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:

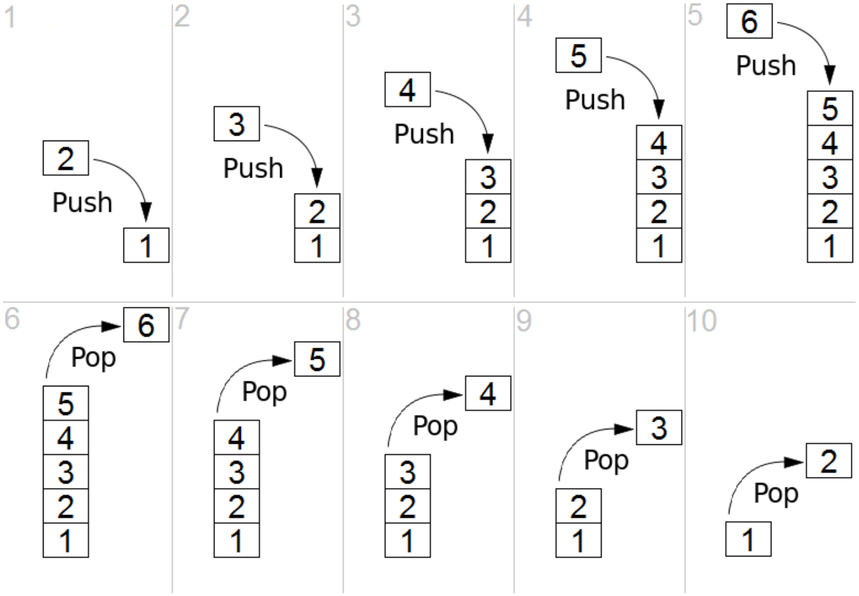
- `Push(Key)`: adds key to collection
- `Key Top()`: returns most recently-added key
- `Key Pop()`: removes and returns most recently-added key

Definition

Stack: Abstract data type with the following operations:

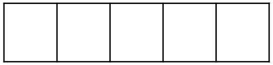
- `Push(Key)`: adds key to collection
- `Key Top()`: returns most recently-added key
- `Key Pop()`: removes and returns most recently-added key
- `Boolean Empty()`: are there any elements?

Last In First Out (LIFO)



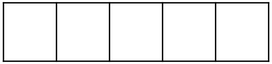
Stack Implementation with Array

numElements: 0



Stack Implementation with Array

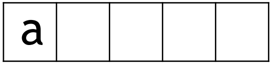
numElements: 0



`Push(a)`

Stack Implementation with Array

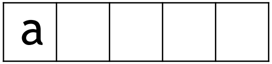
numElements: 1



`Push(a)`

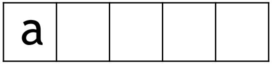
Stack Implementation with Array

numElements: 1



Stack Implementation with Array

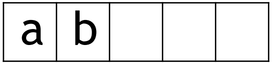
numElements: 1



Push (b)

Stack Implementation with Array

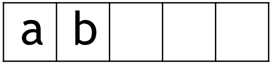
numElements: 2



Push (b)

Stack Implementation with Array

numElements: 2



Stack Implementation with Array

numElements: 2

a	b			

`Top()`

Stack Implementation with Array

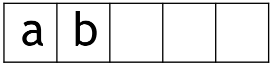
numElements: 2

a	b			

`Top()` $\longrightarrow$ `b`

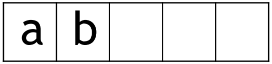
Stack Implementation with Array

numElements: 2



Stack Implementation with Array

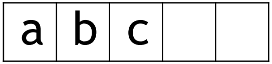
numElements: 2



`Push(c)`

Stack Implementation with Array

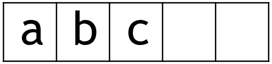
numElements: 3



```
Push(c)
```

Stack Implementation with Array

numElements: 3



Stack Implementation with Array

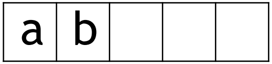
numElements: 3

a	b	c		

`Pop()`

Stack Implementation with Array

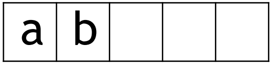
numElements: 2



`Pop()` $\rightarrow$ `c`

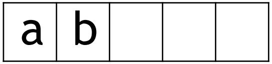
Stack Implementation with Array

numElements: 2



Stack Implementation with Array

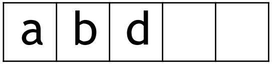
numElements: 2



```
Push(d)
```

Stack Implementation with Array

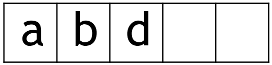
numElements: 3



`Push(d)`

Stack Implementation with Array

numElements: 3



Stack Implementation with Array

numElements: 3

a	b	d		

`Push(e)`

Stack Implementation with Array

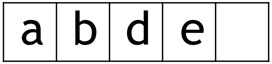
numElements: 4

a	b	d	e	

`Push(e)`

Stack Implementation with Array

numElements: 4



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)`

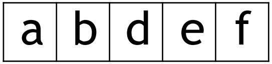
Stack Implementation with Array

numElements: 5

a	b	d	e	f

Stack Implementation with Array

numElements: 5



`Push(g)`

Stack Implementation with Array

numElements: 5

a	b	d	e	f

`Push(g)` $\longrightarrow$ 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

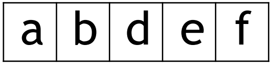
numElements: 5

a	b	d	e	f

`Empty()` $\longrightarrow$ `False`

Stack Implementation with Array

numElements: 5



Stack Implementation with Array

numElements: 5

a	b	d	e	f

```
Pop()
```

Stack Implementation with Array

numElements: 4

a	b	d	e	

`Pop() → f`

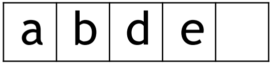
Stack Implementation with Array

numElements: 4

a	b	d	e	

Stack Implementation with Array

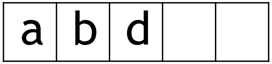
numElements: 4



`Pop()`

Stack Implementation with Array

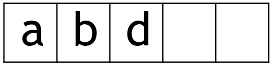
numElements: 3



`Pop() → e`

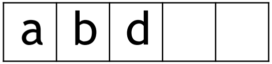
Stack Implementation with Array

numElements: 3



Stack Implementation with Array

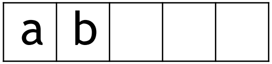
numElements: 3



`Pop()`

Stack Implementation with Array

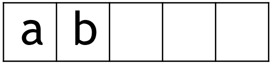
numElements: 2



`Pop() → d`

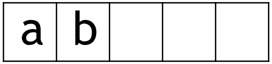
Stack Implementation with Array

numElements: 2



Stack Implementation with Array

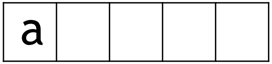
numElements: 2



`Pop()`

Stack Implementation with Array

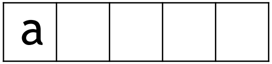
numElements: 1



Pop() $\rightarrow$ b

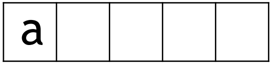
Stack Implementation with Array

numElements: 1



Stack Implementation with Array

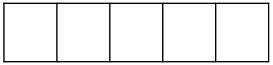
numEElements: 1



`Pop()`

Stack Implementation with Array

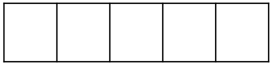
numElements: 0



`Pop()` $\rightarrow$ `a`

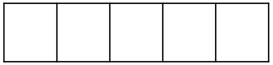
Stack Implementation with Array

numElements: 0



Stack Implementation with Array

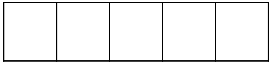
numElements: 0



`Empty()`

Stack Implementation with Array

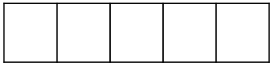
numEmelents: 0



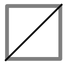
`Empty() → True`

Stack Implementation with Array

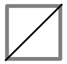
numElements: 0



Stack Implementation with Linked List

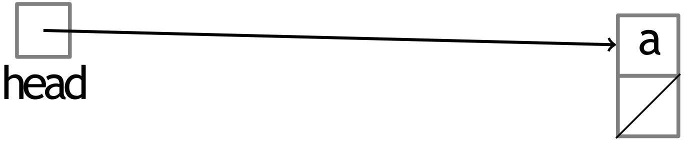


Stack Implementation with Linked List



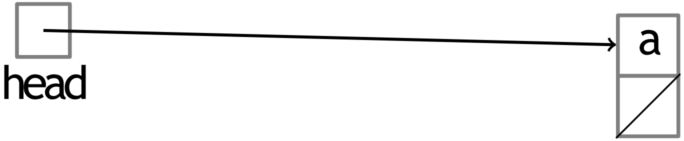
`Push(a)`

Stack Implementation with Linked List

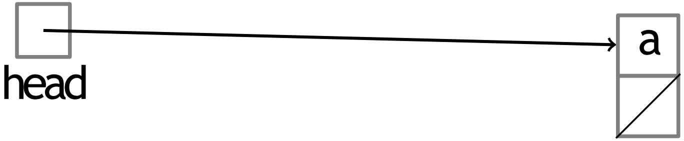


```
Push(a)
```

Stack Implementation with Linked List

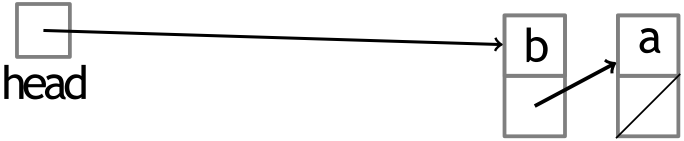


Stack Implementation with Linked List



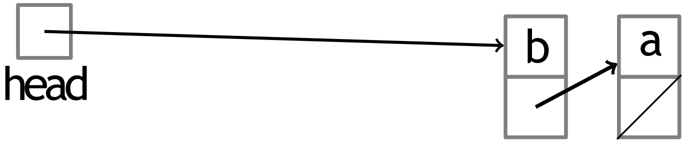
Push(b)

