10301222 โครงสร้างข้อมูลและอัลกอริทึม

Chapter 5

Stack

ผศ.ดร. ปวีณ เขื่อนแก้ว

สาขาวิชาวิทยาการคอมพิวเตอร์ มหาวิทยาลัยแม่โจ้

Stack

มีช่องทางเข้าออกทางเดียว

สิ่งที่ใส่เข้าไปทีหลัง จะถูกนำออกมาใช้ก่อน

Last-in First-out (LIFO)

เรียงทับซ้อน = Stack

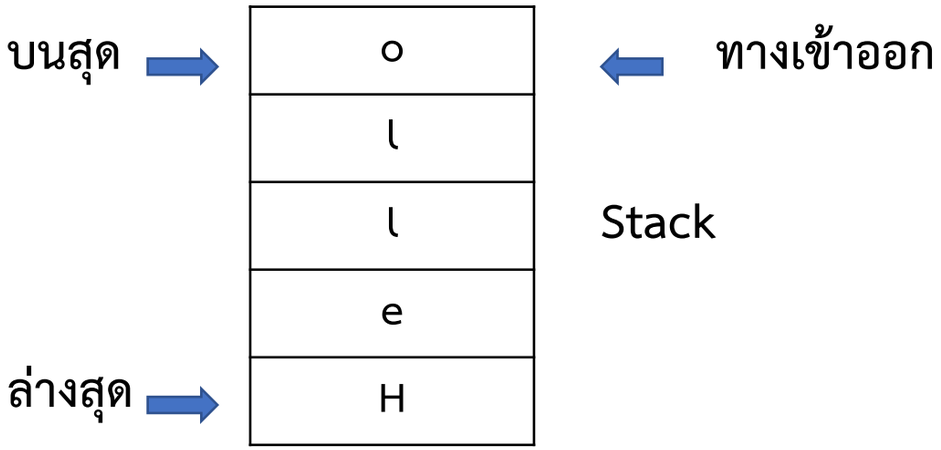
Stack

สแตก (Stack) คือ ชุดของข้อมูลชนิดเดียวกัน (homogeneous) ที่เรียงต่อกัน และจัดลำดับการเก็บข้อมูลแบบเข้าทีหลังและนำออกก่อน มักเรียกว่าไลโฟ (LiFo = Last in First out)

อาจมองว่าข้อมูลเข้าใหม่จะมาเรียงต่ออยู่ด้านบน

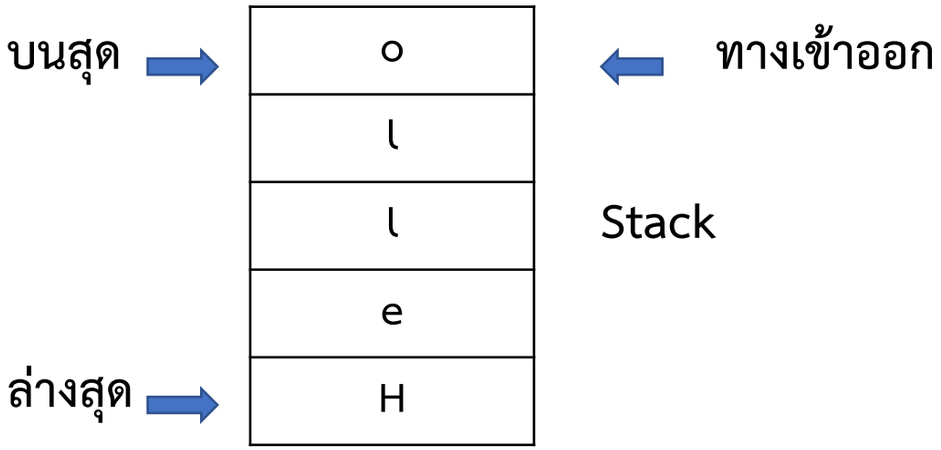
หากเรียกใช้ก็นำด้านบนสุดออกไปก่อน

ดังนั้นข้อมูลที่เข้ามานานที่สุด จะอยู่ล่างสุด และจะอยู่ในสแตกนานที่สุด



Stack

- ข้อมูลใน stack ต้องเป็นชนิดเดียวกัน
- ข้อมูลเรียงตามลำดับการป้อน
- เข้าถึงข้อมูลภายในไม่ได้
- จะมองเห็นเฉพาะข้อมูลที่อยู่บนสุดของ stack เท่านั้น
- ในทางทฤษฎี Stack จะมีขนาดเท่าใด ก็ได้
- ในทางปฏิบัติ ขนาด stack จะถูกจำกัดด้วยขนาดของหน่วยความจำ



Stack

สแตก (Stack) คือโครงสร้างข้อมูลที่ง่ายที่สุด

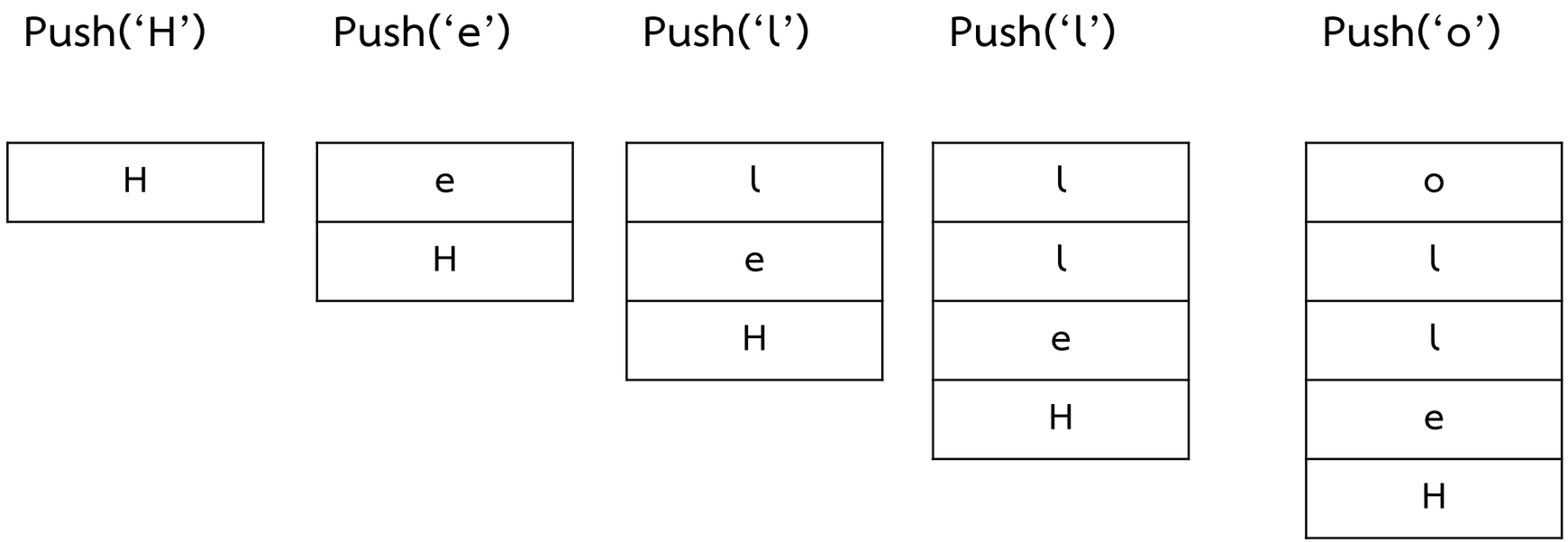
- ทางเข้า-ออก ของข้อมูลมีเพียงตำแหน่งเดียว และเป็นตำแหน่งเดียวกัน
- operation มีเพียง 2 แบบคือ
  - นำข้อมูลเข้าข้อมูล (push)
  - นำข้อมูลออก (pop)

- operation เพิ่มเติม จะมีหรือไม่มีก็ได้
  - ดูข้อมูลตำแหน่งบนสุดโดยไม่นำข้อมูล ออก (peek)
  - isEmpty()
  - isFull()

Stack: operation

การนำเข้าข้อมูล Push()

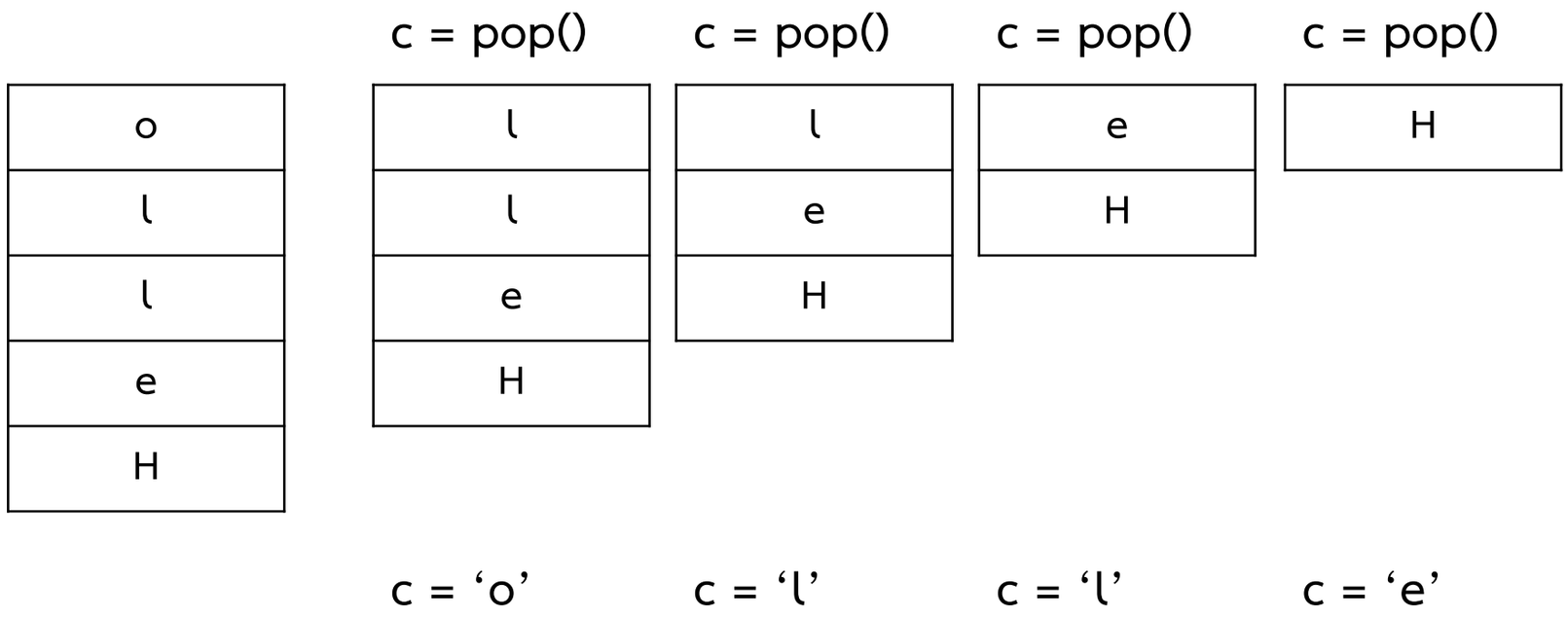
1 หาก stack ยังไม่เต็ม ให้แทรกข้อมูลไว้หน้าสุด



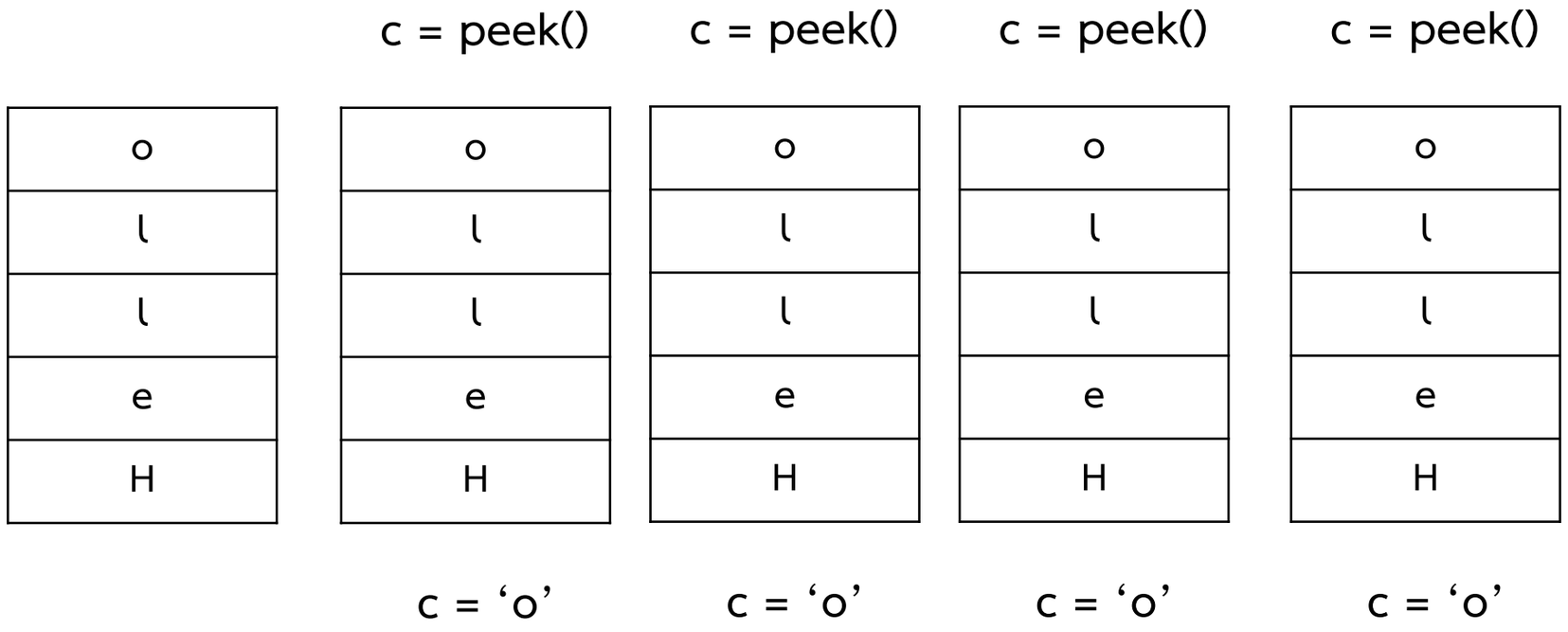
Stack: operation

การนำข้อมูลออก / อ่านข้อมูล Pop()

1 หาก stack ไม่ได้ empty ให้ดึงข้อมูลที่อยู่หน้าสุดออกมา



อ่านข้อมูลโดยไม่นำข้อมูลออก Peek()



Stack: operation

push(1)

push(2)

pop()

peek()

push(3)

pop()

push(4)

push(5)

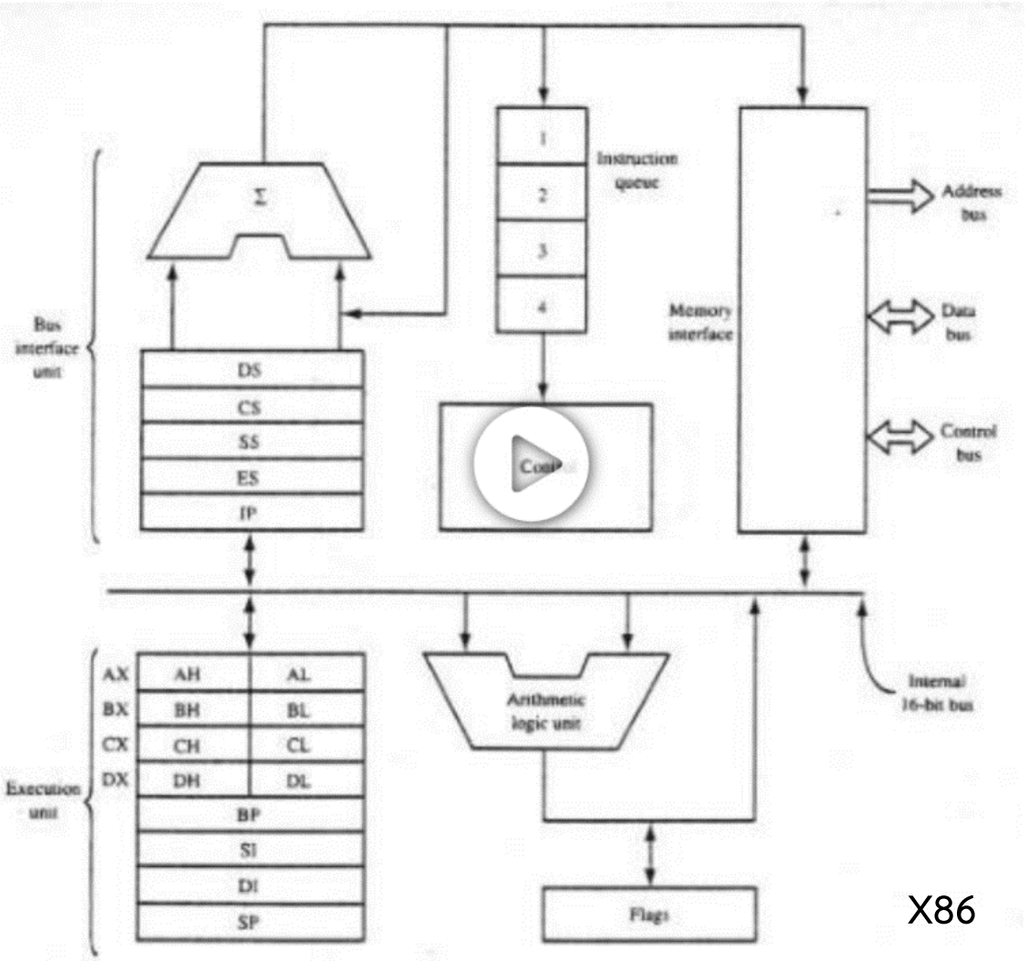
Stack

Stack: Implementation

หน่วยประมวลผล มักจะมีวงจรหน่วยความจำในลักษณะที่เป็น stack อยู่ภายใน แต่อย่างไรก็ตาม เราสามารถสร้างซอฟต์แวร์ stack ขึ้นมาใช้งานได้ โดยมีแนวทางในการสร้าง 2 วิธีคือ

1 ใช้ Array

2 ใช้ Linked list



X86

Stack: Array

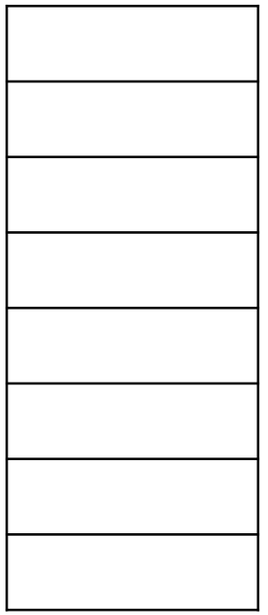
1 สร้าง Array ขึ้นมาขนาดเท่าข้อมูลที่ต้องการเก็บ

2 สร้างตัวแปร top ขึ้นมาเพื่อเก็บตำแหน่งบนสุด

3 หาก stack ไม่มีข้อมูล ให้กดหนด top=-1

4 การ push ให้เพิ่มข้อมูล top ไป 1 และนำข้อมูลใส่ในตำแหน่ง top

5 การ pop ให้อ่านข้อมูลในตำแหน่ง top และลดข้อมูล top ไป 1



top = -1

Stack: Array

1 สร้าง Array ขึ้นมาขนาดเท่าข้อมูลที่ต้องการเก็บ

2 สร้างตัวแปร top ขึ้นมาเพื่อเก็บตำแหน่งบนสุด

3 หาก stack ไม่มีข้อมูล ให้กดหนด top=-1

4 การ push ให้เพิ่มข้อมูล top ไป 1 และนำข้อมูลใส่ในตำแหน่ง top

5 การ pop ให้อ่านข้อมูลในตำแหน่ง top และลดข้อมูล top ไป 1

push('H')

top = -1

Stack: Array

1 สร้าง Array ขึ้นมาขนาดเท่าข้อมูลที่ต้องการเก็บ

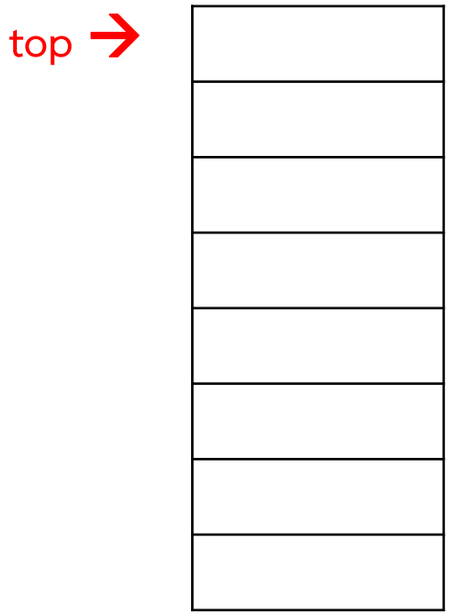
2 สร้างตัวแปร top ขึ้นมาเพื่อเก็บตำแหน่งบนสุด

3 หาก stack ไม่มีข้อมูล ให้กดหนด top=-1

4 การ push ให้เพิ่มข้อมูล top ไป 1 และนำข้อมูลใส่ในตำแหน่ง top

5 การ pop ให้อ่านข้อมูลในตำแหน่ง top และลดข้อมูล top ไป 1

push('H')



