บทที่ 3

การออกแบบภาษา : ความหมาย

(Language Design : Semantics)

3.1 เครื่องคอมพิวเตอร์อย่างง่าย และแบบจำลองสัญลักษณ์

3.2 ชนิด การผูกโยง ตัวปฏิบัติการ และการบังคับ

3.3 การจัดสรรหน่วยเก็บ

3.4 โครงสร้างควบคุม

3.5 โปรซีเดอร์ และพารามิเตอร์

แบบฝึกหัด

บทที่ 3

การออกแบบภาษา : ความหมาย

(Language Design : Semantics)

ตอนจบบทที่ ความคิดของการผูกโยงที่แข็งแรง (strong bond) ระหว่างวากยสัมพันธ์ของภาษา และพฤติกรรมขณะเวลาดำเนินงานของมัน หรือ "ความหมาย" ได้ถูกสร้างขึ้น ซึ่งได้แสดงให้เห็นแล้วว่า การกระจายของนิพจน์ สามารถใช้เพื่อทำตามลำดับการประเมินผล สำหรับการปฏิบัติการแต่ละชนิดอย่างไร

ในบทนี้ เราจะดูรายละเอียดความหมายของภาษาชุดคำสั่งมากขึ้น จำแนกหัวข้อสำคัญเช่นเดียวกับผลเฉลยของมัน ในการทำภาษาชุดคำสั่งร่วมสมัยให้เกิดผล ชนิดสำคัญของหัวข้อเหล่านี้ ได้แก่

- ชนิด การผูกโยง ตัวปฏิบัติการ และการบังคับ
  (Types, binding, operators, and coercion)
- การจัดสรรเนื้อที่หน่วยเก็บ (Storage allocation)
- โครงสร้างควบคุม (Control structures)
- โปรซีเดอร์และพารามิเตอร์ (Procedures and parameters)
- สิ่งแวดล้อมเวลาดำเนินงาน (Run-time environment)

การอภิปรายในหัวข้อเหล่านี้ เริ่มต้นโดยการแนะนำแบบจำลองการแทนที่ ของการจัดองค์กรคอมพิวเตอร์ อย่างง่าย (a simple representative model of computer organization) การกำหนดเลขที่อยู่ (addressing) เซตคำสั่ง (instruction set) สิ่งเหล่านี้ จะนำมาใช้ตลอด ในส่วนที่เหลือของบทนี้ เพื่อแสดงให้เห็นรายละเอียดการทำให้เกิดผล เกี่ยวกับหัวข้อความหมายหลายหัวข้อ ในการออกแบบภาษาชุดคำสั่ง

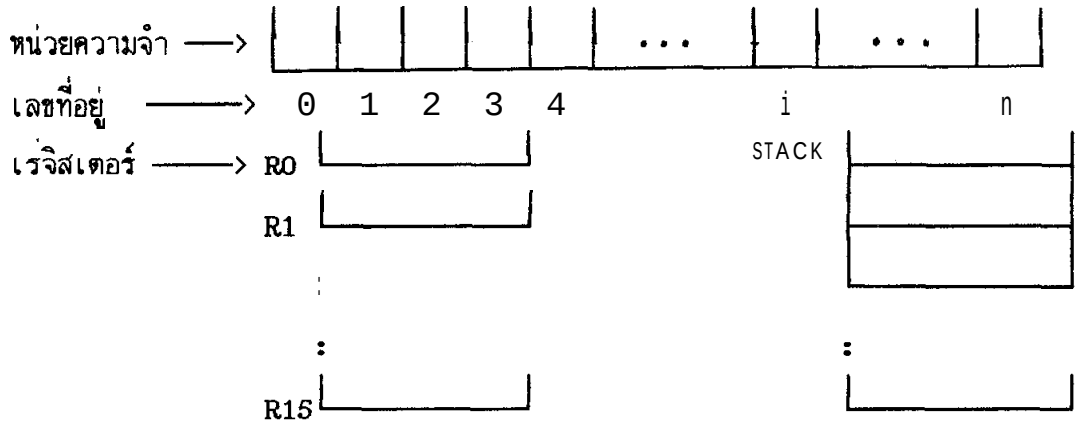
3.1 เครื่องคอมพิวเตอร์อย่างง่าย และแบบจำลองสัญลักษณ์

(A simple machine and notational model)

สมมติว่าผู้อ่าน มีประสบการณ์ เกี่ยวกับการเขียนโปรแกรมภาษาแอสเซมบลี และการจัดองค์กรคอมพิวเตอร์ มาแล้ว อาจเป็นเครื่อง VAX คอมพิวเตอร์ระดับใหญ่ IBM หรือตัวประ-

มวลผลแบบจุลภาค เช่น 8086 แบบจำลองและสัญลักษณ์ที่จะแนะนำในที่นี้ เป็น mirror ของเครื่องชนิดนี้ สำหรับเหตุผลเชิงการเรียนการสอน คือ ความง่ายของเครื่องจริง

ในแบบจำลองนี้ สมมติว่า หน่วยความจำหลักประกอบด้วย ไบต์ชนิด 8 บิต (eight-bit bytes) การเรียงลำดับเลขที่อยู่ เป็นเลขที่อยู่ฐานสิบ นอกจากนี้แล้ว สมมติว่ามีกลุ่มของเรจิสเตอร์ 32 บิต ความเร็วสูง 16 ตัว (16 high speed thirty-two-bit registers) ใช้สำหรับการกำหนดเลขที่อยู่ และการคำนวณ และเลขที่อยู่ของมัน กำหนดโดยสัญลักษณ์ R0, R1, ... และ R15 สมมติต่อไปว่า มีกองซ้อนเวลาดำเนินงาน (run-time stack) หนึ่งชุด มูลค่าของ ไบต์บนสุดของมัน กำหนดเลขที่อยู่เป็น STACK, STACK-1, STACK-2 และอื่นๆ ทั้งหมดแสดงด้วยภาพข้างล่างนี้



ไบต์ติดกัน (adjacent bytes) ของหน่วยความจำ รวมกลุ่มกันสองไบต์ เรียกว่า "words" รวมกันสี่ไบต์ เรียกว่า "longwords"

built-in data types ข้างล่างนี้ สมมติให้กับแบบจำลองคอมพิวเตอร์เครื่องนี้

Type	Code	Memory (bytes)	Range of values
Binary integer	I	2	-2^{15} to $2^{15}-1$
Long integer	L	4	-2^{31} to $2^{31}-1$
Real (float)	E	4	$+10^{75}$, six-digit precision

Type	code	Memory (bytes)	Range of values
Double precision real	D	8	$+10^{75}$, sixteen-digit precision
Address	A	4	0 to $2^{32}-1$
Boolean (logical)	B	1	true, false
Character (ASCII)	C	1	128-character ASCII set
Packed decimal	P	1/2 byte per decimal digit, plus 1/2 byte for sign, rounded to whole byte	

จากตารางข้างต้น ข้อมูลชนิดพื้นฐานเหล่านี้ มีหลายตัว ต้องการเนื้อที่มากกว่าหนึ่งไบต์ ในแต่ละกรณี การอ้างถึงข้อมูล กำหนดโดยเลขที่อยู่ไบต์ซ้ายมือสุดของมัน เพื่อความสะดวกในการเรียนการสอน เลขที่อยู่ หน่วยความจำทั้งหมด และมูลค่าตัวเลขของมัน จะแสดงให้เห็นด้วยสัญลักษณ์ฐานสิบ ตัวอักขระ ทั้งหมดจะเขียน เช่นที่ปรากฏ และค่าบูลีน จะเขียนเป็น **true** และ **false** เซตของคำสั่งของคอมพิวเตอร์เครื่องนี้ มีอยู่สองชนิดดังนี้

(i) opcode operand

(ii) opcode operand1, operand2

คำอธิบาย (comment) จะมีเครื่องหมาย ; นำหน้า บนบรรทัดนั้น กรณีอื่นๆ คำสั่งต้องเป็น หนึ่ง ใน สองรูปแบบข้างต้น รหัสดำเนินการ (opcode) สำหรับคำสั่งหนึ่ง ตัวถูกดำเนินการ (ชนิดที่ i) นิยามดังนี้

Opcode	Meaning
B	แตกกิ่ง หรือย้ายไป เลขที่อยู่ กำหนดโดย "operand"

Opcode	Meaning
CALL	เรียกโปรซีเดอร์ ซึ่งคำสั่งแรก อยู่ที่เลขที่อยู่ กำหนดโดย "operand" และใส่เลขที่อยู่ส่งคืน บนตอนบนของกองซ้อนเวลาดำเนินการ
RETURN	ย้ายไปยังคำสั่ง ซึ่งเลขที่อยู่คืออยู่ที่ตอนบน ของกองซ้อนเวลาดำเนินการ แล้วเอาเลขที่อยู่นั้นออกจากกองซ้อน ไปไว้ที่ตำแหน่ง กำหนดโดย "operand"
PUSH	ใส่ค่าเลขที่อยู่ ของ "operand" บนกองซ้อน
POP	เอาค่าบนสุด ของกองซ้อนออก และไปเก็บที่เลขที่อยู่ "operand"

รหัสดำเนินการชนิดที่ ii ซึ่งมีตัวถูกดำเนินการสองตัว โดยสรุปเป็นดังนี้

Opcode	Meaning
ADD	บวก long integers ณ เลขที่อยู่ กำหนดโดย"operand1" กับ "operand2" เก็บผลลัพธ์ที่ longword address "operand2"
SUB, MUL, DIV	นิยามเช่นเดียวกับ ADD
MOV	ย้าย londword จากเลขที่อยู่ "operand1" ไปยัง เลขที่อยู่ "operand2"
CMP	เปรียบเทียบ longword integers ณ เลขที่อยู่ "operand1" กับ เลขที่อยู่ "operand2" และกำหนดไบต์ STATUS (ดูหน้าถัดไป) ที่ต้องกัน

Opcode	Meaning
GETMAIN	ร้องขอ จากระบบปฏิบัติการ บล็อกของหน่วยความจำหลัก ซึ่งประกอบด้วย "operand1" ไบต์ที่ต่อกัน และปล่อยเลขที่อยู่ของ ไบต์ที่หนึ่งของมันใน longword addressed by "operand2"
FREEMAIN	ปล่อยไปยัง ระบบปฏิบัติการ บล็อกของหน่วยความจำ นิยามโดย "operand1" และ "operand2" เช่นเดียวกับ GETMAIN
CVTxy	แปลงผัน (convert) ค่า ณ เลขที่อยู่ "operand1" ชนิด x ให้เป็นค่าเหมือนกัน ชนิด y และเก็บผลลัพธ์ ณ เลขที่อยู่ "operand2" ในที่นี้ x และ y อาจจะเป็น ข้อมูลชนิดพื้นฐานใด ๆ ของคอมพิวเตอร์ ซึ่งสรุปไว้ตอนต้น (I, L, E, D, A, B, C หรือ P) ตราบใดที่ การแปลงผัน มีเหตุผล ข้อผิดพลาดของการแปลงผัน กำหนดไว้ในไบต์ STATUS ซึ่งจะอธิบายต่อไป การแปลงผัน ของ ค่าความยาวต่าง ๆ ของตัวแปร (ชนิด C และ P) จะอธิบายต่อไป

คำสั่งคำนวณ (ADD, SUB, MUL, DIV, CMP, MOV) มีหลากหลาย สำหรับชนิดอื่นๆ นอกเหนือจากชนิด longword integers, ชนิดต่างๆ เหล่านี้ กำหนดโดย การนำ รหัสชนิดที่เหมาะสม (I, E, D, B หรือ C) มาต่อ กับ รหัสดำเนินการพื้นฐาน

ตัวอย่างเช่น เราอาจกำหนดการบวกของ จำนวนเต็ม สิบหกบิต ด้วย ADDI, การบวก floating-point ด้วย ADDE และอื่น ๆ ในแต่ละกรณี ค่าของตัวถูกดำเนินการ ที่เกิดในการปฏิบัติการ ทั้งคู่ ต้องเป็นชนิดเดียวกัน

คำสั่งคำนวณและการแปลงผัน (CVTxy) บางครั้ง ใช้กับ ค่าของตัวถูกดำเนินการ ซึ่งเป็น ความยาวแปรเปลี่ยนได้ (เช่น ตัวถูกดำเนินการชนิด C และ P) ในกรณีเหล่านี้ การแทนที่ภายใน สมมติว่า ถูกนำหน้าโดย ค่าจำนวนเต็มสิบหก-บิต ซึ่งให้ ความยาวของตัวถูกดำเนิน-

การ มีหน่วยเป็นไบต์ ตัวอย่างเช่น การแทนที่ ทศนิยมอัดแน่น (the packed decimal representation) ของ -21 ใช้เนื้อที่ สี่ไบต์, และ การแทนที่ชนิดไม่อัดแน่น (unpacked) ของมัน เช่น สายอักขระ ใช้เนื้อที่ ห้าไบต์ สองไบต์ สำหรับความยาวและสามไบต์ สำหรับค่า -21 สิ่งนี้ สมมติว่าเป็นผลลัพธ์โดยตรงของ การปฏิบัติการ CVTPC เมื่อ "operand1" หมายถึง เลขที่อยู่ของค่าของทศนิยมอัดแน่น และ "operand2" หมายถึง เลขที่อยู่ของ การแทนที่ตัวอักขระเป้าหมาย

