Fakultas Informatika
School of Computing
Telkom University

CDK2AAB4
STRUKTUR DATA

Linked List Implementation

Queue

SLL last update: Oct 20, 2024

Queue

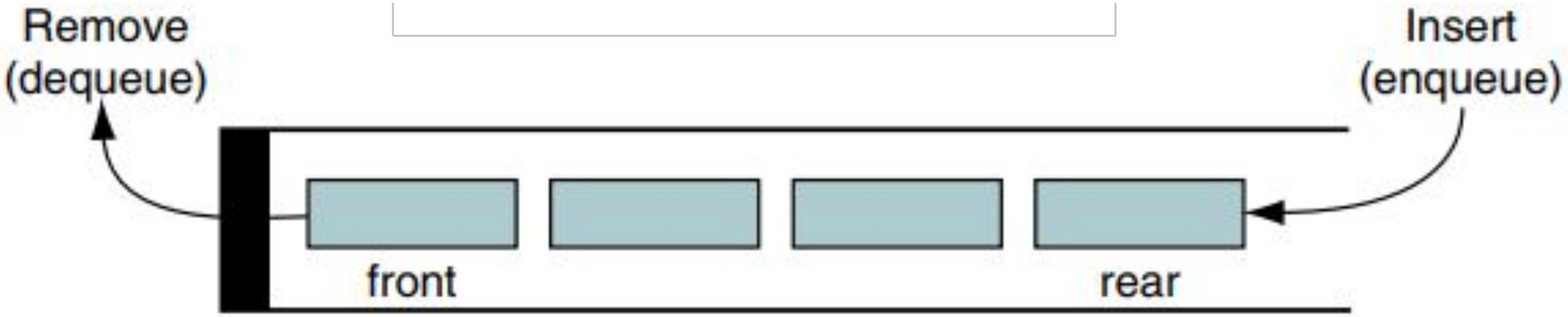
- A queue is a linear list in which data can only be inserted at **one end**, called the **rear**, and deleted from the other end, called the **front**.

People join the queue at the rear
People leave the queue at the front

- In a queue the first item inserted is the first to be removed (**First-In-First-Out, FIFO**)

Ref. Lafore, 2003: Data Structures & Algorithms in Java

Queue

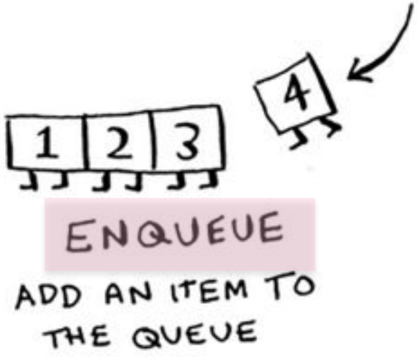


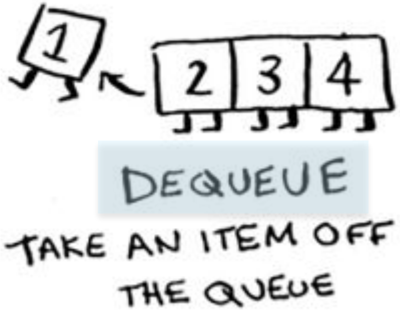
Ref. Gilberg & Forouzan, 2005: Data Structures-
A Pseudocode Approach with C

Primary Queue Operations

- **`enqueue(el)`**—Put the element el at the end of the queue.

- **`dequeue()`**—Take the first element from the queue.





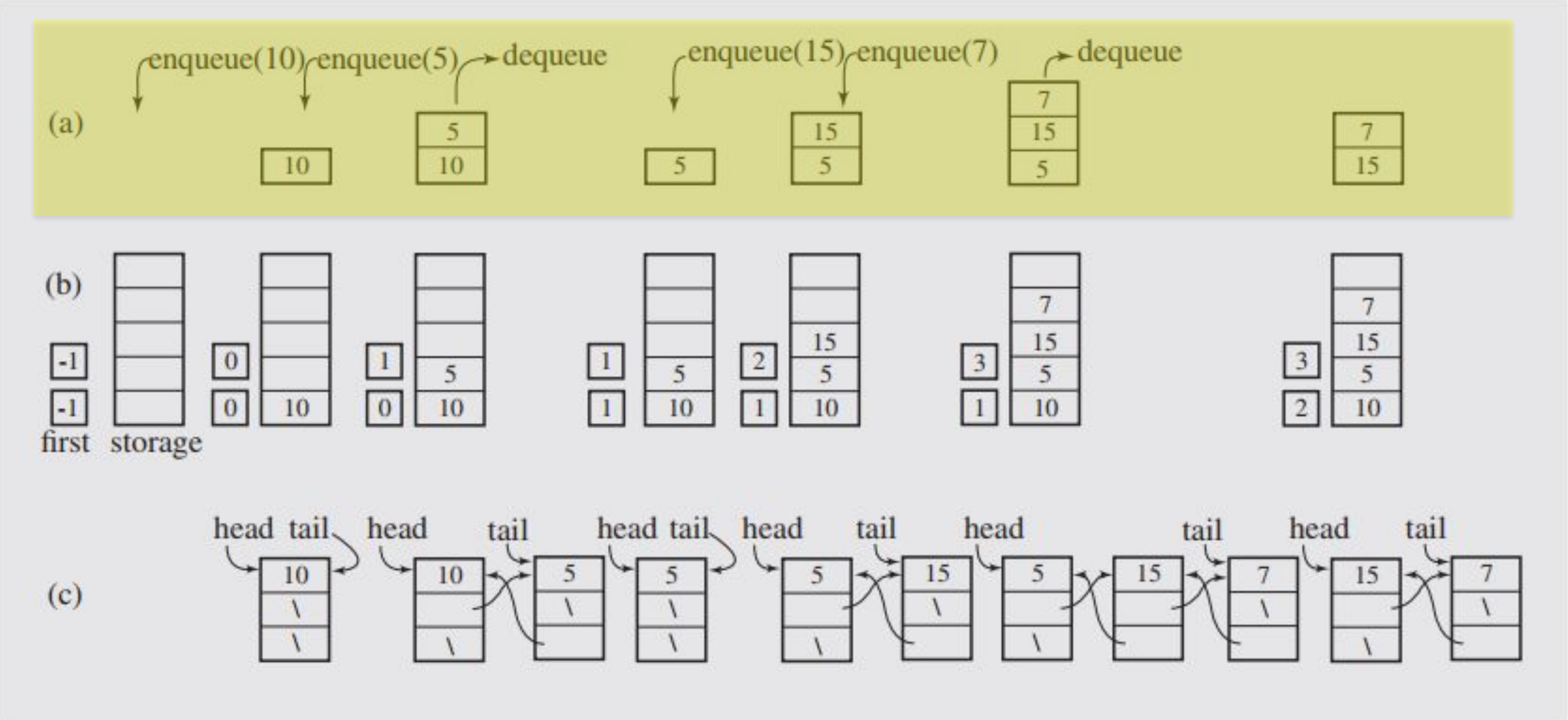
Ref. Drozdek, 2013: Data Structures and Algorithms in C++
Bhargava, 2016: Grokking Algorithms

Auxiliary Queue Operations

- **`isEmpty()`**—Check to see if the queue is empty.
- **`front()`**—Return the first element in the queue without removing it.
- **`size()`**—Return the number of element in the queue.

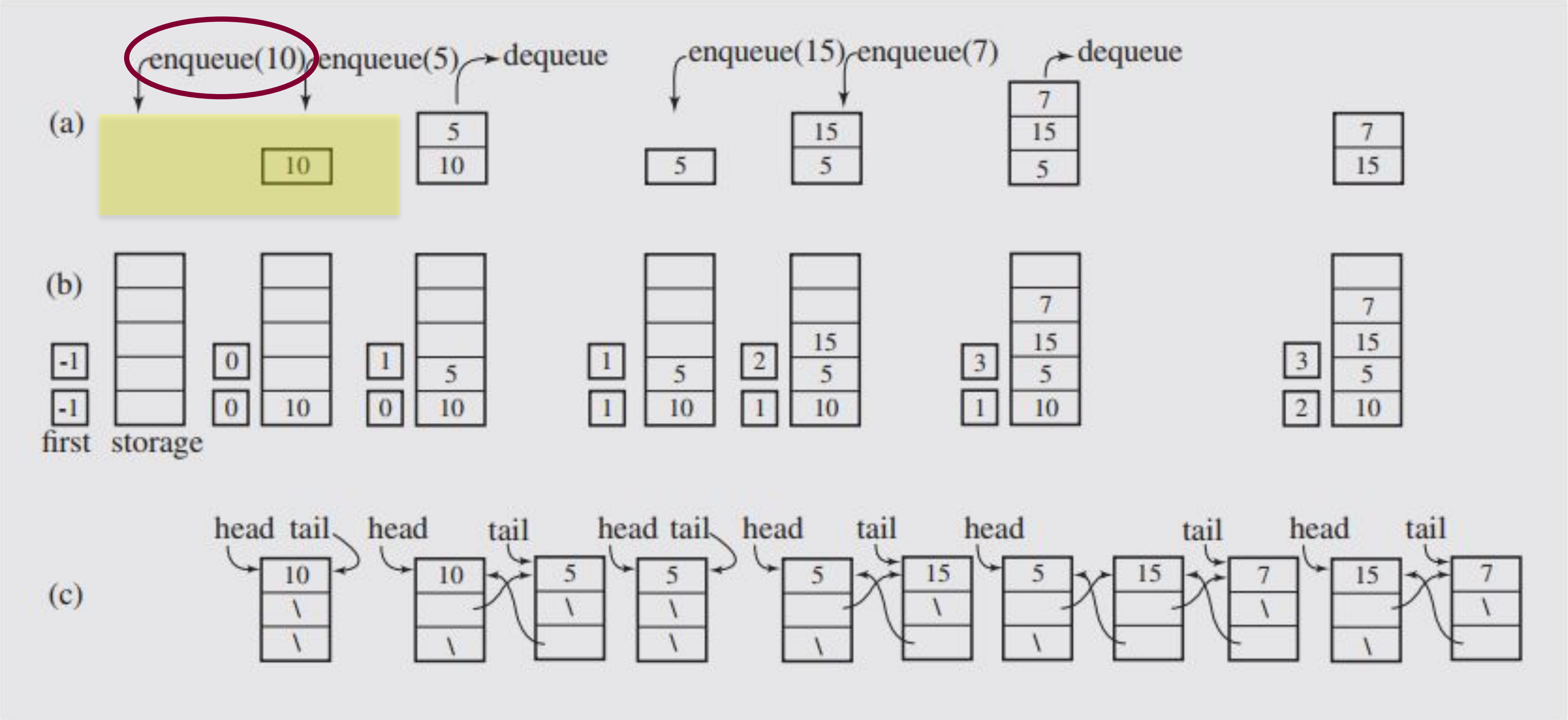
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



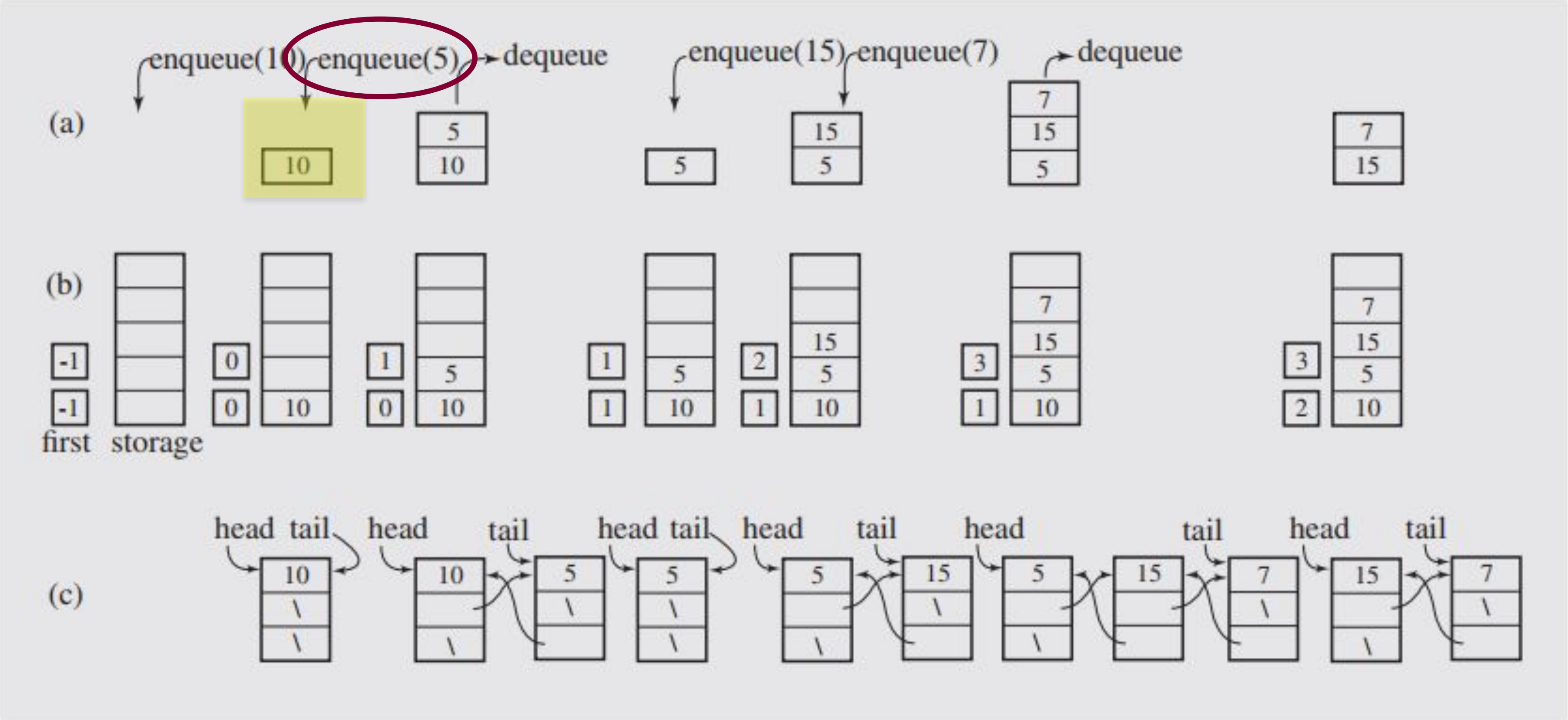
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



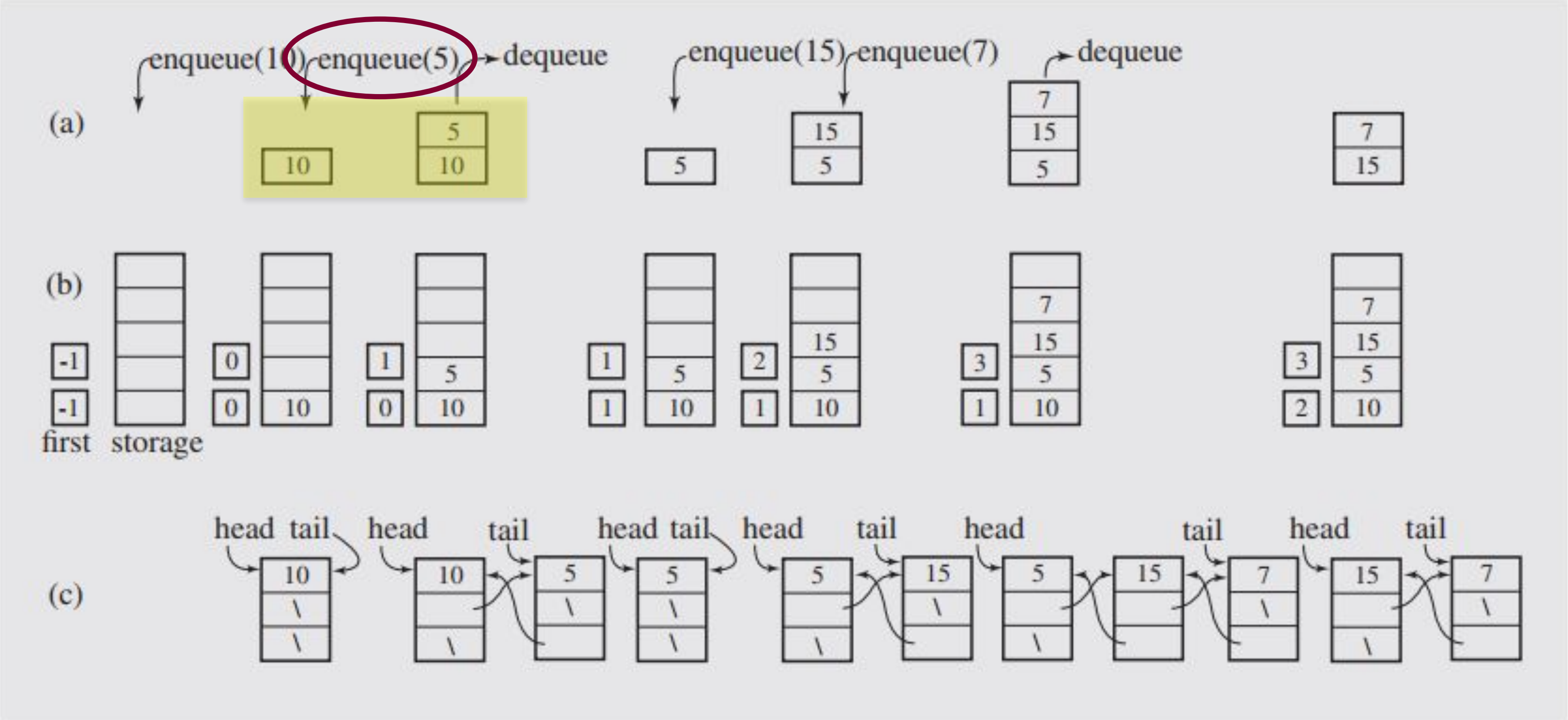
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



