STACKS AND QUEUES

Problem Solving with Computers-II

Stacks – container class available in the C++ STL

- Container class that uses the Last In First Out (LIFO) principle
- Methods

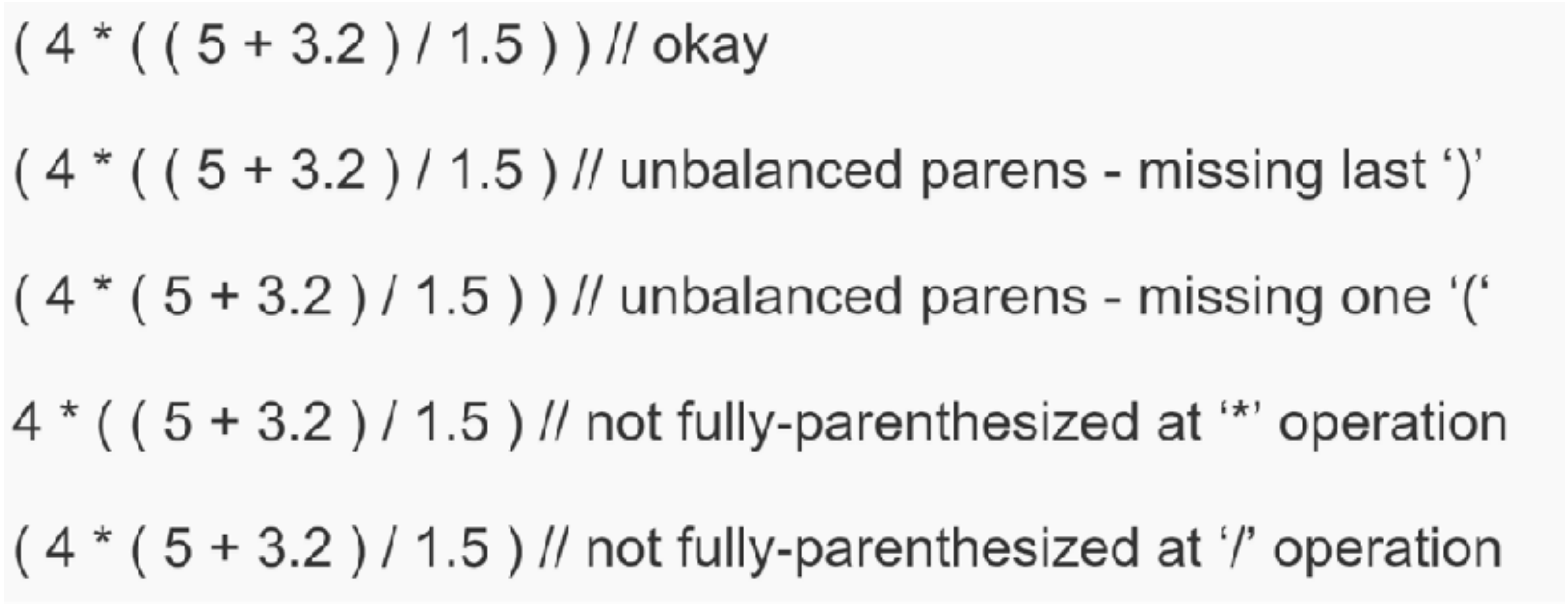
i. push()
ii. pop()
iii. top()
iv. empty()

Demo reversing a string

Notations for evaluating expression

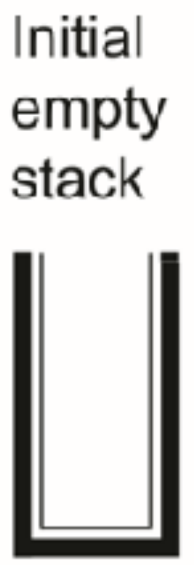
- Infix number operator number (7 + (3 * 5)) – (4 / 2)
- Prefix operators precede the operands
- Postfix operators come after the operands

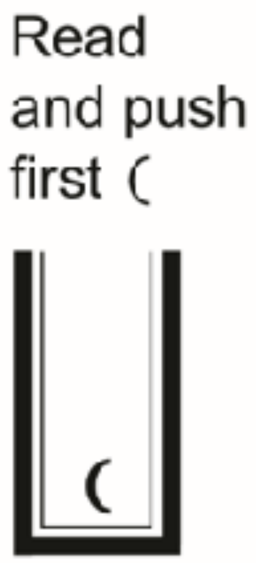
Lab05 – part 1: Evaluate a fully parenthesized infix expression

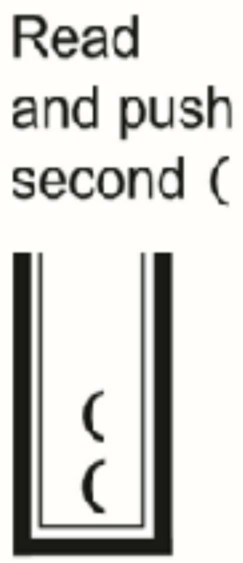


```
( 4 * ( ( 5 + 3.2 ) / 1.5 ) ) // okay
( 4 * ( ( 5 + 3.2 ) / 1.5 ) // unbalanced parens - missing last ‘)’
( 4 * ( 5 + 3.2 ) / 1.5 ) ) // unbalanced parens - missing one ‘(‘
4 * ( ( 5 + 3.2 ) / 1.5 ) // not fully-parenthesized at ‘*’ operation
( 4 * ( 5 + 3.2 ) / 1.5 ) // not fully-parenthesized at ‘/’ operation
```

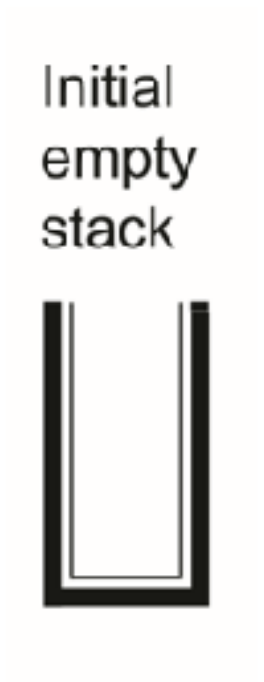
((2 * 2) + (8 + 4))