Stack Implementation with Linked List

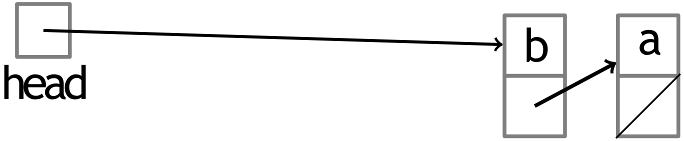


Push(b)

Stack Implementation with Linked List

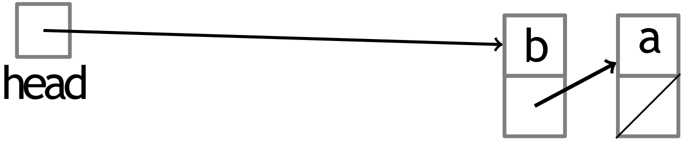


Stack Implementation with Linked List



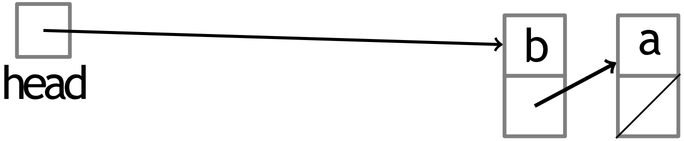
`Top()`

Stack Implementation with Linked List

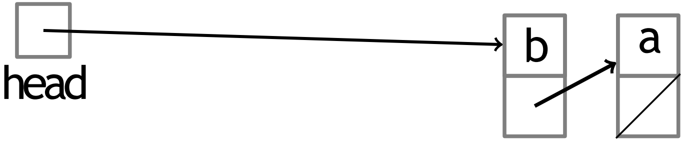


```
Top() → b
```

Stack Implementation with Linked List

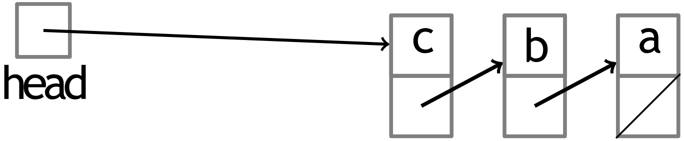


Stack Implementation with Linked List



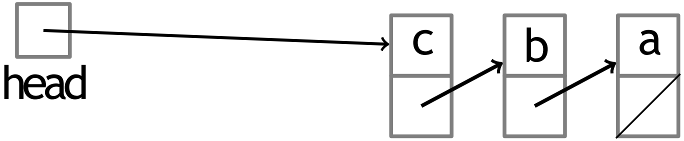
`Push(c)`

Stack Implementation with Linked List

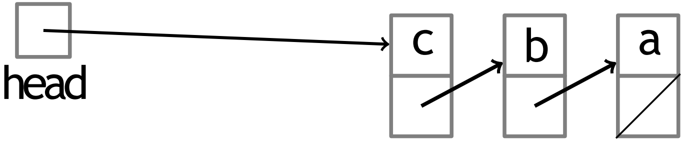


`Push(c)`

Stack Implementation with Linked List

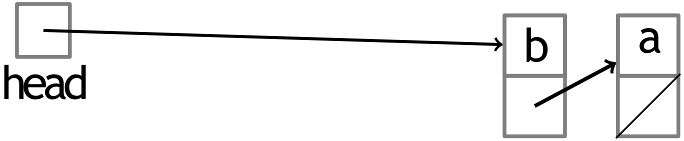


Stack Implementation with Linked List



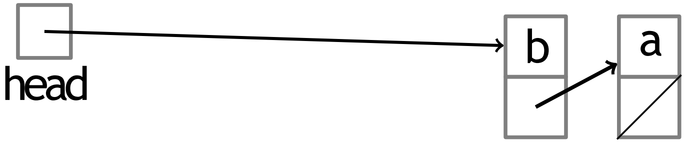
`Pop()`

Stack Implementation with Linked List

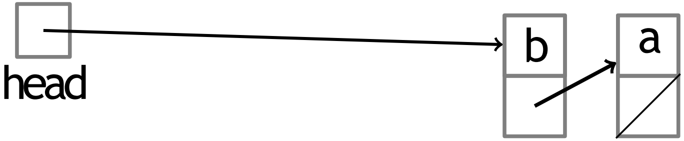


`Pop() → c`

Stack Implementation with Linked List

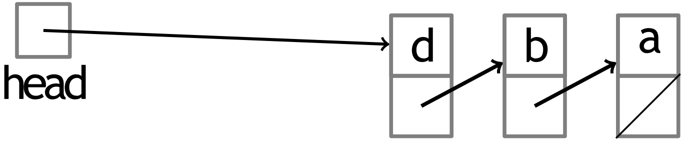


Stack Implementation with Linked List



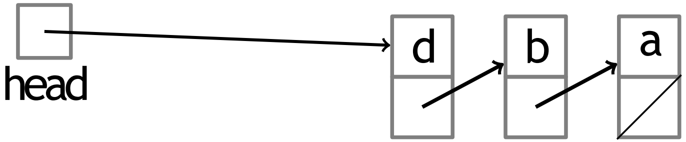
Push(d)

Stack Implementation with Linked List

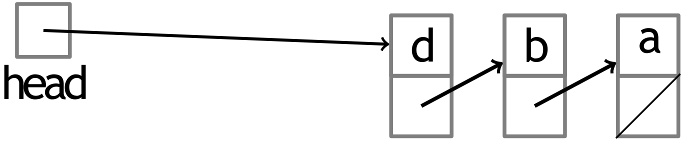


Push(d)

Stack Implementation with Linked List

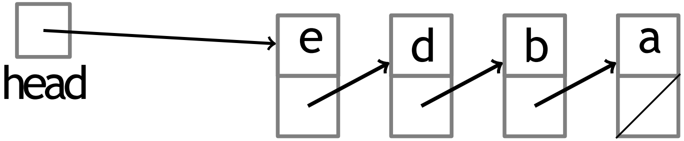


Stack Implementation with Linked List



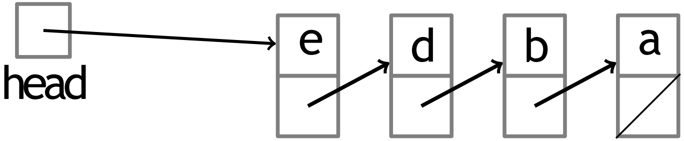
`Push(e)`

Stack Implementation with Linked List

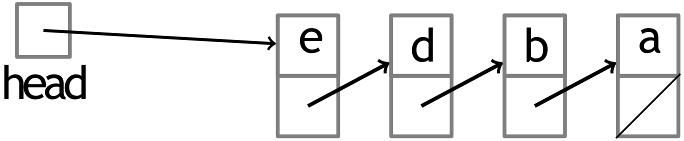


Push(e)

