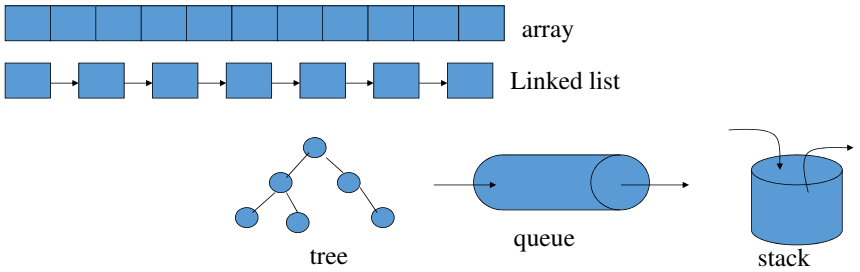


05506006 โครงสร้างข้อมูล

Lecture 4: Array and Stack

ดร.รุ่งรัตน์ เวียงศรีพนาวัลย์

เค้าโครงการบรรยาย

งานประยุกต์

- Array
  - อะเรย์คืออะไร
  - Array in Java
  - การใช้ array ใน Parameter
  - String[] in main() method
  - การ return ค่า เป็นชนิดข้อมูลแบบ array
  - สิ่งที่มักผิดพลาดบ่อยกับการสร้างโครงสร้างข้อมูลแบบอะเรย์
  - ข้อเสียของ อะเรย์

- สแตกคืออะไร
  - การทำงานของสแตก
- ฟังก์ชันในสแตก (Stack Operations)
- การสร้างสแตกด้วยอะเรย์
- การนำสแตกไปใช้งาน
  - การใช้ Stack เปลี่ยนจาก นิพจน์ infix เป็น postfix
  - การใช้ Stack คำนวณนิพจน์แบบ postfix
- การเขียนโปรแกรมย่อย สำหรับใช้งาน

Stack คือ อะไร

- Stack คือ โครงสร้างข้อมูลที่ประกอบด้วยสมาชิกที่เป็นข้อมูลชนิดเดียวกันที่ถูกเก็บเรียงต่อกันเป็นแถว
- การที่จะเพิ่มข้อมูลหรือสมาชิกใหม่เข้าไปใน Stack หรือ การลบสมาชิกออกจาก Stack
  - จะต้องทำจากด้านบนของ Stack (Top of a Stack) เท่านั้น
- ข้อมูลที่เก็บใน Stack จะถูกเก็บเรียงตามลำดับที่ถูกนำเข้ามาใน Stack
- ข้อมูลที่ถูกนำเข้ามาสุดท้ายจะอยู่บนสุดของ Stack และจะต้องถูกเอาออกจาก Stack เป็นตัวแรก เรียกว่า LIFO (Last In First Out)

การทำงานของ Stack

- Stack เป็นโครงสร้างข้อมูลแบบ Dynamic
  - เนื่องจาก (ขนาด) Stack จะมีการเปลี่ยนแปลงทุกครั้งมีการเพิ่มข้อมูลหรือสมาชิกใหม่เข้าไป หรือ ลบสมาชิกออกจาก Stack
- ดังนั้นการทำงานกับ Stack จะมี 2 การทำงานหลักคือ
  - 1. การเพิ่มหรือนำข้อมูลใหม่เข้าไปเก็บใน Stack เรียกว่า Push โดยข้อมูลใหม่นี้จะไปวางที่ด้านบนของ Stack
  - 2. การลบสมาชิกออกจาก Stack เรียกว่า Pop

Stack

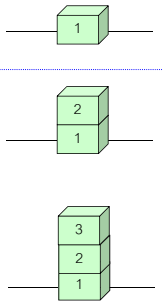
empty() {empty}

Push(block1, Stack)

{Push block1 onto a Stack}

Push(block2, Stack)

Push(block3, Stack)



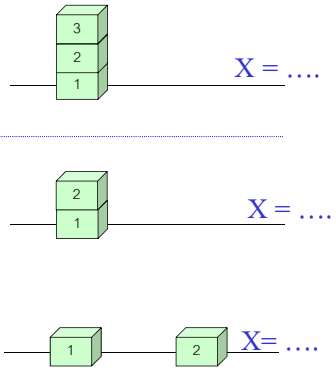
Stack

pop(Stack)

