

# Алгоритмы и структуры данных-1

## Линейные контейнеры. ADT Стек, Очередь, Список

Практическое занятие №2 09.09 — 14.09.2024

2024–2025 учебный год

# ПЛАН

Реализация и применение ADT Стек, ADT Очередь.  
Реализация очереди на паре стеков.

Базовый прием амортизационной оценки сложности.  
Метод банкира

Связный список – циклические сегменты. Алгоритм Флойда

# ADT Cтек

## Last In First Out

# Интерфейс стека

789012
123456

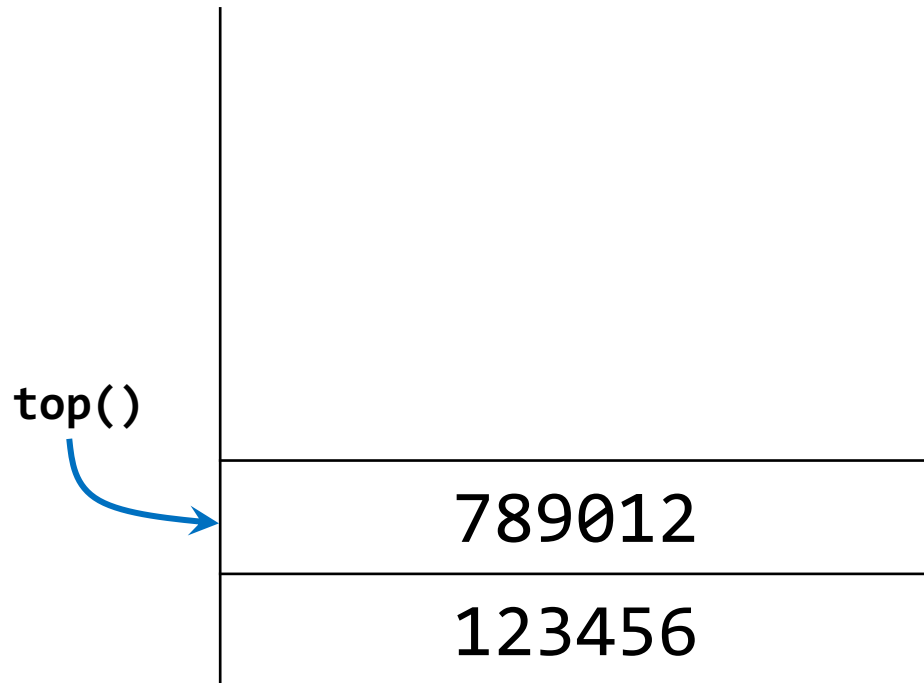


ADT\_Stack.cpp

```
template<typename T>
class Stack {
public:
    void push(T elem);
    T pop();
    T& top();

private:
    //внутренняя структура данных
}
```

# Интерфейс стека **top()**

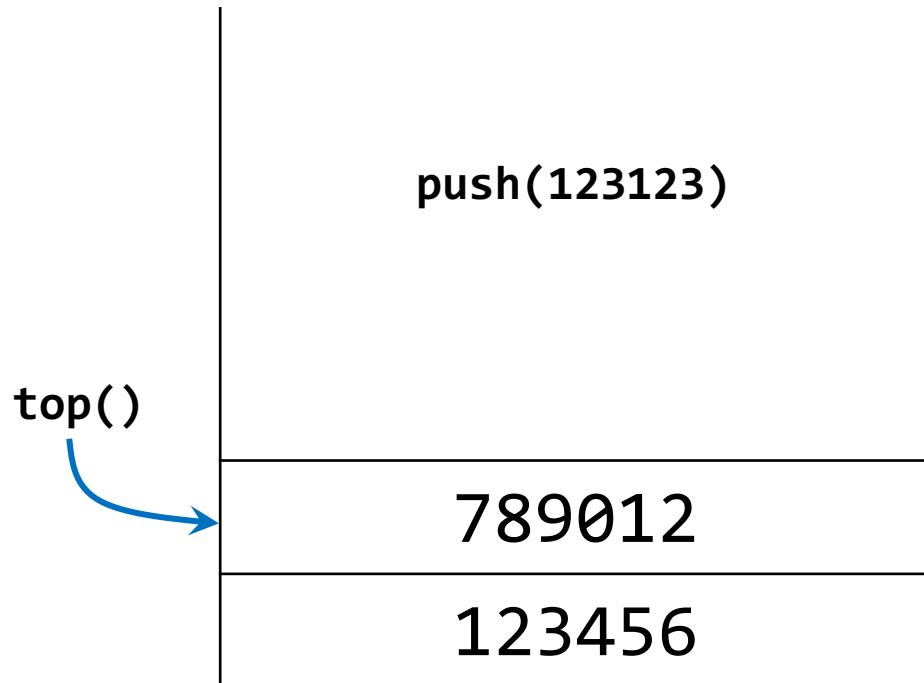


ADT\_Stack.cpp

```
template<typename T>
class Stack {
public:
    void push(T elem);
    T pop();
    T& top();

private:
    //внутренняя структура данных
}
```

# Интерфейс стека `push(T elem)`

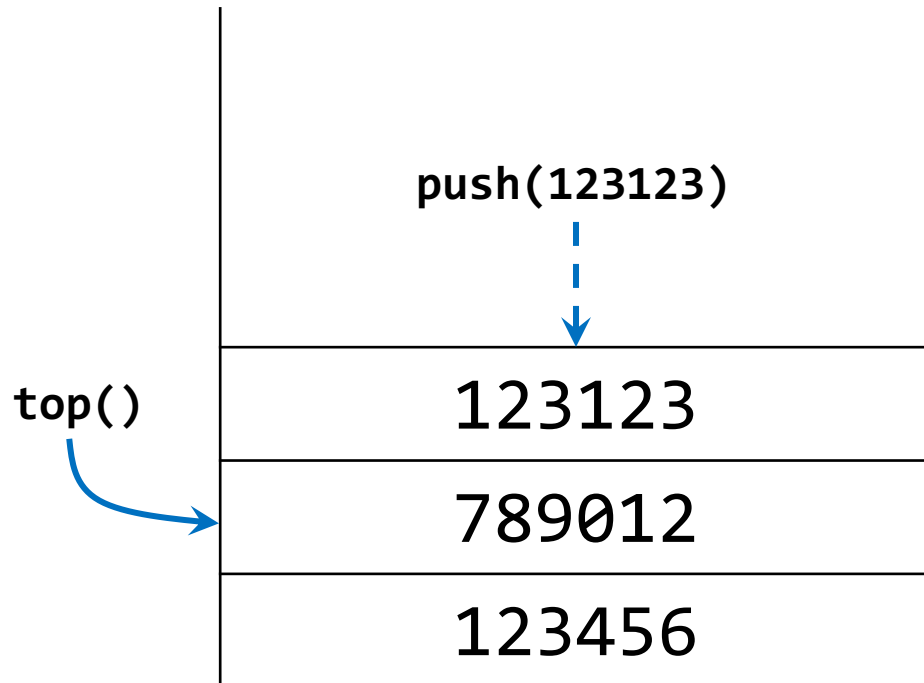


ADT\_Stack.cpp

```
template<typename T>
class Stack {
public:
    void push(T elem);
    T pop();
    T& top();

private:
    //внутренняя структура данных
}
```

# Интерфейс стека `push(T elem)`

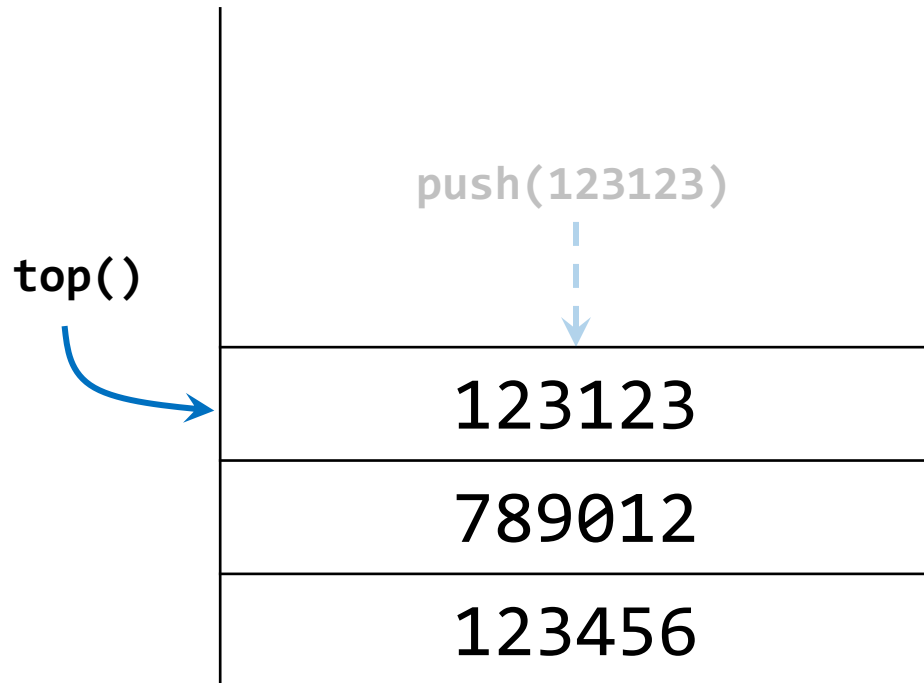


ADT\_Stack.cpp

```
template<typename T>
class Stack {
public:
    void push(T elem);
    T pop();
    T& top();

private:
    //внутренняя структура данных
}
```

# Интерфейс стека `push(T elem)`



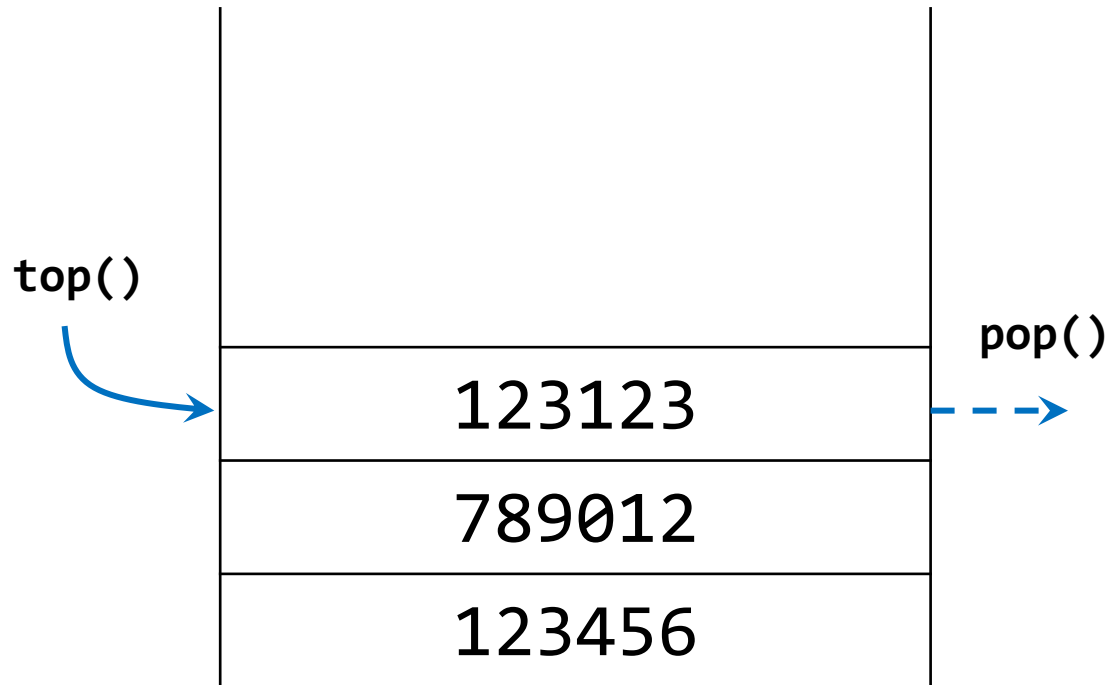
ADT\_Stack.cpp

```
template<typename T>
class Stack {
public:
    void push(T elem);
    T pop();
    T& top();

private:
    //внутренняя структура данных
}
```



# Интерфейс стека **pop()**

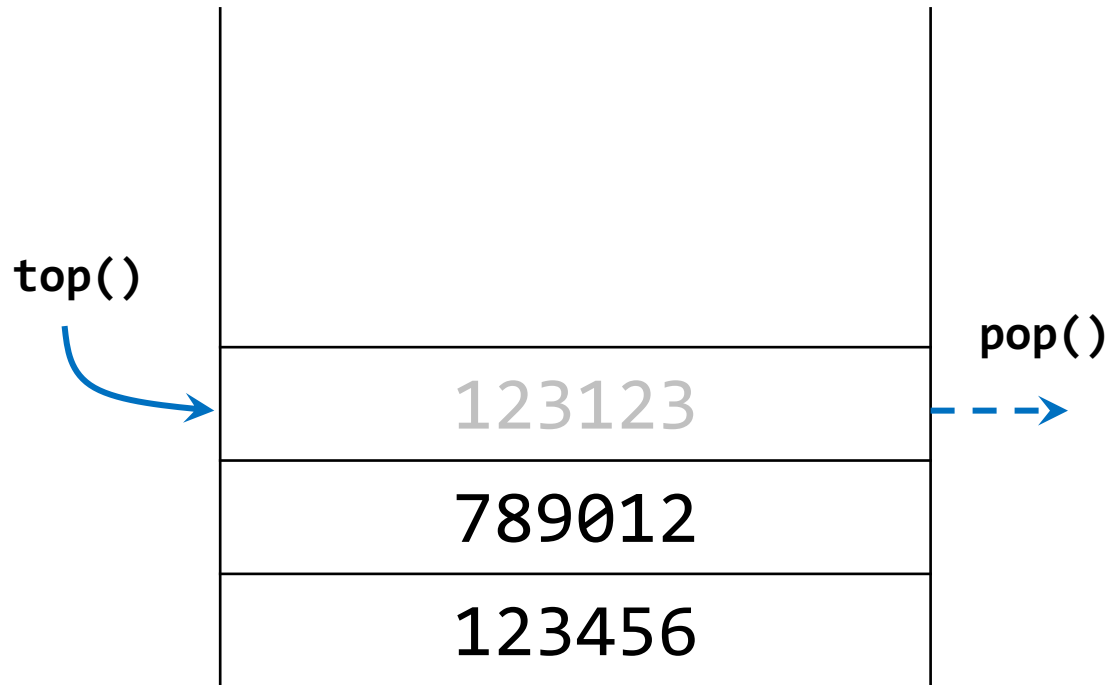



ADT\_Stack.cpp

```
template<typename T>
class Stack {
public:
    void push(T elem);
    T pop();
    T& top();

private:
    //внутренняя структура данных
}
```

# Интерфейс стека **pop()**

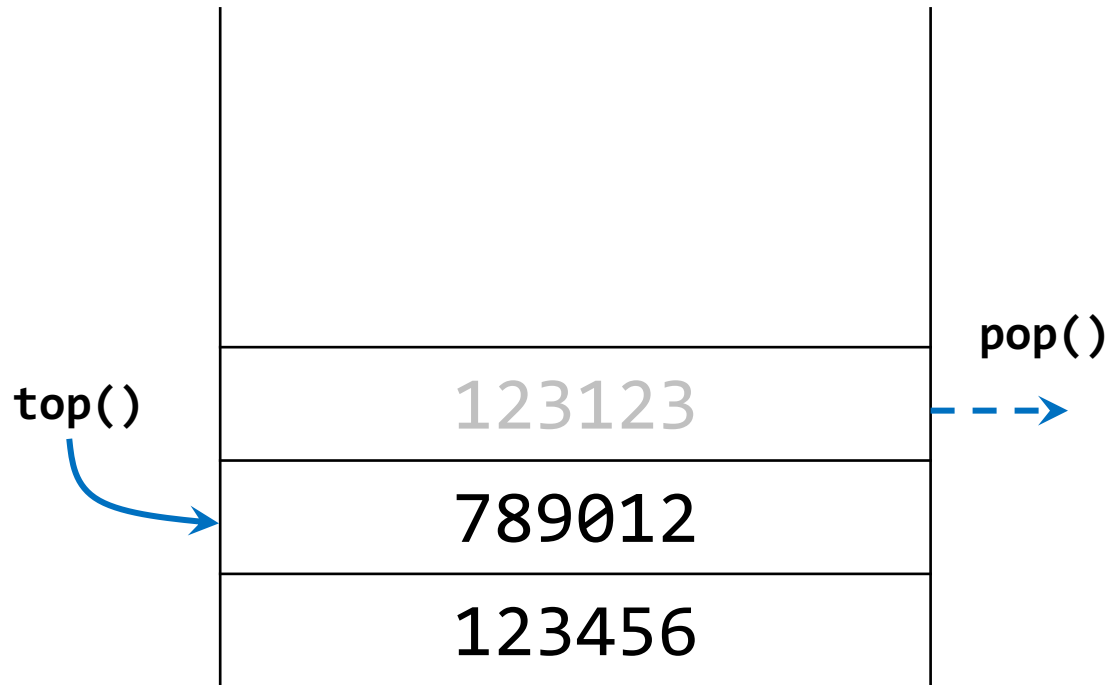


 ADT\_Stack.cpp

```
template<typename T>
class Stack {
public:
    void push(T elem);
    T pop();
    T& top();

private:
    //внутренняя структура данных
}
```

# Интерфейс стека **pop()**



ADT\_Stack.cpp

```
template<typename T>
class Stack {
public:
    void push(T elem);
    T pop();
    T& top();

private:
    //внутренняя структура данных
}
```

# Интерфейс стека – ошибки

789012
123456

Ошибка вставки в стек

**StackOverflow**

Ошибка удаления из стека

**StackUnderflow**

Ошибка чтения вершины

**StackIsEmpty**

# Правильная расстановка скобок в выражении

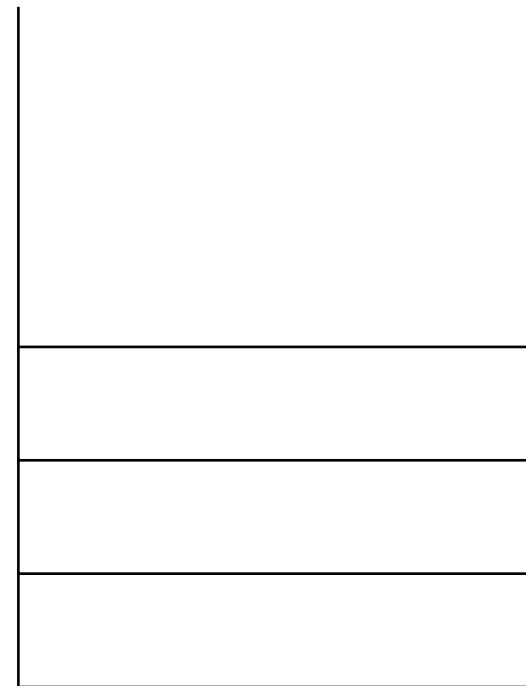
# Правильная скобочная последовательность

**Вход:** Строка **str**, состоящая из конечного числа открывающих '(' и закрывающих ')' скобок.

**Выход:** **YES**, если скобки расставлены верно;  
**NO** – в противном случае

# Проверка расстановки скобок

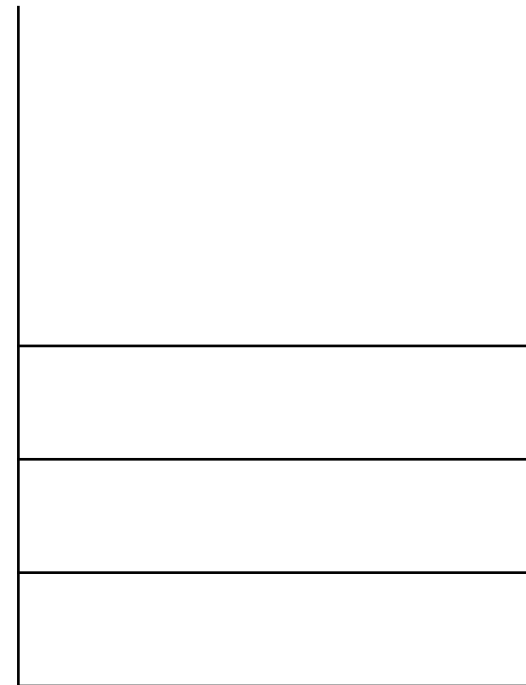
str ( ( ) ) ( )



