



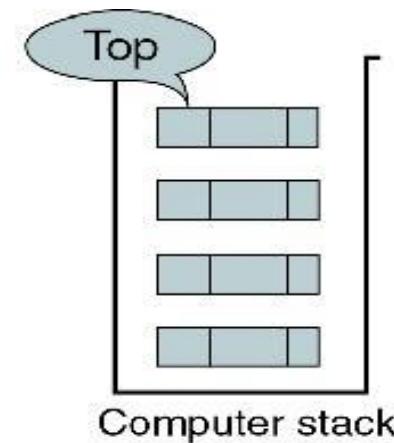
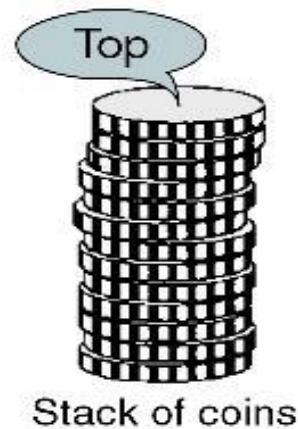
Chapter 3

Stack and Queue

Stack

Stack

- เป็นโครงสร้างข้อมูลแบบ linear list ประเภทหนึ่ง โดยที่
 - จะเพิ่มและลบข้อมูลจาก stack ที่ส่วน top ของ stack เท่านั้น
 - เป็นโครงสร้างแบบ LIFO (Last In First Out) -> เข้าทีหลัง ออกก่อน

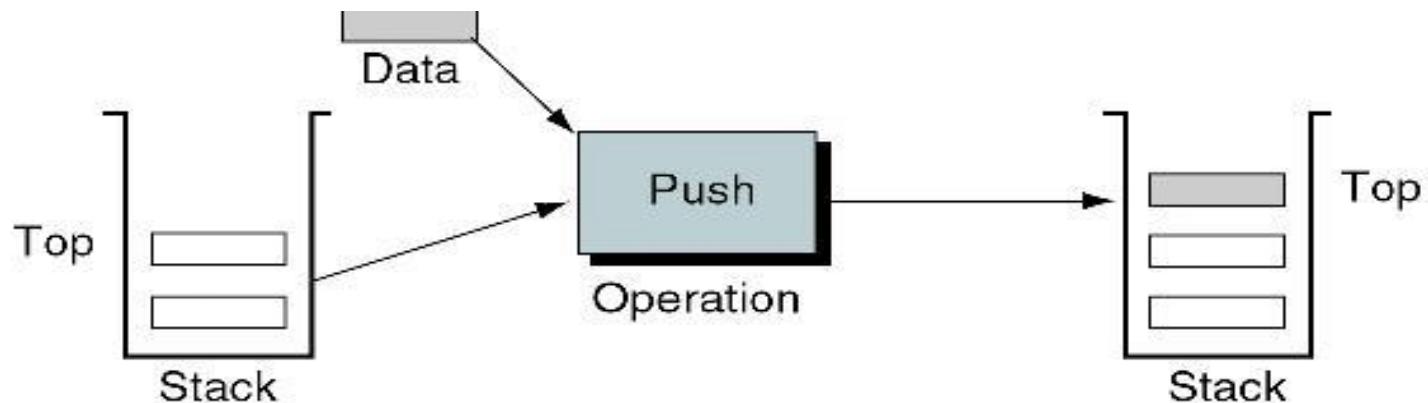


Basic Stack Operations

- Push : ใช้เพิ่มข้อมูลเข้าไปใน stack
- Pop : ใช้อ่านข้อมูลออกจาก stack
- Stack Top : ส่งค่าข้อมูลที่อยู่ณ ตำแหน่ง top ของ stack

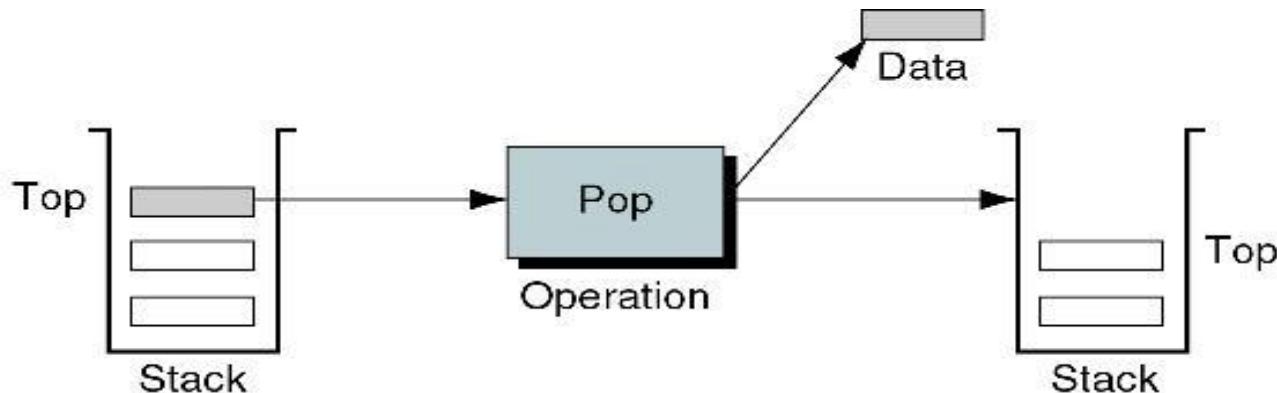
Basic Operation : Push

- เราสามารถเพิ่มข้อมูลเข้าไปที่ส่วน top ของ stack ข้อมูลที่เพิ่มเข้าไป ก็จะอยู่ต่ำแห่ง top ของ stack แทน
- ปัญหา：
 - ถ้า stack เต็ม การเพิ่มข้อมูลลงไปจะทำให้เกิดสถานะ Overflow



Basic Operation : Pop

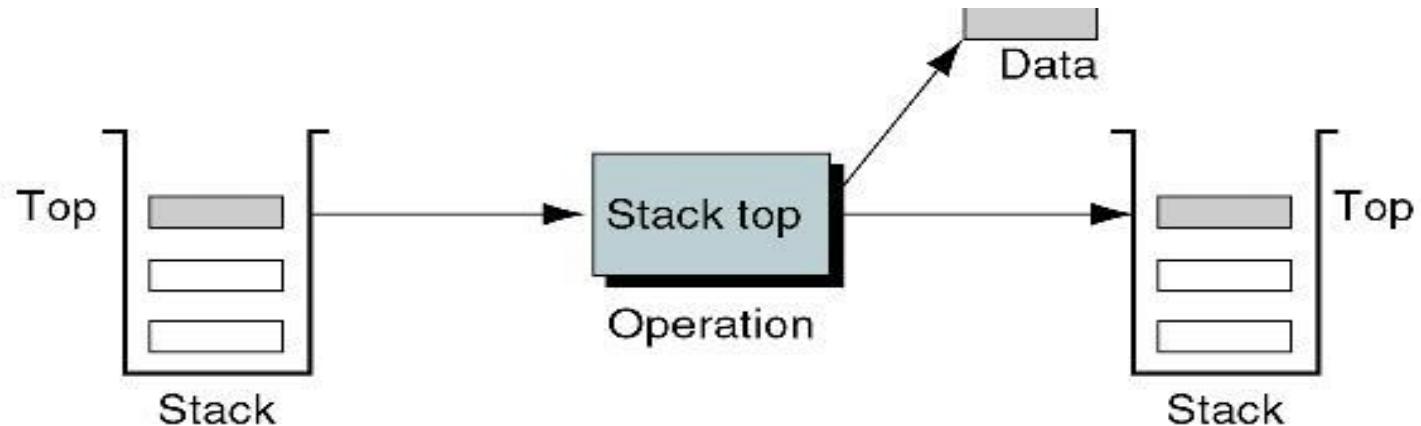
- เป็นการลบข้อมูลที่ตำแหน่ง top ของ stack และคืนค่าข้อมูลนั้น ทำให้ข้อมูลก่อนหน้าจะอยู่ตำแหน่ง top ของ stack แทน
- ปัญหา：
 - ถ้าไม่มีข้อมูลใน stack หรือ stack ว่าง การลบข้อมูลจะทำให้เกิดสถานะ Underflow



