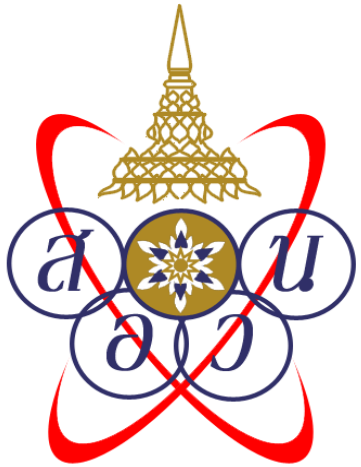


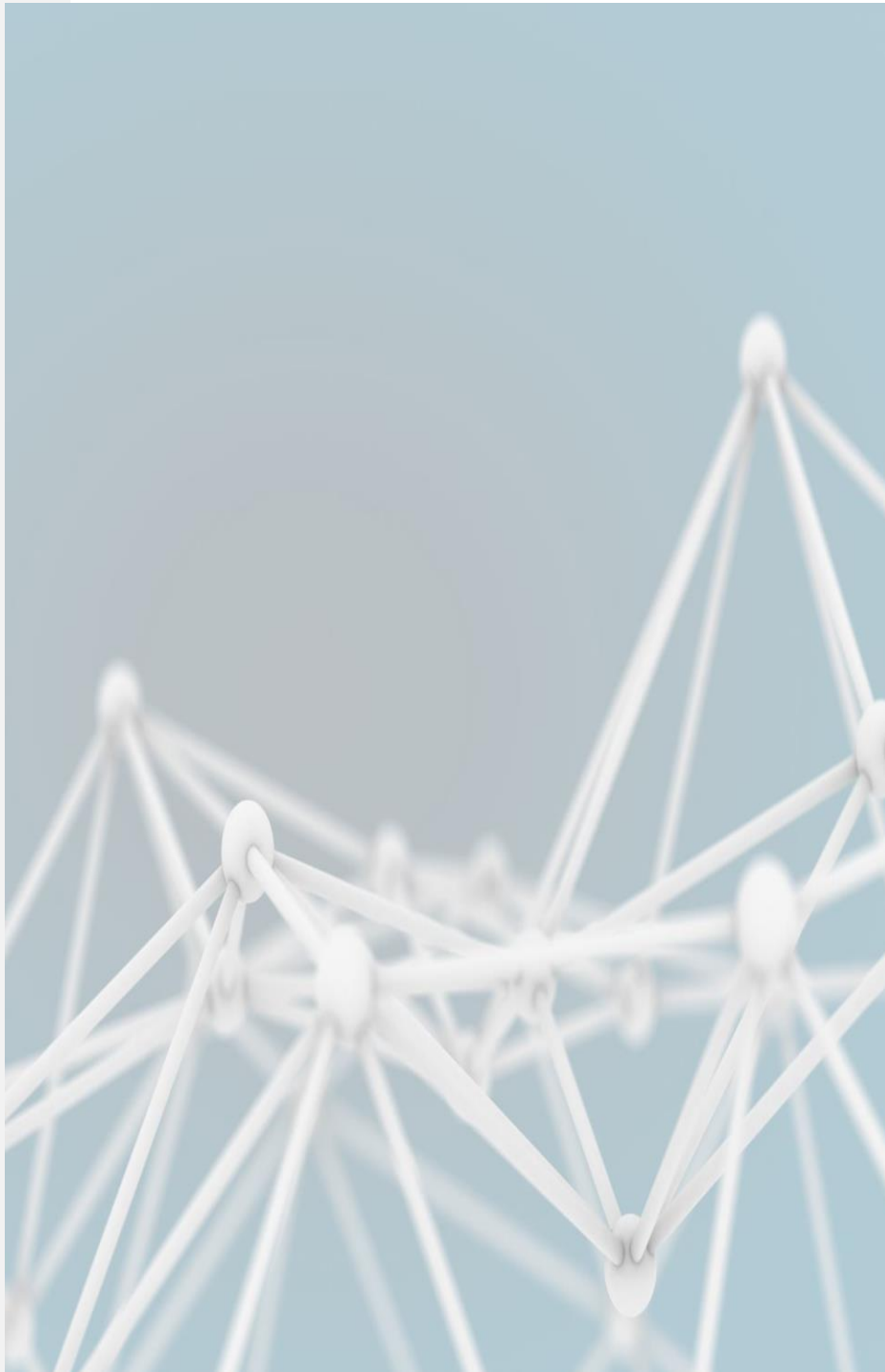
เอกสารประกอบการอบรม
ส่วนที่ 1 วิชาโครงสร้างข้อมูล

ค่ายคอมพิวเตอร์โอลิมปิก สอวน. ค่าย 2 2/2566
ศูนย์โรงเรียนสามเสนวิทยาลัย - มหาวิทยาลัยธรรมศาสตร์
ระหว่างวันที่ 18 มีนาคม – 3 เมษายน 2567

โดย
สาขาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์และเทคโนโลยี
มหาวิทยาลัยธรรมศาสตร์