Stack: Array

1 สร้าง Array ขึ้นมาขนาดเท่าข้อมูลที่ต้องการเก็บ

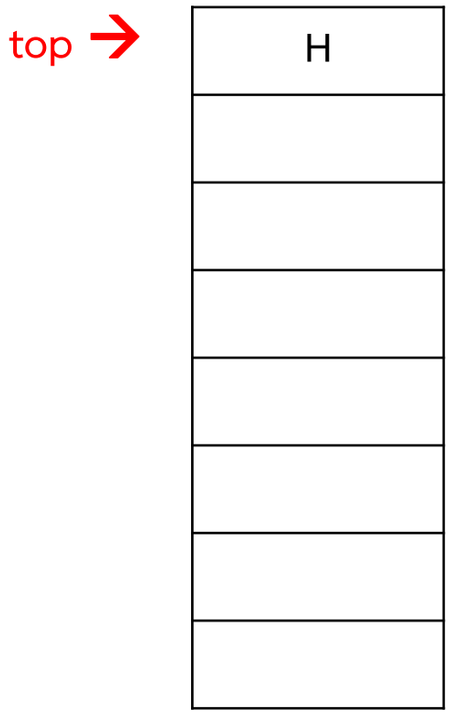
2 สร้างตัวแปร top ขึ้นมาเพื่อเก็บตำแหน่งบนสุด

3 หาก stack ไม่มีข้อมูล ให้กดหนด top=-1

4 การ push ให้เพิ่มข้อมูล top ไป 1 และนำข้อมูลใส่ในตำแหน่ง top

5 การ pop ให้อ่านข้อมูลในตำแหน่ง top และลดข้อมูล top ไป 1

push('H')



Stack: Array

1 สร้าง Array ขึ้นมาขนาดเท่าข้อมูลที่ต้องการเก็บ

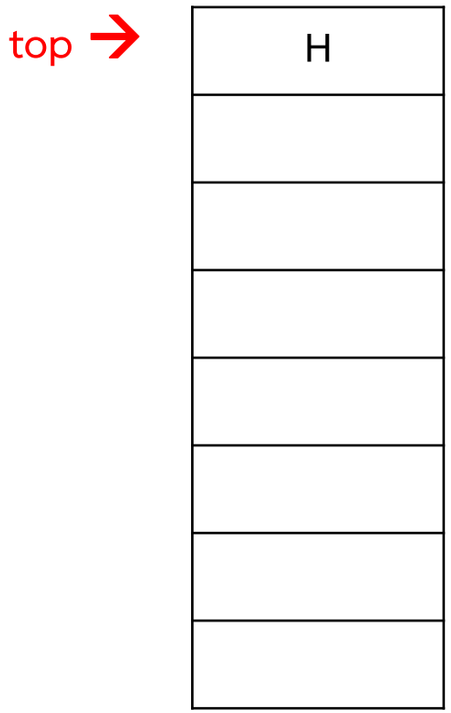
2 สร้างตัวแปร top ขึ้นมาเพื่อเก็บตำแหน่งบนสุด

3 หาก stack ไม่มีข้อมูล ให้กดหนด top=-1

4 การ push ให้เพิ่มข้อมูล top ไป 1 และนำข้อมูลใส่ในตำแหน่ง top

5 การ pop ให้อ่านข้อมูลในตำแหน่ง top และลดข้อมูล top ไป 1

push(‘e’)



Stack: Array

1 สร้าง Array ขึ้นมาขนาดเท่าข้อมูลที่ต้องการเก็บ

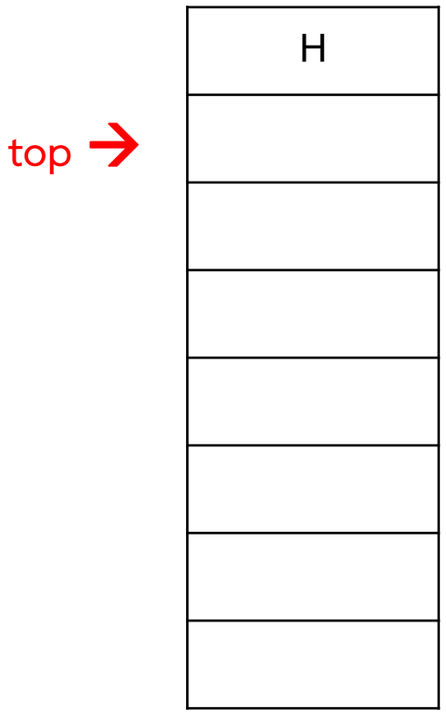
2 สร้างตัวแปร top ขึ้นมาเพื่อเก็บตำแหน่งบนสุด

3 หาก stack ไม่มีข้อมูล ให้กดหนด top=-1

4 การ push ให้เพิ่มข้อมูล top ไป 1 และนำข้อมูลใส่ในตำแหน่ง top

5 การ pop ให้อ่านข้อมูลในตำแหน่ง top และลดข้อมูล top ไป 1

push(‘e’)



Stack: Array

1 สร้าง Array ขึ้นมาขนาดเท่าข้อมูลที่ต้องการเก็บ

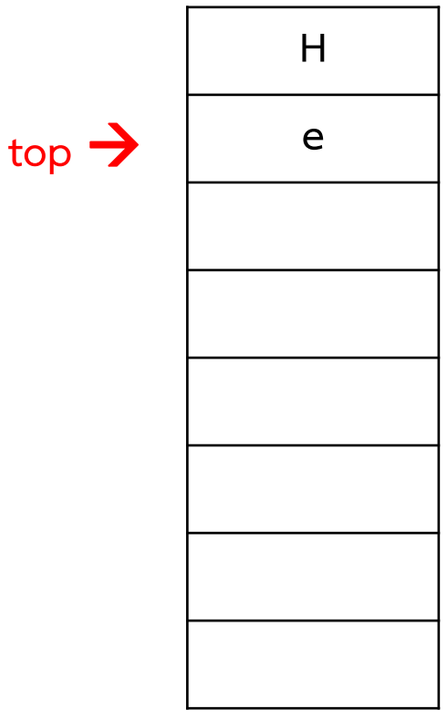
2 สร้างตัวแปร top ขึ้นมาเพื่อเก็บตำแหน่งบนสุด

3 หาก stack ไม่มีข้อมูล ให้กดหนด top=-1

4 การ push ให้เพิ่มข้อมูล top ไป 1 และนำข้อมูลใส่ในตำแหน่ง top

5 การ pop ให้อ่านข้อมูลในตำแหน่ง top และลดข้อมูล top ไป 1

push(‘e’)



Stack: Array

1 สร้าง Array ขึ้นมาขนาดเท่าข้อมูลที่ต้องการเก็บ

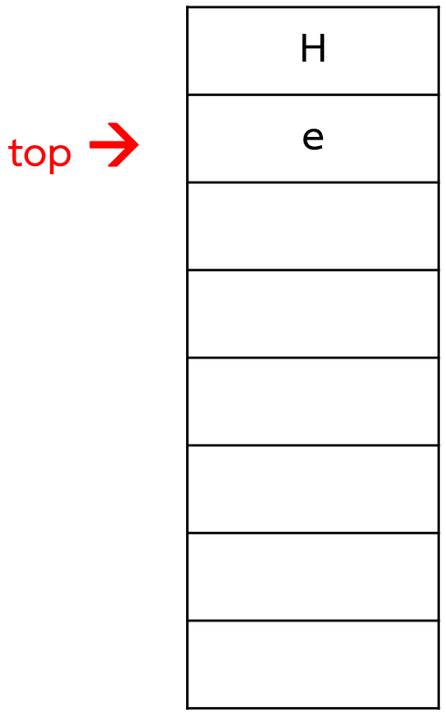
2 สร้างตัวแปร top ขึ้นมาเพื่อเก็บตำแหน่งบนสุด

3 หาก stack ไม่มีข้อมูล ให้กดหนด top=-1

4 การ push ให้เพิ่มข้อมูล top ไป 1 และนำข้อมูลใส่ในตำแหน่ง top

5 การ pop ให้อ่านข้อมูลในตำแหน่ง top และลดข้อมูล top ไป 1

push('l')



Stack: Array

1 สร้าง Array ขึ้นมาขนาดเท่าข้อมูลที่ต้องการเก็บ

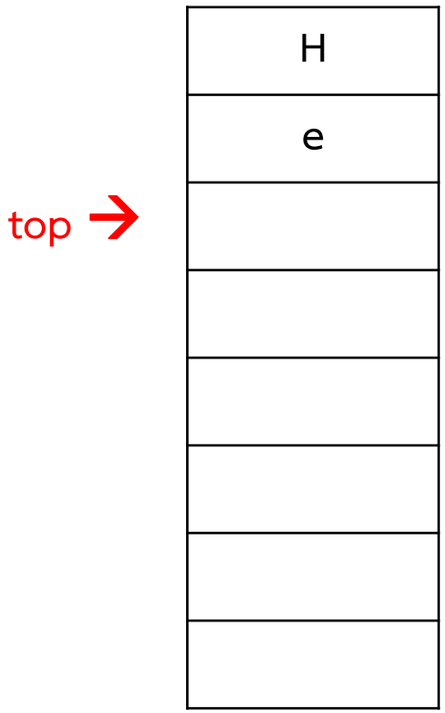
2 สร้างตัวแปร top ขึ้นมาเพื่อเก็บตำแหน่งบนสุด

3 หาก stack ไม่มีข้อมูล ให้กดหนด top=-1

4 การ push ให้เพิ่มข้อมูล top ไป 1 และนำข้อมูลใส่ในตำแหน่ง top

5 การ pop ให้อ่านข้อมูลในตำแหน่ง top และลดข้อมูล top ไป 1

push('l')



Stack: Array

1 สร้าง Array ขึ้นมาขนาดเท่าข้อมูลที่ต้องการเก็บ

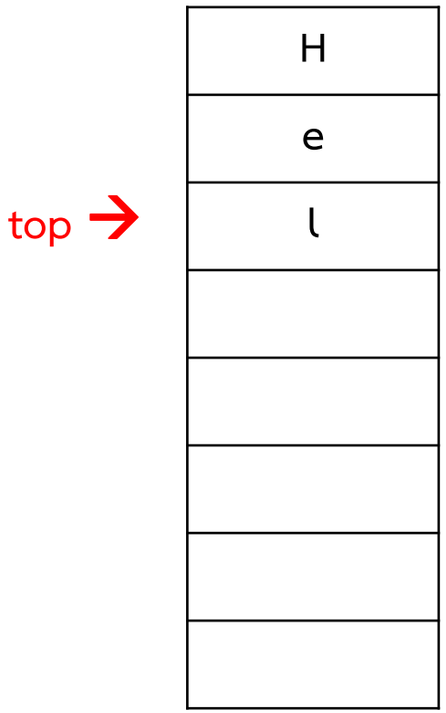
2 สร้างตัวแปร top ขึ้นมาเพื่อเก็บตำแหน่งบนสุด

3 หาก stack ไม่มีข้อมูล ให้กดหนด top=-1

4 การ push ให้เพิ่มข้อมูล top ไป 1 และนำข้อมูลใส่ในตำแหน่ง top

5 การ pop ให้อ่านข้อมูลในตำแหน่ง top และลดข้อมูล top ไป 1

push('l')



Stack: Array

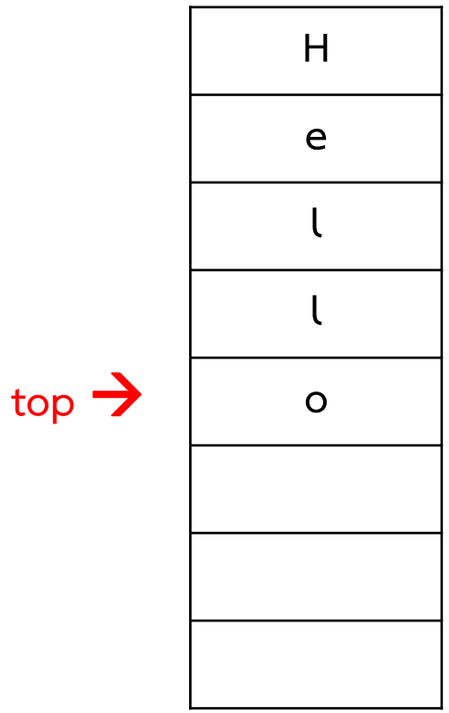
1 สร้าง Array ขึ้นมาขนาดเท่าข้อมูลที่ต้องการเก็บ

2 สร้างตัวแปร top ขึ้นมาเพื่อเก็บตำแหน่งบนสุด

3 หาก stack ไม่มีข้อมูล ให้กดหนด top=-1

4 การ push ให้เพิ่มข้อมูล top ไป 1 และนำข้อมูลใส่ในตำแหน่ง top

5 การ pop ให้อ่านข้อมูลในตำแหน่ง top และลดข้อมูล top ไป 1



Stack: Array

1 สร้าง Array ขึ้นมาขนาดเท่าข้อมูลที่ต้องการเก็บ

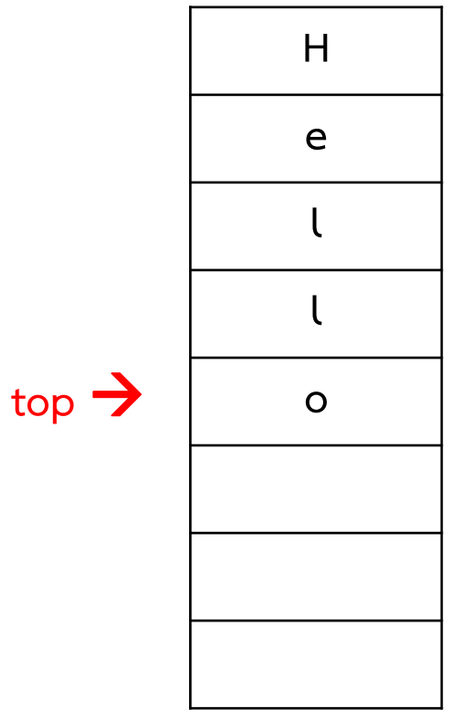
2 สร้างตัวแปร top ขึ้นมาเพื่อเก็บตำแหน่งบนสุด

3 หาก stack ไม่มีข้อมูล ให้กดหนด top=-1

4 การ push ให้เพิ่มข้อมูล top ไป 1 และนำข้อมูลใส่ในตำแหน่ง top

5 การ pop ให้อ่านข้อมูลในตำแหน่ง top และลดข้อมูล top ไป 1

c=pop()



Stack: Array

1 สร้าง Array ขึ้นมาขนาดเท่าข้อมูลที่ต้องการเก็บ

2 สร้างตัวแปร top ขึ้นมาเพื่อเก็บตำแหน่งบนสุด

3 หาก stack ไม่มีข้อมูล ให้กดหนด top=-1

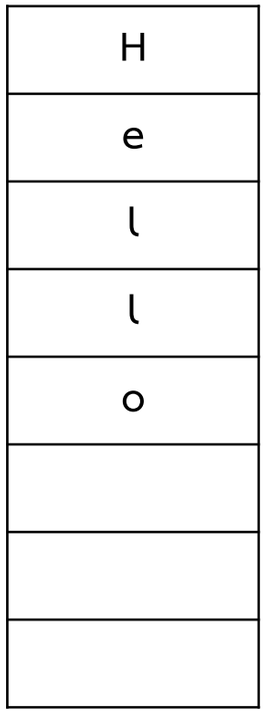
4 การ push ให้เพิ่มข้อมูล top ไป 1 และนำข้อมูลใส่ในตำแหน่ง top

5 การ pop ให้อ่านข้อมูลในตำแหน่ง top และลดข้อมูล top ไป 1

c=pop()

c='o'

top →



Stack: Array

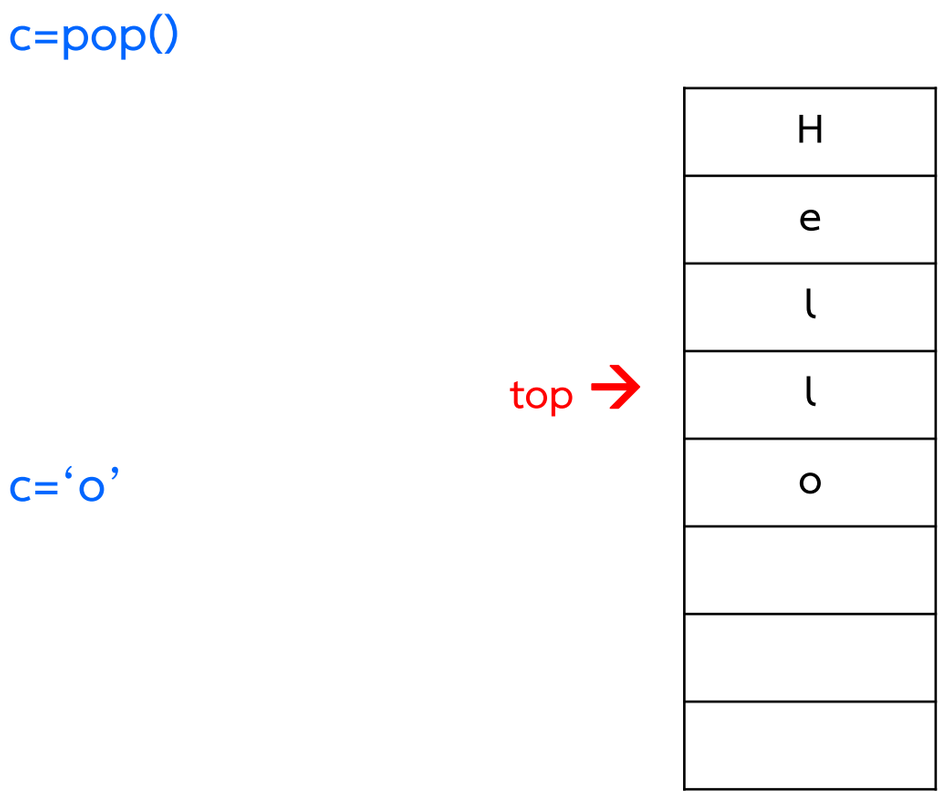
1 สร้าง Array ขึ้นมาขนาดเท่าข้อมูลที่ต้องการเก็บ

2 สร้างตัวแปร top ขึ้นมาเพื่อเก็บตำแหน่งบนสุด

3 หาก stack ไม่มีข้อมูล ให้กดหนด top=-1

4 การ push ให้เพิ่มข้อมูล top ไป 1 และนำข้อมูลใส่ในตำแหน่ง top

5 การ pop ให้อ่านข้อมูลในตำแหน่ง top และลดข้อมูล top ไป 1



Stack: Array

1 สร้าง Array ขึ้นมาขนาดเท่าข้อมูลที่ต้องการเก็บ

2 สร้างตัวแปร top ขึ้นมาเพื่อเก็บตำแหน่งบนสุด

3 หาก stack ไม่มีข้อมูล ให้กดหนด top=-1

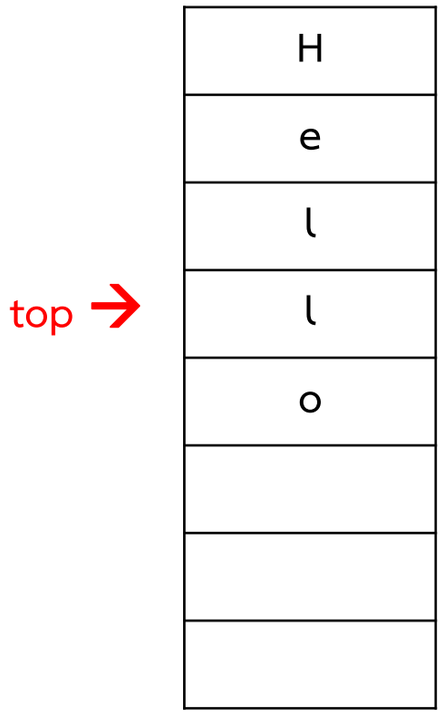
4 การ push ให้เพิ่มข้อมูล top ไป 1 และนำข้อมูลใส่ในตำแหน่ง top

5 การ pop ให้อ่านข้อมูลในตำแหน่ง top และลดข้อมูล top ไป 1

c=pop()

c=pop()

c='o'



Stack: Array

1 สร้าง Array ขึ้นมาขนาดเท่าข้อมูลที่ต้องการเก็บ

2 สร้างตัวแปร top ขึ้นมาเพื่อเก็บตำแหน่งบนสุด

3 หาก stack ไม่มีข้อมูล ให้กดหนด top=-1

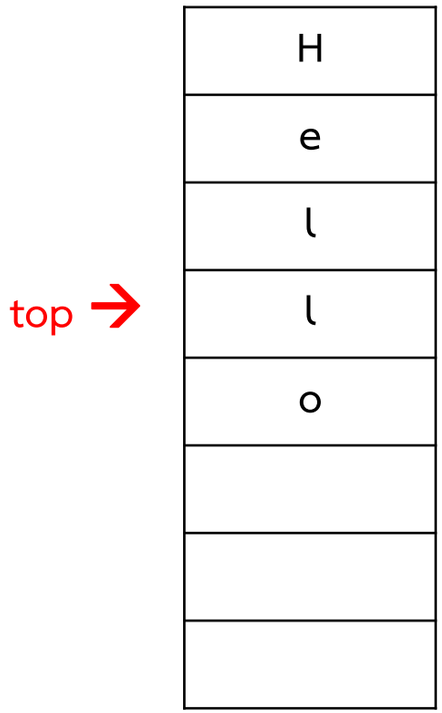
4 การ push ให้เพิ่มข้อมูล top ไป 1 และนำข้อมูลใส่ในตำแหน่ง top