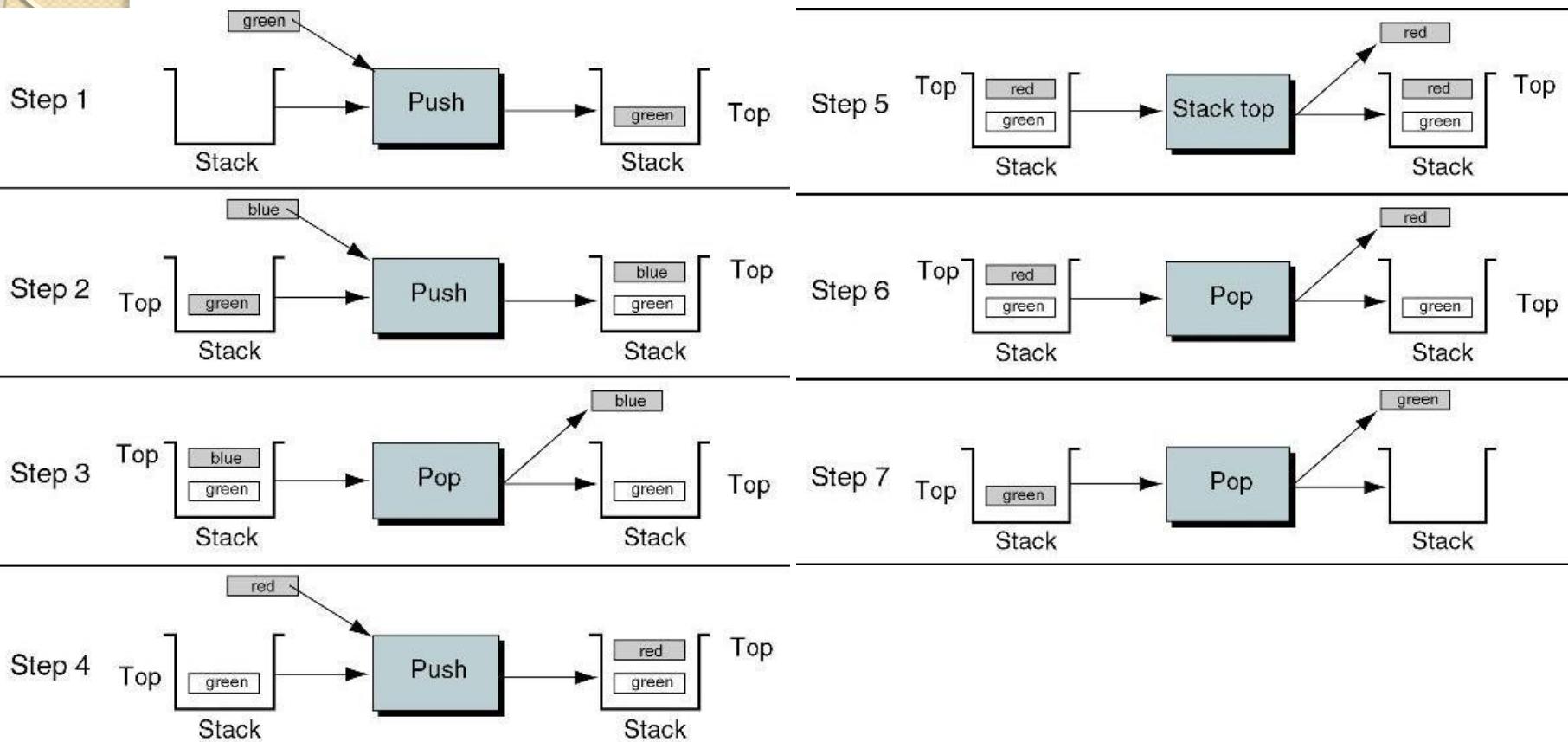
Basic Operation : Stack Top

return

- คืนค่าข้อมูล ณ ตำแหน่ง top ของ stack และไม่ได้ลบข้อมูลนั้น
- ปัญหา：
 - ถ้า stack ว่าง การเอาข้อมูลที่ตำแหน่ง top จะเกิดสถานะ Underflow

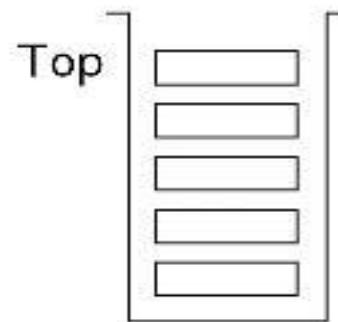


Example

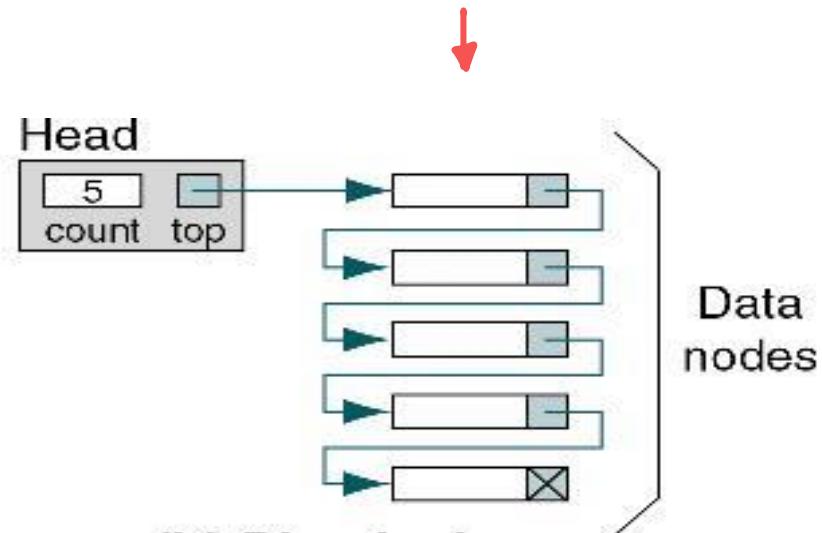


Stack Linked List Implementation

- สามารถสร้างโครงสร้างข้อมูล Stack โดยใช้ **Linked list** มาช่วย



(a) Conceptual



(b) Physical

Stack Data Structure

dNode

 type data

 dNode link

end dNode



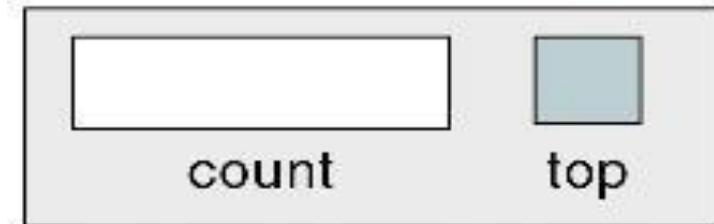
Stack node structure

hNode

 int count

 dNode top

end hNode



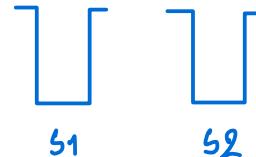
Stack head structure

Stack Algorithm : Create Stack

```
Algorithm createStack
Creates and initializes metadata structure.
Pre    Nothing
Post   Structure created and initialized
Return stack head
1  allocate memory for stack head
2  set count to 0
3  set top to null
4  return stack head
end createStack
```

createStack s_1

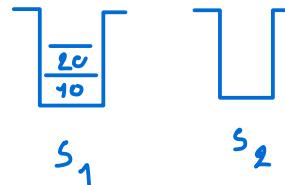
createStack s_2



Stack Algorithm : Push Stack

```
Algorithm pushStack (stack, data)
Insert (push) one item into the stack.
    Pre stack passed by reference
        data contain data to be pushed into stack
    Post data have been pushed in stack
1 allocate new node
2 store data in new node
3 make current top node the second node
4 make new node the top
5 increment stack count
end pushStack
```

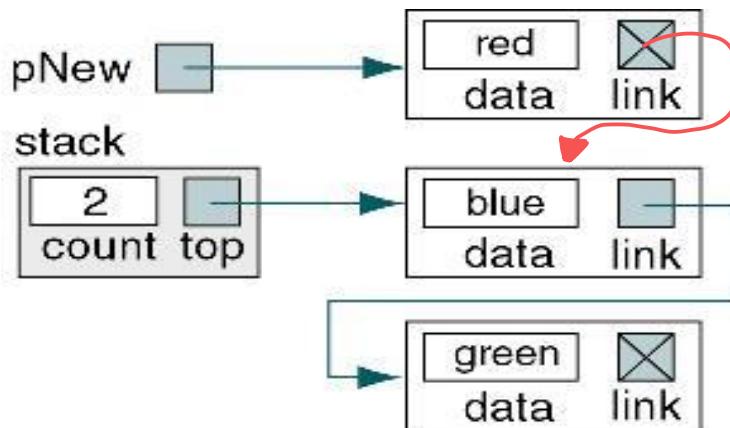
pushStack (s_1 , 10) ; pushStack (s_1 , 20)



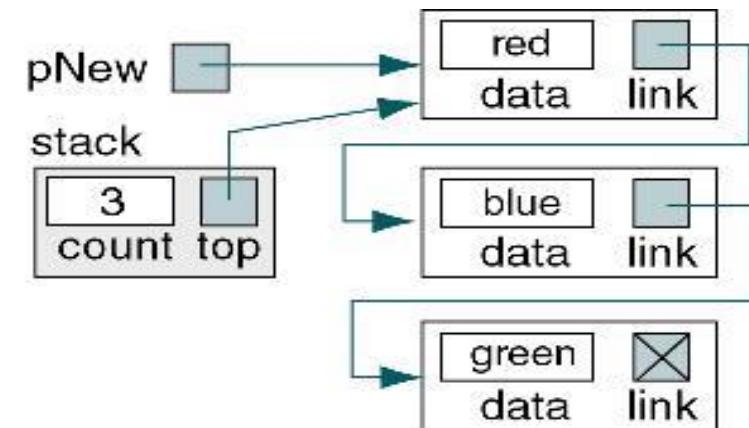
Example : Push Stack

$pNew.link = stack.top$

$stack.top = pNew$



(a) Before



(b) After

Stack Algorithm : Pop Stack

Algorithm by pop

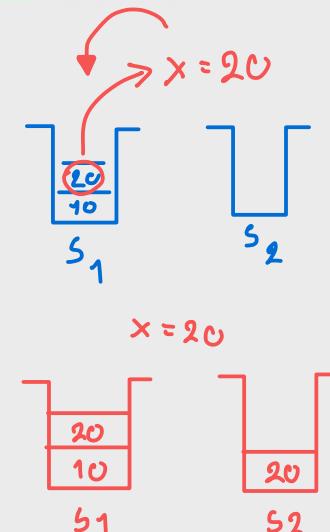
Algorithm popStack (stack, dataOut)

This algorithm pops the item on the top of the stack and returns it to the user.

