QUEUE Assignments in C

**1. Basic Operations Exercise:**

* **Task:** Implement a header file queue.h that declares a Queue structure and prototypes for the basic queue operations: enqueue(), dequeue(), peek(), and isEmpty().
* **Challenge:** Ensure that your implementation can handle different data types using macros or void pointers.

**CODE:**

QUEUE HEADER FILE: (queue.h)

#include <stdio.h>

#include <stdlib.h>

typedef struct Queue{

    int data;

    struct Queue\* next;

} Queue;

Queue\* enqueue(Queue \*front, Queue \*rear, int val){

    // Tries to insert an element to a queue following the FIFO (First In First Out)

    Queue \*node = (Queue\*)malloc(sizeof(Queue));

    node->data=val;//Setting its value

    node->next=NULL;// Pointing it to NULL pointer

    if (rear == NULL){//Checking the overflow condition

        printf("OVERFLOW CONDITION");

        return NULL;

    }

    if (front == NULL){

        front = node;

        rear = node;

    }

    rear->next=node;//Pointing the rear pointer to new node

    return node;

}

int dequeue(Queue \*front){

    //Tries to delete the element following the FIFO

    if (front == NULL){

        printf("UNDERFLOW \n");

        return -1;

    }

    Queue \*temp = front;// Initializing the temporary pointer to front

    int element = temp->data;

    front = front->next;//Moving the front pointer to the next node

    free(temp);// deleting the temporary node

    return element;

}

int peek(Queue\* front){

    //Tries to return the top element of the queue

    if (front == NULL){

        printf("The queue id empty.\n");

        return -1;

    }

    return front->data;

}

int isEmpty(Queue \*front){

    //Checks if the queue is empty

    if (front == NULL){

        return 1;// True

    }

    return 0;// False

}

void display(Queue \*front){

    //Displays all the elements in the queue

    Queue\* temp = front;//Temporary node initialized to front pointer

    printf("QUEUE: \n");

    while (temp != NULL){

        printf("%d - ", temp->data);

        temp = temp->next;

    }

    return;

}

**QUEUE IMPLEMENTATION:**

#include <stdio.h>

#include "queue.h"

int main(){

//Creating a queue with 10 elements

Queue \*node1 = enqueue(node1, node1, 1324);

Queue \*node2 = enqueue(node1, node1, 1325);

Queue \*node3 = enqueue(node1, node2, 1326);

Queue \*node4 = enqueue(node1, node3, 1327);

Queue \*node5 = enqueue(node1, node4, 1328);

Queue \*node6 = enqueue(node1, node5, 1329);

Queue \*node7 = enqueue(node1, node6, 1330);

Queue \*node8 = enqueue(node1, node7, 1331);

Queue \*node9 = enqueue(node1, node8, 1332);

Queue \*node10 = enqueue(node1, node9, 1333);

//Displaying the elements in the queue

display(node1);

printf("\n");

//Dequeuing

int deque = dequeue(node1);

printf("Dequeued elements is %d\n", deque);

display(node2);

printf("\n");

//Displaying the top element in the queue

int top = peek(node2);

printf("The top element is %d", top);

return 0;

}

**2. Application Scenario - Job Scheduling:**

* **Scenario:** You need to simulate a simple job scheduling system where jobs are added to a queue and processed in a FIFO manner.
* **Task:** Develop a C program that uses the queue implemented in queue.h to add jobs, each with a unique ID and priority, and process them in order.
* **Output:** Display the order of job processing.

**3. Real-time Data Processing:**

* **Scenario:** Simulate a scenario where sensor data (e.g., temperature readings) are collected at regular intervals and must be processed in the order they are received.
* **Task:** Implement a system using your queue where sensor data points are enqueued as they arrive and dequeued for processing.
* **Challenge:** Implement overflow handling when the queue reaches its maximum capacity by discarding the oldest data.

**4. Reverse a Queue:**

* **Task:** Write a function to reverse the contents of a queue using only standard queue operations (enqueue, dequeue) and no additional data structures.
* **Output:** Before and after reversing the queue contents.

CODE:

* #include <stdio.h>
* #include "queue.h"
* void reverseQueue(Queue \*front, Queue \*rear){
* if (isEmpty(front) == 1){
* return;
* }
* int deque = dequeue(front);//Storing the first element which is deleted using the FIFO or dequeue
* Queue \*node = enqueue(front, rear, deque);//Enqueuing that dequeue value from the queue
* return;
* }
* int main(){
* Queue \*node1 = enqueue(node1, node1, 1324);
* Queue \*node2 = enqueue(node1, node1, 1325);
* Queue \*node3 = enqueue(node1, node2, 1326);
* Queue \*node4 = enqueue(node1, node3, 1327);
* Queue \*node5 = enqueue(node1, node4, 1328);
* Queue \*node6 = enqueue(node1, node5, 1329);
* Queue \*node7 = enqueue(node1, node6, 1330);
* Queue \*node8 = enqueue(node1, node7, 1331);
* Queue \*node9 = enqueue(node1, node8, 1332);
* Queue \*node10 = enqueue(node1, node9, 1333);
* //Displaying the queue
* reverseQueue(node1, node9);
* display(node1);
* }

**5. Queue Duplication:**

* **Task:** Implement a function duplicateQueue() that creates a copy of an existing queue.
* **Challenge:** Ensure that the original queue remains unmodified after the duplication process.

**6. Concatenate Two Queues:**

* **Task:** Write a function to concatenate two queues, where all elements of the second queue follow all elements of the first queue, and the second queue is left empty.
* **Output:** Display both queues before and after concatenation.

**----------------------------------- Good luck ----------------------------------**