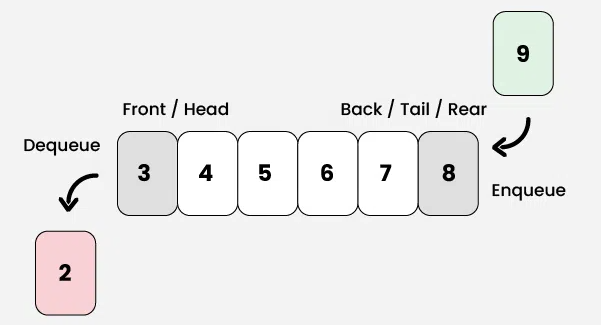
**Введение в структуру данных очереди**

**Структура данных очереди**представляет собой [линейную структуру данных](https://translated.turbopages.org/proxy_u/en-ru.ru.f76bfd99-674ad186-d67c5965-74722d776562/https/www.geeksforgeeks.org/introduction-to-linear-data-structures) которая следует**Принцип FIFO (First In First Out)**, таким образом, первый вставленный элемент является первым, который будет извлечен.



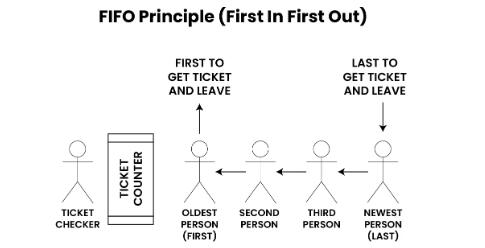
## **Что такое структура данных очереди?**

**Структура данных очереди**- это [линейная структура данных,](https://translated.turbopages.org/proxy_u/en-ru.ru.f76bfd99-674ad186-d67c5965-74722d776562/https/www.geeksforgeeks.org/introduction-to-linear-data-structures) которая открыта с обоих концов и операции выполняются в [порядке поступления первым (FIFO)](https://translated.turbopages.org/proxy_u/en-ru.ru.f76bfd99-674ad186-d67c5965-74722d776562/https/www.geeksforgeeks.org/fifo-first-in-first-out-approach-in-programming).

Мы определяем очередь как список, в котором все добавления в список производятся на одном конце (**в конце очереди**), а все удаления из списка производятся на другом конце (**в начале очереди**). Над элементом, который первым помещается в порядок, сначала выполняется операция удаления.

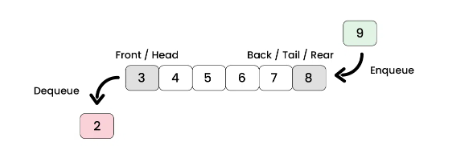
### [**Принцип структуры данных очереди FIFO:**](https://translated.turbopages.org/proxy_u/en-ru.ru.f76bfd99-674ad186-d67c5965-74722d776562/https/www.geeksforgeeks.org/fifo-first-in-first-out-approach-in-programming)

* Очередь похожа на очередь, ожидающую покупки билетов, где первый человек в очереди - это первый обслуженный. (т. е. Первый пришедший первым обслужен).
* Позиция записи в очереди, готовой к обслуживанию, то есть первая запись, которая будет удалена из очереди, называется **передняя** из очереди (иногда, **head** из очереди). Аналогично, позиция последней записи в очереди, то есть добавленной последней, называется **задней** (или**хвост**) очереди.



