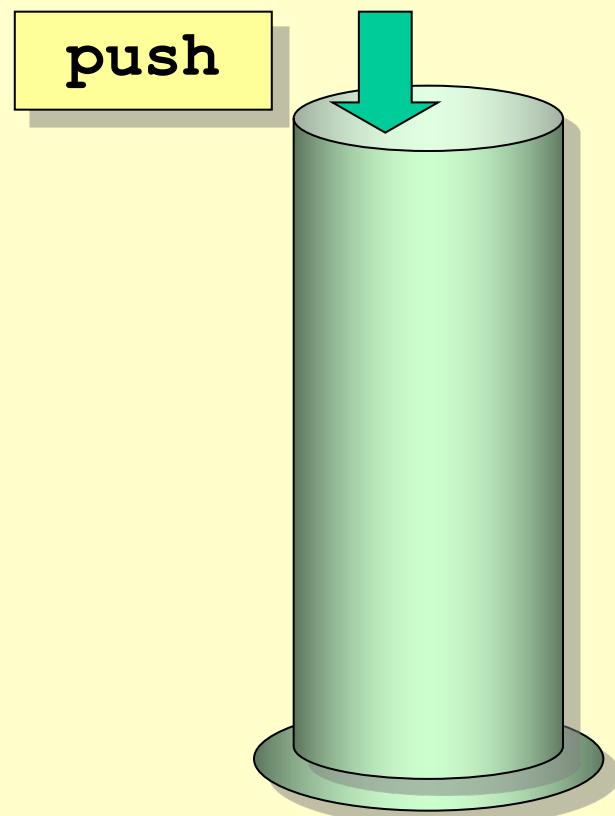


# กองซ้อน

## (Stack)

# การเพิ่ม/ลบข้อมูลในกองซ้อน

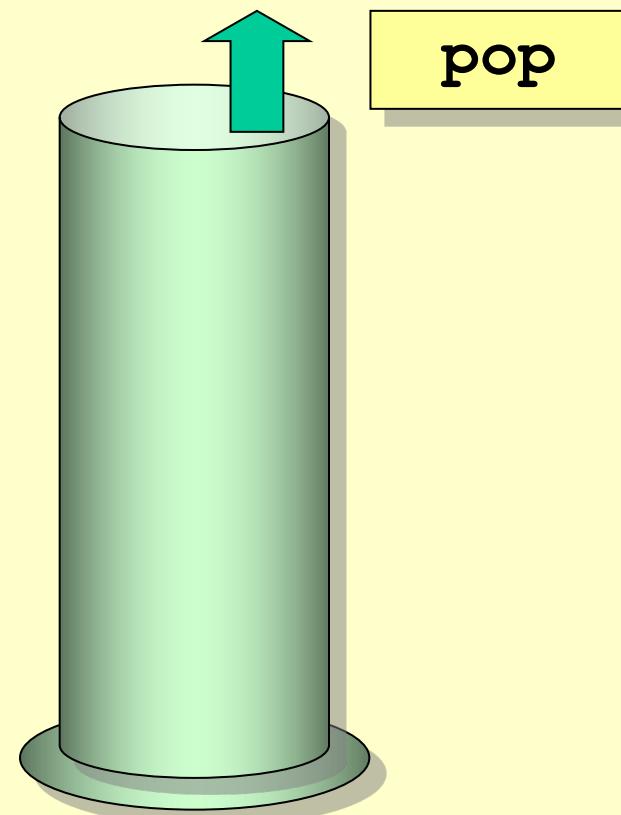
- ข้อมูล เข้าหลัง ออกก่อน (Last-In First-Out)



2-2

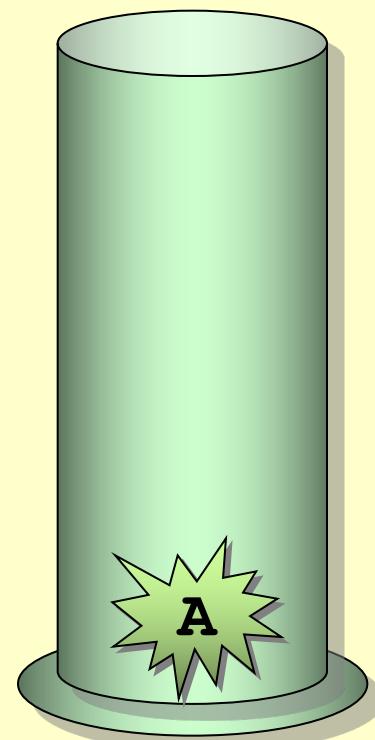
# การเพิ่ม/ลบข้อมูลในกองซ้อน

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



# การเพิ่ม/ลบข้อมูลในกองซ้อน

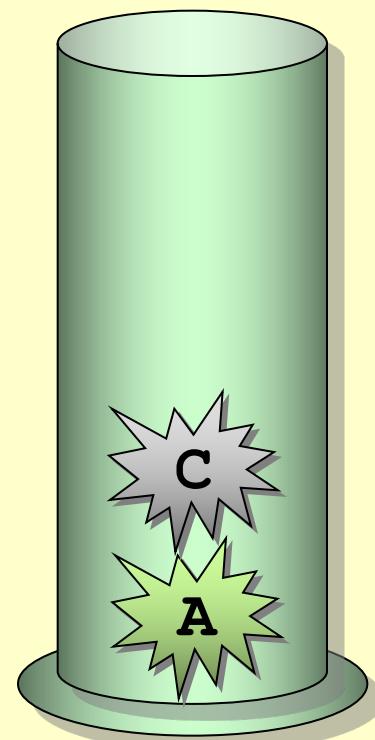
- ข้อมูล เข้าหลัง ออกก่อน (Last-In First-Out)



2-4

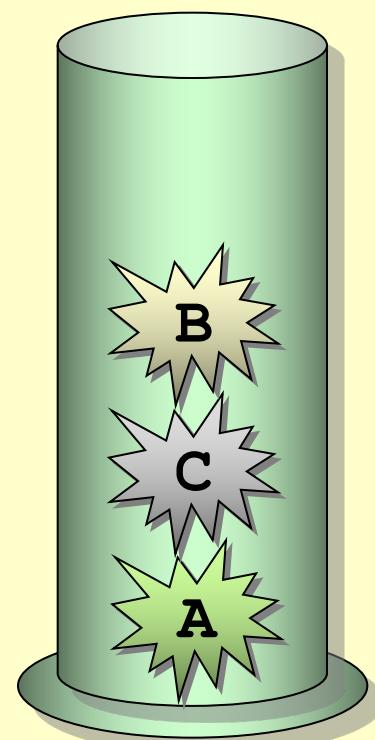
# การเพิ่ม/ลบข้อมูลในกองซ้อน

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



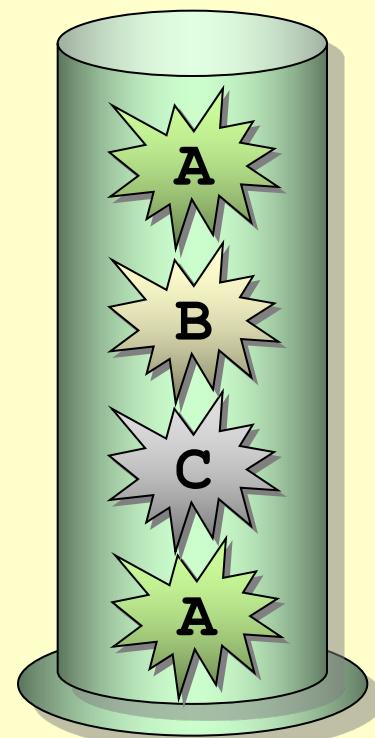
# การเพิ่ม/ลบข้อมูลในกองซ้อน

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



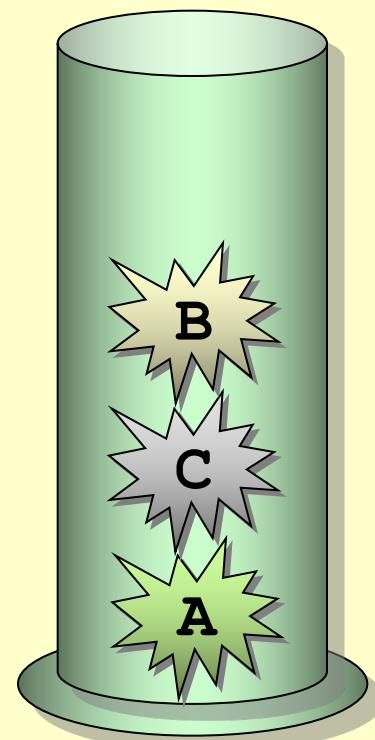
# การเพิ่ม/ลบข้อมูลในกองซ้อน

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



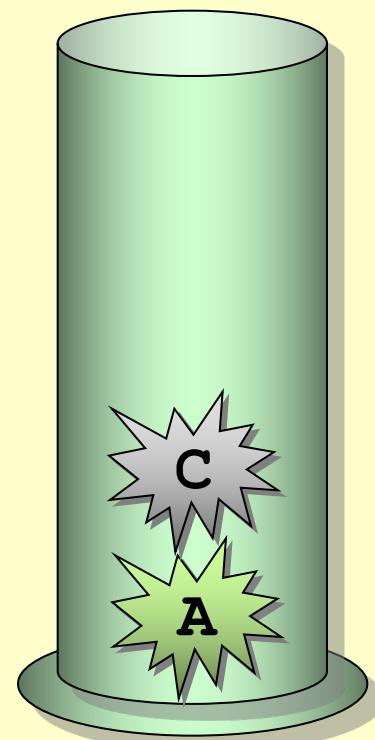
# การเพิ่ม/ลบข้อมูลในกองซ้อน

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



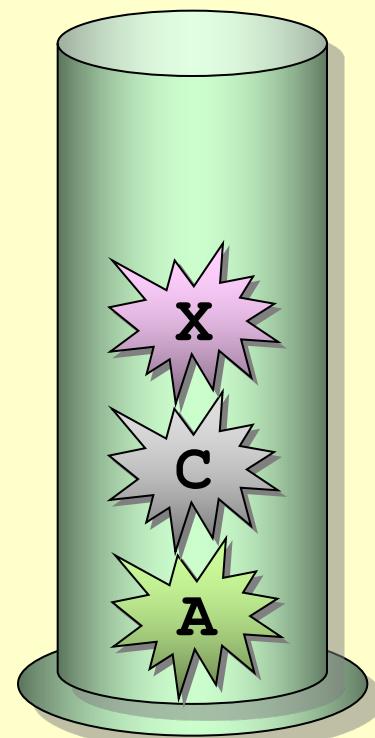
# การเพิ่ม/ลบข้อมูลในกองซ้อน

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



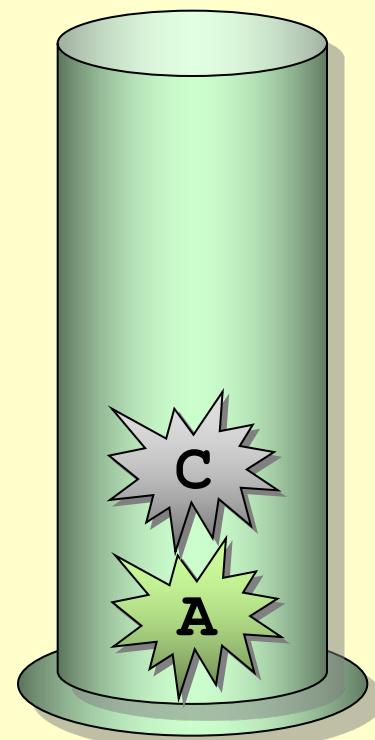
# การเพิ่ม/ลบข้อมูลในกองซ้อน

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



# การเพิ่ม/ลบข้อมูลในกองซ้อน

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



# ກອງຫົວນ : stack

```
bool      empty();  
unsigned   size();  
T         top();  
void      push(T element);  
void      pop();
```

# ตัวอย่างการใช้งาน Stack

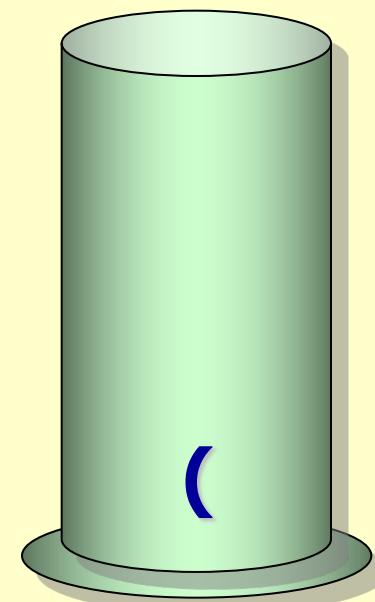
- ◆ การตรวจสอบโครงสร้างแบบช้อนกัน เช่น การใส่สิ่งเล็บ ( ) { } [ ] ...
- ◆ การจัดเก็บตัวแปรและการทำ function calls
- ◆ การประมวลผลนิพจน์ทางคณิตศาสตร์
- ◆ การทำ undo/redo
- ◆ การค้นคำตอบแบบ depth-first search
- ◆ ...

