Binary Search Tree and perform deletion and inorder traversal

```
#include <stdio.h>

#include <stdlib.h>


struct btnode

{

   int value;

   struct btnode *l;

   struct btnode *r;

}*root = NULL, *temp = NULL, *t2, *t1;


void delete1();

void insert();

void delete();

void inorder(struct btnode *t);

void create();

void search(struct btnode *t);

void preorder(struct btnode *t);

void postorder(struct btnode *t);

void search1(struct btnode *t,int data);

int smallest(struct btnode *t);

int largest(struct btnode *t);
```

```
int flag = 1;


void main()
{
   int ch;


   printf("\nOPERATIONS ---");
   printf("\n1 - Insert an element into tree\n");
   printf("2 - Delete an element from the tree\n");
   printf("3 - Inorder Traversal\n");
   printf("4 - Preorder Traversal\n");
   printf("5 - Postorder Traversal\n");
   printf("6 - Exit\n");
   while(1)
   {
      printf("\nEnter your choice : ");
      scanf("%d", &ch);
      switch (ch)
      {
      case 1:
         insert();
         break;
      case 2:
         delete();
```

```
            break;
        case 3:
            inorder(root);
            break;
        case 4:
            preorder(root);
            break;
        case 5:
            postorder(root);
            break;
        case 6:
            exit(0);
        default :
            printf("Wrong choice, Please enter correct choice  ");
            break;
        }
    }
}

/* To insert a node in the tree */
void insert()
{
    create();
    if (root == NULL)
```

```
        root = temp;
    else
        search(root);
}

/* To create a node */
void create()
{
    int data;

    printf("Enter data of node to be inserted : ");
    scanf("%d", &data);
    temp = (struct btnode *)malloc(1*sizeof(struct btnode));
    temp->value = data;
    temp->l = temp->r = NULL;
}

/* Function to search the appropriate position to insert the new node */
void search(struct btnode *t)
{
    if ((temp->value > t->value) && (t->r != NULL))      /* value more than root node value insert at right */
        search(t->r);
    else if ((temp->value > t->value) && (t->r == NULL))
        t->r = temp;
```

```
   else if ((temp->value < t->value) && (t->l != NULL))    /* value less than root node value insert at left */
      search(t->l);
   else if ((temp->value < t->value) && (t->l == NULL))
      t->l = temp;
}

/* recursive function to perform inorder traversal of tree */
void inorder(struct btnode *t)
{
   if (root == NULL)
   {
      printf("No elements in a tree to display");
      return;
   }
   if (t->l != NULL)
      inorder(t->l);
   printf("%d -> ", t->value);
   if (t->r != NULL)
      inorder(t->r);
}

/* To check for the deleted node */
void delete()
{
```

```
    int data;

    if (root == NULL)
    {
        printf("No elements in a tree to delete");
        return;
    }
    printf("Enter the data to be deleted : ");
    scanf("%d", &data);
    t1 = root;
    t2 = root;
    search1(root, data);
}

/* To find the preorder traversal */
void preorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display");
        return;
    }
    printf("%d -> ", t->value);
    if (t->l != NULL)
```

```
        preorder(t->l);
    if (t->r != NULL)
        preorder(t->r);
}

/* To find the postorder traversal */
void postorder(struct btnode *t)
{
    if (root == NULL)
    {
        printf("No elements in a tree to display ");
        return;
    }
    if (t->l != NULL)
        postorder(t->l);
    if (t->r != NULL)
        postorder(t->r);
    printf("%d -> ", t->value);
}

/* Search for the appropriate position to insert the new node */
void search1(struct btnode *t, int data)
{
    if ((data>t->value))
```

```
    {
        t1 = t;
        search1(t->r, data);
    }
    else if ((data < t->value))
    {
        t1 = t;
        search1(t->l, data);
    }
    else if ((data==t->value))
    {
        delete1(t);
    }
}

/* To delete a node */
void delete1(struct btnode *t)
{
    int k;

    /* To delete leaf node */
    if ((t->l == NULL) && (t->r == NULL))
    {
        if (t1->l == t)
```

```
    {
        t1->l = NULL;
    }
    else
    {
        t1->r = NULL;
    }
    t = NULL;
    free(t);
    return;
}

/* To delete node having one left hand child */
else if ((t->r == NULL))
{
    if (t1 == t)
    {
        root = t->l;
        t1 = root;
    }
    else if (t1->l == t)
    {
        t1->l = t->l;
```

```
    }
    else
    {
        t1->r = t->l;
    }
    t = NULL;
    free(t);
    return;
}

/* To delete node having right hand child */
else if (t->l == NULL)
{
    if (t1 == t)
    {
        root = t->r;
        t1 = root;
    }
    else if (t1->r == t)
        t1->r = t->r;
    else
        t1->l = t->r;
    t == NULL;
    free(t);
```

```
        return;
    }

    /* To delete node having two child */
    else if ((t->l != NULL) && (t->r != NULL))
    {
        t2 = root;
        if (t->r != NULL)
        {
            k = smallest(t->r);
            flag = 1;
        }
        else
        {
            k =largest(t->l);
            flag = 2;
        }
        search1(root, k);
        t->value = k;
    }

}

/* To find the smallest element in the right sub tree */
```

```
int smallest(struct btnode *t)
{
   t2 = t;
   if (t->l != NULL)
   {
      t2 = t;
      return(smallest(t->l));
   }
   else
      return (t->value);
}

/* To find the largest element in the left sub tree */
int largest(struct btnode *t)
{
   if (t->r != NULL)
   {
      t2 = t;
      return(largest(t->r));
   }
   else
      return(t->value);
}
```