ไบต์ STATUS เป็น ตำแหน่งควบคุมพิเศษ ซึ่ง กำหนดโดยระบบ เมื่อคำสั่งถูก execute สรุป ดังนี้

Instruction	STATUS byte setting
ADD SUB MUL DIV CVT	0 = no exceptional conditions 1 = overflow (out of range for the given type) 2 = division by zero 3 = operand stored is not of the correct type
CMP	0 = operands equal 1 = operand1 < operand2 2 = operand.1 > operand2 3 = operand stored is not of the correct type

Instruction	STATUS byte setting
PUSH, POP, GETMAIN, FREEMAIN	0 = no exceptional conditions 1 = stack overflow or underflow (PUSH or POP) or failure to complete GETMAIN or FREEMAIN

นอกจากนี้แล้ว ความหลากหลายของ รหัสดำเนินการ ย้าย (B) ข้างล่างนี้ เป็นการแตกกิ่งอย่างมีเงื่อนไข

Opcode	Meaning
BOV	Transfer on overflow, stack overflow, GETMAIN or FREEMAIN error (STATUS = 1)
BZD	Transfer on zero divide (STATUS = 2)
BTY	Transfer on improper operand type (STATUS=3)
BE, BZ	Transfer on equal or zero result (STATUS=0)
BL	Transfer on less (STATUS = 1)
BG	Transfer on greater (STATUS = 2)
BLE	Transfer on less or equal (STATUS = 0 or 1)
BGE	Transfer on greater or equal (STATUS = 0 or 2)
BNE	Transfer on not equal (STATUS = 1 or 2)

ตัวถูกดำเนินการ (operands) ในภาษานี้ปกติ หมายถึง เลขที่อยู่ กำหนดโดย สัญลักษณ์หนึ่งตัว สัญลักษณ์แต่ละตัวอาจเป็นลำดับ 1 ถึง 8 ตัวอักษร เช่น A, B, LOOPS, SUM และอื่นๆ สัญลักษณ์ถูกนิยามโดยการปรากฏทางซ้ายมือ ตอนหน้าของโปรแกรม สัญลักษณ์หนึ่งตัวอาจนิยาม หนึ่ง statement เพื่อเป้าหมาย สำหรับการจัดจุดแตกกิ่ง วิธีง่าย คือ วางตรงตำแหน่งซ้ายมือ ก่อน รหัสดำเนินการ ของ คำสั่งนั้น สัญลักษณ์หนึ่งตัว อาจนิยามเนื้อที่หน่วยเก็บของชนิดข้อมูล อาจจะเริ่มต้นโดย ค่าหนึ่ง ในวิธีข้างล่างนี้

```
symbol    DS    nt
symbol    DC    ntv
```

ในที่นี้ DS ใช้แทน นิยามหน่วยเก็บโดยไม่กำหนดค่าแรกของมัน (define storage without initializing its value)

DC ใช้แทน นิยามค่าคงที่ (define constant) หมายถึง นิยามหน่วยเก็บโดยกำหนดค่าแรก (define storage with initialization)

n อาจจะมีหรือไม่มีก็ได้ หมายถึง integer repetition factor

v หมายถึงค่าแรกที่ให้เก็บ (the initial value to be stored)

ค่าของ t คือ รหัสชนิด built-in อย่างใดอย่างหนึ่งซึ่งได้อธิบายมาแล้ว กับ ความต้องการพิเศษ คือ ความยาว (มีหน่วยเป็นไบต์) สำหรับสายอักขระ (ชนิด C) และ หน่วยเก็บทศนิยมอัดแน่น (ชนิด P) ต้องติดกับ t

ตัวอย่าง แสดงให้เห็นข้อตกลงเหล่านี้

Declaration	Meaning
X D S E	x เป็นค่าจำนวนจริง
A DS 5E	A เป็นแถวลำดับ ของค่าจำนวนจริง 5 ตัว
PI DC E3.14159	P I เป็นค่าคงที่จำนวนจริง มีค่าเท่ากับ 3.14159
M DS Cl0	M เป็นสายอักขระความยาว 10 ไบต์

Declaration	Meaning
M DC C'hello'	M เป็นค่าคงที่สายอักขระความยาว 5 ไบต์ มีค่าเท่ากับ "hello"

ตัวถูกดำเนินการ อาจกำหนดเป็นเรจิสเตอร์ โดยใช้ สัญลักษณ์พิเศษ R0 ถึง R15, อาจกำหนดเป็น กองซ้อนเวลาดำเนินงาน โดยใช้ สัญลักษณ์พิเศษ STACK และตัวแปรต่าง ๆ เพิ่มเติมต่อไปนี้ (เมื่อ X อาจเป็นเลขที่อยู่คงที่ใดๆ, สัญลักษณ์, เลขเรจิสเตอร์, หรือ STACK)

X = มูลค่าของตำแหน่ง X (the contents of location X)

@X = มูลค่าของเลขทีอยู่ กำหนดโดย ตำแหน่ง x (การกำหนดเลขที่อยู่โดยอ้อม) (the contents of the address given by location X (indirect addressing))

x t c (หรือ X-c) = มูลค่าของตำแหน่ง เข้าแทนที่ c ไบต์ จากตำแหน่ง x เมื่อ c หมายถึงค่าคงที่จำนวนเต็ม (the contents of location displaced c bytes from location X, where c denotes an integer constant.)

$X(R_i)$ = มูลค่าของตำแหน่ง เข้าแทนที่ จาก x โดย มูลค่า ของ เรจิสเตอร์ R_i (i = 0, ... 15) นั่นคือ, การกำหนดเลขทีอยู่โดยดรรชนี (the contents of location displaced from X by the contents of register R_i) (i = 0, ... 15); that is indexed addressing)

'string' = any literal character string

#n = any literal integer value n

ไม่มีการจัดการเกี่ยวกับอินพุท-เอ้าพุท (no input-output provisions) รวม

อยู่ในการอธิบายเครื่องคอมพิวเตอร์ โดยย่อนี้ เราสมมติว่า การจัดการอินพุท-เอ้าพุท เสร็จเรียบร้อยแล้ว ผ่าน กลุ่มที่เหมาะสมของ macros ซึ่งจะได้อธิบายภายหลังในบทที่จำเป็น ภาษาเครื่องพื้นฐาน ภายใต้สถาปัตยกรรมที่เป็นอยู่ ความเกี่ยวข้องอื่นๆ ทั้งหมด สมบูรณ์อย่างมีเหตุผล สิ่งนี้จะพอเพียงแน่นอนสำหรับทำให้เกิด หลักของ ความหมายภาษาชุดคำสั่ง ในส่วนที่เหลือของบทนี้ (The basic machine language and underlying architecture are, in all other respects, resonably complete. This will certainly be adequate to motivate the principles of programming language semantics in the remainder of this chapter.)

3.2 ชนิด การผูกโยง ตัวปฏิบัติการ และการบังคับ

(Types, binding, operators, and coercion)

หัวข้อความหมายอันดับแรก เกิดขึ้นเมื่อมีการพิจารณา การทำให้เกิดผลของภาษาชุดคำสั่ง นั่นคือ การแทนที่ ชนิดข้อมูลพื้นฐานต่าง ๆ ในภาษานั้น (when considering the implementation of programming languages is that of representation for the basic data types in the language.) เช่นที่เราสามารถเห็น ภาษาเครื่อง ของเราสนับสนุนโดยตรงกับ ชนิดพื้นฐานส่วนใหญ่ ใน ภาษา Pascal, FORTRAN, COBOL และ PL/1 ความสมนัยนี้ คือ การกล่าวถึงประวัติภาษา และได้สรุปในตารางข้างล่างนี้

Machine type	Corresponding type in ...			
	Pascal	FORTRAN	COBOL	PL/1
I-integer	integer	INTEGER*2	COMP	FIXED BIN(15)
L-long integer	—	INTEGER	COMP-1	FIXED BIN(31)
E-real	real	REAL	COMP-2	FLOAT DEC(6)
D - long real	—	DOUBLE PRECISION	—	FLOAT DEC(16)

Machine type	Corresponding type in . . .			
	Pascal	FORTRAN	COBOL	PL/1
A - address	reference	—		ADDR or POINTER
B-Boolean	Boolean	LOGICAL	—	BIT(1)
C-string	array of Char	CHARACTER	PIC x	CHAR
P-packed decimal	—	—	COMP-3	FIXED DEC

ดังนั้น ความต้องการหน่วยเก็บขณะเวลาดำเนินงาน (run-time storage) สำหรับภาษา ซึ่งตัวแปรต่าง ๆ ได้จัดสรร แบบคงที่ คำนวณโดยใช้ ชนิดนี้ ของความสมนัยเป็นเกณฑ์อย่างหนึ่ง นอกจากนี้แล้ว เพราะว่าตัวแปรทั้งหมดในบางภาษา (โดยเฉพาะ ภาษา FORTRAN และ COBOL) มีการจัดสรรแบบคงที่ ความต้องการหน่วยเก็บ ณ เวลาดำเนินงาน และการจัดสรร สามารถทำให้เสร็จสิ้น ณ เวลาแปลชุดคำสั่ง (การจัดสรรหน่วยเก็บ สำหรับแถวลำดับ, ระเบียน และโครงสร้างเชิงรายการ (list structure) จะอธิบายแยกต่างหาก ในหัวข้อต่อไป)

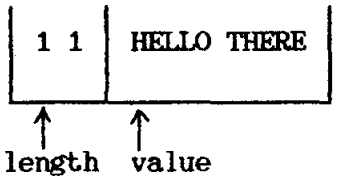
การจัดสรรหน่วยเก็บ สำหรับสายอักขระ และเลขฐานสิบ

(Storage Allocation for Character Strings and Decimal Numbers)

เนื่องจากต้อง ยอมรับข้อตกลงบางอย่าง สำหรับการแทนที่ ความยาวของข้อมูลชนิดความยาวแปรได้ เหล่านี้ การออกแบบภาษา ต้องจัดหาวิธีของการกำหนดความยาวนั้น นอกจากนี้แล้ว ถ้า ความยาวของข้อมูลชนิดนั้น อนุญาตให้ แปรได้ (vary) ระหว่างการปฏิบัติการ ไม่ว่าจะเป็นความยาวสูงสุด ซึ่งต้องระบุในภาษา หรือ การทำให้เกิดผล ต้องเตรียมให้กับ การจัดสรรหน่วยเก็บจอภาพแบบพลวัต (dynamically monitor storage allocation) การแตกกระจาย (fragmentation) และ การเอาโปรแกรมกลับมา (reclaimation) ในวิธีซึ่งชัดเจน

การกำหนดความยาวสูงสุดชัดแจง ถูกสนับสนุน ในภาษา PL/1, SNOBOL ตรงกันข้ามกับยอมให้ สายอักขระแปรเปลี่ยนได้อย่างสมบูรณ์ และ สมมติ ความรับผิดชอบ ทั้งหมด สำหรับการจัดการหน่วยเก็บแบบพลวัต (dynamic storage management)

ถ้าต้องการ สายอักขระความยาวสูงสุด n (ตัวอย่างเช่น การประกาศ ในภาษา PL/I เป็น CHAR(n) VARYING) ดังนั้น การออกแบบเครื่องของเรา ต้องการ บล็อกของหน่วยเก็บความยาว n+2 สำรองไว้ สองไบต์แรก เก็บความยาวปัจจุบัน ของสายอักขระ และไบต์ที่เหลือเก็บค่าปัจจุบันของมัน ตัวอย่างเช่น สายอักขระ "HELLO THERE" จะถูกเก็บดังนี้



สิ่งนี้ต้องใช้เนื้อที่ทั้งหมด 13 ไบต์ ข้อมูลชนิดนี้ สนับสนุนโดยตรงในส่วนขยายของ Pascal (เรียกว่า **string**), FORTRAN และ COBOL เช่นเดียวกับ PL/1

สายอักขระอาจถูกจัดสรร ณ เวลาดำเนินงาน ในบางภาษา เช่น PL/1 และ SNOBOL ในกรณีเช่นนี้ ตัวแปลชุดคำสั่ง ต้อง ก่อกำเนิดรหัส (generated code) เพื่อให้ทำสองสิ่งข้างล่างนี้ ให้เป็นผลสำเร็จ

(i) เอาปริมาณหน่วยเก็บ ซึ่งเท่ากับ ความยาวสูงสุดของตัวแปรที่ได้ประกาศไว้ หรือค่าแรก (Obtain an amount of storage which satisfies the variable's declared maximum length or initial value.)