{Pop the top block of the Stack and put it into X}

Pop(Stack)

Push(X, Stack)

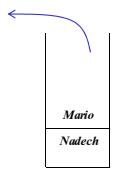


Stack

Push("Nadech") Push("Mario")

Pop() "Mario"

ดึงค่า "Mario" ออกมา เก็บไว้ใน X



Stack Abstract Data Type

- ประกอบไปด้วย 5 operations

1. การสร้าง Stack
2. การเพิ่มข้อมูลลงไปใน Stack
3. การลบข้อมูลออกจาก Stack
4. การตรวจสอบว่า Stack ว่างหรือไม่
   (จำเป็นต้องมีเนื่องจากถ้า Stack ว่างจะทำการลบข้อมูลไม่ได้)
5. การตรวจสอบว่า Stack เต็มหรือไม่

(เนื่องจากถ้า Stack เต็มจะทำการเพิ่มข้อมูลไม่ได้) - ข้อนี้ถ้าโครงสร้างข้อมูลที่นำมาใช้มีขนาดไม่จำกัดเช่น ลิ้งค์ลิสต์ ไม่จำเป็นต้องมี

ฟังก์ชันใน Stack

Operation	คำอธิบาย	ผลลัพธ์
initialize(Stack)	เริ่มสร้าง Empty stack	Stack ว่าง
push(NewElement, Stack)	เพิ่มข้อมูลใหม่เข้าไปใน Stack	
pop(Stack)	ดึงข้อมูลที่อยู่บนสุดออกจาก Stack	
isFull (Stack)	ตรวจสอบว่า Stack มีข้อมูลเก็บอยู่เต็ม Stack ถ้าหาก Stack เต็ม จะคืนค่า True (และไม่สามารถ push ได้) (เฉพาะในกรณีใช้โครงสร้างแบบอะเรย์ ลิ้งค์ลิสต์ไม่จำเป็นต้องมี)	
isEmpty(Stack)	ตรวจสอบว่า Stack ว่างหรือไม่ ถ้าหาก Stack ว่าง จะคืนค่า True (และไม่สามารถ pop ได้)	
peek()	! บอกว่าค่าที่อยู่บนสุดของ Stack คืออะไร	

การนำ stack ไปใช้งาน
1. การเปลี่ยนนิพจน์ Infix เป็น Postfix

- โครงสร้างข้อมูลแบบ Stack เป็นเครื่องมือที่ดีและมักจะถูกนำไปใช้เขียนโปรแกรมในการแก้โจทย์การคำนวณที่มีเครื่องหมายการคำนวณหลายชนิดในข้อเดียวกัน
- นิพจน์ทางคณิตศาสตร์ มี 3 รูปแบบคือ

1. Infix Notation
2. Prefix Notation
3. Postfix Notation

Infix	**: operand1**	**operator**	**operand2**	เช่น A + B
Prefix	**: operator**	**operand1**	**operand2**	เช่น +AB
Postfix	**: operand1**	**operand2**	**operator**	เช่น AB+

คำที่ใช้เรียก

นิพจน์ (**Expression**): **a = b + c * d**

Operands: **a, b, c, d**

Operators: **=, +, –, *, /, %**

การนำ stack ไปใช้งาน:
1. การเปลี่ยนนิพจน์ Infix เป็น Postfix: นิพจน์ทางคณิตศาสตร์

- โดยทั่วไปนิพจน์ทางคณิตศาสตร์ (Arithmetic expressions) จะถูกเขียนในแบบ Infix Notation
- แต่ในการทำงานของเครื่องคิดเลขหรือ Compiler ของภาษาคอมพิวเตอร์ ส่วนใหญ่
  - ต้องแปลงภาษาคอมพิวเตอร์ที่เขียนใน Source Program ไปเป็นคำสั่งในภาษาเครื่อง (Machine Language)
  - ก่อนที่จะประมวลผลนั้น จะแปลง Infix expression ให้อยู่ในรูปของ Postfix expression ก่อน
  - เพราะคำนวณง่ายกว่าในเชิงเทคนิค อีกทั้งไม่มีวงเล็บอยู่ใน Postfix expression
-

