**Fatima Nazir 45317**

**CIRCULAR LINK LIST QUEUE**

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int val) {

data = val;

next = NULL;

}

};

class Queue {

private:

Node\* front;

Node\* rear;

public:

Queue() {

front = rear = NULL;

}

bool isEmpty() {

return front == NULL;

}

bool isFull() {

return false;

}

void enqueue(int val) {

Node\* newNode = new Node(val);

if (isEmpty()) {

front = rear = newNode;

rear->next = front; // Point rear to front to create circularity

} else {

rear->next = newNode;

rear = newNode;

rear->next = front; // Maintain circular link

}

cout << val << " enqueued to the queue." << endl;

}

void dequeue() {

if (isEmpty()) {

cout << "Queue is empty." << endl;

return;

}

Node\* temp = front;

if (front == rear) { // Only one element in the queue

front = rear = NULL;

} else {

front = front->next;

rear->next = front; // Maintain circular link

}

cout << temp->data << " dequeued from the queue." << endl;

delete temp;

}

void display() {

if (isEmpty()) {

cout << "Queue is empty." << endl;

return;

}

Node\* temp = front;

cout << "Queue elements: ";

do {

cout << temp->data << " ";

temp = temp->next;

} while (temp != front);

cout << endl;

}

};

int main() {

Queue q;

q.enqueue(10);

q.enqueue(20);

q.enqueue(30);

q.display();

q.dequeue();

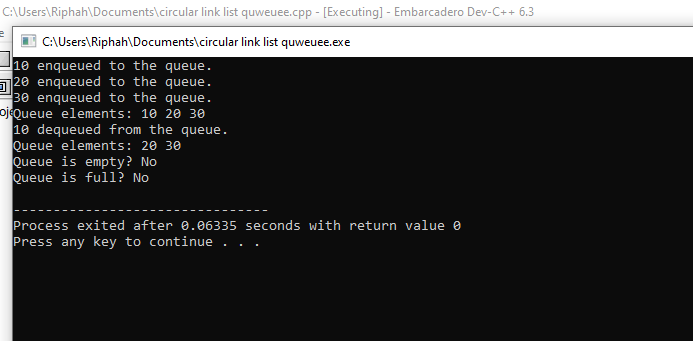
q.display();

cout << "Queue is empty? " << (q.isEmpty() ? "Yes" : "No") << endl;

cout << "Queue is full? " << (q.isFull() ? "Yes" : "No") << endl;

return 0;

}



**CIRCULAR QUEUE ARRAY**

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int val) {

data = val;

next = NULL;

}

};

class Queue {

private:

Node\* front;

Node\* rear;

public:

Queue() {

front = rear = NULL;

}

bool isEmpty() {

return front == NULL;

}

bool isFull() {

return false;

}

void enqueue(int val) {

Node\* newNode = new Node(val);

if (isEmpty()) {

front = rear = newNode;

rear->next = front; // Point rear to front to create circularity

} else {

rear->next = newNode;

rear = newNode;

rear->next = front; // Maintain circular link

}

cout << val << " enqueued to the queue." << endl;

}

void dequeue() {

if (isEmpty()) {

cout << "Queue is empty." << endl;

return;

}

Node\* temp = front;

if (front == rear) { // Only one element in the queue

front = rear = NULL;

} else {

front = front->next;

rear->next = front; // Maintain circular link

}

cout << temp->data << " dequeued from the queue." << endl;

delete temp;

}

void display() {

if (isEmpty()) {

cout << "Queue is empty." << endl;

return;

}

Node\* temp = front;

cout << "Queue elements: ";

do {

cout << temp->data << " ";

temp = temp->next;

} while (temp != front);

cout << endl;

}

};

int main() {

Queue q;

q.enqueue(10);

q.enqueue(20);

q.enqueue(30);

q.display();

q.dequeue();

q.display();

cout << "Queue is empty? " << (q.isEmpty() ? "Yes" : "No") << endl;

cout << "Queue is full? " << (q.isFull() ? "Yes" : "No") << endl;

return 0;

}