

DAY-11

1. Write a program to implement a tree and print the output like a tree Format?

Code:

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

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

```
#include <math.h>
```

```
typedef struct Treenode {
```

```
    int data;
```

```
    struct Treenode *left, *right;
```

```
} Treenode;
```

```
typedef struct {
```

```
    Treenode *root;
```

```
} Tree;
```

```
Treenode* newTreenode(int data) {
```

```
    Treenode* node = (Treenode*)malloc(sizeof(Treenode));
```

```
node->data = data;
```

```
node->left = node->right = NULL;
```

```
return node;
```

```
}
```

```
int height(Treenode *root) {
```

```
    if (root == NULL)
```

```
        return 0;
```

```
    int left_height = height(root->left);
```

```
    int right_height = height(root->right);
```

```
    return (left_height > right_height ? left_height : right_height) + 1;
```

```
}
```

```
int getcol(int h) {
```

```
    if (h == 1)
```

```
        return 1;
```

```

    return getcol(h - 1) + getcol(h - 1) + 1;

}

void printTree(int **M, Treenode *root, int col, int row, int height) {

    if (root == NULL)

        return;

    M[row][col] = root->data;

    printTree(M, root->left, col - pow(2, height - 2), row + 1, height - 1);

    printTree(M, root->right, col + pow(2, height - 2), row + 1, height - 1);

}

void TreePrinter(Tree tree) {

    int h = height(tree.root);

    int col = getcol(h);

    int **M = (int **)malloc(h * sizeof(int *));

```

```
for (int i = 0; i < h; i++) {

    M[i] = (int *)malloc(col * sizeof(int));

    for (int j = 0; j < col; j++) {

        M[i][j] = 0;

    }

}

printTree(M, tree.root, col / 2, 0, h);

for (int i = 0; i < h; i++) {

    for (int j = 0; j < col; j++) {

        if (M[i][j] == 0)

            printf(" ");

        else

            printf("%d ", M[i][j]);

    }

}
```

```

    printf("\n");

}

for (int i = 0; i < h; i++) {

    free(M[i]);

}

free(M);

}

Treenode* insertLevelOrder(int arr[], Treenode* root, int i, int n) {

    if (i < n) {

        Treenode *temp = newTreenode(arr[i]);

        root = temp;

        root->left = insertLevelOrder(arr, root->left, 2 * i + 1, n);

        root->right = insertLevelOrder(arr, root->right, 2 * i + 2, n);

```

```
}
```

```
return root;
```

```
}
```

```
int main() {
```

```
    Tree myTree;
```

```
    myTree.root = NULL;
```

```
    int n;
```

```
    printf("Enter the number of nodes in the tree: ");
```

```
    scanf("%d", &n);
```

```
    int *arr = (int *)malloc(n * sizeof(int));
```

```
    printf("Enter the nodes in level order:\n");
```

```
    for (int i = 0; i < n; i++) {
```

```
        scanf("%d", &arr[i]);
```

```
    }
```

```

myTree.root = insertLevelOrder(arr, myTree.root, 0, n);

printf("Tree structure:\n");

TreePrinter(myTree);

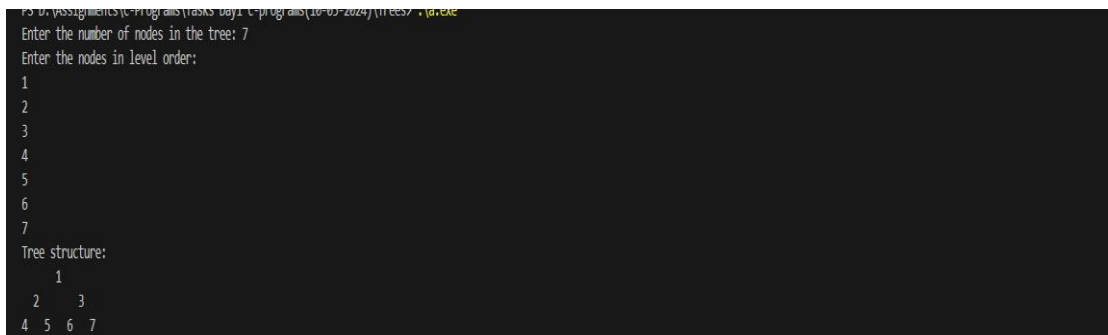
free(arr);

return 0;

}

