Shrihari K Mangle
CSB2-74 12010517

LAB-6

6. Implement a following problem statement in C/C++: Create a Binary Tree and perform Preorder, Inorder and Postorder traversal on it.

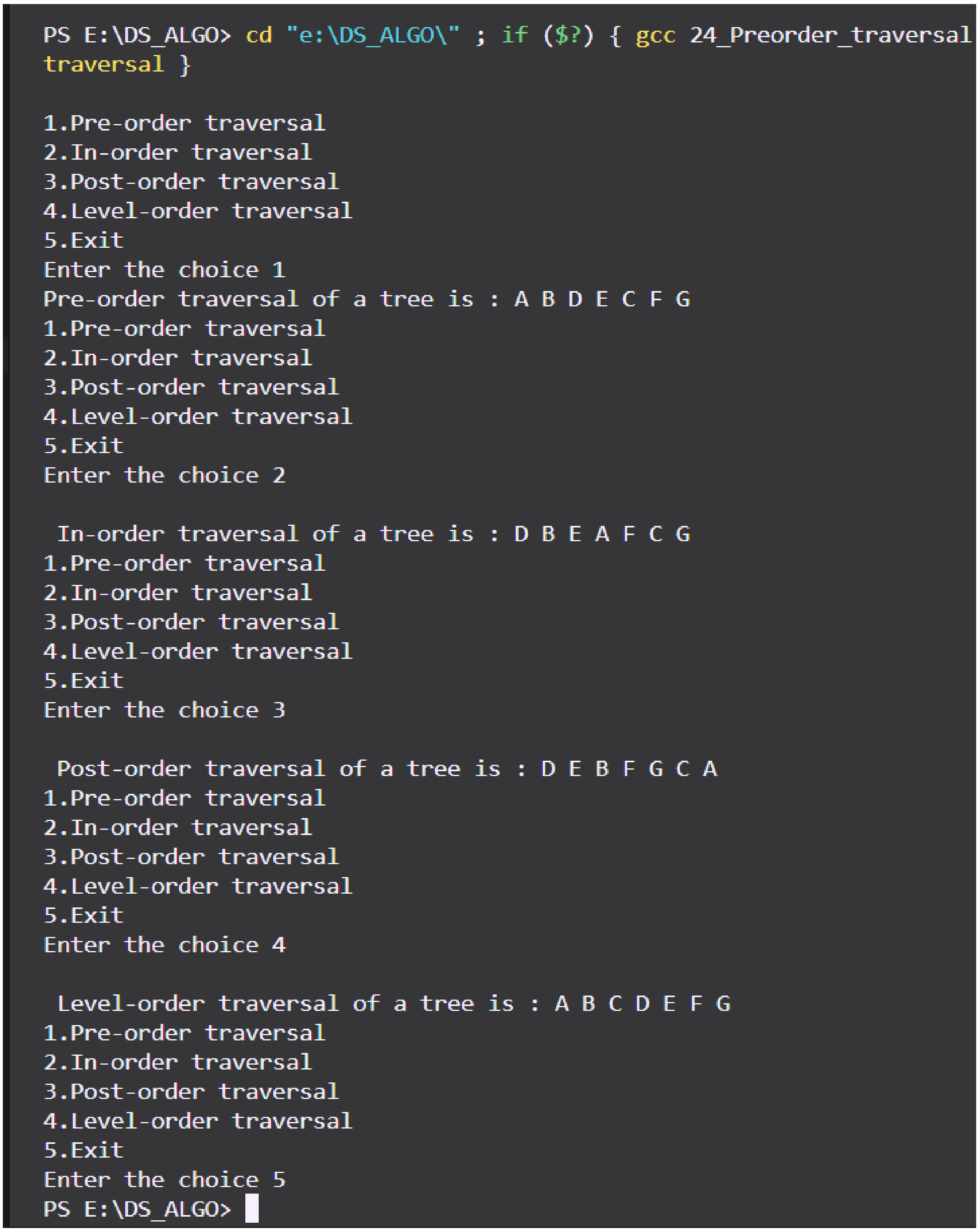
CODE :

```c
#include <stdio.h>
#include <malloc.h>
#define MAX_Q_SIZE 500
typedef struct node
{
    char data;
    struct node *left;
    struct node *right;
} node;

typedef struct stack
{
    struct node *DATA;
    struct stack *next;
} stack;

node *create(char value);
void preorder(node *root);
int emptystack(stack *top);
void push(stack **top, node *root);
node *pop(stack **top);
void inorder(node *root);
void postorder(node *root);
node *top(stack *top);
node **createQueue(int *front, int *rear);

void printLevelOrder(node *root);
void enQueue(struct node **queue, int *rear, struct node *new_node);
node *deQueue(struct node **queue, int *front);

int main()
{
    node *root;
    root = create('A');
    root->left = create('B');
    root->right = create('C');
    (root->left)->left = create('D');
    (root->left)->right = create('E');
    (root->right)->left = create('F');
    (root->right)->right = create('G');

    int ch;
    while (ch != 5)
    {

        printf("\n1.Pre-order traversal\n2.In-order traversal\n3.Post-order
traversal\n4.Level-order traversal\n5.Exit ");
        printf("\nEnter the choice ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printf("Pre-order traversal of a tree is : ");
            preorder(root);
            break;
        case 2:
            printf("\n In-order traversal of a tree is : ");
            inorder(root);
            break;
        case 3:
            printf("\n Post-order traversal of a tree is : ");
            postorder(root);
            break;
        case 4:
            printf("\n Level-order traversal of a tree is : ");
            printLevelOrder(root);
            break;
        case 5:
            exit(0);
        default:
            printf("Enter the correct choice ");
        }
    }
    return 1;
}

node *create(char value)
{
    node *root = (node *)malloc(sizeof(node));
    root->data = value;
    root->left = NULL;
    root->right = NULL;
    return (root);
}

void preorder(node *root)
{
    stack *s = NULL;
    int flag = 1;
```

```c
    while (flag)
    {
        if (root)
        {
            printf("%c ", root->data);
            push(&s, root);
            root = root->left;
        }
        else
        {
            if (!emptystack(s))
            {
                root = pop(&s);
