BTree - Linked List

I. create node of struct datatype having lchild, rchild ~~also~~, parent as well (Let us call the parent ← dad) also a data

II. struct node* searchkey() {

```
1. START
2. int top=0, i=-1
3. tmp = root
4. while (i != -1 || tmp != NULL)
    1. If (tmp → data == key) // key is global
        1. return temp
    2. If (i >= 0 && tmp → lchild == NULL &&
                    tmp → rchild == NULL)
        1. tmp = arr[i] // struct ~~node~~ *arr[
        2. tmp = tmp → rchild
        3. --i
    3. ~~Else~~ If (tmp → lchild == NULL &&
                    tmp → rchild != NULL)
        1. tmp = tmp → rchild;
        4. Else If (tmp -> rchild == NULL &&
                    tmp -> lchild != NULL)
            1. tmp = tmp -> lchild
        5. Else If (tmp -> rchild != NULL &&
                    tmp -> lchild != NULL)
            1. i++
            2. arr[i] = tmp
            3. tmp = tmp -> lchild
        6. Else
            1. tmp = NULL
        7. End If
    5. End While
    ~~6. If (tmp != NULL)~~
        ~~print~~
    6. return tmp
    7. STOP
```

III

```
void bt_deleaf (int pdata)

1. START
2. key = pdata        // key is a global reference
3. leaf = searchkey()
4. if (leaf != NULL)
    1. ~~leaf~~ leafp = leaf -> dad.
```

IV void build_tree (struct node *par)

```
0. START
1. print (" does " + par->data + "have a left node")
2. scanf (" %d ", & L)
3. If (L != 0)
        1. struct node *child;
        2. allocate some space to child
        3. accept the left child value from user
        4. set par -> lchild = child.
        5. set child -> dad = par
        6. build_tree (child);
4. End If
5. Ask user if it must have a right nod
6. If yes accept value as R <- integer (0/1)
7. If (R != 0)
        1. struct node *child;
        2. allocate some space to child
        3. accept the right child value from
                                            u
        4. Set par -> rchild = child
        5. set child -> dad = par.
        6 build_tree (child)
8 End If
9. STOP.
```

V void bt_insert (int pdata)

```
1. START
2. create struct ptr & allocate some space
3. set key = pdata          // key is global
4  parent = searchkey ()
5. if (size ==0)
    1. ~~build tree (ptr)~~ set root = ptr
    2. build_tree (ptr)
6. Else If (parent ==NULL)
    1. print ("pdata +" is not an element of BTree")
    2. free (ptr)
7. Else If (parent->children not NULL)
    1. print (pdata +" has 2 children")
    2. free (ptr)
8. Else
    1. Accept from user the value to insert
    2. If (parent->lchild ==NULL)
        1. add the child to left
        2. set l=1
    3. If (parent->rchild ==NULL && l!=0)
        1. add the child to right
    4.
9. End If
10. increase size
11. STOP
```

VI void dispinorder (struct node *t)

```
1. START
2. if (j < size && t != NULL) //set j = 0 before entering
    1. j++
    2. dispinorder (t -> lchild)
    3. printf ("~~%d~~ \t" + t->data);
    4. dispinorder (t -> rchild)
3. End If
4. STOP
```

VII void dispreorder(struct node *t)

```
1. START
2. if (j < size && t != NULL) //set j = 0 before entering
    1. j++
    2. ~~dispre~~ print ("\t" + t->data);
    3. dispreorder (t -> lchild)
    4. dispreorder (t -> rchild)
3. End If
4. STOP
```

