Tree Traversal :
1.In Order Tree Traversal Algorithm.
Until all nodes are traversed –
Step 1 – Recursively traverse left subtree.
Step 2 – Visit root node.
Step 3 – Recursively traverse right subtree.

2. Pre order Tree Traversal
Until all nodes are traversed –
Step 1 – Visit root node.
Step 2 – Recursively traverse left subtree.
Step 3 – Recursively traverse right subtree.

3. Post order Tree Traversal
Until all nodes are traversed –
Step 1 – Recursively traverse left subtree.
Step 2 – Recursively traverse right subtree.
Step 3 – Visit root node.

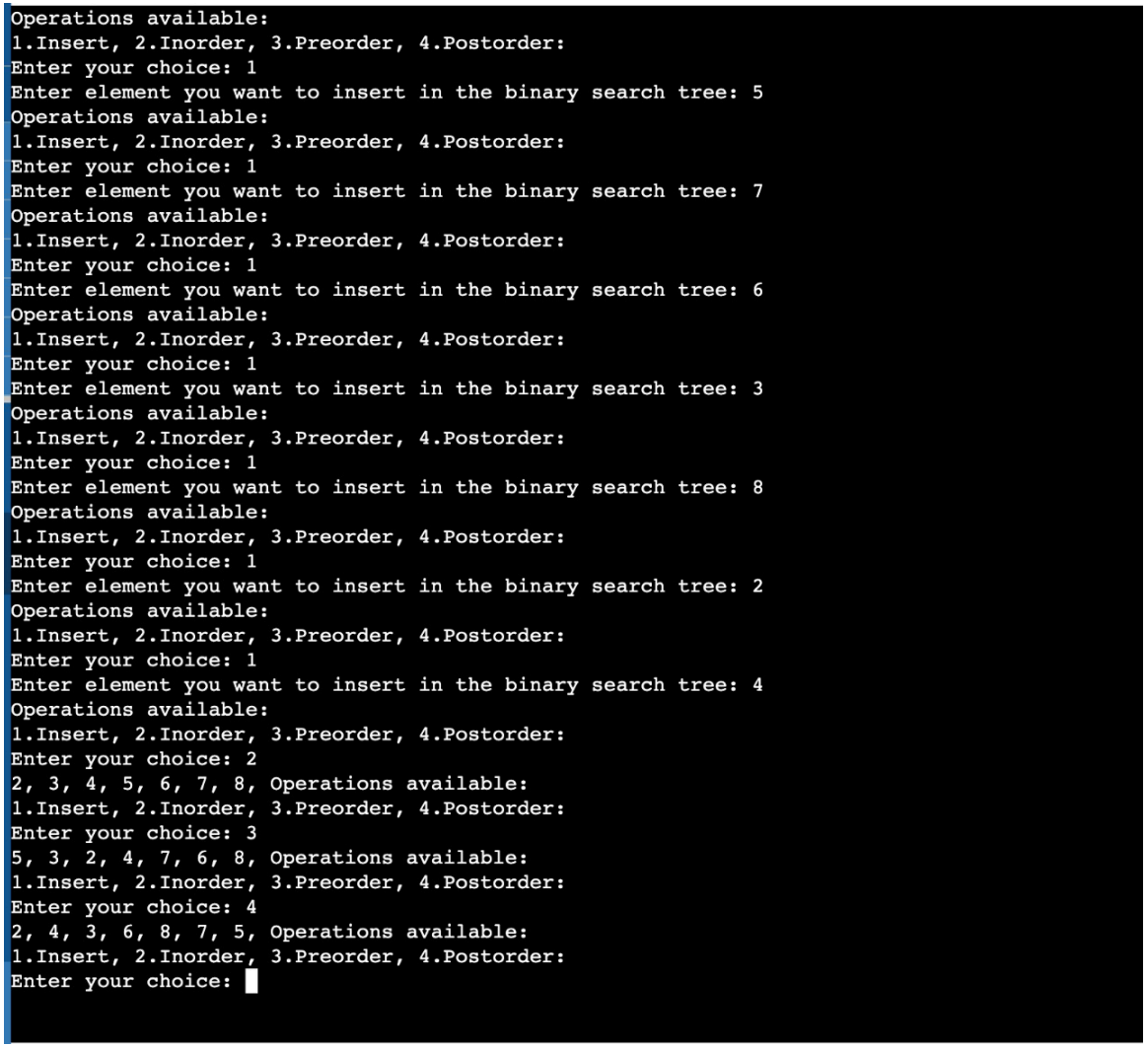
Source Code:

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

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

struct node* insert(struct node* T, int x){
```

```
    if(T == NULL){
        T = (struct node*)malloc(sizeof(struct node));
        T->data = x;
        T->left = NULL;
        T->right = NULL;
    }
    else if(x < T->data){
        T->left = insert(T->left, x);
    }
    else if(x >= T->data){
        T->right = insert(T->right, x);
    }
    return T;
}

void inorder(struct node* T){
    if(T != NULL){
        inorder(T->left);
        printf("%d, ", T->data);
        inorder(T->right);
    }
}

void preorder(struct node* T){
    if(T != NULL){
        printf("%d, ", T->data);
```

```
        preorder(T->left);
        preorder(T->right);
    }
}

void postorder(struct node* T){
    if(T != NULL){
        postorder(T->left);
        postorder(T->right);
        printf("%d, ", T->data);
    }
}

int main()
{
    struct node* root = NULL;
    struct node* min_node, *max_node;

    do{
        int ch, x;
        printf("Operations available:\n");
        printf("1.Insert, 2.Inorder, 3.Preorder, 4.Postorder:\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);

        switch(ch){
```

```
            case 1:
                printf("Enter element you want to insert in the binary search tree: ");
                scanf("%d", &x);
                root = insert(root, x);
                break;
            case 2:
                inorder(root);
                break;
            case 3:
                preorder(root);
                break;
            case 4:
                postorder(root);
                break;
            default:
                exit(0);
        }
    }while(1);

    return 0;
}
```



Input and output.

Binary search tree:

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

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

struct node* insert(struct node* T, int x){
    if(T == NULL){
        T = (struct node*)malloc(sizeof(struct node));
        T->data = x;
        T->left = NULL;
        T->right = NULL;
    }
    else if(x < T->data){
        T->left = insert(T->left, x);
    }
    else if(x >= T->data){
        T->right = insert(T->right, x);
    }
    return T;
}
```

```
struct node* minValueNode(struct node* node)
{
   struct node* current = node;

   while (current && current->left != NULL){
      current = current->left;
   }
   return current;
}

struct node* deleteNode(struct node* root, int key)
{
   if (root == NULL){
      return root;
   }
   if (key < root->data){
      root->left = deleteNode(root->left, key);
   }
   else if (key > root->data){
      root->right = deleteNode(root->right, key);
   }
   else {

      if (root->left == NULL) {
         struct node* temp = root->right;
         free(root);
```

```
            return temp;
        }
        else if (root->right == NULL) {
            struct node* temp = root->left;
            free(root);
            return temp;
        }
        struct node* temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}


void inorder(struct node* T){
    if(T != NULL){
        inorder(T->left);
        printf("%d, ", T->data);
        inorder(T->right);
    }
}

void preorder(struct node* T){
    if(T != NULL){
        printf("%d, ", T->data);
```

```
        preorder(T->left);
        preorder(T->right);
    }
}

void postorder(struct node* T){
    if(T != NULL){
        postorder(T->left);
        postorder(T->right);
        printf("%d, ", T->data);
    }
}

struct node* find_min(struct node* T){
    if(T == NULL){
        return T;
    }
    else{
        return find_min(T->left);
    }
}

struct node* find_max(struct node* T){
    if(T == NULL){
        return T;
    }
```

```
    else{
        return find_max(T->right);
    }
}

void find_ele(struct node *T, int x){
    struct node* temp = T;

    while(temp != NULL){
        if(x == temp->data){
            break;
        }
        else if(x < temp->data){
            temp = temp->left;
        }
        else{
            temp = temp->right;
        }
    }

    if(T != NULL){
        printf("Element is present.\n");
    }
    else{
        printf("Element is not present.\n");
    }
```

```
}

int main()
{
    struct node* root = NULL;
    struct node* min_node, *max_node;

    do{
        int ch, x, s_ele, del;
        printf("Operations available:\n");
        printf("1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Find Minimum Element, 6. Find Maximum Element, 7.Find
element, 8.Delete:\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);

        switch(ch){
            case 1:
                printf("Enter element you want to insert in the binary search tree: ");
                scanf("%d", &x);
                root = insert(root, x);
                break;
            case 2:
                inorder(root);
                break;
            case 3:
                preorder(root);
```

```
            break;
        case 4:
            postorder(root);
            break;
        case 5:
            min_node = find_min(root);
            printf("The smallest element is: %d", min_node->data);
            break;
        case 6:
            max_node = find_max(root);
            printf("The biggest element is: %d", max_node->data);
            break;
        case 7:
            printf("Enter the element you want to search: ");
            scanf("%d", &s_ele);
            find_ele(root, s_ele);
            break;
        case 8:
            printf("Enter the element you want to delete: ");
            scanf("%d", &del);
            root = deleteNode(root, del);
            printf("After deletion:\n");
            inorder(root);
            break;
        default:
            exit(0);
```

```
    }
  }while(1);

  return 0;
```

```
Operations available:
1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Find Minimum Element, 6. Find Maximum Element, 7.Find element, 8.Delete:
Enter your choice: 1
Enter element you want to insert in the binary search tree: 8
Operations available:
1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Find Minimum Element, 6. Find Maximum Element, 7.Find element, 8.Delete:
Enter your choice: 1
Enter element you want to insert in the binary search tree: 2
Operations available:
1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Find Minimum Element, 6. Find Maximum Element, 7.Find element, 8.Delete:
Enter your choice: 1
Enter element you want to insert in the binary search tree: 4
Operations available:
1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Find Minimum Element, 6. Find Maximum Element, 7.Find element, 8.Delete:
Enter your choice: 2
2, 3, 4, 5, 6, 7, 8, Operations available:
1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Find Minimum Element, 6. Find Maximum Element, 7.Find element, 8.Delete:
Enter your choice: 8
Enter the element you want to delete: 5
After deletion:
2, 3, 4, 6, 7, 8, Operations available:
1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Find Minimum Element, 6. Find Maximum Element, 7.Find element, 8.Delete:
Enter your choice: 8
Enter the element you want to delete: 8
After deletion:
2, 3, 4, 6, 7, Operations available:
1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Find Minimum Element, 6. Find Maximum Element, 7.Find element, 8.Delete:
Enter your choice: 8
Enter the element you want to delete: 4
After deletion:
2, 3, 6, 7, Operations available:
1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Find Minimum Element, 6. Find Maximum Element, 7.Find element, 8.Delete:
Enter your choice: 2
2, 3, 6, 7, Operations available:
1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Find Minimum Element, 6. Find Maximum Element, 7.Find element, 8.Delete:
Enter your choice: 3
6, 3, 2, 7, Operations available:
1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Find Minimum Element, 6. Find Maximum Element, 7.Find element, 8.Delete:
Enter your choice: 4
2, 3, 7, 6, Operations available:
1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Find Minimum Element, 6. Find Maximum Element, 7.Find element, 8.Delete:
Enter your choice:
```

```
}
```