การนำ stack ไปใช้งาน
การเขียนนิพจน์ Infix

- การเขียนนิพจน์ทางคณิตศาสตร์ ในแบบ Infix Notation จำเป็นต้องมีวงเล็บ ( ) มาช่วยแบ่งแยกการคำนวณหรือเพื่อให้คำนวณได้ถูกต้อง

Precedence rules:

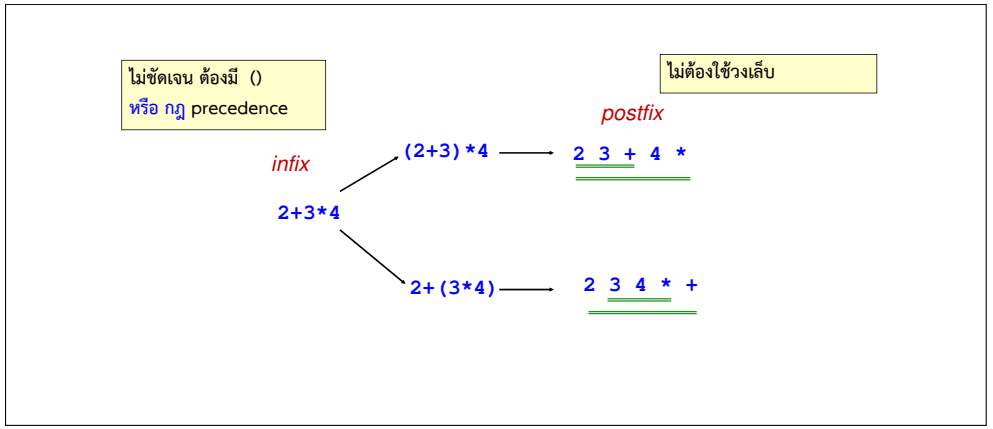
- Operator บาง operator มี priorities สูงกว่า operator อื่น เช่น : * และ / มี precedence สูงกว่า + และ –.
- สำหรับ operators ที่มี precedence เท่ากัน (เช่น * และ /) จะ process จากซ้ายเป็นขวา

- ตัวอย่างเช่น 2 * (3 + 4) = ?

2 * 3 + 4 = ?

แปลง 2 * (3 + 4) ให้อยู่ในรูปของ Postfix Notation ???

การนำ stack ไปใช้งาน:
1. การเปลี่ยนนิพจน์ Infix เป็น Postfix: Infix Vs Postfix



การนำ stack ไปใช้งาน: 1. การเปลี่ยนนิพจน์ Infix เป็น Postfix:
ขั้นตอนในการเปลี่ยน Infix เป็น Postfix โดยใช้สแตก

1. Scan infix expression จากซ้ายไปขวา
2. ถ้าพบ operand เพิ่ม operand นี้ลงไปใน postfix expression.
3. ถ้าพบ "(" , push ลงไปใน stack.
4. ถ้าพบ ")"
   - ทำ 2 step นี้ซ้ำ จนกว่าจะเจอ "("
     1. pop ค่าออกมาจาก stack
     2. นำค่าที่ pop ออกมาใส่ลงไปใน postfix expression
   - นำค่า "(". ออกไป

5. ถ้าพบ operator
   - ถ้า operator มีค่า precedence ต่ำกว่าหรือเท่ากับ เครื่องหมายที่อยู่บนสุดใน stack
     1. pop เครื่องหมายออกจาก stack
     2. ใส่เครื่องหมายที่ pop ออกมาลงใน postfix expression
   - ทำซ้ำขั้นตอนที่ 1 และ 2 จนกว่า operator มีค่า precedence สูงกว่า
     3. push operator ลง stack
6. เมื่อทำครบทุกตัวแล้ว ให้ทำ
   - ทำ 2 step นี้ซ้ำจนกว่า stack จะว่าง
     1. Pop เครื่องหมายออกจาก Stack
     2. ใส่ เครื่องหมายนั้นลงใน postfix expression

การนำ stack ไปใช้งาน: 1. การเปลี่ยนนิพจน์ Infix เป็น Postfix:
ตัวอย่าง การเปลี่ยน Infix เป็น Postfix โดยใช้สแตก (1)

