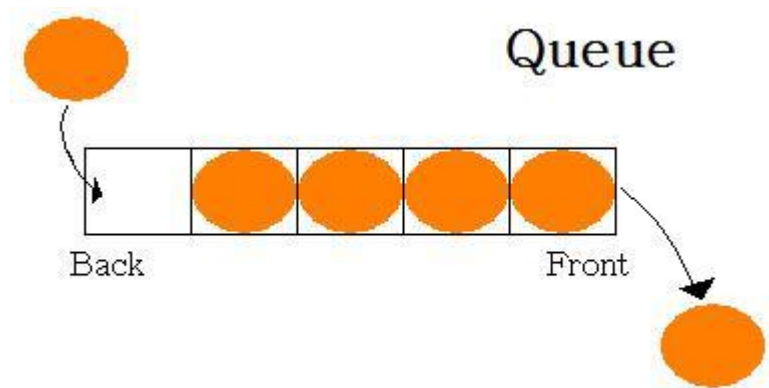


Опашка

доц. д-р Нора Ангелова

Опашка

- Съставна хомогенна линейна структура от данни
- „първи влязъл - пръв излязъл“ (FIFO)



Опашка

Логическо представяне

- Крайна редица от елементи от един и същ тип.

Операции:

- Включване - допустима е само за единия край на опашката (край на опашката).
- Изключване - допустима е само за другия край на опашката (начало на опашката, глава).

Достъп

Възможен е достъп само до елемента, който се намира в началото на опашката.

Достъпът е **пряк**.

Опашка

Операции:

- `empty()` – проверка дали опашката е празна.
- `push(x)` – включване на елемент в опашката.
- `pop()` – изключване на елемент от опашката.
- `head()` – достъп до първия елемент на опашката.

Опашка

Физическо представяне

- последователно – посредством масив
- свързано

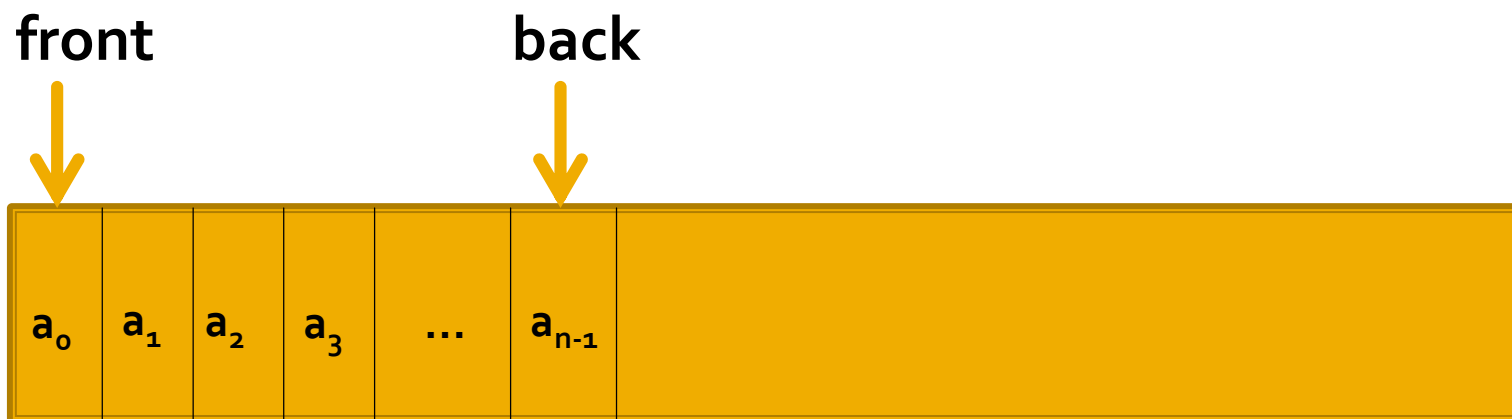
Реализации на опашка

Последователно представяне

- Запазва се блок от памет, в който опашката расте и се съкращава.

