บทที่ 4 กองซ้อน(stacks)

4.1 บทนิยาม

4.1.1 รายการเชิงเส้น

4.1.2 กองซ้อน

4.1.3 การปฏิบัติการบนกองซ้อน

4.2 กองซ้อนใน COBOL และ Pascal

4.2.1 เก็บกองซ้อนในแถวลำดับ

4.2.2 การประกาศกองซ้อน

4.2.3 การปฏิบัติการบนกองซ้อน

4.3 ตัวอย่างงานประยุกต์ของกองซ้อน

4.3.1 การจับคู่วงเล็บ

4.3.2 การเรียกซ้ำ

4.3.3 สัญกรณ์เติมหลัง

4.4 การแปลงส่งไปยังหน่วยเก็บ

แบบฝึกหัด

IT 204 71

บทที่ 4
กองซ้อน
(Stack)

โครงสร้างข้อมูลแบบเชิงเส้นที่ปกติใช้ร่วมกันมากที่สุดชนิดหนึ่งได้แก่กองซ้อน แนะนำการปฏิบัติการซึ่งนิยามโครงสร้างข้อมูลกองซ้อน หลังจากนั้นอภิปรายสิ่งอำนวยความสะดวกสำหรับการประกาศและการคุมแต่งกองซ้อน เนื้อหาในบทนี้ส่วนมากอุทิศให้กับตัวอย่างของการใช้กองซ้อน

4.1 บทนิยาม (Definitions)

4.1.1 รายการเชิงเส้น (Linear list)

รายการเชิงเส้น หมายถึง โครงสร้างข้อมูลซึ่งประกอบด้วยเซตแบบอันดับของสมาชิก จำนวนสมาชิกในรายการผันแปรได้

(A linear list is a data structure comprised of an ordered set of elements; the number of elements in the list can vary.)

รายการเชิงเส้นชื่อ A มีสมาชิก T ตัว ใช้สัญลักษณ์ดังนี้

$$A = [A_1 , A_2 , \ldots , A_T]$$

ถ้า T = 0 จะเรียก A ว่ารายการว่าง (empty list หรือ null list)

สมาชิกสามารถออกจากตำแหน่งใด ๆ ก็ได้ในรายการเชิงเส้น และสมาชิกตัวใหม่สามารถใส่เข้าไปในตำแหน่งใด ๆ ก็ได้ในรายการ เพราะฉะนั้นรายการสามารถโตขึ้นหรือเล็กลงได้ตลอดเวลา

4.1.2 กองซ้อน (Stack)

กองซ้อนหมายถึงรายการเชิงเส้นกรณีพิเศษซึ่งปฏิบัติการใส่และการลบถูกจำกัดให้เกิดขึ้นที่ปลายหนึ่งด้านซึ่งเรียกว่า top ของกองซ้อน

(A **stack*** is a special case of linear list in which insertion and deletion operations are restricted to occur only at one end, which is referred to as the stack's top.)

สมาชิกตัวบนสุดของกองซ้อน S ใช้สัญลักษณ์TOP(S) สำหรับกองซ้อน S เมื่อ

$$S = [S_1, S_2, \ldots , S_T]$$

TOP(S) คือ S_T

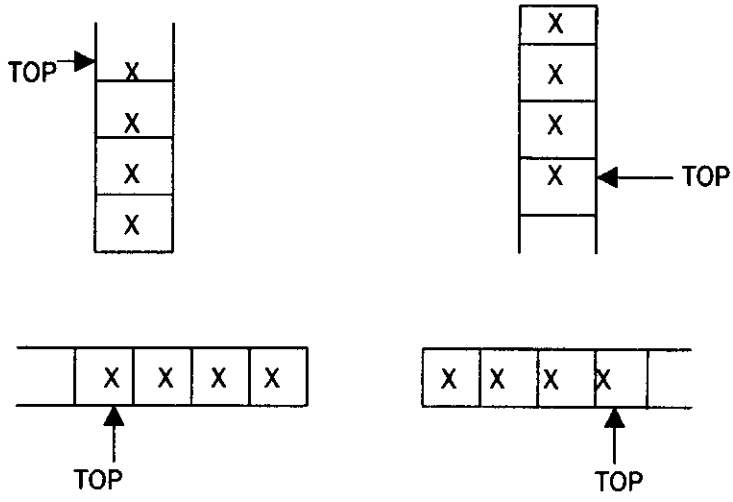
จำนวนสมาชิกในกองซ้อน S ใช้สัญลักษณ์ Noel(S)

Noel(S) คือลักษณะประจำของกองซ้อน S มีค่าเป็นจำนวนเต็ม

---

*หรือเรียกว่า LAST-IN-FIRST-OUT list

จากกองซ้อน S ข้างต้น Noel(S) คือ T
สมาชิกตัวที่ Noel ของกองซ้อนคือสมาชิกตัวบนสุดรูปภาพเชิงตรรกะของกองซ้อน สมาชิกตัวบนสุดอาจอยู่ บน หรือล่าง หรือเมื่อวาดภาพกองซ้อนตามแนวนอน สมาชิกตัวบนสุด อาจเป็น ซ้ายมือหรือขวามือ (รูป 4- 1)



รูป 4-1 Drawings of stacks

ภาพในรูปใดก็ตามใน 4 แบบข้างต้นนี้ใช้ได้หมด ในที่นี้เราใช้ข้อตกลงของการให้ top อยู่ตอนบน กรณีใดก็ตาม สำหรับกองซ้อน

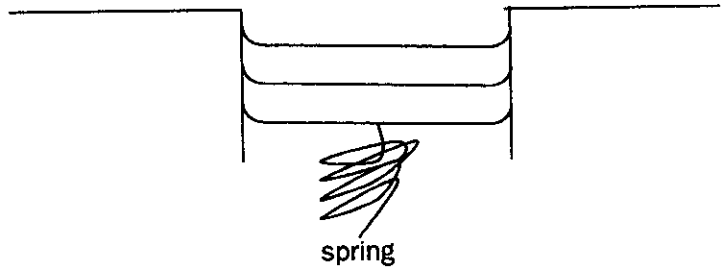
$$S = [S_1, S_2, \ldots\ldots, S_{Noel}]$$

เราพูดว่า สมาชิก S_i อยู่ข้างบนสมาชิก S_j สำหรับ $I > J$ ดังนั้น S_i จึงจะถูกนำออกมาจากกองซ้อนก่อนสมาชิกใด ๆ ซึ่งอยู่ตอนล่างของมัน S_i จึงมีเวลาอยู่ในกองซ้อนน้อยกว่าสมาชิกอื่น ๆ ที่อยู่ตอนล่างของมันเมื่อกองซ้อนโตขึ้น หรือเล็กลง สมาชิกตัวบน (top element) จะเปลี่ยนแปลงตลอดแต่สมาชิกตัวล่าง (bottom element) ยังคงที่

ตัวอย่าง

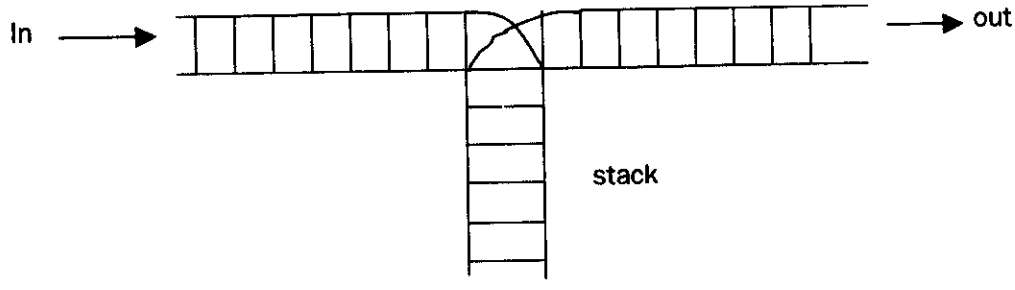
ตัวอย่างของกองซ้อนได้แก่ กองของถาดในร้านขายอาหาร(cafeteria)ซึ่งมีกลไกสปริงอยู่ข้างล่างดังนั้นเฉพาะถาดใบบนเท่านั้นที่มองเห็นและหยิบไปใช้ได้ (รูป 4-2)