5 การ pop ให้อ่านข้อมูลในตำแหน่ง top และลดข้อมูล top ไป 1

c=pop()

c=pop()

c='l'



Stack: Array

1 สร้าง Array ขึ้นมาขนาดเท่าข้อมูลที่ต้องการเก็บ

2 สร้างตัวแปร top ขึ้นมาเพื่อเก็บตำแหน่งบนสุด

3 หาก stack ไม่มีข้อมูล ให้กดหนด top=-1

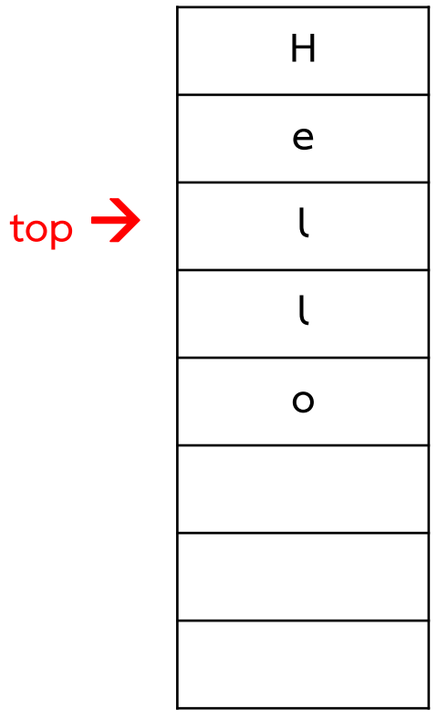
4 การ push ให้เพิ่มข้อมูล top ไป 1 และนำข้อมูลใส่ในตำแหน่ง top

5 การ pop ให้อ่านข้อมูลในตำแหน่ง top และลดข้อมูล top ไป 1

c=pop()

c=pop()

c='l'



Stack: Array

1 สร้าง Array ขึ้นมาขนาดเท่าข้อมูลที่ต้องการเก็บ

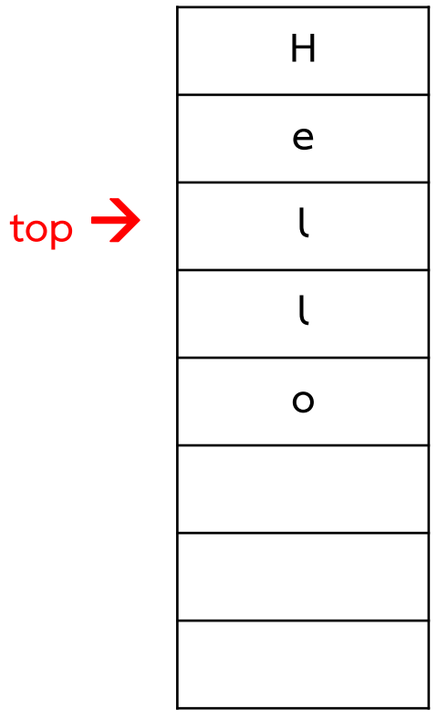
2 สร้างตัวแปร top ขึ้นมาเพื่อเก็บตำแหน่งบนสุด

3 หาก stack ไม่มีข้อมูล ให้กดหนด top=-1

4 การ push ให้เพิ่มข้อมูล top ไป 1 และนำข้อมูลใส่ในตำแหน่ง top

5 การ pop ให้อ่านข้อมูลในตำแหน่ง top และลดข้อมูล top ไป 1

push('x')



Stack: Array

1 สร้าง Array ขึ้นมาขนาดเท่าข้อมูลที่ต้องการเก็บ

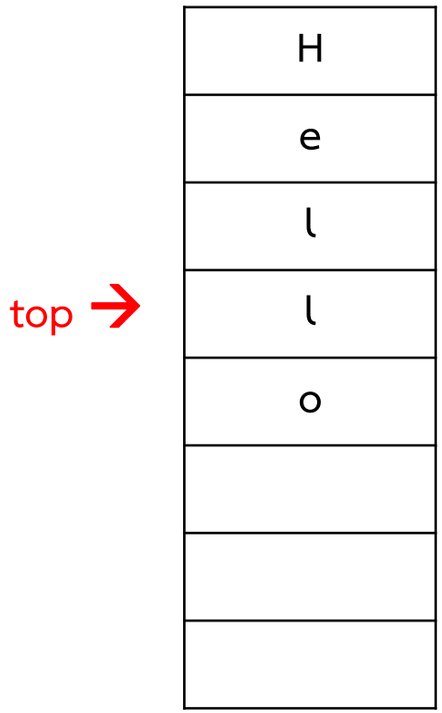
2 สร้างตัวแปร top ขึ้นมาเพื่อเก็บตำแหน่งบนสุด

3 หาก stack ไม่มีข้อมูล ให้กดหนด top=-1

4 การ push ให้เพิ่มข้อมูล top ไป 1 และนำข้อมูลใส่ในตำแหน่ง top

5 การ pop ให้อ่านข้อมูลในตำแหน่ง top และลดข้อมูล top ไป 1

push('x')

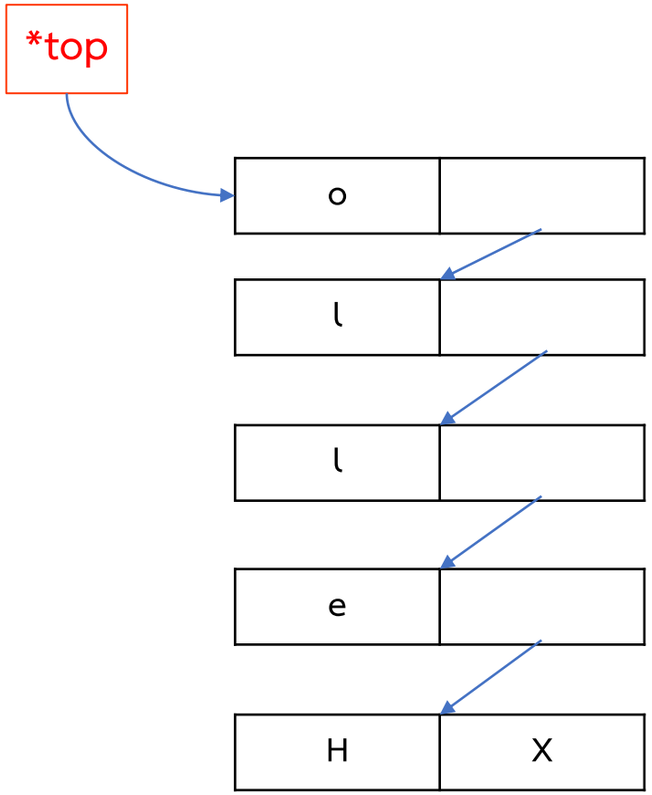


ข้อเสียของการใช้ Array

- ต้องกำหนดขนาดไว้ก่อนตั้งแต่แรก
- เปลี่ยนแปลงขนาดไม่ได้
- หากใส่ข้อมูลไม่ครบ จะสิ้นเปลืองพื้นที่หน่วยความจำ
- หากกำหนดขนาดไว้เพียงพอ โปรแกรมจะหยุดทำงาน

Stack: Linked List

ใช้ Node และ Pointer โดยใช้ top แทนตัวแปร head pointer



Stack: Linked List

ขั้นตอนของการ push()

1 สร้าง node ใหม่

2 ใส่ข้อมูลลงไปใน node ใหม่

3 กำหนด link ชี้ไปยัง node ก่อนหน้า

4 ย้าย *top (*head) มาชี้ node ใหม่

*top

ในบทที่แล้ว ก็คือการแทรก node ไว้หน้าสุดนั่นเอง

Stack: Linked List

ขั้นตอนของการ push()

1 สร้าง node ใหม่

2 ใส่ข้อมูลลงไปใน node ใหม่

3 กำหนด link ชี้ไปยัง node ก่อนหน้า

4 ย้าย *top (*head) มาชี้ node ใหม่

*top

push(‘H’)

Stack: Linked List

ขั้นตอนของการ push()

1 สร้าง node ใหม่

2 ใส่ข้อมูลลงไปใน node ใหม่

3 กำหนด link ชี้ไปยัง node ก่อนหน้า

4 ย้าย *top (*head) มาชี้ node ใหม่

*top

push('H')

Stack: Linked List

ขั้นตอนของการ push()

1 สร้าง node ใหม่

2 ใส่ข้อมูลลงไปใน node ใหม่

3 กำหนด link ชี้ไปยัง node ก่อนหน้า

4 ย้าย *top (*head) มาชี้ node ใหม่

*top

push('H')

Stack: Linked List

ขั้นตอนของการ push()

1 สร้าง node ใหม่

2 ใส่ข้อมูลลงไปใน node ใหม่

3 กำหนด link ชี้ไปยัง node ก่อนหน้า

4 ย้าย *top (*head) มาชี้ node ใหม่

*top

push('H')

Stack: Linked List

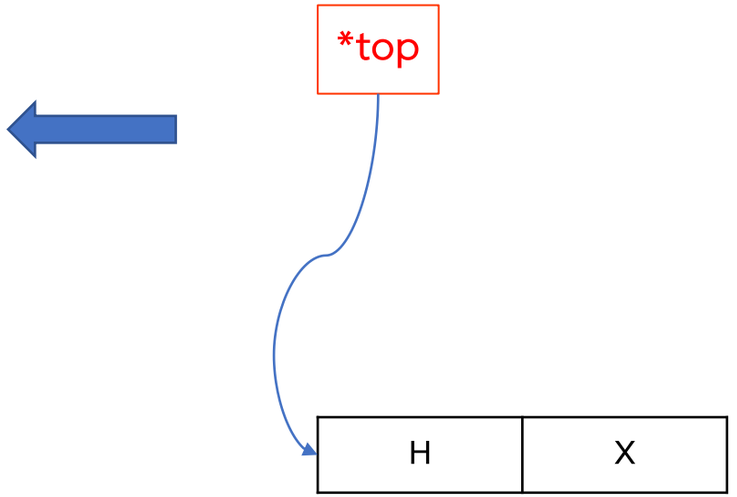
ขั้นตอนของการ push()

1 สร้าง node ใหม่

2 ใส่ข้อมูลลงไปใน node ใหม่

3 กำหนด link ชี้ไปยัง node ก่อนหน้า

4 ย้าย *top (*head) มาชี้ node ใหม่



push('H')

Stack: Linked List

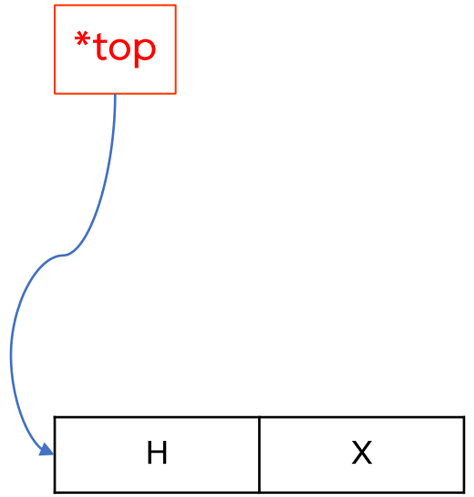
ขั้นตอนของการ push()

1 สร้าง node ใหม่

2 ใส่ข้อมูลลงไปใน node ใหม่

3 กำหนด link ชี้ไปยัง node ก่อนหน้า

4 ย้าย *top (*head) มาชี้ node ใหม่



push('e')

Stack: Linked List

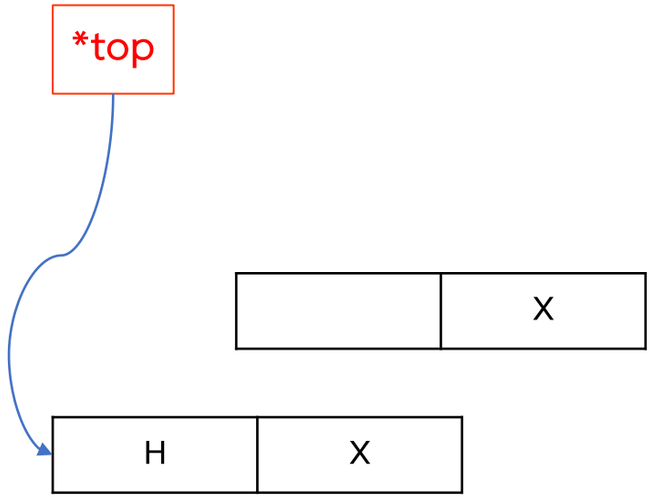
ขั้นตอนของการ push()

1 สร้าง node ใหม่

2 ใส่ข้อมูลลงไปใน node ใหม่

3 กำหนด link ชี้ไปยัง node ก่อนหน้า

4 ย้าย *top (*head) มาชี้ node ใหม่



push('e')

Stack: Linked List

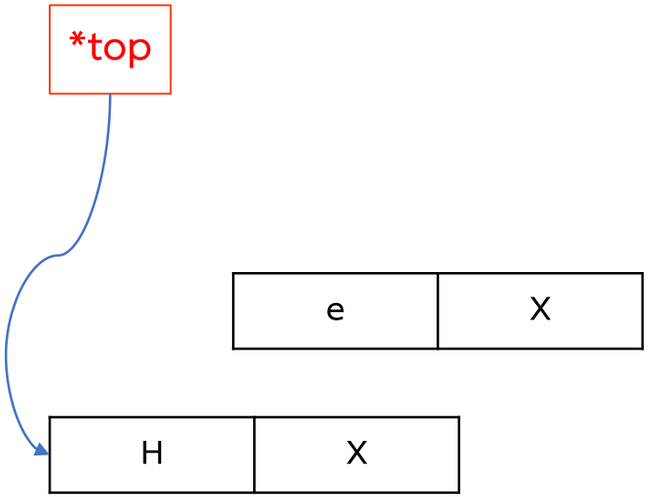
ขั้นตอนของการ push()

1 สร้าง node ใหม่

2 ใส่ข้อมูลลงไปใน node ใหม่

3 กำหนด link ชี้ไปยัง node ก่อนหน้า

4 ย้าย *top (*head) มาชี้ node ใหม่



push(‘e’)

Stack: Linked List

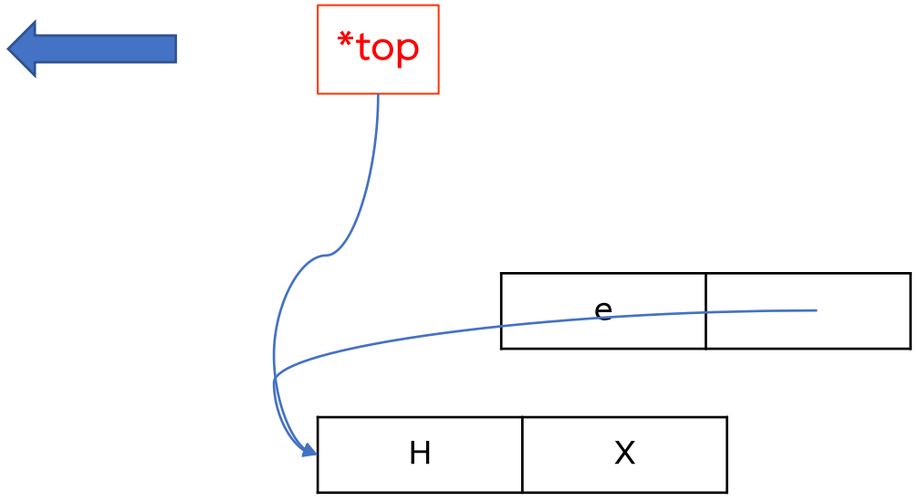
ขั้นตอนของการ push()

1 สร้าง node ใหม่

2 ใส่ข้อมูลลงไปใน node ใหม่

3 กำหนด link ชี้ไปยัง node ก่อนหน้า

4 ย้าย *top (*head) มาชี้ node ใหม่



push('e')

Stack: Linked List

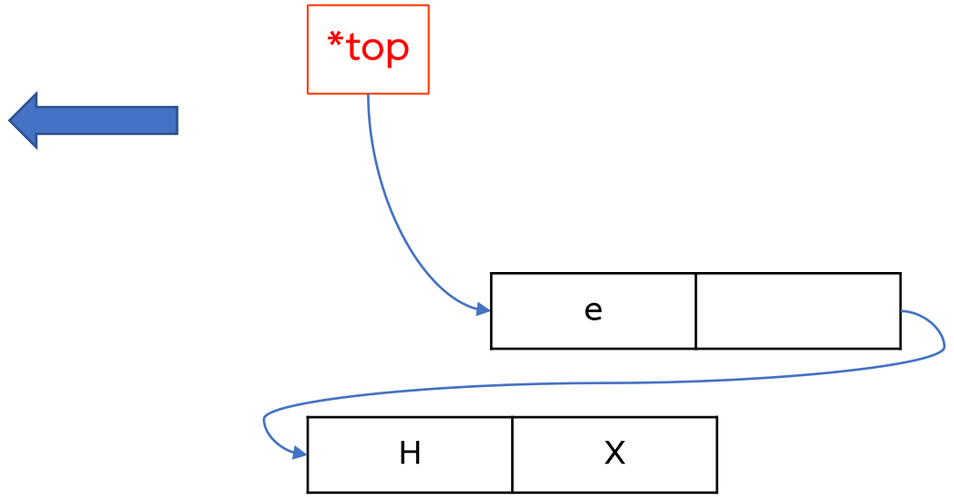
ขั้นตอนของการ push()

1 สร้าง node ใหม่

2 ใส่ข้อมูลลงไปใน node ใหม่

3 กำหนด link ชี้ไปยัง node ก่อนหน้า

4 ย้าย *top (*head) มาชี้ node ใหม่



push('e')

Stack: Linked List

ขั้นตอนของการ pop()

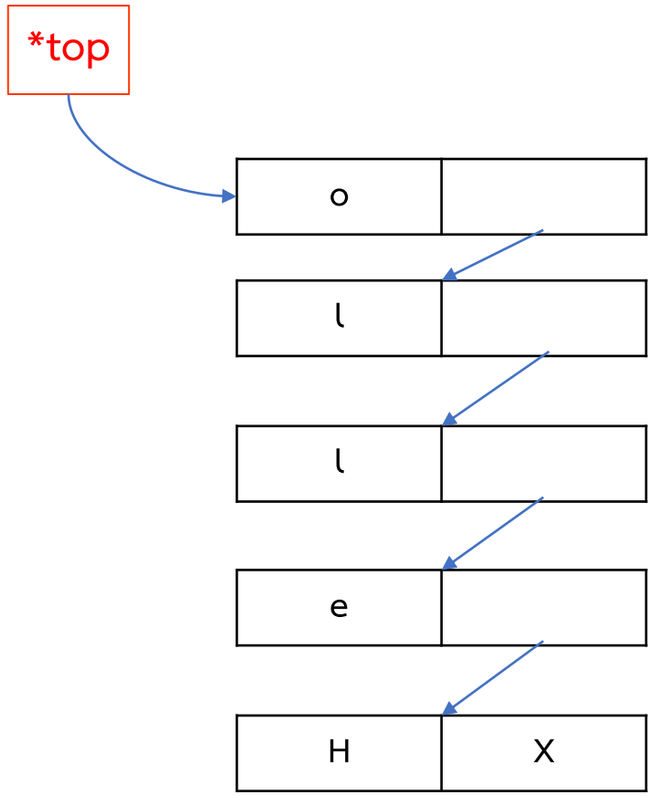
1 อ่านข้อมูลที่อยู่ใน node แรก

2 สร้าง pointer ตัวที่ 2 เพื่อชี้ไปยัง node แรก

3 เปลี่ยน *top (*head) ให้ชี้ไปยัง node ที่สอง

4 ลบ node ที่ pointer ตัวที่ 2 ชี้อยู่

c=pop()



Stack: Linked List

ขั้นตอนของการ pop()

1 อ่านข้อมูลที่อยู่ใน node แรก

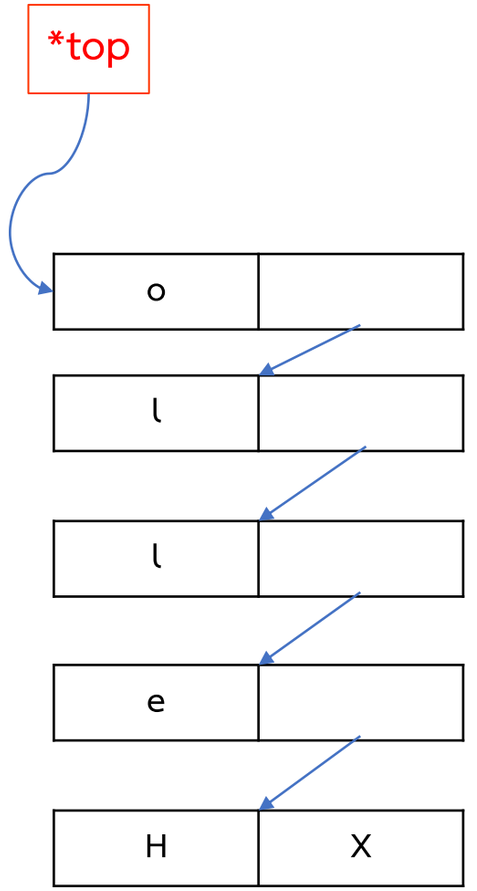
2 สร้าง pointer ตัวที่ 2 เพื่อชี้ไปยัง node แรก