Stack<char>

# Проверка расстановки скобок

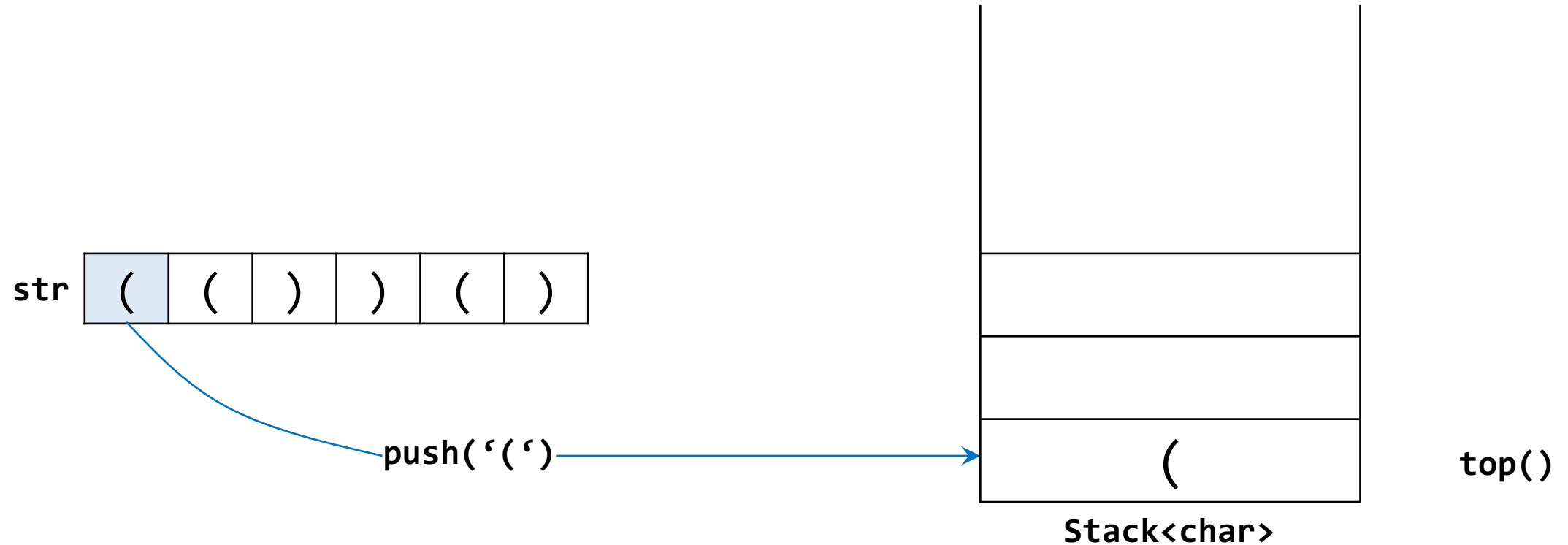
str ( ( ) ) ( )



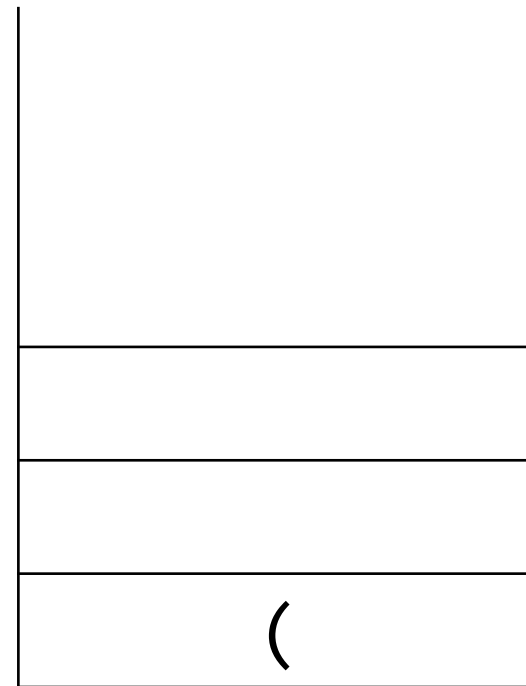
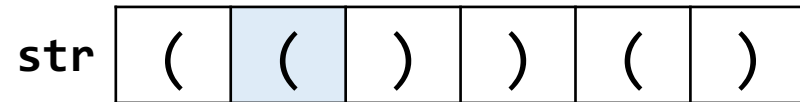
Stack<char>



# Проверка расстановки скобок

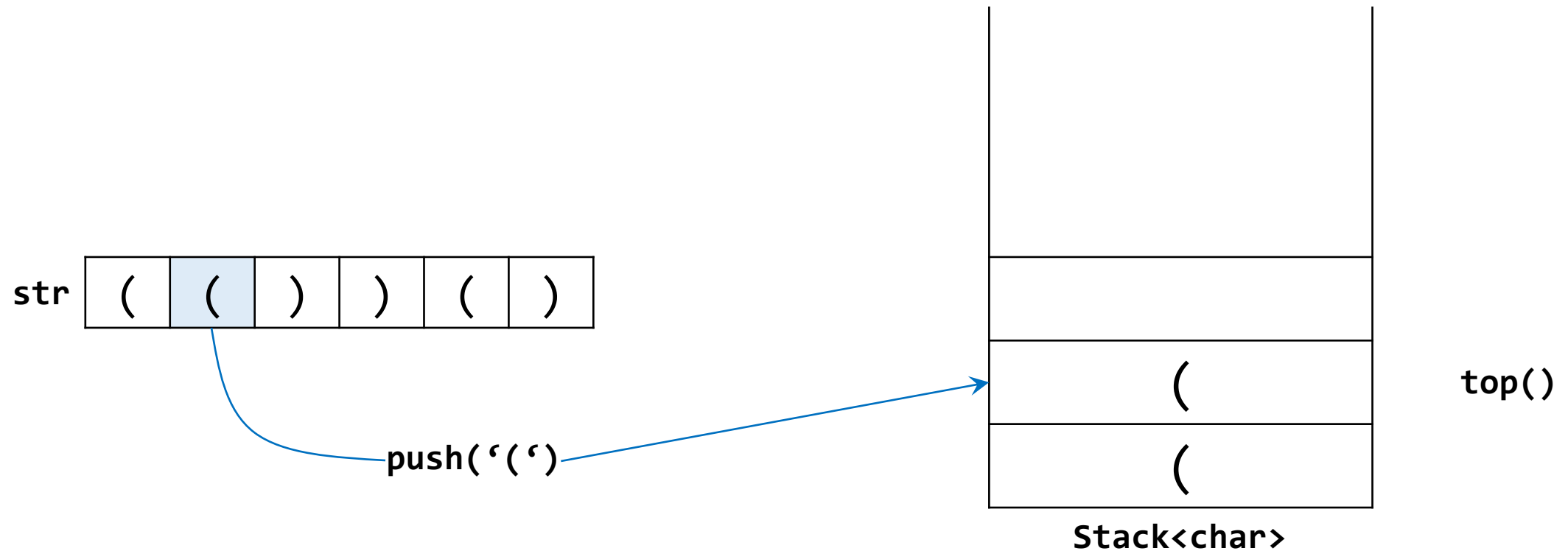


# Проверка расстановки скобок

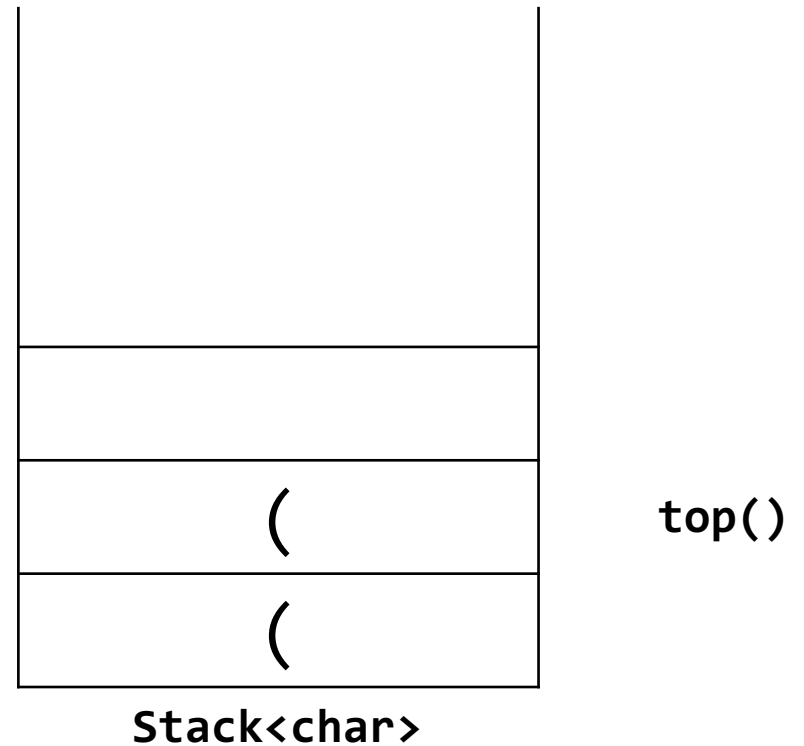
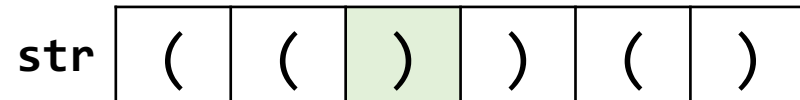


Stack<char>

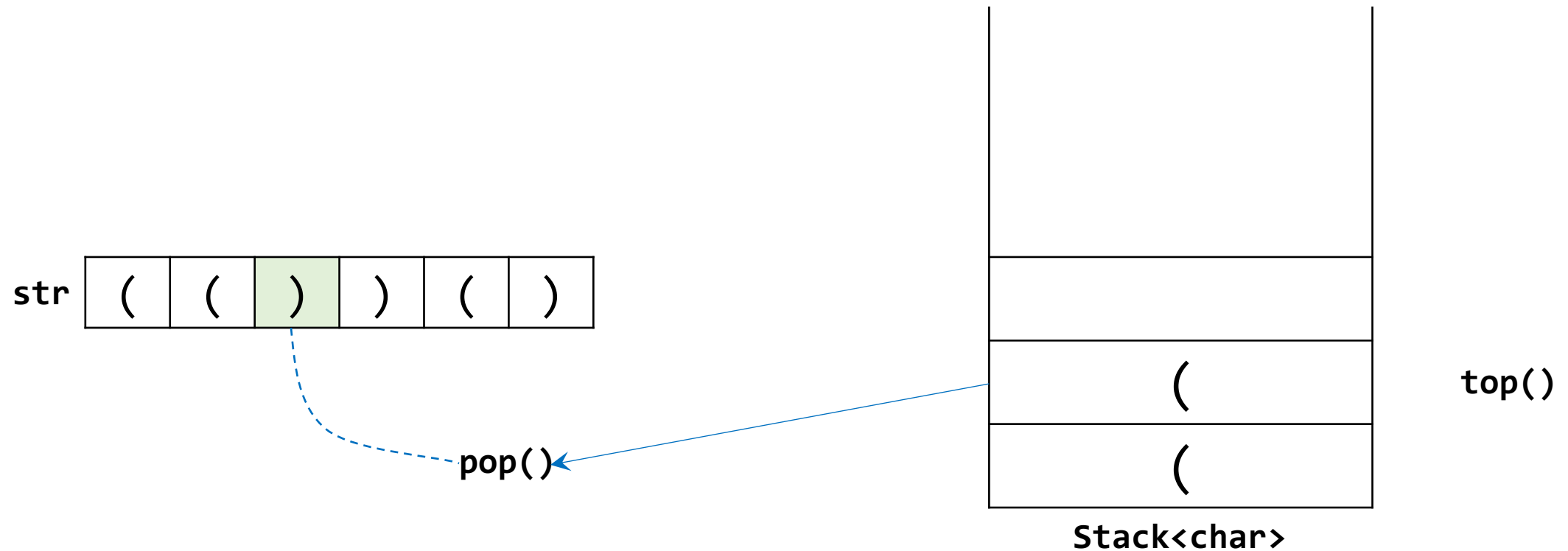
# Проверка расстановки скобок



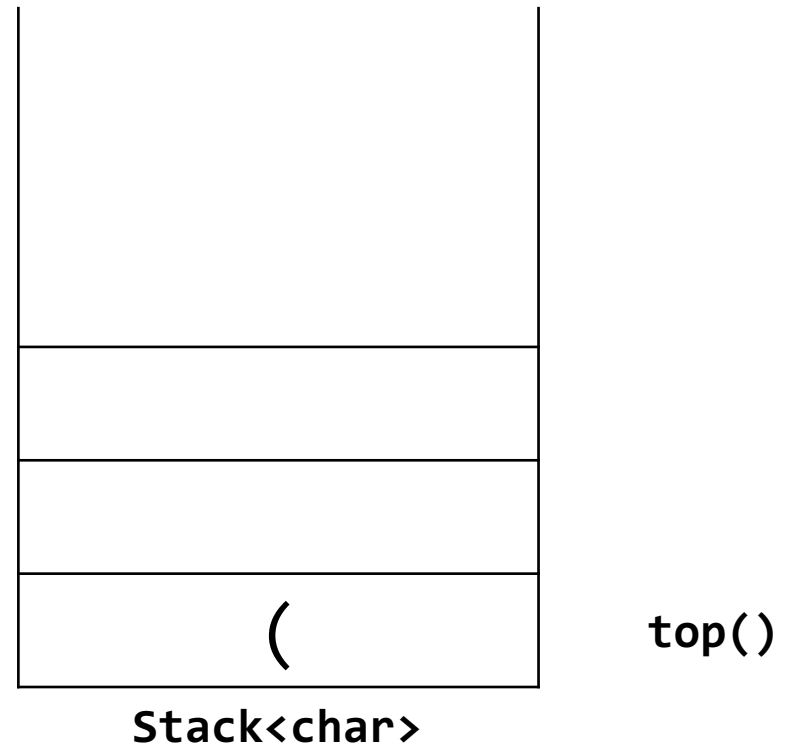
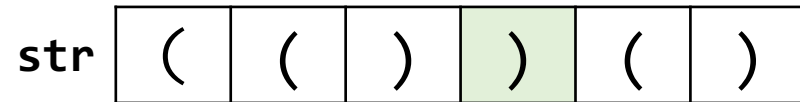
# Проверка расстановки скобок



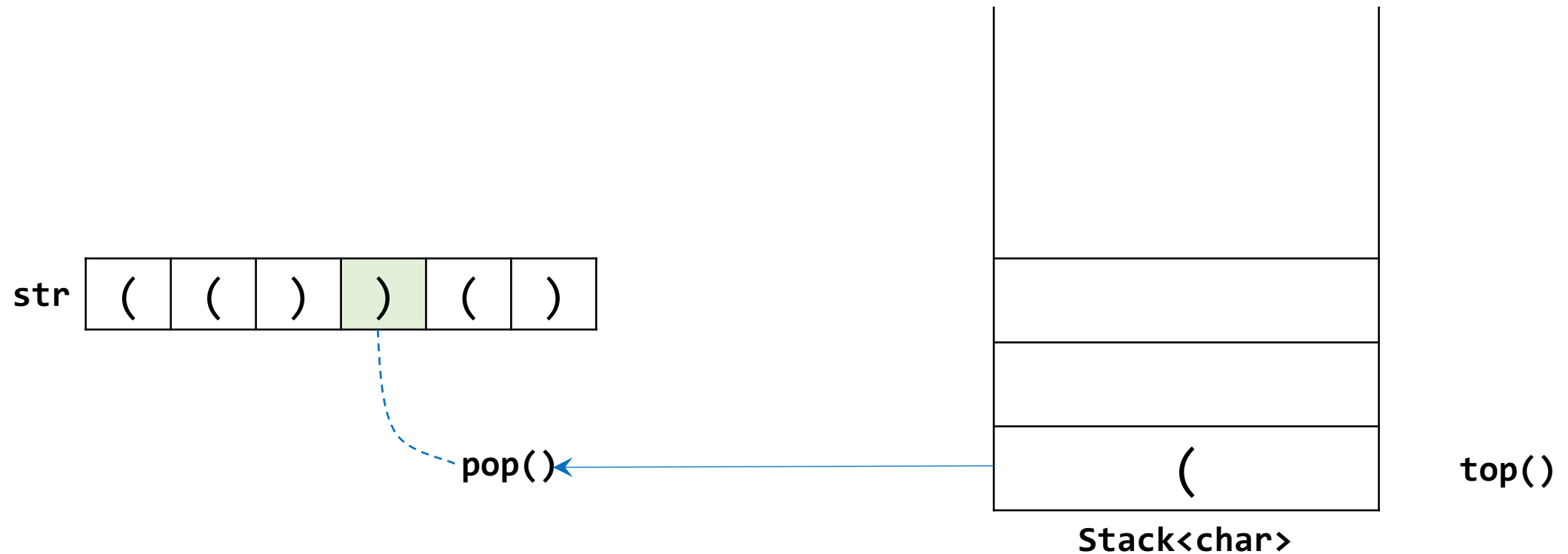
# Проверка расстановки скобок



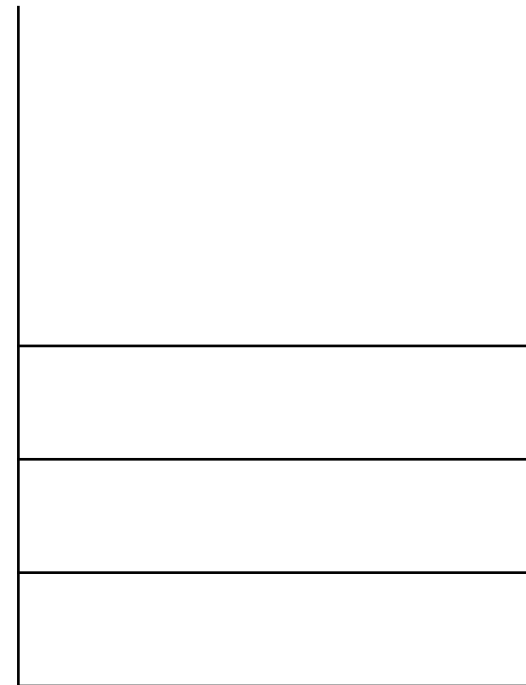
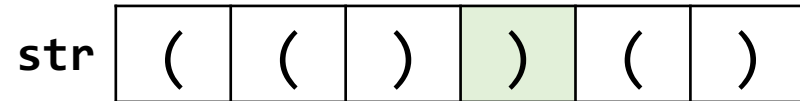
# Проверка расстановки скобок



# Проверка расстановки скобок



# Проверка расстановки скобок



Stack<char>



# Проверка расстановки скобок

Когда скобки расставлены верно?

# Проверка расстановки скобок

Когда скобки расставлены верно?

Прочитана **вся входная строка** `str`, а в стеке  
**не осталось** открывающих скобок

# Проверка расстановки скобок

Когда скобки расставлены верно?

Прочитана **вся входная строка** `str`, а в стеке  
**не осталось** открывающих скобок

Как характеризуется сложность алгоритма?

# Проверка расстановки скобок

Когда скобки расставлены верно?

Прочитана **вся входная строка** `str`, а в стеке  
**не осталось** открывающих скобок

Как характеризуется сложность алгоритма?

$\Theta(n)$  по времени и  $\Theta(n)$  для стека (по памяти)  
 $n$  — длина входной строки `str`

# Проверка расстановки скобок

Когда скобки расставлены верно?

Прочитана **вся входная строка** `str`, а в стеке  
**не осталось** открывающих скобок

Как характеризуется сложность алгоритма?

$\Theta(n)$  по времени и  $\Theta(n)$  для стека (по памяти)  
 $n$  — длина входной строки `str`

можно реализовать и проще, но применение стека дает наиболее универсальный способ решения этой задачи

# Применение стека

Организация рекурсивных вызовов

Операции Undo/Redo

Вычисление значения арифметического выражения в различных нотациях

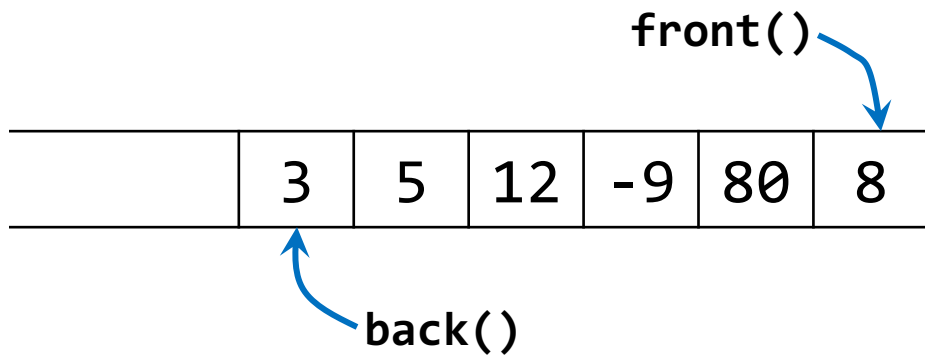
Алгоритмы с возвратом backtracking

...