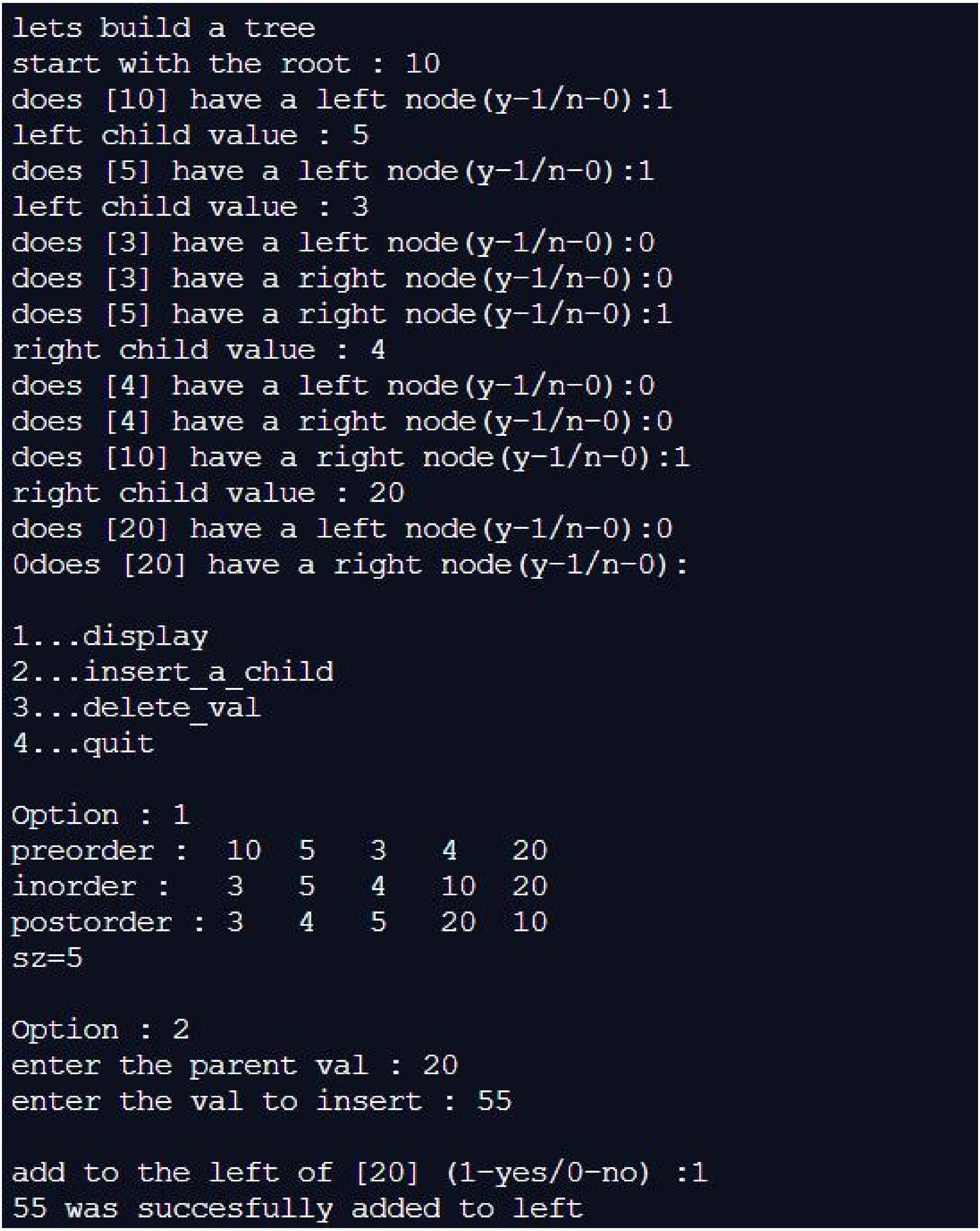
VIII void dispostorder (struct node *t)

```
1. START
2. if (j < size && t != NULL)
    1. j++
    2. dispostorder (t -> lchild)
    3. dispostorder (t -> rchild)
    4. print ("\t" + t -> data
3 End If
4 STOP
```

