

CS221 Data Structure

Chapter 6

Queues



By Dr. Paween Khoenkaw
Computer Science MJU



Queues



มีช่องทางเข้า 1 ทางและช่องทางออก 1 ทาง

สิ่งที่ใส่เข้าไปก่อน จะถูกนำออกมาใช้ก่อน

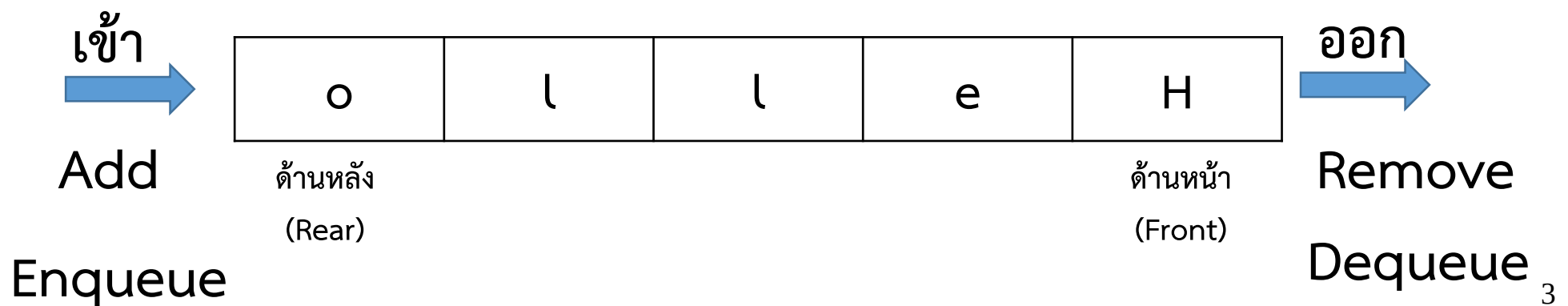
First-in First-out (FIFO)

First come First serve

Queue = แถวลำดับ

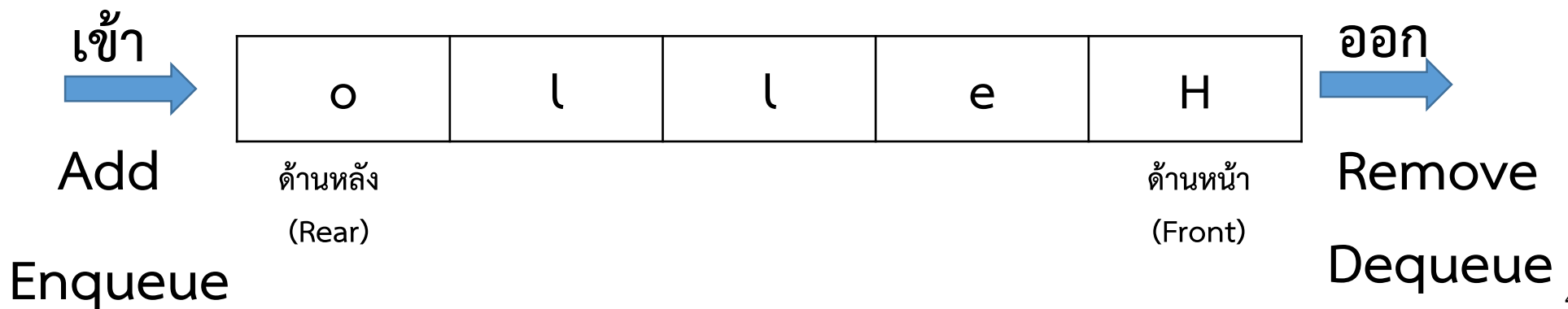
Queues

แถวคอย หรือ คิว (อังกฤษ: **queue**) เป็นแบบชนิดข้อมูลนามธรรมที่มีลักษณะการเรียงลำดับข้อมูล การดำเนินการในแถวคอยจะแบ่งเป็น การเพิ่มข้อมูลไปที่ส่วนหลังสุดของแถวคอย และการดึงข้อมูลออกจากส่วนหน้าสุดของแถวคอย เข้าออกในลักษณะการเข้าก่อนออกก่อน (First In First Out: FIFO) ในโครงสร้างข้อมูลลักษณะเข้าก่อนออกก่อนนี้ ข้อมูลแรกสุดที่ถูกเพิ่มเข้าไปในแถวคอยจะเป็นข้อมูลแรกที่ถูกดึงออก ซึ่งก็เท่ากับว่า ความจำเป็นที่ว่า เมื่อมีข้อมูลหนึ่งถูกเพิ่มเข้ามาแล้ว ข้อมูลที่ถูกเพิ่มก่อนหน้านี้ทั้งหมดจะต้องถูกดึงออกก่อนที่ข้อมูลใหม่จะถูกใช้งาน คล้ายกับการเข้าแถวซื้อของในชีวิตประจำวัน



Queues

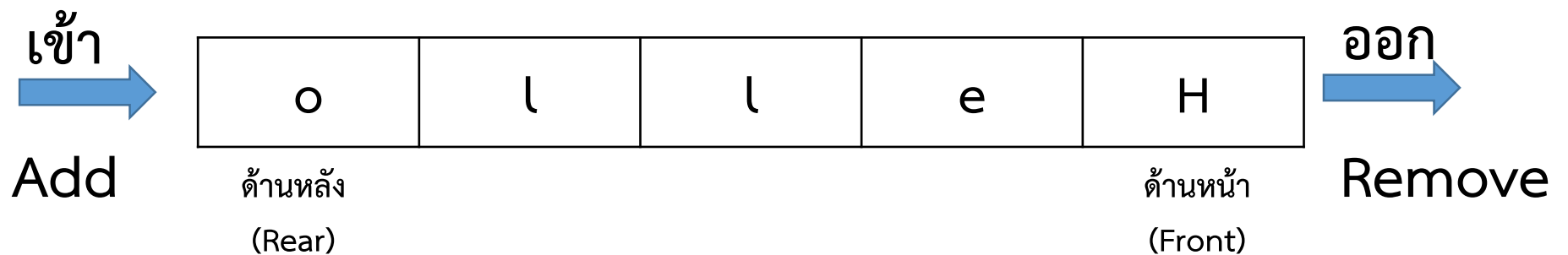
- ข้อมูลในแถวคอยเป็นแบบ Homogenous
- ข้อมูลเข้าและออกคนละด้าน
- ข้อมูลเข้าทางด้านหลัง
- ข้อมูลออกทางด้านหน้า
- เข้าก่อน ออกก่อน ตามลำดับ
- ความสัมพันธ์เป็นเชิงเส้น
- เข้าถึงข้อมูลภายในแถวคอยโดยตรงไม่ได้



Queues

Operation ของแถวคอย

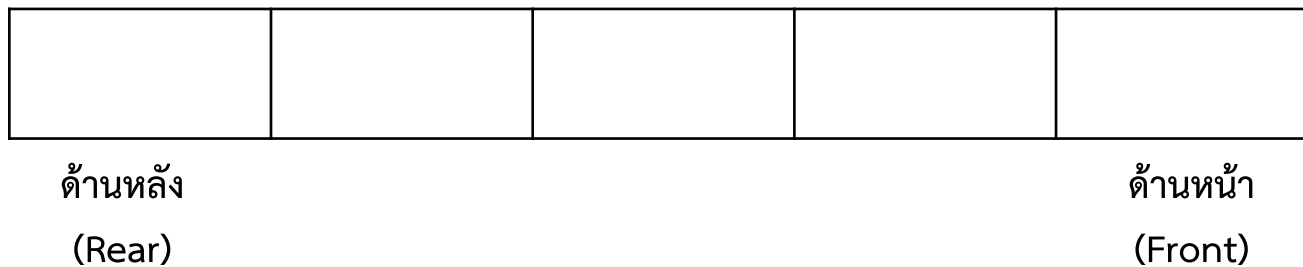
- สร้างแถวคอย
- เพิ่มข้อมูล (Add)
- ลบข้อมูล (Remove)
- อ่านข้อมูลโดยไม่ลบ (peek)
- isEmpty()
- isFull()
- ตรวจสอบขนาด



Queues

เพิ่มข้อมูล (Add, Enqueue, Insert, Append)

- 1 ตรวจสอบพื้นที่ว่าง หากไม่เพียงพอให้รายงาน overflow
- 2 เพิ่มข้อมูลเข้าไปทางด้านหลัง

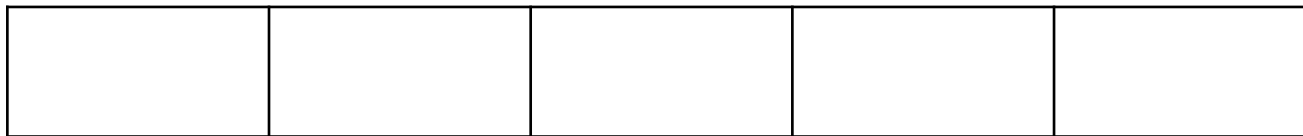


Queues

เพิ่มข้อมูล (Add, Enqueue, Insert, Append)

- 1 ตรวจสอบพื้นที่ว่าง หากไม่เพียงพอให้รายงาน overflow
- 2 เพิ่มข้อมูลเข้าไปทางด้านหลัง

Add('H')



ด้านหลัง
(Rear)

ด้านหน้า
(Front)

Queues

เพิ่มข้อมูล (Add, Enqueue, Insert, Append)

- 1 ตรวจสอบพื้นที่ว่าง หากไม่เพียงพอให้รายงาน overflow
- 2 เพิ่มข้อมูลเข้าไปทางด้านหลัง

Add('e')

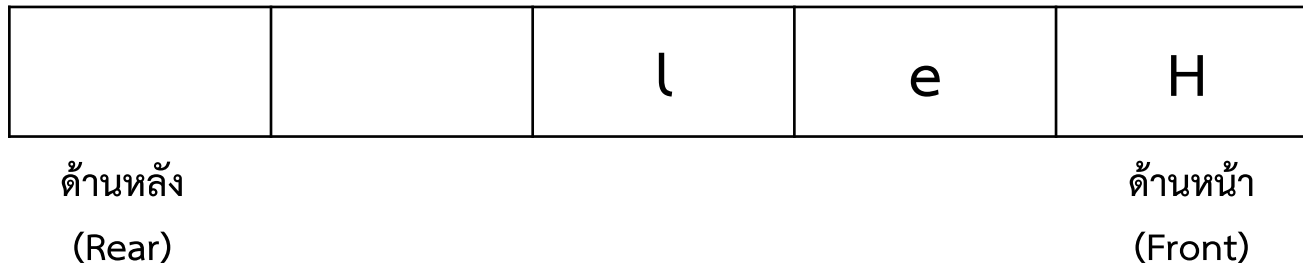


Queues

เพิ่มข้อมูล (Add, Enqueue, Insert, Append)

- 1 ตรวจสอบพื้นที่ว่าง หากไม่เพียงพอให้รายงาน overflow
- 2 เพิ่มข้อมูลเข้าไปทางด้านหลัง

Add('l')



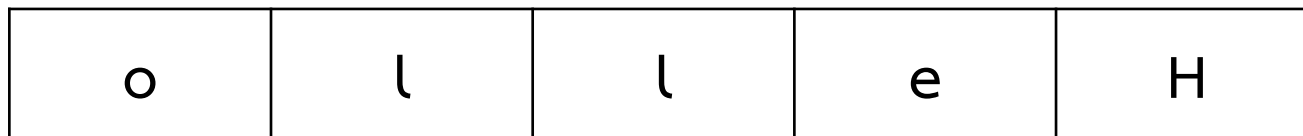
Queues

ลบข้อมูล (Remove, Dequeue, Delete, Serve)

1 ลบข้อมูลทางด้านหน้าออกไป

2 หากแถวคอยไม่มีข้อมูลให้รายงาน underflow

c=Remove()



ด้านหลัง
(Rear)

ด้านหน้า
(Front)