(ii) กำหนดค่าแรก ให้กับตัวแปรนั้น (Initialize the value of that variable)

ตัวอย่าง คำสั่งภาษา PL/1

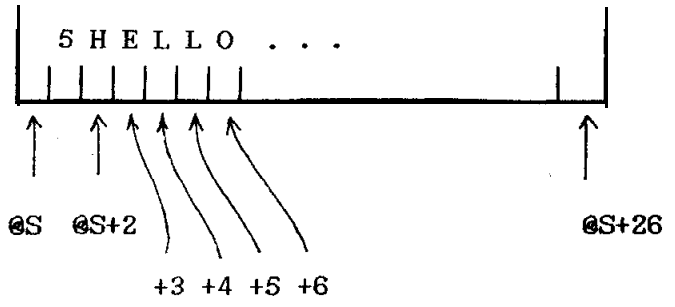
```
DCL S CHAR(25) VAR INIT ('HELLO');
```

ณ เวลา ซึ่งโปรซีเดอร์ที่มี การประกาศนี้ ถูกใช้งาน (activated) รหัสเครื่อง ข้างล่างนี้ ต้องถูก กระทำการ (execute)

```
GETMAIN #27, S          ; obtain 27 bytes for S
MOVI    #5, @S          ; initialize the length of S
```

```
MOVC    'HELLO', @S+2      ; initialize value of S
```

ดังนั้น การเริ่มต้นหน่วยเก็บข้างล่างนี้ สำหรับ s จะเกิดขึ้น



ขณะนี้การอ้างถึงสายอักขระนี้ จะถูกกระทำโดยนิพจน์ @S+2, และการอ้างถึงทั้งหมดของ ความยาวของมัน จะกระทำโดย นิพจน์ @S

<u>การผูกโยง</u> (Binding) ในสาระทั่วไปส่วนใหญ่ของมัน แนวคิดของ "การผูกโยง" เสนอแนะ การกระทำของสมาชิกในโปรแกรมที่เกี่ยวข้องกัน (เช่น ตัวแปร คำสั่ง หรือโปรซีเดอร์) ด้วยความหมายอย่างหนึ่ง หรือคุณสมบัติเวลาดำเนินงาน (เช่น ชนิด บล็อกของหน่วยเก็บ หรือค่าข้อมูล) (In its most general sense, the notion of "binding" suggests an act of associating a particular program element (such as a variable, a statement, or a procedure) with a particular semantic, or run-time property (such as a type, a block of storage, or a data value).)

การผูกโยงบางอย่างเกิดขึ้นเมื่อเขียนโปรแกรม เช่น การเลือก ชนิดข้อมูล ให้กับตัวแปร บางอย่างเกิดขึ้น เมื่อโปรแกรมถูกแปล เช่น การเลือกคำสั่งเครื่อง ให้กับ หนึ่งคำสั่ง ยังมีการผูกโยงอื่นๆ อีก ซึ่งจะไม่เกิดขึ้น จนกระทั่งเวลาดำเนินงาน เช่น การจัดสรรหน่วยเก็บให้กับตัวแปรเฉพาะที่ใน โปรซีเดอร์ถูกเรียก สุดท้าย เราสามารถมองที่เวลา เมื่อเริ่มต้นการนิยามภาษา (ปกติเกิดขึ้นโดย นักเขียนโปรแกรม) บางทีเป็นเวลาผูกโยงที่สำคัญที่สุดของทั้งหมด ณ เวลานั้น เช่น สมาชิก เป็นเซตของชนิดที่เป็นไปได้ทั้งหมด, เซตของ statements และโครงสร้างควบคุม และอื่น ๆ ถูกเลือกขึ้น และกำหนด การแทนที่วากยสัมพันธ์ และ

ความหมายถูกต้องโดยแท้ ให้ภาษานั้น

ตัวอย่าง จงพิจารณาการผูกโยงของตัวแปร ชื่อ I กับตำแหน่งหน่วยเก็บ ชนิด integer ในหลายภาษา (Pascal, FORTRAN, COBOL, PL/1) สิ่งนี้ กระทำแล้ว เมื่อเขียนโปรแกรม ไม่ว่าจะเป็นการประกาศชัดแจ้ง หรือโดยนัย (กฎ I ถึง N) ของ FORTRAN ในภาษา SNOBOL การผูกโยงนี้ จะไม่เกิดขึ้น จนกระทั่งถึงเวลาดำเนินงาน (run time) เพราะว่า ชนิดของ ตัวแปร SNOBOL อาจผันแปรได้ ขณะกระทำการ

ในทางตรงกันข้าม การผูกโยง ของ ตัวแปร I ชนิด integer กับ ตำแหน่งภายในโปรแกรมภาษาเครื่อง อาจเกิดขึ้น ณ เวลาแปล (เช่น หน่วยเก็บชนิด STATIC ของ PL/1, FORTRAN, และ COBOL) หรือระหว่างกระทำการ (เช่น หน่วยเก็บอื่น ๆ ใน PL/1), ตัวแปรเฉพาะที่ของ Pascal, และตัวแปรทั้งหมด ใน เวอร์ชัน การตีความของ SNOBOL, APL, และ LISP) การผูกโยง อันก่อนหน้า เรียกว่า การจัดสรรหน่วยเก็บแบบคงที่ (static storage allocation) และภาษาซึ่งผูกโยงตัวแปรทั้งหมด ในวิธีนี้ เรียกว่า ภาษาแบบคงที่ (static languages) ส่วนการจัดสรรหน่วยเก็บแบบพลวัต (dynamic storage allocation) และ ภาษาแบบพลวัต (dynamic languages) ซึ่งเป็นรูปแบบ สำหรับวิธีการผูกโยงอันหลัง

การผูกโยงของตัวแปรกับค่า ปกติเกิดขึ้น ณ เวลากระทำการและเกิดขึ้น ครั้งแล้วครั้งเล่า อย่างไรก็ตาม ค่าแรก (เช่นใน PL/1's INIT attribute, FORTRAN's DATA specification และ COBOL's VALUE clause) เป็นการกำหนดค่า ณ เวลาแปลโปรแกรม

ผล (outcome) ที่สำคัญอย่างหนึ่ง ของเวลาการผูกโยงที่แตกต่างกัน ปรากฏในการจัดการแฟ้มอินพุทมาตรฐานของภาษา Pascal ในสิ่งแวดล้อมแบบกลุ่มและแบบทันที (Pascal's handling of the standard "input" file in batch and interactive environments) สำหรับอินพุทกลุ่ม (batch input) แฟ้มถูกผูกโยงกับโปรแกรม ณ การเริ่มต้น ของการปฏิบัติการ เพื่อที่ว่าการปรากฏของแฟ้มว่าง (ทำเครื่องหมายโดยฟังก์ชัน "eof") ถูกทราบ ณ การเริ่มต้นของ การปฏิบัติการ สำหรับอินพุทแบบทันที (interactive input) แฟ้มจะไม่ถูกผูกโยงจนกระทั่งคำสั่ง read คำสั่งแรก ถูก กระทำการ ดังนั้น แฟ้มว่าง จึงไม่สามารถ

ตรวจพบ (detected) จนกระทั่งเวลานั้น ดังนั้น ตรรกะการเขียนโปรแกรม Pascal แตกต่างอย่างมีนัยสำคัญเมื่อมีการสวิตช์กลุ่มข้อมูล (awitching) ระหว่าง อินพุทแบบกลุ่มและอินพุทแบบทันที และเหตุผล ได้อธิบายแล้วโดย เวลาการผูกโยง ที่แตกต่างกัน

ตัวปฏิบัติการและการบังคับ (Operators and Coercion)

ความเบี่ยงเบนสำคัญสิ่งแรก ซึ่งเกิดขึ้นระหว่าง โปรแกรมที่เขียนด้วยภาษาระดับสูง และ การทำภาษาเครื่องของมัน ให้เกิดผล เกิดขึ้นใน อรรถศาสตร์ (semantics) ของการคำนวณ และการกำหนดค่า

ตัวอย่าง คำสั่งกำหนดค่าของ Pascal

```
A := B + 2 * C
```

จะมี การปฏิบัติการ "ซ่อน" เพิ่มเติมหลายอย่าง ภายใต้การบวก, การคูณ และการกำหนดค่า ซึ่งปรากฏชัดแจ้งในการตีความคำสั่งนี้ อย่างตรงไปตรงมามากที่สุด คือ ให้ A, B และ C ทั้งหมดนี้ เป็น ตัวแปรชนิด **integer** และการกระจายของมัน คือ ลำดับของคำสั่งเครื่องข้างล่างนี้

```
MOV  #2, R1          ; R1 := 2
MUL  C, R1           ; R1 := 2 * C
ADD  B, R1           ; R1 := B + 2 * C
MOV  R1, A           ; A := B + 2 * C
```

ในที่นี้ ลำดับของคำสั่งเครื่อง น้อยที่สุด เพราะว่าตัวถูกดำเนินการทั้งหมด เป็นชนิดเดียวกัน

อย่างไรก็ตาม ถ้าเราสมมติว่า มีความกังวลใจน้อยที่สุด ระหว่างชนิดของ A, B และ C รหัสเครื่องซึ่งเป็นผลลัพธ์ จะเป็นไปในทางที่เข้าไปสู่การแปลงผัน ชนิดที่จำเป็น นั่นคือ ตัวแปลชุดคำสั่ง ต้องใส่คำสั่ง CTVxy ที่เหมาะสม และจัดตำแหน่งหน่วยเก็บชั่วคราว ให้พบกับความต้องการที่เข้มงวดของคำสั่งคำนวณเครื่อง (machine arithmetic instructions) นอกจากนี้แล้ว มันใส่การทดสอบเวลาดำเนินงานที่เกี่ยวข้อง เพื่อให้เชื่อมั่นว่า การแปลงผันการใส่ เกิดขึ้นโดยปราศจาก ความผิดพลาดทันที กิจกรรมทั้งหมดนี้ รวมกันเรียกว่า **การบังคับ**

(coercion)

นั่นคือ **การบังคับ** หมายถึงการแปลงผัน อย่างถูกต้อง ของค่าข้อมูลชนิดหนึ่ง ให้เป็นค่าข้อมูลอีกชนิดหนึ่งที่มีความหมายเหมือนกัน เพื่อให้เป็นไปตาม ขีดจำกัดอรรถศาสตร์ ของ เซต คำสั่งเครื่อง หรือเนื้อความรวม ที่ซึ่ง ค่าเริ่มต้นเกิดขึ้น (That is, "coercion" is the proper conversion of a data value of a particular type to an "equivalent" data value of another type, in order to satisfy the semantic constraints of the machine instruction set or the global context where that original value occurs.) ในคำสั่งกำหนดค่าข้างต้นนั้น ชนิดต่างๆ ของการบังคับที่เป็นไปได้มีดังนี้

Coercion	cause
of B to reel	B is integer and the value of 2*C is real
of B+2*C to real	B+2*C is integer and A is real

เหล่านี้เป็นการบังคับเชิงตัวเลขสองชนิดเท่านั้น ซึ่งเกิดขึ้นในภาษา Pascal เนื่องจาก stringent constraints บนชนิด ซึ่ง นิพจน์ และตัวแปรทางด้านซ้ายมือมี ภาษา ซึ่งวาง constraints เช่นนี้ และการบังคับ เกิด ณ เวลาแปลชุดคำสั่ง เรียกว่า <u>strongly typed language</u>

เพื่อแสดงให้เห็น การบังคับ สำหรับตัวอย่าง Pascal นี้ สมมติว่า A และ B เป็นตัวแปรชนิด **real** ส่วน C เป็นตัวแปรชนิด **integer** รหัสเครื่องที่มีความหมายเหมือนกันจะปรากฏดังนี้

```
MOV   #2, R1          ; R1 := 2
MUL   C, R1           ; R1 := 2 * C
CVTLE R1, R1          ; Coercion 2 * C to real
ADDE  B, R1           ; R1 := B + 2 * C
```

MOVE R1, A ; A := B + 2 * C

การเปลี่ยนแปลงจาก เวอร์ชันแรก ได้ขีดเส้นใต้ไว้ ในเวอร์ชันนี้ จะเห็นว่า ผลคูณ integer 2*C ต้องถูกแปลงผัน ไปเป็นชนิดของ B เพื่อให้เป็นการบวกชนิด real (ADDE) และคำสั่งกำหนดค่า (MOVE) ซึ่งจะใช้ในสองขั้นตอนถัดไป ความจริงที่ว่า การปฏิบัติการคำนวณเครื่องต้องการตัวถูกดำเนินการทั้งคู่ เป็นชนิดเดียวกัน โดยทั่วไป ชนิดของการบังคับ ซึ่งจำเป็นจะต้องทำตาม

การบังคับอัตโนมัติ ไม่ได้เกิดขึ้นเฉพาะ ภายในคำสั่งคำนวณและคำสั่งกำหนดค่าเท่านั้น ภาษาส่วนใหญ่ กระทำเช่นเดียวกัน ตัวอย่างเช่น เมื่อใดก็ตาม ค่าอินพุท มีการค้นคืน และชนิดภายนอกของมัน ไม่ตรงกัน (disagree) กับชนิดตัวแปรซึ่งถูกเก็บไว้ ในทางปฏิบัติ สิ่งนี้เป็นจริงของตัวเลข ซึ่งเข้าไป ในการตอบรับ กับคำสั่ง read ในภาษา Pascal ดังนั้น

read(X)

เมื่อได้รับการ execute จะทำให้เกิดการบังคับ ของ สายอักขระ ของเลขฐานสิบ, เครื่องหมาย และจุดทศนิยม ให้มีความหมายเหมือนกับ การแทนที่ภายใน ขึ้นอยู่กับชนิด X ถ้า X เป็น real ตัวอย่างเช่น รหัสข้างล่างนี้ จะปรากฏหลังจาก การเคลื่อนย้าย สายอักขระนั้นจากแป้นพิมพ์ ไปยังบัฟเฟอร์ (BUFFER) ในหน่วยความจำ

CVTCE BUFFER, X

BTY ERROR

คำสั่งแรก แปลงผันค่า อินพุท BUFFER ให้เป็น real และคำสั่งที่สองทดสอบให้เห็นว่า ไม่มีข้อผิดพลาดเกิดขึ้นในกระบวนการ ข้อผิดพลาดเช่นนี้ จะเกิดขึ้น ตัวอย่างเช่น เมื่อสายอักขระซึ่งพิมพ์นั้น ไม่ใช่ตัวแทนเลขที่ถูกต้อง เช่น 22A