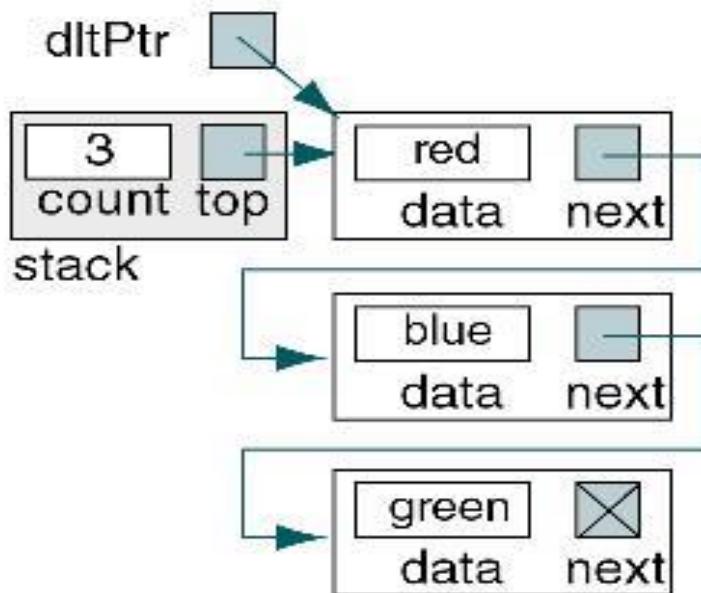
Pre stack passed by reference
dataOut is reference variable to receive data
Post Data have been returned to calling algorithm
Return true if successful; false if underflow

```
1 if (stack empty)
    1 set success to false
2 else
    1 set dataOut to data in top node
    2 make second node the top node
    3 decrement stack count
    4 set success to true
3 end if
4 return success
end popStack
```

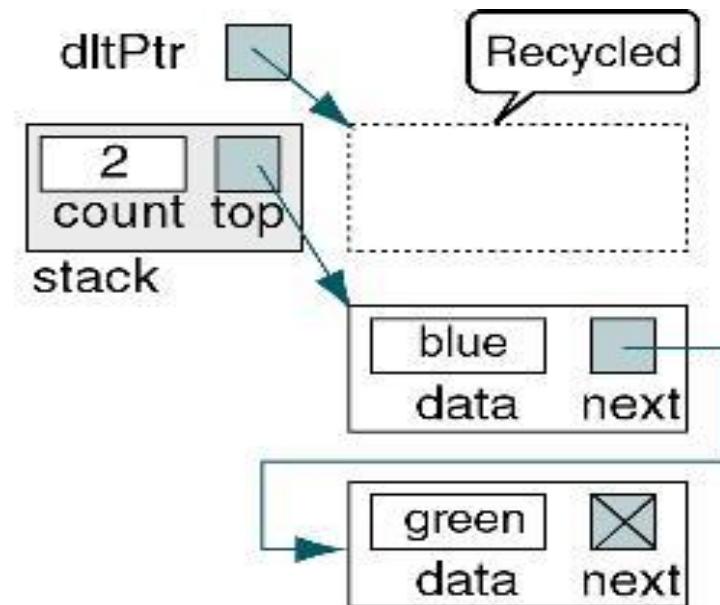
popStack (s_1, x)
pushStack (s_2, x)
pushStack (s_1, x)



Example : Pop Stack



(a) Before



(b) After

Stack Algorithm : Stack Top

```
Algorithm stackTop (stack, dataOut)
```

This algorithm retrieves the data from the top of the stack without changing the stack.

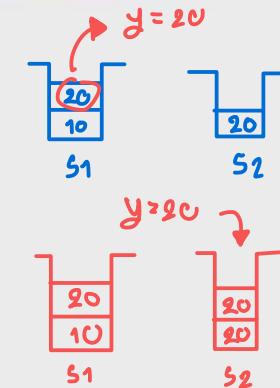
Pre stack is metadata structure to a valid stack
 dataOut is reference variable to receive data

Post Data have been returned to calling algorithm

Return true if data returned, false if underflow

```
1 if (stack empty)
  1 set success to false
2 else
  1 set dataOut to data in top node
  2 set success to true
3 end if
4 return success
end stackTop
```

stackTop (s_1, y)
pushStack (s_2, y)

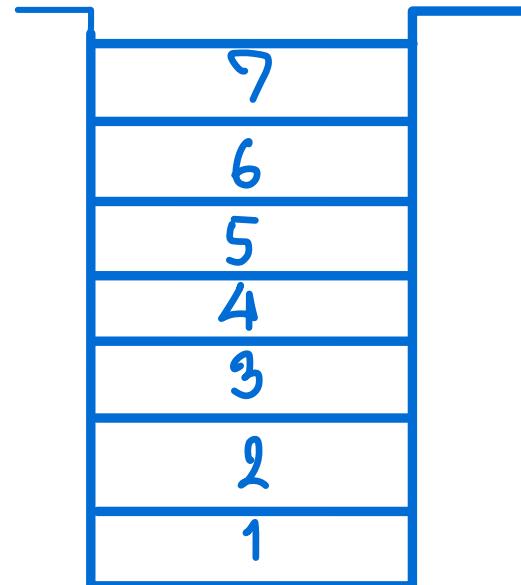


Stack Application

- Reversing data
- Parsing data
- Postponing data usage
- Backtracking step

Reversing Data : Reverse a List

- ใช้ Stack มาช่วยในการพิมพ์ค่าของข้อมูลในลิสต์แบบย้อนกลับ
- ทำได้ง่ายๆ แค่อ่านข้อมูลเก็บลง stack ไปจนหมดแล้วจึง pop ข้อมูลและพิมพ์ลงหน้าจอจนหมด stack
- Input = 1 2 3 4 5 6 7
- Output = 7 6 5 4 3 2 1



Parsing : Parse Parenthesis

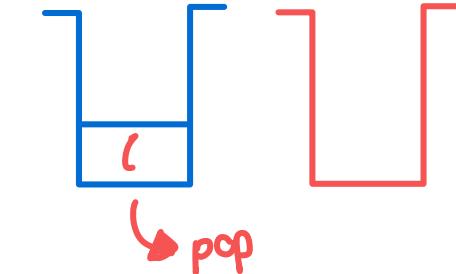
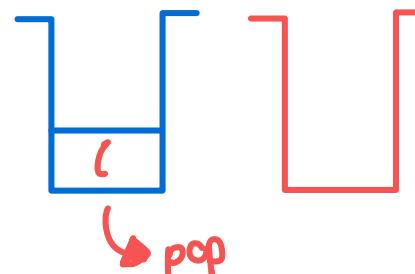
- ใช้ stack มาช่วยตรวจสอบโค้ดโปรแกรมว่า มีการกำหนดวงเล็บเปิดปิดถูกต้องหรือไม่ (ครบคู่หรือไม่)
push ไม่ถูกต้อง จึง pop ไม่ได้ กรณี underflow
- เช่น $((A+B)/C, (A+B)/C)$ มีการกำหนดวงเล็บไม่ถูกต้อง

~~push~~ ~~pop~~ ~~push~~

push pop
((A + B))

push pop
((C + D))

$((B + 10) \neq (10 + B))$



Parsing : Parse Parenthesis

```
Algorithm parseParen
```

This algorithm reads a source program and parses it to make sure all opening-closing parentheses are paired.

```
1 loop (more data)
  1 read (character)
  2 if (opening parenthesis)
    1 pushStack (stack, character)
  3 else
    1 if (closing parenthesis)
      1 if (emptyStack (stack))
        1 print (Error: Closing parenthesis not matched)
      2 else
        1 popStack(stack)
      3 end if
    2 end if
  4 end if
2 end loop
3 if (not emptyStack (stack))
  1 print (Error: Opening parenthesis not matched)
end parseParen
```

Postponement :

Infix-Postfix Converting

Infix : A +B

Prefix : +AB

Postfix : AB +

- Infix notation : โอเปอเรเตอร์จะอยู่ระหว่างโอเปอเรนด์ 2 ตัว เช่น A+B
- Postfix notation : โอเปอเรเตอร์จะอยู่หลังโอเปอเรนด์ทั้งสอง เช่น AB+
- กฎที่ใช้แปลง Infix เป็น Postfix Notation
 - จัดวงเล็บให้กับโจทย์
 - เปลี่ยนรูปแบบ infix ให้เป็น postfix โดยเริ่มทำจากวงเล็บในสุดก่อน
 - เอารงเล็บออก

Converting Infix to Postfix

$$5+3 \times 4 \rightarrow 19$$

- แปลงประโยคนี้ ให้อยู่ในรูปแบบ Postfix

$$A + B * C$$

- ใส่วงเล็บ

$$(A + (B * C))$$

- เปลี่ยนเป็นรูปแบบ Postfix โดยเริ่มจากการเล็บในสุดก่อน

$$(A + (B C *))$$

$$(A (B C *)) +$$

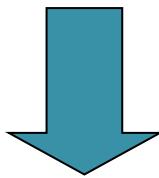
- อ่านวงเล็บออก

$$A B C * +$$

สังเกต ถ้าจัดกลุ่มประโยค infix ได้ ก็แปลงเป็น Postfix ได้ แล้วเราจัดกลุ่มอย่างไรละ

