Stack

โครงสร้างข้อมูลและอัลกอริทึมเบื้องต้น 305214 /235012

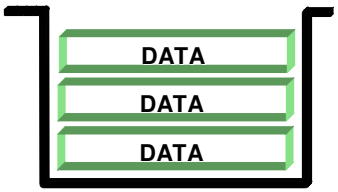
Topic

- Stack Abstract Data Type
- Stack Operation
- Array Stack
- Linked List Stack
- Applied Stack
  - Infix & Postfix Expression

Stack

กองซ้อน (stack) เป็นโครงสร้างข้อมูลที่มีคุณสมบัติ

- เป็นข้อมูลชนิดเดียวกัน
- เรียงต่อกันตามลำดับเชิงเส้น (linear)
- เข้าทีหลังออกก่อน Last in First out (LIFO)
- กระทำกับข้อมูลบนสุดเท่านั้น



Stack Operation

Basic Operation

- Push การใส่ข้อมูลลงใน stack (ข้อมูลอยู่บนสุด)
- Pop การนำข้อมูลออกจาก stack (ข้อมูลบนสุด)
- Top การแสดงข้อมูลที่อยู่บนสุด

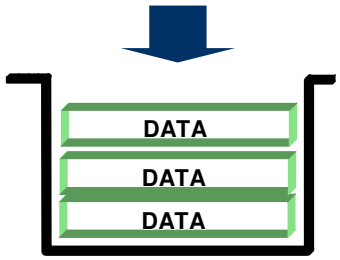
Other Operation

- Create สร้าง stack ว่าง
- IsEmpty ตรวจสอบว่าเป็น stack ว่าง
- Destroy / Clear เลิกใช้ทำลาย หรือ ทำให้เป็น stack ว่าง
- Count / IsFull อื่นๆ เช่นนับจำนวน และตรวจสอบว่าเต็ม

Stack Operation

Push – การใส่ข้อมูลลงใน stack

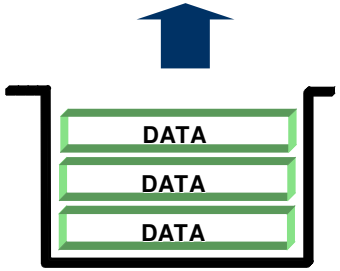
- ต้องใส่ข้อมูลด้านบนสุดของ stack เสมอ
- การ push ข้อมูลตัวต่อไปจะทับตัวก่อนหน้า
- ต้องตรวจสอบว่า stack เต็มหรือไม่



Stack Operation

Pop – นำข้อมูลออกจาก stack

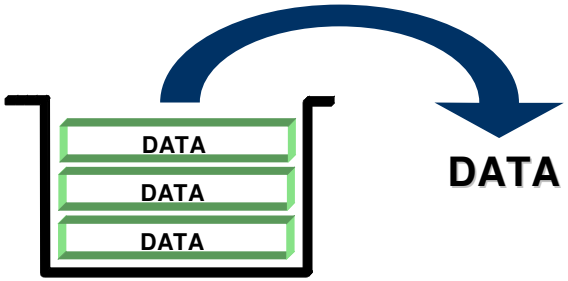
- ต้องนำข้อมูลด้านบนสุดของ stack ออกก่อนเสมอ
- การ pop จะทำการนำข้อมูลออกเรียกลำดับกันไปจากด้านบน
- ต้องตรวจสอบว่า stack ว่างหรือไม่



Stack Operation

Top – แสดงข้อมูลที่อยู่บนสุดของ stack

- แสดงค่าข้อมูลที่อยู่ด้านบนสุดของ stack
- ไม่ได้นำข้อมูลบนสุดออกมาจาก stack
- ต้องตรวจสอบว่า stack ว่างหรือไม่

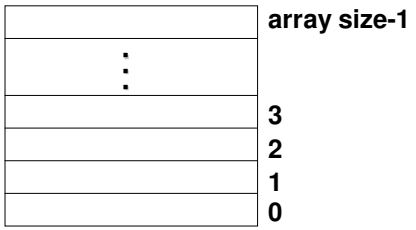


Array Stack

การสร้าง stack ด้วย array

- สร้าง array เพื่อเก็บข้อมูล stack
- มีการจองเนื้อที่สำหรับเก็บข้อมูลเท่าขนาดของ array
- เก็บค่า top ของ stack เป็น index ของ array

top stack top (integer)



