```c
// Name: Ayush Gupta
// Roll: mns2025027

// 1. For a given binary tree, write a program to find the level
for the user given element.
// (assuming the level of root node is 0)
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
   char data;
   struct node *left, *right;
} Node;

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

Node *queue[100];
int front = 0, rear = 0;

void levelorder(Node *root)
{
   if (!root) return;

   front = rear = 0;
   queue[rear++] = root;
```

```
    while (front < rear)
    {
        Node *current = queue[front++];

        printf("%c", current->data);

        if (current->left) queue[rear++] = current->left;
        if (current->right) queue[rear++] = current->right;
    }
}

int findlevel(Node *root, char val, int level)
{
    if (!root) return -1;
    if (val == root->data) return level;

    int leftlevel = findlevel(root->left, val, level + 1);
    if (leftlevel != -1) return leftlevel;
    return findlevel(root->right, val, level + 1);
}

int main()
{
    Node *mult1 = createNode('*');
    Node *mult2 = createNode('*');
    Node *add1 = createNode('+');
    Node *add2 = createNode('+');
    Node *sub = createNode('-');
    Node *div = createNode('/');

    Node *two1 = createNode('2');
```

```
    Node *two2 = createNode('2');
    Node *three = createNode('3');
    Node *four = createNode('4');
    Node *five = createNode('5');
    Node *six = createNode('6');
    Node *eight = createNode('8');

    mult1->left = three;
    mult1->right = four;
    div->left = six;
    div->right = two1;
    add1->left = mult1;
    add1->right = div;
    add2->left = eight;
    add2->right = two2;
    sub->left = add1;
    sub->right = add2;
    mult2->left = five;
    mult2->right = sub;

    char val;
    printf("Enter the value to search: ", &val);
    scanf("%c", &val);

    printf("Your value %c exists at the level %d", val,
findlevel(mult2, val, 0));
    return 0;
}

// 2. For a given binary tree, write a program to traverse the
tree using level-order traversing.
#include <stdio.h>
```

```c
#include <stdlib.h>

typedef struct node
{
   char data;
   struct node *left, *right;
} Node;

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

Node *queue[100];
int front = 0, rear = 0;

void levelorder(Node *root)
{
   if (!root) return;

   front = rear = 0;
   queue[rear++] = root;

   while (front < rear)
   {
       Node *current = queue[front++];

       printf("%c", current->data);
```

```
        if (current->left) queue[rear++] = current->left;
        if (current->right) queue[rear++] = current->right;
    }
}

int main()
{
    Node *mult1 = createNode('*');
    Node *mult2 = createNode('*');
    Node *add1 = createNode('+');
    Node *add2 = createNode('+');
    Node *sub = createNode('-');
    Node *div = createNode('/');

    Node *two1 = createNode('2');
    Node *two2 = createNode('2');
    Node *three = createNode('3');
    Node *four = createNode('4');
    Node *five = createNode('5');
    Node *six = createNode('6');
    Node *eight = createNode('8');

    mult1->left = three;
    mult1->right = four;
    div->left = six;
    div->right = two1;
    add1->left = mult1;
    add1->right = div;
    add2->left = eight;
    add2->right = two2;
    sub->left = add1;
    sub->right = add2;
```

```
   mult2->left = five;
   mult2->right = sub;

   levelorder(mult2);
   return 0;
}

// 3. For a given binary tree, write a program to count the
total number of nodes, internal
// nodes and external nodes.
#include <stdio.h>
#include <stdlib.h>

int internal = 0, external = 0, total = 0;

typedef struct node
{
   char data;
   struct node *left, *right;
} Node;

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

void countNodes(Node *root)
{
   if (!root) return;
```

```
    total++;

    if (root->left == NULL && root->right == NULL)
        external++;
    else
        internal++;
    countNodes(root->left);
    countNodes(root->right);
}

int main()
{
    Node *mult1 = createNode('*');
    Node *mult2 = createNode('*');
    Node *add1 = createNode('+');
    Node *add2 = createNode('+');
    Node *sub = createNode('-');
    Node *div = createNode('/');

    Node *two1 = createNode('2');
    Node *two2 = createNode('2');
    Node *three = createNode('3');
    Node *four = createNode('4');
    Node *five = createNode('5');
    Node *six = createNode('6');
    Node *eight = createNode('8');

    mult1->left = three;
    mult1->right = four;
    div->left = six;
    div->right = two1;
```

```
   add1->left = mult1;
   add1->right = div;
   add2->left = eight;
   add2->right = two2;
   sub->left = add1;
   sub->right = add2;
   mult2->left = five;
   mult2->right = sub;

   countNodes(mult2);

   printf("\nNumber of internal nodes: %d ", internal);
   printf("\nNumber of external nodes: %d ", external);
   printf("\nNumber of total nodes: %d ", total);
   return 0;
}

// 4. For a given binary tree, write a program to create a
binary tree that is a mirror image of
// the given binary tree.
