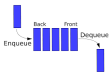


Queue

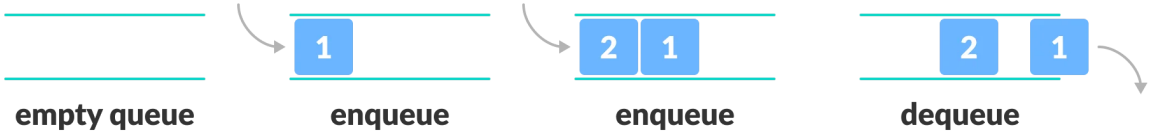
Queue is an abstract linear data structure in which operations are performed in a particular manner

The Manner in which operations are performed in queue is First in First Out (FIFO) or first come first served.

A queue can be implemented using array as well as using Linked List.

Queues provides 4 basic operations for interaction

1. `enqueue()-` Adds an element at the end of queue
2. `dequeue()` - Removes an element from from to the queue
3. `isEmpty()` - Check if queue is empty or not.
4. `peek()` - Return the element present at the front of queue



Why & When Queue??

A queue is used when we want to manage things in a first come first serve basis

Some real life usage of queues are as follow

- Serving request on a single shared resource like a printer, CPU Task scheduling etc.
- All types of customer service (like railway reservation) centers software to hold people and calling them in order.
- Buffer for a device like keyboard

Working of a Queue

- two pointers `front` and `back`
- `front` track the first element of the queue
- `back` track the last element of the queue
- initially set value of `front` and `back` as -1

Enqueue

- check if the queue is full
- for the first element, increment `front` to 0
- increase `back` by 1
- add new element in position pointed by `back`

Dequeue

- check if queue is empty
- increase `front` by 1

Peek

- check if queue is empty
- return `arr[front]`

Empty

- check if `front = -1` or `front>back` and return the true or false accordingly

Pros & Cons of Queue

Aa Pros	Cons

Pros	Cons
Does not take much effort to add a new element as compared to array	You have no access to underneath element, only element present on top and bottom is accessible
In Queue we can allocate memory dynamically	Because of dynamic memory allocation if we not use all memory space then there will be wastage of memory and we'll suffer memory leak.
Can handle multiple clients at a time as we can insert from one end and remove from another	Lookup is slow

Time complexity for Queue

Operation	Complexity	Explanation
Untitled		
Enque (Insertion)	O(1)	We just have to push element at the end of Queue or LL which is a 1 step process
Deque (Deletion)	O(1)	We just have to remove element at the front of Queue or LL which is a 1 step process
Top (Get Front)	O(1)	We have to return the value of element present at top which is a one step process
Searching	O(N)	As we can access only the topmost and bottom most element of queue so we have to traverse whole queue to search a particular element

Code Implementation

Implementing a Queue using array

```
#include <iostream>

using namespace std;

#define n 100

class queue{
    int* arr;
    int front;
    int back;

    public:
    queue(){
```

```
        arr = new int[n];
        front = -1;
        back = -1;
    }

    void push(int x){
        if(back == n-1){
            cout<<"Queue Overflow\n";
            return;
        }
        back++;
        arr[back] = x;

        if(front == -1){
            front++;
        }
    }

    void pop(){
        if(front == -1 || front>back){
            cout<<"Queue Underflow\n";
            return;
        }

        front++;
    }

    int peek(){
        if(front == -1 || front>back){
            cout<<"Queue is empty\n";
            return -1;
        }
        return arr[front];
    }

    bool empty(){
        return (front == -1 || front>back);
    }

};

int main(){

    queue q;
    q.push(1);
    q.push(2);
    q.push(3);

    cout<<q.peek()<<endl;
    q.pop();

    cout<<q.peek()<<endl;
    q.pop();

    cout<<q.peek()<<endl;
    q.pop();

    cout<<q.peek()<<endl;
    q.pop();

    cout<<q.empty()<<endl;
```

```
    return 0;
}
```