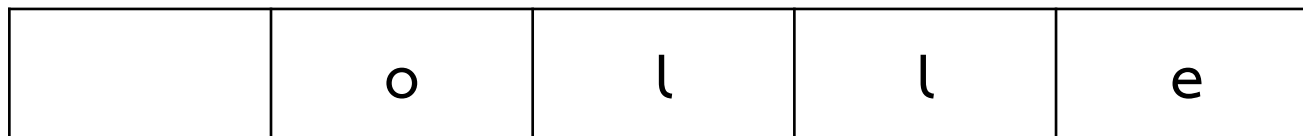
Queues

ลบข้อมูล (Remove, Dequeue, Delete, Serve)

1 ลบข้อมูลทางด้านหน้าออกไป

2 หากแถวคอยไม่มีข้อมูลให้รายงาน underflow

`c=Remove()`



ด้านหลัง
(Rear)

ด้านหน้า
(Front)

`c= H`

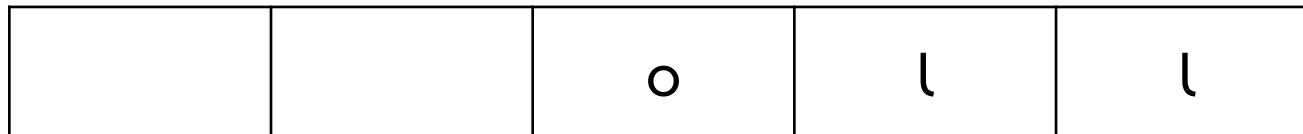
Queues

ลบข้อมูล (Remove, Dequeue, Delete, Serve)

1 ลบข้อมูลทางด้านหน้าออกไป

2 หากแถวคอยไม่มีข้อมูลให้รายงาน underflow

`c=Remove()`



ด้านหลัง
(Rear)

ด้านหน้า
(Front)

`c = e`

การสร้างแถวคอย

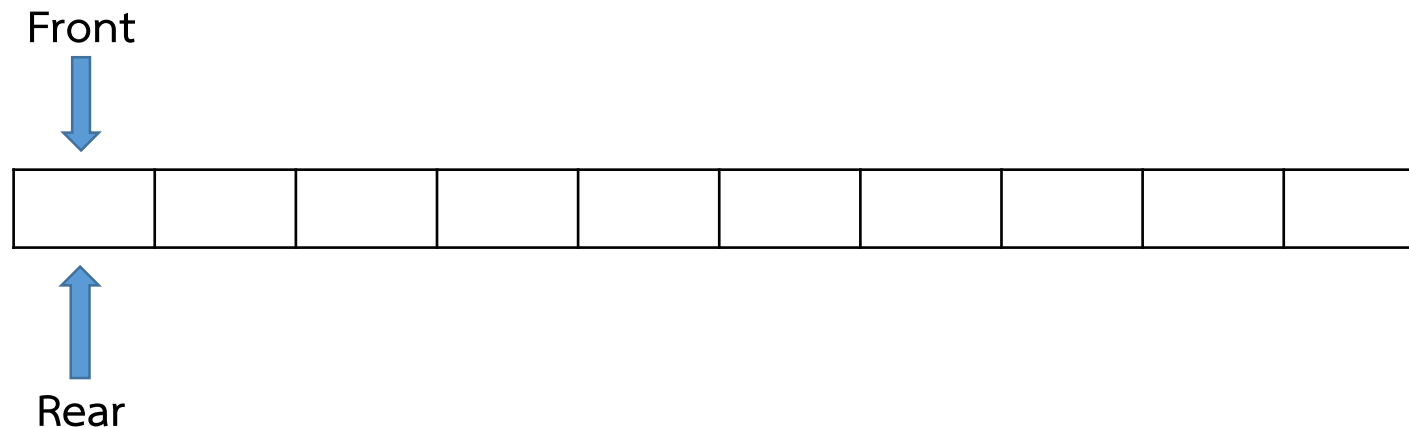
เวลาในการนำข้อมูล เข้า – ออก ต้องเป็น $O(1)$
ทำได้ไหม ?

1 ใช้ Array

2 ใช้ Linked list

การสร้างแถวคอยด้วย Array

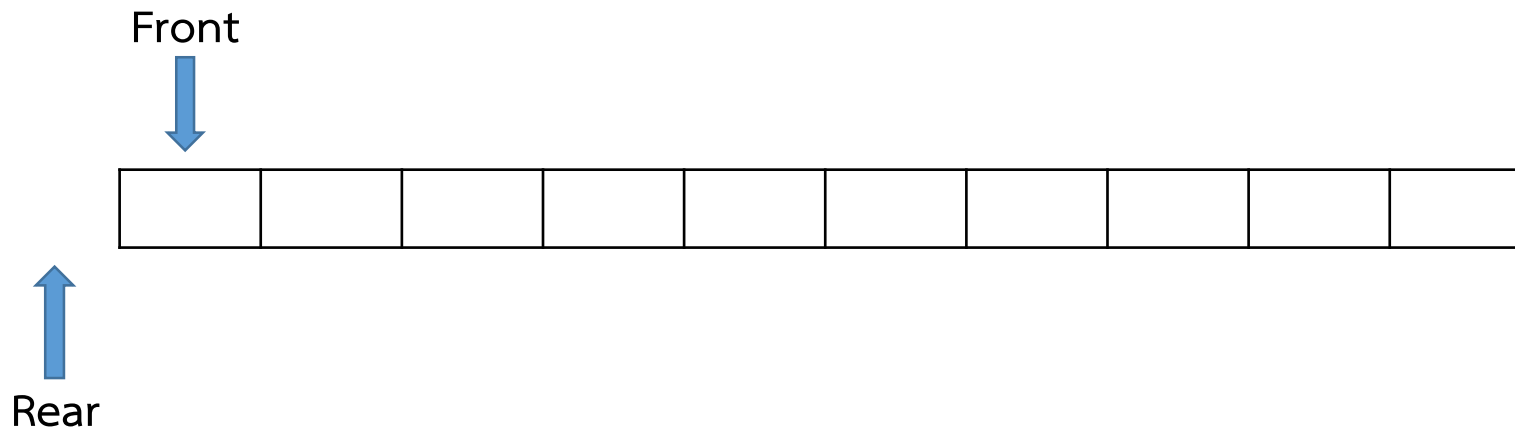
- 1 สร้าง Array ขึ้นมาเพื่อใช้สำหรับเก็บข้อมูลภายในแถวคอย
- 2 สร้างตัวแปรขึ้นมาอีก 2 ตัวเพื่อเก็บตำแหน่งหน้าและหลัง (Front , Rear)
- 3 การ add และ remove จะกระทำโดยการเปลี่ยนตำแหน่งของ Front และ Rear



การเพิ่มข้อมูล

- 1) ตำแหน่งเริ่มต้นของ Rear เป็น -1 เมื่อไม่มีข้อมูล
- 2) เพิ่มตำแหน่งของ Rear ไป 1 และทำการใส่ข้อมูลในตำแหน่ง Rear

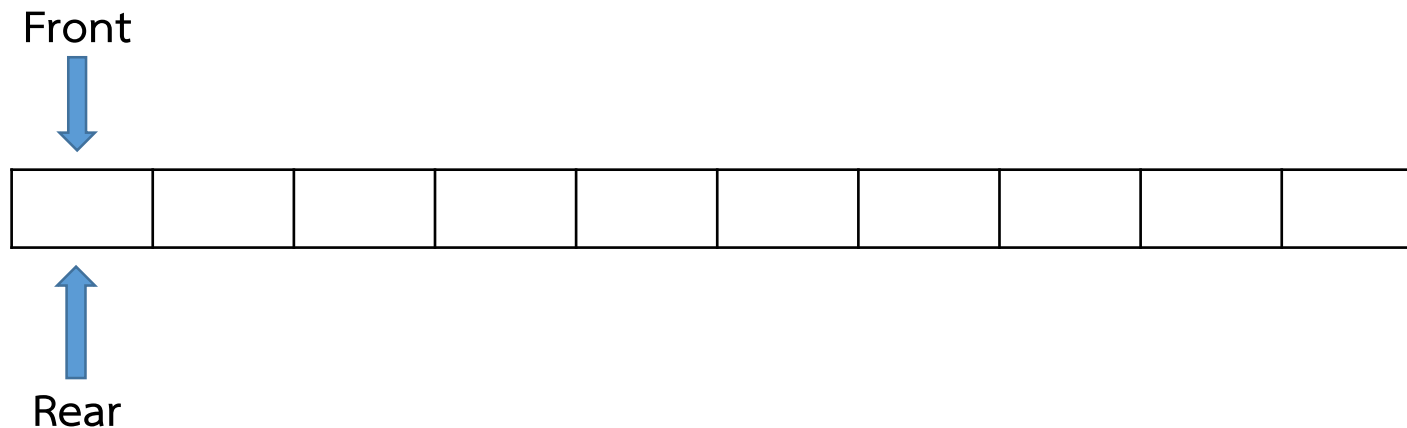
add('H')



การเพิ่มข้อมูล

- 1) ตำแหน่งเริ่มต้นของ Rear เป็น -1 เมื่อไม่มีข้อมูล
- 2) เพิ่มตำแหน่งของ Rear ไป 1 และทำการใส่ข้อมูลในตำแหน่ง Rear

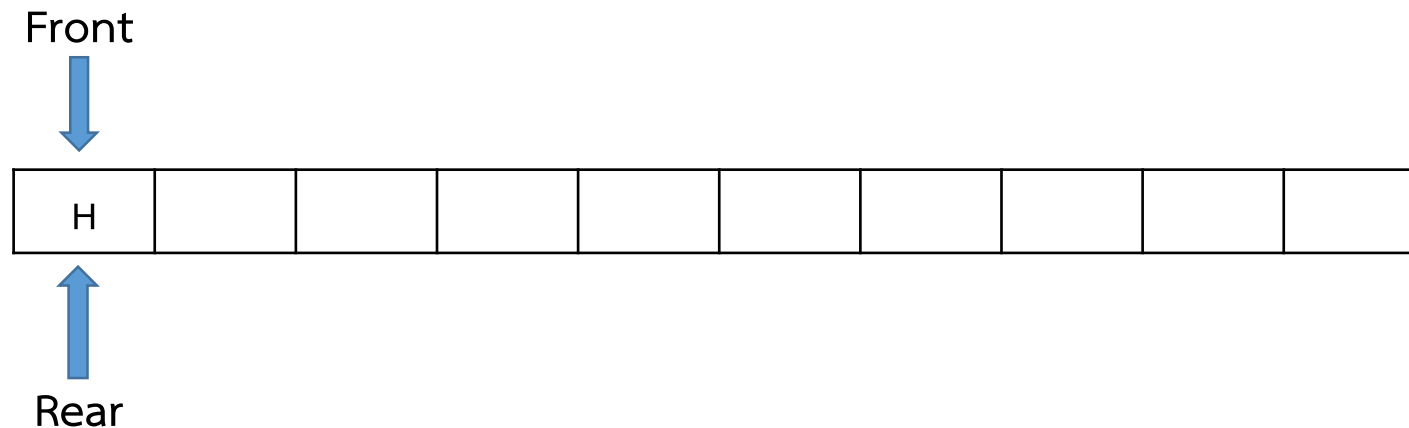
add('H')



การเพิ่มข้อมูล

- 1) ตำแหน่งเริ่มต้นของ Rear เป็น -1 เมื่อไม่มีข้อมูล
- 2) เพิ่มตำแหน่งของ Rear ไป 1 และทำการใส่ข้อมูลในตำแหน่ง Rear

