

Name- Yashwant Chandrakant Bhosale
MIS – 612303039
SY comp Div 1

Assignment 4: Queue

Write files: queue.c queue.h and testqueue.c
Submit a single file: MISID.tar.gz

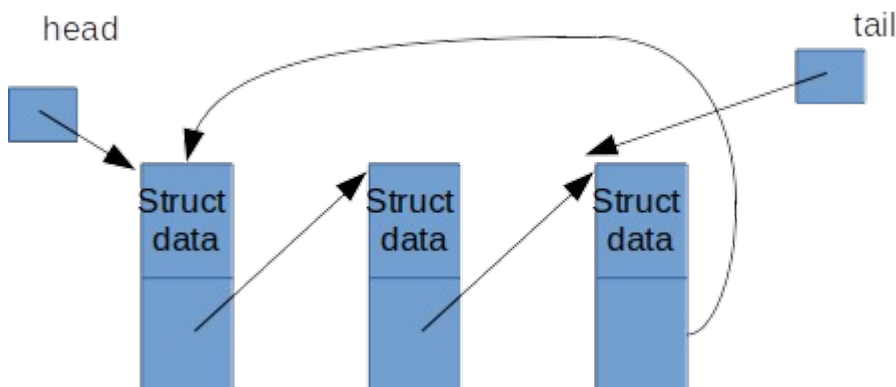
Implement a queue of structures. The queue stores the following structure:

```
typedef struct data {  
    char name[16];  
    unsigned int age;  
}data;
```

Write the following queue functions: **qinit, enq, deq, qfull, qempty**;
Define the proper prototypes in **queue.h** and write the code in **queue.c**

The queue implementation should be done using singly linked circularly connected structures (nodes), where each node will store one instance of 'data'.
You can use head and tail - both pointers

In **qfull**, you may simply assume that the queue is never full and write code accordingly. The real challenging part (for no extra marks) is to detect full if malloc fails, but still make sure that qfull() can detect it and enq() always works if qfull() said not-full.
Here is a diagram of how your queue will store the data: Note that tail is optional and you may or may not use it.



Do a proper typedef of the queue type in **queue.h**

Use provided main() function in a file called **testqueue.c** to test your queue. Make sure that your code compiles with this code and you can test it.

Solution:

Code:

queue.h : Contains the struct declaration and function prototypes required for the program.

```
/* Declaration for data */
typedef struct data {
    char name[16];
    unsigned int age;
} data;

/* Node declaration of the linked list */
typedef struct node {
    data d;
    struct node *next;
} node;

/* Queue Struct Declaration */
typedef struct {
    node *head, *tail;
    short is_full;
} queue;

void qinit(queue *q); // Function to initialize the queue
void enq(queue *q, data d); // Function to insert an element to the queue
data deq(queue *q); // Function to remove an element from the queue
short qfull(queue *q); // Function to check if queue is full
short qempty(queue *q); // Function to check if queue is empty
void print_queue(queue *q); // Function to print the queue
```

queue.c : Contains all the function definitions required for the program.

```
#include <stdio.h>
#include <stdlib.h>
#include "queue.h"

void qinit(queue *q) {
    if(!q) return;
    /* Initialize head and tail to NULL */
    q->head = NULL;
    q->tail = NULL;
    return;
}
```

```

void enq(queue *q, data d) {
    if(!q) return;

    node *nn = (node *) malloc(sizeof(node));

    /* If malloc fails to allocate memory the is_full flag in queue data structure will be
       set, so that we will be able to detect that queue is full */
    if(!nn) {
        q->is_full = 1; // set is_full flag if memory allocation fails
        return;
    }else {
        q->is_full = 0;
    }

    nn->d = d;
    if(q->head == NULL) {
        q->head = nn;
        q->tail = nn;
        nn->next = nn; // Point to the first node to make the circular list
    }else {
        q->tail->next = nn;
        nn->next = q->head; // Point to the first node to make the circular list
        q->tail = nn; // Update the tail pointer
    }
    return;
}

data deq(queue *q) {
    data d = {"", -1}; // Invalid data

    if(!q) return d; // if queue is not initilized return invalid data
    else if(qempty(q)) return d; // if queue is empty return invalid data

    node *p = q->head;

    // If queue contains single node
    if(q->head == q->tail) {
        d = p->d;
        q->head = q->tail = NULL;
    }else {
        q->head = p->next; // Update head
        q->tail->next = q->head; // Update next node of the tail to ensure circular queue
        d = p->d; // Removed data
    }

    free(p);
    return d;
}

short qempty(queue *q) {
    if(!q) return 1;

    if(q->head == NULL)
        return 1;
    else
        return 0;
}

short qfull(queue *q) {
    return q->is_full;
}

