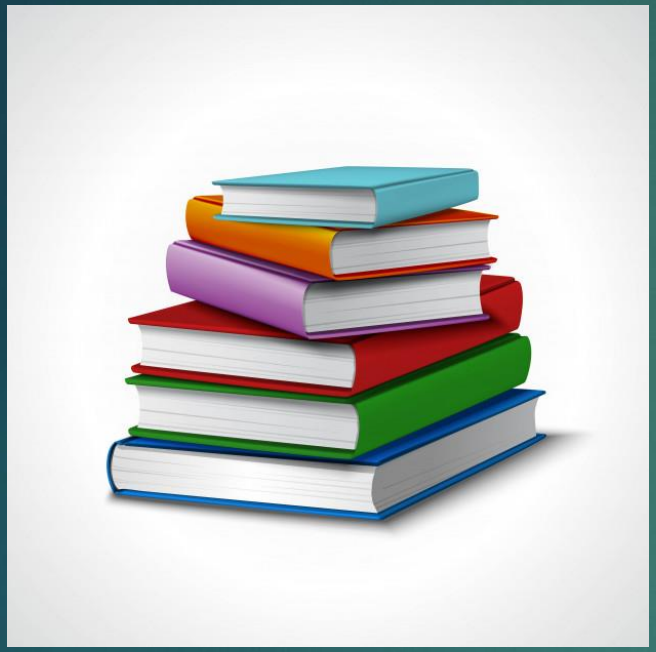
Лекция 4
Стеки и очереди

Стеки и очереди



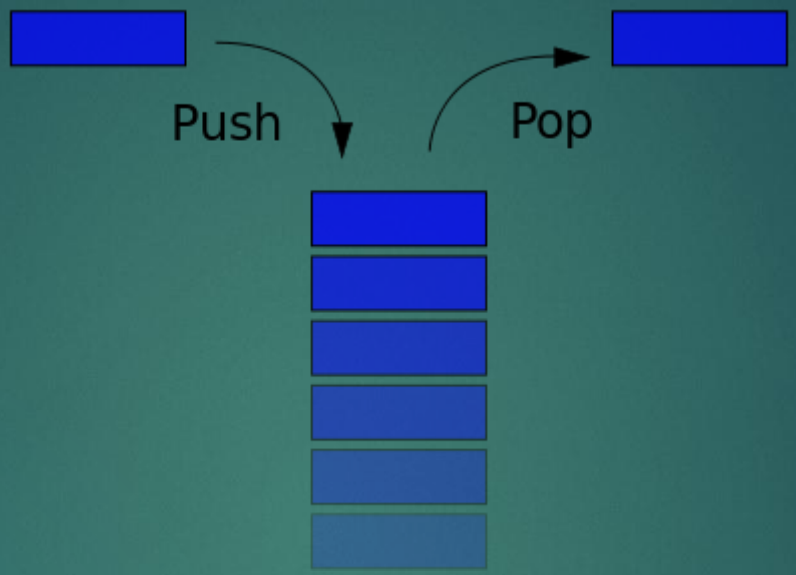
Стеки, очереди и приоритетные очереди являются более абстрактными сущностями, чем массивы и многие другие структуры данных. Они определяются, прежде всего, своим интерфейсом: набором разрешенных операций, которые могут выполняться с ними. Базовый механизм, используемый для их реализации, обычно остается невидимым для пользователя.

Стеки и очереди

В стеке доступен только один элемент данных: тот, который был в него вставлен последним. Удалив этот элемент, пользователь получает доступ к предпоследнему элементу и т. д. Такой механизм доступа удобен во многих ситуациях, связанных с программированием.

Многие микропроцессоры имеют стековую архитектуру. При вызове метода адрес возврата и аргументы заносятся в стек, а при выходе они извлекаются из стека. Операции со стеком встроены в микропроцессор.

Стеки и очереди



Основные операции со стеком — *вставка* (занесение) элемента в стек и *извлечение* из стека — выполняются только на вершине стека, то есть с его верхним элементом. Говорят, что стек работает по принципу LIFO (Last-In-First-Out), потому что последний занесенный в стек элемент будет первым извлечен из него.

Реализация стека

Реализовать стек на языке Java

Реализация стека на языке Java

```
class SimpleStack
{
        private int maxSize; // Размер массива
        private long[] stackArray;
        private int top; // Вершина стека


        …
}
```

Реализация стека на языке Java

```
public SimpleStack(int s) // Конструктор
{
        maxSize = s; // Определение размера стека
        stackArray = new long[maxSize]; // Создание массива
        top = -1; // Пока нет ни одного элемента
}
```

Реализация стека на языке Java

```
public void push(long j) // Размещение элемента на вершине стека
{
        stackArray[++top] = j; // Увеличение top, вставка элемента
}
```

Реализация стека на языке Java

```
public long pop() // Извлечение элемента с вершины стека
{
        return stackArray[top--]; // Извлечение элемента, уменьшение top
}
```

Реализация стека на языке Java

```
public long peek() // Чтение элемента с вершины стека
{
        return stackArray[top];
}
```

Реализация стека на языке Java

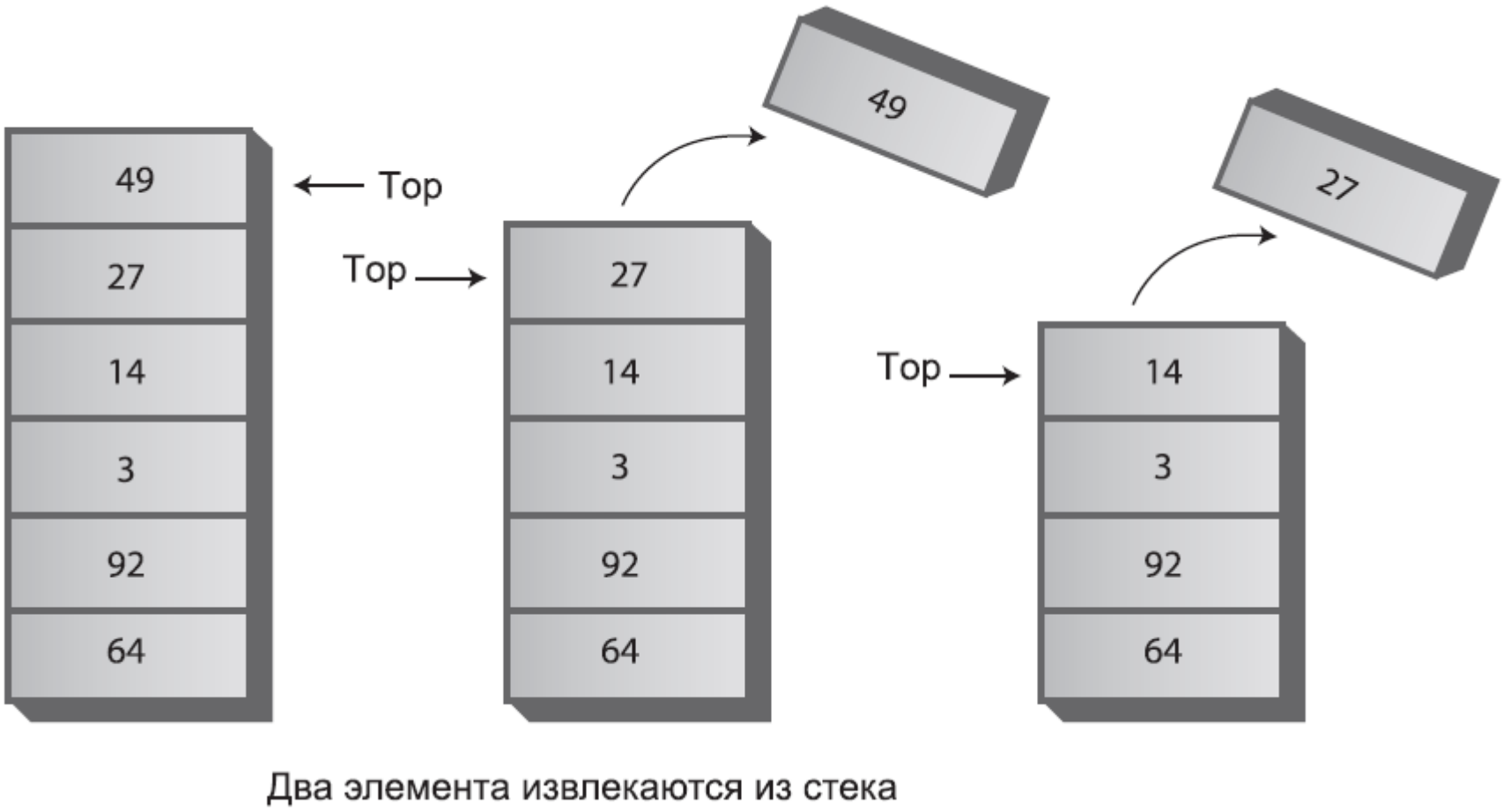
```
public boolean isEmpty() // True, если стек пуст
{
        return (top == -1);
}

public boolean isFull() // True, если стек полон
{
        return (top == maxSize-1);
}
```