Опашка - последователно представяне

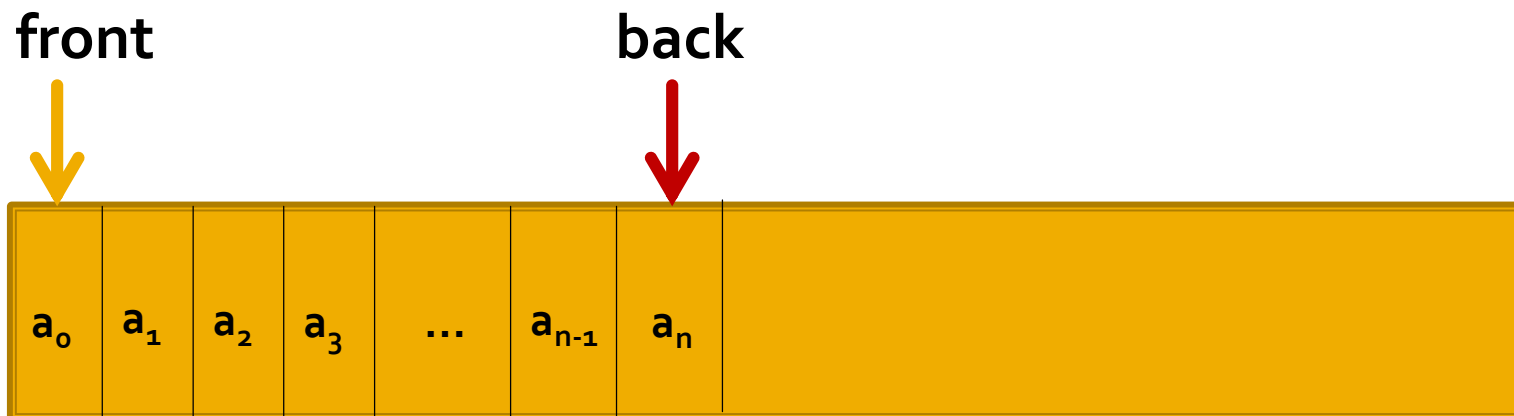
- Статичен масив с ограничен капацитет
 - `front` – индекс на първия елемент
 - `back` – индекс на последен елемент на опашката