AVL TREE:

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

struct node{
    int key;
    struct node *left;
    struct node *right;
    int height;
};

int getheight(struct node *n){
    if(n == NULL){
        return 0;
    }
    else{
        return n->height;
    }
}

struct node *create(int x){
    struct node newnode = (struct node)malloc(sizeof(struct node));
```

```
    newnode->key = x;
    newnode->left = NULL;
    newnode->right = NULL;
    newnode->height = 1;
    return newnode;
}

int max(int a, int b){
    // if(a>b){
    //     return a;
    // }
    // else{
    //     return b;
    // }
    return a>b?a:b;
}

int getBalanceFactor(struct node* n){
    if(n == NULL){
        return 0;
    }
    else{
        return getheight(n->left) - getheight(n->right);
    }
}
```

```
struct node *rightrotate(struct node *y){
    struct node *x = y->left;
    struct node *T2 = x->right;
    x->right = y;
    y->left = T2;
    y->height = max(getheight(y->right), getheight(y->left)) + 1;
    x->height = max(getheight(x->right), getheight(x->left)) + 1;
    return x;
}

struct node *leftrotate(struct node *x){
    struct node *y = x->right;
    struct node *T2 = y->left;
    y->left = x;
    x->right = T2;
    y->height = max(getheight(y->right), getheight(y->left)) + 1;
    x->height = max(getheight(x->right), getheight(x->left)) + 1;
    return y;
}

void inorder(struct node* T){
    if(T != NULL){
        inorder(T->left);
        printf("%d, ", T->key);
        inorder(T->right);
    }
```

```
}

void preorder(struct node* T){
    if(T != NULL){
        printf("%d, ", T->key);
        preorder(T->left);
        preorder(T->right);
    }
}

void postorder(struct node* T){
    if(T != NULL){
        postorder(T->left);
        postorder(T->right);
        printf("%d, ", T->key);
    }
}

struct node*insert(struct node * T, int data){
    if(T == NULL){
        return create(data);
    }
    if(data < T->key){
        T->left = insert(T->left, data);
    }
    else if(data > T->key){
```