## **Представление структуры данных очереди:**

На рисунке ниже показано, как мы представляем структуру данных очереди:



## [**Типы структуры данных очереди:**](https://translated.turbopages.org/proxy_u/en-ru.ru.f76bfd99-674ad186-d67c5965-74722d776562/https/www.geeksforgeeks.org/different-types-of-queues-and-its-applications)

Структуру данных очереди можно разделить на 4 типа:

Существуют различные типы очередей:

1. **Простая очередь:** Простая очередь просто следует **FIFO**Структура. Мы можем вставить элемент только в конце и удалить элемент из начала очереди.
2. [**Двусторонняя очередь (Dequeue)**](https://translated.turbopages.org/proxy_u/en-ru.ru.f76bfd99-674ad186-d67c5965-74722d776562/https/www.geeksforgeeks.org/deque-set-1-introduction-applications)**:** В двусторонней очереди операции вставки и удаления могут выполняться с обоих концов. Они бывают двух типов:
   * **Очередь с ограниченным вводом:**Это простая очередь. В очереди этого типа входные данные могут быть получены только с одного конца, но удаление может быть выполнено с любого из концов.
   * **Очередь с ограниченным выводом:** Это тоже простая очередь. В очереди этого типа входные данные могут приниматься с обоих концов, но удаление может выполняться только с одного конца.
3. [**Циклическая очередь:**](https://translated.turbopages.org/proxy_u/en-ru.ru.f76bfd99-674ad186-d67c5965-74722d776562/https/www.geeksforgeeks.org/introduction-to-circular-queue)Это особый тип очереди, в которой последняя позиция соединяется обратно с первой позицией. Здесь также операции выполняются в порядке FIFO.
4. [**Приоритетная очередь**](https://translated.turbopages.org/proxy_u/en-ru.ru.f76bfd99-674ad186-d67c5965-74722d776562/https/www.geeksforgeeks.org/priority-queue-set-1-introduction)**:** Приоритетная очередь - это специальная очередь, доступ к элементам в которой осуществляется на основе присвоенного им приоритета. Они бывают двух типов:
   * **Очередь с возрастающим приоритетом:**В очереди с возрастающим приоритетом элементы располагаются в порядке возрастания их значений приоритета. Элемент с наименьшим значением приоритета выбирается первым.
   * **Очередь с приоритетом по убыванию:**В очереди с приоритетом по убыванию элементы располагаются в порядке убывания их значений приоритета. Элемент с наибольшим приоритетом выбирается первым.

# 

## [**Основные операции в структуре данных очереди:**](https://translated.turbopages.org/proxy_u/en-ru.ru.f76bfd99-674ad186-d67c5965-74722d776562/https/www.geeksforgeeks.org/basic-operations-for-queue-in-data-structure)

Некоторые из основных операций для очереди в структуре данных следующие:

1. **Постановка в очередь (Enqueue):** Добавляет (или сохраняет) элемент в конец очереди.
2. **Удаление из очереди (Dequeue):** Удаление элементов из очереди.
3. **Просмотр или передняя часть (Peek or front):** Получает элемент данных, доступный на переднем узле очереди, без его удаления.
4. **rear:** Эта операция возвращает элемент в конце очереди, не удаляя его.
5. **Заполнено (isFull)**: проверяет, заполнена ли очередь.
6. **isEmpty**: Проверяет, пуста ли очередь.

Существует несколько вспомогательных операций (auxiliary operations):

## **1. Операция постановки в очередь в структуре данных очереди:**

Операция Enqueue() в очереди **добавляет (или сохраняет) элемент в конец очереди**.  
Для постановки (вставки) данных в очередь необходимо предпринять следующие шаги:

* **Шаг 1:** Проверьте, заполнена ли очередь.
* **Шаг 2:**Если очередь заполнена, верните ошибку переполнения и завершите работу.
* **Шаг 3:** Если очередь не заполнена, увеличьте задний указатель, чтобы он указывал на следующее пустое место.
* **Шаг 4:** Добавьте элемент данных в расположение очереди, куда указывает конец.
* **Шаг 5:** возврат завершен успешно.