                root = root->right;
            }
            else
            {
                flag = 0;
            }
        }
    }
}

int emptystack(stack *top)
{
    if (top == NULL)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

node *top(stack *top)
{
    return top->DATA;
}
void push(stack **top, node *root)
{
    stack *new_top = (stack *)malloc(sizeof(stack));
    new_top->DATA = root;
    new_top->next = (*top);
    (*top) = new_top;
}

node *pop(stack **top)
{
    stack *new_top;
    node *root;
    root = (*top)->DATA;
    new_top = (*top);

    (*top) = (*top)->next;
    free(new_top);
    return (root);
}

void inorder(node *root)
{
    int flag = 1;
    stack *s = NULL;

    while (flag)
    {
        if (root)
        {
            push(&s, root);
            root = root->left;
        }
        else
        {
            if (!emptystack(s))
            {
                root = pop(&s);
                printf("%c ", root->data);
                root = root->right;
            }
            else
            {
                flag = 0;
            }
        }
    }
}

void postorder(node *root)
{
    stack *s = NULL;
    node *previous = NULL;

    do
    {
        while (root != NULL)
        {
            push(&s, root);
            root = root->left;
        }
        while (root == NULL && !emptystack(s))
        {
            root = top(s);
            if (root->right == NULL || root->right == previous)
            {
                printf("%c ", root->data);
                previous = pop(&s);
                root = NULL;
            }
            else
            {
                root = root->right;
            }
        }
    } while (!emptystack(s));
}

void printLevelOrder(node *root)
{
    int rear, front;
    struct node **queue = createQueue(&front, &rear);
    struct node *temp_node = root;

    while (temp_node)
    {
        printf("%c ", temp_node->data);

        if (temp_node->left)
            enQueue(queue, &rear, temp_node->left);

        if (temp_node->right)
            enQueue(queue, &rear, temp_node->right);

        temp_node = deQueue(queue, &front);
    }
}

node **createQueue(int *front, int *rear)
{
    node **queue = (node **)malloc(
        sizeof(node *) * MAX_Q_SIZE);

    *front = *rear = 0;
    return queue;
}
void enQueue(struct node **queue, int *rear,
             struct node *new_node)
{
    queue[*rear] = new_node;
    (*rear)++;
}

node *deQueue(struct node **queue, int *front)
```

```
{
    (*front)++;
    return queue[*front - 1];
}
```