Реализация стека на языке Java

Новый элемент заносится в стек

Реализация стека на языке Java



Два элемента извлекаются из стека

Перестановка букв в слове

Общая задача:
Вывести слово, в котором буквы переставлены в обратном порядке.

Требования:

- Создать класс-утилиту, который будет работать со словом (переставлять буквы в слове). Класс должен хранить исходное слово и слово, в котором буквы переставлены в обратном порядке.
- Алгоритм для перестановки букв в слове реализовать при помощи стека.
- Использовать свой собственный класс, реализующий Стек

Перестановка букв в слове

```
class Reverser
{
        private String input; // Входная строка
        private String output; // Выходная строка

        public Reverser(String in) // Конструктор
        { input = in; }

        ...
}
```

```
public String doRev()  { // Перестановка символов
        int stackSize = input.length(); // Определение размера стека
        StackX theStack = new StackX(stackSize); // Создание стека
        for(int j=0; j<input.length(); j++) {
                char ch = input.charAt(j); // Чтение символа из входного потока
                theStack.push(ch); // Занесение в стек
        }
        output = "";
        while( !theStack.isEmpty() ) {
                char ch = theStack.pop(); // Извлечение символа из стека
                output = output + ch; // Присоединение к выходной строке
        }
        return output;}
```

Поиск парных скобок

Написать класс для проверки парных скобок в строке

Пример:
c[d] // Правильно
a{b[c]d}e // Правильно
a{b(c]d}e // Неправильно;] не соответствует (
a[b{c}d]e} // Неправильно; у завершающей скобки } нет пары
a{b(c) // Неправильно; у открывающей скобки { нет пары

```
class BracketChecker
{
    private String input; // Входная строка

    public BracketChecker(String in) // Конструктор
        { input = in; }

}
```

```
public void check() {
        int stackSize = input.length(); // Определение размера стека
        StackX theStack = new StackX(stackSize); // Создание стека

        for(int j=0; j<input.length(); j++) { // Последовательное чтение
                char ch = input.charAt(j); // Чтение символа
                switch(ch) {
                        case ‘{‘: // Открывающие скобки
                        case '[':
                        case '(':
                                theStack.push(ch); // Занести в стек
                                break;
                        case ‘}’: // Закрывающие скобки
                        case ']':
                        case ')':
                                if( !theStack.isEmpty() ){   // Если стек не пуст,
                                        char chx = theStack.pop(); // Извлечь и проверить
                                        if( (ch=='}’ && chx!=’{‘) ||(ch==']' && chx!='[') || (ch==')' && chx!='(') )
                                                System.out.println("Error: "+ch+" at "+j);
                                } else // Преждевременная нехватка элементов
                                        System.out.println("Error: "+ch+" at "+j);
                                break;
                        default: // Для других символов действия не выполняются
                                break;
                }
        }
        // В этой точке обработаны все символы
        if( !theStack.isEmpty() )
                System.out.println("Error: missing right delimiter");
}
```

Эффективность стеков

Занесение и извлечение элементов из стека, реализованного в классе Stack, выполняется за время $O(1)$. Иначе говоря, время выполнения операции не зависит от количества элементов в стеке; следовательно, операция выполняется очень быстро, не требуя ни сравнений, ни перемещений.

Очереди

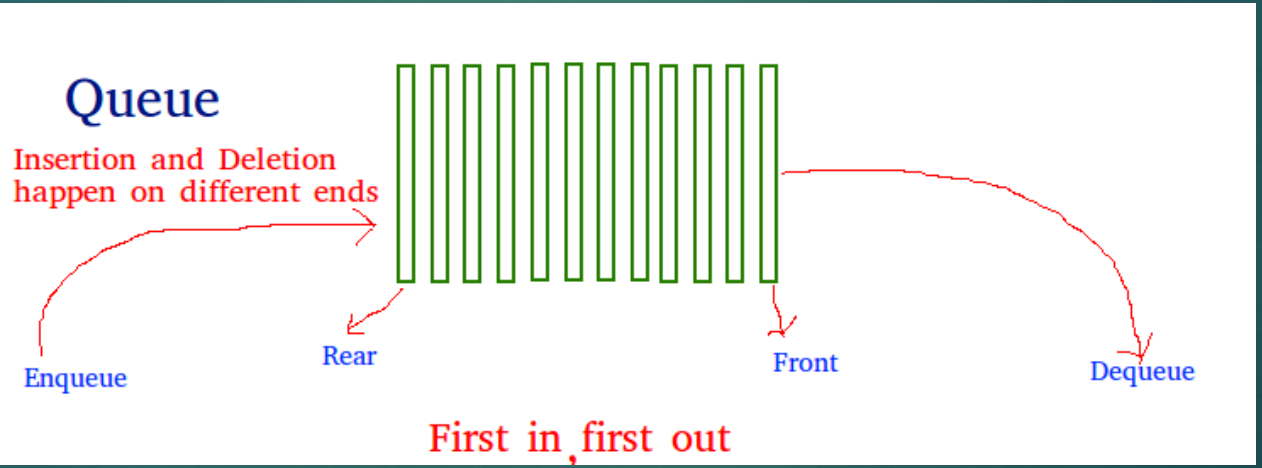
Структура данных, называемая в информатике *очередью*, напоминает стек, но в очереди первым извлекается элемент, вставленный первым (FIFO, First-In-First-Out), тогда как в стеке, как мы видели, первым извлекается элемент, вставленный последним (LIFO). Она работает по тому же принципу, что и очередь в кино: человек, первым вставшим в очередь, первым доберется до кассы и купит билет. Тот, кто встанет в очередь последним, последним купит билет (или не купит, если билеты будут распроданы).

