หน่วยที่ 4

โครงสร้างข้อมูลแบบคิว

หัวข้อเรื่อง

1. โครงสร้างข้อมูลแบบคิว
2. การสร้างคิวด้วยโครงสร้างข้อมูลแบบอาร์เรย์
3. การดำเนินการกับคิว
4. การนำข้อมูลเข้าสู่คิว
5. การนำข้อมูลออกจากคิว
6. คิววงกลม

สาระสำคัญ

โครงสร้างข้อมูลแบบคิว (Queues Structure) เป็นโครงสร้างข้อมูลแบบเชิงเส้น มีลักษณะเป็นแถวรอคอย ลำดับการนำข้อมูลเข้าออกจากคิวมีความสำคัญ นั่นคือ ข้อมูลที่เข้าไปในคิวก่อนจะถูกนำออกจากคิวก่อนและข้อมูลที่เข้าไปทีหลังจะถูกนำออกมาทีหลัง จึงเรียกโครงสร้างข้อมูลแบบคิวว่าเป็นโครงสร้างข้อมูลแบบ **เข้าก่อนออกก่อน** (First In, First Out หรือ FIFO) ลักษณะโครงสร้างข้อมูลแบบคิวสามารถพบเห็นได้บ่อยๆ ในการดำเนินชีวิตประจำวัน เช่น การเข้าคิวรอซื้ออาหารในโรงอาหาร การเข้าคิวขึ้นรถประจำทาง การเข้าคิวรอตรวจโรคในโรงพยาบาล การเข้าคิวซื้อบัตรชมคอนเสิร์ต เป็นต้น สำหรับการประยุกต์ใช้งานโครงสร้างข้อมูลแบบคิวในงานคอมพิวเตอร์ เช่น การเข้าคิวของโพรเซสหรืองานต่างๆ เพื่อรอการประมวลผลจากซีพียูตามลำดับ

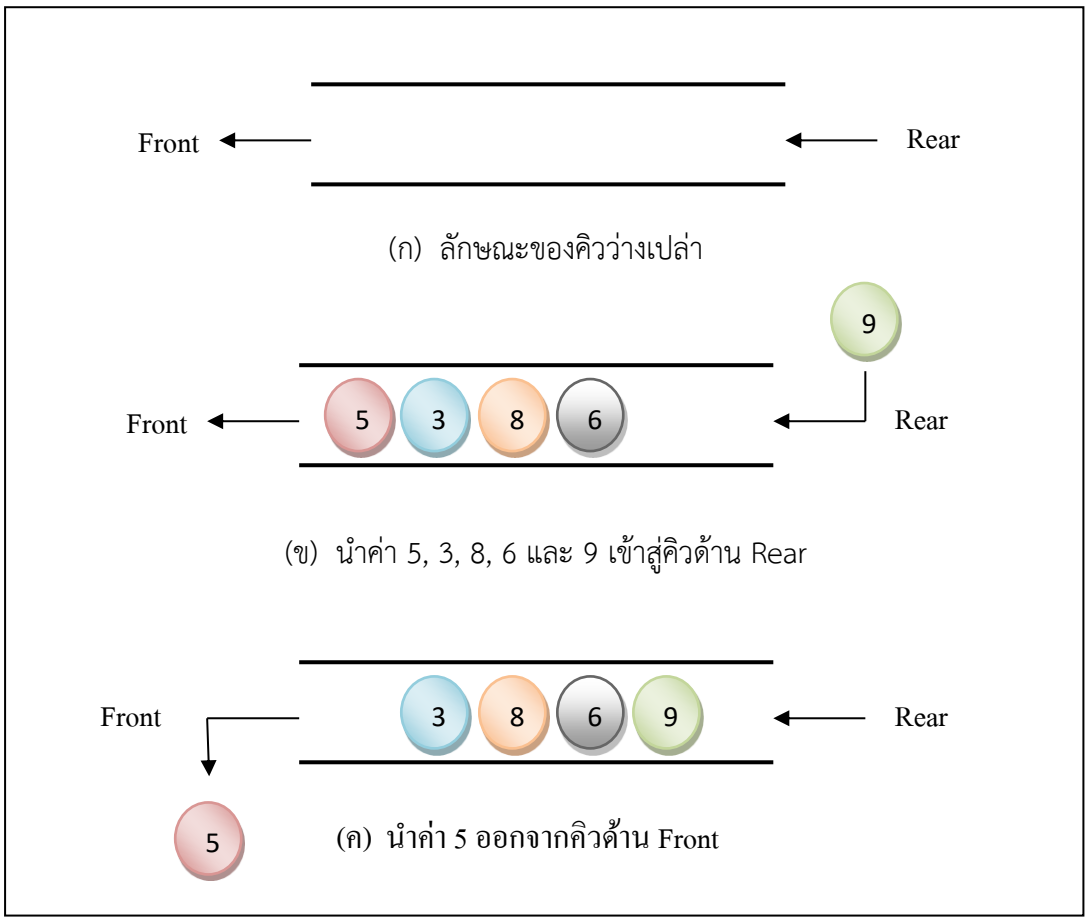
จุดประสงค์เชิงพฤติกรรม

1. อธิบายโครงสร้างข้อมูลแบบคิวได้
2. อธิบายการสร้างคิวด้วยโครงสร้างข้อมูลแบบอาร์เรย์ได้
3. อธิบายลักษณะการดำเนินการกับคิวได้
4. เขียนอัลกอริทึมการนำข้อมูลเข้าสู่คิวได้
5. เขียนอัลกอริทึมการนำข้อมูลออกจากคิวได้
6. อธิบายคิววงกลมได้
7. อธิบายการตรวจสอบคิวเต็มในคิววงกลมได้
8. อธิบายการตรวจสอบคิวว่างในคิววงกลมได้
9. อธิบายคำศัพท์ที่เกี่ยวข้องกับโครงสร้างข้อมูลแบบคิวได้
10. ประยุกต์โครงสร้างข้อมูลแบบคิวมาเขียนเป็นโปรแกรมได้

โครงสร้างข้อมูลแบบคิว

โครงสร้างข้อมูลแบบคิว (Queues Structure) เป็นโครงสร้างข้อมูลแบบเชิงเส้น มีลักษณะเป็นแถวรอคอย นั่นคือมีการจัดเรียงลำดับข้อมูลในการเข้าและออกจากคิวอย่างเป็นลำดับ โดยข้อมูลใดเข้าไปในคิวก่อนก็จะถูกนำออกจากคิวไปก่อน ข้อมูลใดเข้าไปในคิวทีหลังก็จะถูกนำออกไปทีหลัง จากลักษณะการทำงานเช่นนี้จึงเรียกโครงสร้างข้อมูลแบบคิวว่าเป็นโครงสร้างข้อมูลแบบ **เข้าก่อนออกก่อน** (First In, First Out หรือ FIFO) ซึ่งแตกต่างจากโครงสร้างข้อมูลแบบสแตกที่มีลักษณะแบบ **เข้าหลังออกก่อน** (Last In, First Out หรือ LIFO)

โครงสร้างข้อมูลแบบคิวจะนำข้อมูลเข้าสู่คิวและออกจากคิวคนละทาง นั่นคือในการนำข้อมูลเข้าสู่คิวจะกระทำตรงส่วนท้ายของคิวที่เรียกว่า Rear ในขณะที่การนำข้อมูลออกจากคิวจะกระทำตรงส่วนหัวของคิวที่เรียกว่า Front ซึ่งสามารถแสดงลักษณะการทำงานของคิวได้ดังรูปที่ 4.1