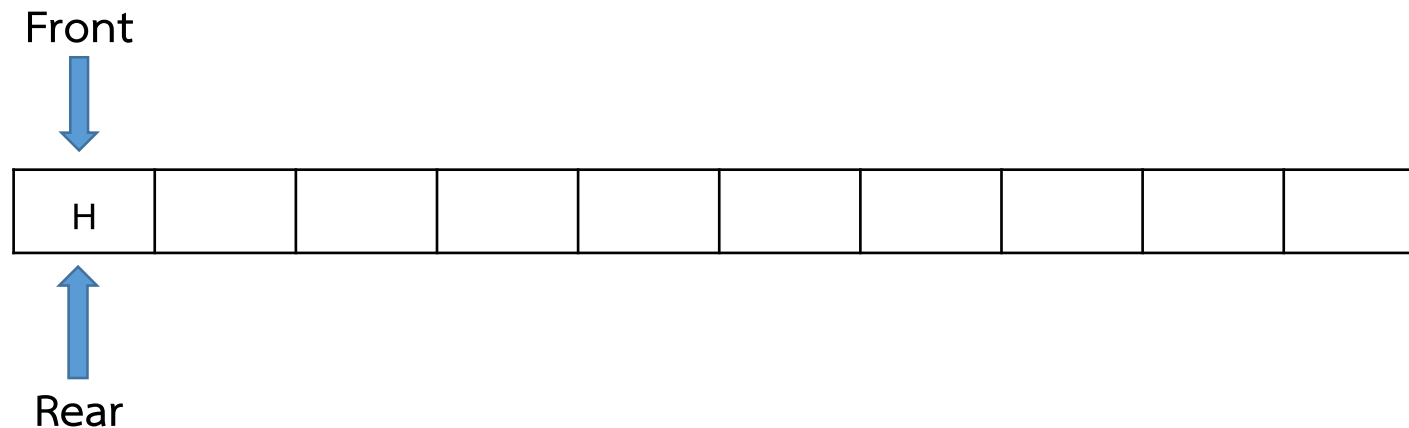
add('H')



การเพิ่มข้อมูล

- 1) ตำแหน่งเริ่มต้นของ Rear เป็น -1 เมื่อไม่มีข้อมูล
- 2) เพิ่มตำแหน่งของ Rear ไป 1 และทำการใส่ข้อมูลในตำแหน่ง Rear

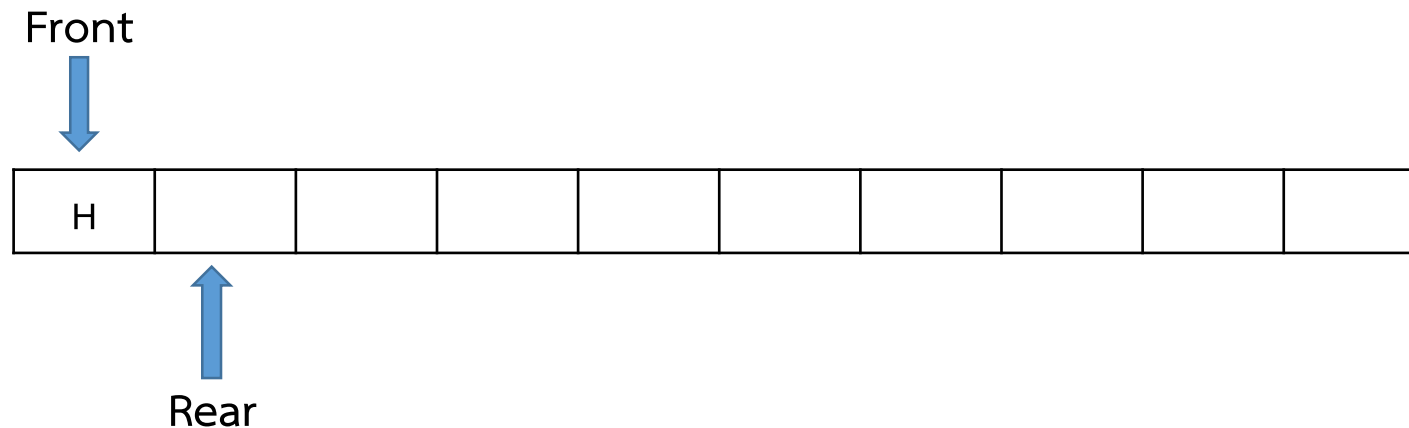
add('e')



การเพิ่มข้อมูล

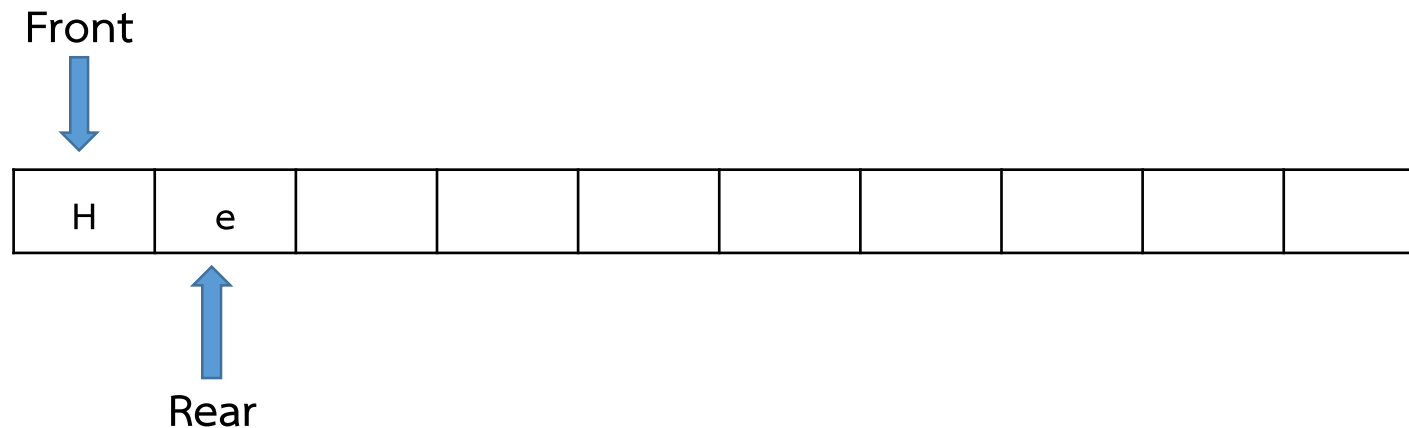
- 1) ตำแหน่งเริ่มต้นของ Rear เป็น -1 เมื่อไม่มีข้อมูล
- 2) เพิ่มตำแหน่งของ Rear ไป 1 และทำการใส่ข้อมูลในตำแหน่ง Rear

add('e')



การเพิ่มข้อมูล

- 1) ตำแหน่งเริ่มต้นของ Rear เป็น -1 เมื่อไม่มีข้อมูล
- 2) เพิ่มตำแหน่งของ Rear ไป 1 และทำการใส่ข้อมูลในตำแหน่ง Rear