```

```

void print_queue(queue *q) {
    if(!q) return;

    if(qempty(q)) return;

    printf("[\t");

    node *p = q→head;
    while(p→next ≠ q→head){
        printf("{ %s, %d }\t", p→d.name, p→d.age);
        p=p→next;
    }

    if(p→next = q→head)
        printf("{ %s, %d }\t", p→d.name, p→d.age);

    printf("]\n");
    return;
}

```

main.c : Contains the main execution and flow of the program. It first reads input from the input keeps reading until user presses 'ctrl + d' on the keyboard (so that scanf returns -1). Then prints all the elements one by one as it dequeue them from the queue.

```

#include <stdio.h>

#include "queue.h"
int main() {
    queue q;
    data d;

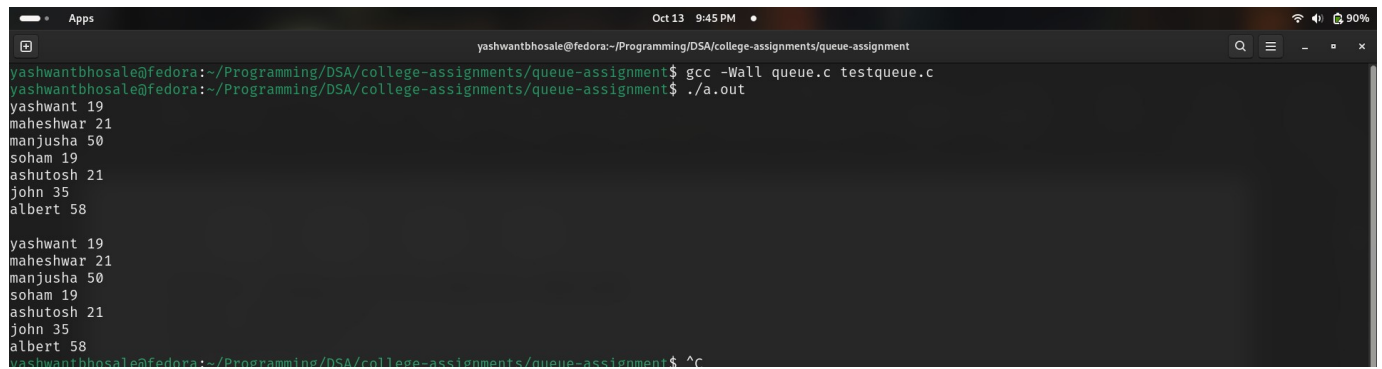
    qinit(&q);
    while(scanf("%s%u", d.name, &(d.age)) ≠ -1)
        if(!qfull(&q))
            enq(&q, d);

    // We may exit the loop if queue becomes full

    while(!qempty(&q)) {
        d = deq(&q);
        printf("%s %u\n", d.name, d.age);
    }
    return 0;
}

```

Output:



```
yashwantbhosale@fedora:~/Programming/DSA/college-assignments/queue-assignment$ gcc -Wall queue.c testqueue.c
yashwantbhosale@fedora:~/Programming/DSA/college-assignments/queue-assignment$ ./a.out
yashwant 19
maheshwar 21
manjusha 50
soham 19
ashutosh 21
john 35
albert 58

yashwant 19
maheshwar 21
manjusha 50
soham 19
ashutosh 21
john 35
albert 58
yashwantbhosale@fedora:~/Programming/DSA/college-assignments/queue-assignment$ ^C
```

Here, as the user is entering the data elements, they are being **pushed inside the queue** and when the program concludes (after pressing '**ctrl + d**' on keyboard) the elements are popped one by one and printed. Here, we can see that elements come out in **First In First Out pattern** or **FIFO**.