บางภาษา เช่น PL/1 กับตรงกันข้ามจาก Pascal และยอมให้ ชนิดที่ไม่จับคู่กัน ของทุก ๆ ชนิดภายในโปรแกรม เหล่านี้ เรียกว่า "weakly typed" languages และจากประสบการณ์ แสดงให้เห็นว่า ปรัชญานี้ มีแนวโน้ม ที่จะกระตุ้น ปัญหา และศักยภาพ สไตล์การเขียนโปรแกรมที่ไม่น่าเชื่อถือ (experience shows that this philosophy tends to encourage questionable and potentially unreliable programming styles.) ตัวอย่างคำสั่งกำหนดค่าข้างต้น ในภาษา PL/1

```
A = B + 2 * C
```

เราอาจประกาศ ให้ A เป็นสายอักขระ, B เป็นสายบิต และ C เป็นตัวแปร ชนิด **real** ในกรณีนี้ ลำดับของคำสั่งเครื่องที่จำเป็นจะเป็นดังนี้

```
MOVE      #2, R1          ; R1 := 2
MULE      C, R1           ; R1 := 2 * C
CVTEβ     R1, R1          ; convert 2 * C to Boolean
BYTE      ERROR           ; cheack for success
OR        B, R1           ; logical "or" of B with 2*C
CVTBC     R1, 4           ; convert B+2*C to character
BYTE      ERROR           ; check for success
```

(ในที่นี้ คำสั่งซึ่งขีดเส้นใต้ ไม่ได้ถูกนิยาม สำหรับคอมพิวเตอร์ของเรา แต่เป็นคำสั่งเครื่องเพิ่มเติม ซึ่งยอมรับได้) ปัญหารวม ในที่นี้ คือ อันตราย ซึ่ง weakly typed languages อยู่บนความไม่สงสัยของ นักเขียนโปรแกรม ข้อผิดพลาด ซึ่งไม่ได้สังเกต และไม่ได้คาดการณ์ไว้ล่วงหน้า จนกระทั่งเวลาดำเนินการ ทำให้ยากมากขึ้นที่ละตรวจพบ การใช้โดย ไม่มีขีดจำกัดของการบังคับอัตโนมัติ เช่นนี้ เพิ่ม "power" การเขียนโปรแกรมที่รายจ่ายของ ความชัดแจ้ง และความน่าเชื่อถือ ในภาษา

ในเนื้อหา ของ ระเบียน อินพุท-เอ้าพุท ฟังก์ชัน ไม่มีการบังคับเกิดขึ้น โดยอัตโนมัติ นั่นคือ เมื่อหนึ่งระเบียน ถูกย้ายไปหรือมาจากแฟ้มภายนอก เขตข้อมูล ซึ่งประกอบเป็นระเบียน จะยังคงอยู่ในรูปแบบเข้มงวด การบังคับที่จำเป็นใดๆ สำหรับการประมวลผลภายหลัง ต้องกำหนดในอีกคำสั่งหนึ่งแยกต่างหาก ตัวอย่างนี้เกิดขึ้นในภาษา COBOL การแทนที่คำสั่ง DISPLAY ของเขตข้อมูล เลขฐานสิบในแฟ้ม ถูกบังคับชัดแจ้ง ให้เป็นการแทนที่ภายในที่มีความหมายเหมือนกัน ก่อนนำไปใช้ในการคำนวณใดๆ การบังคับนั้นปกติ กระทำด้วย คำสั่ง MOVE

3.3 การจัดสรรหน่วยเก็บ (Storage Allocation)

การจัดสรร หน่วยเก็บ สำหรับแถวลำดับ ระเบียน และรายการโยง ค่อนข้างซับซ้อนมากกว่า ค่าหนึ่งค่า (scalar value) โดยเฉพาะเมื่อเกิดขึ้น ณ เวลาดำเนินงาน (at run

time) ในหัวข้อนี้ เราแนะนำเทคนิคพื้นฐานและอภิปรายหัวข้อต่างๆ ซึ่งเกี่ยวข้องกับ การจัดสรร และการกำหนดเลขที่อยู่ ของค่าต่างๆ ใน โครงสร้างข้อมูลเช่นนั้น

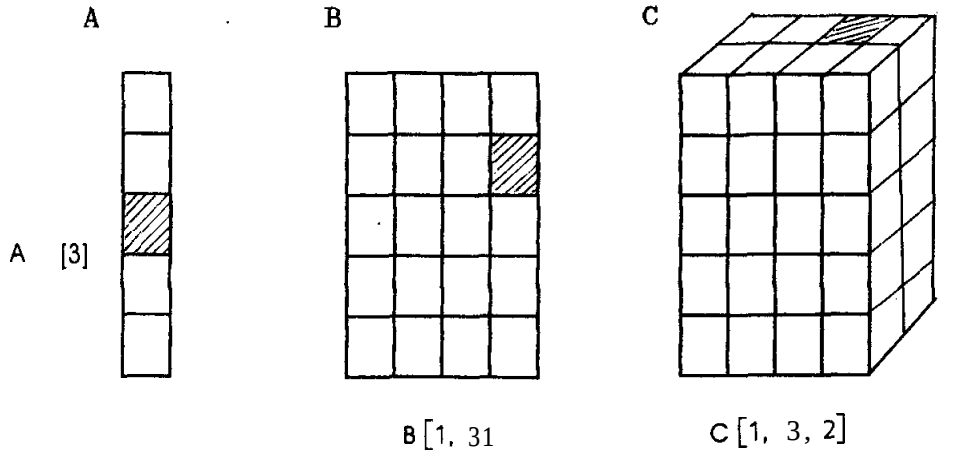
การจัดสรรหน่วยเก็บแถวลำดับ (Array Storage Allocation)

ปัญหาของการจัดสรรหน่วยเก็บ สำหรับแถวลำดับ หมายถึง สิ่งสำคัญอย่างหนึ่ง ของการแปลงส่งโครงสร้าง n มิติ ให้เป็น ลำดับ ของตำแหน่งหน่วยความจำติดกัน 1 มิติ และ การนิยามความสมนัย ระหว่างเซตของค่าดรรชนีล่าง (subscript in values) ในภาษาต้นฉบับ กับ เลขที่อยู่เกี่ยวข้องของมัน ในเวอร์ชันภาษาเครื่อง

ในภาษาส่วนใหญ่ แถวลำดับหมายถึง กลุ่มของค่าต่างๆ n มิติ ค่าทุกตัวเป็นชนิดเดียวกัน และสมาชิกหนึ่งตัวในแต่ละมิติ ถูกอ้างถึงโดย ดรรชนีล่าง จำนวนเต็ม ภายในช่วงพิสัย ซึ่งนิยามมาแล้ว (an array is an n-dimensional collection of values, all having the same type, and in each dimension an entry is referenced by an integer subscript within a predefined interval range.)

ตัวอย่าง แถวลำดับ 1 มิติ, 2 มิติ และ 3 มิติ A, B และ C แสดงให้เห็นในรูป 3-1 ประกาศในภาษา Pascal ดังนี้

```
var  A : array [1..5] of integer;
     B : array [0..4, 0..3] of string [5];
     C : array [1..5, 1..4, 1..2] of real;
```



รูป 3 - 1

อะไรที่เปลี่ยนจากแถวลำดับ หนึ่งใน 3 รูปนี้ เป็นสิ่งถัดไป

Description of array	A	B	C
จำนวนมิติ (n)	1	2	3
จำนวนสมาชิกในแต่ละมิติ ($k_1,k_2,\ldots,k_n$)	5	5,4	5,4,2
พิสัยดรรชนีล่าง ในแต่ละมิติ ($l_1..u_1,\ldots,$ $l_n..u_n$)	1..5	0..4, 0..3	1..5,1..4, 1..2
ชนิด (สมาชิกแต่ละตัวใช้เนื้อที่ c bytes)	integer (C=2)	string [5] (C = 7)	real (C = 4)

ลักษณะเหล่านี้ รวมเข้าด้วยกัน เพื่ออธิบาย (describe) แถวลำดับ ในรูปแบบของมันเรียกว่า โดป เวกเตอร์ (dope vector) หรือ ระเบียนอธิบายแถวลำดับ (array description record) ซึ่งรูปแบบทั่วไป เป็นดังนี้

address of 1st element	type	c	n	k_1	.	.	.	k_n	l_1 u_1	...	l_n u_n

สิ่งนี้ให้สาระสนเทศ (information) ทั้งหมดที่จำเป็นต้องใช้เพื่อ คำนวณเลขที่อยู่ ของสมาชิกแต่ละตัวในแถวลำดับ ตัวอย่างเช่น โดป เวกเตอร์ สำหรับแถวลำดับ A, B และ C ซึ่งประกาศไว้ข้างต้น จะเป็นดังนี้

address of A[1]	integer	2	1	5	1 5

address of B[0,0]	string[5]	7	2	5	4	0 / 4	0 / 3

address of C[1,1,1]	real	4	3	5	4	2	1 / 5	1 / 4	1 / 2

โดป เวกเตอร์ ประกอบด้วย สาระสนเทศ พอเพียงที่ใช้ในการคำนวณ สำหรับเซตของค่าดรรชนีล่าง ที่กำหนดให้นั้น ไม่ว่ามันจะถูกต้อง หรือไม่ ภายในพิสัยของแถวลำดับ เช่นที่ประกาศ สาระสนเทศนี้ กำหนดโดย $l_i..u_i$ สำหรับแต่ละมิติ i

มีสองหัวข้อที่คลุม (govern) การผูกโยง โดป เวกเตอร์ ของแถวลำดับ กับสูตรซึ่งใช้จัดสรรเลขที่อยู่ ให้กับสมาชิกแต่ละตัว

สิ่งแรก การผูกโยงของ โดป เวกเตอร์ ของแถวลำดับ จะไม่สามารถกระทำได้จนกว่าจะทราบมิติของมัน ในบางภาษา เช่น FORTRAN, Pascal และ COBOL สิ่งนี้ ทราบ ณ เวลาแปลโปรแกรม (at compile time) การประกาศแถวลำดับ ต้องเป็นค่าขอบเขตคงที่ (constant bounds) ส่วนภาษาอื่น ๆ เช่น Pl/1, SNOBOL และ APL ขอบเขตของแถวลำดับจะไม่ทราบ จนกระทั่งถึงเวลาดำเนินงาน (until run time)

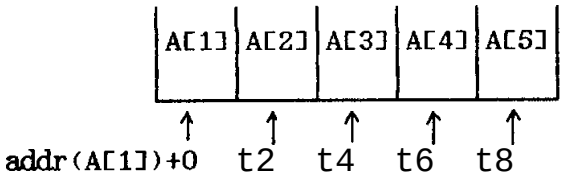
ตัวอย่าง การประกาศของภาษา PL/1

```
DCL A(N);
```

ซึ่ง ณ เวลาดำเนินงาน กำหนดว่า แถวลำดับ A มีสมาชิก N ตัว และ N เป็นตัวแปร ซึ่งจะไม่ทราบค่า จนกระทั่งพบ การประกาศนี้ ดังนั้น ในกรณีนี้ โดป เวกเตอร์ สำหรับ A ต้องกำหนดค่าของมันอย่างพลวัต และหน่วยความจำของแถวลำดับ ต้องเป็นการจัดสรรแบบพลวัต

หัวข้อที่สอง ต้องกระทำก่อน การคำนวณเลขที่อยู่ ของ สมาชิก แถวลำดับ คือ การเลือกของ

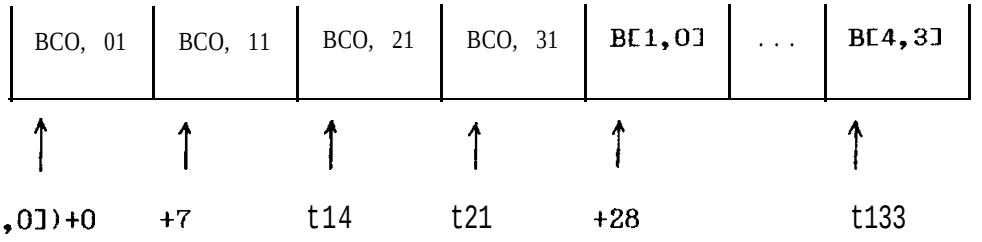
การเรียงอันดับ (ordering) สำหรับสมาชิกของแถวลำดับ ในหน่วยความจำ สำหรับแถวลำดับหนึ่งมิติ การนำเสนอนี้ ไม่มีปัญหา สมาชิกได้จัดเรียงอันดับค่าดรรชนีล่าง จากน้อยไปหามาก ในไบต์ต่อเนื่องกันของหน่วยเก็บ ตัวอย่างเช่น แถวลำดับ A ซึ่งประกาศข้างต้น ได้รับการจัดสรร โดยสมาชิกหนึ่งตัวใช้เนื้อที่ 2 ไบต์ ดังนี้



