GeeksQuiz

Computer science mock tests for geeks

Search

- Home
- Latest Questions
- Articles
- C/C++ Programs
- Contribute
- Books

Subscribe

Queue | Set 1(Introduction and Array Implementation)

February 1, 2014

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 **In**First **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
// 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;
};

// function to create a queue of given capacity. It initializes size of
// queue as 0
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;
}

// 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)
{
    if (isEmpty(queue))
        return INT_MIN;
    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;
}
```

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.

Linked list implementation is easier, we will soon be discussing linked list implementation in next article.

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

Related Questions:

- Binary Heap
- Delete all occurrences of a given key in a linked list
- How to create mergable stack?
- Deque | Set 1 (Introduction and Applications)
- A data structure for n elements and O(1) operations
- Convert left-right representation of a bianry tree to down-right
- Print level order traversal line by line
- C Program for Red Black Tree Insertion

Like 5 Tweet 0 +1 0

17 Comments GeeksQuiz Login

Recommend Share Sort by Best

Join the discussion…

nitish • a year 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.

1 • Reply • Share ›

GeeksforGeeks Mod nitish • a year ago

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

Reply • Share ›

Aditya Goel • 4 days 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;
```

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

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

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

2. In function rear()

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

• Reply • Share ›

swasti • 15 days ago
What is the difference with this implementation and circular queue implementation? Please comment?
• Reply • Share ›

Shubham Aggarwal • a month 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 months ago
what does it mean
(queue->front + 1)%queue->capacity;
• Reply • Share ›

Kartik Nagpal ↪ Sujit Roy • a month ago
In above impl. we dequeue from queue->front and enqueue at queue->rear

Lets say i have a queue->array with capacity 7
queue->array : [_ , _ , f , e , k , y , _]
At a given instant,
queue->front = 2
queue->rear = 5

1). Do enqueue('q'):
queue->array : [_ , _ , f , e , k , y , q]
Now,

queue->front = 2 (unchanged)
queue->rear = (5 + 1) % 7 = 6 (this is the position at which 'q' was inserted)

2). Do enqueue('x'):
queue->array : [x , _ , f , e , k , y , q]
Now,
queue->front = 2 (unchanged)
queue->rear = (6 + 1) % 7 = 0 (this is the position at which 'x' was inserted)

see more

• Reply • Share ›

Sean • 2 months ago

How can we impliment queue as array to hold structures. I tried the following code to hold tree nodes in a queue using array but it is giving me segmentation fault in enque operation. Any ideas?

```
/*tree node*/
struct node
{
int val;
struct node* left;
struct node* right;
};

/*function to create new node of a tree*/
struct node* newNode(int data)
{
struct node* new_node=(struct node*) malloc(sizeof(struct node));
new_node->val=data;
new_node->left=NULL;
```

see more

• Reply • Share ›

Ashish Jaiswal • 2 months ago

Without use of queue->size variable

```
#include<stdio.h>

#include<stdlib.h>

#include<limits.h>

typedef struct queue
```

{

int front,rear;

unsigned capacity;

int*array;

}Queue;

Queue*createstack(unsigned capacity)

see more

^ | ⌄ • Reply • Share ›

Ankit Marothi • 5 months ago

Why is there a need to use size? Since, the capacity of the queue will always be equal to the capacity that you fill the structure with, also this should be the right value for the capacity. Why not increase and decrease the value of size rather than increasing it beyond capacity. Also, this would lead to issues like the integer capacity if reached would lead to a round off in which case the maximum allowed operations on the queue will be equal to the maximum limit of integers, after that the program may behave incorrectly.

^ | ⌄ • Reply • Share ›

pratik • 5 months ago

what is the use of header file limits.h here??

^ | ⌄ • Reply • Share ›

Ankit Marothi ➦ pratik • 5 months ago

To get the upper limit and lower limit of each datatype. The definitions of upper limit and lower limits of primitive types is stored in limits.h

^ | ⌄ • Reply • Share ›

Kim Jong-il • 7 months 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.

• Reply • Share ›

Swati • 8 months ago

I think in dequeue operation

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

is wrong and it should be

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

as after removing an element from front, front should point to an element before that . correct me if i am wrong

• Reply • Share ›

Abhishek Kannojia Swati • 8 months ago

No, it is correct. In Dequeue operation element is removed from front of the queue, and the next queue element is after this removed element therefore front should be incremented. See below figure

```
10   20   30   40   50
^                    ^
front             rear

Dequeue()

     20   30   40   50
     ^               ^
     front         rear
```

1 • Reply • Share ›

Swati Abhishek Kannojia • 8 months ago

thanks !!

• Reply • Share ›

typing.. • 9 months ago

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

• Reply • Share ›

Subscribe Add Disqus to your site Privacy

GeeksQuiz

Like

13,852 people like GeeksQuiz.

Facebook social plugin

-
-
- # Categories
    - Articles (126)
        - Algorithms (24)
        - C (17)
        - C++ (17)
        - Data Structures (29)
        - DBMS (1)
        - Interview Experiences (6)
        - Java (2)
        - Operating Systems (1)
        - Puzzle (13)
        - Searching and Sorting (10)
    - Programs (35)
    - Quizzes (2,106)
        - Aptitude (2)
        - Computer Science Quizzes (2,103)
            - Algorithms (147)
            - C (207)
            - C++ (129)
            - Computer Organization and Architecture (1)
            - Data Structures (132)

- DBMS (2)
- GATE (1,406)
- Java (51)
- Operating Systems (28)
- Web technologies (1)

Recent Discussions

- rajeshsethi

  Can we use the concept of "Union of two arrays"...

  Print All Distinct Elements of a given integer array · 6 hours ago

- Siva Krishna

  "The 98'th prisoner decides his answer on the...

  Puzzle 13 | (100 Prisoners with Red/Black Hats) · 11 hours ago

- Thippesh Ks

  Its clearly mentioned that " A set of...

  Check if characters of a given string can be rearranged to form a palindrome · 21 hours ago

- MingWei Jie

  I doubt the iterative version has a bug....

  Binary Search · 1 day ago

- pavithra

  can anyone help me by answering this question...

  Bubble Sort · 1 day ago

- Nitin Maurya

  how 8 is inserted after node 7 ?

  Linked List | Set 2 (Inserting a node) · 1 day ago

Valid XHTML Strict 1.0
Powered by WordPress & MooTools | MiniMoo 1.3.4