Converting Infix to Postfix

$(A + (B * C) - (D / E))$



$ABC^*+DE/-$

stack used for collect operator

if

→ op > Top → push ដោយក្រឡើ

else

→ op ≤ Top → pop op នៅលម្អាត
until push op ជាទុល្ខន

when finish if stack ≠ null

pop all

Infix	Stack	Postfix
(a) $A + B * C - D / E$ <i>push</i>		
(b) $+ B * C - D / E$		A
(c) $B * C - D / E$	$+$	$A B$
(d) $* C - D / E$ <i>push</i>	$*$	$A B *$
(e) $C - D / E$ <i>push</i> \ominus	$*$ $+$	$A B *$
(f) $- D / E$ <i>push</i> \ominus	\ominus	$A B C$
(g) D / E <i>push</i>		$A B C \ominus$
(h) $/ E$		$A B C \ominus +$
(i) E		$A B C \ominus + D$
(j)		$A B C \ominus + D E$
(k)		$A B C \ominus + D E \ominus$

Algorithm : Converting Infix to Postfix

```
Algorithm inToPostFix (formula)
Convert infix formula to postfix.
    Pre    formula is infix notation that has been edited
           to ensure that there are no syntactical errors
    Post   postfix formula has been formatted as a string
    Return postfix formula
1 createStack (stack)
2 loop (for each character in formula)
    1 if (character is open parenthesis)
        1 pushStack (stack, character)
    2 elseif (character is close parenthesis)
        1 popStack (stack, character)
        2 loop (character not open parenthesis)
            1 concatenate character to postFixExpr
            2 popStack (stack, character)
        3 end loop
    3 elseif (character is operator)
        Test priority of token to token at top of stack
        1 stackTop (stack, topToken)
        2 loop (not emptyStack (stack)
                AND priority(character) <= priority(topToken))
            1 popStack (stack, tokenOut)
            2 concatenate tokenOut to postFixExpr
            3 stackTop (stack, topToken)
```

Algorithm :

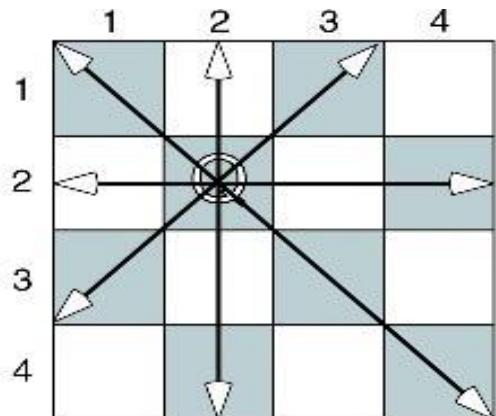
Converting Infix to Postfix (cont.)

```
    3  end loop
    4  pushStack (stack, token)
4  else
    Character is operand
    1 Concatenate token to postFixExpr
5  end if
3 end loop
Input formula empty. Pop stack to postFix
4 loop (not emptyStack (stack))
    1 popStack (stack, character)
    2 concatenate token to postFixExpr
5 end loop
6 return postFix
end intToPostFix
```

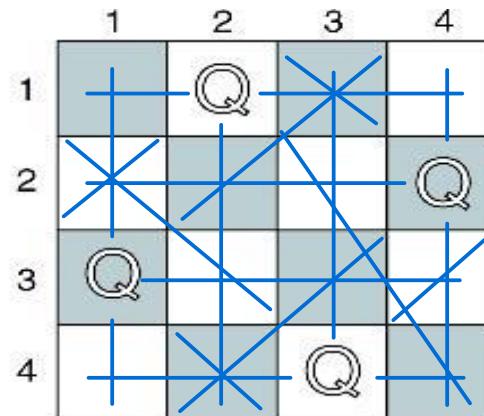
Backtracking : 4 Queens Problems

- Four Queens Problem

- พยายามวาง Queen 4 ตัวบนกระดาน โดยที่ Queen แต่ละตัวจะไม่อยู่ในทิศการเดินของ Queen ตัวอื่น

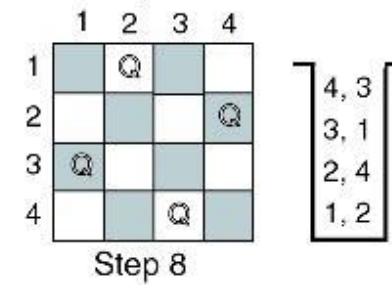
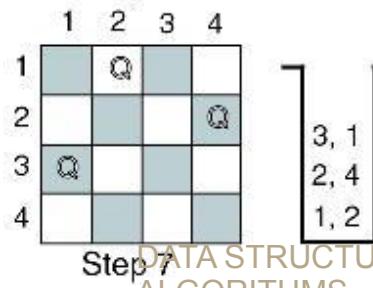
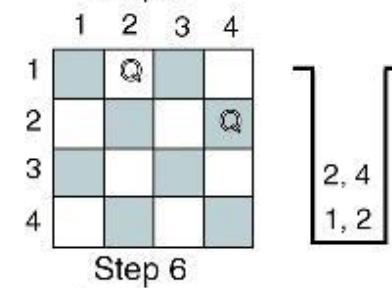
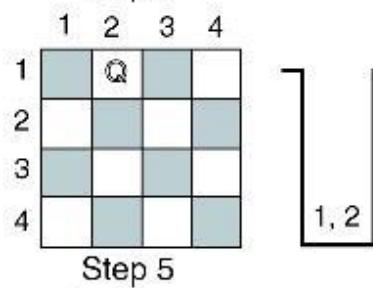
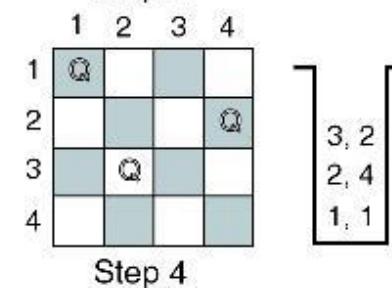
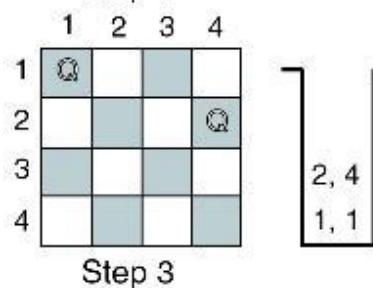
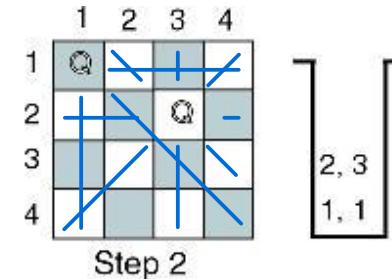
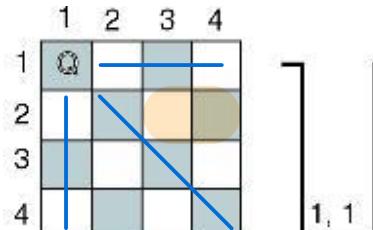


(a) Queen capture rules



(b) First four queens solution

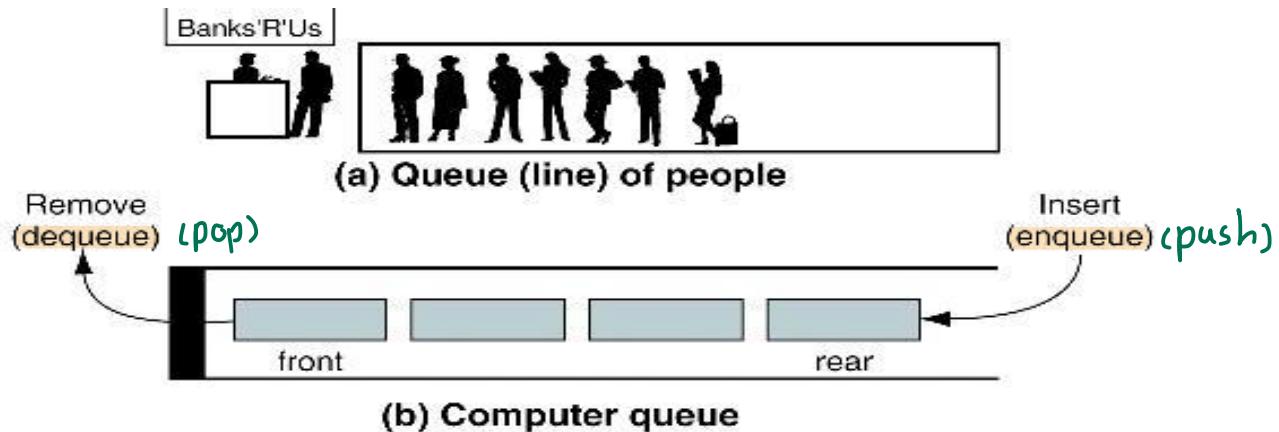
Four Queens Solution



Queue

Queues

- เป็นโครงสร้างข้อมูลแบบ linear list ประเภทหนึ่ง โดยที่
 - จะเพิ่มข้อมูลได้ที่ส่วนท้าย (rear) ของคิวเท่านั้น
 - จะลบข้อมูลได้ที่ส่วนหัว (front) ของคิวเท่านั้น
 - เป็นโครงสร้างแบบ FIFO (First In First Out) -> เข้ามาก่อน ออกก่อน