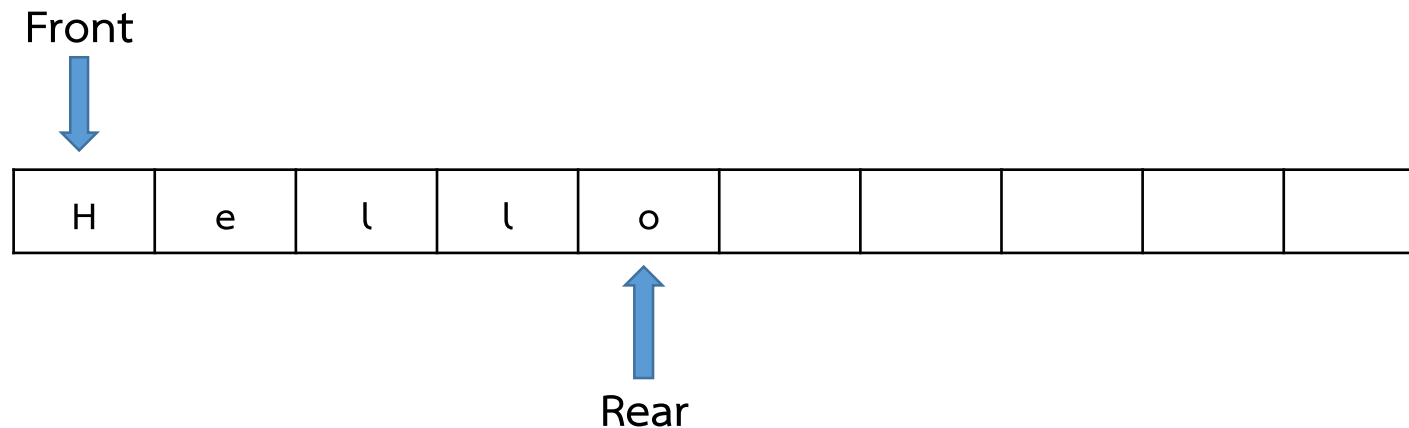


การลบข้อมูล

- 1) อ่านข้อมูลจากเพิ่มตำแหน่งของ front
- 2) เลื่อนตำแหน่ง front ไป 1

`c = remove()`

`c = H`

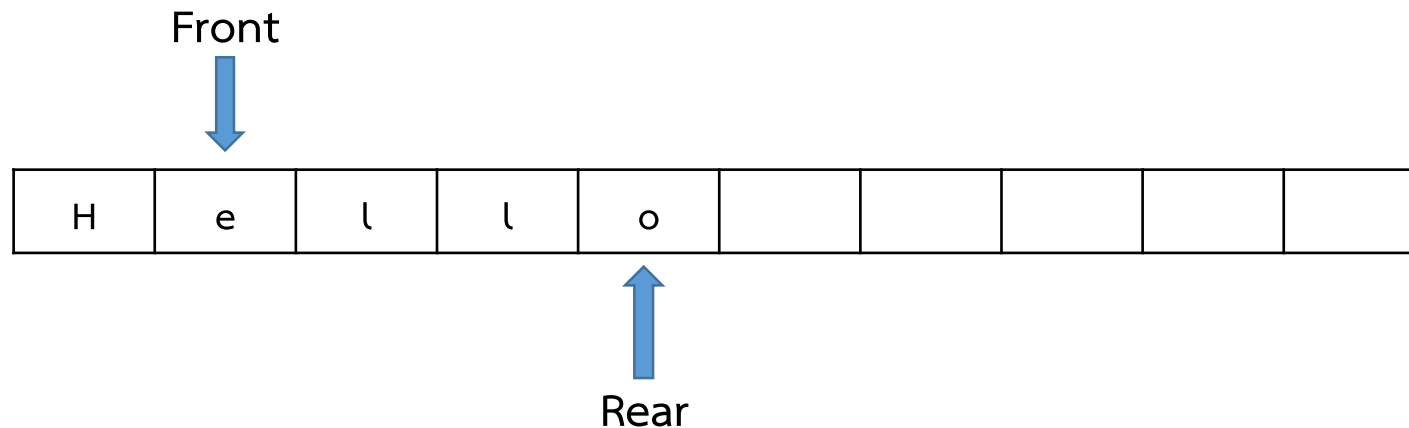


การลบข้อมูล

- 1) อ่านข้อมูลจากเพิ่มตำแหน่งของ front
- 2) เลื่อนตำแหน่ง front ไป 1

`c = remove()`

`c = e`

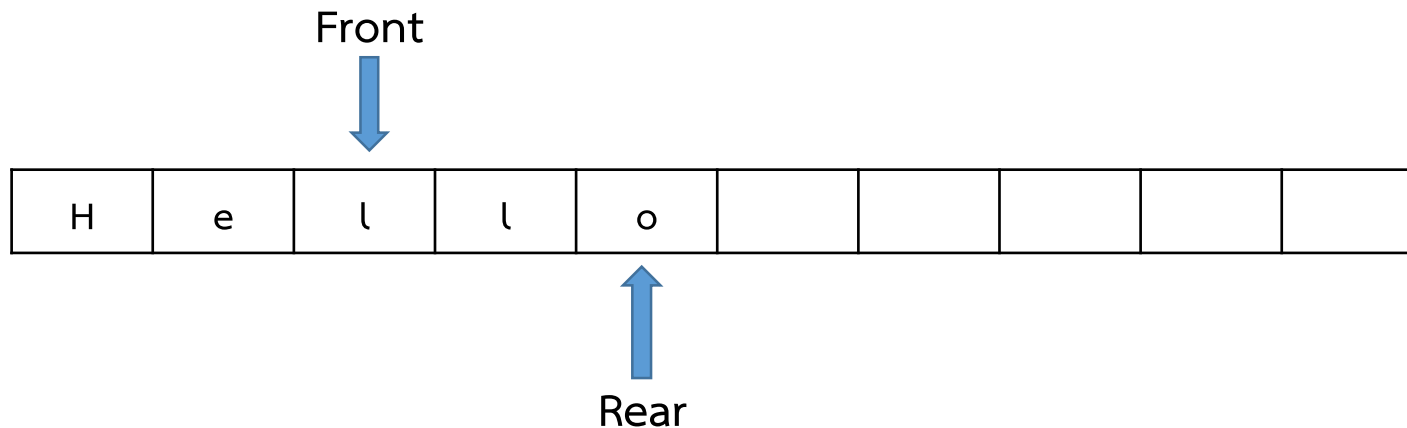


การลบข้อมูล

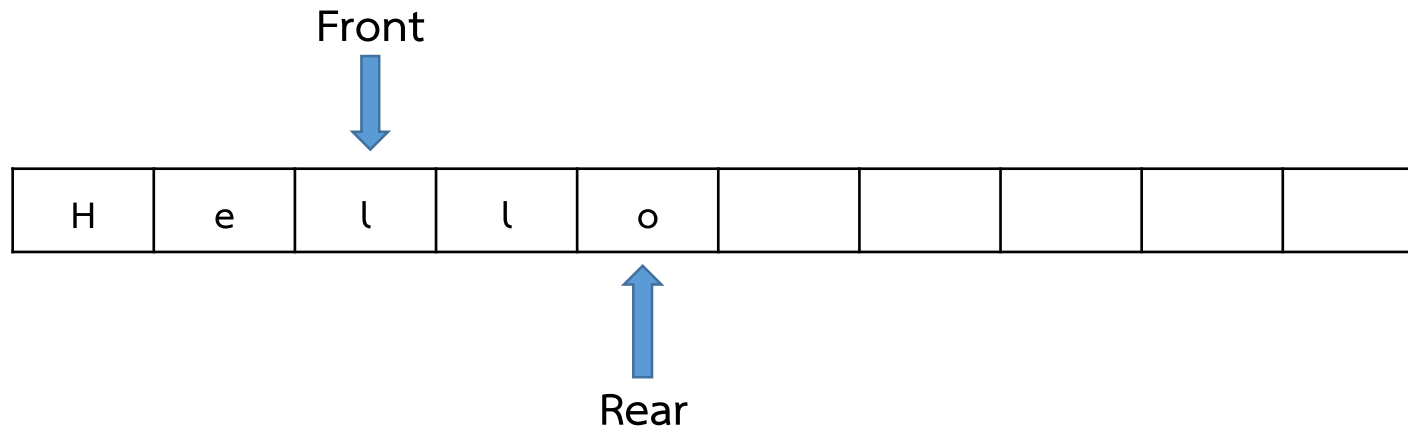
- 1) อ่านข้อมูลจากเพิ่มตำแหน่งของ front
- 2) เลื่อนตำแหน่ง front ไป 1

`c = remove()`

`c = l`



Queues: Array Implementation

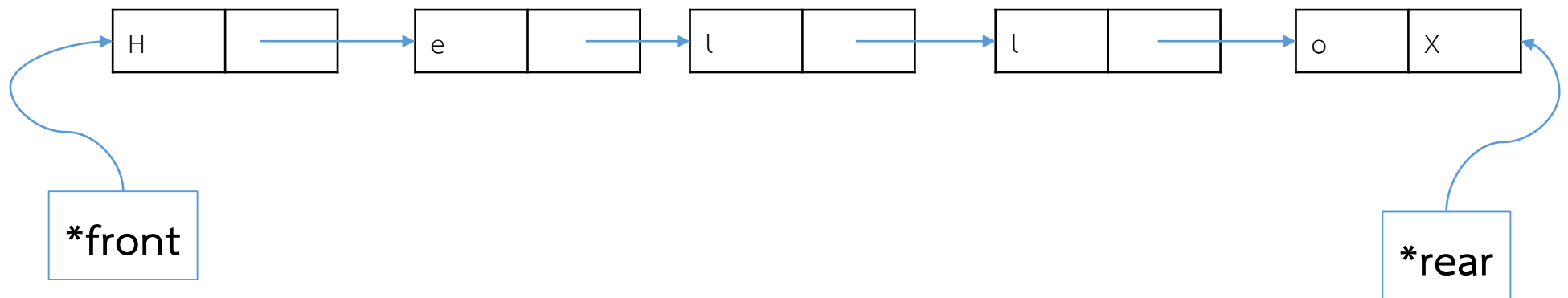


- ความซับซ้อนของการเพิ่มและลบ เป็น $O(1)$
- ขนาดของแถวคอยเปลี่ยนแปลงไม่ได้
- ตำแหน่งของข้อมูลที่ถูกลบไปแล้ว ไม่สามารถนำกลับมาใช้ซ้ำได้
- หาก Rear เลื่อนไปถึงตำแหน่งขวาสุด จะเกิดการ overflow
- หาก front มากกว่า Rear จะเกิด underflow
- ปัญหา overflow แก้ได้ด้วยการเลื่อนข้อมูลไปทางด้านซ้าย แต่เสียเวลา $O(N)$

Queues: Linked list Implementation

การสร้างแถวคอยด้วย Linked List

- 1 ข้อมูลของแถวคอยจะถูกบันทึกในรูปแบบ node
- 2 ใช้ pointer จำนวน 2 ตัวเพื่อระบุตำแหน่ง front และ rear
- 3 การ add คือการเพิ่ม node ต่อจาก rear (tail)
- 4 การ remove คือการลบ node แรก



Queues: Linked list Implementation

การเพิ่มข้อมูล (add)

1 สร้าง node ใหม่

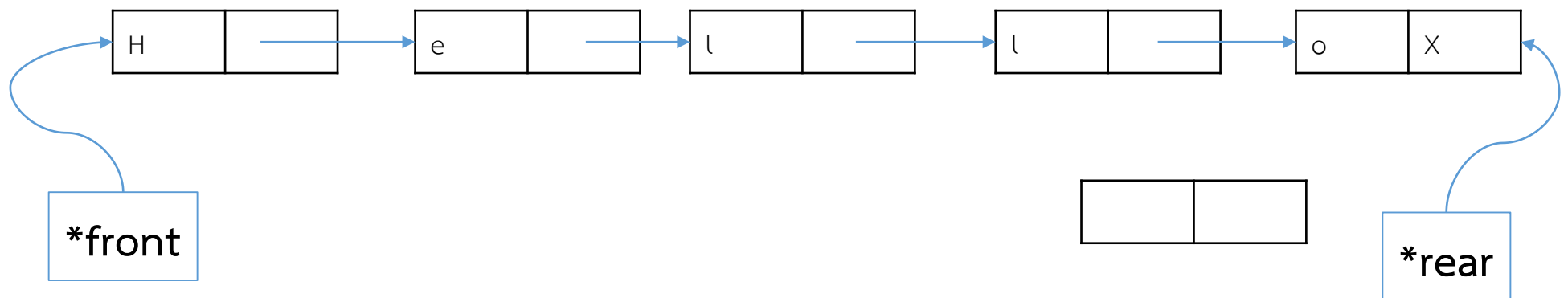
2 ใส่ข้อมูลลงใน node ใหม่

3 กำหนด next node เป็น null

4 กำหนด next node ของ node ที่ rear pointer ชี้อยู่ ให้ชี้มาที่ node ใหม่

4 กำหนด rear pointer ชี้มาที่ node ใหม่

add('W')



Queues: Linked list Implementation

การเพิ่มข้อมูล (add)

1 สร้าง node ใหม่

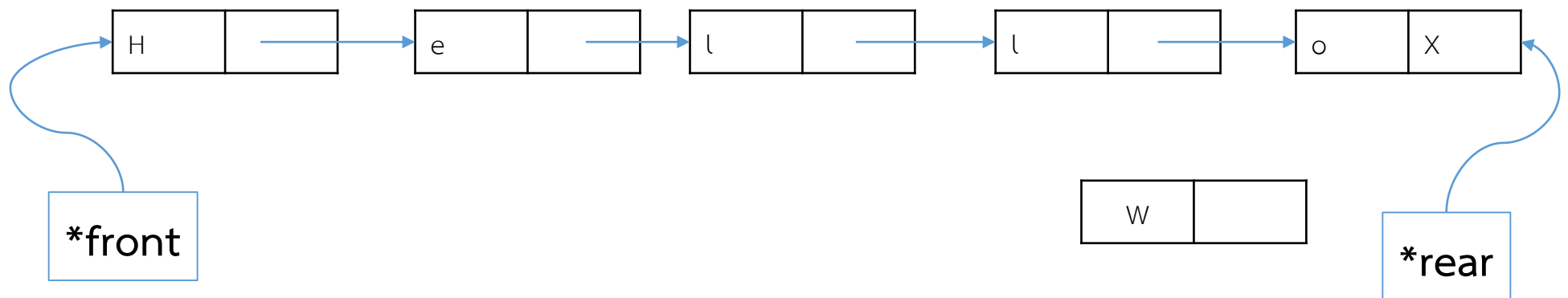
2 ใส่ข้อมูลลงใน node ใหม่

3 กำหนด next node เป็น null

4 กำหนด next node ของ node ที่ rear pointer ชี้อยู่ ให้ชี้มาที่ node ใหม่

4 กำหนด rear pointer ชี้มาที่ node ใหม่

add('W')



Queues: Linked list Implementation

การเพิ่มข้อมูล (add)

1 สร้าง node ใหม่

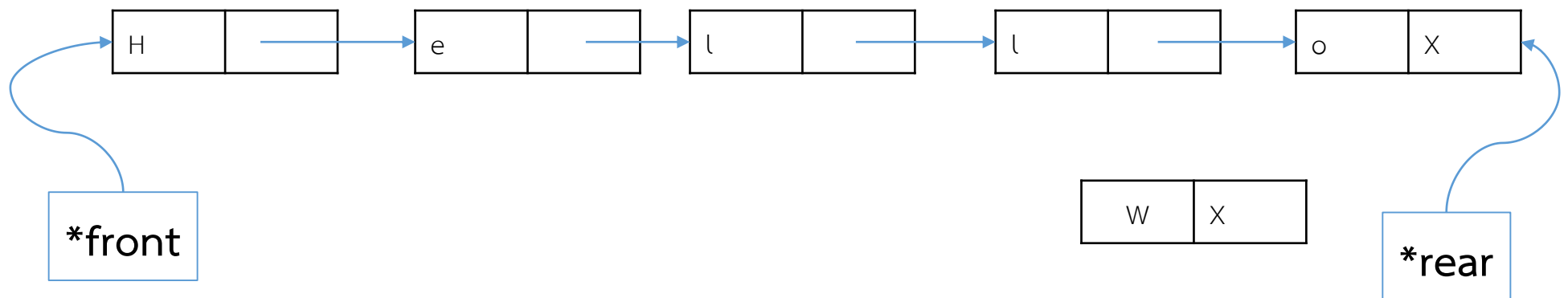
2 ใส่ข้อมูลลงใน node ใหม่

3 กำหนด next node เป็น null

4 กำหนด next node ของ node ที่ rear pointer ชี้อยู่ ให้ชี้มาที่ node ใหม่

4 กำหนด rear pointer ชี้มาที่ node ใหม่

add('W')



Queues: Linked list Implementation

การเพิ่มข้อมูล (add)

1 สร้าง node ใหม่

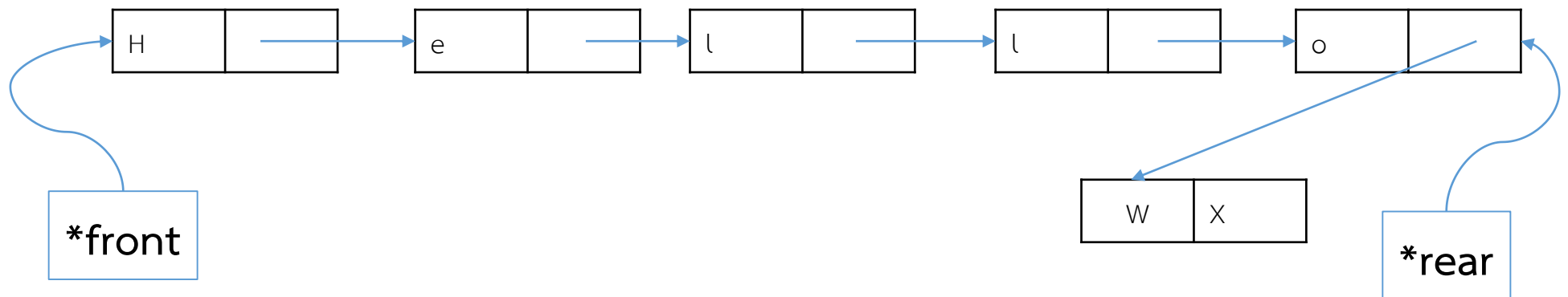
2 ใส่ข้อมูลลงใน node ใหม่

3 กำหนด next node เป็น null

4 กำหนด next node ของ node ที่ rear pointer ชี้อยู่ ให้ชี้มาที่ node ใหม่

4 กำหนด rear pointer ชี้มาที่ node ใหม่

add('W')