# ADT Очередь

## First In First Out

# Интерфейс очереди

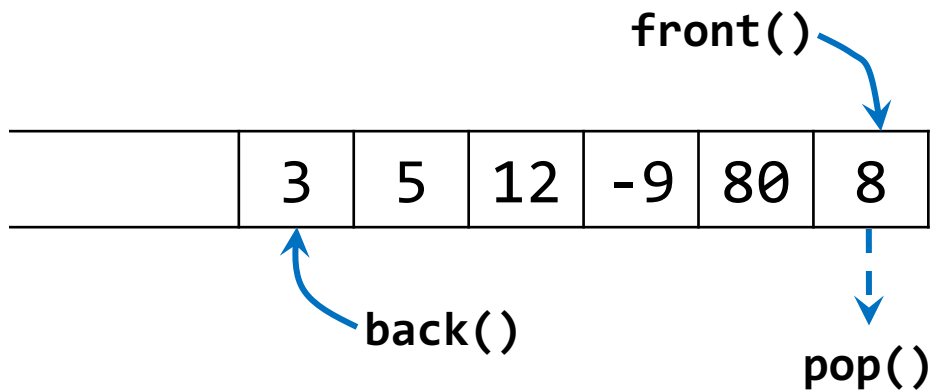


ADT\_Queue.cpp

```
template<typename T>
class Queue {
public:
    void push(T elem);
    T pop();
    T& front();
    T& back();
private:
    // внутренняя структура данных
}
```



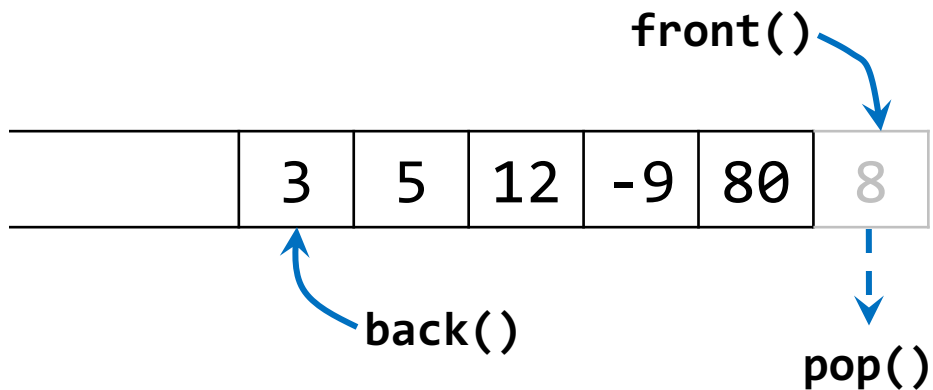
# Интерфейс очереди



```
ADT_Queue.cpp

template<typename T>
class Queue {
public:
    void push(T elem);
    T pop();
    T& front();
    T& back();
private:
    // внутренняя структура данных
}
```

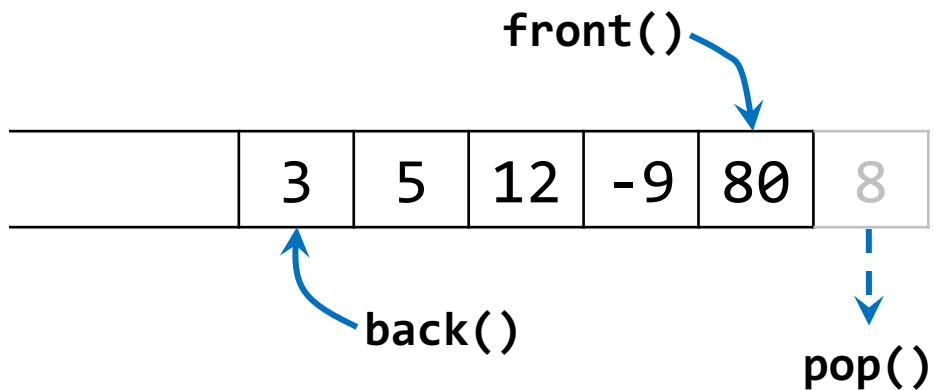
# Интерфейс очереди



```
ADT_Queue.cpp

template<typename T>
class Queue {
public:
    void push(T elem);
    T pop();
    T& front();
    T& back();
private:
    // внутренняя структура данных
}
```

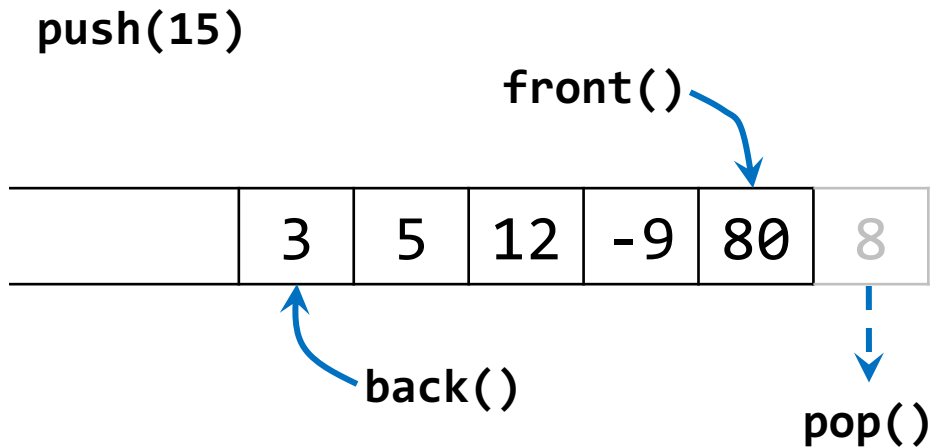
# Интерфейс очереди



ADT\_Queue.cpp

```
template<typename T>
class Queue {
public:
    void push(T elem);
    T pop();
    T& front();
    T& back();
private:
    // внутренняя структура данных
}
```

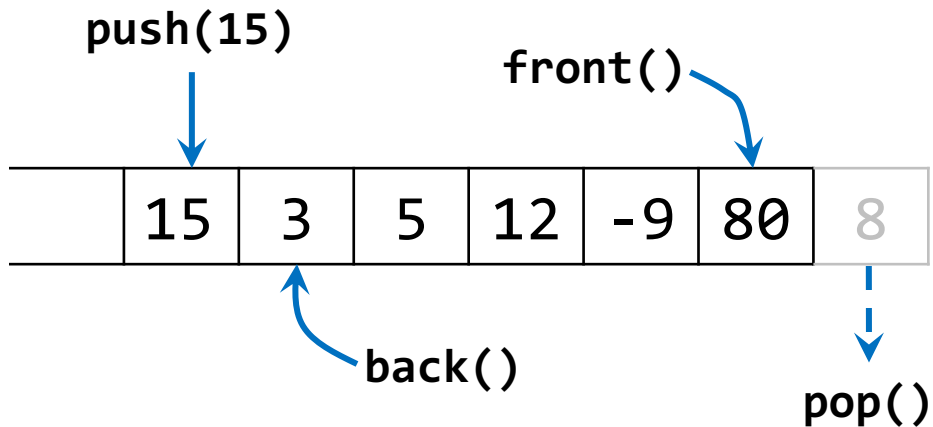
# Интерфейс очереди



```
ADT_Queue.cpp

template<typename T>
class Queue {
public:
    void push(T elem);
    T pop();
    T& front();
    T& back();
private:
    // внутренняя структура данных
}
```

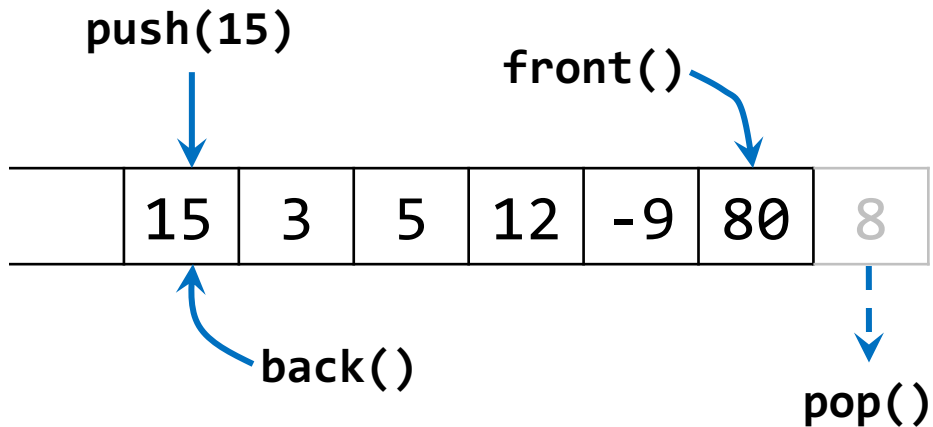
# Интерфейс очереди



```
ADT_Queue.cpp

template<typename T>
class Queue {
public:
    void push(T elem);
    T pop();
    T& front();
    T& back();
private:
    // внутренняя структура данных
}
```

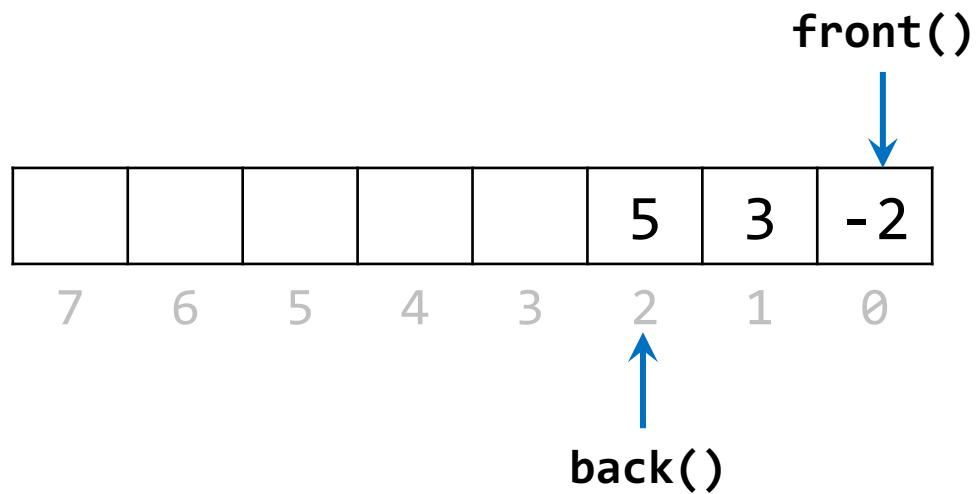
# Интерфейс очереди



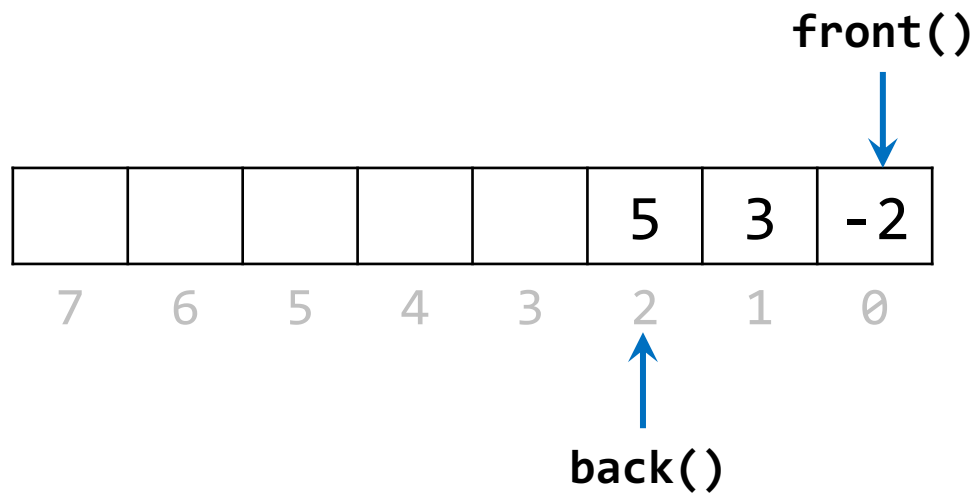
```
ADT_Queue.cpp

template<typename T>
class Queue {
public:
    void push(T elem);
    T pop();
    T& front();
    T& back();
private:
    // внутренняя структура данных
}
```

# Очередь на простом массиве



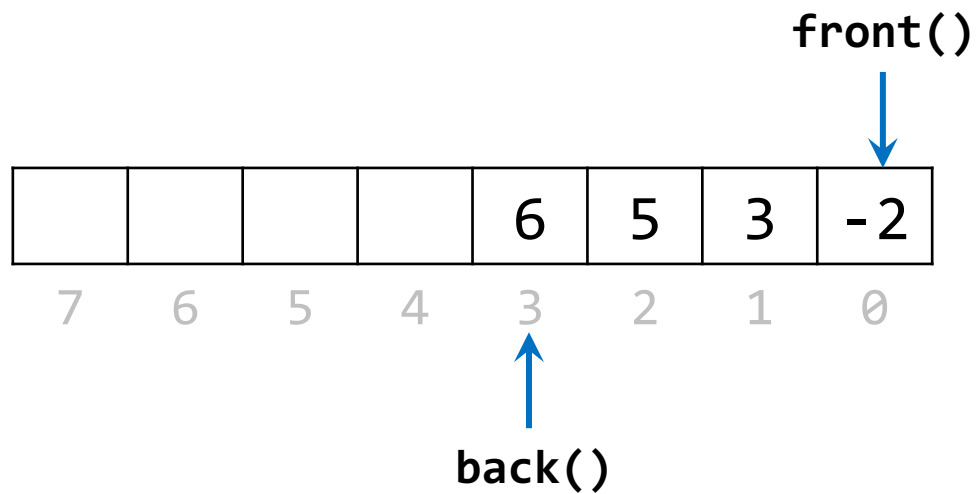
# Очередь на простом массиве



`push(6)`



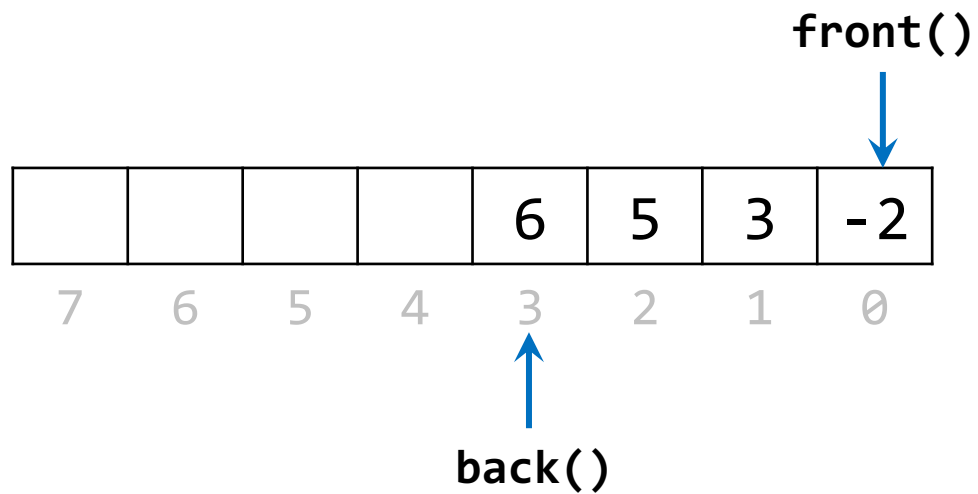
# Очередь на простом массиве



push(6)

back += 1

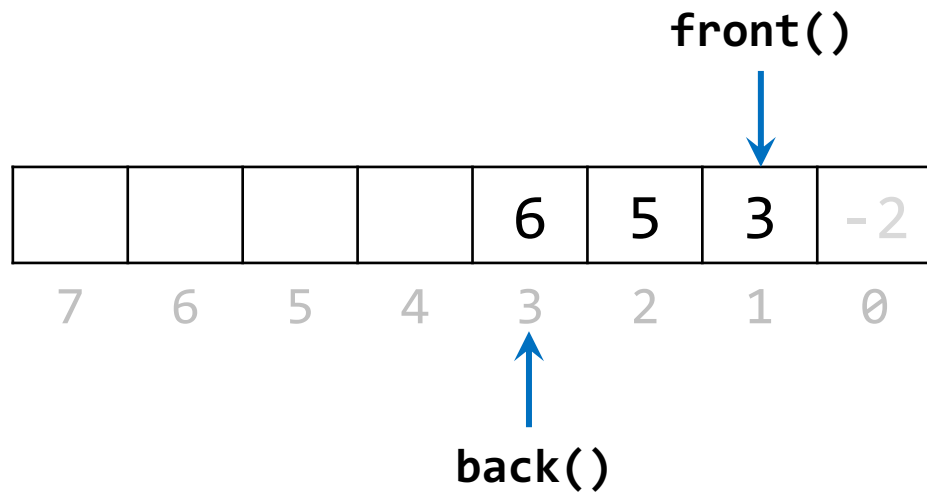
# Очередь на простом массиве



`push(6)`  
`pop()`

`back += 1`

# Очередь на простом массиве



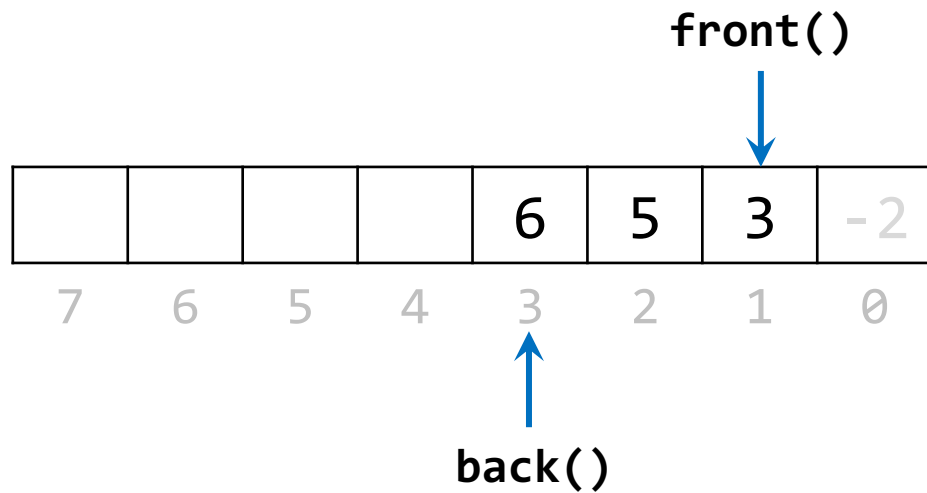
`push(6)`

`pop()`

`back += 1`

`front += 1`

# Очередь на простом массиве



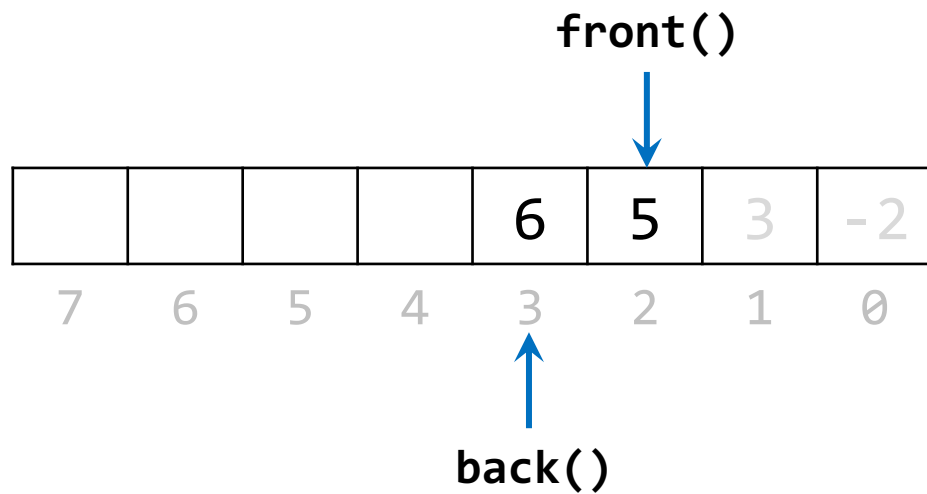
`push(6)`

`pop()`

`back += 1`

`front += 1`

# Очередь на простом массиве



push(6)

back += 1

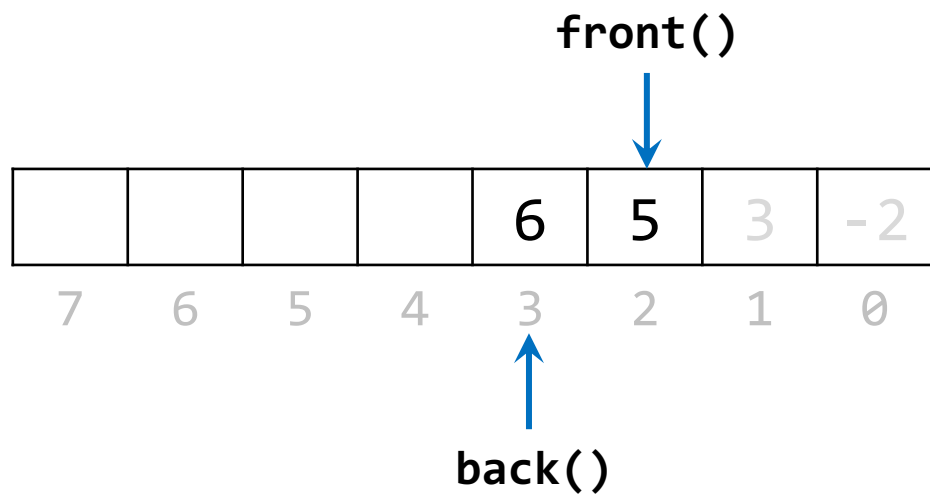
pop()

front += 1

pop()