# 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

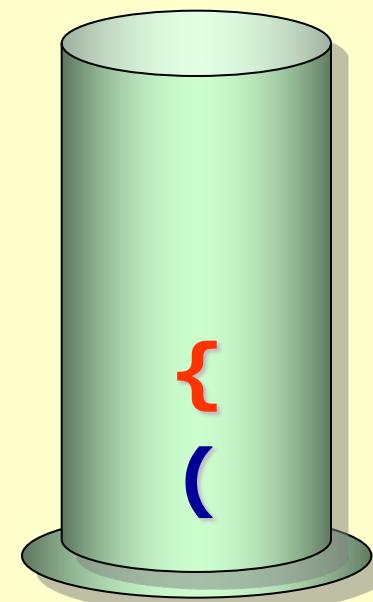
# ตัวอย่าง ๑

- ( { ( ) [ { } ] } )



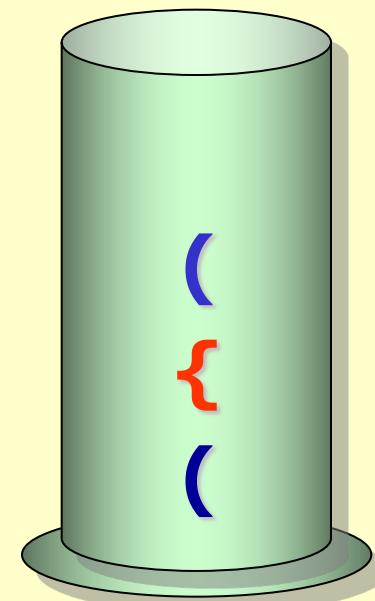
# ตัวอย่าง ๑

- ( { ( ) [ { } ] } )



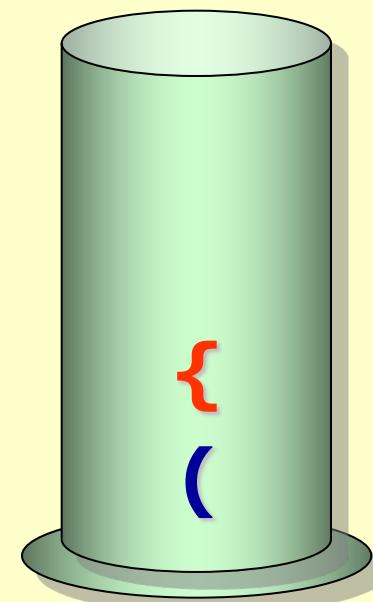
# ตัวอย่าง ๑

- ( { ( ) [ { } ] } )



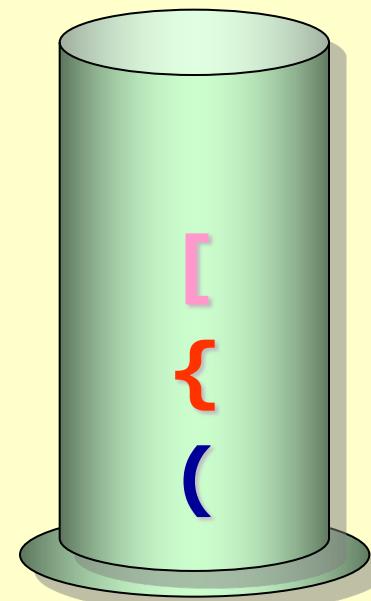
# ตัวอย่าง ๑

- ( { ( ) [ { } ] } )



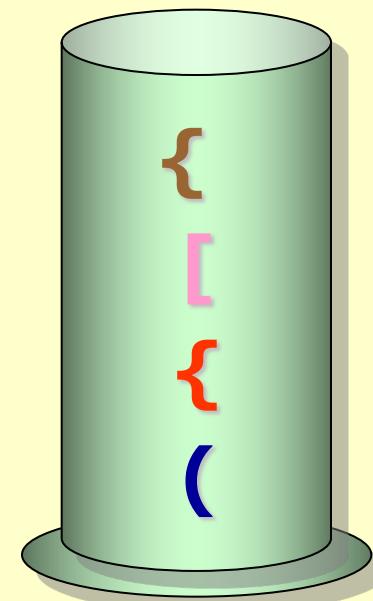
# ตัวอย่าง ๑

• ( { ( ) [ { } ] } )



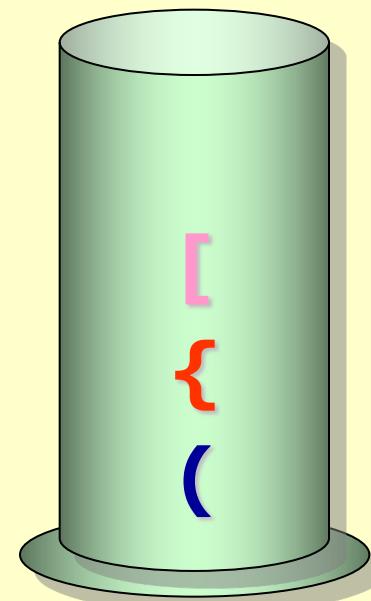
# ตัวอย่าง ๑

- ( { ( ) [ { } ] } )



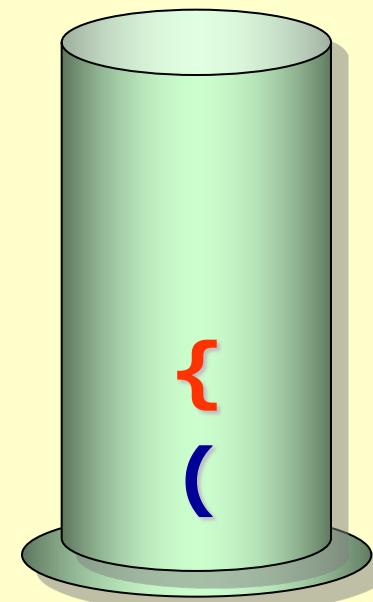
# ตัวอย่าง ๑

- ( { ( ) [ { } ] } )



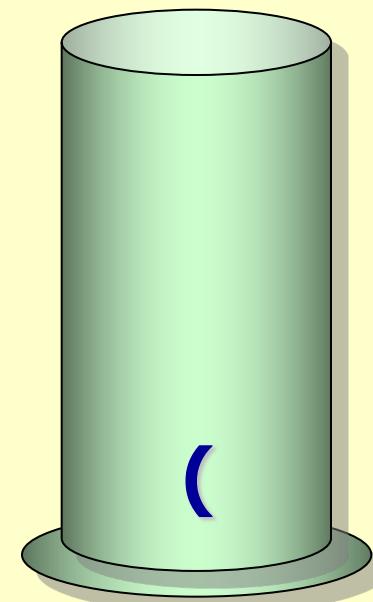
# ตัวอย่าง ๑

- ( { ( ) [ { } ] } )



# ตัวอย่าง ๑

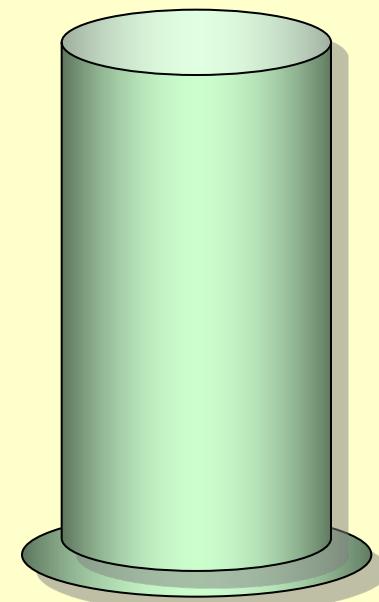
- ( { ( ) [ { } ] } )



# ตัวอย่าง ๑

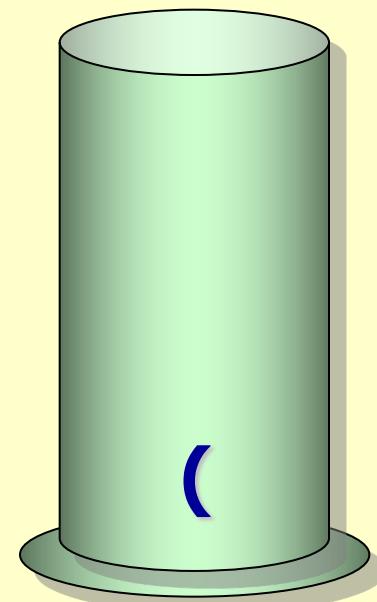
• ( { ( ) [ { } ] } )

ถูกต้อง



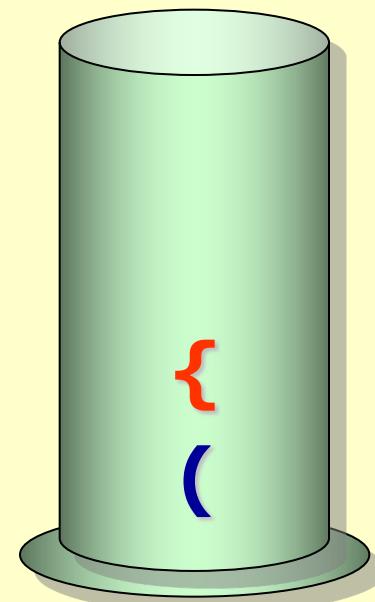
# ตัวอย่าง ๒

- ( { ( ) [ { ) } ] }



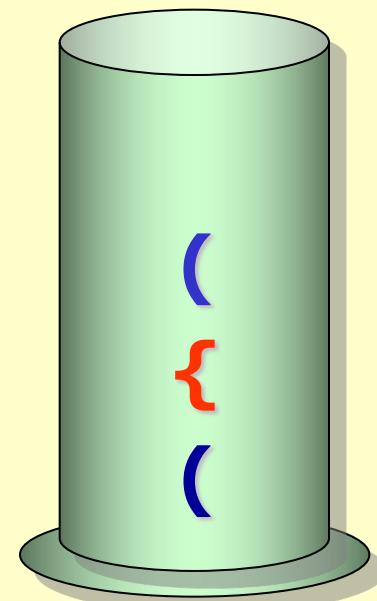
# ตัวอย่าง ๒

- ( { ( ) [ { ) ] } )



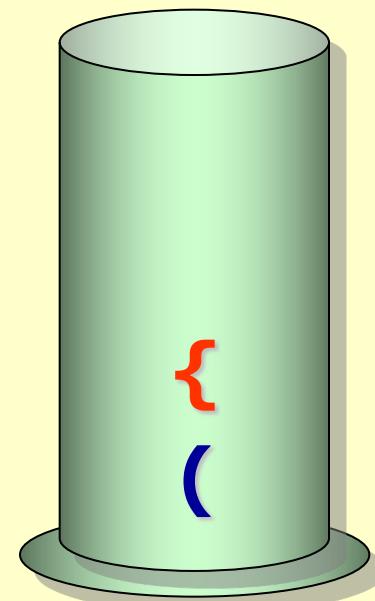
# ตัวอย่าง ๒

- ( { ( ) [ { ) ] } )



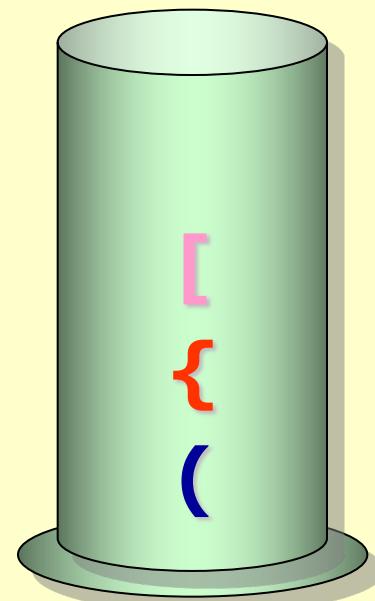
# ตัวอย่าง ๒

• ( { ( ) [ { ) } ] }



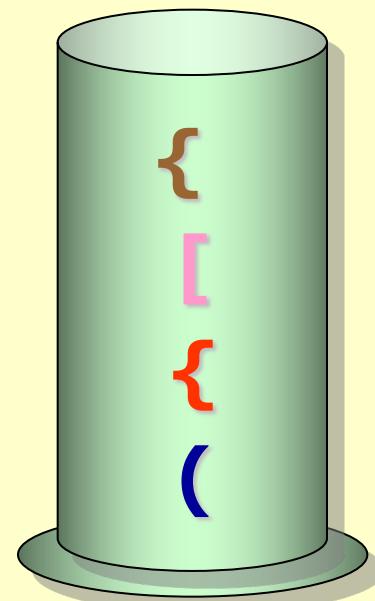
# ตัวอย่าง ๒

- ( { ( ) [ { ) ] } )



# ตัวอย่าง ๒

- ( { ( ) [ { ) ] } )

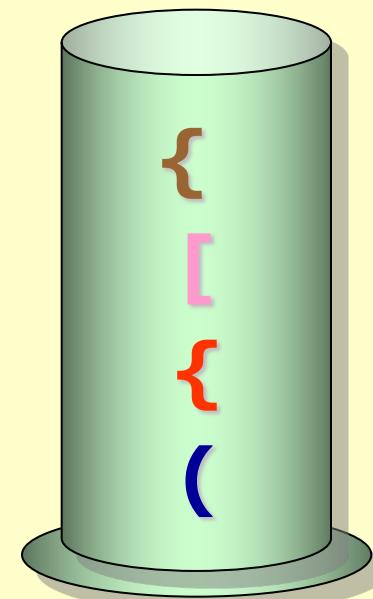


# ตัวอย่าง ๒

• ( { ( ) [ { ) ] } )

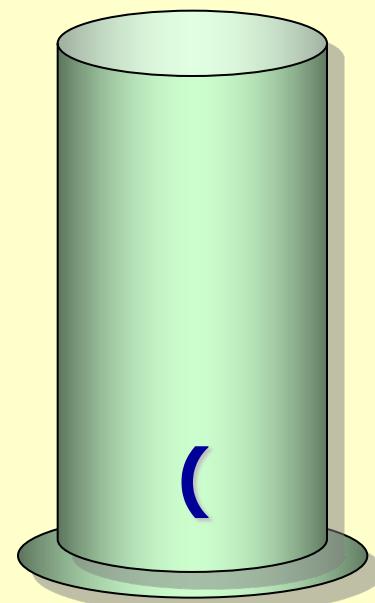


วงเล็บปิด  
ไม่ตรงกับเปิด  
ผิด !!



