Outline

Definition

Stack: Abstract data type with the following operations:

Definition

Stack: Abstract data type with the following operations:

- `Push(Key)`: adds key to collection

Definition

Stack: Abstract data type with the following operations:

- `Push(Key)`: adds key to collection
- `Key Top()`: returns most recently-added key

Definition

Stack: Abstract data type with the following operations:

- `Push(Key)`: adds key to collection
- `Key Top()`: returns most recently-added key
- `Key Pop()`: removes and returns most recently-added key

Definition

Stack: Abstract data type with the following operations:

- `Push(Key)`: adds key to collection
- `Key Top()`: returns most recently-added key
- `Key Pop()`: removes and returns most recently-added key
- `Boolean Empty()`: are there any elements?

Balanced Brackets

Input: A string *str* consisting of '(', ')', '[', ']' characters.

Output: Return whether or not the string's parentheses and square brackets are balanced.

Balanced Brackets

Balanced:

- ''([])[]()'',
- ''((([([])]))())''

Unbalanced:

- ''([]]()''
- ''][''
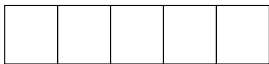
IsBalanced(*str*)

```
Stack stack
for char in str:
  if char in ['(', '[']:
    stack.Push(char)
  else:
    if stack.Empty():  return False
    top ← stack.Pop()
    if (top = '[' and char != ']') or
      (top = '(' and char != ')'):
      return False
return stack.Empty()
```

Stack Implementation with Array

numElements: 0

Stack Implementation with Array
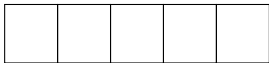
numElements: 0

Push(a)

Stack Implementation with Array

numElements: 1

a				

Push(a)

Stack Implementation with Array

numElements: 1

a				

Stack Implementation with Array

numElements: 1

a				
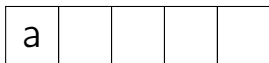
Push(b)

Stack Implementation with Array

numElements: 2

a	b			

Push(b)

Stack Implementation with Array

numElements: 2

a	b			

Stack Implementation with Array

numElements: 2

a	b			

`Top()`

Stack Implementation with Array

numElements: 2

a	b			

`Top()`→ `b`

Stack Implementation with Array

numElements: 2

a	b			

Stack Implementation with Array

numElements: 2

a	b			

Push(c)

Stack Implementation with Array

numElements: 3

a	b	c		

`Push(c)`

Stack Implementation with Array

numElements: 3

a	b	c		

Stack Implementation with Array

numElements: 3

a	b	c		

Pop()

Stack Implementation with Array

numElements: 2

a	b			

Pop()→ c

Stack Implementation with Array

numElements: 2

a	b			

Stack Implementation with Array

numElements: 2

a	b			

Push(d)

Stack Implementation with Array

numElements: 3

| a | b | d | | |

Push(d)

Stack Implementation with Array

numElements: 3

a	b	d		

Stack Implementation with Array

numElements: 3

a	b	d		

Push(e)

Stack Implementation with Array

numElements: 4

a	b	d	e	

`Push(e)`

Stack Implementation with Array

numElements: 4

a	b	d	e	

Stack Implementation with Array

numElements: 4

a	b	d	e	

Push(f)

Stack Implementation with Array

numElements: 5

a	b	d	e	f

Push(f)

Stack Implementation with Array

numElements: 5

| a | b | d | e | f |

Stack Implementation with Array

numElements: 5

a	b	d	e	f

Push(g)

Stack Implementation with Array

numElements: 5

a	b	d	e	f

$\text{Push}(g) \rightarrow$ ERROR

Stack Implementation with Array

numElements: 5

| a | b | d | e | f |

Stack Implementation with Array

numElements: 5

a	b	d	e	f

```
Empty()
```

Stack Implementation with Array

numElements: 5

a	b	d	e	f

Empty() → False

Stack Implementation with Array

numElements: 5

a	b	d	e	f

Stack Implementation with Array

numElements: 5

| a | b | d | e | f |

Pop()

Stack Implementation with Array

numElements: 4

a	b	d	e	

`Pop()`$\to$ `f`

Stack Implementation with Array

numElements: 4

a	b	d	e	

Stack Implementation with Array

numElements: 4

a	b	d	e	

`Pop()`

Stack Implementation with Array

numElements: 3

a	b	d		

Pop()→ e

Stack Implementation with Array

numElements: 3

a	b	d		

Stack Implementation with Array

numElements: 3

a	b	d		

```
Pop()
```

Stack Implementation with Array

numElements: 2

a	b			

Pop()$\to$ d

Stack Implementation with Array

numElements: 2

a	b			

Stack Implementation with Array

numElements: 2

a	b			

Pop()

Stack Implementation with Array
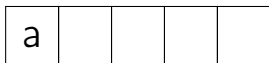
numElements: 1

a				

Pop()$\to$ b

Stack Implementation with Array

numElements: 1

a				

Stack Implementation with Array

numElements: 1

a				

Pop()

Stack Implementation with Array
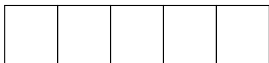
numElements: 0

`Pop()→ a`

Stack Implementation with Array
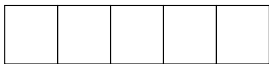
numElements: 0

Stack Implementation with Array
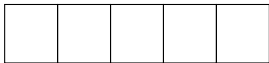
numElements: 0

Empty()

Stack Implementation with Array
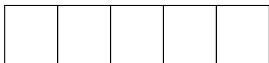
numElements: 0

Empty() → True

Stack Implementation with Array

numElements: 0

Stack Implementation with Linked List

head

Stack Implementation with Linked List

head

`Push(a)`

Stack Implementation with Linked List

head

a

Push(a)

Stack Implementation with Linked List

head

Stack Implementation with Linked List

head

Push(b)

Stack Implementation with Linked List

head

b

a

Push(b)

Stack Implementation with Linked List

Stack Implementation with Linked List

head

b

a

`Top()`

Stack Implementation with Linked List

head

b   a
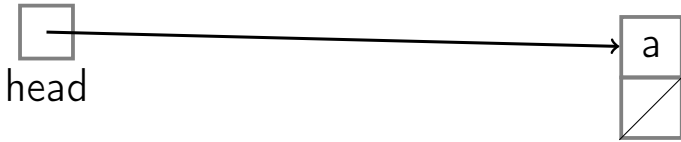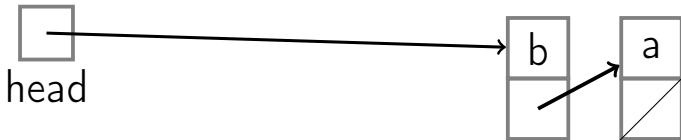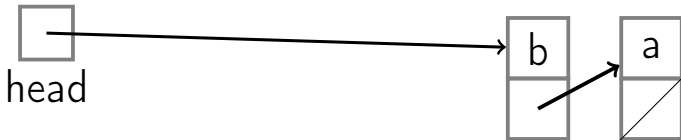
Top()→ b