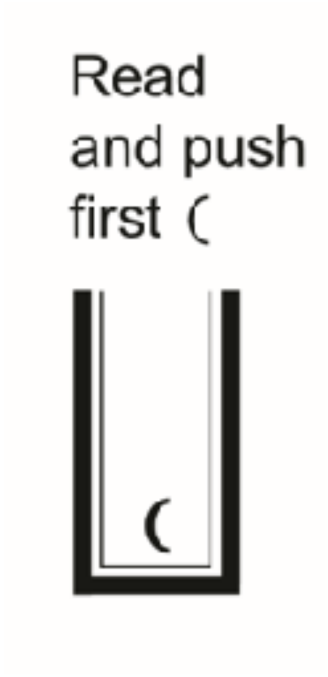

Initial
empty
stack

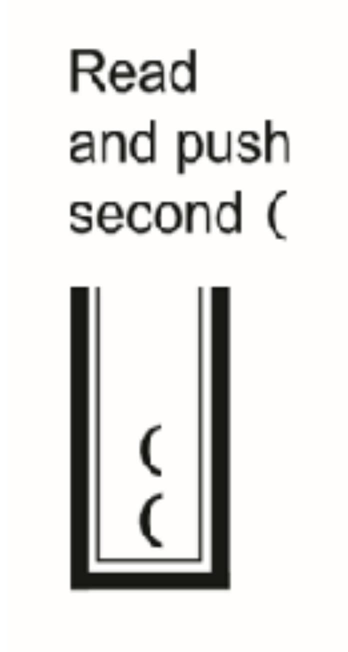

Read
and push
first (
(


Read
and push
second (
(
(

((2 * 2) + (8 + 4))



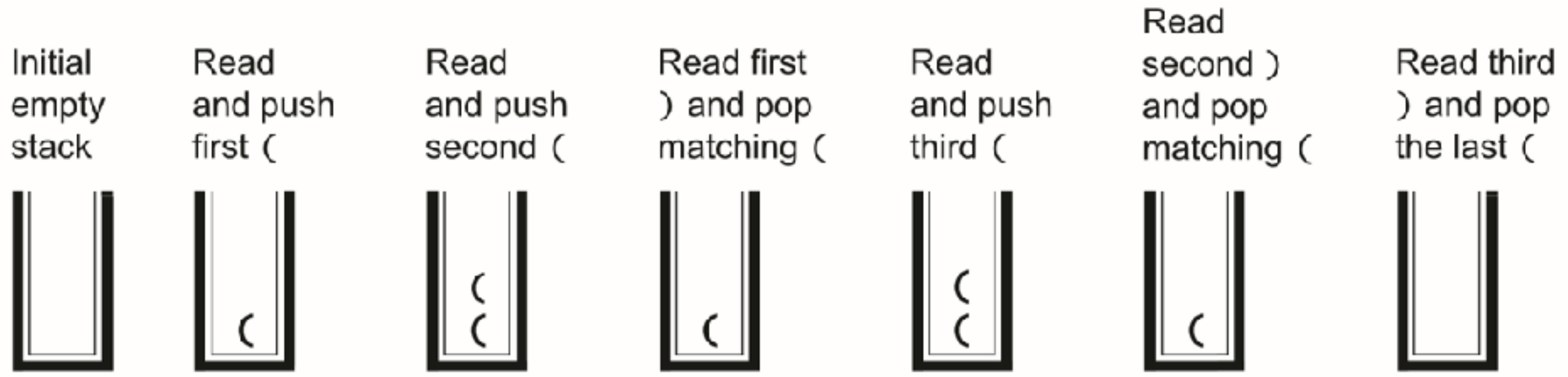




What should **be the next step after the first right parenthesis is encountered**?

A. Push the right parenthesis onto the stack
B. If the stack is not empty pop the next item on the top of the stack
C. Ignore the right parenthesis and continue checking the next character
D. None of the above

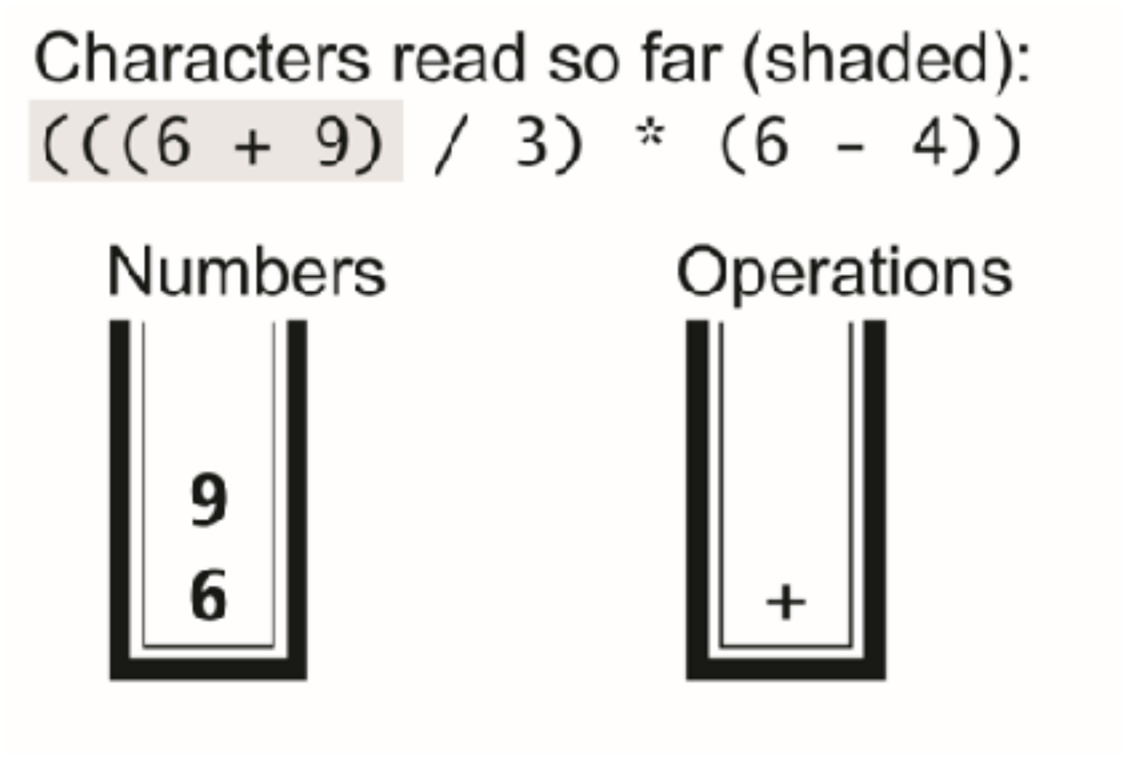
((2 * 2) + (8 + 4))