Опашка - последователно представяне

Реализация с масив

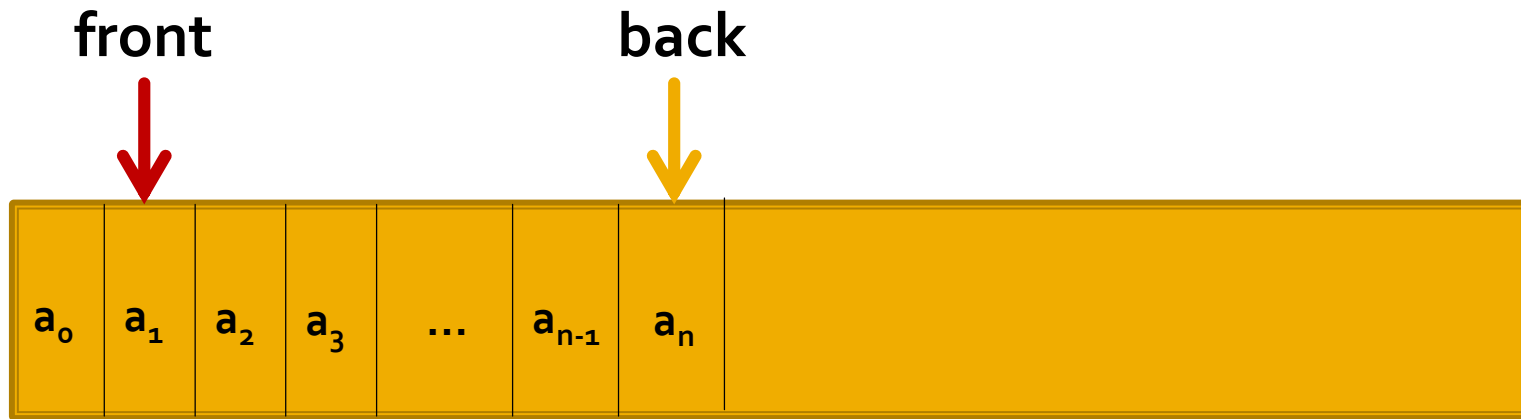
- `push (an)` – включва елемента a_n



Опашка - последователно представяне

Реализация с масив

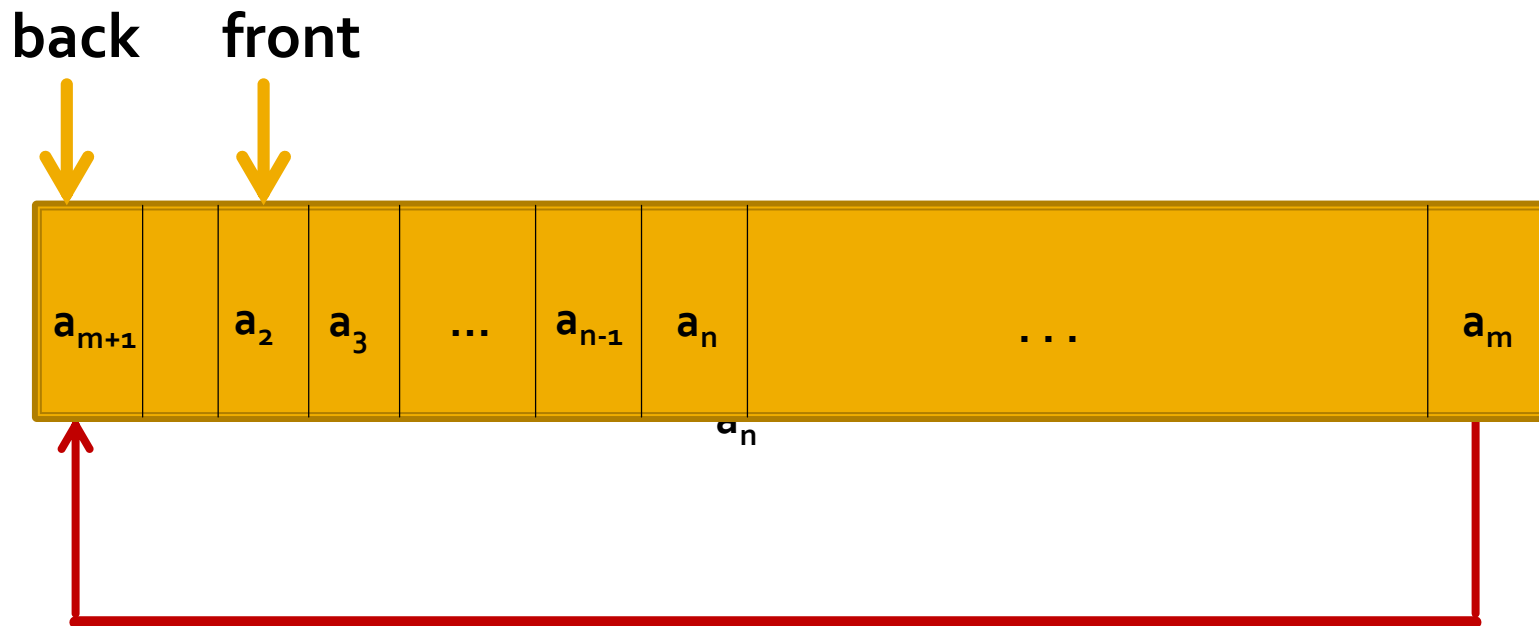
- `push (an)` – включва елемента a_n
- `pop ()` – изключва елемент



Опашка - последователно представяне

Реализация с масив

- `push (an)` – включва елемента a_n
- `pop ()` – изключва елемент
- ЦИКЛИЧНОСТ



Опашка - последователно представяне

```
const int MAX_SIZE = 100;

template <typename T>
class Queue {
    T elements[MAX_SIZE];
    int front, back, size; // Индекси и текущ брой на елементите

    bool full() const;
public:
    Queue(); // Създаване на празна опашка

    bool empty() const; // Проверка дали опашка е празна
    void push(T const& x); // Включване на елемент
    void pop(); // Изключване на елемент
    T front() const; // Достъп до първия елемент в опашка
};
```

