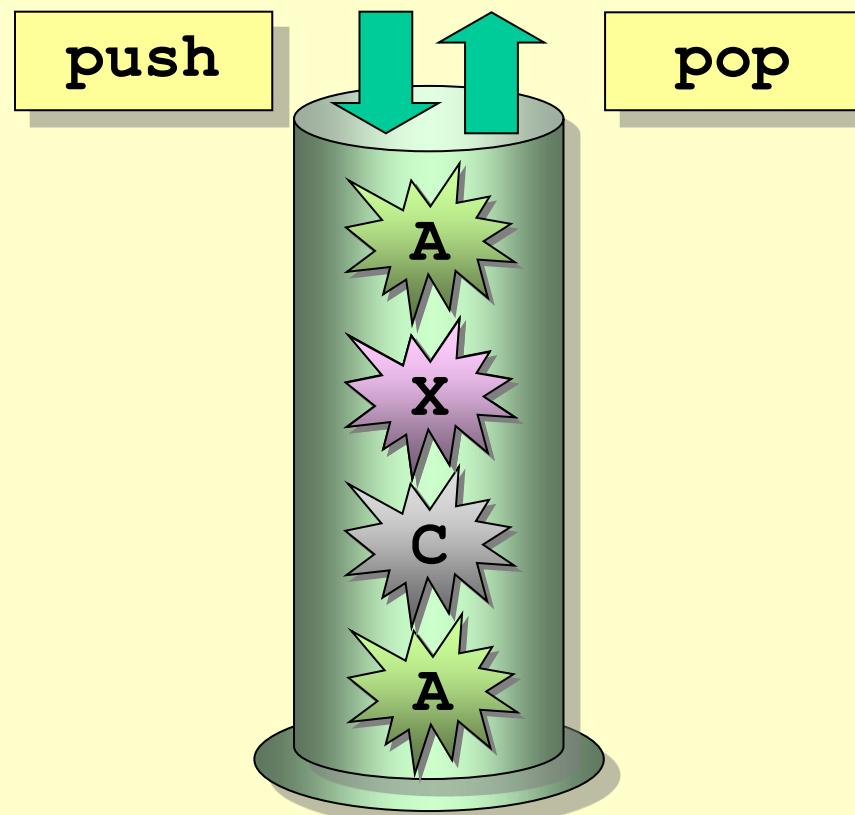


กองซ้อน

(Stack)

การเพิ่ม/ลบข้อมูลในกองช้อน

- ข้อมูล เข้าหลัง ออกก่อน (Last-In First-Out)



ກອງຫົວນ : stack

bool	empty () ;
unsigned	size () ;
T	<u>top () ;</u> ←
void	push (T element) ; ←
void	pop () ; ← ເລີ່ມ ດັວນ

ເອົາໃຫຍ່

ກົດໜີ້

ข้อควรระวัง

- `top()` ตอนไม่มีข้อมูล → พัง
- `pop()` ตอนไม่มีข้อมูล → พัง
- `stack` ไม่มี iterator
 - ดูได้เฉพาะข้อมูลที่พึ่งใส่เข้าไปล่าสุด (`top`)
 - ไม่มี `begin()`, `end()` ให้ใช้
 - ถ้าอยากดูข้อมูลทุกตัวใน `stack` ต้องจำใจ `pop` ข้อมูลออกมา

ตัวอย่างการใช้งาน Stack

- ◆ การตรวจสอบโครงสร้างแบบข้อมูลกัน เช่น การใส่สิ่งเล็บ () { } [] ...
 - ◆ การจัดเก็บตัวแปรและการทำ function calls
 - ◆ การประมวลผลนิพจน์ทางคณิตศาสตร์
 - ◆ การทำ undo/redo
 - ◆ การค้นคำตอบแบบ depth-first search
 - ◆ ...
- 
↑
↑
↑
↓ Discrete Math