Array Stack

Array Stack Operation

- push, pop, top อ้างอิงจากค่า top ของ stack
- IsEmpty ตรวจสอบจากค่า top (ให้เป็นค่าใดค่าหนึ่งเช่นให้เท่ากับ array size)
- IsFull ตรวจสอบ stack เต็มโดยดู top ของ stack เท่ากับ array size - 1 หรือไม่
- Count เป็นค่า top + 1

Array Stack

ข้อดี

- ใช้โครงสร้างข้อมูลที่ง่ายไม่ซับซ้อน

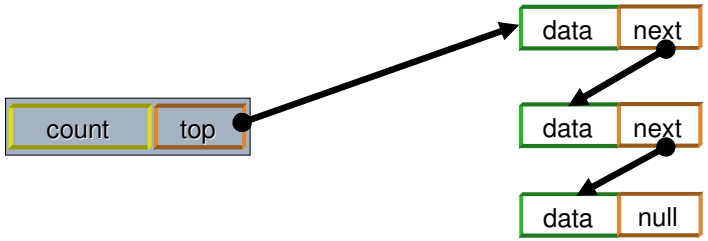
ข้อเสีย

- ต้องจองพื้นที่แน่นอนในหน่วยความจำ (static memory allocation)
- เกิดปัญหา stack เต็ม
- ไม่ได้ใช้พื้นที่หน่วยความจำอย่างมีประสิทธิภาพ

Linked List Stack

การสร้าง stack ด้วย Linked List

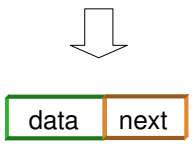
- สร้าง Linked List เพื่อเก็บข้อมูล stack
- มีการจองเนื้อที่สำหรับเก็บเป็นแบบ dynamic memory allocation
- เก็บค่า top ของ stack เป็น pointer ที่ชี้ไปยัง list
- เก็บค่า count เพื่อทำการนับจำนวน



Linked List Stack

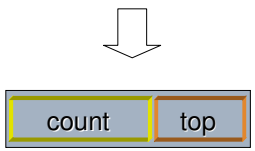
การสร้างโหนดข้อมูล

```
typedef struct node
{
int n_int;
node *next;
}node_list;
```



การสร้างหัวของ stack

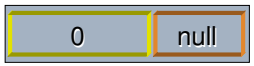
```
typedef struct stack
{
int count;
node_list *top;
}STACK;
```



Linked List Stack

Create

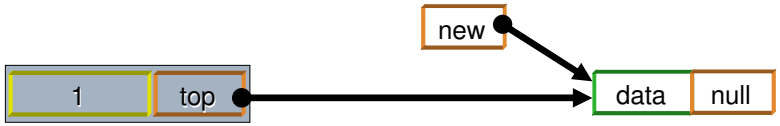
1. จองพื้นที่สำหรับหัวของ stack
2. ให้ count เป็น 0
3. ให้ top เป็น null



Linked List Stack

Push (กรณีที่ stack ว่าง)

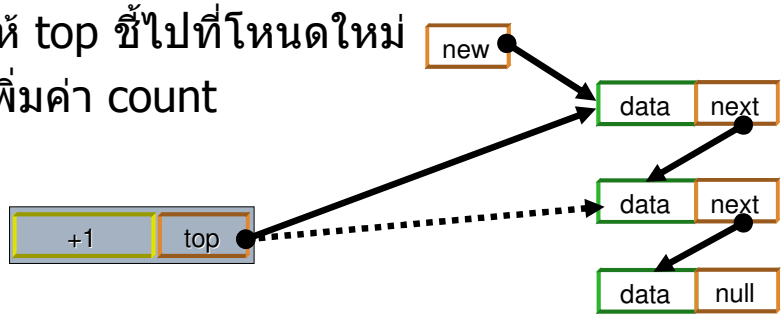
1. จองพื้นที่สำหรับโหนดข้อมูล
2. กำหนดข้อมูลของโหนดใหม่
3. ให้ next ของโหนดใหม่เป็น null
4. ให้ top ชี้ไปที่โหนดใหม่
5. ให้ count เป็น 1



Linked List Stack

Push (กรณีที่มีข้อมูลใน stack)

