Binary Search Tree

I create a struct node having int data & rchild, lchild, dad pointers as well.

II struct node *succparent (struct node *ptr)

1. START
2. create struct *x, *par.
3. par = ptr
4. If (par -> rchild != NULL)
   1. x = par -> rchild
   2. while (x -> lchild != NULL)
      1. par = x
      2. x = x -> lchild
   3. End while
5. Else
   1. x = par -> lchild
   2. while (x -> rchild != NULL)
      1. par = x
      2. x = x -> rchild
   3. End while
6. End If
7. return x.
8. STOP

III struct node *succ (struct node *ptr)

1. START
2. create struct *x
3. if (ptr → rchild != NULL)

~~1. succ (p → rchild)~~

~~Else If~~ 1. x = ptr → rchild
2. while (x → lchild != NULL)
1. x = x → lchild
3. End While
4. Else If (ptr → lchild != NULL)
1. x = ptr → lchild
2. while (x → rchild != NULL)
1. x = x → rchild
3. End While
5. End If
6. return x;
7. STOP

IV void dispinorder(struct node *root)

1. START
2. If (root != NULL)
1. dispinorder (root → lchild);
2. print ("\t" + root → data)
3. dispinorder (root → rchild)

3 End If

4 STOP

V void dispreorder (struct node *root)

1. START
2. If (root != NULL)
   1. print ("\t" + root → data)
   2. dispreorder (root → lchild)
   3. dispreorder (root → rchild)
3. Else If
4. STOP

VI void dispostorder (struct node *root)

1. START
2. If (root != NULL)
   1. dispostorder (root → lchild)
   2. dispostorder (root → rchild)
   3. print ("\t" + root → data)
3. End If
4. STOP

VII void insertbst()

1. START
2. Create struct node *x & allocate some space
3. Accept a value from user & add it to x→data

4. If (root == NULL)
    root = x

5. Else
    1. tmp = root
    2. while (tmp != NULL)
        1. parent = tmp
        2. If (x->data < tmp->data)
            1. tmp = tmp->lchild
        3. Else If (x->data > tmp->data)
            1. tmp = tmp->rchild.
        4 Else
            1. tmp = NULL
        5. End If
    3. End while
    4. If (x->data == parent->data)
        1. free (x) // to prevent duplicates
        2. return ;
    5. End If
    6. If (~~tmp == NULL~~) (x->data < parent->data)
        1. parent->lchild = x,
    7 Else
        1. parent->rchild = x;
    8. End If
    9. x->dad = parent
6 End If

7 Size ++

8 STOP

VIII void popitem ( ) {

1. START
2. ptr = root , flag = False
3. Accept & store the item to remove
4. While (ptr != NULL && flag == False)
   1. if(item < ptr -> data)
      1. parent = ptr
      2. ptr = ptr -> lchild
   2. else if(item == ptr -> data)
      1. flag = True
   3. else (item > ptr -> data)
      1. parent = ptr
      2. ptr = ptr -> rchild
5. End While.
6. If (flag == False)
   1. print ("Item not found")
   2. Exit
7. /* Decision case for deletion */
   1. If (ptr -> lchild == NULL && ptr -> rchild == NULL)
      1. CASE = 1

2. Else If (ptr → lchild != NULL &&
ptr → rchild != NULL)

1. CASE = 3

3 Else

1. CASE = 2

4. End If

8. If (CASE == 1)

1. if (parent → lchild == ptr)

1. parent → lchild = NULL

2. Else

1 parent → rchild = NULL

3. End If

4. return ptr

9. If (CASE == 2)

1. if (parent → lchild == ptr)

1. if (ptr → lchild == NULL)

1: parent → lchild = ptr → rchild

2. Else

1. parent → lchild = ptr → lchild

2. Else if (parent → rchild == ptr)

1. if (ptr → rchild == NULL)

1. parent → rchild = ptr → rchild

2. Else

1. parent → rchild = ptr → lchild

3. End If

3. End If

4 return ptr

10 If (CASE == 3)

1 succ = succ x (ptr)

2 succpa = succp(ptr)

3. set the child of succ to the child of succpa

4. free (succ)

11. End If

12. STOP

IX int main()