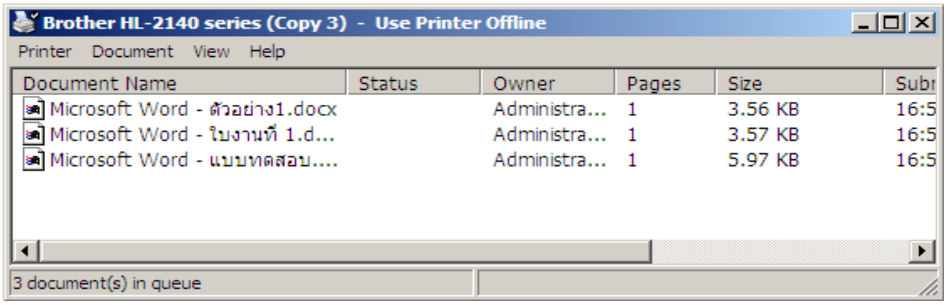
รูปที่ 4.1 แสดงลักษณะการทำงานของคิว

จากรูปที่ 4.1 (ก) เป็นลักษณะของคิวว่างเปล่า เมื่อมีการนำข้อมูลคือ 5, 3, 8, 6 และ 9 เข้าสู่คิวก็จะนำเข้าทางด้าน Rear ซึ่งข้อมูลจะถูกจัดเก็บในคิวเรียงตามลำดับ ดังรูป (ข) แต่เมื่อมีการนำข้อมูลออกจากคิวข้อมูลที่เข้าไปในคิวก่อนคือ 5 จะถูกนำออกมาก่อนทางด้าน Front ดังรูป (ค)

ตัวอย่างการทำงานของคิวที่พบเห็นโดยทั่วไปในชีวิตประจำวัน เช่น การเข้าคิวซื้ออาหารในโรงอาหาร การเข้าคิวรับบริการฝาก-ถอนเงินในธนาคาร การเข้าคิวซื้อตั๋วชมภาพยนตร์ การเข้าคิวรอรับการตรวจโรคในโรงพยาบาล การเข้าคิวส่งงานครู เป็นต้น โดยผู้ที่เข้ามาก่อนก็จะได้รับบริการก่อน ผู้มาทีหลังก็จะต้องรอจนกระทั่งถึงคิวตัวเอง

รูปที่ 4.2 นักศึกษาเข้าคิวรอส่งงานครู

สำหรับการประยุกต์ใช้งานโครงสร้างข้อมูลแบบคิวในงานคอมพิวเตอร์ เช่น การเข้าคิวของโพรเซสหรืองานต่างๆ เพื่อรอการประมวลผลจากซีพียูตามลำดับ และตัวอย่างคิวที่เห็นได้ชัดเจน คือ การเข้าคิวของเอกสารเพื่อรอพิมพ์ผลลัพธ์จากเครื่องพิมพ์ จะเห็นได้ว่าเมื่อเราส่งงานออกไปพิมพ์ทางเครื่องพิมพ์หลายๆ งานพร้อมกัน งานต่างๆ เหล่านั้นจะถูกจัดคิวเพื่อรอพิมพ์ตามลำดับ งานใดส่งมาก่อนก็ถูกพิมพ์ก่อน งานใดส่งมาทีหลังก็จะพิมพ์ทีหลัง



รูปที่ 4.3 งานต่างๆ ที่ส่งพิมพ์จะถูกจัดคิวเพื่อรอพิมพ์ตามลำดับ

การสร้างคิวด้วยโครงสร้างข้อมูลแบบอาร์เรย์

การสร้างคิวด้วยโครงสร้างข้อมูลแบบอาร์เรย์ หมายถึง การแทนที่ข้อมูลของคิวด้วยอาร์เรย์ ซึ่งเป็นการจัดสรรพื้นที่หน่วยความจำแบบคงที่ นั่นคือมีการกำหนดขนาดของคิวไว้ล่วงหน้าว่ามีขนาดเท่าใด และจะมีการจัดสรรพื้นที่หน่วยความจำให้เลย ซึ่งลักษณะเช่นนี้อาจเกิดความผิดพลาดกรณีคิวเต็มได้หากเพิ่มจำนวนข้อมูลมากกว่าที่กำหนดไว้ ดังนั้นจึงต้องกำหนดขนาดของคิวให้เพียงพอกับจำนวนข้อมูล และต้องกำหนดตัวแปรขึ้นมา 2 ตัว คือ Front และ Rear เพื่อใช้ชี้ข้อมูลที่อยู่ส่วนหัวและส่วนท้ายของคิว

การประกาศตัวแปรอาร์เรย์ให้มีโครงสร้างข้อมูลแบบคิวด้วยภาษาซี

```
#define MAX 5
int Queue[MAX]
int Front = -1;
int Rear = -1;
```

เป็นการประกาศตัวแปรอาร์เรย์ใช้แทนคิวชื่อ Queue สามารถเก็บข้อมูลได้สูงสุด 5 ค่า และประกาศตัวแปร Front และ Rear เป็นชนิดเลขจำนวนเต็มเพื่อใช้ชี้ข้อมูลแรกและข้อมูลสุดท้ายในคิว โดยกำหนดให้เก็บค่า -1 หมายความว่า ยังไม่ได้ชี้ข้อมูลใดๆ เป็นการบ่งบอกให้รู้ว่าขณะนี้คิวว่างเปล่าไม่มีข้อมูล

การดำเนินการกับคิว

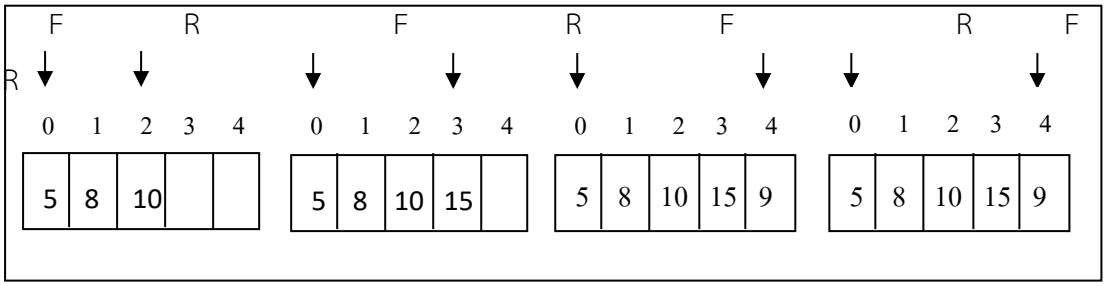
การดำเนินการกับคิวประกอบด้วย 2 ลักษณะใหญ่ๆ คือ

1. การนำข้อมูลเข้าสู่คิว
2. การนำข้อมูลออกจากคิว

กรณีที่สร้างคิวด้วยโครงสร้างข้อมูลแบบอาร์เรย์ที่มีการกำหนดขนาดของคิวที่แน่นอน ก่อนจะนำข้อมูลเข้าสู่คิวหรือนำข้อมูลออกจากคิวจะต้องทำการตรวจสอบคิวก่อนว่าคิวนั้นมีข้อมูลอยู่เต็มหรือไม่ หรือคิวนั้นว่างเปล่าหรือไม่เพื่อป้องกันความผิดพลาดที่อาจเกิดขึ้น และจะกำหนดตัวแปร 2 ตัว คือ Front หรือ F และ Rear หรือ R เพื่อใช้ชี้ข้อมูลแรกและข้อมูลสุดท้ายในคิว

การนำข้อมูลเข้าสู่คิว

การนำข้อมูลเข้าสู่คิว จะเรียกว่าการ **Enqueue** หลังจากที่ได้นำข้อมูลเข้าสู่คิวแล้ว ข้อมูลใหม่ที่เพิ่มเข้าไปจะถูกนำไปต่อท้ายทางด้าน R ในการสั่งให้นำข้อมูลเข้าสู่คิวนั้นจะต้องมั่นใจว่ามีพื้นที่ว่างในคิวพอที่จะเพิ่มข้อมูลใหม่เข้าไปได้ เพราะถ้าหากไม่มีที่ว่างหรือคิวเต็มแล้วจะทำให้เกิดความผิดพลาดที่เรียกว่า **Queue Overflow** ขึ้น ดังรูปที่ 4.4