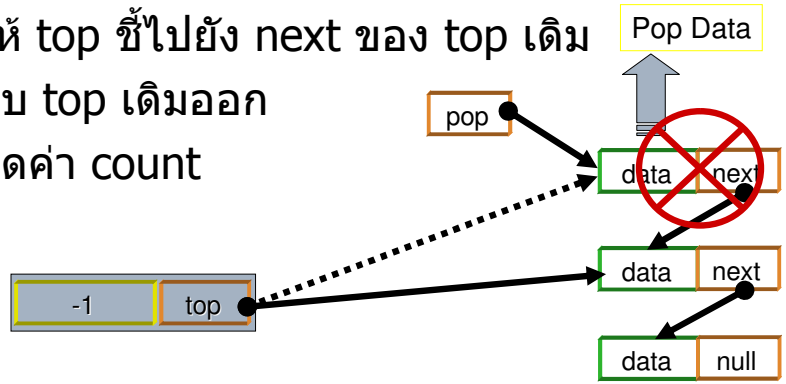
1. จองพื้นที่สำหรับโหนดข้อมูล
2. กำหนดข้อมูลของโหนดใหม่
3. ให้ next ของโหนดใหม่ชี้ไปที่ top เดิม
4. ให้ top ชี้ไปที่โหนดใหม่
5. เพิ่มค่า count



Linked List Stack

Pop

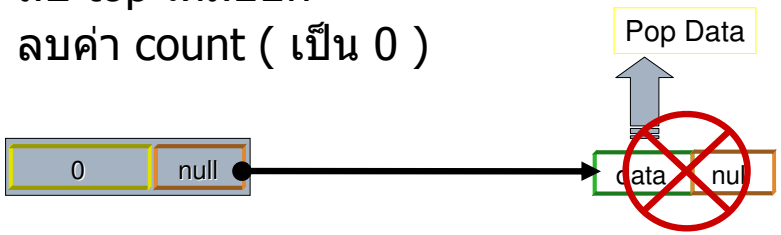
1. ตรวจสอบว่าไม่เป็น stack ว่าง
2. คืนค่าข้อมูลของ node top
3. ให้ top ชี้ไปยัง next ของ top เดิม
4. ลบ top เดิมออก
5. ลดค่า count



Linked List Stack

Pop (กรณีที่เหลือโหนดสุดท้าย)

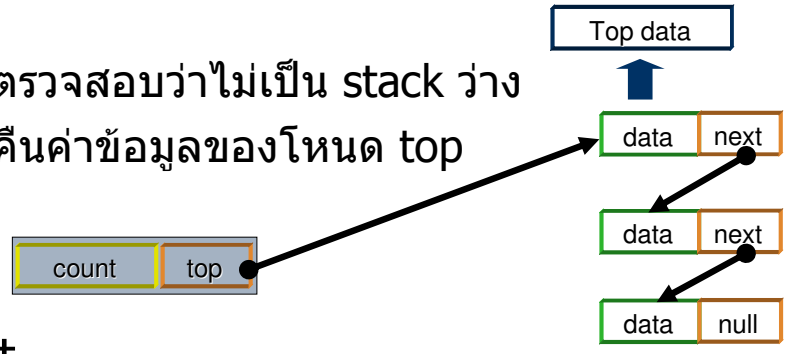
1. ตรวจสอบว่าเหลือโหนดสุดท้าย (next ของ top เป็น null หรือ count เป็น 1)
2. คืนค่าข้อมูลของ node top
3. ให้ top เป็น null
4. ลบ top เดิมออก
5. ลบค่า count (เป็น 0)



