Queue | Set 1 (Introduction and Array Implementation)

Like Stack, Queue is a linear structure which follows a particular order in which the operations are performed. The order is **F**irst **I**n **F**irst **O**ut (FIFO). A good example of queue is any queue of consumers for a resource where the consumer that came first is served first.

The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

Operations on Queue:

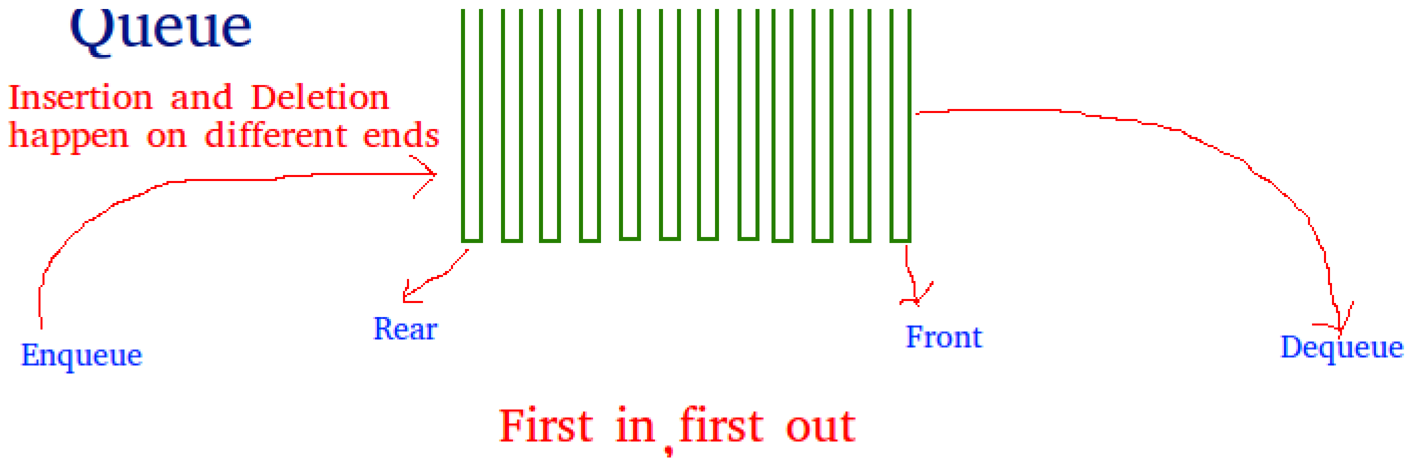
Mainly the following four basic operations are performed on queue:

Enqueue: Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.

Dequeue: Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition.

Front: Get the front item from queue.

Rear: Get the last item from queue.



Applications of Queue:

Queue is used when things don't have to be processed immediatly, but have to be processed in **F**irst **I**n**F**irst **O**ut order like Breadth First Search. This property of Queue makes it also useful in following kind of scenarios.

1) When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling.

2) When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.

See this for more detailed applications of Queue and Stack.

Array implementation Of Queue

For implementing queue, we need to keep track of two indices, front and rear. We enqueue an item at the rear and dequeue an item from front. If we simply increment front and rear indices, then there may be problems, front may reach end of the array. The solution to this problem is to increase front and rear in circular manner (See this for details)

```
// C program for array implementation of queue
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

// A structure to represent a queue
struct Queue
{
    int front, rear, size;
    unsigned capacity;
    int* array;
};
```

```
{
    struct Queue* queue = (struct Queue*) malloc(sizeof(struct Queue));
    queue->capacity = capacity;
    queue->front = queue->size = 0;
    queue->rear = capacity - 1;  // This is important, see the enqueue
    queue->array = (int*) malloc(queue->capacity * sizeof(int));
    return queue;
}

// Queue is full when size becomes equal to the capacity
int isFull(struct Queue* queue)
{  return (queue->size == queue->capacity);  }

// Queue is empty when size is 0
int isEmpty(struct Queue* queue)
{  return (queue->size == 0); }

// Function to add an item to the queue.  It changes rear and size
void enqueue(struct Queue* queue, int item)
{
    if (isFull(queue))
        return;
    queue->rear = (queue->rear + 1)%queue->capacity;
    queue->array[queue->rear] = item;
    queue->size = queue->size + 1;
    printf("%d enqueued to queue\n", item);
}

// Function to remove an item from queue.  It changes front and size
int dequeue(struct Queue* queue)
{
    if (isEmpty(queue))
        return INT_MIN;
    int item = queue->array[queue->front];
    queue->front = (queue->front + 1)%queue->capacity;
    queue->size = queue->size - 1;
    return item;
}

// Function to get front of queue
int front(struct Queue* queue)
{
    if (isEmpty(queue))
        return INT_MIN;
    return queue->array[queue->front];
}

// Function to get rear of queue
int rear(struct Queue* queue)
```