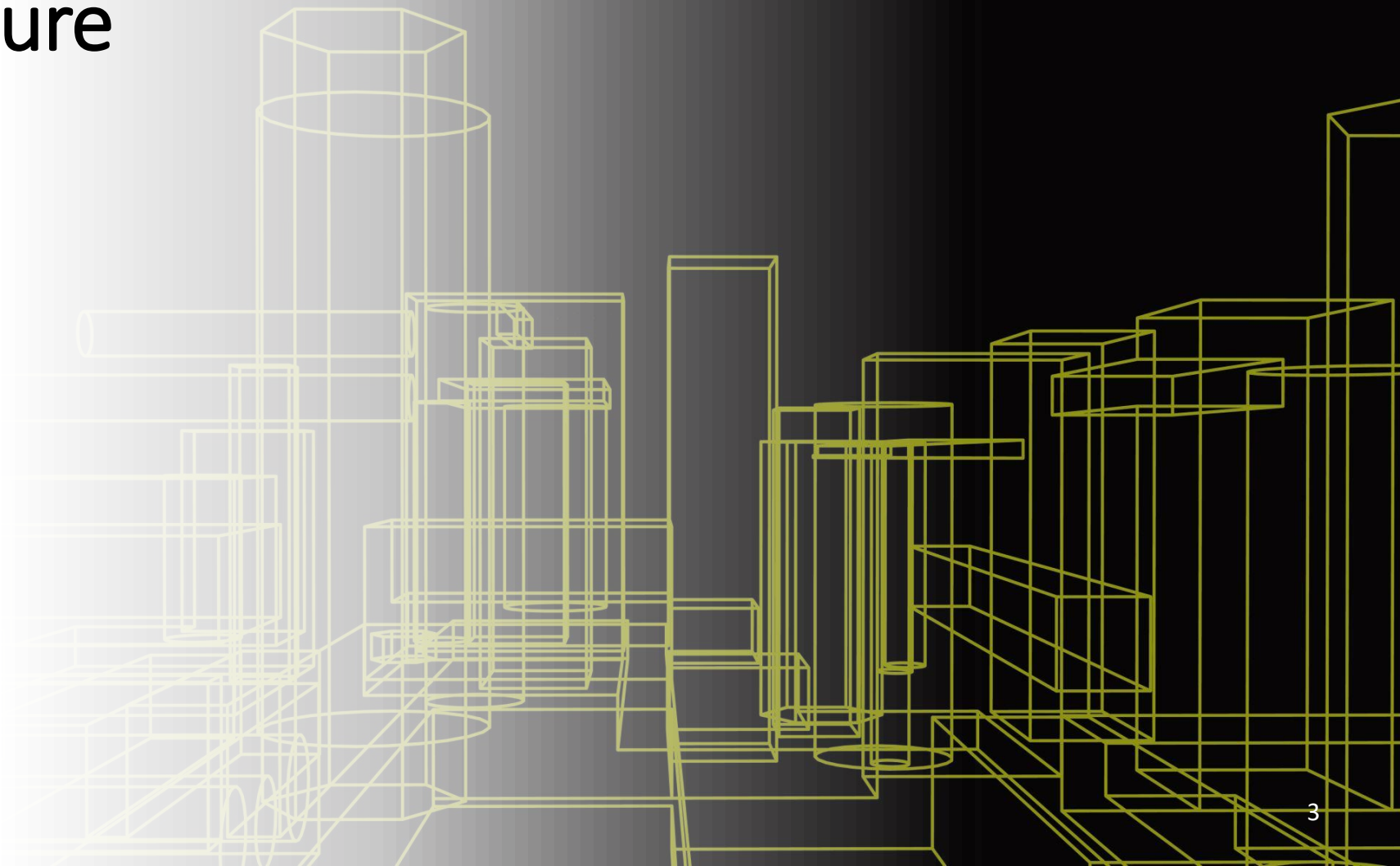
ส่วนที่ 1: โครงสร้างข้อมูล

ผศ.ดร.วิรัตน์ จารีวงศ์ไพบูลย์
อาจารย์ นุชชากร งามเสาวรส



Linear Data Structure

- List
- Stack
- Queue

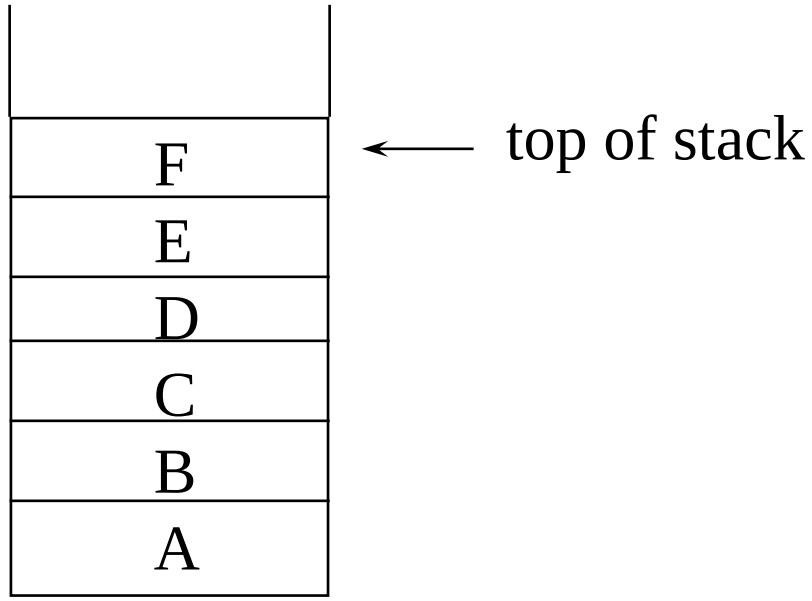


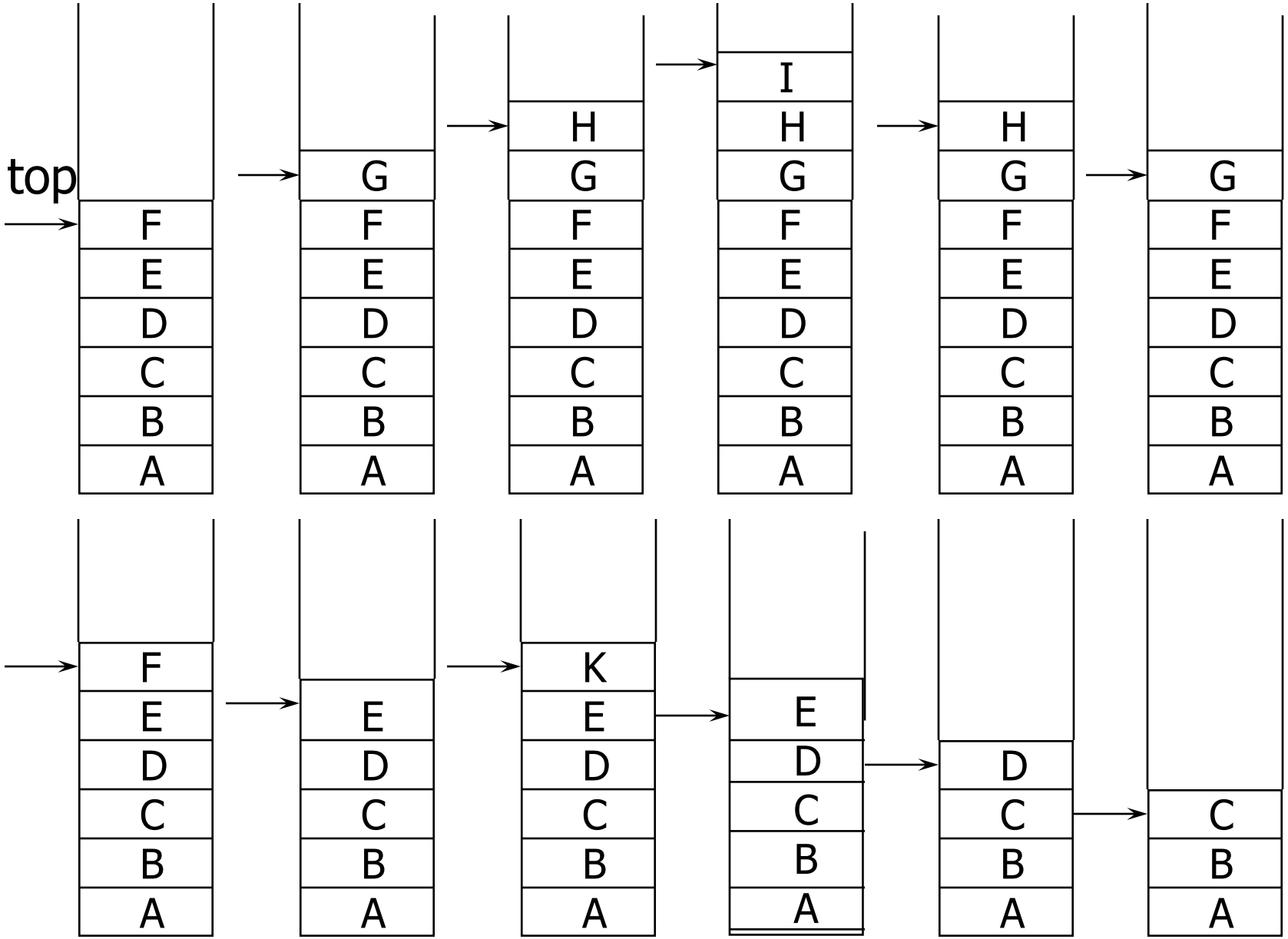
สแตก (Stack)

Abstract Data Type

สแตก (Stack)

- สแตก คือ กลุ่มของข้อมูลที่จัดเรียงกันอย่างมีลำดับ โดยการเพิ่มหรือลบข้อมูลจะกระทำที่จุดจุดเดียวกันคือที่ด้านบนของสแตกที่เรียกว่า top ของสแตก
- ข้อมูลที่ถูกเพิ่มเข้าไปในสแตกตัวสุดท้ายจะเป็นข้อมูลตัวแรกที่ถูกลบออกจากสแตก ทำให้ในบางครั้งเราจึงเรียก สแตกว่าเป็นลิสต์แบบ *last-in, first-out* (LIFO)




top
F
E
D
C
B
A
G
F
E
D
C
B
A
H
G
F
E
D
C
B
A
I
H
G
F
E
D
C
B
A
H
G
F
E
D
C
B
A
G
F
E
D
C
B
A
F
E
D
C
B
A
E
D
C
B
A
K
E
D
C
B
A
E
D
C
B
A
D
C
B
A
C
B
A