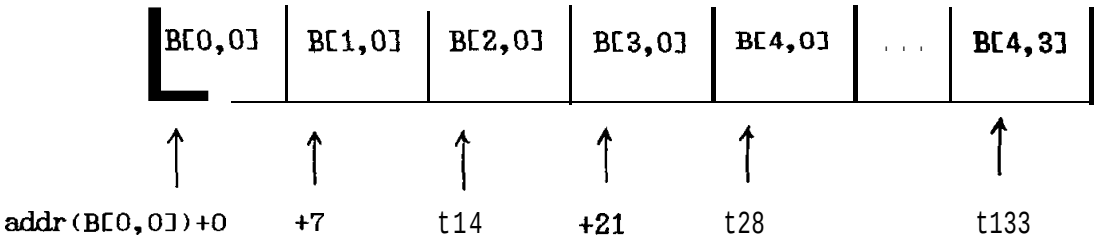
โปรดสังเกตว่า การแทนที่ (displacements) ของสมาชิกต่อเนื่องกัน ใน A เกิดขึ้น โดยเพิ่มครั้งละสองไบต์ เพราะว่าสมาชิกแต่ละตัว ต้องจัดที่อยู่ให้ค่าจำนวนเต็ม

สำหรับแถวลำดับที่มีมิติสูงขึ้น มียุทธวิธีให้เลือกสองแบบ สำหรับการจัดสรรหน่วยเก็บคือ เรียงอันดับตามแถว (row-major order) หรือ เรียงอันดับตามสดมภ์ (column-major order) ภาษาส่วนใหญ่ใช้รูปแบบแรก แต่บางภาษา (รวมทั้ง FORTRAN) ใช้รูปแบบที่สอง การเรียงอันดับตามแถว หมายถึง แถวลำดับถูกเก็บ โดยที่ แต่ละแถว ใช้ บล็อกของหน่วยเก็บติดกัน ส่วนการเรียงอันดับตามสดมภ์ หมายถึงแต่ละสดมภ์ เก็บในหน่วยเก็บติดกัน

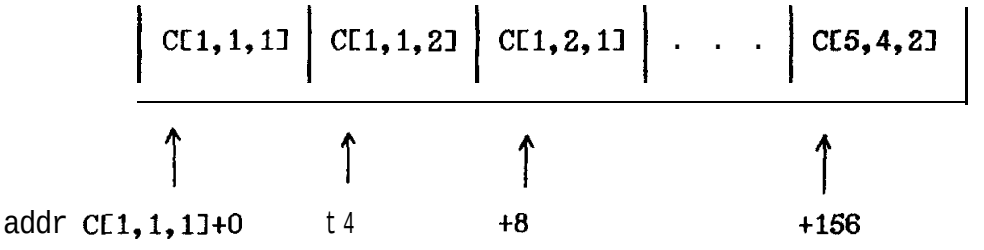
<u>ตัวอย่าง</u> แผนภาพข้างล่างนี้ แสดงให้เห็นว่า การประกาศ B ข้างต้น เก็บโดยเรียงอันดับตามแถว เป็นดังนี้



ในทางตรงกันข้าม การจัดสรร B เรียงอันดับตามสดมภ์ จะให้ผลลัพธ์ดังนี้



แถวลำดับ C สามมิติ ซึ่งประกาศข้างต้น ใช้ยุทธวิธีเหมือนกัน ภาพข้างล่างนี้เป็นการแทนที่หน่วยเก็บแบบแถว



ขณะนี้รูปแบบชัดเจน สำหรับแถวลำดับหนึ่งมิติ ให้ a_1 เป็นเลขที่อยู่ของสมาชิกตัวที่ i คำนวณดังนี้

$$addr(a_i) = addr(a_{l_1}) + c(i-l_1)$$

ในที่นี้ ความหมายของ l_1 และ c เหมือนกับในโดป เวกเตอร์ คือ เป็นขอบเขตล่างของพิสัยดรรชนีล่าง และจำนวนไบต์ สำหรับสมาชิกแต่ละตัว ตามลำดับ

สำหรับแถวลำดับสองมิติและสามมิติ เก็บเรียงอันดับแบบแถว การคำนวณจะเป็นดังนี้

$$addr(a_{ij}) = addr(a_{l_1, l_2}) + c[(i-l_1)k_2 + (j-l_2)]$$

$$addr(a_{ijk}) = addr(a_{l_1, l_2, l_3}) + c[((i-l_1)k_2+(j-l_2))k_3+(k-l_3)]$$

สำหรับแถวลำดับ n มิติ เรียงอันดับแบบแถว เลขที่อยู่ของ $a_{i_1 i_2 \ldots i_n}$ คำนวณโดยวิธีทั่วไปดังนี้

$$(a_{i_1 i_2 \ldots i_n}) = \text{addr}(a_{l_1 l_2 \ldots l_n}) + c[(\ldots((i_1-l_1)k_2+(i_2-l_2)k_3+\ldots+(i_n-l_n))]$$

<u>ตัวอย่าง</u> จงคำนวณเลขที่อยู่ของ ข้อมูล A[4], B[4,2] และ C[2,1,1] สำหรับแถวลำดับซึ่งประกาศข้างต้น โดยใช้สูตร

addr(A[4]) = addr(A[1]) t 2(4-1) = addr(A[1]) t 6

addr(B[4,2]) = addr(B[0,0]+7[(4-0)4+(2-0)] = addr(B[0,0]) t 126

addr(C[2,1,1]) = addr(C[1,1,1])+4[((2-1)4+(1-1))2+(1-1)]

= addr(C[1,1,1]) t 32

ผู้อ่านควรจะตรวจสอบว่า การคำนวณเหล่านี้ ให้ การแทนที่เหมือนกับที่แสดงให้เห็นข้อมูลเหล่านี้ในแผนภาพ สำหรับ A, B และ C

การทำให้เกิดผลของแถวลำดับ เกี่ยวข้องกับ สามขั้นตอนดังนี้

(1) การจัดสรร และกำหนดค่าให้กับ โดป เวกเตอร์ (allocation and assignment of values to the dope vector)

(2) การจัดสรรหน่วยเก็บ สำหรับแถวลำดับเอง (allocation of storage for the *array* itself)

(3) คำนวณการแทนที่ หรือค่าดรรชนี สำหรับสมาชิกของแถวลำดับ เมื่อกำหนดเซตของดรรชนีล่างให้ (calculation of a displacement, or index value, for an array element when given a set of subscripts.)

อาจจะมี ขั้นตอนที่สี่ การก่อกำเนิดของการตรวจสอบเวลาดำเนินงาน สำหรับ ดรรชนีล่างออกนอกพิสัย (Optionally, a fourth provision is advised; generation of a run-time check for subscripts out of range.)

กรณีแย่ที่สุด คือ ทั้งสี่ขั้นตอน ต้องรอ จนกระทั่งถึงเวลาดำเนินงาน ต่อไปจะแสดงให้

เห็น รหัสเครื่องที่จำเป็น สำหรับสี่ขั้นตอนนี้ จงพิจารณาแถวลำดับ B ซึ่งประกาศไว้ตอนต้น

1. การจัดสรร และการเริ่มต้นของโดป เวกเตอร์ สำหรับ B

```
GETMAIN   #40,BDOPE          ; obtain a block of storage
MOV #7,@BDOPE t 8            ; length of an entry (c)
MOV #2,@BDOPE t 12           ; no of dimensions (n)
MOV #5,@BDOPE t 16           ; no of rows (k1)
MOV #4,@BDOPE t 20           ; no of columns (k2)
MOV #0,@BDOPE t 24           ; lower bound (l1)
MOV #4,@BDOPE t 28           ; upper bound (u1)
MOV #0,@BDOPE t 32           ; lower bound (l2)
MOV #3,@BDOPE t 36           ; upper bound (u2)
```

2. จัดสรรหน่วยเก็บสำหรับ B เอง หลังจากการคำนวณ จำนวนหน่วยเก็บที่ต้องการ $(c * k_1 * k_2)$

```
MOV  BBDOPE  t  16,  R1      ; จำนวนแถว (k1) อยู่ที่ @BDOPE+16
MOV  @BDOPE  t  20,  R1      ; จำนวนสดมภ์ (k2) อยู่ที่ @BDOPE+20
MOV  @BDOPE  t  8,  R1       ; ความยาวของสมาชิก (c) อยู่ที่
                             ; @BDOPE+8
GETMAIN RI,  B               ; addr(B[0,0]) ---> B
```

3. คำนวณ เลขที่อยู่ของ B(I,J) กำหนดตัวแปร I และ J ซึ่งค่าของมันสมมติว่ามีอยู่แล้ว

```
MOV I,  R1
```

```
SUB @BDOPE t 24, R1        ; I-l1
MUL @BDOPE t 20, R1        ; (I-l1) * k2
MOVI J, R2
SUB @BDOPE t 32, R2        ; J-l2
ADD R1, R2                 ; (I-l1) * k2 t (J-l2)
MUL @BDOPE t 8, R2         ; C * [(I-l1) * k2 +(J-l2)]
ADD @BDOPE, R2             ; C * [(I-l1) * k2 +(J-l2)]
                           ; t addr(B[0,0])
```

ขณะนี้ เลขที่อยู่ของ B[I,J] อยู่ในเรจิสเตอร์ R2 พร้อมสำหรับใช้ โดยการอ้างถึงของคำสั่งใด ๆ

4. ตรวจสอบว่า ดรรชนีล่าง I และ J อยู่ในขอบเขต $l_1..u_1$ และ $l_2..u_2$ ตามลำดับ และย้ายไป ERROR ถ้าไม่อยู่ในขอบเขต

```
CMP  I,@BDOPE t 24
BL   ERROR                 ; I < l1
CMP  I,@BDOPE t 28
BG   ERROR                 ; I > u1
CMP  J,@BDOPE t 32
BL   ERROR                 ; J < l2
CMP  J,@BDOPE t 36
BG   ERROR                 ; J > u2
```

ขั้นที่ 3 ปกติ การทำให้เกิดผล มีรูปแบบที่ดีกว่า กล่าวคือ สูตรการคำนวณเลขที่อยู่สามารถเขียนใหม่โดยใช้ จำนวนของการปฏิบัติการคำนวณ น้อยกว่า ณ เวลาดำเนินงาน สิ่งนี้

สำคัญ สำหรับประสิทธิภาพของ โปรแกรมคณิตศาสตร์ ขนาดใหญ่ ซึ่ง ปกติ ปฏิบัติการอ้างถึงแถวลำดับ มีจำนวนเป็นหลายพัน ในการดำเนินงานหนึ่งครั้ง (in a single run) สำหรับแถวลำดับ 2 มิติ เทอมคงที่ เอารวมเข้าด้วยกัน และเขียนการคำนวณเลขที่อยู่ใหม่ ดังนี้

$$addr(a_{ij}) = addr\ (a_{l_1 l_2})\quad t\ c * (k_2 * i\ t\ j) - c * (k_2 * l_1\ t\ l_2)$$

ขณะนี้เทอมที่ 1 และเทอมที่ 3 เป็นค่าคงที่ (constants) และผลบวกของมันคำนวณแล้ว และเก็บใน โดป เวกเตอร์ ณ เวลาที่มีการพัฒนาเวกเตอร์ การคำนวณ เวลาดำเนินงาน ส่วนที่เหลือ จะลดการคูณสองครั้งและการบวกสองครั้ง ซึ่งเมื่อเปรียบเทียบแล้ว ดีกว่า รหัสเวลาดำเนินงานข้างต้น การประหยัดที่มีนัยสำคัญ มากกว่า สามารถเห็นได้ชัดเจน สำหรับแถวลำดับมิติสูง ๆ

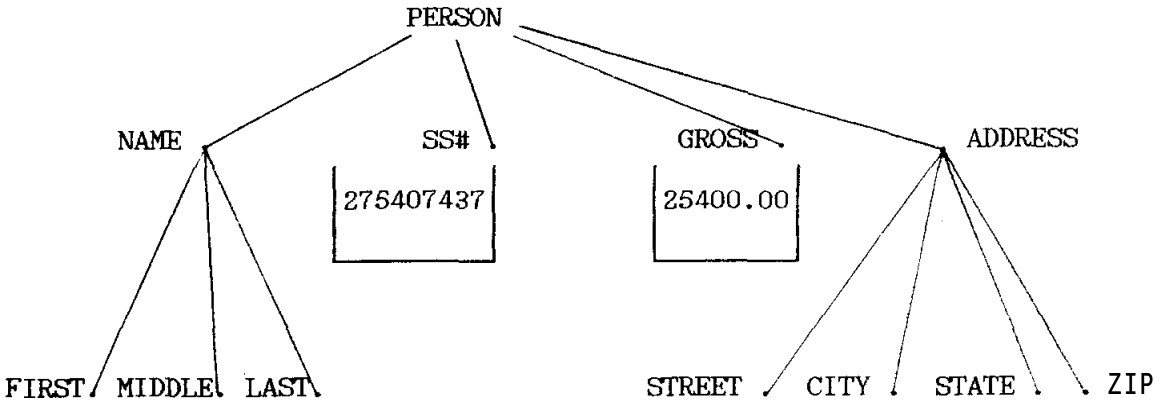
โครงสร้างระเบียน (Record Structures)