Linked List Stack

Top

1. ตรวจสอบว่าไม่เป็น stack ว่าง
2. คืนค่าข้อมูลของโหนด top



Count

1. คืนค่า count

Linked List Stack

IsEmpty

1. คืนค่า true เมื่อ count เป็น 0

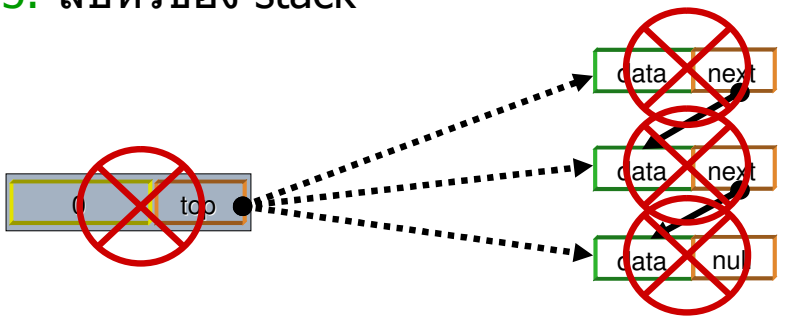
IsFull

1. ตรวจสอบ memory ไม่มีที่เหลือสำหรับโหนดข้อมูล
2. คืนค่า true

Linked List Stack

Destroy

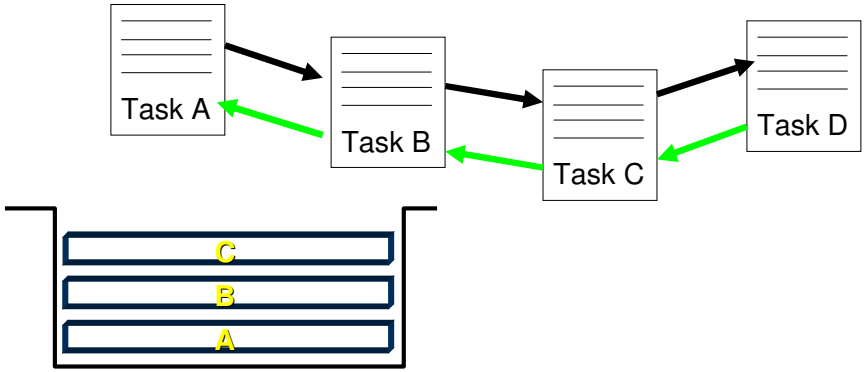
1. ตรวจสอบว่า stack ไม่ว่าง
2. ทำการวน loop ลบ (pop) จนกระทั่ง stack ว่าง
3. ลบหัวของ stack



Applied Stack

ประยุกต์ใช้ stack

- เป็นการเก็บค่าแบบ LIFO
- เหมาะกับการใช้งานในลักษณะ call function



Applied Stack

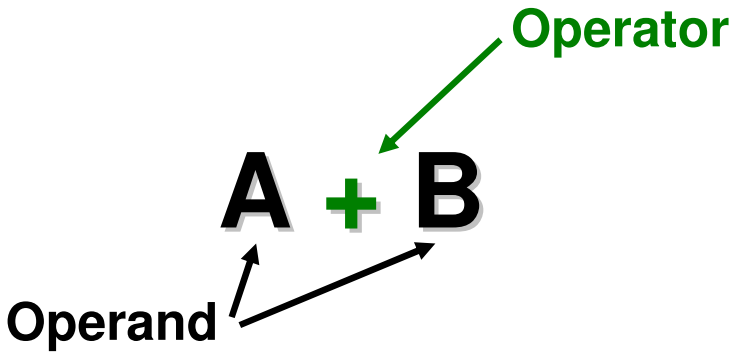
การใช้ stack ในการคำนวณนิพจน์ทางคณิตศาสตร์

- นิพจน์ Infix , Prefix , Postfix
- การคำนวณนิพจน์ Postfix โดยใช้ Stack
- การแปลงนิพจน์ Infix เป็น Postfix

Applied Stack

นิพจน์ Infix

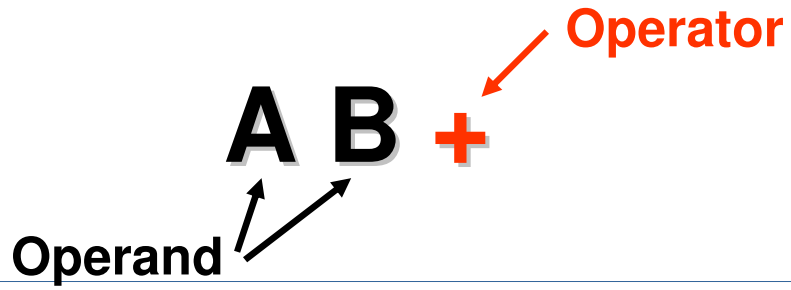
- นิพจน์ที่คนอ่านเข้าใจเป็นนิพจน์แบบ infix
- Operator อยู่ระหว่าง Operand



Applied Stack

นิพจน์ Postfix

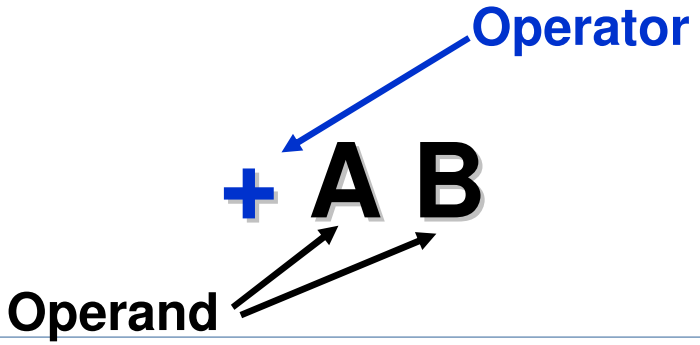
- Operator อยู่หลัง Operand
- คอมพิวเตอร์ใช้ในการคำนวณได้สะดวกกว่า มีการทำการคำนวณจากซ้ายไปขวาตามลำดับ
- ไม่ต้องใช้วงเล็บในการสร้างนิพจน์