Queues: Linked list Implementation

การเพิ่มข้อมูล (add)

1 สร้าง node ใหม่

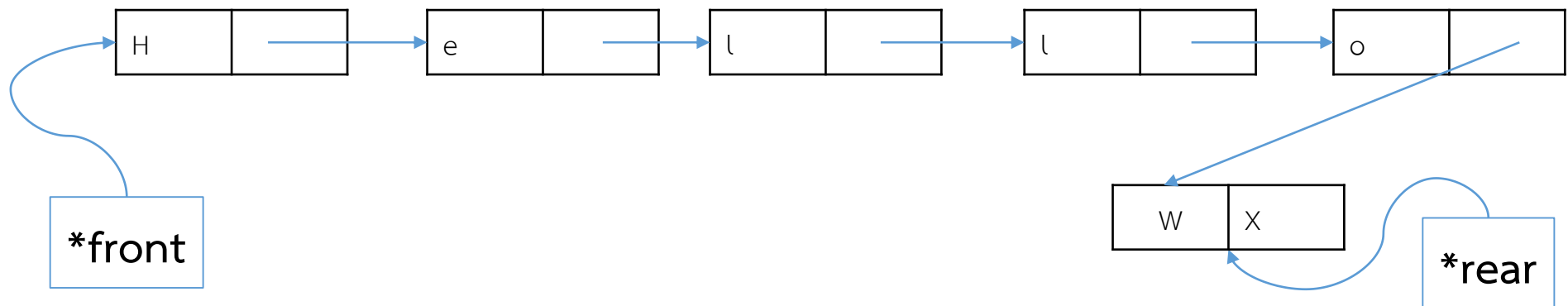
2 ใส่ข้อมูลลงใน node ใหม่

3 กำหนด next node เป็น null

4 กำหนด next node ของ node ที่ rear pointer ชี้อยู่ ให้ชี้มาที่ node ใหม่

4 กำหนด rear pointer ชี้มาที่ node ใหม่

add('W')

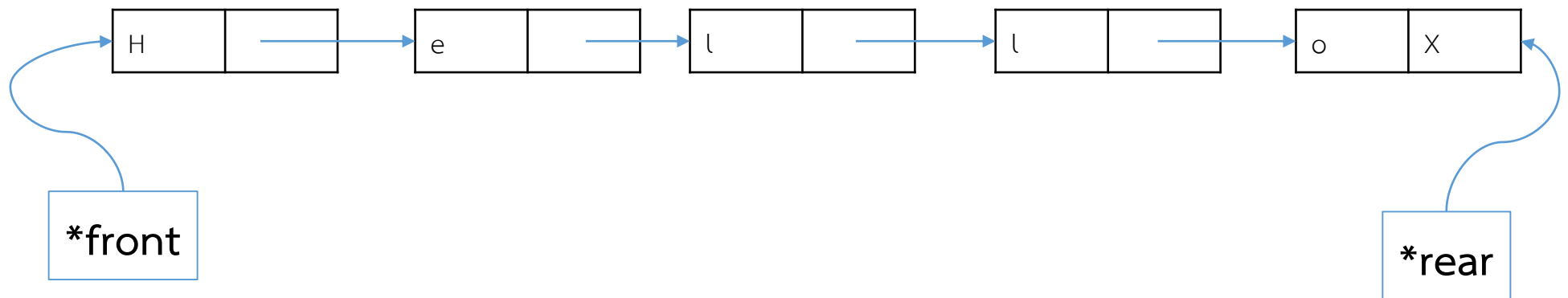


Queues: Linked list Implementation

การลบข้อมูล (remove)

- 1 สร้าง pointer ชั่วคราวเพื่อชี้ไปตำแหน่งเดียวกับ front
- 2 สร้าง node ชั่วคราวขึ้นมาและสำเนาข้อมูลของ node แรกมาไว้ที่นี้
- 3 เปลี่ยนตำแหน่ง front pointer มาชี้ node ถัดไป
- 4 ลบ node ที่ถูกชี้โดย pointer ชั่วคราวออกไป
- 4 คำนวณค่าของ node ชั่วคราวที่บันทึกไว้ออกมา

remove()

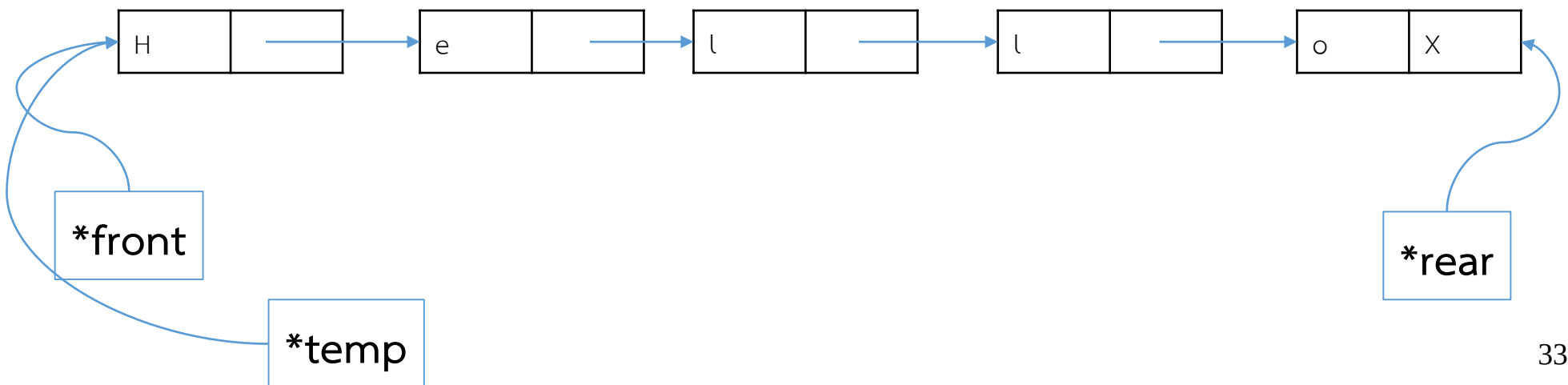


Queues: Linked list Implementation

การลบข้อมูล (remove)

- 1 สร้าง pointer ชั่วคราวเพื่อชี้ไปตำแหน่งเดียวกับ front
- 2 สร้าง node ชั่วคราวขึ้นมาและสำเนาข้อมูลของ node แรกมาไว้ที่นี้
- 3 เปลี่ยนตำแหน่ง front pointer มาชี้ node ถัดไป
- 4 ลบ node ที่ถูกชี้โดย pointer ชั่วคราวออกไป
- 4 คำนวณค่าของ node ชั่วคราวที่บันทึกไว้ออกมา

remove()

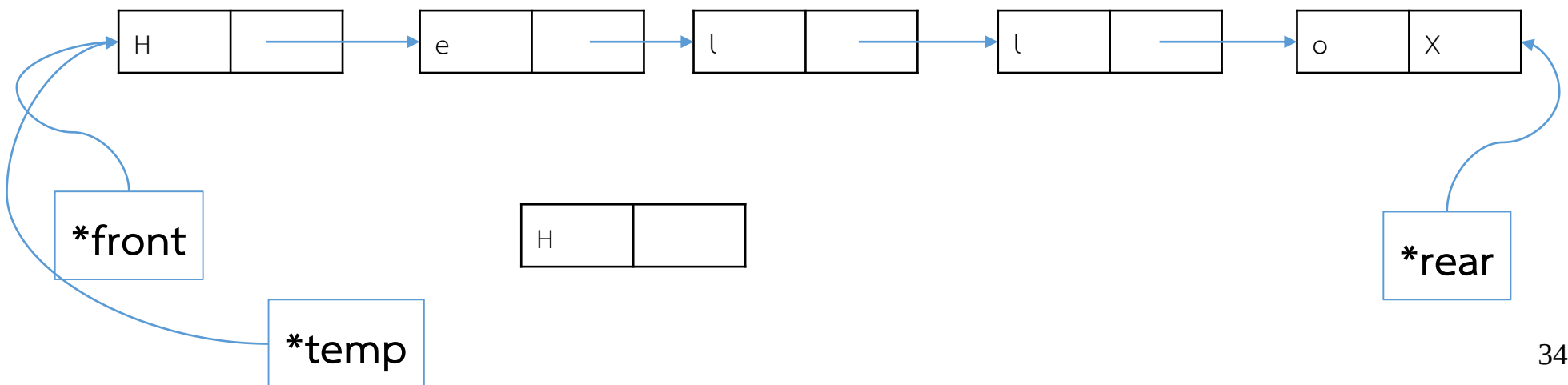


Queues: Linked list Implementation

การลบข้อมูล (remove)

- 1 สร้าง pointer ชั่วคราวเพื่อชี้ไปตำแหน่งเดียวกับ front
- 2 สร้าง node ชั่วคราวขึ้นมาและสำเนาข้อมูลของ node แรกมาไว้ที่นี่
- 3 เปลี่ยนตำแหน่ง front pointer มาชี้ node ถัดไป
- 4 ลบ node ที่ถูกชี้โดย pointer ชั่วคราวออกไป
- 4 คำนวณค่าของ node ชั่วคราวที่บันทึกไว้ออกมา

remove()

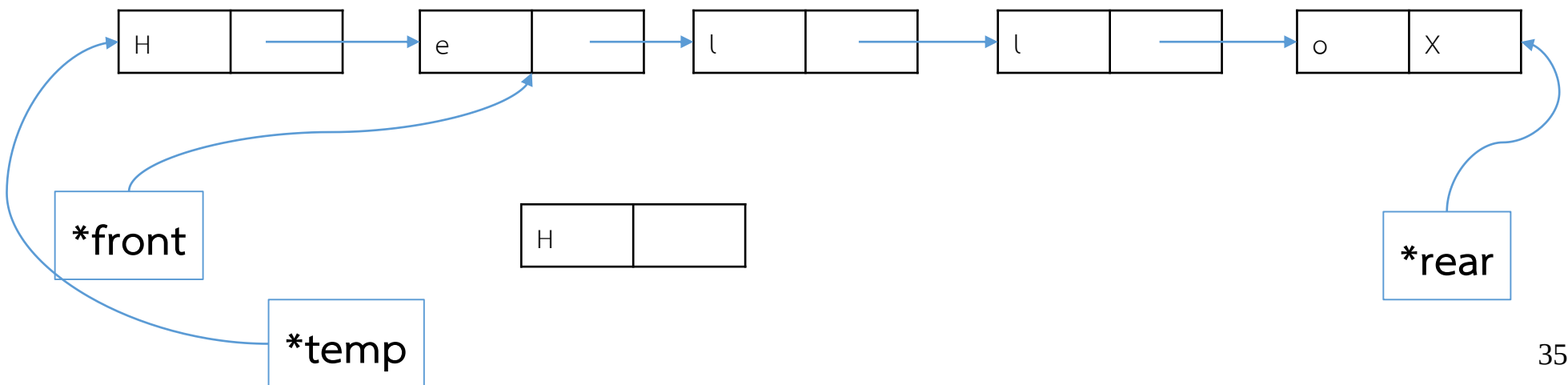


Queues: Linked list Implementation

การลบข้อมูล (remove)