Stack Implementation with Linked List
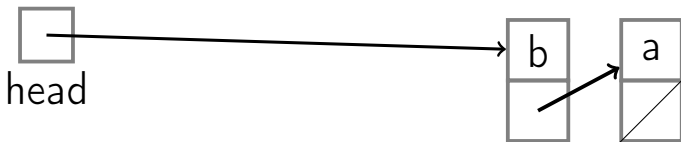
Stack Implementation with Linked List

Push(c)

Stack Implementation with Linked List
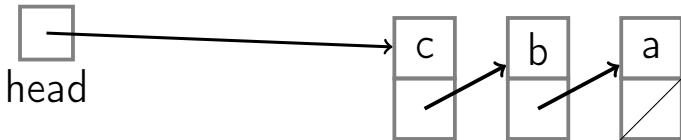
head

Push(c)

Stack Implementation with Linked List

Stack Implementation with Linked List

head

c b a

Pop()

Stack Implementation with Linked List

head

b   a

Pop()→ c

Stack Implementation with Linked List

Stack Implementation with Linked List

head

b   a

Push(d)

Stack Implementation with Linked List

head

d   b   a

Push(d)

Stack Implementation with Linked List

Stack Implementation with Linked List

head

Push(e)

Stack Implementation with Linked List

Push(e)

Stack Implementation with Linked List
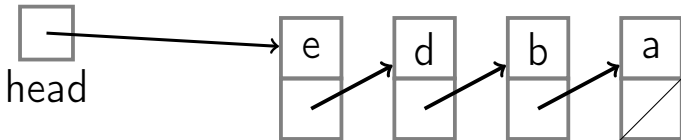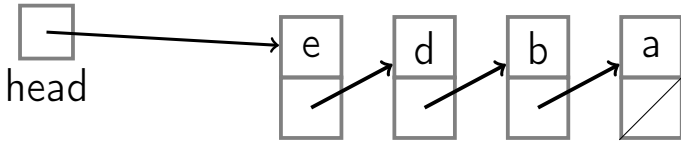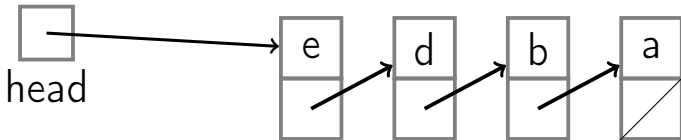
Stack Implementation with Linked List


```
Push(f)
```

Stack Implementation with Linked List

Push(f)

Stack Implementation with Linked List

Stack Implementation with Linked List

head

f e d b a

`Empty()`

Stack Implementation with Linked List

head

f   e   d   b   a
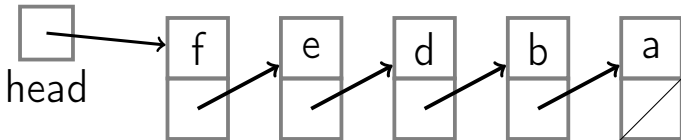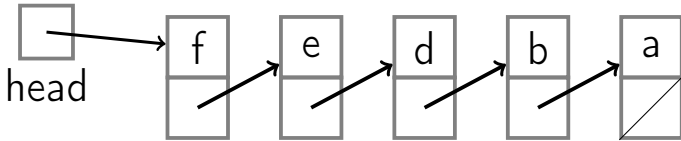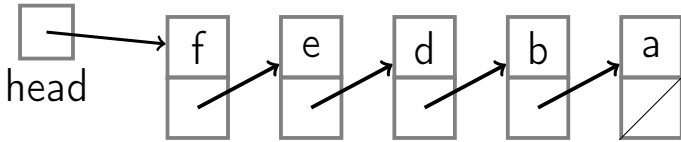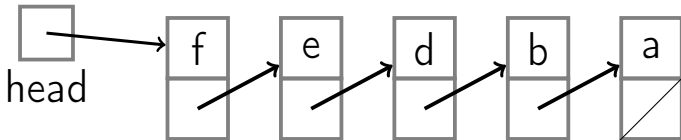
Empty() → False
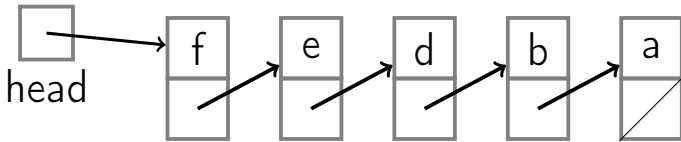
Stack Implementation with Linked List

Stack Implementation with Linked List

Pop()

Stack Implementation with Linked List

head

e   d   b   a

Pop()→ f

Stack Implementation with Linked List

Stack Implementation with Linked List

head

e   d   b   a

Pop()

Stack Implementation with Linked List

head

Pop() → e

Stack Implementation with Linked List

Stack Implementation with Linked List

head

d   b   a

Pop()

Stack Implementation with Linked List

head

b

a

$\mathtt{Pop()} \rightarrow \mathtt{d}$

Stack Implementation with Linked List

Stack Implementation with Linked List

head

b

a

Pop()

Stack Implementation with Linked List

head

a

Pop()$\to$ b

Stack Implementation with Linked List

Stack Implementation with Linked List

head

a

Pop()

Stack Implementation with Linked List

head

$\text{Pop()} \rightarrow \text{a}$

Stack Implementation with Linked List

head

Stack Implementation with Linked List

head

`Empty()`

Stack Implementation with Linked List

□

head

`Empty()` $\rightarrow$ `True`

Stack Implementation with Linked List

head

Stack Implementation with Linked List

Stack Implementation with Linked List

Stack Implementation with Linked List

Summary

- Stacks can be implemented with either an array or a linked list.

Summary

- Stacks can be implemented with either an array or a linked list.
- Each stack operation is $O(1)$: Push, Pop, Top, Empty.

Summary

- Stacks can be implemented with either an array or a linked list.
- Each stack operation is $O(1)$: Push, Pop, Top, Empty.
- Stacks are ocassionaly known as LIFO queues.

Outline

Definition

Queue: Abstract data type with the following operations:

Definition

Queue: Abstract data type with the following operations:

- `Enqueue(Key)`: adds key to collection

Definition

Queue: Abstract data type with the following operations:

- `Enqueue(Key)`: adds key to collection
- `Key Dequeue()`: removes and returns least recently-added key

Definition

Queue: Abstract data type with the following operations:

- `Enqueue(Key)`: adds key to collection
- `Key Dequeue()`: removes and returns least recently-added key
- `Boolean Empty()`: are there any elements?

Definition

Queue: Abstract data type with the following operations:

- `Enqueue(Key)`: adds key to collection
- `Key Dequeue()`: removes and returns least recently-added key
- `Boolean Empty()`: are there any elements?