Operations ทั่วไปของสแตก

กำหนดให้ *s* แทนสแตกและ *i* แทนข้อมูล

- push(s, i) : เพิ่มข้อมูล i ที่ตำแหน่ง top ในสแตก s
- pop(s) : ลบข้อมูลที่ตำแหน่ง top ของสแตกและส่งกลับข้อมูลนั้น
- isEmpty(s) : ตรวจสอบว่าสแตก s ว่างหรือไม่
- stackTop(s) : ดูข้อมูลที่ตำแหน่ง top ของสแตกโดยก็อบปี้ข้อมูลออกมา และไม่ลบข้อมูลออกจากสแตก

Implementation of Stack

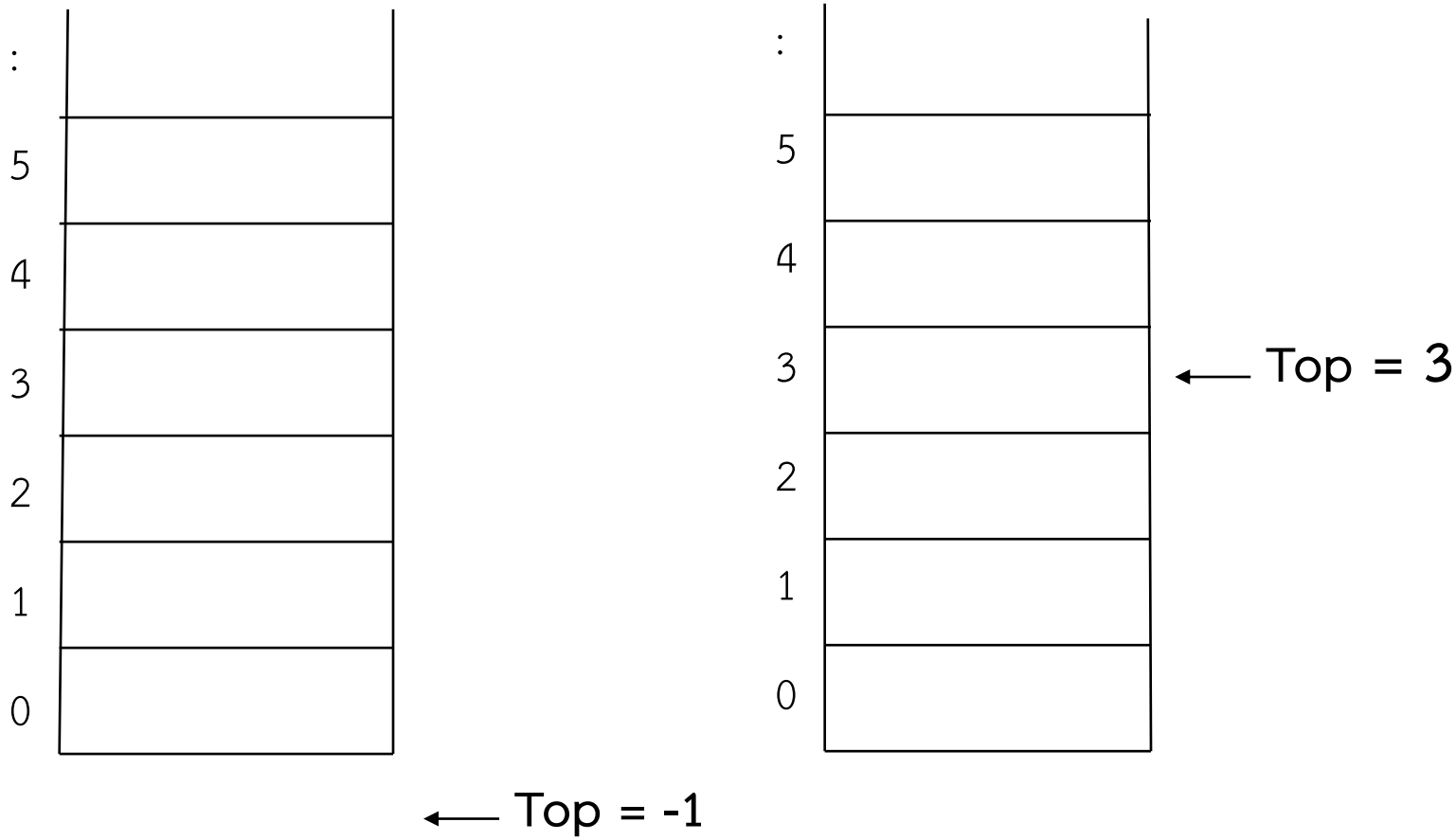
การสร้างสแตกด้วยอะเรย์

การสร้าง stack ในภาษา C สามารถกำหนดเป็นโครงสร้างที่ประกอบไปด้วยข้อมูลสองส่วนคือ

- ตำแหน่งบนสุดของแสตก (top)
- และอะเรย์ที่ใช้เก็บข้อมูล (item)

```
#define STACKSIZE 100
struct stack {
    int top;
    int items[STACKSIZE];
};
struct stack s;
```

สถานะของสแตก



(a) แสตกว่าง

(b) แสตกที่มีข้อมูล 4 ตัว

ฟังก์ชัน isEmpty()

การตรวจสอบ stack ว่าเป็น stack ว่างหรือไม่ สามารถทำได้โดยการ ตรวจสอบค่า top มีค่าเป็น -1 หรือไม่

```
int isEmpty(struct stack *ps)
{
        if (ps->top == -1)
                return (1);
        else
                return (0);
}  /* end isEmpty */
```

ฟังก์ชัน push()

เพิ่มข้อมูล x ที่ตำแหน่ง top ในสแตก s

```
void push(struct stack *ps, int x)
{
    if (ps->top == STACKSIZE-1) {
        printf("%s", "stack overflow");
        exit(1);
    }
    else
        ps->items[++(ps->top)] = x;
    return;
}  /* end push */
```

ฟังก์ชัน pop()

ลบข้อมูลที่ตำแหน่ง top ของสแตก
และส่งกลับข้อมูลนั้น

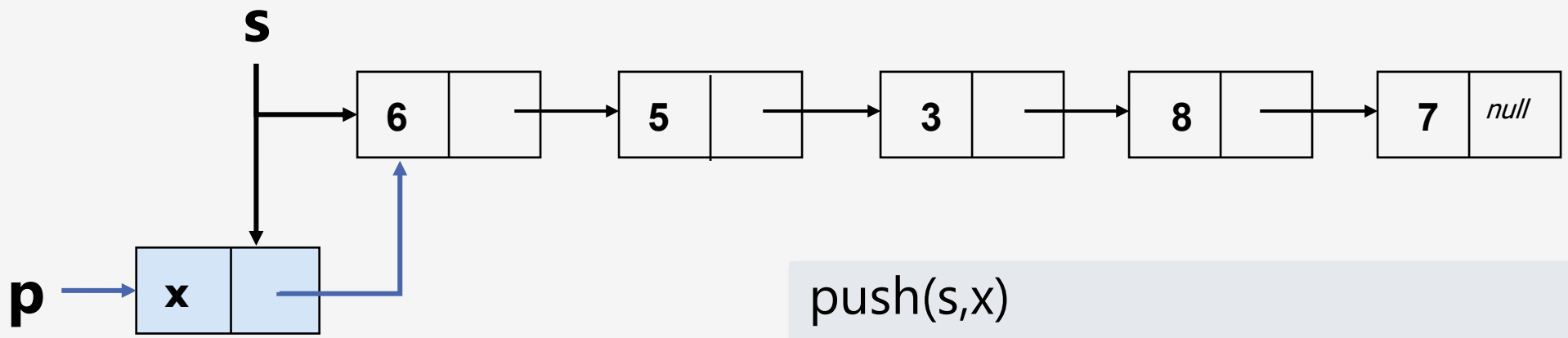
```
int pop(struct stack *ps)
{
    if (isEmpty(ps))  {
        printf("%s", "stack underflow");
        exit(1);
    }  /* end if */
    return (ps->items[ps->top--]);
}  /* end pop */
```

