Q. 1. Create binary tree and perform recursive traversals.

/\*Create binary tree and perform recursive traversals\*/

#include <iostream>

using namespace std;

// Structure for a node of a binary tree

struct Node {

int data;

Node\* left;

Node\* right;

};

// Function to create a new node and return its address

Node\* getNewNode(int data) {

Node\* newNode = new Node();

newNode->data = data;

newNode->left = newNode->right = NULL;

return newNode;

}

// Recursive function to do pre-order traversal of the binary tree

void preOrder(Node\* root) {

if (root == NULL) return;

cout << root->data << " ";

preOrder(root->left);

preOrder(root->right);

}

// Recursive function to do in-order traversal of the binary tree

void inOrder(Node\* root) {

if (root == NULL) return;

inOrder(root->left);

cout << root->data << " ";

inOrder(root->right);

}

// Recursive function to do post-order traversal of the binary tree

void postOrder(Node\* root) {

if (root == NULL) return;

postOrder(root->left);

postOrder(root->right);

cout << root->data << " ";

}

int main() {

Node\* root = getNewNode(1);

root->left = getNewNode(2);

root->right = getNewNode(3);

root->left->left = getNewNode(4);

root->left->right = getNewNode(5);

cout << "Pre-order traversal: ";

preOrder(root);

cout << endl;

cout << "In-order traversal: ";

inOrder(root);

cout << endl;

cout << "Post-order traversal: ";

postOrder(root);

cout << endl;

return 0;

}

Q. 2. Implement circular queue using arrays.

/\*Implement circular queue using arrays. \*/

#include <iostream>

using namespace std;

class CircularQueue {

int \*queue, size, front, rear;

public:

CircularQueue(int s) {

size = s;

queue = new int[size];

front = rear = -1;

}

void enqueue(int x);

int dequeue();

void display();

};

void CircularQueue::enqueue(int x) {

if ((front == 0 && rear == size - 1) || (front == rear + 1)) {

cout << "Queue is full\n";

return;

}

else if (front == -1) {

front = rear = 0;

}

else if (rear == size - 1 && front != 0) {

rear = 0;

}

else {

rear++;

}

queue[rear] = x;

}

int CircularQueue::dequeue() {

if (front == -1) {

cout << "Queue is empty\n";

return -1;

}

int x = queue[front];

if (front == rear) {

front = rear = -1;

}

else if (front == size - 1) {

front = 0;

}

else {

front++;

}

return x;

}

void CircularQueue::display() {

if (front == -1) {

cout << "Queue is empty\n";

return;

}

if (rear >= front) {

for (int i = front; i <= rear; i++)

cout << queue[i] << " ";

}

else {

for (int i = front; i < size; i++)

cout << queue[i] << " ";

for (int i = 0; i <= rear; i++)

cout << queue[i] << " ";

}

}

int main() {

CircularQueue q(5);

q.enqueue(1);

q.enqueue(2);

q.enqueue(3);

q.enqueue(4);

q.enqueue(5);

q.enqueue(6);

q.display();

cout << endl;

q.dequeue();

q.dequeue();

q.display();

cout << endl;

return 0;

}