1. **Tree Traversal**

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
struct node *left;
int element;
struct node *right;
};
typedef struct node Node;
Node *Insert(Node *Tree, int e);
void Inorder(Node *Tree);
void Preorder(Node *Tree);
void Postorder(Node *Tree);
int main()
{
Node *Tree = NULL;
int n, i, e, ch;
printf("Enter number of nodes in the tree : ");
scanf("%d", &n);
printf("Enter the elements :\n");
for (i = 1; i <= n; i++)
 {
 scanf("%d", &e);
 Tree = Insert(Tree, e);
 }
do
 {
 printf("1. Inorder \n2. Preorder \n3. Postorder \n4. Exit\n");
 printf("Enter your choice : ");
 scanf("%d", &ch);
 switch (ch)
{
 case 1:
 Inorder(Tree);
 printf("\n");
 break;
 case 2:
 Preorder(Tree);
 printf("\n");
 break;
 case 3:
 Postorder(Tree);
 printf("\n");
```

```
 break;
 }
 } while (ch <= 3);
return 0;
}
Node *Insert(Node *Tree, int e)
{
Node *NewNode = malloc(sizeof(Node));
if (Tree == NULL)
 {
 NewNode->element = e;
 NewNode->left = NULL;
 NewNode->right = NULL;
 Tree = NewNode;
 }
else if (e < Tree->element)
 {
 Tree->left = Insert(Tree->left, e);
 }
else if (e > Tree->element)
 {
 Tree->right = Insert(Tree->right, e);
 }
return Tree;
}
void Inorder(Node *Tree)
{
if (Tree != NULL)
 {
 Inorder(Tree->left);
 printf("%d\t", Tree->element);
 Inorder(Tree->right);
 }
}
void Preorder(Node *Tree)
{
if (Tree != NULL)
 {
 printf("%d\t", Tree->element);
 Preorder(Tree->left);
 Preorder(Tree->right);
 }
}
void Postorder(Node *Tree)
```

```
{
if (Tree != NULL)
 {
 Postorder(Tree->left);
 Postorder(Tree->right);
 printf("%d\t", Tree->element);
 }
}
```

OUTPUT

```
Enter number of nodes in the tree : 7
Enter the elements :
10
6
20
8
9
15
21
1. Inorder
2. Preorder
3. Postorder
4. Exit
Enter your choice : 1
6       8       9       10      15      20      21
1. Inorder
2. Preorder
3. Postorder
4. Exit
Enter your choice : 2
10      6       8       9       20      15      21
1. Inorder
2. Preorder
3. Postorder
4. Exit
Enter your choice : 3
9       8       6       15      21      20      10
1. Inorder
2. Preorder
3. Postorder
4. Exit
Enter your choice : 4
```

2. **Binary Search Tree**

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
int data;
struct Node* left;
struct Node* right;
};
struct Node* createNode(int value) {
struct Node* newNode  = (struct Node*)malloc(sizeof(struct Node));
newNode->data =value;
newNode->left = NULL;
newNode->right =NULL;
return newNode;
}
struct Node* insert(struct Node* root, int value) {
if (root == NULL) {
return createNode(value);
}
if (value < root->data) {
root->left =insert(root->left, value);
} else if (value > root->data) {
root->right =insert(root->right, value);
}
return root;
}
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 value) {
if (root == NULL) {
return root;
}
if (value < root->data) {
root->left =deleteNode(root->left,value);
} else if (value > root->data) {
root->right =deleteNode(root->right,value);
} 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;
}
struct Node* search(struct Node* root, int value) {
if (root == NULL || root->data == value) {
return root;
}
if (root->data < value) {
return search(root->right, value);
}
return search(root->left,value);
}
void display(struct Node*root) {
if (root != NULL) {
display(root->left);
printf("%d ", root->data);
display(root->right);
}
}
int main() {
struct Node* root =NULL;
root = insert(root, 50);
insert(root, 30);
insert(root, 20);
insert(root, 40);
insert(root, 70);
insert(root, 60);
insert(root, 80);
printf("Binary Search Tree Inorder Traversal: ");
display(root);
printf("\n");
root = deleteNode(root,20);
printf("Binary Search Tree Inorder Traversal after deleting 20: ");
```

```
display(root);
printf("\n");
struct Node*searchResult =search(root, 30);
if (searchResult !=NULL) {
printf("Element 30 found in the Binary Search Tree.\n");
} else {
printf("Element 30 not found in the Binary Search Tree.\n");
}
return 0;
}
```

OUTPUT

```
Binary Search Tree Inorder Traversal: 20 30 40 50 60 70 80
Binary Search Tree Inorder Traversal after deleting 20: 30 40 50 60 70 80
Element 30 found in the Binary Search Tree.
```

3. **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;
    return n->height;
}

struct Node *createNode(int key){
    struct Node* node = (struct Node *) malloc(sizeof(struct Node));
    node->key = key;
    node->left = NULL;
    node->right = NULL;
    node->height = 1;
    return node;
}

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

int getBalanceFactor(struct Node * n){
    if(n==NULL){
        return 0;
    }
    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;
```

```
    x->height = max(getHeight(x->right), getHeight(x->left)) + 1;
    y->height = max(getHeight(y->right), getHeight(y->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;

    x->height = max(getHeight(x->right), getHeight(x->left)) + 1;
    y->height = max(getHeight(y->right), getHeight(y->left)) + 1;

    return y;
}

struct Node *insert(struct Node *node, int key){
    if (node == NULL)
        return  createNode(key);

    if (key < node->key)
        node->left  = insert(node->left, key);
    else if (key > node->key)
        node->right = insert(node->right, key);

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

    // Left Left Case
        if(bf>1 && key < node->left->key){
            return rightRotate(node);
        }
    // Right Right Case
        if(bf<-1 && key > node->right->key){
            return leftRotate(node);
        }
    // Left Right Case
    if(bf>1 && key > node->left->key){
            node->left = leftRotate(node->left);
            return rightRotate(node);
```

```
        }
    // Right Left Case
    if(bf<-1 && key < node->right->key){
            node->right = rightRotate(node->right);
            return leftRotate(node);
        }
    return node;
}

void preOrder(struct Node *root)
{
    if(root != NULL)
    {
        printf("%d ", root->key);
        preOrder(root->left);
        preOrder(root->right);
    }
}

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


    root = insert(root, 1);
    root = insert(root, 2);
    root = insert(root, 4);
    root = insert(root, 5);
    root = insert(root, 6);
    root = insert(root, 3);
    preOrder(root);
    return 0;
}
```

OUTPUT

4 2 1 3 5 6