การใส่ถาดอีกหนึ่งใบให้วางตอนบนของกองซ้อนซึ่งสปริงใต้จะกดสปริงลง ทำให้เห็นถาดบนสุดใบเดียว การหยิบถาดออกมาหนึ่งใบสปริงใต้กองซ้อนจะเบาขึ้น ทำให้ถาดใบถัดไปปรากฏที่ระดับโต๊ะ



รูป 4-2 Example stack or trays

อีกตัวอย่างหนึ่งของกองซ้อนคือระบบรถไฟ (railway system) ซึ่งสับเปลี่ยนขบวนรถไฟจากตำแหน่งหนึ่งไปยังอีกตำแหน่งหนึ่ง



รูป 4-3 tracks forming a stack

รถไฟขบวนท้ายสุดที่เข้ามาบนกองซ้อน คือ ขบวนแรกที่ออกจากกองซ้อน

4.1.3 **การปฏิบัติการบนกองซ้อน** (Stack Operations)

การปฏิบัติการพื้นฐานสี่ชนิดซึ่งถูกต้องสำหรับข้อมูลชนิดกองซ้อน:

1. Create (stack)
2. Isempty (stack)
3. Push (element, stack)
4. Pop (stack)

The Create (S) operator returns an empty stack with name S. By definition

Noel (Create (S)) is 0

and Top (create (S)) is null

ตัวปฏิบัติการ Isempty บอกว่ากองซ้อนว่างหรือไม่ว่างตัวถูกกระทำคือ stack ผลลัพธ์เป็น boolean

Isempty (S) เป็นจริงถ้ากองซ้อน S ว่าง (นั่นคือ if noel (S) = 0) และเป็นเท็จในกรณีอื่น ๆ

โปรดสังเกตว่า Isempty (Create(S)) is true

การใส่สมาชิกบนกองซ้อนเรียกว่า 1pushing the stack

Push (E,S) adds element E to the top of stack S.

ผลลัพธ์คือ กองซ้อนมีขนาดเพิ่มขึ้น โปรดสังเกตว่า

Top (push (E,S)) is E

Push (E,S) ทำให้เพิ่มค่าของ Noel(S)

ผลลัพธ์ของการใส่สมาชิกบนกองซ้อนใด ๆ ก็ตามไม่ทำให้เป็นกองซ้อนว่าง

Isempty (Push (E,S)) is false

การลบสมาชิกออกจากกองซ้อนเรียกว่า popping the stack

Pop(S) removes an element from the top of stack S.

ผลลัพธ์คือ กองซ้อนมีขนาดลดลง ถ้าสมาชิกตัวที่ลบออก (deleted element) ต้องการเก็บไว้การกระทำนี้ควรจะต้องทำก่อนปฏิบัติการ pop

โปรดสังเกตว่า Pop(S) ทำให้ Noel(S) มีค่าลดลงจะเกิด error ถ้ามีความพยายามpop สมาชิกออกจากกองซ้อนว่าง

Pop (Create (S)) yields an error condition

ถ้าไม่มีเหตุการณ์ข้างต้นนี้ Pop(S) ทำให้ Noel (S) มีค่าลดลงและเปลี่ยนแปลง Top(S)

ตัวปฏิบัติการ pop ลบล้างผลลัพธ์ของตัวปฏิบัติการ push

(The pop operator undoes the results of the push operator:)

Pop (Push (E,S)) is S

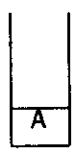
ตัวอย่าง เริ่มต้นจากกองซ้อนว่างจงพิจารณาผลกระทบของลำดับของ pushes และ pops เราแทนกองซ้อนว่าง S ด้วยรูปภาพ ในรูป 4-4 (a)

S

รูป 4-4 (a)

Noel(S) = 0 , Top(S) undefined

ขั้นแรกใส่สมาชิก A ทำให้ S = [A]

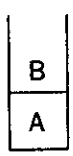


S

รูป 4-4 (b)

Noel(S) = 1, Top(S) = A

หลังจากนั้น ใส่สมาชิก B ทำให้ S = [A , B]

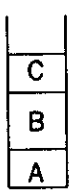


S

รูป 4-4 (c)

Noel (S) = 2, Top(S) = B

หลังจากนั้นใส่สมาชิก C ทำให้ S = [A , B , C]

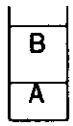


S

รูป 4-4 (d)

Noel (S) = 3, Top(S) = C

เอาสมาชิกออกหนึ่งตัวทำให้ S = [A , B]

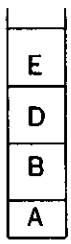


S

รูป 4-4 (e)

Noel (S) = 2 , Top(S) = B

หลังจากนั้น ใส่สมาชิกเพิ่มอีกสองตัว ชื่อ D และ E ทำให้ S = [A , B , D , E]

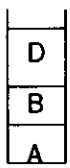


S

รูป 4-4 (f)

Noel (S) = 4, Top(S) = E

หลังจากนั้น เอาสมาชิกออกจากกองซ้อนหนึ่งตัว ทำให้ S = [A , B , D]



S

รูป 4-4 (g)

Noel (S) = 3, Top(S) = D

เช่นนี้เรื่อยไป กองซ้อนเรียกว่าถูกปฏิบัติการในลักษณะเข้าทีหลังออกก่อน (last – in – first – out) สมาชิกถูกลบออกแบบย้อนลำดับกับการใส่สมาชิกบนกองซ้อน

4.2 กองซ้อนใน COBOL และ Pascal (Stacks in COBOL and Pascal)

4.2.1 เก็บกองซ้อนในแถวลำดับ (Housing Stacks in Arrays)

ถึงแม้ว่าจะมีการใช้กองซ้อนอย่างกว้างขวางแต่ภาษาโปรแกรมส่วนมากไม่มีโครงสร้างข้อมูล built–in ชนิด stack ให้ใช้ ในการเขียนโปรแกรมแก้ปัญหาซึ่งเรียกใช้กองซ้อนโปรแกรมเมอร์ใช้คุณสมบัติที่มีอยู่ของภาษาเพื่อจำลองแบบการปฏิบัติการของกองซ้อน มีหลายวิธีเพื่อแทนที่กองซ้อนซึ่งวิธีที่ง่ายที่สุดเพื่อแทนที่กองซ้อนคือเก็บกองซ้อนในแถวลำดับ

กองซ้อนสามารถเก็บในแถวลำดับได้ แต่สิ่งที่สำคัญที่จะต้องจำไว้คือ กองซ้อนและแถวลำดับเป็นโครงสร้างข้อมูลสองชนิดที่แตกต่างกัน มีกฎเกณฑ์แตกต่างกัน

ข้อแรก ไม่มีข้อจำกัดใด ๆ กับสมาชิกในการใส่หรือลบออกจากแถวลำดับ ดังนั้นเป็นความรับผิดชอบของโปรแกรมเมอร์ที่จะบังคับกฎ LIFO สำหรับกองซ้อนซึ่งเก็บในแถวลำดับ