การจัดสรรหน่วยเก็บและกำหนดเลขที่อยู่ สำหรับโครงสร้าง ซึ่งแตกต่างจากแถวลำดับ จำนวนหน่วยเก็บทั้งหมด สำหรับ หนึ่งระเบียน คือ ผลรวมของความต้องการนั้นสำหรับข้อมูลแต่ละตัวซึ่งประกอบเข้าด้วยกัน ทุกตัว โดยปกติ มีชนิดแตกต่างกัน (the total amount of storage needed for a record is the sum of that required for each of its constituents, all of which typically have different types.) อันดับซึ่งหน่วยเก็บได้ถูกจัดสรรให้กับข้อมูลพื้นฐาน ใน โครงสร้าง นิยามโดย อันดับ ของการปรากฏในการประกาศระเบียน การแทนที่ของสมาชิกแต่ละตัวจากการเริ่มต้นของโครงสร้าง คำนวณโดยผลบวกของความยาวตามลำดับ มีหน่วยเป็นไบต์ ของสมาชิกทั้งหมด ซึ่งนำหน้ามันในการประกาศ

ตัวอย่าง จงพิจารณา โครงสร้าง ซึ่งแสดงให้เห็นในรูป 3-2 สิ่งนี้ประกาศใน PL/1 ดังนี้

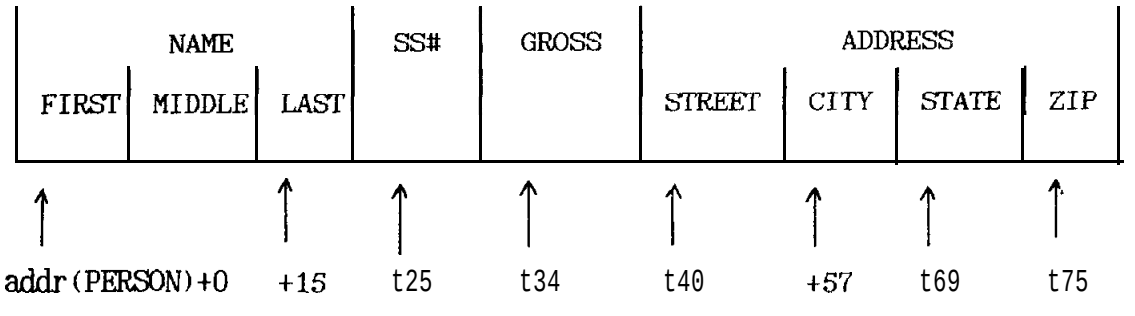
```
DCL    1  PERSON,
          2  NAME,
             3  FIRST     CHAR(10),
             3  MIDDLE    CHAR(5),
             3  LAST      CHAR(10),
```

```
2 SS#   PIC   '9(9)'
2 GROSS FIXED DEC(7,2),
2 ADDRESS,
   3 STREET    CHAR(17),
   3 CITY      CHAR(12),
   3 STATE     CHAR(6),
   3 ZIP       PIC '99999';
```



รูป 3-2

การจัดสรรหน่วยเก็บ สำหรับโครงสร้างนี้ จะเป็นดังนี้

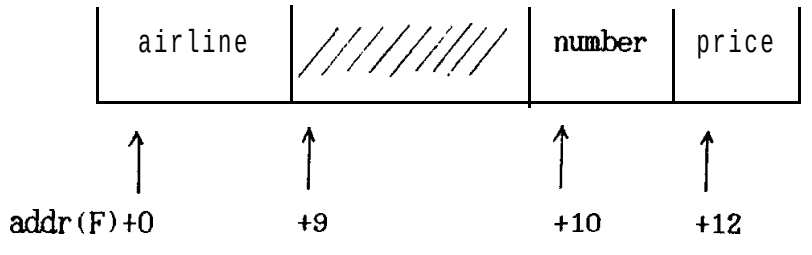


ดังนั้น ความยาวทั้งหมดของระเบียน เท่ากับ 80 ไบต์ เป็นตัวเลข ซึ่งนำไปใช้คำนวณ ตัวอย่าง เช่น ขนาดบัฟเฟอร์ สำหรับ การย้าย อินพุท-เอ้าพุท ของระเบียนเช่นนี้ ไปยัง แฟ้มภายนอก (external files)

ในคอมพิวเตอร์บางเครื่อง การจัดสรรหน่วยเก็บสำหรับระเบียนจะซับซ้อนโดยการวางขอบเขต (boundary alignment) ที่ต้องการสำหรับชนิดของเขตข้อมูลบางอย่าง ตัวอย่างเช่น จำนวนเต็ม และจำนวนจริง อาจต้องการแนวของเลขที่อยู่ ซึ่งเป็นผลคูณของ สองไบต์หรือสี่ไบต์ ตามลำดับ การทำสิ่งนี้ ให้เป็นผลสำเร็จ ไบต์เสริมเต็ม (padding bytes) ตั้งแต่หนึ่งไบต์ขึ้นไป บางครั้งต้องใส่ ก่อน เขตข้อมูลเช่นนี้ เพื่อที่จะ รักษาการวางแนวให้ถูกต้อง จงพิจารณาตัวอย่างการประกาศ Pascal ต่อไปนี้ สำหรับระเบียนสายการบิน และตัวแปร F

```
type   flight  =  record
                  airline  : string [9];
                  number   : integer:
                  price    : real
                  end;
var    F: flight;
```

การจัดสรร หน่วยเก็บสำหรับ F กับไบต์เสริมเต็ม ซึ่งแรเงาไว้ แสดงให้เห็นดังนี้



จำนวนไบต์ทั้งหมด สำหรับระเบียน คือ 15 ไม่ใช่ 14 เนื่องจากมีการใส่ ไบต์เสริมเต็ม เพิ่มก่อน เขตข้อมูล F.number

ข้อจำกัด ขึ้นอยู่กับตัวเครื่องเช่นนี้ เป็นภาระกับการเคลื่อนย้ายได้ (porttability) ของโปรแกรม เว้นเสียแต่ว่า นักเขียนโปรแกรม จะมีความรอบคอบพอเพียงที่จะ คาดการณ์ล่วงหน้า การทำให้เกิดผลในอนาคตทั้งหมด (ซึ่งค่อนข้างเป็นไปไม่ได้) เมื่อโปรแกรมจะดำเนินงานในที่สุด จะเห็นชัดเจนว่า กรณีนี้ และกรณีอื่น ๆ ที่เหมือนกับกรณีนี้ ซึ่งการใส่ไบต์เสริมเต็มอัตโนมัติ ควรจะหลีกเลี่ยงอย่างเต็มที่ โดยการจัดการเขตข้อมูลของระเบียนอย่างเป็นระบบ หรือกรณีอื่น โดยการใส่อย่างชัดแจ้ง ของ "dummy fields" ภายใน การประกาศชนิดระเบียนของมันเอง กรณีหลัง ปกติเป็นผลเฉลยที่ทำมากกว่า เพราะว่ากรณีแรกผลลัพธ์ในการจัดการใหม่ จะถูกทำลาย (destroys) บนระดับหนึ่ง การเรียงอันดับดีที่สุดระหว่างเขตข้อมูล

โครงสร้างลิสป์แบบพลวัต (Dynamic List Structures)

โครงสร้างลิสป์แบบพลวัต เกิดขึ้นเสมอ ในภาษา Pascal, PL/1, SNOBOL, LISP และ Ada แต่ไม่มีในภาษา FORTRAN หรือ COBOL เพราะว่างานทางด้านวิทยาศาสตร์และงานทางด้านประมวลผลข้อมูล มีการใช้รายการแบบพลวัต อย่างจำกัด

การทำให้เกิดผลของรายการแบบพลวัต ขึ้นอยู่กับ

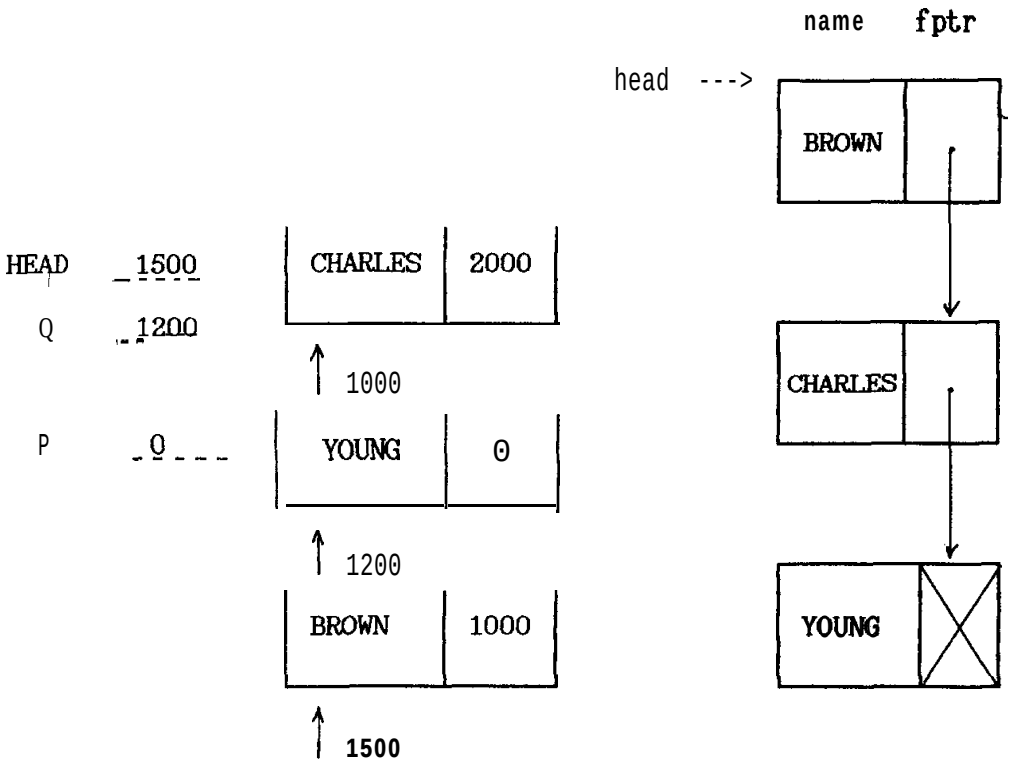
(1) รูปแบบการแทนที่สำหรับตัวชี้ (a representation scheme for pointers)

(2) วิธีจัดสรรแบบพลวัต และบล็อกอิสระของหน่วยเก็บ สำหรับโหนดแต่ละตัวในรายการ (a method for dynamically allocating and freeing blocks of storage for individual nodes in the list)

(3) ยุทธวิธี สำหรับ การเรียกกลับหน่วยเก็บอัตโนมัติ ซึ่งได้ปล่อยไปโดยโปรแกรม เมื่อหน่วยเก็บ ที่ว่างอื่น ๆ ทั้งหมดได้ถูกใช้ ยุทธวิธีนี้เรียกว่า <u>การรวบรวมขยะ</u> (garbage collection) และมันเป็นสมาชิกร่วม ของภาษาเช่น SNOBOL และ LISP ซึ่งไม่มีสิ่งอำนวยความสะดวกจัดการหน่วยเก็บ ไม่เห็นชัดแจ้ง กับ โปรแกรมตัวมันเอง

ในหัวข้อนี้ เราจะมุ่งสนใจเฉพาะการแทนที่เวลาดำเนินงาน ข้อ (1) และ (2) ข้างต้น และแสดงให้เห็น การแทนที่ภายในเครื่องของ คำสั่งประมวลผลรายการ ของภาษา Pascal และ PL/1 อย่างง่าย ณ เวลาดำเนินงาน

ตัวอย่าง การแทนที่เวลาดำเนินงาน สำหรับรายการ ในรูป 3-3 ภาษา Pascal อธิบายได้ดังนี้



รูป 3-3

ในที่นี้ ตัวชี้ HEAD, Q, P และ FPTR (ในโหนดสามตัว) เป็นการทำให้เกิดผลจริง หมายถึง **เลขที่อยู่** (address) และโหนดแต่ละตัวของรายการ ไม่ใช่เนื้อที่แผ่นเดียวกัน (non-contiguous) และบล็อกเลขที่อยู่ของหน่วยเก็บเป็นแบบสุ่ม โหนดแต่ละตัวคือ บล็อกหน่วยเก็บขนาด 21 ไบต์ ได้มาจากคำสั่งเครื่อง GETMAIN เลขที่อยู่ 1000,1,200 และ 1500 เป็นเลขที่อยู่ ซึ่งกำหนดให้ สมมติว่า ได้มาจาก การปฏิบัติการของ คำสั่ง GETMAIN สามครั้ง เพื่อความสะดวก ตัวชี้ **nil** ถูกแทนที่ ด้วยเลขที่อยู่ 0 ตามข้อตกลงเหล่านี้ เราสามารถยอมรับการทำให้เกิดผลของคำสั่งประมวลผลรายการแต่ละชุด คำสั่ง Pascal สองคำสั่ง

```
new(P);
head := P
```

7-2

ถูกทำให้เกิดผลในคำสั่งภาษาเครื่องดังนี้

```
GETMAIN #21, P
MOV  P, HEAD
```

การกำหนดค่าให้กับเขตข้อมูล ภายใน โหนด ซึ่งมีเลขที่อยู่เป็น P เช่น เขตข้อมูล NAME ด้วยค่า "BROWN" กระทำโดย การกำหนดเลขที่อยู่โดยอ้อม (indirect addressing) ดังนี้

```
MOVC  'BROWN', @P
```

ในที่นี้ การสมนัยระหว่าง การกำหนดเลขที่อยู่โดยอ้อม (@) กับความคิดซึ่ง ตัวชี้ Pascal ส่งไปโดยทันที สุดท้าย การเชื่อมจากโหนดหนึ่งตัว ไปยัง โหนดถัดไป ในรายการ ถูกกระทำด้วยลำดับต่อไปนี้ ซึ่งสมนัยกับ คำสั่ง Pascal ซึ่งแสดงไว้ขวามือ ในรูป คำอธิบาย

```
MOV  P, Q          ;  Q := P
GETMAIN #21, P     ;  new(P)
MOV P,  @Q+17      ;  Q↑. fptr  := P
```