FIFO: First-In, First-Out

Queue Implementation with Linked List

head

tail

Queue Implementation with Linked List

head

tail

Enqueue(a)

Queue Implementation with Linked List
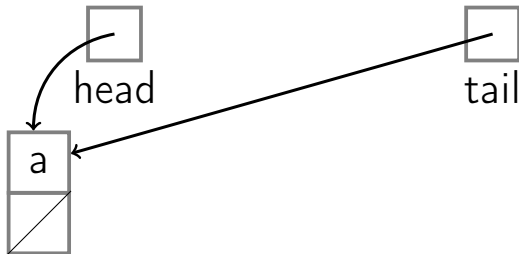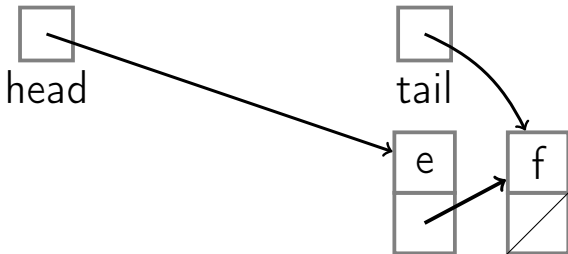

Enqueue(a)

Queue Implementation with Linked List

Queue Implementation with Linked List

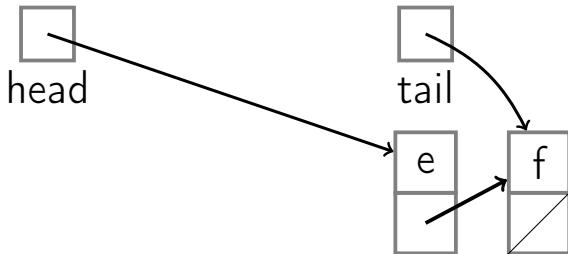head

tail

a
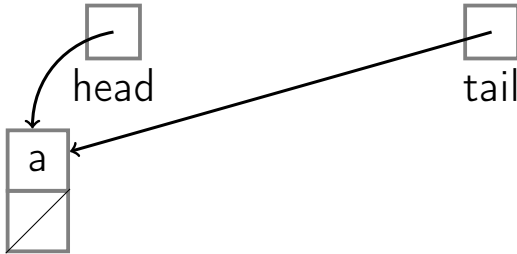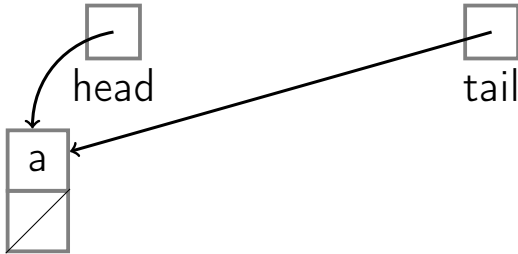
Enqueue(b)
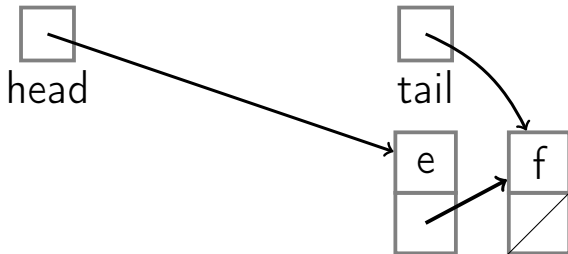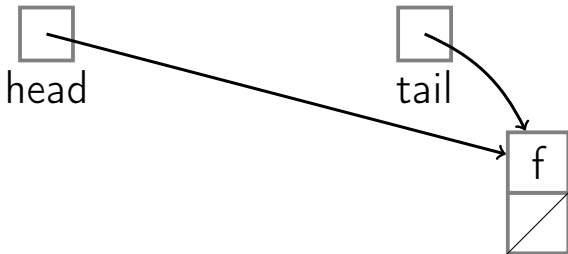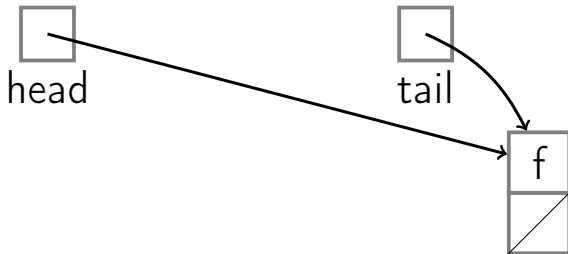
Queue Implementation with Linked List
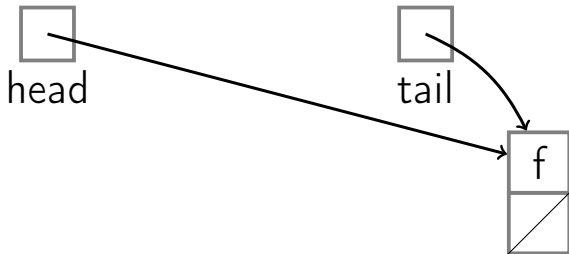

Enqueue(b)

Queue Implementation with Linked List
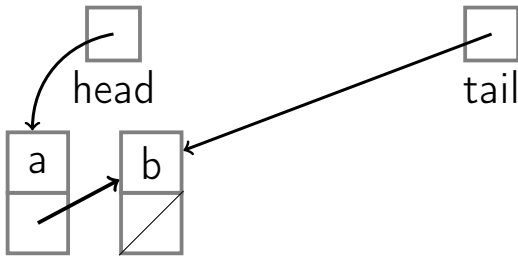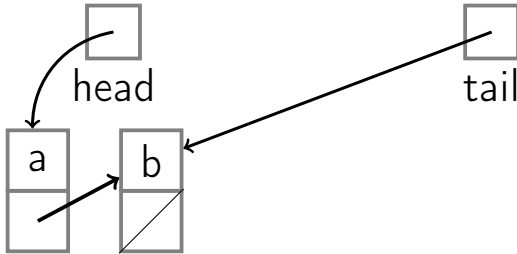
Queue Implementation with Linked List
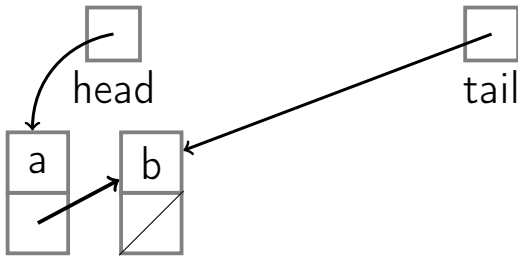
Empty()

Queue Implementation with Linked List
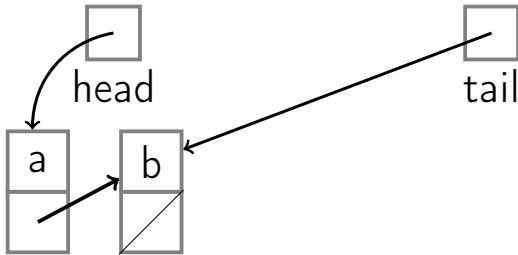
Empty()$\rightarrow$ False