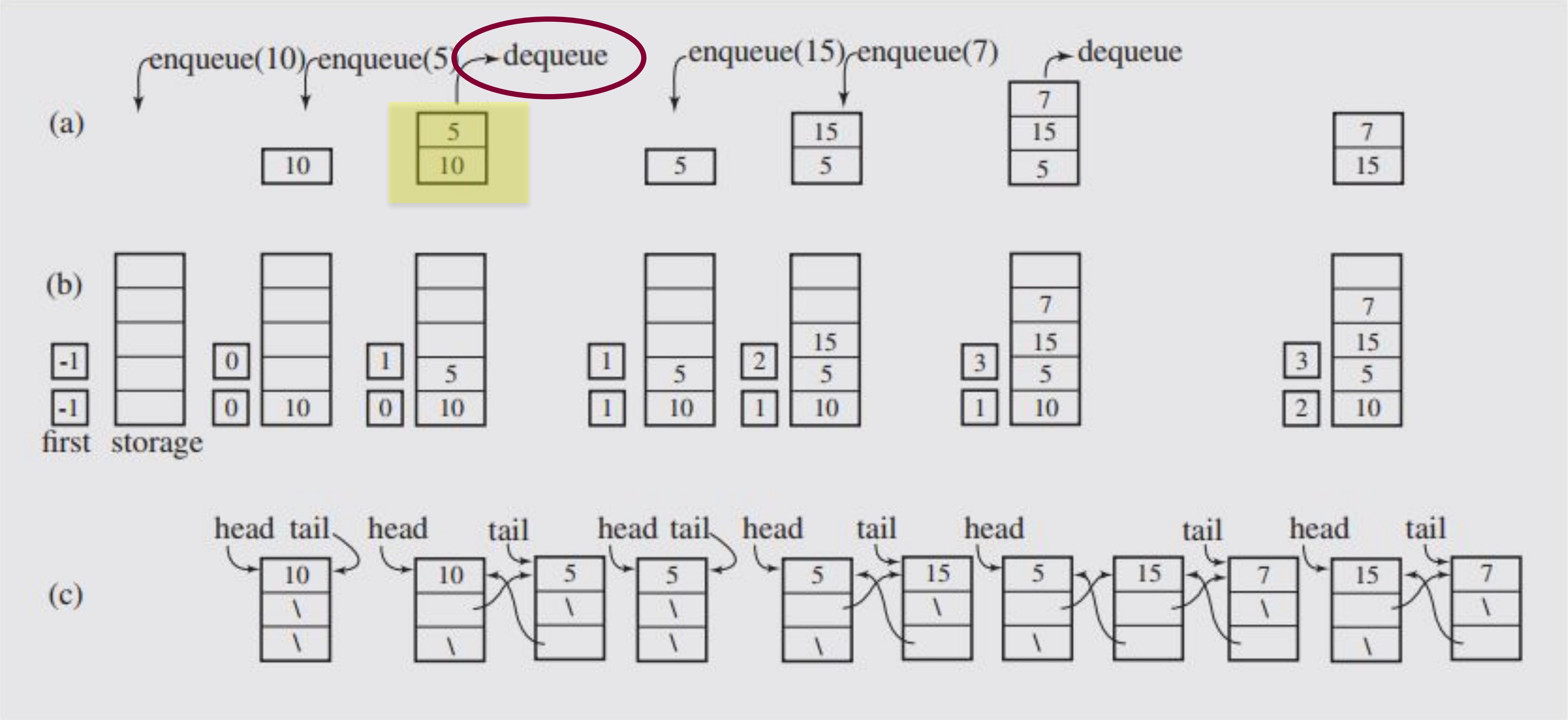
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



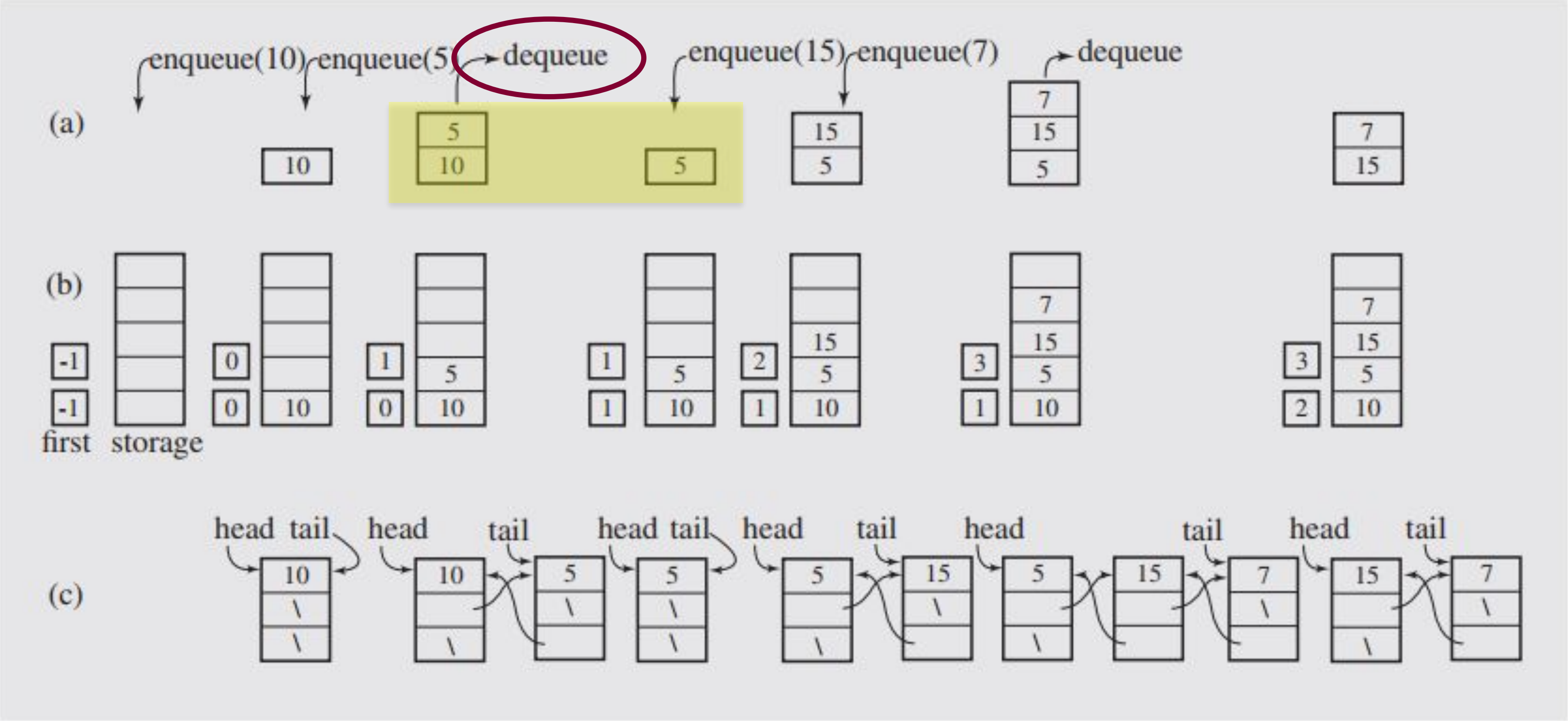
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



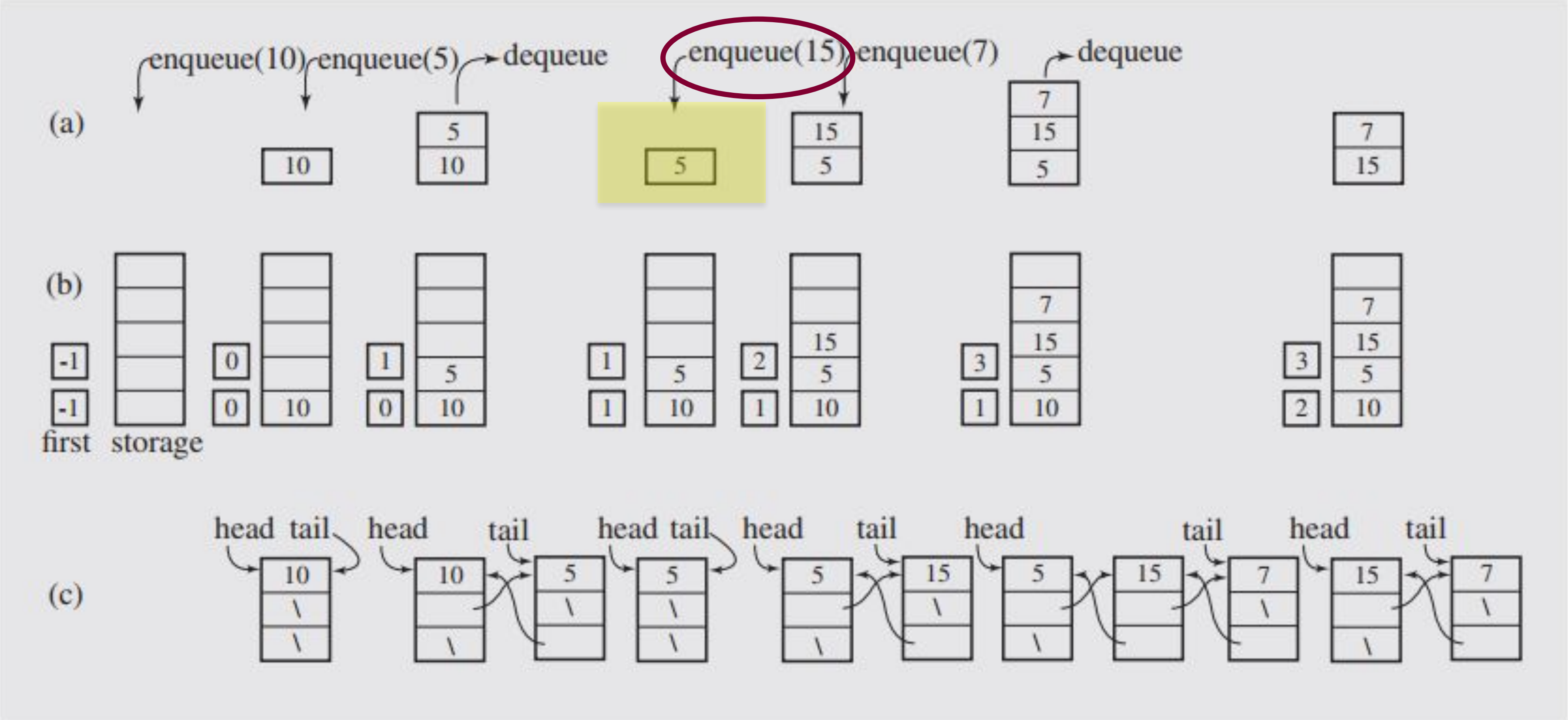
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



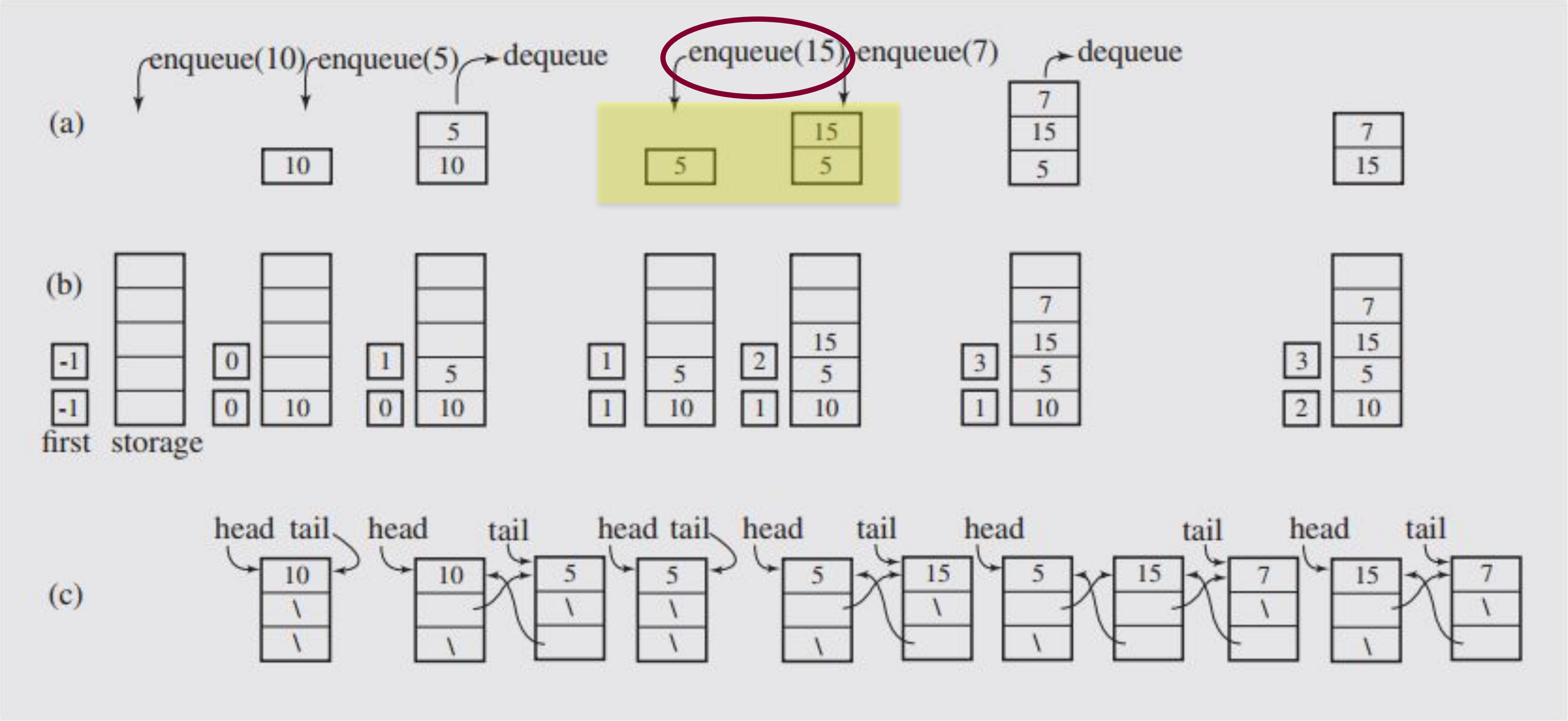
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



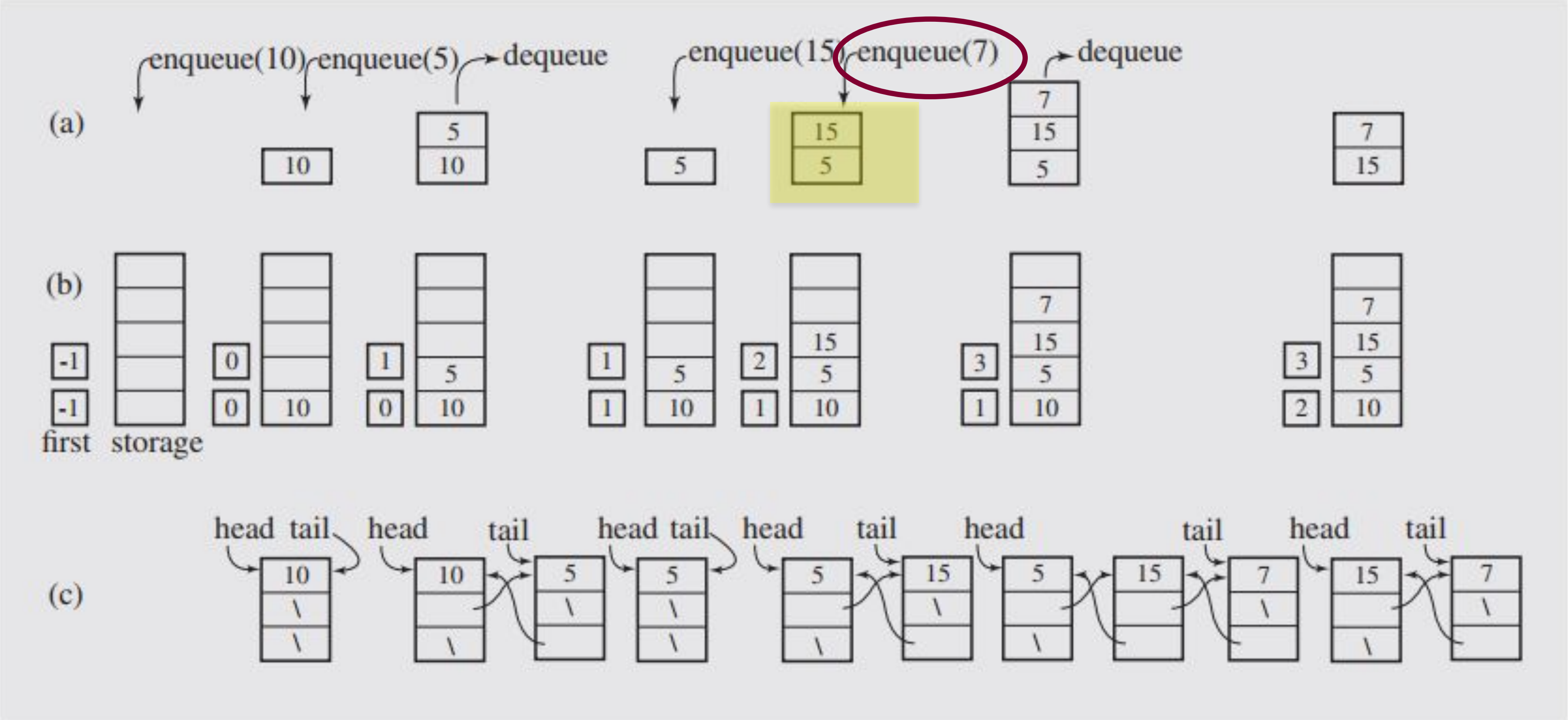
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



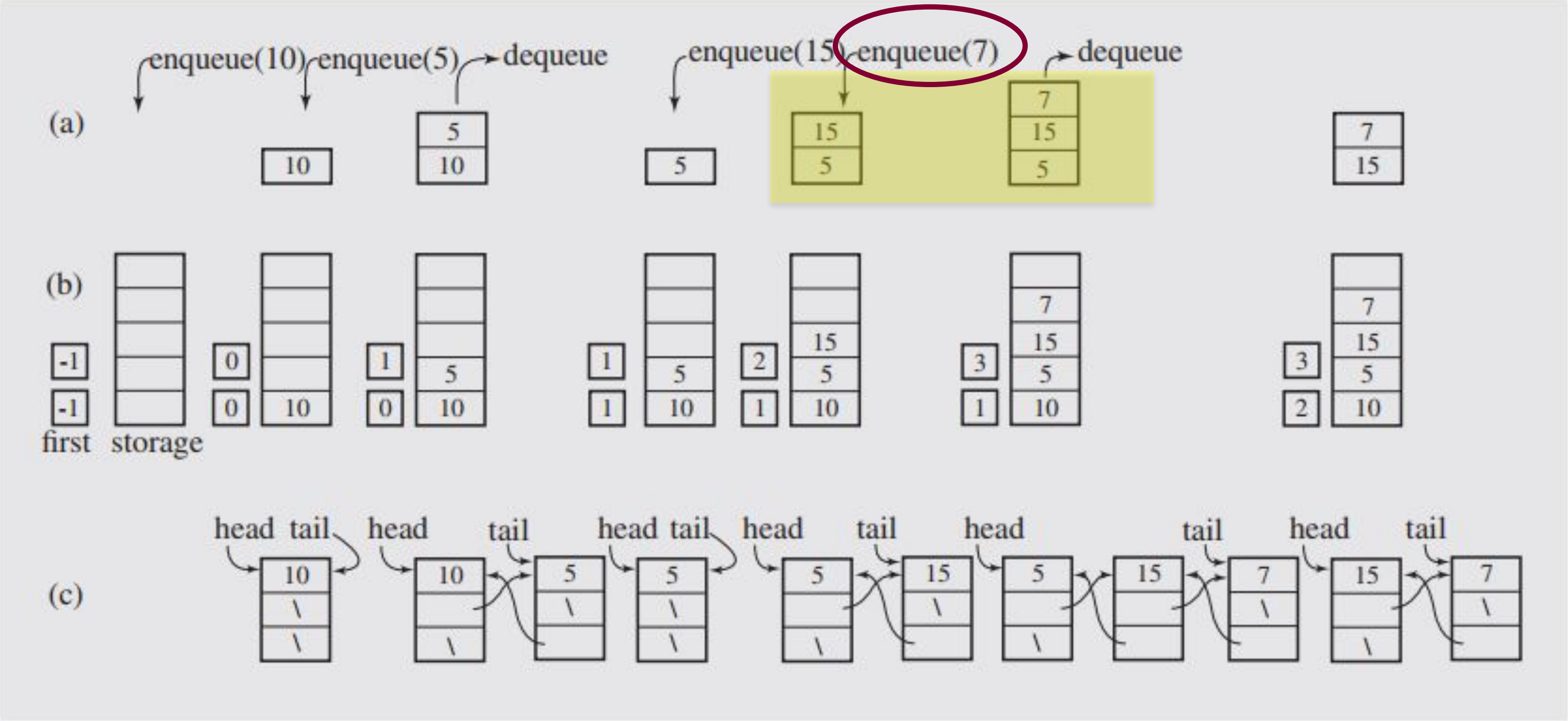
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



