1. WRITE A C PROGRAM FOR BINARY TREE.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
   int data;
   struct Node* left;
   struct Node* right;
} Node;
int main() {
   Node* root = (Node*)malloc(sizeof(Node));
   root->data = 50;
   root->left = (Node*)malloc(sizeof(Node));
   root->right = (Node*)malloc(sizeof(Node));
   root->left->data = 30;
   root->left->left = (Node*)malloc(sizeof(Node));
   root->left->right = NULL;
   root->right->data = 70;
   root->right->left = NULL;
   root->right->right = (Node*)malloc(sizeof(Node));
   root->left->left->data = 20;
   root->left->left->left = NULL;
   root->left->left->right = NULL;
   root->right->right->data = 80;
   root->right->right->left = NULL;
   root->right->right->right = NULL;
   printf("In-order traversal: ");
   Node* stack[100];
   int top = -1;
   Node* current = root;
   while (current != NULL || top != -1) {
      while (current != NULL) {
         stack[++top] = current;
         current = current->left;
      }
      current = stack[top--];
      printf("%d ", current->data);
      current = current->right;
   }
   printf("\n");
   printf("Pre-order traversal: ");
   top = -1;
   current = root;
   while (current != NULL || top != -1) {
      while (current != NULL) {
         printf("%d ", current->data);
         stack[++top] = current;
         current = current->left;
      }
      current = stack[top--];
      current = current->right;
   }
   printf("\n");
   free(root->left->left);
   free(root->left);
   free(root->right->right);
   free(root->right);
   free(root);
   return 0;
}
```

SAMPLE OUTPUT:

```
In-order traversal: 20 30 50 70 80
Pre-order traversal: 50 30 20 70 80 \
```

2. WRITE A C PROGRAM FOR BINARY SEARCH TREE.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;
    struct Node* left;
    struct Node* right;
} Node;
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
Node* insertNode(Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (data < root->data) {
        root->left = insertNode(root->left, data);
    } else if (data > root->data) {
        root->right = insertNode(root->right, data);
    }
    return root;
}
Node* findMin(Node* root) {
    while (root->left != NULL) {
        root = root->left;
    }
    return root;
}
Node* deleteNode(Node* root, int data) {
    if (root == NULL) {
        return root;
    }
    if (data < root->data) {
        root->left = deleteNode(root->left, data);
    } else if (data > root->data) {
        root->right = deleteNode(root->right, data);
    } else {
        if (root->left == NULL) {
            Node* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            Node* temp = root->left;
            free(root);
            return temp;
        }
        Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}
Node* searchNode(Node* root, int data) {
    if (root == NULL || root->data == data) {
        return root;
    }
    if (data < root->data) {
        return searchNode(root->left, data);
    }
    return searchNode(root->right, data);
```

```
}
void inorderTraversal(Node* root) {
   if (root != NULL) {
      inorderTraversal(root->left);
      printf("%d ", root->data);
      inorderTraversal(root->right);
   }
}
int main() {
   Node* root = NULL;
   root = insertNode(root, 50);
   root = insertNode(root, 30);
   root = insertNode(root, 20);
   root = insertNode(root, 40);
   root = insertNode(root, 70);
   root = insertNode(root, 60);
   root = insertNode(root, 80);
   printf("In-order traversal: ");
   inorderTraversal(root);
   printf("\n");
   int searchValue = 40;
   Node* searchResult = searchNode(root, searchValue);
   if (searchResult != NULL) {
      printf("Value %d found in the tree.\n", searchValue);
   } else {
      printf("Value %d not found in the tree.\n", searchValue);
   }
   int deleteValue = 20;
   root = deleteNode(root, deleteValue);
   printf("In-order traversal after deleting : ", deleteValue);
   inorderTraversal(root);
   printf("\n");
   while (root != NULL) {
      root = deleteNode(root, root->data);
   }
   return 0;
}
```

SAMPLE OUTPUT:

```
In-order traversal: 20 30 40 50 60 70 80
Value 40 found in the tree.
In-order traversal after deleting : 30 40 50 60 70 80
```

3. WRITE A C PROGRAM FOR BINARY TREE TRAVERSAL (IN ORDER, PRE ORDER AND POST ORDER).

```
#include <stdio.h>
#include <stdlib.h>
int main() {
   typedef struct Node {
      int data;
      struct Node* left;
      struct Node* right;
   } Node;
   Node* root = (Node*)malloc(sizeof(Node));
   root->data = 1;
   root->left = (Node*)malloc(sizeof(Node));
   root->right = (Node*)malloc(sizeof(Node));
   root->left->data = 2;
   root->left->left = (Node*)malloc(sizeof(Node));
   root->left->right = (Node*)malloc(sizeof(Node));
   root->right->data = 3;
   root->right->left = NULL;
   root->right->right = (Node*)malloc(sizeof(Node));
   root->left->left->data = 4;
   root->left->left->left = NULL;
```

```
    root->left->left->right = NULL;
    root->left->right->data = 5;
    root->left->right->left = NULL;
    root->left->right->right = NULL;
    root->right->right->data = 6;
    root->right->right->left = NULL;
    root->right->right->right = NULL;
    printf("In-order traversal: ");
    Node* stack[100];
    int top = -1;
    Node* current = root;
    while (current != NULL || top != -1) {
        while (current != NULL) {
            stack[++top] = current;
            current = current->left;
        }
        current = stack[top--];
        printf("%d ", current->data);
        current = current->right;
    }
    printf("\n");
    printf("Pre-order traversal: ");
    top = -1;
    current = root;
    while (current != NULL || top != -1) {
        while (current != NULL) {
            printf("%d ", current->data);
            stack[++top] = current;
            current = current->left;
        }
        current = stack[top--];
        current = current->right;
    }
    printf("\n");
    printf("Post-order traversal: ");
    Node* temp;
    top = -1;
    Node* lastVisited = NULL;
    stack[++top] = root;
    while (top != -1) {
        temp = stack[top];
        if ((temp->left == NULL && temp->right == NULL) ||
            (lastVisited != NULL && (lastVisited == temp->left || lastVisited == temp->right))) {
            printf("%d ", temp->data);
            lastVisited = temp;
            top--;
        } else {
            if (temp->right != NULL) {
                stack[++top] = temp->right;
            }
            if (temp->left != NULL) {
                stack[++top] = temp->left;
            }
        }
    }
    printf("\n");
    free(root->left->left);
    free(root->left->right);
    free(root->left);
    free(root->right->right);
    free(root->right);
    free(root);
    return 0;
}
```

SAMPLE OUTPUT:

In-order traversal: 4 2 5 1 3 6
Pre-order traversal: 1 2 4 5 3 6
Post-order traversal: 4 5 2 6 3 1