Stack Implementation with Linked List

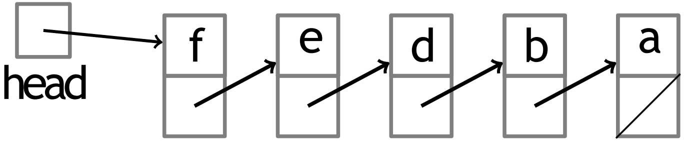


Stack Implementation with Linked List



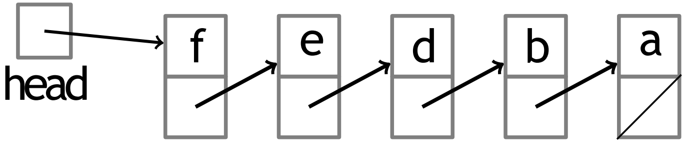
`Push(f)`

Stack Implementation with Linked List

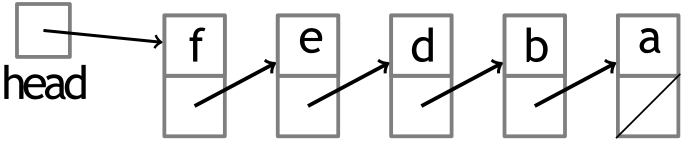


`Push(f)`

Stack Implementation with Linked List

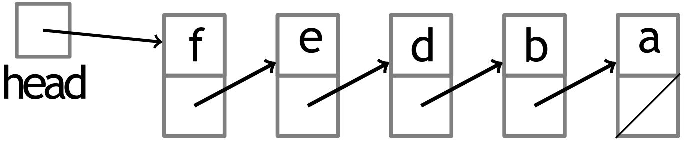


Stack Implementation with Linked List



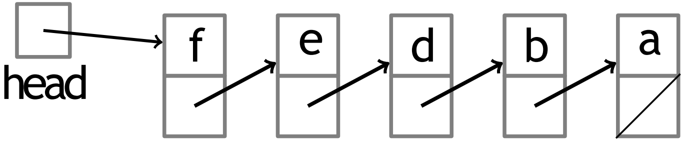
`Empty()`

Stack Implementation with Linked List

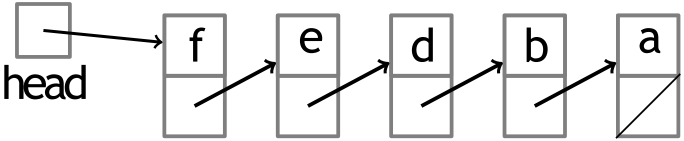


`Empty() → False`

Stack Implementation with Linked List

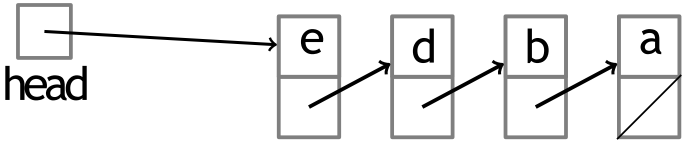


Stack Implementation with Linked List



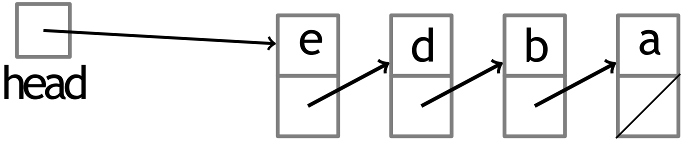
Pop()

Stack Implementation with Linked List

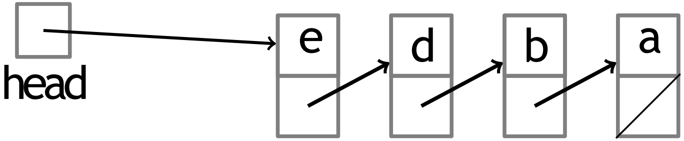


$\text{Pop}() \rightarrow f$

Stack Implementation with Linked List

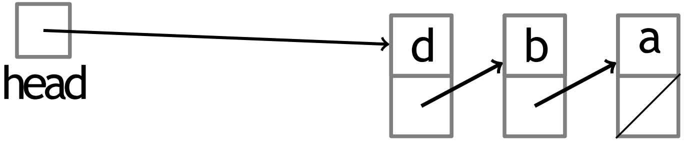


Stack Implementation with Linked List



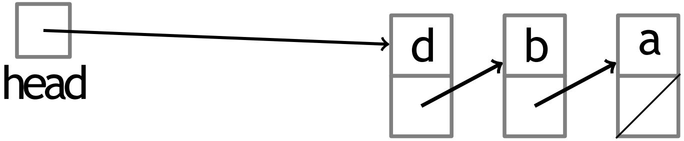
Pop()

Stack Implementation with Linked List

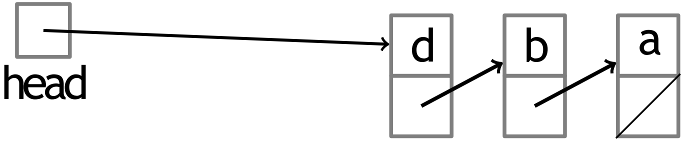


`Pop() → e`

Stack Implementation with Linked List

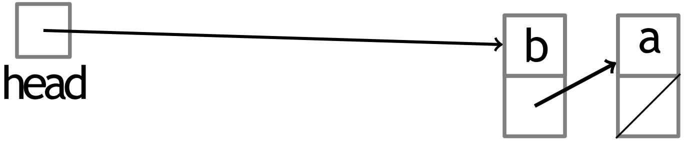


Stack Implementation with Linked List



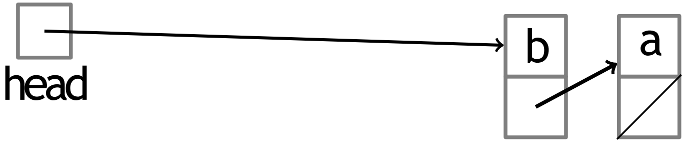
Pop ( )

Stack Implementation with Linked List

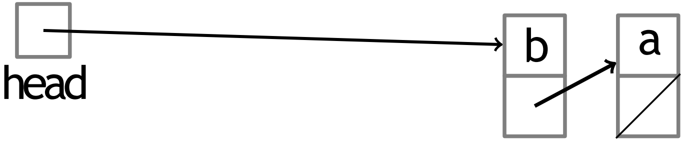


Pop() $\longrightarrow$ d

Stack Implementation with Linked List

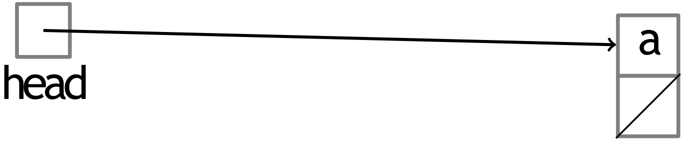


Stack Implementation with Linked List



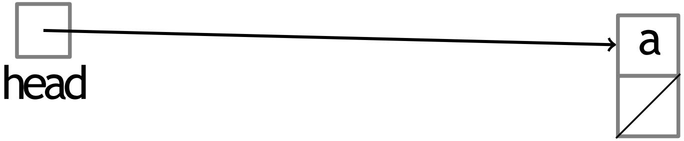
`Pop()`

Stack Implementation with Linked List

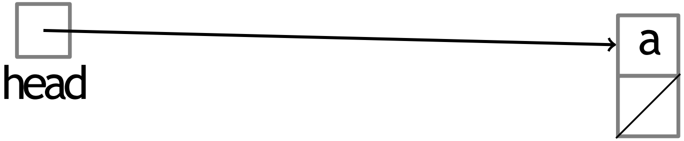


`Pop() ⟶ b`

Stack Implementation with Linked List

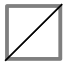


Stack Implementation with Linked List



`Pop()`

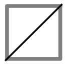
Stack Implementation with Linked List



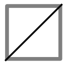
head

`Pop() → a`

Stack Implementation with Linked List



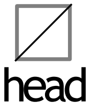
Stack Implementation with Linked List



head

`Empty()`

Stack Implementation with Linked List



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

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.