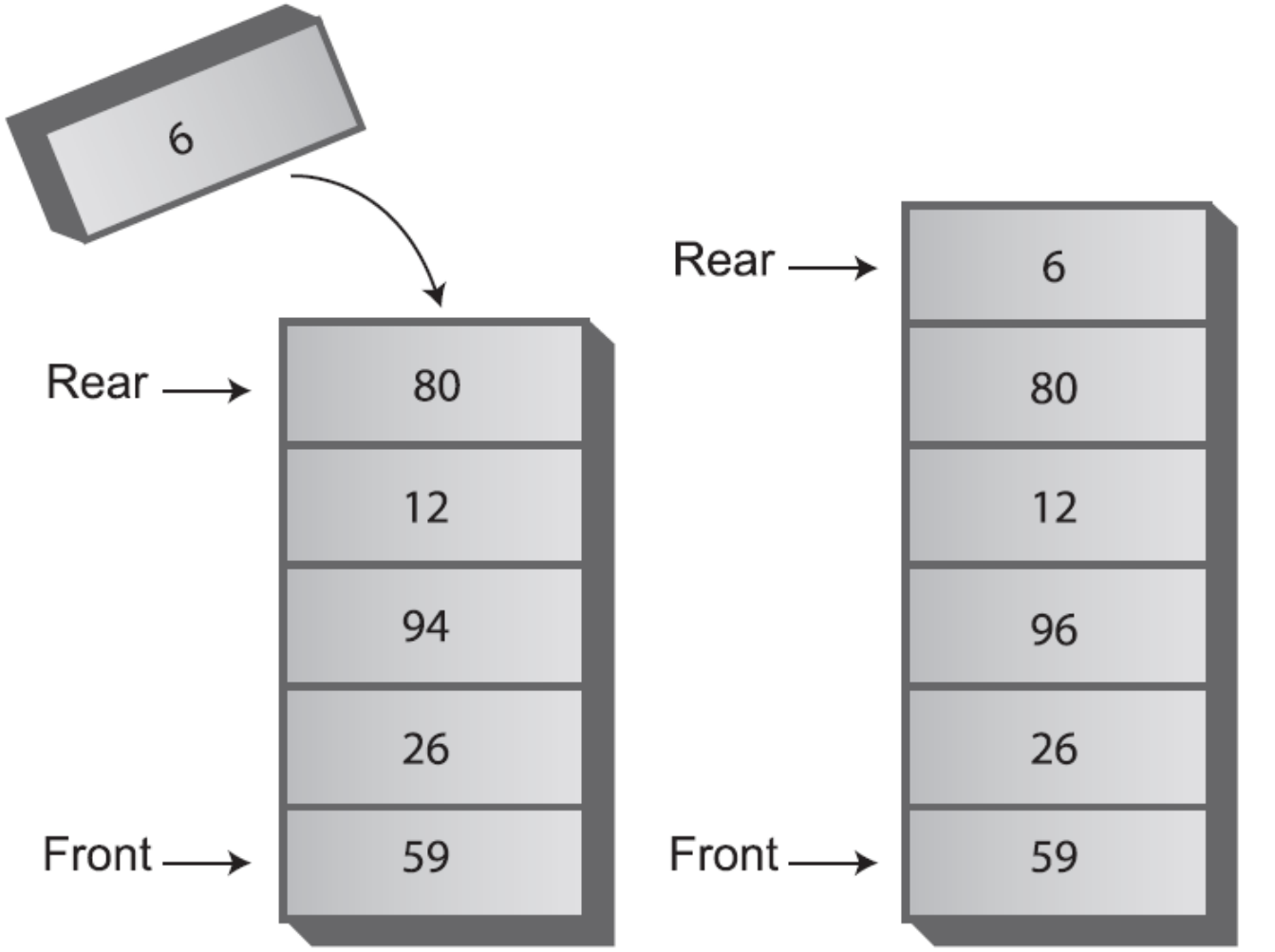
Очереди

В операционной системе вашего компьютера (и в сети) трудятся различные очереди, незаметно выполняющие свои обязанности. В очереди печати задания ждут освобождения принтера. Данные, вводимые с клавиатуры, тоже сохраняются в очереди. Если вы работаете в текстовом редакторе, а компьютер на короткое время отвлекся на выполнение другой операции, нажатия клавиш не будут потеряны; они ожидают в очереди, пока у редактора не появится свободное время для их получения. Очередь обеспечивает хранение нажатий клавиш в исходной последовательности до момента обработки.