Initial empty stack
Read and push first (
Read and push second (
Read first) and pop matching (
Read and push third (
Read second) and pop matching (
Read third) and pop the last (
(
(
(
(
(
(
(

Evaluating a fully parenthesized infix expression

```
(((6 + 9)/3)*(6 - 4))
```

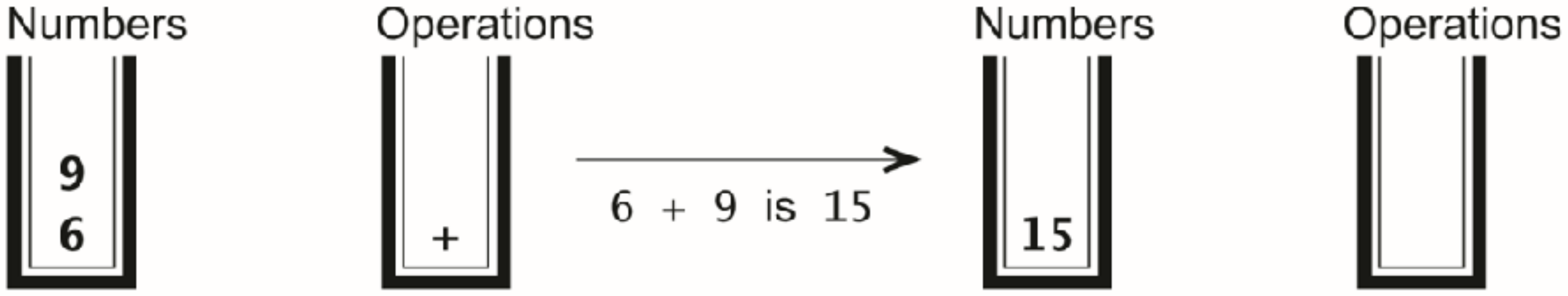
Evaluating a fully parenthesized infix expression



Evaluating a fully parenthesized infix expression

Characters read so far (shaded):

(((6 + 9) / 3) * (6 - 4))



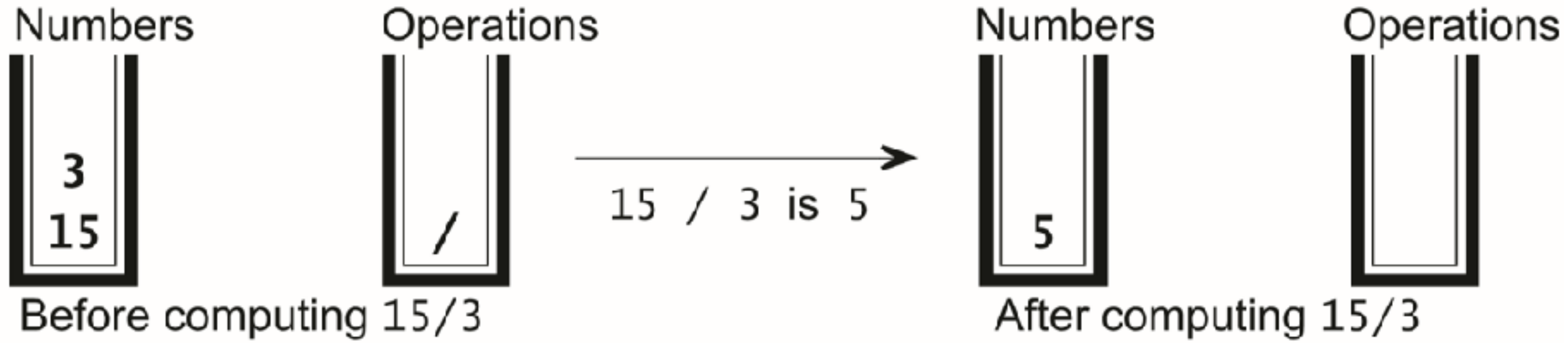
Before computing 6 + 9

After computing 6 + 9

Evaluating a fully parenthesized infix expression

Characters read so far (shaded):

```
(((6 + 9) / 3) * (6 - 4))
```



Before computing 15/3

After computing 15/3

Lab 05, part2 :
Evaluating post fix expressions using a single stack

Postfix: 7 3 5 * + 4 2 / -

Infix: (7 + (3 * 5)) – (4 / 2)

Small group exercise

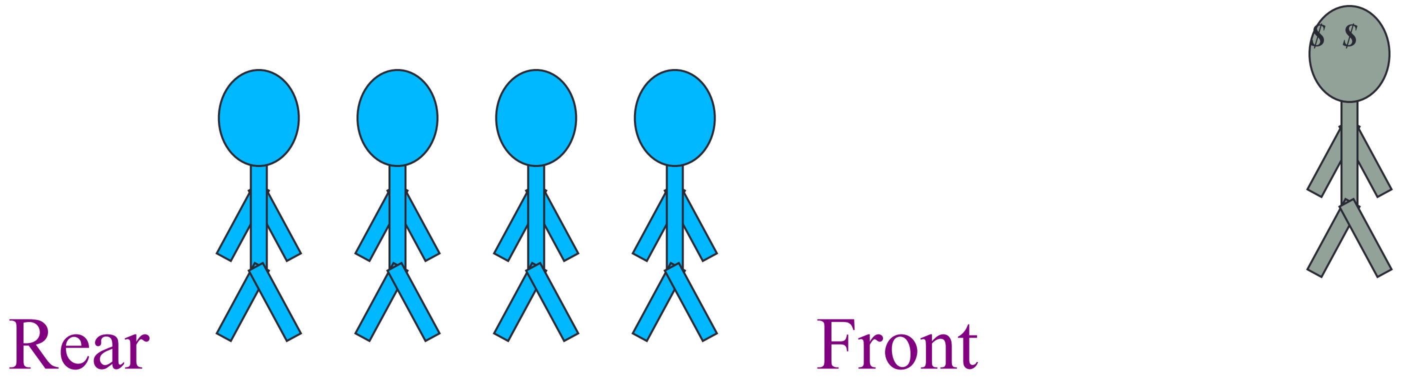
Write a ADT called in minStack that provides the following methods

- push() // inserts an element to the “top” of the minStack
- pop() // removes the last element that was pushed on the stack
- top () // returns the last element that was pushed on the stack
- min() // returns the minimum value of the elements stored so far