(ก) Enqueue (15) (ข) Enqueue (9) (ค) Enqueue (20) (ง) * Queue Overflow

รูปที่ 4.4 แสดงการนำข้อมูลเข้าสู่คิว (Enqueue) ที่สร้างด้วยอาร์เรย์ขนาด 5 ช่อง

จากรูปที่ 4.4 (ก) คิวมีข้อมูลอยู่แล้ว 3 ค่า คือ 5, 8 และ 10 โดยมี F ชี้ที่ 5 บอกให้รู้ว่า 5 เป็นข้อมูลแรกในคิว และ R ชี้ที่ 10 บอกให้รู้ว่า 10 เป็นข้อมูลสุดท้ายที่เข้าไปในคิว และเมื่อมีการ Enqueue (15) ค่า 15 จะถูกเพิ่มลงในคิวทางด้าน R ต่อท้าย 10 และ R จะชี้ที่ 15 ซึ่งถือว่าเป็นข้อมูลสุดท้ายในคิว ดังรูป (ข) เมื่อมีการ Enqueue (9) เข้าไปอีกค่า 9 ก็จะไปต่อท้าย 15 และ R ก็จะเลื่อนไปชี้ที่ 9 ดังรูป (ค) ซึ่งจะเห็นได้ว่าขณะนี้ R ชี้ที่ตำแหน่งสุดท้ายในคิวแล้ว เมื่อมีการ Enqueue (20) เข้าไปอีก ตอนนี้จะเกิดความผิดพลาดที่เรียกว่า “Queue Overflow” เนื่องจากคิวเต็ม ดังรูป (ง)

การนำข้อมูลเข้าสู่คิวสามารถเขียนลำดับขั้นตอนการทำงานได้ดังอัลกอริทึมที่ 4.2

อัลกอริทึมที่ 4.1 ลำดับขั้นตอนการนำข้อมูลเข้าสู่คิว

1. ตรวจสอบว่าคิวเต็มหรือไม่ โดยเปรียบเทียบค่า Rear กับตำแหน่งสุดท้ายของตัวแปรอาร์เรย์
   1.1 ถ้าคิวเต็ม (Rear เท่ากับตำแหน่งสุดท้ายของตัวแปรอาร์เรย์) ให้แจ้งให้ทราบว่าคิวเต็ม แล้วเลิกการทำงาน
   1.2 ถ้าคิวยังไม่เต็มให้เพิ่มค่า Rear อีก 1 เพื่อเลื่อน Rear ไปตำแหน่งถัดไป
2. นำข้อมูลเข้าสู่คิวในตำแหน่ง Rear
3. ถ้าข้อมูลที่นำเข้าสู่คิวเป็นข้อมูลแรกในคิวให้ Front ชี้ไปที่ข้อมูลนั้น

จากอัลกอริทึมที่ 4.1 สามารถนำมาเขียนเป็นฟังก์ชั่นในภาษาซีได้ดังโปรแกรมที่ 4.1

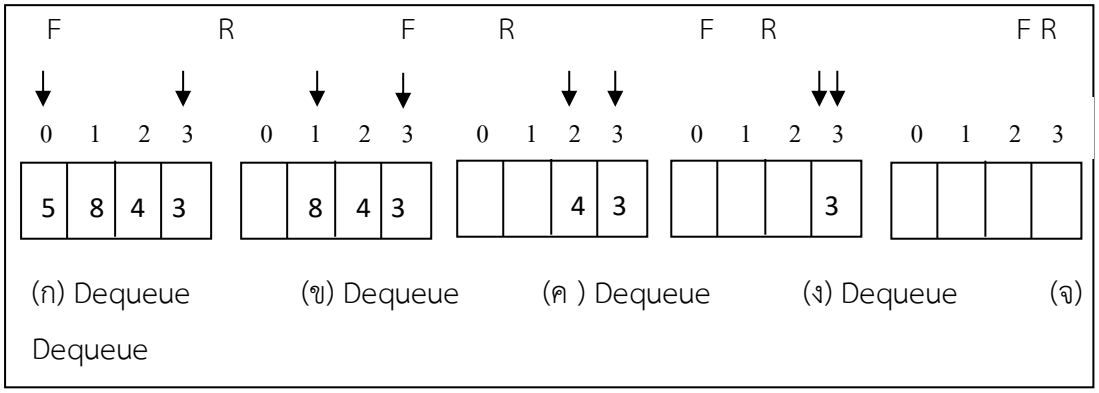
โปรแกรมที่ 4.1 ฟังก์ชั่นการนำข้อมูลเข้าสู่คิว (Enqueue)

```
int Enqueue(int data)
{
        if (Rear<MAX-1)
        {
                Rear++;
                Queue[Rear] = data;
                if (Front == -1)
                        Front = 0;
                return 1;
        }
        return -1;
}
```

จากโปรแกรมที่ 4.1 การทำงานของฟังก์ชั่น Enqueue เริ่มด้วยการตรวจสอบเงื่อนไข คือ Rear < MAX-1 หรือไม่ (ที่ต้อง -1 เพราะเลขลำดับดัชนีของอาร์เรย์ในภาษาซีเริ่มต้นที่ 0) ถ้าใช่แสดงว่าคิวยังไม่เต็มให้เพิ่มค่า Rear อีก 1 (R++;) และนำข้อมูลที่ส่งมาทางพารามิเตอร์ data เก็บลงในตัวแปร Queue ในตำแหน่ง Rear จากนั้นทำการตรวจสอบว่าค่า Front เท่ากับ -1 (Front==-1) หรือไม่ ถ้าใช่แสดงว่าข้อมูลที่เก็บลงไปเป็นข้อมูลแรกในคิว ก็ปรับให้ Front = 0 และ return 1 กลับไปยังจุดที่เรียกใช้เป็นการบอกว่านำข้อมูลเข้าสู่คิวเรียบร้อยแล้ว แต่ถ้าเงื่อนไขแรกไม่เป็นจริง (Rear ไม่น้อยกว่า MAX-1) จะลงมาทำที่ return -1 นั่นคือฟังก์ชั่นจะคืนค่า -1 กลับไปยังจุดที่เรียกใช้เพื่อบอกว่าคิวเต็มแล้วนั่นเอง

การนำข้อมูลออกจากคิว

การนำข้อมูลออกจากคิว จะเรียกว่าการ **Dequeue** เป็นการนำข้อมูลตัวแรกหรือตัวที่อยู่หน้าสุดในตำแหน่ง F ออกจากคิว แล้วทำการคืนค่าไปยังยูสเซอร์หรือโมดูลที่เรียกใช้ ในการ Dequeue ถ้าหากคิวว่างเปล่าไม่มีข้อมูลอยู่เลยจะเกิดความผิดพลาดที่เรียกว่า **Queue Underflow**



* Queue Underflow

รูปที่ 4.5 แสดงการนำข้อมูลออกจากคิว (Dequeue)

