DATA STRUCTURE

DAY 5 – 30/07/2024

1.Binary tree

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
 typedef struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
} TreeNode;
 TreeNode* createNode(int data) {
    TreeNode* newNode = (TreeNode*)malloc(sizeof(TreeNode));
    if (newNode == NULL) {
        printf("Memory allocation error\n");
        exit(1);
    }
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
 TreeNode* insertNode(TreeNode* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
     TreeNode* queue[100];
    int front = 0, rear = 0;
    queue[rear++] = root;
    while (front < rear) {
        TreeNode* current = queue[front++];
        if (current->left == NULL) {
            current->left = createNode(data);
            return root;
        } else {
            queue[rear++] = current->left;
        }
        if (current->right == NULL) {
            current->right = createNode(data);
            return root;
        } else {
            queue[rear++] = current->right;
        }
    }
    return root;
}
 void inOrderTraversal(TreeNode* root) {
    if (root != NULL) {
        inOrderTraversal(root->left);
        printf("%d ", root->data);
        inOrderTraversal(root->right);
    }
}
 void preOrderTraversal(TreeNode* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preOrderTraversal(root->left);
        preOrderTraversal(root->right);
    }
}
void postOrderTraversal(TreeNode* root) {
    if (root != NULL) {
        postOrderTraversal(root->left);
        postOrderTraversal(root->right);
        printf("%d ", root->data);
    }
}
void levelOrderTraversal(TreeNode* root) {
    if (root == NULL) return;
    TreeNode* queue[100];
    int front = 0, rear = 0;
    queue[rear++] = root;
    while (front < rear) {
        TreeNode* current = queue[front++];
        printf("%d ", current->data);
        if (current->left != NULL) {
            queue[rear++] = current->left;
        }
        if (current->right != NULL) {
            queue[rear++] = current->right;
        }
    }
}
void freeTree(TreeNode* root) {
    if (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}
int main() {
    TreeNode* root = NULL;
     root = insertNode(root, 1);
    insertNode(root, 2);
    insertNode(root, 3);
    insertNode(root, 4);
    insertNode(root, 5);
    insertNode(root, 6);
    insertNode(root, 7);
     printf("In-order traversal: ");
    inOrderTraversal(root);
    printf("\n");
    printf("Pre-order traversal: ");
    preOrderTraversal(root);
    printf("\n");
    printf("Post-order traversal: ");
    postOrderTraversal(root);
    printf("\n");
    printf("Level-order traversal: ");
    levelOrderTraversal(root);
    printf("\n");
    freeTree(root);
    return 0;
}
```

OUTPUT:

```
In-order traversal: 4 2 5 1 6 3 7

Pre-order traversal: 1 2 4 5 3 6 7

Post-order traversal: 4 5 2 6 7 3 1

Level-order traversal: 1 2 3 4 5 6 7
```

2.Binary Search tree tranversal

PROGRAM:

```
#include <stdio.h>

#include <stdlib.h>

 typedef struct TreeNode {

    int data;

    struct TreeNode* left;

    struct TreeNode* right;

} TreeNode;

 TreeNode* createNode(int data) {

    TreeNode* newNode = (TreeNode*)malloc(sizeof(TreeNode));

    if (newNode == NULL) {

        printf("Memory allocation error\n");

        exit(1);

    }

    newNode->data = data;

    newNode->left = NULL;

    newNode->right = NULL;

    return newNode;

}

 TreeNode* insertNode(TreeNode* root, int data) {

    if (root == NULL) {

        return createNode(data);

    }
    if (data < root->data) {
        root->left = insertNode(root->left, data);
    } else {
        root->right = insertNode(root->right, data);
    }