ข้อที่สอง ไม่มีสิ่งใดในแนวคิดของกองซ้อนซึ่งป้องกันมันจากชนิดต่าง ๆ ของสมาชิกข้อจำกัดเทียมอีกข้อหนึ่งคือ เมื่อโปรแกรมเมอร์ต้องระบุ upper bound บนค่าดรรชนีล่างของแถวลำดับตัวแปรกองซ้อนไม่มีข้อจำกัดบนจำนวนสูงสุดของสมาชิกซึ่งอาจจะมีอยู่ ณ. เวลาใด ๆ ที่กำหนดให้อย่างไรก็ตาม ตัวแปรกองซ้อนซึ่งเก็บในแถวลำดับมีขีดจำกัดกับการจัดสรรเนื้อที่ให้กับแถวลำดับนั้นความจริง กองซ้อนโตและเล็กลงตลอดเวลา แต่แถวลำดับมีขนาดคงที่

4.2.2 การประกาศกองซ้อน (Declaring Stacks)

จงพิจารณาการประกาศของตัวแปรกองซ้อนชื่อ S สมมติว่าสมาชิกแต่ละตัวของ S เป็น integer และ S มีสมาชิกมากที่สุด 100 ตัว
นอกเหนือจากประกาศแถวลำดับซึ่งจะเก็บกองซ้อน S แล้วเราต้องประกาศ ตัวแปร TOP–PTR ซึ่งค่าของมันจะเป็นดรรชนีล่างของสมาชิกตัวบนปัจจุบันของกองซ้อนเราตั้งชื่อการรวมกันของแถวลำดับแบะ top indicator ว่า STACK–STRUCT ด้วยการแทนที่ดังนี้

Noel(S) = TOP–PRT
Isempty (S) เป็นจริงเมื่อ TOP–PTR = 0
และเป็นเท็จ เมื่อ TOP–PTR > 0

ภาษา COBOL :

```
01    STACK–STRUCT.
      02   S  OCCURS 100 TIMES   PICTURE  9(5).
      02    TOP–PTR               PICTURE  9(3).
```

ภาษา Pascal :

```
type   stackstruct = record stack : array [1..100] of integer;
```

```
                                    topptr: integer;
                            end;
        var     S : stackstruct;
```

ถ้าในโปรแกรมเดียวกันมีกองซ้อนหลายชุดกองซ้อนแต่ละชุดต้องมี top indicator ของมันเอง (โปรดสังเกตว่า เราใช้ชื่อ TOP-PTR เพราะว่า TOP เป็นคำสงวนในภาษา COBOL)

4.2.3 **การปฏิบัติการบนกองซ้อน** (Stack Operations)

คอมไพเลอร์ (compiler) ไม่สามารถตรวจจับการฝ่าฝืนกฎ LIFO ของการปฏิบัติการ สำหรับกองซ้อนซึ่งเก็บในแถวลำดับได้ ดังนั้นโปรแกรมเมอร์จึงต้องระมัดระวังไม่ใช่ดรรชนีกับ S ที่ใดก็ตามยกเว้นการใช้เป็นค่าของ TOP-PTR

การปฏิบัติการของการใส่และการลบออกจาก S สามารถเขียนโปรแกรมได้ข้างล่างนี้ เราใช้ EON เป็นสมาชิกที่จะใส่บน S และใช้ EOFF เป็นสมาชิกตัวที่ลบออกจาก S โปรดสังเกตว่าเราเก็บ EOEF ให้ตัวแปร NOEL-MAX เป็นจำนวนสูงสุดของสมาชิกซึ่งเก็บในแถวลำดับของ S ในที่นี้ NOEL-MAX = 100 จะแสดงให้เห็นว่า application กำหนด action ที่จะกระทำเมื่อเกิดเงื่อนไข overflow (พยายามที่จะใส่สมาชิกบนกองซ้อนเต็ม) หรือเงื่อนไข underflow (พยายามลบสมาชิกออกจากกองซ้อนว่าง)

ใน COBOL paragraphs :

```
PUSH.
        IF TOP-PTR < NOEL-MAX
        THEN COMPUTE TOP-PTR = TOP-PTR + 1
                MOVE EON TO S(TOP-PTR)
        ELSE overflow-condition.
POP.
        IF TOP-PTR > 0
        THEN MOVE S(TOP-PTR) TO EOFF
                COMPUTE TOP-PTR = TOP-PTR - 1
        ELSE  underflow-condition.
```

ใน Pascal procedures:

```
procedure push (eon: integer);
begin
        if (s.topptr < noelmax)
        then begin
                s.topptr := s.topptr + 1 ;
                s.stack [s.topptr] := eon
             end;
        else OVERFLOW – CONDITION.
end;

procedure pop (var eoff : integer);
begin
        if (s.topptr > 0)
        then begin
                eoff := s.stack [s.topptr];
                s.topptr := s.topptr – 1
             end;
        else UNDERFLOW – CONDITION
end;
```

สมมติว่า S เป็นตัวแปรส่วนกลาง (global variable) สิ่งสำคัญที่ต้องระวังอีกอย่างหนึ่งคือ กองซ้อนที่เก็บในแถวลำดับ ความรับผิดชอบของโปรแกรมเมอร์ ต้องมั่นใจว่า เฉพาะ top pointer เท่านั้นที่ใช้ เป็นตัวชี้ของกองซ้อน

มีวิธีอื่น ๆ ที่ใช้แทนที่ตัวแปรกองซ้อนเมื่อภาษาโปรแกรมไม่มีข้อมูล built – in ชนิด stack ให้ใช้ ซึ่งจะได้อภิปรายในบทที่ 6 เรื่องรายการโยงต่อไป

4.3 **ตัวอย่างงานประยุกต์ของกองซ้อน** (Example applications of stacks)

กองซ้อนถูกนำมาใช้แก้ปัญหาอย่างกว้างขวาง นำมาใช้โดยคอมไพเลอร์ , โดยระบบปฏิบัติการและโดยโปรแกรมประยุกต์ ในหัวข้อนี้เราวางแนวทางการใช้กองซ้อนหลายแบบและยังมีอื่น ๆ อีกมาก

4.3.1 ตัวอย่าง : การจับคู่วงเล็บ (Example : Matching Parentheses)