การตรวจสอบการใส่วงเล็บ

- ถูก :
- ผิด : เปิดปิดไม่ตรงกัน ([])
- ผิด : มีปิดมากไป ({ () }) }
- ผิด : มีเปิดมากไป ({ () }
- วิธีทำ
 - อ่านมาทีละตัว
 - ถ้าเป็นวงเล็บเปิด ให้ push ลง stack
 - ถ้าเป็นวงเล็บปิด ให้ pop จาก stack มาตรวจสอบว่าเป็น วงเล็บเปิดที่ตรงกันวงเล็บปิดที่พบหรือไม่
 - เมื่อได้อยาก pop ถ้า isEmpty แสดงว่า ปิดมีมากไป
 - เมื่ออ่านเสร็จหมด stack ยังมีข้อมูล แสดงว่า เปิดมีมากไป

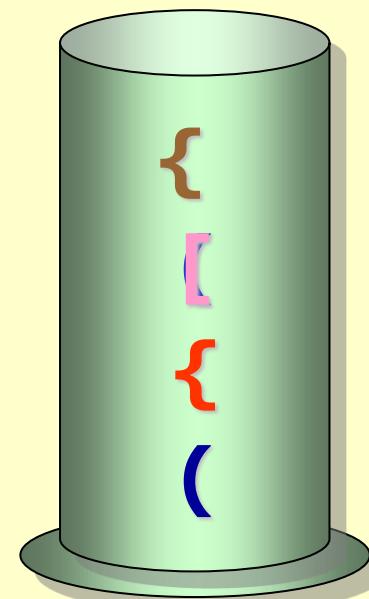
1 2 3
()() })
(),

ตัวอย่าง ๑

- ({ () [{ }] })



ถูกต้อง

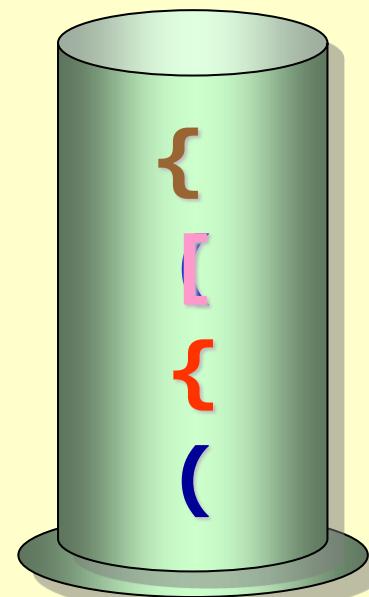


ตัวอย่าง ๒

- ({ () [{)] })



วงเล็บปิด
ไม่ตรงกับเปิด
ผิด !!

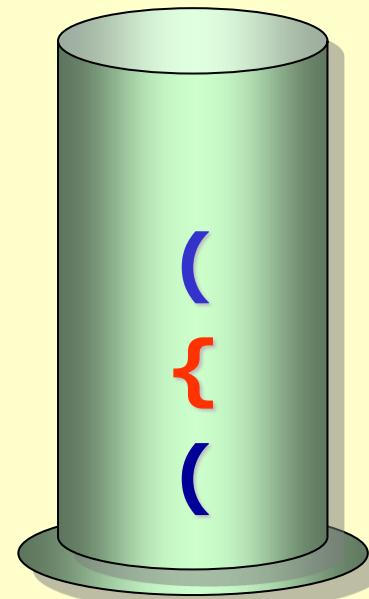


ตัวอย่าง ๓

- ({ () })]



ยังไม่หมด
แต่ stack ว่าง
ผิด !!