CRACKING the CODING INTERVIEW
189 PROGRAMMING QUESTIONS & SOLUTIONS
GAYLE LAAKMANN MCDOWELL
6th EDITION

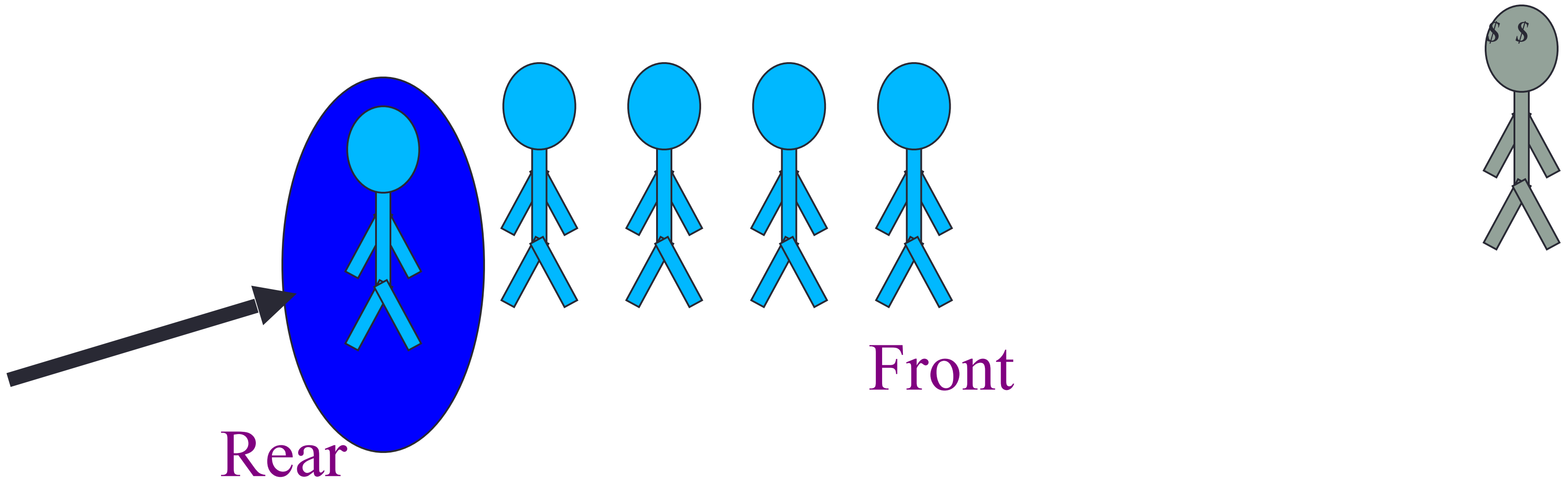
The Queue Operations

- A queue is like a line of people waiting for a bank teller. The queue has a **front** and a **rear**.



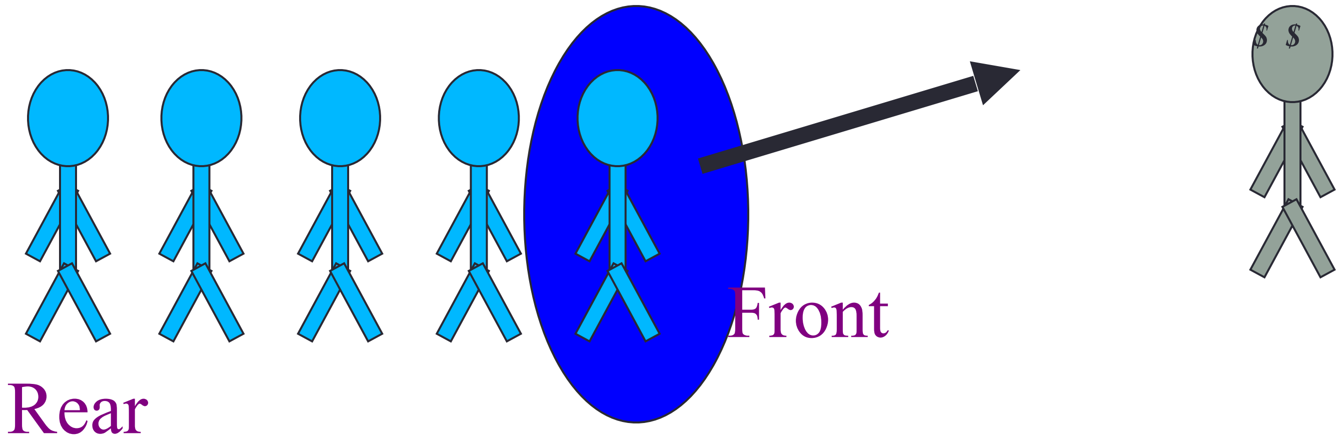
The Queue Operations

- New people must enter the queue at the rear. The C++ queue class calls this a **push**, although it is usually called an **enqueue** operation.



The Queue Operations

- When an item is taken from the queue, it always comes from the front. The C++ queue calls this a **pop**, although it is usually called a **dequeue** operation.