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()
```

Assignment

1. Reverse individual words using stack
    - Input : Hello World
    - Output : olleH dlroW
2. Given an expression string exp , write a program to examine whether the pairs and the orders of “{“,”}”,”(“,”)”,”[“,”]” are correct in exp. For example, the program should print true for exp = “[()]{}{[()()]()}” and false for exp = “[(])”
3. Write getMin() and getMax() functions for stack

Step by step explanation :

Suppose the elements are pushed on to the stack in the order {4, 2, 14, 1, 18}

Step 1 : Push 4, Current max : 4

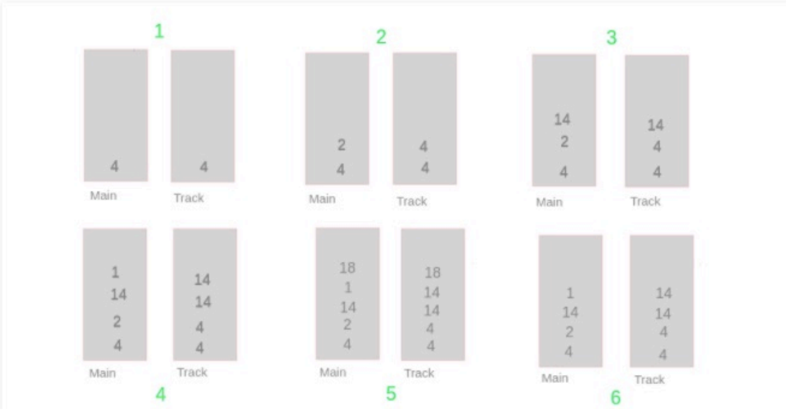
Step 2 : Push 2, Current max : 4

Step 3 : Push 14, Current max : 14

Step 4 : Push 1, Current max : 14

Step 5 : Push 18, Current max : 18

Step 6 : Pop 18, Current max : 14



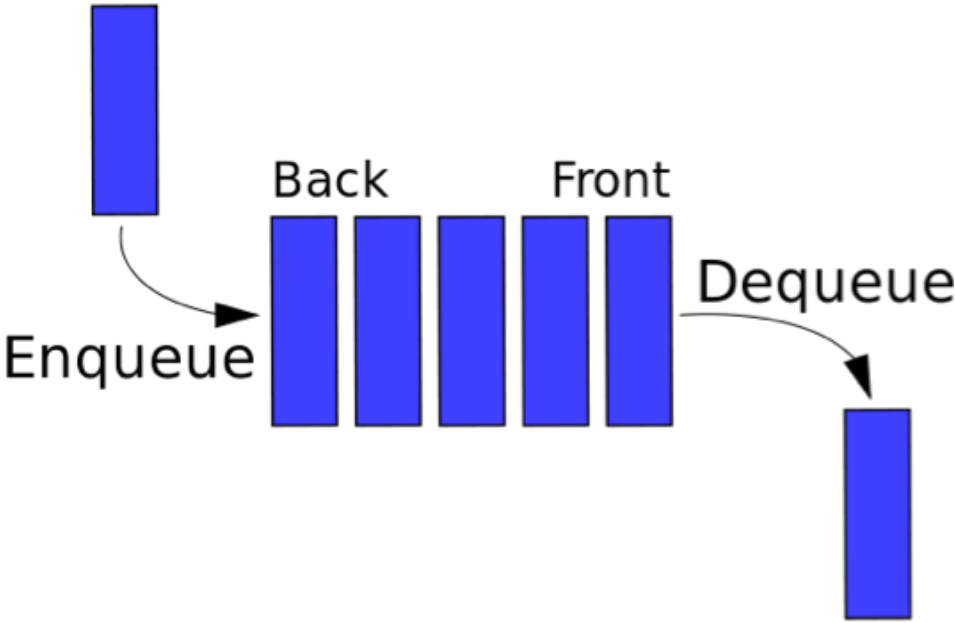
Outline

Definition

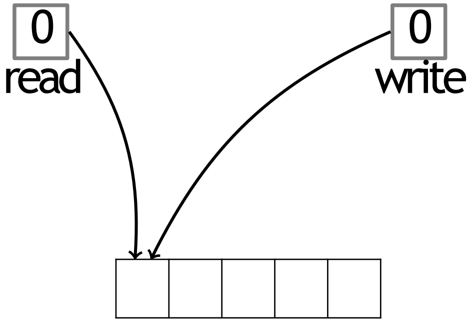
Queue: Abstract data type with the following operations:

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

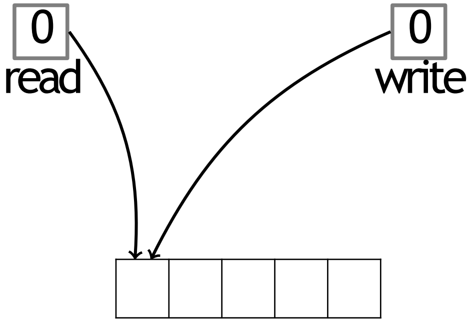
FIFO: First-In, First-Out


Back
Front
Enqueue
Dequeue

Queue Implementation with Array

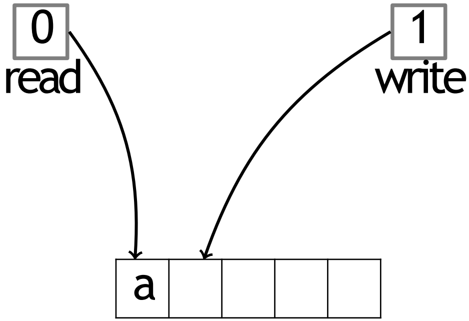


Queue Implementation with Array



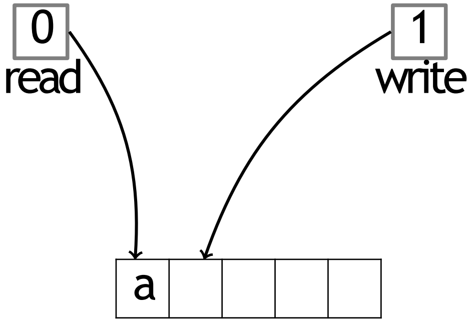
Enqueue(a)

Queue Implementation with Array

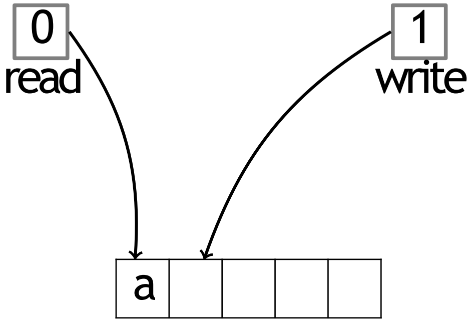


Enqueue(a)

Queue Implementation with Array

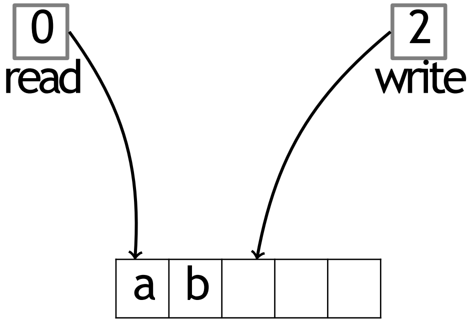


Queue Implementation with Array



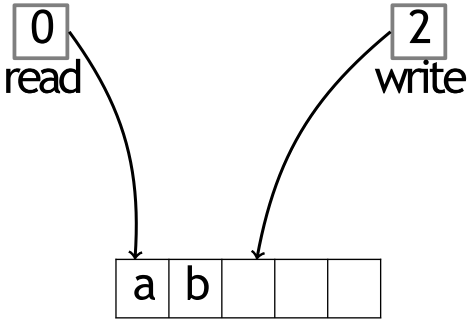
Enqueue (b)

Queue Implementation with Array

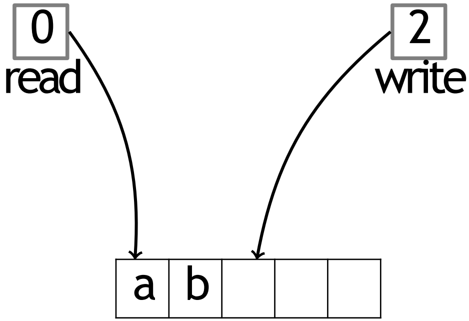


Enqueue (b)

Queue Implementation with Array

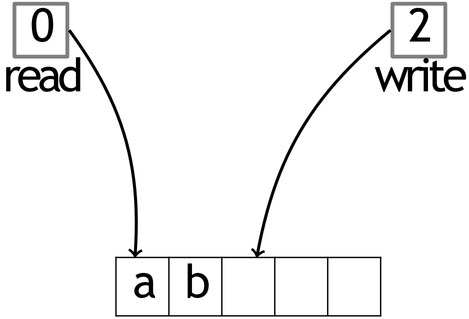


Queue Implementation with Array



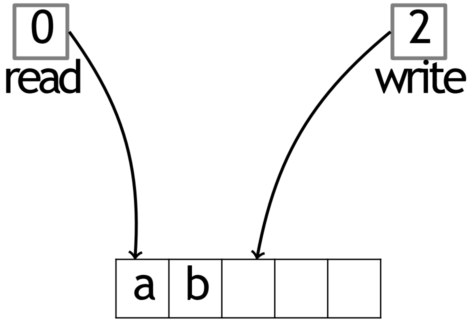
`Empty()`

Queue Implementation with Array