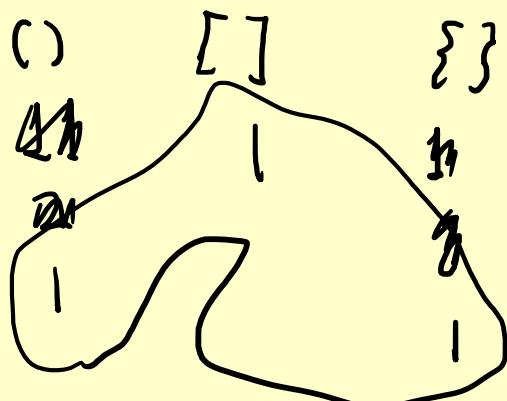


ตัวอย่าง ๔

- ({ () [{ }

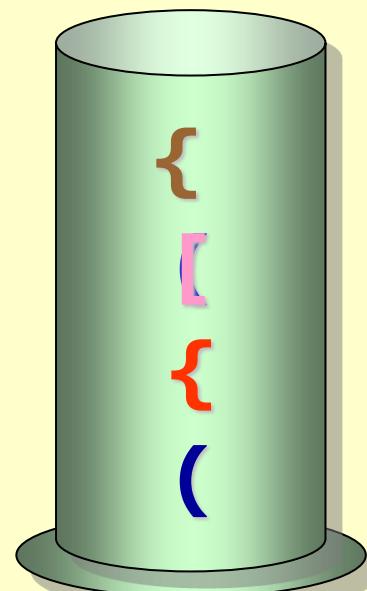


หมดแล้ว
แต่ stack ไม่ว่าง
ผิด !!



↓ ↓ ↓
([)]

([)] { }
↓ ↓ ↓ ↓
0



การใช้กองข้อมูลใน java virtual machine

- ตัว jvm ใช้กองข้อมูล (java stack) ในการเก็บ
 - สถานะของการเรียกเมท็อด
 - พารามิเตอร์ และ local variables
 - ที่เก็บชั่วคราวเพื่อการคำนวณ (operand stack)

```
public class Jeng3 {  
    public static void main(String[] a) {  
        main(args);  
    }  
}
```

Exception in thread "main" java.lang.StackOverflowError
at Jeng3.main(Jeng3.java:3)

```
int main() {  
    main();  
}
```

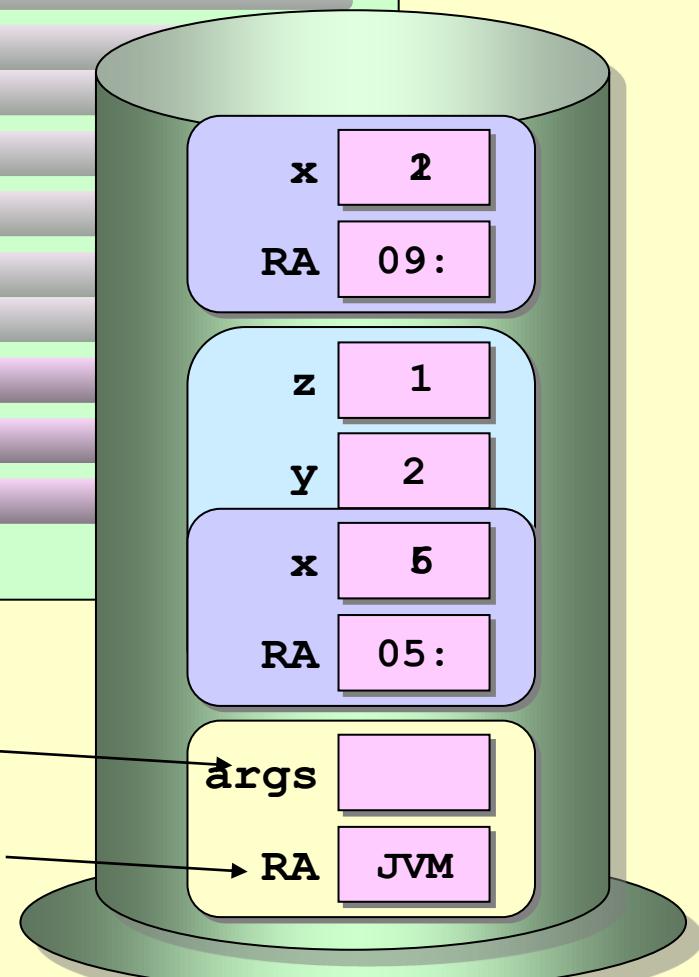
Process terminated with status -1073741571