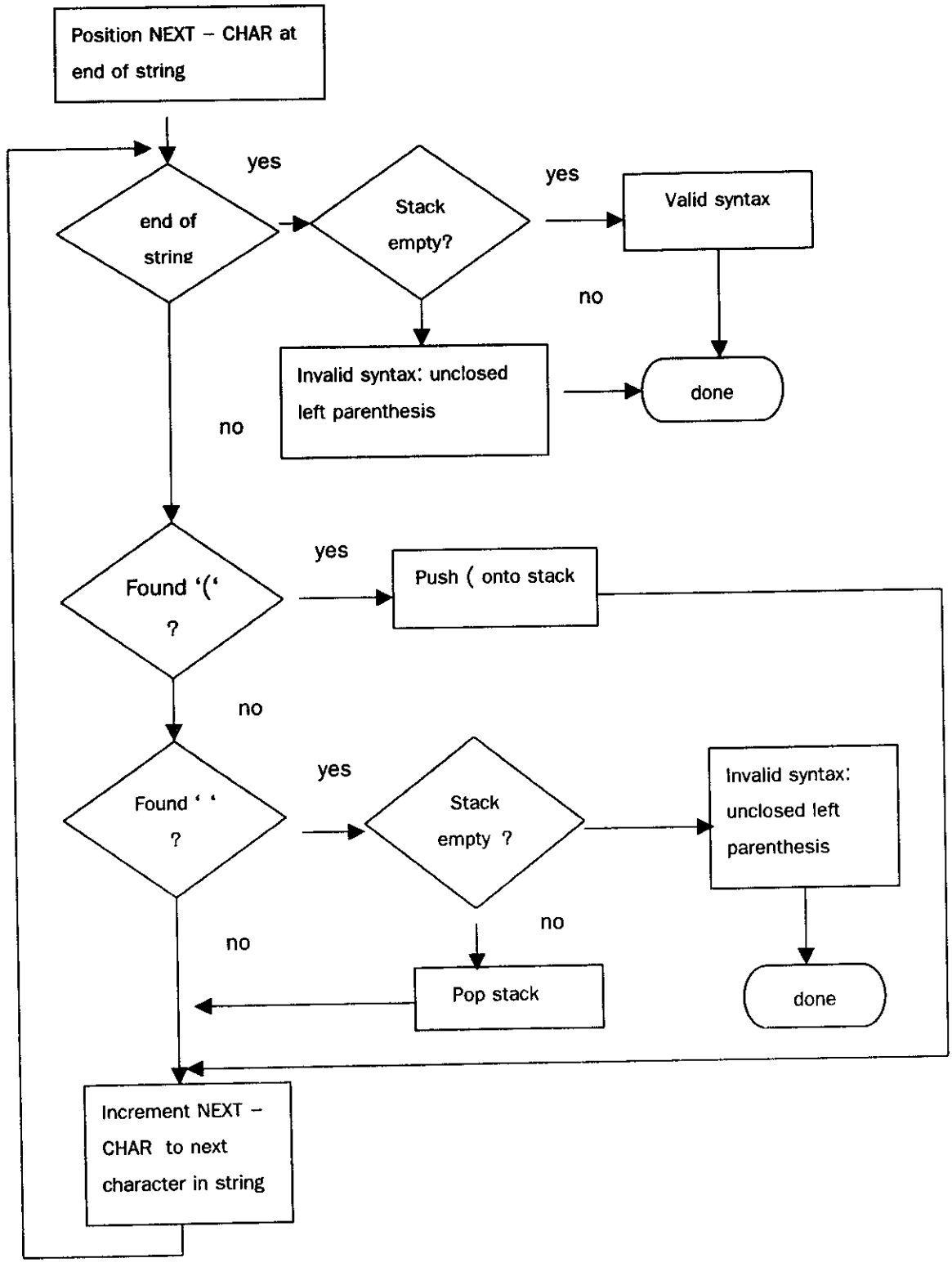
งานอย่างหนึ่งของคอมไพเลอร์ คือ เพื่อตรวจทานว่าโปรแกรมเมอร์ยึดมั่นกฎไวยากรณ์ (วากยสัมพันธ์) ของภาษาโปรแกรม

จงพิจารณาปัญหาการตรวจทานวากยสัมพันธ์เพื่อให้มั่นใจว่าวงเล็บเปิดแต่ละตัวมีวงเล็บปิดที่สมนัยกัน กองซ้อนสามารถนำมาใช้เพื่อโปรซีเดอร์การจับคู่ทำได้ง่ายอัลกอริทึมง่าย เรากราดตรวจ (scan) สายอักขระสมาชิกจากซ้ายไปขวา เมื่อใดก็ตามที่พบวงเล็บเปิดให้นำไปใส่บนกองซ้อน เมื่อใดก็ตามที่พบวงเล็บปิด ให้ตรวจสอบสถานะของกองซ้อน ถ้ากองซ้อนว่างเพราะฉะนั้นเราพบวงเล็บปิดซึ่งไม่มีวงเล็บเปิดจึงเป็นข้อผิดพลาด ถ้ากองซ้อนไม่ว่าง เราพบคู่ของวงเล็บปิดให้ pop วงเล็บเปิดออกจากกองซ้อน ถ้ากองซ้อนไม่ว่างที่ตอนจบของสายอักขระแสดงว่าอย่างน้อยที่สุดมีวงเล็บเปิดหนึ่งตัว ณ. จุดใด ๆ ในการกราดตรวจ จำนวนของสมาชิกในกองซ้อน คือ ความลึกการซ้อนกันของวงเล็บผังงานของอัลกอริทึมนี้กำหนดในรูป 4–5 เราเขียนโปรแกรมรูทีนนี้ด้วยภาษา COBOL, สายอักขระที่ถูกกราดตรวจถูกเก็บทีละตัวอักขระในแถวลำดับชื่อ STRING

กองซ้อนเก็บในแถวลำดับชื่อ STACK สมมติว่าจำนวนสูงสุดของตัวอักขระในสายอักขระ คือ 80 ตัวและสายอักขระ (actual string) จบด้วย semi colon(;)

โครงสร้างข้อมูลนิยามดังนี้

```
01    STRACK – STRUCT
        02    STACK OCCURS 80 TIMES PICTURE X.
        02    TOP–PTR PICTURE 99 VALUE 0.
01    STRING .
        02    CHAR OCCURS 80 TIMES PICTURE X.
01    NEXT–CHAR    PICTURE 99.
```



รูป 4-5 Logic for matching parentheses

โครงสร้างเหล่านี้จัดการโดยรหัสต่อไปนี้ :

```
        PERFORM SCAN-NEXT-CHAR VARYING NEXT-CHAR
                        FROM 1 BY 1 UNTIL NEXT-CHAR > 80
                        OR CHAR(NEXT - CHAR) = " ; " .
        IF TOP-PTR = 0
                        Valid syntax
        ELSE invalid-syntax-unclose-left-parenthesis.
        เมื่อ
SCAN-NEXT-CHAR.
        IF CHAR (NEXT-CHAR) = " ("
                PERFORM PUSH.
        ELSE IF CHAR(NEXT-CHAR) = " ) "
                PERFORM POP.
PUSH.
        COMPUTE  TOP-PTR = TOP-PTR + 1.
        MOVE CHAR (NEXT-CHAR) TO STACK (TOP-PTR).
POP.
        IF TOP-PTR = 0
                Invalid-syntax-no-matching-left-parenthesis
        ELSE  COMPUTE  TOP-PTR = TOP-PTR - 1.
```

ปัญหา syntax-verification ที่น่าสนใจมากขึ้นคือการสร้างอัลกอริทึม สำหรับการจับคู่ไม่ใช่แค่วงเล็บเล็กเปิดและปิดแต่ให้รวมถึงวงเล็บปีกกาเปิด และปิด วงเล็บใหญ่เปิดและปิดในสายอักขระชุดเดียวกันด้วย

ตัวอย่างเช่น อัลกอริทึมควรจะสามารถมีสายอักขระข้างล่างนี้เป็นอินพุตและบอก ว่าสัญลักษณ์วงเล็บใหญ่จับคู่ถูกต้องหรือไม่

```
({ A + B * C / ([ D + E - (F + G) * (A + B )] * (F - G + B) }/
                                (D - E - (A + B)))
```

จงสร้างอัลกอริทึมและเขียนโปรแกรม ปัญหานี้จะต้องใช้กองซ้อนทั้งหมดกี่ชุด

4.3.2 ตัวอย่าง : การเรียกซ้ำ (Example : Recursion)

กองซ้อนปกตินำมาใช้เก็บ track ว่าโปรแกรมอยู่ที่ใดเมื่อโปรแกรมนี้มีโปรซีเดอร์ซึ่งเรียกตัวมันเอง โปรซีเดอร์เหล่านี้เรียกว่า โปรซีเดอร์เรียกซ้ำ (recursive procedures)

ตัวอย่าง จงพิจารณาปัญหาของการกลายเป็นเศรษฐีเงินล้านเพื่อให้มีเงินหนึ่งล้านบาทในเวลา 25 ปี นับจากขณะนี้วันนี้เราจะต้องฝากเงินในธนาคารเป็นจำนวนเงินเท่าใดเมื่ออัตราดอกเบี้ยเงินฝากทบต้นอยู่ที่ 8 % ต่อปีวิธีหนึ่งของการเขียนโปรแกรมนี้คือใช้โปรซีเดอร์เรียกซ้ำ