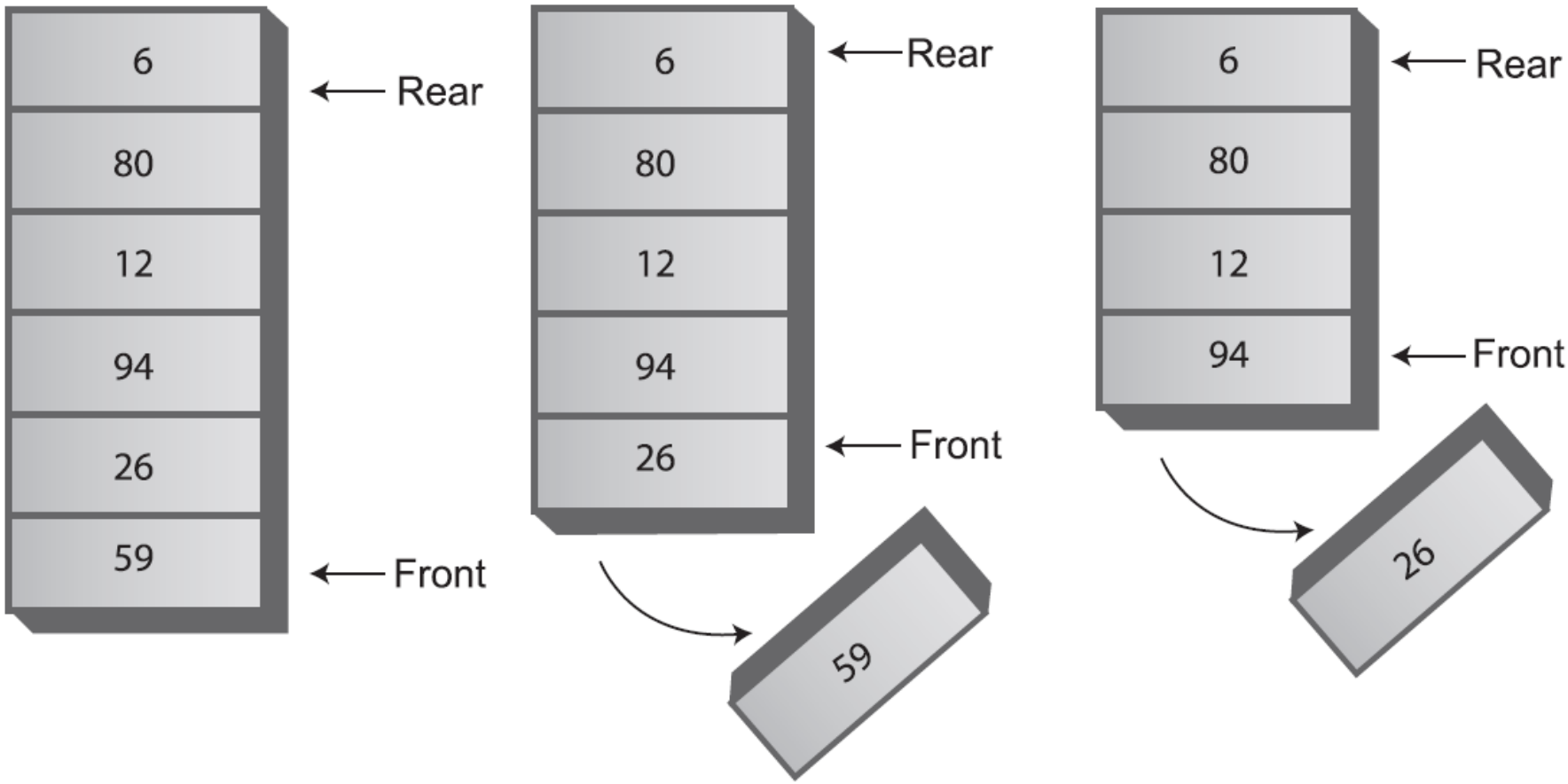
Очереди

Две основные операции с очередью — *вставка* элемента в конец очереди и *извлечение* элемента с начала очереди. Все происходит так же, как в очереди в кино: человек становится в конец очереди, ждет, добирается до кассы, покупает билет и покидает очередь из ее начала.