Java Stack

```
01:public class StackFrame {  
02:    public static void main(String[] args) {  
03:        a(3, 2);  
04:        b(5);  
05:    }  
06:    static void a(int x, int y) {  
07:        int z = x/y;  
08:        b(z);  
09:    }  
10:    static void b(int x) {  
11:        ++x;  
12:    }  
13:}
```

กรอบกองซ้อน
(Stack Frame)

parameters &
local variables
return address



นิพจน์ Infix และ Postfix

$$a+b$$

- ## infix (ເຕີມກລາງ)

- a + b * c / d - 2,

– ต้องกำหนดลำดับการคำนวณหลังของ operators

– ໃຫ້ງເລີບຊ່າຍ

postfix (เติมหลัง)

$$- \cancel{a b c * d / + 2 -}, \quad a b + \cancel{c * d 2 -} /$$

- ลำดับการทำงานของ operators คือจาก ซ้ายไปขวา

- ไม่จำเป็นต้องมีวงเล็บ

S V O

$$(a + b) * c / (d - 2)$$

HP

S O V

a b

S Q V

05V

V 50

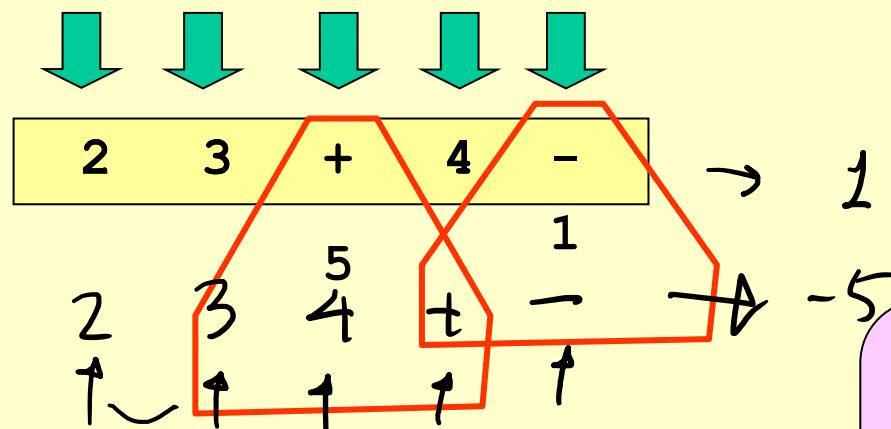
✓ o S

1	2	+	3	*	4	1	-	/
3	3	*	4	1	-	/		
9	4	1	-	/				
9	3	/						

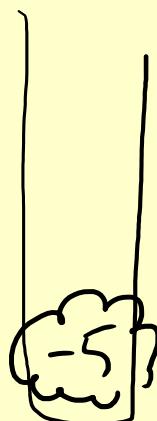
การใช้กองซ้อนคำนวณค่าของนิพจน์เติมหลัง

- $2\ 3\ +\ 4\ 5\ -\ 6\ *\ +$ มีค่าเท่าไร ?
- สามารถใช้ stack ช่วยหาค่าของนิพจน์ postfix
- วิธีทำ
 - ดูทีละตัวใน postfix จากซ้ายไปขวา
 - ถ้าเป็น operand ให้ push ของ stack
 - ถ้าเป็น operator ให้ pop operands จาก stack ตามที่ operator ต้องการมาประมวลผล และ push ผลลัพธ์
 - ทำเสร็จ คำตอบจะอยู่ที่ top of stack

ตัวอย่าง

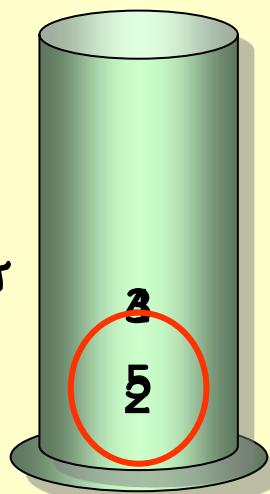


ผลลัพธ์อยู่
บนกองขอน



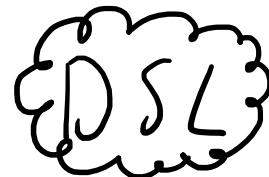
$$3 + 4 = 7$$

$$2 - 7 = -5$$



jvm เป็น stack machine

```
int a = 1;
```



```
int b = 2;
```

```
int c = 3;
```

```
int d = a + b * (c + a);
```

infix

postfix

```
iconst_1  
istore_1
```

```
iconst_2  
istore_2
```

```
iconst_3  
istore_3
```

```
iload_1  
iload_2
```

```
iload_3  
iload_1
```

```
iadd
```

```
imul
```

```
iadd
```

```
istore_4
```

