

Queue Data Structure

What is a Queue?

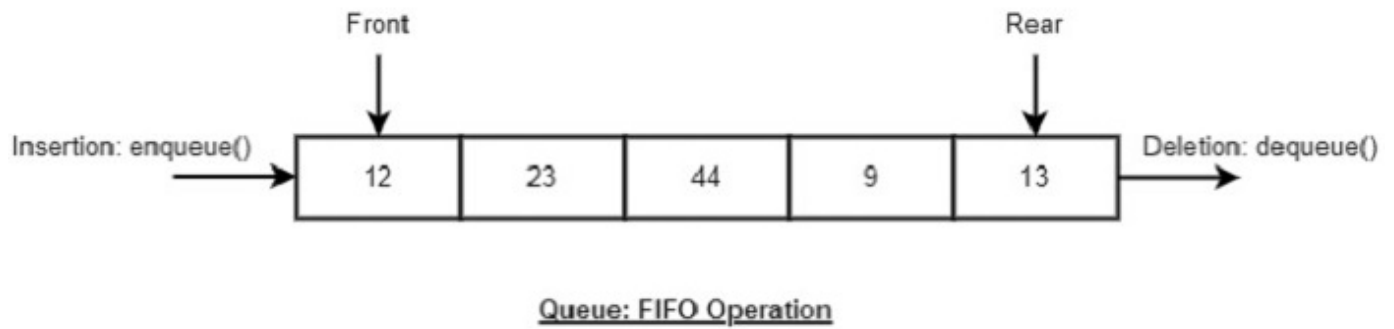
A **queue** is a linear data structure where elements are stored in the FIFO (First In First Out) principle where the first element inserted would be the first element to be accessed. A queue is an Abstract Data Type (ADT) similar to stack, the thing that makes queue different from stack is that a queue is open at both its ends. The data is inserted into the queue through one end and deleted from it using the other end. Queue is very frequently used in most programming languages.



A real-world example of queue can be a single-lane one-way road, where the vehicle enters first, exits first. More real-world examples can be seen as queues at the ticket windows and bus-stops.

Representation of Queues

Similar to the stack ADT, a queue ADT can also be implemented using arrays, linked lists, or pointers. As a small example in this tutorial, we implement queues using a one-dimensional array.



Basic Operations in Queue

Queue operations also include initialization of a queue, usage and permanently deleting the data from the memory.

The most fundamental operations in the queue ADT include: enqueue(), dequeue(), peek(), isFull(), isEmpty(). These are all built-in operations to carry out data manipulation and to check the status of the queue.

Queue uses two pointers – **front** and **rear**. The front pointer accesses the data from the front end (helping in enqueueing) while the rear pointer accesses data from the rear end (helping in dequeuing).

Queue Insertion Operation: Enqueue()

The enqueue() is a data manipulation operation that is used to insert elements into the stack. The following algorithm describes the enqueue() operation in a simpler way.

Algorithm

1. START
2. Check if the queue is full.
3. If the queue is full, produce overflow error and exit.
4. If the queue is not full, increment rear pointer to point the next empty space.
5. Add data element to the queue location, where the rear is pointing.
6. return success.
7. END

Example

Following are the implementations of this operation in various programming languages –

C

C++

Java

Python

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX 6
int intArray[MAX];
int front = 0;
int rear = -1;
int itemCount = 0;
bool isFull(){
    return itemCount == MAX;
}
bool isEmpty(){
    return itemCount == 0;
}
int removeData(){
    int data = intArray[front++];
    if(front == MAX) {
        front = 0;
    }
    itemCount--;
    return data;
}
void insert(int data){
    if(!isFull()) {
        if(rear == MAX-1) {
            rear = -1;
        }
        intArray[++rear] = data;
        itemCount++;
    }
}
```

```
}  
int main(){  
    insert(3);  
    insert(5);  
    insert(9);  
    insert(1);  
    insert(12);  
    insert(15);  
    printf("Queue: ");  
    while(!isEmpty()) {  
        int n = removeData();  
        printf("%d ",n);  
    }  
}
```

Output

Queue: 3 5 9 1 12 15

Queue Deletion Operation: dequeue()

The dequeue() is a data manipulation operation that is used to remove elements from the stack. The following algorithm describes the dequeue() operation in a simpler way.

Algorithm

1. START
2. Check if the queue is empty.
3. If the queue is empty, produce underflow error and exit.
4. If the queue is not empty, access the data where front is pointing.
5. Increment front pointer to point to the next available data element.
6. Return success.
7. END

Example

Following are the implementations of this operation in various programming languages –

C

C++

Java

Python

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX 6
int intArray[MAX];
int front = 0;
int rear = -1;
int itemCount = 0;
bool isFull(){
    return itemCount == MAX;
}
bool isEmpty(){
    return itemCount == 0;
}
void insert(int data){
    if(!isFull()) {
        if(rear == MAX-1) {
            rear = -1;
        }
        intArray[++rear] = data;
        itemCount++;
    }
}
int removeData(){
    int data = intArray[front++];
    if(front == MAX) {
        front = 0;
    }
    itemCount--;
    return data;
}
```

```
}  
int main(){  
    int i;  
  
    /* insert 5 items */  
    insert(3);  
    insert(5);  
    insert(9);  
    insert(1);  
    insert(12);  
    insert(15);  
    printf("Queue: ");  
    for(i = 0; i < MAX; i++)  
        printf("%d ", intArray[i]);  
  
    // remove one item  
    int num = removeData();  
    printf("\nElement removed: %d\n", num);  
    printf("Updated Queue: ");  
    while(!isEmpty()) {  
        int n = removeData();  
        printf("%d ", n);  
    }  
}
```