ให้ศึกษาปัญหาในวิธีข้างล่างนี้ : สมมติว่าเรากำลังอยู่สบาย 25 ปี ในอนาคต และขณะนี้มีเงินหนึ่งล้านบาท เราจะมีเงินฝากในธนาคารเท่าใดใน ปีที่ 24 ณ. เวลานั้นอัตราดอกเบี้ย 8 % ต่อปี และให้เงินหนึ่งล้านบาทเมื่อครบ 1 ปี นั่นคือ หนึ่งล้านบาท หารด้วย 1.08

วิธีคิด เงินต้นปีที่ 25 + ดอกเบี้ยอัตรา 8% ฝากครบหนึ่งปีสิ้นปีได้เงิน 1,000,000 บาท

$$\text{เงินต้นปี} + \text{เงินต้นปี} * 8\% = 1{,}000{,}000$$

$$\text{เงินต้นปี} * \left(1 + \frac{8}{100}\right) = 1{,}000{,}000$$

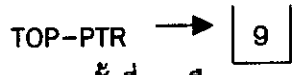
$$\text{เงินต้นปี} * (1.08) = 1{,}000{,}000$$

$$\text{เงินต้นปี} = \frac{1{,}000{,}000}{1.08}$$

เราจะต้องมีเงินในปีที่ 23 เท่าใดจากนั้นลดลงทีละหนึ่งปีเช่นนี้เรื่อยไป โปรแกรม Pascal ข้างล่างนี้ใช้ recursive procedure เรียก compound เพื่อคำนวณหาเงินฝากในปีที่ 1 เพื่อให้ถึงเป้าหมายมีเงินหนึ่งล้านบาทในอีก 25 ปีข้างหน้า

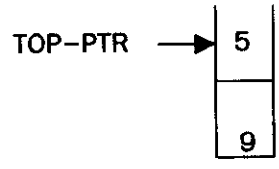
```
                Program retire (output);
                 Const  goal = 1000000 ; rate = 0.08;
                 Var     nowhave : real ; yr : integer;
 1               procedure compound (var yr :integer);
                   { determines  amount in saving 1 year earlier }
 2                 begin if (yr > 0) then
 3                       begin nowhave : = nowhave / (1.0 + rate);
 4                             yr := yr - 1;
 5                             compound(yr)
                        end;
 6                 end; {compound)
 7          begin    yr := 25 ; { main program block }
 8                   nowhave := goal;
 9                   compound(yr);
10                   write(nowhave); writeln;
11          end.
```

ค่าของตัวแปร yr เปลี่ยนแปลงทุกครั้งที่มีการเรียกโปรซีเดอร์ compound เริ่มต้นที่ 25 ไปจนถึง 0 จึงจบโปรแกรมกองซ้อนถูกนำมาใช้โดยระบบเพื่อเก็บ track ว่าโปรแกรม retire อยู่ที่ใด ณ. เวลาที่กำหนดให้ระหว่างกระทำการทุกครั้งที่โปรซีเดอร์ compound ถูกเรียก calling address ถูกนำไปใส่บนกองซ้อนทุกครั้งที่ออกจาก compound กองซ้อนถูก popped และ control ส่งกลับมายัง address ที่เคยอยู่บนสุดการเรียก compound ครั้งแรกอยู่ที่ statement 9 และกองซ้อนควบคุม (control stack) จะแสดงในรูป 4- 6(a)



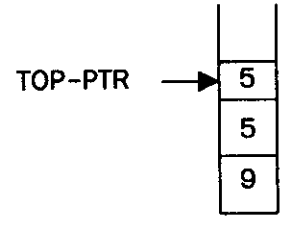
รูป 4-6(a)

การเรียก compound ครั้งที่สองคือจาก statement 5 เมื่อ yr มีค่าเท่ากับ24ขณะที่กองซ้อนควบคุมแสดงในรูป 4-6(b)



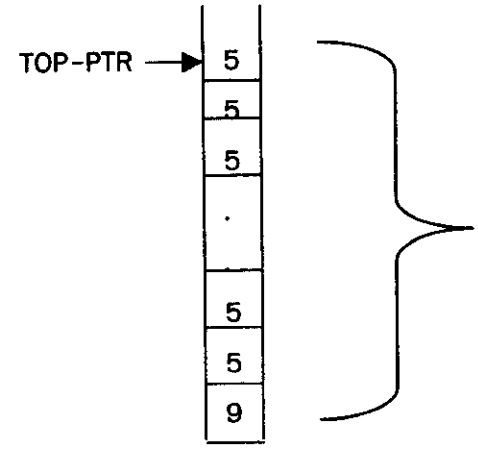
รูป 4-6 (b)

การเรียก compound ครั้งที่สามจาก statement 5 เมื่อ yr มีค่าเท่ากับ 23 ดังนี้ กองซ้อนควบคุมแสดงใน รูป 4-6 (c)



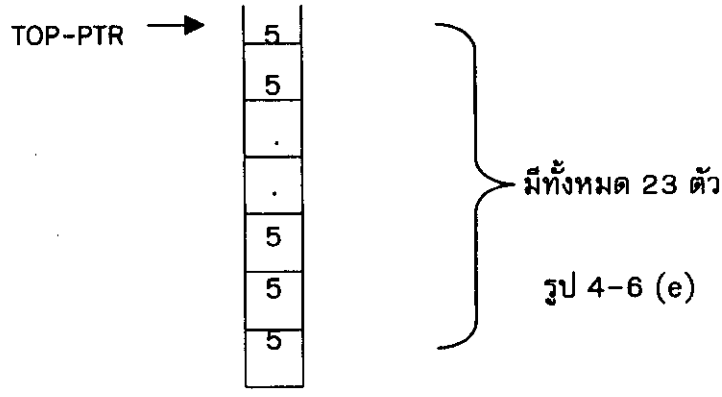
รูป 4-6 (c)

หลังจากเรียกโปรซีเดอร์ compound 25 ครั้ง กองซ้อนแสดงในรูป 4-6 (d)



รูป 4-6 (d)

มีการเรียก 24 ครั้งจาก statement 5 และสุดท้าย yr มีค่าเท่ากับ 0 ขณะนี้กองซ้อนถูกแก้คืนจากการเรียกแต่ละครั้งของการเรียก compound จำเป็นต้องส่งกลับในลำดับ (LIFO) จนกระทั่งถึงการเรียกครั้งแรกเมื่อส่งกลับจากการเรียก compound ท้ายสุด การ popping กองซ้อนเช่นนั้นแสดงในรูป 4-6 (e)



รูป 4-6 (e)

สุดท้าย addresses ทั้งหมดจะถูก popped ออกจากกองซ้อนควบคุมและส่งกลับมายัง statement 9 ข้อความสั่งถัดไปคือ พิมพ์ค่าของ nowhave ซึ่งเป็นจำนวนเงินที่จะต้องนำฝาก เพื่อให้ได้เงินหนึ่งล้านบาทในอีก 25 ปีข้างหน้าตัวอย่างนี้ คอมไพเลอร์ของ Pascal สร้างกองซ้อนและจัดหาคำสั่งต่าง ๆ ที่ถูกต้องสำหรับการใส่และการลบออกจากกองซ้อนในตัวอย่างก่อนหน้านั้น โปรแกรมเมอร์ภาษา COBOL ต้องรับผิดชอบต่อการจัดการ(handle)กองซ้อนโดยตนเองคอมไพเลอร์บางตัวไม่สามารถจัดการการเรียกโปรซีเดอร์แบบ recursive ได้ เพราะว่าไม่มีความต้องการกลไกของกองซ้อน เช่น ภาษา COBOL และ FORTRAN ดังนั้นก่อนที่จะเขียนโปรแกรมควรตรวจสอบจากคู่มือของภาษา

