

**Aim: Implementation of Circular Linked List ADT**

**Objective**: Circular Linked List can be used to manage the Computing Resources of the computer. Data Structure such as stacks and queue are implemented with the help of circular linked list.

**Theory :**

**Algorithm**

1. Initialize a circular linked list with a pointer to the head node.

2. Define a structure for the resource node containing information about each resource.

3. Create functions for adding, removing, searching, displaying, and managing resources.

4. InitializeCircularList():

- Initialize the head pointer as NULL.

5. AddResource():

- Create and add a new resource node to the circular linked list.

6. RemoveResource():

- Remove a specific resource node from the list.

7. SearchResource():

- Find a resource node by ID or other criteria.

8. DisplayResources():

- Display the list of available resources.

9. ManageResources():

- Implement allocation, deallocation, or modification operations.

10. Main Program:

- Initialize the circular linked list using InitializeCircularList().

- Display a menu for user actions (e.g., Add, Remove, Search, Display, Manage, Exit).

- Call corresponding functions based on user choices.

- Repeat the menu until the user exits the program.

11. Exit the program when the user chooses to exit.

12. Test the program with various resource management scenarios.

**Code :**

#include <stdio.h>

#include <conio.h>

#define SIZE 5

int items[SIZE];

int front = -1,rear = -1;

// Check if the queue is full

int isFull() {

if ((front == rear + 1) || (front == 0 && rear == SIZE - 1)) return 1;

return 0;

}

// Check if the queue is empty

int isEmpty() {

if (front == -1) return 1;

return 0;

}

// Adding an element

void enQueue(int element) {

if (isFull())

printf("\n Queue is full!!\n");

else {

if (front == -1) front = 0;

rear = (rear + 1) % SIZE;

items[rear] = element;

printf("\n Inserted ->%d",element);

}

}

// Removing an element

int deQueue() {

int element;

if (isEmpty()) {

printf("\n Queue is empty !! \n");

return (-1);

} else{

element = items[front];

if (front == rear) {

front = -1;

rear = -1;

}

// Q has only one element, so we reset the

// queue after dequeing it. ?

else {

front = (front + 1) % SIZE;

}

printf("\n Deleted element ->%d \n", element);

return (element);

}

}

// Display the queue

void display() {

int i;

if (isEmpty())

printf("\n Empty Queue\n");

else {

printf("\n Front ->%d", front);

printf("\n Items ->");

for (i = front; i != rear; i = (i + 1) % SIZE) {

printf("%d", items[i]);

}

printf("%d", items[i]);

printf("\n Rear ->%d \n", rear);

}

}

int main() {

// Fails because front = -1

deQueue();

enQueue(1);

enQueue(2);

enQueue(3);

enQueue(4);

enQueue(5);

// Fails to enqueue because front == 0 &amp;&amp; rear == SIZE - 1

enQueue(6);

display();

deQueue();

display();

enQueue(7);

display();

// Fails to enqueue because front == rear + 1

enQueue(8);

getch();

return 0;

}

**Output:**



**Conclusion:**Circular Linked Lists are ideal for efficiently managing computing resources and implementing data structures like stacks and queues. Their cyclic nature, constant-time operations, space efficiency, and adaptability make them a valuable choice for optimizing resource utilization and enhancing system efficiency in computing environments