**1. Node Class**

* Represents a node with data (job number), priority, and pointers to the next and previous nodes.

**2. createNode Function**

* Dynamically allocates memory for a new node.
* Initializes its data and priority based on the provided parameters.
* Returns a pointer to the created node.

**3. deque Function**

* Removes a node from the front of the queue.
* Returns the job number of the dequeued node.
* Handles the case when the queue is empty.

**4. enqueue Function**

* Adds a new node to the end of the queue.
* Returns a pointer to the updated rear node.

**5. printQ Function**

* Prints the elements of a queue (forward or reverse).
* Used for displaying the job execution sequence.

**6. display Function**

* Displays the job execution sequence for high, medium, and low priority queues.

**7. addJob Function**

* Adds a job to the corresponding priority queue (High, Medium, Low).
* Updates the rear pointer for the respective queue.
* Returns a priority code based on the queue.

**8. main Function**

* Initializes variables, queues, and flags for high, medium, and low priority queues.
* Uses a loop to add jobs to the queues based on priority.
* Displays the job execution sequence after adding initial jobs.
* Adds a new job with priority 'M'.
* Displays the job execution sequence after adding the new job.
* Dequeues some jobs from different queues and displays the updated sequence.

**Note:**

* Priority codes: 1 - High, 2 - Medium, 3 - Low
* The program simulates job scheduling with different priorities.
* Queues are implemented as doubly-linked lists.
* The program prints the job execution sequence after each operation.

Algorithm:

1. \*Node Class Definition:\*

- Define a class Node with integer data (job no), character priority (H, M, or L), and pointers to the next and previous nodes.

2. \*\*Node Creation Function (createNode):\*\*

- Dynamically allocate memory for a new node.

- Set the data and priority values.

- Print a message indicating the job scheduled.

- Return a pointer to the created node.

3. \*\*Dequeuing Function (deque):\*\*

- Check if the front pointer is NULL. If true, print "QEmpty" and return -1.

- Create a temporary pointer and point it to the front.

- Move the front pointer to the next node.

- Retrieve the data of the node.

- Delete the node.

- Return the retrieved data.

4. \*\*Enqueuing Function (enqueue):\*\*

- Create a new node using the createNode function.

- Update the pointers to link the new node at the end of the queue.

- Return the updated rear pointer.

5. \*\*Print Queue Function (printQ):\*\*

- Traverse the queue from the head node either forward or backward.

- Print the job number and priority for each node.

6. \*\*Display Function (display):\*\*

- Print the job execution sequence by calling printQ for each priority queue (High, Medium, and Low).

7. \*\*Add Job Function (addJob):\*\*

- Based on the priority, call enqueue for the respective priority queue (High, Medium, or Low).

- Return an integer indicating the priority of the added job (1 for High, 2 for Medium, 3 for Low).

8. \*\*Main Function (main):\*\*

- Declare arrays for job numbers (arr) and priorities (prior).

- Declare pointers for the front and rear of each priority queue.

- Initialize flags for each priority queue.

- Iterate over the job numbers and priorities to create and enqueue nodes based on priorities.

- Display the job execution sequence after initial jobs.

- Add a new job with priority 'M' and display the updated sequence.

- Dequeue some jobs from the high priority queue and display the updated sequence.

- Dequeue some jobs from the medium priority queue and display the updated sequence.

- Dequeue some jobs from the low priority queue and display the final sequence.

9. \*Return 0 to Indicate Successful Execution.\*