การใช้กองช้อนช่วยแปลง infix เป็น postfix

- input : infix expression
- output : postfix expression
- ขั้นตอนการทำงาน
 - ดูแต่ละตัวใน infix
 - ถ้าเป็น operand นำไปต่อท้าย output
 - ถ้าเป็น operator
 - อาจ pop operator ออกไปต่อท้าย output
 - push operator ลงกองช้อน
 - เมื่อดู infix ครบตัวแล้ว
 - ให้ pop operators ทุกตัวออกไปต่อท้าย output

infix -> postfix

```
string infix2postfix(string &infix) {  
    int n = infix.length();  
    string postfix = "";  
    stack<char> s;  
    for (int i=0; i<n; i++) {  
        char token = infix[i];  
        if (token เป็น operand) {  
            postfix += token;  
        } else {  
            while(????) {  
                postfix += s.top();  
                s.pop();  
            }  
            s.push(token);  
        }  
    }  
    while(!s.empty()) {postfix += s.top(); s.pop();}  
    return postfix;  
}
```

123+74

1 + 4

ดูทีละตัว

ถ้าเป็น operand, เพิ่มต่อในผลลัพธ์

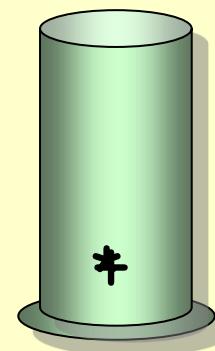
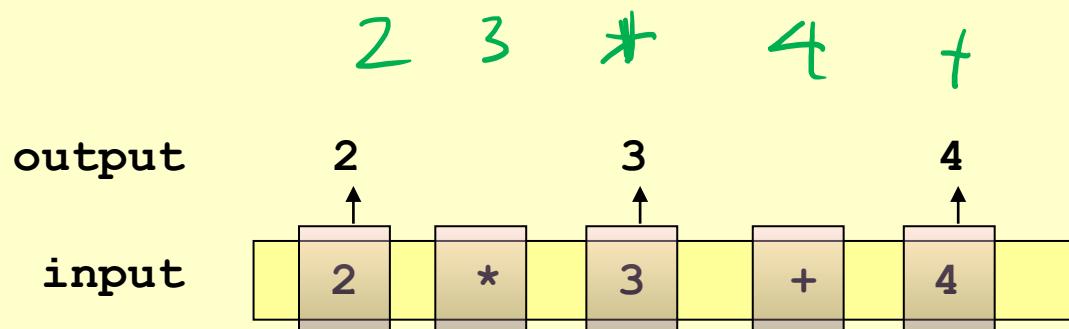
ถ้าเป็น operator

อาจ pop operators ใน stack
ออกเป็นผลลัพธ์

ตามด้วยการ push operator ตัวใหม่

© S. Prasitjutrakul 2013

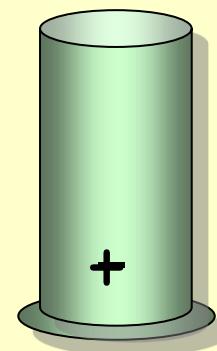
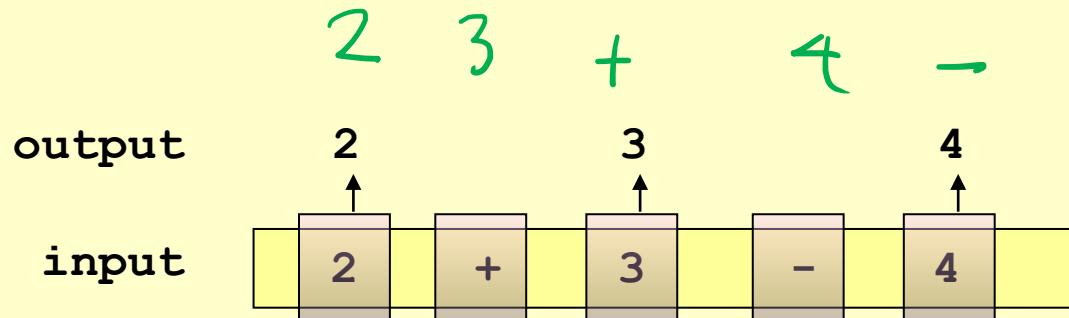
ความสำคัญของ operators



