VISHWAKARMA INSTITUTE OF TECHNOLOGY

NAME	Arpit Sudhir Vidhale
ROLL NO.	60
DIVISION	CS-D
BATCH	B3
PRN NO.	12111229

DS LAB ASSIGNMENT 5

Question:

Tree traversal without recursion.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 50

struct node
{
    struct node *lchild;
    int info;
    struct node *rchild;
};

struct node *insert_nrec(struct node *root, int ikey);
void nrec_pre(struct node *root);
void nrec_in(struct node *root);
void nrec_post(struct node *root);
void display(struct node *ptr, int level);

struct node *queue[MAX];
int front = -1, rear = -1;
void insert_queue(struct node *item);
struct node *del_queue();
int queue_empty();
```

```
struct node *stack[MAX];
int top = -1;
void push_stack(struct node *item);
struct node *pop_stack();
int stack_empty();

int main()
{
    struct node *root = NULL, *ptr;
    int choice, k;

    while (1)
    {
        printf("\n");
        printf("1.Insert\n");
        printf("2.Display\n");
        printf("3.Preorder Traversal\n");
        printf("4.Inorder Traversal\n");
        printf("5.Postorder Traversal\n");
        printf("6.Quit\n");
        printf("\nEnter your choice : ");
        scanf("%d", &choice);

        switch (choice)
        {

        case 1:
            printf("\nEnter the key to be inserted :
            "); scanf("%d", &k);
            root = insert_nrec(root, k);
            break;

        case 2:
            printf("\n");
            display(root, 0);
            printf("\n");
            break;

        case 3:
            printf("\n Preorder Traversal ->");
            nrec_pre(root);
```

```
            break;

        case 4:
            printf("\n Inorder Traversal ->");
            nrec_in(root);
            break;
        case 5:
            printf("\n Postorder Traversal ->");
            nrec_post(root);
            break;

        case 6:
            exit(1);

        default:
            printf("\nWrong choice\n");
        }
    }

    return 0;

}
struct node *insert_nrec(struct node *root, int
ikey) {
    struct node *tmp, *par, *ptr;

    ptr = root;
    par = NULL;

    while (ptr != NULL)
    {
        par = ptr;
        if (ikey < ptr->info)
            ptr = ptr->lchild;
        else if (ikey > ptr->info)
            ptr = ptr->rchild;
        else
        {
            printf("\nDuplicate key");
            return root;
        }
    }
```

```
    tmp = (struct node *)malloc(sizeof(struct
    node)); tmp->info = ikey;
    tmp->lchild = NULL;
    tmp->rchild = NULL;

    if (par == NULL)
        root = tmp;
    else if (ikey < par->info)
        par->lchild = tmp;
    else
        par->rchild = tmp;

    return root;
}

void nrec_pre(struct node *root)
{
    struct node *ptr = root;
    if (ptr == NULL)
    {
        printf("Tree is
        empty\n"); return;
    }
    push_stack(ptr);
    while (!stack_empty())
    {
        ptr = pop_stack();
        printf("%d ", ptr->info);
        if (ptr->rchild != NULL)
            push_stack(ptr->rchild);
        if (ptr->lchild != NULL)
            push_stack(ptr->lchild);
    }
    printf("\n");
}

void nrec_in(struct node *root)
{
    struct node *ptr = root;

    if (ptr == NULL)
```

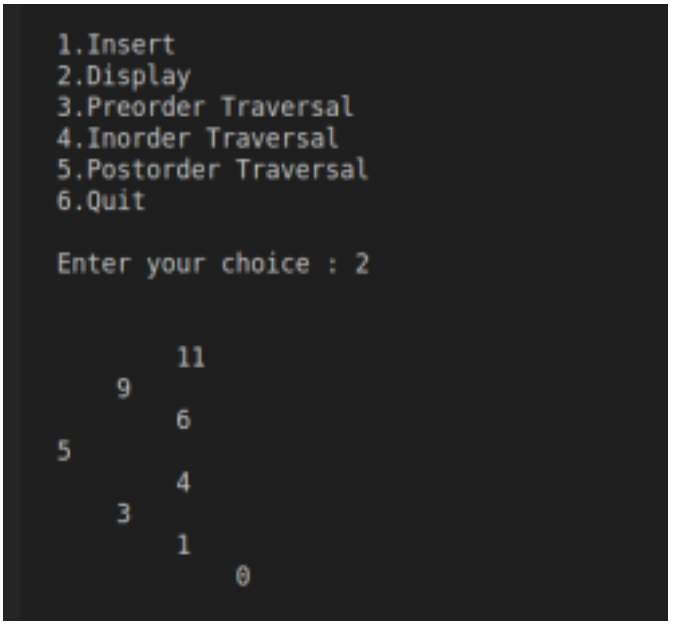
```
    {
        printf("Tree is
        empty\n"); return;
    }
    while (1)
    {
        while (ptr->lchild != NULL)
        {
            push_stack(ptr);
            ptr = ptr->lchild;
        }