4.3.3 ตัวอย่าง : สัญกรณ์เติมหลัง (Example : Postfix Notation)

งานประยุกต์ชุดที่สามของกองซ้อนคือ ในการแปลนิพจน์คำนวณในภาษาโปรแกรมระดับสูง คอมไพเลอร์จำเป็นต้องแปลรูปแบบปกติของการแทนที่ข้อความสั่งคำนวณ (เรียกว่า สัญกรณ์เติมกลาง) ให้เป็นรูปแบบซึ่งง่ายในการนำไปใช้ก่อกำเหนิดรหัสจุดหมาย (object code) สำหรับตัวดำเนินการแบบทวิภาค (binary operators) เช่น การบวก การลบ การคูณ การหารและการยกกำลัง ตัวดำเนินการในสัญกรณ์เติมกลางอยู่ระหว่างตัวถูกดำเนินการสองตัว ตัวอย่างเช่น

A + B , E ↑ F

กองซ้อนถูกนำมาใช้เพื่อแปลงสัญกรณ์นี้ให้เป็นสัญกรณ์เติมหลัง ซึ่งตัวถูกกระทำทั้งคู่อยู่หน้าตัวดำเนินการ ตัวอย่างเช่น

A B + , E F ↑

รูปแบบนี้ ง่ายขึ้นสำหรับคอมไพเลอร์ที่จะจัดกระทำ

จงพิจารณาอัลกอริทึมข้างล่างนี้ซึ่งแปลงผันนิพจน์จากสัญกรณ์เติมกลางให้เป็นสัญกรณ์เติมหลังกองซ้อนใช้เพื่อเก็บ infix operators จนกระทั่งตัวถูกกระทำของมันถูกราดตรวจ นิพจน์ถูกกราดตรวจจากซ้ายไปขวาทีละตัวอักขระ

มีกฎพื้นฐานสี่ข้อดังนี้

1. If the symbol is " (" it is pushed onto the operator stack.
2. If the symbol is ") " it pops everything from the operator stack down to the first ") " The operators go to output as they are popped off the stack.

   The ") " is popped but does not go to output.
3. If the symbol is an operator, then if the operator on the top of the operator stack is of the same or higher precedence, that operator is poped and goes to output , continuing in this fashion until the first left parenthesis or an operator of lower precedence is met in the operator stack.

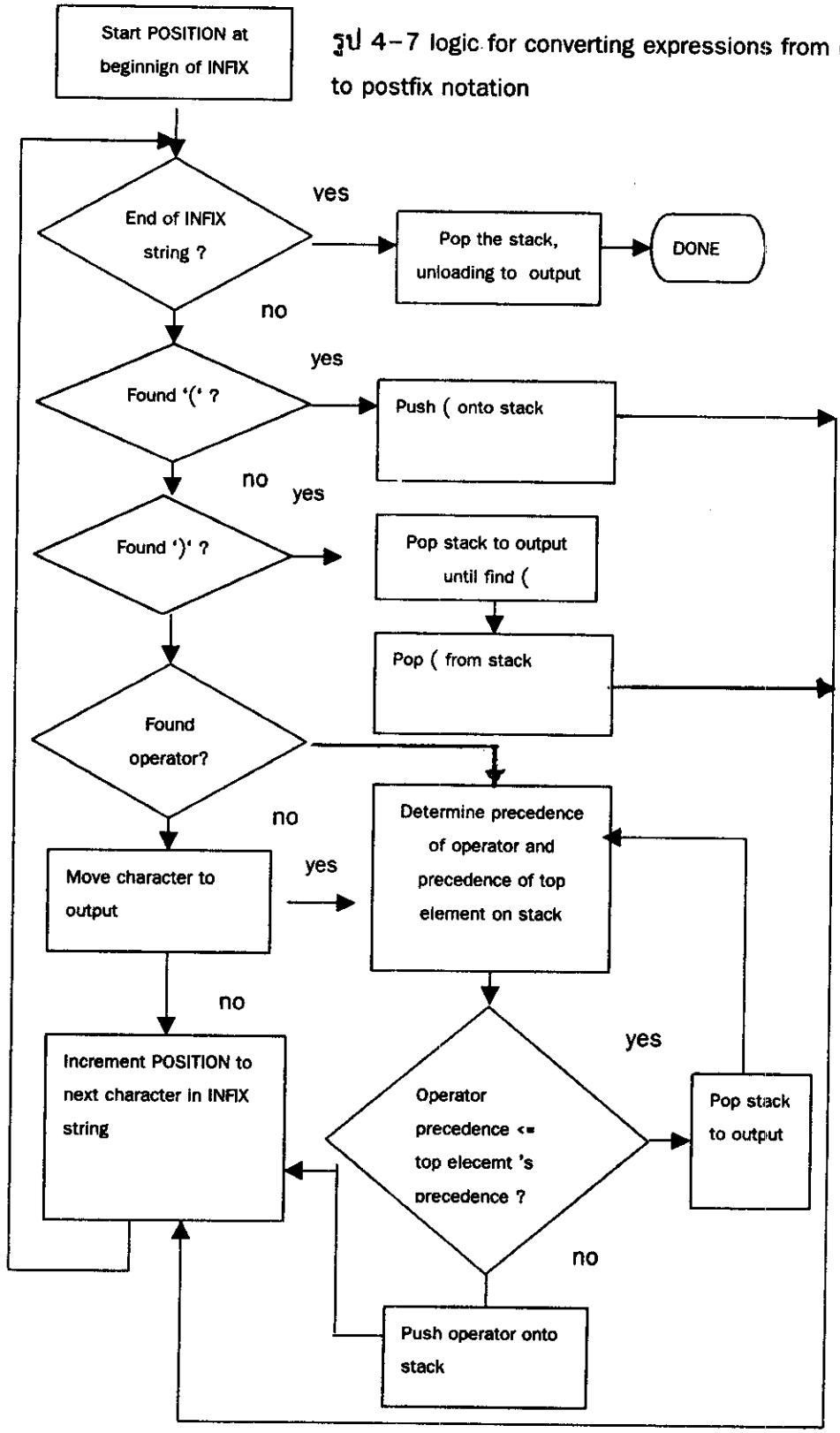
   When that situation occurs, then the current operator is pushed onto the stack.
4. If the symbol is an operand, it goes directly to output.

   The terminating symbol, here the semicolon, pops everything off the stack.

   A flowchart of the algorithm is shown in Fig 4–7

   Let us first consider use of this algorithm with just there operator precedence levels.

Highest :	Exponentiation (↑)
Middle :	Multiplication (*), Division (/)
Lowest :	Addition(+) , subtraction (-)



รูป 4-7 logic for converting expressions from infix to postfix notation

รูป 4-8 แสดง operator stack และเอาต์พุต คือ อักขระแต่ละตัวของนิพจน์คำนวณเติมกลาง

((A+ B) * C/D +E↑F) / G;

ที่ถูกกราดตรวจ

Time point	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Character scanned	(	(	A	+	B	)	*	C	/	D	+	E	↑	F	)	/	G	;
Operator TOP → stack	(	((	((	+ ((	+ ((	(	* (	* (	/ (	/ (	+ (	+ (	↑ + (	↑ + (		/		
Output			A		B	+		C	*	D	/	E		F	↑+		G	/

รูป 4-8 operator stack contents as each character is scanned

นิพจน์เติมหลังที่เป็นผลลัพธ์คือ

AB+C*D/EF↑ + G

โปรดสังเกตว่า วงเล็บทั้งหมดถูกลบออกจากนิพจน์สำหรับนิพจน์เติมกลางนั้นจำเป็นต้องมีวงเล็บไว้เพื่อระบุการทำก่อนของตัวปฏิบัติการ ในสัญกรณ์เติมหลัง การทำก่อนของตัวปฏิบัติการแสดงโดยอันดับของตัวปฏิบัติการ สายอักขระเติมหลัง (postfix string) ถูกกราดตรวจจากซ้ายไปขวา

