Show Course Contents ⊕

Overview Q&A Downloads Announcements

C Code For Queue and its Operations Using Arrays in Data Structure

In the last tutorial, we finished learning concepts behind implementing the basic operations of a queue ADT using arrays. Those were enqueue, dequeue, *isEmpty* and *isFull.* Today we will learn how to program all of those. Without further ado, let's move to our editors!

I have attached the whole source code for your referral. Follow it while understanding.

Understanding the code snippet below:

1. First of all, start by creating a struct named *queue*, and define all of its four members we discussed yesterday. An integer variable *size* to store the size of the array, another integer variable *f* to store the front end index, and an integer variable *r* to store the index of the rear end. Then, define an integer pointer *arr* to store the address of the dynamically allocated array.

```
struct queue
{
    int size;
    int f;
    int r;
    int* arr;
};
```

Code Snippet 1: Declaring struct queue

2. In main, declare a struct queue *q,* and initialize its instances. Declare some size of the array, let 100. Initialize both f and r with -1. And allocate memory in heap for *arr* using malloc. Don't forget to include the header file <stdlib.h>

```
    struct queue q;
    q.size = 4;
    q.f = q.r = 0;
    q.arr = (int*) malloc(q.size*sizeof(int));
```

Code Snippet 2: Defining and initializing a struct element q

3. Creating Enqueue:

Create a void function *enqueue,* pass the pointer to the struct queue *q,* and the value to insert as parameters. First of all, check if the queue is full by calling the *isFull* function. If it returns 1, then print the condition of the queue overflow and return. Else, increase the r value of q using the arrow operator, and insert the new value at the index r of the array *arr.*

```
void enqueue(struct queue *q, int val){
    if(isFull(q)){
        printf("This Queue is full\n");
    }
    else{
        q->r++;
        q->arr[q->r] = val;
        printf("Enqued element: %d\n", val);
    }
}
```

Code Snippet 3: Creating the enqueue function

4. Creating isFull:

Create an integer function *isFull,* and pass into it the pointer to the struct queue *q* as the only parameter. In the function, check if the *r* element of struct queue *q* is equal to the (size element)-1. If it is, then there is no space left in the queue to insert elements anymore, hence return 1, else 0.

```
int isFull(struct queue *q){
    if(q->r==q->size-1){
        return 1;
    }
    return 0;
}
```

Code Snippet 4: Creating the isFull function

5. Creating Dequeue:

Create an integer function *dequeue,* and pass the pointer to the struct queue *q,* as the only parameter into it. In the function, first of all, check if the queue is already not empty by calling the *isEmpty* function. If it returns 1, then print the condition of the queue underflow and return. Else, increase the *f* value of *q* using the arrow operator, and store the value at the index *f* of the array in some integer variable *a*. Later, return *a*.

```
int dequeue(struct queue *q){
    int a = -1;
    if(isEmpty(q)){
        printf("This Queue is empty\n");
    }
    else{
        q->f++;
        a = q->arr[q->f];
    }
    return a;
}
```

Code Snippet 5: Creating the dequeue function

6. Creating isEmpty:

Create an integer function *isEmpty,* and pass into it the pointer to the struct queue *q,* as the only parameter. Inside the function, check if the *r* element of the *q is* equal to the *f* element of the *q*. Intuitively speaking, the difference between the values of *f & r* is the size of the queue. And if they both are equal, the size is 0. Therefore, if they are equal, return 1, else return 0.

```
int isEmpty(struct queue *q){
    if(q->r==q->f){
        return 1;
    }
    return 0;
}
```

Code Snippet 6: Creating the isEmpty function

Here is the whole source code:

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

struct queue
{
    int size;
    int f;
    int r;
    int* arr;
};


int isEmpty(struct queue *q){
    if(q->r==q->f){
        return 1;
    }
    return 0;
}

int isFull(struct queue *q){
    if(q->r==q->size-1){
        return 1;
    }
    return 0;
}

void enqueue(struct queue *q, int val){
    if(isFull(q)){
        printf("This Queue is full\n");
    }
    else{
        q->r++;
        q->arr[q->r] = val;
        printf("Enqued element: %d\n", val);
    }
}

int dequeue(struct queue *q){
    int a = -1;
    if(isEmpty(q)){
        printf("This Queue is empty\n");
    }
    else{
        q->f++;
        a = q->arr[q->f];
    }
    return a;
}

int main(){
    struct queue q;
    q.size = 4;
    q.f = q.r = 0;
    q.arr = (int*) malloc(q.size*sizeof(int));

    // Enqueue few elements
    enqueue(&q, 12);
    enqueue(&q, 15);
    enqueue(&q, 1);
    printf("Dequeuing element %d\n", dequeue(&q));
    printf("Dequeuing element %d\n", dequeue(&q));
    printf("Dequeuing element %d\n", dequeue(&q));
    enqueue(&q, 45);
    enqueue(&q, 45);
    enqueue(&q, 45);

    if(isEmpty(&q)){
        printf("Queue is empty\n");
    }
    if(isFull(&q)){
        printf("Queue is full\n");
    }

    return 0;
}
```

Code Snippet 7: Implementing a queue and its operations using arrays

Let's now check if our functions work all good. Since we have not inserted any element in the queue yet, we'll see what the *isEmpty* function has to say.

```
    if(isEmpty(&q)){
        printf("Queue is empty\n");
    }
```

Code Snippet 8: Using the isEmpty function

The output we received was:

```
Queue is empty
PS D:\MyData\Business\code playground\Ds & Algo with Notes\Code>
```

Figure 1: Output of the above program

Let us now insert/enqueue some elements inside the queue.

```
    enqueue(&q, 12);
    enqueue(&q, 15);
```

Code Snippet 9: Using the enqueue function

Our terminal had the following output:

```
Enqued element: 12
Enqued element: 15
PS D:\MyData\Business\code playground\Ds & Algo with Notes\Code>
```

```
    printf("Dequeuing element %d\n", dequeue(&q));
```

Code Snippet 10: Using the dequeue function

And the output we received was:

```
Dequeuing element 12
Dequeuing element 15
PS D:\MyData\Business\code playground\Ds & Algo with Notes\Code>
```

Figure 3: Output of the above program

Every output was very accurate to my knowledge. And here we complete learning the basics of queues. Each of these topics was covered in detail. You may want to go through them a second time if you still couldn't get through. But you will surely understand everything. It's time we move ahead with our new topic.

I appreciate your support throughout. I hope you enjoyed the tutorial. If you genuinely appreciate my work, please let your friends know about this course too. If you haven't checked out the whole playlist yet, move on to codewithharry.com or my YouTube channel to access it. See you all in the next tutorial where we'll learn another variation of queues**, circular queues**. Till then, keep coding.

Previous Next

CodeWithHarry