```

Output:



```

PS D:\Assignments\c-programs\tasks\day1\c-programs\10-03-2024\trees> .\a.exe
Enter the number of nodes in the tree: 7
Enter the nodes in level order:
1
2
3
4
5
6
7
Tree structure:
1
2 3
4 5 6 7

```

2. Write a program to implement AVL tree by using Linked list ?

Code:

```

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

struct Node {

    int key;

    struct Node *left;

```

```

    struct Node *right;

    int height;

};

int max(int a, int b);

int height(struct Node *N) {

    if (N == NULL)

        return 0;

    return N->height;

}

int max(int a, int b) {

    return (a > b) ? a : b;

}

struct Node *newNode(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);

}

```



```

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(height(y->left), height(y->right)) + 1;

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

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

    return y;

}

```

```

int getBalance(struct Node *N) {

    if (N == NULL)

        return 0;

}

```

```

return height(N->left) - height(N->right);
}

struct Node *insertNode(struct Node *node, int key) {

// Find the correct position to insertNode the node and insertNode it

if (node == NULL)

return (newNode(key));

if (key < node->key)

node->left = insertNode(node->left, key);

else if (key > node->key)

node->right = insertNode(node->right, key);

else

return node;

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

int balance = getBalance(node);

if (balance > 1 && key < node->left->key)

return rightRotate(node);

if (balance < -1 && key > node->right->key)

return leftRotate(node);

if (balance > 1 && key > node->left->key) {

node->left = leftRotate(node->left);

return rightRotate(node);

}

```

```

if (balance < -1 && key < node->right->key) {

    node->right = rightRotate(node->right);

    return leftRotate(node);

}

return node;

}

struct Node *minValueNode(struct Node *node) {

    struct Node *current = node;

    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(height(root->left),

    height(root->right));

int balance = getBalance(root);

if (balance > 1 && getBalance(root->left) >= 0)

    return rightRotate(root);

if (balance > 1 && getBalance(root->left) < 0) {

```

```

    root->left = leftRotate(root->left);

    return rightRotate(root);

}

if (balance < -1 && getBalance(root->right) <= 0)

    return leftRotate(root);

if (balance < -1 && getBalance(root->right) > 0) {

    root->right = rightRotate(root->right);

    return leftRotate(root);

}

return root;

}

void printPreOrder(struct Node *root) {

    if (root != NULL) {

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

        printPreOrder(root->left);

        printPreOrder(root->right);

    }

}

void inorder(struct Node *root){
    if(root==NULL){
        // printf("The tree is empty\n");
        return ;
    }
    inorder(root->left);
    printf("%d ", root->key);

```

```

        inorder(root->right);
    }
void postorder(struct Node *root){
    if(root==NULL){
        // printf("The tree is empty\n");
        return ;
    }
    postorder(root->left);
    postorder( root->right);
    printf("%d ",root->key);
}

```

```

int main() {

    struct Node *root = NULL;

    root = insertNode(root, 2);

    root = insertNode(root, 1);

    root = insertNode(root, 7);

    root = insertNode(root, 4);

    root = insertNode(root, 5);

    root = insertNode(root, 3);

    root = insertNode(root, 8);

    printPreOrder(root);

    root = deleteNode(root, 3);

    printf("\nAfter deletion: ");

    struct Node *r=root;
    printf("\nPre order traversal:\n");
    printPreOrder(root);
    printf("\nIn order traversal:\n");
    inorder(root);
    printf("\nPost order traversal:\n");
    postorder(root);
}

```

```
    return 0;  
}
```

Output:

```
4 2 1 3 7 5 8  
After deletion:  
Pre order traversal:  
4 2 1 7 5 8  
In order traversal:  
1 2 4 5 7 8  
Post order traversal:  
1 2 5 8 7 4
```