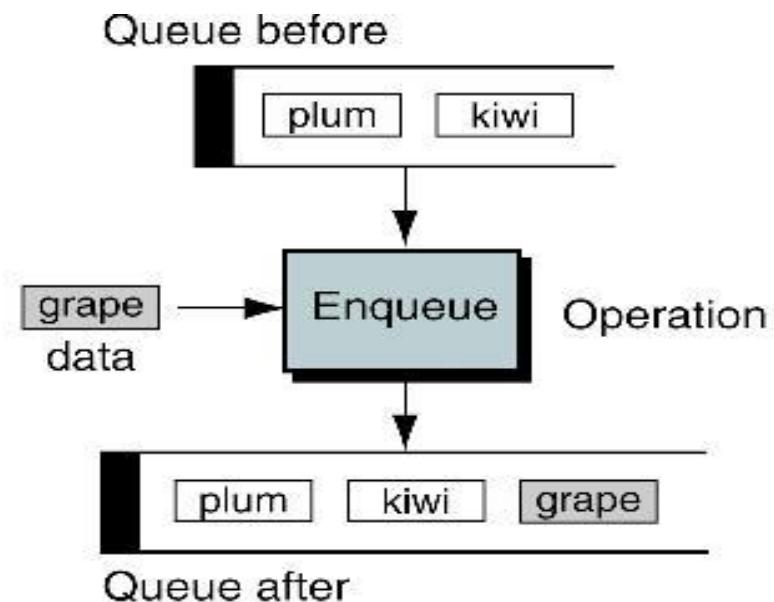


Queue Operations

- Enqueue : เพิ่มข้อมูลเข้าไปในคิวที่ส่วนท้าย (rear) ของคิว
- Dequeue : ลบข้อมูลออกจากคิวที่ส่วนหัว (front) ของคิว
- Queue Front : ส่งค่าข้อมูลที่ตำแหน่งหัวคิว
- Queue Rear : ส่งค่าข้อมูลที่ตำแหน่งท้ายคิว

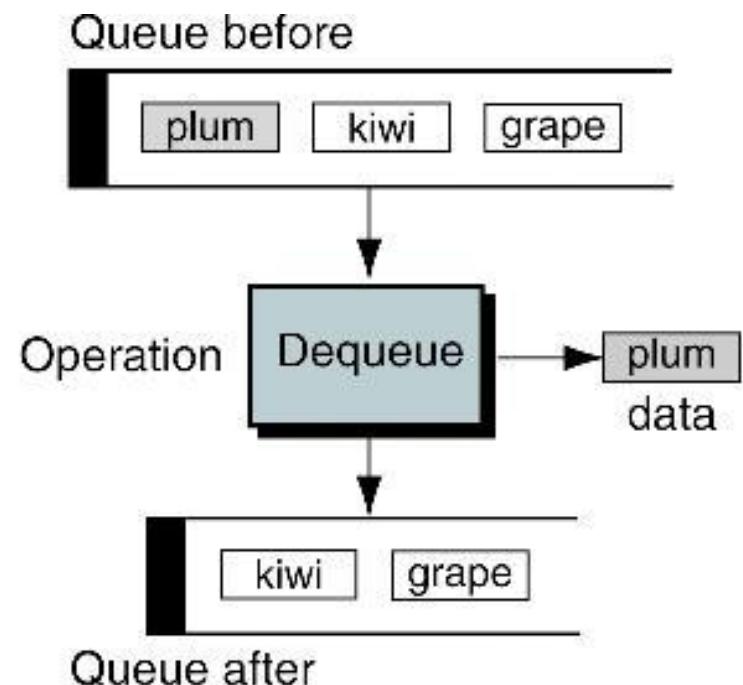
Enqueue

- เป็นการเพิ่มข้อมูลเข้าที่ส่วนท้าย (rear) ของคิว ทำให้ข้อมูลนั้นอยู่ตำแหน่งท้ายสุดของคิว
- ปัญหา：
 - ถ้าคิวเต็ม การเพิ่มข้อมูลลงไปจะทำให้เกิดสถานะ Overflow



Dequeue

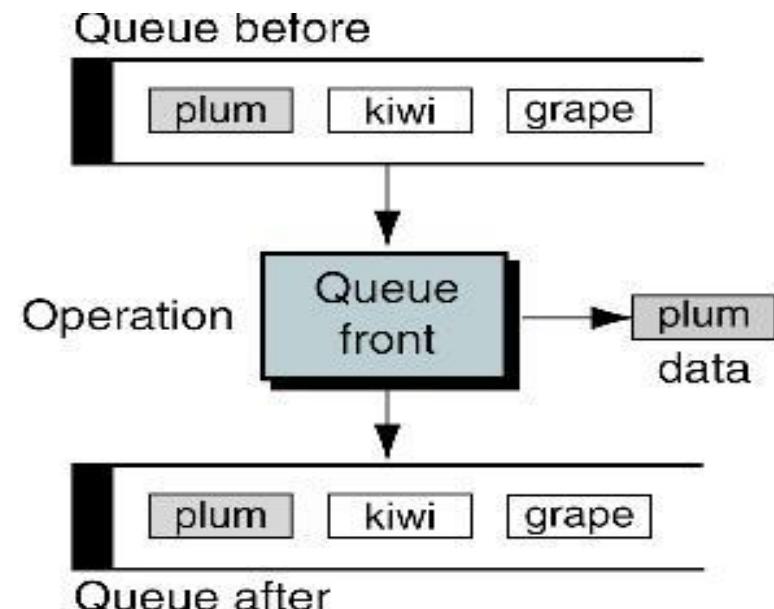
- เป็นการลบข้อมูลออกจากคิวที่ส่วนหัวของคิวแล้วคืนค่าข้อมูลนั้น ทำให้ข้อมูลถัดไปอยู่ตำแหน่งหัวคิวแทน
- ปัญหา：
 - ถ้าคิวว่าง การลบข้อมูลออก จะทำให้เกิดสถานะ Underflow



Queue Front

return

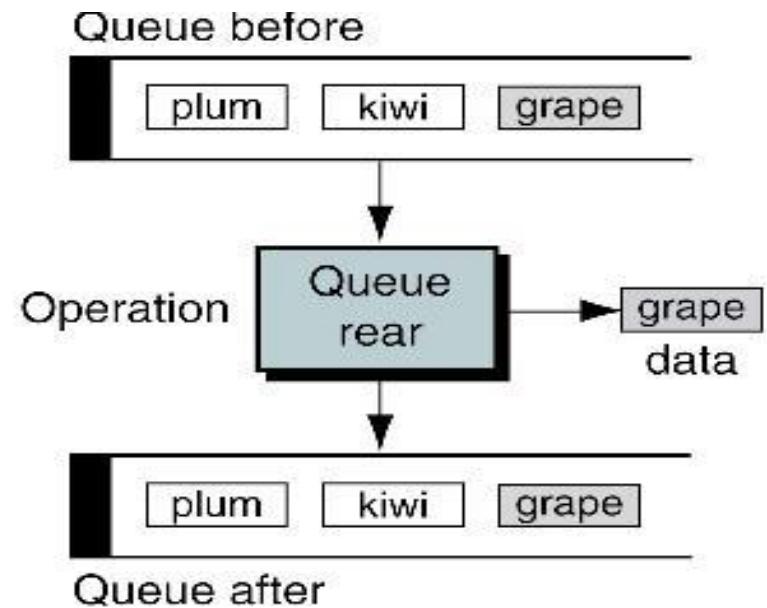
- เป็นการส่งค่าข้อมูลที่ตำแห่งหัวคิว โดยไม่ได้ลบข้อมูลนั้น
- ปัญหา：
 - ถ้าคิuw่าง การพยายามดึงข้อมูลที่ตำแห่งหัวคิวจะทำให้เกิดสถานะ Underflow



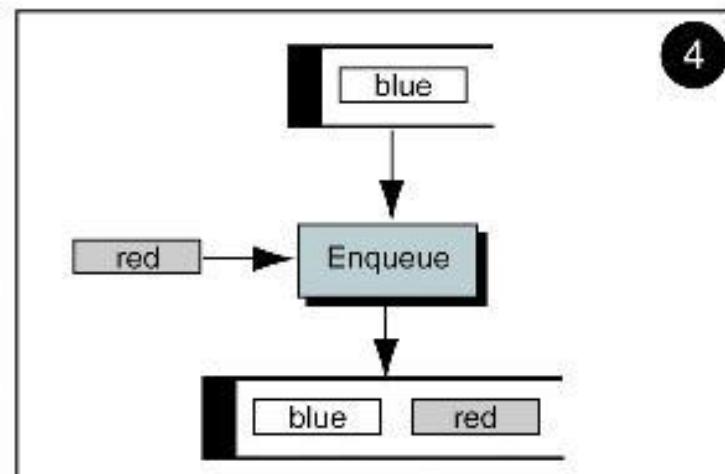
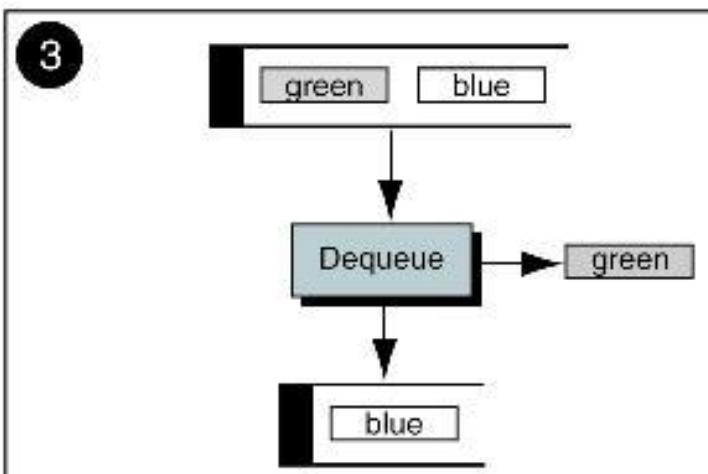
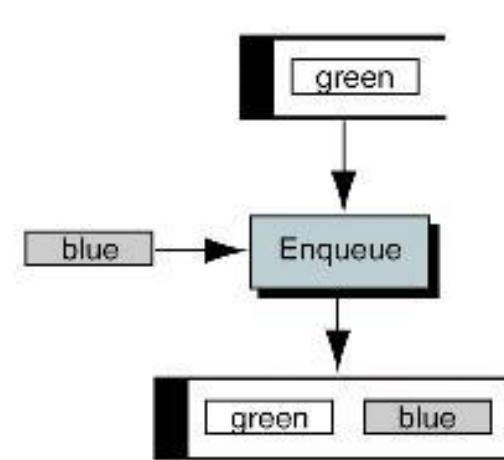
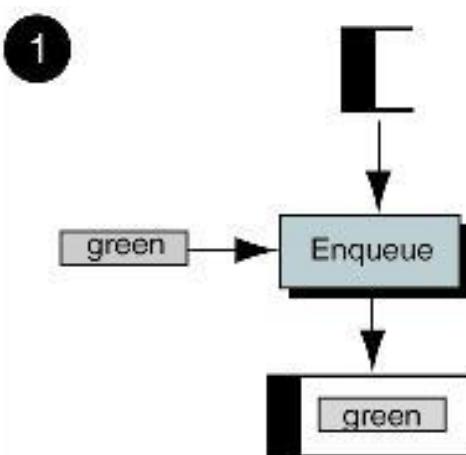
Queue Rear