Example: a – (b + c * d) / e

		Stack (ล่าง ขึ้น บน)	PostFix Expression
1	a		a
2	-	-	a
3	(	- (	a
4	b	- (	a b

1. a เป็น operand เขียนใน PostFix Expression ได้เลย

2. - เป็น operator Stack ว่าง push ลงได้เลย

3. (push ลงได้เลย

การนำ stack ไปใช้งาน: 1. การเปลี่ยนนิพจน์ Infix เป็น Postfix:
ตัวอย่าง การเปลี่ยน Infix เป็น Postfix โดยใช้สแตก (2)

Example: a – (b + c * d) / e

		Stack (ล่าง ขึ้น บน)	PostFix Expression
1	a	a	a
2	-	-	a
3	(	- (	a
4	b	- (	a b
5	+	- (+	a b
6	c	- (+	a b c
7	*	- (+ *	a b c

5. เปรียบเทียบ + กับ (+ มี precedence สูงกว่า push +

6. c เป็น operand ใส่ลงใน postFix

7. เปรียบเทียบ * กับ + ... มี precedence ..... สูงกว่า ดังนั้น .....

การนำ stack ไปใช้งาน: 1. การเปลี่ยนนิพจน์ Infix เป็น Postfix: ตัวอย่าง การเปลี่ยน Infix เป็น Postfix โดยใช้สแตก (3)

Example: a – (b + c * d) / e

		Stack (ล่าง ขึ้น บน)	PostFix Expression
1	a	a	a
2	-	-	a
3	(	- (	a
4	b	- (	a b
5	+	- (+	a b
6	c	- (+	a b c
7	*	- (+ *	a b c
8	d	- (+ *	a b c d

8. d เป็น operand ใส่ d ลง PostFix Expression

การนำ stack ไปใช้งาน: 1. การเปลี่ยนนิพจน์ Infix เป็น Postfix: ตัวอย่าง การเปลี่ยน Infix เป็น Postfix โดยใช้สแตก (4)

Example: a – (b + c * d) / e

		Stack (ล่าง ขึ้น บน)	PostFix Expression
1	a	a	a
2	-	-	a
3	(	- (	
4	b	- (	
5	+	- (+	a b
6	c	- (+	a b c
7	*	- (+ *	a b c
8	d	- (+ *	a b c d
9.1	)	- (+	a b c d *
9.2		- (	a b c d * +
9.3		-	a b c d * +

9.) => pop * ออกไปใส่ใน postfix expression =>
pop ออกไปใส่ใน postfix expression => pop (ออก ทิ้ง)

9.3 pop (ออก ทิ้ง

การนำ stack ไปใช้งาน: 1. การเปลี่ยนนิพจน์ Infix เป็น Postfix: ตัวอย่าง การเปลี่ยน Infix เป็น Postfix โดยใช้สแตก (5)

Example: a – (b + c * d) / e

		Stack (ล่าง ขึ้น บน)	PostFix Expression
1	a	a	a
2	-	-	
3	(	- (	
4	b	- (	a b
5	+	- (+	
6	c	- (+	
7	*	- (+ *	a b c
8	d	- (+ *	a b c d
9	)	-	a b c d * +
10	/	- /	a b c d * +
11	e	- /	a b c d * + e
12.1		-	a b c d * + e /
			a b c d * + e / -

10. เปรียบเทียบ / กับ – เนื่องจาก / preference มากกว่า ดังนั้น push ลงไปใน Stack

12. ทำการ pop จนกว่า stack จะว่าง
pop / => pop -

การนำ stack ไปใช้งาน: 1. การเปลี่ยนนิพจน์ Infix เป็น Postfix: ตัวอย่าง การเปลี่ยน Infix เป็น Postfix โดยใช้สแตก (6)

Example: a – (b + c * d) / e

		Stack (ล่าง ขึ้น บน)	PostFix Expression
1	a	a	a
2	-	-	
3	(	- (	
4	b	- (	a b
5	+	- (	
6	c	- (	
7	*	- (+ *	a b c
8	d	- (+ *	a b c d
9	)	-	a b c d * +
10	/	- /	a b c d * +
11	e	- /	a b c d * + e
12.1		-	a b c d * + e /
			a b c d * + e / -