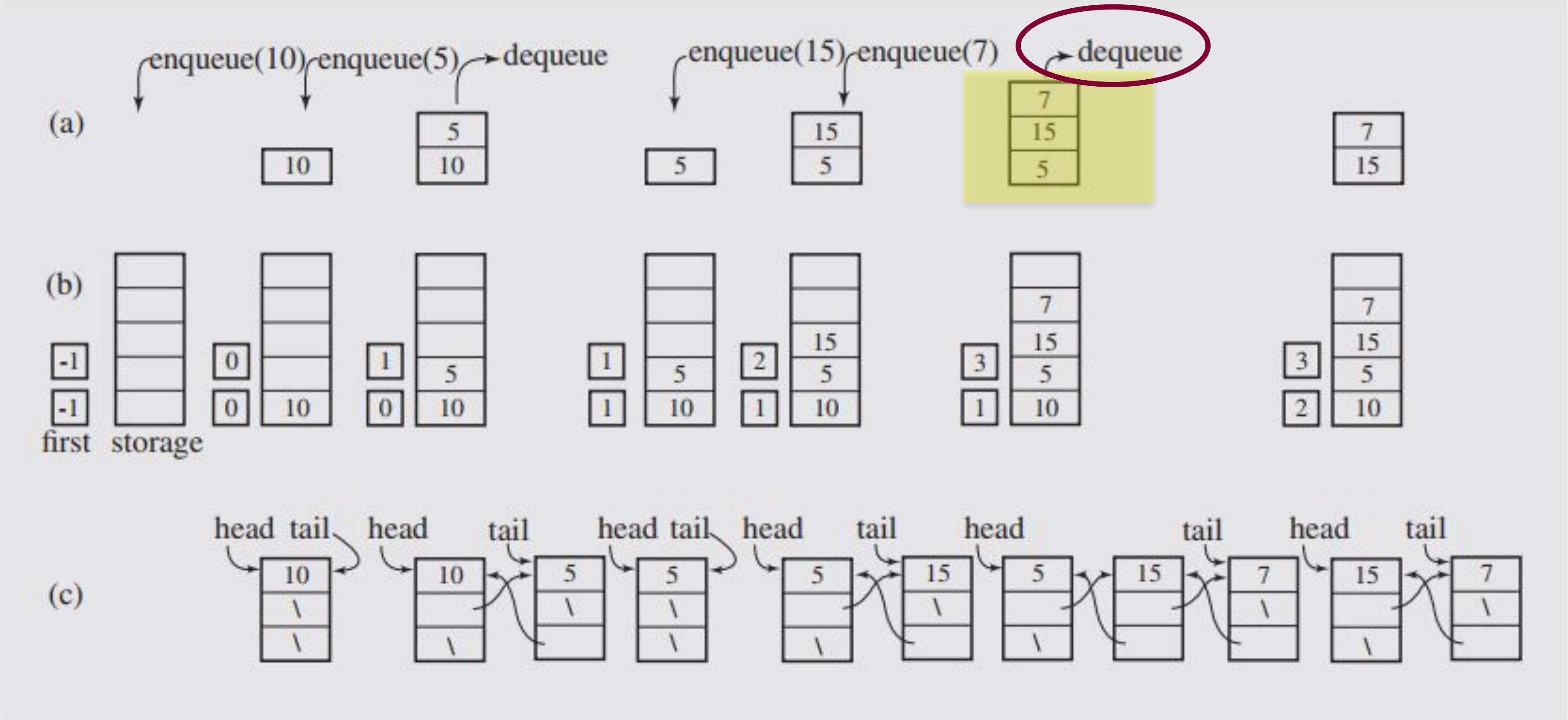
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



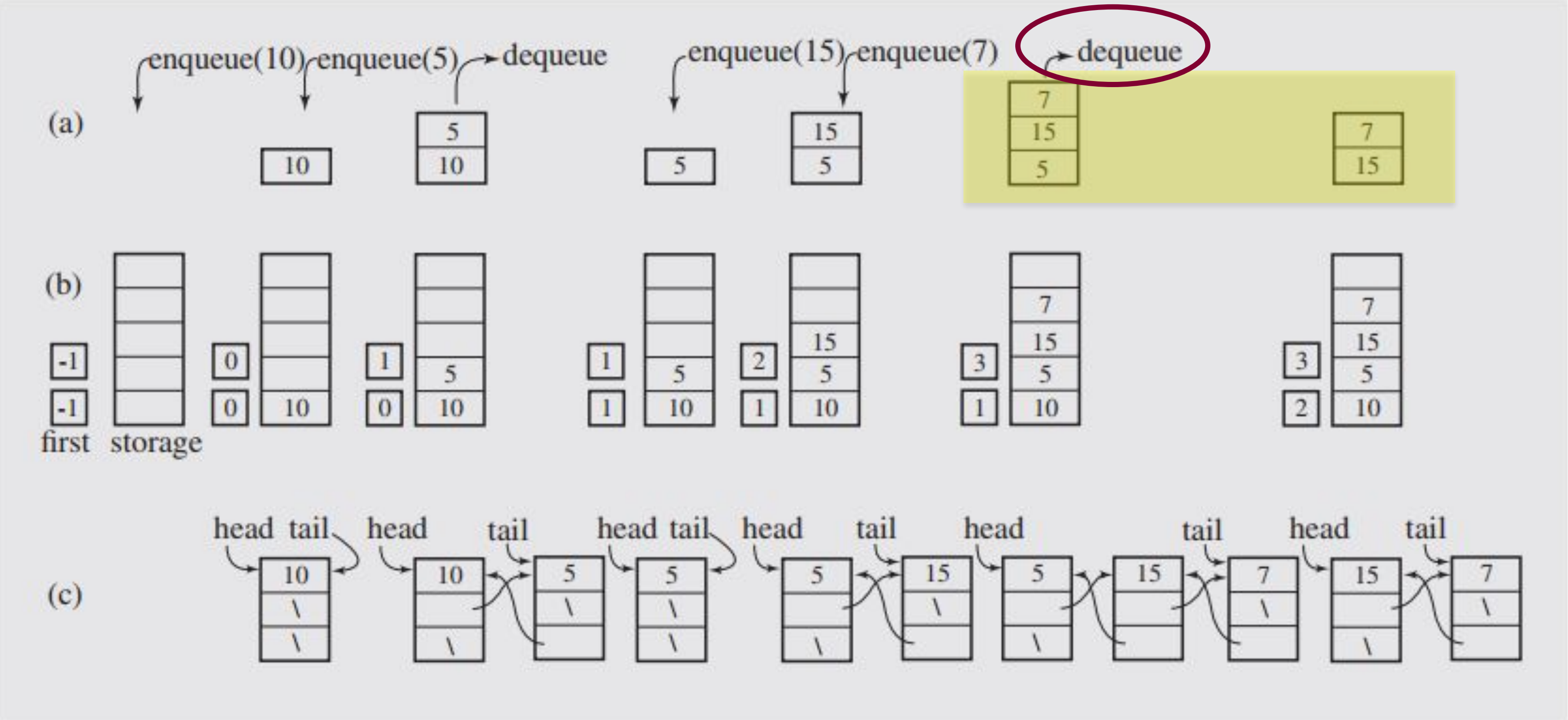
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



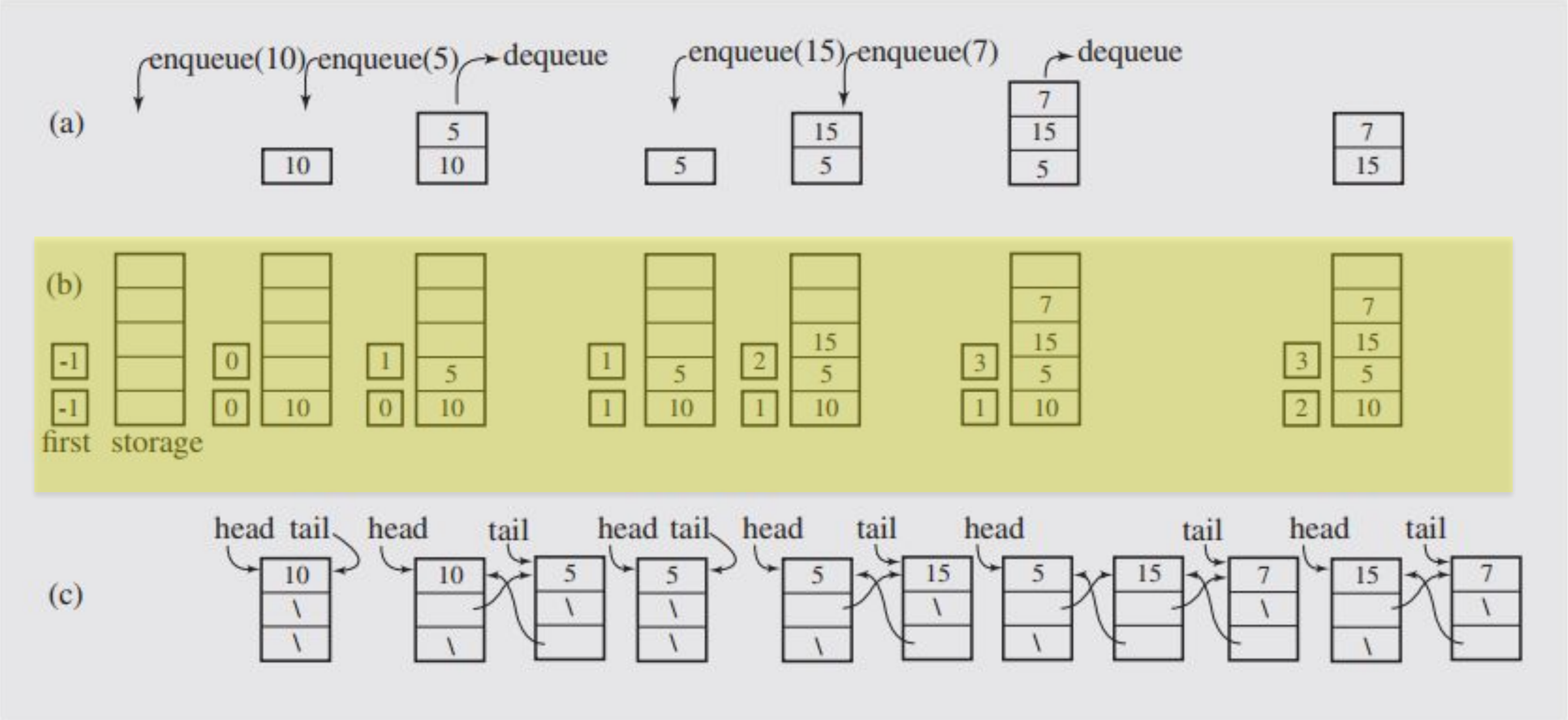
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



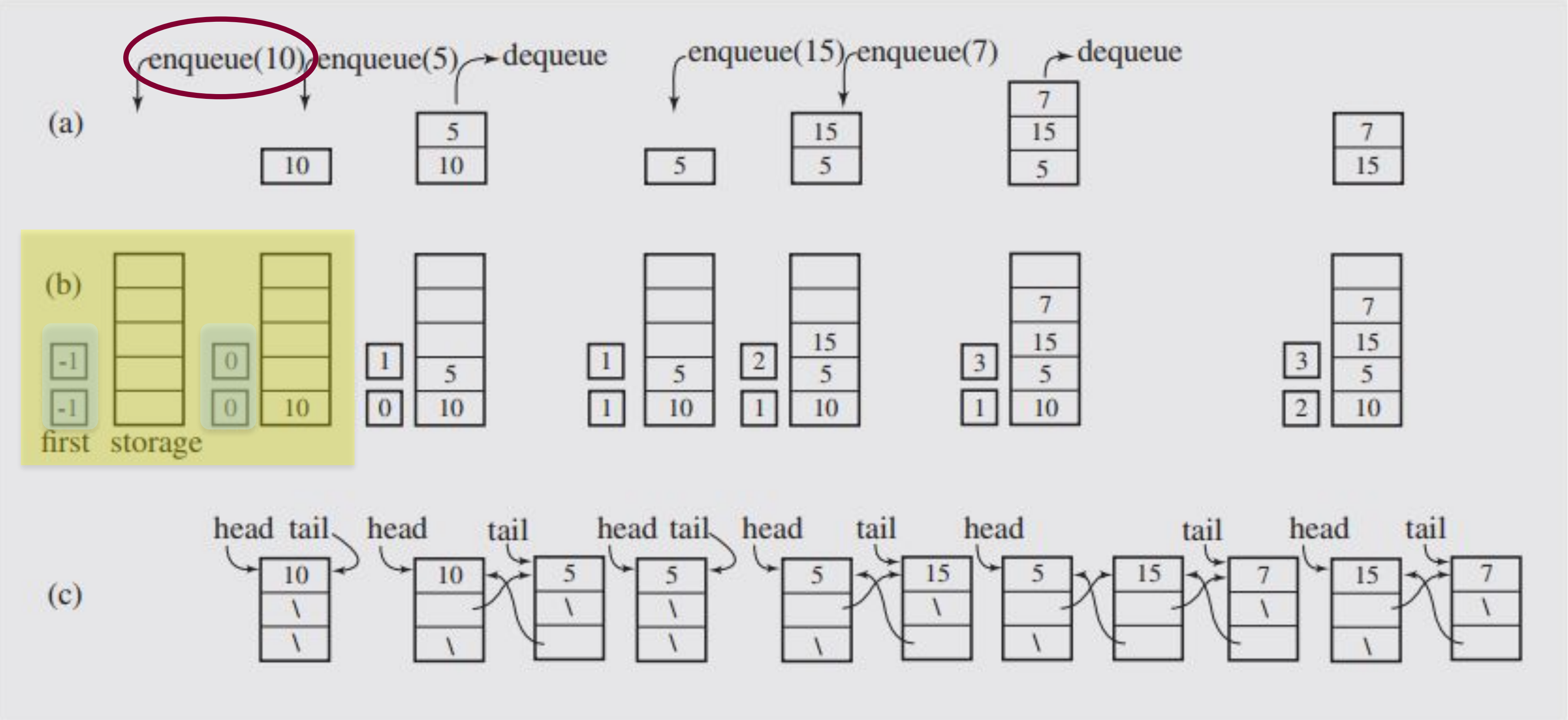
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



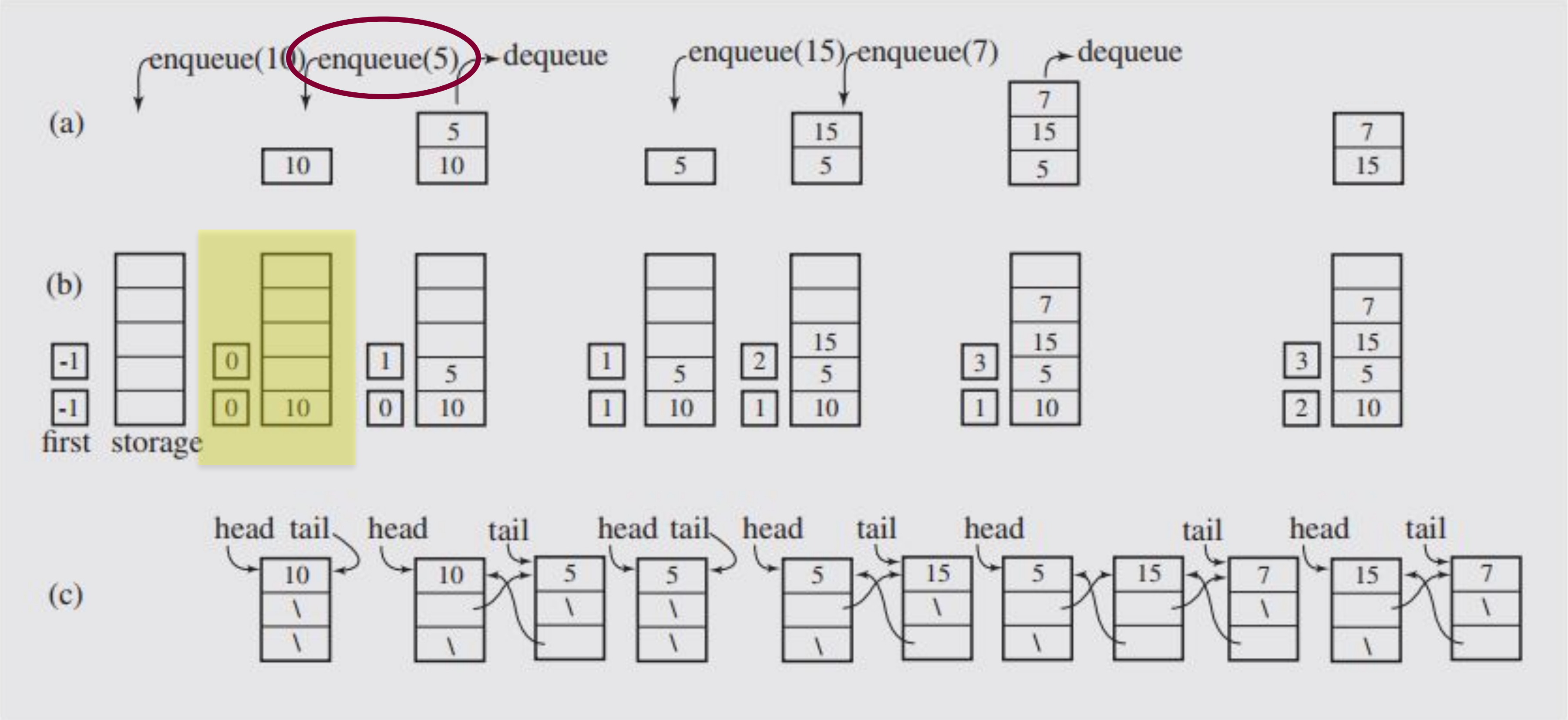
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