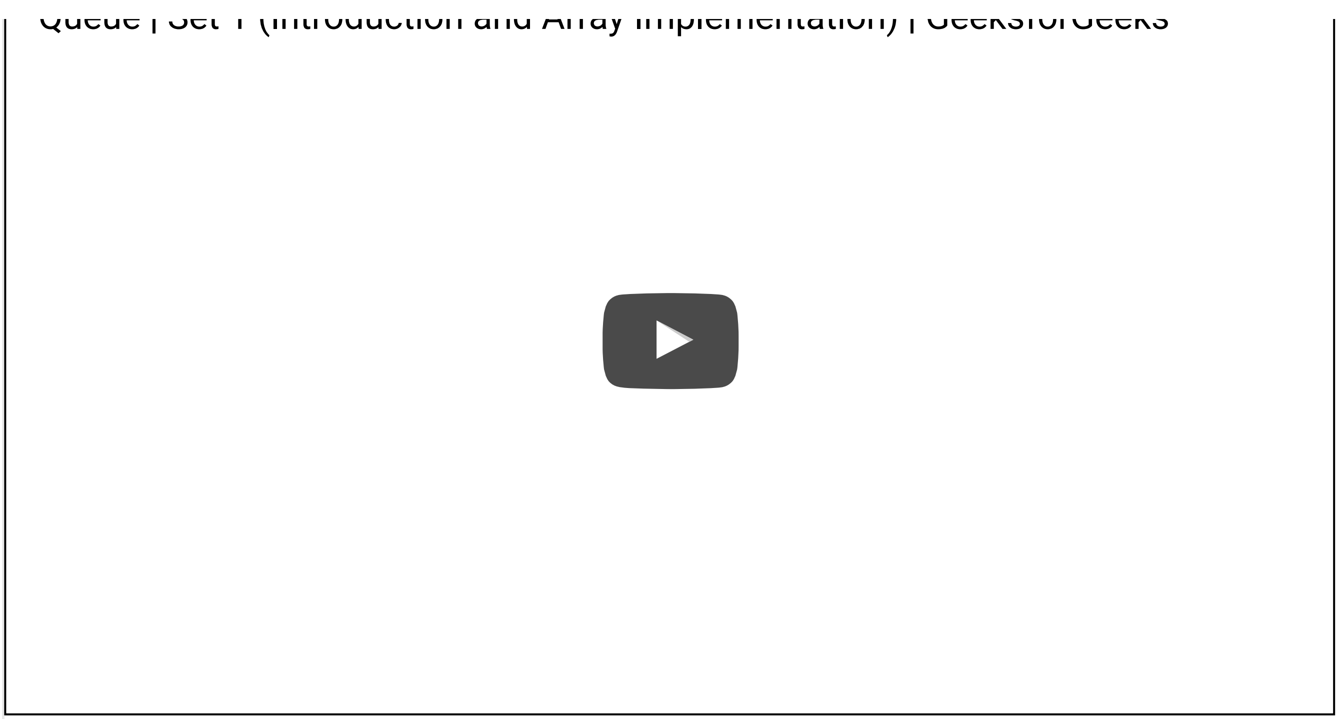
```
    return queue->array[queue->rear];
}

// Driver program to test above functions./
int main()
{
    struct Queue* queue = createQueue(1000);

    enqueue(queue, 10);
    enqueue(queue, 20);
    enqueue(queue, 30);
    enqueue(queue, 40);

    printf("%d dequeued from queue\n", dequeue(queue));

    printf("Front item is %d\n", front(queue));
    printf("Rear item is %d\n", rear(queue));

    return 0;
}
```

Run on IDE

Output:

```
10 enqueued to queue
20 enqueued to queue
30 enqueued to queue
40 enqueued to queue
10 dequeued from queue
Front item is 20
Rear item is 40
```

Time Complexity: Time complexity of all operations like enqueue(), dequeue(), isFull(), isEmpty(), front() and rear() is O(1). There is no loop in any of the operations.