10. เปรียบเทียบ / กับ – เนื่องจาก / preference มากกว่า ดังนั้น push ลงไปใน Stack

12. ทำการ pop จนกว่า stack จะว่าง
pop / => pop -

การนำ stack ไปใช้งาน:
1. การเปลี่ยนนิพจน์ Infix เป็น Postfix: แบบฝึกหัด

- 1. จงแปลงนิพจน์ต่อไปนี้เป็น Postfix Expression โดยใช้ stack
  - 8 + (7 + 6 / 3 – 4) / 2 * 5
  - (X – (Y + Z * A)) / (B+A)

การนำ stack ไปใช้งาน:
2. การใช้ stack คำนวณนิพจน์แบบ postfix

- get the next token
- ถ้าเป็น operand
  - Push ลง stack
- ถ้าไม่ใช่แต่ เป็น operator
  1. pop ค่าออกจาก stack มาเป็น operand ทางขวา
  2. pop ค่าถัดมาเป็น operand ทางซ้าย
  3. ทำการคำนวณ
  4. push ผลลัพธ์ลงสู่ stack
- เมื่อทำจนครบ ผลลัพธ์คือค่าที่อยู่บนสุดของ Stack (top of stack)

การนำ stack ไปใช้งาน:
2. การใช้ stack คำนวณนิพจน์แบบ postfix : ตัวอย่าง (1)

10 2 – 3 – 3 2 + /

ข้อมูลนำเข้า	Operand1	Operand 2	ผลลัพธ์	Stack
10				10
2				10, 2
-	10	2	(10-2) =8	8
3				8, 3
-	8	3	(8-3)=5	5
3				5, 3
2				5, 3, 2
+	3	2	(3-2)=5	5, 5
/	5	5	(5/5)=1	1

การนำ stack ไปใช้งาน:
2. การใช้ stack คำนวณนิพจน์แบบ postfix : ตัวอย่าง (2)

a b c d * + e / -

ข้อมูลนำเข้า	Operand1	Operand 2	ผลลัพธ์	Stack
a				a
b				a, b
c				a, b, c
d				a, b, c, d
*	c	d	(c*d)	a, b, (c*d)
+	b	(c*d)	(b+(c*d))	a, (b+(c*d))
e				a, (b+(c*d)), e
/	(b+(c*d))	e	((b+(c*d)) /e)	a, ((b+(c*d)) /e)
-	a	((b+(c*d)) /e)	a- ((b+(c*d)) /e)	a- ((b+(c*d)) /e)

การนำ stack ไปใช้งาน:

2. การใช้ stack คำนวณนิพจน์แบบ postfix :แบบฝึกหัด

4 5 6 + - 1 * 6 3 / -

ข้อมูลนำเข้า	Operand1	Operand 2	ผลลัพธ์	Stack

การสร้างสแตกด้วยอะเรย์

- ภาษาโปรแกรมไม่มีโครงสร้างข้อมูลแบบสแตกไว้ให้
- จึงมีการประยุกต์นำอะเรย์มาใช้ในการสร้างสแตก
- แต่การที่อะเรย์และสแตกเป็นโครงสร้างข้อมูลต่างชนิดกัน
  - การเพิ่มหรือลบข้อมูลในอะเรย์ทำที่ตำแหน่งใดในระยะเวลาหนึ่งๆ ได้
  - การเพิ่ม (Push) หรือ การลบ (Pop) ข้อมูลออกจากสะแตก จะทำได้ที่ตำแหน่งบนสุดเท่านั้น
- ในการนำอะเรย์มาประยุกต์ใช้ในการสร้างสแตก
  - จึงต้องมีตัวแปรสำหรับบ่งชี้สมาชิกตัวบนสุดใน Stack (Top)