จากรูปที่ 4.5 (ก) คิวมีข้อมูลอยู่เต็มโดยมี F ชี้ที่ข้อมูลแรกคือ 5 และ R ชี้ที่ข้อมูลสุดท้าย คือ 3 เมื่อมีการ Dequeue ข้อมูลที่อยู่ในตำแหน่ง F คือ 5 จะถูกนำออกมาจากคิวก่อน และ F จะเลื่อนไปชี้ที่ค่าถัดไป คือ 8 ซึ่งถือว่าเป็นข้อมูลแรกในคิวขณะนี้ ดังรูป (ข) เมื่อมีการ Dequeue อีก 8 ซึ่งอยู่ในตำแหน่ง F ก็จะถูกนำออกจากคิว และ F ก็จะเลื่อนไปชี้ที่ค่าถัดไปคือ 4 ดังรูป (ค) ซึ่งจะกระทำเช่นนี้ไปเรื่อยๆ จนกระทั่ง F และ R อยู่ตำแหน่งเดียวกันแสดงว่าเหลือข้อมูลตัวสุดท้ายในคิว ดังรูป (ง) เมื่อมีการ Dequeue อีก ตอนนี้ข้อมูลตัวสุดท้ายถูกนำออกไปทำให้คิวว่างเปล่า F และ R จะไม่ได้ชี้ข้อมูลใด ดังรูป (จ) และถ้าหากให้ทำการ Dequeue อีกตอนนี้จะเกิดความผิดพลาดที่เรียกว่า Queue Underflow เนื่องจากเป็นการให้นำข้อมูลออกจากคิวที่ว่างเปล่า

การนำข้อมูลออกจากคิวสามารถเขียนลำดับขั้นตอนการทำงานได้ดังอัลกอริทึมที่ 4.2

อัลกอริทึมที่ 4.2 ลำดับขั้นตอนการนำข้อมูลออกจากคิว

1. ตรวจสอบว่าคิวว่างหรือไม่ โดยการเปรียบเทียบค่า Front กับ -1
   1.1 ถ้า Front = -1 แสดงว่า คิวว่าง ให้แจ้งให้ทราบว่าคิวว่าง แล้วจบการทำงาน
   1.2 ถ้า Front ≠ -1 แสดงว่ายังมีข้อมูลอยู่ในคิว ให้นำข้อมูลในตำแหน่ง Front ออกมา เก็บไว้ในตัวแปร
2. ตรวจสอบว่าข้อมูลที่นำออกมาเป็นข้อมูลสุดท้ายหรือไม่ โดยเปรียบเทียบค่า Front กับ Rear
   2.1 ถ้า Front > Rear แสดงว่าข้อมูลที่นำออกมาเป็นข้อมูลสุดท้าย ให้ปรับค่าของ Front และ Rear เป็น -1
   2.2 ถ้า Front <= Rear แสดงว่ายังไม่ใช่ข้อมูลสุดท้าย ให้เพิ่มค่า Front อีก 1 เพื่อเลื่อน

จากอัลกอริทึมที่ 4.2 สามารถนำมาเขียนเป็นฟังก์ชั่นในภาษาซีได้ดังโปรแกรมที่ 4.2

โปรแกรมที่ 4.2 ฟังก์ชั่นการนำข้อมูลออกจากคิว (Dequeue)

```
int Dequeue()
   {
         int  x;
         if (Front!=-1)
        {
               x = Queue[Front];
               if (Front>Rear)
              {
                   Front = -1;
                   Rear = -1;
                   return -1;
              }
   else
                   Front++;
    }
    else
            return -1;
     return x;
   }
```

จากโปรแกรมที่ 4.2 การทำงานของฟังก์ชั่น Dequeue จะเริ่มด้วยการตรวจสอบคิวก่อนว่าคิววางหรือไม่ โดยเปรียบเทียบค่าตัวแปร Front กับ -1 ถ้า Front เท่ากับ -1 แสดงว่าคิววางเปล่าไม่มีข้อมูล ฟังก์ชั่น Dequeue จะคืนค่า -1 กลับมา แต่ถ้า Front ไม่เท่ากับ -1 แสดงว่าคิวมีข้อมูลอยู่ ให้นำข้อมูลในคิวตำแหน่ง Front ออกมาเก็บไว้ในตัวแปร x จากนั้นก็จะตรวจสอบต่อไปอีกว่า Front > Rear หรือไม่ ถ้าใช่แสดงว่าได้นำข้อมูลออกจากคิวหมดแล้วก็ให้ Front และ Rear มีค่าเท่ากับ -1 และคืนค่า -1 กลับมาเพื่อบอกให้ทราบว่าข้อมูลหมดแล้วตอนนี้คิววางเปล่า แต่ถ้า Front ไม่มากกว่า Rear แสดงว่าข้อมูลที่นำออกมายังไม่ใช่ข้อมูลสุดท้ายก็ให้เพิ่มค่า Front (Front++;) เพื่อเลื่อนไปชี้ข้อมูลที่จะนำออกมาเป็นลำดับต่อไป

เพื่อให้เข้าใจลักษณะการดำเนินงานของโครงสร้างข้อมูลแบบคิวชัดเจนยิ่งขึ้น ให้พิจารณาโปรแกรมที่ 4.3 ซึ่งได้มีการใช้ return คืนค่าจากฟังก์ชั่นกลับมา ทำให้ทราบว่าฟังก์ชั่นทำงานสำเร็จหรือไม่ และทำให้ทราบสถานะของคิวด้วยว่าเป็นอย่างไร เพื่อจะได้สั่งให้ดำเนินการอย่างอื่นต่อไป

โปรแกรมที่ 4.3 โปรแกรมทดสอบการดำเนินการกับคิว

```
#include<stdio.h>
#include<conio.h>
#define MAX 5

int Queue[MAX];
int Front=-1;
int Rear=-1;

int Enqueue(int data)    /* ฟังก์ชั่นการนำข้อมูลเข้าสู่คิว  */
{
        if (Rear<MAX-1)
        {
                Rear++;
                Queue[Rear] = data;
                if (Front == -1)
                     Front=0;
                return 1;
         }
         return -1;
}

int Dequeue()      /* ฟังก์ชั่นการนำข้อมูลออกจากคิว  */
{
 int x;
 if (Front!=-1)
 {
       x = Queue[Front];
       if (Front> Rear)
```

```
        {
            Front = -1;
            Rear = -1;
            return -1;
          }
          else
            Front++;
      }
      else
          return -1;
      return x;
   }
void main()    /*  ฟังก์ชั่นหลัก  */
{
  int i;
  int  ch;
  clrscr();
  Enqueue(5);
  Enqueue(10);
  Enqueue(15);
  Enqueue(20);
  printf("Enqueue result =%d\n",Enqueue(25));
  printf("Enqueue result =%d\n",Enqueue(30));
```

(โปรแกรมต่อ)

```
 printf("\n < Dequeue > \n");
 for (i=1;i<=10;i++)
 {
     ch=Dequeue();
     if (ch!=-1)
           printf("%d. data = %d\n",i,ch);
     else
    {
           printf("Queue is Empty");
           break;
      }
 }
 getch();
}
```