Queue Implementation with Linked List
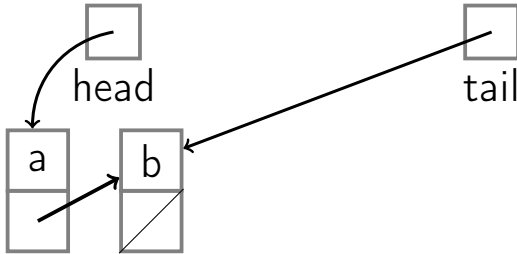
Queue Implementation with Linked List
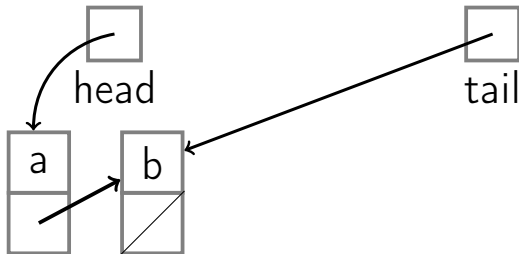
Enqueue(c)

Queue Implementation with Linked List
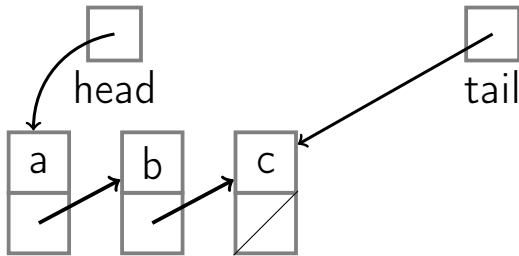
Enqueue(c)

Queue Implementation with Linked List
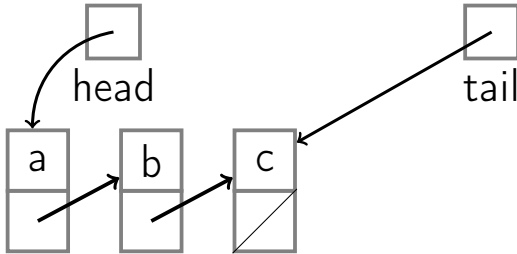
Queue Implementation with Linked List
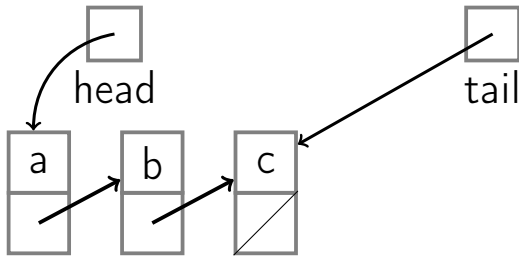
Dequeue()

Queue Implementation with Linked List

Dequeue() $\to$ a

Queue Implementation with Linked List
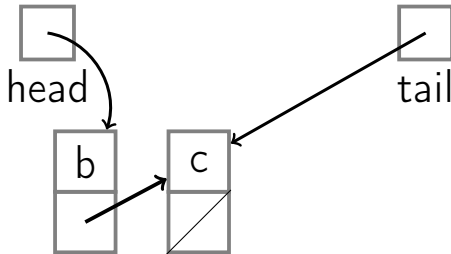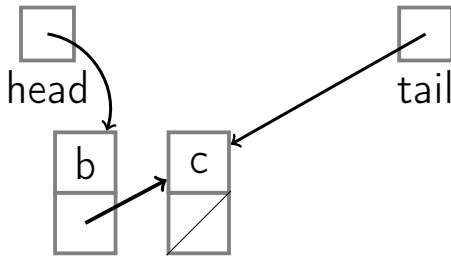
Queue Implementation with Linked List

Enqueue(d)

Queue Implementation with Linked List

Enqueue(d)

Queue Implementation with Linked List
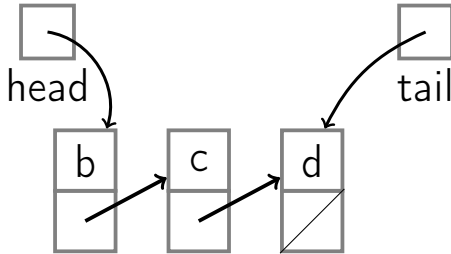
Queue Implementation with Linked List
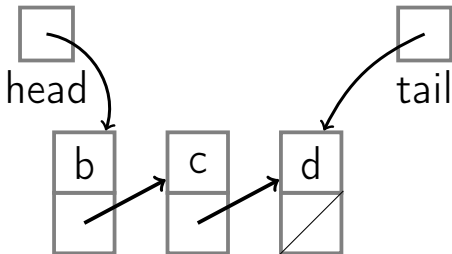
Enqueue(e)

Queue Implementation with Linked List
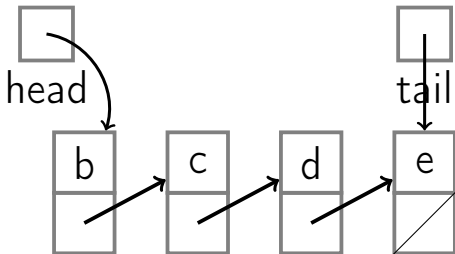
Enqueue(e)

Queue Implementation with Linked List
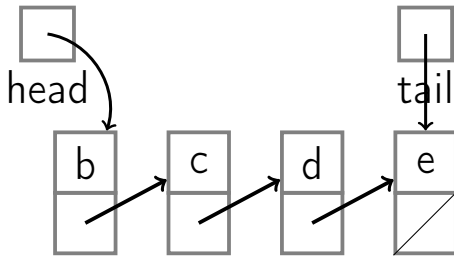
Queue Implementation with Linked List
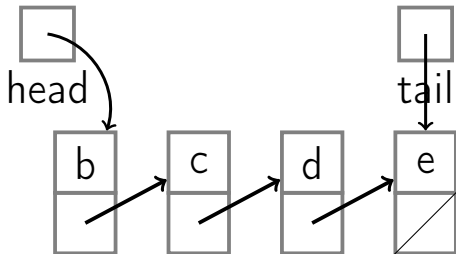
Enqueue(f)

Queue Implementation with Linked List

Enqueue(f)

Queue Implementation with Linked List

Queue Implementation with Linked List
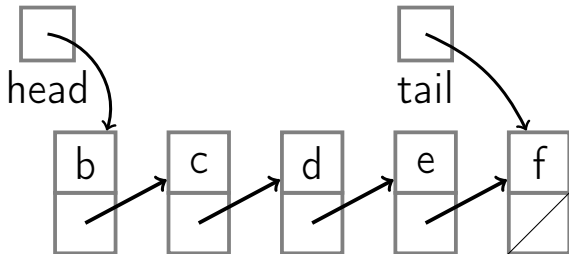
Dequeue()

Queue Implementation with Linked List

Dequeue()→ b

Queue Implementation with Linked List
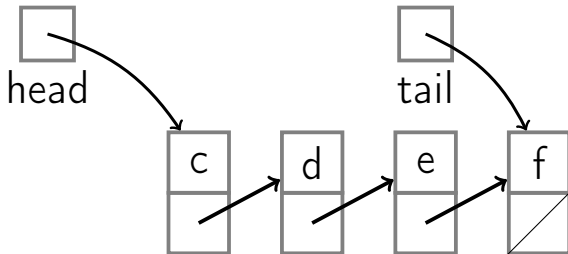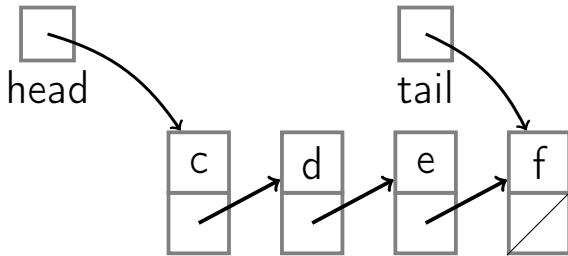
Queue Implementation with Linked List