- 1 สร้าง pointer ชั่วคราวเพื่อชี้ไปตำแหน่งเดียวกับ front
- 2 สร้าง node ชั่วคราวขึ้นมาและสำเนาข้อมูลของ node แรกมาไว้ที่นี่
- 3 เปลี่ยนตำแหน่ง front pointer มาชี้ node ถัดไป
- 4 ลบ node ที่ถูกชี้โดย pointer ชั่วคราวออกไป
- 4 คำนวณค่าของ node ชั่วคราวที่บันทึกไว้ออกมา

remove()

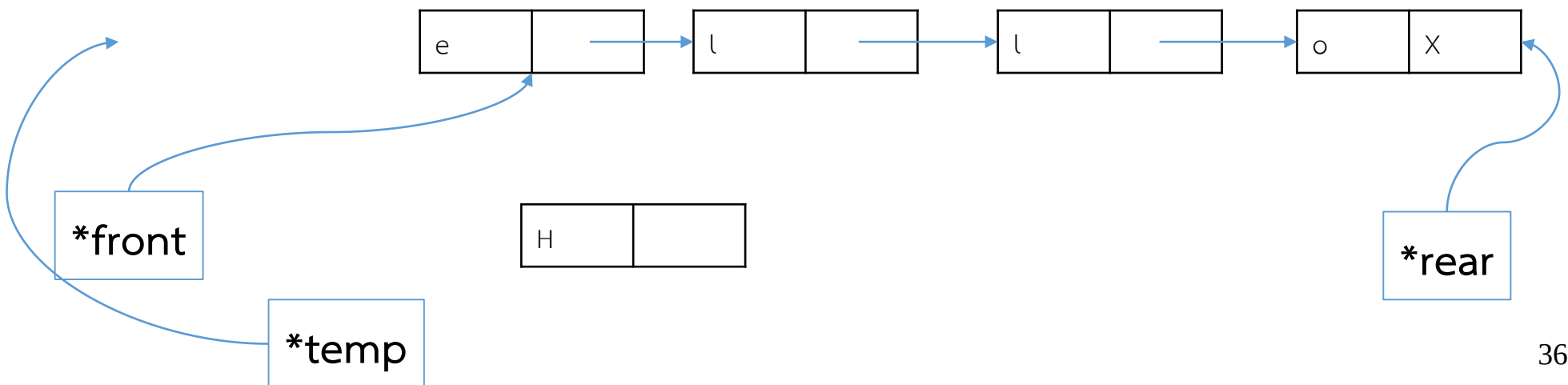


Queues: Linked list Implementation

การลบข้อมูล (remove)

- 1 สร้าง pointer ชั่วคราวเพื่อชี้ไปตำแหน่งเดียวกับ front
- 2 สร้าง node ชั่วคราวขึ้นมาและสำเนาข้อมูลของ node แรกมาไว้ที่นี่
- 3 เปลี่ยนตำแหน่ง front pointer มาชี้ node ถัดไป
- 4 ลบ node ที่ถูกชี้โดย pointer ชั่วคราวออกไป
- 4 คำนวณค่าของ node ชั่วคราวที่บันทึกไว้ออกมา

remove()

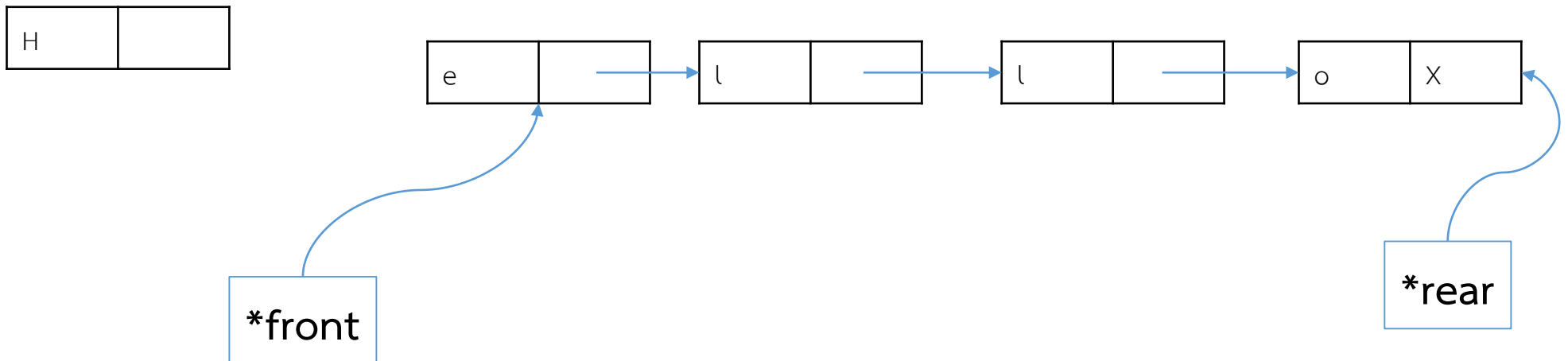


Queues: Linked list Implementation

การลบข้อมูล (remove)

- 1 สร้าง pointer ชั่วคราวเพื่อชี้ไปตำแหน่งเดียวกับ front
- 2 สร้าง node ชั่วคราวขึ้นมาและสำเนาข้อมูลของ node แรกมาไว้ที่นี่
- 3 เปลี่ยนตำแหน่ง front pointer มาชี้ node ถัดไป
- 4 ลบ node ที่ถูกชี้โดย pointer ชั่วคราวออกไป
- 4 คำนวณค่าของ node ชั่วคราวที่บันทึกไว้ออกมา

remove()



Queues: Linked list Implementation

การสร้างแถวคอยด้วยน Linked List

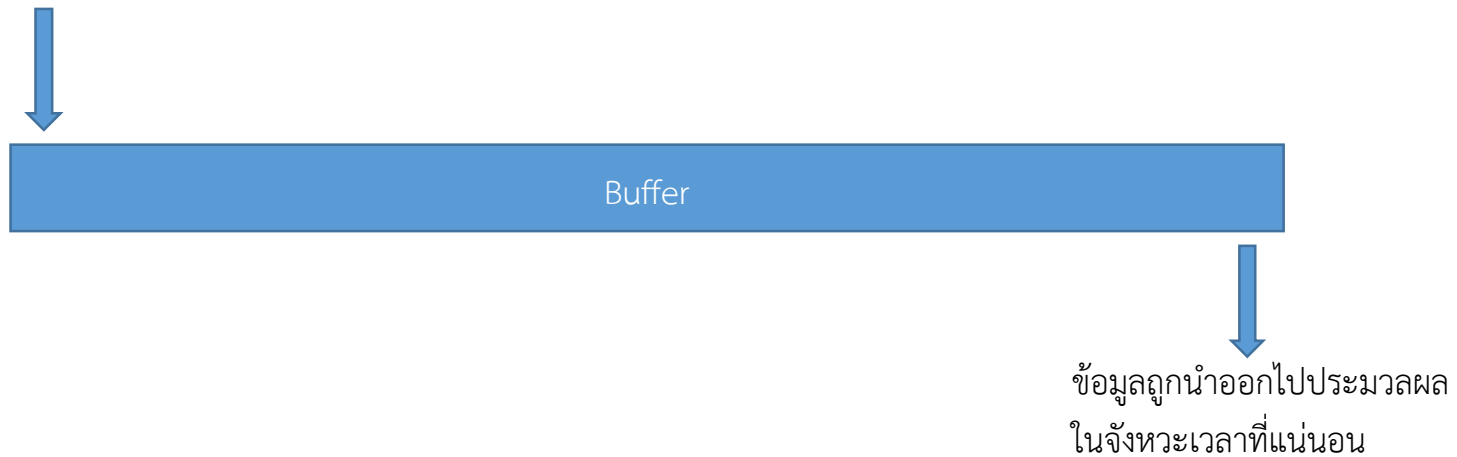
- เปลี่ยนแปลงขนาดได้
- นำตำแหน่งหน่วยความจำมาใช้ใหม่ได้
- มีความคล่องตัว

Queues:

แถวลายมักนิยมใช้เพื่อเป็น Buffer ในการสื่อสารระหว่างอุปกรณ์ที่มีความเร็วไม่เท่ากัน

- การรับตัวอักษรจากแป้นพิมพ์
- การประมวลผลคำสั่งของโปรแกรม
- การรับข้อมูลของเครื่องพิมพ์
- การส่งข้อมูลผ่านระบบเครือข่าย

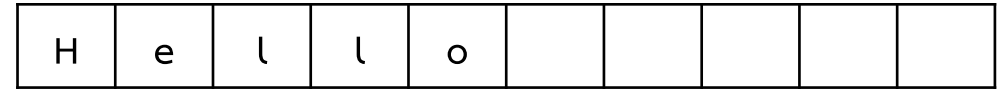
ข้อมูลเข้ามาเต็มเรื่อย ๆ คาดเตาจังหวะเวลาไม่ได้



ถ้าข้อมูลถูกนำออกไปประมวลผลไม่ทันจะเกิด buffer overflow

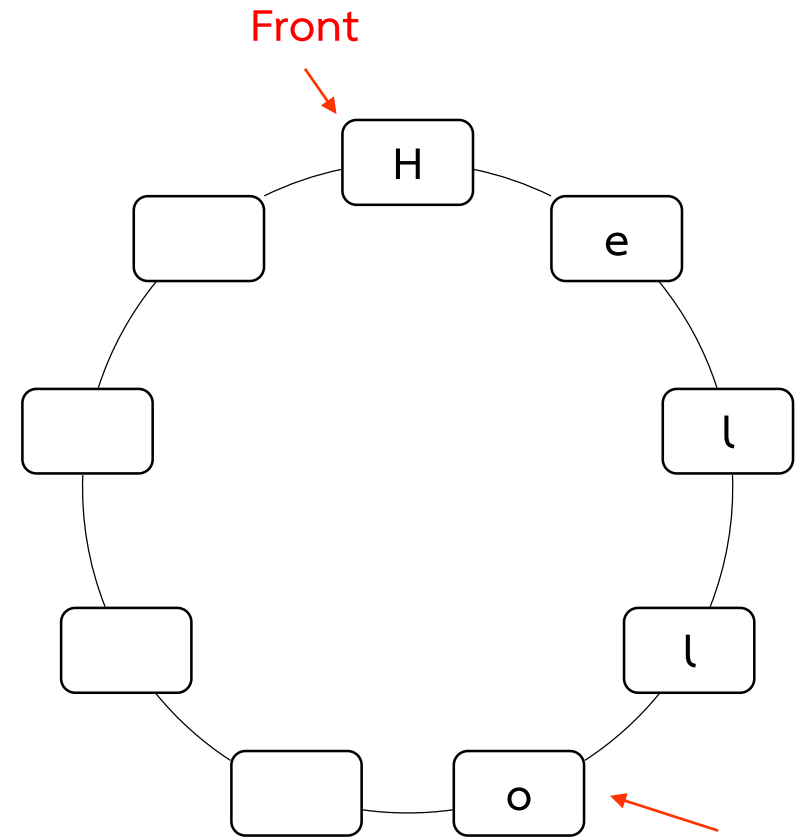
Queues: Circular Queues

Circular Queues คือแถวคอยที่เป็นวงแหวน



Linear Queue

- ใช้งานได้หลากหลายกว่า Linear Queues
- นิยมใช้จัดตารางงาน
- ใช้สร้าง Buffer สำหรับพักข้อมูลในอุปกรณ์เครือข่าย