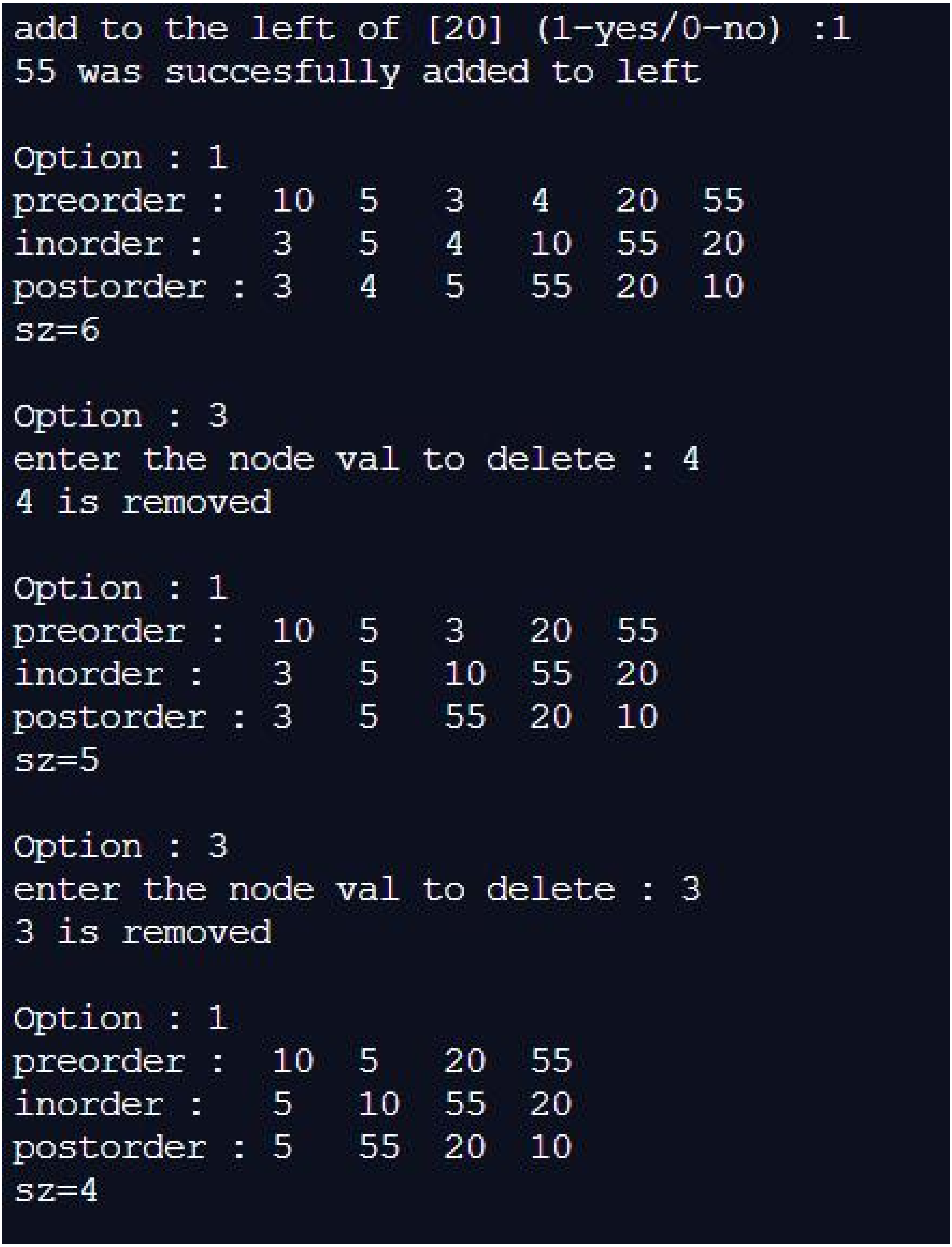
```
#include <stdio.h>
#include <stdlib.h>

#define n 65

int i,x=0,sz=0,key,R,L;


struct node{
    int data;
    struct node *rchild;
    struct node *lchild;
    struct node *dad;
} *root=NULL ,*tmp ,*t2 ,*parent ,*leaf ,*leafp ,*arr[50];

struct node* searchkey2(){
    int top=0,i=-1;
    tmp=root;
    while(i!=-1 || tmp!=NULL){
        if(tmp->data==key)
            return tmp;
        if(i>=0 && tmp->lchild==NULL && tmp->rchild==NULL){
            tmp=arr[i];
            tmp=tmp->rchild;
            --i;
        }
        else if(tmp->lchild==NULL && tmp->rchild!=NULL)
            tmp=tmp->rchild;
        else if(tmp->rchild==NULL && tmp->lchild!=NULL)
            tmp=tmp->lchild;
        else if(tmp->lchild!=NULL && tmp->rchild!=NULL){
            i++;
            arr[i]=tmp;
            tmp=tmp->lchild;
        }
        else
            tmp=NULL;
    }
    if(tmp!=NULL)
         printf("searchkey2 ---------- %d\n",tmp->data);
    return tmp;
}

void bt_deleaf(int pdata){
    key=pdata;
    leaf=searchkey2();
    if(leaf!=NULL){
        leafp=leaf->dad;
        //printf("searchleafpar ---------- %d\n",leafp->data);
    }
    if(leaf==NULL)
        printf("Node does not exist\n");
    else if(leaf->lchild==NULL && leaf->rchild==NULL){
        if(root->data!=leaf->data){
            if(leafp->lchild ==leaf)
                leafp->lchild =NULL;
            else if(leafp->rchild ==leaf)
                leafp->rchild =NULL;
        }
        else
            root=NULL;
        printf("%d is removed\n",leaf->data);
        free(leaf);
        sz--;
    }
    else
        printf("%d is not a leaf node\n",pdata);
}

void build_tree(struct node *par){
    sz++;

    printf("does [%d] have a left node(y-1/n-0):",par->data);
    scanf("%d",&L);
    if(L!=0){
        struct node *child;
        child=(struct node*)malloc(sizeof(struct node));

        printf("left child value : ");
        scanf("%d",&(child->data));
        // child->lchild=NULL;
        // child->rchild=NULL;

        par->lchild=child;
        child->dad=par;
        build_tree(child);
    }

    printf("does [%d] have a right node(y-1/n-0):",par->data);
    scanf("%d",&R);
    if(R!=0){
        struct node *child;
        child=(struct node*)malloc(sizeof(struct node));

        printf("right child value : ");
        scanf("%d",&(child->data));
        child->lchild=NULL;
        child->rchild=NULL;
        par->rchild=child;
        child->dad=par;
        build_tree(child);
    }
}

void bt_insert(int pdata){