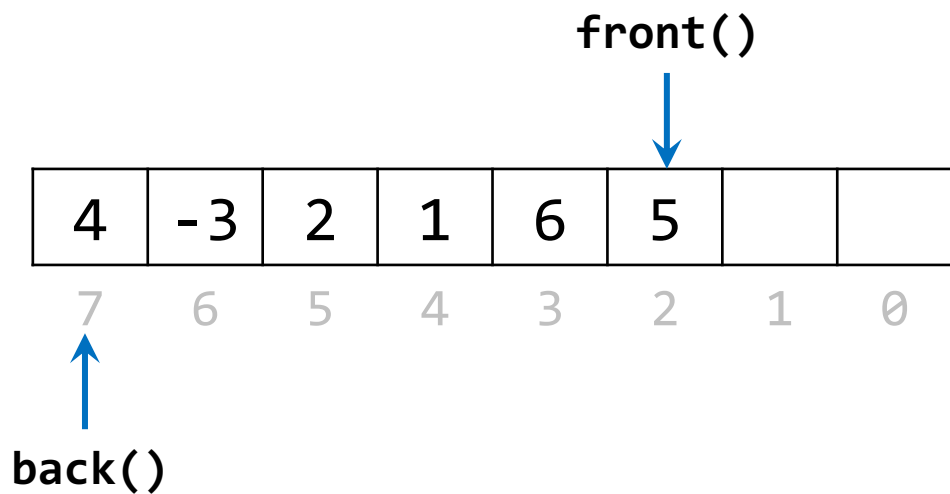
front += 1

# Очередь на простом массиве



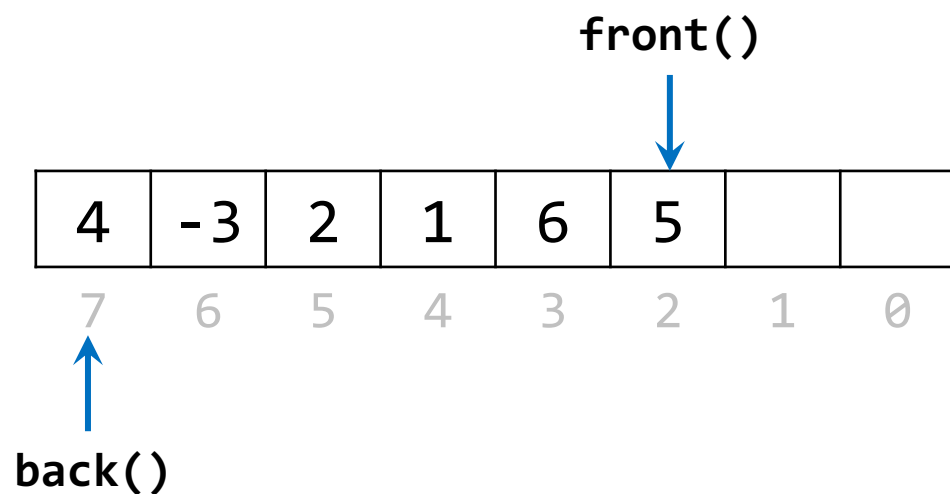
push(6)	back += 1
pop()	front += 1
pop()	front += 1
push(1)	back += 1
push(2)	back += 1
push(-3)	back += 1
push(4)	back += 1

# Очередь на простом массиве



<code>push(6)</code>	<code>back += 1</code>
<code>pop()</code>	<code>front += 1</code>
<code>pop()</code>	<code>front += 1</code>
<code>push(1)</code>	<code>back += 1</code>
<code>push(2)</code>	<code>back += 1</code>
<code>push(-3)</code>	<code>back += 1</code>
<code>push(4)</code>	<code>back += 1</code>

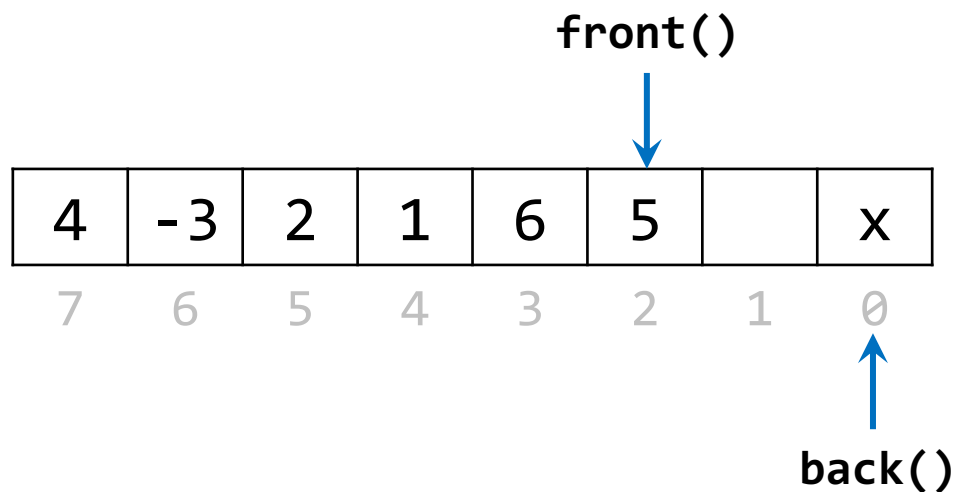
# Очередь на простом массиве



push(6)	back += 1
pop()	front += 1
pop()	front += 1
push(1)	back += 1
push(2)	back += 1
push(-3)	back += 1
push(4)	back += 1
push(x)	???



# Очередь на простом массиве



```
push(6)           back += 1
pop()             front += 1
pop()             front += 1
push(1)           back += 1
push(2)           back += 1
push(-3)          back += 1
push(4)           back += 1
push(x)           back = (back + 1) % n
```

# Применение очереди

Поиск в ширину (BFS) в графовых структурах данных

Выделение времени процессора

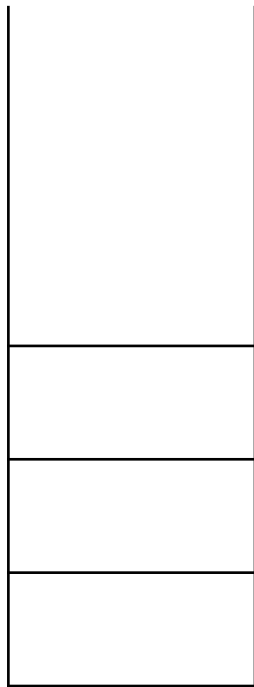
Асинхронная коммуникация

Сетевые протоколы

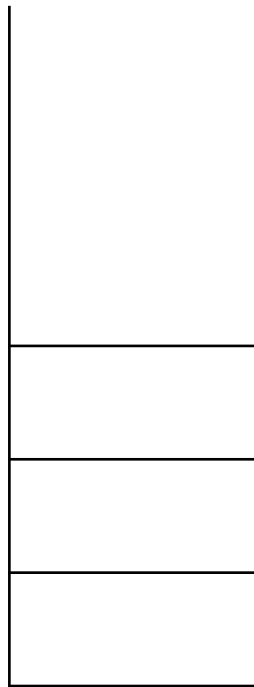
...

ADT Очередь на двух стеках

# Очередь – реализация №2



pushStack

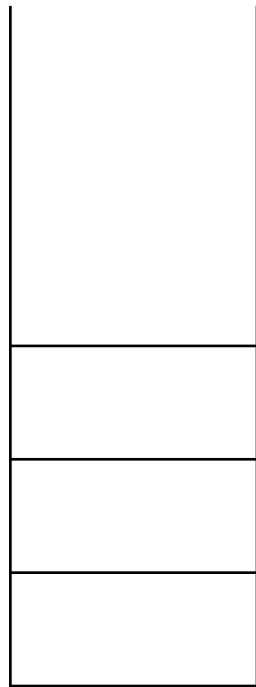


popStack

ADT\_Queue.cpp

```
template<typename T>
class Queue {
public:
    void push(T elem);
    T pop();
    T& front();
    T& back();
private:
    Stack<T> pushStack;
    Stack<T> popStack;
}
```

# Очередь – реализация №2



**pushStack**

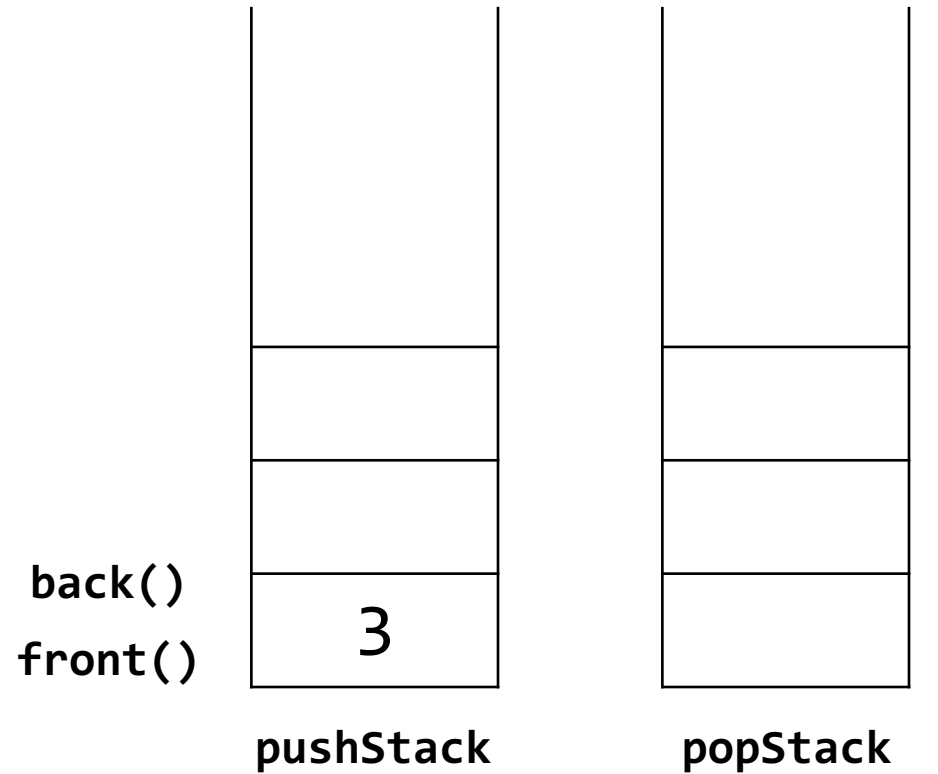


**popStack**

push(3)

**pushStack.push(3)**

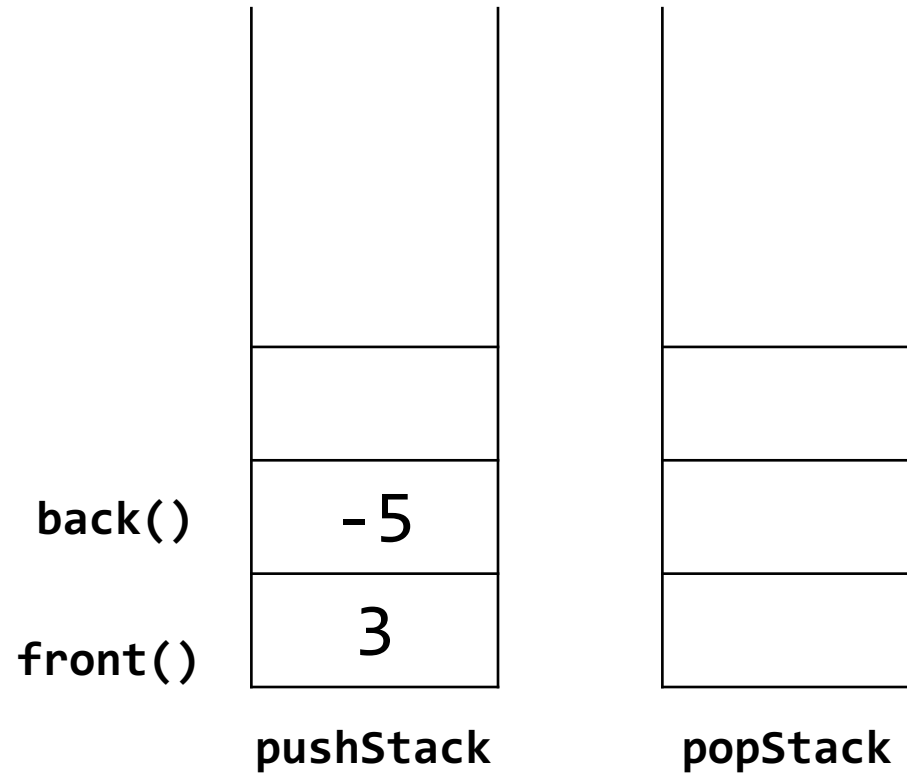
# Очередь – реализация №2



`push(3)`

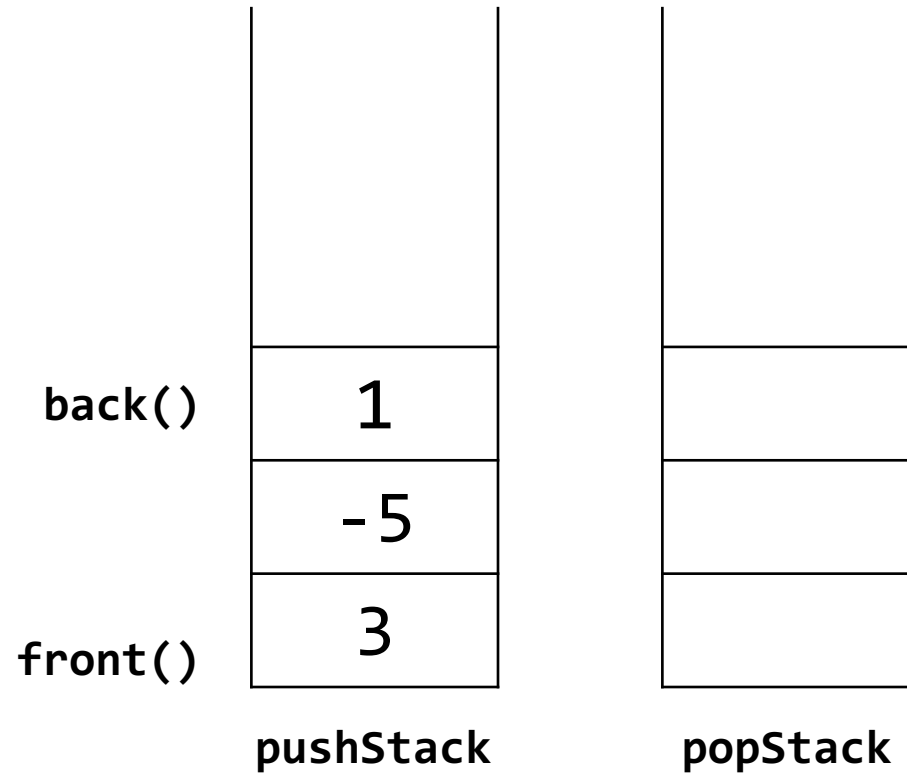
`pushStack.push(3)`

# Очередь – реализация №2



<code>push(3)</code>	<code>pushStack.push(3)</code>
<code>push(-5)</code>	<code>pushStack.push(-5)</code>

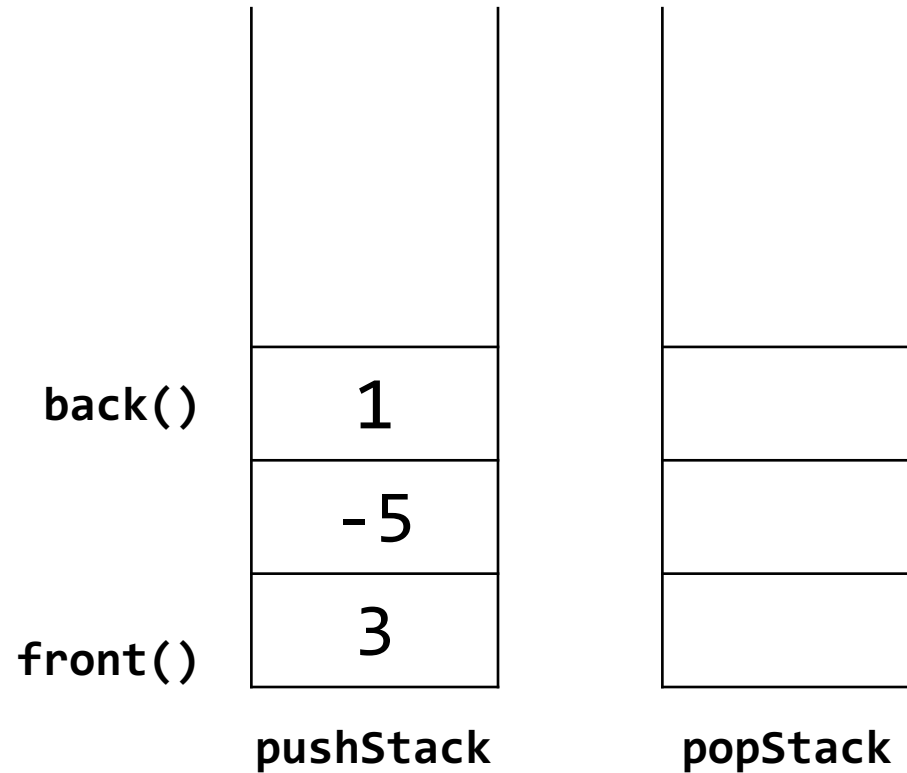
# Очередь – реализация №2



<code>push(3)</code>	<code>pushStack_.push(3)</code>
<code>push(-5)</code>	<code>pushStack_.push(-5)</code>
<code>push(1)</code>	<code>pushStack_.push(1)</code>

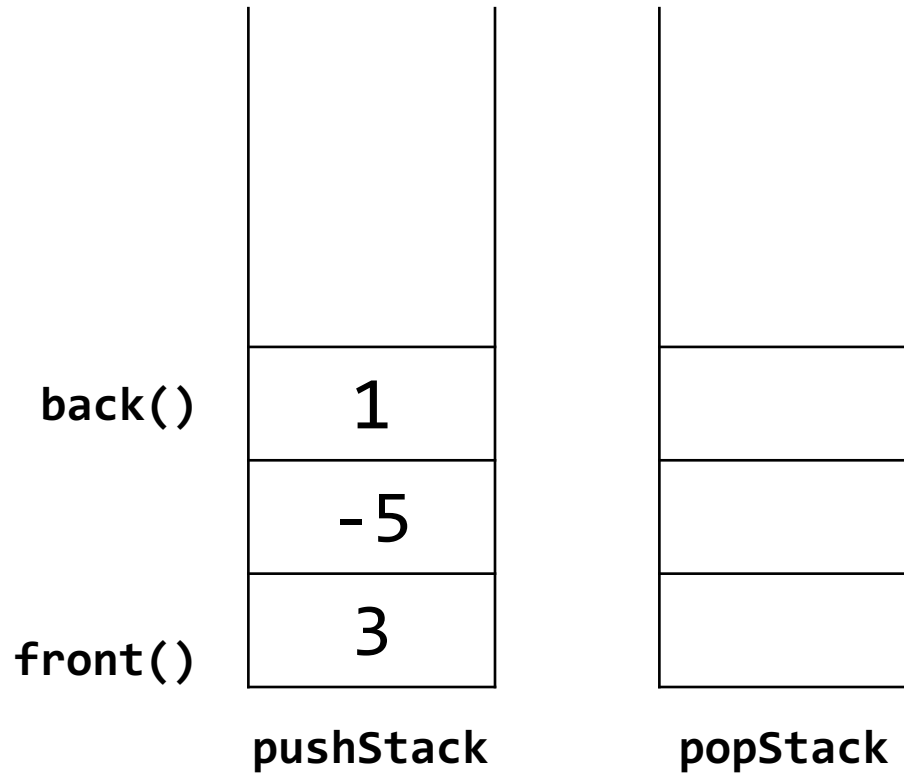


# Очередь – реализация №2



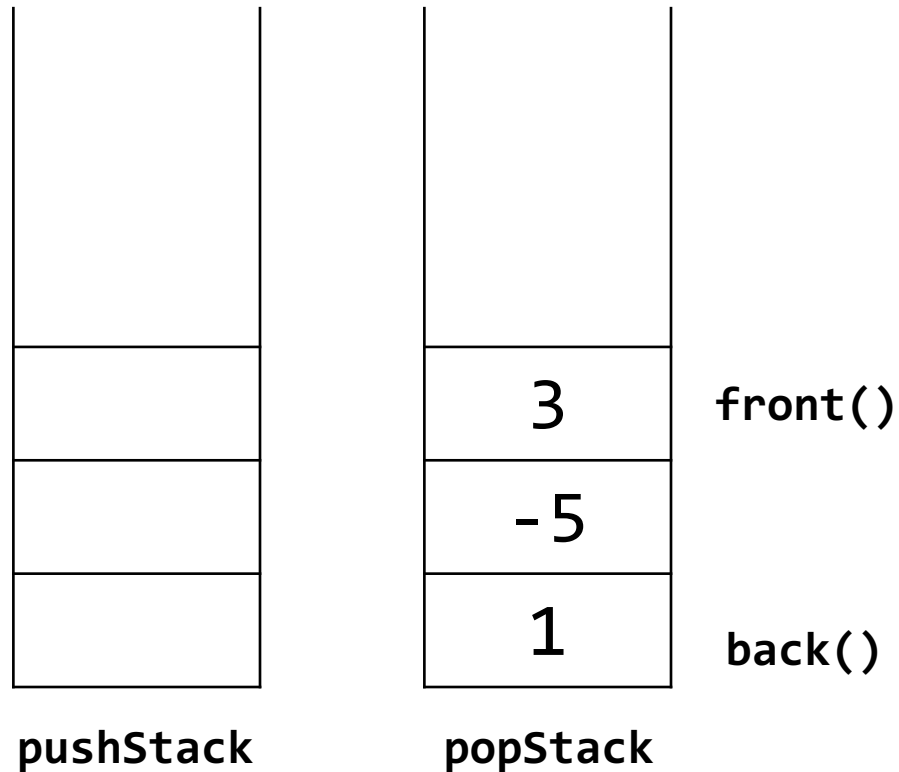
<code>push(3)</code>	<code>pushStack_.push(3)</code>
<code>push(-5)</code>	<code>pushStack_.push(-5)</code>
<code>push(1)</code>	<code>pushStack_.push(1)</code>
<code>pop()</code>	