ผลรัน

Enqueue result =1

Enqueue result =-1

< Dequeue >

1. Data = 5

2. Data = 10

3. Data = 15

4. Data = 20

5. Data = 25

Queue is Empty

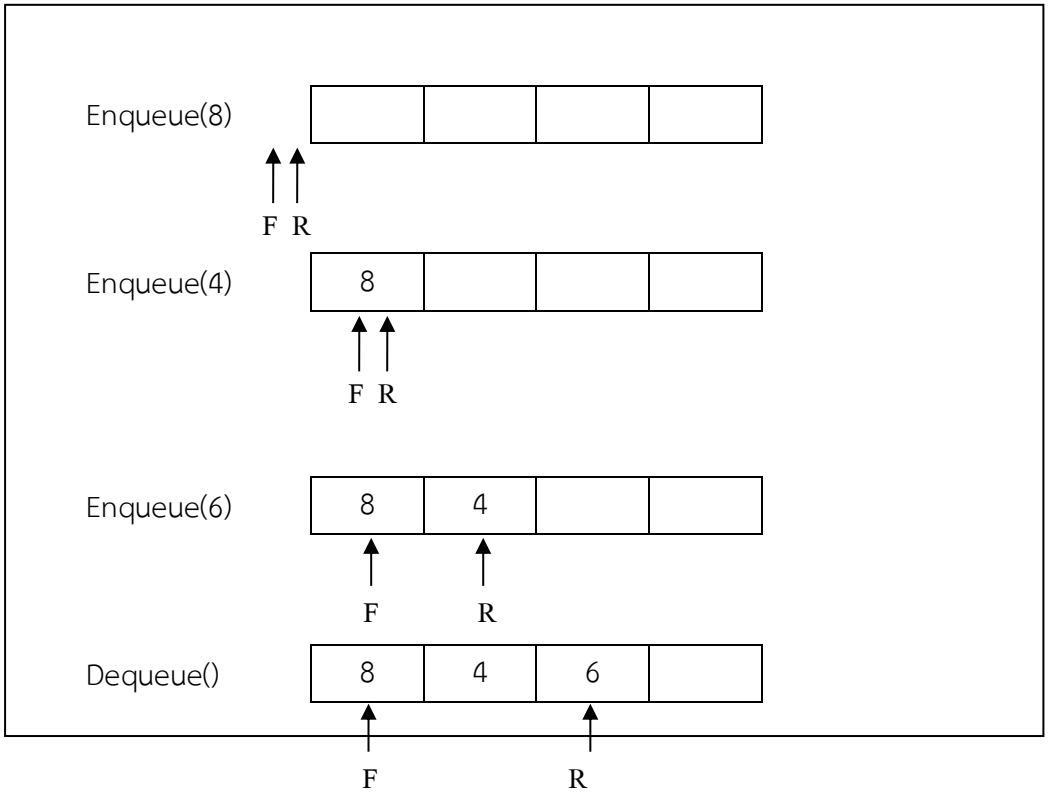
โปรแกรมที่ 4.3 ผลรันที่ได้จะเห็นว่า ถ้าฟังก์ชั่น Enqueue สามารถนำข้อมูลเข้าสู่คิวได้จะคืนค่ามาเป็น 1 แต่ถ้าไม่สามารถนำข้อมูลเข้าสู่คิวได้จะคืนค่ามาเป็น -1 ซึ่งจากการเรียกใช้ฟังก์ชั่น Enqueue 6 ครั้ง โดยในครั้งที่ 5 และ 6 สั่งให้แสดงค่า return ออกมา ผลที่ได้ก็คือครั้งที่ 5

แสดงค่าที่คืนมาเป็น 1 แสดงว่าสามารถนำ 25 เข้าสู่คิวได้ แต่ในครั้งที่ 6 แสดงค่าที่คืนมาเป็น -1 แสดงว่าไม่สามารถนำค่า 30 เข้าสู่คิวได้เนื่องจากคิวเต็ม ต่อมาได้เรียกใช้ฟังก์ชั่น Dequeue เพื่อนำข้อมูลออกจากคิวโดยใช้ลูป for เรียกใช้ฟังก์ชั่น 10 ครั้ง แต่คิวมีข้อมูลอยู่เพียง 5 ค่า ดังนั้นในครั้งที่ 6 ฟังก์ชั่น Dequeue ไม่สามารถดำเนินการได้สำเร็จเนื่องจากข้อมูลหมด จึงได้คืนค่ากลับมาเป็น -1 และเราได้ใช้ค่านี้ไปประยุกต์ใช้เพื่อแจ้งสถานะของคิว โดยให้แสดงข้อความ “Queue is Empty” เพื่อบอกให้ทราบว่าคิวว่างเปล่าไม่มีข้อมูล จะเห็นได้ว่าค่า return จากฟังก์ชั่นมีประโยชน์อย่างมาก ทำให้เราสามารถทราบสถานะทำงานของคิวว่าเป็นอย่างไร โดยเราสามารถนำค่า return จากฟังก์ชั่นไปปรับเขียนโปรแกรมเพื่อตรวจสอบและให้แจ้งสถานะของคิวได้ว่าคิวว่างหรือคิวเต็ม

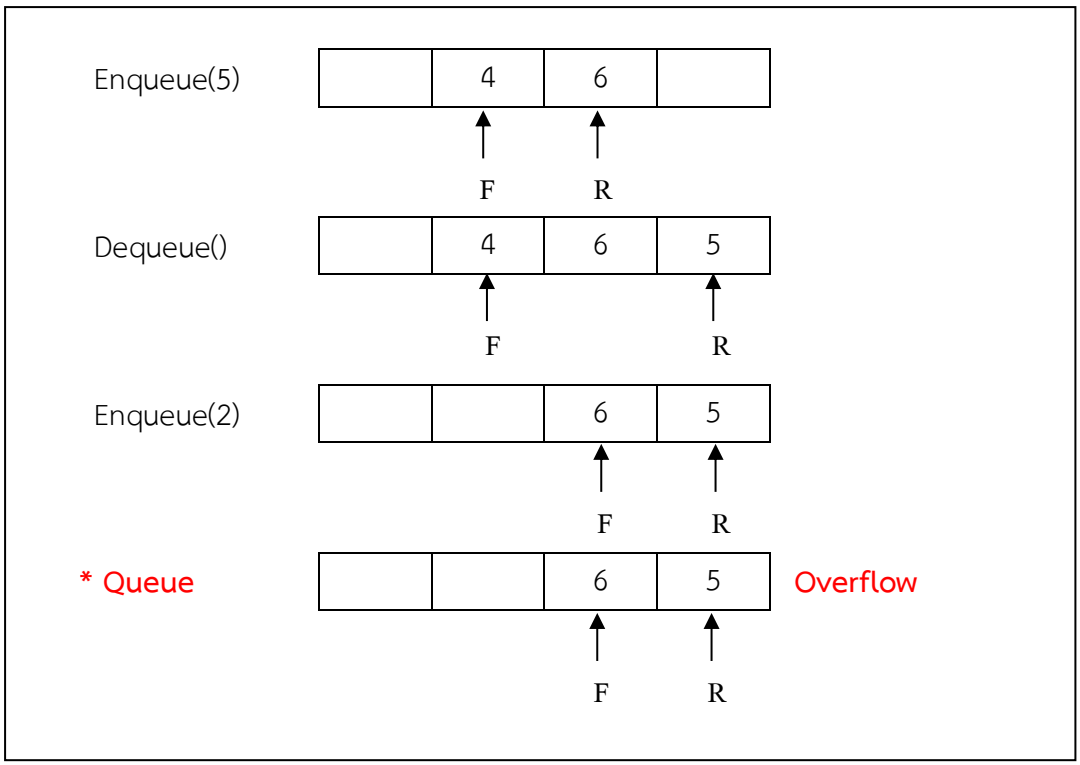
จากโปรแกรมที่ 4.3 เพื่อให้เห็นภาพการทำงานของคิวได้ชัดเจนขึ้น เราสามารถเขียนแผนภาพการทำงานของคิวได้ดังตัวอย่างที่ 4.1

ตัวอย่างที่ 4.1 จงเขียนแผนภาพการดำเนินการกับคิวซึ่งถูกสร้างด้วยอาร์เรย์ขนาด 4 ช่อง ตามลำดับดังนี้ Enqueue(8), Enqueue(4), Enqueue(6), Dequeue(), Enqueue(5), Dequeue(), Enqueue(2)