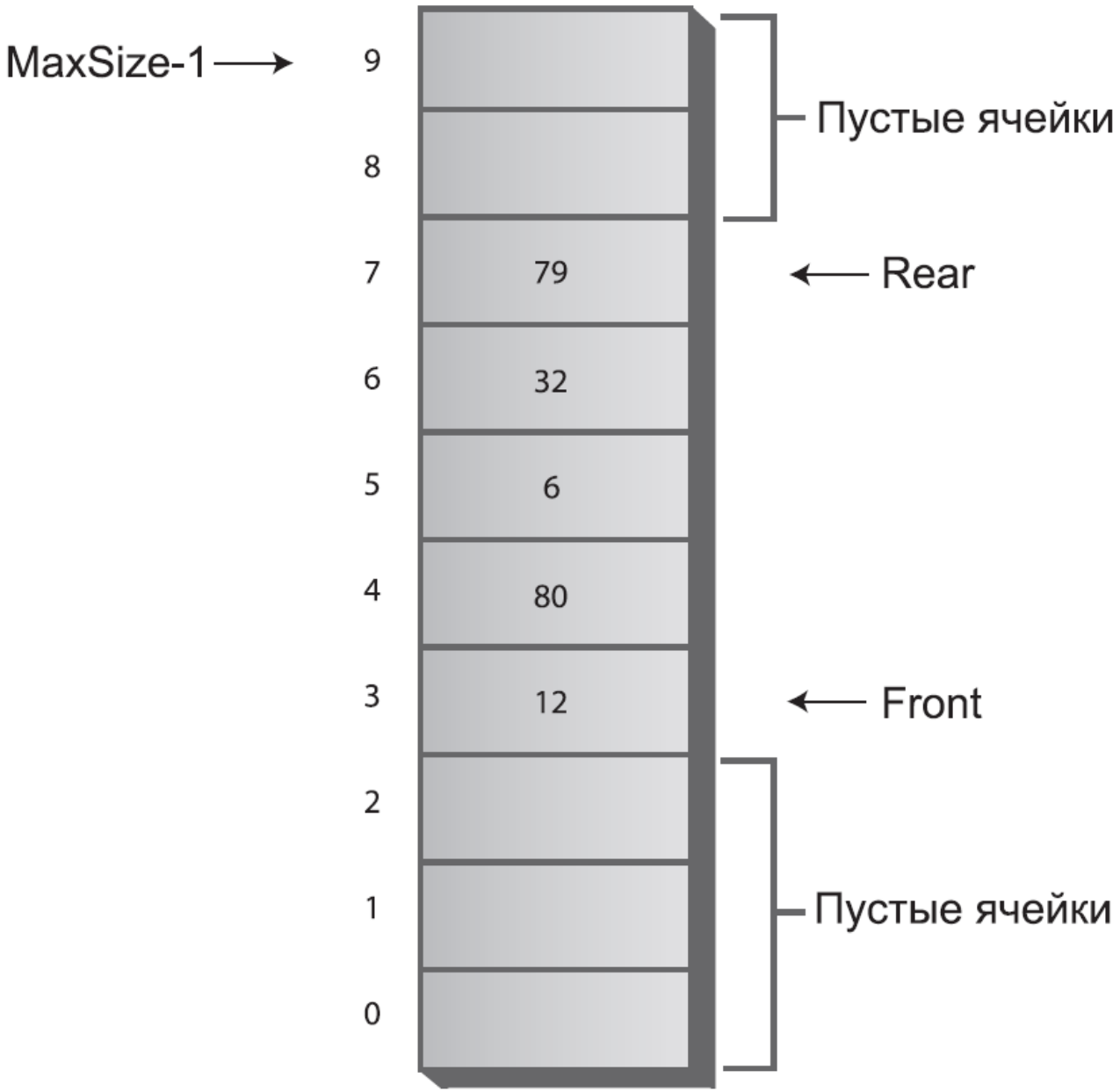


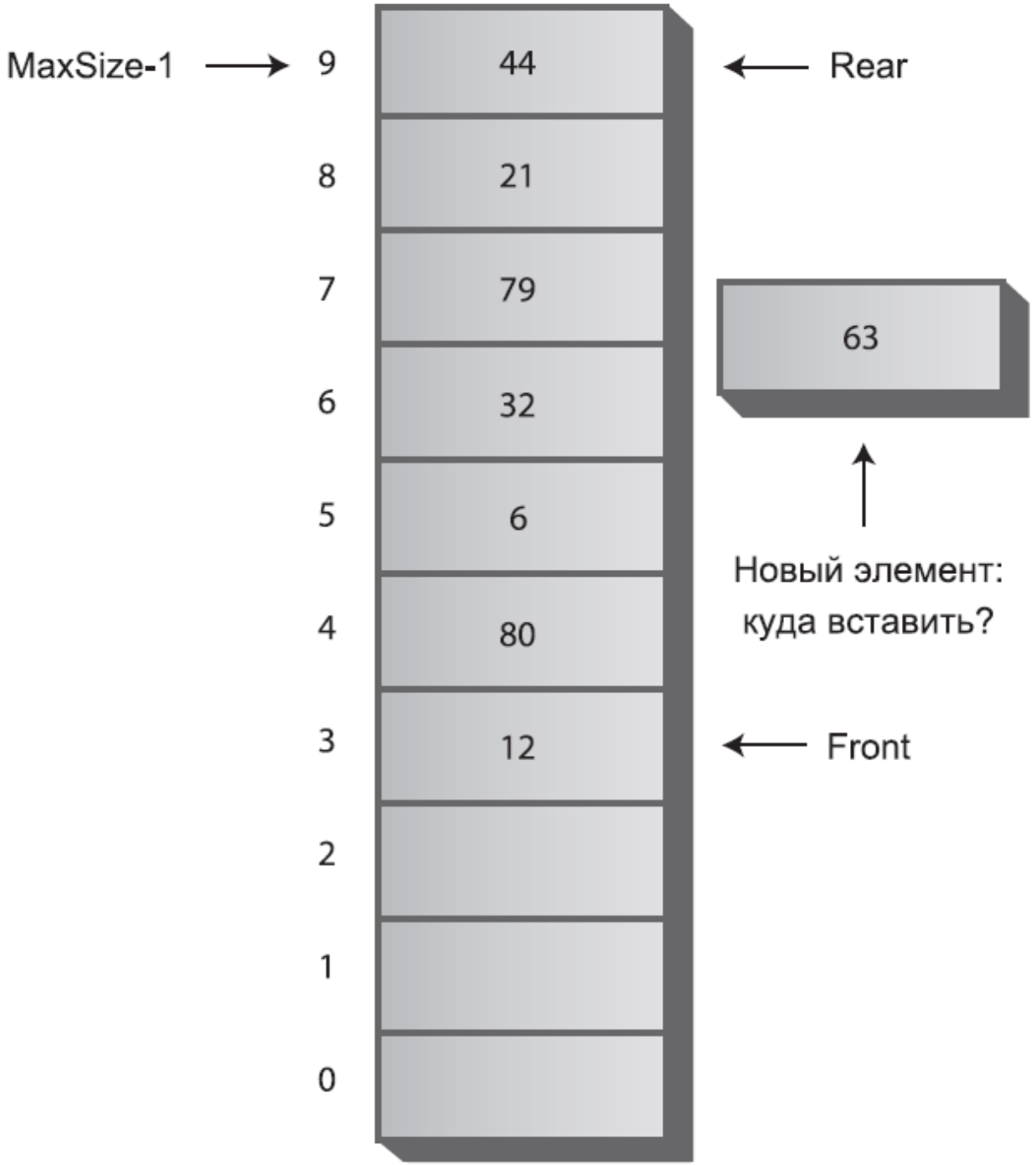


Новый элемент вставляется в конец очереди



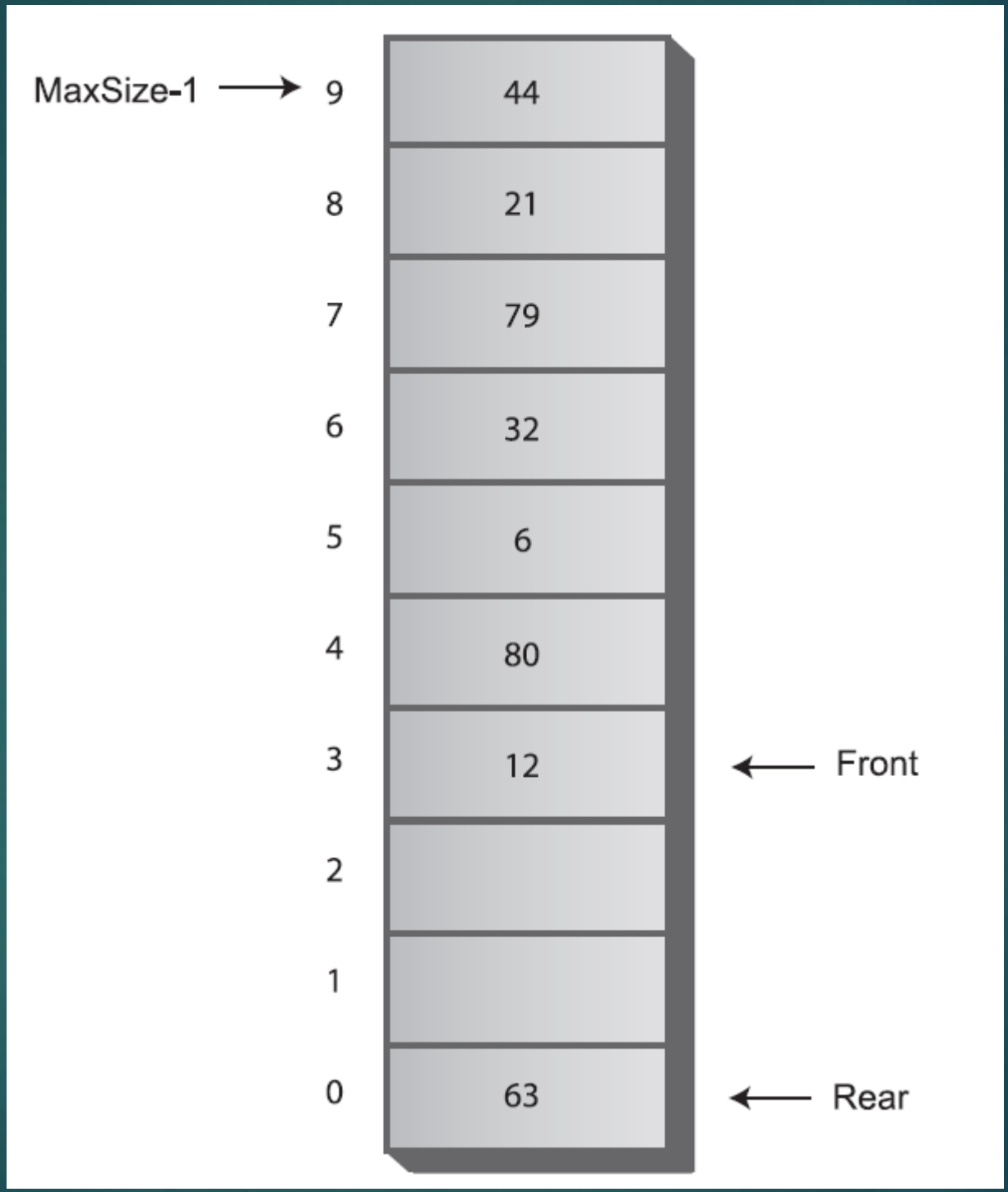
Извлечение двух элементов от начала очереди