return

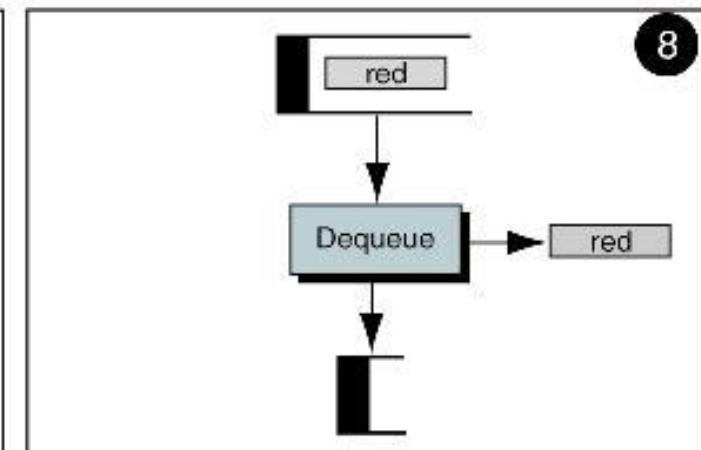
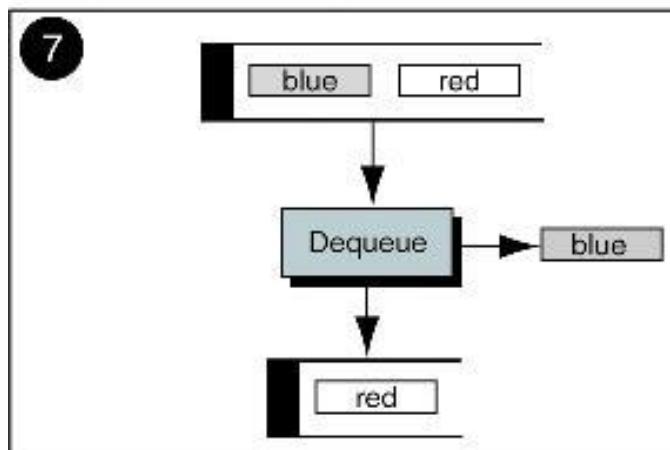
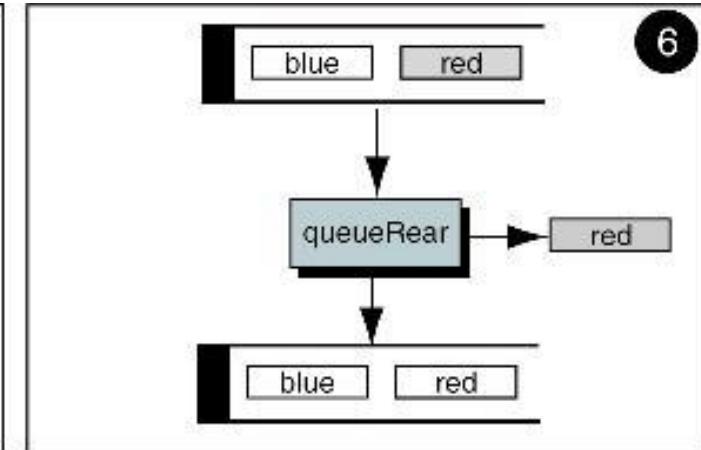
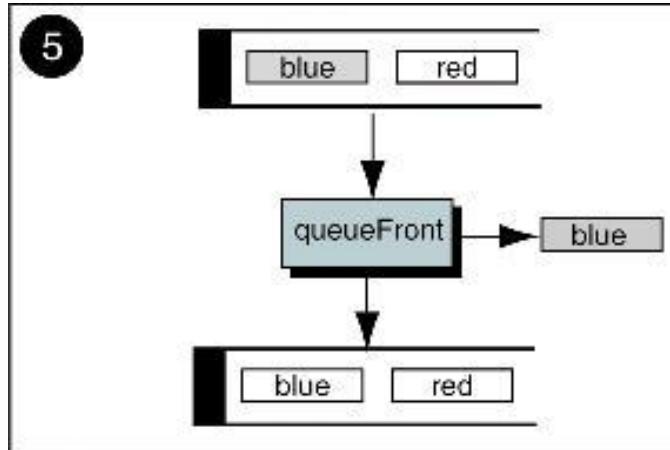
- เป็นการส่งค่าข้อมูลที่ **ตำแหน่งท้ายคิว** โดยไม่ได้ลบข้อมูลนั้น
- ปัญหา：
 - ถ้าคิวว่าง** การพยายามดึงข้อมูลที่ **ตำแหน่งท้ายคิว** จะทำให้เกิดสถานะ **Underflow**



Queue Example



Queue Example (cont.)

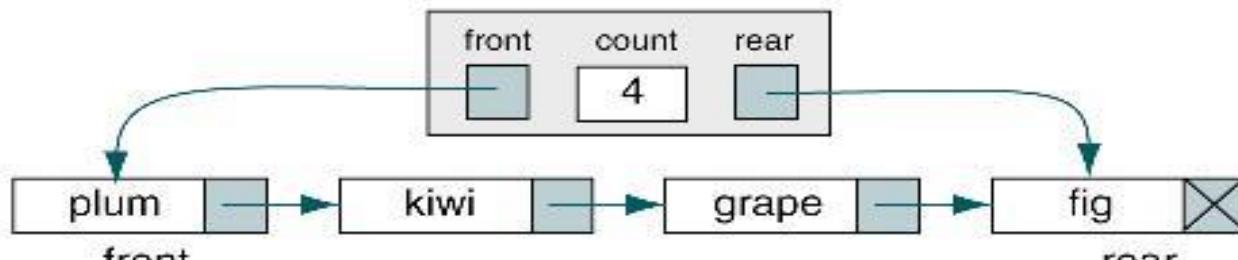


Queue Linked List Design

- สามารถสร้างโครงสร้างข้อมูล Queue โดยใช้ Linked list มาช่วย



(a) Conceptual queue



(b) Physical queue

Queue Data Structure

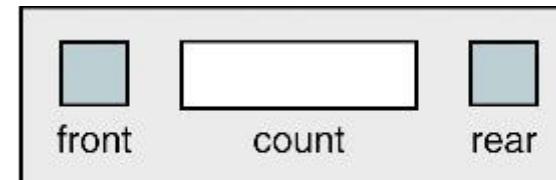
queueH

 dNode front

 dNode rear

 int count

end queueH



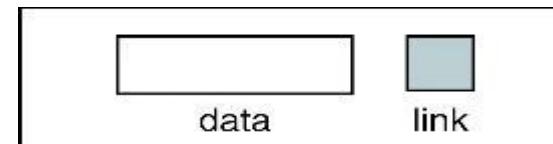
Head structure

dNode

 type data

 dNode link

end dNode



Node structure

Queue Algorithms : Create Queue

```
Algorithm createQueue
Creates and initializes queue structure.
Pre    queue is a metadata structure
Post   metadata elements have been initialized
Return queue head
1 allocate queue head
2 set queue front to null
3 set queue rear to null
4 set queue count to 0
5 return queue head
end createQueue
```

createQueue (Q_1)

Q_1



Queue Algorithms : Enqueue

```
Algorithm enqueue (queue, dataIn)
```

This algorithm inserts data into a queue.