การเขียนโปรแกรมย่อย สำหรับใช้งาน Stack: Initialization

การตั้งค่า Initialization

- 1. สร้าง Array ที่มีขนาดเท่ากับ ขนาดของ Stack ที่ผู้ใช้ต้องการ
  - ให้ขนาดที่ต้องการ กำหนดผ่านตัวแปรชื่อ Maxsize
- 2. สร้างตัวแปรชื่อ top ให้ชี้ไปที่ตำแหน่งบนสุดของ Stack
  - เนื่องจากตอนสร้าง array ยังไม่มีสมาชิก ดังนั้น ให้ top ชี้ไปที่ -1
- 3. สำหรับชนิดข้อมูลที่เก็บใน Stack นั้น ขึ้นอยู่กับว่าเราต้องการเก็บข้อมูลชนิดอะไร ให้สร้าง array ของ ข้อมูลชนิดนั้น ในที่นี้ ให้เป็น int

กรณีไม่ทำเป็น **object**

```
int maxSize;
int[] a;
a = new int[maxSize];
int top = -1;
```

กรณีเป็น **object**

```
int maxSize;
int[] a;
int top;
public MyStack(int s)
{
      maxSize = s;
      a = new int[maxSize];
      top = -1;
}
```


การเขียนโปรแกรมย่อย สำหรับใช้งาน Stack: IsEmpty()

Method IsEmpty() เพื่อทดสอบว่า Stack ว่างหรือไม่

- ถ้า Stack ว่าง top = -1

```
boolean IsEmpty()
{
   return(top==-1);
}
```

```
boolean IsEmpty()
{
   if (top==-1) return(-1);
 else return(1);
}
```

การเขียนโปรแกรมย่อย สำหรับใช้งาน Stack: IsFull()

- โดยทฤษฎีแล้ว โครงสร้างข้อมูลชนิด Stack จะไม่มีเหตุการณ์ที่ Stack เต็ม แต่เนื่องจากใช้โครงสร้าง Array มาสร้าง Stack และ Array มีขนาดจำกัด จึงสามารถเกิดเหตุการณ์ที่ Stack เต็มได้
- ดังนั้นทุกครั้งก่อนที่จะนำสมาชิกใหม่ไปเก็บในStack (Push) จึงควรตรวจสอบก่อนว่า Array (Stack) เต็มหรือยัง
- Method IsFull () เพื่อทดสอบว่า Stack เต็มหรือไม่
  - ถ้า Stack เต็ม top = MaxSize-1

```
boolean IsFull()
{
   return(top==MaxSize-1);
}
```


การเขียนโปรแกรมย่อย สำหรับใช้งาน Stack: push

- การนำสมาชิกใหม่เพิ่มเข้าไปใน Stack
  - ดังนั้น Method นี้ ให้ชื่อว่า Push method นี้ไม่มีการ return ค่า
  - ต้องมีการบอกว่า สมาชิกใหม่ที่เพิ่มเข้าไปคืออะไร
    - จึงต้องมี Parameter ซึ่งมีข้อมูลเป็นชนิดเดียวกับข้อมูลของ Stack
    - push(int value)
  - ทุกครั้งที่ push ค่า top ต้องเพิ่มขึ้นหนึ่ง top = top+1;

```
void  push(int value)
{
   a[++top] = value;
}
```

ถ้า Stack เต็มจะไม่สามารถเพิ่มเข้าไปใน Stack ได้ จึงต้องทำการตรวจสอบก่อนว่า Stack นี้เต็มหรือไม่ มิฉะนั้นโปรแกรมจะเกิด Error

```
void  push(int value)
{
   if (!isFull())
      a[++top] = value;
   else
      System.out.println(" Stack is Full. Cannot push");
}
```


การเขียนโปรแกรมย่อย สำหรับใช้งาน Stack: pop

- การนำสมาชิกออกจาก Stack
  - Method นี้ ให้ชื่อว่า pop method มีการ return ค่า เป็น ค่าของข้อมูลของ Stack ที่อยู่บนสุด
    - int pop()
  - ทุกครั้งที่ pop ข้อมูลจะถูกนำออกดังนั้น ตำแหน่ง top จะลดลงหนึ่ง
    - top=top-1;

```
int pop()
{
   return(a[top--]);
}
```

ถ้า Stack ว่างจะไม่สามารถนำข้อมูลออกมาได้ จึงต้องทำการตรวจสอบก่อนว่า Stack นี้ว่างหรือไม่ มิฉะนั้นโปรแกรมจะเกิด Error

```
int  pop()
{
   if (!isEmpty())
      return(a[top--]);
   else
   {  System.out.println(" Stack  is Empty.");
      return(-1);
   }
}
```