Dequeue()

Queue Implementation with Linked List

head   tail

| d | e | f |
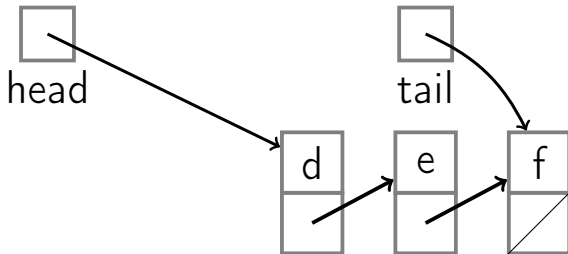
Dequeue() $\rightarrow$ c

Queue Implementation with Linked List
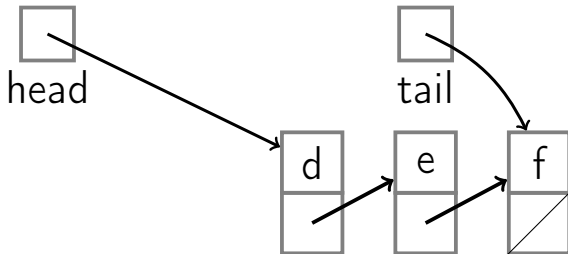
Queue Implementation with Linked List
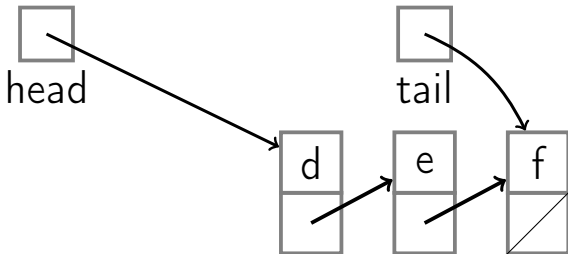
Dequeue()

Queue Implementation with Linked List

Queue Implementation with Linked List

Queue Implementation with Linked List

Dequeue()

Queue Implementation with Linked List

Dequeue() $\rightarrow$ e

Queue Implementation with Linked List

Queue Implementation with Linked List

Dequeue()

Queue Implementation with Linked List

head

tail

Dequeue()$\to$ f

Queue Implementation with Linked List

head   tail

Queue Implementation with Linked List

head

tail

Empty()

Queue Implementation with Linked List

head

tail

`Empty() → True`

Queue Implementation with Linked List

head   tail

Queue Implementation with Linked List

- Enqueue: use `List.PushBack`

Queue Implementation with Linked List

- Enqueue: use `List.PushBack`
- Dequeue: use `List.TopFront` and `List.PopFront`

Queue Implementation with Linked List

- Enqueue: use `List.PushBack`
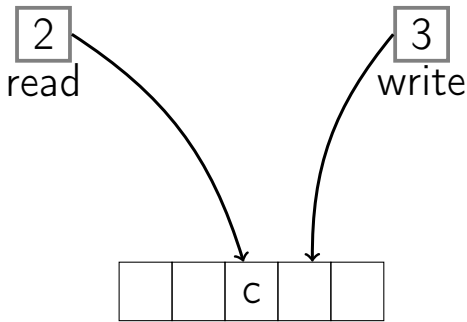- Dequeue: use `List.TopFront` and `List.PopFront`
- Empty: use `List.Empty`
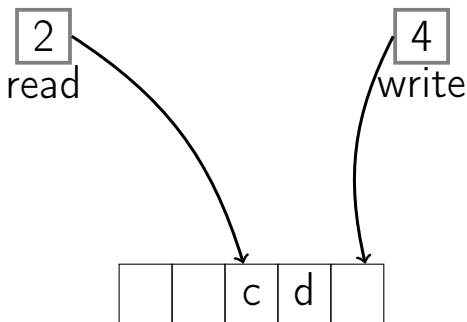
Queue Implementation with Array
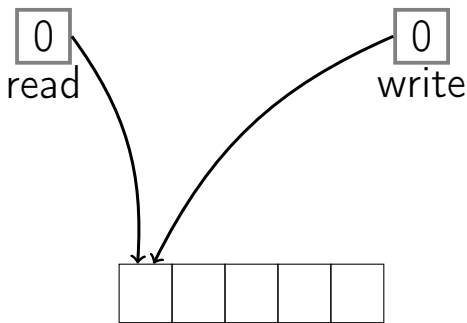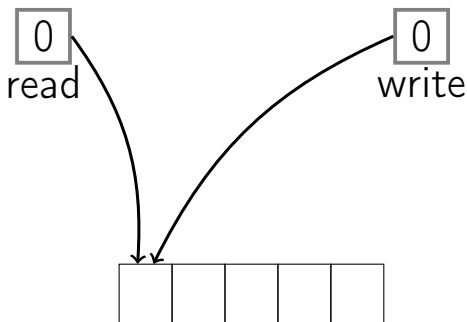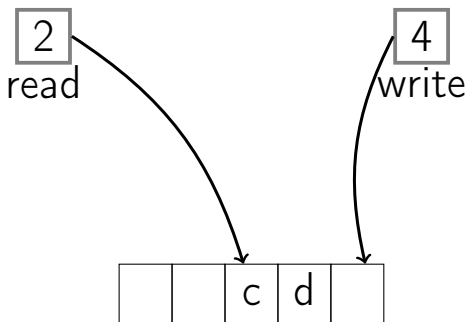
Queue Implementation with Array

Enqueue(a)
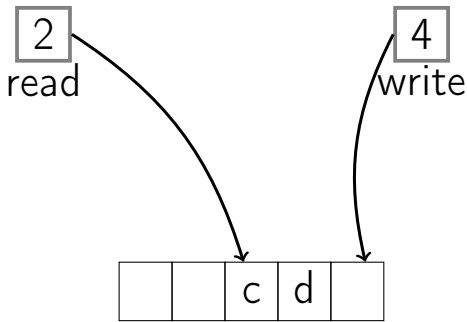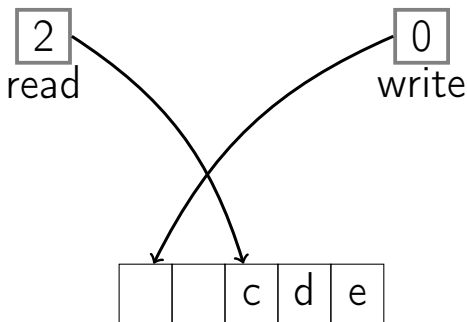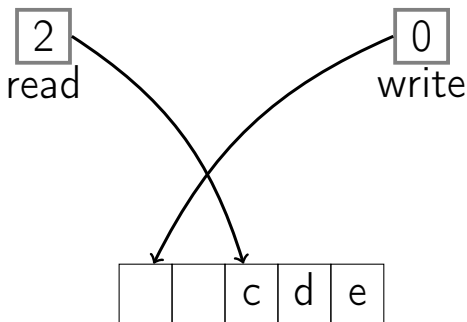
Queue Implementation with Array