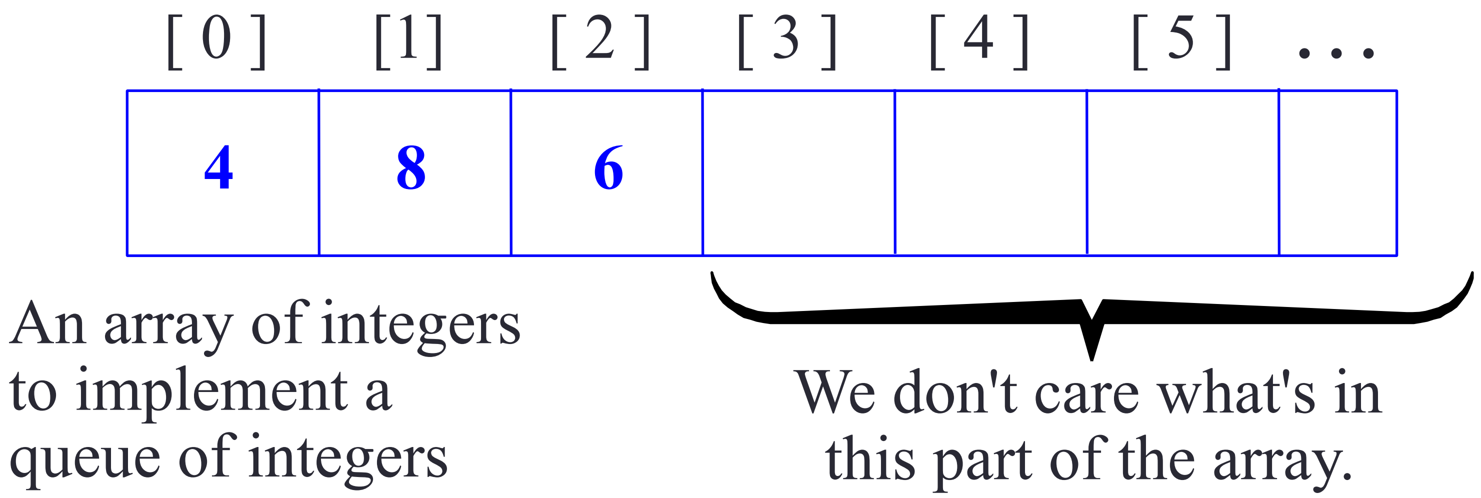
The Queue Class

- The C++ standard template library has a queue template class.
- The template parameter is the type of the items that can be put in the queue.

```
template <class Item>
class queue<Item>
{
public:
        queue( );
        void push(const Item& entry);
        void pop( );
        bool empty( ) const;
        Item front( ) const;
        …
```

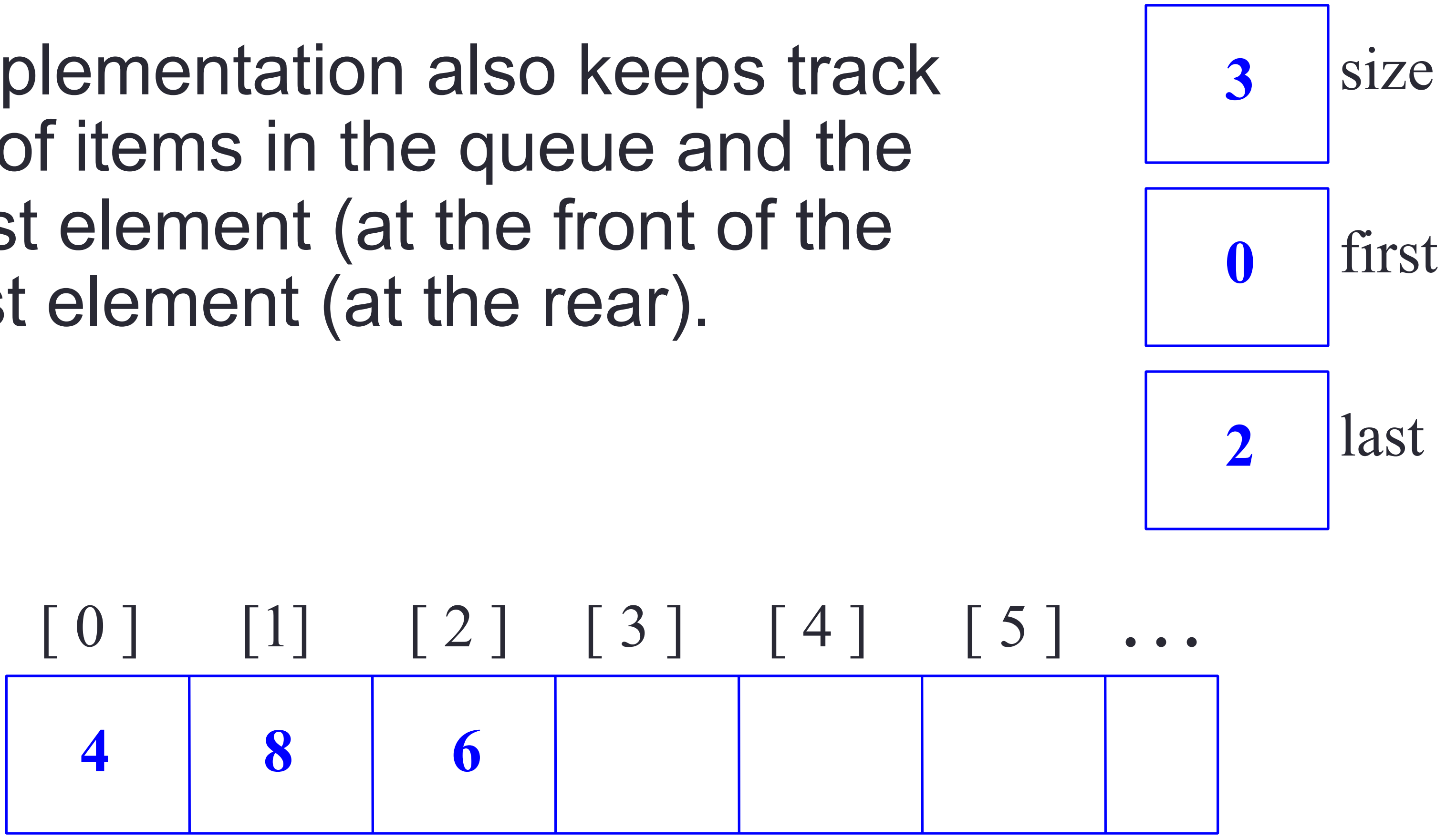
Array Implementation

- A queue can be implemented with an array, as shown here. For example, this queue contains the integers 4 (at the front), 8 and 6 (at the rear).


[0]
[1]
[2]
[3]
[4]
[5]
. . .
4
8
6
An array of integers to implement a queue of integers
We don't care what's in this part of the array.

