 Deletion
 Enter your selection: 1
Enter data: 40

 1 to continue, 0 to exit: 1
1 Create, 2 Preorder, 3 Inorder, 4 Postorder
 5 Search, 6 Insertion, 7 Deletion
 Enter your selection: 6
```

```
                struct node* temp=findMin(root->right)
                root->key=temp->key
                root->right=Delete(root->right,temp->key)
            End if
        End if
```

13. In main()

```
        Declare integer variables ch,flag and item.
        Begin do while loop.
        Display the options.
        Accept choice from user as ch.
        switch(ch)
            case 1:
                Take input from user as item
                create(item)
            case 2:
                Preorder(root)
            case 3:
                Inorder(root)
            case 4:
                Postorder(root)
            case 5:
                Take input from user as item
                search(root,item)
            case 6:
                Take input from user as item
                insertion(root,item)
            case 7:
                Take input from user as item
                Delete(root,item)
            default:
                print Invalid entry
        Take input from user to continue or not as flag.
        Close while loop if flag!=1.
```

14. Stop

```
Enter data: 30

 1 to continue, 0 to exit: 1
1 Create, 2 Preorder, 3 Inorder, 4 Postorder
 5 Search, 6 Insertion, 7 Deletion
 Enter your selection: 6
Enter data: 50

 1 to continue, 0 to exit: 1
1 Create, 2 Preorder, 3 Inorder, 4 Postorder
 5 Search, 6 Insertion, 7 Deletion
 Enter your selection: 6
Enter data: 48

 1 to continue, 0 to exit: 1
1 Create, 2 Preorder, 3 Inorder, 4 Postorder
 5 Search, 6 Insertion, 7 Deletion
 Enter your selection: 6
Enter data: 21

 1 to continue, 0 to exit: 1
1 Create, 2 Preorder, 3 Inorder, 4 Postorder
 5 Search, 6 Insertion, 7 Deletion
 Enter your selection: 6
Enter data: 80

 1 to continue, 0 to exit: 1
1 Create, 2 Preorder, 3 Inorder, 4 Postorder
 5 Search, 6 Insertion, 7 Deletion
 Enter your selection: 2
40 30 21 50 48 80
 1 to continue, 0 to exit: 1
1 Create, 2 Preorder, 3 Inorder, 4 Postorder
 5 Search, 6 Insertion, 7 Deletion
 Enter your selection: 3
21 30 40 48 50 80
 1 to continue, 0 to exit: 1
1 Create, 2 Preorder, 3 Inorder, 4 Postorder
 5 Search, 6 Insertion, 7 Deletion
 Enter your selection: 4
21 30 48 80 50 40
 1 to continue, 0 to exit: 1
1 Create, 2 Preorder, 3 Inorder, 4 Postorder
 5 Search, 6 Insertion, 7 Deletion
 Enter your selection: 5
Enter data: 40
The node exists
 1 to continue, 0 to exit: 1
1 Create, 2 Preorder, 3 Inorder, 4 Postorder
 5 Search, 6 Insertion, 7 Deletion
 Enter your selection: 7
Enter data: 40

 1 to continue, 0 to exit: 1
1 Create, 2 Preorder, 3 Inorder, 4 Postorder
 5 Search, 6 Insertion, 7 Deletion
 Enter your selection: 2
48 30 21 50 80
 1 to continue, 0 to exit: 0
```

Result:

Program has been executed successfully and obtained the output.