* มาก่อน และสำคัญกว่า +

pop ออก ถ้า operator บนกองช้อนสำคัญกว่า operator ใหม่

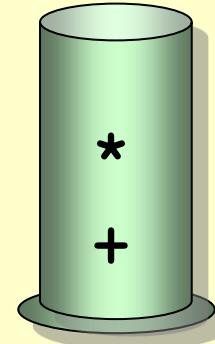
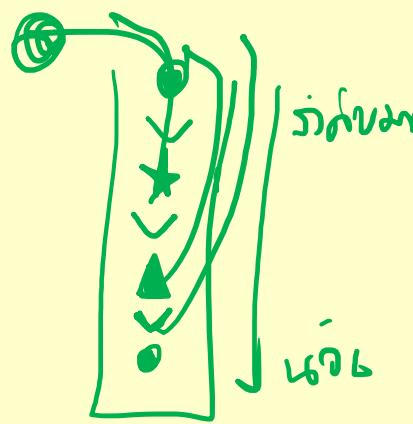
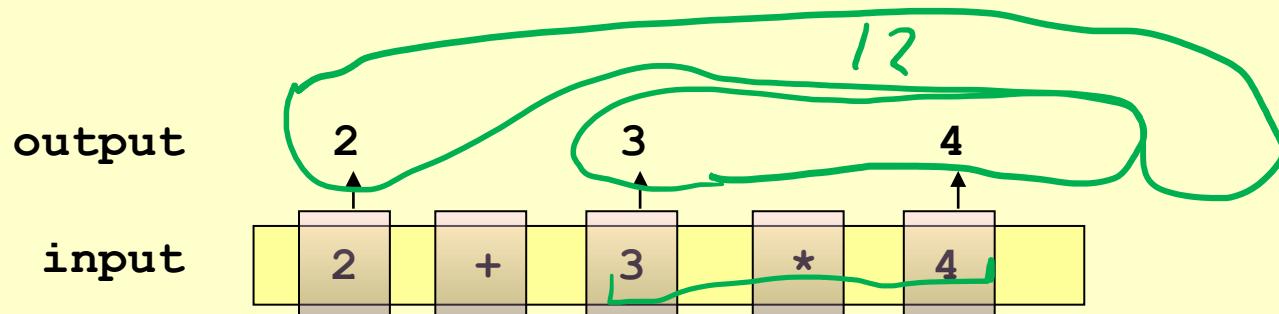
ความสำคัญของ operators



+ มา ก่อน
แล้ว สำคัญ เท่า กับ -

pop ออก ถ้า operator บน กอง ช้อน
สำคัญ ไม่น้อย กว่า operator ใหม่

ความสำคัญของ operators



+ มาก่อน แต่ * สำคัญกว่า +

push ทับ ถ้า operator ใหม่สำคัญกว่า operator บนกองข้อมูล

การเปรียบเทียบความสำคัญของ operators

```
string infix2postfix(string &infix) {  
    int n = infix.length();  
    string postfix = "";  
    stack<char> s;  
    for (int i=0; i<n; i++) {  
        char token = infix[i];  
        if (priority[token] == 0) {  
            postfix += token;  
        } else {  
            int p = priority[token];  
            while( !s.empty() && priority[s.top()] >= p ) {  
                postfix += s.top(); s.pop();  
            }  
            s.push(token);  
        }  
    }  
    while(!s.empty()) { postfix += s.top(); s.pop(); }  
    return postfix;  
}
```

lazy evaluation

while (E && S)

while (priority[s.top()] >= p)