3 เปลี่ยน *top (*head) ให้ชี้ไปยัง node ที่สอง

4 ลบ node ที่ pointer ตัวที่ 2 ชี้อยู่

c=pop()

c='o'



Stack: Linked List

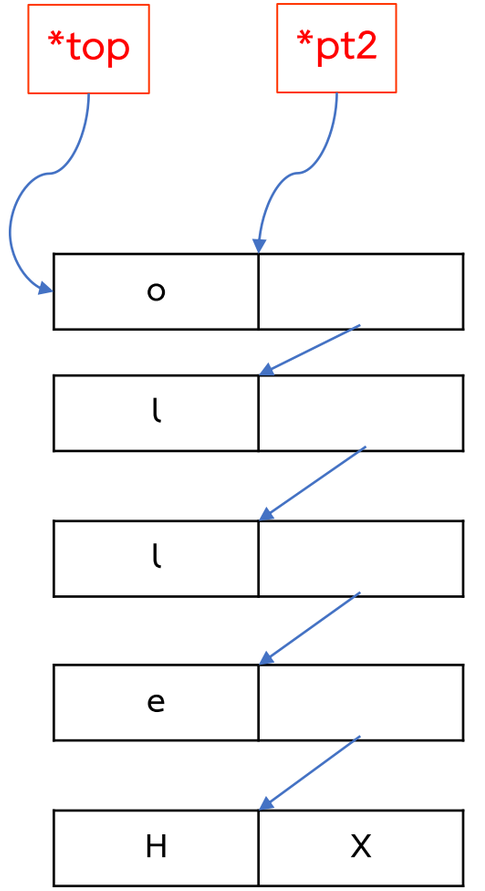
ขั้นตอนของการ pop()

1 อ่านข้อมูลที่อยู่ใน node แรก

2 สร้าง pointer ตัวที่ 2 เพื่อชี้ไปยัง node แรก

3 เปลี่ยน *top (*head) ให้ชี้ไปยัง node ที่สอง

4 ลบ node ที่ pointer ตัวที่ 2 ชี้อยู่



c=pop()

c='o'

Stack: Linked List

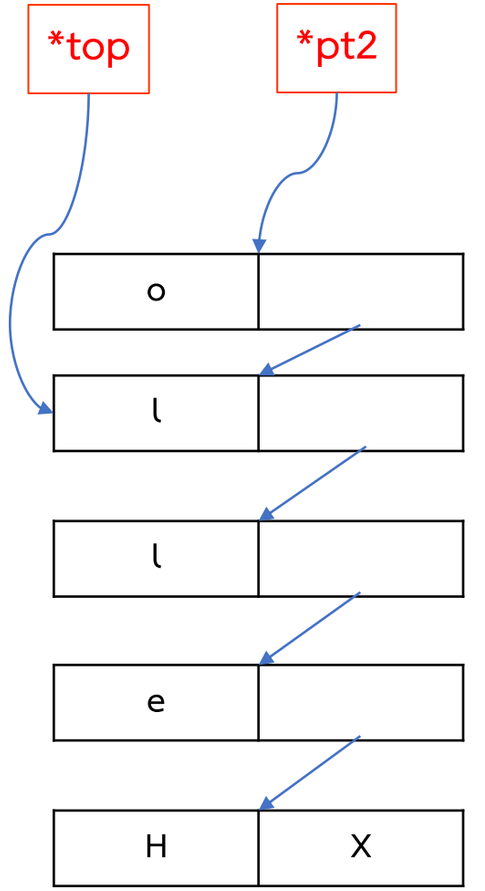
ขั้นตอนของการ pop()

1 อ่านข้อมูลที่อยู่ใน node แรก

2 สร้าง pointer ตัวที่ 2 เพื่อชี้ไปยัง node แรก

3 เปลี่ยน *top (*head) ให้ชี้ไปยัง node ที่สอง

4 ลบ node ที่ pointer ตัวที่ 2 ชี้อยู่



c=pop()

c='o'

Stack: Linked List

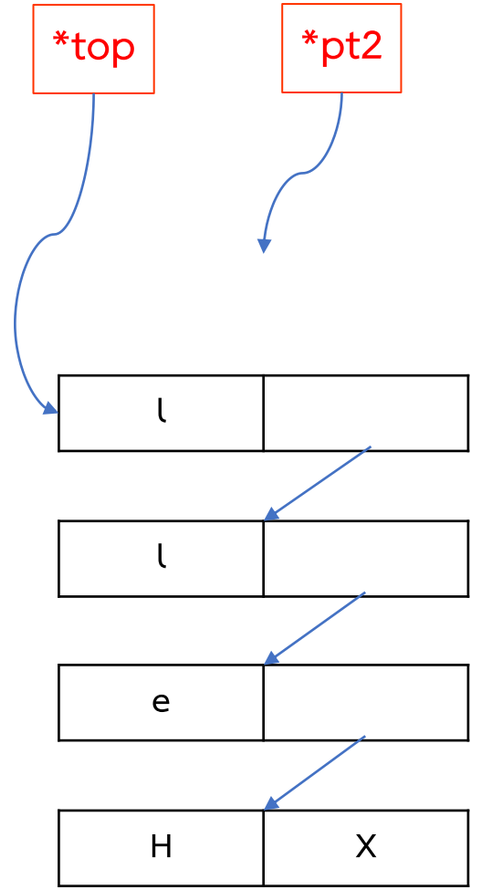
ขั้นตอนของการ pop()

1 อ่านข้อมูลที่อยู่ใน node แรก

2 สร้าง pointer ตัวที่ 2 เพื่อชี้ไปยัง node แรก

3 เปลี่ยน *top (*head) ให้ชี้ไปยัง node ที่สอง

4 ลบ node ที่ pointer ตัวที่ 2 ชี้อยู่



c=pop()

c='o'

Stack: Linked List

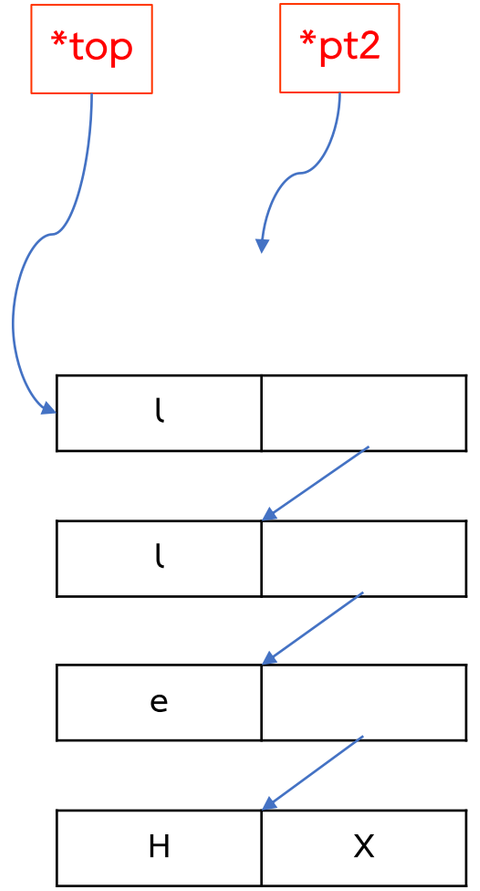
ขั้นตอนของการ pop()

1 อ่านข้อมูลที่อยู่ใน node แรก

2 สร้าง pointer ตัวที่ 2 เพื่อชี้ไปยัง node แรก

3 เปลี่ยน *top (*head) ให้ชี้ไปยัง node ที่สอง

4 ลบ node ที่ pointer ตัวที่ 2 ชี้อยู่



c=pop()

c=pop()

c='o'

Stack: Linked List

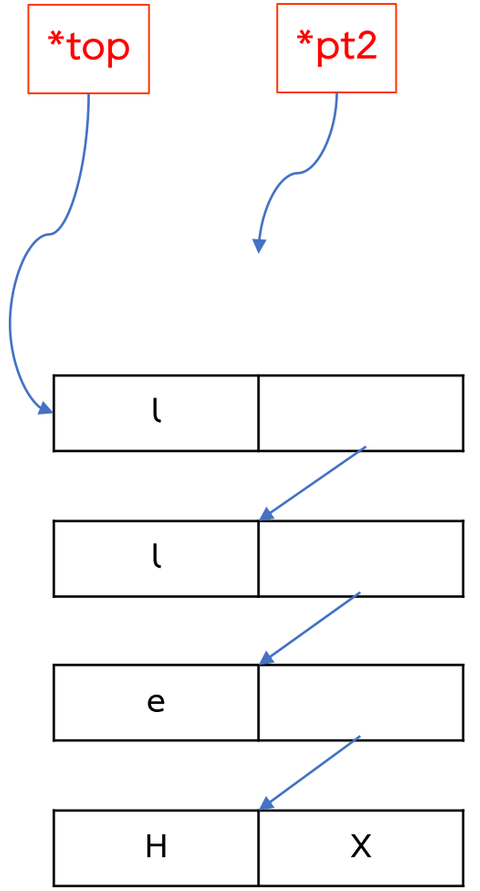
ขั้นตอนของการ pop()

1 อ่านข้อมูลที่อยู่ใน node แรก

2 สร้าง pointer ตัวที่ 2 เพื่อชี้ไปยัง node แรก

3 เปลี่ยน *top (*head) ให้ชี้ไปยัง node ที่สอง

4 ลบ node ที่ pointer ตัวที่ 2 ชี้อยู่



c=pop()

c=pop()

c='l'

Stack: Linked List

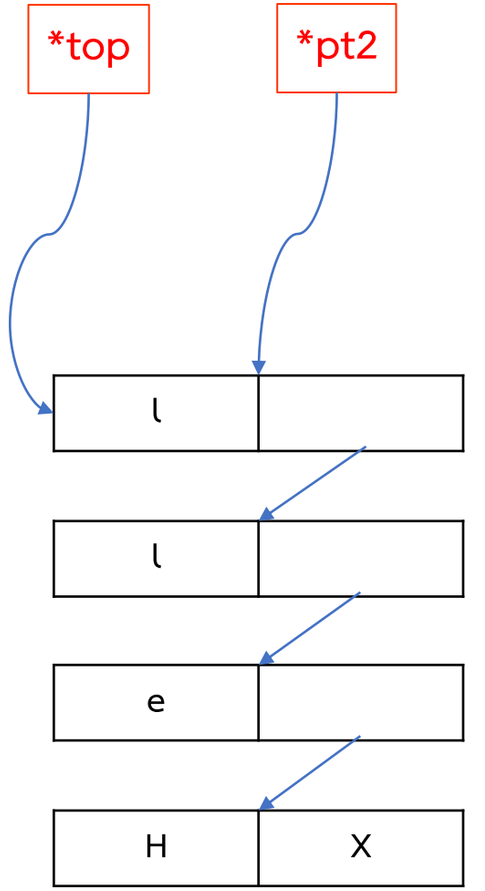
ขั้นตอนของการ pop()

1 อ่านข้อมูลที่อยู่ใน node แรก

2 สร้าง pointer ตัวที่ 2 เพื่อชี้ไปยัง node แรก

3 เปลี่ยน *top (*head) ให้ชี้ไปยัง node ที่สอง

4 ลบ node ที่ pointer ตัวที่ 2 ชี้อยู่



c=pop()

c=pop()

c='l'

Stack: Linked List

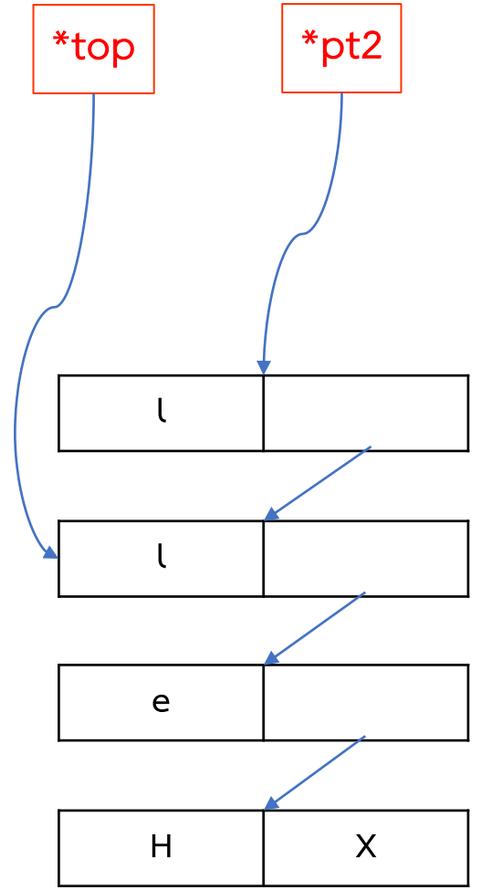
ขั้นตอนของการ pop()

1 อ่านข้อมูลที่อยู่ใน node แรก

2 สร้าง pointer ตัวที่ 2 เพื่อชี้ไปยัง node แรก

3 เปลี่ยน *top (*head) ให้ชี้ไปยัง node ที่สอง

4 ลบ node ที่ pointer ตัวที่ 2 ชี้อยู่



c=pop()

c=pop()

c='l'

Stack: Linked List

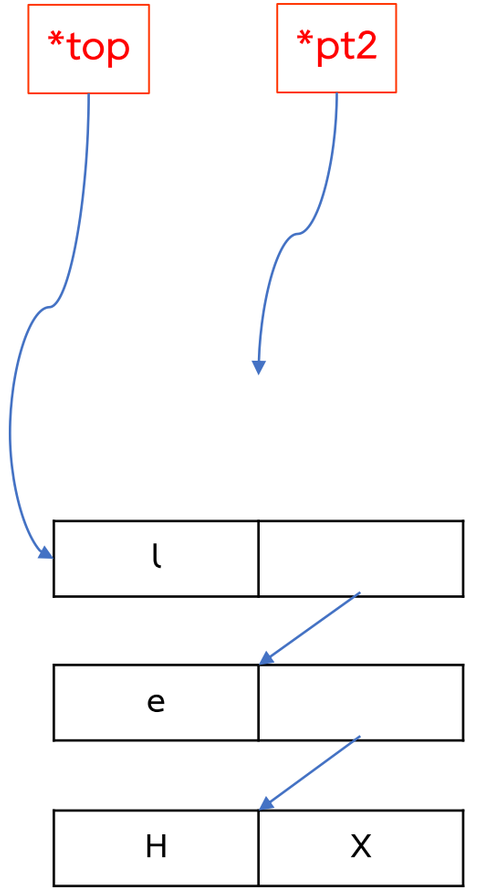
ขั้นตอนของการ pop()

1 อ่านข้อมูลที่อยู่ใน node แรก

2 สร้าง pointer ตัวที่ 2 เพื่อชี้ไปยัง node แรก

3 เปลี่ยน *top (*head) ให้ชี้ไปยัง node ที่สอง

4 ลบ node ที่ pointer ตัวที่ 2 ชี้อยู่



c=pop()

c=pop()

c='l'

Stack: Linked List

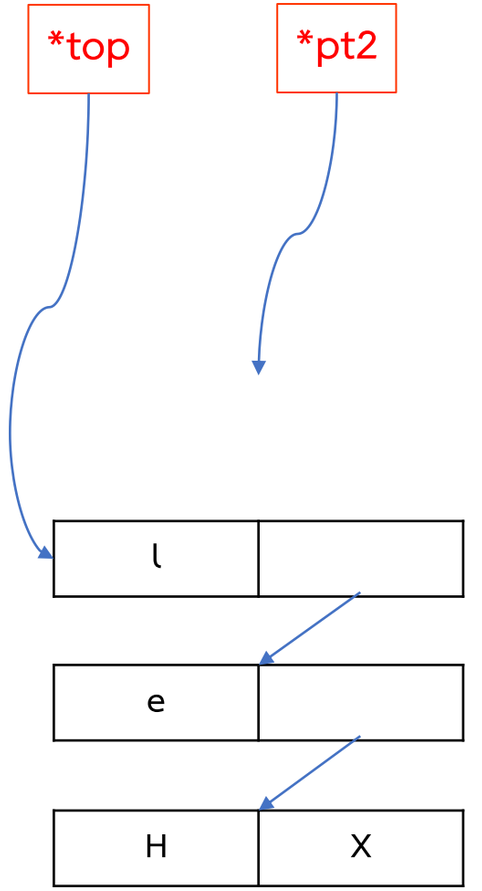
ขั้นตอนของการ pop()

1 อ่านข้อมูลที่อยู่ใน node แรก

2 สร้าง pointer ตัวที่ 2 เพื่อชี้ไปยัง node แรก

3 เปลี่ยน *top (*head) ให้ชี้ไปยัง node ที่สอง

4 ลบ node ที่ pointer ตัวที่ 2 ชี้อยู่



c=pop()

c=pop()

c='l'

Stack: Linked List

เปรียบเทียบกับ Array

- Array สร้างได้ง่ายกว่า แต่เหมาะกับข้อมูลขนาดเล็ก
- Linked list สร้างยากกว่า แต่ยืดหยุ่นกว่า
    - เปลี่ยนแปลงขนาดได้
    - เหมาะกับข้อมูลที่ไม่รู้ขนาดแน่นอน

ไม่ว่าจะสร้างด้วย Linked list หรือ Array ความซับซ้อนทางเวลาจะเท่ากัน

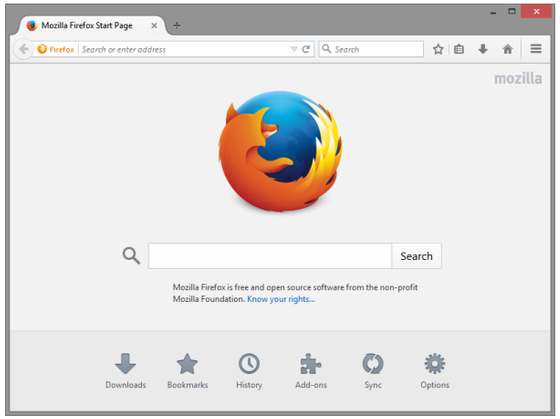
Operation	Array	Linked List
push()	O(1)	O(1)
pop()	O(1)	O(1)
peek()	O(1)	O(1)
isEmpty()	O(1)	O(1)
isFull()	O(1)	O(1)

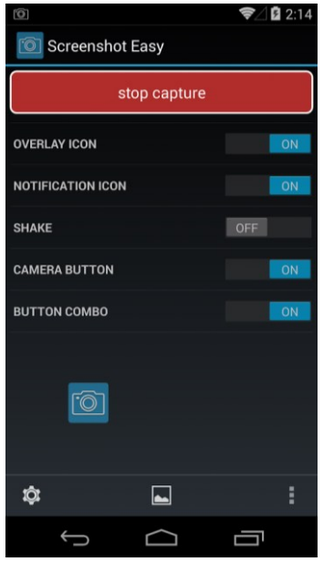
*แต่เมื่อโปรแกรมทำงานจบ ถ้าใช้ Linked list จะต้องเสียเวลาเพิ่มอีก O(N) เพื่อลบ node คืนหน่วยความจำ

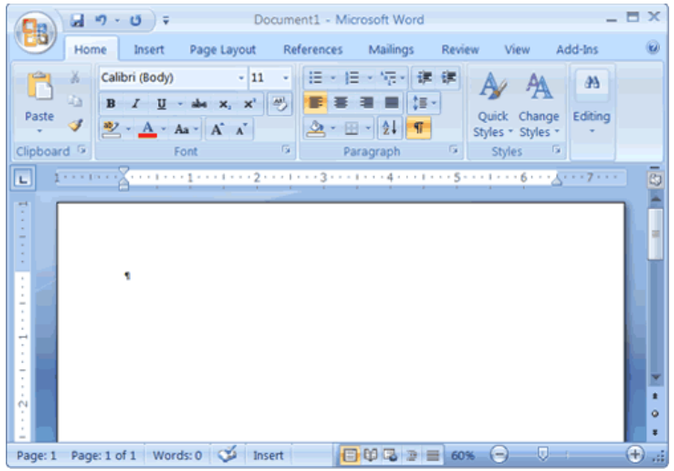
Stack: Application

Stack ใช้ทำอะไรได้บ้าง

งานที่จำเป็นต้องมีการ Track back ตัวอย่างเช่น







Stack: Application

Stack ใช้ทำอะไรได้บ้าง

ประมวลผลที่เกี่ยวข้องกับภาษาหรือไวยากรณ์

ตัวอย่างเช่นการตรวจสอบวงเล็บ

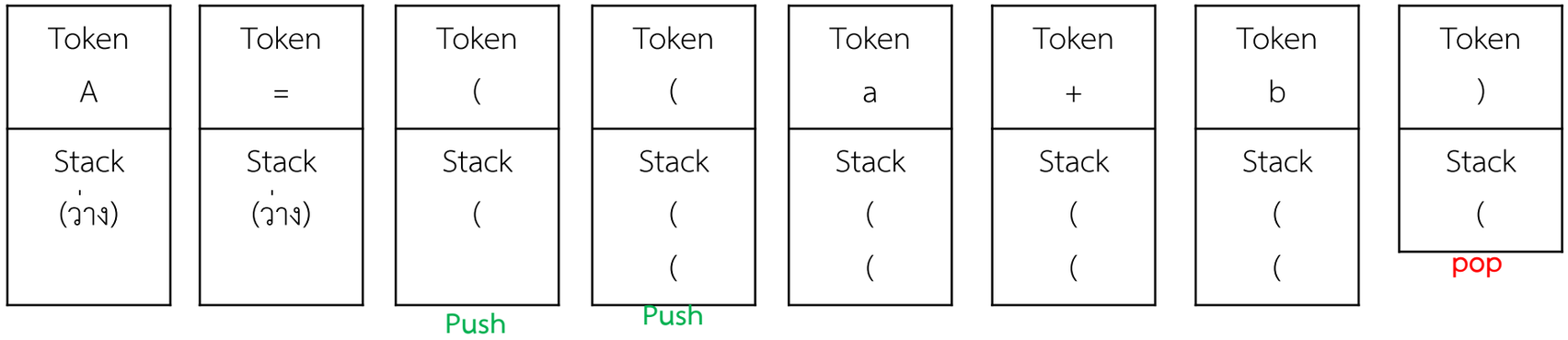
1) กวาดประโยคทีละตัวเริ่มจากซ้ายไปขวา
2) ถ้าเจอ '(' ให้ push ลง stack
3) ถ้าเจอ ')' ให้ pop ออกมาจาก stack หากไม่มีข้อมูลใน stack แสดงว่าใช้วงเล็บไม่ถูกต้อง
4) หากจบประโยคแล้ว stack ไม่มีข้อมูลแสดงว่าใช้วงเล็บได้ถูกต้อง แต่หาก stack ไม่ว่าง แสดงว่าใช้วงเล็บไม่ถูกต้อง