การทำโหนดให้เป็นอิสระ ทำให้เกิดผลดังนี้

```
FREEMAIN #21, P  ;  dispose(P)
```

การแสดงให้เห็นโดยสรุปนี้ ปัญหาสองอย่างซึ่ง อาจเกิดขึ้นระหว่างการพัฒนา โปร-แกรมประมวลผลรายการ คือ

1. มันเป็นไปไม่ได้ที่จะตามรอยตัวแปรอย่างมีความหมาย (meaningfully trace variables) ระหว่างการปฏิบัติการ ของ โปรแกรมการประมวลผลรายการ
2. โหนด หรือ เซกเม้นต์ ของรายการโยง อาจจะเป็น "เนื้อที่สูญหาย" (lost in space) เช่น ไม่มีตัวชี้อ้างถึงมัน แม้ว่าจะยังไม่ได้ทำให้เป็นอิสระก็ตาม

ปัญหาแรกเกิดขึ้นเพราะว่า ตัวชี้ เป็นเพียงเลขที่อยู่ และเลขที่อยู่นั้น ไม่มีความหมายอะไร กับนักเขียนโปรแกรม เหมือนกับ ค่าของ ตัวแปร **integer** หรือ ตัวแปร **real**

ปัญหาที่สอง เกิดขึ้นเนื่องจาก ผลลัพธ์ของจำนวน ซึ่งไม่เป็นปกติ ภายในตรรกะของโปรแกรม เช่นลำดับของ Pascal ต่อไปนี้

```
new(P);
P := nil
```

คำสั่งที่สอง ไม่ต่อกับ ตัวชี้ P จากโหนดซึ่งเพิ่งจัดสรร ทำให้ไม่มีทางอ้างถึงต่อไป หรือ ปล่อยโหนดนี้ แต่ โหนดนั้นยังคงครอบครองบล็อกหน่วยเก็บ จนกระทั่ง จบ โปรแกรม หรือ เนื้อที่ว่างทั้งหมดถูกใช้ไป

ในกรณีเหตุการณ์หลัง การปฏิบัติการของคำสั่ง "new" ถัดไป จะขัดจังหวะการปฏิบัติการ การหยุดการทำงานระยะสั้นของโปรแกรม ระบบอาจจัดหาเครื่องมือ กระทำ การรวมขยะ ณ จุดนี้ นั่นคือ การกวาดอย่างเป็นระบบ บล็อกเนื้อที่จัดสรรทั้งหมด ถูกกระทำขึ้นและเนื้อที่เหล่านี้ ซึ่งอ้างโดย ไม่มีตัวชี้ สามารถเอากลับคืนได้ ดังนั้น โนหดต่าง ๆ ในรายการของเรา ซึ่งเป็น "เนื้อที่สูญหาย" ขณะนี้เป็นอิสระ และการปฏิบัติการโปรแกรม จะสามารถกลับคืนได้ปกติ

การรวมขยะ เป็น สิ่งสำคัญ โดยเฉพาะ ในภาษา ซึ่งนักเขียนโปรแกรมไม่สามารถควบคุม การจัดสรรหน่วยเก็บ และการให้อิสระหน่วยเก็บ โดยตรง เช่น ภาษา LISP และ SNOBOL ส่วนภาษาอื่น ๆ เช่น PL/1 และ Pascal ปล่อยความรับผิดชอบนี้ ให้กับนักเขียนโปรแกรม และปกติจะ ไม่มีสิ่งอำนวยความสะดวก การรวมขยะอัตโนมัติ (no automatic garbage collection facility) เพราะว่า สิ่งนี้ โดยทั่วไป เป็นหัวข้อซับซ้อน ผู้อ่าน ควรอ้างถึงตำราอื่นๆ สำหรับการอภิปราย ในรายละเอียดของยุทธวิธี การรวมขยะต่างๆ และอัลกอริทึม กรณีศึกษา 5 - Job Scheduler รวมรูปแบบการรวมขยะอย่างง่าย (ดูภาคผนวก E) ในเนื้อหาของปัญหาการเขียนโปรแกรมระบบปฏิบัติการ

3-4 โครงสร้างควบคุม (Control Structures)

เนื่องจากภาษาเครื่อง ไม่มีโครงสร้างควบคุมให้ใช้ เหมือนในภาษาระดับสูง เช่น **if ... then ... else, while** และ **for** โครงสร้างเหล่านี้ ต้องถูก จำลอง (simulated ขึ้น โดยใช้ การรวมกัน ของคำสั่ง เปรียบเทียบและคำสั่ง ย้าย ในระดับต่ำ โดยปกติ ตัวแปลชุดคำสั่ง จะก่อกำเนิด "template" หรือ รูปแบบของคำสั่งเครื่องสมนัยกับแต่ละชุดของโครงสร้างควบคุมเหล่านี้ (Compilers typically generate a "template" or pattern of machine instructions corresponding to each of these control structures.) สำหรับเครื่องคอมพิวเตอร์ของเรา templates เหล่านี้ ได้แก่

Control Structure	Template
if [b] then [s]	COMP [b], true BNE $L0001 [s] $L0001
if [b] then [s_1] else [s_2]	COMP [b], true BNE $L0001 [$s_1$] B $L0002 $L0001 [$s_2$] $L0002

for i:= [e] to [e]
do [s]
	MOV [e_1], i
	$L0001 CMP i, [$e_1$]
	BG $L0002
	[s]
	ADD #1, i
	B $L0001
	$L0002

Control Structure	Template
while [b] do [s]	$L0001 COMP [b], true BNE $L0002 [s] B $L0001 $L0002
case i of v_1: [s_1]; v_2: [ ]; ⋮ v_n: [s_n] end	CMP i , v_1 BNE $L0002 [$s_1$] B $L000$_{n+1}$ $L0002 CMP i, v_2 BNE $L0003 [$s_2$] B $L000$_{n+1}$ $L000$_n$ CMP i, v_n BNE $L000$_{n+1}$ [s_n] $L000$_{n+1}$

ใน templates เหล่านี้ สมาชิกในรูปสี่เหลี่ยม (boxed elements) หมายถึง นิพจน์ หรือ คำสั่ง ซึ่ง จำเป็นต้องขยายต่อไปให้เป็นรหัสเครื่องและเลเบล $L0001, $L0002, etc หมายถึง compiler generated symbols ซึ่งเป็นจุดแยก (branch points) สำหรับคำสั่งอื่นๆ ใน template ดังนั้น template เหล่านี้ จึงเหมือนกับ macros ของภาษาแอสเซมบลี

และแต่ละชุด ถูก "expanded" ในบรรทัด โดยตัวแปลชุดคำสั่ง ซึ่งสมนัยกับโครงสร้างควบคุมที่พบใน ภาษาต้นฉบับ (source language)

3-5 โปรซีเดอร์ และพารามิเตอร์ (Procedures and Parameters)

บางที โครงสร้างควบคุม เริ่มแรก ในการออกแบบภาษาชุดคำสั่ง คือ การเรียกโปรซีเดอร์ (procedure invocation) ซึ่งประกอบเป็นมูลฐานสำหรับการพัฒนาโดยลำดับ ของการเขียนโปรแกรมแบบมอดุลาร์ และแนวคิดออกแบบโครงสร้าง และสิ่งเหล่านี้ เป็นเครื่องมือพื้นฐานใน โครงการวิศวกรรม ซอฟต์แวร์ชั้นสูง ในงานประยุกต์ทุกด้าน มองจากจุดของการออกแบบภาษา มีสองหัวข้อสำคัญ ที่เกี่ยวข้อง ในการเรียกโปรซีเดอร์

1. การโยงการควบคุม (Linkage of control)
2. การโยงพารามิเตอร์ และอาร์กิวเม้นต์

(Linkage of parameters and arguments)

เราจะอภิปรายสองหัวข้อนี้ แยกต่างหากจากกันเป็น 2 ตอน ตามลำดับ ดังนี้

การโยงการควบคุมระหว่างโปรซีเดอร์ (Linkage of Control between Procedures)

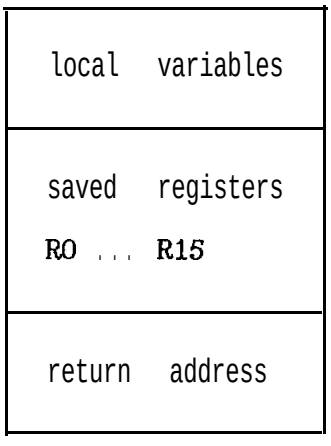
เมื่อ โปรซีเดอร์หนึ่ง ถูกเรียก มีการโยงกลับ ไปยังจุดของการเรียก หรือ **จุดกลับคืน** (return point) สิ่งนี้สมนัยกับ **เลขที่อยู่ส่งกลับ** (return address) ในภาษาเครื่อง นอกจากนี้แล้ว ตัวแปรเฉพาะที่ทั้งหมด ซึ่งประกาศภายในโปรซีเดอร์ ถ้าเป็นชนิดพลวัต จะถูกจัดสรรเนื้อที่หน่วยเก็บให้ และกำหนดค่าเริ่มต้นที่เหมาะสม

(when a procedure is invoked, a link back to the point of invocation, or "return point", must be establishd. This corresponds to the "return address" in machine language parlance. In addition, all local variables declared within the procedure, if they are dynamic, must be allocated memory space and initialized as appropriate.)

ในการทำขั้นตอนเหล่านี้ ให้บรรลุผล สมมติให้มี **แถวคอยเวลาดำเนินงาน** (a run-

time stack) ทุกครั้งที่ โปรซีเดอร์หนึ่งถูกเรียก หรือ ถูกใช้งาน (activated) จะมีระเบียนปฏิบัติการ (a activation record) ใส่บนกองซ้อน เช่นเดียวกัน เมื่อใดก็ตาม โปรซีเดอร์ส่งคืนการควบคุม (returns control) ไปยัง โปรซีเดอร์เรียก ระเบียนปฏิบัติการบนสุดจะถูกนำออกจากกองซ้อน ดังนั้น ณ จุดใดๆ ระหว่างการปฏิบัติการ ภาพ (image) เวลาดำเนินงานของลำดับชั้นของโปรซีเดอร์ปฏิบัติการทั้งหมด ได้มาโดยการตรวจสอบ ชุด (series) ของระเบียนปฏิบัติการ ซึ่งประกอบกัน ในกองซ้อนเวลาดำเนินการ ในภาษาเครื่องของเรา กองซ้อนเวลาดำเนินการ จัดไว้แล้ว ในภาษาอื่นๆ กองซ้อนเวลาดำเนินงาน อาจไม่ได้มีการจัดไว้จึงต้องมีการ ทำแบบจำลองขึ้น

มูลค่าของหนึ่งระเบียนการใช้งานนิยามดังนี้ (The contents of an activation record can be defined as follows:)



ในที่นี้ เรามีการจัดการเพิ่ม โดยเก็บมูลค่าของเรจิสเตอร์ R0 ... R1 ไว้ เพื่อว่า โปรซีเดอร์จะได้นำกลับมาใช้ได้อีกอย่างอิสระ เช่นเดียวกับตัวแปรเฉพาะที่ ค่าเหล่านี้ จะกลับมาเก็บ (restore) ในเรจิสเตอร์ หลังจากกลับคืนจากโปรซีเดอร์

ดังนั้น การเรียกโปรซีเดอร์ ทำให้เกิดผล ดังนี้

CALL name

เมื่อ name เป็นเลขที่อยู่สัญลักษณ์ (symbolic address) ของคำสั่งปฏิบัติการ คำสั่งแรกใน

โปรซีเดอร์ ณ เวลาที่คำสั่งนี้ ถูกกระทำการเลขที่อยู่กลับ (return address) จะถูกใส่อัตโนมัติบน กองซ้อนเวลาดำเนินงาน ดังนั้นจึงเป็นการเริ่มต้น ระเบียนการใช้งานใหม่ ส่วนที่เหลือของระเบียนการใช้งานจะถูกเก็บทันที โดยสองสามคำสั่งแรกของโปรซีเดอร์โดยตัวมันเอง นั่นคือ "entry sequence" ใน โปรซีเดอร์ ต้องเป็นดังนี้

1. ใส่เรจิสเตอร์ R0 ... R15 บนกองซ้อน (Push the registers R0 ... R15 onto the stack)
2. ใส่ และกำหนดค่าแรก ให้กับตัวแปรเฉพาะที่บนกองซ้อน (Push and initialize its local variables on the stack)

ถัดจาก ลำดับข้างต้นนี้จะเป็น ตัวโปรซีเดอร์ และสุดท้าย "return sequence" เอาระเบียนการใช้งานออกจากกองซ้อน อย่างเป็นระบบ และเก็บมูลค่าเรจิสเตอร์ สำหรับโปรแกรมเรียก นั่นคือ ขั้นตอนเหล่านี้

1. เอาตัวแปรเฉพาะที่ออกจากกองซ้อน
   (Pop the local variables from the stack)
2. เก็บค่าใหม่ของเรจิสเตอร์ R0 ... R15 จากกองซ้อน
   (Restore the registers R0 ... R15 from the stacks.)
3. ส่งกลับไปยังโปรซีเดอร์เรียก (RETURN to the invoking procedure) และเอาออกเลขที่อยู่ส่งกลับจากกองซ้อน ในเวลาเดียวกัน