หากค่าความสำคัญของ
operator ตัวใหม่ที่พบ

pop ออก ถ้า operator บนกองข้อมูล
สำคัญไม่น้อยกว่า operator ใหม่

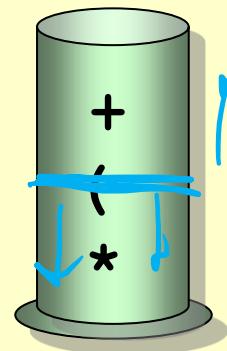
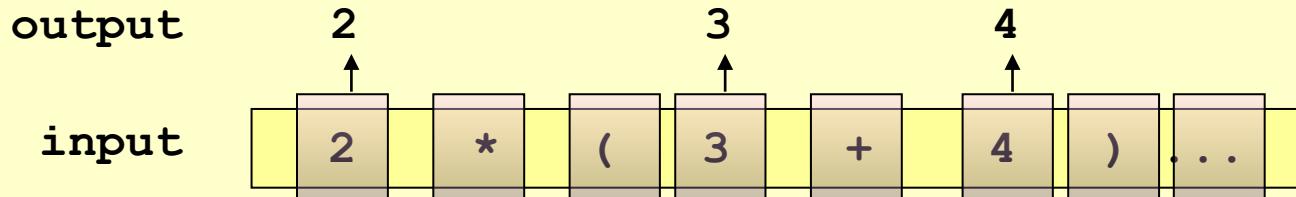
การหาความสำคัญของ operators

- \wedge แทนการยกกำลัง 7
- \wedge ทำก่อน + - * /
- *, / ทำก่อน +, -
- * สำคัญเท่ากับ / 5
- + สำคัญเท่ากับ - 3

```
map<char,int> priority =  
    {{'+',3}, {'-',3}, {'*',5}, {'/',5}, {'^',7}};
```

```
int p = priority[token];  
while( !s.empty() && priority[s.top()] >= p ) {  
    postfix += s.top(); s.pop();  
}
```

ขั้นตอนขึ้นเมื่อมีวงเล็บใน infix



- ภายนอกวงเล็บเสมือนเป็นนิพจน์ย่อย
- พบรหัสเปิด push เสมอ (มีความสำคัญมาก)
- วงเล็บเปิดในกองช้อน มีความสำคัญน้อยมาก
- พบรหัสปิด ลุยก์ pop จนกว่าจะพบรหัสเปิด

ความสำคัญของ operators กรณีมีวงเล็บ

- ขณะพิจารณาวงเล็บเปิด
 - มีความสำคัญมาก, จึง push (เสมอ
- แต่พิจารณาวงเล็บเปิดอยู่ในกองข้อมูล
 - มีความสำคัญน้อยสุด, เพื่อให้ตัวอื่น push ทับ, ยกเว้น)

```
int p = priority[token];
while( !s.empty() && priority[s.top()] >= p ) {
    postfix += s.top(); s.pop();
}
s.push(token);
```

```
int p = outPriority[token];
while( !s.empty() && inPriority[s.top()] >= p ) {
    postfix += s.top(); s.pop();
}
if (token == ')') s.pop(); else s.push(token);
```

ความสำคัญของ operators กรณีมีวงเล็บ

	+	-	*	/	[^]	()
ขนาดที่พบในนิพจน์ (อยู่นอกกองช้อน)	3	3	5	5	7	9	1
ขนาดอยู่ในกองช้อน	3	3	5	5	7	0	

```
map<char,int> outPriority =  
{{'+',3},{'-',3}, {'*',5}, {'/',5}, {'^',7}, {'(' ,9}, {')' ,1}};
```

```
map<char,int> inPriority =  
{{'+',3},{'-',3}, {'*',5}, {'/',5}, {'^',7}, {'(' ,0}};
```

operator ที่ทำการซ้ายไปขวา

	+	-	*	/	[^]	()
ขณะที่พบในนิพจน์ (อยู่นอกกองข้อมูล)	3	3	5	5	8	9	1
ขณะอยู่ในกองข้อมูล	3	3	5	5	7	0	