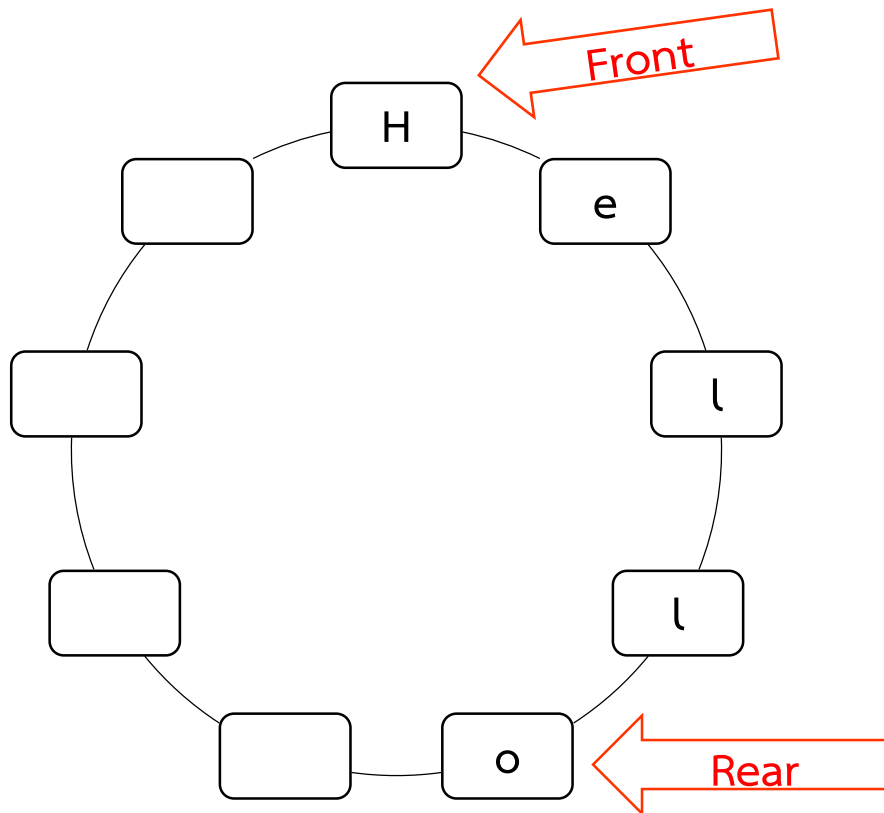


Circular Queue

Queues: Circular Queues

Circular Queues

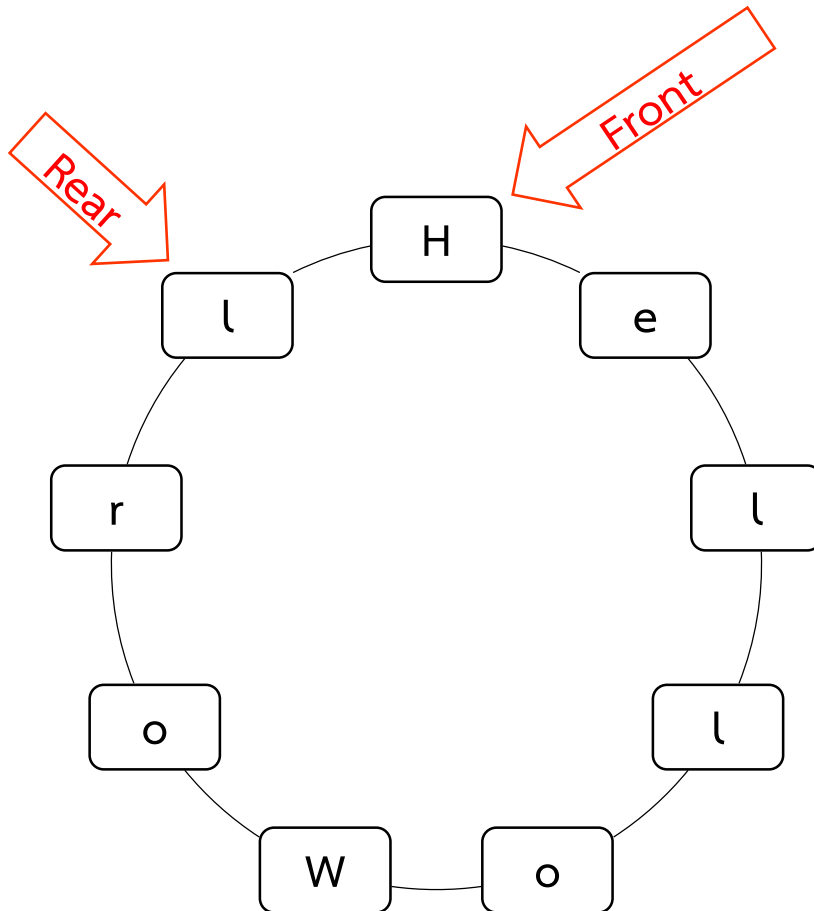
- Operation ของ Circular Queue จะเหมือนกับ Linear Queue
- แต่การเขียนโปรแกรมจะแตกต่างกันเล็กน้อยในส่วนของ `isFull()` และ `Add()`
- สามารถเกิด overflow ขึ้นได้



Queues: Circular Queues

Circular Queues

- Operation ของ Circular Queue จะเหมือนกับ Linear Queue
- แต่การเขียนโปรแกรมจะแตกต่างกันเล็กน้อยในส่วนของ `isFull()` และ `Add()`
- สามารถเกิด overflow ขึ้นได้

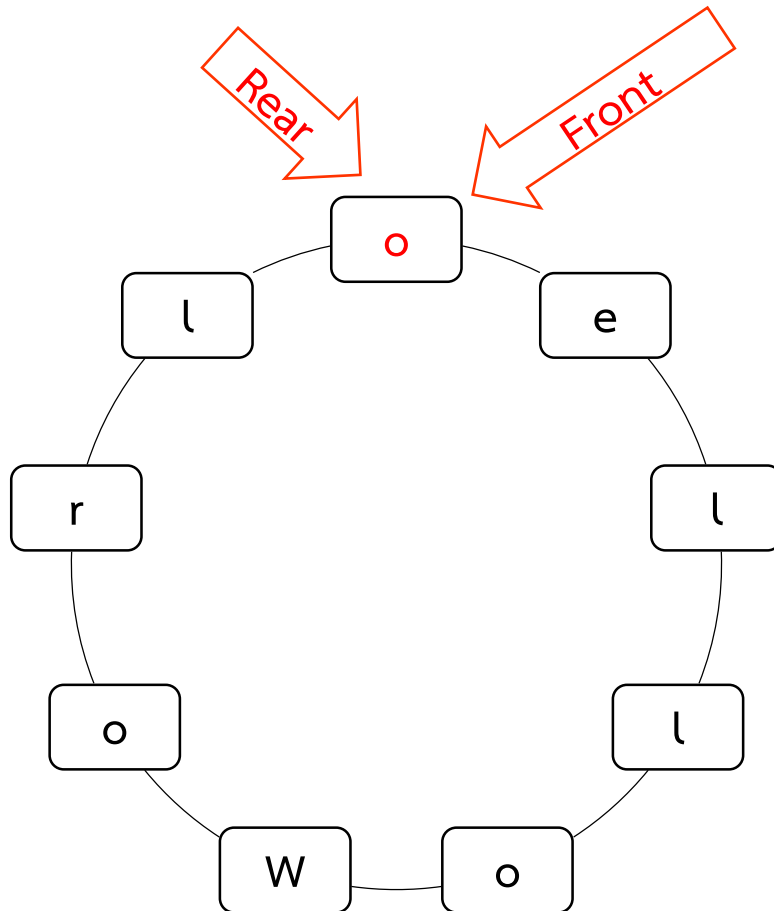


isFull() = 1
ข้อมูลเต็มแถวคอย

Queues: Circular Queues

Circular Queues

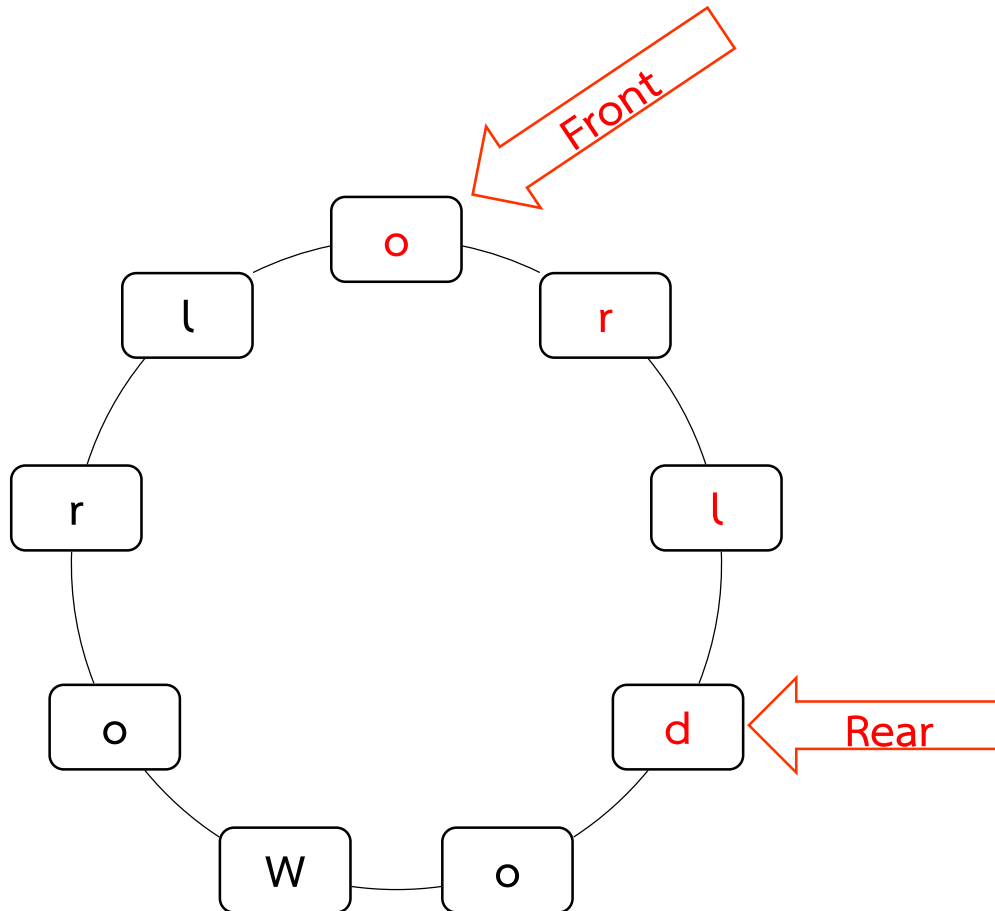
- Operation ของ Circular Queue จะเหมือนกับ Linear Queue
- แต่การเขียนโปรแกรมจะแตกต่างกันเล็กน้อยในส่วนของ `isFull()` และ `Add()`
- สามารถเกิด overflow ขึ้นได้



Queues: Circular Queues

Circular Queues

- Operation ของ Circular Queue จะเหมือนกับ Linear Queue
- แต่การเขียนโปรแกรมจะแตกต่างกันเล็กน้อยในส่วนของ `isFull()` และ `Add()`
- สามารถเกิด overflow ขึ้นได้

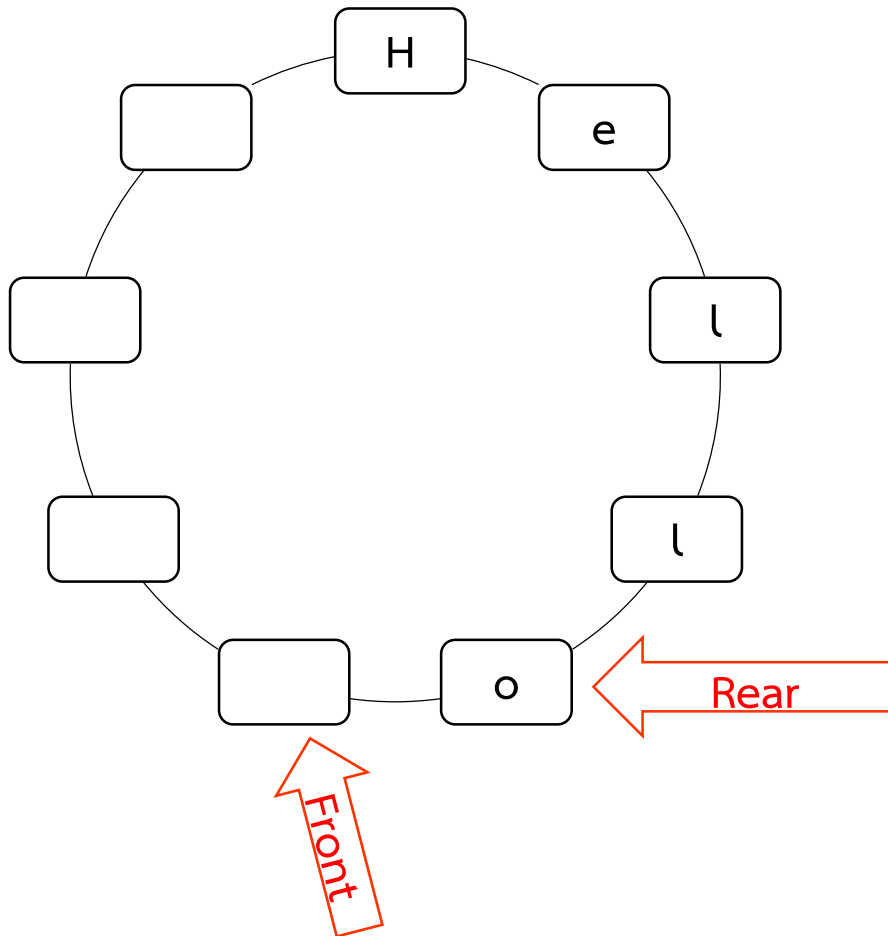


เกิด Overflow

Queues: Circular Queues

Circular Queues

- Operation ของ Circular Queue จะเหมือนกับ Linear Queue
- แต่การเขียนโปรแกรมจะแตกต่างกันเล็กน้อยในส่วนของ `isFull()` และ `Add()`
- สามารถเกิด overflow ขึ้นได้