OUTPUT :

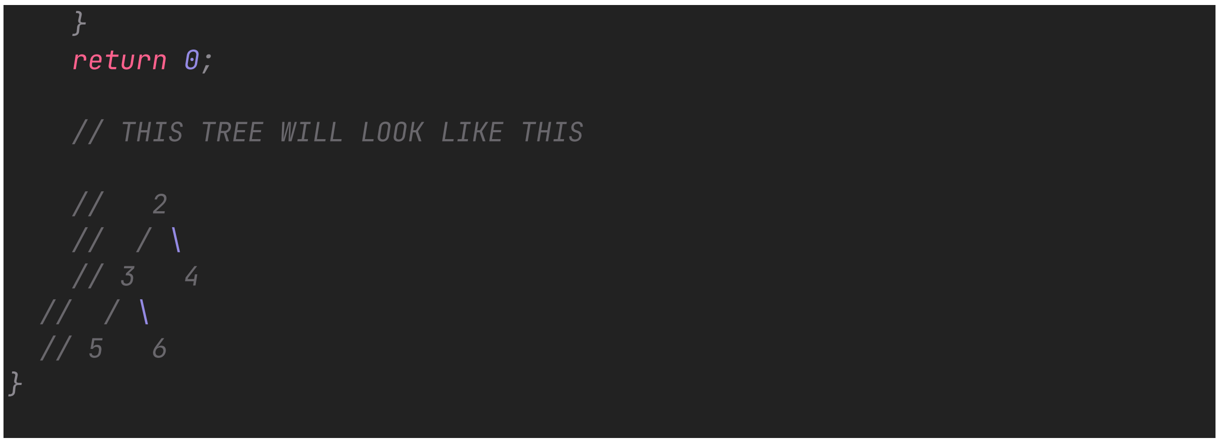


```
PS E:\DS_ALGO> cd "e:\DS_ALGO\" ; if ($?) { gcc 24_Preorder_traversal
traversal }

1.Pre-order traversal
2.In-order traversal
3.Post-order traversal
4.Level-order traversal
5.Exit
Enter the choice 1
Pre-order traversal of a tree is : A B D E C F G
1.Pre-order traversal
2.In-order traversal
3.Post-order traversal
4.Level-order traversal
5.Exit
Enter the choice 2

 In-order traversal of a tree is : D B E A F C G
1.Pre-order traversal
2.In-order traversal
3.Post-order traversal
4.Level-order traversal
5.Exit
Enter the choice 3

 Post-order traversal of a tree is : D E B F G C A
1.Pre-order traversal
2.In-order traversal
3.Post-order traversal
4.Level-order traversal
5.Exit
Enter the choice 4

 Level-order traversal of a tree is : A B C D E F G
1.Pre-order traversal
2.In-order traversal
3.Post-order traversal
4.Level-order traversal
5.Exit
Enter the choice 5
PS E:\DS_ALGO> 
```

Shrihari K Mangle
CSB2-74   12010517

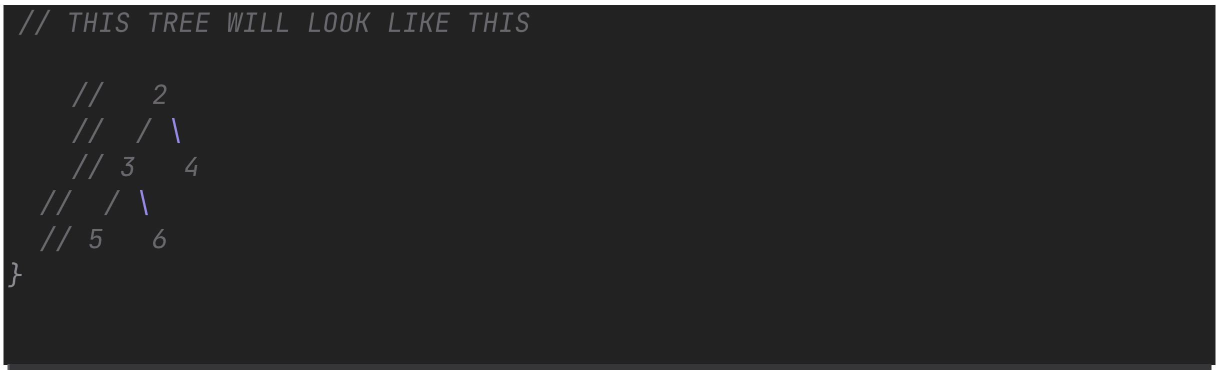
RECURSIVE CODE:

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

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

struct node* createNode(int data)
{
    struct node*p;
    p = (struct node *)malloc(sizeof(struct node));
    p->data = data;
    p->left = NULL;
    p->right = NULL;

    return p;
}

void preOrder(struct node *root)
{
    if (root!=NULL)
    {
        printf("%d ", root->data);
        preOrder(root->left);
        preOrder(root->right);
    }

}

void postOrder(struct node *root)
{
    if (root!=NULL)
    {
        postOrder(root->left);
        postOrder(root->right);
        printf("%d ", root->data);
    }