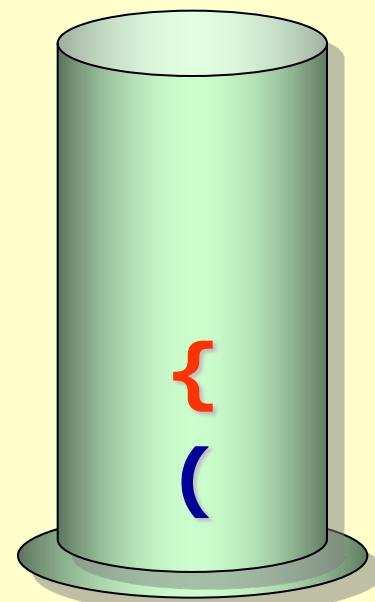
# ตัวอย่าง ๓

- ( { ( ) } ) ]



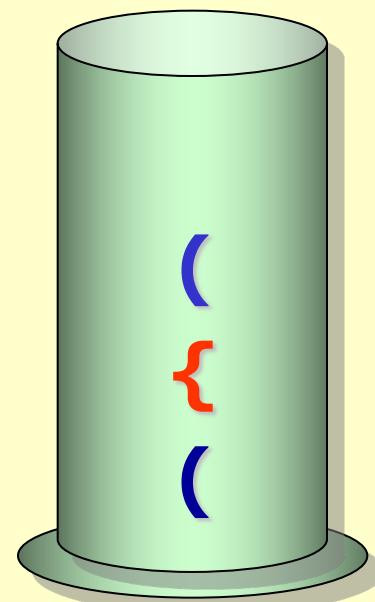
# ตัวอย่าง ๓

• ( { ( ) } ) ]



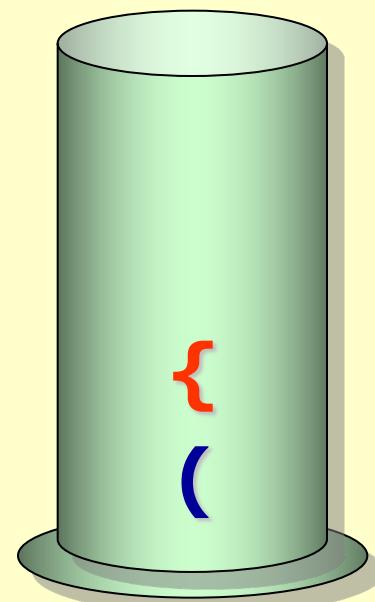
# ตัวอย่าง ๓

• ( { ( ) } ) ]



# ตัวอย่าง ๓

• ( { ( ) } ) ]

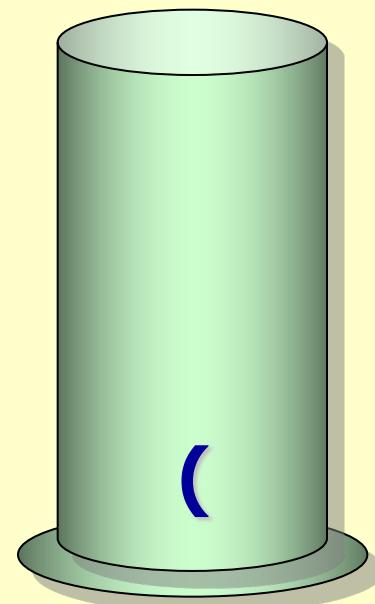


8-10

10/11/66 107

# ตัวอย่าง ๓

• ( { ( ) } ) ]

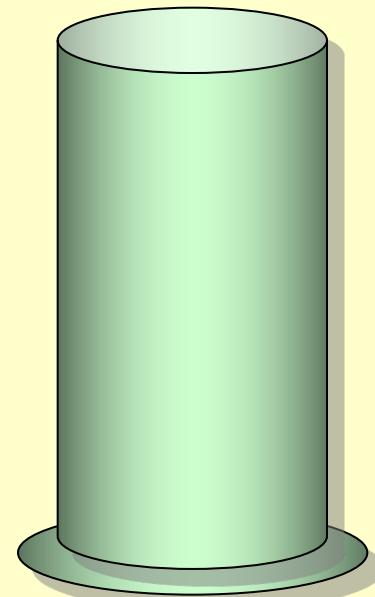


# ตัวอย่าง ๓

• ( { ( ) } ) ]



ยังไม่หมด  
แต่ stack ว่าง  
ผิด !!

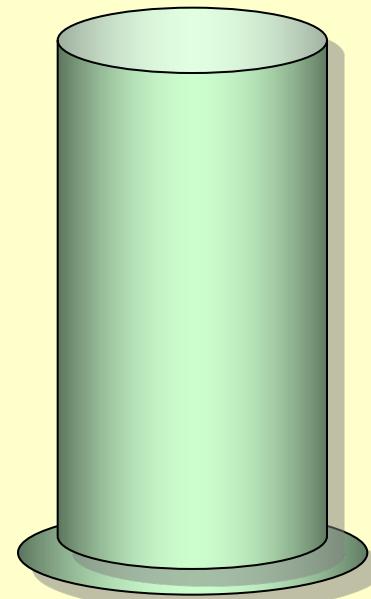


# ตัวอย่าง ๓

• ( { ( ) } ) ]

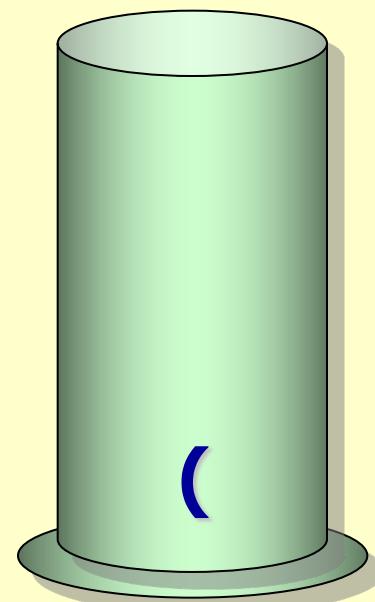


ยังไม่หมด  
แต่ stack ว่าง  
ผิด !!



# ตัวอย่าง ๔

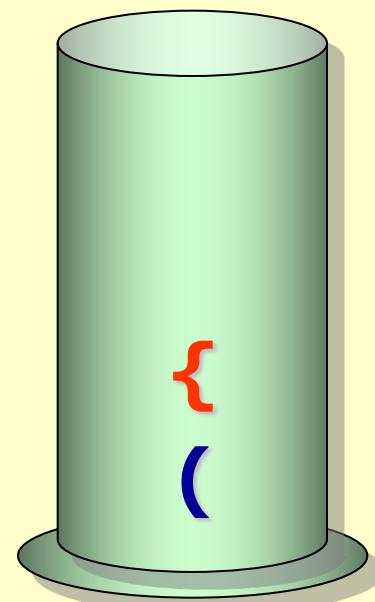
- ( { ( ) [ { }



9-3

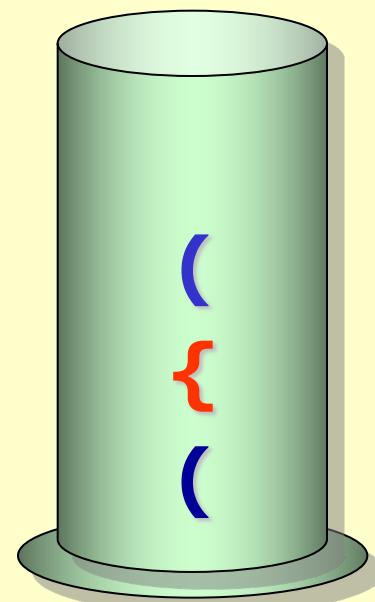
# ตัวอย่าง ๔

- ( { ( ) [ { }



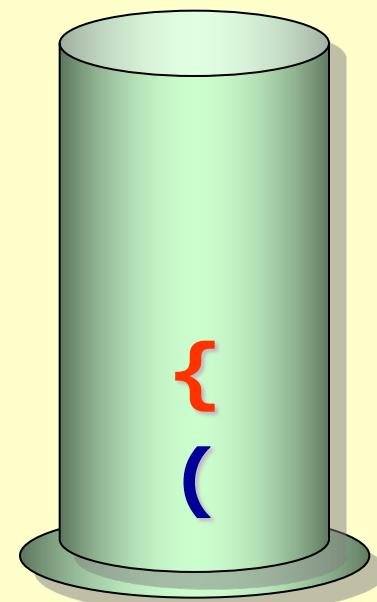
# ตัวอย่าง ๔

• ( { ( ) [ { }



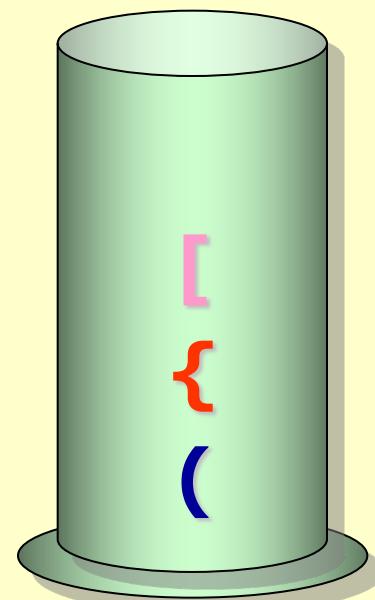
# ตัวอย่าง ๔

• ( { ( ) [ { }



# ตัวอย่าง ๔

• ( { ( ) [ { }

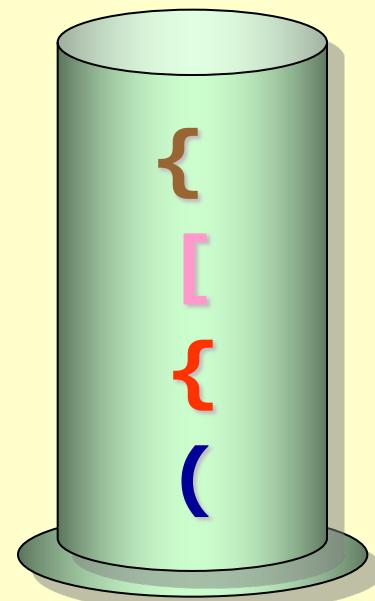


9-11

10/11/66 124

# ตัวอย่าง ๔

• ( { ( ) [ { }



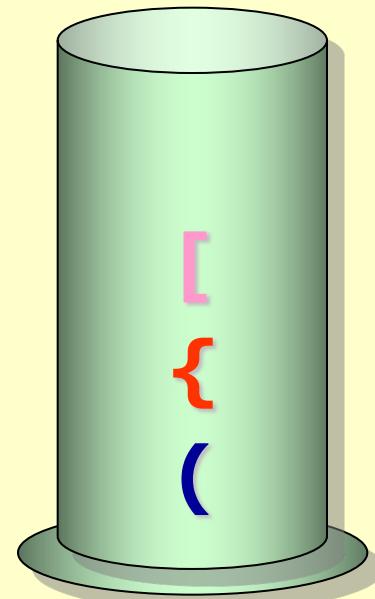
9-13

10/11/66 126

# ตัวอย่าง ๔

- ( { ( ) [ { }

หมดแล้ว  
แต่ stack ไม่ว่าง  
ผิด !!



# การใช้กองช้อนภายใน java virtual machine

- ตัว jvm ใช้กองช้อน (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();  
}
```

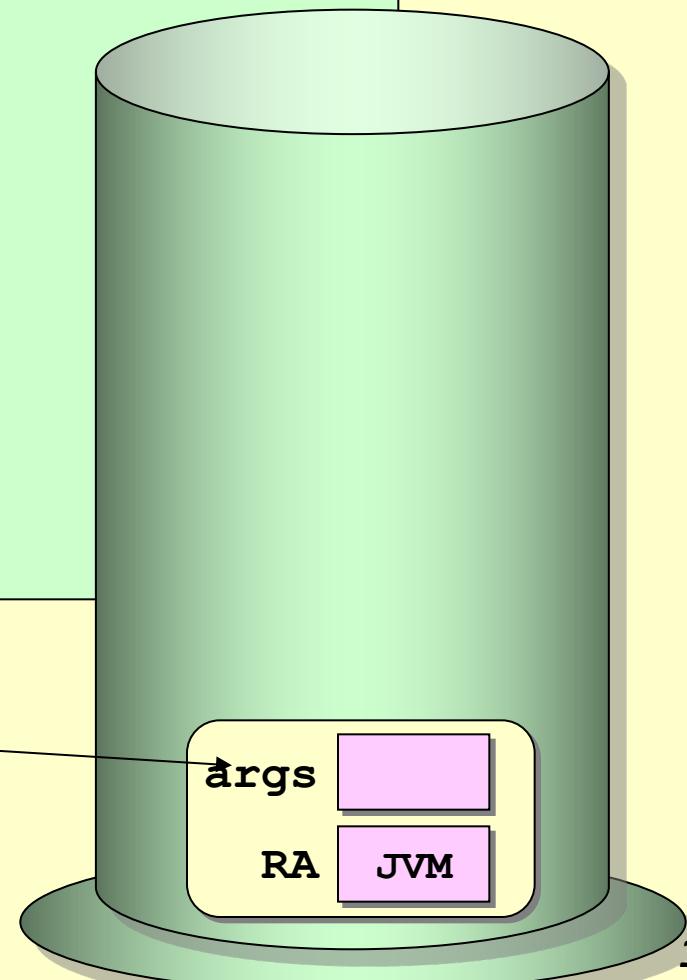
Process terminated with status -1073741571

# 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)

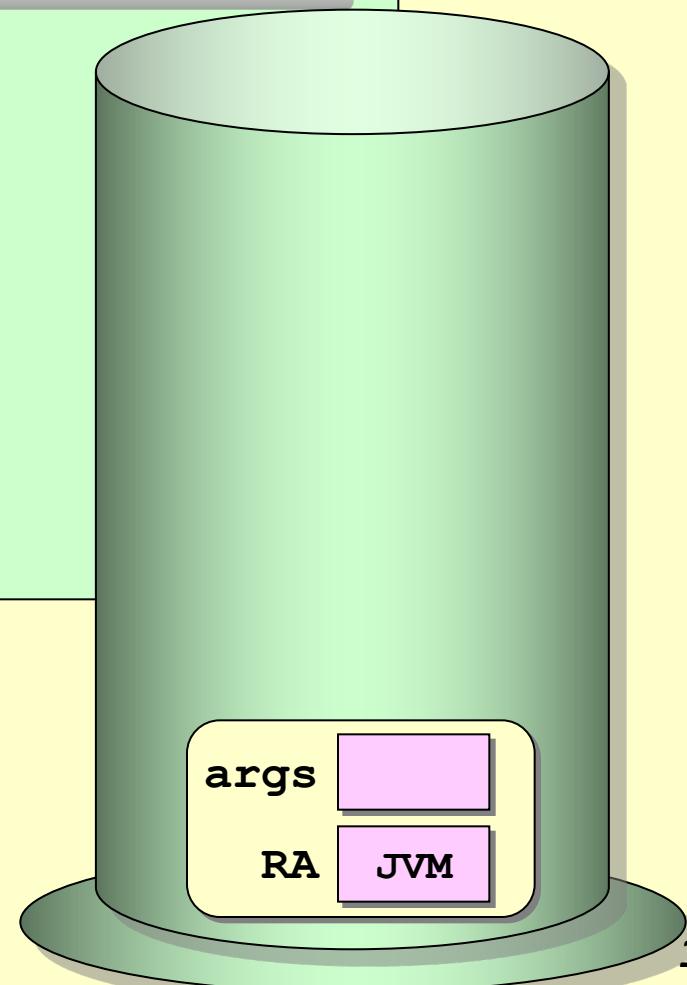
parameters &  
local variables



# 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:}
```

