**NAME** :- Manish Shashikant Jadhav

**UID** :-

**BRANCH** :- Comps -B. **BRANCH:** B.

**EXPERIMENT 2: Implement of given problem statement using Queue.**

**SUBJECT** :- DS (DATA STRUCTURES)

**TOPIC** **1** :- Implementation of Circular Queue using array.

**CODE** :-

/\*

 \* File: circular\_queue.c

 \* Author: Manish Jadhav

 \* Email: manishsj289@gmail.com

 \* Created: September 16, 2023

 \* Description: This program implements a Queue ADT with a circular array

 \*/

#include <stdio.h>

#include <stdlib.h>

struct Queue

{

    int front;

    int rear;

    int size;

    char \*array;

};

// 1 -> Initialize

struct Queue \*initialize\_queue(struct Queue \*queue, int size)

{

    queue->size = size;

    queue->front = queue->rear = 0;

    queue->array = (char \*)malloc(size \* sizeof(char));

}

// 2 -> isEmpty

int isEmpty(struct Queue \*queue)

{

    if (queue->rear == queue->front)

    {

        return 1;

    }

    return 0;

}

// 3 -> isFull

int isFull(struct Queue \*queue)

{

    if ((queue->rear + 1) % queue->size == queue->front)

    {

        return 1;

    }

    return 0;

}

// 4 -> enqueue

void enqueue(struct Queue \*queue, char item)

{

    if (isFull(queue))

    {

        printf("Queue Overflow\n");

    }

    else

    {

        queue->rear = (queue->rear + 1) % queue->size;

        queue->array[queue->rear] = item;

        printf("Enqueued element: %c\n", item);

    }

}

// 5 -> dequeue

char dequeue(struct Queue \*queue)

{

    char val = '\0';

    if (isEmpty(queue))

    {

        printf("Empty queue\n");

    }

    else

    {

        queue->front = (queue->front + 1) % queue->size;

        val = queue->array[queue->front];

    }

    return val;

}

// 6 -> front

char front(struct Queue\* queue){

    if (isEmpty(queue)) {

        printf("Queue is empty\n");

        return '\0';

    }

      printf("The front is : %c\n", queue->array[(queue->front + 1) % queue->size]);

}

// 7 -> rear

char rear(struct Queue\* queue){

    if (isEmpty(queue)) {

        printf("Queue is empty\n");

        return '\0';

    }

    printf("The rear is : %c\n", queue->array[queue->rear]);

}

void display(struct Queue \*queue)

{

    if (isEmpty(queue))

    {

        printf("Queue is empty\n");

        return;

    }

    printf("Queue elements: ");

    int i = queue->front;

    do

    {

        i = (i + 1) % queue->size;

        printf("%c ", queue->array[i]);

    } while (i != queue->rear);

    printf("\n");

}

int main()

{

    struct Queue q;

    initialize\_queue(&q, 5);

    enqueue(&q, '8');

    enqueue(&q, '9');

    enqueue(&q, '1');

    enqueue(&q, '3');

    display(&q);

    front(&q);

    rear(&q);

    printf("Dequeued element: %c\n", dequeue(&q));

    printf("Dequeued element: %c\n", dequeue(&q));

    front(&q);

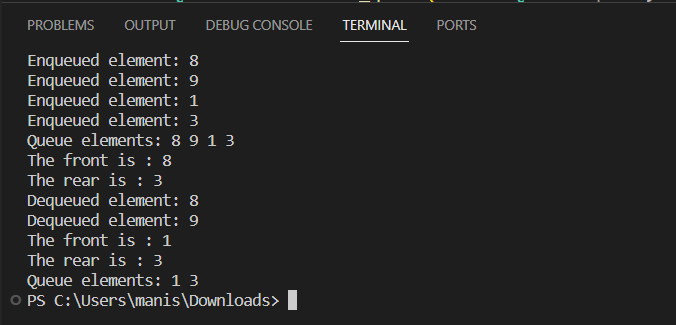
    rear(&q);

    display(&q);

    return 0;

}

**OUTPUT** :-



**Algorithm:**

**1. Initialize Queue**

- Create a structure called `Queue` with fields for front, rear, size, and an array to hold elements.

- Implement an `initialize\_queue` function that takes a pointer to a `Queue` structure and a size as input.

- Set the size of the queue and initialize both front and rear to 0.

- Allocate memory for the array using `malloc`.

**2. isEmpty**

- Implement an `isEmpty` function that takes a pointer to a `Queue` structure.

- Check if the rear and front of the queue are at the same index; if they are, the queue is empty and return 1; otherwise, return 0.

**3. isFull**

- Implement an `isFull` function that takes a pointer to a `Queue` structure.

- Check if the next position after the rear is equal to the front; if it is, the queue is full and return 1; otherwise, return 0.

**4. Enqueue**

- Implement an `enqueue` function that takes a pointer to a `Queue` structure and an element to be added.