กำหนดนิพจน์ชั่วคราว ซึ่งเกี่ยวข้องเฉพาะการปฏิบัติการแบบทวิภาคดังนี้

ขั้นที่ 1	T_1 = AB+	สายอักขระคือ T_1 C*D/EF↑ + G/
ขั้นที่ 2	T_2 = T_1 C*	สายอักขระคือ T_2 D/EF↑ + G/
ขั้นที่ 3	T_3 = T_2 D/	สายอักขระคือ T_3 EF↑ + G/
ขั้นที่ 4	T_4 = EF ↑	สายอักขระคือ T_4 T_3 + G/
ขั้นที่ 5	T_5 = T_3 T_4	สายอักขระคือ T_5 G/
ขั้นที่ 6	T_6 = T_5G/	สายอักขระคือ T_6

โปรดสังเกตว่าไม่มีตัวปฏิบัติการใด ๆ ถูกกราดตรวจจนกระทั่งหลังจากมีตัวถูกดำเนินการสองตัวถูกกราดตรวจแล้ว

อัลกอริทึมนี้เขียนโปรแกรมใน COBOL ตัวแปรต่าง ๆ ประกาศใน DATA DIVISION ดังนี้

```
    01 OP-STACK.
       02 STACK OCCURS 10 TIMES  PICTURE X.
       02 TOP-PTR              PICTURE     9(3).
       02 NOEL-MAX             PICTURE 9(3) VALUE 100.
01     IN-STRING.
       02 INFIX OCCURS 100 TIMES PICTURE X.
01     HOUSEKEEPING.
       02 POSITION             PICTURE     9(3).
       02 NEXT-CHAR            PICTURE X.
       02 EOFF                 PICTURE X.
       02 OP-PRECEDENCE            PICTURE 9.
       02 TOP-PRECEDENCE PICTURE 9.
01     PRECEDENCE-RULE.
       02 CHAR-CHECK           PICTURE X.
            88 LEVEL-0 VALUE ' +' , ' -' .
            88 LEVEL-1 VALUE ' * ', ' / '.
            88 LEVEL-2 VALUE ' ↑ ' .
            88 LEVEL-3 VALUE ' ( '.
       02 PRECEDENCE-LEVEL     PICTURE 9.
```

ใน PROCEDURE DIVISION เขียนดังนี้

```
        COMPUTE      TOP-PTR = 0.
        PERFORM SCAN-NEXT-CHAR
                     VARYING POSITION FROM 1 BY 1
                     UNTIL POSITION > NOEL-MAX OR
                     INFIX (POSITION) = ' ; '.
        PERFORM POP-STACK-TO-OUTPUT
                     UNTIL TOP-PTR = 0.

SCAN-NEXT-CHAR.
        MOVE INFIX (POSITION ) TO NEXT-CHAR.
        IF NEXT-CHAR = ' ('
              PERFORM PUSH-STACK.
        ELSE IF NEXT-CHAR = ' ) '
              PERFORM POP-STACK-TO-OUTPUT
                     UNTIL STACK (TOP-PTR) = ' ('
              PERFORM POP-STACK
        ELSE IF NEXT-CHAR = '↑' OR ' * ' OR ' / ' OR ' + ' OR ' - '
                     PERFORM FIND-OP-PRECEDENCE
                     PERFORM FIND-TOP-PRECEDENCE
                     PERFORM EMPTY-STACK-TO-LOWER
                              - PRECEDENCE  UNTIL
                            OP-PRECEDENCE > TOP-PRECEDENCE
                     PERFORM PUSH-STACK
              ELSE DISPLAY NEXT-CHAR.
PUSH-STACK .
        COMPUTE TOP-PTR = TOP-PTR + 1.
        IF TOP-PTR > NOEL-MAX
              error-condition
        ELSE MOVE NEXT-CHAR TO STACK (TOP-PTR).
```

```
POP-STACK.
        IF TOP-PTR = 0
                error - condition
        ELSE MOVE STACK (TOP-PTR) TO EOFF
                COMPUTE TOP-PTR  = TOP-PTR - 1.
POP-STACK-TO-OUTPUT.
        PERFORM POP-STACK.
        DISPLAY EOFF.
FIND-OP-PRECEDENCE.
        MOVE NEXT-CHAR TO CHAR-CHECK.
        PERFORM FIND-PRECEDENCE.
        MOVE PRECEDENCE-LEVEL TO OP-PRECEDENCE.
FIND-TOP-PRECEDENCE.
        IF TOP-PTR > 0
                MOVE STACK (TOP-PTR) TO CHAR-CHECK.
                PERFORM FIND-PRECEDENCE
                MOVE PRECEDENCE-LEVEL TO TOP-PRECEDENCE
        ELSE MOVE 0 TO TOP-PRECEDENCE.
EMPTY-STACK-TO-LOWER-PRECEDENCE.
        PERFORM POP-STACK-TO-OUTPUT.
        PERFORM FIND-TOP-PRECEDENCE.
FIND-PRECEDENCE.
        IF LEVEL-0
                COMPUTE PRECEDENCE-LEVEL = 0
        ELSE IF LEVEL-1
                        COMPUTE PRECEDECE-LEVEL = 1
                ELSE IF LEVEL-2
                        COMPUTE PRECEDENCE-LEVEL = 2
                        ELSE IF LEVEL-3
                                COMPUTE PRECEDENCE-LEVEL = 3
                                ELSE error-condition.
```

อีกหนึ่งทางเลือกของการแทนที่นิพจน์คำนวณ คือ รูปแบบเติมหน้า (prefix form) ซึ่งตัวดำเนินการอยู่หน้าตัวถูกดำเนินการของมัน

ตัวอย่างเช่น +AB, ↑EF

แบบฝึกหัดท้ายบทนี้ ให้เราพิจารณาว่าต้องมีการเปลี่ยนแปลงอะไรบ้างกระทำกับอัลกอริทึมเติมหลังข้างต้นเพื่อให้ก่อกำเนิดสายอักขระซึ่งเป็นรูปแบบเติมหน้าไม่ใช่รูปแบบเติมหลัง

ในแบบฝึกหัดยังแนะนำว่าให้สนับสนุนการแปลงผันอัลกอริทึมเพื่อรวมตัวดำเนินการสัมพันธ์ (< , <= , = , > = , >) , ตัวดำเนินการแบบบลู (and , or, not) และตัวดำเนินการเอกภาค (เช่น - , +) ซึ่งปกติการทำก่อนของตัวดำเนินการเป็นดังนี้

ระดับ 6 : unary- , unary+ , not
ระดับ 5 : ↑
ระดับ 4 : *, /
ระดับ 3 : + , -
ระดับ 2 : <, <=, =, >=,>
ระดับ 1 : and
ระดับ 0 : or

โปรดสังเกตว่ามีการกำหนด high priority นี้ให้กับ COBOL routine ข้างต้นด้วย

4.4 การแปลงส่งไปยังหน่วยเก็บ (Mapping to Storage)