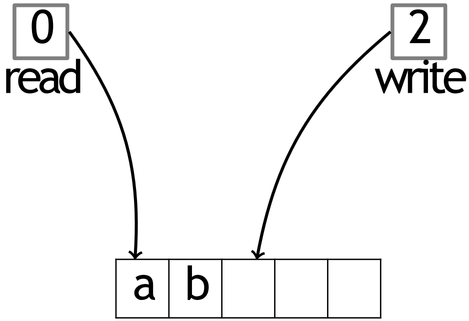


`Empty()` $\longrightarrow$ `False`

Queue Implementation with Array

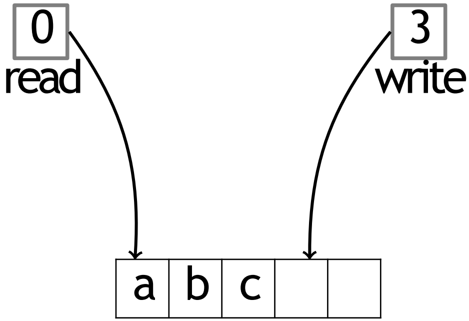


Queue Implementation with Array



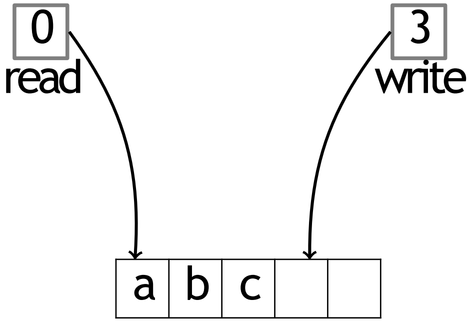
Enqueue(c)

Queue Implementation with Array

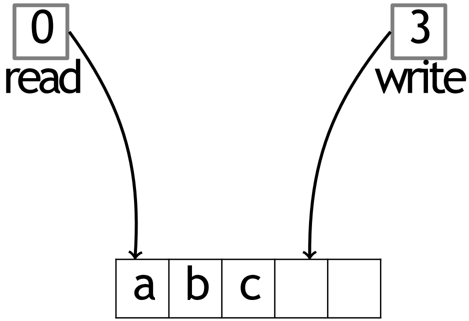


Enqueue(c)

Queue Implementation with Array

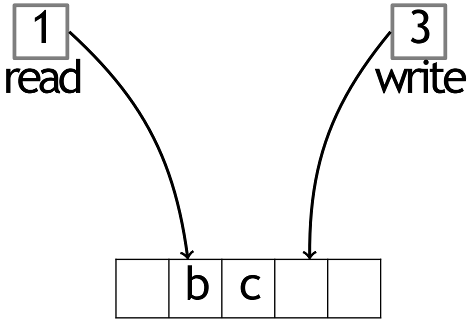


Queue Implementation with Array



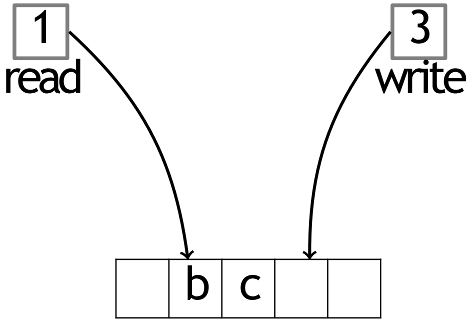
`Dequeue()`

Queue Implementation with Array

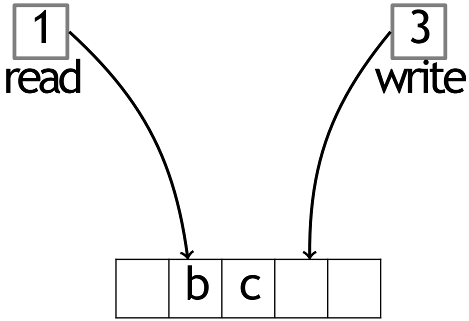


`Dequeue()` $\rightarrow$ `a`

Queue Implementation with Array

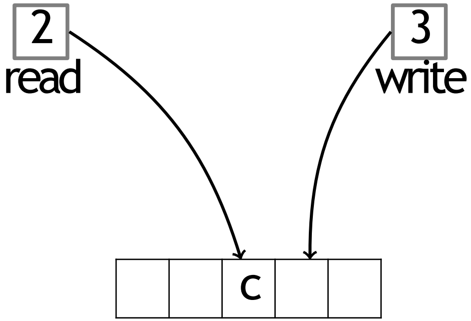


Queue Implementation with Array



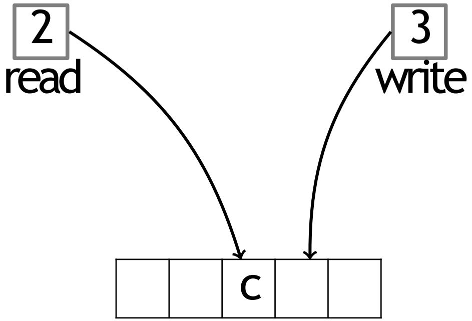
`Dequeue()`

Queue Implementation with Array

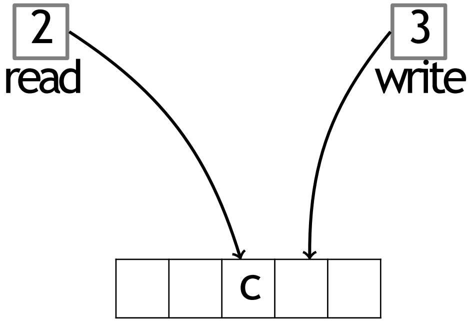


`Dequeue()` $\longrightarrow$ `b`

Queue Implementation with Array

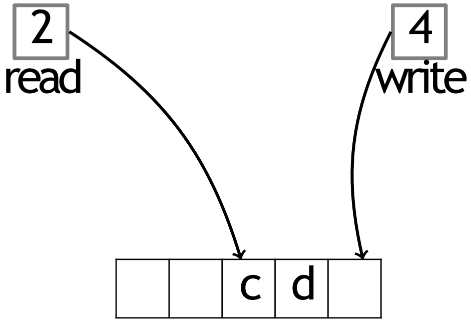


Queue Implementation with Array



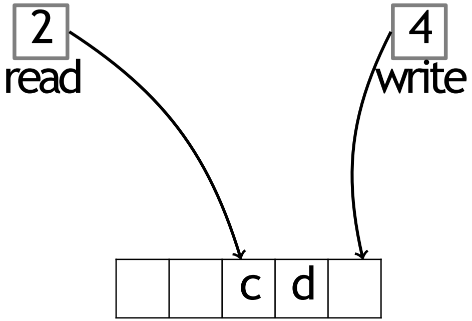
`Enqueue(d)`

Queue Implementation with Array

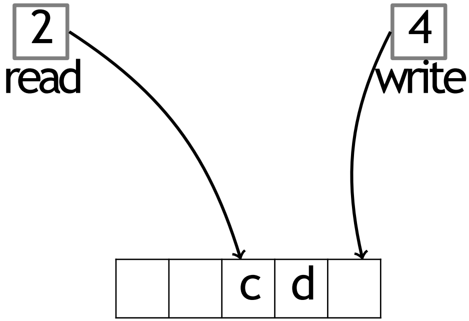


Enqueue(d)

Queue Implementation with Array

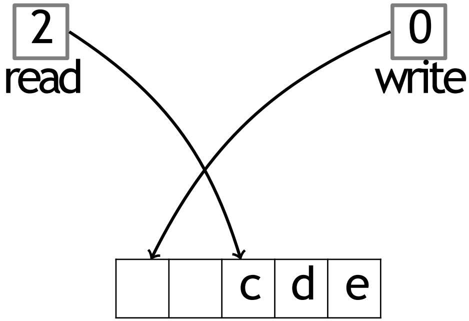


Queue Implementation with Array



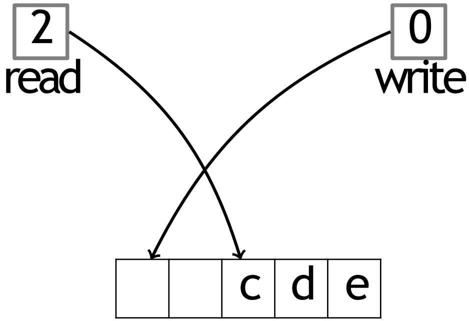
`Enqueue(e)`

Queue Implementation with Array

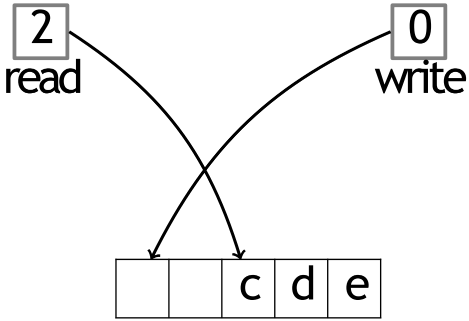


`Enqueue(e)`

Queue Implementation with Array

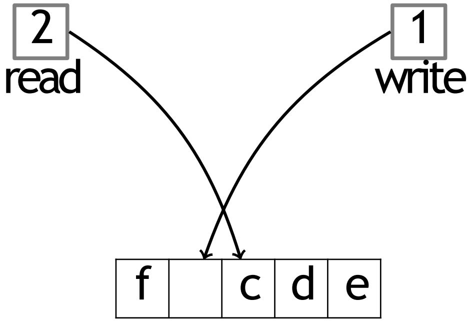


Queue Implementation with Array



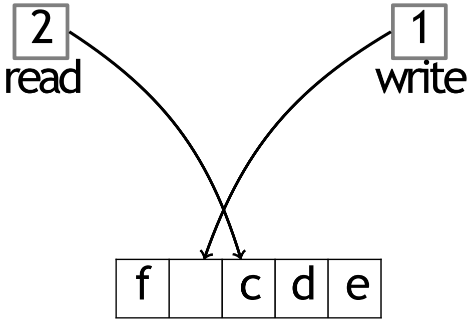
`Enqueue(f)`

Queue Implementation with Array

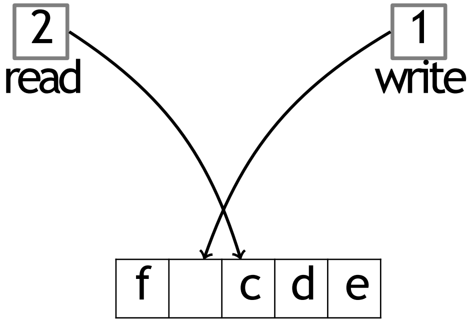


`Enqueue(f)`

Queue Implementation with Array

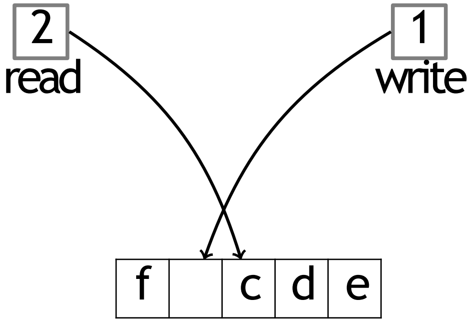


Queue Implementation with Array



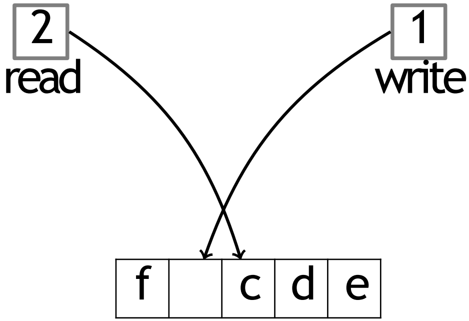
Enqueue(g)