- Check if the queue is full using the `isFull` function. If it's full, print "Queue Overflow."

- Otherwise, increment the rear pointer to the next position, and store the element at that position in the array.

**5. Dequeue**

- Implement a `dequeue` function that takes a pointer to a `Queue` structure.

- Check if the queue is empty using the `isEmpty` function. If it's empty, print "Empty queue" and return a default character (e.g., '\0').

- Otherwise, increment the front pointer to the next position and return the element at that position in the array.

**6. Front**

- Implement a `front` function that takes a pointer to a `Queue` structure.

- Check if the queue is empty using the `isEmpty` function. If it's empty, print "Queue is empty" and return a default character (e.g., '\0').

- Otherwise, print the element at the next position after the front in the array.

**7.Rear**

- Implement a `rear` function that takes a pointer to a `Queue` structure.

- Check if the queue is empty using the `isEmpty` function. If it's empty, print "Queue is empty" and return a default character (e.g., '\0').

- Otherwise, print the element at the rear position in the array

**8. Display Queue**

- Implement a `display` function that takes a pointer to a `Queue` structure.

- Check if the queue is empty using the `isEmpty` function. If it's empty, print "Queue is empty."

- Otherwise, loop through the elements in the queue, starting from the next position after the front and ending at the rear position, and print each element.

**9. Main Function**

- In the `main` function:

- Declare a `Queue` structure variable `q`.

- Initialize the queue using the `initialize\_queue` function with a size of 5.

- Enqueue several elements into the queue.

- Display the queue.

- Retrieve and print the front and rear elements.

- Dequeue two elements and print them.

- Display the updated queue.

**TOPIC** **2** :- N-Series and Chill

**CODE** :-

/\*

 \* File: nqueues.c

 \* Author: Manish Jadhav

 \* Email: manishsj289@gmail.com

 \* Created: September 16, 2023

 \* Description: This program implements an n-series queues data structure

 \*/

/\*

 N-Series and chill

\*/

#include "queue.c"

#include <stdio.h>

#include <stdlib.h>

struct NQueues

{

    int \*front;

    int \*rear;

    unsigned total\_queues;

    unsigned size\_per\_queue;

    struct Queue \*array[100];

};

struct NQueues \*createNQueues(int n, int capacity\_per\_queue)

{

    struct NQueues \*nqueues = (struct NQueues \*)malloc(sizeof(struct NQueues));

    nqueues->total\_queues = n;

    nqueues->size\_per\_queue = capacity\_per\_queue;

    for (int i = 0; i < n; i++)

    {

        nqueues->array[i] = initialize\_queue(capacity\_per\_queue);

    }

    return nqueues;

}

void add\_episode(struct NQueues \*nqueues, int episode\_id, int queue\_number)

{

    enqueue(nqueues->array[queue\_number], episode\_id);

}

char watch\_next\_episode(struct NQueues \*nqueues, int queue\_number)

{

    char ep = dequeue(nqueues->array[queue\_number]);

    return ep;

}

void display\_queue(struct NQueues \*nqueues, int queue\_number)

{

    display(nqueues->array[queue\_number]);

}

void display\_all(struct NQueues \*nqueues)

{

    for (int i = 0; i < nqueues->total\_queues; i++)

    {

        printf("> Queue No: %d", i + 1);

        display(nqueues->array[i]);

    }

}

int main()

{

    struct NQueues \*nqueues = createNQueues(3, 5);

    add\_episode(nqueues, 77, 0);

    add\_episode(nqueues, 65, 0);

    add\_episode(nqueues, 78, 0);

    add\_episode(nqueues, 73, 1);

    add\_episode(nqueues, 83, 1);

    add\_episode(nqueues, 72, 1);

    add\_episode(nqueues, 80, 2);

    add\_episode(nqueues, 85, 2);

    add\_episode(nqueues, 82, 2);

    display\_all(nqueues);

    char next\_episode = watch\_next\_episode(nqueues, 0);

    printf("Watched episode: %d : %c\n", next\_episode, next\_episode);

    printf("Queue after watching episode:\n");

    display\_queue(nqueues, 0);

    next\_episode = watch\_next\_episode(nqueues, 0);

    next\_episode = watch\_next\_episode(nqueues, 0);

    // Underflow check

    next\_episode = watch\_next\_episode(nqueues, 0);

    // Empty Queue Display

    printf("\nQueue after watching episode:");

    display\_queue(nqueues, 0);