1. START

2 call insert, display & pop as per users choice

3. STOP.

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

int sz=0,array[20],i=0;

struct node{
    int data;
    struct node *rchild;
    struct node *lchild;
    struct node *dad;
} *root=NULL ,*tmp ,*parent ,*succ ,*succp;

struct node *succparent(struct node *ptr){
    struct node *x,*par;
    par=ptr;

    if(par->rchild !=NULL){     //right subtree
        x=par->rchild;
        while(x->lchild != NULL){
            par=x;
            x=x->lchild;
        }
    }
    else{                       //left subtree
        x=par->lchild;
        while(x->rchild != NULL){
            par=x;
            x=x->rchild;
        }
    }

    return par;
}

struct node* succx(struct node *ptr){
    struct node* x;

    if(ptr->rchild !=NULL){     //right subtree
        x=ptr->rchild;
        while(x->lchild != NULL)
            x=x->lchild;
    }
    else{                       //left subtree
        x=ptr->lchild;
        while(x->rchild != NULL)
            x=x->rchild;
    }

    return x;
}
```

```
void dispinorder(struct node *root){
    if(root != NULL){
        dispinorder(root->lchild);
        printf("%d\t",root->data);
        dispinorder(root->rchild);
    }
}
void dispreorder(struct node *root){
    if(root != NULL){
        printf("%d\t",root->data);
        dispreorder(root->lchild);
        dispreorder(root->rchild);
    }
}
void dispostorder(struct node *root){
    if(root != NULL){
        dispostorder(root->lchild);
        dispostorder(root->rchild);
        printf("%d\t",root->data);
    }
}

void insertbst(){
    int val;
    struct node *x;
    x=(struct node*)malloc(sizeof(struct node));

    printf("enter the val--");
    scanf("%d",&(x->data));
    x->rchild = NULL;
    x->lchild = NULL;

    array[i]=x->data;
    i++;

    if(root==NULL){
        x->dad=NULL;
        root=x;
    }
    else{
        tmp = root;
        while(tmp != NULL){
            parent=tmp;
            if(x->data < tmp->data)
                tmp=tmp->lchild;
            else if(x->data > tmp->data)
                tmp=tmp->rchild;
            else
                tmp=NULL;
```

```
        }
        if(x->data == parent->data){
            free(x);
            //sz--;
            return ;
        }
        if(tmp!=NULL)
            parent=tmp->dad;
        if(x->data < parent->data)
            parent->lchild = x;
        else if(x->data > parent->data)
            parent->rchild = x;
        x->dad=parent;
    }
    sz++;
}
void popitem(){
    int val, flag=0;
    printf("remove me--");
    scanf("%d",&val);
    //printf("1\n");

    for(int index=0;index<=i;index++)
        if(array[index]==val)
            flag=1;
    if(root!=NULL)
        tmp = root;
    //#################finding tmp and parent
    while(flag==1 && tmp != NULL){
        if(val==tmp->data){
            flag=1;
            break;
        }

        parent=tmp;
        if(val < tmp->data){
            tmp=tmp->lchild;
        }
        else if(val > tmp->data){
            tmp=tmp->rchild;
        }
    }
    //########################################
    //printf("1\n");

    if(tmp->dad !=NULL)
        parent=tmp->dad;
    else
        parent=tmp;
```

```
if(flag!=1)
    printf("Item not found\n");
else{
    if(tmp->lchild==NULL && tmp->rchild==NULL){  // no child
        if(root != tmp){
            if(parent->lchild == tmp)
                parent->lchild=NULL;
            else
                parent->rchild=NULL;
        }
        else
            root=NULL;
    }
    else if(tmp->lchild==NULL || tmp->rchild==NULL){ // one child

        if(root != tmp){
            if(parent->lchild == tmp){
                if(tmp->rchild == NULL){// && tmp->lchild != NULL)
                    parent->lchild = tmp->lchild;
                    tmp->lchild->dad=parent;
                }
                else if(tmp->lchild==NULL){// && tmp->rchild !=NULL)
                    parent->lchild = tmp->rchild;
                    tmp->rchild->dad=parent;
                }
            }
            else if(parent->rchild == tmp){
                if(tmp->rchild==NULL){// && tmp->lchild != NULL)
                    parent->rchild = tmp->lchild;
                    tmp->lchild->dad=parent;
                }
                else if(tmp->lchild==NULL){// && tmp->rchild !=NULL)
                    parent->rchild = tmp->rchild;
                    tmp->rchild->dad=parent;
                }
            }
        }
        else{
            if(tmp->lchild != NULL)
                root=tmp->lchild;
            else
                root=tmp->rchild;
        }
    }
    else{                                 // two children

        succ =succx(tmp); //will not have 2 child
        if(succ!=NULL){
            succp = succparent(tmp);
            //succp=succ->dad;
```