2 + 3 + 4 + 5

2 3 + 4 + 5 +

```

int p = outPriority[token];
while( !s.empty() && inPriority[s.top()] >= p ) {
    postfix += s.top(); s.pop();
}
if (token == ')') s.pop(); else s.push(token);

```

operator ที่ทำจากขวาไปซ้าย

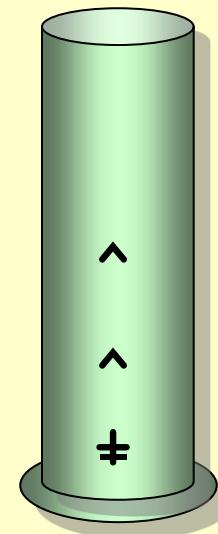
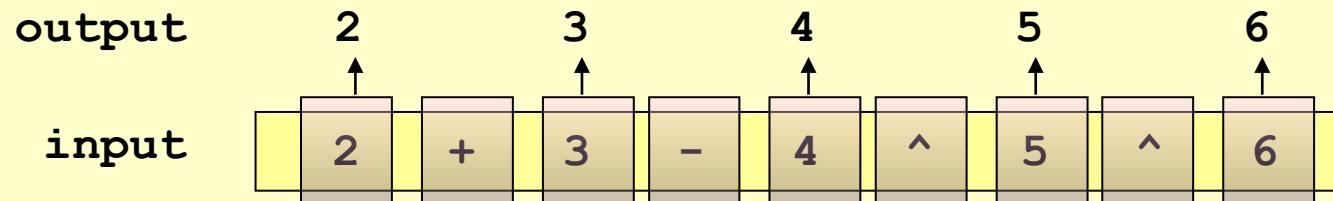
$2^{3^{4^5}} = 2^{\left(3^{\left(4^5\right)}\right)}$	<table border="1"> <thead> <tr> <th>+</th><th>-</th><th>*</th><th>/</th><th>^</th><th>(</th><th>)</th></tr> </thead> <tbody> <tr> <td>2</td><td>2</td><td>4</td><td>4</td><td>8</td><td>9</td><td>1</td></tr> <tr> <td>^</td><td>^</td><td>^</td><td>^</td><td>7</td><td>✓</td><td></td></tr> <tr> <td>3</td><td>3</td><td>5</td><td>5</td><td>0</td><td></td><td></td></tr> </tbody> </table>	+	-	*	/	^	()	2	2	4	4	8	9	1	^	^	^	^	7	✓		3	3	5	5	0		
+	-	*	/	^	()																							
2	2	4	4	8	9	1																							
^	^	^	^	7	✓																								
3	3	5	5	0																									
ขณะอยู่ในกองข้อมูล	<table border="1"> <tr> <td>2</td><td>3</td><td>4</td><td>5</td><td></td><td></td><td></td></tr> <tr> <td>^</td><td>^</td><td>^</td><td></td><td>^</td><td>^</td><td>^</td></tr> </table>	2	3	4	5				^	^	^		^	^	^														
2	3	4	5																										
^	^	^		^	^	^																							

```

int p = outPriority[token];
while( !s.empty() && inPriority[s.top()] >= p ) {
    postfix += s.top(); s.pop();
}
if (token == ')') s.pop(); else s.push(token);

```

ตัวอย่าง



สรุป

- ❖ ประยุกต์กองซ้อนในการแก้ปัญหาหลากหลาย
- ❖ การดำเนินการหลัก : push / pop / top
- ❖ สร้างกองซ้อนได้ง่ายด้วยอารเรย์
- ❖ ถ้าจ่องขนาดให้เพียงพอ การทำงานทุกครั้งเป็น $\Theta(1)$

ຖາມຕัวນາ
push
pop
top
/ စဉ်သော ဝါယာ
ဒုတိယံကပ် မေတ္တာမှု input