การดำเนินการ เริ่มต้น คิวว่างเปล่า F และ R ยังไม่ได้ชี้ค่าใดๆ

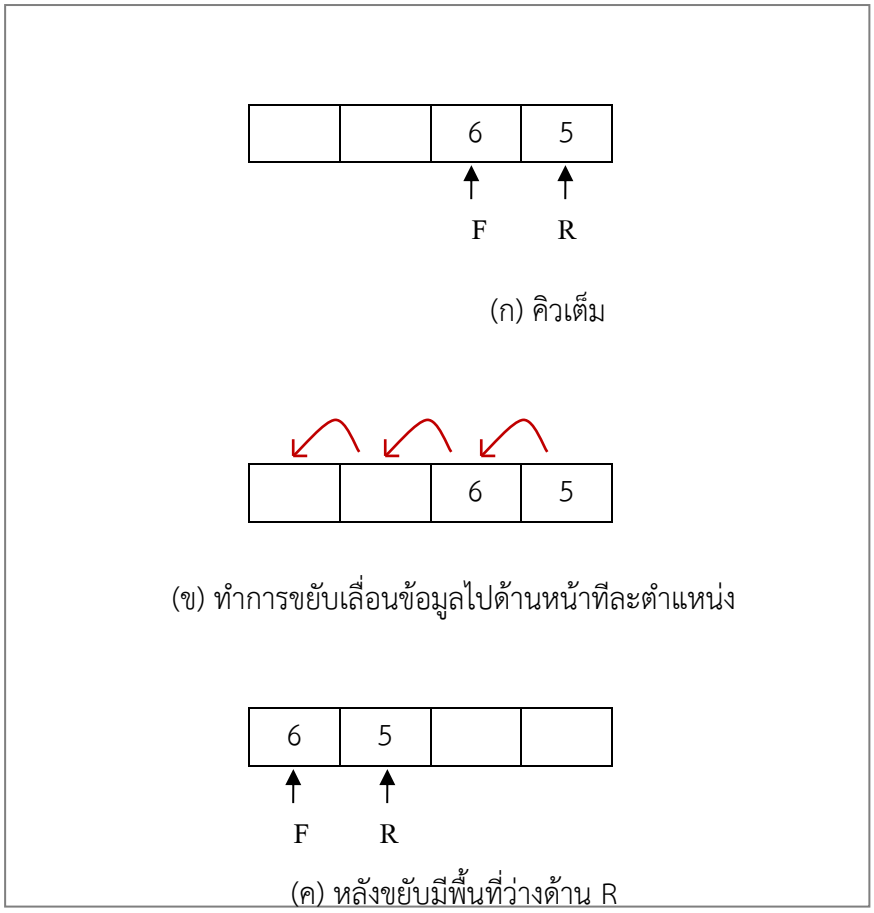


ตัวอย่างที่ 4.1 (ต่อ)



* เมื่อมีการ Enqueue(2) ในขณะที่คิวเต็มแล้ว (R ชี้ตำแหน่งสุดท้ายในคิว) จึงเกิดความผิดพลาด "Queue Overflow"

จากโปรแกรมที่ 4.3 และแผนภาพการดำเนินการกับคิวตามตัวอย่างที่ 4.1 จะเห็นได้ว่าการเพิ่มข้อมูลเข้าไปในคิวที่สร้างด้วยโครงสร้างข้อมูลแบบอาร์เรย์นั้น ถ้าหากทำการเพิ่มข้อมูลต่อท้ายจนกระทั่ง R ชี้ที่ตำแหน่งสุดท้ายในอาร์เรย์จะถือว่าคิวเต็ม ถึงแม้ว่าจะนำข้อมูลในส่วนหน้าด้าน F ออกไปแล้วและมีพื้นที่ว่างอยู่ก็ไม่สามารถนำข้อมูลเข้าไปได้ การแก้ปัญหาให้สามารถนำพื้นที่ว่างมาเก็บข้อมูลได้อีกจนกระทั่งคิวเต็มจริงๆ จะต้องทำการจัดเรียงข้อมูลในคิวใหม่ โดยการขยับเลื่อนข้อมูลในคิวทีละตัวไปข้างหน้าทีละตำแหน่งจนหมด หลังขยับเรียบร้อยแล้วก็จะทำให้เกิดพื้นที่ว่างด้าน R ทำให้สามารถเพิ่มข้อมูลต่อท้ายได้อีก ดังรูปที่ 4.6



รูปที่ 4.6 การจัดเรียงตำแหน่งข้อมูลในคิวใหม่

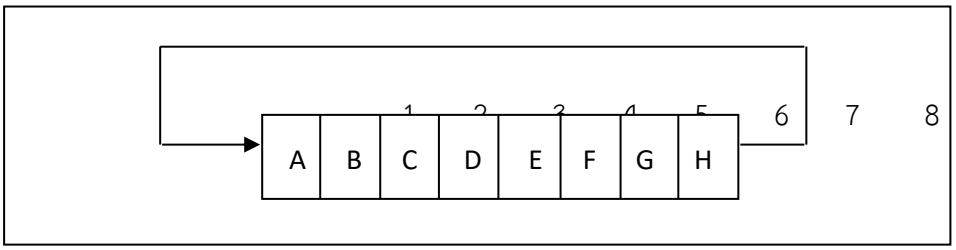
การจัดเรียงตำแหน่งข้อมูลภายในคิวใหม่นี้ ใช้ได้กับคิวที่บรรจุข้อมูลจำนวนไม่มากนัก แต่ถ้าเป็นคิวที่มีขนาดใหญ่ และภายในบรรจุฟิลด์มากมายในหนึ่งเรคอร์ด การใช้เทคนิคนี้จะใช้เวลามากในการประมวลผลเพื่อเรียงลำดับข้อมูลในคิวใหม่ เนื่องจากเมื่อมีการ Dequeue แต่ละครั้งก็จะต้องขยับตำแหน่งใหม่ทั้งหมด ดังนั้นวิธีการที่จะแก้ปัญหานี้ ก็คือใช้ **คิววงกลม**

คิววงกลม

คิววงกลม (Circular Queue) เป็นคิวที่คิดค้นมาเพื่อแก้ปัญหาที่เกิดขึ้นในกรณีที่ยังมีพื้นที่เหลือในการเก็บข้อมูลด้านหน้า Front แต่ส่วนท้าย Rear เต็มแล้ว ซึ่งจะทำให้สามารถ Enqueue ข้อมูลเข้ามาได้อีก จนกว่าคิวจะเต็มจริงๆ

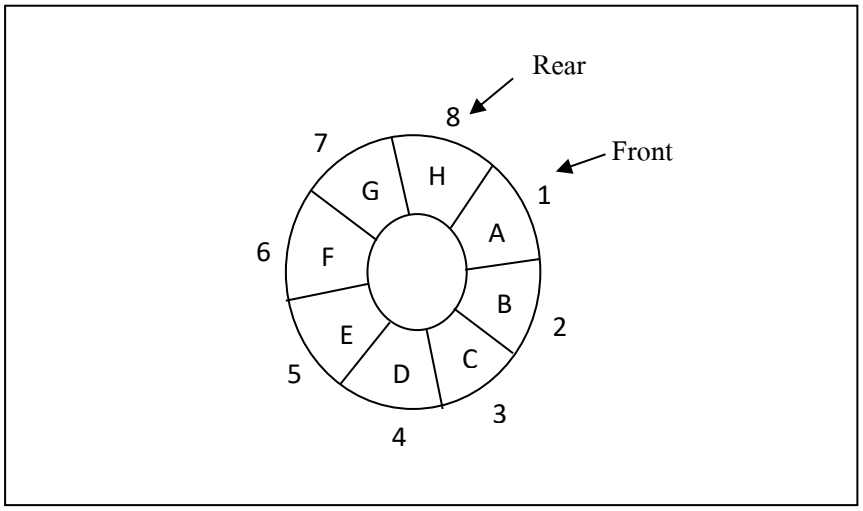
ลักษณะของคิววงกลม

ลักษณะของคิววงกลมเป็นการนำเอาตำแหน่งสุดท้ายของคิวมาเชื่อมต่อกับตำแหน่งแรกของคิว จึงทำให้มองเห็นคิวเป็นวงกลม ดังรูปที่ 4.7