```
            }
            else
                succp=succ;


//printf("succp=%d,succ=%d,tmp=%d,par=%d\n",succp->data,succ->data,tmp->data,parent
->data);
            if(succ->dad==succp){
                if(succp->lchild == succ){
                    if(succ->rchild!=NULL && succ->lchild==NULL){
                        succp->lchild =succ->rchild;
                        succp->lchild->dad=succp;
                    }
                    else if(succ->lchild!=NULL && succ->rchild==NULL){
                        succp->lchild=succ->lchild;
                        succp->lchild->dad=succp;
                    }
                    else if(succ->lchild==NULL && succ->rchild==NULL){
                        succp->lchild=NULL;
                    }
                }
                else if(succp->rchild == succ){
                    if(succ->rchild!=NULL && succ->lchild==NULL){
                        succp->rchild =succ->rchild;
                        succp->rchild->dad=succp;
                    }
                    else if(succ->lchild!=NULL && succ->rchild==NULL){
                        succp->rchild=succ->lchild;
                        succp->rchild->dad=succp;
                    }
                    else if(succ->lchild==NULL && succ->rchild==NULL){
                        succp->rchild=NULL;
                    }
                }
                if(tmp->lchild != succ)
                    succ->lchild = tmp->lchild;
                else
                    succ->lchild = NULL;

                if(tmp->rchild != succ)
                    succ->rchild = tmp->rchild;
                else
                    succ->rchild = NULL;

                if (tmp->dad!=NULL)
                    succ->dad = tmp->dad;
                else{
                    succ->dad=NULL;
                    root=succ;
                }
```

```
                if(succ->lchild!=NULL)
                    succ->lchild->dad=succ;
                if(succ->rchild!=NULL)
                    succ->rchild->dad=succ;

                //if(tmp->dad ==parent){
                    if(parent->lchild == tmp)
                        parent->lchild = succ;
                    else if(parent->rchild == tmp)
                        parent->rchild = succ;
                //}
                //tmp=succp;
            }
            else{
                if(succ==NULL)
                    printf("empty tree");
                if(succ==succp){
                    root=NULL;
                    tmp=succ;
                }
            }

//printf("succp=%d,succ=%d,tmp=%d,par=%d\n",succp->data,succ->data,tmp->data,parent
->data);
        }
        free(tmp);
        sz--;
    }

}

int main(){
    int choice;
    int pos;
    printf("1...traversal\n");
    printf("2...insert\n");
    printf("3...popitem\n");
    printf("4...quit\n");

    int quit=1;
    while(quit!=0){
        printf("\nOption : ");
        scanf("%d",&choice);
        switch(choice){
                case 1: if(root!=NULL){
                    printf("preorder : \t");
                    dispreorder(root);
                    printf("\ninorder : \t");
                    dispinorder(root);
                    printf("\npostorder : ");
```

```
                    dispostorder(root);
                    printf("\nsz=%d\n",sz);
                }
                else
                    printf("Empty tree\n");
                break;
            case 2: insertbst();
                break;
            case 3:
                if(sz>0)
                    popitem();
                else
                    printf("no more elements to pop");
                break;
            case 4: quit=0;
                break;
            default:
                printf("\n1...traversal\n");
                printf("2...insert\n");
                printf("3...popitem\n");
                printf("4...quit\n");
        }
    }

    return 0;
}
```

```
1...traversal
2...insert
3...popitem
4...quit

Option : 1
Empty tree

Option : 2
enter the val--5

Option : 2
enter the val--9

Option : 2
enter the val--1

Option : 2
enter the val--7

Option : 2
enter the val--2

Option : 1
preorder :  5   1   2   9   7
inorder :   1   2   5   7   9
postorder : 2   1   7   9   5
sz=5

Option : 3
remove me--9
```

```
Option : 3
remove me--9

Option : 3
remove me--7

Option : 1
preorder :  5   1   2
inorder :   1   2   5
postorder : 2   1   5
sz=3

Option : 3
remove me--5

Option : 1
preorder :  1   2
inorder :   1   2
postorder : 2   1
sz=2

Option : 3
remove me--1

Option : 3
remove me--2

Option : 3
no more elements to pop
Option : 1
Empty tree
```