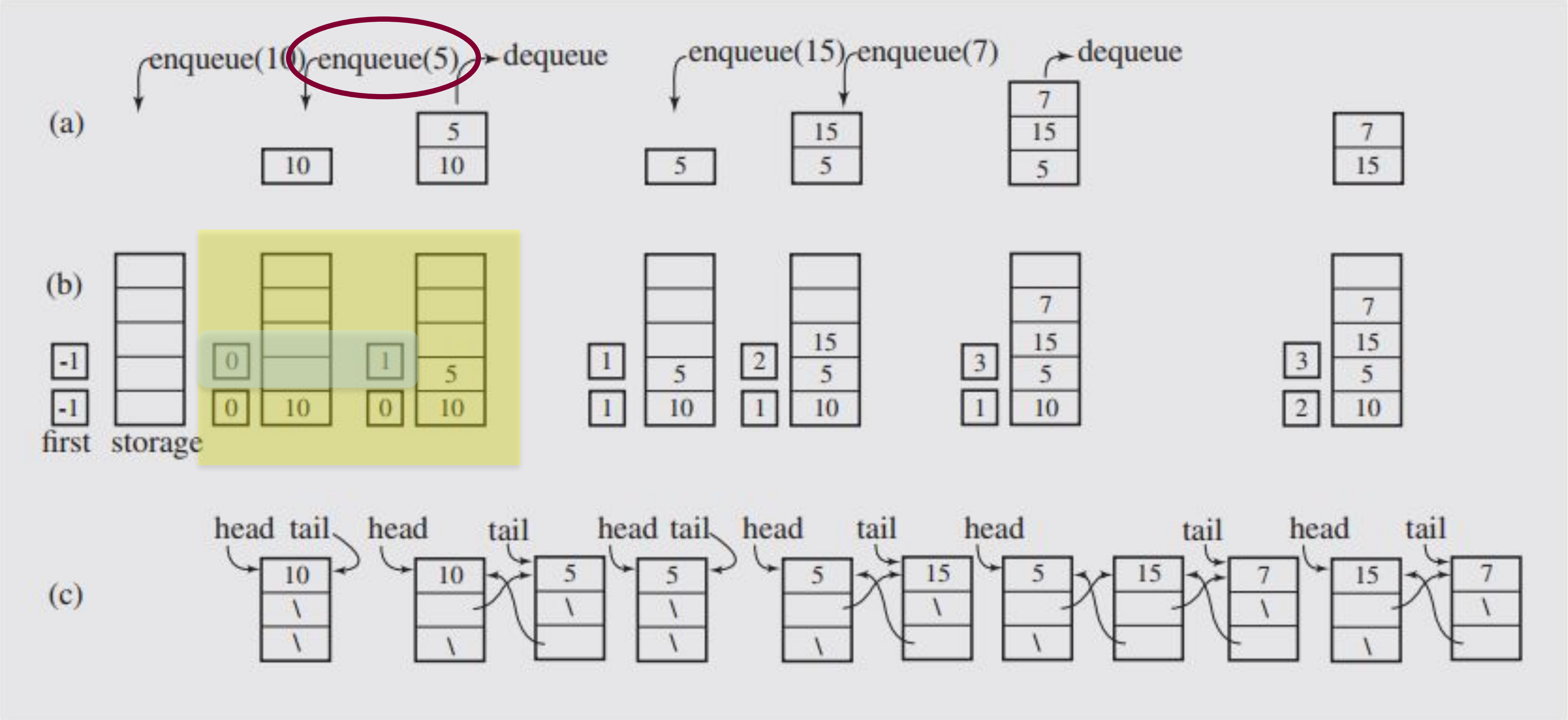
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



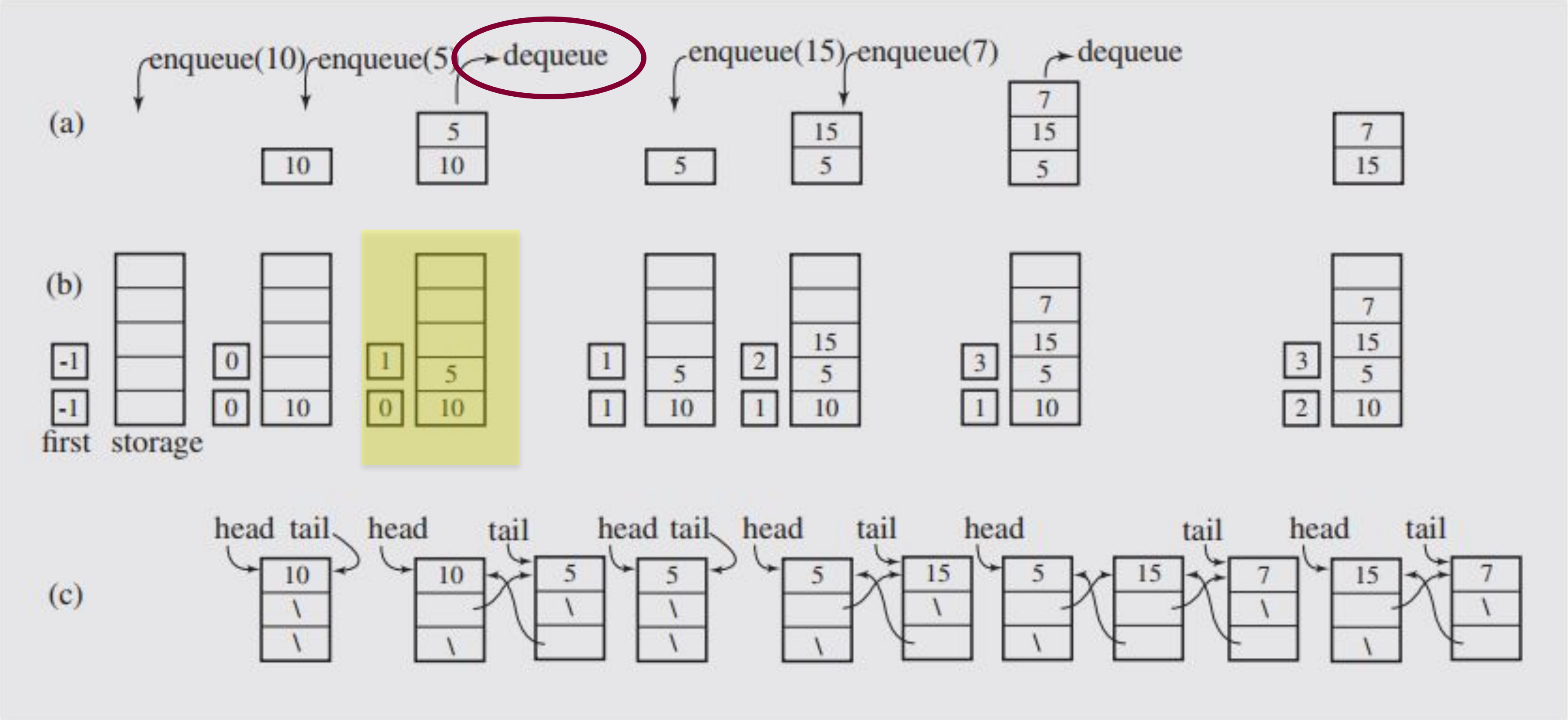
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



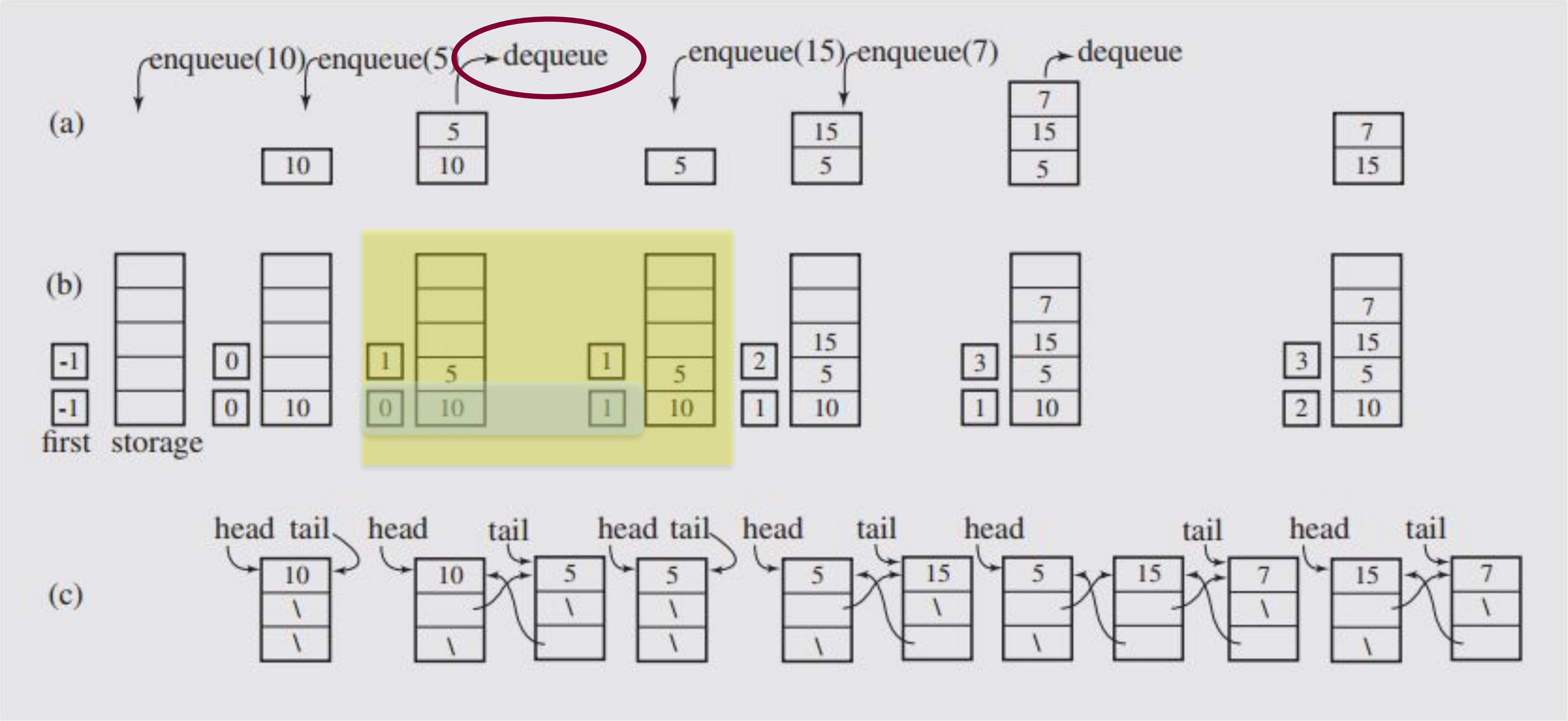
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



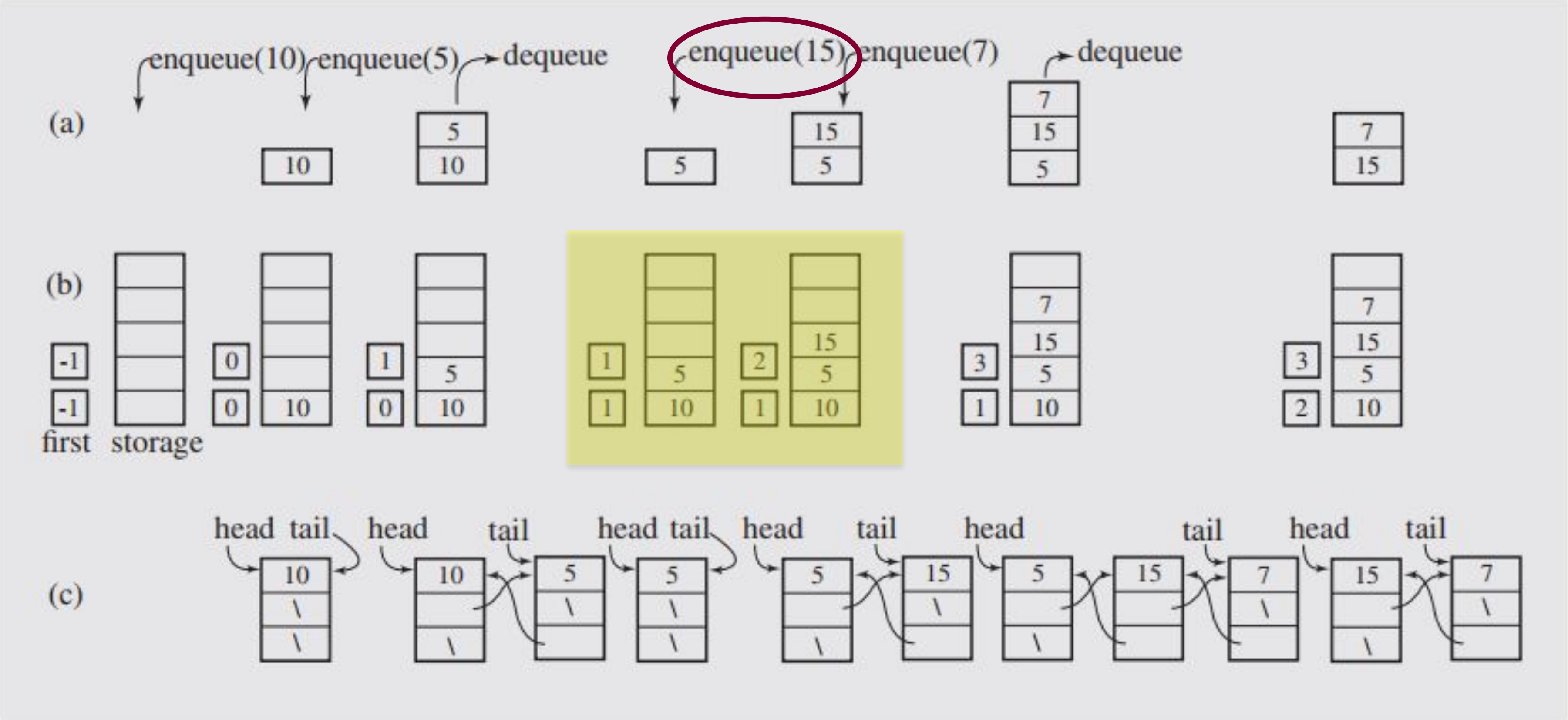
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



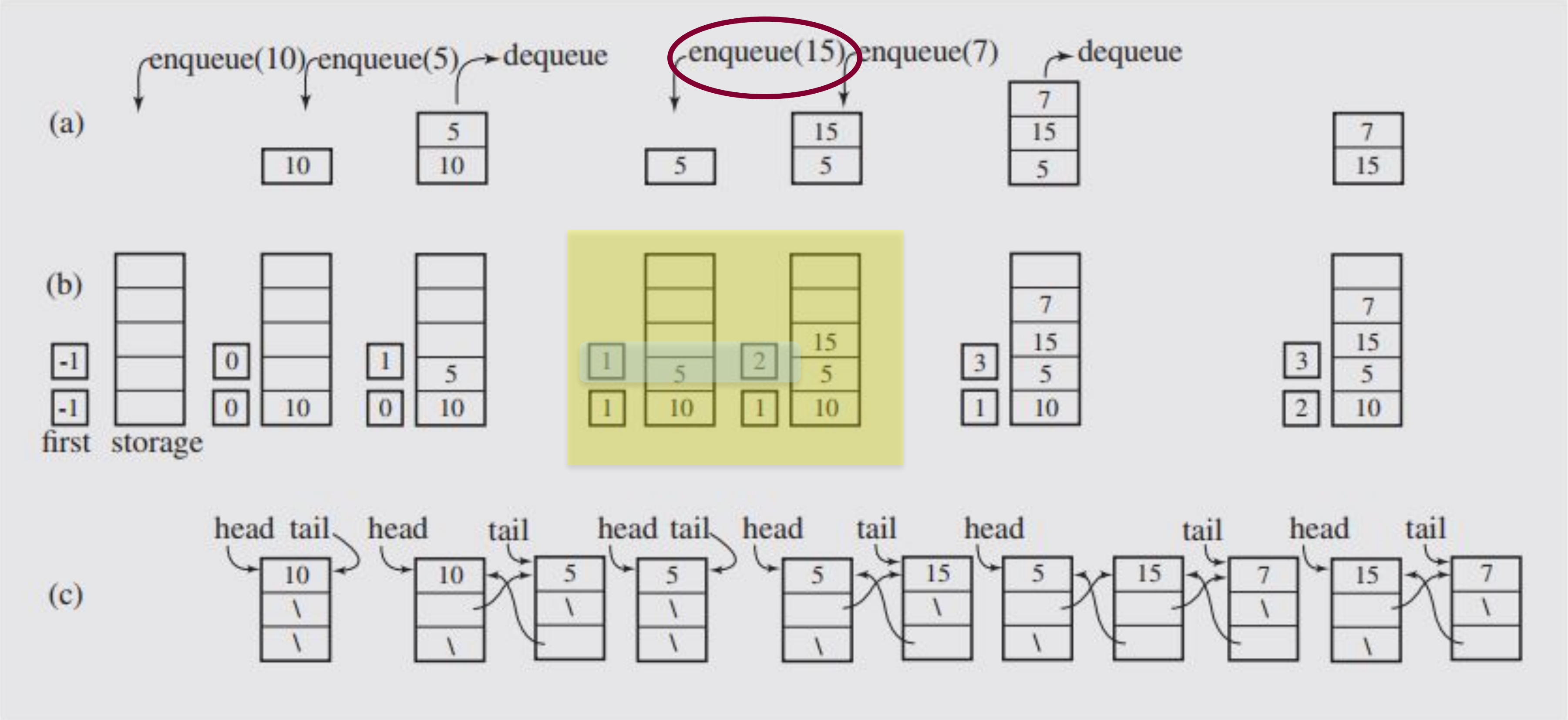
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