Pre queue is a metadata structure

Post dataIn has been inserted

Return true if successful, false if overflow

```
1 if (queue full)
```

```
1 return false
```

```
2 end if
```

```
3 allocate (new node)
```

```
4 move dataIn to new node data
```

```
5 set new node next to null pointer
```

```
6 if (empty queue)
```

Inserting into null queue

```
1 set queue front to address of new data
```

```
7 else
```

Point old rear to new node

```
1 set next pointer of rear node to address of new node
```

```
8 end if
```

```
9 set queue rear to address of new node
```

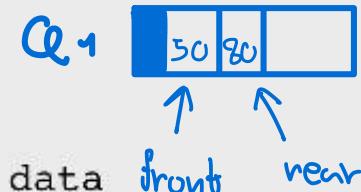
```
10 increment queue count
```

```
11 return true
```

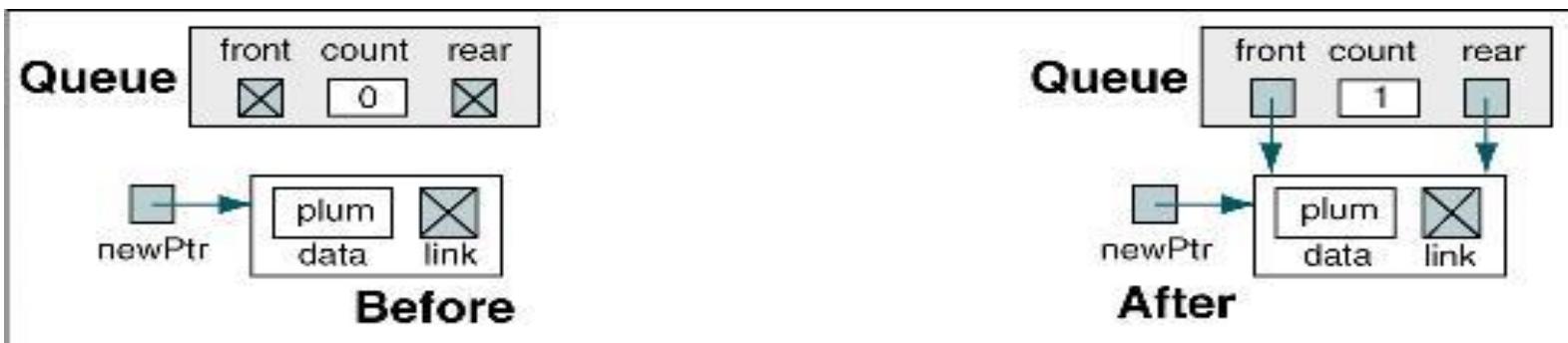
```
end enqueue
```

enqueue (Q1, 50)

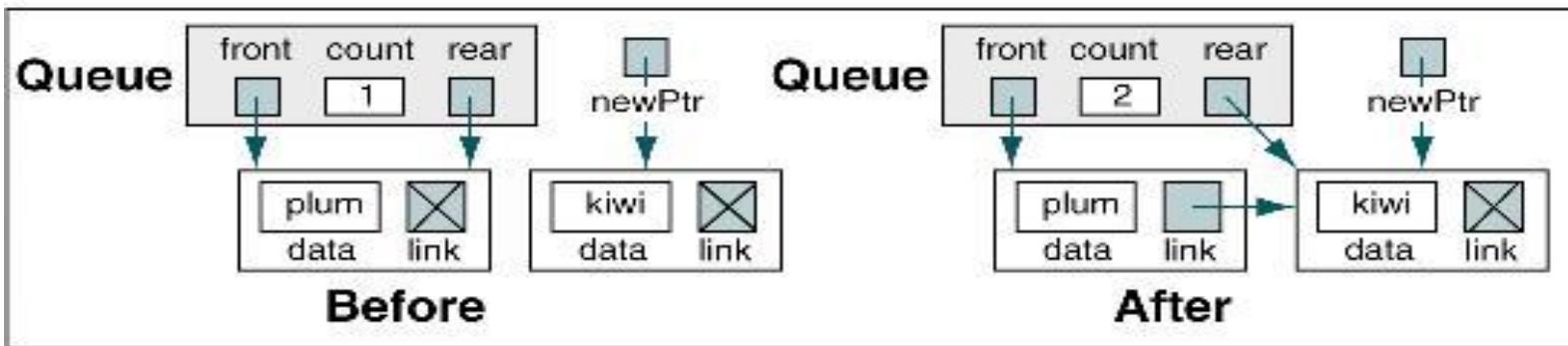
enqueue (Q1, 80)



Example : Enqueue



(a) Case 1: insert into empty queue



(b) Case 2: insert into queue with data

Queue Algorithms : Dequeue

Algorithm dequeue (queue, item)

This algorithm deletes a node from a queue.

Pre queue is a metadata structure

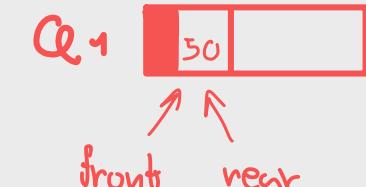
 item is a reference to calling algorithm variable

Post data at queue front returned to user through item
 and front element deleted

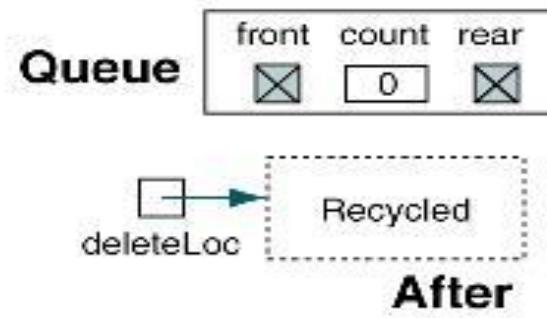
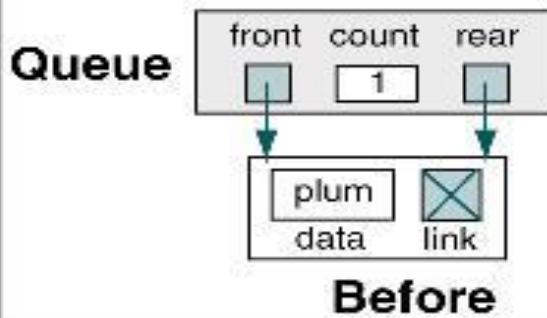
Return true if successful, false if underflow

```
1 if (queue empty)
  1 return false
2 end if
3 move front data to item
4 if (only 1 node in queue)
  Deleting only item in queue
  1 set queue rear to null
5 end if
6 set queue front to queue front next
7 decrement queue count
8 return true
end dequeue
```

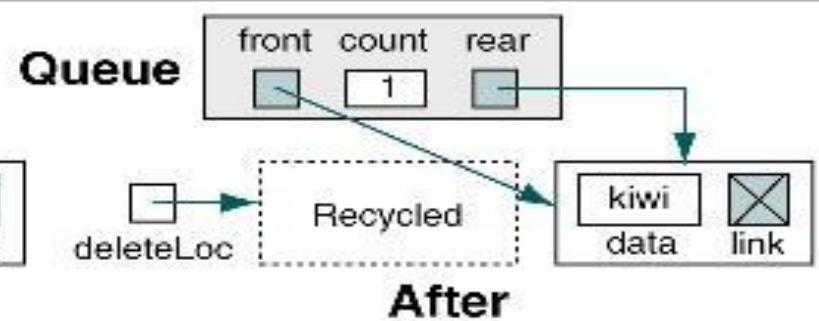
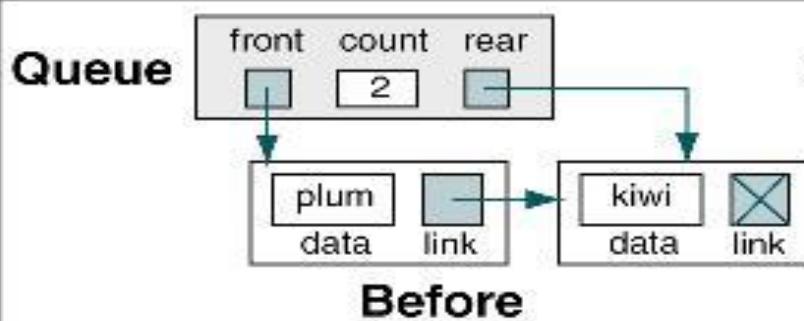
dequeue (Q1, 80)



Example : Dequeue



(a) Case 1: delete only item in queue



(b) Case 2: delete item at front of queue

Queue Algorithms : Queue Front

```
Algorithm queueFront (queue, dataOut)
Retrieves data at the front of the queue without changing
queue contents.

    Pre    queue is a metadata structure
           dataOut is a reference to calling algorithm variable
    Post   data passed back to caller
    Return true if successful, false if underflow

1 if (queue empty)
    1 return false
2 end if
3 move data at front of queue to dataOut
4 return true
end queueFront
```

Queue Application :

Categorizing Data

- ให้จัดกลุ่มของข้อมูลตามลำดับในลิสต์ กำหนดให้มีลิสต์ของตัวเลขต่อไปนี้
 - 3 22 12 6 10 34 65 29 9 30 81 4 5 19 20 57 44 99
- กำหนดให้จัดข้อมูลออกเป็น 4 กลุ่ม
 - กลุ่ม 1 ชุดข้อมูลที่มีค่าต่ำกว่า 10
 - กลุ่ม 2 ชุดข้อมูลที่มีค่าระหว่าง 10 ถึง 19
 - กลุ่ม 3 ชุดข้อมูลที่มีค่าระหว่าง 20 ถึง 29
 - กลุ่ม 4 ชุดข้อมูลที่มีค่าตั้งแต่ 30 ขึ้นไป
- ผลลัพธ์ คือ | 3 6 9 4 5 | 12 10 19 | 22 29 20 | 34 65 30 81 57 44 99 |