ฟังก์ชัน stackTop()

ดูข้อมูลที่ตำแหน่ง top ของสแตกโดยคัดลอกข้อมูลออกมา และไม่ลบข้อมูลออกจากสแตก

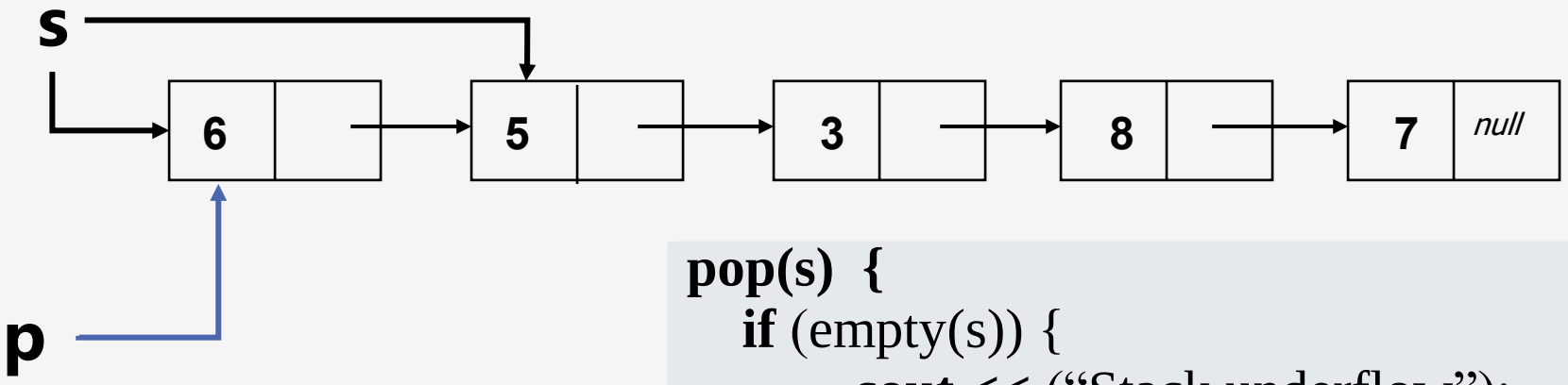
```
int pop(struct stack *ps)
{
    if (isEmpty(ps))  {
        printf("%", "stack underflow");
        exit(1);
    }  /* end if */
    return (ps->item[ps->top]);
}  /* end pop */
```

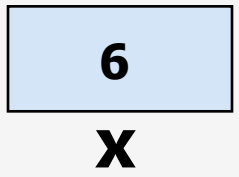
การสร้างสแตกด้วยลิงค์ลิสต์ – push()




```
push(s,x)
{
        p = getnode();
        info(p) = x;
        next(p) = s;
        s = p;
}
```

การสร้างสแตกด้วยลิงค์ลิสต์ – pop()






```
pop(s)  {
    if (empty(s)) {
            cout << (“Stack underflow”);
            exit(1);
     }else {
            p = s;
            s = next(p);
            x = info(p);
           freenode(p);
      }
      return x;
}
```

การใช้สแตกในการแก้ปัญหา

ตัวอย่าง : พิจารณานิพจน์ทางคณิตศาสตร์ดังต่อไปนี้ :

{[A+B] - [C * (D - E)]}

เราต้องตรวจสอบว่า

1. จำนวนวงเล็บเปิดเท่ากับจำนวนวงเล็บปิดหรือไม่
2. ทุกๆ วงเล็บปิดจะต้องมีวงเล็บเปิดที่คู่กันอยู่ก่อนหน้าเสมอหรือไม่
3. วงเล็บเปิดและวงเล็บปิดเป็นวงเล็บชนิดเดียวกันหรือไม่

วิธีการแก้ปัญหาโดยไม่ใช้สแตก

เราอาจใช้วิธีการนับวงเล็บ โดยนำจำนวนวงเล็บเปิดที่นับได้ลบด้วยจำนวนวงเล็บปิดที่นับได้

- ถ้าผลลัพธ์เป็นศูนย์ แสดงว่าจำนวนวงเล็บเปิดเท่ากับจำนวนวงเล็บปิด
- ค่าที่ได้จากการนับ ณ ขณะใดๆ ต้องไม่เป็นค่าลบ เนื่องจากเราเอาจำนวนวงเล็บเปิดลบด้วยจำนวนวงเล็บปิด ถ้าค่าเป็นลบแสดงว่าวงเล็บปิดมาก่อนวงเล็บเปิด

ตัวอย่าง :

```
7 - ( ( x * ( ( x + y ) / ( j - 3 ) ) + y ) / ( 4 - 2.5 ) )
0 0 1 2 2 2 3 4 4 4 4 3 3 4 4 4 4 3 2 2 2 1 1 2 2 2 2 1 0
```

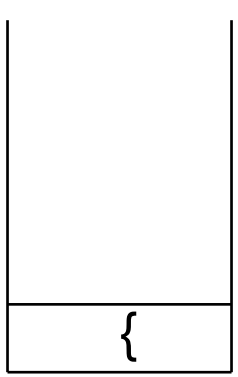
```
( ( A + B )          ( A + B ) ) - ( C + D
1 2 2 2 2 1          1 1 1 1 0 -1 -1 0 0 0 0
```

ถ้ามีการใช้วงเล็บหลายแบบ เช่น

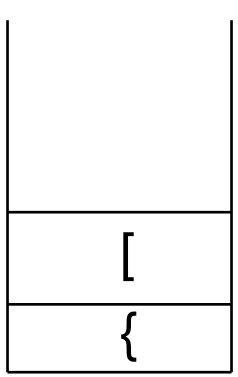
{[A+B] - [C*(D-E)]}

จะแก้ปัญหาอย่างไร เพราะวิธีนับอาจจะไม่ได้ผล

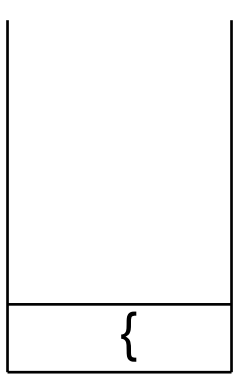
การแก้ปัญหา ==> กรณีมีการใช้วงเล็บหลายแบบ เช่น {[A+B] - [C*(D-E)]}



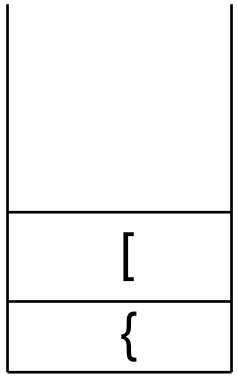
{...



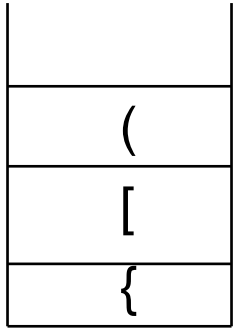
{[...



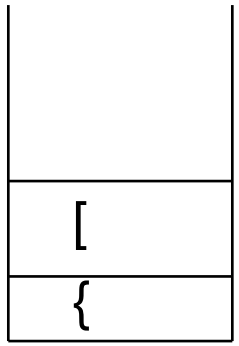
{[A+B]...