Enqueue(a)

Queue Implementation with Array

Queue Implementation with Array

Enqueue(b)

Queue Implementation with Array

Enqueue(b)

Queue Implementation with Array

Queue Implementation with Array


```
Empty()
```
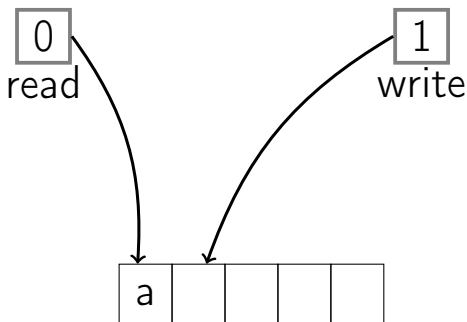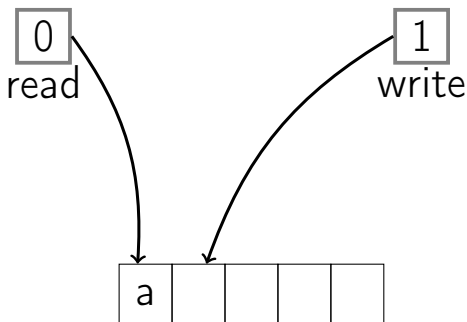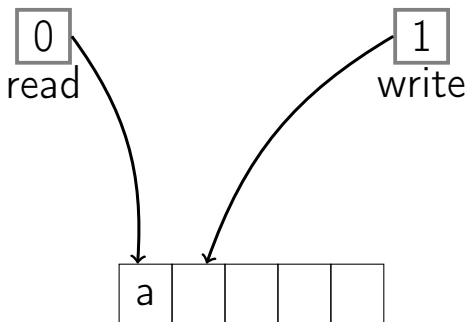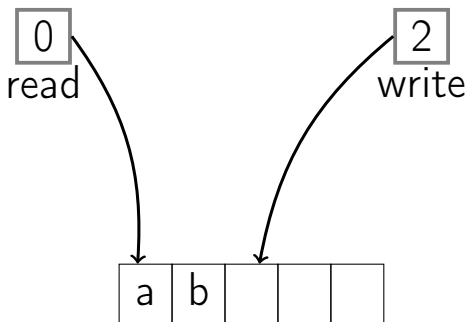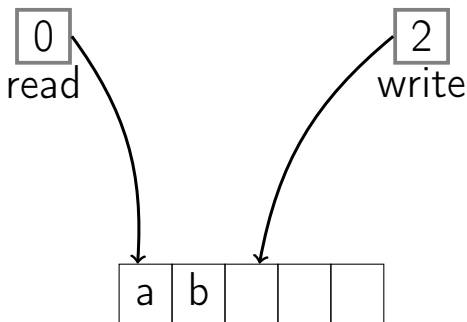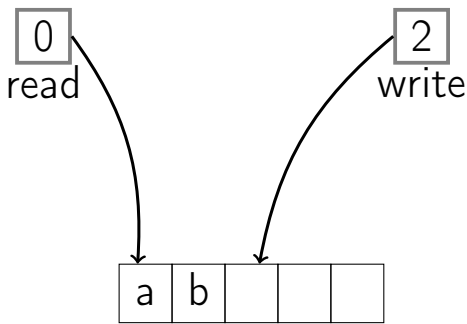
Queue Implementation with Array
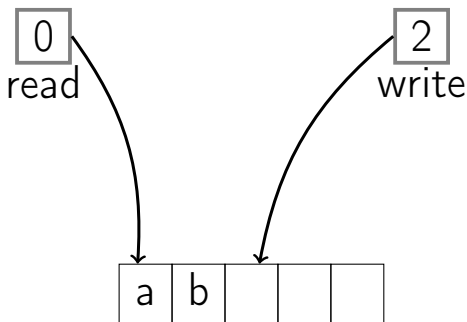
Empty() → False

Queue Implementation with Array
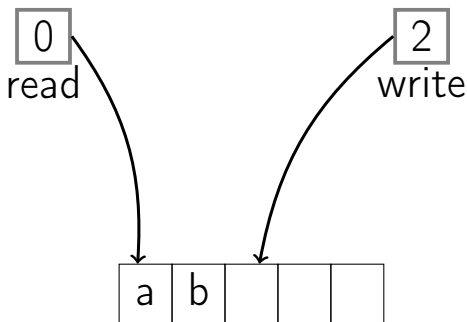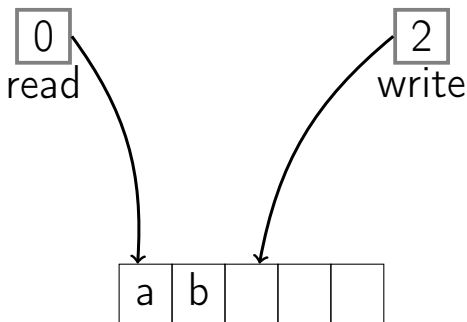
Queue Implementation with Array

Enqueue(c)

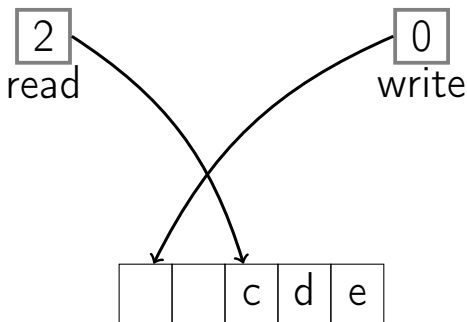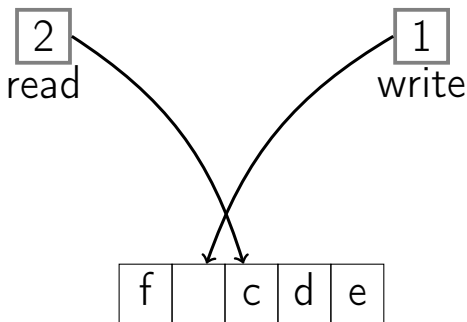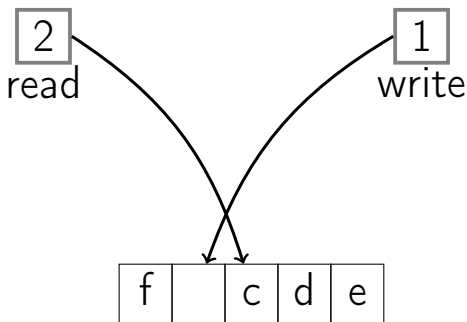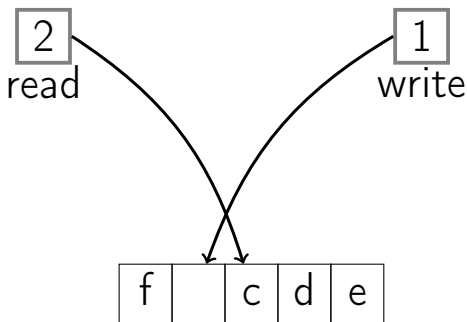# Queue Implementation with Array
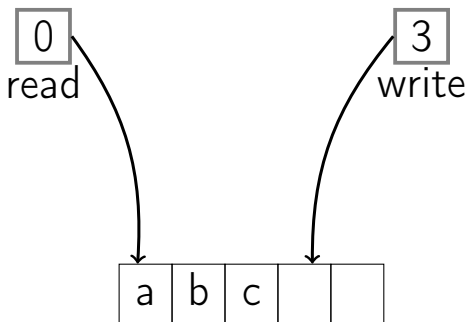
Enqueue(c)