# Очередь – реализация №2



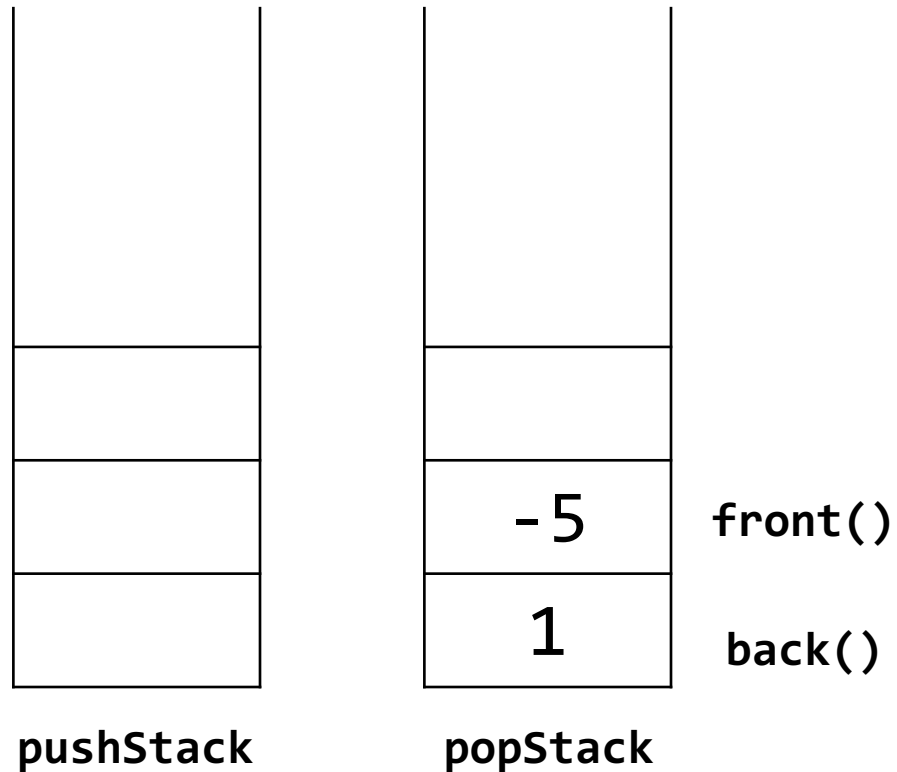
push(3)	<code>pushStack_.push(3)</code>
push(-5)	<code>pushStack_.push(-5)</code>
push(1)	<code>pushStack_.push(1)</code>
pop()	<code>popStack.push(pushStack.pop())</code>
	<code>popStack.push(pushStack.pop())</code>
	<code>popStack.push(pushStack.pop())</code>
	<code>popStack.push(pushStack.pop())</code>

# Очередь – реализация №2



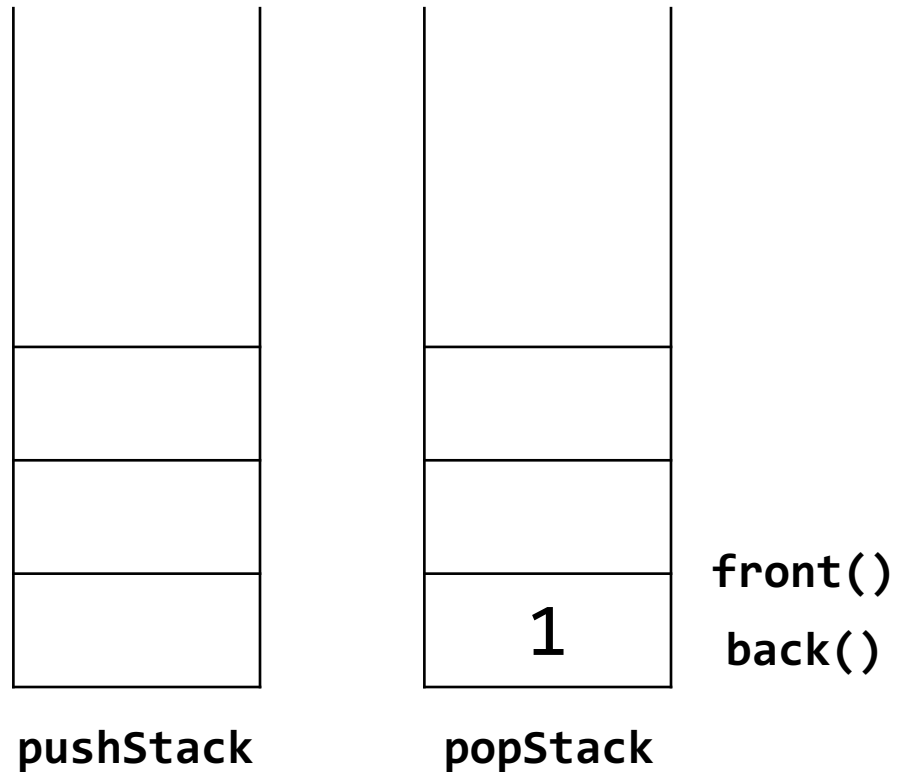
<code>push(3)</code>	<code>pushStack_.push(3)</code>
<code>push(-5)</code>	<code>pushStack_.push(-5)</code>
<code>push(1)</code>	<code>pushStack_.push(1)</code>
<code>pop()</code>	<code>popStack.push(pushStack.pop())</code>
	<code>popStack.push(pushStack.pop())</code>
	<code>popStack.push(pushStack.pop())</code>

# Очередь – реализация №2



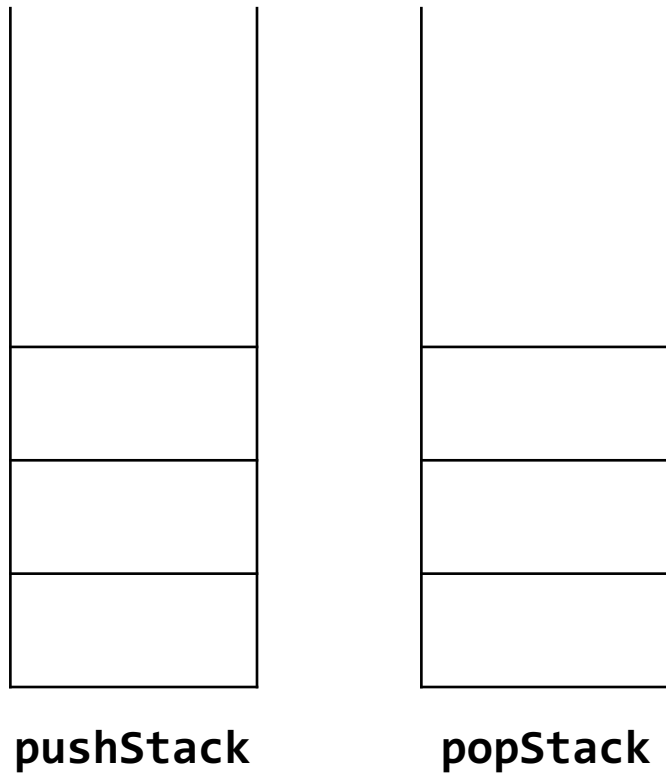
push(3)	<b>pushStack_.push(3)</b>
push(-5)	<b>pushStack_.push(-5)</b>
push(1)	<b>pushStack_.push(1)</b>
pop()	<b>popStack.push(pushStack.pop())</b>
	<b>popStack.push(pushStack.pop())</b>
	<b>popStack.push(pushStack.pop())</b>
	<b>popStack.pop()</b>

# Очередь – реализация №2



<code>push(3)</code>	<code>pushStack_.push(3)</code>
<code>push(-5)</code>	<code>pushStack_.push(-5)</code>
<code>push(1)</code>	<code>pushStack_.push(1)</code>
<code>pop()</code>	<code>popStack.push(pushStack.pop())</code>
	<code>popStack.push(pushStack.pop())</code>
	<code>popStack.push(pushStack.pop())</code>
	<code>popStack.pop()</code>
<code>pop()</code>	<code>popStack.pop()</code>

# Очередь – реализация №2



<code>push(3)</code>	<code>pushStack_.push(3)</code>
<code>push(-5)</code>	<code>pushStack_.push(-5)</code>
<code>push(1)</code>	<code>pushStack_.push(1)</code>
<code>pop()</code>	<code>popStack.push(pushStack.pop())</code>
	<code>popStack.push(pushStack.pop())</code>
	<code>popStack.push(pushStack.pop())</code>
	<code>popStack.pop()</code>
<code>pop()</code>	<code>popStack.pop()</code>
<code>pop()</code>	<code>popStack.pop()</code>

# Очередь на двух стеках — СЛОЖНОСТЬ

		фактическая реализация	сложность	комментарий
Queue.push()		pushStack.push()	$\Theta(1)$	простое добавление объекта в стек
Queue.pop()	!popStack.empty()	popStack.pop()	$\Theta(1)$	простое удаление объекта из стека
	popStack.empty()	popStack.push(...) popStack.push(...) ... popStack.push(...) popStack.pop()	$\Theta(S) + \Theta(1)$ $S$ — размер стека для вставки	<ul style="list-style-type: none"><li>○ перекладывание из pushStack в popStack</li><li>○ удаление из стека</li></ul>

# Очередь на двух стеках — СЛОЖНОСТЬ

		фактическая реализация	сложность	комментарий
Queue.push()		pushStack.push()	$\Theta(1)$	простое добавление объекта в стек
Queue.pop()	!popStack.empty()	popStack.pop()	$\Theta(1)$	простое удаление объекта из стека
	popStack.empty()	popStack.push(...) popStack.push(...) ... popStack.push(...) popStack.pop()	$\Theta(S) + \Theta(1)$ $S$ — размер стека для вставки	<ul style="list-style-type: none"><li>○ перекладывание из pushStack в popStack</li><li>○ удаление из стека</li></ul>

для того, чтобы вычислить обобщенную оценку операции **Queue.pop()**, обратимся к способам **усреднения** временной сложности



## средняя сложность $\bar{T}$

- выборкой из **какого** **распределения** являются входные данные?
- какова **вероятность** того, что массив уже будет отсортирован?

....

$\bar{T}$  — **математическое ожидание** временной сложности

[пример на доске]

## амортизированная сложность $\dot{T}$

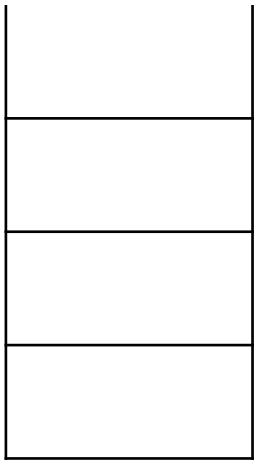
- предположения о входных данных **не выдвигаются**
- усреднение **влияния долгих** операций среди  $n$  операций
- амортизированная сложность:

$$\dot{c}(op) = t(op) + \text{амортизация, где } op \text{ — некая операция}$$

$\dot{T}_n = \sum_{i=1}^n \dot{c}(op_i)$  — **суммарная сложность** выполнения  $n$  операций

[пример далее...]

# Очередь на двух стеках — метод банкира



pushStack



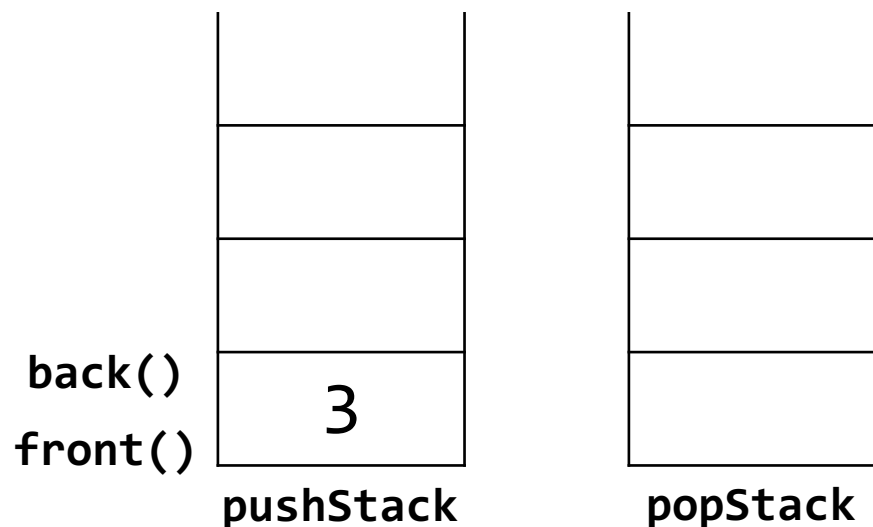
popStack

сложность стандартных операций со стеком:

$$t(\text{push}) = t(\text{pop}) = 1$$

операция $op$	амортизация		$\dot{c}(op)$
	отложено	потрачено	

# Очередь на двух стеках — метод банкира



сложность стандартных операций со стеком:

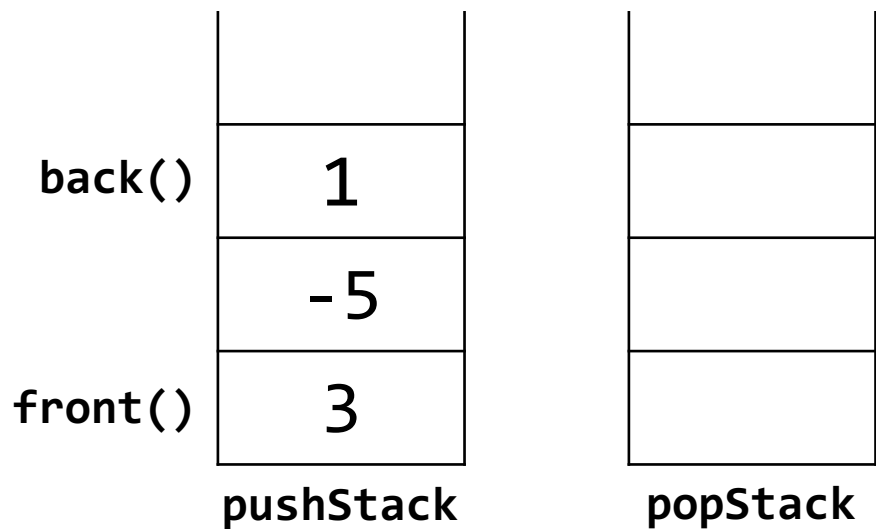
$$t(\text{push}) = t(\text{pop}) = 1$$

операция $op$	амортизация		$\dot{c}(op)$
	отложено	потрачено	
<code>push(3)</code>	1 1	—	$1 + 2 - 0 = 3$

при выполнении простых операций  
откладываем ресурсы для выполнения  
долгих операций впоследствии

амортизация — разница между  
отложенными и потраченными ресурсами

# Очередь на двух стеках — метод банкира

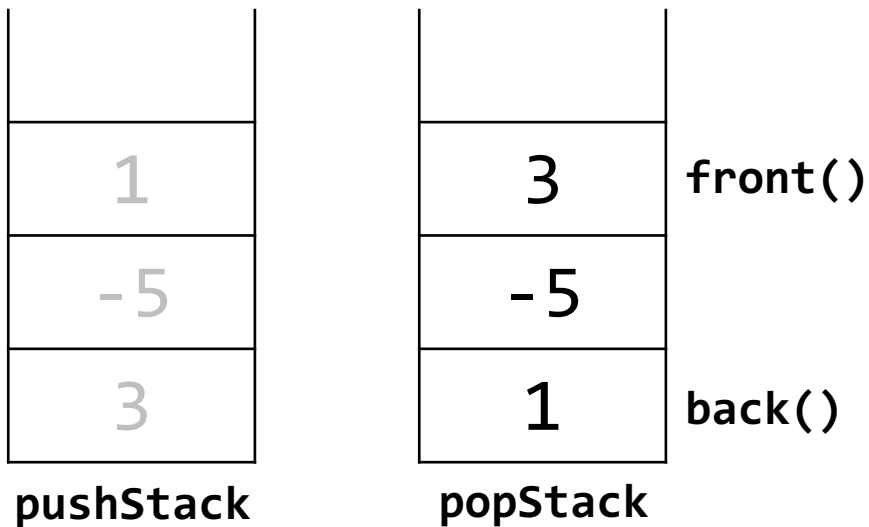


операция $op$	амортизация		$\dot{c}(op)$
	отложено	потрачено	
<code>push(3)</code>	1 1	—	$1 + 2 - 0 = 3$
<code>push(-5)</code>	1 1	—	$1 + 2 - 0 = 3$
<code>push(1)</code>	1 1	—	$1 + 2 - 0 = 3$

сложность стандартных операций со стеком:

$$t(\text{push}) = t(\text{pop}) = 1$$

# Очередь на двух стеках — метод банкира



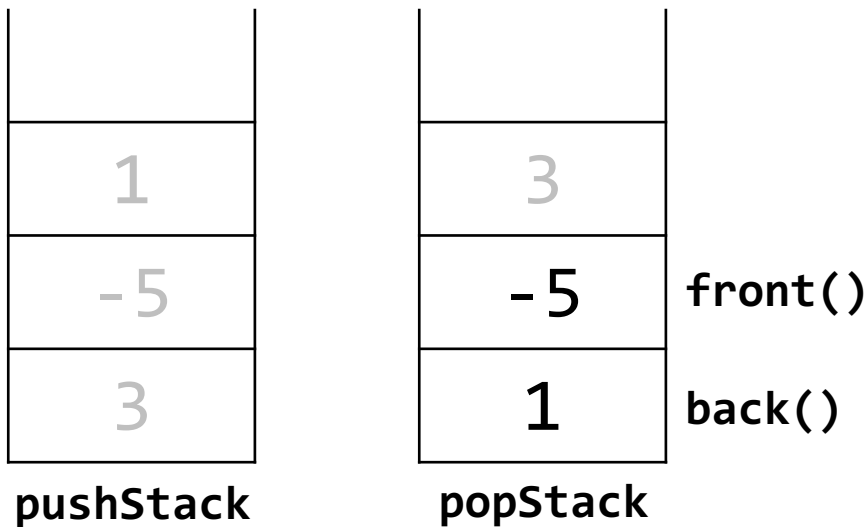
front()

back()

сложность стандартных операций со стеком:  
 $t(\text{push}) = t(\text{pop}) = 1$

операция <i>op</i>		амортизация		$\dot{c}(op)$
		отложено	потрачено	
push(3)		1 1	—	$1 + 2 - 0 = 3$
push(-5)		1 1	—	$1 + 2 - 0 = 3$
push(1)		1 1	—	$1 + 2 - 0 = 3$
pop()	pushStack.pop()	—	1	
	popStack.push(1)	—	1	
	pushStack.pop()	—	1	
	popStack.push(-5)	—	1	
	pushStack.pop()	—	1	
	popStack.push(3)	—	1	

# Очередь на двух стеках — метод банкира



front()

back()

сложность стандартных операций со стеком:  
 $t(\text{push}) = t(\text{pop}) = 1$