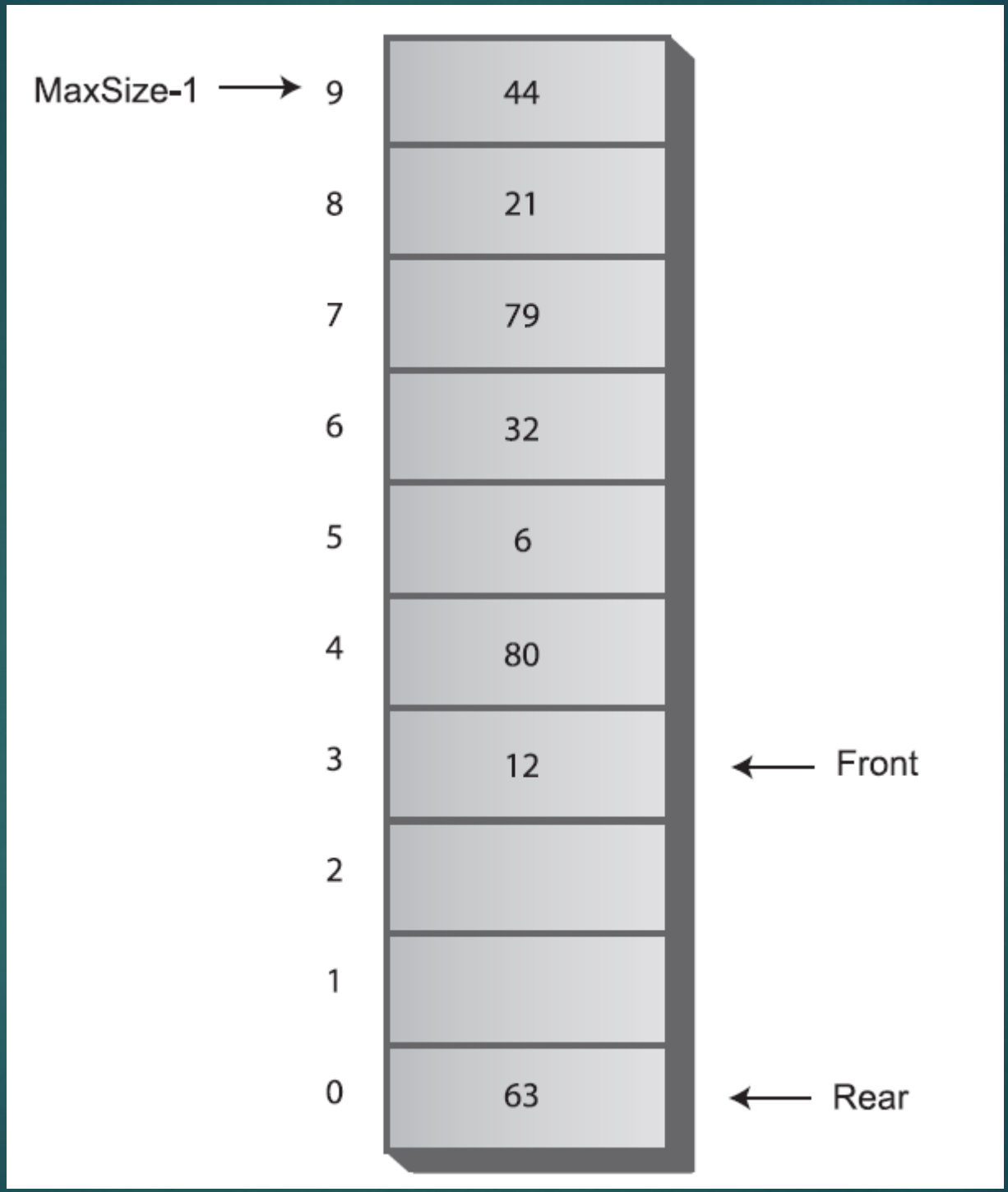

MaxSize-1
9
8
7
79
6
32
5
6
4
80
3
12
2
1
0
Пустые ячейки
Rear
Front
Пустые ячейки


MaxSize-1
9
8
7
6
5
4
3
2
1
0
44
21
79
32
6
80
12
Rear
63
Новый элемент:
куда вставить?
Front

Циклический перенос

Для решения проблемы со вставкой новых элементов в очередь, в которой имеются свободные ячейки, маркеры Front и Rear при достижении границы массива перемещается к его началу. Такая структура данных называется *циклической очередью* (также иногда встречается термин *кольцевой буфер*).


MaxSize-1
9
8
7
6
5
4
3
2
1
0
44
21
79
32
6
80
12
63
Front
Rear


MaxSize-1
9
8
7
6
5
4
3
2
1
0
44
21
79
32
6
80
12
63
Front
Rear

Реализация очереди

Реализовать очередь на языке Java

```
class Queue
{
    private int maxSize;
    private long[] queArray;
    private int front;
    private int rear;
    private int nItems;

    public Queue(int s) { // Конструктор
        maxSize = s;
        queArray = new long[maxSize];
        front = 0;
        rear = -1;
        nItems = 0;
    }
```

```
public void insert(long j) {  // Вставка элемента в конец очереди
    if(rear == maxSize-1) // Циклический перенос
        rear = -1;
    queArray[++rear] = j; // Увеличение rear и вставка
    nItems++; // Увеличение количества элементов
}

public long remove()  { // Извлечение элемента в начале очереди
    long temp = queArray[front++]; // Выборка и увеличение front
    if(front == maxSize) // Циклический перенос
        front = 0;
    nItems--; // Уменьшение количества элементов
    return temp;
}
```

```
public long peekFront() { // Чтение элемента в начале очереди
    return queArray[front];
}

public boolean isEmpty()  { // true, если очередь пуста
    return (nItems==0);
}

public boolean isFull()  { // true, если очередь заполнена
    return (nItems==maxSize);
}

public int size()  { // Количество элементов в очереди
    return nItems;
}
```

Эффективность очередей

Вставка и извлечение элементов очереди, как и элементов стека, выполняются за время $O(1)$.