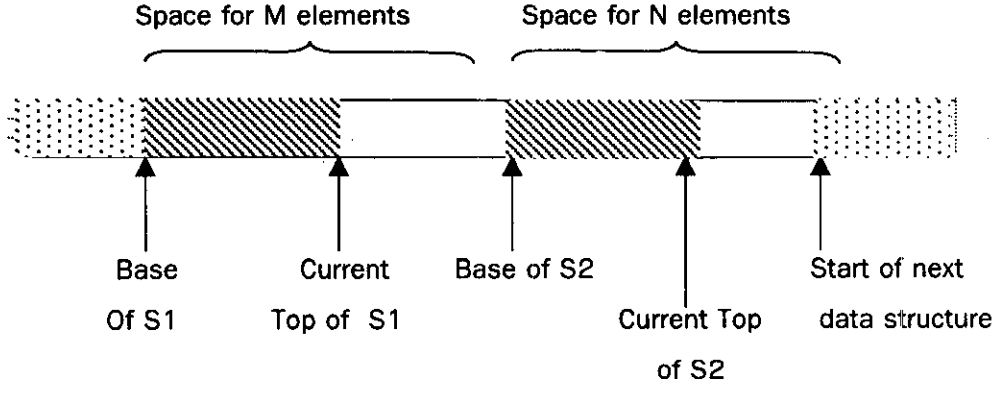
เราได้อภิปรายการใช้แถวลำดับเพื่อเก็บกองซ้อนไปแล้วจะเห็นได้ว่าแถวลำดับหนึ่งมิติโดยทั่วไปแปลงส่งไปยังหน่วยเก็บในลักษณะแบบลำดับเชิงกายภาพ เซตของแบบที่อยู่ที่เป็นพื้นที่ติดกันถูกจัดสรรให้กับสมาชิกของแถวลำดับ การแปลงส่งแบบง่ายที่สุดของกองซ้อนไปยังหน่วยเก็บใช้วิธีเดียวกัน กองซ้อนถูกกำหนดเลขที่อยู่ฐาน(base address)ซึ่งคงที่ตลอดเวลาหลังจากนั้นกองซ้อนจะโตขึ้นในตำแหน่งที่ประชิดกัน

พื้นที่ใช้ร่วมกัน (Sharing Space)

เราได้พิจารณาการจัดสรรหน่วยเก็บเฉพาะกองซ้อนหนึ่งชุดเท่านั้น ขณะนี้สมมติว่ามีกองซ้อนสองชุดจำเป็นต้องอยู่ในหน่วยความจำทั้งคู่ ถ้ากองซ้อน S1 มีสมาชิก

มากที่สุด M ตัวและกองซ้อน S2 มีสมาชิกมากที่สุด N ตัว การจัดสรรหน่วยความจำแสดงในรูป 4-9

S1 and S2 will never run into each other, yet neither of them will ever overflow; together they can accommodate up to N elements. Stack S1 grows to the right and stack S2 grows to the left.

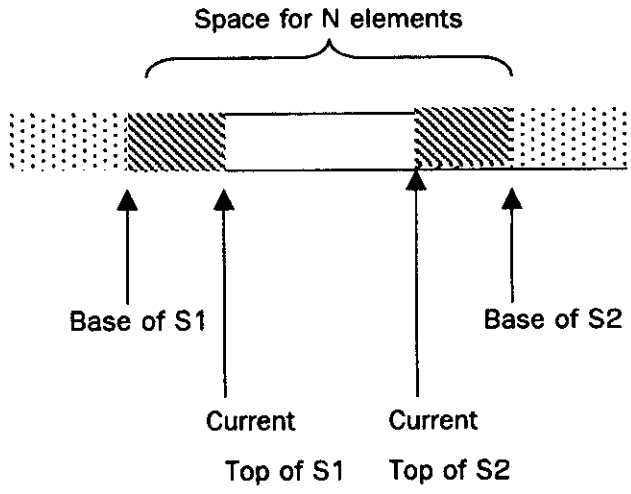


รูป 4-9 Allocation of space to two stacks

สิ่งนี้ไม่มีประสิทธิภาพ โดยเฉพาะถ้ากองซ้อน S1และ S2 ไม่เคยเต็ม ณ.เวลาเดียวกัน จงพิจารณากรณีที่เราทราบแล้วว่าจำนวนสมาชิกใน S1 และS2 รวมกันแล้วไม่เกิน N ตัว

$$Noel\ (S1) + Noel(S2) \leq N$$

อีกหนึ่งทางเลือกคือจัดสรรเนื้อที่เฉพาะเมื่อรวมกันแล้วสูงสุดเท่ากับ N ให้กองซ้อนสองชุดหันหน้าเข้าหากัน โดยที่กองซ้อนแต่ละชุดโตขึ้นเข้าหากัน ดูรูป 4-10



รูป 4-10 Alternative allocation of space to two stacks, growing toward each other

แบบฝึกหัด

1. จงเติมคำในช่องว่าง
   (a) ถ้า pop กองซ้อนว่างผลลัพธ์คือ. . . . . .
   (b) ถ้า push สมาชิก i บนกองซ้อนอย่างถูกต้อง. . . . . .
   Top = ...
   (c) Top กองซ้อนว่าง =. . . . . .
   (d) ถ้า pop และจากนั้น push สมาชิก i
   Top = ...
   (e) กองซ้อนสร้างใหม่มีสมาชิกกี่ตัว. . . . . .
2. จงบอกผลลัพธ์ของข้อความยืนยันข้างล่างนี้
   (a) Isempty (Create(S)) คือ. . . . . .
   (b) Isempty (Push(i,S)) คือ. . . . . .
   (c) Pop(Create(S)) คือ . . . . . .
3. จงบอกผลลัพธ์ของ Top(Push(i,s)) . . . . . .
4. จงบอกผลลัพธ์ของ Pop (Push(i,S))
5. กองซ้อนถูกทำให้เกิดผลในทางปฏิบัติใน FORTRAN อย่างไร
6. จงเขียนอัลกอริทึมใส่สมาชิกบนกองซ้อน
7. จงเขียนอัลกอริทึมลบสมาชิกออกจากกองซ้อน
8. จงเขียนอัลกอริทึมสร้างและบรรจุกองซ้อน
9. จงเขียนรายการประยุกต์ของกองซ้อนสามชนิดและให้เหตุผลว่าทำไมแต่ละชนิดการใช้กองซ้อนจึงดีกว่าใช้แถวลำดับ
10. อัลกอริทึม Push ตรวจสอบเงื่อนไขส่วนล้น (overflow condition) และอัลกอริทึม POP ตรวจสอบเงื่อนไขน้อยเกินเก็บ (underflow condition) จงให้ความหมายของ overflow และ underflow ทำไมจึงตรวจสอบเงื่อนไขเช่นนั้นอะไรคือกิจกรรมที่เหมาะสมเพื่อกระทำกับเงื่อนไขแต่ละชนิดนั้น
11. จงเขียนรูปแบบเติมหลัง (postfix form) สำหรับนิพจน์ข้างล่างนี้
   (a) A*B-(C+D) – (E- F) + F/H ↑ I
   (B) ((B*C) +C/D↑F) + G
   (C) A ↑ B * C- D + E/F/(G+H)
12. จงเขียนอัลกอริทึมลบ item ออกจากกองซ้อนและกำหนดให้ค่าที่ลบออกให้กับตัวแปร x

13. จงเขียนโปรแกรมซึ่งใช้กองซ้อนเพื่อตรวจสอบการจับคู่วงเล็บเปิดและวงเล็บปิด (วงเล็บเล็ก , วงเล็บใหญ่ และวงเล็บปีกกา) ในสายอักขระหนึ่งชุด
14. จงเขียนอัลกอริทึมซึ่งใช้กองซ้อนเพื่อแปลงผันนิพจน์คำนวณจากสัญกรณ์เติมกลาง (infix notation) ให้เป็นสัญกรณ์เติมหน้า (prefix notation)
15. จงเขียนโปรแกรมเพื่อทำให้เกิดในทางปฏิบัติของอัลกอริทึมแปลงผัน infix – to-prefix
16. จงเขียนโปรแกรม Pascal เพื่อแปลงผันนิพจน์คำนวณจากสัญกรณ์เติมกลางให้เป็นสัญกรณ์เติมหลัง