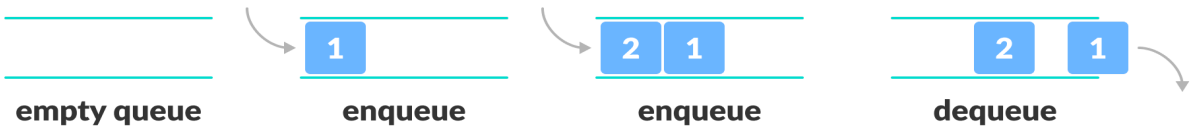
QUEUE

A queue is a useful data structure in programming. It is similar to the ticket queue outside a cinema hall, where the first person entering the queue is the first person who gets the ticket.

Queue follows the First In First Out (FIFO) rule - the item that goes in first is the item that comes out first.



FIFO Representation of Queue

In the above image, since 1 was kept in the queue before 2, it is the first to be removed from the queue as well. It follows the FIFO rule.

In programming terms, putting items in the queue is called enqueue, and removing items from the queue is called dequeue.

Basic Operations of Queue

A queue is an object (an abstract data structure - ADT) that allows the following operations:

- **Enqueue:** Add an element to the end of the queue
- **Dequeue:** Remove an element from the front of the queue
- **IsEmpty:** Check if the queue is empty
- **IsFull:** Check if the queue is full
- **Peek:** Get the value of the front of the queue without removing it

---

Working of Queue

Queue operations work as follows:

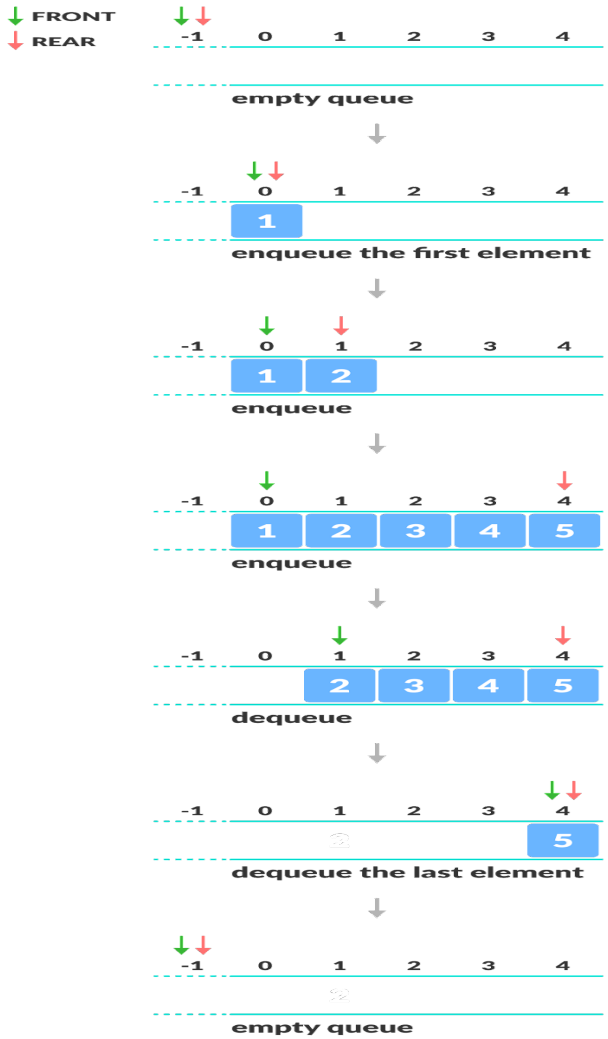
- two pointers FRONT and REAR
- FRONT track the first element of the queue
- REAR track the last element of the queue
- initially, set value of FRONT and REAR to -1

Enqueue Operation

- check if the queue is full
- for the first element, set the value of FRONT to 0
- increase the REAR index by 1
- add the new element in the position pointed to by REAR

Dequeue Operation

- check if the queue is empty
- return the value pointed by FRONT
- increase the FRONT index by 1
- for the last element, reset the values of FRONT and REAR to -1