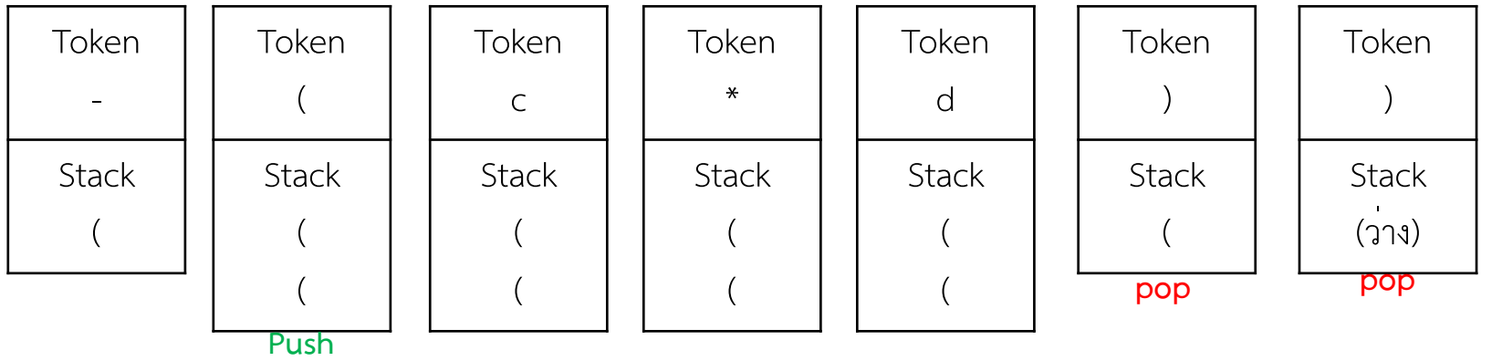
Stack: Application

Stack ใช้ทำอะไรได้บ้าง
ตัวอย่างเช่นการตรวจสอบวงเล็บ

1) กวาดประโยคทีละตัวเริ่มจากซ้ายไปขวา
2) ถ้าเจอ '(' ให้ push ลง stack
3) ถ้าเจอ ')' ให้ pop ออกมาจาก stack หากไม่มีข้อมูลใน stack แสดงว่าใช้วงเล็บไม่ถูกต้อง
4) หากจบประโยคแล้ว stack ไม่มีข้อมูลแสดงว่าใช้วงเล็บได้ถูกต้อง แต่หาก stack ไม่ว่าง แสดงว่าใช้วงเล็บไม่ถูกต้อง

A=((a+b) – (c * d))

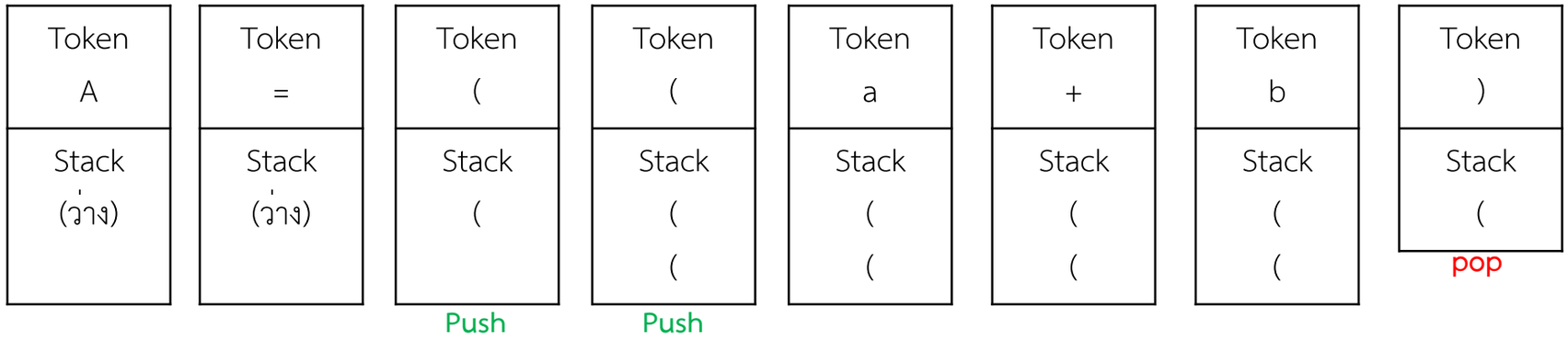


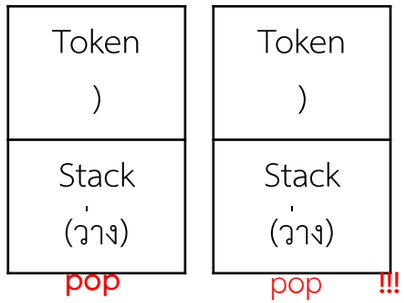


Stack ใช้ทำอะไรได้บ้าง
ตัวอย่างเช่นการตรวจสอบวงเล็บ

1) กวาดประโยคทีละตัวเริ่มจากซ้ายไปขวา
2) ถ้าเจอ '(' ให้ push ลง stack
3) ถ้าเจอ ')' ให้ pop ออกมาจาก stack หากไม่มีข้อมูลใน stack แสดงว่าใช้วงเล็บไม่ถูกต้อง
4) หากจบประโยคแล้ว stack ไม่มีข้อมูลแสดงว่าใช้วงเล็บได้ถูกต้อง แต่หาก stack ไม่ว่าง แสดงว่าใช้วงเล็บไม่ถูกต้อง

A=((a+b))) – (c * d))





!!! Stack ไม่มีข้อมูล จึง pop ไม่ได้

A=((a+b))) – (c * d))

compiler ใช้วิธีนี้ในการตรวจสอบ วงเล็บ
หรือหา source code ที่ไม่ถูกต้อง

Stack ใช้ทำอะไรได้บ้าง

ประมวลผลที่เกี่ยวข้องกับภาษาหรือไวยากรณ์

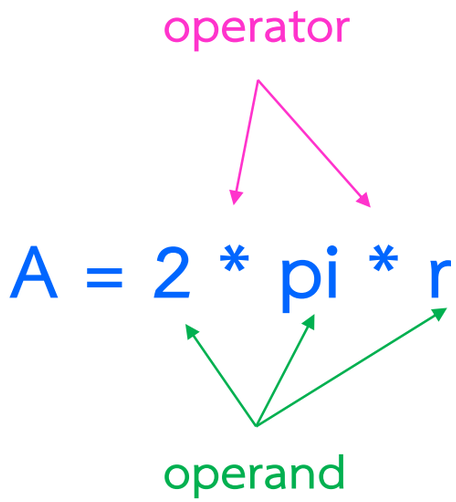
แปลงนิพจน์จาก Infix ไปเป็น Postfix

Stack: Infix to Postfix Conversion

นิพจน์ทางคณิตศาสตร์มีส่วนประกอบอยู่ 2 ชนิดคือ

Operator หรือตัวกระทำ เช่น + - * /

Operand หรือตัวถูกกระทำ เช่น a b c d หรือจำนวนเลข



การเขียนนิพจน์ทางคณิตศาสตร์มีวิธีวาง operator และ operand ได้ 3 วิธีคือ

Infix

Prefix

Postfix

Stack: Infix to Postfix Conversion

Infix จะเขียนโดยวาง operator ไว้ระหว่าง operand

$$A = 1 + 2 + 3 * 5 * 6 + 3 \wedge 2$$

เป็นวิธีเขียนที่เก่าที่สุด มีความกำกวมของลำดับการคำนวณ

เพื่อให้เข้าใจตรงกัน จึงต้องมีการนิยามลำดับความสำคัญของการคำนวณ (Precedence)

1) ยกกำลัง
2) คูณ หาร
3) บวก ลบ

หาก operator มีความสำคัญเท่ากัน ให้กระทำตัวที่อยู่ด้านซ้ายมือก่อน

สามารถใช้วงเล็บ ในการบังคับลำดับการคำนวณไม่ให้เป็นไปตามกฎนี้ได้

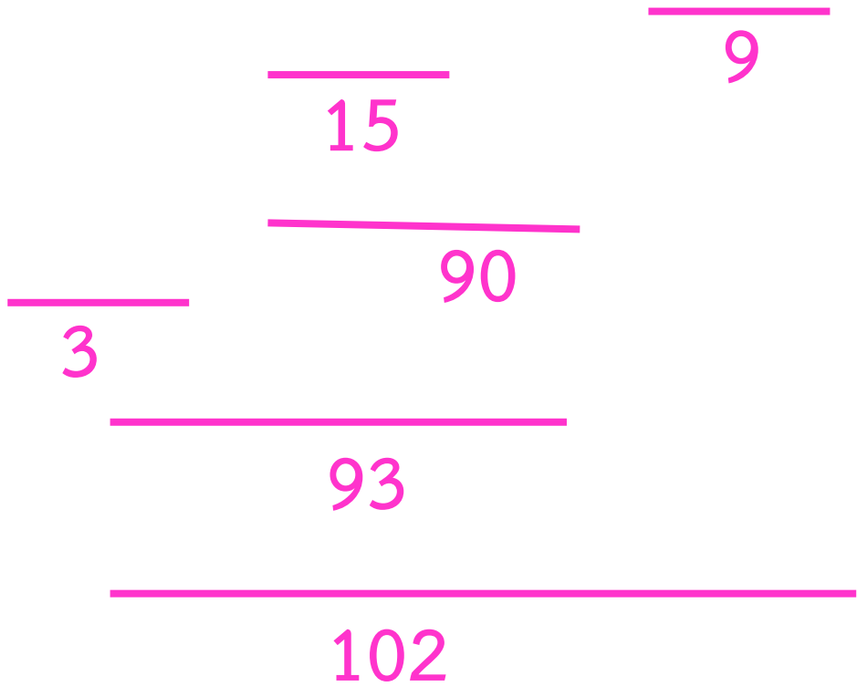
Stack: Infix to Postfix Conversion

Infix คือจะเขียนโดยวาง operator ไว้ระหว่าง operand

A= 1 + 2 + 3 * 5 * 6 + 3 ^ 2

1) ยกกำลัง
2) คูณ หาร
3) บวก ลบ

หาก operator มีความสำคัญเท่ากัน
ให้กระทำตัวที่อยู่ด้านซ้ายมือก่อน



คอมพิวเตอร์ ไม่ได้ใช้รูปแบบนี้ในการคำนวณทางคณิตศาสตร์
เพราะมันซับซ้อนเกินไปสำหรับเครื่องจักร

Stack: Infix to Postfix Conversion

Prefix จะเขียนโดยวาง operator ไว้หน้า operand

A = + 2 3

ไม่มีความกำกวม การคำนวณจะเรียงจากหน้าไปหลัง

คล้ายๆ ADD 2 , 3

Infix : A= 1 + 2 + 3 * 5 * 6 + 3 ^ 2

Posfix : A= + + 1 2 + * * 3 5 6 ^ 3 2

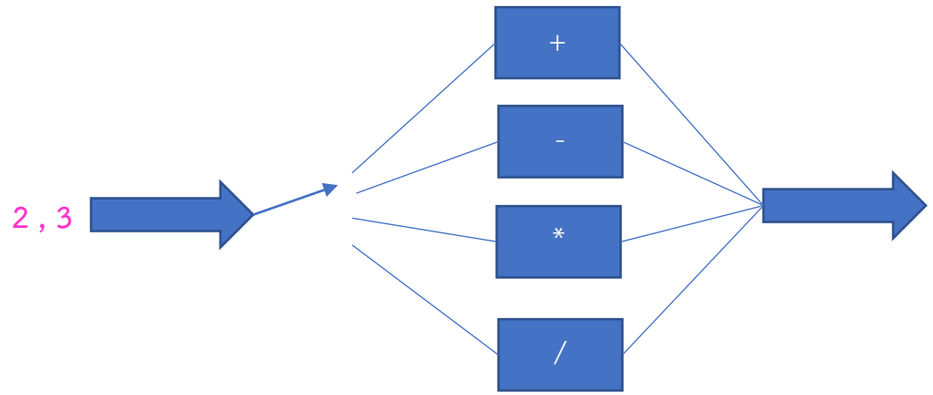
Stack: Infix to Postfix Conversion

Postfix จะเขียนโดยวาง operator ไว้หลัง operand

A = 2 3 +

ไม่มีความกำกวม การคำนวณจะเรียงจากหน้าไปหลัง

คล้ายๆ



Infix : A= 1 + 2 + 3 * 5 * 6 + 3 ^ 2

Posfix : A=1 2 + 3 5 * 6 * + 3 2 ^ +

Stack: Infix to Postfix Conversion

Compiler จะทำการแปลงนิพจน์ให้อยู่ในรูป prefix หรือ postfix ก่อนจึงจะคำนวณได้
โดยใช้ Algorithm ดังต่อไปนี้

Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

ยกกำลัง → คูณ หาร → บวก ลบ

Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

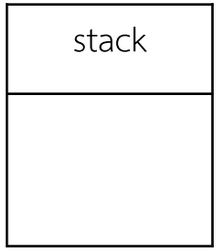
ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1



Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

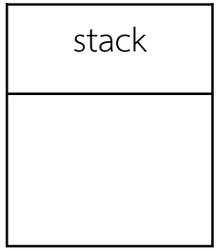
ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1



Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1

stack
+

Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1

stack
+

Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2

stack
+

Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2

stack
+

Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 +

stack

Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 +

stack
+

Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

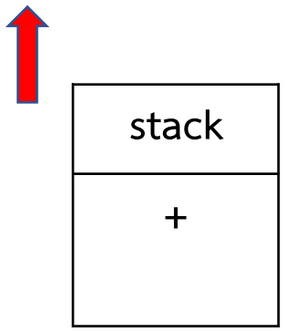
ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 +



Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

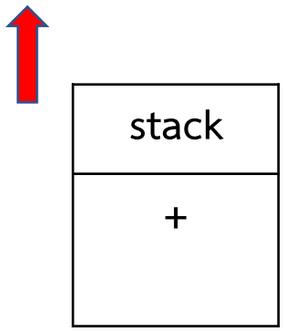
ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 + 3



Stack: Infix to Postfix Conversion

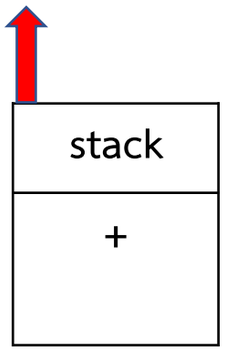
การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2



output

1 2 + 3

Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

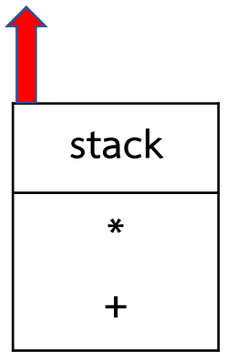
ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 + 3



Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

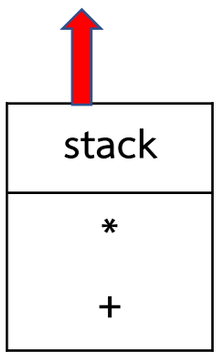
ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 + 3



Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

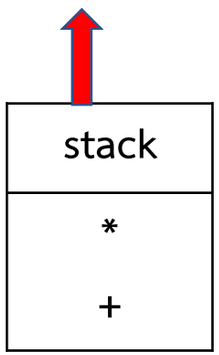
ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 + 3 5



Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

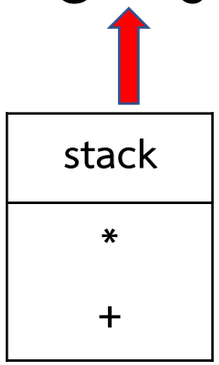
ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 + 3 5



Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

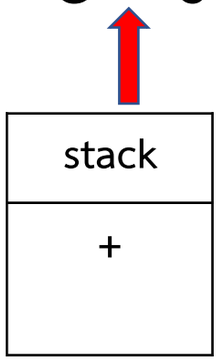
ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 + 3 5 *



Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

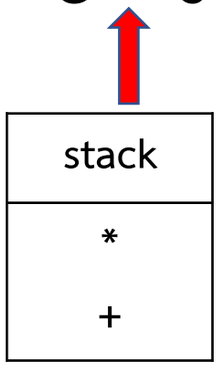
ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 + 3 5 *



Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

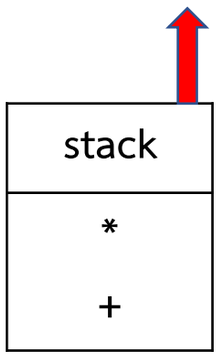
ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 + 3 5 *



Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

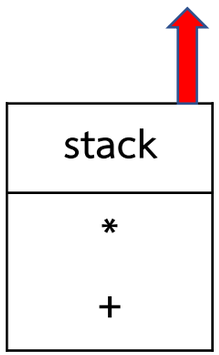
ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

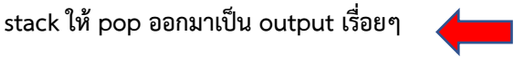
1 2 + 3 5 * 6



Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก



ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

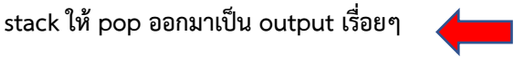
1 2 + 3 5 * 6

stack
*
+

Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก



ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 + 3 5 * 6 *

stack
+

Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

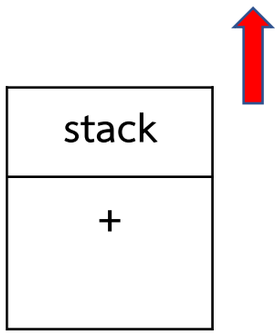
ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 + 3 5 * 6 *



Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

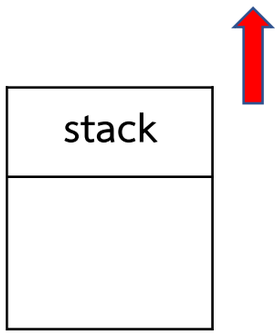
ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 + 3 5 * 6 * +



Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

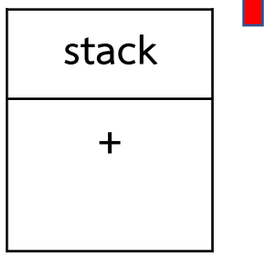
ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 + 3 5 * 6 * +



Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 + 3 5 * 6 * +

stack
+

Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 + 3 5 * 6 * + 3

stack
+

Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 + 3 5 * 6 * + 3

stack
+

Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 + 3 5 * 6 * + 3

stack
^
+

Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 + 3 5 * 6 * + 3

stack
^
+

Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 + 3 5 * 6 * + 3 2

stack
^
+

Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 + 3 5 * 6 * + 3 2 ^

stack
+

Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

1) เริ่มกวาดนิพจน์ทีละตัว เรียกว่า token โดยเริ่มจาก ซ้ายไปขวา
2) หาก token เป็น operand ให้ ส่งไปยัง output
3) หาก token เป็น operator และ stack ว่าง หรือ top คือ (ให้ push ลง stack
4) หาก token เป็น (ให้ push ลง stack
5) หาก token เป็น) ให้ pop stack ออกมาเป็น output เรื่อยๆจนกว่าจะเจอ (และกำจัดวงเล็บออก
6) หาก token มี precedence สูงกว่า top ของ stack ให้ push ลง stack
7) หาก token มี precedence เทียบเท่ากับ top ของ stack ให้ pop ออกมาเป็น output และ push token ลงไปแทน
8) หาก token มี precedence ต่ำกว่า top ของ stack ให้ pop ออกมาเป็น output เรื่อยๆ และกลับไปทำข้อ 6
9) หากจบนิพจน์แล้ว ให้ pop ออกมาเป็น output ให้หมด หากมีลงเล็บให้ลบออก