Queue Implementation with Array

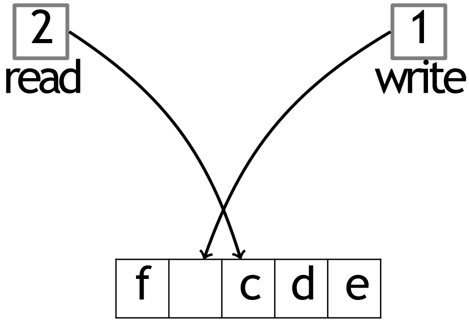


`Enqueue(g)` $\longrightarrow$ `ERROR`

Queue Implementation with Array

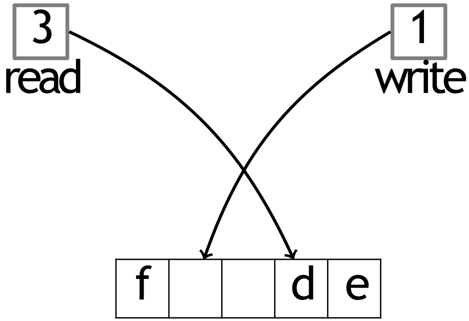


Queue Implementation with Array



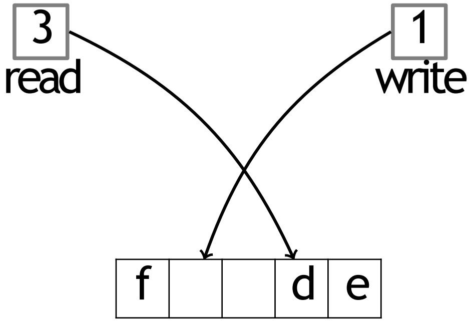
`Dequeue()`

Queue Implementation with Array

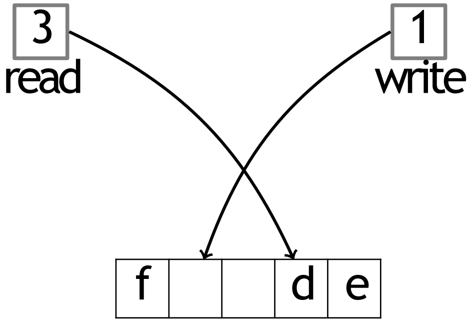


Dequeue() → c

Queue Implementation with Array

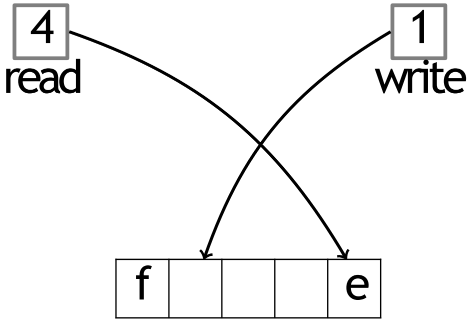


Queue Implementation with Array



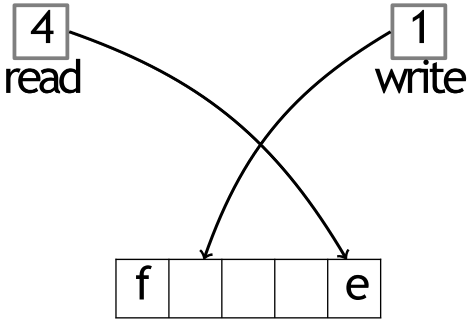
```
Dequeue()
```

Queue Implementation with Array

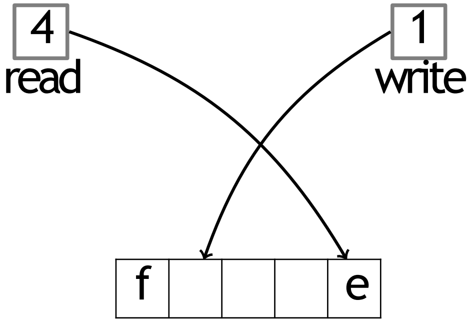


Dequeue() → d

Queue Implementation with Array

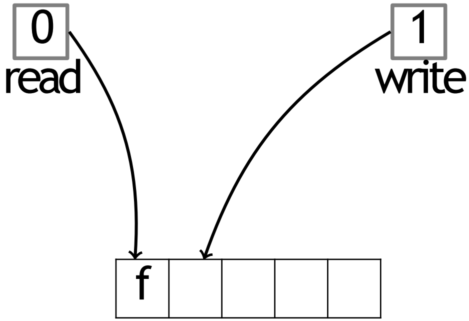


Queue Implementation with Array



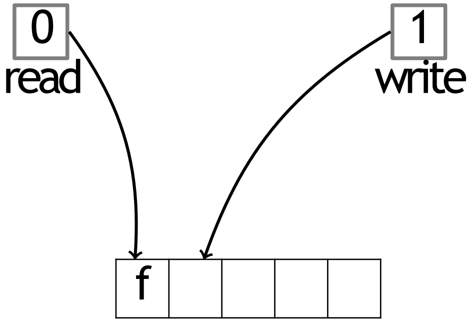
`Dequeue()`

Queue Implementation with Array

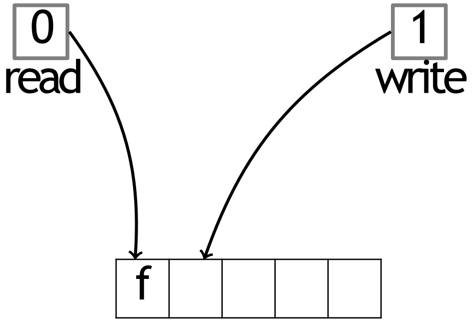


Dequeue() $\rightarrow$ e

Queue Implementation with Array

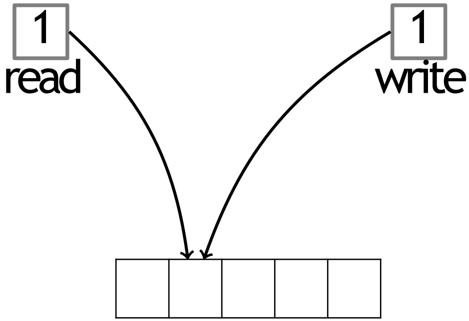


Queue Implementation with Array



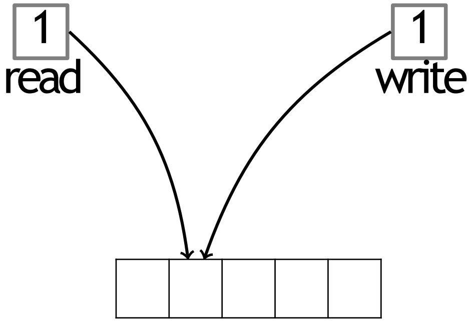
`Dequeue()`

Queue Implementation with Array

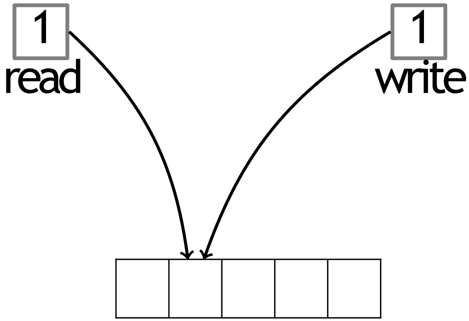


Dequeue() $\rightarrow$ f

Queue Implementation with Array

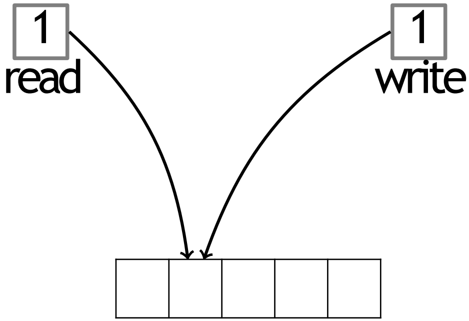


Queue Implementation with Array



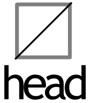
`Empty()`

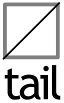
Queue Implementation with Array



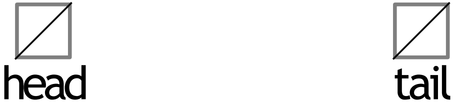
`Empty()` $\longrightarrow$ `True`

Queue Implementation with Linked List



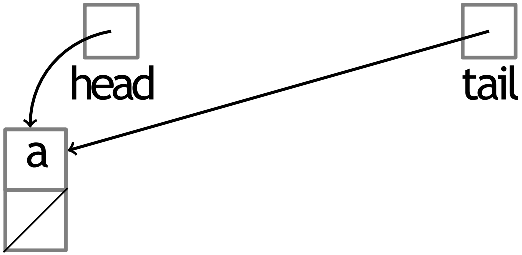


Queue Implementation with Linked List



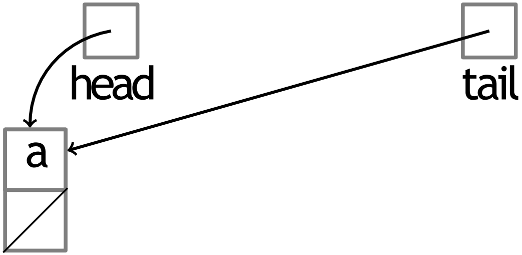
`Enqueue(a)`

Queue Implementation with Linked List



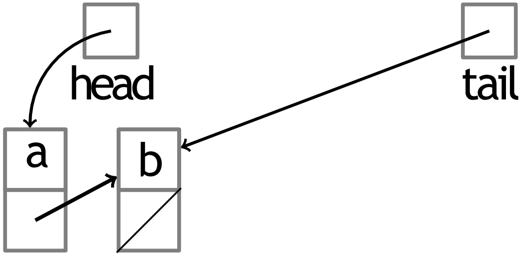
Enqueue(a)

Queue Implementation with Linked List



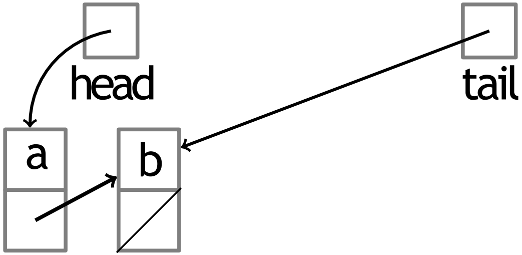