Queue | Set 1 (Introduction and Array Implementation) | GeeksforGeeks

Linked list implementation is easier, it is discussed here: Queue | Set 2 (Linked List Implementation)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

GATE CS Notes (According to Official GATE 2017 Syllabus)

GATE CS Corner

See Placement Course for placement preparation, GATE Corner for GATE CS Preparation and Quiz Corner for all Quizzes on GeeksQuiz.

Category: Queue

(Login to Rate and Mark)

Average Rating : **5/5.0**

Based on **7** vote(s)

★★★★★

2.1

Average Difficulty : **2.1/5.0**

Based on **61** vote(s)

Add to TODO List

Mark as DONE

61 Comments **GeeksQuiz** 1 **Login**

♡ **Recommend** 5 **Share** **Sort by Best**

typing.. • 3 years ago

I think there is not need of the size variable in queue structure, we can perform all operations without it.

9 ^ | ˅ • Reply • Share ›

Bofin Babu ➦ typing.. • 2 years ago

In the program described, if there's no size variable we can't check the boundary conditions ie. whether the queue is empty or full. And this is important here since it is circular.

1 ^ | ˅ • Reply • Share ›

Afsar Alam ➦ typing.. • 8 months ago

Here I am assuming queue is circular one:
If you wont use size variable then in case of dequeuing/poping there will not be any problem because rear and front will be equal when queue is empty and we can skip using size variable here but while enqueing/pushing you will not be able to inspect whether the queue is full or not .

^ | ˅ • Reply • Share ›

Kim Jong-il • 3 years ago

```
struct Queue* createQueue(unsigned capacity)
{
    struct Queue* queue = (struct Queue*) malloc(sizeof(struct Queue));
    queue->capacity = capacity;
    queue->front = queue->size = 0;
    queue->rear = capacity - 1;  // This is important, see the enqueue
    queue->array = (int*) malloc(queue->capacity * sizeof(int));
    return queue;
}
```

@GeeksforGeeks Is not here queue->rear= queue->front; Initially both front and rear both pointng to the same location. Since It is an array implementation then it should be set to -1

bhavik gujarati ➦ Kim Jong-il • a year ago

This is circular array. So, initialization wouldn't be to -1 of front and rear.

And initially it won't be "queue->rear= queue->front" because while doing enqueue operation, we are incrementing rear first and then adding item to the queue and while doing dequeue operation, we are popping an element from queue and then incrementing front. So, initially rear is 1 less than front (in circular manner). This mechanism is to make sure that, at all the time, front will be pointing to the first element of the queue and rear will be pointing to the last element of the queue.

^ | ˅ • Reply • Share ›

nitish • 3 years ago

Enqueue: Adds an item to the queue. If the ""stack"" is full, then it is said to be an Overflow condition. it will not be stack ,it will be queue..
just a printing mistake.

2 ^ | ˅ • Reply • Share ›

GeeksforGeeks Mod ➦ nitish • 3 years ago

Thanks for pointing this out. We have corrected the typo.

^ | ˅ • Reply • Share ›

AllergicToBitches • 2 years ago

We can initialize rear as -1 with no change in code.

```
queue->rear = -1;
```

Or can initalize it with 0 with slight modifications in our code

```
queue->rear = 0;
```

1. In function enqueue(), enqueue item before incrementing rear.

```
queue->array[queue->rear] = item;
queue->rear = (queue->rear + 1)%queue->capacity;
```

```
return queue->array[queue->rear-1];
```

1 ^ | v • Reply • Share ›

vasavi ➦ AllergicToBitches • 10 months ago

a block for new entry is creates, when we use rear+1
so then can add element to array

^ | v • Reply • Share ›

PJ • 25 days ago

Hi can you please give code in java? I did read from other sites and thought will give the java code myself. However I feel my knowledge is limited. So requesting.
Thanks

^ | v • Reply • Share ›

Akhil • a month ago

Easy implementation in java :- http://code.geeksforgeeks.o...