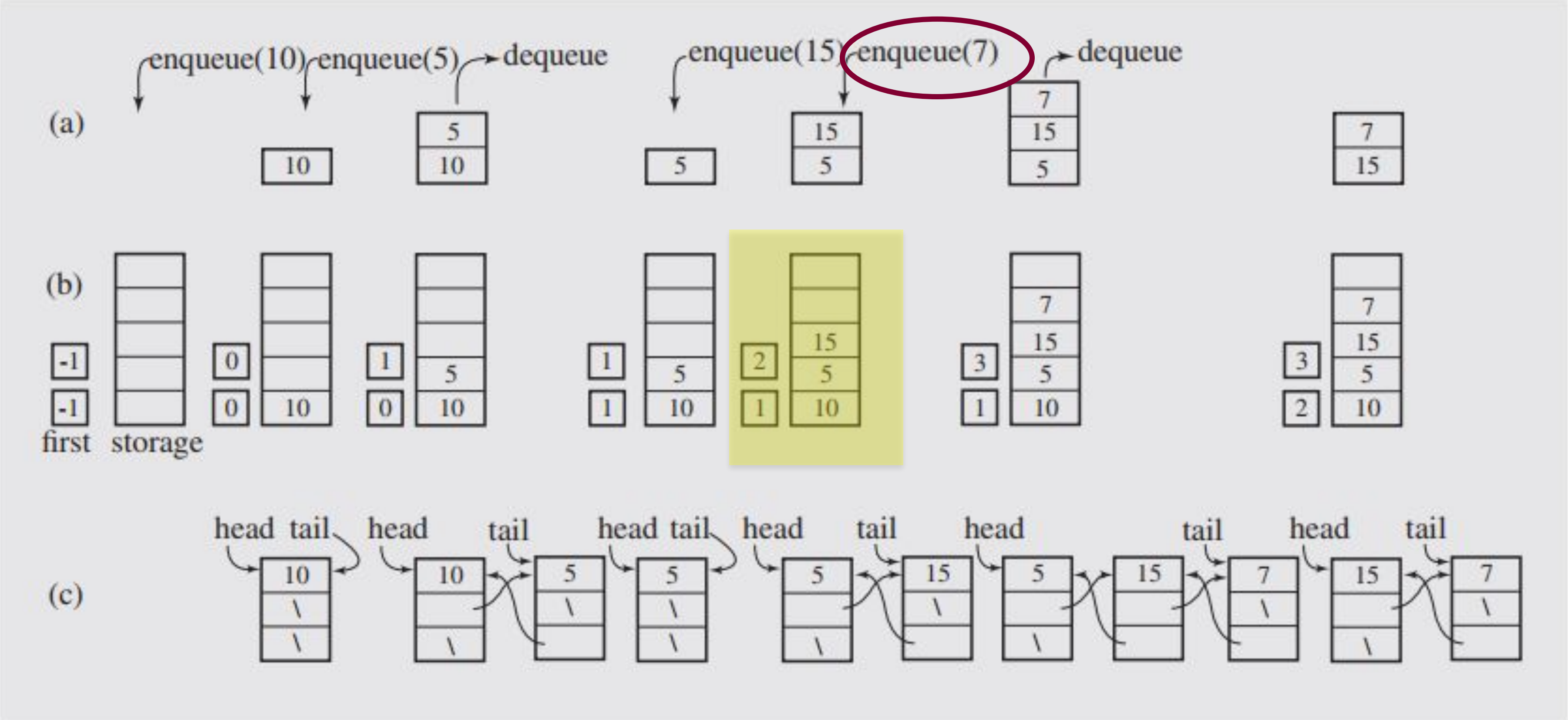
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



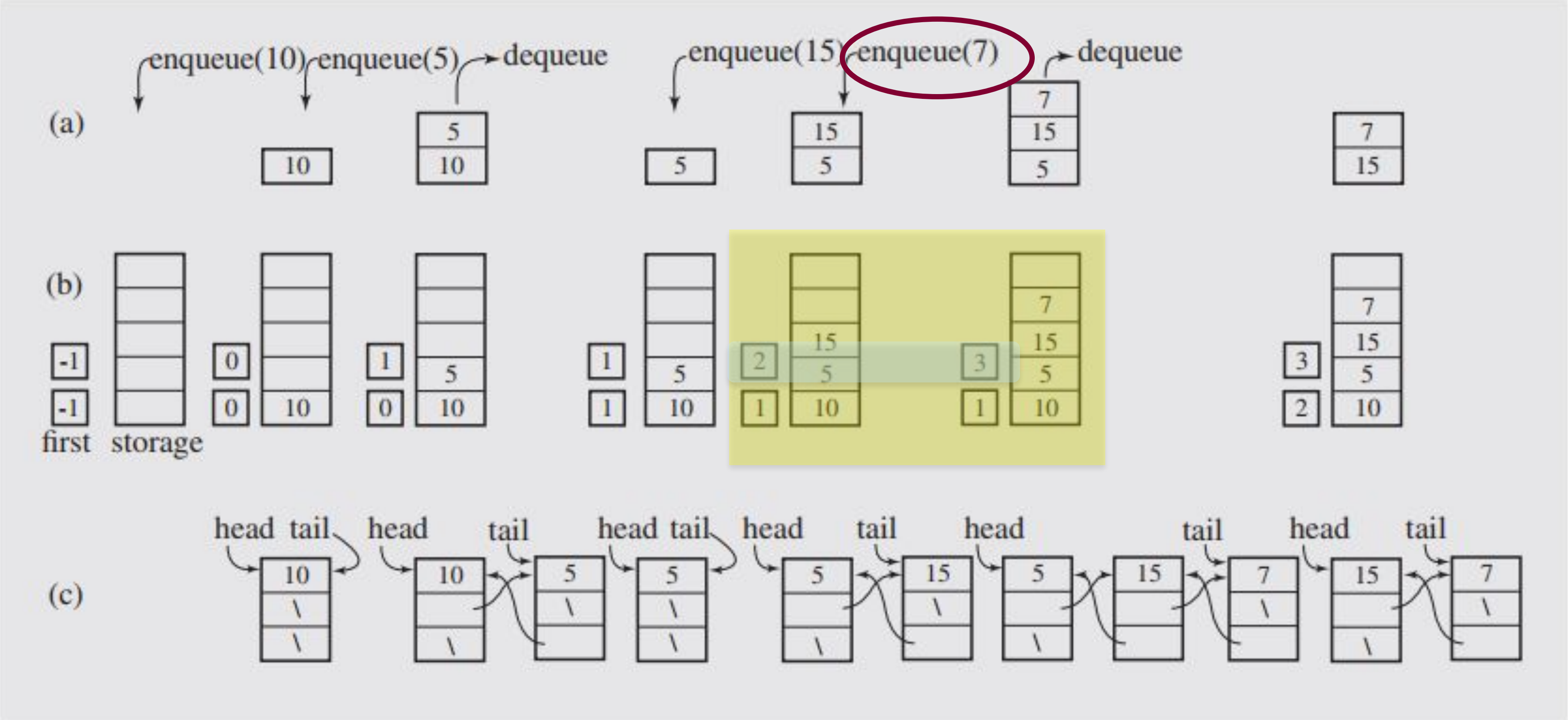
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



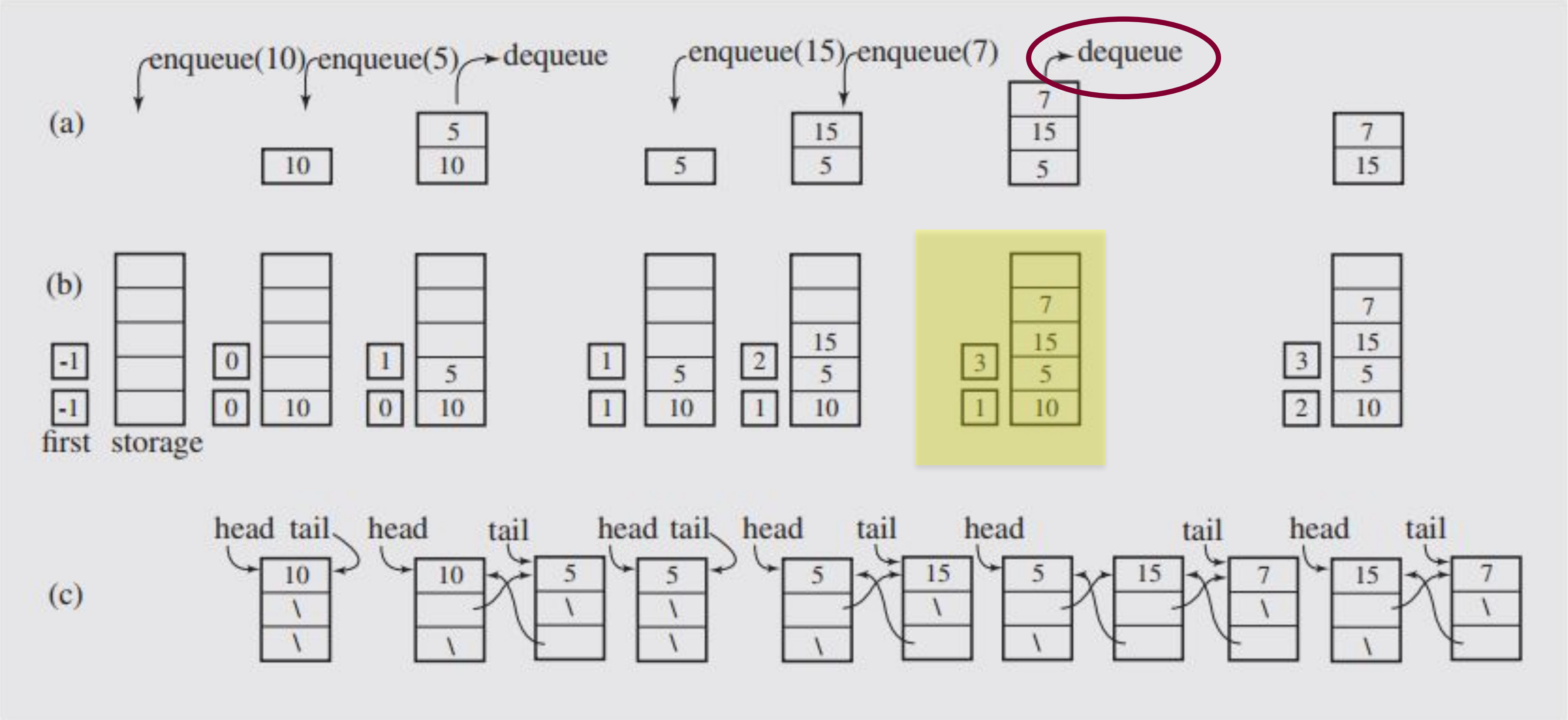
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



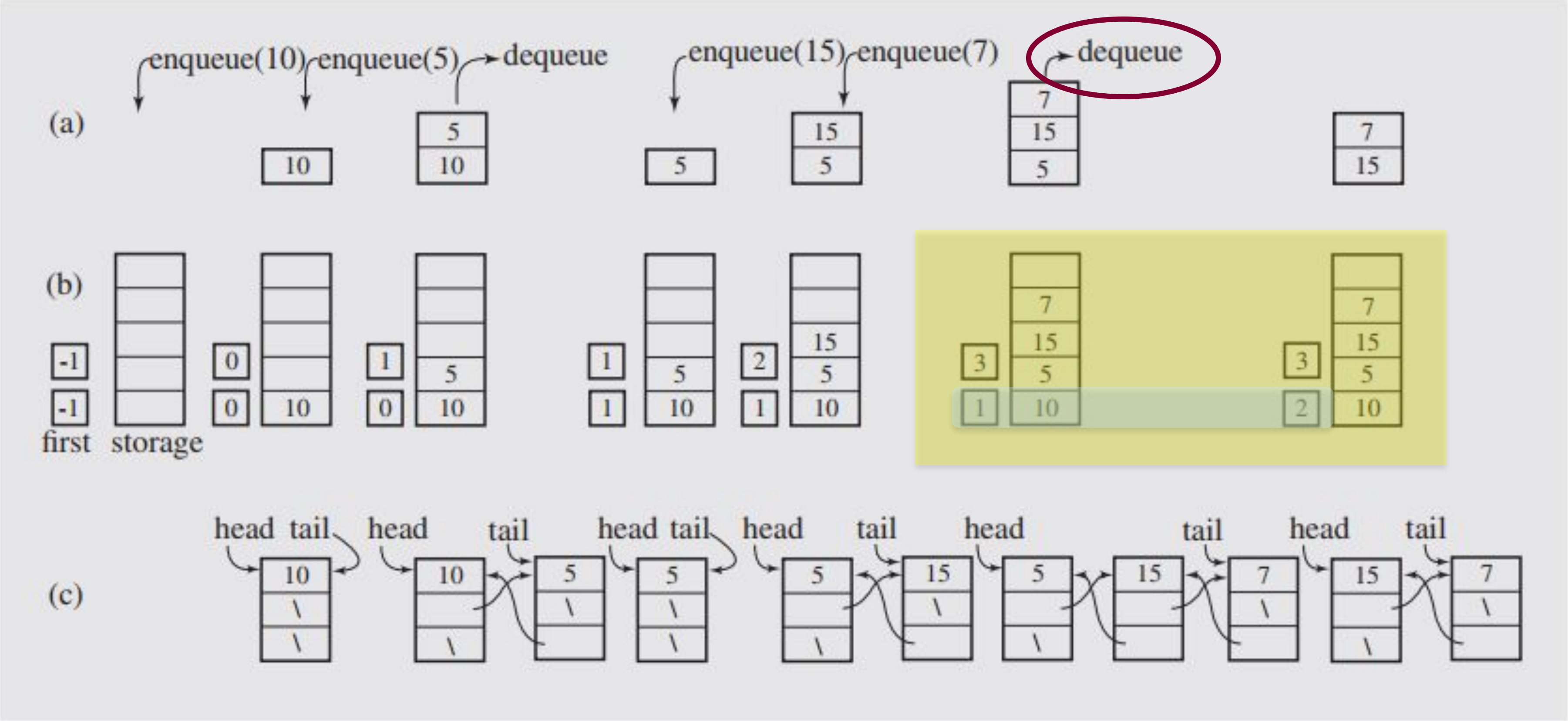
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.

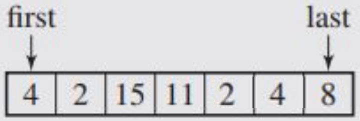


Ref. Drozdek, 2013: Data Structures and Algorithms in C++

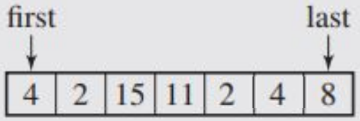
FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



Ref. Drozdek, 2013: Data Structures and Algorithms in C++



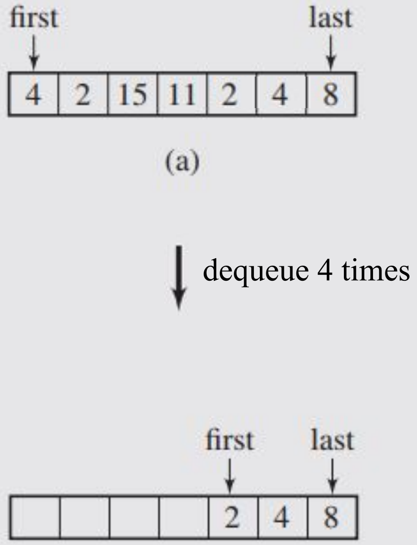
(a)



(a)

↓ dequeue 4 times

Ref. Drozdek, 2013: Data Structures and Algorithms in C++

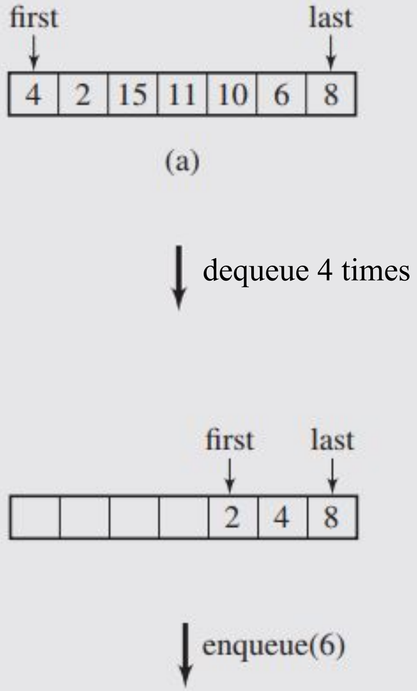


4	2	15	11	2	4	8

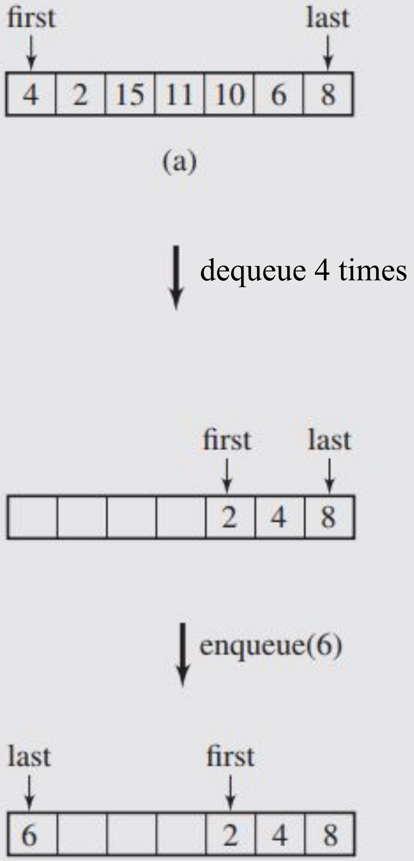
(a)

				2	4	8

Ref. Drozdek, 2013: Data Structures and Algorithms in C++



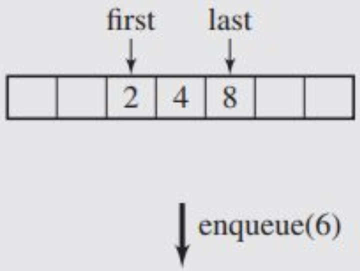
Ref. Drozdek, 2013: Data Structures and Algorithms in C++



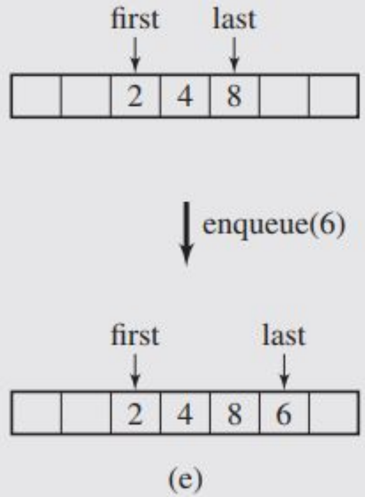
(a)

(d)

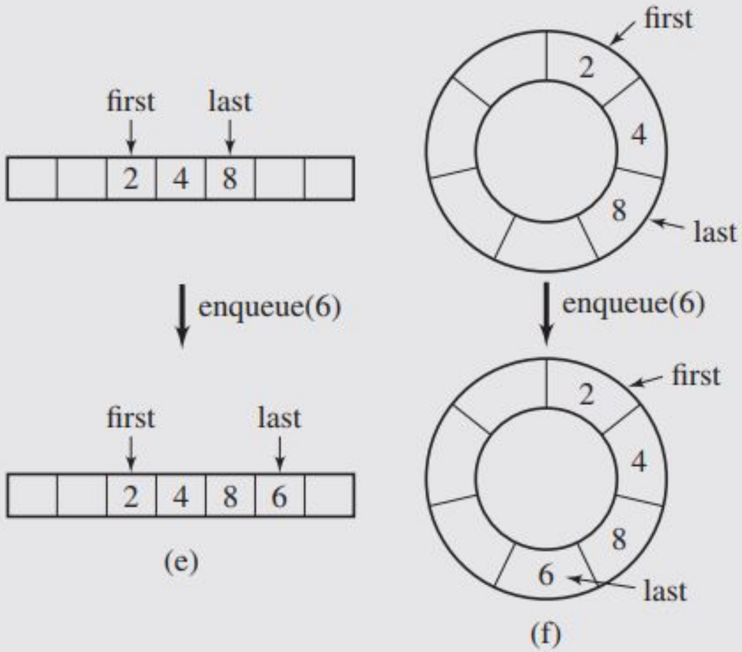
Ref. Drozdek, 2013: Data Structures and Algorithms in C++



