NAME : P.VINEETHA

DATA STRUCTURE : DAY-5

1. Write a C programming for Binary Search Tree.

```
#include <stdio.h>

#include <stdlib.h>

struct Node {

        int key;

        struct Node *left, *right;

};

struct Node* createNode(int key) {

        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

        newNode->key = key;

        newNode->left = newNode->right = NULL;

        return newNode;

}

struct Node* insert(struct Node* root, int key) {

        if (root == NULL) return createNode(key);

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

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

        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 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) {
            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->key = temp->key;
        root->right = deleteNode(root->right, temp->key);
    }
```

```
        return root;

}

struct Node* search(struct Node* root, int key) {

        if (root == NULL || root->key == key) return root;

        if (root->key < key) return search(root->right, key);

        return search(root->left, key);

}

void inorder(struct Node* root) {

        if (root != NULL) {

                inorder(root->left);

                printf("%d ", root->key);

                inorder(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("Inorder traversal of the BST: ");

        inorder(root);
```

```
printf("\n\nDelete 20\n");

        root = deleteNode(root, 20);

        printf("Inorder traversal of the modified BST: ");

        inorder(root);

printf("\n\nSearch for 40: ");

        if (search(root, 40) != NULL)

                printf("Key found");

        else

                printf("Key not found");

return 0;

}
```

OUTPUT:

```
Inorder traversal of the modified BST: 30 40 50 60 70 80

Delete 20

Inorder traversal of the modified BST: 30 40 50 60 70 80

Search for 40: Key found
```

2 . Write a c programming for   Binary tree.

```
#include<stdio.h>

#include<stdlib.h>

struct BTnode

{

int keyVal;
```

```
struct BTnode *leftNode;
struct BTnode *rightNode;
};
struct BTnode *getNode(int value)
{
struct BTnode *newNode = malloc(sizeof(struct BTnode));
newNode->keyVal = value;
newNode->leftNode = NULL;
newNode->rightNode = NULL;
return newNode;
}
struct BTnode *insert(struct BTnode *rootNode, int value)
{
if(rootNode == NULL)
return getNode(value);
if(rootNode->keyVal < value)
rootNode->rightNode = insert(rootNode->rightNode,value);
else if(rootNode->keyVal > value)
rootNode->leftNode = insert(rootNode->leftNode,value);
return rootNode;
}
void insertorder(struct BTnode *rootNode)
{
if(rootNode == NULL)
return;
```

```
insertorder(rootNode->leftNode);

printf("%d ",rootNode->keyVal);

insertorder(rootNode->rightNode);

}

int main()

{

struct BTnode *rootNode = NULL;

rootNode = insert(rootNode,7);

rootNode = insert(rootNode,4);

rootNode = insert(rootNode,8);

rootNode = insert(rootNode,1);

rootNode = insert(rootNode,5);

rootNode = insert(rootNode,2);

rootNode = insert(rootNode,9);

rootNode = insert(rootNode,3);

insertorder(rootNode);

return 0;

}
```

OUTPUT :

```
1 2 3 4 5 7 8 9
```

3. Write a C program for Binary Tree Traversal using Inorder,Preorder,and Postorder.

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

```
#include <stdlib.h>

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

struct Node* createNode(int data) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = data;
        newNode->left = NULL;
        newNode->right = NULL;
        return newNode;
}

void inOrder(struct Node* root) {
        if (root == NULL) return;
        inOrder(root->left);
        printf("%d ", root->data);
        inOrder(root->right);
}

void preOrder(struct Node* root) {
        if (root == NULL) return;
        printf("%d ", root->data);

        preOrder(root->left);

        preOrder(root->right);

}


void postOrder(struct Node* root) {

        if (root == NULL) return;

        postOrder(root->left);

        postOrder(root->right);

        printf("%d ", root->data);

}


int main() {

        struct Node* root = createNode(1);

        root->left = createNode(2);

        root->right = createNode(3);

        root->left->left = createNode(4);

        root->left->right = createNode(5);


        printf("Inorder traversal: ");

        inOrder(root);

        printf("\n");


        printf("Preorder traversal: ");

        preOrder(root);
```

```
    printf("\n");

    printf("Postorder traversal: ");
    postOrder(root);
    printf("\n");

    return 0;
}
```

OUTPUT :

```
Inorder traversal: 4 2 5 1 3
Preorder traversal: 1 2 4 5 3
Postorder traversal: 4 5 2 3 1
```