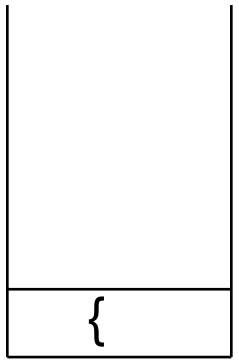
{[A+B]-[...



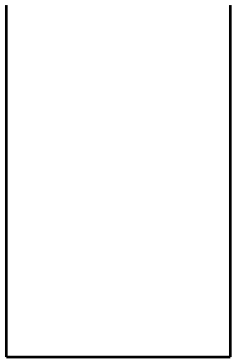
{[A+B]-[C*(...



{[A+B]-[C*(D-E)...



{[A+B]-[C*(D-E)]...



{[A+B]-[C*(D-E)]}

อัลกอริทึม

```
valid = true;
s = the empty stack;
while (we have not read the entire string) {
        read the next symbol (symb ) of the string ;
        if  (symb == '(' ) || (symb == '[' ) || (symb == '{' )
            push(s, symb);
        if  (symb == ')' ) || (symb == ']' ) || (symb == '}' )
             if  (empty(s))
                 valid = false;
            else {
                  i = pop(s);
            if ( i  is not the matching opener of symb)
                 valid = false;
            }
}
if  (!empty(s)) valid = false;
```

คิว (Queue)

คิว (QUEUES)

- **คิว** เป็นกลุ่มหรือลิสต์ของข้อมูลที่เรียงกันอย่างมีลำดับ การเพิ่มข้อมูลในคิวจะเพิ่มที่ด้านท้ายหรือที่เรียกว่า rear ของคิว ส่วนการลบข้อมูลจะลบที่ด้านหน้าหรือ front ของคิว
- จากคุณลักษณะเฉพาะดังกล่าว ในบางครั้งเราจึงเรียกคิวว่าเป็น first-in, first-out หรือ FIFO list

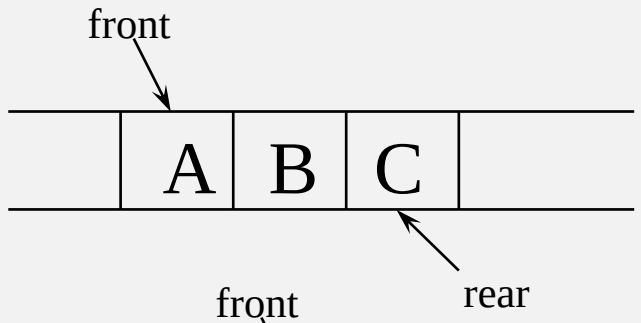
Operations ทั่วไปของคิว

enqueue(*q*,*x*) เพิ่มข้อมูล x ที่ตำแหน่งท้ายของคิว *q*

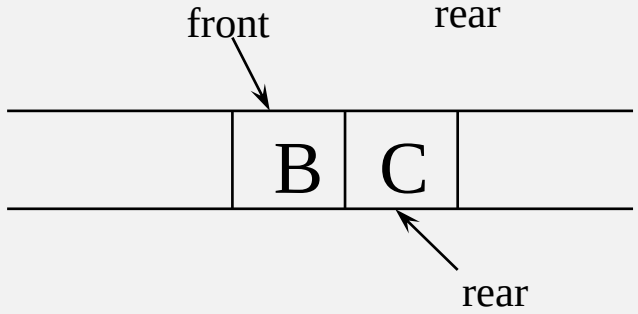
x = dequeue(*q*) ลบข้อมูลที่ตำแหน่งแรก (front) ของคิว q แล้วส่งกลับค่าให้ตัวแปร x

empty(*q*) ตรวจสอบว่าคิว q เป็นคิวว่างหรือไม่ (คิวว่างคือคิวที่ไม่มีข้อมูลเลย)

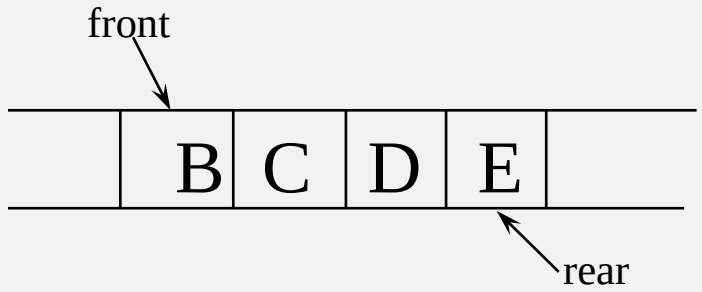
ตัวอย่างคิว



enqueue(q, A);
enqueue(q, B);
enqueue(q, C);



x = dequeue(q);



enqueue(q, D);
enqueue(q, E);

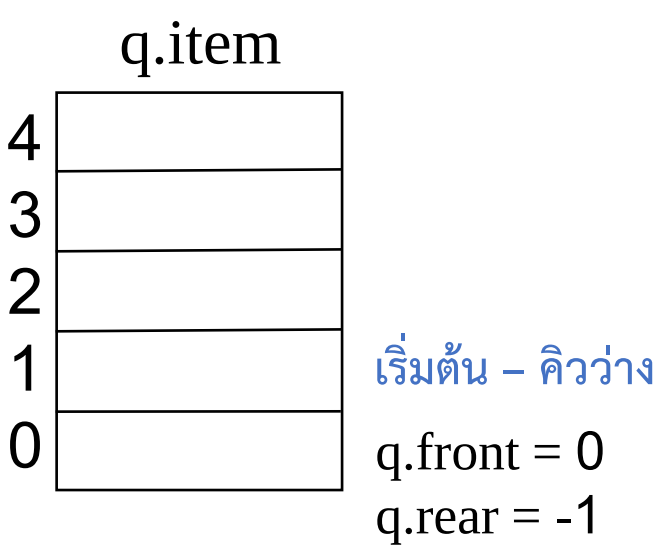
การสร้างคิวด้วยอะเรย์

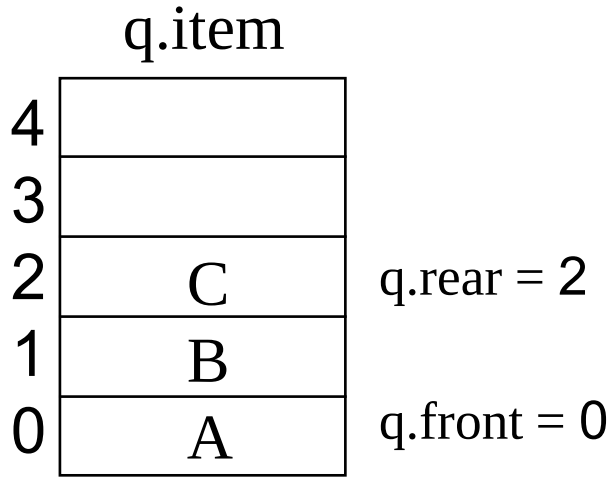
```
#define MAXQUEUE 5
struct queue {
    int items[MAXQUEUE];
    int front;
    int rear;
}q;
```

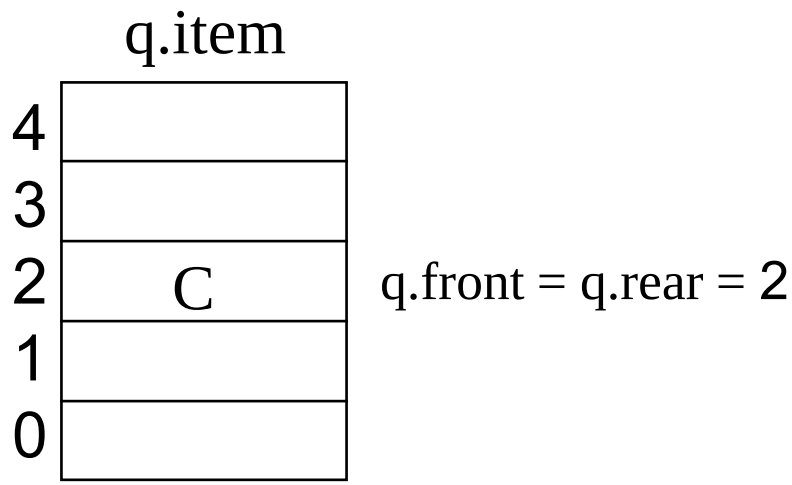
enqueue(q, x) สามารถเขียนได้ดังนี้ q.items [++q.rear] = x;