Опашка - последователно представяне

```
// Задава индексите на първата празна позиция
template <typename T>
Queue<T>::Queue() : front(0), back(0), size(0) {}

template <typename T>
bool Queue<T>::full() const {
    return size == MAX_SIZE;
}

template <typename T>
bool Queue<T>::empty() const {
    return size == 0;
}
```

Опашка - последователно представяне

```
template <typename T>
T Queue<T>::front() const {
    if (empty()) {
        throw std::runtime_error("Can not get elem from an empty queue");
    }
    return elements[front];
}
```

```
template <typename T>
void Queue<T>::push(T const& x) {
    if (full()) {
        throw std::runtime_error("The queue is full!");
    }

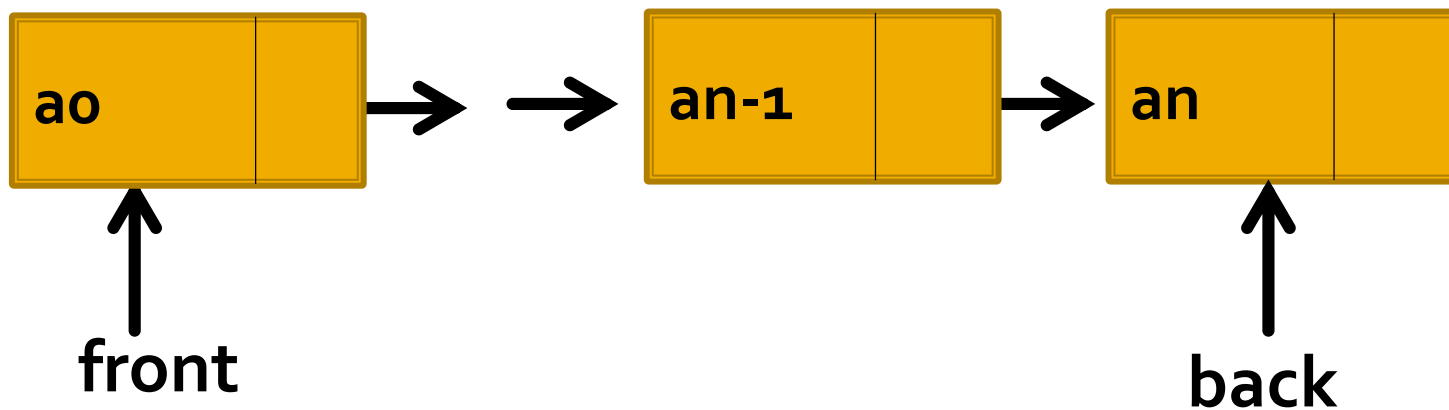
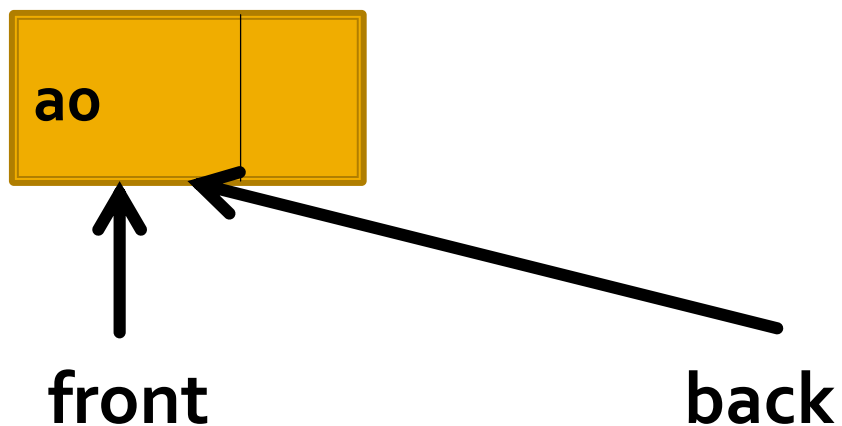
    elements[back] = x;
    size++;
    back++;
    back = back % MAX_SIZE;
}
```

Опашка - последователно представяне

```
template <typename T>
void Queue<T>::pop() {
    if (empty()) {
        throw std::runtime_error("Can not delete element from an empty queue");
    }
    size--;
    front++;
    front = front % MAX_SIZE;
}
```