Queue Implementation with Array
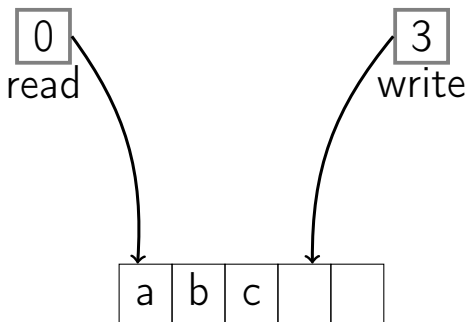
Queue Implementation with Array
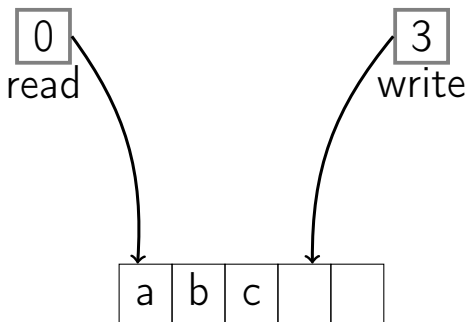
Dequeue()

Queue Implementation with Array

Dequeue() $\to$ a

Queue Implementation with Array

Queue Implementation with Array


```
Dequeue()
```
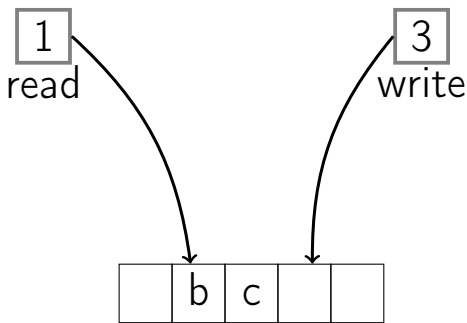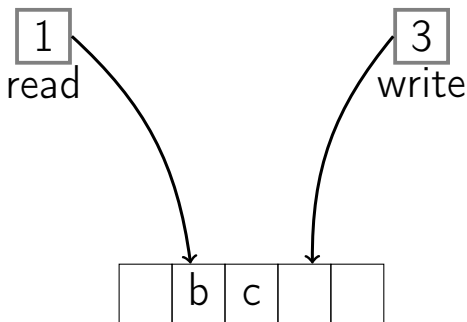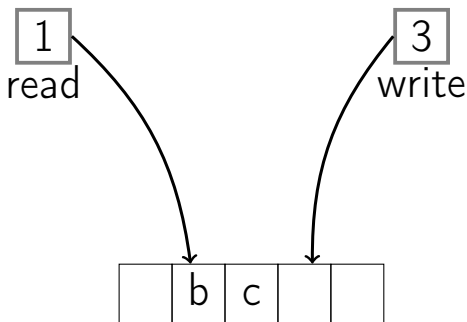
Queue Implementation with Array
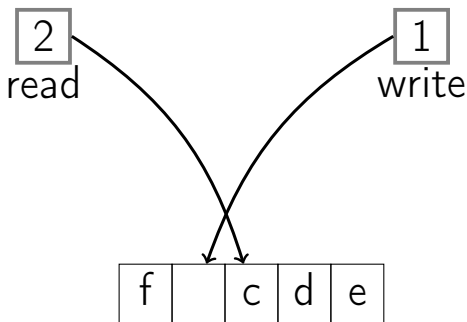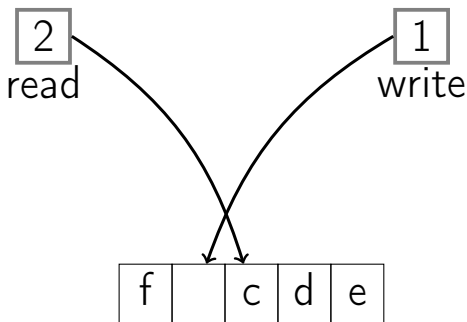

Dequeue() → b

Queue Implementation with Array
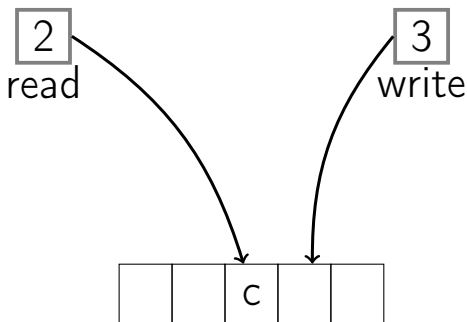
Queue Implementation with Array
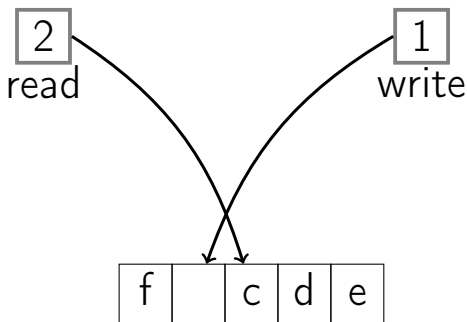

Enqueue(d)

Queue Implementation with Array

Enqueue(d)

Queue Implementation with Array

Queue Implementation with Array

Enqueue(e)

Queue Implementation with Array

Enqueue(e)

Queue Implementation with Array

Queue Implementation with Array

Enqueue(f)

Queue Implementation with Array

Enqueue(f)

Queue Implementation with Array

Queue Implementation with Array

Enqueue(g)