    return root;
}
 TreeNode* findMin(TreeNode* root) {
    while (root && root->left != NULL) {
        root = root->left;
    }
    return root;
}
 TreeNode* deleteNode(TreeNode* 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) {
            TreeNode* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            TreeNode* temp = root->left;
            free(root);
            return temp;
        }
        TreeNode* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}
 void inOrderTraversal(TreeNode* root) {
    if (root != NULL) {
        inOrderTraversal(root->left);
        printf("%d ", root->data);
        inOrderTraversal(root->right);
    }
}
 TreeNode* searchNode(TreeNode* root, int data) {
    if (root == NULL || root->data == data) {
        return root;
    }
    if (data < root->data) {
        return searchNode(root->left, data);
    } else {
        return searchNode(root->right, data);
    }
}
void freeTree(TreeNode* root) {
    if (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}
int main() {
    TreeNode* root = NULL;
     int valuesToInsert[] = {20, 15, 25, 12, 18, 65, 45};
    for (int i = 0; i < sizeof(valuesToInsert) / sizeof(valuesToInsert[0]); i++) {
        root = insertNode(root, valuesToInsert[i]);
    }
    printf("In-order traversal before deletion: ");
    inOrderTraversal(root);
    printf("\n");
    int valuesToDelete[] = {18, 65, 14};
    for (int i = 0; i < sizeof(valuesToDelete) / sizeof(valuesToDelete[0]); i++) {
        root = deleteNode(root, valuesToDelete[i]);
    }
    printf("In-order traversal after deletion: ");
    inOrderTraversal(root);
    printf("\n");
    int valuesToSearch[] = {15, 18, 45, 14};
    for (int i = 0; i < sizeof(valuesToSearch) / sizeof(valuesToSearch[0]); i++) {
        TreeNode* result = searchNode(root, valuesToSearch[i]);
        if (result) {
            printf("Value %d found in the BST.\n", valuesToSearch[i]);
        } else {
            printf("Value %d not found in the BST.\n", valuesToSearch[i]);
        }
    }
    freeTree(root);
    return 0;
```

```
}
```

OUTPUT:

```
In-order traversal before deletion: 12 15 18 20 25 45 65
In-order traversal after deletion: 12 15 20 25
Value 15 found in the BST.
Value 18 not found in the BST.
Value 45 not found in the BST.
Value 14 not found in the BST.
```

3.Binary tree tranverse

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
typedef struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
} TreeNode;
 TreeNode* createNode(int data) {
    TreeNode* newNode = (TreeNode*)malloc(sizeof(TreeNode));
    if (newNode == NULL) {
        printf("Memory allocation error\n");
        exit(1);
    }
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
void inOrderTraversal(TreeNode* root) {
    if (root != NULL) {
        inOrderTraversal(root->left);
        printf("%d ", root->data);
        inOrderTraversal(root->right);
    }
}
void preOrderTraversal(TreeNode* root) {
    if (root != NULL) {
        printf("%d ", root->data);
        preOrderTraversal(root->left);
        preOrderTraversal(root->right);
    }
}
void postOrderTraversal(TreeNode* root) {
    if (root != NULL) {
        postOrderTraversal(root->left);
        postOrderTraversal(root->right);
        printf("%d ", root->data);
    }
}
void levelOrderTraversal(TreeNode* root) {
    if (root == NULL) return;
    TreeNode** queue = (TreeNode**)malloc(sizeof(TreeNode*) * 100); // assuming a max of 100 nodes
    int front = 0;
    int rear = 0;
    queue[rear++] = root;
    while (front < rear) {
        TreeNode* current = queue[front++];
        printf("%d ", current->data);
        if (current->left != NULL) {
            queue[rear++] = current->left;
        }
        if (current->right != NULL) {
            queue[rear++] = current->right;
        }
    }
    free(queue);
}
 void freeTree(TreeNode* root) {
    if (root != NULL) {
        freeTree(root->left);
        freeTree(root->right);
        free(root);
    }
}
int main() {
    TreeNode* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);
    root->right->left = createNode(6);
    root->right->right = createNode(7);
     printf("In-order traversal: ");
    inOrderTraversal(root);
    printf("\n");
    printf("Pre-order traversal: ");
    preOrderTraversal(root);
    printf("\n");
    printf("Post-order traversal: ");
    postOrderTraversal(root);
    printf("\n");
    printf("Level-order traversal: ");
    levelOrderTraversal(root);
    printf("\n");
     freeTree(root);
    return 0;
}
```

OUTPUT:

```
In-order traversal: 4 2 5 1 6 3 7
Pre-order traversal: 1 2 4 5 3 6 7
Post-order traversal: 4 5 2 6 7 3 1
Level-order traversal: 1 2 3 4 5 6 7
```