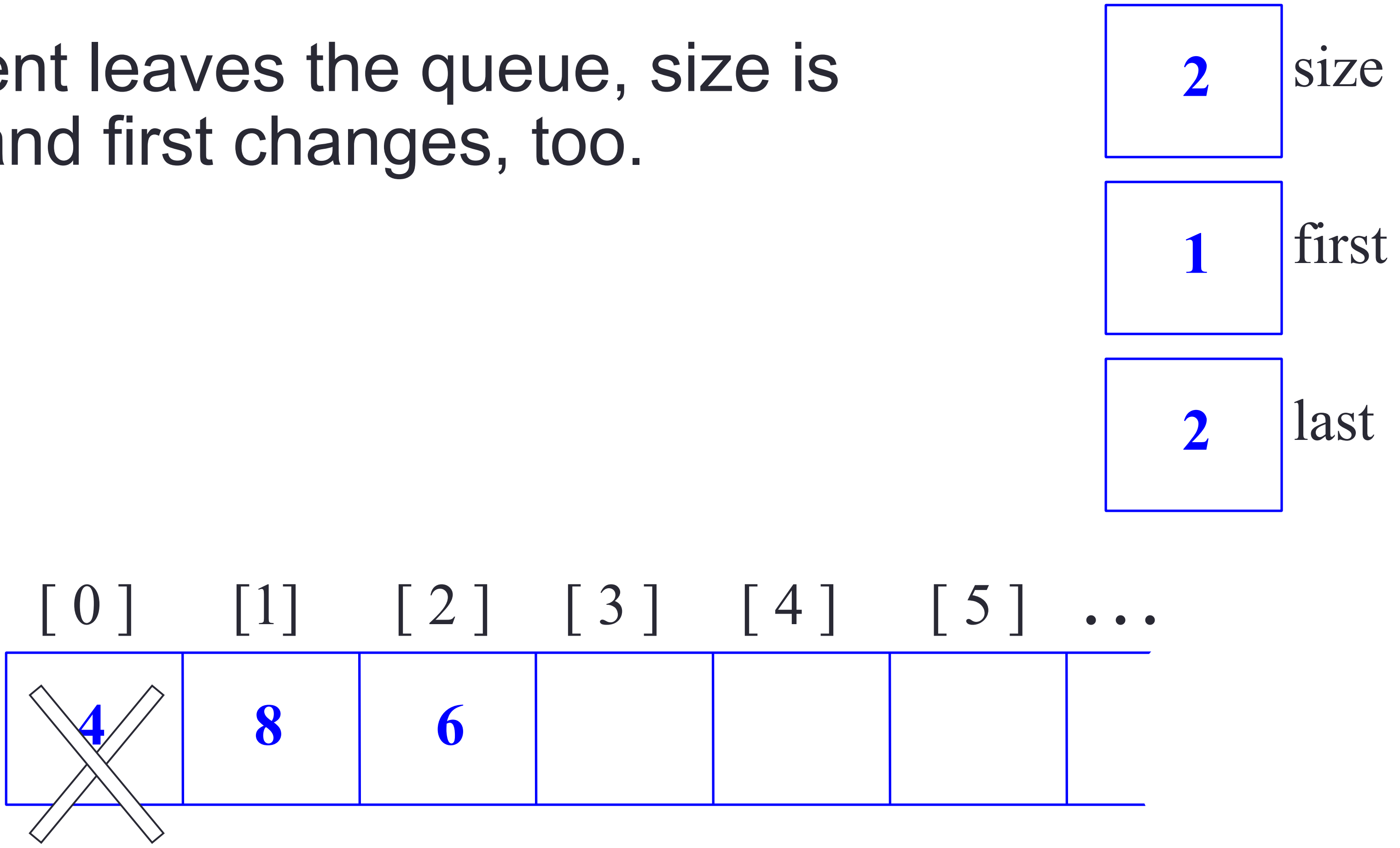
Array Implementation

- The easiest implementation also keeps track of the number of items in the queue and the index of the first element (at the front of the queue), the last element (at the rear).



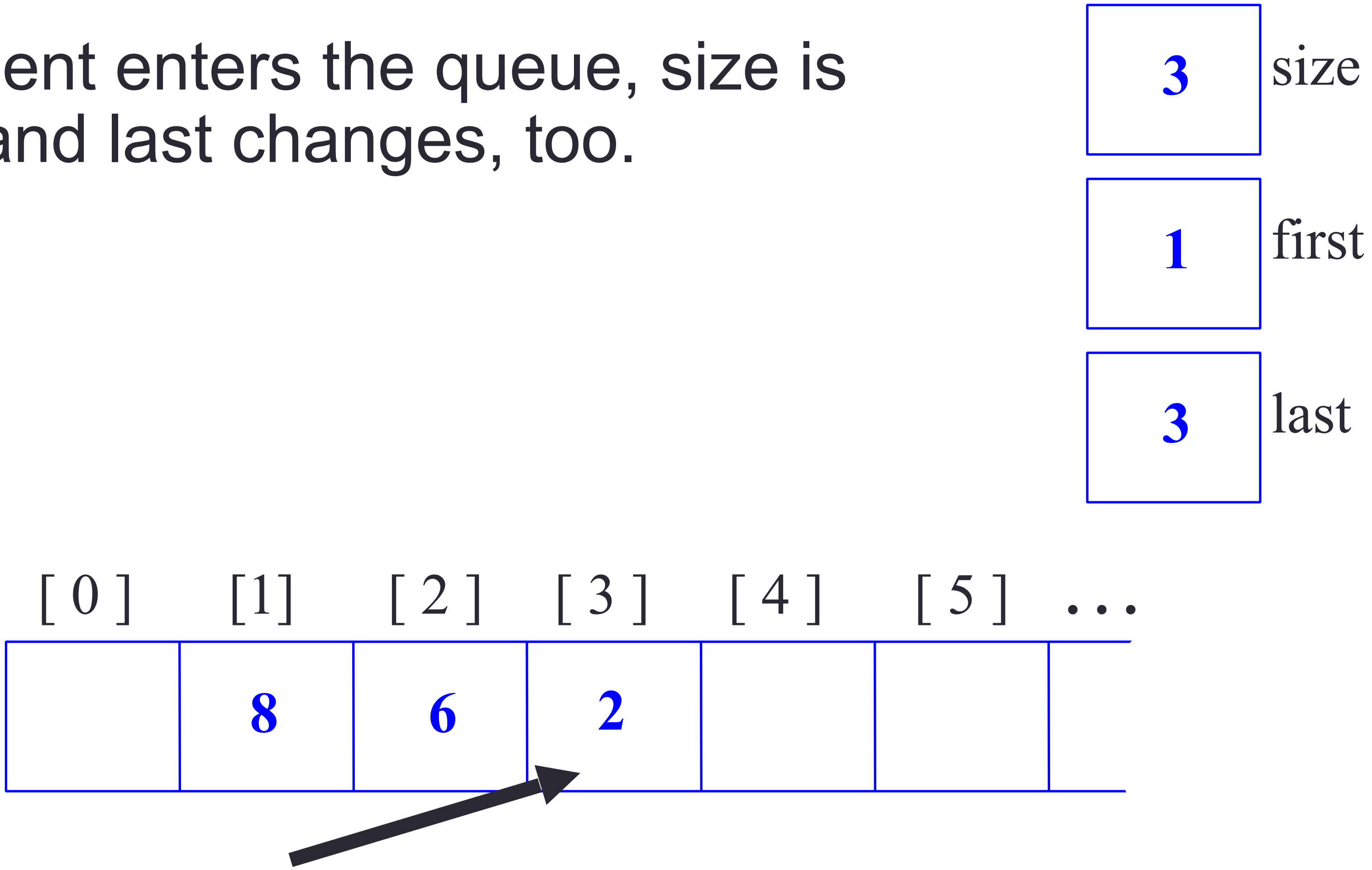
A Dequeue Operation

- When an element leaves the queue, size is decremented, and first changes, too.