}

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

}


int main()
{
    struct node * p1 = createNode(2);
    struct node * p2 = createNode(3);
    struct node * p3 = createNode(4);
    struct node * p4 = createNode(5);
    struct node * p5 = createNode(6);

    p1->left = p2;
    p1->right = p3;
    p2->left = p4;
    p2->right = p5;

    int no, choice;

    while (1)
    {
        printf("\n1.PREORDER\n2.INORDER\n3.POSTORDER\n4.EXIT\n");
        printf("Enter choice : ");
        scanf("%d", &choice);
        printf("\n");

        switch (choice)
        {
        case 1:
            printf("PREORDER TRAVERSAL IS : ");
            preOrder(p1);
            break;

        case 2:

            printf("INORDER TRAVERSAL IS : ");
            inOrder(p1);
            break;

        case 3:
            printf("POSTORDER TRAVERSAL IS : ");
            postOrder(p1);
            break;

        case 4:
            exit(0);

        default:
            break;
        }
```

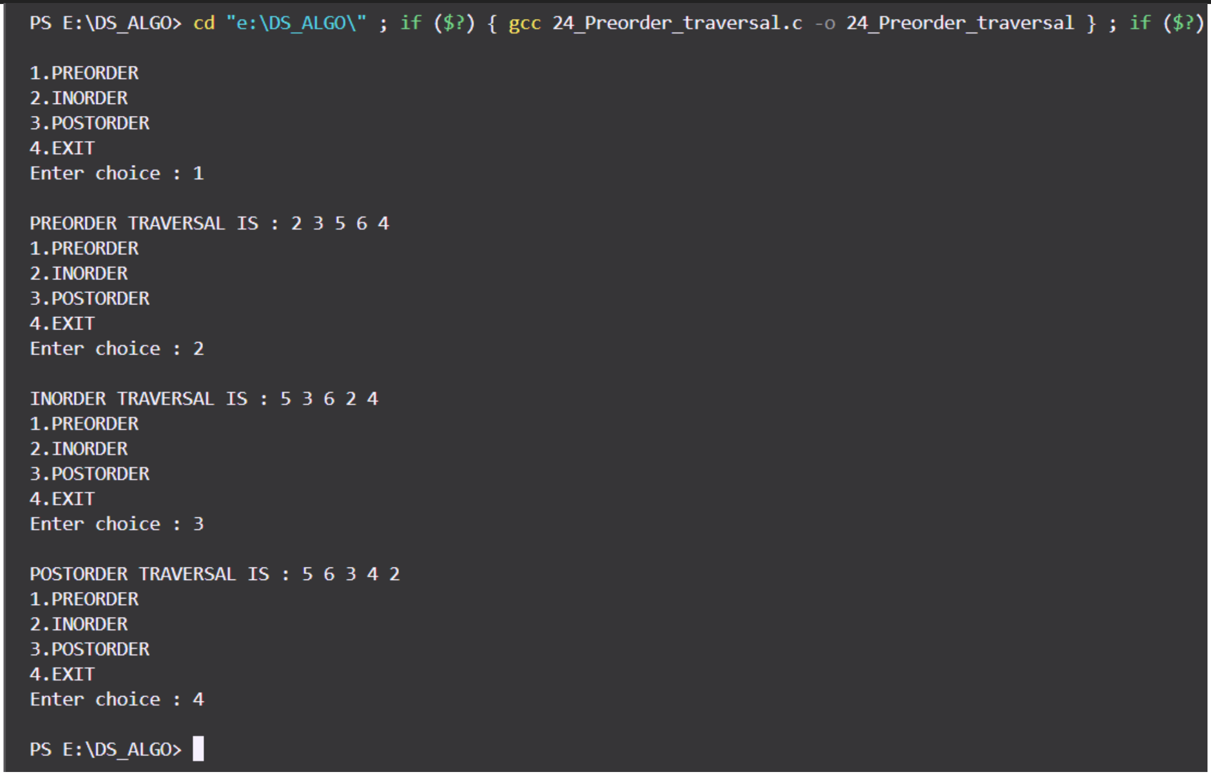


```
    }
    return 0;

    // THIS TREE WILL LOOK LIKE THIS

    //    2
    //   / \
    // 3    4
  //   / \
  // 5    6
}
```

OUTPUT :



```
// THIS TREE WILL LOOK LIKE THIS

    //    2
    //   / \
    // 3    4
  //   / \
  // 5    6
}
```



```
PS E:\DS_ALGO> cd "e:\DS_ALGO\" ; if ($?) { gcc 24_Preorder_traversal.c -o 24_Preorder_traversal } ; if ($?)

1.PREORDER
2.INORDER
3.POSTORDER
4.EXIT
Enter choice : 1

PREORDER TRAVERSAL IS : 2 3 5 6 4
1.PREORDER
2.INORDER
3.POSTORDER
4.EXIT
Enter choice : 2

INORDER TRAVERSAL IS : 5 3 6 2 4
1.PREORDER
2.INORDER
3.POSTORDER
4.EXIT
Enter choice : 3

POSTORDER TRAVERSAL IS : 5 6 3 4 2
1.PREORDER
2.INORDER
3.POSTORDER
4.EXIT
Enter choice : 4

PS E:\DS_ALGO> █
```