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. Write a menu driven program to perform following operations on singly linked list: Create,

reverse, search, count and Display

/\*Write a menu driven program to perform following

operations on singly linked list: Create, reverse,

search, count and Display.

\*/

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

class LinkedList {

private:

Node\* head;

int count;

public:

LinkedList() {

head = NULL;

count = 0;

}

void create() {

int data;

cout << "Enter the data for the node: ";

cin >> data;

Node\* newNode = new Node();

newNode->data = data;

newNode->next = head;

head = newNode;

count++;

}

void reverse() {

Node\* prev = NULL;

Node\* current = head;

Node\* next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

head = prev;

}

int search(int key) {

Node\* current = head;

int index = 0;

while (current != NULL) {

if (current->data == key) {

return index;

}

current = current->next;

index++;

}

return -1;

}

int countNodes() {

return count;

}

void display() {

Node\* current = head;

while (current != NULL) {

cout << current->data << " ";

current = current->next;

}

cout << endl;

}

};

int main() {

int choice;

LinkedList list;

while (true) {

cout << "1. Create Node" << endl;

cout << "2. Reverse List" << endl;

cout << "3. Search Element" << endl;

cout << "4. Count Nodes" << endl;

cout << "5. Display List" << endl;

cout << "6. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

list.create();

break;

case 2:

list.reverse();

break;

case 3: {

int key;

cout << "Enter the element to be searched: ";

cin >> key;

int index = list.search(key);

if (index == -1) {

cout << "Element not found." << endl;

} else {

cout << "Element found at index: " << index << endl;

}

break;

}

case 4:

cout << "Number of nodes: " << list.countNodes() << endl;

break;

case 5:

list.display();

break;

case 6:

return 0;

default:

cout << "Invalid choice. Please enter a valid choice." << endl;

}

}

return 0;

}