```
        T->right = insert(T->right, data);
    }

    T->height = max(getheight(T->left), getheight(T->right)) + 1;
    int bf = getBalanceFactor(T);

    // LEFT LEFT CASE
    if(bf > 1 && data < T->left->key){
        return rightrotate(T);
    }

    // RIGHT RIGHT CASE
    if(bf < -1 && data > T->right->key){
        return leftrotate(T);
    }

    // LEFT RIGHT CASE
    if(bf > 1 && data > T->left->key){
        T->left = leftrotate(T->left);
        return rightrotate(T);
    }

    //RIGHT LEFT CASE
    if(bf < -1 && data < T->right->key){
        T->right = rightrotate(T->right);
        return leftrotate(T);
```

```
    }

    return T;
}

struct node * minValueNode(struct node* node)
{
    struct node* current = node;

    /* loop down to find the leftmost leaf */
    while (current->left != NULL)
        current = current->left;

    return current;
}

struct node* deleteNode(struct node* root, int key)
{
    if (root == NULL)
        return root;

    if ( key < root->key )
        root->left = deleteNode(root->left, key);

    else if( key > root->key )
        root->right = deleteNode(root->right, key);
```

```
    else
    {
        if( (root->left == NULL) || (root->right == NULL) )
        {
            struct node *temp = root->left ? root->left :root->right;


            if (temp == NULL)
            {
                temp = root;
                root = NULL;
            }
            else
             *root = *temp;
            free(temp);
        }
        else
        {
            struct node* temp = minValueNode(root->right);
            root->key = temp->key;
            root->right = deleteNode(root->right, temp->key);
        }
    }

    if (root == NULL){
```

```
    return root;
  }

  root->height = 1 + max(getheight(root->left),getheight(root->right));


  int bf = getBalanceFactor(root);


  // Left Left Case
  if (bf > 1 && getBalanceFactor(root->left) >= 0)
      return rightrotate(root);

  // Left Right Case
  if (bf > 1 && getBalanceFactor(root->left) < 0)
  {
      root->left =  leftrotate(root->left);
      return rightrotate(root);
  }

  // Right Right Case
  if (bf < -1 && getBalanceFactor(root->right) <= 0)
      return leftrotate(root);

  // Right Left Case
  if (bf < -1 && getBalanceFactor(root->right) > 0)
```

```
    {
        root->right = rightrotate(root->right);
        return leftrotate(root);
    }

    return root;
}

int main()
{
    struct node* root = NULL;

    do{
        int ch, x, del;
        printf("Operations available:\n");
        printf("1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Deletion\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);

        switch(ch){
            case 1:
                printf("Enter element you want to insert in the binary search tree: ");
                scanf("%d", &x);
                root = insert(root, x);
                break;
            case 2:
```

```
            inorder(root);
            break;
        case 3:
            preorder(root);
            break;
        case 4:
            postorder(root);
            break;
        case 5:
            printf("Enter element that you want to delete: ");
            scanf("%d", &del);
            root = deleteNode(root, del);
            printf("After deletion of %d\n", del);
            preorder(root);
            break;
        default:
            exit(0);
      }
   }while(1);


   return 0;
}
```

```
1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Deletion
Enter your choice: 1
Enter element you want to insert in the binary search tree: 5
Operations available:
1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Deletion
Enter your choice: 1
Enter element you want to insert in the binary search tree: 10
Operations available:
1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Deletion
Enter your choice: 1
Enter element you want to insert in the binary search tree: 0
Operations available:
1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Deletion
Enter your choice: 1
Enter element you want to insert in the binary search tree: 6
Operations available:
1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Deletion
Enter your choice: 1
Enter element you want to insert in the binary search tree: 11
Operations available:
1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Deletion
Enter your choice: 1
Enter element you want to insert in the binary search tree: -1
Operations available:
1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Deletion
Enter your choice: 1
Enter element you want to insert in the binary search tree: 1
Operations available:
1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Deletion
Enter your choice: 1
Enter element you want to insert in the binary search tree: 2
Operations available:
1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Deletion
Enter your choice: 3
9, 1, 0, -1, 5, 2, 6, 10, 11, Operations available:
1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Deletion
Enter your choice: 5
Enter element that you want to delete: 10
After deletion of 10
1, 0, -1, 9, 5, 2, 6, 11, Operations available:
1.Insert, 2.Inorder, 3.Preorder, 4.Postorder, 5.Deletion
Enter your choice:
```