### Реализация постановки в очередь:

void queueEnqueue(int data)

{

*// Check queue is full or not*

**if** (capacity == rear) {

printf("**\n**Queue is full**\n**");

**return**;

}

*// Insert element at the rear*

**else** {

queue[rear] = data;

rear++;

}

**return**;

}

### Реализация постановки в очередь:

## **2. Операция удаления из очереди в структуре данных очереди:**

Удаляет (или получает доступ) первый элемент из очереди.  
Для выполнения операции удаления из очереди выполняются следующие шаги:

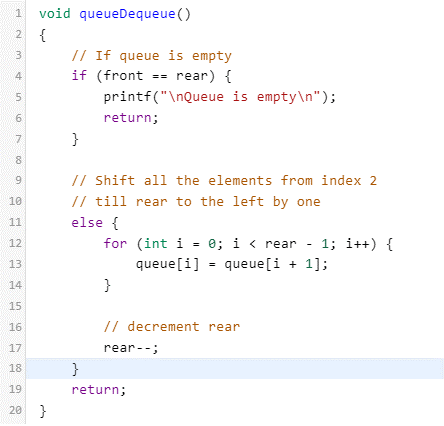
* **Шаг 1:** Проверьте, пуста ли очередь.
* **Шаг 2:** Если очередь пуста, верните ошибку underflow и завершите работу.
* **Шаг 3:** Если очередь не пуста, получите доступ к данным там, где указано начало.
* **Шаг 4:** Увеличьте передний указатель, чтобы он указывал на следующий доступный элемент данных.
* **Шаг 5:** Успешный возврат.

## **2. Операция удаления из очереди в структуре данных очереди:**

Удаляет (или получает доступ) первый элемент из очереди.  
Для выполнения операции удаления из очереди выполняются следующие шаги:

* **Шаг 1:** Проверьте, пуста ли очередь.
* **Шаг 2:** Если очередь пуста, верните ошибку underflow и завершите работу.
* **Шаг 3:** Если очередь не пуста, получите доступ к данным там, где указано начало.
* **Шаг 4:** Увеличьте передний указатель, чтобы он указывал на следующий доступный элемент данных.
* **Шаг 5:** Успешный возврат.

### Реализация удаления из очереди:



## **3. Начальная операция в структуре данных очереди:**

Эта операция возвращает элемент во внешнем интерфейсе, не удаляя его.

// Function to get front of queue

int front(Queue\* queue)

{

if (isempty(queue))

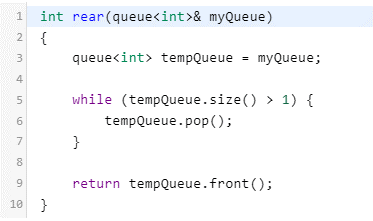
return INT\_MIN;

return queue->arr[queue->front];

}

## **4. Обратная операция в структуре данных очереди:**

Эта операция возвращает элемент в конце, не удаляя его.



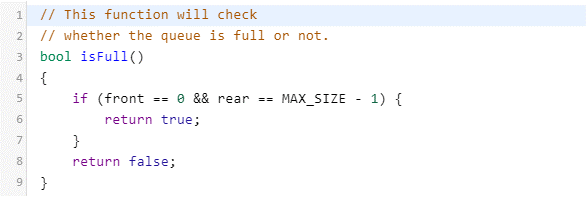
## **5. Операция isEmpty в структуре данных очереди:**

Эта операция возвращает логическое значение, указывающее, пуста очередь или нет.

## 

## **6. Правильная работа в структуре очереди:**

Эта операция возвращает логическое значение, указывающее, заполнена очередь или нет.



## Реализация структуры данных очереди:

Очередь может быть реализована с использованием следующих структур данных:

* [Реализация очереди с использованием массивов](https://translated.turbopages.org/proxy_u/en-ru.ru.f76bfd99-674ad186-d67c5965-74722d776562/https/www.geeksforgeeks.org/introduction-and-array-implementation-of-queue)
* [Реализация очереди с использованием связанного списка](https://translated.turbopages.org/proxy_u/en-ru.ru.f76bfd99-674ad186-d67c5965-74722d776562/https/www.geeksforgeeks.org/queue-linked-list-implementation)

Ниже мы обсудили реализацию Queue:

// Implementation of queue(enqueue, dequeue).

