Write a program to implement the following operations of Binary Search Tree.

i. Insert
ii. Traverse (Inorder, Preorder, Postorder)
iii. Delete

Program-

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

typedef struct Node
{
      struct Node *left;
      int data;
      struct Node *right;
} node;

// Creation
node *createNode(int element)
{
      node *newnode = (node *)malloc(sizeof(node));
      newnode->data = element;
      newnode->left = newnode->right = NULL;
      return newnode;
}

// Insertion
void insertion(node **root, int element)
{
      node *prev = NULL, *newnode, *loc = *root;
      while (loc != NULL)
      {
            prev = loc;
            if (loc->data == element)
            {
                  printf("Element exists.\n");
                  return;
            }
            else if (element < loc->data)
                  loc = loc->left;
            else
                  loc = loc->right;
      }
      newnode = createNode(element);
```

```
	if (prev == NULL)
		*root = newnode;
	else if (element < prev->data)
		prev->left = newnode;
	else
		prev->right = newnode;
	return;
}

// Traversal
void inorder(node *root)
{
	if (root != NULL)
	{
		inorder(root->left);
		printf("%d ", root->data);
		inorder(root->right);
	}
	else
	{
		printf("BST doesn't exists.\n");
	}
}

void preorder(node *root)
{
	if (root != NULL)
	{
		printf("%d ", root->data);
		preorder(root->left);
		preorder(root->right);
	}
	else
	{
		printf("BST doesn't exists.\n");
	}
}

void postorder(node *root)
{
	if (root != NULL)
	{
		postorder(root->left);
		postorder(root->right);
```

```
            printf("%d ", root->data);
      }
      else
      {
            printf("BST doesn't exists.\n");
      }
}

// Deletion
void deletion(node **root, int item)
{
      if (*root == NULL)
      {
            printf("BST doesn't exists.\n");
            return;
      }
      node *par = NULL, *loc = *root, *child, *parsuc, *suc;
      while (loc != NULL && loc->data != item)
      {
            par = loc;
            if (item < loc->data)
                  loc = loc->left;
            else
                  loc = loc->right;
      }
      if (loc == NULL)
      {
            printf("\nNode containing %d does not exist.\n",
item);
            return;
      }
      if (loc->left == NULL && loc->right == NULL)
            child = NULL;
      else if (loc->left == NULL)
            child = loc->right;
      else if (loc->right == NULL)
            child = loc->left;
      else
      {
            suc = loc->right;
            parsuc = loc;
            while (suc->left != NULL)
            {
                  parsuc = suc;
```

```
                suc = suc->left;
            }
            if (parsuc != loc)
            {
                parsuc->left == suc->right;
                suc->right = loc->right;
            }
            suc->left = loc->left;
            child = suc;
        }
        if (par != NULL)
        {
            if (loc == par->left)
                par->left = child;
            else
                par->right = child;
        }
        else
            *root = child;
        printf("\n%d deleted.\n", item);
        free(loc);
        return;
}

int main()
{
        int num, element;
        node *root = NULL;
        while (1)
        {
            printf("\nOperations to be performed:\n");
            printf("1. Insertion.\n");
            printf("2. Deletion.\n");
            printf("3. Traversal.\n");

            printf("\nEnter the operation (1-3 or 0 to exit):
");
            scanf("%d", &num);

            switch (num)
            {
                case 1:
                    printf("\nEnter the element to be
inserted: ");
```

```
                        scanf("%d", &element);
                        insertion(&root, element);
                        break;

                case 2:
                        printf("\nEnter the element to be
      deleted: ");
                        scanf("%d", &element);
                        deletion(&root, element);
                        break;

                case 3:
                        printf("\nPreorder Traversal: ");
                        preorder(root);
                        printf("\nInorder Traversal: ");
                        inorder(root);
                        printf("\nPostorder Traversal: ");
                        postorder(root);
                        printf("\n");
                        break;

                case 0:
                        exit(0);

                default:
                        printf("Invalid option! Try again.\n");
                }
        }
        return (0);
}
```