Опашка – свързано представяне

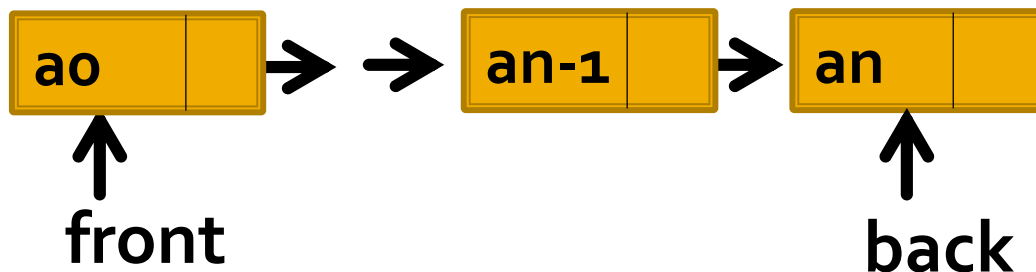
Свързано представяне



Опашка – свързано представяне

Свързано представяне

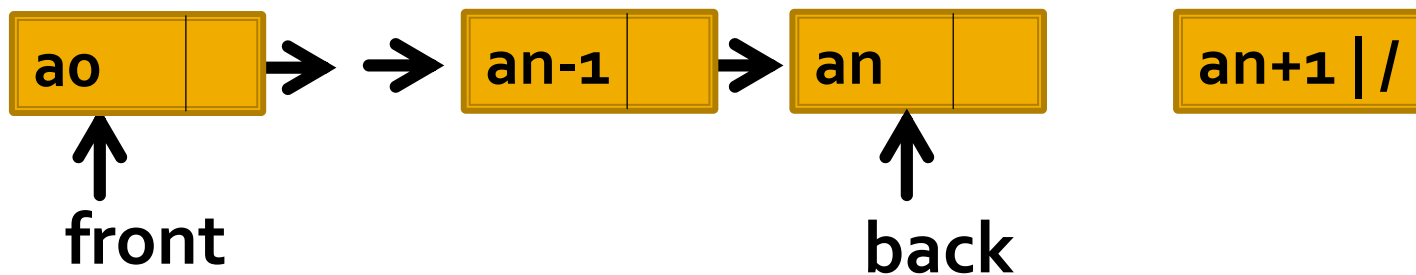
- Добавяне на елемент



Опашка – свързано представяне

Свързано представяне

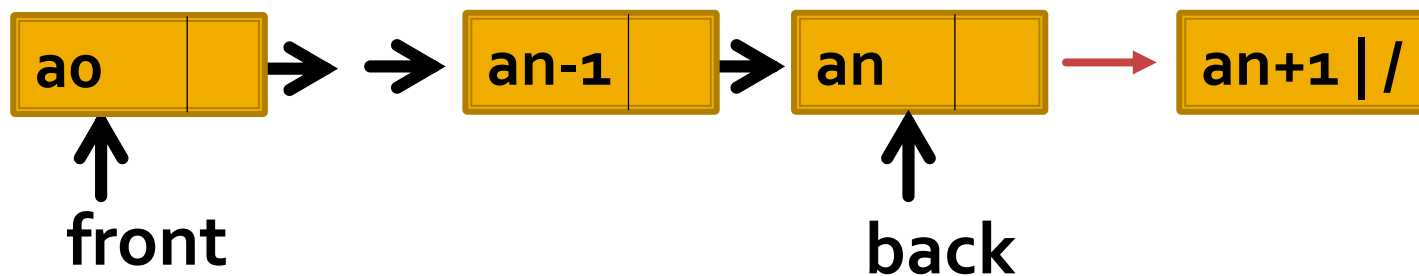
- Добавяне на елемент



Опашка – свързано представяне

Свързано представяне

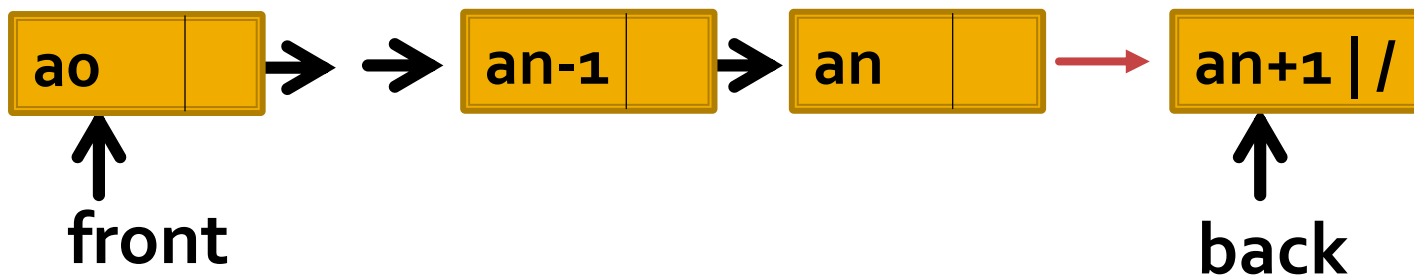
- Добавяне на елемент



Опашка – свързано представяне

Свързано представяне

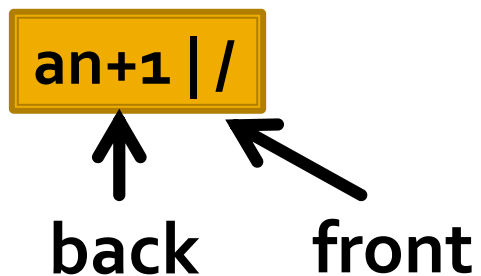
- Добавяне на елемент



Опашка – свързано представяне

Свързано представяне

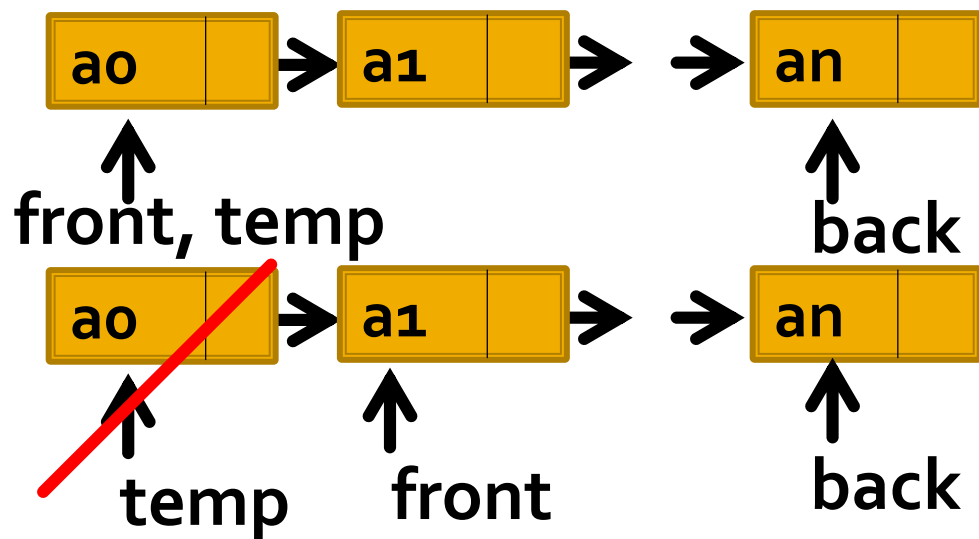
- Добавяне на първи елемент



Опашка – свързано представяне

Свързано представяне

- Добавяне на елемент
- Премахване на елемент



Опашка – свързано представяне

```
template <typename T>  
struct QueueElement {  
    T data;  
    QueueElement<T>* next;  
};
```



STL (Опашка)

`std::queue<T>`

`#include <queue>`

Интерфейс:

- `queue()` — създаване на празна опашка
- `empty()` — проверка за празнота на опашка
- `push(x)` — включване на първи елемент в опашката (`void`)
- `pop()` — изключване на последен елемент от опашката (`void`)
- `front()` — първи елемент в опашката (`reference || const_reference`)
- `back()` — последен елемент в опашката (`reference || const_reference`)
- `size()` — дължина на опашката

- `==, !=, <=, >=` — лексикографско сравнение на две опашки

Следва продължение...