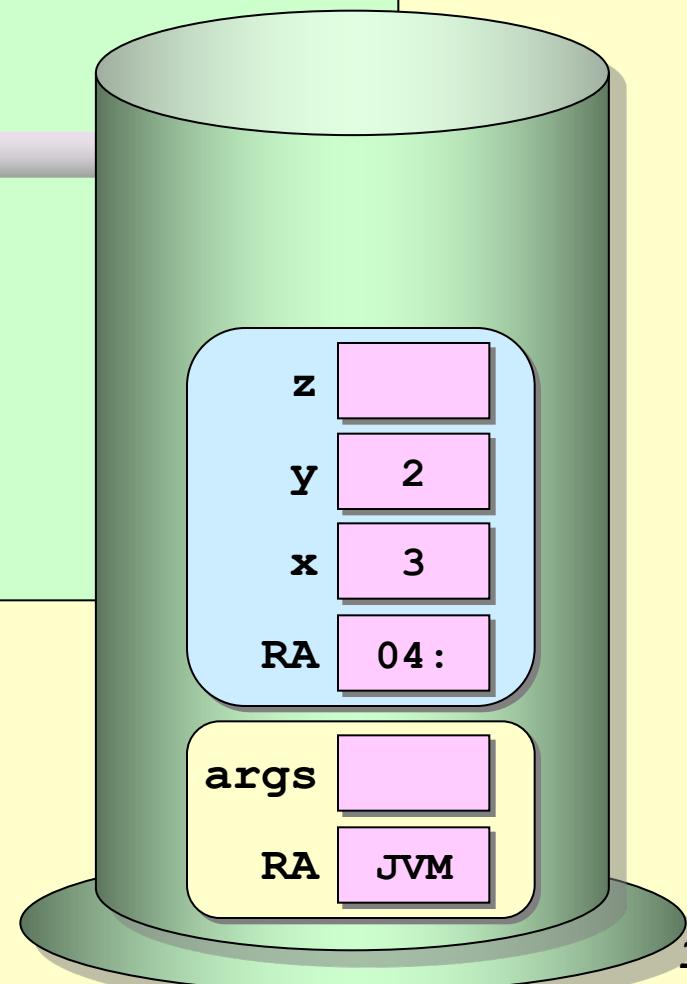
กรอบกองชอน  
(Stack Frame)



# 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:}
```

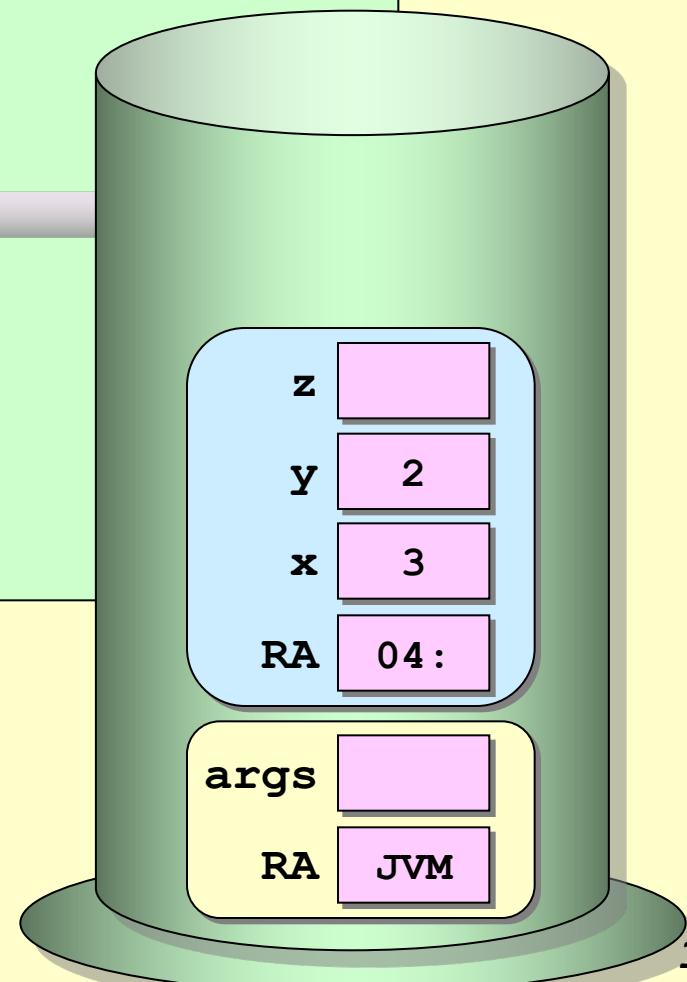
กรอบกองชอน  
(Stack Frame)



# 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:}
```

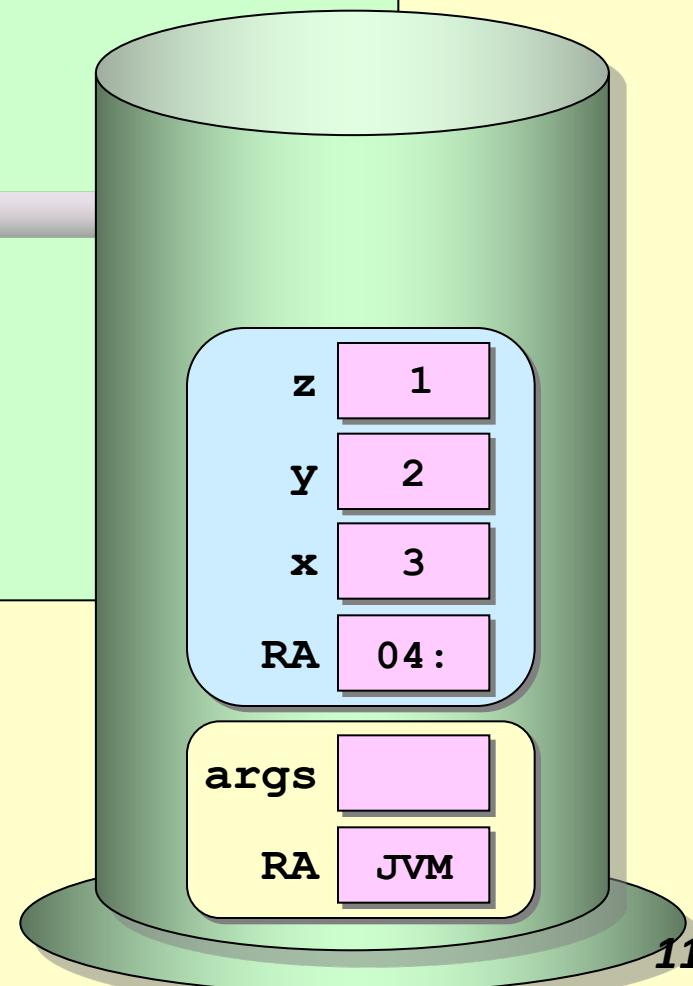
กรอบกองชอน  
(Stack Frame)



# 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:}
```

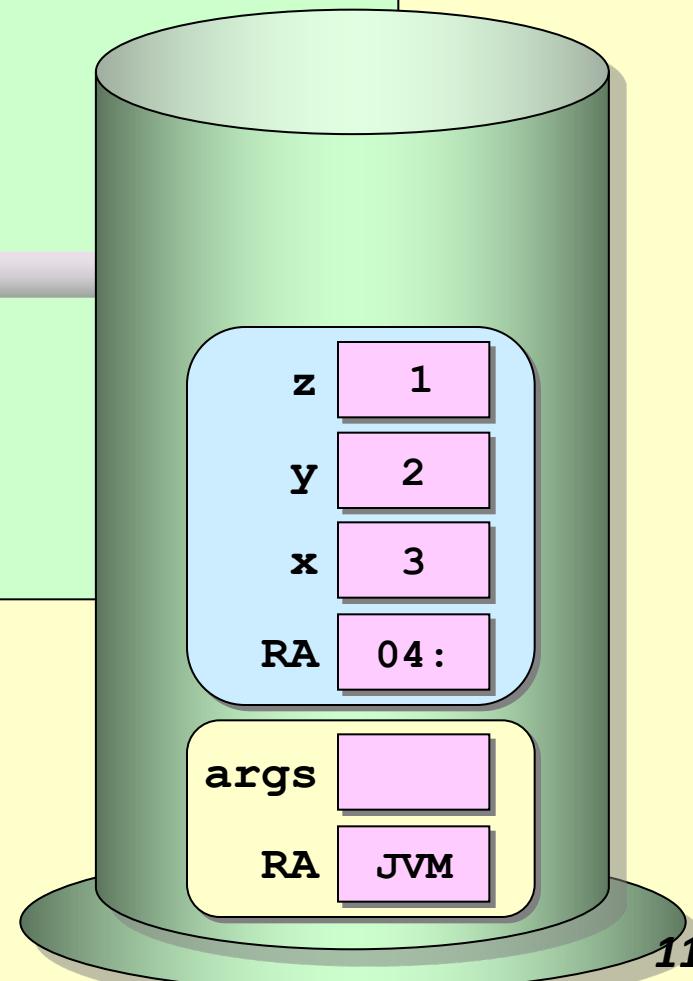
กรอบกองชอน  
(Stack Frame)



# 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:}
```

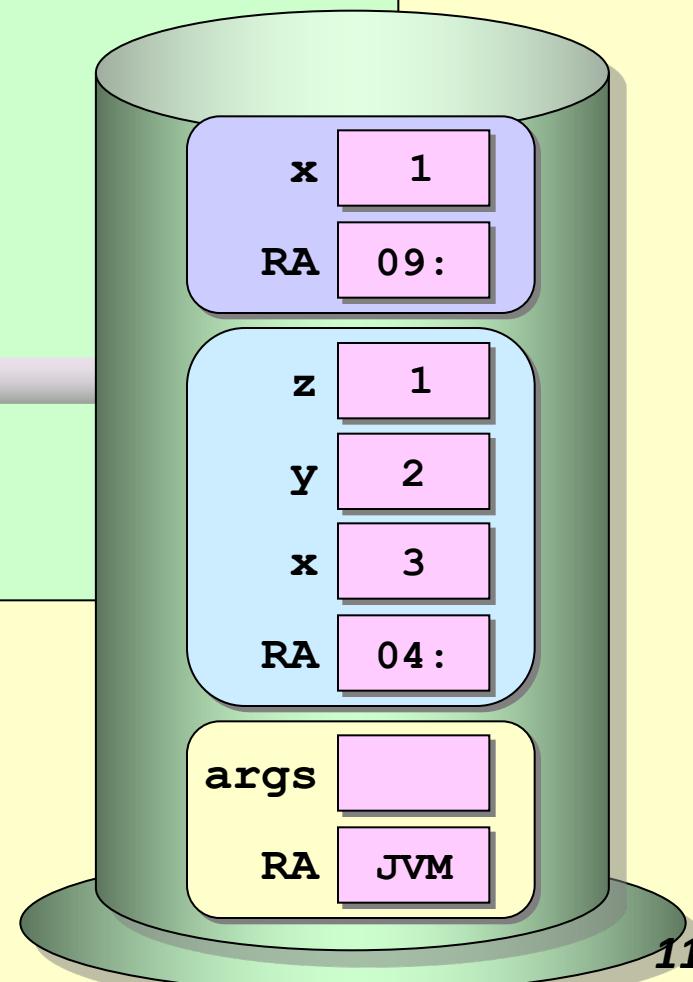
กรอบกองซ้อน  
(Stack Frame)



# 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:}
```

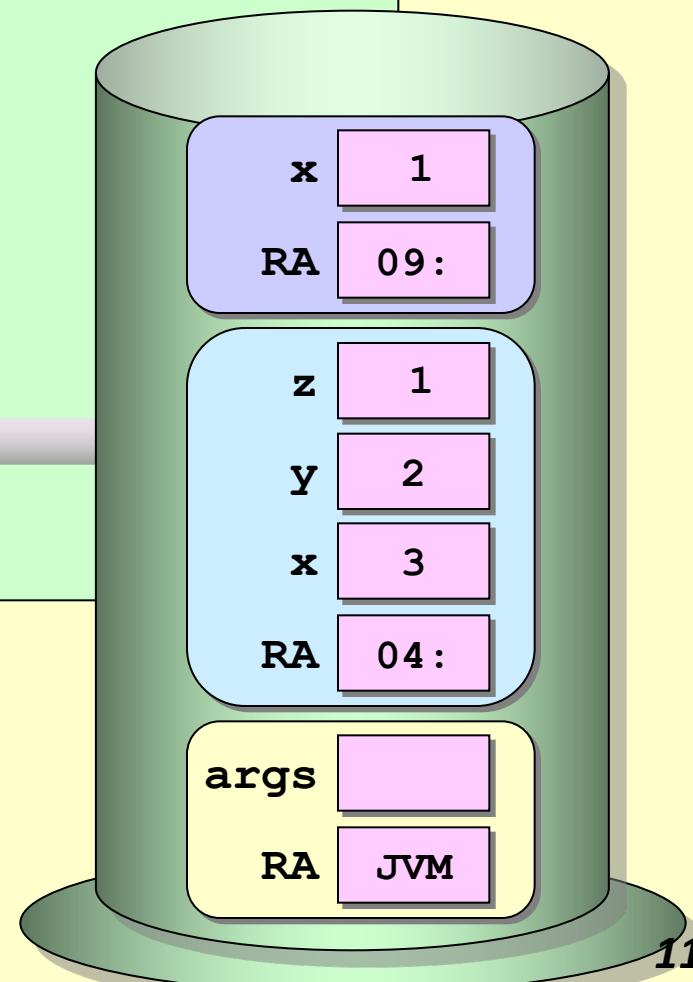
กรอบกองชอน  
(Stack Frame)



# 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:}
```

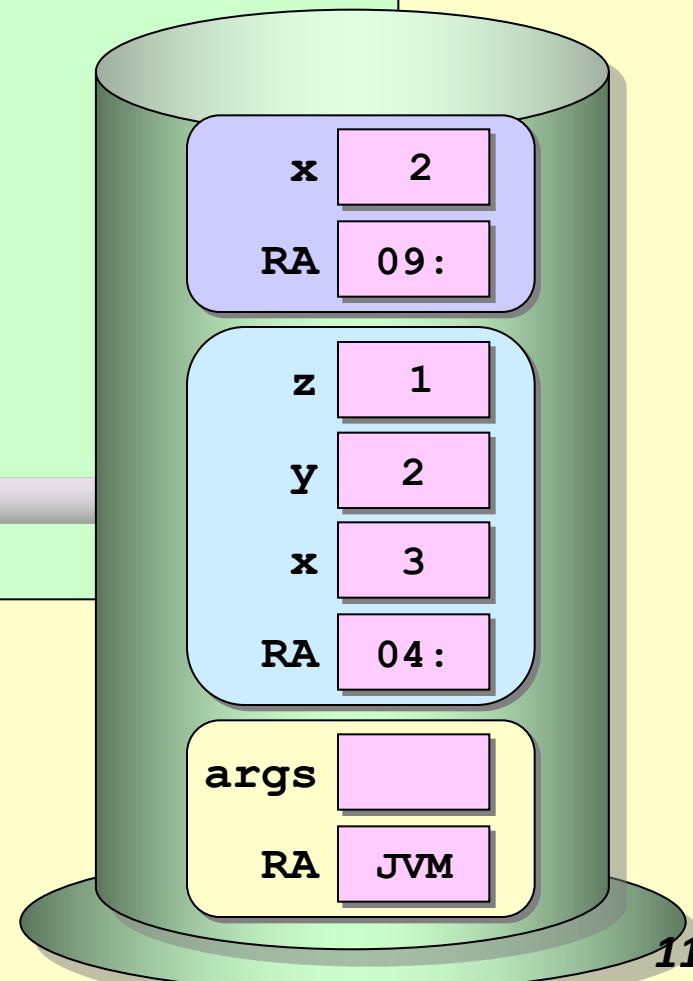
กรอบกองชอน  
(Stack Frame)