    int l=0,r=0;
    struct node *ptr;
    ptr=(struct node*)malloc(sizeof(struct node));
    ptr->lchild=NULL;
    ptr->rchild=NULL;

    key=pdata;
    parent=searchkey2();
    if(sz==0){
        printf("root : %d\n",pdata);
        ptr->data=pdata;
        ptr->dad=NULL;
        root=ptr;
        ptr->lchild=NULL;
        ptr->rchild=NULL;
        build_tree(root);
    }
    else if(parent==NULL){
        printf("%d is not an element of Btree\n",pdata);
        free(ptr);
    }
    else if(parent->lchild!=NULL && parent->rchild!=NULL){
        printf("%d has two children so insertion not possible\n", pdata);
        free(ptr);
    }
    else{

        printf("enter the val to insert : ");
        scanf("%d",&ptr->data);
        if(parent->lchild==NULL){

            printf("\nadd to the left of [%d] (1-yes/0-no) :",parent->data);
            scanf("%d",&l);
            if(l!=0) {//&& node->lchild<=sz){
                parent->lchild=ptr;
                ptr->dad=parent;
                printf("%d was succesfully added to left\n",ptr->data);
            }
        }
        if(parent->rchild==NULL && l==0){

            printf("\nadd to the right of [%d] (1-yes/0-no) :",parent->data);
            scanf("%d",&r);
            if(r!=0) {//&& node->rchild<=sz)
                parent->rchild=ptr;
                ptr->dad=parent;
                printf("%d was succesfully added to right\n",ptr->data);
            }
        }
    sz++;
    }
}

int j=0;
void dispinorder(struct node *t){
    if(j<sz && t!=NULL){
        j++;
        dispinorder(t->lchild);
        printf("%d\t",t->data);
        dispinorder(t->rchild);
    }
}
void dispreorder(struct node *t){
    if(j<sz &&t!=NULL){
        j++;
        printf("%d\t",t->data);
        dispreorder(t->lchild);
        dispreorder(t->rchild);
    }
}
void dispostorder(struct node *t){
    if(j<sz){
        j++;
        if(t->lchild!=NULL && t->lchild->data!=0)
            dispostorder(t->lchild);
        if(t->rchild!=NULL && t->rchild->data!=0)
            dispostorder(t->rchild);
        printf("%d\t",t->data);
    }
}
int main() {

    struct node *ptr;
    ptr=(struct node*)malloc(sizeof(struct node));
    printf("lets build a tree\nstart with the root : ");
    scanf("%d",&(ptr->data));
    ptr->dad=NULL;
    root=ptr;
    ptr->lchild=NULL;
    ptr->rchild=NULL;
    build_tree(root);
```

```
int choice, pos, pdata, item;
printf("\n1...display\n");
printf("2...insert_a_child\n");
printf("3...delete_val\n");
printf("4...quit\n");
int quit=1;
while(quit!=0){
    printf("\nOption : ");