ยกกำลัง → คูณ หาร → บวก ลบ

Input

1 + 2 + 3 * 5 * 6 + 3 ^ 2

output

1 2 + 3 5 * 6 * + 3 2 ^ +

stack

Stack: Infix to Postfix Conversion

การแปลง Infix ไปเป็น postfix

Token	Stack	Output
1	ว่าง	1
+	+	1
2	+	1 2
+	+	1 2 +
3	+	1 2 + 3
*	*+	1 2 + 3
5	*+	1 2 + 3 5
*	*+	1 2 + 3 5 *
6	*+	1 2 + 3 5 * 6
+	+	1 2 + 3 5 * 6 * +
3	+	1 2 + 3 5 * 6 * + 3
^	^+	1 2 + 3 5 * 6 * + 3
2	^+	1 2 + 3 5 * 6 * + 3 2
	+	1 2 + 3 5 * 6 * + 3 2 ^
	ว่าง	1 2 + 3 5 * 6 * + 3 2 ^ +

Infix : A= 1 + 2 + 3 * 5 * 6 + 3 ^ 2

Prefix : A=1 2 + 3 5 * 6 * + 3 2 ^ +

Stack: Infix to Postfix Conversion

Token	Stack	Output	หมายเหตุ
(	(		
A	(	A	
+	+(	A	
B	+(	AB	
)	(	AB+	
	ว่าง	AB+	กำจัดวงเล็บ
*	*	AB+	
(	(*	AB+	
C	(*	AB+C	
^	^(*	AB+C	
(	(^(*	AB+C	
D	(^(*	AB+CD	
-	-(^(*	AB+CD	
E	-(^(*	AB+CDE	
)	(^(*	AB+CDE-	
	^(*	AB+CDE-	กำจัดวงเล็บ
*	(*	AB+CDE-^	
	(	AB+CDE-^	
F	*(*	AB+CDE-^F	
)	)*	AB+CDE-^F*	
	*	AB+CDE-^F*	กำจัดวงเล็บ
-	*	AB+CDE-^F*	
	-	AB+CDE-^F**	
G	-	AB+CDE-^F**G	
		AB+CDE-^F**G-	pop ข้อมูลที่ค้าง

Infix : (A+B)*(C^(D-E)*F)-G

Prefix : AB+CDE-^F**G-

Stack: Infix to Postfix Conversion

การแปลภาษา C จะมี operator เพิ่ม และจะมีกฎ Associativity เพิ่มขึ้นมา

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

Infix : A= 1 + 2 + 3 * 5 * 6 + 3 ^ 2

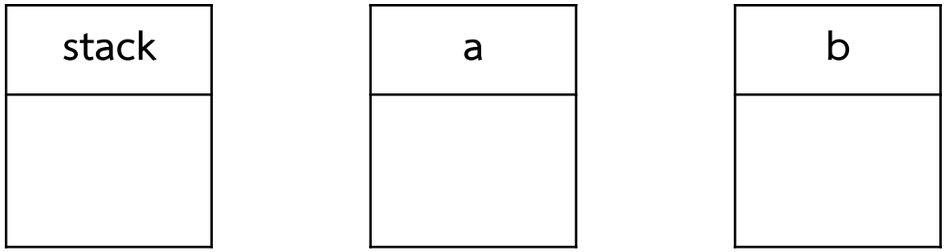
Posfix : A=1 2 + 3 5 * 6 * + 3 2 ^ +

! ไม่ยาก !

ใช้ตัวแปร 2 ตัวและ stack 1 ตัว

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

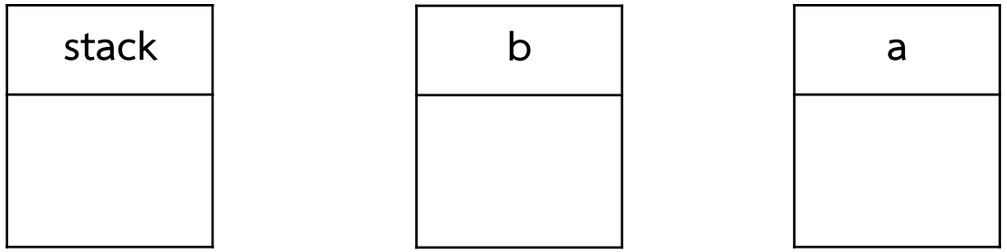
1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ ค่าที่อยู่ใน stack คือคำตอบ



การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

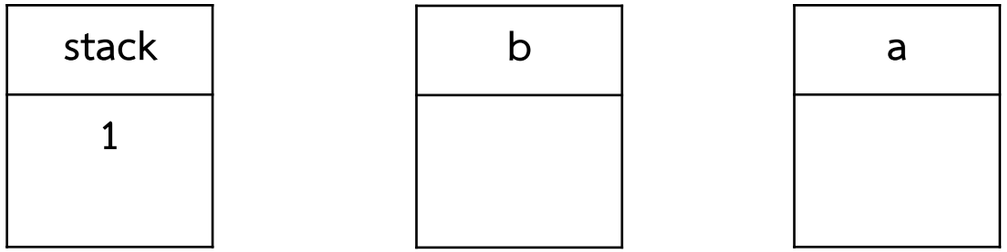
1 2 + 3 5 * 6 * + 3 2 ^ +



การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

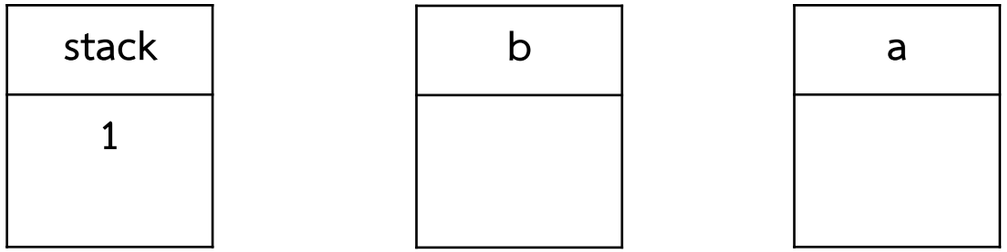
1 2 + 3 5 * 6 * + 3 2 ^ +



การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +



การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack
2
1

b

a

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack
2
1

b

a

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack
1

b

a
2

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack

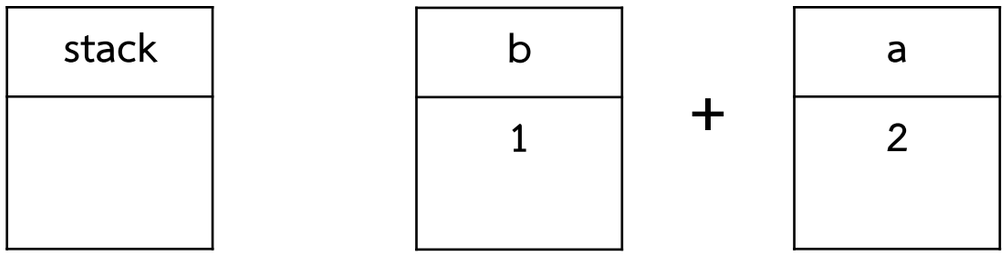
b
1

a
2

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

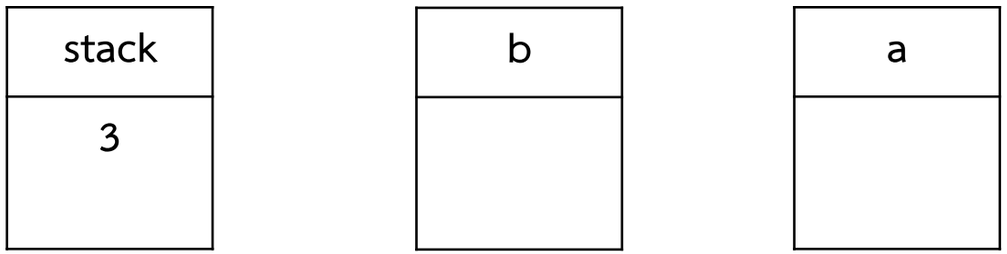
1 2 + 3 5 * 6 * + 3 2 ^ +



การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

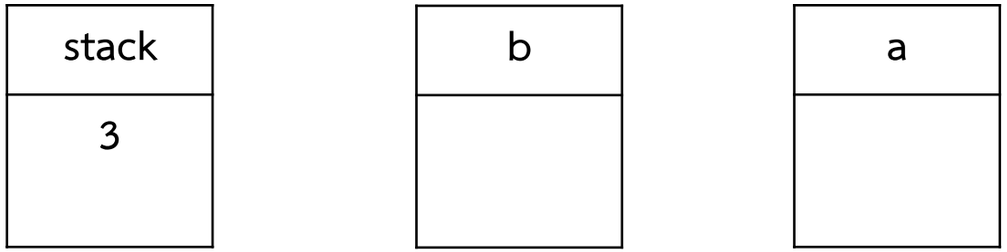
1 2 + 3 5 * 6 * + 3 2 ^ +



การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +



การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack
3
3

b

a

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack
3
3

b

a

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack
5
3
3

b

a

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack
5
3
3

b

a

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack
3

b
3

a
5

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack
15
3

b

a

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack
15
3

b

a

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack
6
15
3

b

a

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack
6
15
3

b

a

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack
3

b
15

a
6

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack
90
3

b

a

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack
90
3

b

a

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack

b
3

a
90

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack
93

b

a

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack
93

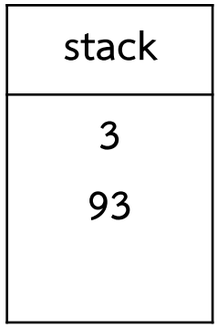
b

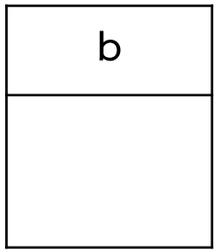
a

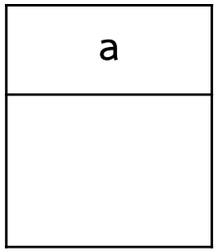
การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +







การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack
2
3
93

b

a

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack
2
3
93

b

a

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack
93

b
3

a
2

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack
9
93

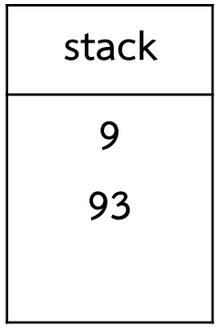
b

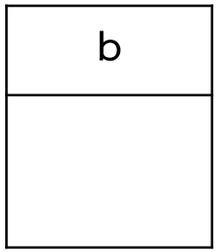
a

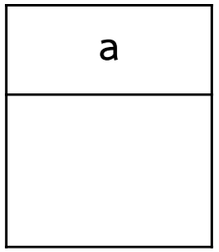
การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +







การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ

1 2 + 3 5 * 6 * + 3 2 ^ +

stack

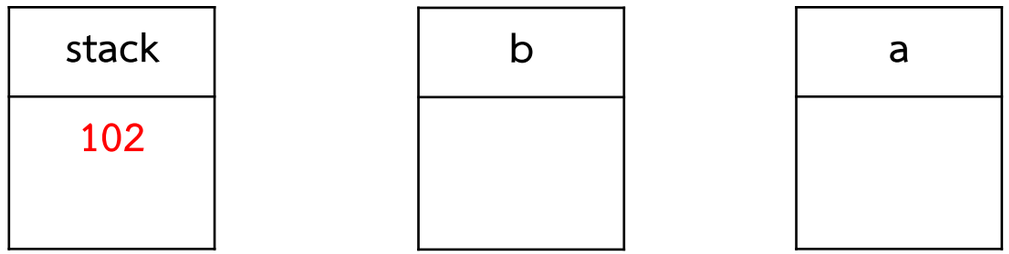
b
93

a
9

การหาคำตอบของนิพจน์ที่อยู่ในรูปของ Postfix

1) เริ่มกวาด Postfix จากซ้ายไปขวา
2) หาก token เป็น operand ให้ push ลง stack
3) หาก token เป็น operator ให้ pop ครั้งที่ 1 ใส่ตัวแปร a และ pop อีก 1 ครั้งใส่ตัวแปร b
4) คำนวณ b token a และ push คำตอบ กลับลง stack
5) ทำซ้ำจนครบ ค่าที่อยู่ใน stack คือคำตอบ

1 2 + 3 5 * 6 * + 3 2 ^ +



Stack: Application

Stack ใช้ทำอะไรได้บ้าง

ใช้ในการประมวลผล function call

Stack: Application

Stack ใช้ทำอะไรได้บ้าง

```
#include<stdio.h>
float calArea(float r);
float square(float area);
void main(){
   float r=10.0;
   float area=0;
   area=calArea(r);
   printf("%f",area);
}
float calArea(float r)
{       float PI=3.14129;
        float area=0;
        area= PI * square(r);
        return area;
}
float square(float area){
        float power=0;
        power=area*area;
        return power;
}
```

Stack: Application

การคอมไฟล์ โปรแกรม

```
#include<stdio.h>
float calArea(float r);
float square(float area);
void main(){
   float r=10.0;
   float area=0;
   area=calArea(r);
   printf("%f",area);
}
float calArea(float r)
{       float PI=3.14129;
        float area=0;
        area= PI * square(r);
        return area;
}
float square(float area){
        float power=0;
        power=area*area;
        return power;
}
```

Address=xxxxxx

calArea()

Stack: Application

การคอมไฟล์ โปรแกรม

```
#include<stdio.h>
float calArea(float r);
float square(float area);
void main(){
   float r=10.0;
   float area=0;
   area=calArea(r);
   printf("%f",area);
}
float calArea(float r)
{       float PI=3.14129;
        float area=0;
        area= PI * square(r);
        return area;
}
float square(float area){
        float power=0;
        power=area*area;
        return power;
}
```

Stack: Application

การคอมไฟล์ โปรแกรม

```
#include<stdio.h>
float calArea(float r);
float square(float area);
void main(){
   float r=10.0;
   float area=0;
   area=calArea(r);
   printf("%f",area);
}
float calArea(float r)
{       float PI=3.14129;
        float area=0;
        area= PI * square(r);
        return area;
}
float square(float area){
        float power=0;
        power=area*area;
        return power;
}
```

Address=xxxxxx
calArea()

Address=xxxxxx
square()

Address=xxxxxx
main()
84 D2 74 3C 83 E1 01 74 2B B9
01 00 00 00 48 83 C0 01 0F B6
10 80 FA 20 7E E6 41 89 C8 41
83 F0 01 80 FA 22 41 0F 44
C8.....

Stack: Application

การคอมไฟล์ โปรแกรม

```
#include<stdio.h>
float calArea(float r);
float square(float area);
void main(){
   float r=10.0;
   float area=0;
   area=calArea(r);
   printf("%f",area);
}
float calArea(float r)
{       float PI=3.14129;
        float area=0;
        area= PI * square(r);
        return area;
}
float square(float area){
        float power=0;
        power=area*area;
        return power;
}
```

Address=xxxxxx
calArea()
C3 0F B7 44 24 60 E9 05 FF
FF FF 0F 1F 44 00 00 48 8B
35 29 2F 00 00 BF 01 00 00
00 8B 06 83 F8 01 0F 85 EF
.....

Address=xxxxxx
square()

Address=xxxxxx
main()
84 D2 74 3C 83 E1 01 74 2B B9
01 00 00 00 48 83 C0 01 0F B6
10 80 FA 20 7E E6 41 89 C8 41
83 F0 01 80 FA 22 41 0F 44
C8.....

Stack: Application

การคอมไฟล์ โปรแกรม

```
#include<stdio.h>
float calArea(float r);
float square(float area);
void main(){
   float r=10.0;
   float area=0;
   area=calArea(r);
   printf("%f",area);
}
float calArea(float r)
{      float PI=3.14129;
       float area=0;
       area= PI * square(r);
       return area;
}
float square(float area){
       float power=0;
       power=area*area;
       return power;
}
```

Address=xxxxxx
calArea()
C3 0F B7 44 24 60 E9 05 FF
FF FF 0F 1F 44 00 00 48 8B
35 29 2F 00 00 BF 01 00 00
00 8B 06 83 F8 01 0F 85 EF
.....

Address=xxxxxx
square()
48 8B 15 19 2F 00 00 48 8B
0D 02 2F 00 00 C7 06 01 00
00 00 E8 7F 16 00 00 E9 8C
FD FF FF 45 31 ED E9 E0 FE
....

Address=xxxxxx
main()
84 D2 74 3C 83 E1 01 74 2B B9
01 00 00 00 48 83 C0 01 0F B6
10 80 FA 20 7E E6 41 89 C8 41
83 F0 01 80 FA 22 41 0F 44
C8.....

Stack: Application

การคอมไฟล์ โปรแกรม

```
#include<stdio.h>
float calArea(float r);
float square(float area);
void main(){
   float r=10.0;
   float area=0;
   area=calArea(r);
   printf("%f",area);
}
float calArea(float r)
{       float PI=3.14129;
        float area=0;
        area= PI * square(r);
        return area;
}
float square(float area){
        float power=0;
        power=area*area;
        return power;
}
```

Address=xxxxxx
calArea()
C3 0F B7 44 24 60 E9 05 FF
FF FF 0F 1F 44 00 00 48 8B
35 29 2F 00 00 BF 01 00 00
00 8B 06 83 F8 01 0F 85 EF
.....

Address=xxxxxx
square()
48 8B 15 19 2F 00 00 48 8B
0D 02 2F 00 00 C7 06 01 00
00 00 E8 7F 16 00 00 E9 8C
FD FF FF 45 31 ED E9 E0 FE
....

Address= 100H
main()
84 D2 74 3C 83 E1 01 74 2B B9
01 00 00 00 48 83 C0 01 0F B6
10 80 FA 20 7E E6 41 89 C8 41
83 F0 01 80 FA 22 41 0F 44
C8.....

Stack: Application

การคอมไฟล์ โปรแกรม

```
#include<stdio.h>
float calArea(float r);
float square(float area);
void main(){
   float r=10.0;
   float area=0;
   area=calArea(r);
   printf("%f",area);
}
float calArea(float r)
{      float PI=3.14129;
       float area=0;
       area= PI * square(r);
       return area;
}
float square(float area){
       float power=0;
       power=area*area;
       return power;
}
```

Address= 200H
calArea()
C3 0F B7 44 24 60 E9 05 FF
FF FF 0F 1F 44 00 00 48 8B
35 29 2F 00 00 BF 01 00 00
00 8B 06 83 F8 01 0F 85 EF
.....

Address= 300H
square()
48 8B 15 19 2F 00 00 48 8B
0D 02 2F 00 00 C7 06 01 00
00 00 E8 7F 16 00 00 E9 8C
FD FF FF 45 31 ED E9 E0 FE
....

Address= 100H
main()
84 D2 74 3C 83 E1 01 74 2B B9
01 00 00 00 48 83 C0 01 0F B6
10 80 FA 20 7E E6 41 89 C8 41
83 F0 01 80 FA 22 41 0F 44
C8.....

Stack: Application

การคอมไฟล์ โปรแกรม

```
        #include<stdio.h>
        float calArea(float r);
        float square(float area);
100     void main(){
110        float r=10.0;
120        float area=0;
130        area=calArea(r);
140        printf("%f",area);
        }
200     float calArea(float r)
210     {       float PI=3.14129;
220             float area=0;
230             area= PI * square(r);
240             return area;
        }
300     float square(float area){
310             float power=0;
320             power=area*area;
330             return power;
        }
```

Address= 100H
main()
84 D2 74 3C 83 E1 01 74 2B B9
01 00 00 00 48 83 C0 01 0F B6
10 80 FA 20 7E E6 41 89 C8 41
83 F0 01 80 FA 22 41 0F 44
C8.....

Address= 200H
calArea()
C3 0F B7 44 24 60 E9 05 FF FF
FF 0F 1F 44 00 00 48 8B 35 29
2F 00 00 BF 01 00 00 00 8B 06
83 F8 01 0F 85 EF .....

Address= 300H
square()
48 8B 15 19 2F 00 00 48 8B 0D
02 2F 00 00 C7 06 01 00 00 00
E8 7F 16 00 00 E9 8C FD FF FF
45 31 ED E9 E0 FE ....

