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



struct Node
   {
      int data;
      struct Node* prev;
      struct Node* next;
   };

void inorder(struct Node* root)
{
   if(root)
      {
         inorder(root->prev);
         printf("%d\t",root->data);
         inorder(root->next);
      }
}



void Insert_node(struct Node** root,int key)
   {
      struct Node* current;
      struct Node* p;
      struct Node* newnode;
      newnode=(struct Node *)malloc(sizeof(struct Node));
      newnode->data=key;
      newnode->prev=NULL;
      newnode->next=NULL;
      p=*root;
      if(*root==NULL)
      {
         *root=newnode;

         printf("it is in the root insertion:\n");
      }

      else
```

```
    {   current=*root;
        while(current)
        {
            p=current;
            if(newnode->data > current->data )
            {   printf("%d\n",current->data);
                current=current->next;
            }
            else
            {   printf("%d\n",current->data);
                current=current->prev;
            }
        }
        if(newnode->data > p->data)
        {   printf("it is in the without root: if\n");
            p->next=newnode;
            printf("%d\n",newnode->data);

        }
        else
        {   printf("it is in the without root:else\n");
            p->prev=newnode;
            printf("%d\n",newnode->data);


        }
    }
  }


void deletion(struct Node** root,int key)
  {
     struct Node* current=*root;
     struct Node* p;
     int flag=0;
      while(current!=NULL)
        {

            if(key==current->data)
            {   flag=1;
                printf("flag1:\n");
                break;
            }
```

```
        if(key > current->data)
        {   p=current;
            current=current->next;

        }
        else
        {
            p=current;
            current=current->prev;

        }

    }

if(flag==1)
{   printf("satisfy:\n");
    if(current->prev==NULL & current->next==NULL)
    {   printf("satisfy:\n");
        if(current==p->next)
        {
            p->next=NULL;
        }
        else
        {
            p->prev=NULL;
        }
    }

    // this is code for the deletion of node when node does not have the right child

    else if(current->prev!=NULL & current->next==NULL)
    {
        p->prev=current->prev;
        current=NULL;
    }

    // this is the code for the deletion of node when node does not have the left children
    else if(current->prev==NULL & current->next!=NULL)
    {
        p->prev=current->next;
        current=NULL;
    }
```

```
        // this is code for the deletion of a node when left and right child are not null
        else if(current->prev!=NULL & current->next!=NULL)
        {
           if(current->next->prev==NULL & current->next->next==NULL & p->next==current)
           {
              p->next=current->next;
              current->next->prev=current->prev;
              free(current);
           }
           else if(current->next->prev==NULL & current->next->next==NULL &
p->prev==current)
           {

              p->prev=current->next;
              current->next->prev=current->prev;
              free(current);
           }



        }

      }

   }
int main()
   {
      struct Node* root;
      root=NULL;
      Insert_node(&root,50);
      Insert_node(&root,80);

      Insert_node(&root,60);
      Insert_node(&root,90);
      Insert_node(&root,40);
      Insert_node(&root,30);

      Insert_node(&root,45);
      deletion(&root,40);
      deletion(&root,80);
```

```
   /* Insert_node(&root,45);
    Insert_node(&root,65);
    Insert_node(&root,85);
    Insert_node(&root,75);
    Insert_node(&root,70);
    deletion(&root,55);
    Insert_node(&root,63);
    Insert_node(&root,95);
    */
    inorder(root);
    return 0;

}
```