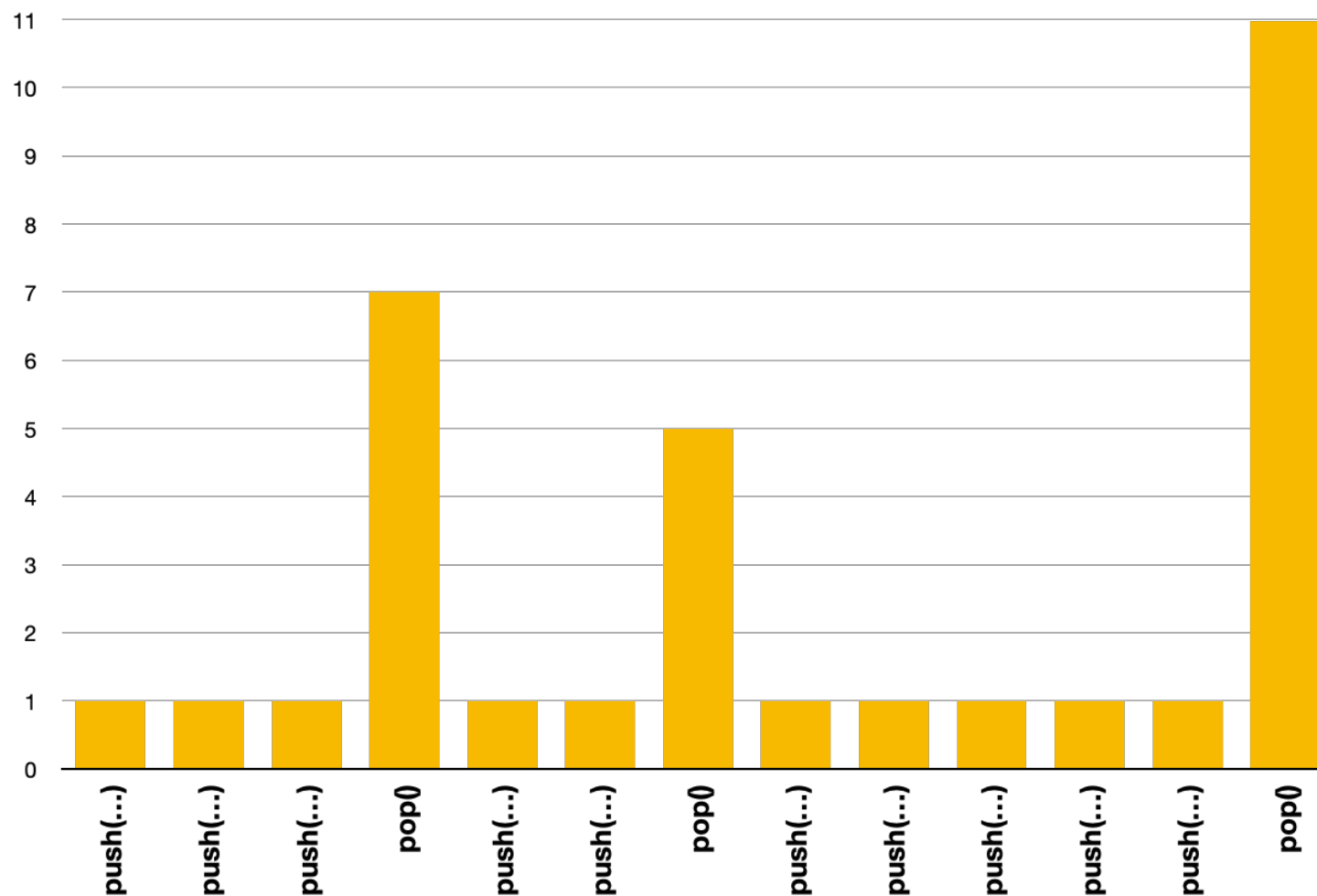
операция <i>op</i>		амортизация		$\dot{c}(op)$
		отложено	потрачено	
push(3)		1 1	—	$1 + 2 - 0 = 3$
push(-5)		1 1	—	$1 + 2 - 0 = 3$
push(1)		1 1	—	$1 + 2 - 0 = 3$
pop()	pushStack.pop()	—	1	$7 + 2 - 6 = 3$
	popStack.push(1)	—	1	
	pushStack.pop()	—	1	
	popStack.push(-5)	—	1	
	pushStack.pop()	—	1	
	popStack.push(3)	—	1	
	popStack.pop()	1 1	—	

# Очередь на двух стеках — СЛОЖНОСТЬ

		фактическая реализация	сложность	комментарий
Queue.push()		pushStack_.push()	$\Theta(1)$	простое добавление объекта в стек
Queue.pop()	!popStack_.empty()	popStack_.pop()	$\Theta(1)$	простое удаление объекта из стека
	popStack_.empty()	popStack_.push(...) popStack_.push(...) ... popStack_.push(...) popStack_.pop()	$\Theta(1)$ амортизированная	<ul style="list-style-type: none"><li>○ перекладывание из pushStack_ в popStack_</li><li>○ удаление из стека</li></ul>

динамику сложности выполнения последовательности операций вставки и удаления с очередью на двух стеках можно также условно представить в виде гистограммы

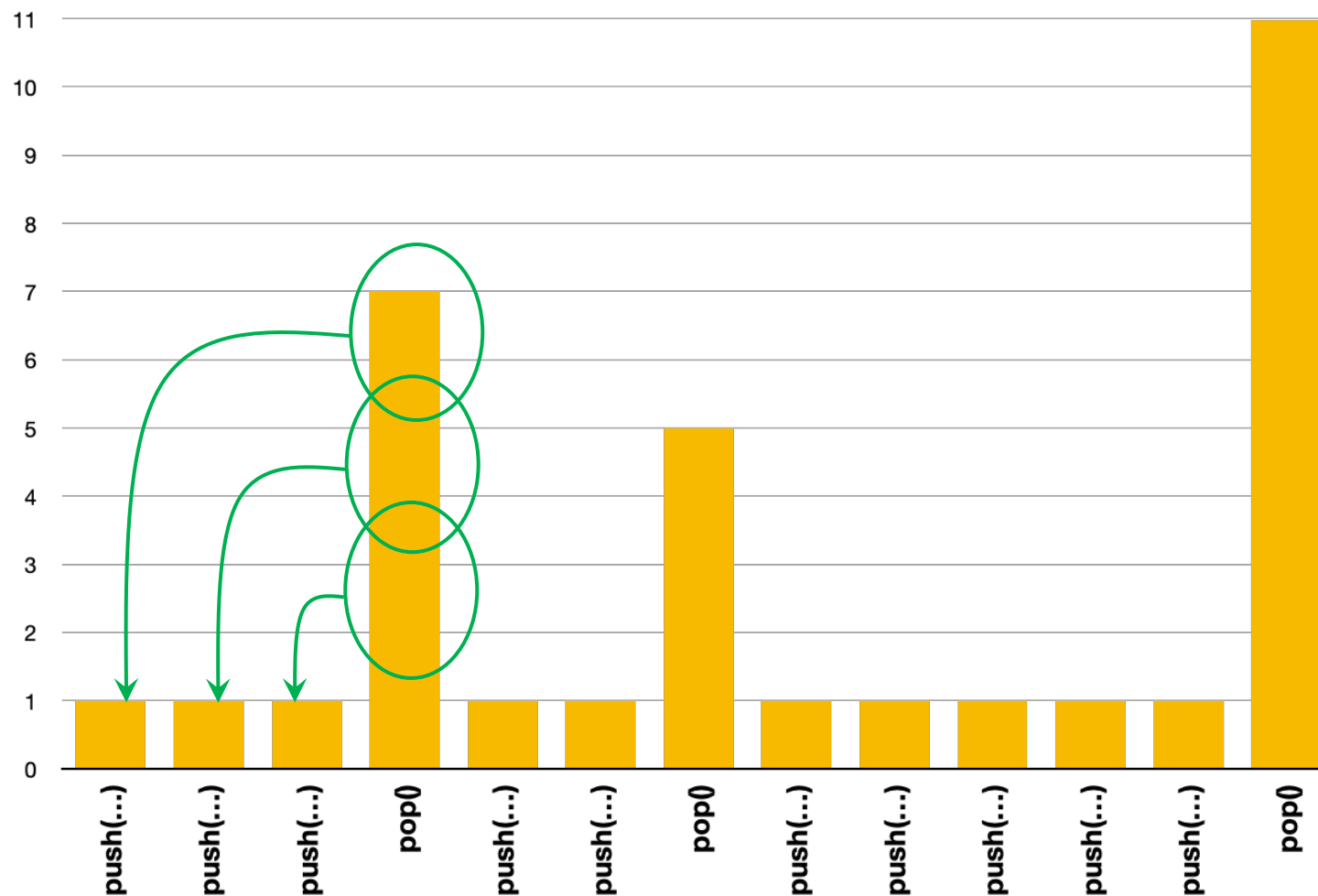
стандартные сложности основных операций  $t(op)$



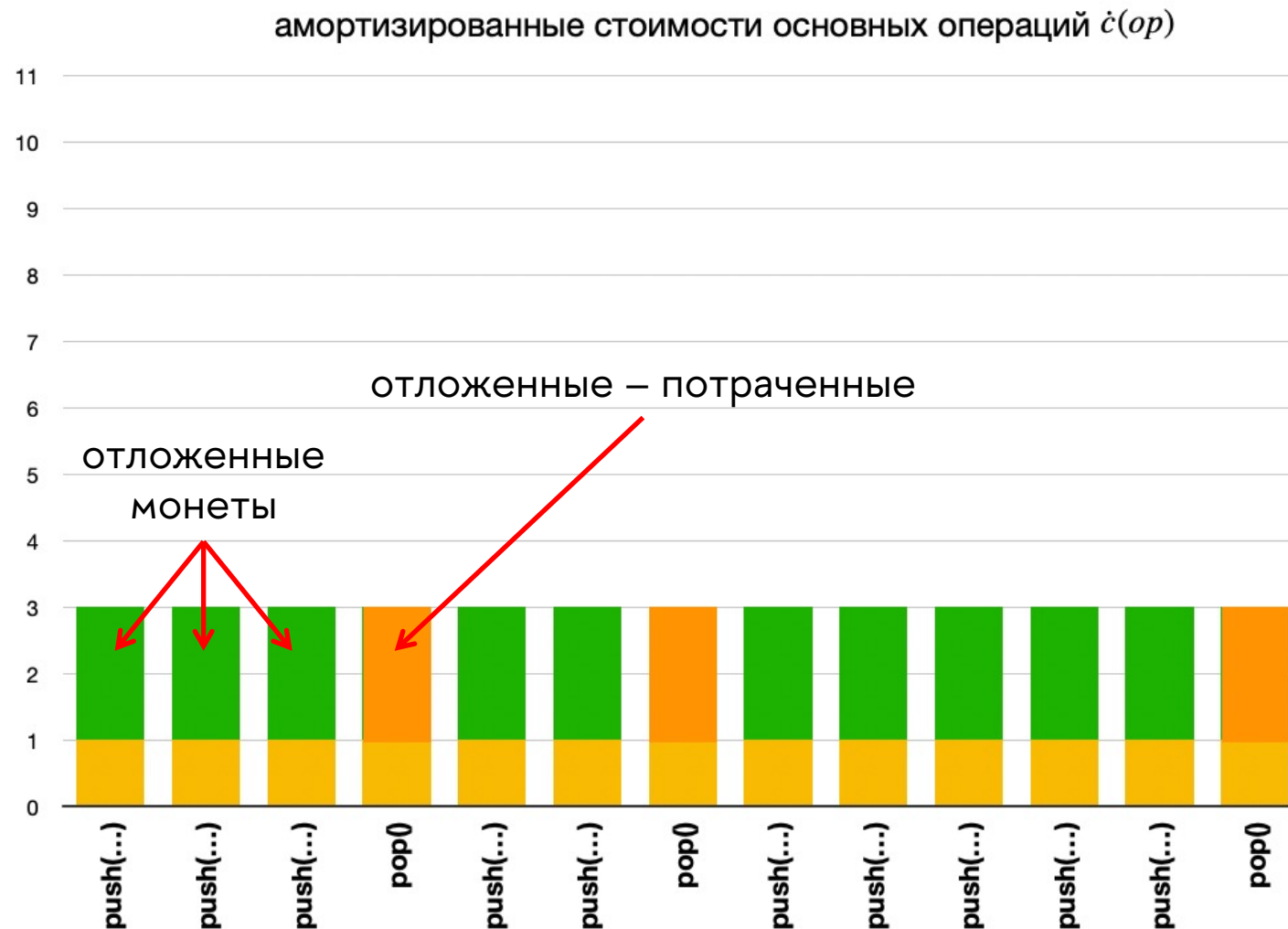


при амортизированном подсчете, мы интуитивно перераспределяем сложность долгих операций на сложность предшествующих быстрых операций

стандартные сложности основных операций  $t(op)$



при амортизированном подсчете, мы интуитивно перераспределяем сложность долгих операций на сложность предшествующих быстрых операций



# ADT Стек/Очередь

## ассоциативные операции

# Стек с поддержкой **минимума**

- ассоциативная операция:  $\min(a, \min(b, c)) = \min(\min(a, b), c)$
- реализация стека с поддержкой хранения **промежуточных минимумов** для заданной входной последовательности целых чисел

# Стек с поддержкой **минимума**

- ассоциативная операция:  $\min(a, \min(b, c)) = \min(\min(a, b), c)$
- реализация стека с поддержкой хранения **промежуточных минимумов** для заданной входной последовательности целых чисел

top()	9	1
	6	1
	1	1

1	6	9	-8	3
---	---	---	----	---

```
stack.push(1, min(1, stack.top().second()))
```

```
stack.push(6, min(6, stack.top().second()))
```

```
stack.push(9, min(9, stack.top().second()))
```

# Стек с поддержкой **минимума**

- ассоциативная операция:  $\min(a, \min(b, c)) = \min(\min(a, b), c)$
- реализация стека с поддержкой хранения **промежуточных минимумов** для заданной входной последовательности целых чисел

top()	9	1
	6	1
	1	1

1	6	9	-8	3
---	---	---	----	---

```
stack.push(1, min(1, stack.top().second()))
```

```
stack.push(6, min(6, stack.top().second()))
```

```
stack.push(9, min(9, stack.top().second()))
```

на вершине стека хранится минимум **среди всех введенных на данный момент** значений исходной последовательности —  $\Theta(1)$

# Очередь с поддержкой минимума

- ассоциативная операция:  $\min(a, \min(b, c)) = \min(\min(a, b), c)$
- стандартная реализации очереди не обладает такими же быстрыми возможностями вычисления  $\min$ , как стек ПОЧЕМУ?

# Очередь с поддержкой **минимума**

- ассоциативная операция:  $\min(a, \min(b, c)) = \min(\min(a, b), c)$
- стандартная реализации очереди **не обладает** такими же быстрыми возможностями вычисления  $\min$ , как стек

back()	3	-2	
	-2	-2	
front()	1	1	
	pushStack	popStack	

1	-2	3	5	0	-4
---	----	---	---	---	----

push(1)

push(-2)

push(3)



# Очередь с поддержкой **минимума**

- ассоциативная операция:  $\min(a, \min(b, c)) = \min(\min(a, b), c)$
- стандартная реализации очереди **не обладает** такими же быстрыми возможностями вычисления  $\min$ , как стек

back()	3	-2	
	-2	-2	
front()	1	1	
	pushStack	popStack	

1	-2	3	5	0	-4
---	----	---	---	---	----

push(1)

push(-2)

push(3)

pop()

# Очередь с поддержкой **минимума**

- ассоциативная операция:  $\min(a, \min(b, c)) = \min(\min(a, b), c)$
- стандартная реализации очереди **не обладает** такими же быстрыми возможностями вычисления  $\min$ , как стек

3	-2
-2	-2
1	1

pushStack

1	-2
-2	-2
3	3

popStack

front()

back()

1	-2	3	5	0	-4
---	----	---	---	---	----

push(1)

push(-2)

push(3)

pop()

текущий min — **popStack.top().second()**

# Очередь с поддержкой **минимума**

- ассоциативная операция:  $\min(a, \min(b, c)) = \min(\min(a, b), c)$
- стандартная реализации очереди **не обладает** такими же быстрыми возможностями вычисления  $\min$ , как стек

back()	-4	-4	front()
	0	0	
	5	5	
pushStack		popStack	

1	-2	3	5	0	-4
---	----	---	---	---	----

push(5)

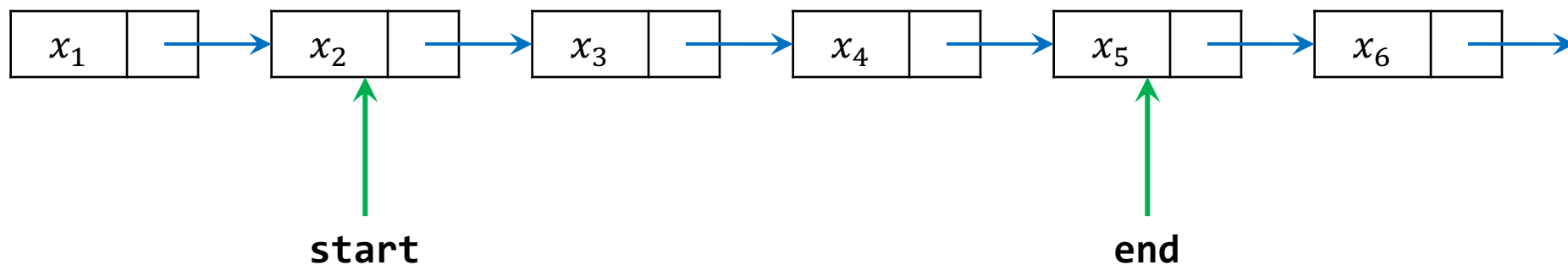
push(0)

push(-4)

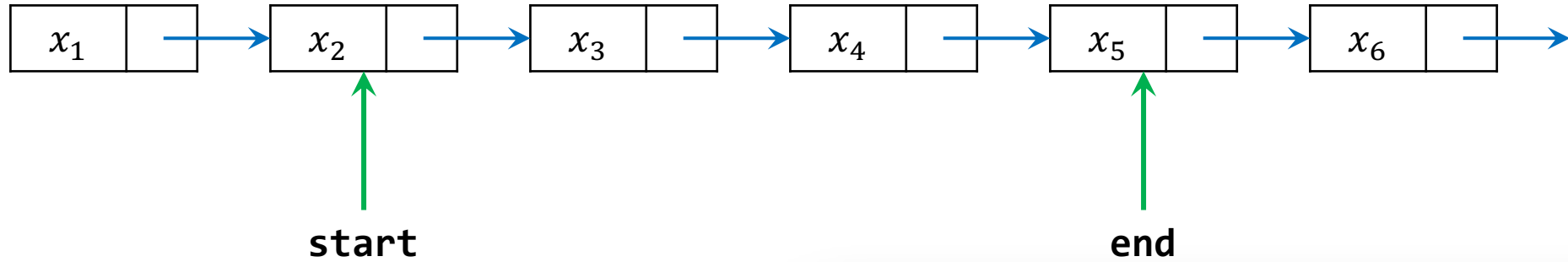
$\min = \min(\text{pushStack.top().second()},$   
 $\text{popStack.top().second()})$

# Связный список

# Сегмент списка



выделенный набор узлов списка, ограниченный указателем начала (**start**) и указателем конца (**end**)



```
isSegment_recursive.cpp

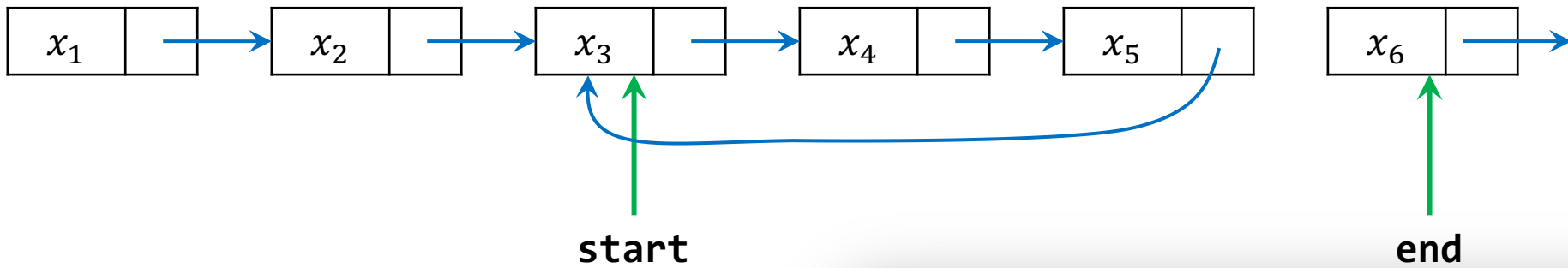
bool isSegment(Node* start, Node* end) {
    if (start == nullptr) return false;
    if (start == end) return true;

    return isSegment(start->next, end);
}
```

```
isSegment_loop.cpp

bool isSegment(Node* start, Node* end) {
    Node* p = start;
    while (p != nullptr) {
        if (p == end) return true;
        p = p->next;
    }

    return false;
}
```



