WEEK-11

Date:24-09-2025

List of programs:

1. Write a C program to implement Circular Queue operations using arrays.
2. Write a C program to implement Recursive Binary Tree Traversals(In-Order, Pre-Order, Post-Order).

1.Aim: Write a C program to implement Circular Queue operations using arrays.

Program:

```
#include<stdio.h>
#include<stdlib.h>
#define size 3
int Queue[size];
int front=-1,rear=-1;
void enqueue(int x)
{
   if(front==(rear+1)%size)
   {
      printf("circular queue is full");
   }
   else
   {
      if(front==-1&&rear==-1)
      {
         front=rear=0;
      }
      else
      {
```

```
            rear=(rear+1)%size;
        }
        Queue[rear]=x;
    }
}
void dequeue()
{
    if(front==-1)
    {
        printf("queue is empty");
    }
    else
    {
        printf("deleted element is %d",Queue[front]);
        if(front==rear)
        {
            front=rear=-1;
        }
        else
        {
            front=(front+1)%size;
        }
    }
}
void display()
{
    int i;
    if(front==-1)
```

```
    {
        printf("queue is empty");
    }
    else
    {
        for(i=front;i!=rear;i=(i+1)%size)
        {
            printf("%d\n",Queue[i]);
        }
        printf("%d\n",Queue[i]);
    }
}

void main()
{
int ch,num;
printf("\n:: circular queue using arrays ::\n");
while(1)
{
printf("\nMAIN MENU:\n1.enqueue\n2.dequeue\n3.DISPLAY\n4.EXIT\n\nENTER YOUR
CHOICE:");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("ENTER THE QUEUE ELEMENT : ");
          scanf("%d",&num);
          enqueue(num);
          break;
```

```
case 2: dequeue();
        break;
case 3: display();
        break;
case 4:exit(0);
        break;
default:printf("Invalid Choice : ");
}
}
}
```

Output:

```
:: circular queue using arrays ::

MAIN MENU:
1.enqueue
2.dequeue
3.DISPLAY
4.EXIT

ENTER YOUR CHOICE:1
ENTER THE QUEUE ELEMENT : 5

MAIN MENU:
1.enqueue
2.dequeue
3.DISPLAY
4.EXIT

ENTER YOUR CHOICE:1
ENTER THE QUEUE ELEMENT : 10

MAIN MENU:
1.enqueue
2.dequeue
3.DISPLAY
4.EXIT
```

```
ENTER YOUR CHOICE:1
ENTER THE QUEUE ELEMENT : 15

MAIN MENU:
1.enqueue
2.dequeue
3.DISPLAY
4.EXIT

ENTER YOUR CHOICE:3
5
10
15

MAIN MENU:
1.enqueue
2.dequeue
3.DISPLAY
4.EXIT

ENTER YOUR CHOICE:2
deleted element is 5
MAIN MENU:
1.enqueue
2.dequeue
```

```
ENTER YOUR CHOICE:4


=== Code Execution Successful ===
```

2. Aim: To Write a C program to implement Recursive Binary Tree Traversals(In-Order, Pre-Order, Post-Order).

Program:

```
#include <stdio.h>
#include <stdlib.h>
    struct Node
{
    int data;
    struct Node* left;
    struct Node* right;
} ;
Struct node* createNode(int data)
 {
    Struct node* newNode = (struct node*)malloc(sizeof(struct node));
    if (newNode == NULL)
 {
       printf("Memory allocation failed!\n");
       exit(1);
    }
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
Struct node* insert(struct node* root, int data)
 {
    if (root == NULL)
    {
```

```
        return createNode(data);
    }
    if (data < root->data)
    {
        root->left = insert(root->left, data);
    }
    else if (data > root->data)
    {
        root->right = insert(root->right, data);
    }
else
    return root;
}
void inorderTraversal(struct node* root)
{
    if (root != NULL)
{
        inorderTraversal(root->left);
        printf("%d ", root->data);
        inorderTraversal(root->right);
    }
}
void preorderTraversal(struct node* root)
{
    if (root != NULL) {
        printf("%d ", root->data);
        preorderTraversal(root->left);
        preorderTraversal(root->right);
```

```
    }
}
void postorderTraversal(struct node* root)
{
    if (root != NULL)
{
        postorderTraversal(root->left);
        postorderTraversal(root->right);
        printf("%d ", root->data);
    }
}
int main()
{
    Struct node* root = NULL;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 70);
    insert(root, 20);
    insert(root, 40);
    insert(root, 60);
    insert(root, 80);
    printf("In-order traversal: ");
    inorderTraversal(root);
    printf("\n");
    printf("Pre-order traversal: ");
    preorderTraversal(root);
    printf("\n");
    printf("Post-order traversal: ");
```

```
    postorderTraversal(root);
    printf("\n");
    return 0;
}
```

Output:

```
In-order traversal: 20 30 40 50 60 70 80
Pre-order traversal: 50 30 20 40 70 60 80
Post-order traversal: 20 40 30 60 80 70 50


=== Code Execution Successful ===
```

Inferences:

- The main advantage of circular queues using arrays is **no wasted space** → unlike simple linear array queue where freed positions at the start cannot be reused and it has efficient memory utilization.
- The **recursive approach** simplifies both the **tree creation** and the **tree traversal** processes, making the code clean and easier to understand.
- The **in-order traversal (Left → Root → Right)** displays nodes in **sorted order** only if the binary tree satisfies the **Binary Search Tree (BST)** property.
- The **pre-order traversal (Root → Left → Right)** is useful for **copying or saving** the tree structure
- The **post-order traversal (Left → Right → Root)** is useful for **deleting** the entire tree or evaluating **expression trees**.
- Recursion ensures that every subtree is processed independently, following the same traversal logic — demonstrating **divide-and-conquer** principle.