11. Deque, implementation using std::vector/std::list, Implementation with circular buffer and the corresponding implementation in STL, Complexity analysis for Deque.

> **Дек** — это структура данных, представляющая собой двустороннюю очередь. Она позволяет добавлять и удалять элементы как с начала, так и с конца, обеспечивая гибкость в работе с данными.

Ключевые свойства дека:

- Дек поддерживает операции *вставки и удаления* с обоих концов.
- Может использоваться для задач, требующих управления *очередями с обоих концов*

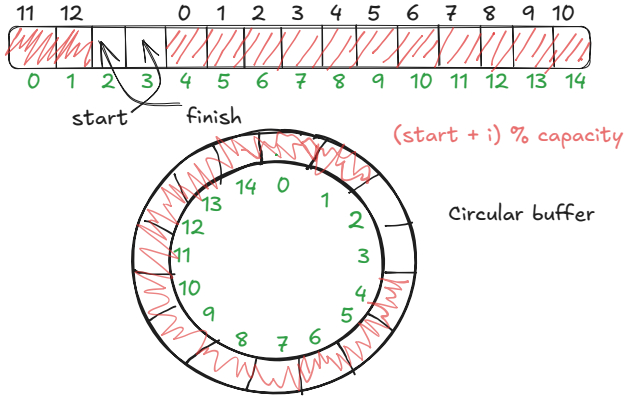
Типы операций над деком:

- **Добавление элементов**:
  - `push_front(value)` — добавить элемент в начало.
  - `push_back(value)` — добавить элемент в конец.
- **Удаление элементов**:
  - `pop_front()` — удалить элемент из начала.
  - `pop_back()` — удалить элемент с конца.
- **Просмотр элементов**:
  - `front()` — возвращает элемент в начале дека.
  - `back()` — возвращает элемент в конце дека.

Реализация дека:

Дек можно реализовать на основе разных структур данных:

1. **На основе массива** (`std::vector`):
   Удобен для реализации, но вставка/удаление в начало требует сдвига элементов, что делает операции с началом дека менее эффективными (`O(n)`).
   -> Использование циклического буфера устраняет эту проблему, позволяя использовать `O(1)` для вставки и удаления в обоих концах.



- Зелёным отмечены индексы вектора - настоящие, чёрным -дека.
- У дека есть итераторы на начало и конец (указывает на элемент после последнего).
- Для получения `i` -ого зеленого индекса: `(start(зеленый) + i (чёрный)) % capacity`.
- Когда `finish` достигает `start`, необходимо делать реалокацию памяти (сделать её, допустим, в 2 раза больше).

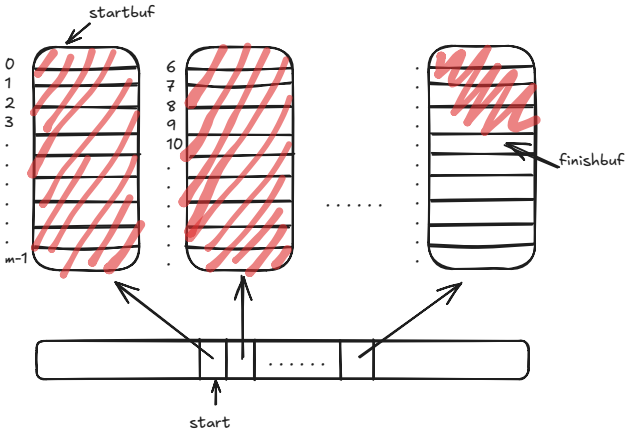
Визуально,то из себя представляет circular buffer - начало соединено с концом. Амортизировано `push_back` и `push_front` работают за `O(1)`.

```
#include <vector>
#include <iostream>

class DequeVector {
private:
    std::vector<int> data;
public:
    void push_front(int value) {
        data.insert(data.begin(), value); // O(n)
    }
    void push_back(int value) {
        data.push_back(value); // O(1)
    }
    void pop_front() {
        if (!data.empty()) {
            data.erase(data.begin()); // O(n)
        }
    }
    void pop_back() {
        if (!data.empty()) {
            data.pop_back(); // O(1)
        }
    }
    int front() const {
        return data.front();
    }
    int back() const {
        return data.back();
    }
};

int main() {
    DequeVector dq;
    dq.push_back(10);
    dq.push_front(5);
    dq.push_back(20);
    std::cout << "Front: " << dq.front() << ", Back: " << dq.back() << std::endl;
    dq.pop_front();
    dq.pop_back();
    std::cout << "Front after pop: " << dq.front() << std::endl;
    return 0;
}
```

2.**На основе двусвязного списка** (`std::list`):
- *Вставка и удаление* выполняются за *O(1)* как с начала, так и с конца, так как в списке не нужно сдвигать элементы.
- Однако *доступ по индексу* в списке менее эффективен (*O(n)*).

```
#include <list>
#include <iostream>

class DequeList {
private:
    std::list<int> data;
```

```
public:
    // Push an element to the front
    void push_front(int value) {
        data.push_front(value); // O(1)
    }

    // Push an element to the back
    void push_back(int value) {
        data.push_back(value); // O(1)
    }

    // Pop an element from the front
    void pop_front() {
        if (!data.empty()) {
            data.pop_front(); // O(1)
        }
    }

    // Pop an element from the back
    void pop_back() {
        if (!data.empty()) {
            data.pop_back(); // O(1)
        }
    }

    // Get the front element
    int front() const {
        return data.front();
    }

    // Get the back element
    int back() const {
        return data.back();
    }
};

int main() {
    DequeList dq;

    // Push elements
    dq.push_back(10);
    dq.push_front(5);
    dq.push_back(20);

    // Print front and back elements
    std::cout << "Front: " << dq.front() << ", Back: " << dq.back() << std::endl;

    // Pop elements
    dq.pop_front();
    dq.pop_back();

    // Print front element after popping
    std::cout << "Front after pop: " << dq.front() << std::endl;

    return 0;
}
```



3. **Стандартная реализация** `std::deque`

- Обычно для реализации дек в c++ хранится не циклический буффер, а просто буфферы (достаточно большие) отдельно друг от друга, а так же указатели на эти буфферы.
- Буфферы имеют стандартный размер `m`.
- При `push` и `pop` индексы меняются самостоятельно.
- Найдем `i`-ый элемент:
  - рассмотрим случай когда `startbuf` находится в самом начале:
    - сначала нужно найти в каком буффере находится элемент - `[i/m]`
    - а *индекс* в самом буффере - `i%m`.

Ассимптотика сложности `deque`

Operation	Complexity
push_front	O(1)
push_back	O(1)
pop_front	O(1)
pop_back	O(1)
index	O(1)

Сравнение с другими структурами

Data Structure	push_back	pop_back	push_front	pop_front	erase	insert	operator[]
vector	O(1) ~ O(n)	O(1)	- O(n)	- O(n)	O(n)	O(n)	O(1)
forward_list	O(1)	O(1)	O(1)	O(1)	O(1) (after)	O(1) (after)	- O(n)
list	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)	- O(n)
deque	O(1)	O(1)	O(1)	O(1)	O(n)	O(n)	O(1)