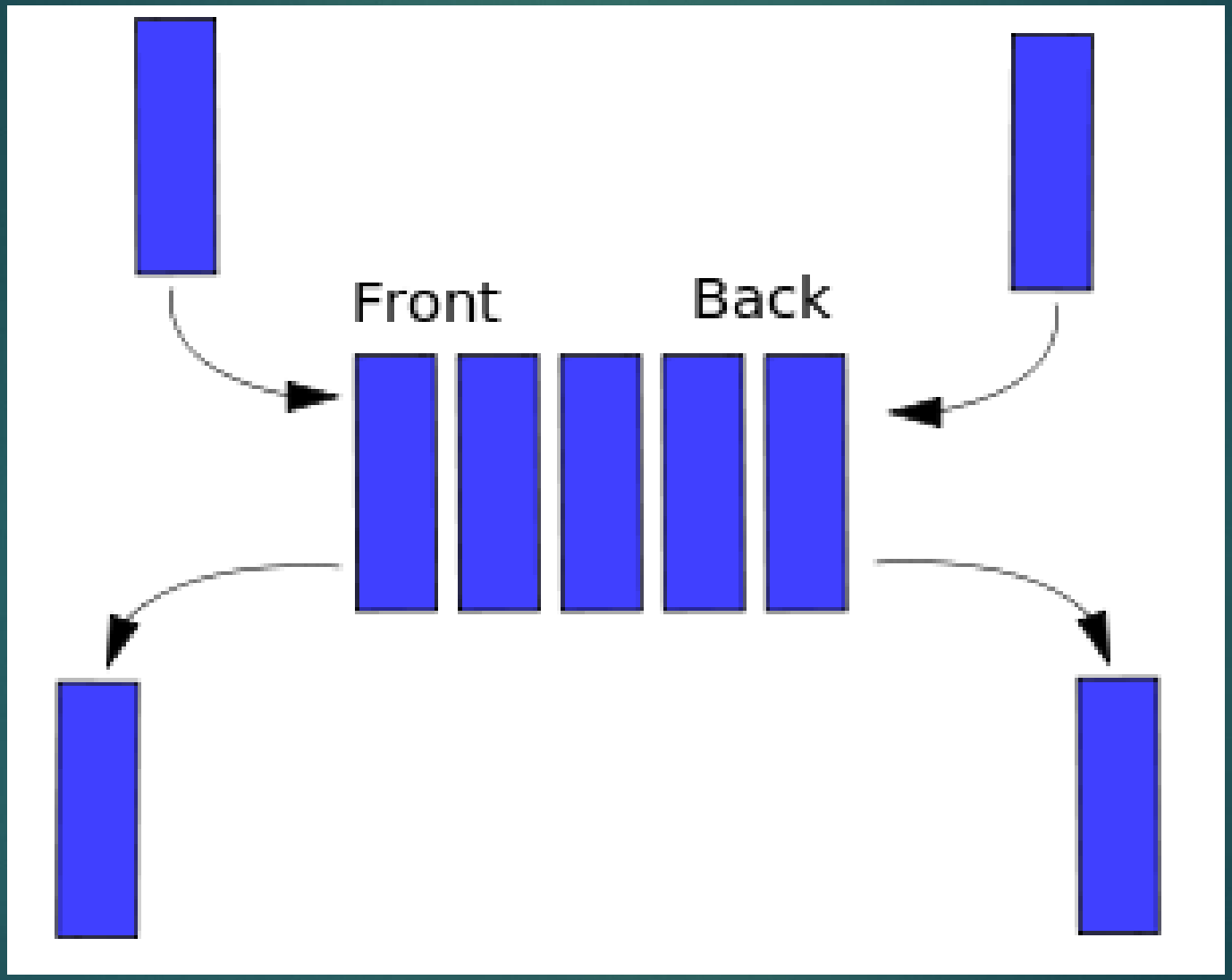
Дек

Дек (deque) представляет собой двустороннюю очередь. И вставка, и удаление элементов могут производиться с обоих концов. Соответствующие методы могут называться insertLeft()/insertRight() и removeLeft()/removeRight().

Если ограничиться только методами insertLeft() и removeLeft() (или их эквивалентами для правого конца), дек работает как стек. Если же ограничиться методами insertLeft() и removeRight() (или противоположной парой), он работает как очередь.

По своей гибкости деки превосходят и стеки, и очереди; иногда они используются в библиотеках классов-контейнеров для реализации обеих разновидностей.Тем не менее используются они реже стеков или очередей, поэтому подробно рассматривать мы их тоже не будем.

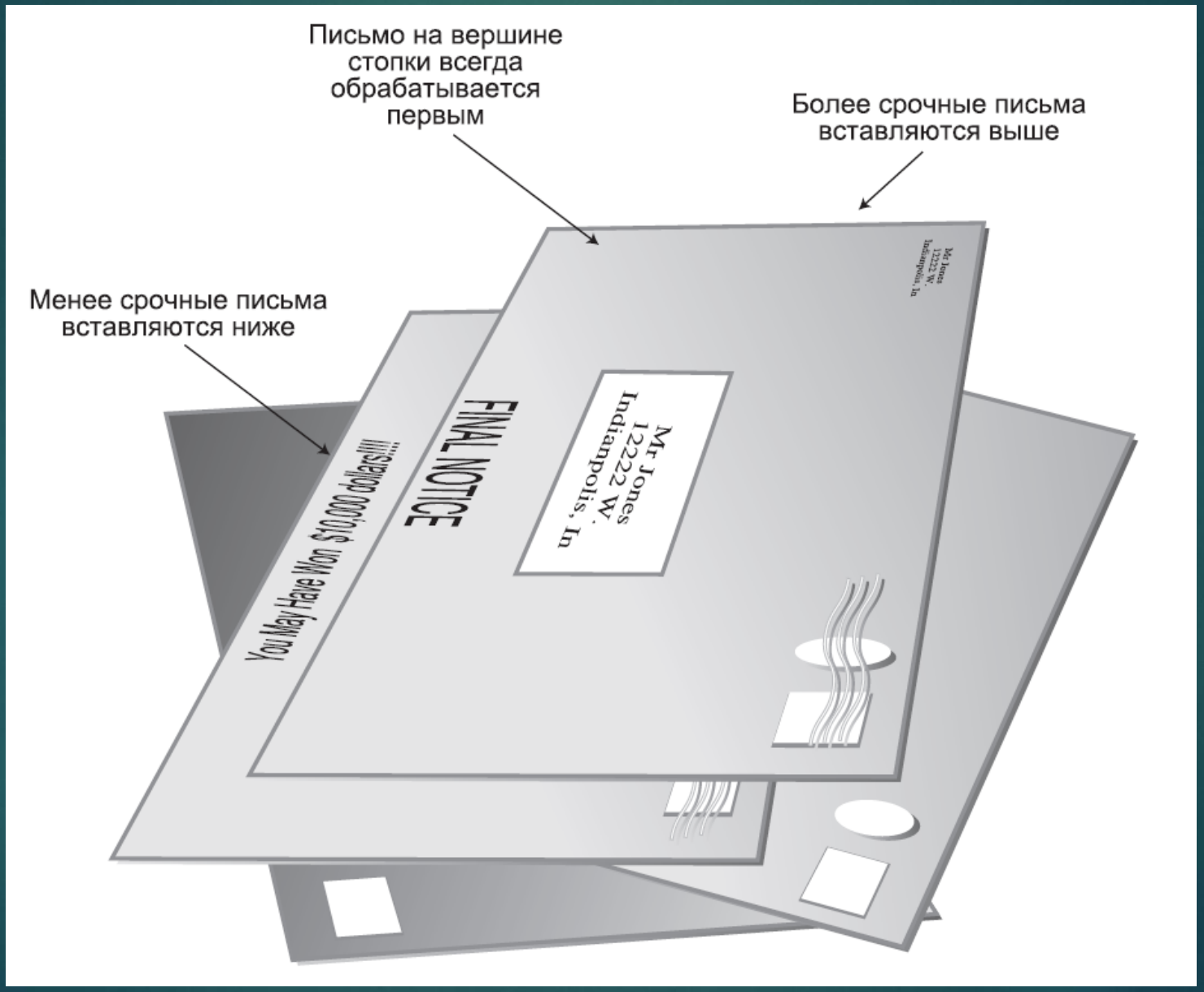
Дек



Приоритетные очереди

Приоритетная очередь является более специализированной структурой данных, чем стек или очередь, однако и он неожиданно часто оказывается полезным. У приоритетной очереди, как и у обычной, имеется начало и конец, а элементы извлекаются от начала. Но у приоритетной очереди элементы упорядочиваются по ключу, так что элемент с наименьшим (в некоторых реализациях — наибольшим) значением ключа всегда находится в начале. Новые элементы вставляются в позициях, сохраняющих порядок сортировки.

Приоритетные очереди



Эффективность приоритетных очередей

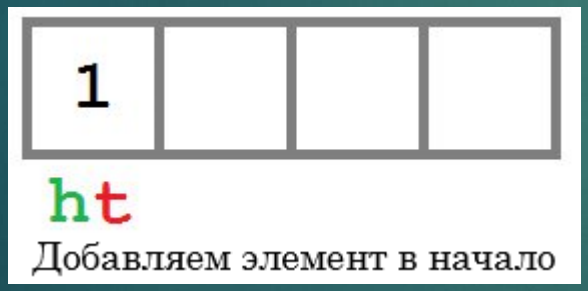
Простая реализации приоритетной очереди вставка выполняется за время $O(N)$, а извлечение — за время $O(1)$.