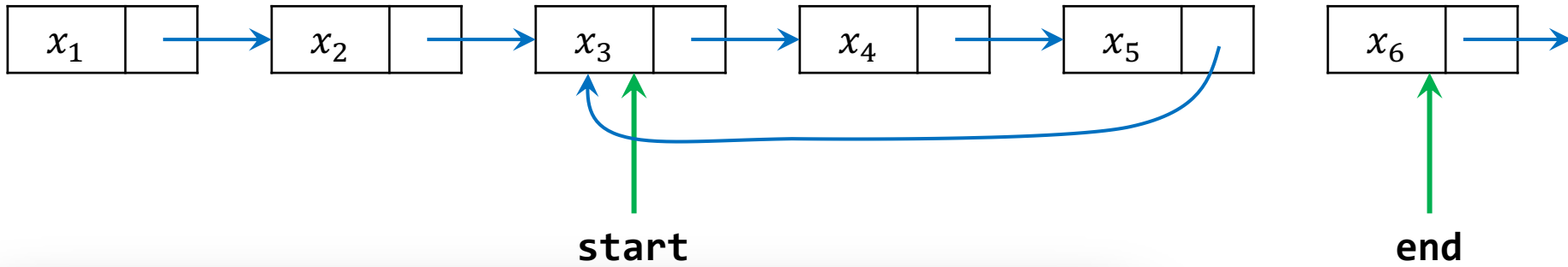
isSegment\_recursive.cpp

```
bool isSegment(Node* start, Node* end) {  
    if (start == end) return true;  
    if (start == null) return false;  
    return isSegment(start->next, end);  
}
```

для связанного списка с циклической структурой, предложенный алгоритм проверки может не завершиться... ПОЧЕМУ?

isSegment\_loop.cpp

```
bool isSegment(Node* start, Node* end) {  
    Node* p = start;  
    while (p != null) {  
        if (p == end) return true;  
        p = p->next;  
    }  
    return false;  
}
```

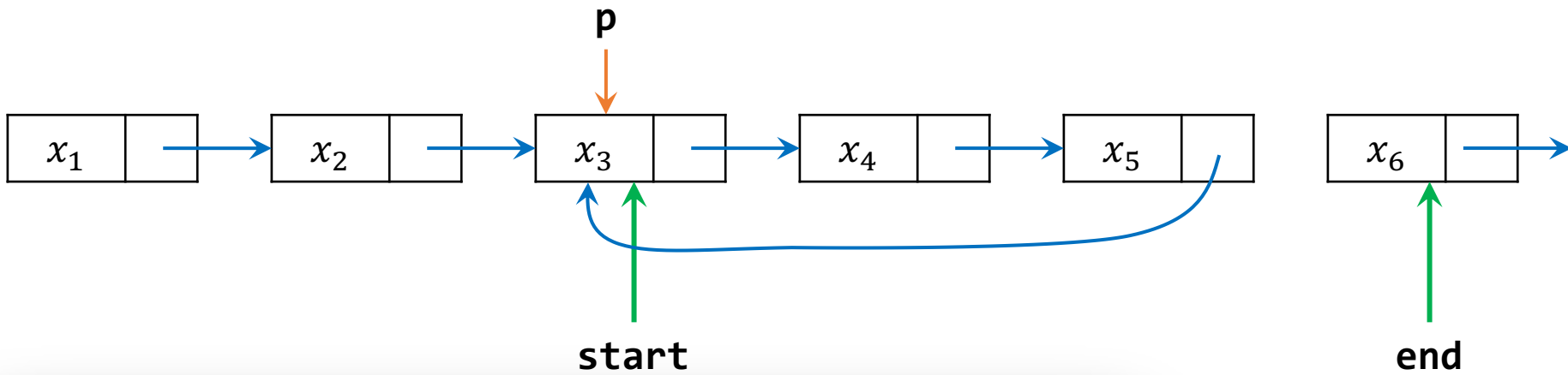


```
isAcyclic_quadratic.cpp

bool isAcyclic(Node* start, Node* end) {
    for (Node* p = start; p != end; p = p->next) {
        if (p == nullptr) return true;
        for (Node* q = start; q != p; q = q->next) {
            if (q == p->next) return false;
        }
    }

    return true;
}
```

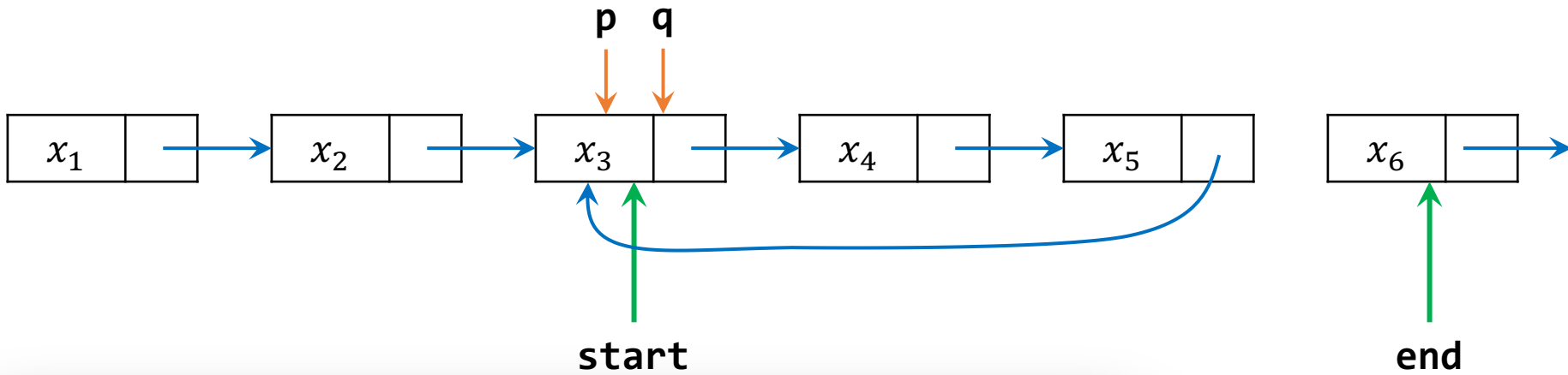




```
isAcyclic_quadratic.cpp

bool isAcyclic(Node* start, Node* end) {
    → for (Node* p = start; p != end; p = p->next) {
        if (p == nullptr) return true;
        for (Node* q = start; q != p; q = q->next) {
            if (q == p->next) return false;
        }
    }

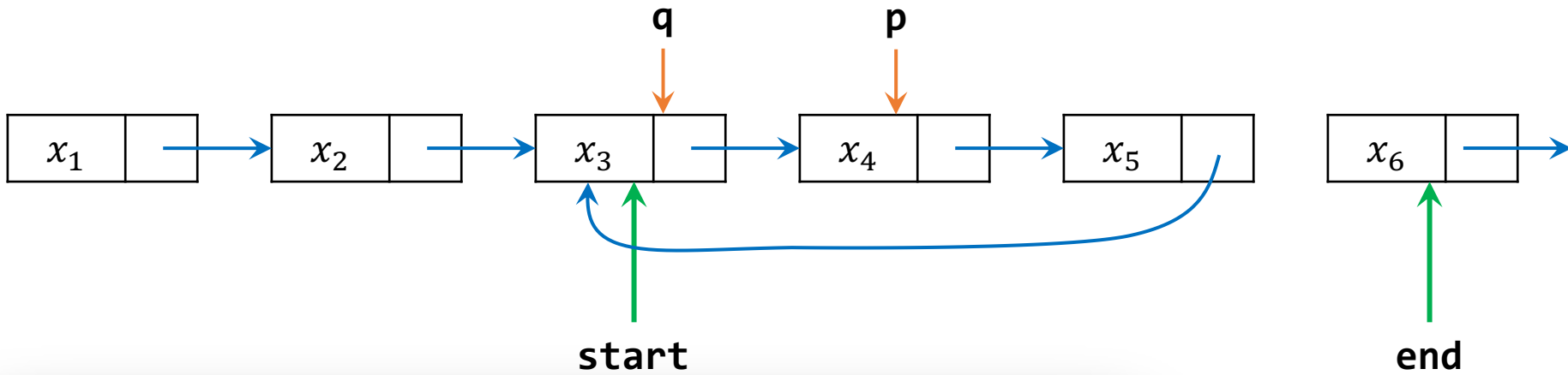
    return true;
}
```



```
isAcyclic_quadratic.cpp

bool isAcyclic(Node* start, Node* end) {
    for (Node* p = start; p != end; p = p->next) {
        if (p == nullptr) return true;
        ➔ for (Node* q = start; q != p; q = q->next) {
            if (q == p->next) return false;
        }
    }

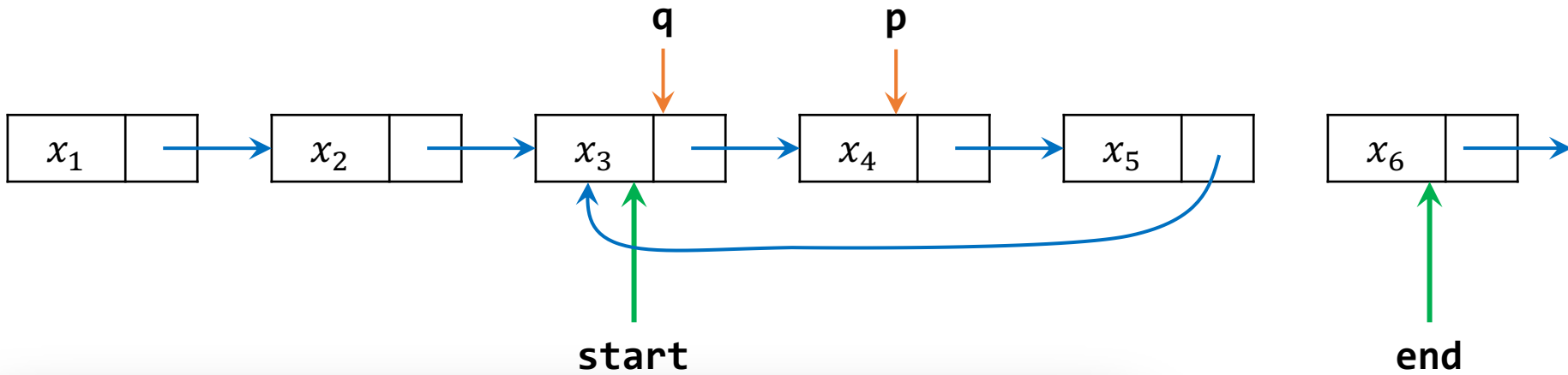
    return true;
}
```



```
isAcyclic_quadratic.cpp

bool isAcyclic(Node* start, Node* end) {
    ➔ for (Node* p = start; p != end; p = p->next) {
        if (p == nullptr) return true;
        for (Node* q = start; q != p; q = q->next) {
            if (q == p->next) return false;
        }
    }

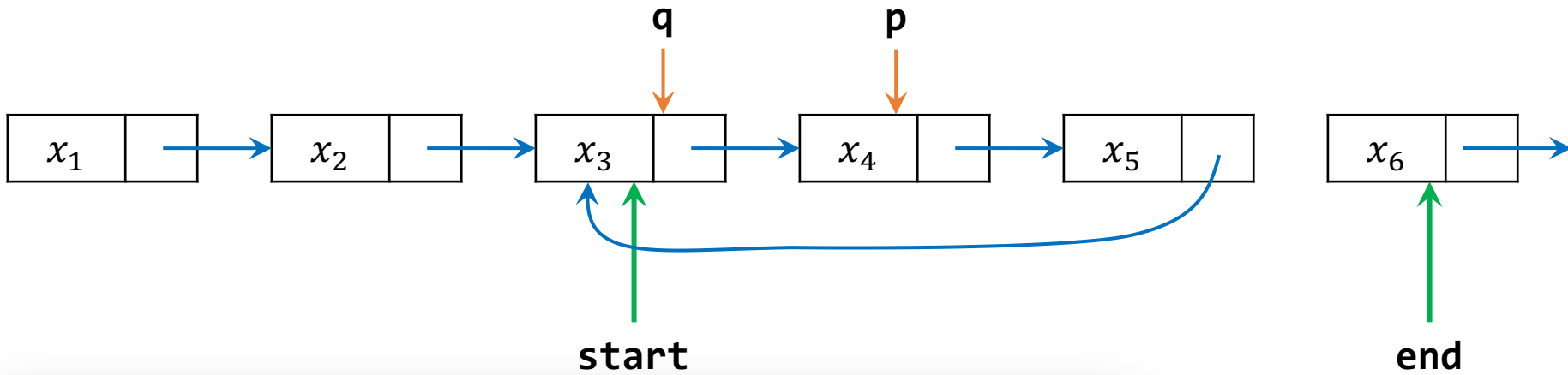
    return true;
}
```



```
isAcyclic_quadratic.cpp

bool isAcyclic(Node* start, Node* end) {
    for (Node* p = start; p != end; p = p->next) {
        if (p == nullptr) return true;
        ➔ for (Node* q = start; q != p; q = q->next) {
            if (q == p->next) return false;
        }
    }

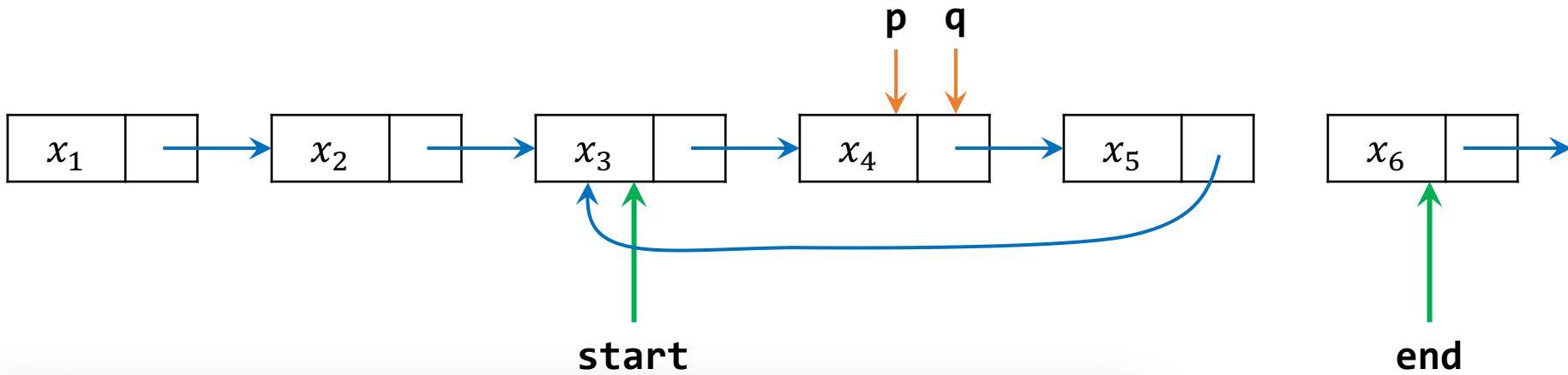
    return true;
}
```



```
isAcyclic_quadratic.cpp

bool isAcyclic(Node* start, Node* end) {
    for (Node* p = start; p != end; p = p->next) {
        if (p == nullptr) return true;
        for (Node* q = start; q != p; q = q->next) {
            ➔ if (q == p->next) return false;
        }
    }

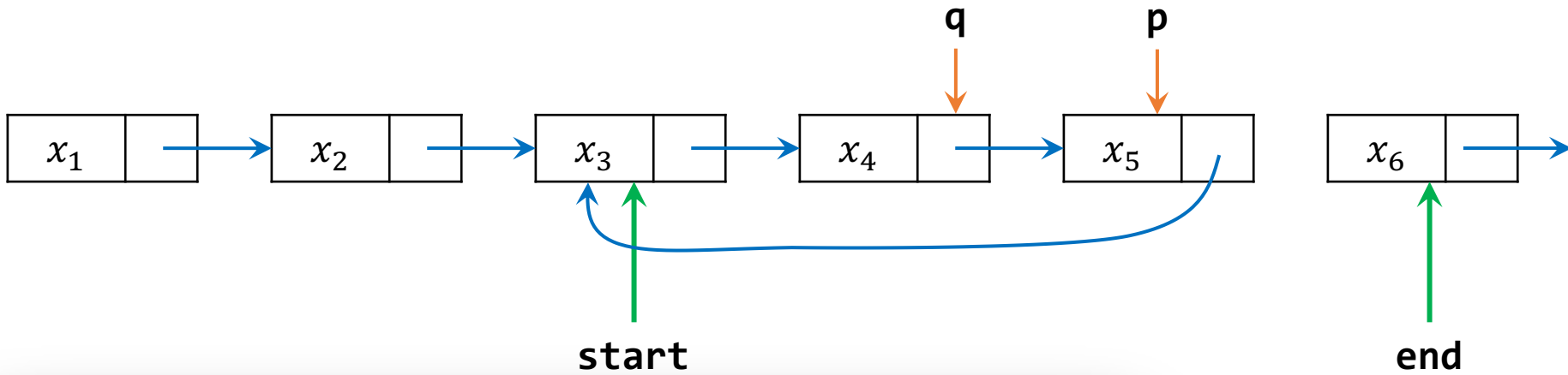
    return true;
}
```



```
isAcyclic_quadratic.cpp

bool isAcyclic(Node* start, Node* end) {
    for (Node* p = start; p != end; p = p->next) {
        if (p == nullptr) return true;
        ➔ for (Node* q = start; q != p; q = q->next) {
            if (q == p->next) return false;
        }
    }

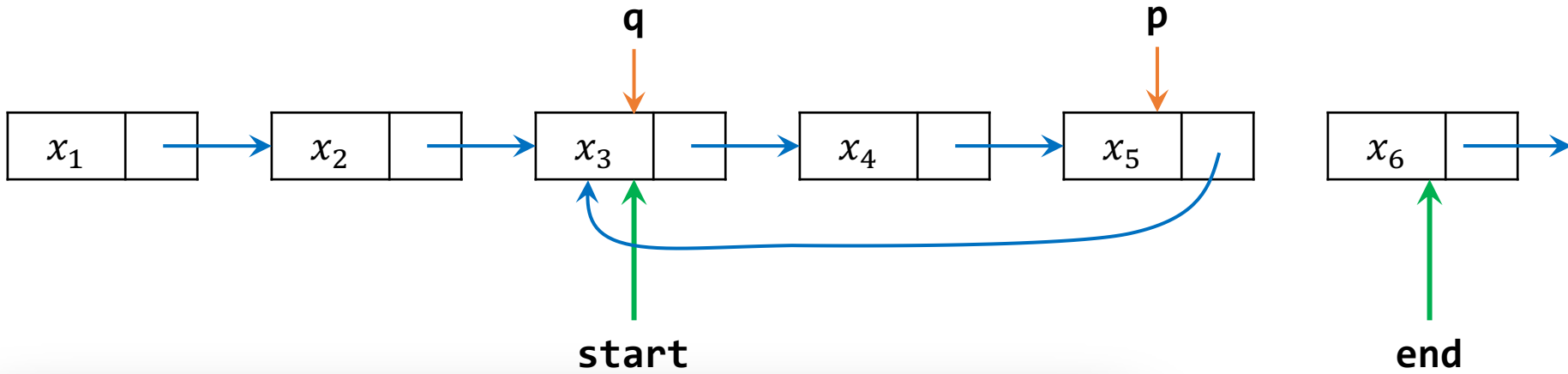
    return true;
}
```



```
isAcyclic_quadratic.cpp

bool isAcyclic(Node* start, Node* end) {
    ➔ for (Node* p = start; p != end; p = p->next) {
        if (p == nullptr) return true;
        for (Node* q = start; q != p; q = q->next) {
            if (q == p->next) return false;
        }
    }

    return true;
}
```

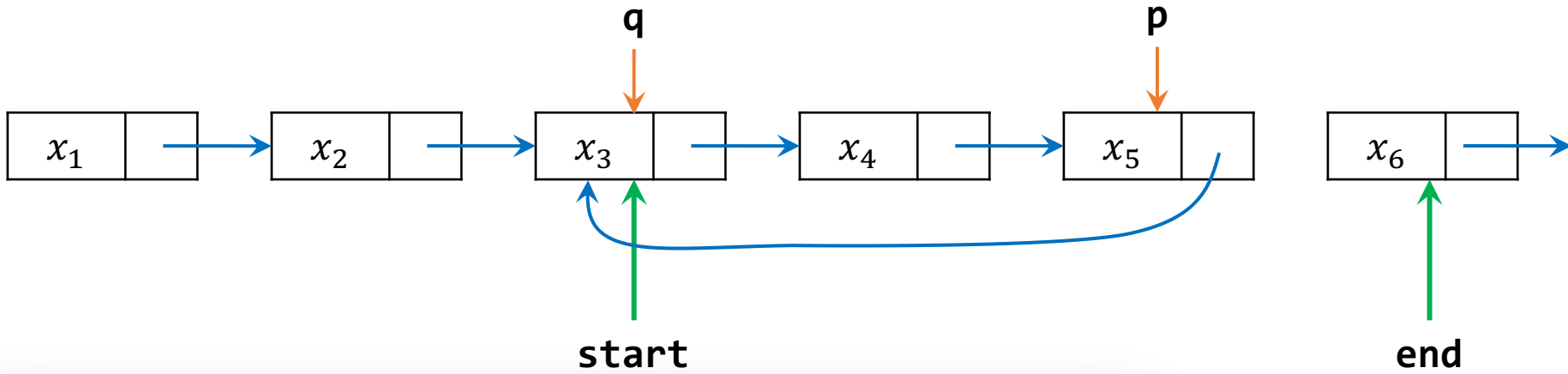


```
isAcyclic_quadratic.cpp

bool isAcyclic(Node* start, Node* end) {
    for (Node* p = start; p != end; p = p->next) {
        if (p == nullptr) return true;
        → for (Node* q = start; q != p; q = q->next) {
            if (q == p->next) return false;
        }
    }

    return true;
}
```