Algorithm : Category Queue

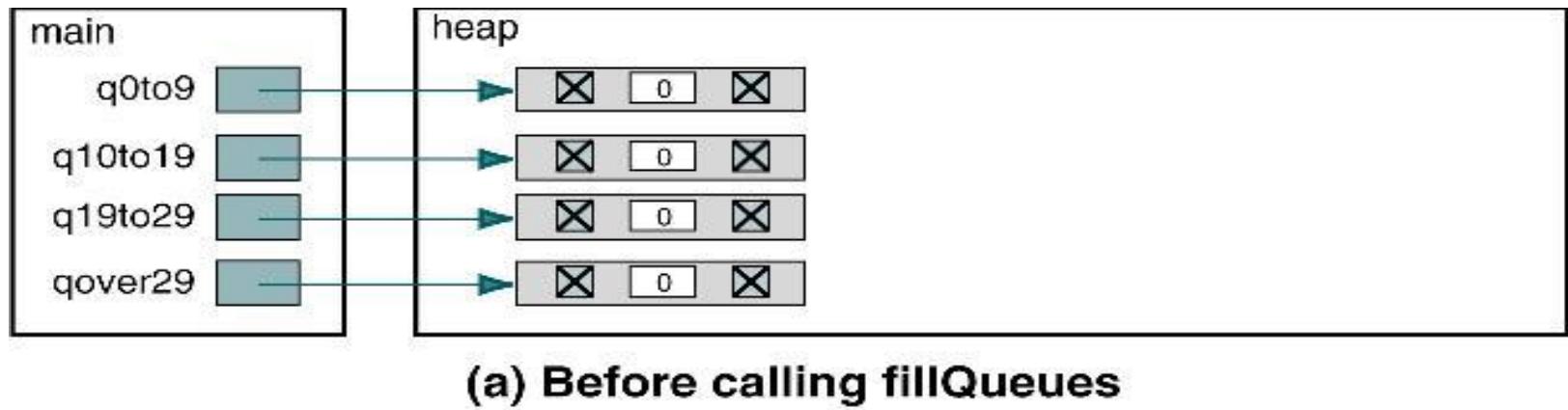
```
Algorithm categorize
    Group a list of numbers into four groups using four queues.
    Written by:
    Date:
1 createQueue (q0to9)
2 createQueue (q10to19)
3 createQueue (q20to29)
4 createQueue (qOver29)
5 fillQueues (q0to9, q10to19, q20to29, qOver29)
6 printQueues (q0to9, q10to19, q20to29, qOver29)
end categorize
```

Algorithm : Category Queue (cont.)

```
Algorithm fillQueues (q0to9, q10to19, q20to29, qOver29)
This algorithm reads data from the keyboard and places them in
one of four queues.

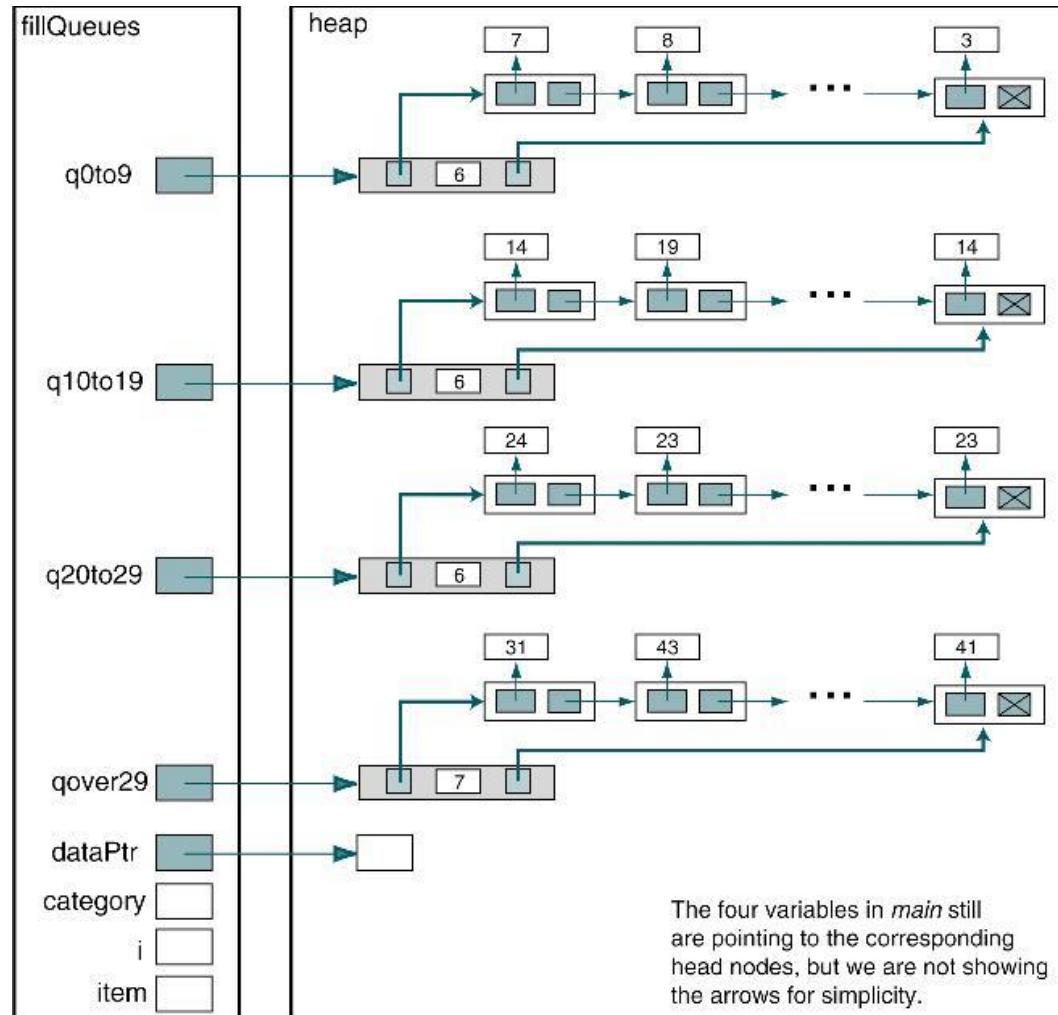
    Pre  all four queues have been created
    Post queues filled with data
1 loop (not end of data)
    1 read (number)
    2 if (number < 10)
        1 enqueue (q0to9, number)
    3 elseif (number < 20)
        1 enqueue (q10to19, number)
    4 elseif (number < 30)
        1 enqueue (q20to29, number)
    5 else
        1 enqueue (qOver29, number)
    6 end if
2 end loop
end fillQueues
```

Structures for Categorizing Data



Structures for Categorizing Data

(cont.)



(b) After calling `fillQueues`

Quiz 1

- จะประมวลผลคำสั่งต่อไปนี้ พร้อมวิเคราะห์ Stack ผลลัพธ์เมื่อทำทุกคำสั่งเสร็จแล้ว
(กำหนด $s1$ และ $s2$ เป็น Empty Stack)
 - pushStack($s1, 6$)
 - pushStack($s1, 5$)
 - pushStack($s1, 4$)
 - pushStack($s1, 3$)
 - pushStack($s1, 2$)
 - pushStack($s1, 1$)
 - Loop not emptyStack($s1$)
 - popStack($s1, x$)
 - popStack($s1, x$)
 - pushStack($s2, x$)
 - end Loop

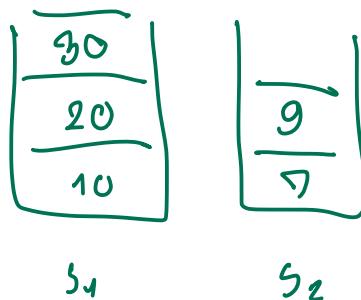
Quiz 2

- ให้ดาวดรูป queue Q ผลลัพธ์ เมื่อได้รับข้อมูล และมีการทำงานต่อไปนี้
 - กำหนด data : 9, 72, 1, 43, 29, 0, 34, 62, 3, 56, 0, 34
- ```
createQueue(Q)

loop (not end of file)
 read number
 if (number is greater than 5)
 enqueue(Q,number)
 else
 queueRear(Q,x)
 enqueue(Q,x)
 end if
end loop
```

# Quiz 3-4

- จงวาดภาพแสดงการแปลง Infix Expression เป็น Postfix Expression โดยใช้ Stack
  - $A * (B + C) - D * F - E$
- จงเขียนอัลกอริทึม copyStack(stack1,stack2) ที่ใช้คัดลอกข้อมูลของ stack1 ให้กับ stack2 (ให้มีลำดับข้อมูลเหมือนกัน)

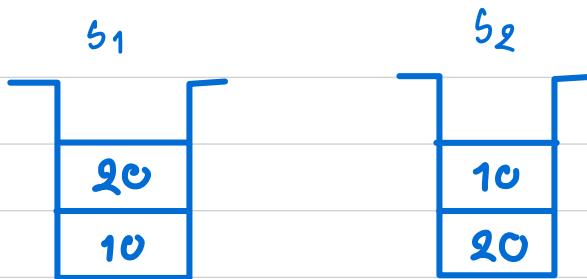


Algo copyStack( $S_1, S_2$ )  
loop  $S_2$  is not empty :  
    popStack( $S_2, x$ )  
end loop  
end copyStack

# Quiz 5

- ให้เขียนอัลกอริทึม concatQueue (ใช้คำสั่งของ queue ADT ได้)
  - รูปแบบ : concatQueue(Q1,Q2)
  - นำข้อมูลของ Q1 ต่อท้ายด้วย ข้อมูลของ Q2 และนำผลลัพธ์ที่ได้เก็บไว้ใน Q1
  - Q2 ยังคงเหมือนเดิม

/ createStack ( $S_1$ )



/ createStack ( $S_2$ )

/ createQueue ( $Q_1$ )



/ enqueue ( $Q_1, 10$ )

$$x = 10$$

$$y = 20$$

$$z = 10$$

$$A = 20$$

/ enqueue ( $Q_1, 20$ )

/ queueFront ( $Q_1, x$ )

/ pushStack ( $S_1, x$ )

/ queueRear ( $Q_1, y$ )

/ pushStack ( $S_1, y$ )

/ pushStack ( $S_2, z$ )

/ pushStack ( $S_2, y$ )

/ stackTop ( $S_1, A$ )

/ dequeue ( $Q_1, z$ )

/ enqueue ( $Q_1, A + Z$ )