# 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:}
```

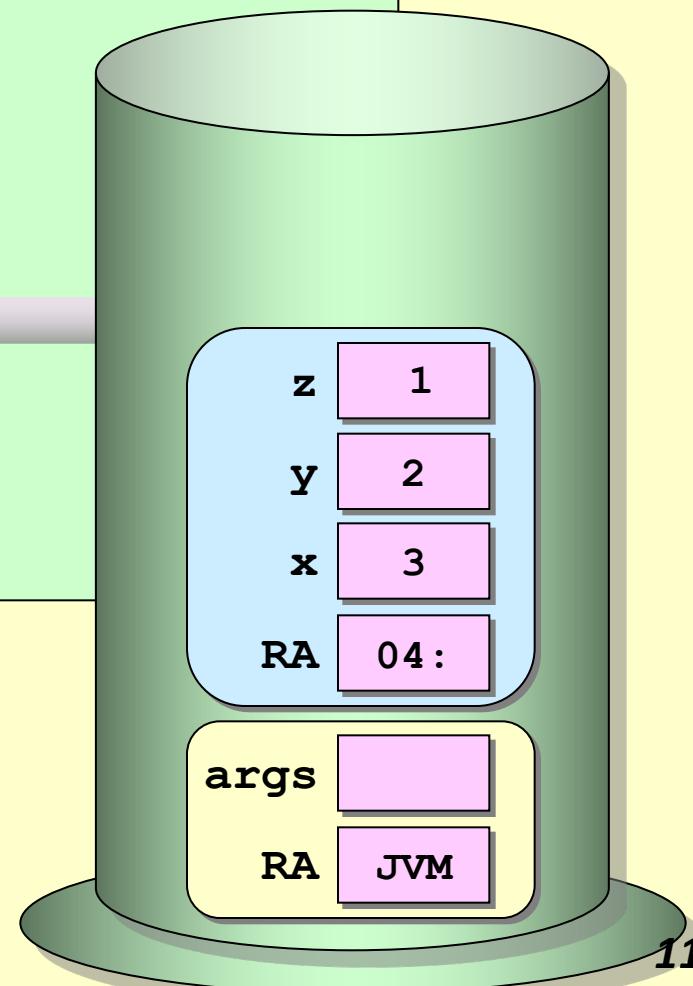
กรอบกองชอน  
(Stack Frame)



# 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:}
```

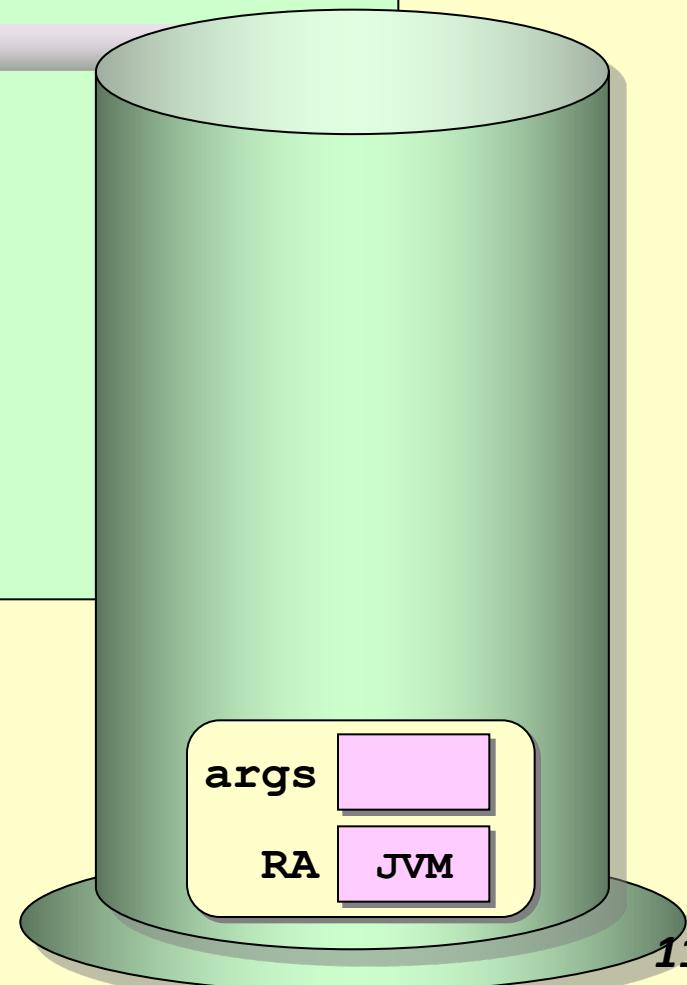
กรอบกองข้อมูล  
(Stack Frame)



# 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:}
```

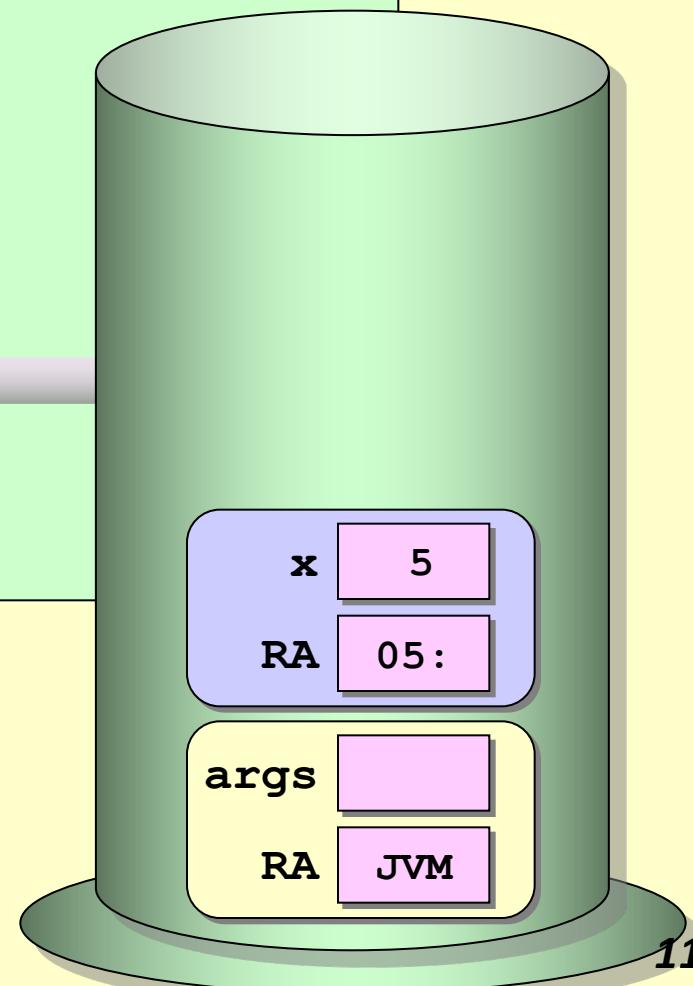
กรอบกองซ้อน  
(Stack Frame)



# 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:}
```

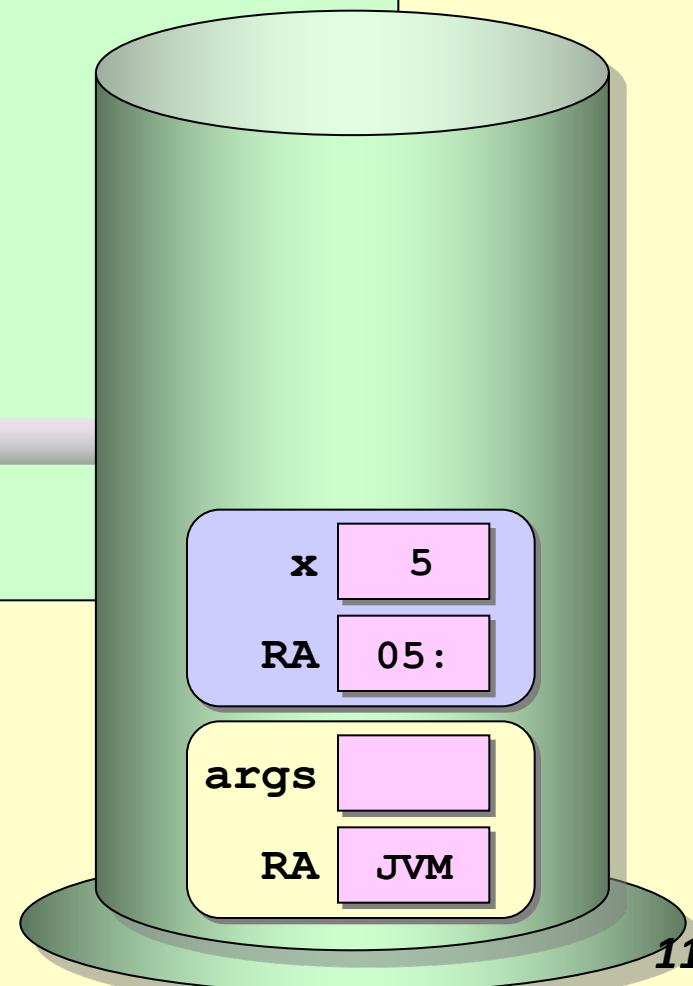
กรอบกองข้อมูล  
(Stack Frame)



# 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:}
```

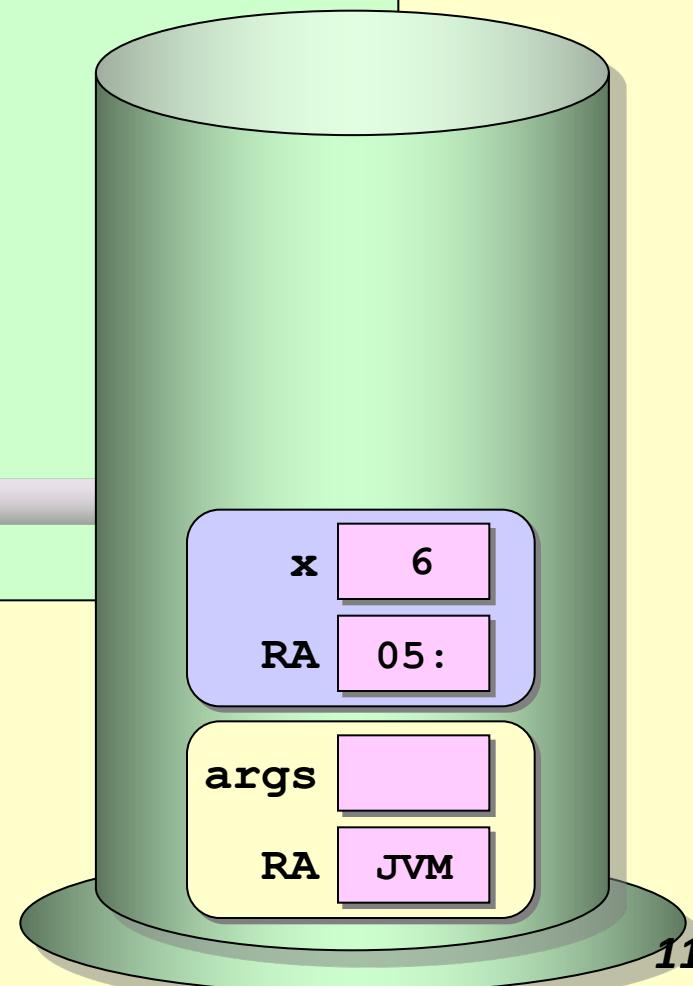
กรอบกองชอน  
(Stack Frame)



# 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:}
```

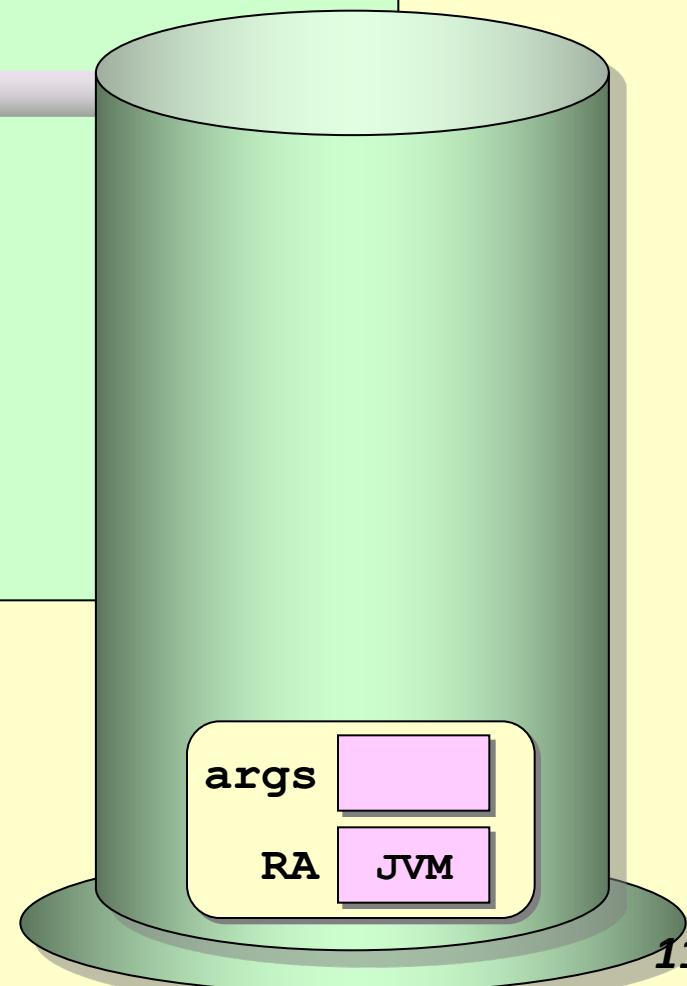
กรอบกองข้อมูล  
(Stack Frame)



