**Waleed Afzal  
23p-0566**

**3D  
Data Structure Lab**

**Queue (Array + LinkedList)**

***Task1:***#include<iostream>

#include<string>

using namespace std;

class Queue {

int\* array;

int queueSize;

int front;

int rear;

int count;

public:

Queue(int s) {

array = new int[s];

queueSize = s;

front = -1;

rear = -1;

count = 0;

}

bool isEmpty() {

if (front == -1) {

return true;

}

else {

return false;

}

}

bool isFull() {

if (rear == queueSize - 1) {

return true;

}

else {

return false;

}

}

void enqueue(int num) {

if (isFull()) {

cout << "OVERFLOW!!" << endl;

return;

}

if (isEmpty()) {

rear = front = 0;

}

else {

rear++;

}

array[rear] = num;

cout << "Data Enqueued: " << num << endl;

count++;

}

int dequeue() {

int num = 0;

if (isEmpty()) {

cout << "UNDERFLOW!!" << endl;

return -1;

}

num = array[front];

if (front == rear) {

// num = array[front];

front = rear = -1;

}

else {

front++;

}

cout << "Data Dequeued: " << num << endl;

count--;

return num;

}

void display() {

front = 0;

cout << "Queue Data List: " << endl;

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

cout << array[i] << endl;

}

cout << endl;

}

};

int main() {

Queue Q1(10);

Q1.enqueue(2);

Q1.enqueue(4);

Q1.enqueue(6);

Q1.enqueue(8);

Q1.enqueue(10);

Q1.dequeue();

// cout << num << endl;

Q1.enqueue(44);

Q1.enqueue(100);

Q1.display();

system("pause");

return 0;

}

***Output:  
A screenshot of a computer

Description automatically generated***

***Task2:***

#include<iostream>

#include<string>

using namespace std;

class CircularQueue {

int\* array;

int queueSize;

int front;

int rear;

int count;

public:

CircularQueue(int s) {

array = new int[s];

queueSize = s;

front = -1;

rear = -1;

count = 0;

}

bool isEmpty() {

if (rear == -1 && front == -1) {

return true;

}

else {

return false;

}

}

bool isFull() {

if ((rear + 1) % queueSize == front) {

return true;

}

else {

return false;

}

}

void enqueue(int num) {

if (isFull()) {

cout << "OVERFLOW!!" << endl;

return;

}

else if (isEmpty()) {

rear = front = 0;

}

else {

rear = (rear + 1) % queueSize;

}

cout << "Data Enqueued: " << num << endl;

count++;

array[rear] = num;

}

int dequeue() {

int num = 0;

if (isEmpty()) {

cout << "UNDERFLOW!!" << endl;

return -1;

}

num = array[front];

array[front] = -1;

if (front == rear) {

front = rear = -1;

}

else {

front = (front + 1) % queueSize;

}

cout << "Data Dequeued: " << num << endl;

count--;

return num;

}

void display() {

cout << "Displaying Circular Queue (-1 Shows the Index is Empty):\n";

for (int i = 0;i <= count;i++) {

cout << array[i] << " ";

}

cout << endl;

}

};

int main() {

CircularQueue Q1(10);

Q1.enqueue(1);

Q1.enqueue(2);

Q1.enqueue(3);

Q1.enqueue(5);

Q1.enqueue(10);

Q1.dequeue();

Q1.enqueue(24);

Q1.dequeue();

Q1.enqueue(60);

Q1.display();

system("pause");

return 0;

}

***Output:***A screenshot of a computer

Description automatically generated

***Task3:***#include<iostream>

#include<string>

using namespace std;

class CircularQueue {

int\* array;

int queueSize;

int front;

int rear;

int count;

int poppedCount;

public:

CircularQueue(int s) {

array = new int[s];

queueSize = s;

front = -1;

rear = -1;

count = 0;

poppedCount = 0;

}

bool isEmpty() {

if (rear == -1 && front == -1) {

return true;

}

else {

return false;

}

}

bool isFull() {

if ((rear + 1) % queueSize == front) {

return true;

}

else {

return false;

}

}

void enqueue(int num) {

if (isFull()) {

cout << "OVERFLOW!!" << endl;

return;

}

else if (isEmpty()) {

rear = front = 0;

}

else {

rear = (rear + 1) % queueSize;

}

cout << "Data Enqueued: " << num << endl;

count++;

array[rear] = num;

}

int dequeue() {

int num = 0;

if (isEmpty()) {

cout << "UNDERFLOW!!" << endl;

return -1;

}

num = array[front];

array[front] = -1;

if (front == rear) {

front = rear = -1;

}

else {

front = (front + 1) % queueSize;

}

cout << "Data Dequeued: " << num << endl;

poppedCount++;

count--;

return num;

}

int getSize() {

// cout<<"Queue Size is: "<<count<<endl;

return count;

}

//using this function in main()

bool reverse() {

if (isEmpty()) {

cout << "Queue is empty, nothing to reverse." << endl;

return false;

}

return reverseQueue(front, rear);

}

bool reverseQueue(int start, int end) {

// Base case: stop when start meets or crosses end

if (start == end || (end + 1) % queueSize == start) {

return true;

}

// Swap elements at start and end

int temp = array[start];

array[start] = array[end];

array[end] = temp;

// Move start forward and end backward

start = (start + 1) % queueSize;

end = (end - 1 + queueSize) % queueSize;

// Recursive call with updated indices

return reverseQueue(start, end);

}

void display() {

cout << "Displaying Circular Queue (-1 Shows the Index is Empty):\n";

for (int i = 0;i <= count + 1;i++) {

cout << array[i] << " ";

}

cout << endl;

}

};

int main() {

CircularQueue Q1(10);

Q1.enqueue(1);

Q1.enqueue(2);

Q1.enqueue(2);

Q1.enqueue(3);

Q1.enqueue(5);

Q1.enqueue(10);

Q1.dequeue();

Q1.enqueue(24);

Q1.dequeue();

Q1.enqueue(60);

Q1.display();

int size = Q1.getSize();

if (Q1.reverse()) {

cout << "Queue is Reversed: " << endl;

Q1.display();

}

Q1.getSize();

system("pause");

return 0;

}

***Output:***