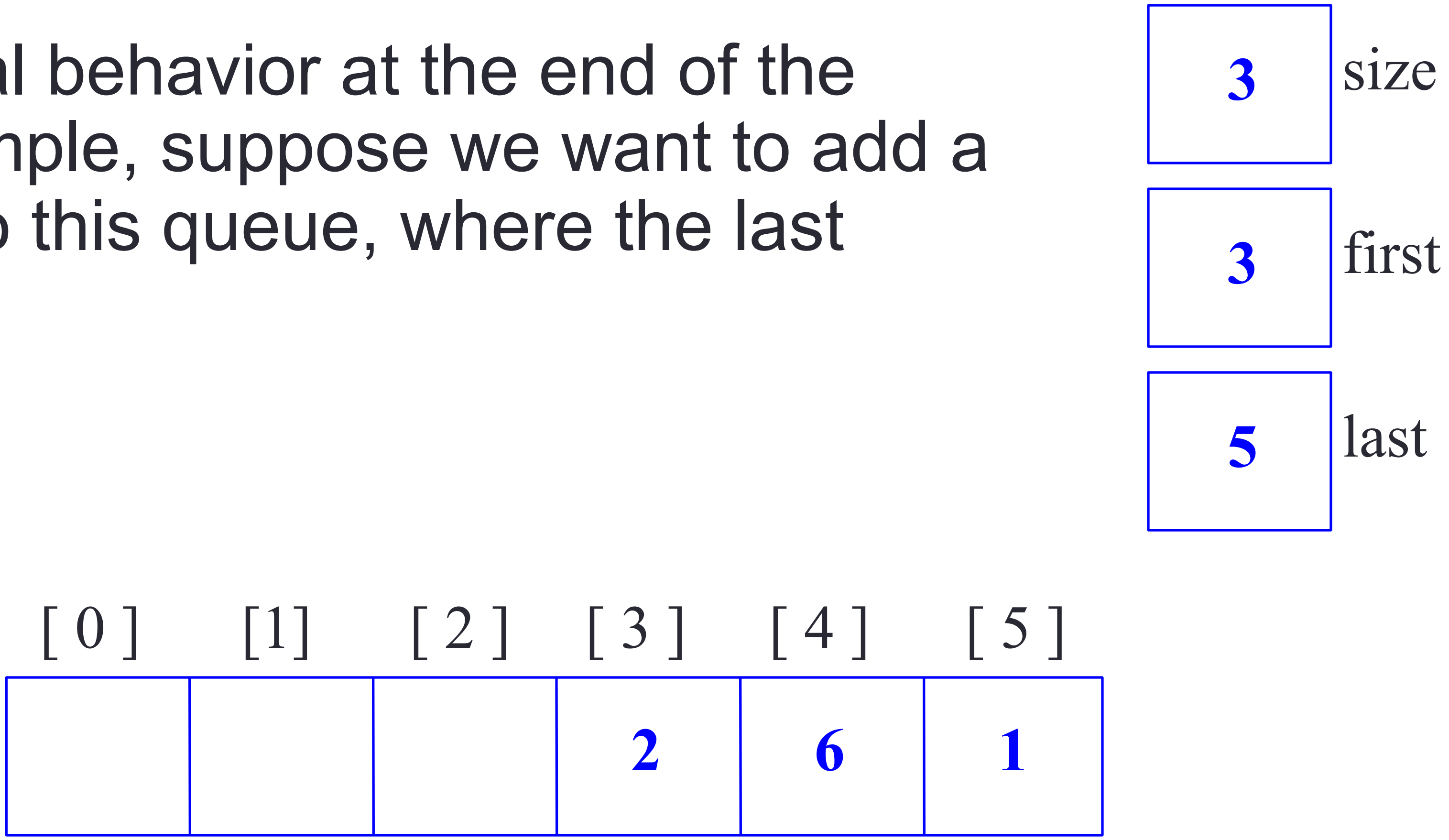
An Enqueue Operation

- When an element enters the queue, size is incremented, and last changes, too.