Applied Stack

นิพจน์ Prefix

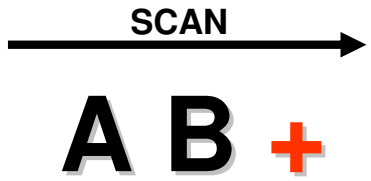
- Operator อยู่หน้า Operand
- ทำการคำนวณจากขวาไปซ้าย



Applied Stack

การประมวลผลนิพจน์ Postfix

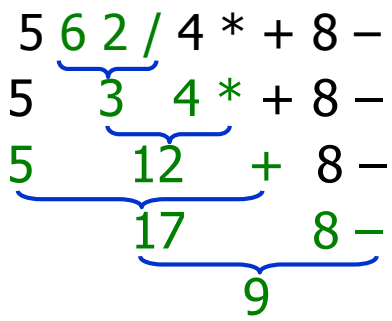
- ทำการ scan จากซ้ายมาขวา
- เมื่อเจอ operator ก็จะทำการ operate สองตัวที่อยู่ก่อนหน้า
- เหมาะกับการใช้ stack ในการเก็บ operand



Applied Stack

ตัวอย่างการประมวลผลนิพจน์ Postfix

5 6 2 / 4 * + 8 –



Applied Stack

การประมวลผลนิพจน์ Postfix โดยใช้ stack

- ทำการ scan จากซ้ายไปขวา
- ถ้าเป็น operand ให้ push stack
- ถ้าเป็น operator ให้ pop ออกมาสองตัวนำมา operate แล้ว push ค่ากลับไปยัง stack
- วนทำจน scan หมด จะได้คำตอบใน stack

Applied Stack

ตัวอย่างการประมวลผลนิพจน์ Postfix โดยใช้ stack

5 6 2 / 4 * + 8 –

push 5	[5]
push 6	[6, 5]
push 2	[2, 6, 5]
'/' pop 2, 6 → 6/2 push 3	[3, 5]
push 4	[4, 3, 5]
'*' pop 4, 3 → 3*4 push 12	[12, 5]

Applied Stack

ตัวอย่างการประมวลผลนิพจน์ Postfix โดยใช้ stack(ต่อ)

5 6 2 / 4 * + 8 –

'*' pop 4, 3 → 3*4 push 12	[12, 5]
'+' pop 12, 5 → 5+12 push 17	[17]
push 8	[8, 17]
'-' pop 8, 17 → 17-8 push 9	[9]

Answer = 9

Applied Stack

การแปลงนิพจน์ Infix เป็น Postfix

(5 + ((6 / 2) * 4)) – 8
(5 + ((6 2 /) * 4)) – 8
(5 + (6 2 / 4 *)) – 8
(5 6 2 / 4 * +) – 8
5 6 2 / 4 * + 8 –

Applied Stack

การแปลงนิพจน์ Infix เป็น Postfix

- Scan จากซ้ายไปขวา
- Operand เอาไว้ในผลลัพธ์
- '(' ให้ push เข้า stack
- ')' ให้ pop ไปไว้ในผลลัพธ์จนกว่าจะเจอ '('
- Operator
  - ถ้า priority สูงกว่าหรือเท่ากับให้ push เข้า (→ + - → * /
  - ไม่ ให้ pop ไปไว้ในผลลัพธ์จนกว่าจะ push ได้
- ทำซ้ำจนหมด แล้ว pop ที่เหลือไปเป็นผลลัพธ์

Applied Stack

ตัวอย่างการแปลงนิพจน์ Infix เป็น Postfix

(5 + ((6 / 2) * 4)) – 8

Token	Stack	Postfix
(	(	
5	(	5
+	(+	5
(	(+ (	5
(	(+ ((	5
6	(+ ((	5 6
/	(+ ((/	5 6
2	(+ ((/	5 6 2

Applied Stack

แปลงนิพจน์ Infix เป็น Postfix (5 + ((6 / 2) * 4)) – 8

Token	Stack	Postfix
)	(+ (	5 6 2 /
*	(+ (*	5 6 2 /
4	(+ (*	5 6 2 / 4
)	(+	5 6 2 / 4 *
)		5 6 2 / 4 * +
-	-	5 6 2 / 4 * +
8	-	5 6 2 / 4 * + 8
		5 6 2 / 4 * + 8 -

คำถามข้อสงสัย