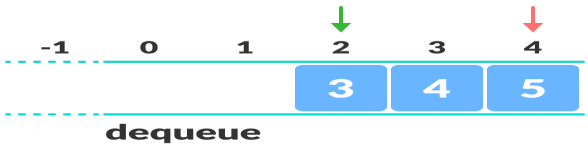
Enqueue and Dequeue Operations

Queue Implementations in C++

```
#include <iostream>
#define SIZE 5

using namespace std;

class Queue {
   private:
  int items[SIZE], front, rear;

   public:
  Queue() {
    front = -1;
    rear = -1;
  }

  bool isFull() {
  if (front == 0 && rear == SIZE - 1)
{
      return true;
    }
    return false;
  }

  bool isEmpty() {
    if (front == -1)
      return true;
    else
      return false;
  }

  void enQueue(int element) {
    if (isFull()) {
      cout << "Queue is full";
    } else {
      if (front == -1) front = 0;
      rear++;
      items[rear] = element;
      cout << endl
         << "Inserted " << element <<
endl;
    }
  }

  int deQueue() {
    int element;
    if (isEmpty()) {
      cout << "Queue is empty" <<
endl;
      return (-1);
    }
          else {
      element = items[front];
      if (front >= rear) {
         front = -1;
        rear = -1;
      }      else {
        front++;
      }
      cout<<endl<< "Deleted -> " <<
element<<endl;
      return (element);
    }
  }

  void display() {
        int i;
    if (isEmpty()) {
  cout<<endl<< "Empty Queue" << endl;
    } else {
      cout << endl<< "Front index-> "
<< front;
      cout<<endl<< "Items -> ";
      for (i = front; i <= rear; i++)
       cout << items[i] << "  ";
      cout<<endl<< "Rear index-> " <<
rear << endl;
    }
  }
};

int main() {
  Queue q;
  q.deQueue();
  q.enQueue(1);
  q.enQueue(2);
  q.enQueue(3);
  q.enQueue(4);
  q.enQueue(5);
  q.enQueue(6);
  q.display();
  q.deQueue();
  q.display();
  return 0;
}
```

Limitations of Queue

As you can see in the image below, after a bit of enqueuing and dequeuing, the size of the queue has been reduced.



Limitation of a queue

And we can only add indexes 0 and 1 only when the queue is reset (when all the elements have been dequeued).

After REAR reaches the last index, if we can store extra elements in the empty spaces (0 and 1), we can make use of the empty spaces. This is implemented by a modified queue called the circular queue.

Complexity Analysis

The complexity of enqueue and dequeue operations in a queue using an array is O(1). If you use pop(N) in python code, then the complexity might be O(n) depending on the position of the item to be popped.

Applications of Queue

- **CPU scheduling, Disk Scheduling**
- **When data is transferred asynchronously between two processes.The queue is used for synchronization. For example: IO Buffers, pipes, file IO, etc**
- **Handling of interrupts in real-time systems.**
- **Call Center phone systems use Queues to hold people calling them in order.**

Type of queue :

A queue is a useful data structure in programming. It is similar to the ticket queue outside a cinema hall, where the first person entering the queue is the first person who gets the ticket.

There are four different types of queues:

- **Simple Queue**

- Circular Queue
- Priority Queue
- Double Ended Queue

Simple Queue :

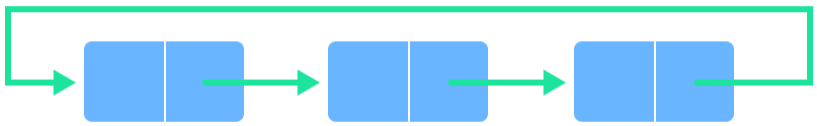
In a simple queue, insertion takes place at the rear and removal occurs at the front. It strictly follows the FIFO (First in First out) rule.

Simple Queue Representation

To learn more, visit Queue Data Structure.

Circular Queue :

In a circular queue, the last element points to the first element making a circular link.



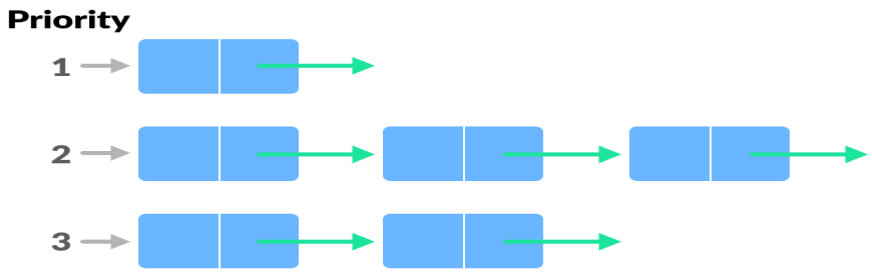
Circular Queue Representation

The main advantage of a circular queue over a simple queue is better memory utilization. If the last position is full and the first position is empty, we can insert an element in the first position. This action is not possible in a simple queue.

To learn more, visit Circular Queue Data Structure.

Priority Queue :

A priority queue is a special type of queue in which each element is associated with a priority and is served according to its priority. If elements with the same priority occur, they are served according to their order in the queue.



Priority Queue Representation

Insertion occurs based on the arrival of the values and removal occurs based on priority.

To learn more, visit Priority Queue Data Structure.

Deque (Double Ended Queue) :

In a double ended queue, insertion and removal of elements can be performed from either from the front or rear. Thus, it does not follow the FIFO (First In First Out) rule.