Stack: Application

การรันฟังก์ชัน

```
#include<stdio.h>
float calArea(float r);
float square(float area);
100  void main(){
110     float r=10.0;
120     float area=0;
130     area=calArea(r);
140     printf("%f",area);
     }
200  float calArea(float r)
210  {       float PI=3.14129;
220          float area=0;
230          area= PI * square(r);
240          return area;
     }
300  float square(float area){
310          float power=0;
320          power=area*area;
330          return power;
     }
```

function main()
Address= 100

Stack Frame

Stack: Application

การรันฟังก์ชัน

```
                #include<stdio.h>
                float calArea(float r);
                float square(float area);
100             void main(){
110                float r=10.0;
120                float area=0;
130                area=calArea(r);
140                printf("%f",area);
                }
200             float calArea(float r)
210             {       float PI=3.14129;
220                     float area=0;
230                     area= PI * square(r);
240                     return area;
                }
300             float square(float area){
310                     float power=0;
320                     power=area*area;
330                     return power;
                }
```

function main()

Address= 120

r=10.0

area=0

Stack Frame

Stack: Application

การรันฟังก์ชัน

```
                #include<stdio.h>
                float calArea(float r);
                float square(float area);
100             void main(){
110                float r=10.0;
120                float area=0;
130                area=calArea(r);
140                printf("%f",area);
                }
200             float calArea(float r)
210             {       float PI=3.14129;
220                     float area=0;
230                     area= PI * square(r);
240                     return area;
                }
300             float square(float area){
310                     float power=0;
320                     power=area*area;
330                     return power;
                }
```

function main()

Address= 130

r=10.0

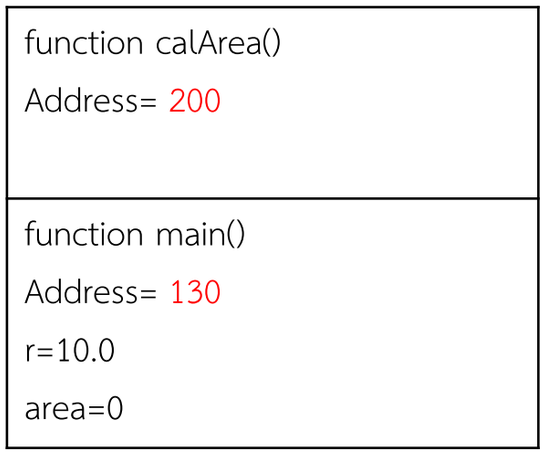
area=0

Stack Frame

Stack: Application

การรันฟังก์ชัน

```
#include<stdio.h>
float calArea(float r);
float square(float area);
100  void main(){
110     float r=10.0;
120     float area=0;
130     area=calArea(r);
140     printf("%f",area);
     }
200  float calArea(float r)
210  {       float PI=3.14129;
220          float area=0;
230          area= PI * square(r);
240          return area;
     }
300  float square(float area){
310          float power=0;
320          power=area*area;
330          return power;
     }
```



Stack Frame

Stack: Application

การรันฟังก์ชัน

```
                #include<stdio.h>
                float calArea(float r);
                float square(float area);
100             void main(){
110                float r=10.0;
120                float area=0;
130                area=calArea(r);
140                printf("%f",area);
                }
200             float calArea(float r)
210             {       float PI=3.14129;
220                     float area=0;
230                     area= PI * square(r);
240                     return area;
                }
300             float square(float area){
310                     float power=0;
320                     power=area*area;
330                     return power;
                }
```

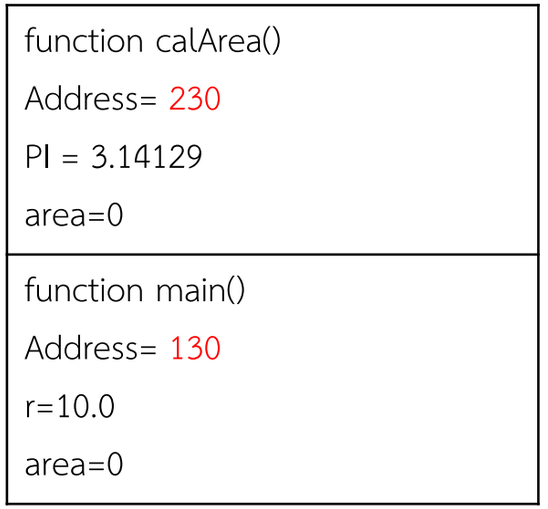
function calArea()
Address= 220
PI = 3.14129
area=0
function main()
Address= 130
r=10.0
area=0

Stack Frame

Stack: Application

การรันฟังก์ชัน

```
                #include<stdio.h>
                float calArea(float r);
                float square(float area);
100             void main(){
110                float r=10.0;
120                float area=0;
130                area=calArea(r);
140                printf("%f",area);
                }
200             float calArea(float r)
210             {       float PI=3.14129;
220                     float area=0;
230                     area= PI * square(r);
240                     return area;
                }
300             float square(float area){
310                     float power=0;
320                     power=area*area;
330                     return power;
                }
```

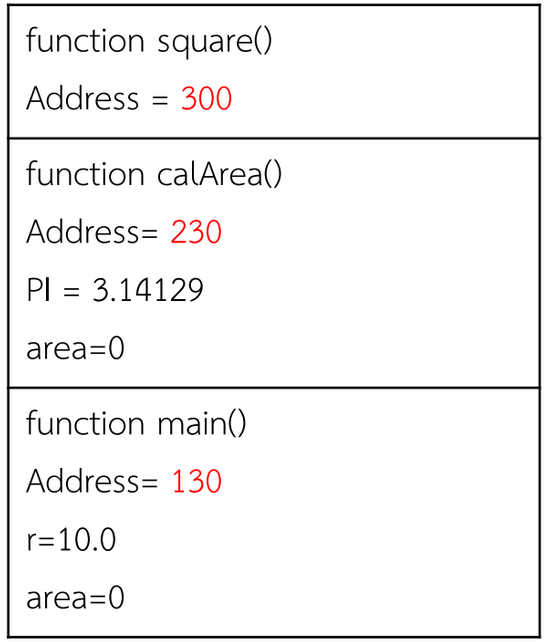


Stack Frame

Stack: Application

การรันฟังก์ชัน

```
#include<stdio.h>
float calArea(float r);
float square(float area);
100  void main(){
110     float r=10.0;
120     float area=0;
130     area=calArea(r);
140     printf("%f",area);
     }
200  float calArea(float r)
210  {       float PI=3.14129;
220          float area=0;
230          area= PI * square(r);
240          return area;
     }
300  float square(float area){
310          float power=0;
320          power=area*area;
330          return power;
     }
```



Stack Frame

Stack: Application

การรันฟังก์ชัน

```
#include<stdio.h>
float calArea(float r);
float square(float area);
100  void main(){
110     float r=10.0;
120     float area=0;
130     area=calArea(r);
140     printf("%f",area);
    }
200  float calArea(float r)
210  {      float PI=3.14129;
220         float area=0;
230         area= PI * square(r);
240         return area;
    }
300  float square(float area){
310         float power=0;
320         power=area*area;
330         return power;
    }
```

function square()
Address = 310
power = 0
function calArea()
Address= 230
PI = 3.14129
area=0
function main()
Address= 130
r=10.0
area=0

Stack Frame

Stack: Application

การรันฟังก์ชัน

```
#include<stdio.h>
float calArea(float r);
float square(float area);
100  void main(){
110     float r=10.0;
120     float area=0;
130     area=calArea(r);
140     printf("%f",area);
    }
200  float calArea(float r)
210  {      float PI=3.14129;
220         float area=0;
230         area= PI * square(r);
240         return area;
    }
300  float square(float area){
310         float power=0;
320         power=area*area;
330         return power;
    }
```

function square()
Address = 320
power = 100
function calArea()
Address= 230
PI = 3.14129
area=0
function main()
Address= 130
r=10.0
area=0

Stack Frame

Stack: Application

การรันฟังก์ชัน

```
#include<stdio.h>
float calArea(float r);
float square(float area);
100  void main(){
110     float r=10.0;
120     float area=0;
130     area=calArea(r);
140     printf("%f",area);
     }
200  float calArea(float r)
210  {       float PI=3.14129;
220          float area=0;
230          area= PI * square(r);
240          return area;
     }
300  float square(float area){
310          float power=0;
320          power=area*area;
330          return power;
     }
```

function square()
Address = 330
power = 100
function calArea()
Address= 230
PI = 3.14129
area=0
function main()
Address= 130
r=10.0
area=0

Stack Frame

Stack: Application

การรันฟังก์ชัน

```
#include<stdio.h>
float calArea(float r);
float square(float area);
100  void main(){
110     float r=10.0;
120     float area=0;
130     area=calArea(r);
140     printf("%f",area);
    }
200  float calArea(float r)
210  {      float PI=3.14129;
220         float area=0;
230         area= PI * square(r);
240         return area;
    }
300  float square(float area){
310         float power=0;
320         power=area*area;
330         return power;
    }
```

function calArea()
Address= 230
PI = 3.14129
area=0
function main()
Address= 130
r=10.0
area=0

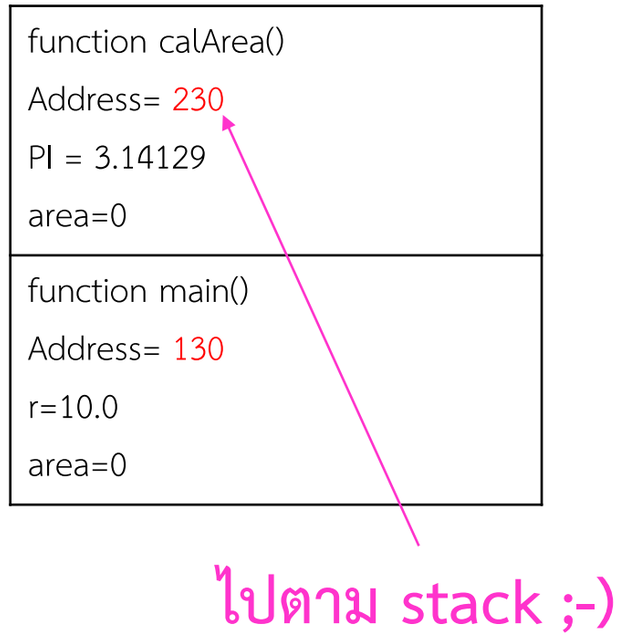
สิ้นสุดหน่วยความจำแล้ว ไปไหนต่อ ?

Stack Frame

Stack: Application

การรันฟังก์ชัน

```
        #include<stdio.h>
        float calArea(float r);
        float square(float area);
100     void main(){
110        float r=10.0;
120        float area=0;
130        area=calArea(r);
140        printf("%f",area);
        }
200     float calArea(float r)
210     {       float PI=3.14129;
220             float area=0;
230             area= PI * square(r);
240             return area;
        }
300     float square(float area){
310             float power=0;
320             power=area*area;
330             return power;
        }
```



Stack Frame

Stack: Application

การรันฟังก์ชัน

```
                #include<stdio.h>
                float calArea(float r);
                float square(float area);
100             void main(){
110                float r=10.0;
120                float area=0;
130                area=calArea(r);
140                printf("%f",area);
                }
200             float calArea(float r)
210             {       float PI=3.14129;
220                     float area=0;
230                     area= PI * square(r);
240                     return area;
                }
300             float square(float area){
310                     float power=0;
320                     power=area*area;
330                     return power;
                }
```

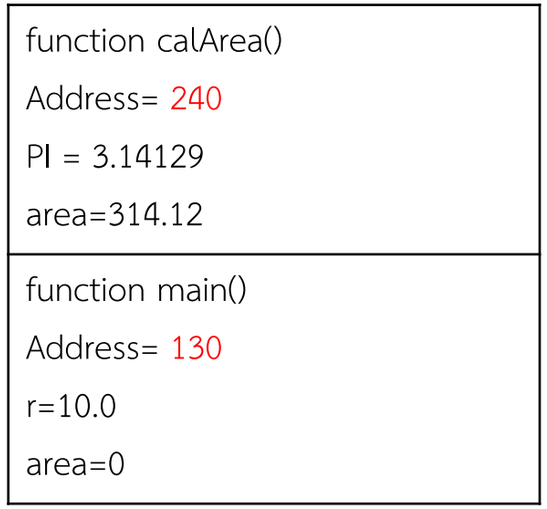
function calArea()
Address= 230
PI = 3.14129
area=0
function main()
Address= 130
r=10.0
area=0

Stack Frame

Stack: Application

การรันฟังก์ชัน

```
#include<stdio.h>
float calArea(float r);
float square(float area);
100  void main(){
110     float r=10.0;
120     float area=0;
130     area=calArea(r);
140     printf("%f",area);
     }
200  float calArea(float r)
210  {       float PI=3.14129;
220          float area=0;
230          area= PI * square(r);
240          return area;
     }
300  float square(float area){
310          float power=0;
320          power=area*area;
330          return power;
     }
```

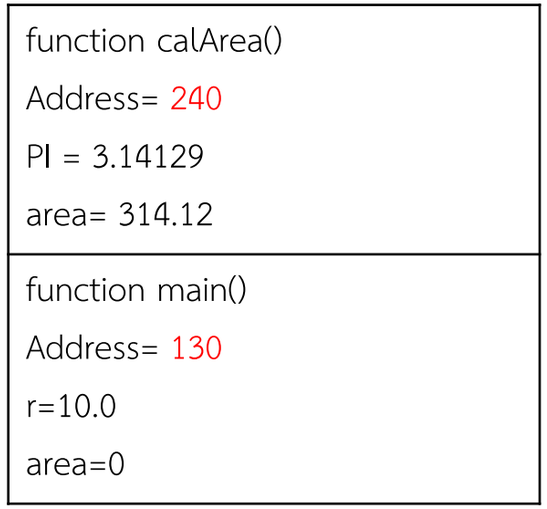


Stack Frame

Stack: Application

การรันฟังก์ชัน

```
#include<stdio.h>
float calArea(float r);
float square(float area);
100  void main(){
110     float r=10.0;
120     float area=0;
130     area=calArea(r);
140     printf("%f",area);
     }
200  float calArea(float r)
210  {       float PI=3.14129;
220          float area=0;
230          area= PI * square(r);
240          return area;
     }
300  float square(float area){
310          float power=0;
320          power=area*area;
330          return power;
     }
```

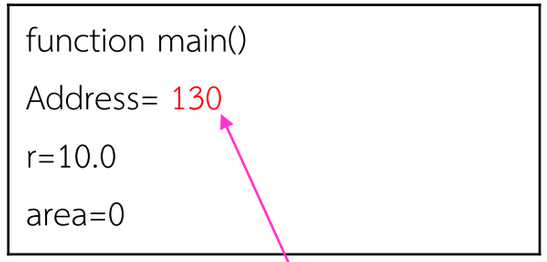


Stack Frame

Stack: Application

การรันฟังก์ชัน

```
        #include<stdio.h>
        float calArea(float r);
        float square(float area);
100     void main(){
110        float r=10.0;
120        float area=0;
130        area=calArea(r);
140        printf("%f",area);
        }
200     float calArea(float r)
210     {       float PI=3.14129;
220             float area=0;
230             area= PI * square(r);
240             return area;
        }
300     float square(float area){
310             float power=0;
320             power=area*area;
330             return power;
        }
```



ไปตาม stack ;-)

Stack Frame

Stack: Application

การรันฟังก์ชัน

```
                #include<stdio.h>
                float calArea(float r);
                float square(float area);
100             void main(){
110                float r=10.0;
120                float area=0;
130                area=calArea(r);
140                printf("%f",area);
                }
200             float calArea(float r)
210             {       float PI=3.14129;
220                     float area=0;
230                     area= PI * square(r);
240                     return area;
                }
300             float square(float area){
310                     float power=0;
320                     power=area*area;
330                     return power;
                }
```

function main()

Address= 130

r=10.0

area=0

Stack Frame

Stack: Application

การรันฟังก์ชัน

```
#include<stdio.h>
float calArea(float r);
float square(float area);
100  void main(){
110     float r=10.0;
120     float area=0;
130     area=calArea(r);
140     printf("%f",area);
     }
200  float calArea(float r)
210  {       float PI=3.14129;
220          float area=0;
230          area= PI * square(r);
240          return area;
     }
300  float square(float area){
310          float power=0;
320          power=area*area;
330          return power;
     }
```

function main()

Address= 130

r=10.0

area=314.12

Stack Frame

Stack: Application

การรันฟังก์ชัน

```
#include<stdio.h>
float calArea(float r);
float square(float area);
100  void main(){
110     float r=10.0;
120     float area=0;
130     area=calArea(r);
140     printf("%f",area);
     }
200  float calArea(float r)
210  {       float PI=3.14129;
220          float area=0;
230          area= PI * square(r);
240          return area;
     }
300  float square(float area){
310          float power=0;
320          power=area*area;
330          return power;
     }
```

function main()

Address= 140

r=10.0

area=314.12

Stack Frame

Stack: Application

การรันฟังก์ชัน

```
#include<stdio.h>
float calArea(float r);
float square(float area);
100  void main(){
110     float r=10.0;
120     float area=0;
130     area=calArea(r);
140     printf("%f",area);
    }
200  float calArea(float r)
210  {      float PI=3.14129;
220         float area=0;
230         area= PI * square(r);
240         return area;
    }
300  float square(float area){
310         float power=0;
320         power=area*area;
330         return power;
    }
```

Stack Frame

Stack: Application

การรันฟังก์ชัน

```
#include<stdio.h>
float calArea(float r);
float square(float area);
100  void main(){
110     float r=10.0;
120     float area=0;
130     area=calArea(r);
140     printf("%f",area);
     }
200  float calArea(float r)
210  {       float PI=3.14129;
220          float area=0;
230          area= PI * square(r);
240          return area;
     }
300  float square(float area){
310          float power=0;
320          power=area*area;
330          return power;
     }
```

Stack Frame

โปรแกรมสิ้นสุดการทำงานเมื่อ Stack ไม่มีข้อมูล

Stack: Application

คำสั่ง call ของ X86

Original 8086/8088 instructions [edit]

Original 8086/8088 instruction set

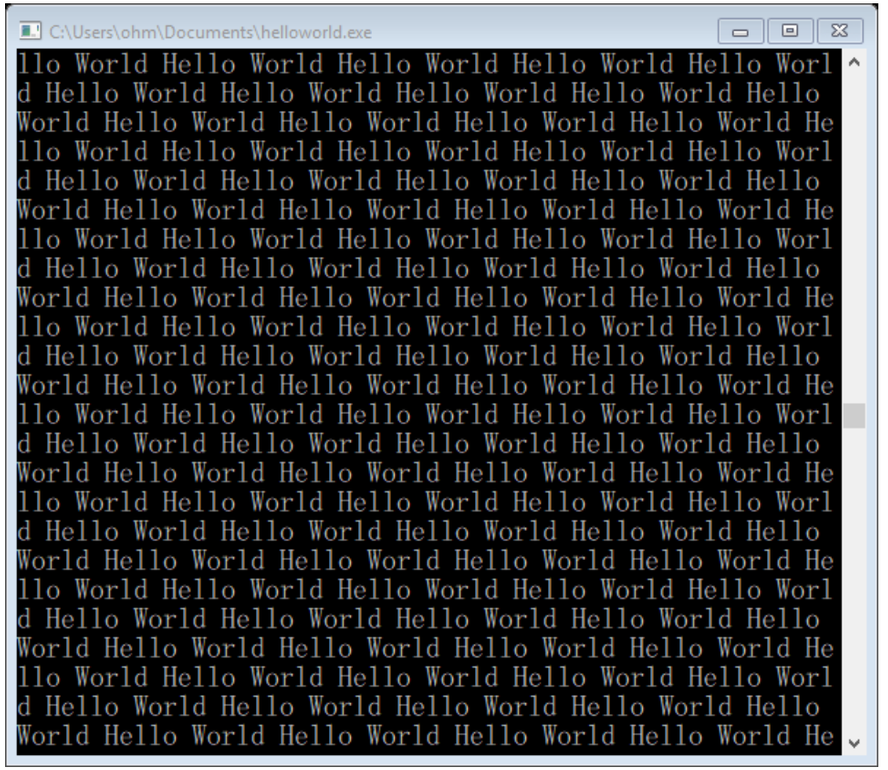
Instruction	Meaning	Notes	Opcode
AAA	ASCII adjust AL after addition	used with unpacked binary coded decimal	0x37
AAD	ASCII adjust AX before division	8086/8088 datasheet documents only base 10 version of the AAD instruction (opcode 0xD5 0x0A), but any other base will work. Later Intel's documentation has the generic form too. NEC V20 and V30 (and possibly other NEC V-series CPUs) always use base 10, and ignore the argument, causing a number of incompatibilities	0xD5
AAM	ASCII adjust AX after multiplication	Only base 10 version (Operand is 0xA) is documented, see notes for AAD	0xD4
AAS	ASCII adjust AL after subtraction		0x3F
ADC	Add with carry	`destination := destination + source + carry_flag`	0x10…0x15, 0x80/2…0x83/2
ADD	Add	(1) `r/m += r/imm;` (2) `r += m/imm;`	0x00…0x05, 0x80/0…0x83/0
AND	Logical AND	(1) `r/m &= r/imm;` (2) `r &= m/imm;`	0x20…0x25, 0x80/4…0x83/4
CALL	Call procedure	`push eip; eip points to the instruction directly after the call`	0x9A, 0xE8, 0xFF/2, 0xFF/3
CBW	Convert byte to word		0x98
CLC	Clear carry flag	`CF = 0;`	0xF8

การ call จะ push ตำแหน่งโปรแกรมปัจจุบัน ลง stack เพื่อให้กลับมาที่เดิมได้

Stack ใช้ทำอะไรได้บ้าง

ใช้ในเขียนโปรแกรมแบบ Recursive

```
#include<stdio.h>
void main(){
   printf("Hello World ");
   main();
}
```