การเขียนโปรแกรมย่อย สำหรับใช้งาน Stack: peek()

- จะให้ค่าสมาชิกที่อยู่บนสุด
- มีการ return ค่า เป็น ค่าของข้อมูลของ Stack ที่อยู่บนสุด
- แต่ไม่มีการนำค่าออกจาก Stack
  - int peek()

```
int peek()
{
   return(a[top]);
}
```

ถ้า Stack ว่างจะไม่สามารถนำข้อมูลออกมาได้ จึงต้องทำการตรวจสอบก่อนว่า Stack นี้ว่างหรือไม่ มิฉะนั้นโปรแกรมจะเกิด Error

```
int  peek ()
   if (!isEmpty())
      return(a[top]);
   else
   {
      System.out.println(" Stack is Empty.");
      return(-1);
   }
}
```

การเขียนโปรแกรมย่อย สำหรับใช้งาน Stack: การพิมพ์ค่าของข้อมูลใน Stack

หนึ่งวิธีคือ

- ให้ pop ทีละตัวออกมาพิมพ์
  - การที่จะรู้ได้ว่า pop ออกมาหมดหรือยังให้เช็คทุกครั้งก่อน pop ว่า Stack
  - ใช้ loop while

```
while (!IsEmpty())
  {System.out.print(pop()+ " " );
```

```
int value;
while (!IsEmpty())
  { value = pop();
    System.out.print(value+ " " );
  }
```

```
int peek()
{
   return(a[top]);
}
```

ถ้า Stack ว่างจะไม่สามารถนำข้อมูลออกมาได้ จึงต้องทำการตรวจสอบก่อนว่า Stack นี้ว่างหรือไม่ มิฉะนั้นโปรแกรมจะเกิด Error

ตัวอย่างการประกาศใช้งาน: กรณีไม่ใช้ object

```
import java.io.*;
class StackApp9
{//สร้างตัวแปร
static int top, maxSize, a[];
public static void main(String[] args)
{ /* Initializtion */
     top = -1; maxSize=4;
     a = new int[maxSize];
   //ใส่ค่าลงไปใน Stack
     push(20); push(40);
     push(60);push(80);
     while(!isEmpty())
 System.out.print(pop()+"");
}
```

```
public static void push(int value)
{          a[top]=value;}

public static int pop()
{ return(a[          ])

public static int peek()
{ return          }

public static boolean isEmpty()
{ if (          return true; }

public static boolean isFull()
{ if                return true;}

public static int size()
{ return(top+1)}
```


เป็น object ชนิด Stack ชื่อ class MyStack

```
public class MyStack {
   int maxSize;
   int[] stackArray;
   int top;

   public MyStack(int s)
   {
      maxSize = s;
      stackArray = new int[maxSize];
      top = -1;
   }

   void  push(int value)
   {
      if (!isFull())
         stackArray[++top] = value;
      else
         System.out.println("Stack is Full. Cannot push");
   }
```

```
int  pop()
{
   if (!isEmpty())
      return(stackArray[top--]);
   else
  { System.out.println("Stack is Empty.");
    return(-1);}
}

int peek()
{
    if (!isEmpty())
       return(stackArray[top]);
    else
    {   System.out.println("Stack is Empty.");
    return(-1);}
}

boolean isEmpty()
{
      return(top==-1);
}

boolean isFull()
{
      return(top==maxSize-1);
}
}
```


สร้าง class main มาเรียกใช้

```
public class TestStack {
  public static void main(String[] args) {
     MyStack theStack = new MyStack(10);
     theStack.push(10);
     theStack.push(20);
     theStack.push(30);
     theStack.push(40);
     theStack.push(50);
     while (!theStack.isEmpty()) {
        long value = theStack.pop();
        System.out.print(value);
        System.out.print(" ");
     }
     System.out.println("");
  }
}
```

อ้างอิง

- สไลด์ประกอบการสอนวิชา SE 311 Algorithms Design and Analysis โดย ผศ.ดร. สมศรี บัณฑิตวิไล