# 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:}
```

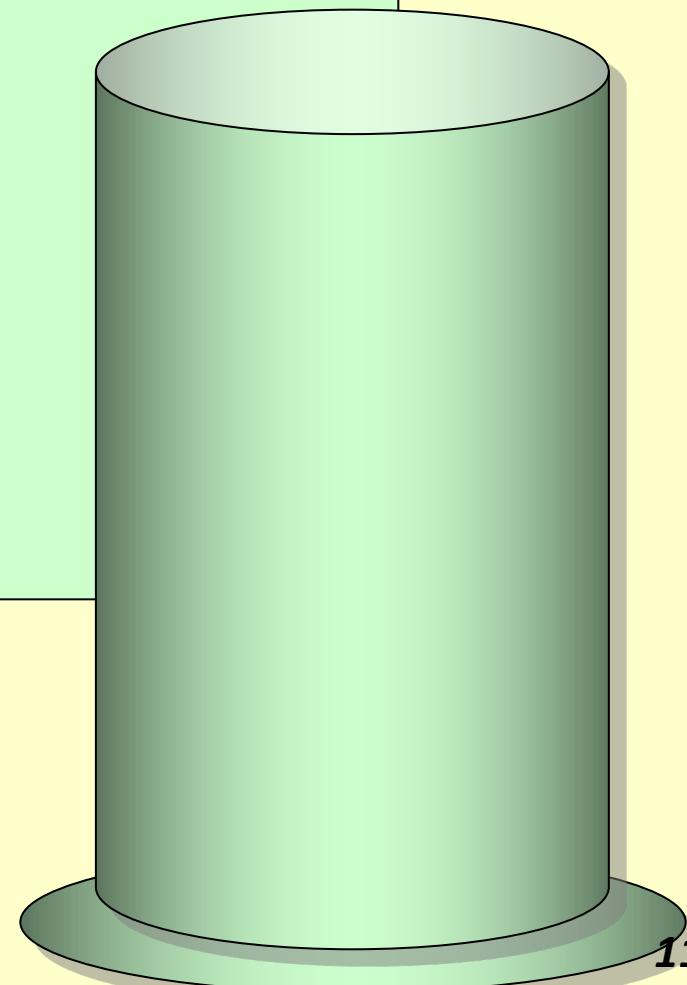
กรอบกองซ้อน  
(Stack Frame)



# 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:}
```

กรอบกองช้อน  
(Stack Frame)



# นิพจน์ Infix และ Postfix

- infix (เติมกลาง)
  - $a + b * c / d - 2$ ,  $(a + b) * c / (d - 2)$
  - ต้องกำหนดลำดับการทำงานของ operators
  - ใช้วงเล็บช่วย
- postfix (เติมหลัง)
  - $a b c * d / + 2 -$ ,  $a b + c * d 2 - /$
  - ลำดับการทำงานของ operators คือจาก ซ้ายไปขวา
  - ไม่จำเป็นต้องมีวงเล็บ

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

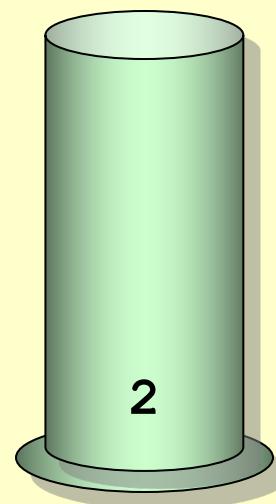
# การใช้กองซ้อนคำนวนค่าของนิพจน์เติมหลัง

- $2\ 3\ +\ 4\ 5\ -\ 6\ *\ +$  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



$$\boxed{2 \quad 3 \quad + \quad 4 \quad -}$$



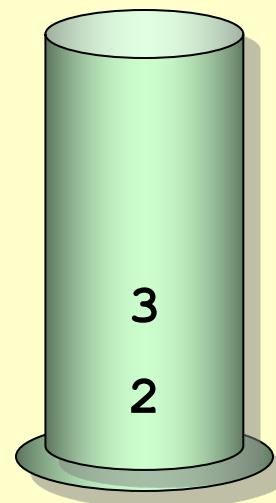
14-4

10/11/66 182

# ตัวอย่าง



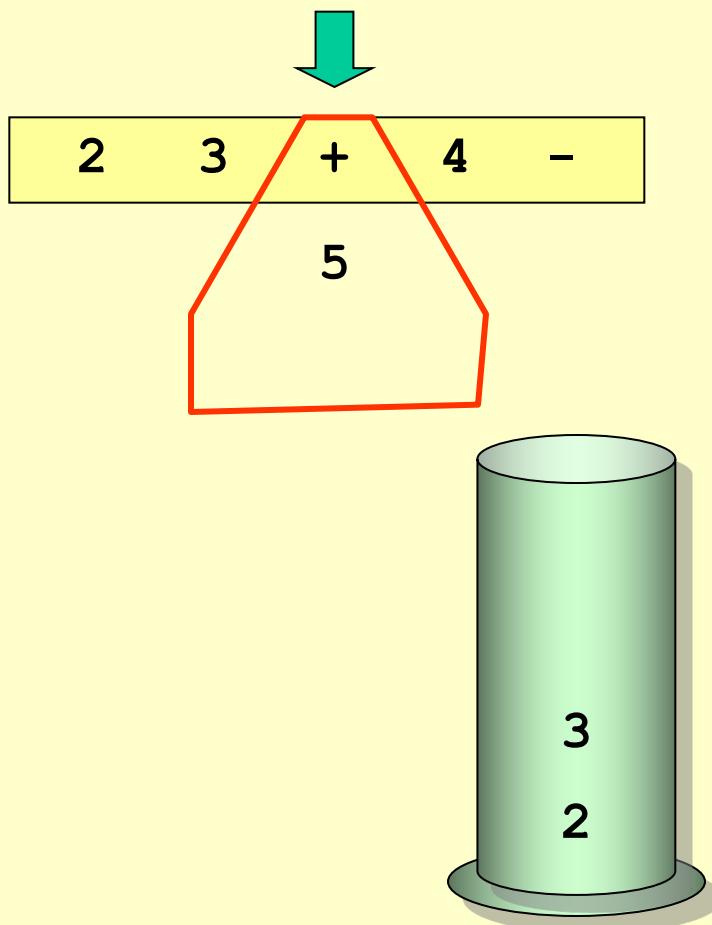
$$\begin{array}{r} 2 \quad 3 \\ + \quad 4 \\ \hline \end{array}$$



14-7

10/11/66 185

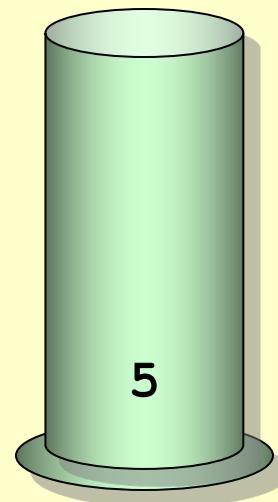
# ตัวอย่าง



# ตัวอย่าง

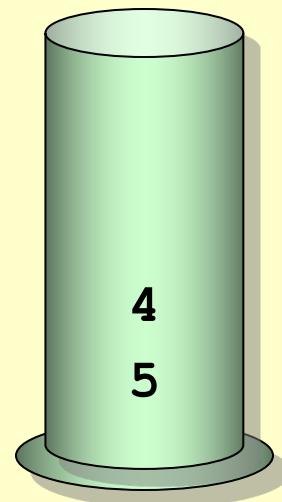


2	3	+	4	-
---	---	---	---	---



# ตัวอย่าง

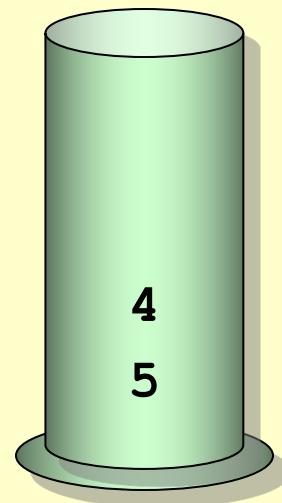
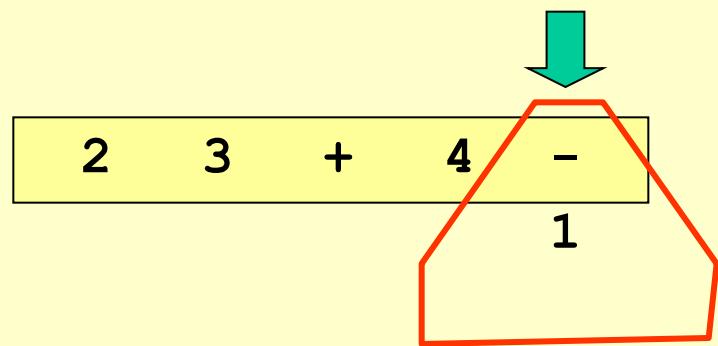
$$\begin{array}{ccccccc} & & & & & & \\ & & & & & & \\ \textcolor{red}{2} & \textcolor{blue}{3} & + & \textcolor{red}{4} & - & \textcolor{blue}{5} & \end{array}$$



14-18

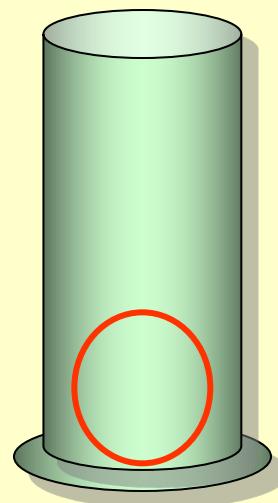
10/11/66 196

# ตัวอย่าง



# ตัวอย่าง

$$\begin{array}{ccccccc} 2 & 3 & + & 4 & - \\ & & & & 1 \end{array}$$



ผลลัพธ์อยู่  
บนกองขอน

# jvm เป็น stack machine

```
int a = 1;
```

```
iconst_1  
istore_1
```

```
int b = 2;
```

```
iconst_2  
istore_2
```

```
int c = 3;
```

```
iconst_3  
istore_3
```

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

```
iload_1
```

```
iload_2
```

```
iload_3
```

```
iload_1
```

```
iadd
```

```
imul
```

```
iadd
```

```
istore_4
```

infix

postfix

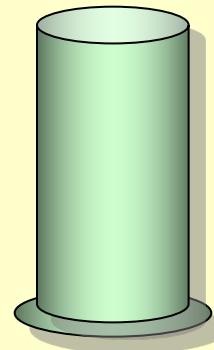
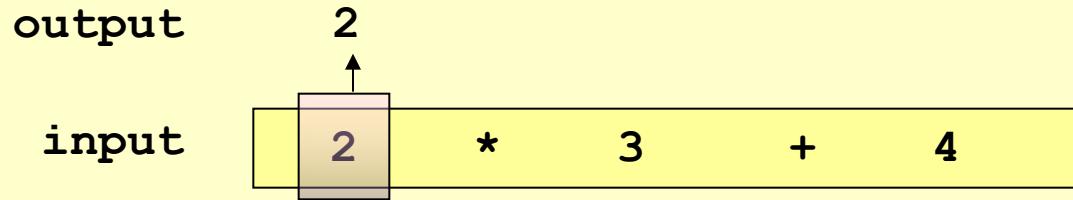
# การใช้กองช้อนช่วยแปลง infix เป็น postfix

- 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

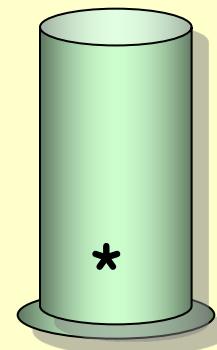
# infix -> postfix

```
string infix2postfix(string &infix) {  
    int n = infix.length();  
    string postfix = "";  
    stack<char> s;  
    for (int i=0; i<n; i++) { ← ดูที่ลະตัว  
        char token = infix[i];  
        if ( token เป็น operand ) {  
            postfix += token; ← ถ้าเป็น operand, เพิ่มต่อในผลลัพธ์  
        } else {  
            while( ???? ) {  
                postfix += s.top(); ← ถ้าเป็น operator  
                อาจ pop operators ใน stack  
                ออกเป็นผลลัพธ์  
                s.pop();  
            }  
            s.push(token); ← ตามด้วยการ push operator ตัวใหม่  
        }  
    }  
    while(!s.empty()) {postfix += s.top(); s.pop();}  
    return postfix;  
}
```

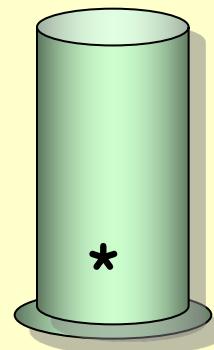
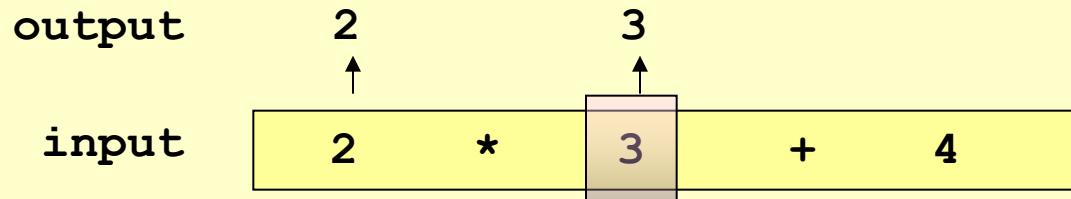
# ความสำคัญของ operators



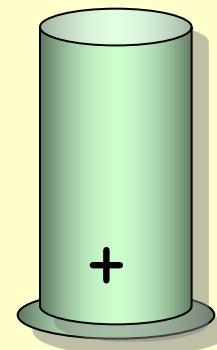
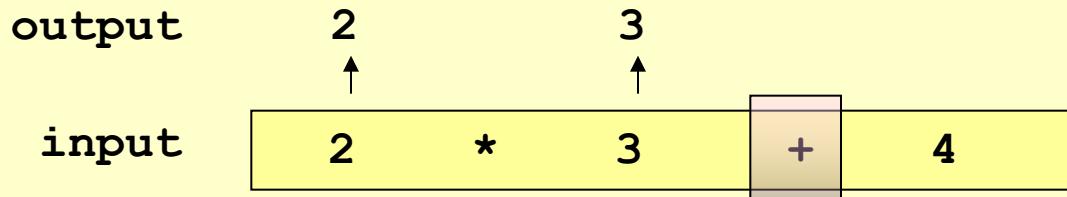
# ความสำคัญของ operators