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
   char data;
   struct node *left, *right;
} Node;

Node *createNode(char data)
{
   Node *newNode = malloc(sizeof(Node));
```

```
    newNode->data = data;
    newNode->right = newNode->left = NULL;
    return newNode;
}

void mirror(Node *root)
{
    if (!root) return;
    Node *temp = root->left;
    root->left = root->right;
    root->right = temp;

    mirror(root->left);
    mirror(root->right);
}

void inorder(Node *root)
{
    if (!root) return;
    if (root->right || root->left) printf("[");
    inorder(root->left);
    printf(" %c ", root->data);
    inorder(root->right);
    if (root->right || root->left) printf("]");
}

int main()
{
    Node *mult1 = createNode('*');
    Node *mult2 = createNode('*');
    Node *add1 = createNode('+');
    Node *add2 = createNode('+');
```

```
    Node *sub = createNode('-');
    Node *div = createNode('/');

    Node *two1 = createNode('2');
    Node *two2 = createNode('2');
    Node *three = createNode('3');
    Node *four = createNode('4');
    Node *five = createNode('5');
    Node *six = createNode('6');
    Node *eight = createNode('8');

    mult1->left = three;
    mult1->right = four;
    div->left = six;
    div->right = two1;
    add1->left = mult1;
    add1->right = div;
    add2->left = eight;
    add2->right = two2;
    sub->left = add1;
    sub->right = add2;
    mult2->left = five;
    mult2->right = sub;

    inorder(mult2);
    mirror(mult2);
    printf("\n");
    inorder(mult2);
    return 0;
}
```

```c
// 5. Write a program to create a binary tree from preorder and
inorder traversal given by the
// user.
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
   char data;
   struct node *left, *right;
} Node;

Node *createNode(char data)
{
   Node *n = malloc(sizeof(Node));
   n->data = data;
   n->left = n->right = NULL;
   return n;
}

int search(char inorder[], int start, int end, char value)
{
   for (int i = start; i <= end; i++)
       if (inorder[i] == value) return i;
   return -1;
}

Node *buildTree(char inorder[], char preorder[], int inStart,
int inEnd, int *preIndex)
{
   if (inStart > inEnd) return NULL;
   char curr = preorder[*preIndex];
```

```
    (*preIndex)++;
    Node *node = createNode(curr);
    if (inStart == inEnd) return node;
    int inIndex = search(inorder, inStart, inEnd, curr);
    node->left = buildTree(inorder, preorder, inStart, inIndex -
1, preIndex);
    node->right = buildTree(inorder, preorder, inIndex + 1,
inEnd, preIndex);
    return node;
}

void printPostorder(Node *root)
{
    if (!root) return;
    printPostorder(root->left);
    printPostorder(root->right);
    printf("%c ", root->data);
}

int main()
{
    int n;
    printf("Enter number of nodes: ");
    scanf("%d", &n);

    char preorder[n], inorder[n];
    printf("Enter preorder traversal: ");
    for (int i = 0; i < n; i++) scanf(" %c", &preorder[i]);
    printf("Enter inorder traversal: ");
    for (int i = 0; i < n; i++) scanf(" %c", &inorder[i]);

    int preIndex = 0;
```

```
   Node *root = buildTree(inorder, preorder, 0, n - 1,
&preIndex);

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

   return 0;
}

// 6. For the binary tree created in program-5, check whether
the given binary tree is a strict
// binary tree or not.
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
   char data;
   struct node *left, *right;
} Node;

Node *createNode(char data)
{
   Node *n = malloc(sizeof(Node));
   n->data = data;
   n->left = n->right = NULL;
   return n;
}

int isStrictBinaryTree(Node *root)
{
```

```
    if (!root) return 1;
    if (!root->left && !root->right) return 1;
    if (root->left && root->right)
        return isStrictBinaryTree(root->left) &&
isStrictBinaryTree(root->right);
    return 0;
}

int main()
{
    Node *root = createNode('A');
    root->left = createNode('B');
    root->right = createNode('C');
    root->left->left = createNode('D');
    root->left->right = createNode('E');

    if (isStrictBinaryTree(root))
        printf("The tree is a Strict Binary Tree\n");
    else
        printf("The tree is NOT a Strict Binary Tree\n");

    return 0;
}
```