Q. 1. 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;

}

Q. 2. Create binary tree. Find height of the tree and print leaf nodes. Find mirror image, print

original and mirror image using level-wise printing.

#include <bits/stdc++.h>

using namespace std;

/\* A binary tree node has data, pointer

to left child and a pointer to right child \*/

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* newNode(int data)

{

struct Node\* node

= (struct Node\*)malloc(sizeof(struct Node));

node->data = data;

node->left = NULL;

node->right = NULL;

return (node);

}

void mirror(struct Node\* node)

{

if (node == NULL)

return;

else {

struct Node\* temp;

/\* do the subtrees \*/

mirror(node->left);

mirror(node->right);

/\* swap the pointers in this node \*/

temp = node->left;

node->left = node->right;

node->right = temp;

}

}

/\* Helper function to print

Inorder traversal.\*/

void inOrder(struct Node\* node)

{

if (node == NULL)

return;

inOrder(node->left);

cout << node->data << " ";

inOrder(node->right);

}

// Driver Code

int main()

{

struct Node\* root = newNode(1);

root->left = newNode(2);

root->right = newNode(3);

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

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

/\* Print inorder traversal of the input tree \*/

cout << "Inorder traversal of the constructed"

<< " tree is" << endl;

inOrder(root);

/\* Convert tree to its mirror \*/

mirror(root);

/\* Print inorder traversal of the mirror tree \*/

cout << "\nInorder traversal of the mirror tree"

<< " is \n";

inOrder(root);

return 0;

}