# ความสำคัญของ operators



# ความสำคัญของ operators

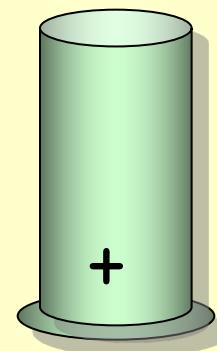


\* มาก่อน และสำคัญกว่า +

pop ออก ถ้า operator บนกองช้อนสำคัญกว่า operator ใหม่

# ความสำคัญของ operators

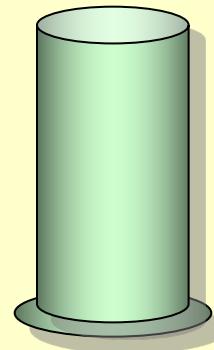
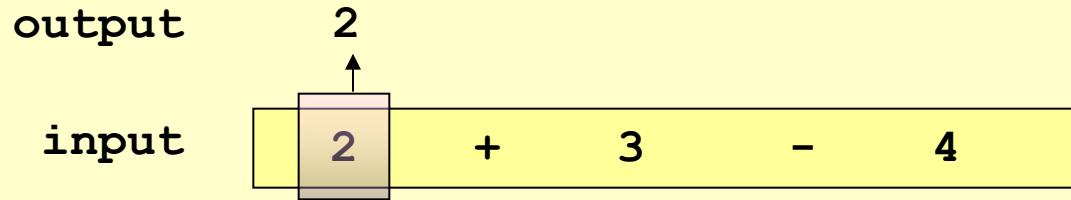
output	2	3	4		
input	2	*	3	+	4



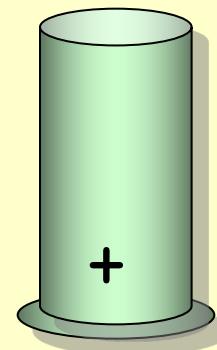
\* มาก่อน และสำคัญกว่า +

pop ออก ถ้า operator บนกองช้อนสำคัญกว่า operator ใหม่

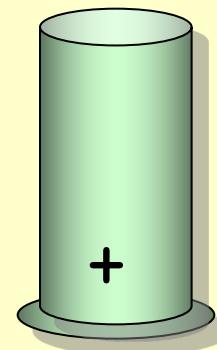
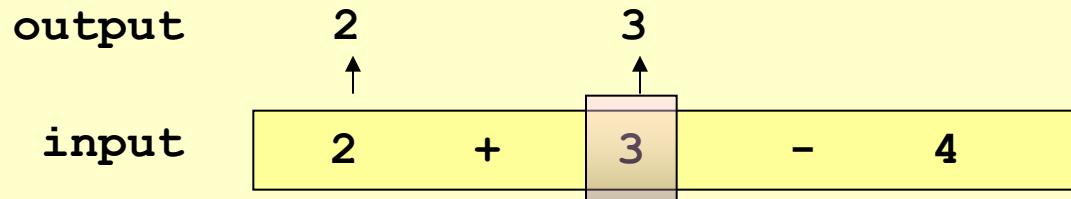
# ความสำคัญของ operators



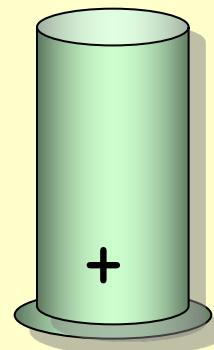
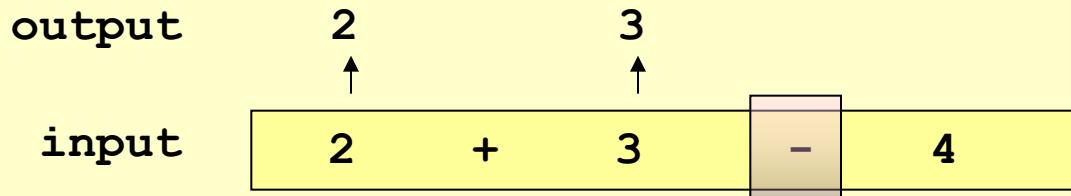
# ความสำคัญของ operators



# ความสำคัญของ operators



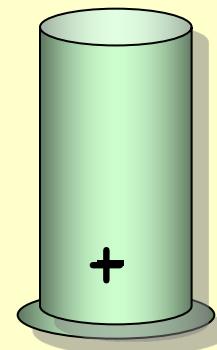
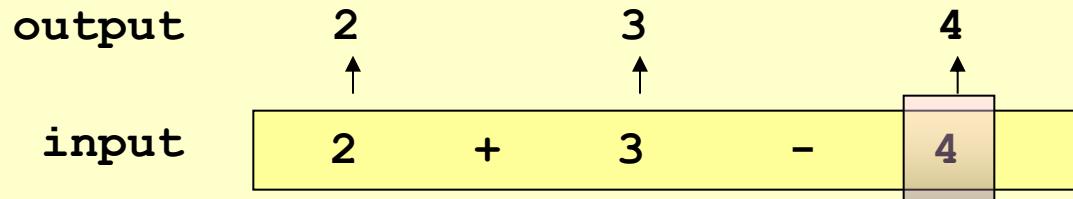
# ความสำคัญของ operators



+ มา ก่อน  
แล ะ สำคัญ เท่า กับ -

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

# ความสำคัญของ operators

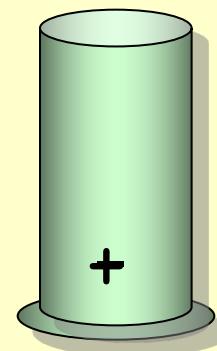


+ มา ก่อน  
แล ะ สำคัญ เท่า กับ -

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

# ความสำคัญของ operators

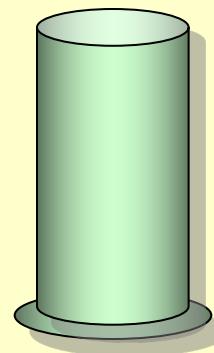
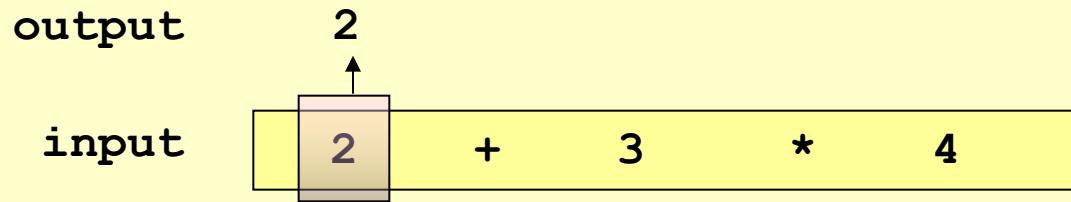
output	2	3	4
input	2	+	3
	-	4	



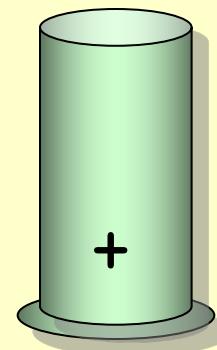
+ มา ก่อน  
แล ะ สำคัญ เท่า กับ -

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

# ความสำคัญของ operators



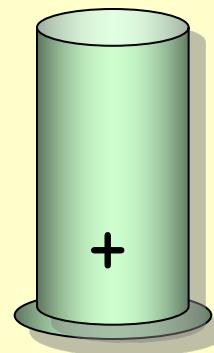
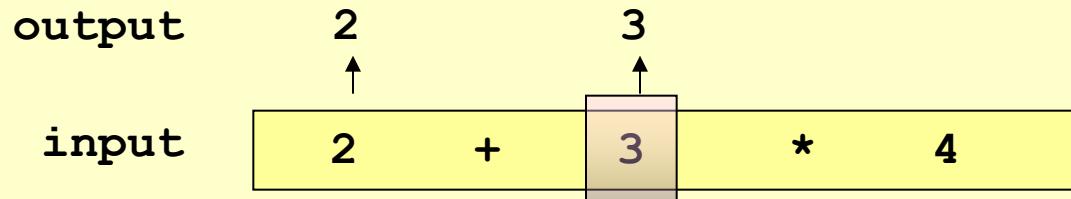
# ความสำคัญของ operators



20-5

10/11/66 257

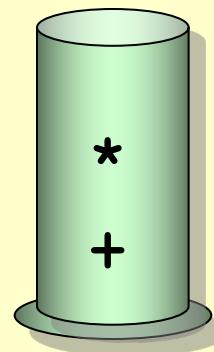
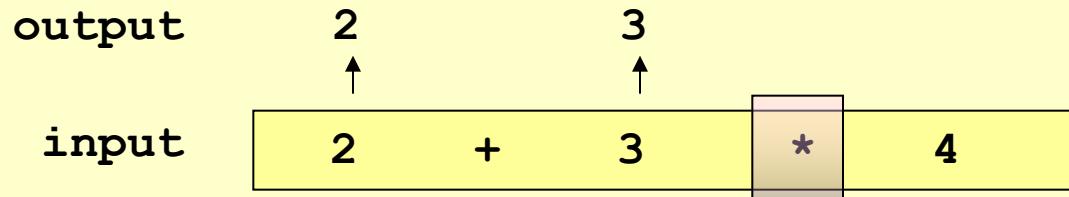
# ความสำคัญของ operators



20-7

10/11/66 259

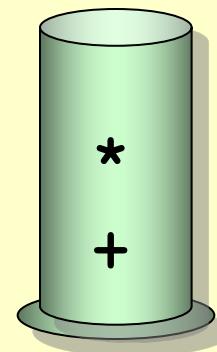
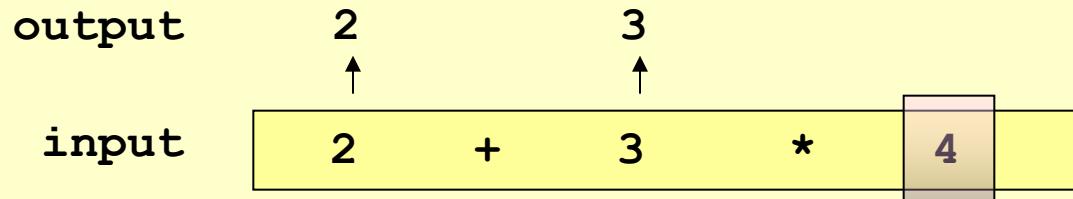
# ความสำคัญของ operators



+ มา ก่อน แต่ \* สำคัญกว่า +

push ทับ ถ้า operator ใหม่สำคัญกว่า operator บนกองข้อน

# ความสำคัญของ operators

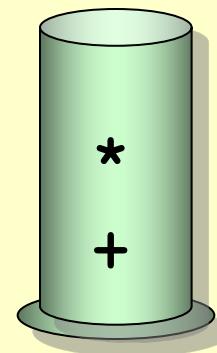


+ มา ก่อน แต่ \* สำคัญกว่า +

push ทับ ถ้า operator ใหม่สำคัญกว่า operator บนกองข้อน

# ความสำคัญของ operators

output	2	3	4
input	2	+	3



+ มา ก่อน แต่ \* สำคัญกว่า +

push ทับ ถ้า operator ใหม่สำคัญกว่า operator บนกองข้อน

# 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;  
}
```

หากค่าความสำคัญของ operator ต่ำให้มีที่พับ

pop ออก ถ้า operator บนกองช้อน  
สำคัญไม่น้อยกว่า operator ใหม่

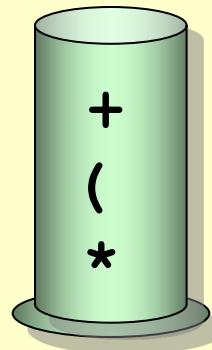
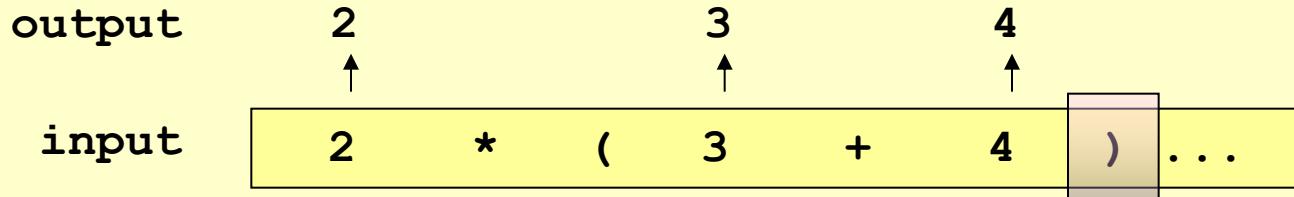
# การหาความสำคัญของ operators

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

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

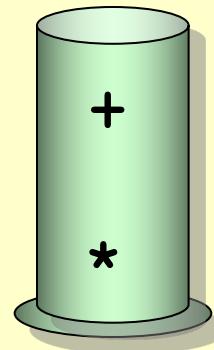
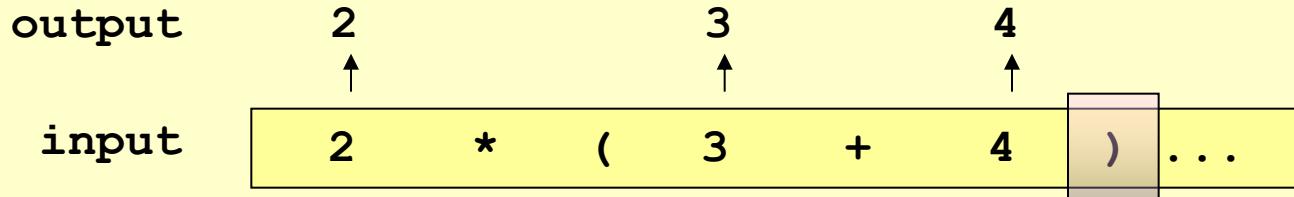
```
int p = priority[token];  
while( !s.empty() && priority[s.top()] >= p ) {  
    postfix += s.top(); s.pop();  
}
```

# ชั้บช้อนขึ้นเมื่อมีวงเล็บใน infix



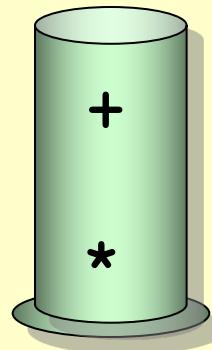
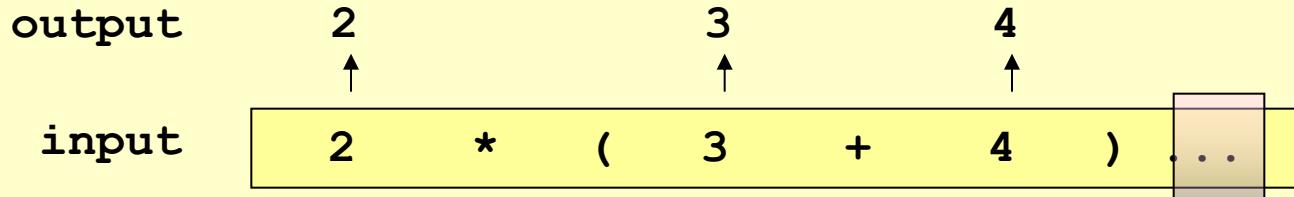
- ภายในวงเล็บเสมือนเป็นนิพจน์ย่อย
- พบวงเล็บเปิด push เสมอ (มีความสำคัญมาก)
- วงเล็บเปิดในกองช้อน มีความสำคัญน้อยมาก
- พบวงเล็บปิด ลุย pop จนกว่าจะพบวงเล็บเปิด

# ชั้บช้อนขึ้นเมื่อมีวงเล็บใน infix



- ภายในวงเล็บเสมือนเป็นนิพจน์ย่อย
- พบวงเล็บเปิด push เสมอ (มีความสำคัญมาก)
- วงเล็บเปิดในกองช้อน มีความสำคัญน้อยมาก
- พบวงเล็บปิด ลุย pop จนกว่าจะพบวงเล็บเปิด

# ชั้บช้อนขึ้นเมื่อมีวงเล็บใน infix



- ภายในวงเล็บเสมือนเป็นนิพจน์ย่อย
- พบวงเล็บเปิด push เสมอ (มีความสำคัญมาก)
- วงเล็บเปิดในกองช้อน มีความสำคัญน้อยมาก
- พบวงเล็บปิด ลุยก pop จนกว่าจะพบวงเล็บเปิด

# ความสำคัญของ operators กรณีมีวงเล็บ

- ขณะพิจารณาวงเล็บเปิด
  - มีความสำคัญมาก, จะ 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

	+	-	*	/	<sup>^</sup>	(	)
ขนาดที่พบในนิพจน์ (อยู่นอกวงกลม)	3	3	5	5	7	9	1
ขนาดอยู่ในวงกลม	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 ที่ทำจากซ้ายไปขวา

	+	-	*	/	<sup>^</sup>	( )
ขณะที่พบในนิพจน์ (อยู่นอกกองข้อมูล)						
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 ที่ทำการซ้ายไปขวา

	+	-	*	/	<sup>^</sup>	( )
ขณะที่พบในนิพจน์ (อยู่นอกกองข้อมูล)	2	2	4	4	6	9
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

$$2^{3^{4^5}} = 2^{\left(3^{\left(4^5\right)}\right)}$$

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

$$2^{3^{4^5}} = 2^{\left(3^{\left(4^5\right)}\right)}$$

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

$$2^{3^{4^5}} = 2^{\left(3^{\left(4^5\right)}\right)}$$

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

$2^{3^{4^5}} = 2^{\left(3^{\left(4^5\right)}\right)}$	+	-	*	/	$\wedge$	(	)
	2	2	4	4	6	9	1
ขณะอยู่ในกองข้อมูล	3	3	5	5	7	0	

2  $\wedge$  3  $\wedge$  4  $\wedge$  5

2 3 4 5  $\wedge$   $\wedge$   $\wedge$

