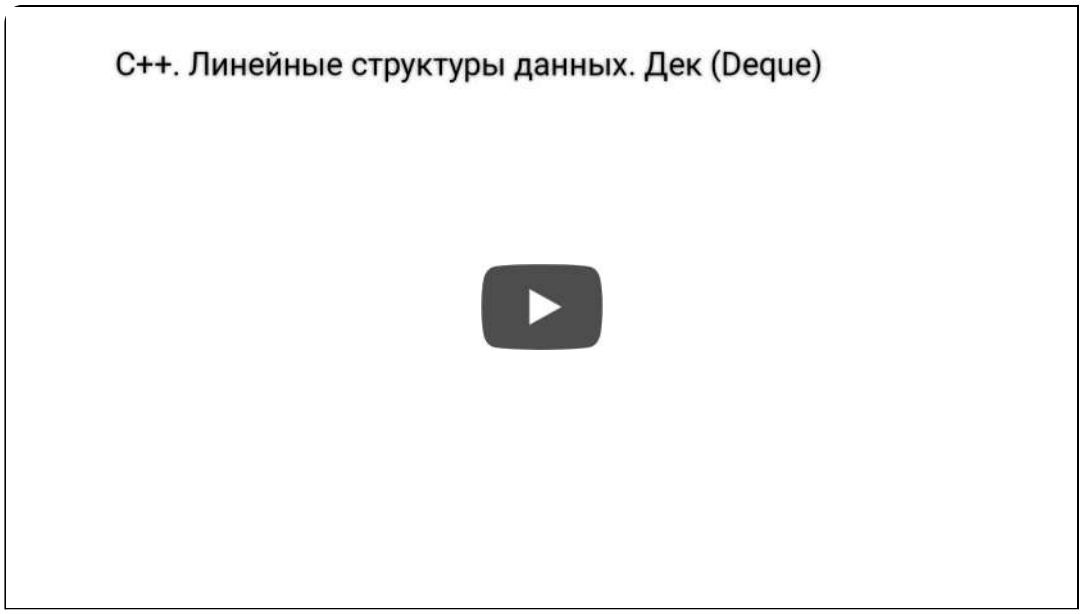
Дек



Дек

Если есть стек, в который элементы кладутся в один конец и извлекаются с того же конца, и очередь, в которой элементы кладутся в один конец, а извлекаются с другого конца, то логичным кажется существование структуры данных, которая называется дек, по английски deque или double-ended queue.

Дек поддерживает следующие операции:

- Добавить элемент в начало дека
- Добавить элемент в конец дека
- Удалить элемент из начала дека
- Удалить элемент из конца дека
- Узнать значение последнего элемента дека
- Узнать значение первого элемента дека
- Узнать размер дека

Операции получения значения элемента и удаления элемента могут быть совмещены. Все операции должны выполняться за $O(1)$ или в среднем за $O(1)$.

Дек можно реализовать так же, как и очередь, — на закольцованном массиве, но необходимо учесть, что при добавлении элементов дек может ползти в любом из двух направлений, поэтому реализация более сложна, чем реализация очереди.

Для реализации на C++ существует специальный контейнер `deque`. Он находится в

библиотеке `deque`.

Приведём пример работы с деком:

```
#include <deque>
#include <iostream>
using namespace std;
int main()
{
    deque<int> D;
    D.push_back(5);
    D.push_front(4);
    cout << D.size() << " " << D.front() << " " << D.back() << endl; // 2 4 5
    D.push_front(6);
    D.pop_back();
    cout << D.size() << " " << D.front() << " " << D.back() << endl; // 2 6 4
    return 0;
}
```

У дека в C++ есть следующие операции:

- `push_back()` — добавить элемент в конец дека
- `push_front()` — добавить элемент в начало дека
- `pop_back()` — удалить элемент из конца дека
- `pop_front()` — удалить элемент из начала дека
- `back()` — узнать значение последнего элемента дека
- `front()` — узнать значение первого элемента дека
- `size()` — узнать размер дека
- `empty()` — проверка дека на пустоту
- `clear()` — очистить дек

Интересной особенностью реализации дека в C++ является то, что вы можете обратиться к элементу дека по индексу, как к элементам массива. Например, если в деке d будет 5 элементов, то к ним можно обращаться как $d[0], d[1], \ldots, d[4]$. Это обращение будет выполняться за $O(1)$, но будет чуть медленнее, чем обращение к элементам вектора или массива. На практике этот же контейнер можно рекомендовать и для работы с очередью — собственный контейнер для работы с очередью реализован именно через дек.