Enqueue (b)

Queue Implementation with Linked List

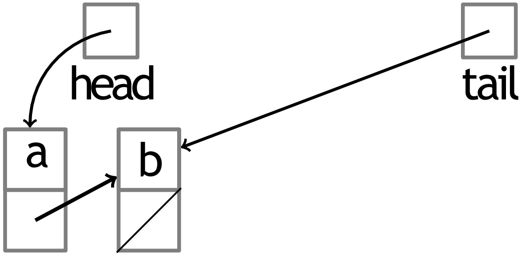


Enqueue (b)

Queue Implementation with Linked List

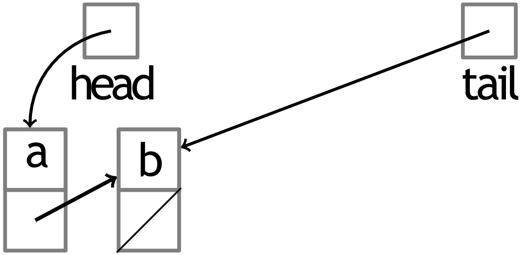


Queue Implementation with Linked List



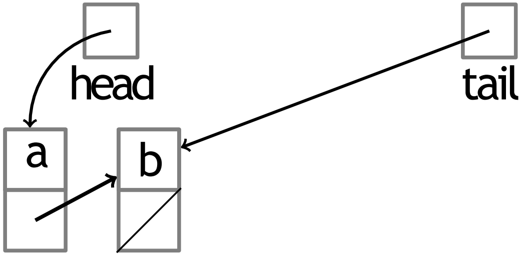
`Empty()`

Queue Implementation with Linked List



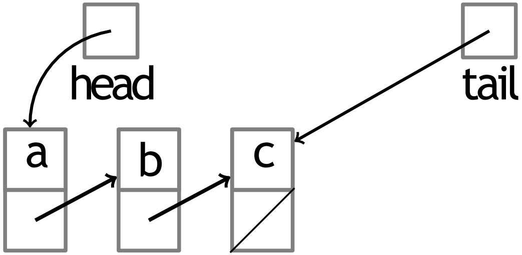
`Empty()` $\longrightarrow$ `False`

Queue Implementation with Linked List



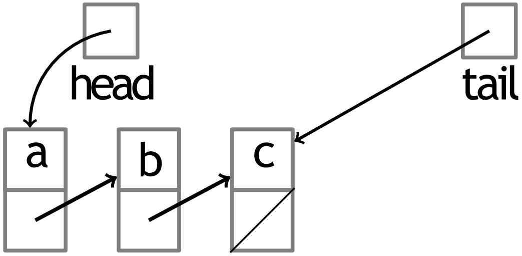
Enqueue(c)

Queue Implementation with Linked List



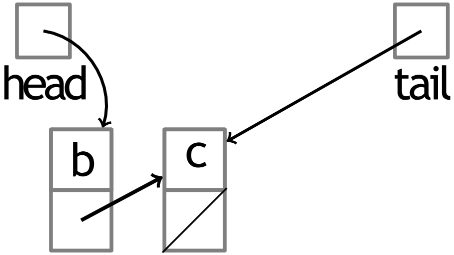
Enqueue(c)

Queue Implementation with Linked List



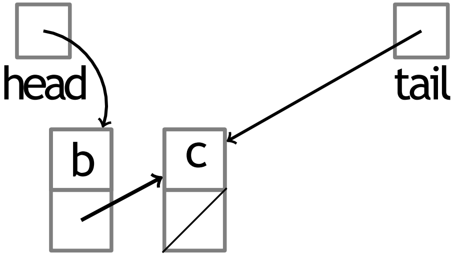
`Dequeue()`

Queue Implementation with Linked List



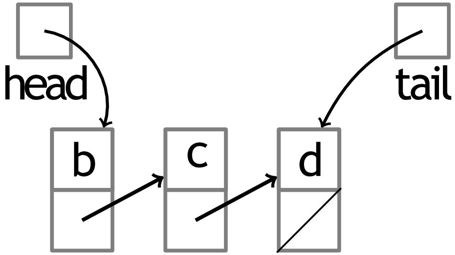
`Dequeue() → a`

Queue Implementation with Linked List



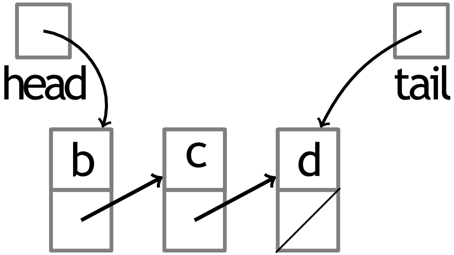
Enqueue(d)

Queue Implementation with Linked List



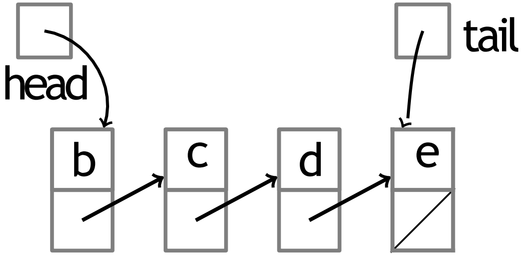
Enqueue(d)

Queue Implementation with Linked List



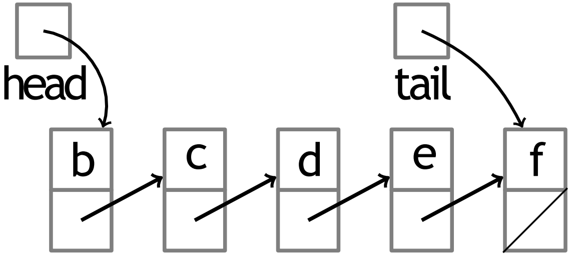
Enqueue(e)

Queue Implementation with Linked List



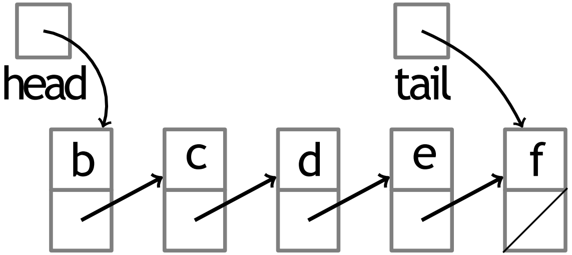
Enqueue(e)

Queue Implementation with Linked List



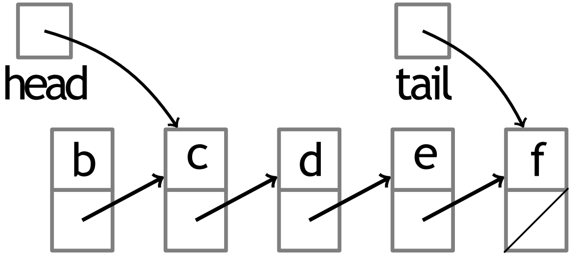
`Enqueue(f)`

Queue Implementation with Linked List



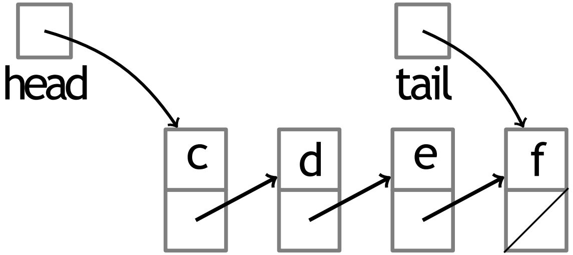
Dequeue()

Queue Implementation with Linked List

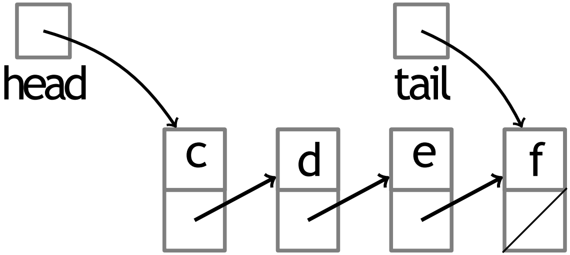


Dequeue()$\longrightarrow$ b

Queue Implementation with Linked List

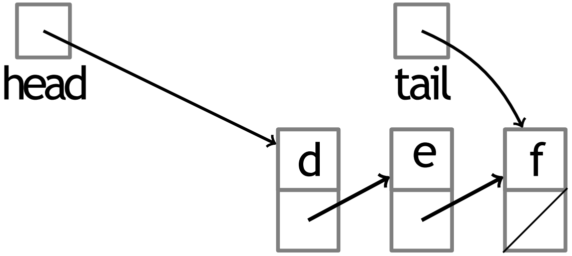


Queue Implementation with Linked List



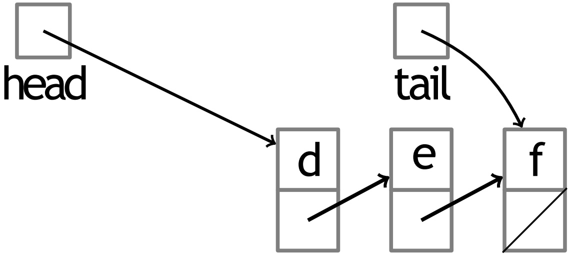
`Deque ()`

Queue Implementation with Linked List

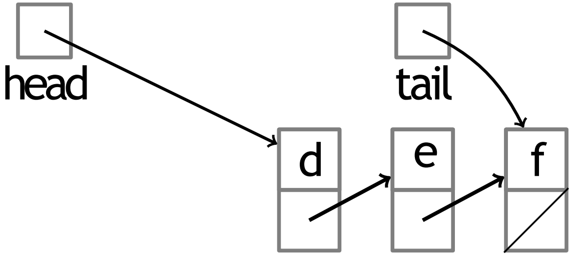


`Dequeue()` $\rightarrow$ `c`

Queue Implementation with Linked List

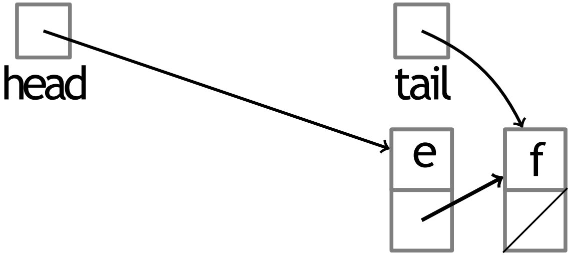