    scanf("%d",&choice);
    switch(choice){
            case 1: if(sz!=0){
                j=0;
                printf("preorder : \t");
                //tmp=root
                dispreorder(root);
                j=0;
                printf("\ninorder : \t");
                //tmp=root
                dispinorder(root);
                j=0;
                printf("\npostorder : ");
                //tmp=root
                dispostorder(root);
                printf("\nsz=%d\n",sz);
            }
            else
                printf("Empty tree\n");
            break;
        case 2: printf("enter the parent val : ");
            scanf("%d",&pdata);

            bt_insert(pdata);
            break;
        case 3: printf("enter the node val to delete : ");
            scanf("%d",&pdata);
            if(root!=NULL)
                bt_deleaf(pdata);
            else
                printf("Btree empty\n");
            break;
        case 4: quit=0;
            break;
        default:
            printf("\n1...display\n");
            printf("2...insert_a_child\n");
            printf("3...delete_val\n");
            printf("4...quit\n");
    }
}
```

```
    return 0;
}
```



```
lets build a tree
start with the root : 10
does [10] have a left node(y-1/n-0):1
left child value : 5
does [5] have a left node(y-1/n-0):1
left child value : 3
does [3] have a left node(y-1/n-0):0
does [3] have a right node(y-1/n-0):0
does [5] have a right node(y-1/n-0):1
right child value : 4
does [4] have a left node(y-1/n-0):0
does [4] have a right node(y-1/n-0):0
does [10] have a right node(y-1/n-0):1
right child value : 20
does [20] have a left node(y-1/n-0):0
0does [20] have a right node(y-1/n-0):

1...display
2...insert_a_child
3...delete_val
4...quit

Option : 1
preorder :  10  5   3   4   20
inorder :   3   5   4   10  20
postorder : 3   4   5   20  10
sz=5

Option : 2
enter the parent val : 20
enter the val to insert : 55

add to the left of [20] (1-yes/0-no) :1
55 was succesfully added to left
```



```
add to the left of [20] (1-yes/0-no) :1
55 was succesfully added to left

Option : 1
preorder :  10  5   3   4   20  55
inorder :   3   5   4   10  55  20
postorder : 3   4   5   55  20  10
sz=6

Option : 3
enter the node val to delete : 4
4 is removed

Option : 1
preorder :  10  5   3   20  55
inorder :   3   5   10  55  20
postorder : 3   5   55  20  10
sz=5

Option : 3
enter the node val to delete : 3
3 is removed

Option : 1
preorder :  10  5   20  55
inorder :   5   10  55  20
postorder : 5   55  20  10
sz=4
```