#include <bits/stdc++.h>

using namespace std;

class Queue {

public:

int front, rear, size;

unsigned cap;

int\* arr;

};

Queue\* createQueue(unsigned cap)

{

Queue\* queue = new Queue();

queue->cap = cap;

queue->front = queue->size = 0;

queue->rear = cap - 1;

queue->arr = new int[(queue->cap \* sizeof(int))];

return queue;

}

int isFull(Queue\* queue)

{

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

}

int isempty(Queue\* queue) { return (queue->size == 0); }

// Function to add an item to the queue.

// It changes rear and size.

void enqueue(Queue\* queue, int item)

{

if (isFull(queue))

return;

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

queue->arr[queue->rear] = item;

queue->size = queue->size + 1;

cout << item << " enqueued to queue\n";

}

// Function to remove an item from queue.

// It changes front and size

int dequeue(Queue\* queue)

{

if (isempty(queue))

return INT\_MIN;

int item = queue->arr[queue->front];

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

queue->size = queue->size - 1;

return item;

}

int front(Queue\* queue)

{

if (isempty(queue))

return INT\_MIN;

return queue->arr[queue->front];

}

int rear(Queue\* queue)

{

if (isempty(queue))

return INT\_MIN;

return queue->arr[queue->rear];

}

// Driver code

int main()

{

Queue\* queue = createQueue(1000);

enqueue(queue, 10);

enqueue(queue, 20);

enqueue(queue, 30);

enqueue(queue, 40);

cout << dequeue(queue);

cout << " dequeued from queue\n";

cout << "Front item is " << front(queue) << endl;

cout << "Rear item is " << rear(queue);

return 0;

}

**Вывод**

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

## Анализ сложности операций со структурой данных очереди:

| **Операции** | **Временная сложность** | **Сложность пространства** |
| --- | --- | --- |
| **Поставить в очередь** | O(1) | O(1) |
| **Удаление из очереди** | O(1) | O(1) |
| **Фронтальная часть** | O(1) | O(1) |
| **Назад** | O(1) | O(1) |
| **Пусто** | O(1) | O(1) |
| **Полностью** | O(1) | O(1) |

## [**Приложения структуры данных очереди:**](https://translated.turbopages.org/proxy_u/en-ru.ru.f76bfd99-674ad186-d67c5965-74722d776562/https/www.geeksforgeeks.org/applications-of-queue-data-structure)

Применение очереди является обычным явлением. В компьютерной системе могут быть очереди задач, ожидающих принтера, доступа к дисковому хранилищу или даже в системе с разделением времени для использования центрального процессора. В рамках одной программы может быть несколько запросов, которые необходимо сохранять в очереди, или одна задача может создавать другие задачи, которые необходимо выполнять по очереди, сохраняя их в очереди.

* Очередь может использоваться при планировании заданий, например, при буферизации принтера.
* Очередь может использоваться там, где у нас есть один ресурс и несколько потребителей.
* В сети очередь используется в таких устройствах, как маршрутизатор / коммутатор и почтовая очередь.
* Очередь может использоваться в различных алгоритмах, таких как поиск по ширине, топологическая сортировка и т.д.

## Преимущества структуры данных очереди:

* Большим объемом данных можно эффективно и с легкостью управлять.
* Такие операции, как вставка и удаление, могут выполняться с легкостью, поскольку это соответствует правилу "первый зашел - первый вышел".
* Очереди полезны, когда определенной услугой пользуются несколько потребителей.
* Очереди обеспечивают высокую скорость обмена данными между процессами.
* Очереди могут использоваться при реализации других структур данных.

## Недостатки структуры данных очереди:

* Такие операции, как вставка и удаление элементов из середины, отнимают много времени.
* Поиск элемента занимает O (N) времени.
* Максимальный размер очереди должен быть определен заранее в случае реализации массива.