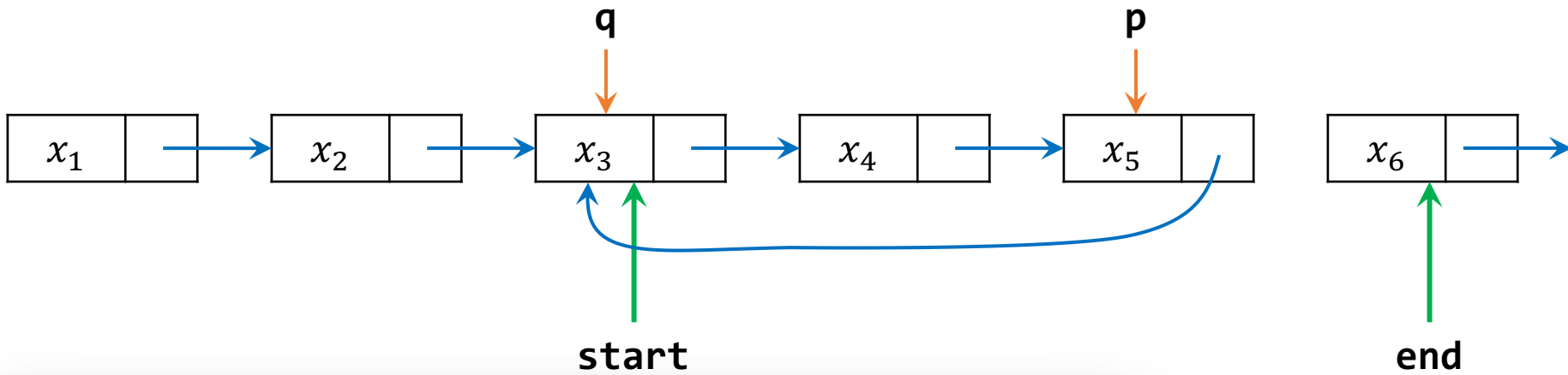




```
isAcyclic_quadratic.cpp

bool isAcyclic(Node* start, Node* end) {
    for (Node* p = start; p != end; p = p->next) {
        if (p == nullptr) return true;
        for (Node* q = start; q != p; q = q->next) {
            ➔ if (q == p->next) return false;
        }
    }

    return true;
}
```



```
isAcyclic_quadratic.cpp

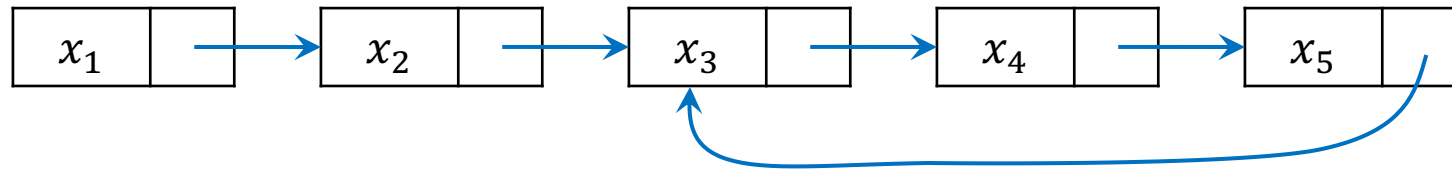
bool isAcyclic(Node* start, Node* end) {
    for (Node* p = start; p != end; p = p->next) {
        if (p == nullptr) return true;
        for (Node* q = start; q != p; q = q->next) {
            if (q == p->next) return false;
        }
    }

    return true;
} →
```

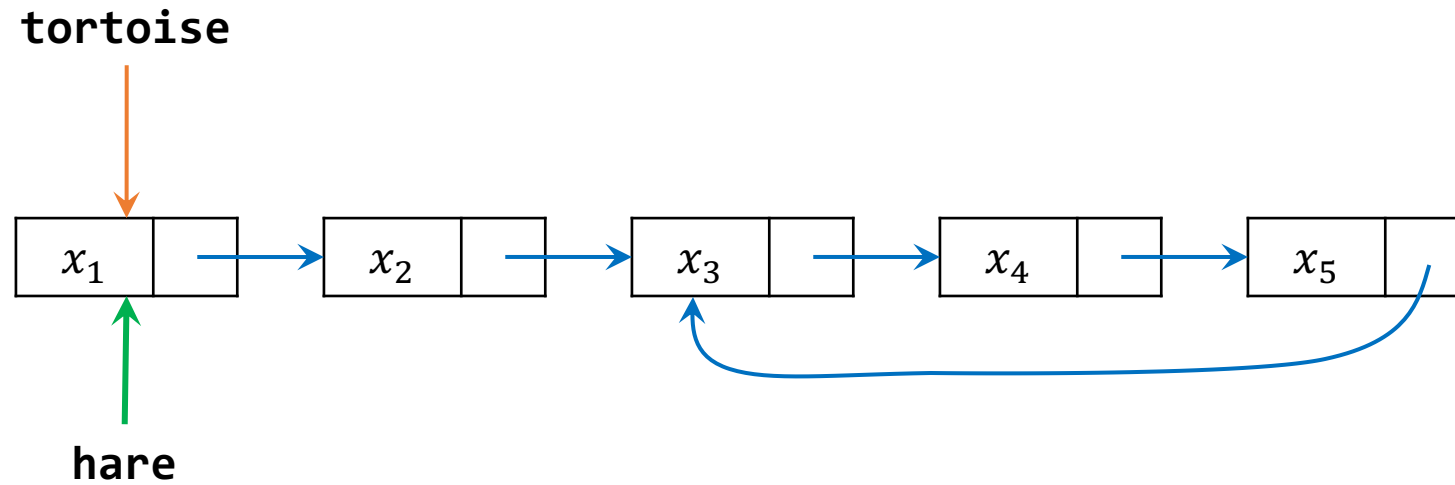
временная  
сложность —  $\Theta(n^2)$ ,  
где  $n$  — размер  
списка

ПОЧЕМУ?

# Проверка на цикл за $\Theta(n)$ - а. Флойда

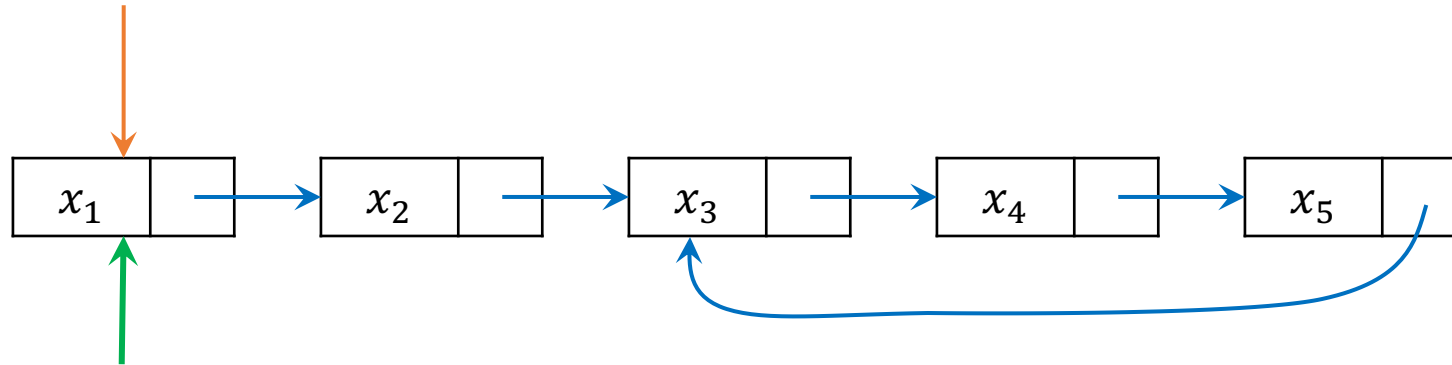


# Проверка на цикл за $\Theta(n)$ - а. Флойда



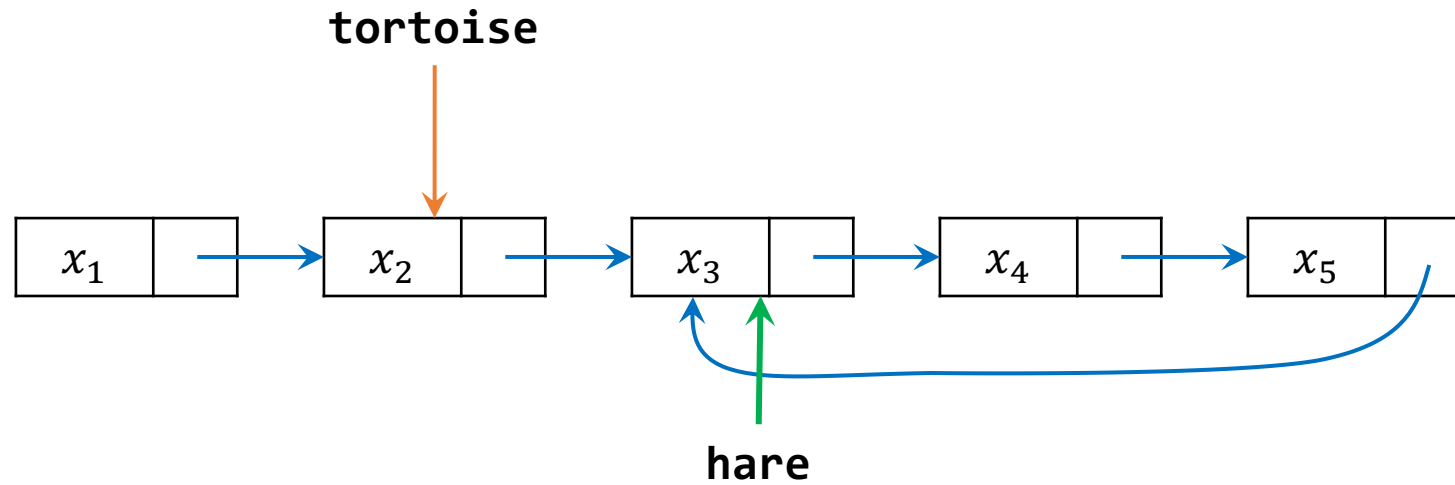
# Проверка на цикл за $\Theta(n)$ - а. Флойда

**tortoise** продвигается вперед **на один** указатель

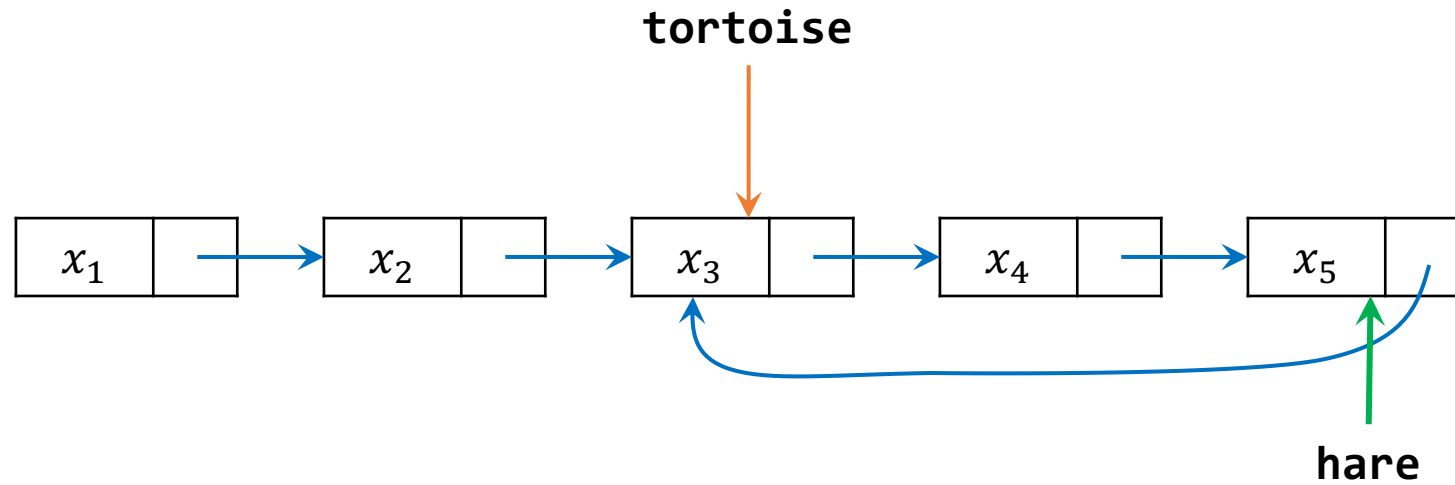


**hare** продвигается вперед **на два** указателя

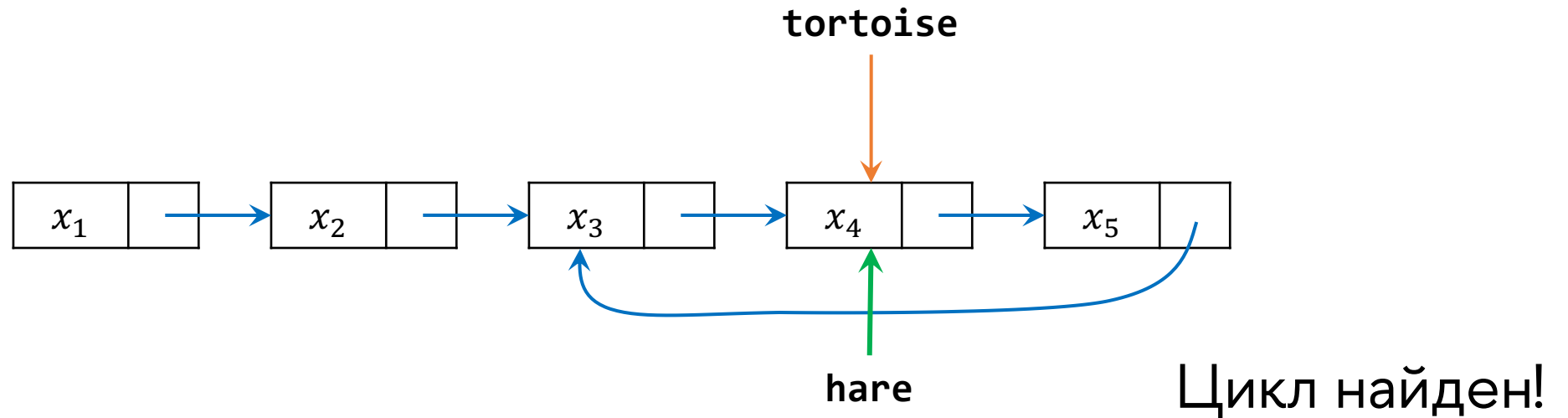
# Проверка на цикл за $\Theta(n)$ - а. Флойда



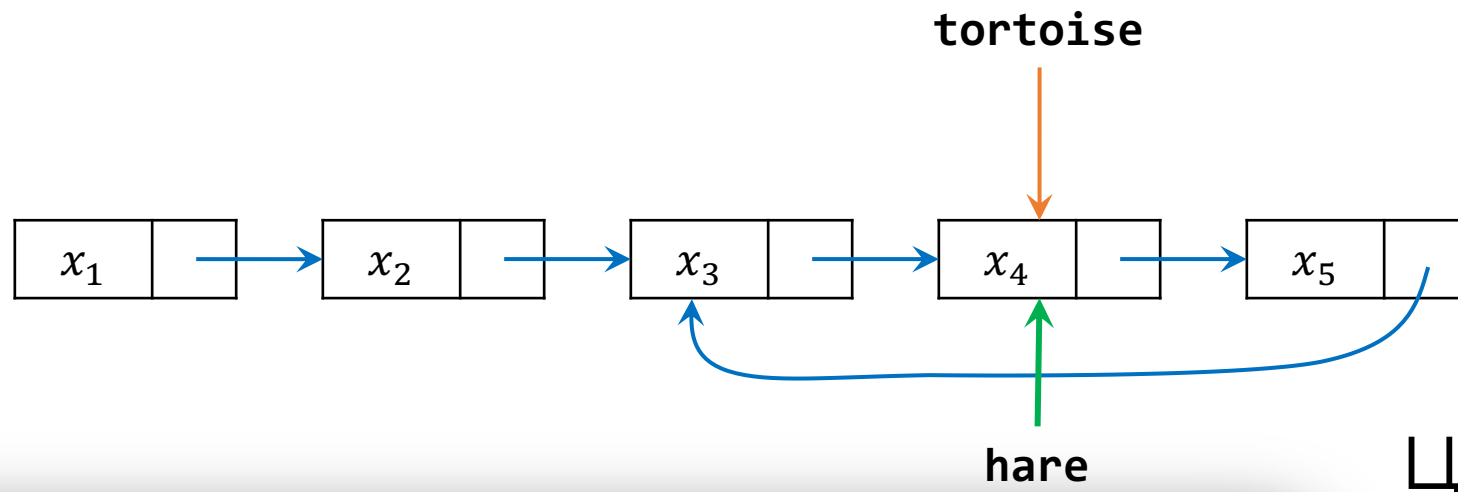
# Проверка на цикл за $\Theta(n)$ - а. Флойда



# Проверка на цикл за $\Theta(n)$ - а. Флойда





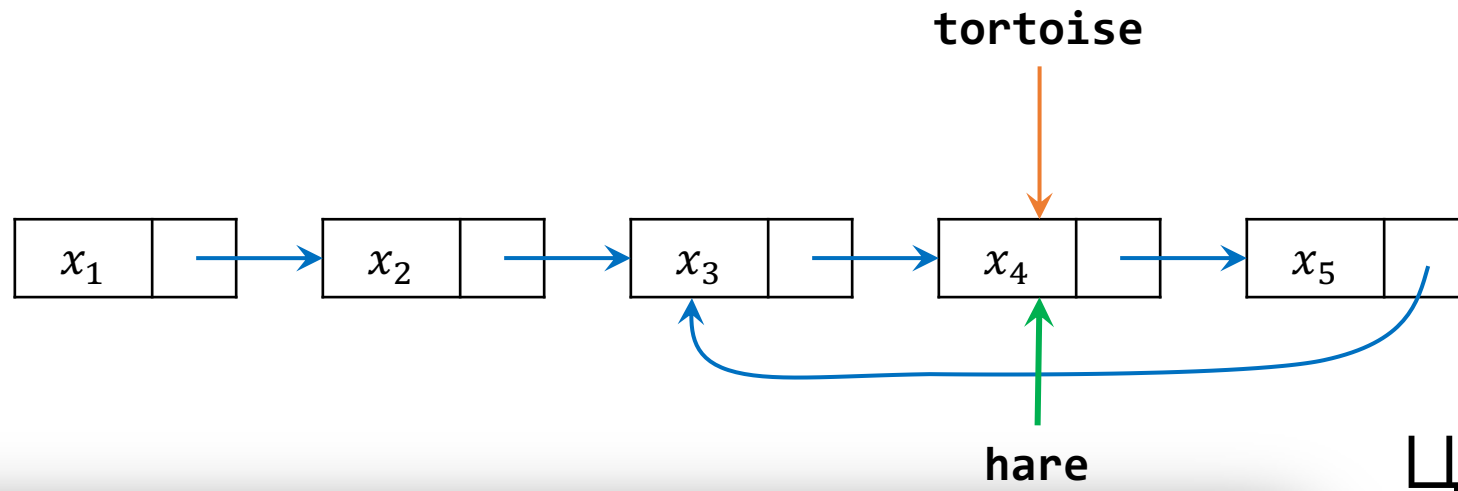


Цикл найден!

isAcyclic\_linear.cpp

```
bool isAcyclic(Node* start) {  
    Node* hare = start;  
    Node* tortoise = start;  
    while (hare != tortoise) {  
        if (hare == nullptr || hare->next == nullptr) return true;  
  
        hare = hare->next->next;  
        tortoise = tortoise->next;  
    }  
  
    return false;  
}
```

нет ли в приведенной реализации ошибок?



isAcyclic\_linear.cpp

```
bool isAcyclic(Node* start) {  
    Node* hare = start;  
    Node* tortoise = start;  
    while (hare != tortoise) {  
        if (hare == nullptr || hare->next == nullptr) return true;  
  
        hare = hare->next->next;  
        tortoise = tortoise->next;  
    }  
  
    return false;  
}
```

нет ли в приведенной  
реализации ошибок?

временная  
сложность —  $\Theta(n)$

ПОЧЕМУ?

# РЕЗЮМЕ

Различные подходы к реализации линейных контейнеров на примере стека и очереди

Амортизационный анализ сложности: много быстрых операций и мало долгих

Связный список: поиск циклического сегмента

# ДОМАШНЕЕ ЗАДАНИЕ

реализовать на C++  
ADT Очередь на двух стеках с  
поддержкой быстрого вычисления  
(промежуточного) минимума среди  
хранящихся целых чисел

```
ADT_Queue.cpp

class Queue {
public:
    void push(int elem);
    int pop();
    int& front();
    int& back();
    int currentMin();
private:
    Stack<int> pushStack_;
    Stack<int> popStack_;
}
```