รูปที่ 4.7 ลักษณะของคิววงกลม

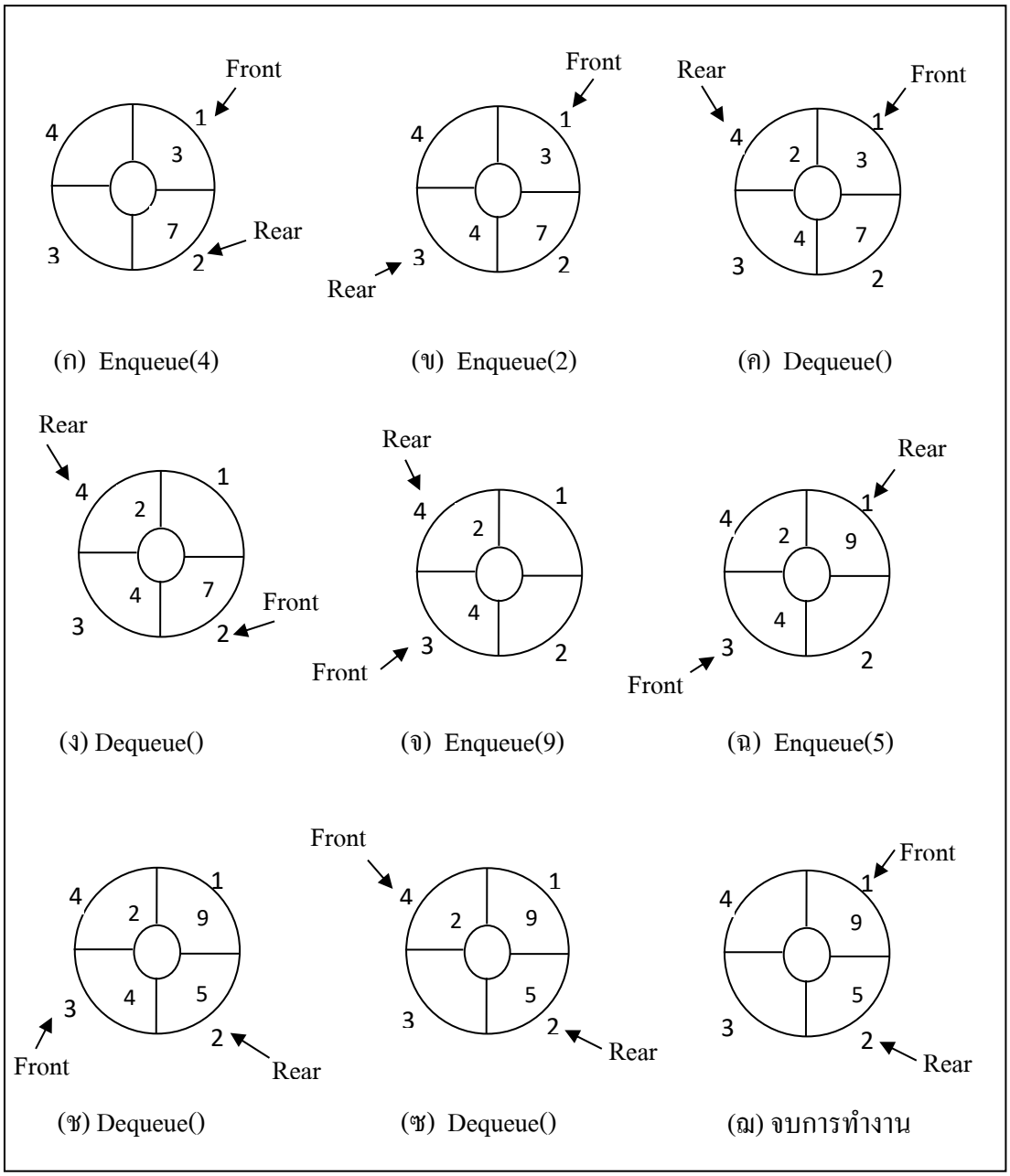
จากรูปที่ 4.7 ให้ส่วนท้ายของคิวที่ตำแหน่ง Queue[8] ไปต่อกับส่วนหน้าที่ตำแหน่ง queue[1] เพื่อให้เข้าใจลักษณะของคิววงกลมได้ง่ายขึ้น สามารถแสดงคิววงกลมได้ดังรูปที่ 4.8



รูปที่ 4.8 ลักษณะของคิววงกลม

การทำงานของคิววงกลม

การทำงานของคิววงกลม ถ้ามีการนำข้อมูลเข้าสู่คิววงกลมจนกระทั่ง R ชี้ที่ตำแหน่งสุดท้ายของคิววงกลม เมื่อสั่งให้นำข้อมูลเข้าสู่คิววงกลมอีกก็จะปรับเลื่อนให้ R มาชี้ที่ตำแหน่งแรกของคิววงกลม และถ้าหากตำแหน่งแรกเป็นพื้นที่ว่างก็สามารถจัดเก็บข้อมูลลงคิววงกลมได้ ในทำนองเดียวกันถ้า F ชี้ที่ตำแหน่งสุดท้ายของคิววงกลม เมื่อมีการนำข้อมูลออกจากคิววงกลมไปแล้วก็จะปรับเลื่อนให้ F มาชี้ที่ตำแหน่งแรกของคิววงกลมเช่นเดียวกัน ซึ่งก็จะวนเช่นนี้ไปเรื่อย ๆ



รูปที่ 4.9 การทำงานของคิววงกลมที่สร้างด้วยอาร์เรย์ขนาด 4 ช่อง

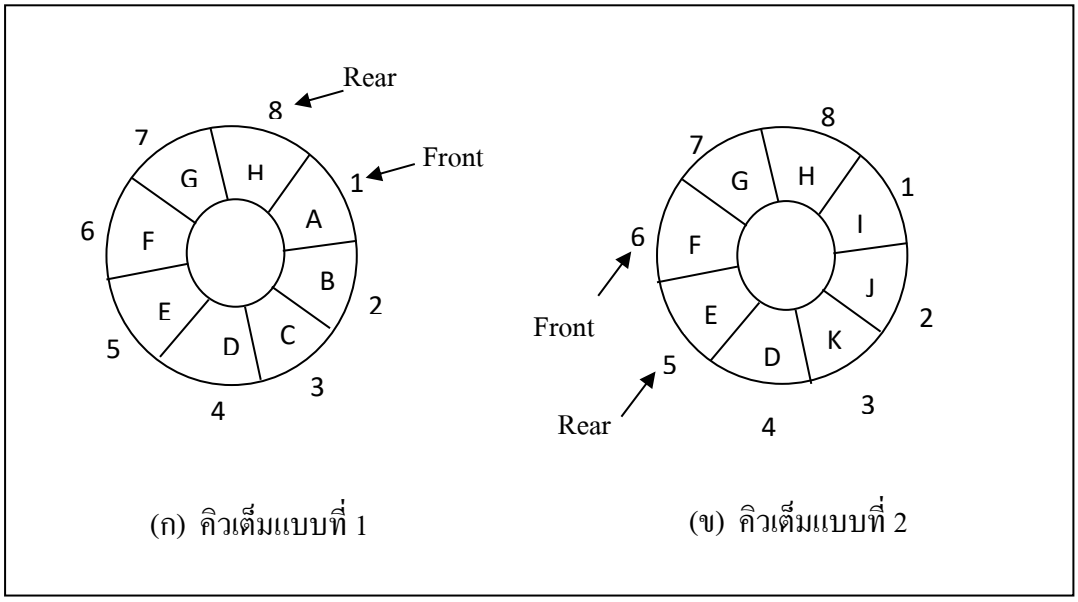
จากรูปที่ 4.9 จะเห็นได้ว่า เมื่อคิววงกลมมีการเชื่อมต่อด้านท้ายกับด้านหน้าเข้าหากัน ทำให้ Front และ Rear สามารถขยับเลื่อนวนไปได้เรื่อยๆ โดยเมื่อ Rear หรือ Front ชี้อยู่ตำแหน่งสุดท้ายของคิววงกลม ถ้าหากมีการ Enqueue หรือ Dequeue อีกก็จะปรับให้ Front และ Rear มาชี้ยังตำแหน่งแรกในคิว