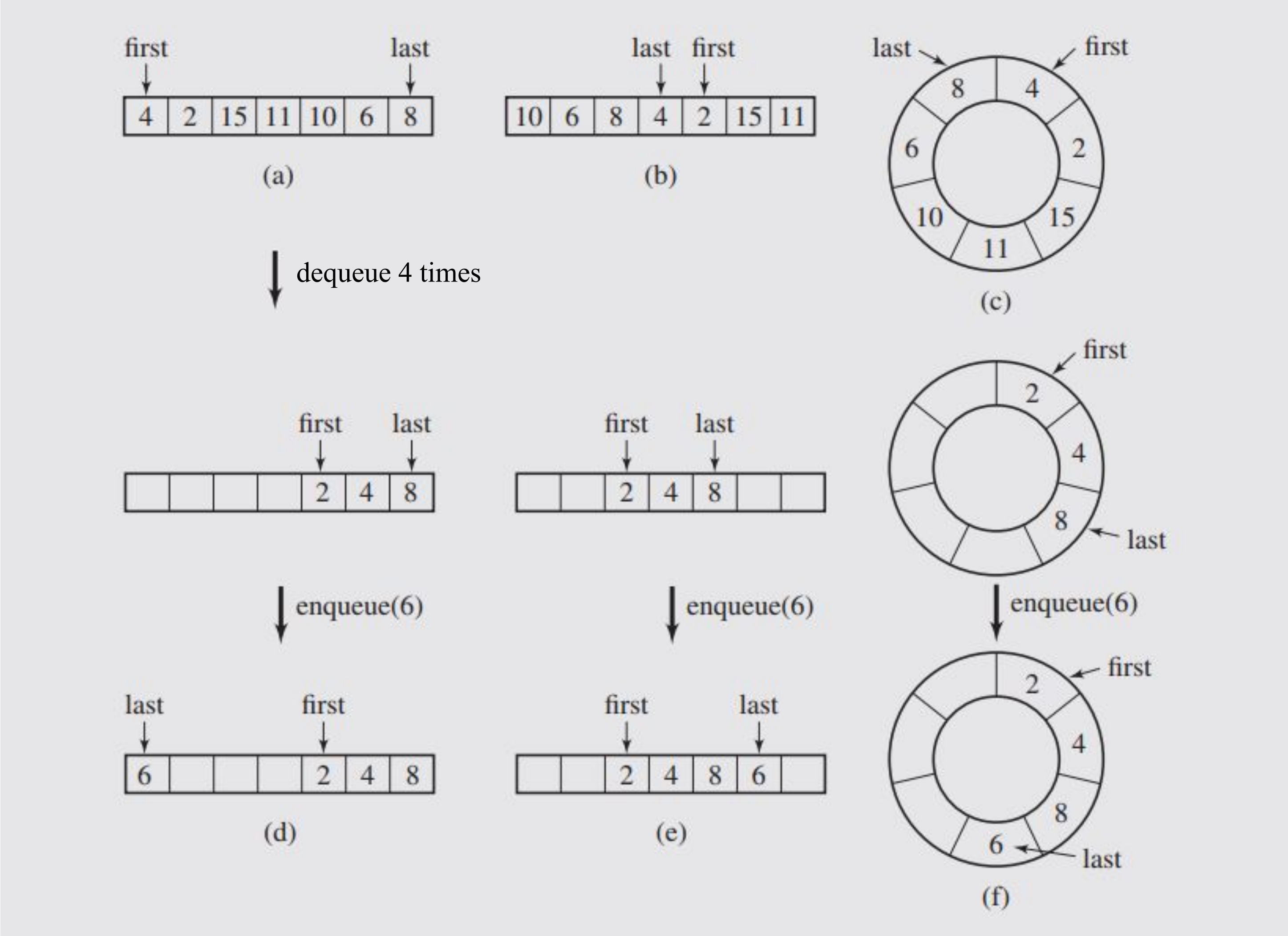
Ref. Drozdek, 2013: Data Structures and Algorithms in C++



Ref. Drozdek, 2013: Data Structures and Algorithms in C++

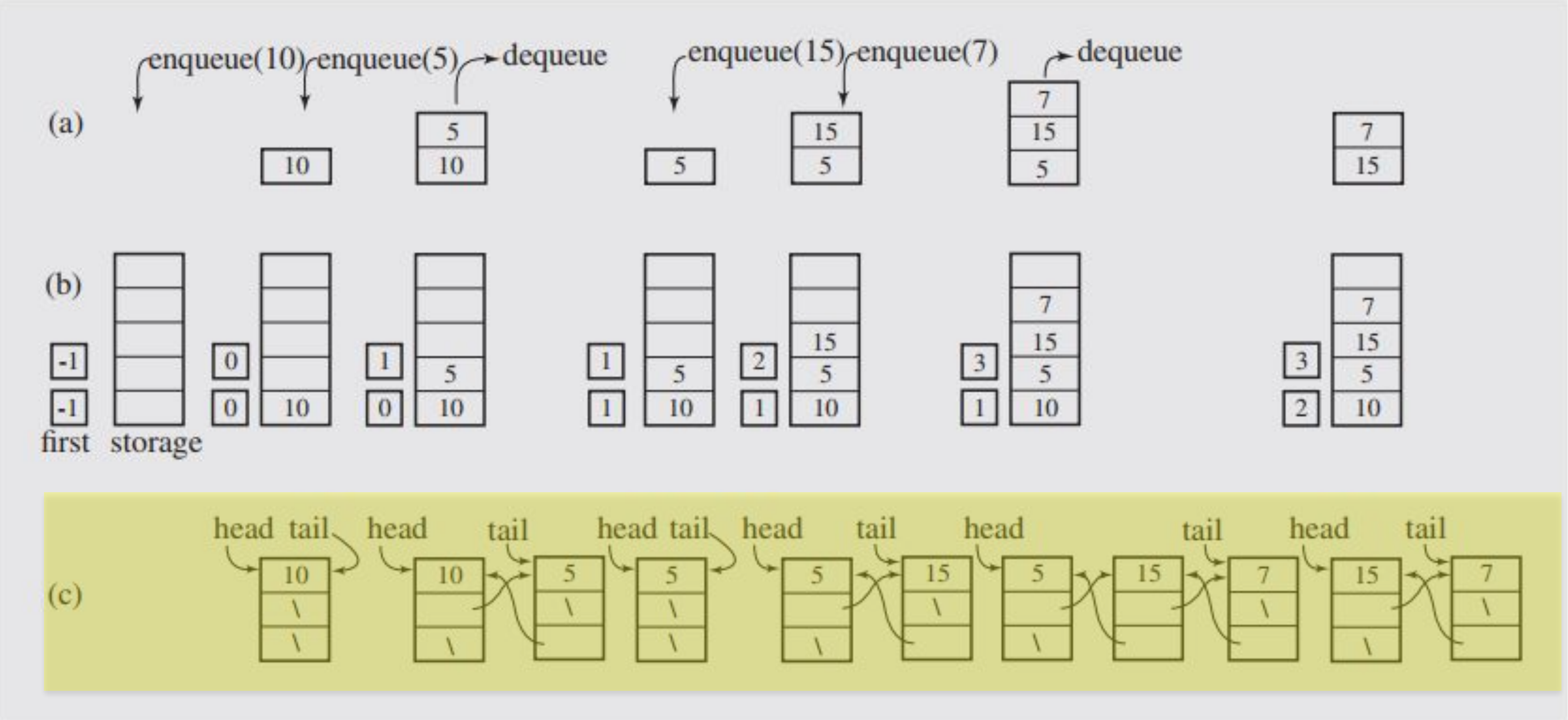


Ref. Drozdek, 2013: Data Structures and Algorithms in C++



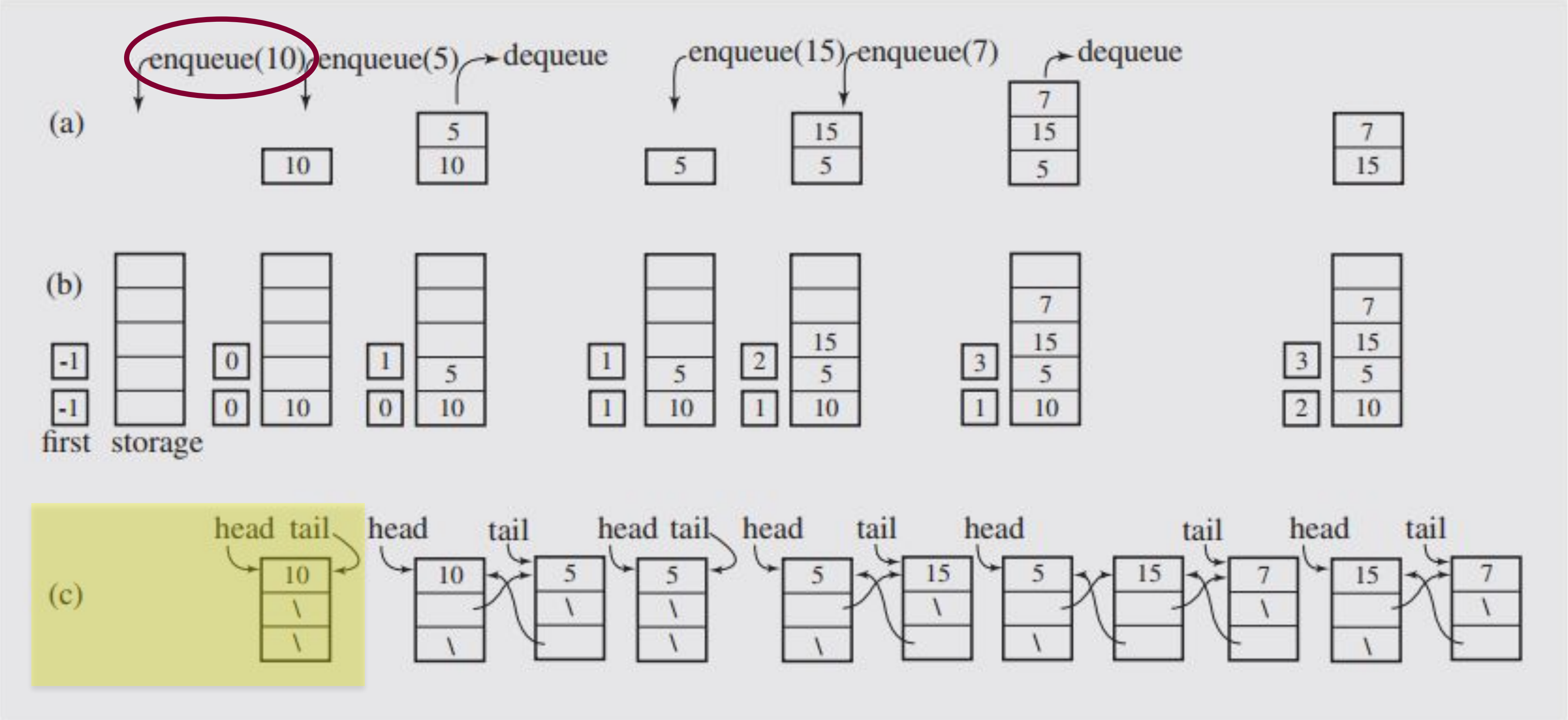
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



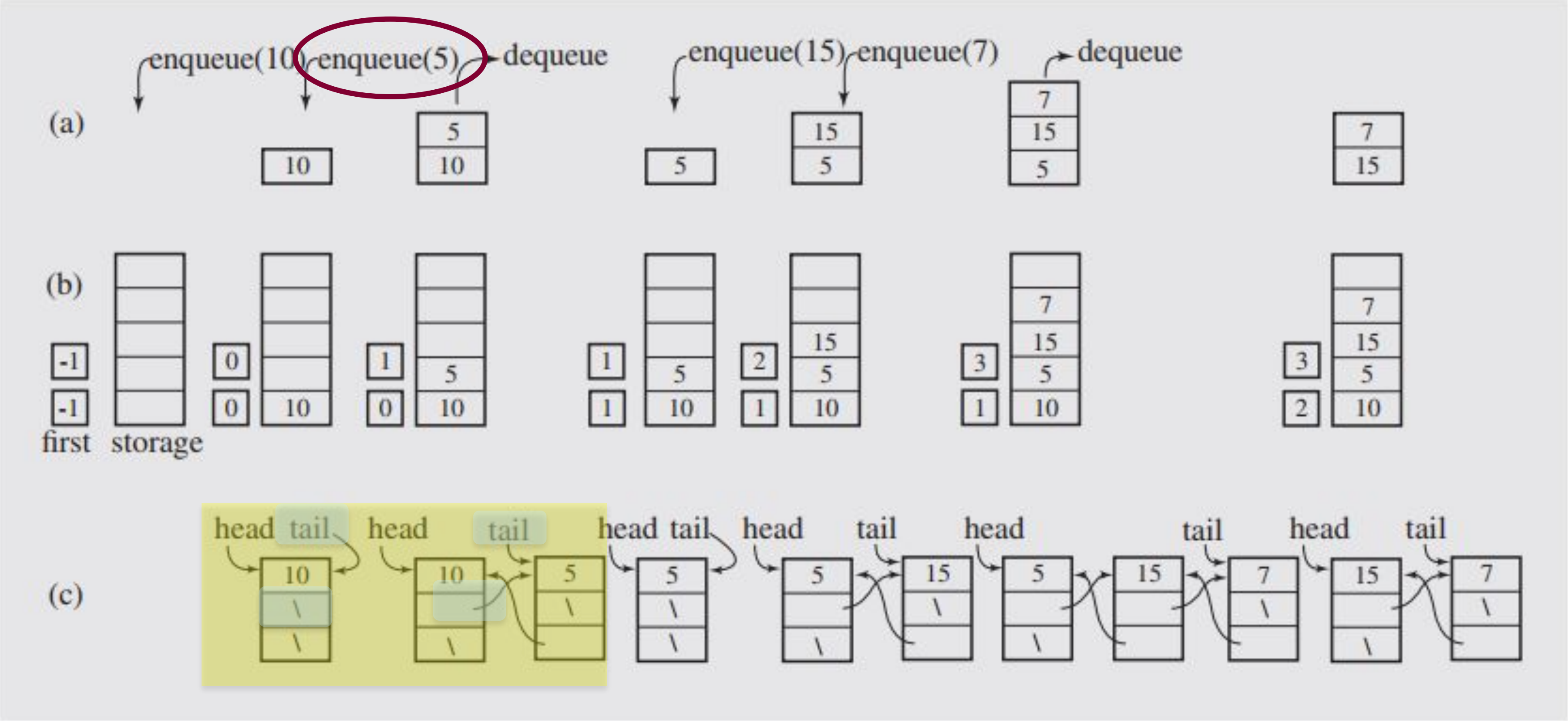
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



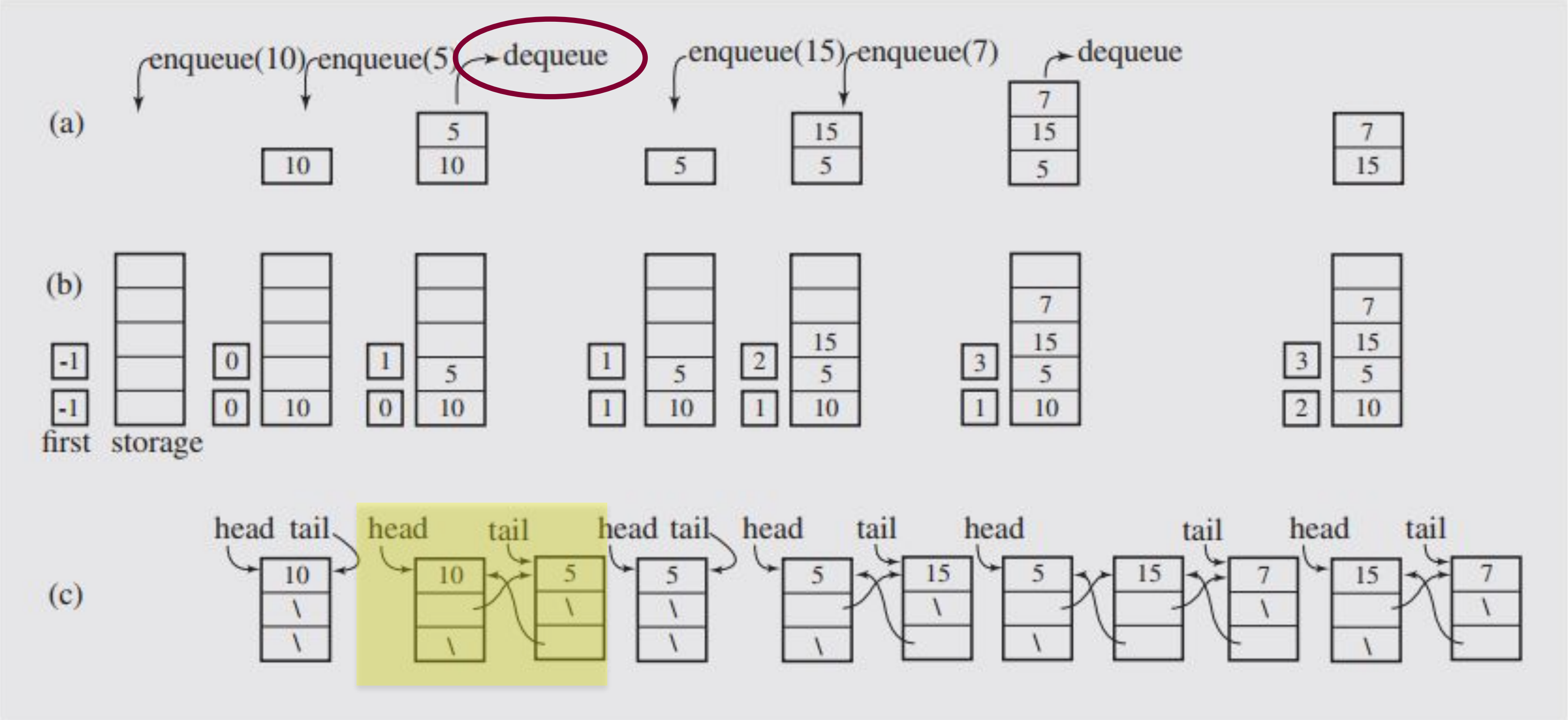
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