    // Free Space

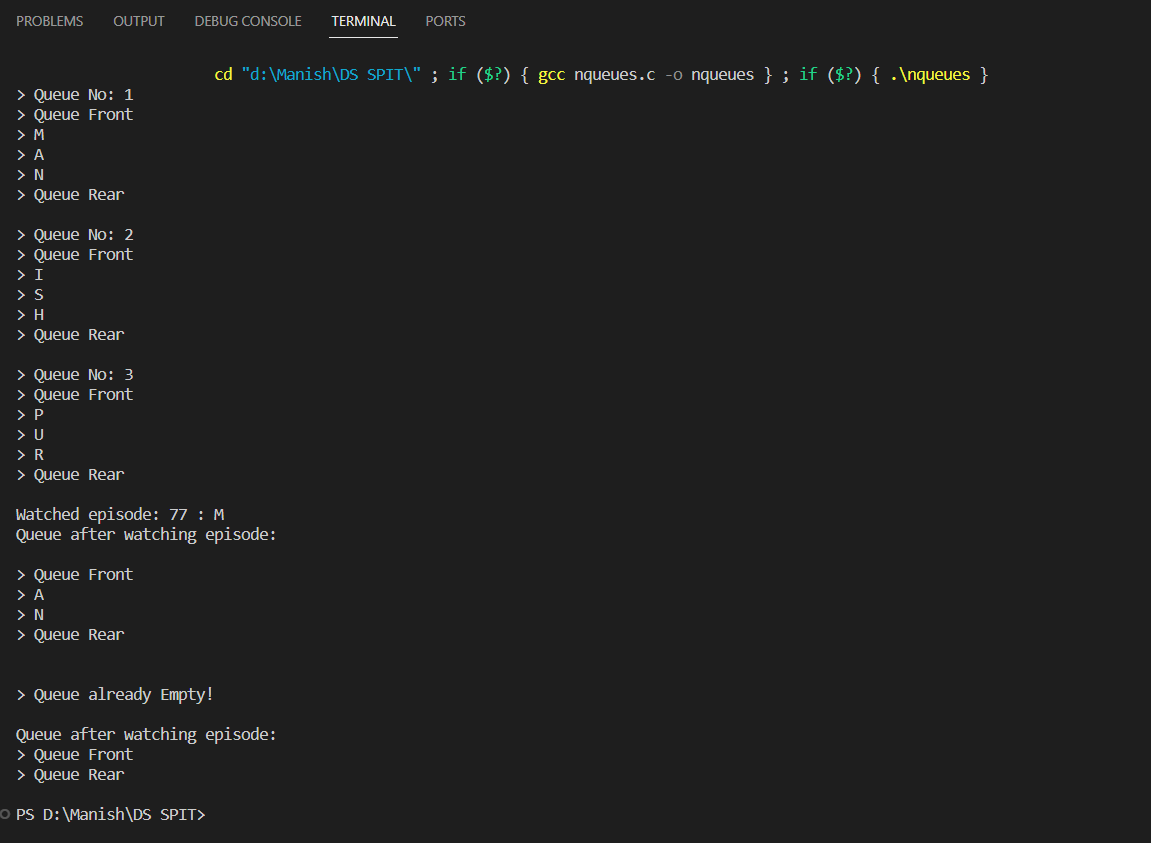
    free(nqueues->array);

    free(nqueues);

   return 0;

}

**OUTPUT** :-



**Algorithm:**

**1. Structure Definition:**

- Define a structure `struct NQueues` to store the N queues. Include fields for the front and rear pointers for each queue, the total number of queues, the size per queue, and an array of pointers to `Queue` structures.

**2. Initialization:**

- Implement a function `createNQueues(int n, int capacity\_per\_queue)` to create the NQueues data structure. Allocate memory for `struct NQueues`. Initialize the total number of queues and the size per queue.

- Initialize each queue in the array using the `initialize\_queue(capacity\_per\_queue)` function from "queue.c".

**3. Adding an Episode:**

- Implement a function `add\_episode(struct NQueues \*nqueues, int episode\_id, int queue\_number)` to add an episode to a specific queue.

- Use the `enqueue()` function from "queue.c" to add the episode to the specified queue.

**4. Watching Next Episode:**

- Implement a function `watch\_next\_episode(struct NQueues \*nqueues, int queue\_number)` to watch the next episode from a specific queue.

- Use the `dequeue()` function from "queue.c" to remove and return the next episode from the specified queue.

**5. Displaying a Queue:**

- Implement a function `display\_queue(struct NQueues \*nqueues, int queue\_number)` to display the contents of a specific queue using the `display()` function from "queue.c".

**6. Displaying All Queues:**

- Implement a function `display\_all(struct NQueues \*nqueues)` to display all the queues, including their contents.

**7. Main Function:**

- In the `main()` function:

- Create an instance of `struct NQueues` with the desired number of queues and capacity per queue using `createNQueues()`. Add episodes to various queues using the `add\_episode()` function.

- Display all the queues using `display\_all()`.

- Watch episodes from a specific queue using `watch\_next\_episode()`.

- Handle underflow conditions when trying to watch episodes from an empty queue.

- Free allocated memory for the data structure and its queues at the end.

A piece of paper with writing on it

Description automatically generated

A piece of paper with writing on it

Description automatically generated

A piece of paper with writing on it

Description automatically generated