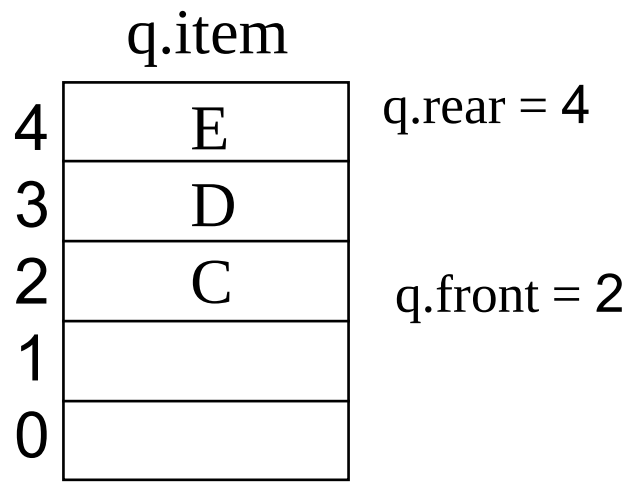
x = dequeue(q) สามารถเขียนได้ดังนี้ x = q.item [q.front++];

ตัวอย่างคิวที่ใช้อะเรย์





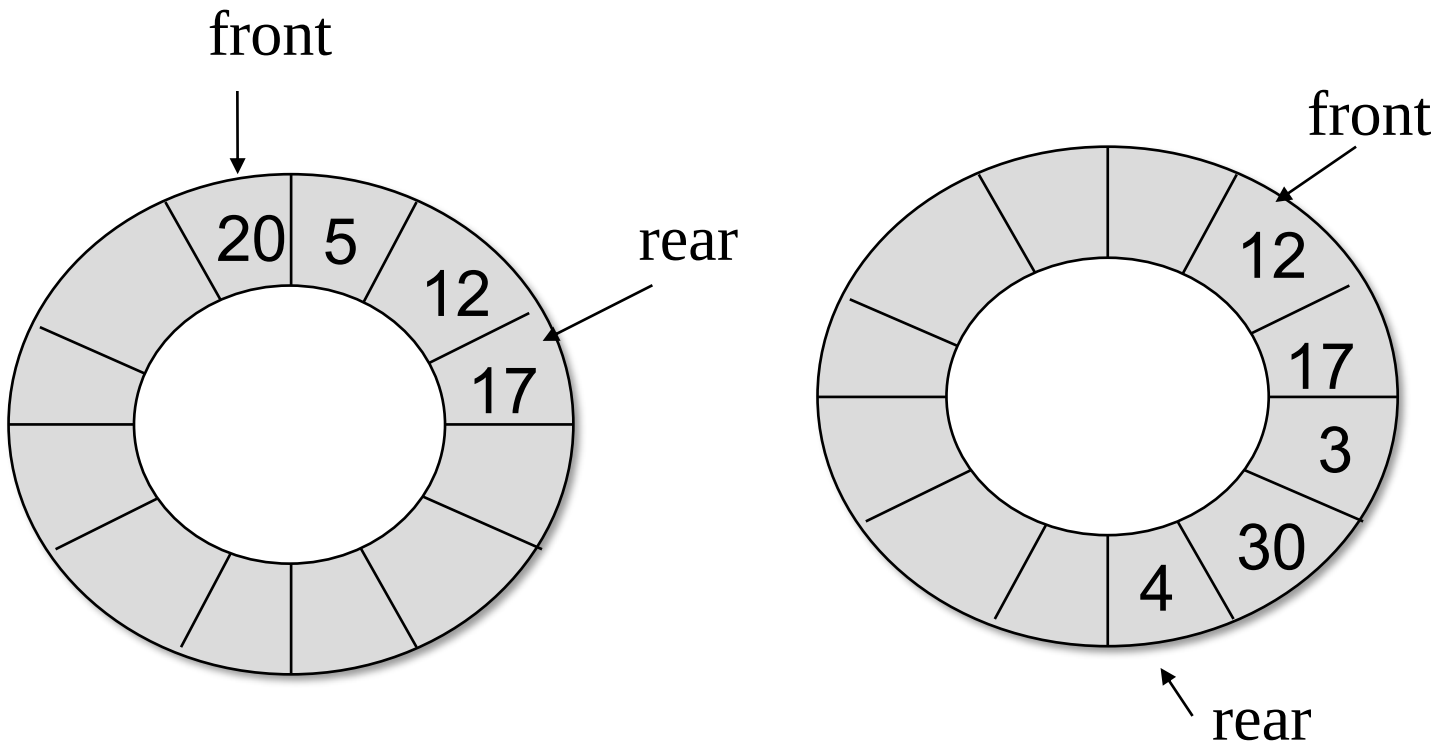




- จำนวนข้อมูล = q.rear - q.front + 1
- แล้วเราจะ INSERT ข้อมูล F เข้าไปใน Q ได้อย่างไร ? เพราะ q.rear = MAXQUEUE -1