^ | v • Reply • Share ›

PJ ➦ Akhil • 24 days ago

Thanks

^ | v • Reply • Share ›

Akhil ➦ PJ • 24 days ago

You're welcome bro :)

^ | v • Reply • Share ›

Manjukeshwar Reddymandadi • 3 months ago

there is a problem with this code here if I enqueue and dequeue some integer, then rear pointer is changed to first position and from now deque is not possible as when enqued only rear pointer moves and front will be at first position.

^ | v • Reply • Share ›

Without using size variable in struct:
https://github.com/ashishni...

^ | v • Reply • Share ›

SACHIN TALAKERI • 6 months ago

queue->rear = (queue->rear + 1)%queue->capacity;
is this line applicable even if (queue->rear== queue->capacity && queue->front==queue->array[0]) ??? will it not overwrite queue->array[0]?
Can anyone will explain

^ | v • Reply • Share ›

Mazhar MIK • 8 months ago

My user friendly code for this.

```
#include <stdio.h>
#include <stdlib.h>
#define size 5

//insert
void Insert(int * queue,int * rear,int *count)
{
int data;
if((*count)==(size))
{
printf("Queue is full\n");
return;
}
printf("Enter the data: ");
scanf("%d",&data);
(*rear)=((*rear)+1);
if((*rear)==(size-1))
```

see more

^ | v • Reply • Share ›

Python Implementation:
http://ideone.com/TP0ERM

^ | v • Reply • Share ›

arjun gulyani ➦ apoorv gupta • 7 months ago

Could you please explain this --->

```
if (self.rear + 1) % self.size == self.front:
print "Queue Full"
```

^ | v • Reply • Share ›

Sonia Jain • 9 months ago

if we are dequeue then are not we rearranging the array.

^ | v • Reply • Share ›

srushtee satardey • 9 months ago

```
queue->array = (int*) malloc(queue->capacity * sizeof(int));
```

can anybody please translate this in c++. thanx

^ | v • Reply • Share ›

Prashant Babber • 9 months ago

We can do this without the ''size'' variable but in that case we have to trade-off with the number of elements in the queue(number of elements = capacity-1). if we start adding elements 1 position after the front,we can distinguish between front==rear i.e if it is an emptyqueue or a fullqueue

^ | v • Reply • Share ›

Prashant Babber ➦ Prashant Babber • 9 months ago

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
struct queue{
int capacity;
int rear;
```

```
};

struct queue *CreateQueue(unsigned capacity){
struct queue *s=(struct queue*)malloc(sizeof(struct queue));
if(!s){
printf("Mem err");
}
s->capacity=capacity;
s->rear=s->front=0;
```

see more

^ | v • Reply • Share ›

Pawnesh Raj → Prashant Babber • 9 months ago

what is the use of %(s->capacity) in isqueueenpty.. ?
why we cant directly compare s->front and s->rear ?//

^ | v • Reply • Share ›

Code@Geeks • 10 months ago

Using Class:
http://ideone.com/u9LOg4

^ | v • Reply • Share ›

Devesh Lall • 10 months ago

Unlike Linkedlist, there is no method by which the implementation could be carried out in Java in case of queue or stacks

^ | v • Reply • Share ›

Balu Chowhan • a year ago

Implemented in java

```
package queueexample;
```

```
int front = -1;
int rear;
int MAXSIZE;
int []arr;

public QueueArray(int MAXSIZE) {
this.MAXSIZE = MAXSIZE;
arr = new int[MAXSIZE];

}

void enqueue( int k){

if(front==-1){
```

see more

^ | ⌄ • Reply • Share ›

Anurag Tangri ➦ Balu Chowhan • 10 months ago

can you please post linkedlist implementation of queue in java

^ | ⌄ • Reply • Share ›

bhavik gujarati • a year ago

@GeeksforGeeks

The problem with non-circular queue is "rear may reach end of array", not "front may reach end of array". Typo in the article.

^ | ⌄ • Reply • Share ›

Rajat Toshniwal • a year ago

Hey, How about altering JUST deQueue part (i.e., Ignoring Front pointer, while decrementing the rer pointer on deQueuing)
Code: http://code.geeksforgeeks.o...

Here is new deQueue()

```
// Function to remove an item from queue. It changes front and size
int deQueue(struct Queue* queue) {
```