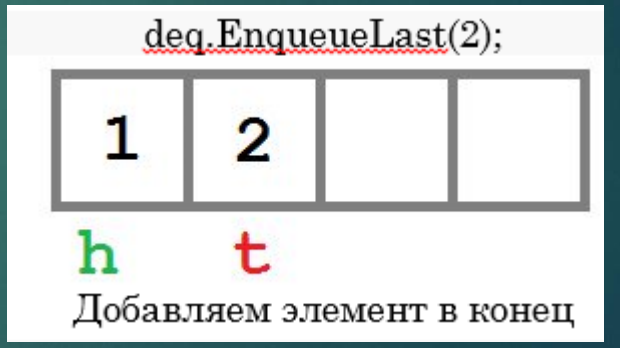
Хранение элементов в массиве

- Как уже было упомянуто, у реализации очереди с использованием массива есть свои преимущества. Она выглядит простой, но на самом деле есть ряд нюансов, которые надо учесть.
- Давайте посмотрим на проблемы, которые могут возникнуть, и на их решение. Кроме того, нам понадобится информация об увеличении внутреннего массива из прошлой статьи о динамических массивах.

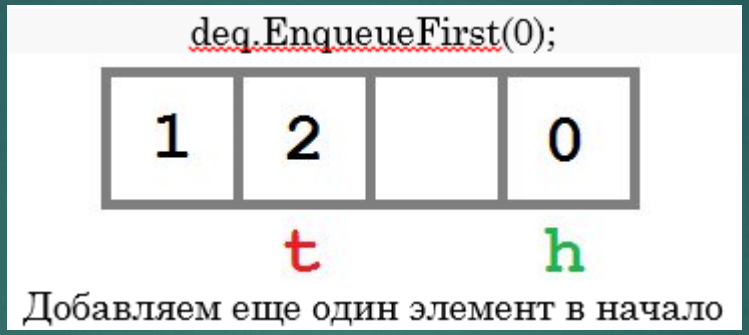
Хранение элементов в массиве

- При создании очереди у нее внутри создается массив нулевой длины. Красные буквы «h» и «t» означают указатели _head и _tail соответственно.
- Deque deq = new Deque();
- deq.EnqueueFirst(1);



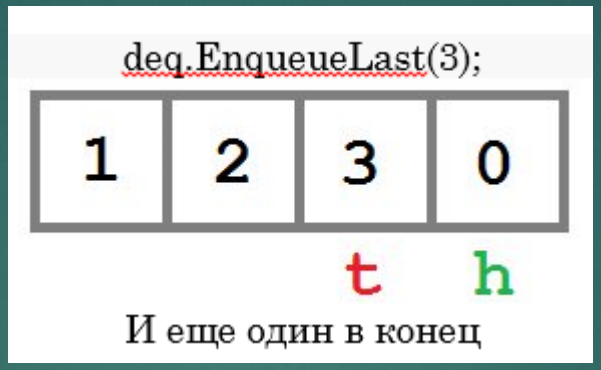


Хранение элементов в массиве



- Обратите внимание: индекс «головы» очереди перескочил в начало списка. Теперь первый элемент, который будет возвращен при вызове метода DequeueFirst — 0 (индекс 3).

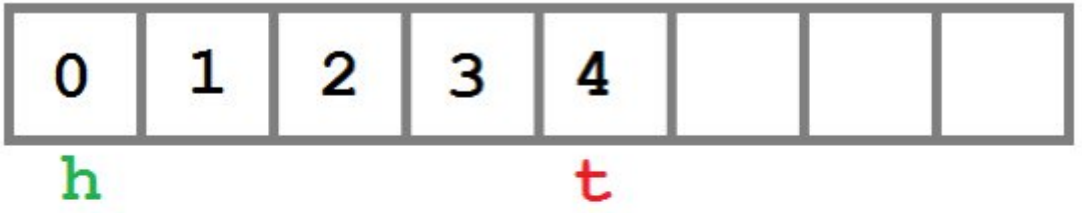
Хранение элементов в массиве



- Массив заполнен, поэтому при добавлении элемента произойдет следующее:
- Алгорим роста определит размер нового массива.
- Элементы скопируются в новый массив с «головы» до «хвоста».
- Добавится новый элемент.

Хранение элементов в массиве

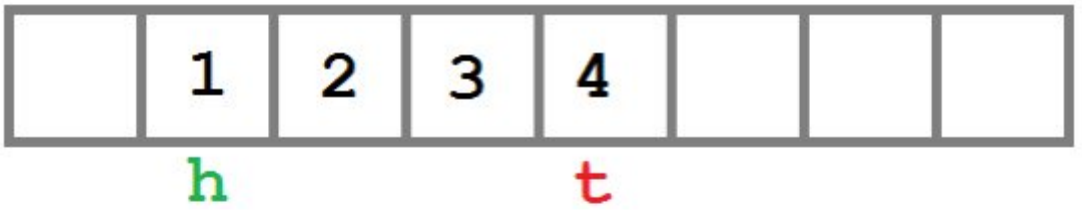
deq.EnqueueLast(4);



Добавляем значение в конец расширенного массива

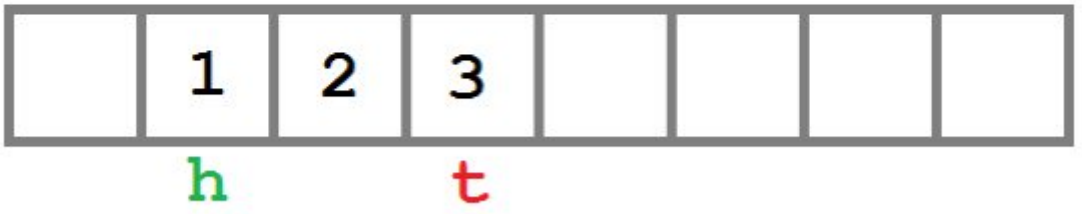
Теперь посмотрим, что происходит при удалении элемента:

deq.DequeueFirst();



Удаляем элемент из начала

deq.DequeueLast();



Удаляем элемент с конца

Хранение элементов в массиве

- Ключевой момент: вне зависимости от вместимости или заполненности внутреннего массива, логически, содержимое очереди — элементы от «головы» до «хвоста» с учетом «закольцованности». Такое поведение также называется «кольцевым буфером».

Обратная польская запись

Обратная польская нотация	Обычная нотация
2 3 +	2 + 3
2 3 * 4 5 * +	(2 * 3) + (4 * 5)
2 3 4 5 6 * + - /	2 / (3 - (4 + (5 * 6)))

Обратная польская запись

Шаг	Оставшаяся цепочка	Стек
1	8, 2,5, *+1,3,2*+4-/	8
2	2,5, *+1,3,2*+4-/	8,2
3	5*+1,3,2*+4-/	8,2,5
4	*+1,3,2*+4-/	8,10
5	+1,3,2*+4-/	18
6	1,3,2*+4-/	18,1
7	3,2*+4-/	18,1,3
8	2*+4-/	18,1,3,2
9	*+4-/	18,1,6
10	+4-/	18,7
11	4-/	18,7,4
12	-/	18,3
13	/	6

Обратная польская запись

																5			
												4				4			
	| =>	7	| =>	4+7	| =>		| =>	3	| =>	3	| =>	3	| =>	4
4		4				11		11		11				11					
4		7		"+"				3		4		5				"			


ПРЕЗЕНТАЦИЯ ОКОНЧЕНА
УЧИТЕ ИНФОРМАТИКУ)