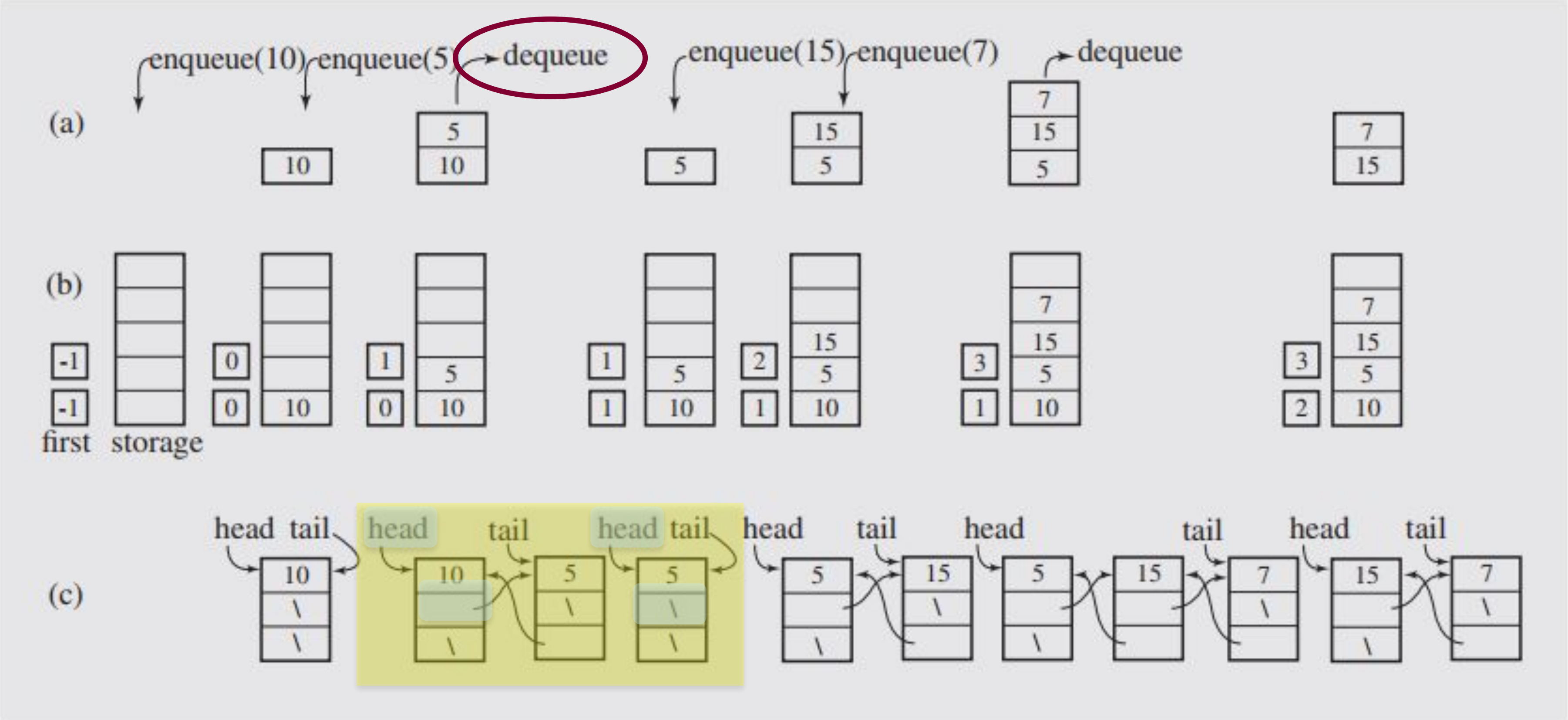
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



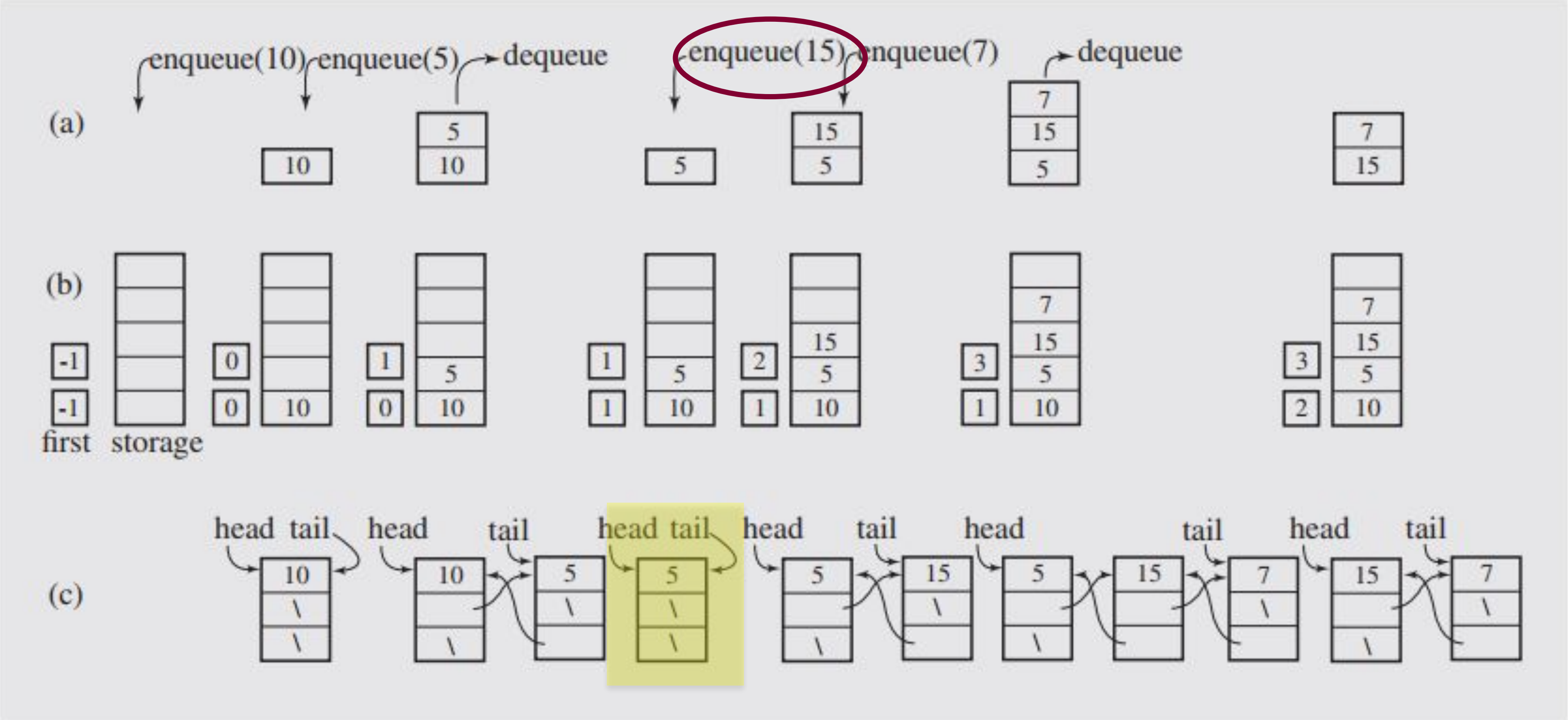
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



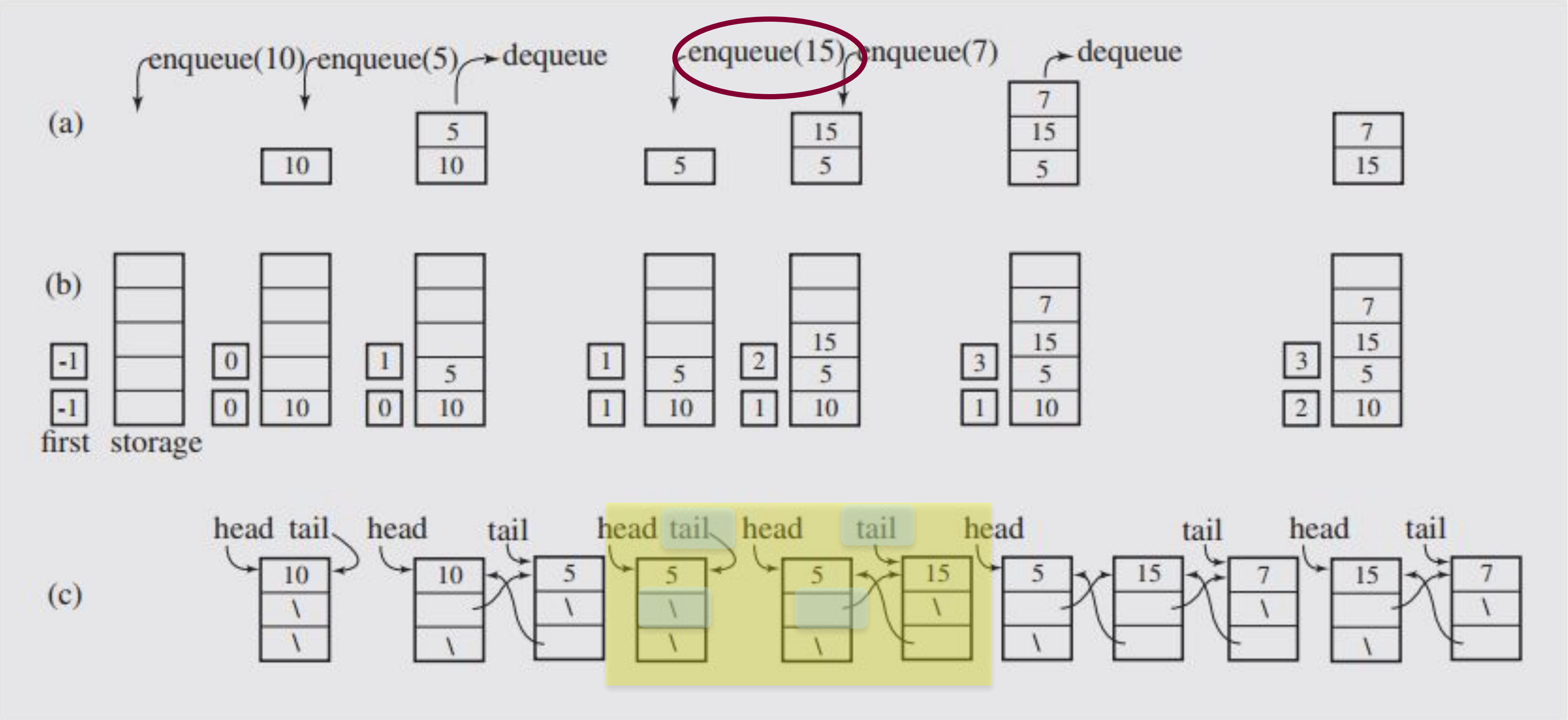
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



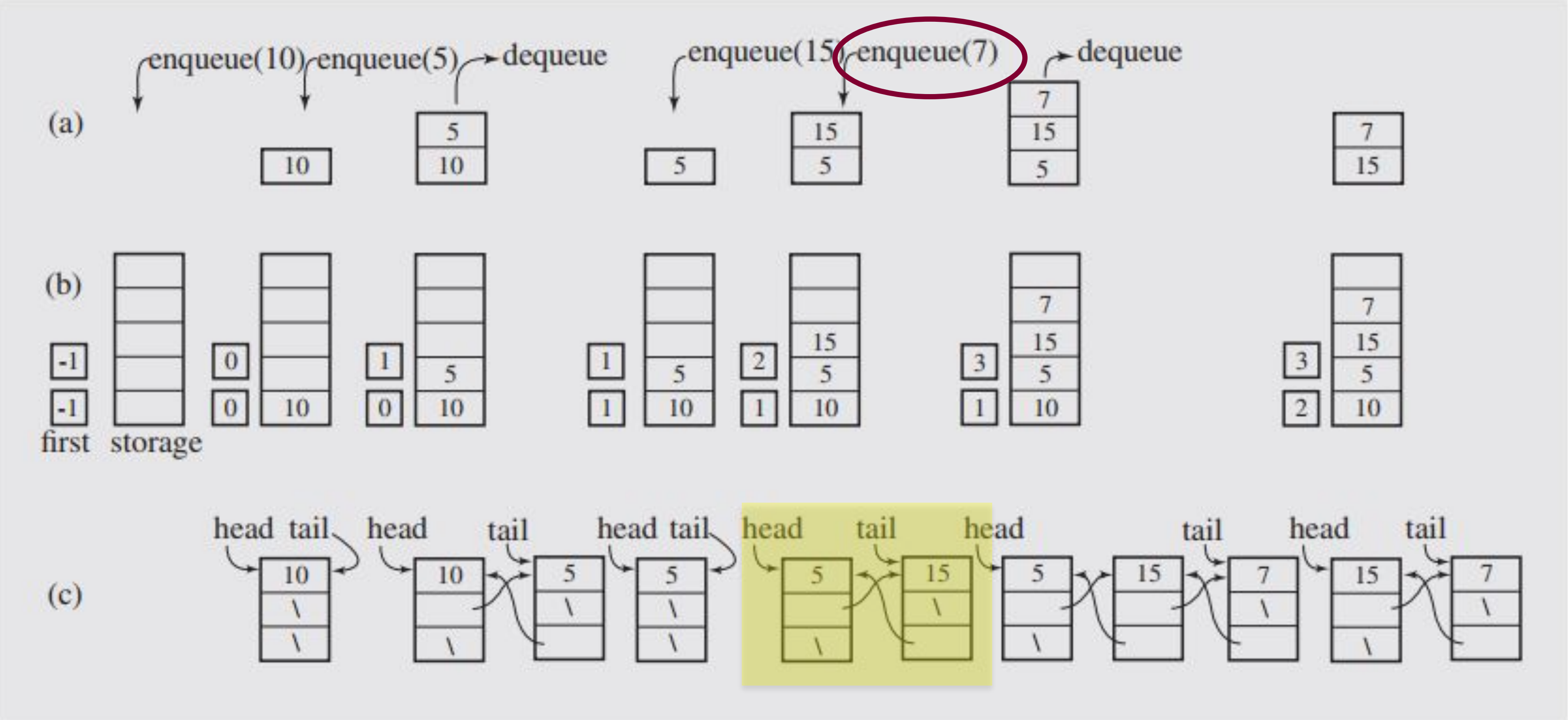
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



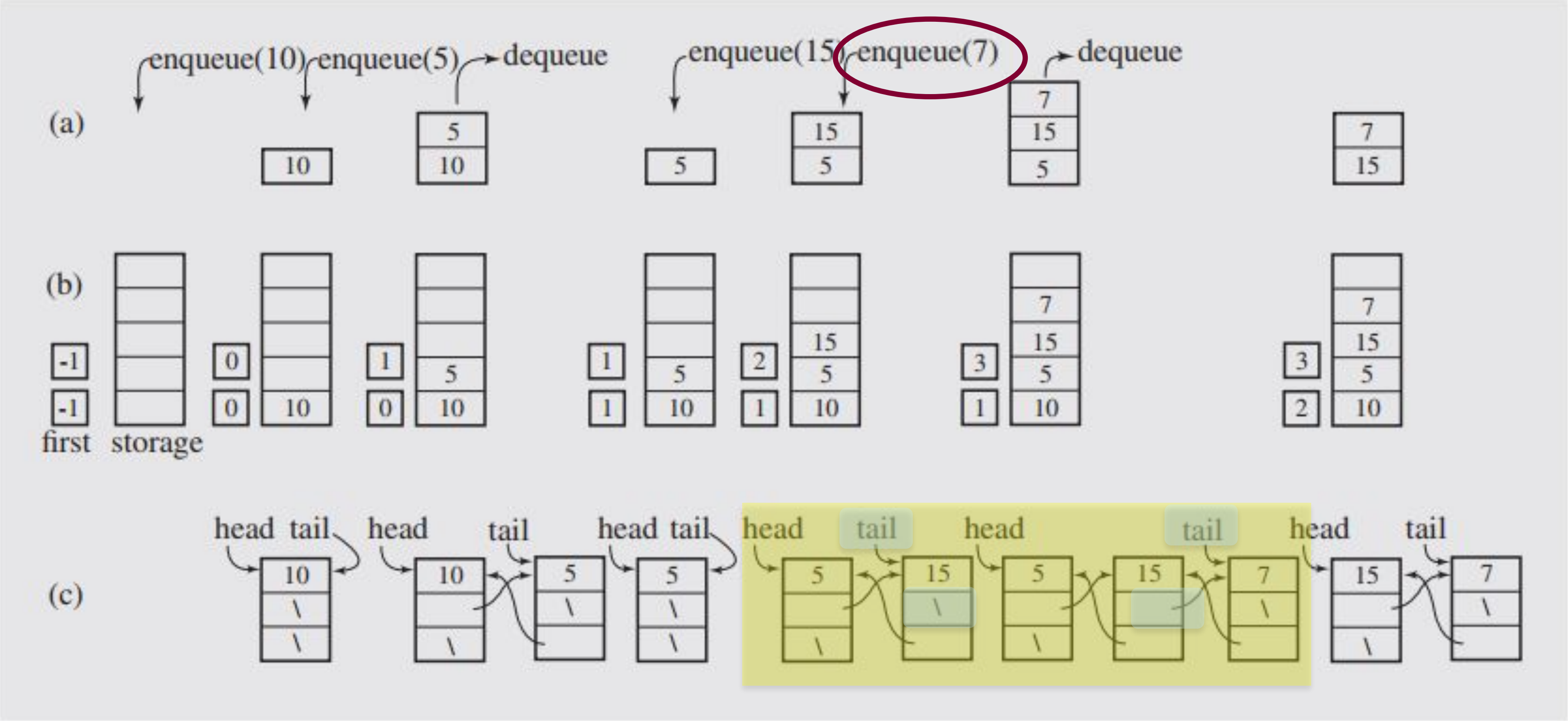
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



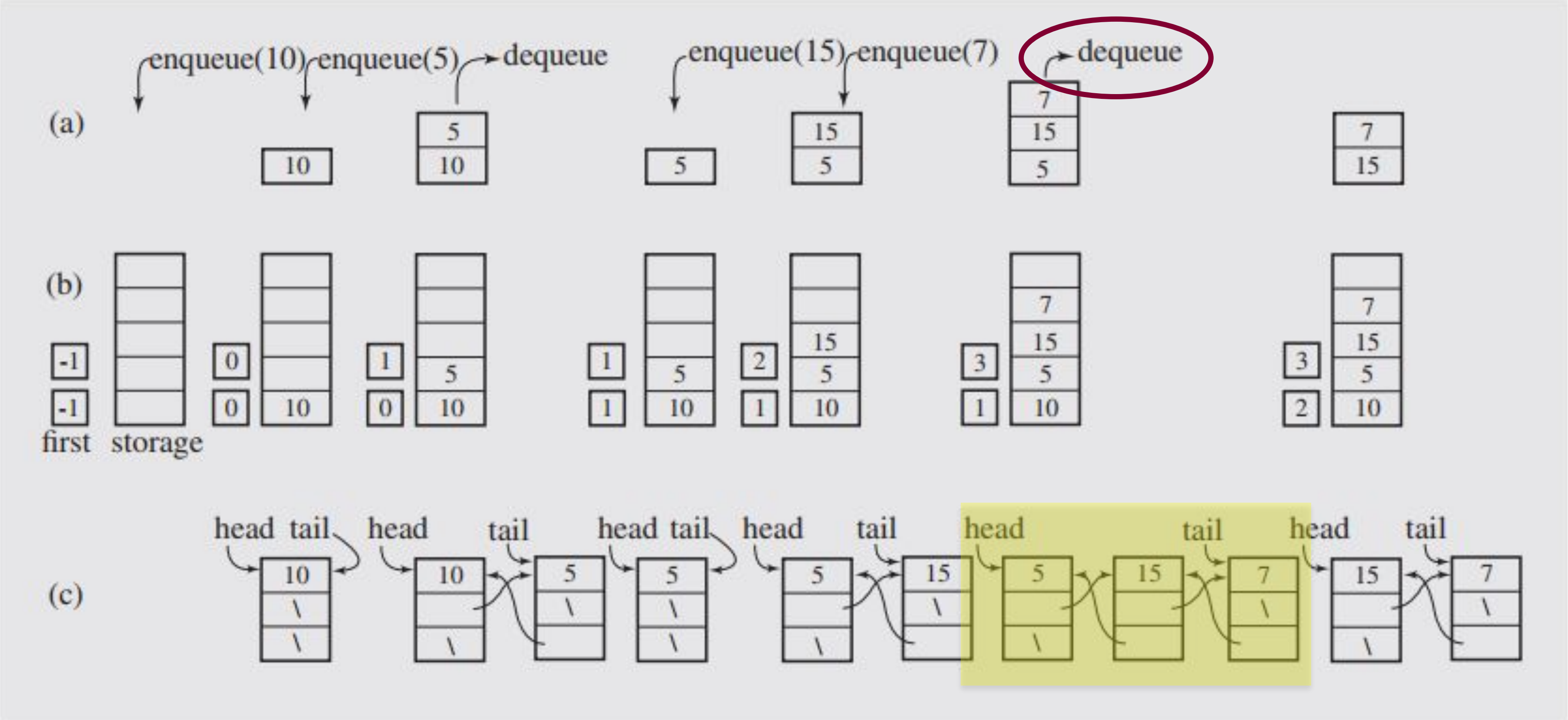
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