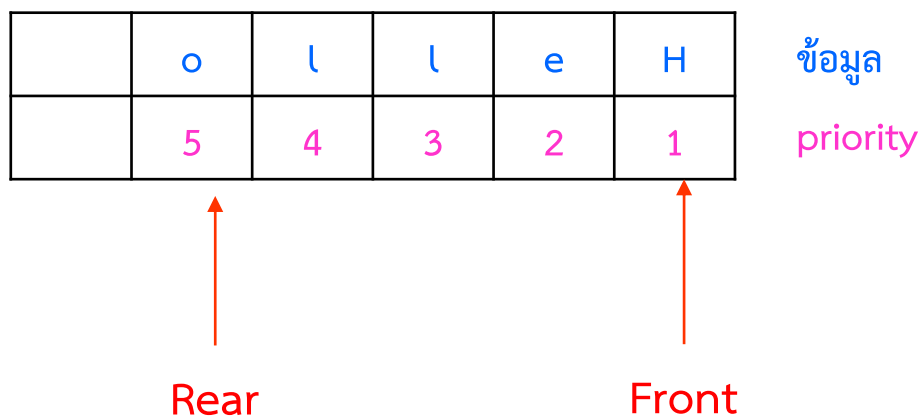


Underflow
ข้อมูลถูกลบมากกว่าที่เติมเข้าไป

Queues: Priority Queues

Priority Queues คือแถวคอยที่สามารถกำหนดลำดับความสำคัญของ element ได้ การนำข้อมูลเข้าและออก จะคํานึงจากระดับความสำคัญเป็นหลัก

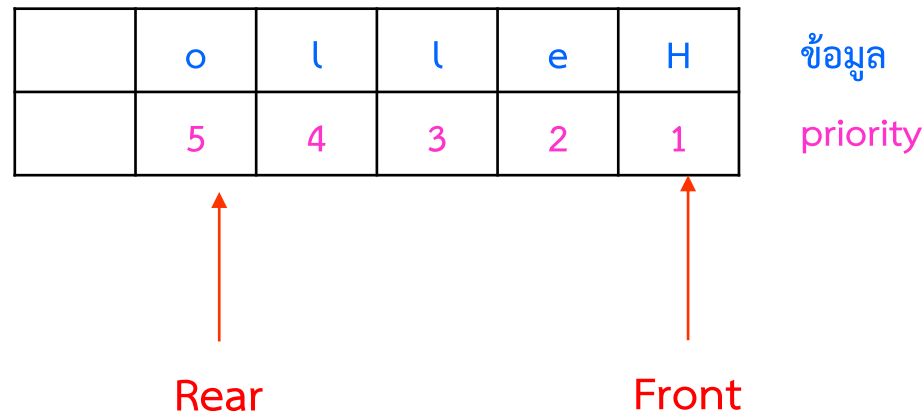
การสร้างจะเพิ่มข้อมูลส่วนของ priority เข้าไปกับข้อมูลที่ต้องการบันทึก โดยการเพิ่มข้อมูลเข้า จะทำการเรียงข้อมูลตามลำดับความสำคัญในการบันทึก



Queues: Priority Queues

การสร้างจะเพิ่มข้อมูลส่วนของ priority เข้าไปกับข้อมูลที่ต้องการบันทึก โดยการเพิ่มข้อมูลเข้า จะทำการเรียงข้อมูลตามลำดับความสำคัญในการบันทึก

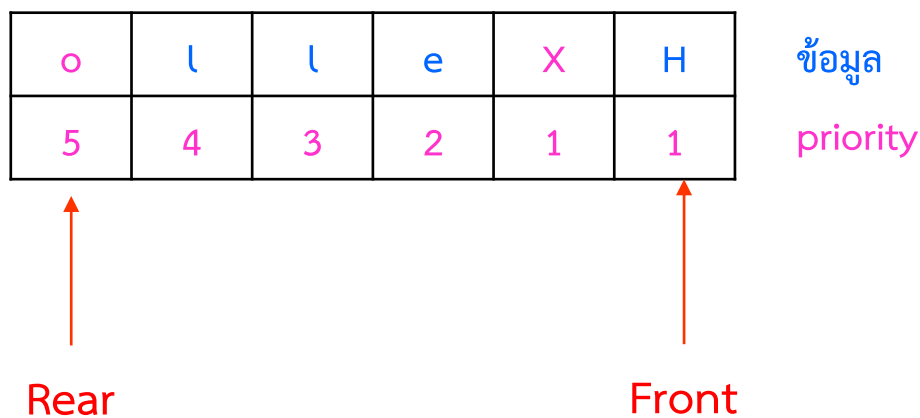
Add('X',1)



Queues: Priority Queues

การสร้างจะเพิ่มข้อมูลส่วนของ priority เข้าไปกับข้อมูลที่ต้องการบันทึก โดยการเพิ่มข้อมูลเข้า จะทำการเรียงข้อมูลตามลำดับความสำคัญในการบันทึก

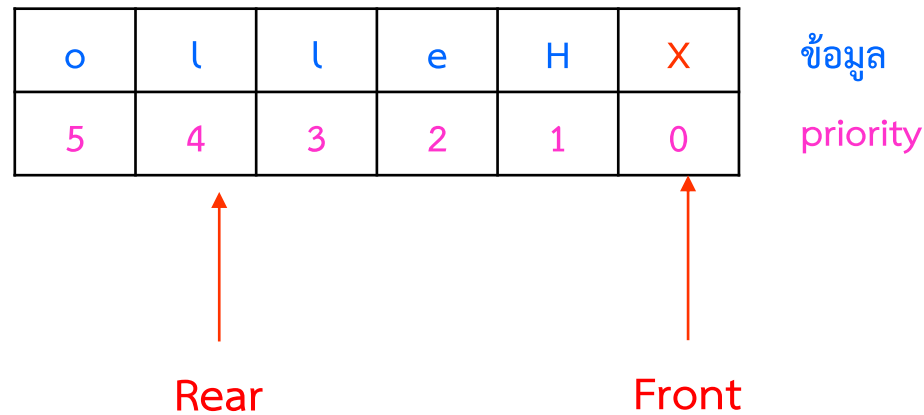
Add('X',1)



Queues: Priority Queues

การสร้างจะเพิ่มข้อมูลส่วนของ priority เข้าไปกับข้อมูลที่ต้องการบันทึก โดยการเพิ่มข้อมูลเข้า จะทำการเรียงข้อมูลตามลำดับความสำคัญในการบันทึก

Add('X',0)



Queues: Priority Queues

การสร้าง Priority Queues สามารถทำได้ 3 วิธีคือ

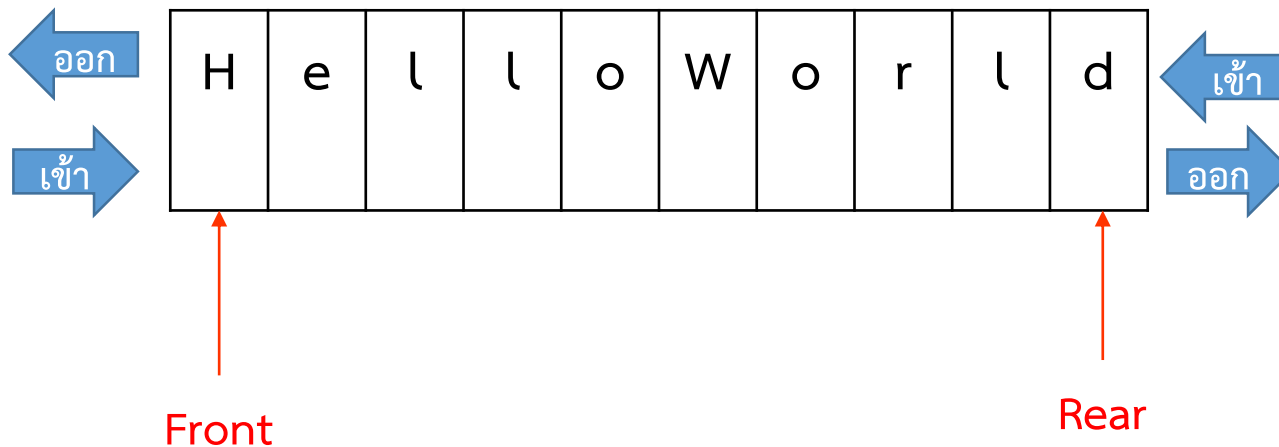
- 1) Array ใช้เวลา add เป็น $O(N)$
- 2) Linked list ใช้เวลา add เป็น $O(N)$
- 3) Heap ใช้เวลา add น้อยกว่า $O(N)$
(จะกล่าวถึงในบทถัดไป)

Queues: Double-Ended Queues

Double-Ended Queues (Deque : ออกเสียงว่า deck)

คือแถวคอยที่สามารถใส่ข้อมูล หรือนำข้อมูลออกได้ทั้ง สองด้าน

- นิยมใช้เนื่องจาก สามารถใช้เป็นแถวลำดับ และทำงานเป็น stack ได้ด้วย
- อาจมีการจำกัดให้ ใส่ข้อมูลเพิ่มได้เพียงด้านเดียว แต่ลบได้สองด้าน จะเรียกว่า Input-restricted Deque
- หากมีการจำกัดให้สามารถลบข้อมูลได้เพียงด้านเดียว แต่เพิ่มข้อมูลได้ทั้งสองด้าน จะเรียกว่า Output-restricted Deque



Queues: Double-Ended Queues

Double-Ended Queues สามารถใช้ในการเลื่อนข้อมูลไปทาง ซ้าย และขวาได้ และหากสร้างให้เป็นแบบ circular ก็สามารรถเลื่อนข้อมูลจาก ท้ายสุด กลับมาหน้าสุดได้ จึงนิยมใช้ในการพัฒนาซอฟต์แวร์

คลาส Deque ของ Java

	First Element (Head)		Last Element (Tail)	
	<i>Throws exception</i>	<i>Special value</i>	<i>Throws exception</i>	<i>Special value</i>
Insert	<code>addFirst(e)</code>	<code>offerFirst(e)</code>	<code>addLast(e)</code>	<code>offerLast(e)</code>
Remove	<code>removeFirst()</code>	<code>pollFirst()</code>	<code>removeLast()</code>	<code>pollLast()</code>
Examine	<code>getFirst()</code>	<code>peekFirst()</code>	<code>getLast()</code>	<code>peekLast()</code>

Queue Method	Equivalent Deque Method
<code>add(e)</code>	<code>addLast(e)</code>
<code>offer(e)</code>	<code>offerLast(e)</code>
<code>remove()</code>	<code>removeFirst()</code>
<code>poll()</code>	<code>pollFirst()</code>
<code>element()</code>	<code>getFirst()</code>
<code>peek()</code>	<code>peekFirst()</code>

Stack Method	Equivalent Deque Method
<code>push(e)</code>	<code>addFirst(e)</code>
<code>pop()</code>	<code>removeFirst()</code>
<code>peek()</code>	<code>peekFirst()</code>