Queue Implementation with Array

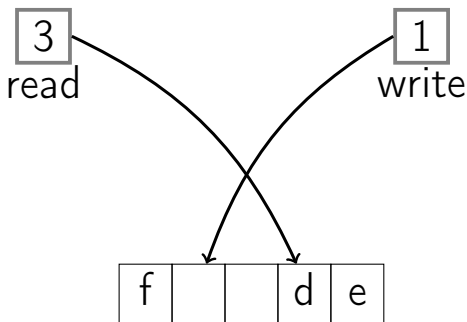Enqueue(g) $\rightarrow$ ERROR

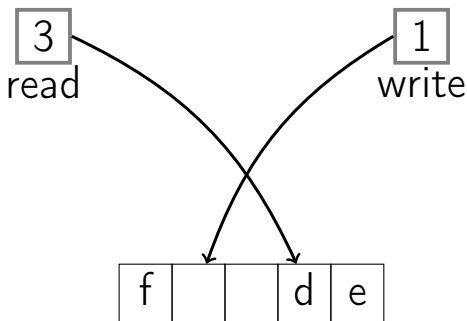# Queue Implementation with Array

Queue Implementation with Array
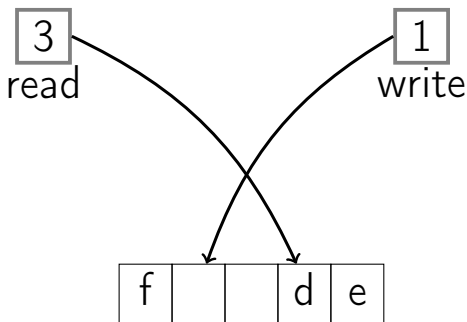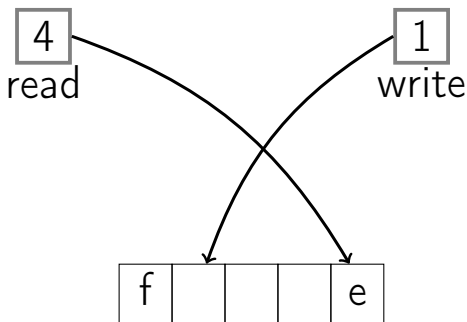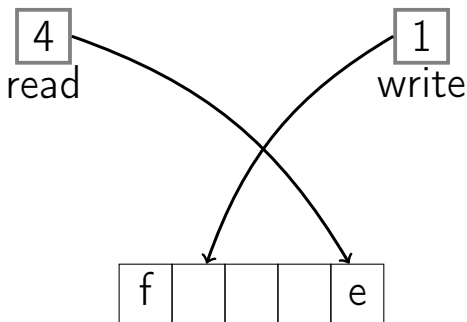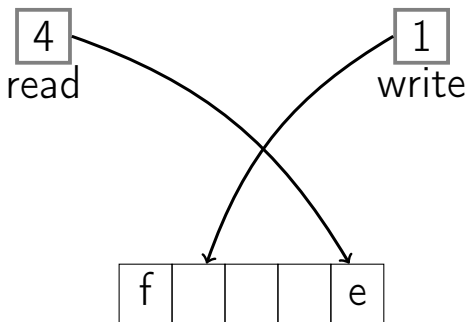

Dequeue()

Queue Implementation with Array

$\text{Dequeue()} \rightarrow$ c

Queue Implementation with Array

Queue Implementation with Array


```
Dequeue()
```

Queue Implementation with Array


```
Dequeue() → d
```
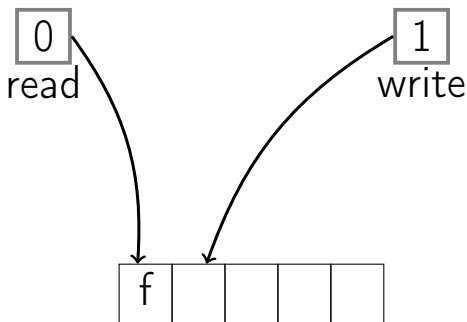
Queue Implementation with Array
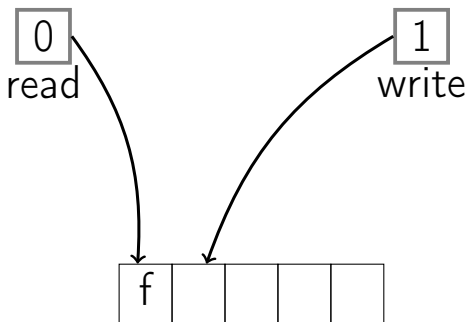
Queue Implementation with Array

Dequeue()

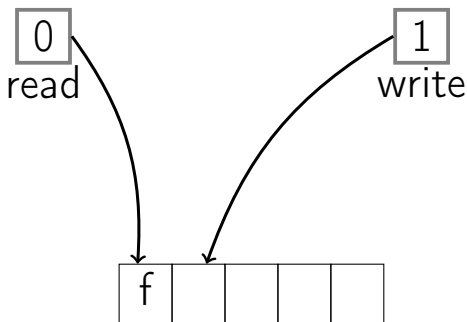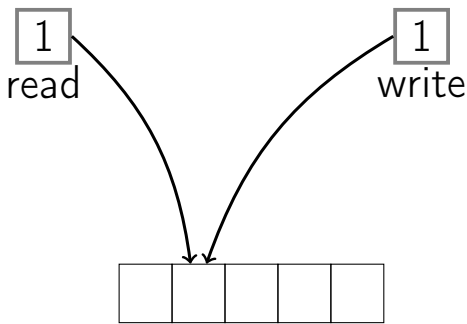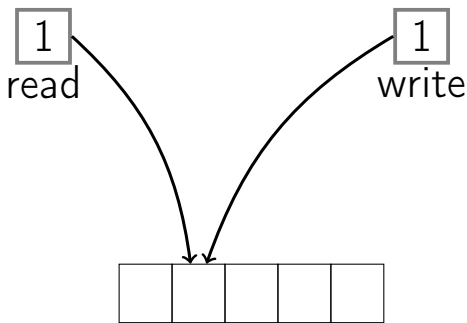# Queue Implementation with Array

Dequeue() $\rightarrow$ e

Queue Implementation with Array

Queue Implementation with Array
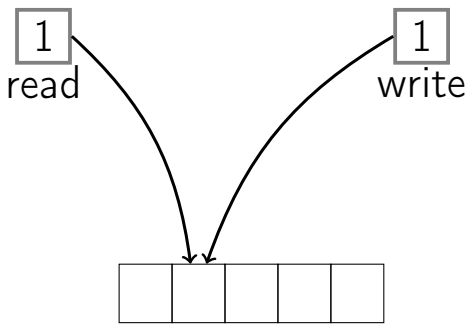

Dequeue()

Queue Implementation with Array
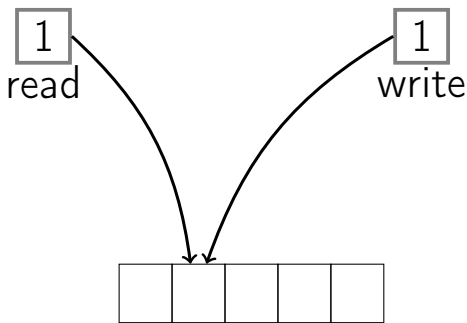

Dequeue() → f

Queue Implementation with Array

Queue Implementation with Array


```
Empty()
```

Queue Implementation with Array

Empty() $\rightarrow$ True

Summary

Summary

- Queues can be implemented with either a linked list (with tail pointer) or an array.

Summary

- Queues can be implemented with either a linked list (with tail pointer) or an array.
- Each queue operation is $O(1)$: Enqueue, Dequeue, Empty.