Recursive

การเวียนซ้ำแบบ Recursive จะต้องเป็นไปตามเงื่อนไขต่อไปนี้

1 การเรียกตัวเองแต่ละครั้ง จะต้องเข้าใกล้คำตอบขึ้นเรื่อย ๆ

2 ต้องมีจุดหยุด

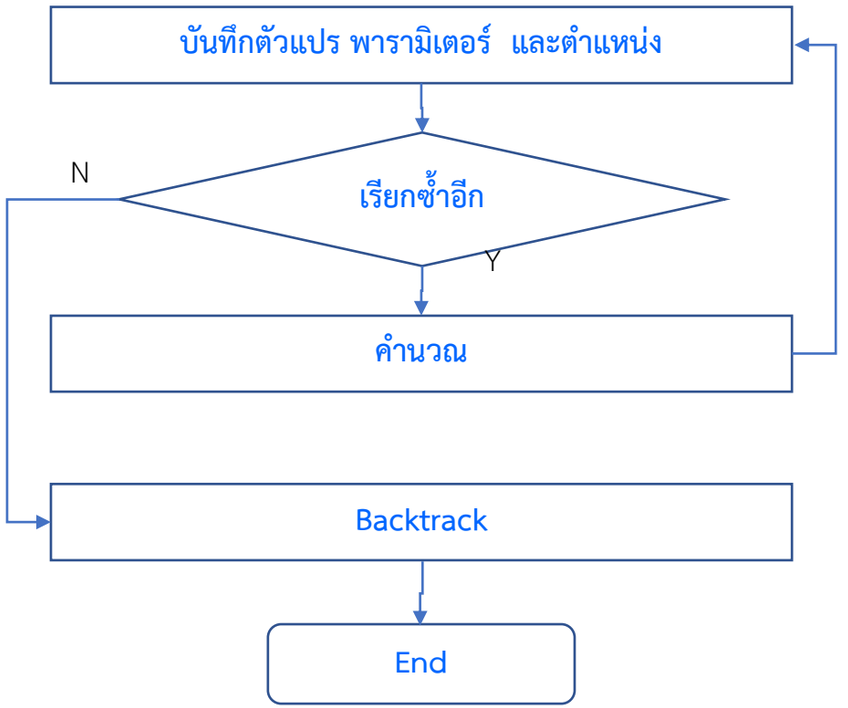
3 มีเงื่อนไขในการกำหนดการเรียกซ้ำ

4 โครงสร้างแบ่งเป็น 3 ส่วนคือ Prologue , Body , Epilogue

Prologue

Body

Epilogue



Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
        #include<stdio.h>

200     int factorial(int n){
210             int m;
220             int fac;
230             if(n==0) return 1;
240             m=n-1;
250             fac=factorial(m);
260             return(n*fac);
        }
100     void main(){
110     printf("%d",factorial(4));
        }
```

Body

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
        #include<stdio.h>

200     int factorial(int n){
210             int m;
220             int fac;
230             if(n==0) return 1;
240             m=n-1;
250             fac=factorial(m);
260             return(n*fac);
        }
100     void main(){
110     printf("%d",factorial(4));
        }
```

Stack Frame

function main()
Address = 110

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
        #include<stdio.h>

200     int factorial(int n){
210             int m;
220             int fac;
230             if(n==0) return 1;
240             m=n-1;
250             fac=factorial(m);
260             return(n*fac);
        }
100     void main(){
110     printf("%d",factorial(4));
        }
```

Stack Frame

function factorial() Address = 220 m=0 , fac = 0 , n=4
function main() Address = 110

Prologue

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
        #include<stdio.h>

200     int factorial(int n){
210             int m;
220             int fac;
230             if(n==0) return 1;
240             m=n-1;
250             fac=factorial(m);
260             return(n*fac);
        }
100     void main(){
110     printf("%d",factorial(4));
        }
```

Stack Frame

function factorial() Address = 230 m=0 , fac = 0 , n=4
function main() Address = 110

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
        #include<stdio.h>

200     int factorial(int n){
210             int m;
220             int fac;
230             if(n==0) return 1;
240             m=n-1;
250             fac=factorial(m);
260             return(n*fac);
        }
100     void main(){
110     printf("%d",factorial(4));
        }
```

Stack Frame

function factorial() Address = 240 m=3 , fac = 0 , n=4
function main() Address = 110

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
        #include<stdio.h>

200     int factorial(int n){
210             int m;
220             int fac;
230             if(n==0) return 1;
240             m=n-1;
250             fac=factorial(m);
260             return(n*fac);
        }
100     void main(){
110     printf("%d",factorial(4));
        }
```

Stack Frame

function factorial() Address = 250 m=3 , fac = 0 , n=4
function main() Address = 110

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
        #include<stdio.h>

200     int factorial(int n){
210             int m;
220             int fac;
230             if(n==0) return 1;
240             m=n-1;
250             fac=factorial(m);
260             return(n*fac);
        }
100     void main(){
110     printf("%d",factorial(4));
        }
```

Stack Frame

function factorial() Address = 220 m=0 , fac = 0 , n=3
function factorial() Address = 250 m=3 , fac = 0 , n=4
function main() Address = 110

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
        #include<stdio.h>

200     int factorial(int n){
210             int m;
220             int fac;
230             if(n==0) return 1;
240             m=n-1;
250             fac=factorial(m);
260             return(n*fac);
        }
100     void main(){
110     printf("%d",factorial(4));
        }
```

Stack Frame

function factorial() Address = 250 m=2 , fac = 0 , n=3
function factorial() Address = 250 m=3 , fac = 0 , n=4
function main() Address = 110

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
#include<stdio.h>

200     int factorial(int n){
210             int m;
220             int fac;
230             if(n==0) return 1;
240             m=n-1;
250             fac=factorial(m);
260             return(n*fac);
        }
100     void main(){
110     printf("%d",factorial(4));
        }
```

Stack Frame

function factorial() Address = 220 m=0 , fac = 0 , n=2
function factorial() Address = 250 m=2 , fac = 0 , n=3
function factorial() Address = 250 m=3 , fac = 0 , n=4
function main() Address = 110

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
        #include<stdio.h>

200     int factorial(int n){
210             int m;
220             int fac;
230             if(n==0) return 1;
240             m=n-1;
250             fac=factorial(m);
260             return(n*fac);
        }
100     void main(){
110     printf("%d",factorial(4));
        }
```

Stack Frame

function factorial() Address = 250 m=1 , fac = 0 , n=2
function factorial() Address = 250 m=2 , fac = 0 , n=3
function factorial() Address = 250 m=3 , fac = 0 , n=4
function main() Address = 110

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
                #include<stdio.h>

200             int factorial(int n){
210                     int m;
220                     int fac;
230                     if(n==0) return 1;
240                     m=n-1;
250                     fac=factorial(m);
260                     return(n*fac);
                }
100             void main(){
110             printf("%d",factorial(4));
                }
```

Stack Frame

function factorial() Address = 220 m=0 , fac = 0 , n=1
function factorial() Address = 250 m=1 , fac = 0 , n=2
function factorial() Address = 250 m=2 , fac = 0 , n=3
function factorial() Address = 250 m=3 , fac = 0 , n=4
function main() Address = 110

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
        #include<stdio.h>

200     int factorial(int n){
210             int m;
220             int fac;
230             if(n==0) return 1;
240             m=n-1;
250             fac=factorial(m);
260             return(n*fac);
        }
100     void main(){
110     printf("%d",factorial(4));
        }
```

Stack Frame

function factorial() Address = 250 m=0 , fac = 0 , n=1
function factorial() Address = 250 m=1 , fac = 0 , n=2
function factorial() Address = 250 m=2 , fac = 0 , n=3
function factorial() Address = 250 m=3 , fac = 0 , n=4
function main() Address = 110

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
#include<stdio.h>

200     int factorial(int n){
210             int m;
220             int fac;
230             if(n==0) return 1;
240             m=n-1;
250             fac=factorial(m);
260             return(n*fac);
        }
100     void main(){
110     printf("%d",factorial(4));
        }
```

Stack Frame
function factorial() Address = 220 m=0 , fac = 0 , n=0
function factorial() Address = 250 m=0 , fac = 0 , n=1
function factorial() Address = 250 m=1 , fac = 0 , n=2
function factorial() Address = 250 m=2 , fac = 0 , n=3
function factorial() Address = 250 m=3 , fac = 0 , n=4
function main()

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
#include<stdio.h>

200     int factorial(int n){
210         int m;
220         int fac;
230         if(n==0) return 1;
240         m=n-1;
250         fac=factorial(m);
260         return(n*fac);
        }
100     void main(){
110     printf("%d",factorial(4));
        }
```

Stack Frame
function factorial() Address = 230 m=0 , fac = 0 , n=0
function factorial() Address = 250 m=0 , fac = 0 , n=1
function factorial() Address = 250 m=1 , fac = 0 , n=2
function factorial() Address = 250 m=2 , fac = 0 , n=3
function factorial() Address = 250 m=3 , fac = 0 , n=4
function main()

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
                #include<stdio.h>

200             int factorial(int n){
210                     int m;
220                     int fac;
230                     if(n==0) return 1;
240                     m=n-1;
250                     fac=factorial(m);
260                     return(n*fac);
                }
100             void main(){
110             printf("%d",factorial(4));
                }
```

Stack Frame
function factorial() Address = 250 m=0 , fac = 0 , n=1
function factorial() Address = 250 m=1 , fac = 0 , n=2
function factorial() Address = 250 m=2 , fac = 0 , n=3
function factorial() Address = 250 m=3 , fac = 0 , n=4
function main() Address = 110

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
                #include<stdio.h>

200             int factorial(int n){
210                     int m;
220                     int fac;
230                     if(n==0) return 1;
240                     m=n-1;
250                     fac=factorial(m);
260                     return(n*fac);
                }
100             void main(){
110             printf("%d",factorial(4));
                }
```

function factorial() Address = 250 m=0 , fac = 1 , n=1
function factorial() Address = 250 m=1 , fac = 0 , n=2
function factorial() Address = 250 m=2 , fac = 0 , n=3
function factorial() Address = 250 m=3 , fac = 0 , n=4
function main() Address = 110

Stack Frame

Epilogue

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
                #include<stdio.h>

200             int factorial(int n){
210                     int m;
220                     int fac;
230                     if(n==0) return 1;
240                     m=n-1;
250                     fac=factorial(m);
260                     return(n*fac);
                }
100             void main(){
110             printf("%d",factorial(4));
                }
```

function factorial() Address = 260 m=0 , fac = 1 , n=1
function factorial() Address = 250 m=1 , fac = 0 , n=2
function factorial() Address = 250 m=2 , fac = 0 , n=3
function factorial() Address = 250 m=3 , fac = 0 , n=4
function main() Address = 110

Stack Frame

Epilogue

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
                #include<stdio.h>

200             int factorial(int n){
210                     int m;
220                     int fac;
230                     if(n==0) return 1;
240                     m=n-1;
250                     fac=factorial(m);
260                     return(n*fac);
                }
100             void main(){
110             printf("%d",factorial(4));
                }
```

function factorial() Address = 250 m=1 , fac = 1 , n=2
function factorial() Address = 250 m=2 , fac = 0 , n=3
function factorial() Address = 250 m=3 , fac = 0 , n=4
function main() Address = 110

Stack Frame

Epilogue

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
                #include<stdio.h>

200             int factorial(int n){
210                     int m;
220                     int fac;
230                     if(n==0) return 1;
240                     m=n-1;
250                     fac=factorial(m);
260                     return(n*fac);
                }
100             void main(){
110             printf("%d",factorial(4));
                }
```

function factorial() Address = 260 m=1 , fac = 1 , n=2
function factorial() Address = 250 m=2 , fac = 0 , n=3
function factorial() Address = 250 m=3 , fac = 0 , n=4
function main() Address = 110

Stack Frame

Epilogue

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
                #include<stdio.h>

200             int factorial(int n){
210                     int m;
220                     int fac;
230                     if(n==0) return 1;
240                     m=n-1;
250                     fac=factorial(m);
260                     return(n*fac);
                }
100             void main(){
110             printf("%d",factorial(4));
                }
```

function factorial()
Address = 250
m=2 , fac = 2 , n=3
function factorial()
Address = 250
m=3 , fac = 0 , n=4
function main()
Address = 110

Stack Frame

Epilogue

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
                #include<stdio.h>

200             int factorial(int n){
210                     int m;
220                     int fac;
230                     if(n==0) return 1;
240                     m=n-1;
250                     fac=factorial(m);
260                     return(n*fac);
                }
100             void main(){
110             printf("%d",factorial(4));
                }
```

function factorial() Address = 260 m=2 , fac = 2 , n=3
function factorial() Address = 250 m=3 , fac = 0 , n=4
function main() Address = 110

Stack Frame

Epilogue

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
                #include<stdio.h>

200             int factorial(int n){
210                     int m;
220                     int fac;
230                     if(n==0) return 1;
240                     m=n-1;
250                     fac=factorial(m);
260                     return(n*fac);
                }
100             void main(){
110             printf("%d",factorial(4));
                }
```

function factorial()
Address = 250
m=3 , fac = 6 , n=4
function main()
Address = 110

Stack Frame

Epilogue

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

```
                #include<stdio.h>

200             int factorial(int n){
210                     int m;
220                     int fac;
230                     if(n==0) return 1;
240                     m=n-1;
250                     fac=factorial(m);
260                     return(n*fac);
                }
100             void main(){
110             printf("%d",factorial(4));
                }
```

function factorial()
Address = 260
m=3 , fac = 6 , n=4
function main()
Address = 110

Stack Frame

Epilogue

Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

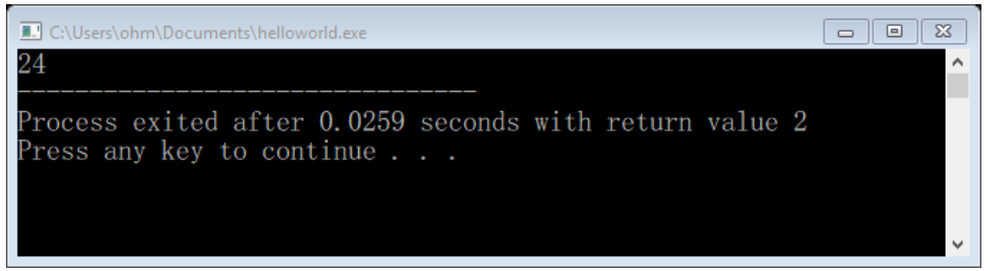
```
#include<stdio.h>

200     int factorial(int n){
210         int m;
220         int fac;
230         if(n==0) return 1;
240         m=n-1;
250         fac=factorial(m);
260         return(n*fac);
        }
100     void main(){
110     printf("%d",factorial(4));
        }
```

Stack Frame

function main()

Address = 110



Recursive

ตัวอย่างโปรแกรมคำนวณค่า 4!

1 x 2 x 3 x 4 = ?

Stack Frame

```
        #include<stdio.h>

200     int factorial(int n){
210         int m;
220         int fac;
230         if(n==0) return 1;
240         m=n-1;
250         fac=factorial(m);
260         return(n*fac);
        }
100     void main(){
110     printf("%d",factorial(4));
        }
```

คำถาม : For loop แปลงเป็น Recursive ได้หรือไม่ ?

ตอบ : สามารถทำได้

คำถาม : การวนลูปแบบ infinite สามารถแปลงเป็น Recursive ได้หรือไม่ ?

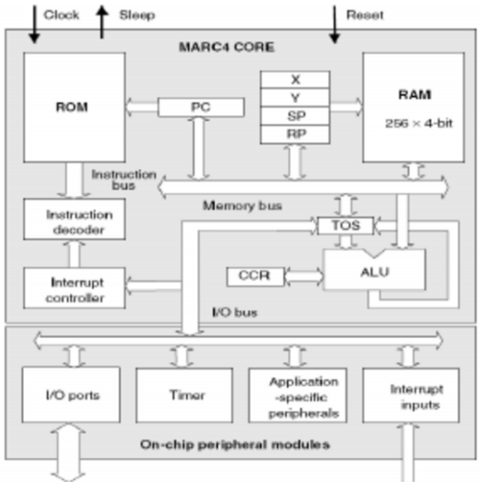
ตอบ : สามารถทำได้ แต่ในทางปฏิบัติ โปรแกรมจะรันไม่ได้
เนื่องจากเราไม่ได้มีหน่วยความจำ stack ขนาด infinite

Stack: Application

Stack ใช้ทำอะไรก็ได้

เป็นไปได้ไหมที่จะสร้าง CPU โดยมีเพียง stack อย่างเดียว

แล้วเขียนโปรแกรมให้ทุกอย่างเป็น Recursive

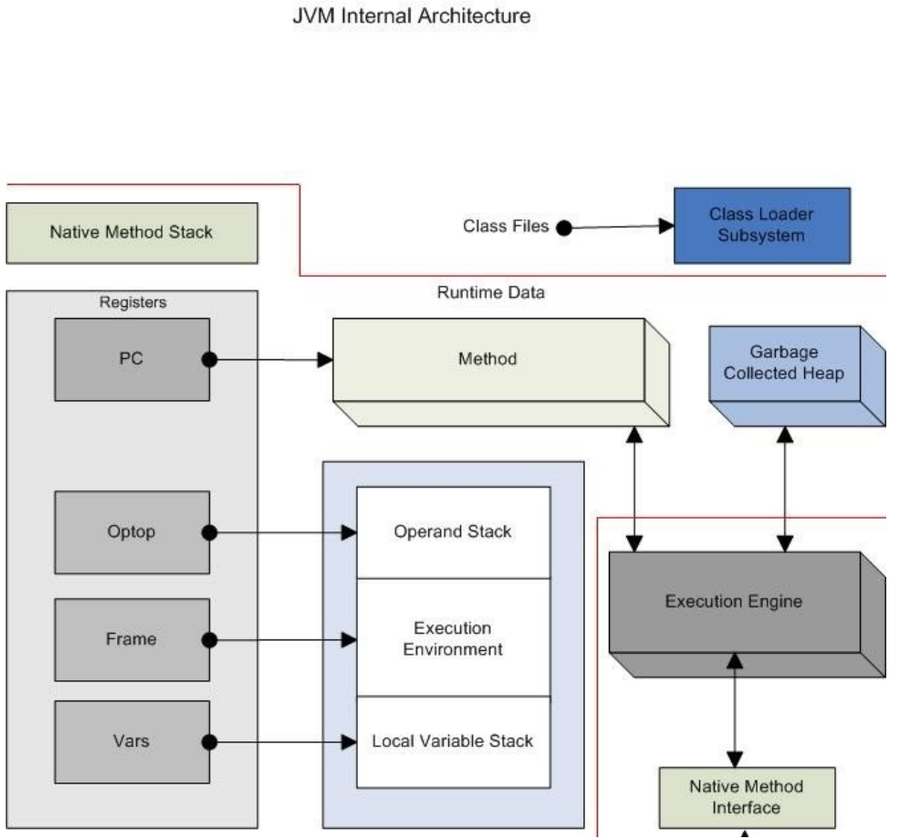


Marc4 ใช้สถาปัตยกรรมแบบ Stack machine

Stack: Application

Stack ใช้ทำอะไรก็ได้

เป็นไปได้ไหมที่จะสร้าง CPU โดยมีเพียง stack อย่างเดียว



Java VM ก็เป็น stack machine

ข้อดีของสถาปัตยกรรมแบบ Stack

- Opcode มีขนาดเล็ก (มีแค่ push และ pop)
- Compiler สร้างง่าย
- โปรแกรมไม่กำกวม (เขียนทุกอย่างเป็น postfix)
- คำนวณได้เร็ว
- CPU มีขนาดเล็ก

ข้อเสีย

- ค้นหาข้อมูลทำได้ช้า (ทางเข้า - ออก มีทางเดียว)
- มี Instruction set ให้ใช้ไม่มาก
- โปรแกรมแบบ stack เขียนยากมาก

Stack: Application

CPU แบบ Stack จึงนิยมใช้สำหรับงานเฉพาะทาง ที่ต้องการความเร็ว
ในการคำนวณสูงๆ เท่านั้น ตัวอย่างเช่น
เครื่องคิดเลข หรือ คอมพิวเตอร์ควบคุมการบิน

Saab JAS 39 Gripen
ใช้ Microprocessor ควบคุมท่าบิน
สามารถโปรแกรมได้

ทำไมเราจึงไม่ค่อยใช้ Stack Machine

8 August 1993,Stockholm

Software Bug

Stack: Application

CPU แบบ Stack จึงนิยมใช้สำหรับงานเฉพาะทาง ที่ต้องการความเร็ว
ในการคำนวณสูงๆ เท่านั้น ตัวอย่างเช่น
เครื่องคิดเลข หรือ คอมพิวเตอร์ควบคุมการบิน

การโปรแกรมทำได้ยาก จึงอาจเกิดความผิดพลาดได้ง่าย ทุกวันนี้ความเร็วของ
CPU ไม่ได้แตกต่างกันมาก ดังนั้น Stack Machine จึงไม่เป็นที่นิยม

Java VM เป็น Stack ก็จริง แต่การเขียนโปรแกรม เราไม่ได้เขียนด้วยภาษาเครื่อง
แต่เขียนด้วยภาษา Java โดยใช้ระบบคอมไพลที่ซับซ้อน จึงทำงานช้า

CPU ที่เราใช้กันส่วนมากจะเป็นแบบ Hybrid ที่ไม่ได้ทำงานบน Stack
เป็นหลัก แต่สามารถเรียกใช้งาน stack เพื่อทำงานบางอย่างได้

Stack: Application

CPU ที่เราใช้กันส่วนมากจะเป็นแบบ Hybrid ที่ไม่ได้ทำงานบน Stack เป็นหลัก แต่สามารถเรียกใช้งาน stack เพื่อทำงานบางอย่างได้

เพราะ Stack นั้นยังคงเป็นเครื่องหลักที่สำคัญมากในหลาย ๆ อัลกอริทึม