        while (ptr->rchild == NULL)
        {
            printf("%d ", ptr->info);
            if (stack_empty())
                return;
            ptr = pop_stack();
        }
        printf("%d ", ptr->info);
        ptr = ptr->rchild;
    }
    printf("\n");
}

void nrec_post(struct node *root)
{
    struct node *ptr = root;
    struct node *q;

    if (ptr == NULL)
    {
        printf("Tree is empty\n");
        return;
    }
    q = root;
    while (1)
    {
        while (ptr->lchild != NULL)
        {
            push_stack(ptr);
            ptr = ptr->lchild;
```

```
        }

        while (ptr->rchild == NULL || ptr->rchild == q)
        {
            printf("%d ", ptr->info);
            q = ptr;
            if (stack_empty())
                return;
            ptr = pop_stack();
        }
        push_stack(ptr);
        ptr = ptr->rchild;
    }
    printf("\n");
}

void insert_queue(struct node *item)
{
    if (rear == MAX - 1)
    {
        printf("Queue Overflow\n");
        return;
    }
    if (front == -1)
        front = 0;
    rear = rear + 1;
    queue[rear] = item;
}

struct node *del_queue()
{
    struct node *item;
    if (front == -1 || front == rear +
    1) {
        printf("Queue
        Underflow\n"); return 0;
    }
    item = queue[front];
    front = front + 1;
    return item;
}
```

```
int queue_empty()
{
    if (front == -1 || front == rear +
        1) return 1;
    else
        return 0;
}


void push_stack(struct node *item)
{
    if (top == (MAX - 1))
    {
        printf("Stack Overflow\n");
        return;
    }
    top = top + 1;
    stack[top] = item;
}

struct node *pop_stack()
{
    struct node *item;
    if (top == -1)
    {
        printf("Stack Underflow....\n");
        exit(1);
    }
    item = stack[top];
    top = top - 1;
    return item;
}

int stack_empty()
{
    if (top == -1)
        return 1;
    else
        return 0;
}
```

```
void display(struct node *ptr, int level)
{
    int i;
    if (ptr == NULL)
        return;
    else
    {
        display(ptr->rchild, level + 1);
        printf("\n");
        for (i = 0; i < level; i++)
            printf(" ");
        printf("%d", ptr->info);
        display(ptr->lchild, level + 1);
    }
}
```

Output:



```
1.Insert
2.Display
3.Preorder Traversal
4.Inorder Traversal
5.Postorder Traversal
6.Quit

Enter your choice : 3

 Preorder Traversal ->5  3  1  0  4  9  6  11

1.Insert
2.Display
3.Preorder Traversal
4.Inorder Traversal
5.Postorder Traversal
6.Quit

Enter your choice : 4

 Inorder Traversal ->0  1  3  4  5  6  9  11
1.Insert
2.Display
3.Preorder Traversal
4.Inorder Traversal
5.Postorder Traversal
6.Quit

Enter your choice : 5

 Postorder Traversal ->0  1  4  3  6  11  9  5
```

Question:

Tree traversal using recursion.

Code:

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

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

void
inorderTraversal (struct node
*root) {
  if (root == NULL)
    return;
  inorderTraversal (root->left);
```

```
  printf ("%d ", root->item);
  inorderTraversal
(root->right); }

void
preorderTraversal (struct node *root)
{
  if (root == NULL)
    return;
  printf ("%d ", root->item);
  preorderTraversal (root->left);
preorderTraversal (root->right);
}

void
postorderTraversal (struct node
*root) {
  if (root == NULL)
    return;
  postorderTraversal (root->left);
  postorderTraversal (root->right);
  printf ("%d ", root->item);
}

struct node *createNode (value)
{
  struct node *newNode = malloc (sizeof (struct
  node)); newNode->item = value;
  newNode->left = NULL;
  newNode->right = NULL;

  return newNode;
}


struct node *insertLeft (struct node *root, int value)
{
  root->left = createNode (value);
  return root->left;
```

```
}


struct node *insertRight (struct node *root, int value)
{
  root->right = createNode (value);
  return root->right;
}

int
main ()
{
  struct node *root = createNode (1);
  insertLeft (root, 12);
  insertRight (root, 9);

  insertLeft (root->left, 5);
  insertRight (root->left, 6);

  printf ("Inorder traversal ->");
  inorderTraversal (root);

  printf ("\n\nPreorder traversal ->");
  preorderTraversal (root);

  printf ("\n\nPostorder traversal ->");
  postorderTraversal (root);
}
```

Output:

```
Inorder traversal ->5 12 6 1 9

Preorder traversal ->1 12 5 6 9

Postorder traversal ->5 6 12 9 1
```