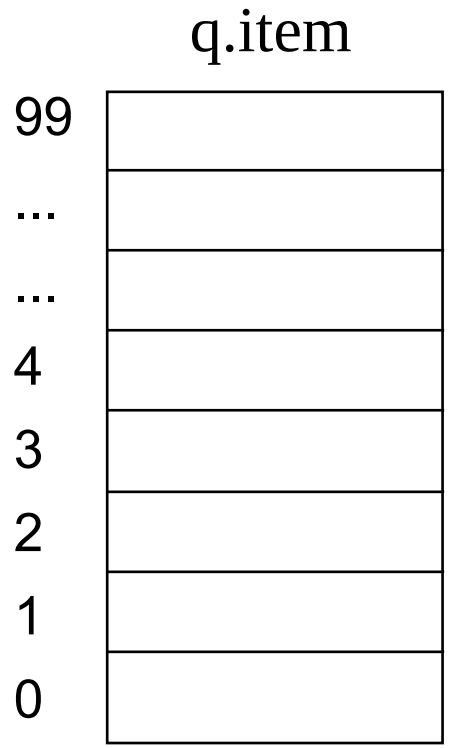
มองอะเรย์ที่เก็บคิวเป็นวงกลม

A Circular Queue



โครงสร้างของ Circular queue

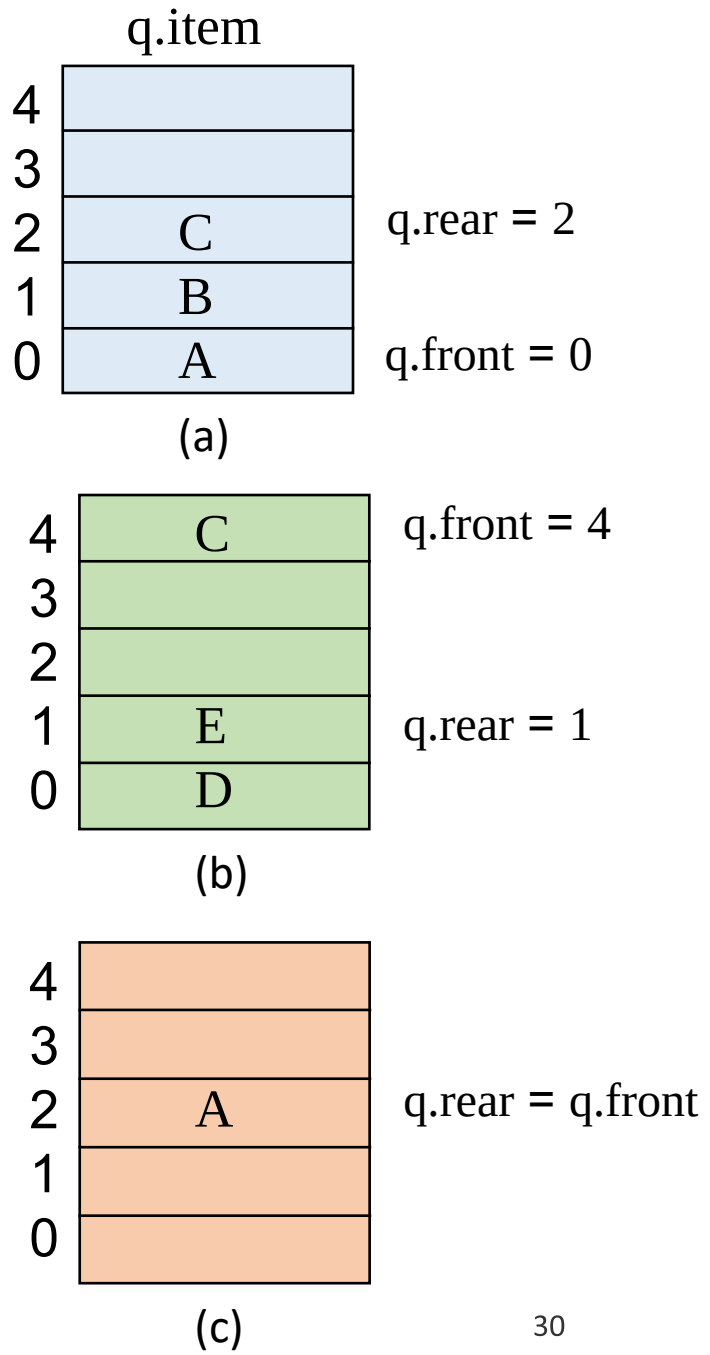
```
#define  MAXQUEUE 100
struct queue {
        int item[MAXQUEUE];
        int front;
        int rear;
        int count;
};

struct queue q;
```



เริ่มต้นกำหนดค่า : q.rear = -1 q.front = -1 q.count = 0

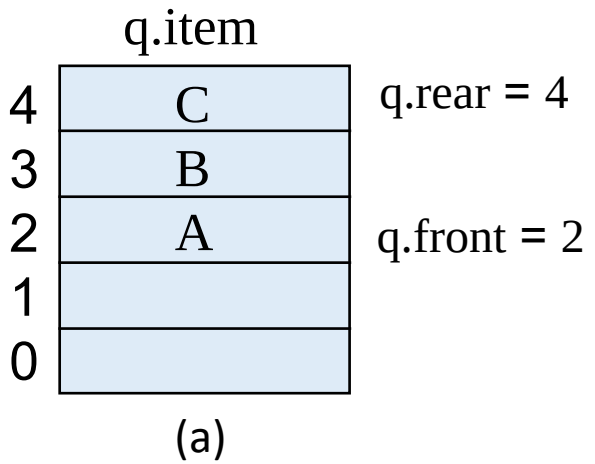
คิว: การลบข้อมูลออกจากคิว

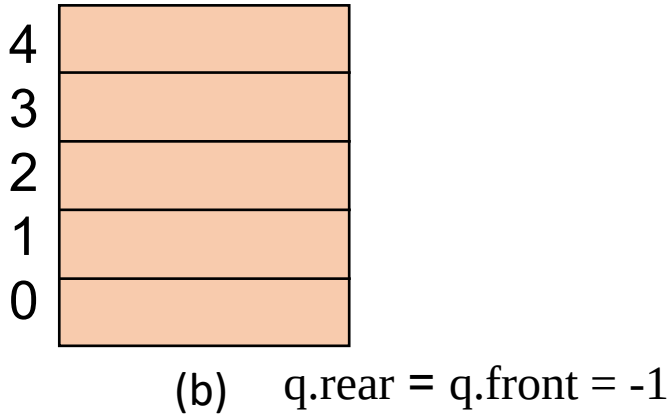
```
int dequeue(struct queue *pq)
{     int frontItem;
       if (isEmpty(pq))  {
            Cout <<(“queue underflow”);
            exit(1);
       }  /* endif */
       frontItem = pq->item[pq->front];
       (pq->front)++;

       if (pq->front = = MAXQUEUE)
            pq->front = 0;

       if (pq->count == 1)
              pq->front = pq->rear = -1

       (pq->count)--;
       return (frontItem);
}
```



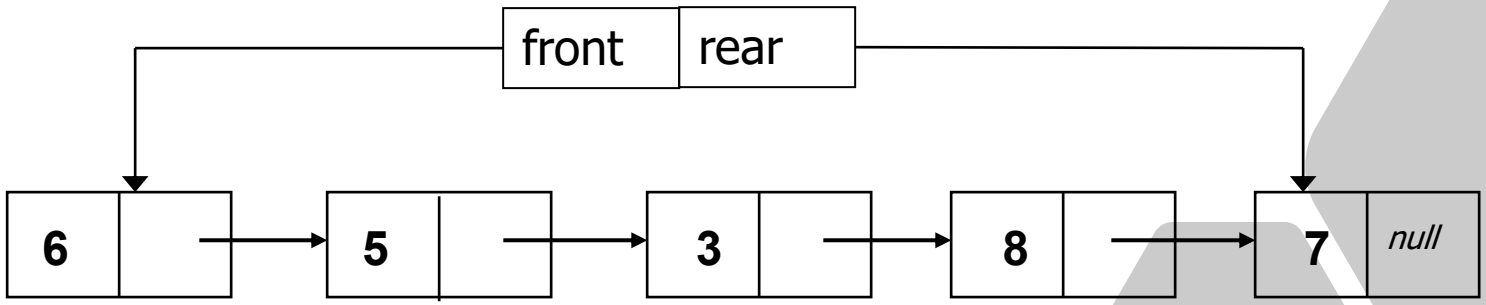
คิว: การใส่ข้อมูลเข้าคิว

```
void enqueue(int x , struct queue *pq)
{
      /*queue is full */
     if ( pq->count = = MAXQUEUE )
          return;
      (pq->rear)++;
      if (pq->rear = = MAXQUEUE)
          pq->rear = 0;

      pq->item[pq->rear] = x;

      if (pq->count = = 0)
          pq->front = 0

      pq->count++;

      return;
}
```





การสร้างคิวด้วยลิ้งค์ลิสต์




```
struct queue {
    NODEPTR front, rear;
};
struct queue q;
```

สร้างคิวด้วยลิ้งค์ลิสต์

```
void enqueue(int x, struct queue *pq)
{
      NODEPTR p;

      p = getnode( );
      p->info = x;
      p->next = NULL;
      if (pq->rear = = NULL)
            pq->front = p;
      else
            (pq->rear)->next = p;
            pq->rear = p;
      pq->rear = p;
}  /* end insert  */
```

```
int dequeue(struct queue *pq)
{
      NODEPTR p;
       int x;
      if (empty(pq)) {
            printf("queue underflow\n");
            exit(1);
      }
      p = pq->front;
      x = p->info;
      pq->front = p->next;
      if (pq-front = = NULL)
            pq->rear = NULL;
      freenode(p);
      return(x);
}  /*  end remove  */
```