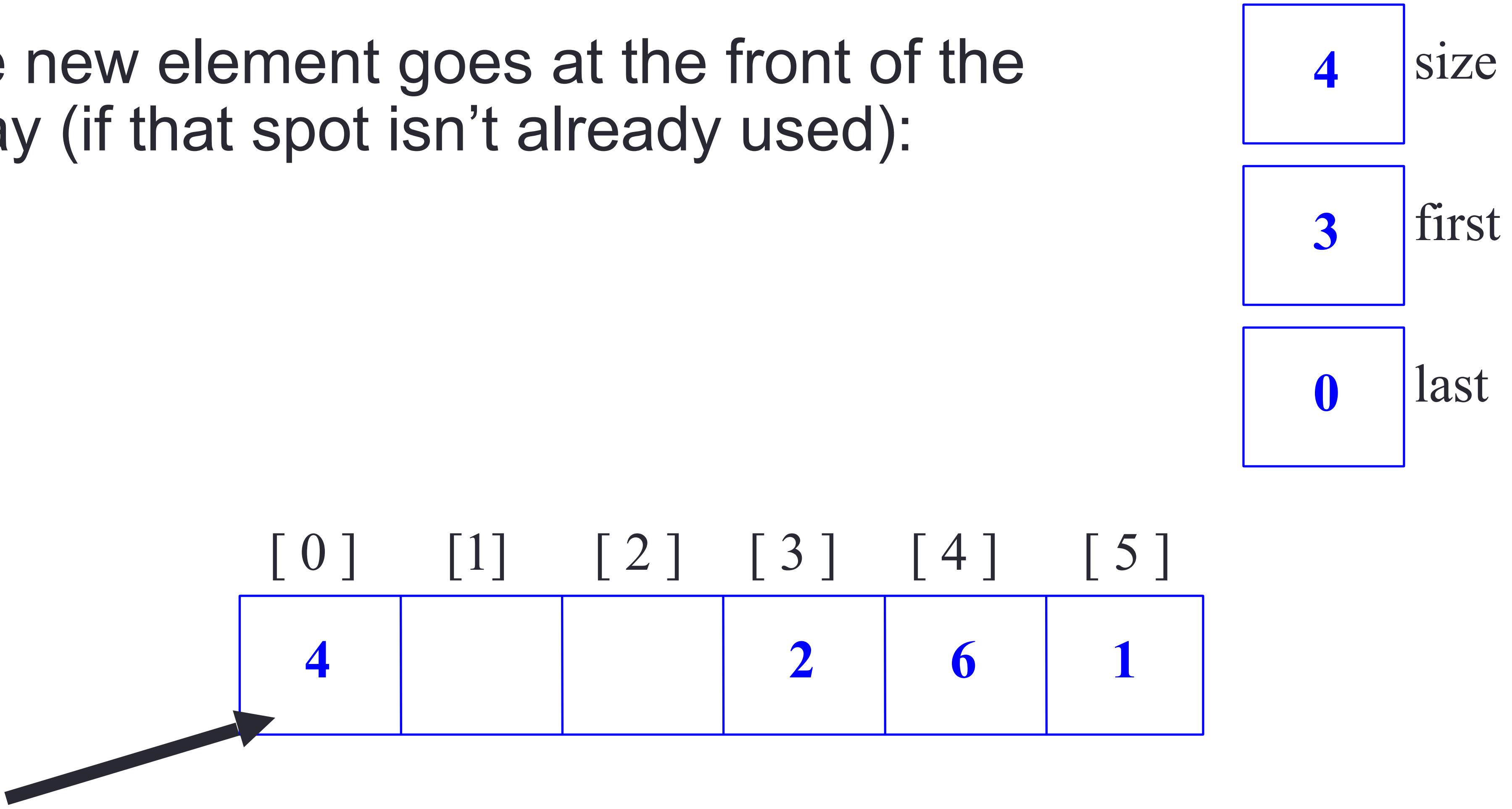
At the End of the Array

- There is special behavior at the end of the array. For example, suppose we want to add a new element to this queue, where the last index is [5]:



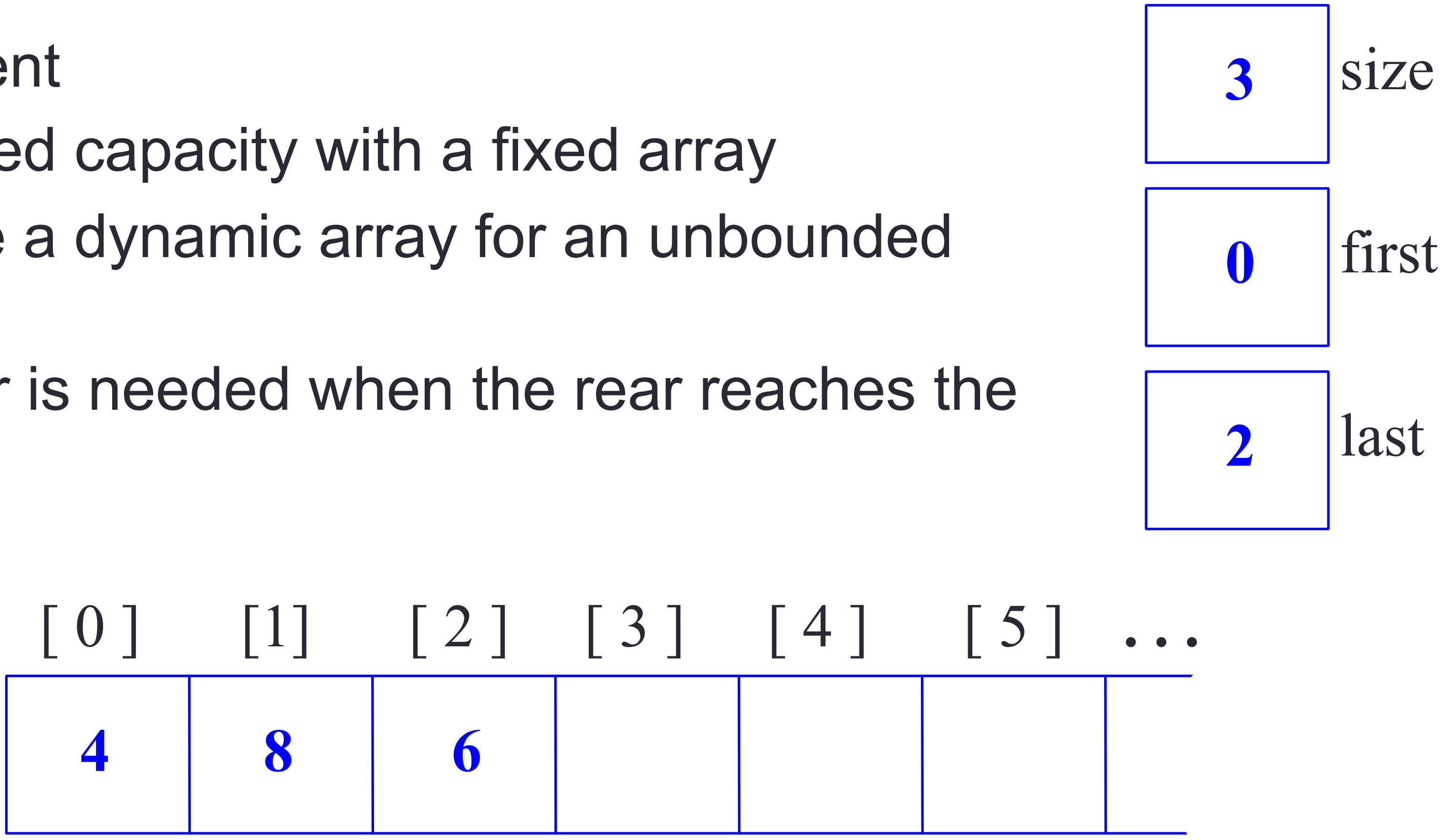
At the End of the Array

- The new element goes at the front of the array (if that spot isn't already used):



Array Implementation

- Easy to implement
- But it has a limited capacity with a fixed array
- Or you must use a dynamic array for an unbounded capacity
- Special behavior is needed when the rear reaches the end of the array.



Summary

- Like stacks, queues have many applications.
- Items enter a queue at the rear and leave a queue at the front.
- Queues can be implemented using an array or using a linked list.