Queue Implementation with Linked List



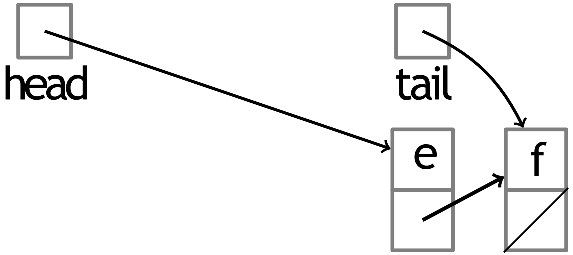
`Deque ()`

Queue Implementation with Linked List

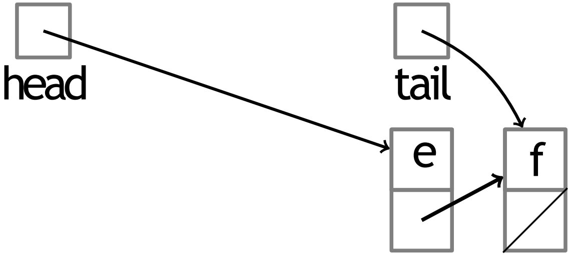


Dequeue() → d

Queue Implementation with Linked List

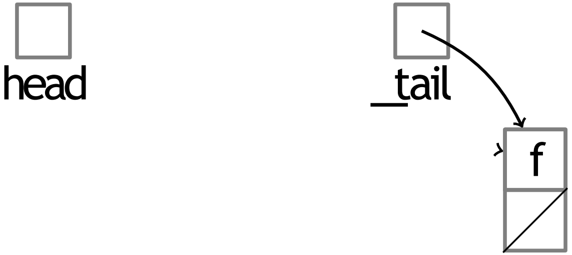


Queue Implementation with Linked List



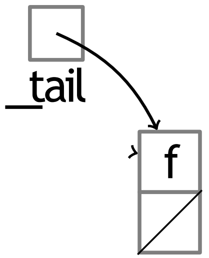
`Dequeue()`

Queue Implementation with Linked List

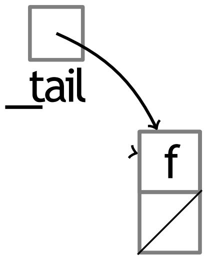


Dequeue() → e

Queue Implementation with Linked List

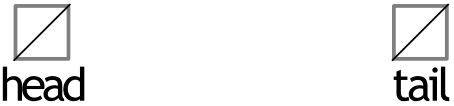


Queue Implementation with Linked List



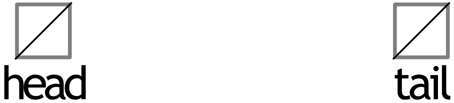
```
Dequeue()
```

Queue Implementation with Linked List

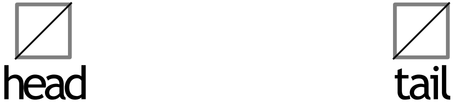


`Dequeue()` $\longrightarrow$ `f`

Queue Implementation with Linked List

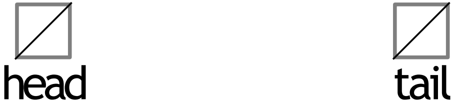


Queue Implementation with Linked List



`Empty()`

Queue Implementation with Linked List



`Empty()` $\longrightarrow$ `True`

Queue Implementation with Linked List

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

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.

Assignment

- Implement a queue using two stacks
- Given a queue of integers of even length, rearrange the elements by interleaving the first half of the queue with the second half of the queue. Only a stack can be used as an auxiliary space. Examples:
  - Input : 1 2 3 4
  - Output : 1 3 2 4
  - Input : 11 12 13 14 15 16 17 18 19 20
  - Output : 11 16 12 17 13 18 14 19 15 20

Following are the steps to solve the problem:

1.Push the first half elements of queue to stack.

2.Enqueue back the stack elements.

4.Again push the first half elements into the stack.

3.Dequeue the first half elements of the queue and enqueue them back.

5.Interleave the elements of queue and stack.