Output:

```
1
2
3
Queue is empty
-1
Queue Underflow
1
```

Implementing a Queue using Linked List

```
#include <iostream>

using namespace std;

class node{
    public:
    int data;
    node* next;

    node(int x){
        data = x;
        next = NULL;
    }
};

class queue{
    node* front;
    node* back;

    public:
    queue(){
        front=NULL;
        back=NULL;
    }

    void push(int x){
        node* n = new node(x);

        if(front == NULL){
            back = n;
            front = n;
            return;
        }

        back->next = n;
        back = n;
    }

    void pop(){
        if(front==NULL){
            cout<<"Queue Underflow\n";
```

```
            return;
        }

        node* todelete = front;
        front = front->next;
        delete todelete;
    }

    int peek(){
        if(front==NULL){
            cout<<"Queue is empty\n";
            return -1;
        }

        return front->data;
    }

    bool empty(){
        return front == NULL;
    }

};

int main(){

    queue q;
    q.push(1);
    q.push(2);
    q.push(3);

    cout<<q.peek()<<endl;
    q.pop();

    cout<<q.peek()<<endl;
    q.pop();

    cout<<q.peek()<<endl;
    q.pop();

    cout<<q.peek()<<endl;
    q.pop();

    cout<<q.empty()<<endl;

    return 0;
}
```

Output:

```
1
2
3
Queue is empty
-1
Queue Underflow
1
```

Using Inbuilt Queue Data Structure from STL

Syntax: `queue <int> queue_name;`

Some functions of Queue:

- `push()` Inserts an element (at end) `O(1)`
- `pop()` Removes/delete an element (from front) `O(1)`
- `front()` Acess the element at the front of queue `O(1)`
- `empty()` Checks if queue is empty or not `O(1)`
- `size()` Returns the size of queue `O(1)`

```
#include <bits/stdc++.h>
using namespace std;

int main() {

  queue <int> que_name; // Decleare a queue

  for (int i = 0; i < 5; i++) {//This loop takes input and adds it to queue (at end)
    int temp; cin >> temp;
    que_name.push(temp);
  }

//This prints the element of queue and clears the queue at the end of loop
  while (!que_name.empty()) {
    cout << que_name.front() << " ";
    que_name.pop();
  }

  return 0;
}
```

Basic Problems:

Reverse A queue

Queue Reversal | Practice | GeeksforGeeks

Given a Queue Q containing N elements. The task is to reverse the Queue. Your task is to complete the function rev(), that reverses the N elements of the queue. Example 1: Input: 6 4 3 1 10 2 6 Output: 6

https://practice.geeksforgeeks.org/problems/queue-reversal/1#

Approach:

We will use stack to reverse the queue as stack follows first in last out principle so it indirectly reverse any given input

- Iterate over the given queue and push all of its elements in a temp stack
- Then iterate over the stack and push all of its element in an temp queue and return this temp queue

```
queue<int> rev(queue<int> q)
{
    // Temp Stack
    stack <int> sta;

    //Iterating over the queue and pushing all of its element in stack
    while(!q.empty())
    {
      sta.push(q.front());
      q.pop();
    }

    queue<int> temp;

//Iterating over the stack and pushing all of its element in a temp queue
    while(!sta.empty())
    {
      temp.push(sta.top());
      sta.pop();
    }

    // returning temp queue
    return temp;
}
```

Queue using Stacks - GeeksforGeeks

The problem is opposite of this post. We are given a stack data structure with push and pop operations, the task is to implement a queue using instances of stack data structure and operations on

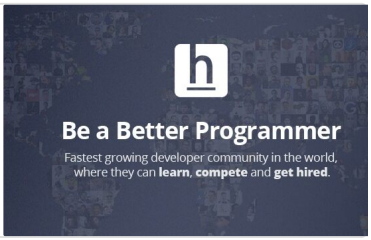
https://www.geeksforgeeks.org/queue-using-stacks/

CP Problems:

Basics of Queues Practice Problems Data Structures | HackerEarth

Solve practice problems for Basics of Queues to test your programming skills. Also go through detailed tutorials to improve your understanding to the topic. | page 1

https://www.hackerearth.com/practice/data-structures/queues/basics-of-queues/practice-problems/



Sliding Window Maximum - LeetCode

Level up your coding skills and quickly land a job. This is the best place to expand your knowledge and get prepared for your next interview.

https://leetcode.com/problems/sliding-window-maximum/