```
return INT_MIN;
int item = queue->array[queue->rear];

if(queue->rear != 0) {
queue->rear = (queue->rear - 1);
} else {
queue->rear = queue->capacity;
}

queue->size = queue->size - 1;
return item;
}
```

⌃ ⌄ • Reply • Share ›

Minakshi Boruah • a year ago

In gate paper it was just mentioned that it's an array not of CIRCULAR. Because for a linear array one of the op'n (dequeu & adjust) would req omega(n).

⌃ ⌄ • Reply • Share ›

kumar • a year ago

why there is no queue deallocation before exit ???

⌃ ⌄ • Reply • Share ›

GOURAV KHANIJOE • a year ago

I think in dequeue function when we are decrementing the size, there should be a check to see if size==0, if yes then rear should be reinitialized to capacity-1....please correct me if i am wrong

⌃ ⌄ • Reply • Share ›

aman bhardwaj • 2 years ago

is it correct????????

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

```
//#define max 100

struct queue

{

int r;

int f;

int cap;

int *array;

};
```

see more

^ | ⌄ • Reply • Share ›

Shallote Dsouza • 2 years ago

Wat can b the ans to dis question..
Queue using one access pointer. Reimplement a queue with al operations taking constant time but only 1 instance variable instead of two

^ | ⌄ • Reply • Share ›

yb • 2 years ago

why is rear initialise to capacity - 1; rather it sud be initialise to -1

^ | ⌄ • Reply • Share ›

garima Choudhary • 2 years ago

queue->front = (queue->front + 1)%queue->capacity;
I dont understand this line ??

^ | ⌄ • Reply • Share ›

paras ➜ garima Choudhary • 2 years ago

sasha • 2 years ago

please someone explain

return (queue->size == queue->capacity);

this statement n what does int_min signifies in the following statement

return INT_MIN;

^ | v • Reply • Share ›

kumar ➦ sasha • a year ago

INT_MIN value is "-2147483648" for 4 byte. Just to indicate that queue is empty.

return (queue->size == queue->capacity); // means if size == capacity it will return true else false.

^ | v • Reply • Share ›

sasha • 2 years ago

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

what is the use of %queue->capacity in this..can't we use without this ?

^ | v • Reply • Share ›

Ankush Jolly ➦ sasha • 2 years ago

http://www.cs.colostate.edu...

This link can help. Actually, they have given that link above to refer. This has been done to make queue array circular.

1 ^ | v • Reply • Share ›

Klaus • 2 years ago

http://ideone.com/sV89vM We can avoid the rear variable.

^ | v • Reply • Share ›

Holden • 2 years ago

^ | v • Reply • Share ›

GeeksforGeeks Mod ➦ Holden • 2 years ago

Hengameh, thanks for writing to us. The current sentence seems fine. Please below cmu link for reference: http://www.cs.cmu.edu/~adam...

2 ^ | v • Reply • Share ›

Ajoy • 2 years ago

In the dequeue function after dequeuing the element if queue gets empty then changing the front and rear pointer to zero also required. In my thinking I feel the code should be as like below,

```
int dequeue(struct Queue* queue)
{
if (isEmpty(queue))
return INT_MIN;
int item = queue->array[queue->front];
queue->size = queue->size - 1;
if (isEmpty(queue))
queue->front = queue->rear = 0;
else

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

return item;
}
```

^ | v • Reply • Share ›

swasti • 2 years ago

What is the difference with this implementation and circular queue implementation? Please comment?

^ | v • Reply • Share ›

Rachit Nagdev ➦ swasti • 2 years ago

This is a circular queue Implementation and implemented by "Use a fill count" (in our case "size") Approach.

-The test for full/empty is simple.

The disadvantages are:
-You need modulo for enqueue and dequeue.
-Enqueue and dequeue operations must share the counter field, so it requires synchronization in multi-threaded situation.

Reference: wikipedia

• Reply • Share ›

Shubham Aggarwal • 2 years ago

Can this be correct?

```
void Enqueue(struct Queue* queue, int data)
{
if(isFull) return;
queue->array[++queue->size] = data; //increasing size and assigning value in single step.
queue->rear++;
}
```

• Reply • Share ›

Sujit Roy • 2 years ago

what does it mean
(queue->front + 1)%queue->capacity;

• Reply • Share ›

Load more comments

Subscribe Add Disqus to your siteAdd DisqusAdd Privacy