การตรวจสอบคิวเต็มในคิววงกลม

การตรวจสอบว่าคิววงกลมเต็มหรือไม่นั้น จะพิจารณาค่าของ Front และ Rear ซึ่งการเกิดคิวเต็มของคิววงกลมมี 2 แบบเท่านั้นคือ

แบบที่ 1 เมื่อ Front อยู่ที่ 1 และ Rear อยู่ที่ตำแหน่งสุดท้ายของคิว ดังรูปที่ 4.10 (ก)

แบบที่ 2 เมื่อ Front อยู่ในตำแหน่งใด ๆ ภายในคิววงกลม (ไม่ใช่ตำแหน่งที่ 1) และ Rear อยู่ในตำแหน่งด้านหน้าของ Front 1 ตำแหน่ง นั่นคือ Rear = Front - 1 หรือ Rear + 1 = Front ดังรูปที่ 4.10 (ข)

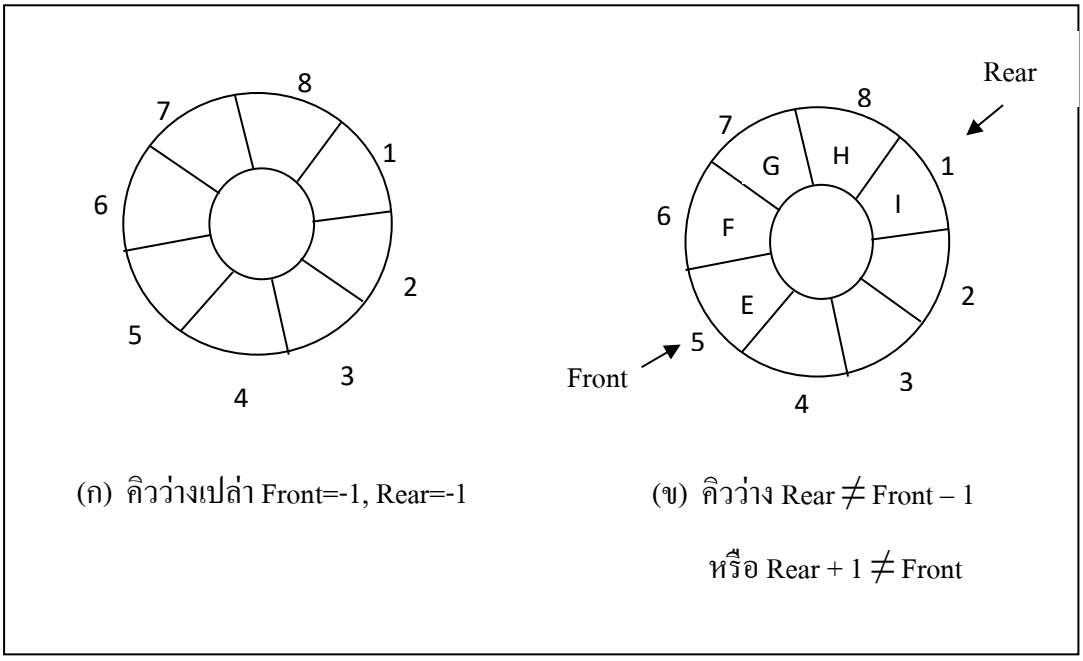


รูปที่ 4.10 แสดงลักษณะคิววงกลมเต็ม

กรณีต้องการตรวจสอบว่าคิววงกลมเต็มหรือไม่นั้น จำเป็นจะต้องนำลักษณะของค่า Front และค่า Rear ทั้งสองแบบมาใช้ในการตรวจสอบร่วมกัน นั่นคือนำทั้ง 2 แบบมาเขียนเป็นเงื่อนไขร่วมกัน

การตรวจสอบว่าคิวว่างในคิววงกลม

การตรวจสอบว่าคิวว่างในคิววงกลมจะมีลักษณะและวิธีการเหมือนกับการตรวจสอบคิวว่างในคิวธรรมดา คือ พิจารณาค่าของ Front และ Rear โดยถ้าค่า Front = -1 และ Rear = -1 ถือว่าคิววงกลมว่างเปล่าไม่มีข้อมูลอยู่เลย นอกจากนี้ ถ้า Rear ≠ Front - 1 หรือ Rear + 1 ≠ Front ก็ถือว่าคิววงกลมยังไม่เต็มยังมีพื้นที่ว่างอยู่สามารถทำการ Enqueue ข้อมูลได้ ดังรูปที่ 4.11



(ก) คิวว่างเปล่า Front=-1, Rear=-1

(ข) คิวว่าง Rear ≠ Front – 1

หรือ Rear + 1 ≠ Front

รูปที่ 4.11 แสดงลักษณะคิววงกลมว่าง

แบบฝึกหัดท้ายหน่วยที่ 4

ตอนที่ 1 จงตอบคำถามต่อไปนี้

1. จงอธิบายโครงสร้างข้อมูลแบบคิว
2. จงยกตัวอย่างคิวในชีวิตประจำวัน
3. จงอธิบายการสร้างคิวด้วยโครงสร้างข้อมูลแบบอาร์เรย์
4. การดำเนินการกับคิวมีกี่ลักษณะอะไรบ้าง
5. จงเขียนอัลกอริทึมการนำข้อมูลเข้าสู่คิว
6. จงเขียนอัลกอริทึมการนำข้อมูลออกจากคิว
7. จงอธิบายลักษณะของคิวเต็ม และวาดรูปประกอบ
8. จงเขียนภาพเหตุการณ์การดำเนินการกับคิวที่สร้างด้วยอาร์เรย์ขนาด 5 ช่อง ดังนี้
   Enqueue(20), Enqueue(17), Enqueue(5), Dequeue(), Dequeue(), Enqueue(10), Enqueue(9), Dequeue(), Enqueue(6)
9. จงอธิบายลักษณะและการทำงานของคิววงกลม
10. จงเขียนภาพเหตุการณ์การทำงานของคิววงกลมที่สร้างด้วยอาร์เรย์ขนาด 5 ช่อง ดังนี้
    Enqueue(3), Enqueue(5), Enqueue(4), Dequeue(), Enqueue(7), Enqueue(9), Dequeue(), Enqueue(11), Dequeue(), Dequeue(), Enqueue(15)
11. จงอธิบายการตรวจสอบคิวเต็มในคิววงกลม พร้อมวาดรูปลักษณะคิวเต็มประกอบ
12. จงอธิบายการตรวจสอบคิวว่างในคิววงกลม พร้อมวาดรูปลักษณะคิวว่างประกอบ

ตอนที่ 2 จงอธิบายคำศัพท์ที่เกี่ยวกับโครงสร้างข้อมูลแบบคิว

1. Enqueue.....

   .....

2. Dequeue.....

   .....

3. Front.....

   .....

4. Rear.....

   .....

5. Queue Underflow.....

   .....

6. Queue Overflow.....

   .....

7. Circular Queue.....

   .....

8. FIFO.....

   .....

9. Queues Structure.....

   .....