```
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

$2^{3^{4^5}} = 2^{\left(3^{\left(4^5\right)}\right)}$	+      -      *      /      ^      (      )
	2      2      4      4      8      9      1
ขณะอยู่ในกองข้อมูล	3      3      5      5      7      0

2    ^    3    ^    4    ^    5

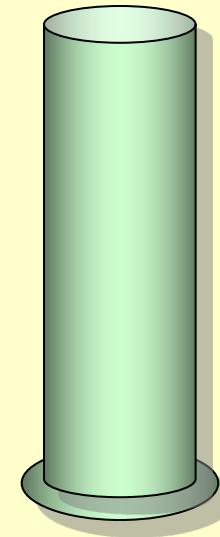
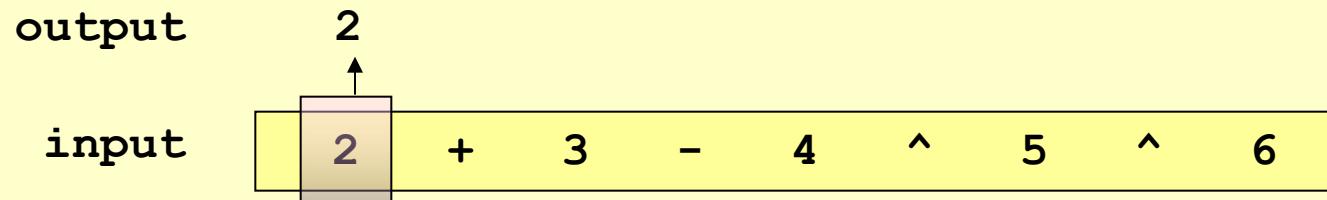
2    3    4    5    ^    ^    ^

```

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

```

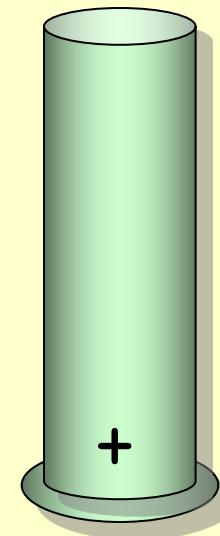
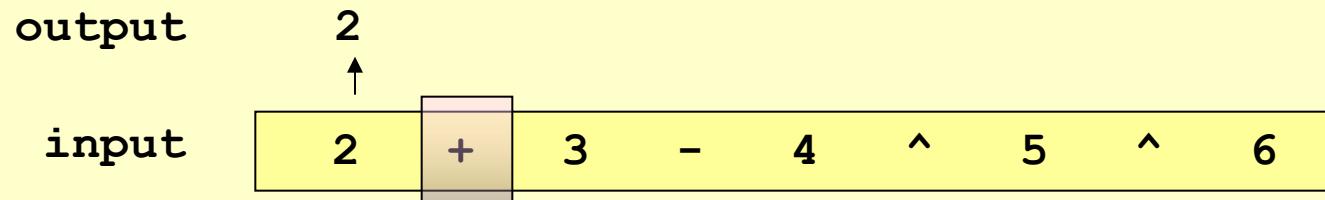
# ตัวอย่าง



28-3

10/11/66 314

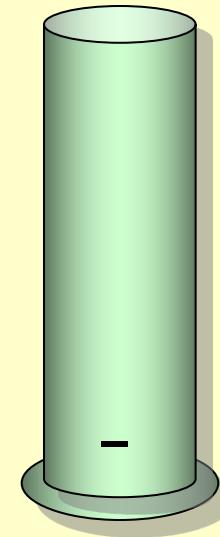
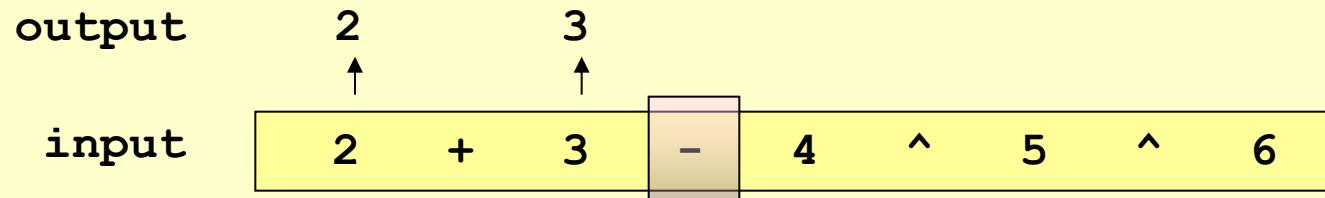
# ตัวอย่าง



28-5

10/11/66 316

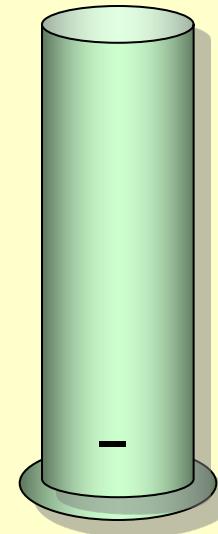
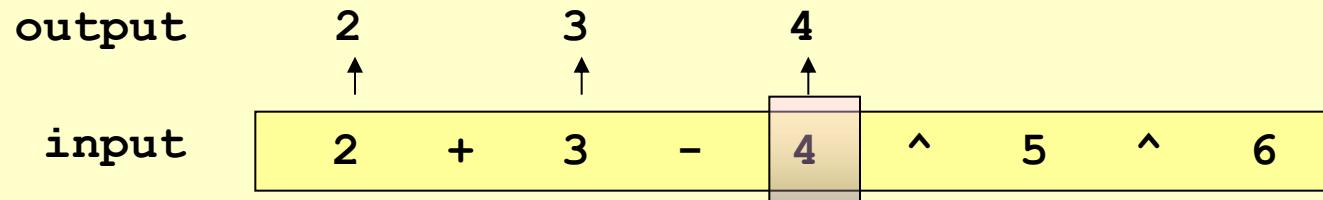
# ตัวอย่าง



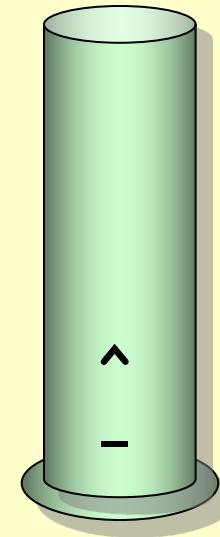
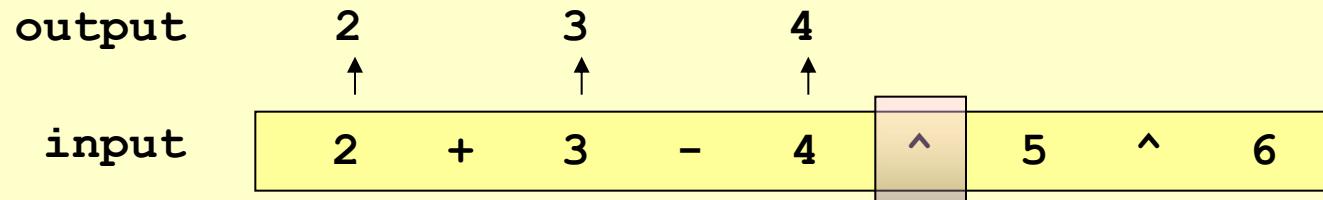
28-10

10/11/66 321

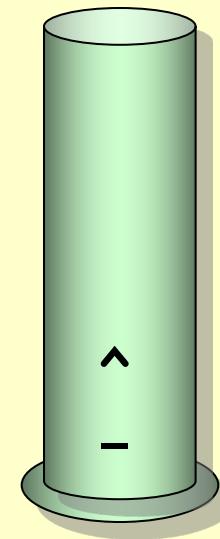
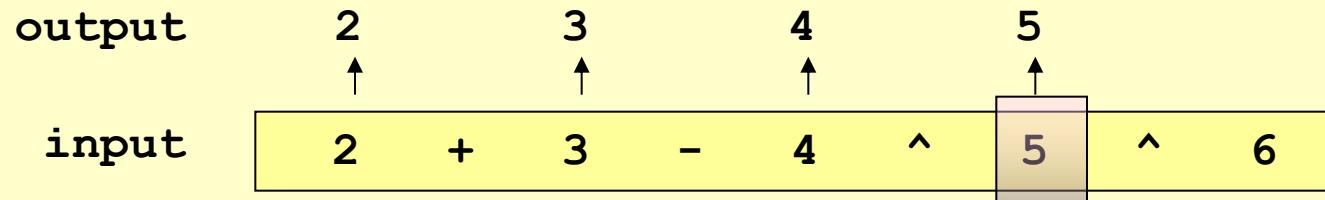
# ตัวอย่าง



# ตัวอย่าง

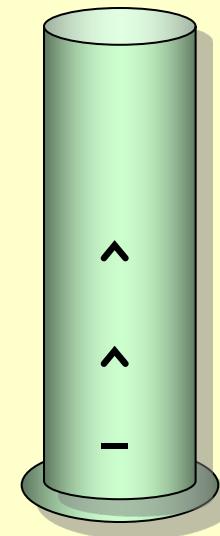


# ตัวอย่าง



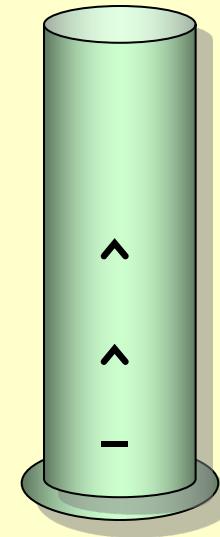
# ตัวอย่าง

output	2	3	4	5				
input	2	+	3	-	4	^	5	6



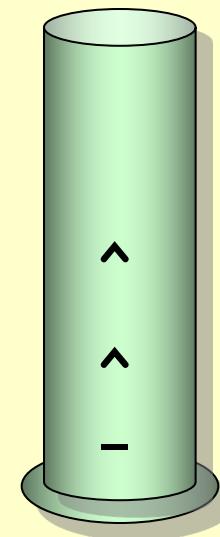
# ตัวอย่าง

output	2	3	4	5	6				
input	2	+	3	-	4	<sup>^</sup>	5	<sup>^</sup>	6



# ตัวอย่าง

output	2	3	4	5	6				
input	2	+	3	-	4	^	5	^	6



# สรุป

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

# แบบฝึกหัด by อ.โต

- เขียนฟังก์ชัน `int getKthData(stack<int> s, int k)`
  - รีเทิร์นデータตัวที่  $k$  ใน stack  $s$  ออกมาน (ให้ถือว่าデータบนสุด เป็นデータลำดับที่ 0)
  - ถ้ารีเทิร์นไม่ได้ ให้ throw exception
  - เสร็จแล้ว  $s$  ต้องไม่เปลี่ยนหรือมีอะไรไป
- เขียนฟังก์ชัน `stack<int> reverse(stack<int> s)`
  - เรียงลำดับของในสแตกให้กลับกับสแตกต้นฉบับให้หมด
- เขียนฟังก์ชัน `double calculate(vector<string> v)`
  - ทำการคำนวณหาค่าตอบของตัวเวคเตอร์ที่เก็บตัวเลข postfix อยู่แล้ว โดยให้ทำกับ  $+,-,*,/$  เท่านั้น โดยใช้สแตกในการแก้ปัญหา