Output

```
Queue: 3 5 9 1 12 15  
Element removed: 3  
Updated Queue: 5 9 1 12 15
```

Queue - The peek() Operation

The peek() is an operation which is used to retrieve the frontmost element in the queue, without deleting it. This operation is used to check the status of the queue with the help of the pointer.

Algorithm

1. START
2. Return the element at the front of the queue
3. END

Example

Following are the implementations of this operation in various programming languages –

C

C++

Java

Python

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX 6
int intArray[MAX];
int front = 0;
int rear = -1;
int itemCount = 0;
int peek(){
    return intArray[front];
}
bool isFull(){
    return itemCount == MAX;
}
void insert(int data){
    if(!isFull()) {
        if(rear == MAX-1) {
            rear = -1;
        }
        intArray[++rear] = data;
        itemCount++;
    }
}
```

```
int main(){
    int i;

    /* insert 5 items */
    insert(3);
    insert(5);
    insert(9);
    insert(1);
    insert(12);
    insert(15);
    printf("Queue: ");
    for(i = 0; i < MAX; i++)
        printf("%d ", intArray[i]);
    printf("\nElement at front: %d\n", peek());
}
```

Output

Queue: 3 5 9 1 12 15
Element at front: 3

Queue - The isFull() Operation

The isFull() operation verifies whether the stack is full.

Algorithm

1. START
2. If the count of queue elements equals the queue size, return true
3. Otherwise, return false
4. END

Example

Following are the implementations of this operation in various programming languages –

C

C++

Java

Python

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX 6
int intArray[MAX];
int front = 0;
int rear = -1;
int itemCount = 0;
bool isFull(){
    return itemCount == MAX;
}
void insert(int data){
    if(!isFull()) {
        if(rear == MAX-1) {
            rear = -1;
        }
        intArray[++rear] = data;
        itemCount++;
    }
}
int main(){
    int i;
    /* insert 5 items */
    insert(3);
    insert(5);
    insert(9);
    insert(1);
    insert(12);
    insert(15);
    printf("Queue: ");
    for(i = 0; i < MAX; i++)
        printf("%d ", intArray[i]);
    printf("\n");
}
```

```
if(isFull()) {  
    printf("Queue is full!\n");  
}  
}
```

Output

```
Queue: 3 5 9 1 12 15  
Queue is full!
```

Queue - The isEmpty() operation

The isEmpty() operation verifies whether the stack is empty. This operation is used to check the status of the stack with the help of top pointer.

Algorithm

1. START
2. If the count of queue elements equals zero, return true
3. Otherwise, return false
4. END

Example

Following are the implementations of this operation in various programming languages –

C

C++

Java

Python

```
#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>  
#include <stdbool.h>  
#define MAX 6  
int intArray[MAX];  
int front = 0;
```

```
int rear = -1;
int itemCount = 0;
bool isEmpty(){
    return itemCount == 0;
}
int main(){
    int i;
    printf("Queue: ");
    for(i = 0; i < MAX; i++)
        printf("%d ", intArray[i]);
    printf("\n");
    if(isEmpty()) {
        printf("Queue is Empty!\n");
    }
}
```

Output

```
Queue: 0 0 0 0 0 0
Queue is Empty!
```

Queue Complete Implementation

Following are the complete implementations of Queue in various programming languages –

C

C++

Java

Python

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX 6
int intArray[MAX];
int front = 0;
int rear = -1;
int itemCount = 0;
```

```
int peek(){
    return intArray[front];
}
bool isEmpty(){
    return itemCount == 0;
}
bool isFull(){
    return itemCount == MAX;
}
int size(){
    return itemCount;
}
void insert(int data){
    if(!isFull()) {
        if(rear == MAX-1) {
            rear = -1;
        }
        intArray[++rear] = data;
        itemCount++;
    }
}
int removeData(){
    int data = intArray[front++];
    if(front == MAX) {
        front = 0;
    }
    itemCount--;
    return data;
}
int main(){

    /* insert 5 items */
    insert(3);
    insert(5);
    insert(9);
    insert(1);
    insert(12);
    insert(15);
    printf("Queue size: %d", size());
```

```
printf("\nQueue: ");
for(int i = 0; i < MAX; i++){
    printf("%d ", intArray[i]);
}
if(isFull()) {
    printf("\nQueue is full!");
}

// remove one item
int num = removeData();
printf("\nElement removed: %d", num);
printf("\nSize of Queue after deletion: %d", size());
printf("\nElement at front: %d", peek());
}
```

Output

Queue size: 6
Queue: 3 5 9 1 12 15
Queue is full!
Element removed: 3
Size of Queue after deletion: 5
Element at front: 5

Queue Implementation in C

Click to check the implementation of [Queue Program using C](#)