Priority Queue

- คือโครงสร้างของคิวที่ตำแหน่งของข้อมูลในคิวไม่ได้บอกตำแหน่งของการเพิ่มหรือลบข้อมูล ข้อมูลแต่ละตัวจะมีค่า priority ที่ต่างกันและค่านี้จะเป็นตัวบอกลำดับการออกจากคิว เราสามารถแบ่งประเภทของ Priority Queue ได้เป็น 2 แบบ คือ
- Ascending priority queue คิวของข้อมูลที่ข้อมูลที่มีค่าน้อยที่สุดหรือมี priority น้อยที่สุด จะถูกลบออกจากคิวก่อน
- Descending priority queue คิวของข้อมูลที่ข้อมูลที่มีค่ามากที่สุดหรือมี priority มากที่สุด จะถูกลบออกจากคิวก่อน

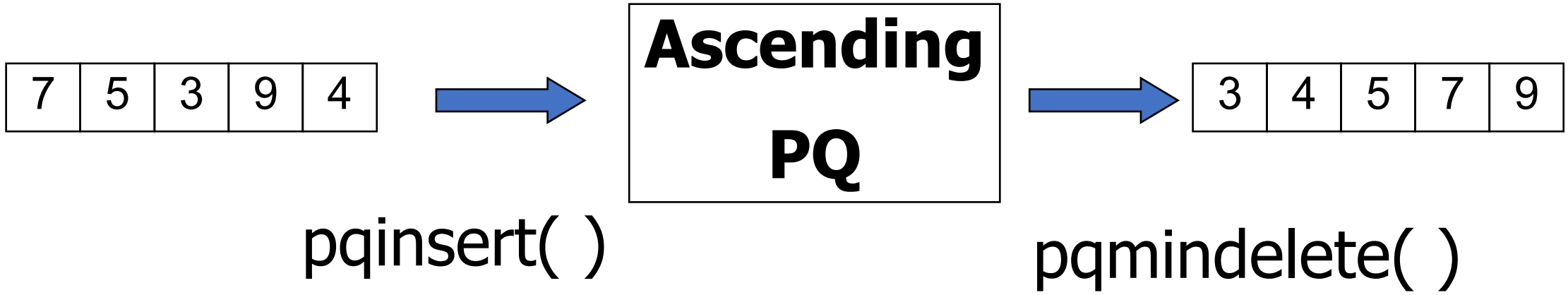
Operations ของ Priority Queue

- กำหนดให้ x เป็นข้อมูล pq เป็น priority queue, apq เป็น ascending priority queue และ dpq เป็น descending priority queue

pqinsert(pq, x)	เพิ่มข้อมูล x ใน pq โดยการเพิ่มเราไม่ได้สนใจว่า ข้อมูลในคิวจะเรียงลำดับอย่างไร
dpqmaxdelete(dpq)	เป็น operation สำหรับ **descending priority queue** โดยทำการลบข้อมูลที่มีค่ามากที่สุดหรือมีค่า priority สูงสุดออกจากคิว
apqmindelete(apq)	เป็น operation สำหรับ **ascending priority queue** โดยทำการลบข้อมูลที่มีค่าน้อยที่สุดหรือมีค่า priority ต่ำสุดออกจากคิว

การเรียงลำดับข้อมูลด้วย Priority Q

- การใช้ operation ของ priority queue ในการเลือกหยิบข้อมูลน้อยที่สุดหรือมากที่สุดในคิวออกทีละตัว และเลือกวางข้อมูลที่ออกจากคิวในอะเรย์ ซึ่งถ้าวางในอะเรย์จากตำแหน่งแรกไปตำแหน่งท้าย หรือเลือกวางข้อมูลจากตำแหน่งท้ายมาตำแหน่งแรก ก็จะได้ลำดับข้อมูลที่แตกต่างกันทั้งจากน้อยไปมากหรือมากไปน้อย



ตัวอย่าง : การเรียงลำดับข้อมูลด้วย dpq

กำหนดให้ dpq เป็น descending priority queue ว่าง

x เป็นอะเรย์ของข้อมูลแบบไม่มีลำดับ ที่มีข้อมูล n จำนวน

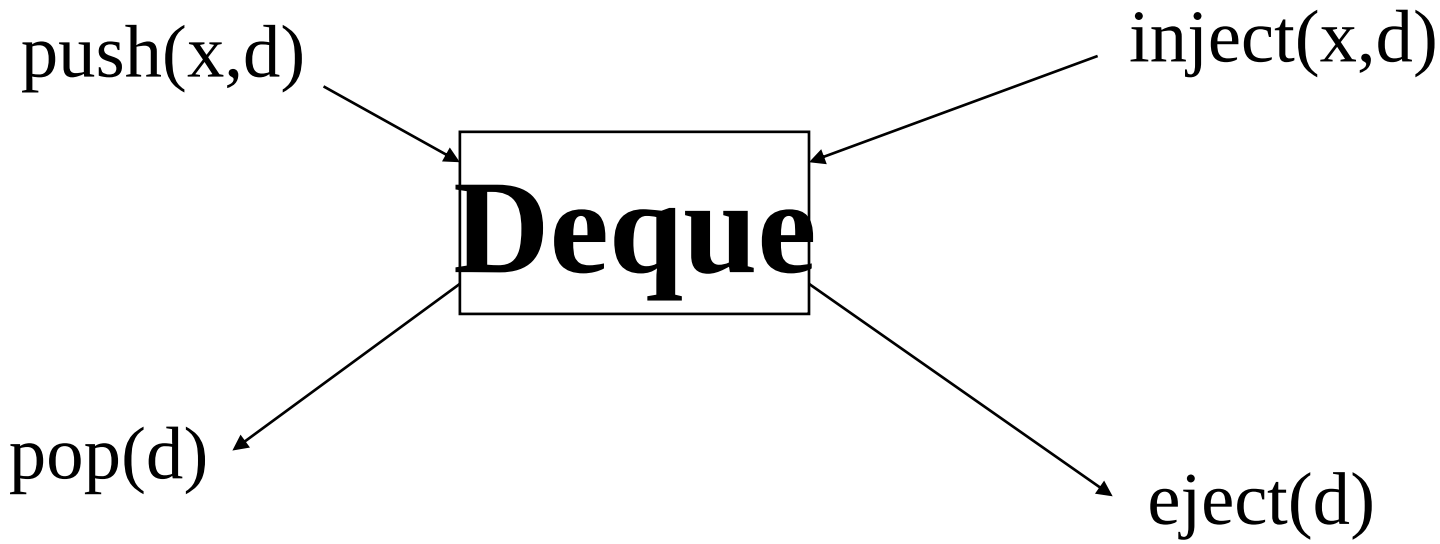
ใส่ข้อมูลในอะเรย์ x ลงใน dpq จนหมด ดังนี้

```
for (i = 0; i < n; i++)
    pqinsert(x[i]);
```

ลบข้อมูลจาก dpq แล้วใส่กลับคืนที่อะเรย์โดยเลือกวางจากตำแหน่งท้ายไปตำแหน่งแรก

```
for (i = n-1; i >= 0; i--)
    x[i] = dpqmaxdelete( );
```

Deque (Double - Ended Queue)



DEQUE

deque คือโครงสร้างข้อมูลที่ประกอบไปด้วยลิสต์ของข้อมูลและ operations ดังต่อไปนี้

- push(x) : การใส่ข้อมูลใหม่ไปที่ด้านหน้าของคิว
- pop() : การลบข้อมูลตัวแรกของคิว
- inject(x) : การใส่ข้อมูลใหม่ไปที่ท้ายของคิว.
- eject() : การลบข้อมูลที่ด้านท้ายของคิว

ลักษณะของ deque ก็เป็นการผสมระหว่างคิวกับสแตกนั่นเอง