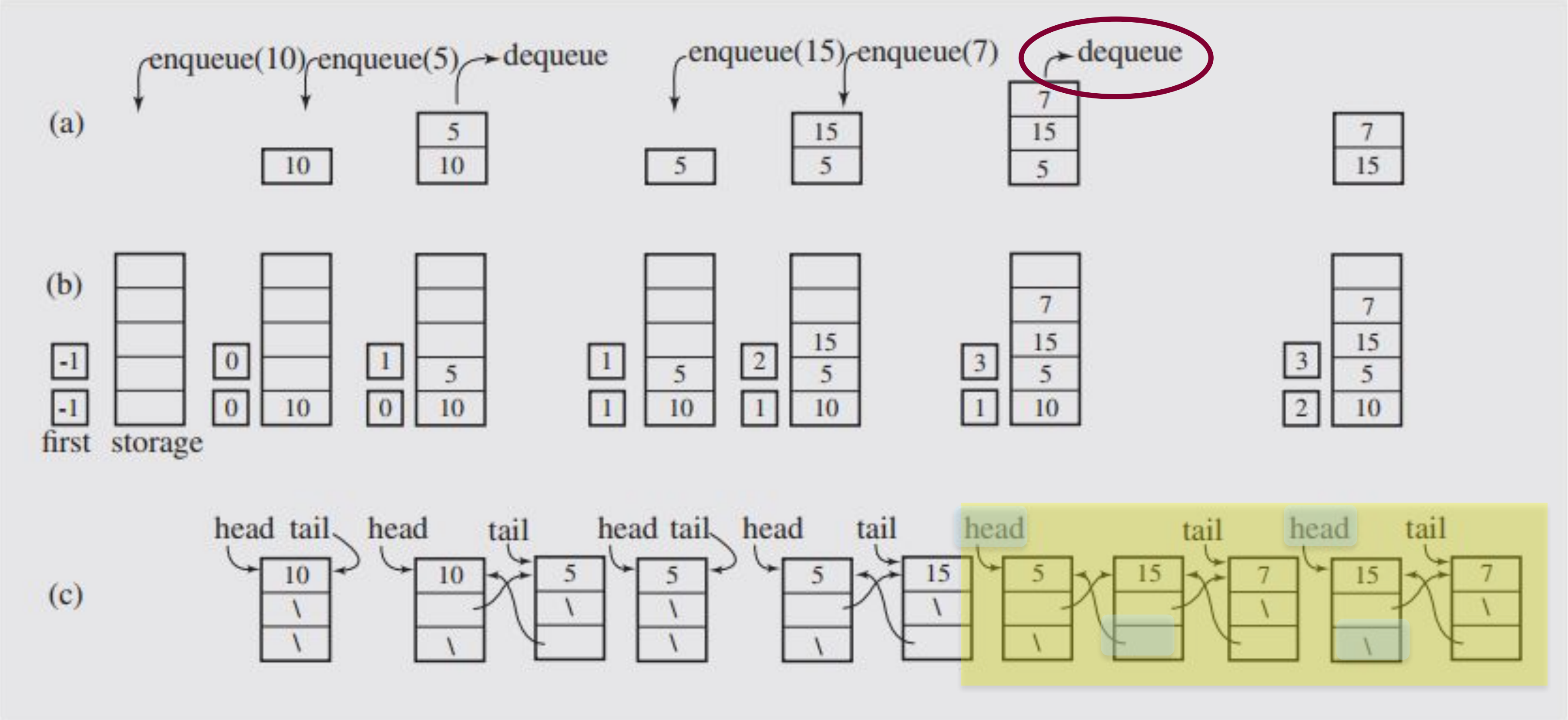
Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



Ref. Drozdek, 2013: Data Structures and Algorithms in C++

FIGURE 4.11 A series of operations executed on (a) an abstract queue and the queue implemented (b) with an array and (c) with a linked list.



Ref. Drozdek, 2013: Data Structures and Algorithms in C++

Queue Applications

- Printer's jobs
  - When jobs are submitted to a printer, they are arranged in order of arrival. Thus, essentially, jobs sent to a printer are placed on a queue.
- Real-life line
  - For instance, lines at ticket counters are queues, because service is first-come first-served.
- File server
  - Users on other machines are given access to files on a first-come first-served basis.

Ref. Weiss, 2014: Data Structures and Algorithm Analysis in C++

Queue Implementations

- Simple circular array-based implementation
- Linked list implementation

Ref. Karumanchi, 2017: Data Structures And Algorithms Made Easy

Queue – Linked List Implementation

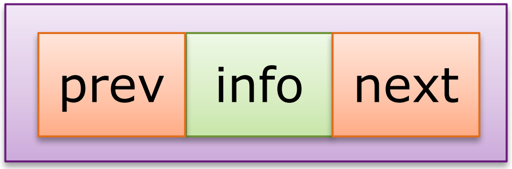
- Doubly Linked List with only
  - Insert Last □ add, enqueue
  - Delete first □ del, dequeue
- L.first replaced by Q.head
- L.last replaced by Q.tail

ADT Queue Element

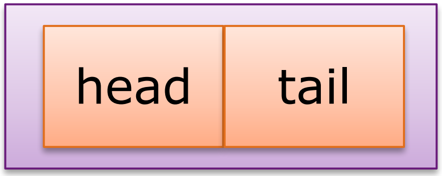
```
type Infotype : integer
type Address : pointer to ElmQueue

type ElmQueue <
   info : Infotype
   next : Address
   prev : Address >

type Queue: <
    head : Address
    tail : Address >
```



ElmQueue



Queue

Queue Operations (Primitives)

- Put the element el at the end of the queue.
- Take the first element from the queue

  procedure dequeue(in/out Q: Queue, P: Address)
-

  function isEmpty(Q : Queue) □ **Boolean**
-

  function front(Q: Queue) □ Address
- Return the number of element in the queue.

  function size(Q: Queue) □ integer

```
type Infotype : integer
type Address : pointer to ElmQueue

type ElmQueue <
  info : Infotype
  next : Address
  prev : Address >

type Queue: <
   head : Address
   tail : Address >
```

Implementation: `createQueue`

```
type Infotype : integer
type Address : pointer to ElmQueue

type ElmQueue <
  info : Infotype
  next : Address
  prev : Address >

type Queue: <
   head : Address
   tail : Address >
```

Implementation: isEmpty

[illegible]

```
type Infotype : integer
type Address : pointer to ElmQueue

type ElmQueue <
  info : Infotype
  next : Address
  prev : Address >

type Queue: <
   head : Address
   tail : Address >
```

Implementation: enqueue

```
type Infotype : integer
type Address : pointer to ElmQueue

type ElmQueue <
  info : Infotype
  next : Address
  prev : Address >

type Queue: <
   head : Address
   tail : Address >
```

Implementation: dequeue

```
type Infotype : integer
type Address : pointer to ElmQueue

type ElmQueue <
  info : Infotype
  next : Address
  prev : Address >

type Queue: <
   head : Address
   tail : Address >
```

Implementation: `front`

Implementation: `size`

```
type Infotype : integer
type Address : pointer to ElmQueue

type ElmQueue <
  info : Infotype
  next : Address
  prev : Address >

type Queue: <
   head : Address
   tail : Address >
```

Queue Operations Illustration

	0x1	0x2	0x3	0x4	0x5
info					
next					
prev					

head	NIL		tail	NIL

isEmpty	True

Queue Operations Illustration

```
enqueue( Q, 5 )
```

	0x1	0x2	0x3	0x4	0x5
info					
next					
prev					

head	NIL

tail	NIL

isEmpty	True

Queue Operations Illustration

```
enqueue( Q, 5 )
```

	0x1	0x2	0x3	0x4	0x5
info	5				
next	NIL				
prev	NIL				

head	0x1

tail	0x1

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
```

	0x1	0x2	0x3	0x4	0x5
info	5				
next	NIL				
prev	NIL				

head	0x1

tail	0x1

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
```

	0x1	0x2	0x3	0x4	0x5
info	5				
next	NIL				
prev	NIL				

head	0x1

tail	0x1

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
```

	0x1	0x2	0x3	0x4	0x5
info	5				
next	NIL				
prev	NIL				

head	0x1

tail	0x1

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
```

	0x1	0x2	0x3	0x4	0x5
info	5	2			
next	0x2	NIL			
prev	NIL	0x1			

head	0x1

tail	0x2

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
```

	0x1	0x2	0x3	0x4	0x5
info	5	2			
next	0x2	NIL			
prev	NIL	0x1			

head	0x1		tail	0x2

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
```

	0x1	0x2	0x3	0x4	0x5
info	5	2			
next	0x2	NIL			
prev	NIL	0x1			

head	0x1

tail	0x2

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
```

	0x1	0x2	0x3	0x4	0x5
info	5	2			
next	0x2	NIL			
prev	NIL	0x1			

head	0x1

tail	0x2

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
```

	0x1	0x2	0x3	0x4	0x5
info	5	2			
next	0x2	NIL			
prev	NIL	0x1			

head	0x1		tail	0x2

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
```

	0x1	0x2	0x3	0x4	0x5
info	5	2	7		
next	0x2	0x3	NIL		
prev	NIL	0x1	0x2		

head	0x1

tail	0x3

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
```

	0x1	0x2	0x3	0x4	0x5
info	5	2	7		
next	0x2	0x3	NIL		
prev	NIL	0x1	0x2		

head	0x1		tail	0x3

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
```

	0x1	0x2	0x3	0x4	0x5
info	5	2	7		
next	0x2	0x3	NIL		
prev	NIL	0x1	0x2		

head	0x1	tail	0x3

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
```

	0x1	0x2	0x3	0x4	0x5
info	5	2	7		
next	0x2	0x3	NIL		
prev	NIL	0x1	0x2		

head	0x1

tail	0x3

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
dequeue( Q )
```

	0x1	0x2	0x3	0x4	0x5
info	5	2	7		
next	0x2	0x3	NIL		
prev	NIL	0x1	0x2		

head	0x1

tail	0x3

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
dequeue( Q )
```

	0x1	0x2	0x3	0x4	0x5
info	5	2	7		
next	NIL	0x3	NIL		
prev	NIL	NIL	0x2		

head	0x2	tail	0x3

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
dequeue( Q )
```

	0x1	0x2	0x3	0x4	0x5
info	5	2	7		
next	NIL	0x3	NIL		
prev	NIL	NIL	0x2		

head	0x2

tail	0x3

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
dequeue( Q )
dequeue( Q )
```

	0x1	0x2	0x3	0x4	0x5
info	5	2	7		
next	NIL	0x3	NIL		
prev	NIL	NIL	0x2		

head	0x2

tail	0x3

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
dequeue( Q )
dequeue( Q )
```

	0x1	0x2	0x3	0x4	0x5
info	5	2	7		
next	NIL	NIL	NIL		
prev	NIL	NIL	NIL		

head	0x3

tail	0x3

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
dequeue( Q )
dequeue( Q )
enqueue( Q, 4 )
```

	0x1	0x2	0x3	0x4	0x5
info	5	2	7		
next	NIL	NIL	NIL		
prev	NIL	NIL	NIL		

head	0x3	tail	0x3

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
dequeue( Q )
dequeue( Q )
enqueue( Q, 4 )
```

	0x1	0x2	0x3	0x4	0x5
info	4	2	7		
next	NIL	NIL	0x1		
prev	0x3	NIL	NIL		

head	0x3

tail	0x1

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
dequeue( Q )
dequeue( Q )
enqueue( Q, 4 )
dequeue( Q )
```

	0x1	0x2	0x3	0x4	0x5
info	4	2	7		
next	NIL	NIL	0x1		
prev	0x3	NIL	NIL		

head	0x3

tail	0x1

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
dequeue( Q )
dequeue( Q )
enqueue( Q, 4 )
dequeue( Q )
```

	0x1	0x2	0x3	0x4	0x5
info	4	2	7		
next	NIL	NIL	NIL		
prev	NIL	NIL	NIL		

head	0x1

tail	0x1

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
dequeue( Q )
dequeue( Q )
enqueue( Q, 4 )
dequeue( Q )
enqueue( Q, 9 )
```

	0x1	0x2	0x3	0x4	0x5
info	4	2	7		
next	NIL	NIL	NIL		
prev	NIL	NIL	NIL		

head	0x1

tail	0x1

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
dequeue( Q )
dequeue( Q )
enqueue( Q, 4 )
dequeue( Q )
enqueue( Q, 9 )
```

	0x1	0x2	0x3	0x4	0x5
info	4	9	7		
next	0x2	NIL	NIL		
prev	NIL	0x1	NIL		

head	0x1

tail	0x2

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
dequeue( Q )
dequeue( Q )
enqueue( Q, 4 )
dequeue( Q )
enqueue( Q, 9 )
enqueue( Q, 4 )
```

	0x1	0x2	0x3	0x4	0x5
info	4	9	7		
next	0x2	NIL	NIL		
prev	NIL	0x1	NIL		

head	0x1	tail	0x2

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
dequeue( Q )
dequeue( Q )
enqueue( Q, 4 )
dequeue( Q )
enqueue( Q, 9 )
enqueue( Q, 4 )
dequeue( Q )
```

	0x1	0x2	0x3	0x4	0x5
info	4	9	4		
next	0x2	0x3	NIL		
prev	NIL	0x1	0x2		

head	0x1	tail	0x3

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
dequeue( Q )
dequeue( Q )
enqueue( Q, 4 )
dequeue( Q )
enqueue( Q, 9 )
enqueue( Q, 4 )
dequeue( Q )
```

	0x1	0x2	0x3	0x4	0x5
info	4	9	4		
next	NIL	0x3	NIL		
prev	NIL	NIL	0x2		

head	0x2

tail	0x3

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
dequeue( Q )
dequeue( Q )
enqueue( Q, 4 )
dequeue( Q )
enqueue( Q, 9 )
enqueue( Q, 4 )
dequeue( Q )
dequeue( Q )
```

	0x1	0x2	0x3	0x4	0x5
info	4	9	4		
next	NIL	0x3	NIL		
prev	NIL	NIL	0x2		

head	0x2	tail	0x3

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
dequeue( Q )
dequeue( Q )
enqueue( Q, 4 )
dequeue( Q )
enqueue( Q, 9 )
enqueue( Q, 4 )
dequeue( Q )
dequeue( Q )
```

	0x1	0x2	0x3	0x4	0x5
info	4	9	4		
next	NIL	NIL	NIL		
prev	NIL	NIL	NIL		

head	0x3

tail	0x3

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
dequeue( Q )
dequeue( Q )
enqueue( Q, 4 )
dequeue( Q )
enqueue( Q, 9 )
enqueue( Q, 4 )
dequeue( Q )
dequeue( Q )
```

	0x1	0x2	0x3	0x4	0x5
info	4	9	4		
next	NIL	NIL	NIL		
prev	NIL	NIL	NIL		

head	0x3		tail	0x3

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
dequeue( Q )
dequeue( Q )
enqueue( Q, 4 )
dequeue( Q )
enqueue( Q, 9 )
enqueue( Q, 4 )
dequeue( Q )
dequeue( Q )
dequeue( Q )
```

	0x1	0x2	0x3	0x4	0x5
info	4	9	4		
next	NIL	NIL	NIL		
prev	NIL	NIL	NIL		

head	0x3

tail	0x3

isEmpty	False

Queue Operations Illustration

```
enqueue( Q, 5 )
enqueue( Q, 2 )
enqueue( Q, 7 )
dequeue( Q )
dequeue( Q )
enqueue( Q, 4 )
dequeue( Q )
enqueue( Q, 9 )
enqueue( Q, 4 )
dequeue( Q )
dequeue( Q )
dequeue( Q )
```

	0x1	0x2	0x3	0x4	0x5
info	4	9	4		
next	NIL	NIL	NIL		
prev	NIL	NIL	NIL		

head	NIL

tail	NIL

isEmpty	True

Question?

TRAIN YOUR BRAIN

Train your Brain!

To access the queue/stack, we are only allowed to use the primitive methods of queue/stack ADT.

- **Problem-1**: Give an algorithm for reversing a queue.
- **Problem-2**: Implement a queue using two stacks.
- **Problem-3**: Implement one stack using two queues.
- **Problem-4**: Given a queue Q containing n elements, transfer these items on to a stack S (initially empty) so that front element of Q appears at the top of the stack and the order of all other items is preserved.
- **Problem-5**: Given an integer k and a queue of integers, how do you reverse the order of the first k elements of the queue, leaving the other elements in the same relative order?
  - For example, if k=4 and queue has the elements [10, 20, 30, 40, 50, 60, 70, 80, 90]; the output should be [40, 30, 20, 10, 50, 60, 70, 80, 90].

Ref. Karumanchi, 2017: Data Structures And Algorithms Made Easy

Train your Brain! (cont.)

- **Problem-6**: Given a queue of integers, rearrange the elements by interleaving the first half of the list with the second half of the list.
    - For example, suppose a queue stores the following sequence of values: [11, 12, 13, 14, 15, 16, 17, 18, 19, 20].
    - Consider the two halves of this list: first half: [11, 12, 13, 14, 15] second half: [16, 17, 18, 19, 20].
    - These are combined in an alternating fashion to form a sequence of interleave pairs:
        - the first values from each half (11 and 16), then the second values from each half (12 and 17),
        - then the third values from each half (13 and 18), and so on.
    - In each pair, the value from the first half appears before the value from the second half.
    - Thus, after the call, the queue stores the following values: [11, 16, 12, 17, 13, 18, 14, 19, 15, 20]

Ref. Karumanchi, 2017: Data Structures And Algorithms Made Easy

Referensi

[1] Karumanchi, N. (2017). Data Structures And Algorithms Made Easy (5th ed.). CareerMonk Pub.

[2] Bhargava, A. Y. (2016). Grokking Algorithms. Manning Pub. Co.

[3] Weiss, M. A. (2014). Data Structures and Algorithm Analysis in C++ (4th ed.). Addison-Wesley Pub.

[4] Drozdek, A. (2013). Data Structures and Algorithms in C++ (4th ed.). Cengage Learning.

[5] Gilberg, R. F. & Forouzan, B. A. (2005). Data Structures- A Pseudocode Approach with C (2nd ed.). Thomson Learning, Inc.

[6] Lafore, R. (2003). Data Structures & Algorithms in Java (2nd ed.). Sams Pub.

THANK YOU