ตัวอย่าง เพื่อแสดงให้เห็นข้อตกลงเหล่านี้ จงพิจารณาโปรแกรม Pascal และโปรซีเดอร์ข้างล่างนี้

Address

```
                Program P;
                    var i, j : integer;
                    procedure Q;
                        var j, k : integer;
                        procedure R;
                            var i, k : integer;
                            begin
                                writeln('enter & exit R')
                            end:
                        begin
2                           writeln('enter Q');
3                           R;
4                           writeln('exit Q')
                        end;
                    procedure S;
                        var i, j : integer;
                        begin
5                           writeln('enter S');
6                           Q;
7                           writeln('exits')
                        end;
```

```
            begin
 8              writeln('enter P');
 9              S;
10              writeln('exit P')
            end.
```

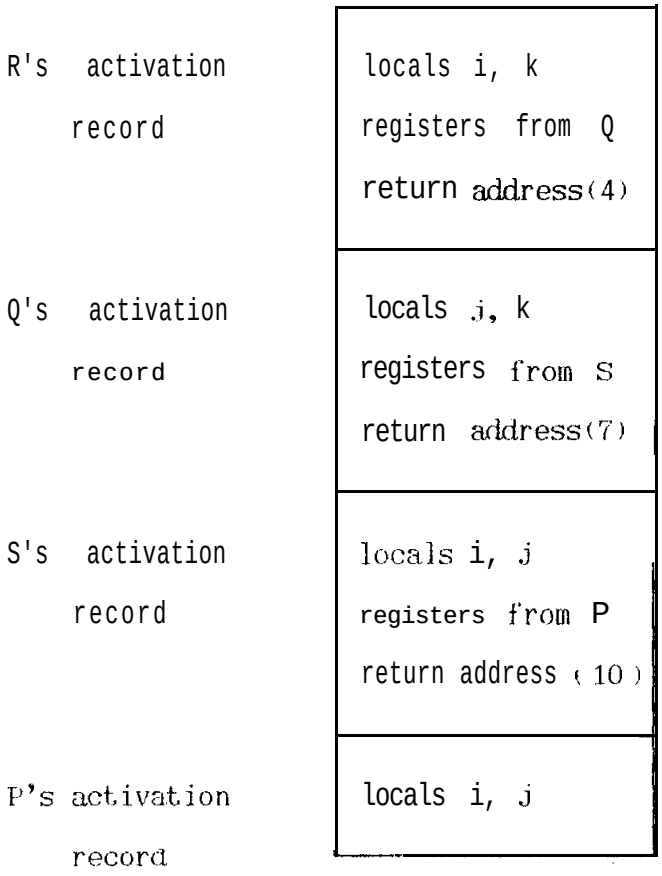
เพื่อให้ง่าย ในที่นี้ เลขที่อยู่ กำหนดให้เฉพาะ คำสั่งปฏิบัติการ (executable instructions) ในลำดับที่มันปรากฏ โดยไม่สนใจว่า ความเป็นจริงนั้น คำสั่งเหล่านี้แต่ละคำสั่งสมนัยกับลำดับ ของคำสั่งเครื่อง ซึ่งจะใช้เนื้อที่มากกว่าหนึ่งไบต์ หลักของการโยงโปรซีเดอร์ อย่างไรก็ตาม ไม่ใช่กระทบกับความง่ายเหล่านี้

ลำดับการปฏิบัติโปรซีเดอร์ สำหรับโปรแกรมนี้ เกิดขึ้นตามลำดับดังนี้

Address	*Activated*	Output	Run-time stack actions
8	P	enter P	PUSH locals(i, j)
9	S		CALL S (push 10)
			save registers from P
5		enter S	PUSHlocals(i, j)
6	Q		CALL Q (push 7)
			save registers from S
2		enter Q	PUSH locals(j, k)
3	R		CALL R (push 4)
			save registers from Q
1		enter &	PUSH locals(i, k)
		exit R	POP locals(i, k)
			restore registers for Q

Address	Activated	Output	Run-time stack actions
4	Q		RETURN
		exit Q	POP locals(j, k)
			restore registers for S
7	S		RETURN
		exit S	POP locals(i, j)
			restore registers for P
10	P		RETURN
		exit. P	POP locals(i, j)

ในตอนกลางของลำดับของเหตุการณ์ต่างๆ นี้ เมื่อ โปรซีเดอร์ R ถูกปฏิบัติ กองซ้อนเวลาดำเนินงาน จะประกอบด้วย ระเบียนการใช้งานต่อไปนี้



จะเห็นได้ชัดเจนว่า การจัดองค์กรเวลาดำเนินงานนี้ (this run-time organization) ครอบคลุมคุณสมบัติ สองอย่างของภาษา คือ

(1) แนวคิดของขอบเขตและโครงสร้างบล็อก (The notion of scope and block structure)

(2) การเวียนบังเกิด (Recursion)

จะเห็นว่า ขอบเขตของตัวแปรเฉพาะที่ทั้งหมด ประกาศภายในหนึ่งโปรซีเดอร์ จะถูกคั่นอัตโนมัติ (the scope of all location variables declared within a procedure is automatically delimited.)

ตัวอย่างเช่น การอ้างทั้งหมดถึง i และ k ในโปรซีเดอร์ R หาได้ (resolve) โดยการดูที่ระเบียนการใช้งานของ R อย่างไรก็ตาม การอ้างถึง ตัวแปรครอบคลุม i จะหาไม่ได้ โดยดู

ที่การค้น โดยตรง ในกองซ้อน ตัวอย่างเช่น ต้องการอ้างถึง ตัวแปรครอบคลุม i จากโปรซีเดอร์ Q การค้นโดยตรงของกองซ้อน จากบนสุด จะให้ตัวแปร i ประกาศใน S ก่อนที่จะถึงตัวแปร i ประกาศใน P ซึ่งจะเป็นการเลือกถูกต้อง

ณ จุดนี้คือ โครงสร้างแบบคงที่ ของ โปรซีเดอร์ ซ้อนกัน นิยาม ขอบเขตของตัวแปรในขณะที่ โครงสร้างแบบพลวัต ไม่เหมือนกันเสมอไป (the **static** structure of nested procedures defines the scope of variables, while the **dynamic** structure is not always the same.) ดังนั้น ขอบเขตของตัวแปรครอบคลุม จะถูกแทนที่อย่างถูกต้อง ณ เวลาดำเนินงาน อย่างไร? ผลเฉลยอยู่ในการวิเคราะห์เชิงวากยสัมพันธ์ของโปรแกรม เมื่อโครงสร้างแบบคงที่ของโปรแกรม ปรากฏจริง ณ เวลานั้น การอ้างถึงแต่ละชุดของตัวแปรครอบคลุม จะหาได้โดยการแสดงชัดแจ้ง ของโปรซีเดอร์ ซึ่งมันถูกประกาศ ดังนั้น การอ้างถึง i ภายใน Q จะถูกต่อท้ายเป็น อ้างถึง i.P (นั่นคือ เป็รเวอร์ชันของ i ประกาศใน P) ไม่ใช่ i.S ดังนั้น ณ เวลาดำเนินงาน การค้น กองซ้อน จะดำเนินต่อไป จนกระทั่ง ถึง ระเบียนปฏิบัติการ P ซึ่งจะพบตำแหน่งที่อยู่ ของ หน่วยตัวอย่างถูกต้องของ i ข้อดีประการที่สองของกองซ้อนเวลาดำเนินงาน คือ มันสนับสนุนการเวียนบังเกิด เราจะแสดง ให้เห็นสิ่งนี้ ในหัวข้อถัดไป หลักจาก ได้แนะนำการทำให้เกิดผลของ การโยงพารามิเตอร์

การโยงพารามิเตอร์และอาร์กิวเม้นต์ (Linkage of Parameters and Arguments) เมื่อเรียก โปรซีเดอร์ หรือ ฟังก์ชัน จะมีการส่ง อาร์กิวเม้นต์จำนวนหนึ่ง และจำนวนนี้สมนัยหนึ่งต่อหนึ่ง กับ พารามิเตอร์ ใน การประกาศของ โปรซีเดอร์ พารามิเตอร์แบ่งออกเป็นสามชนิด ดังนี้

- อินพุทพารามิเตอร์ ส่งค่าให้กับ โปรซีเดอร์ หรือ ฟังก์ชัน (Input parameters supply values to the procedure or function)
- เอ้าพุทพารามิเตอร์ รับมอบผลลัพธ์จาก โปรซีเดอร์ หรือ ฟังก์ชัน (Output parameters deliver results from the procedure or function.) ภาษาส่วนใหญ่ ชื่อของฟังก์ชัน โดยตัวมันเอง ทำบทบาทของ เอ้าพุทพารามิเตอร์ชัดเจน (distinguished output parameter)

เอ้าพุทพารามิเตอร์อื่นๆ ทั้งหมด ในฟังก์ชัน ปกติถูกจำแนกเป็น ผลกระทบข้างเคียง

. อินพุท-เอ้าพุทพารามิเตอร์ ส่งค่าให้และรับมอบผลลัพธ์ จาก โปรซีเดอร์ หรือ ฟังก์ชัน (Input-output parameters supply values to and deliver results from the procedure of function.) โดยปกติ พารามิเตอร์ แถวลำดับ สำหรับโปรซีเดอร์เรียงลำดับ มีคุณลักษณะนี้

ภาษาต่างๆ สนับสนุนทางเลือกเหล่านี้ ในวิธีต่างๆ กัน บางภาษาบังคับการใช้ถูกต้องอย่างเข้มงวด ในความรู้สึกที่ว่าอาร์กิวเม้นต์แต่ละตัว สมนัยกับ อินพุทพารามิเตอร์ จะต้องไม่เปลี่ยนแปลงในตัว (body) ของโปรซีเดอร์ และอาร์กิวเม้นต์แต่ละตัว สมนัยกับ เอ้าพุทพารามิเตอร์ ต้องเปลี่ยนแปลง ภาษาอื่นๆ ส่งความรับผิดชอบนี้ อย่างสมบูรณ์ ให้กับนักเขียนโปรแกรม

นอกจากนี้แล้ว การทำทางเลือกต่างๆ ให้เกิดผล สำหรับพารามิเตอร์ที่มีอยู่ ชนิดต่างๆ ได้สรุปไว้ข้างล่างนี้ ในที่นี้ p หมายถึง พารามิเตอร์, a หมายถึง อาร์กิวเม้นต์ที่สมนัยกัน และ f หมายถึง โปรซีเดอร์ หรือ ฟังก์ชัน

1. **Call by value** ณ เวลาของการเรียก อาร์กิวเม้นต์ a ถูกประเมินผล และค่าของมัน ถูกทำสำเนา (copied) ใน เนื้อที่หน่วยเก็บเฉพาะที่ ซึ่งสมนัยกับ พารามิเตอร์ p เนื้อที่นั้น ถูกใช้โดย การอ้างทั้งหมดถึง p ระหว่างการปฏิบัติการของ f และมันจะถูกทำลาย (รวมทั้ง ตัวแปรเฉพาะที่อื่นๆ ทั้งหมด) เมื่อการเรียกสิ้นสุด

2. **Call by reference** ณ เวลาของการเรียก อาร์กิวเม้นต์ a กลายเป็น มีความหมายเหมือนกับ พารามิเตอร์ p และการอ้างทั้งหมดถึง p ภายใน f เป็นการอ้างอย่างมีประสิทธิผล กับเนื้อที่หน่วยเก็บ สำหรับ a ตัวมันเอง อาร์กิวเม้นต์ a ในกรณีนี้ ต้องเป็น ตัวแปร, แถวลำดับ, หรือ ชื่อโครงสร้าง ซึ่งชนิดตรงกันกับ p

3. **Call by name** ในที่นี้ นิพจน์ a จะไม่ถูกประเมินผล ณ เวลาของการเรียก และแทนที่มันตามตัวอักษร หน่วยทั้งหมดของ p เหมือนกับ textual portion ของ โปรแกรมต้นฉบับ จากนั้น ข้อความผลลัพธ์ ของ f จะถูกปฏิบัติการ (อย่างดีความหมายได้)

4. **Call by value-result** สิ่งนี้เป็นชนิด double-ended call by value เพราะว่า ค่าของ a ถูกเก็บในเนื้อที่หน่วยเก็บเฉพาะที่ สำหรับ p ณ การเริ่มต้น ของ การเรียก และค่าสุดท้าย ของ เนื้อที่นั้น ถูกทำสำเนา กลับไปตัวแปร a ณ ตอนจบของการเรียก

ไม่ใช่ข้อตกลงทั้งหมดนี้ จะแสดงให้เห็นได้ในหนึ่งภาษา ดังนั้น สมมติว่าเป็นภาษา Pascal ซึ่งทางเลือกทั้ง 4 ชนิดนี้ มีให้ใช้ คำสงวน **value, var, name** และ **value-result** ตามลำดับ จะนำมาใช้ เพื่อแสดงข้อตกลงการโยง 4 ชนิดนี้

ตัวอย่าง จงพิจารณา โปรซีเดอร์ Sigma ข้างล่างนี้ ซึ่งคำนวณ ผลบวกของ สมาชิกทั้งหมด n ตัวของแถวลำดับ A

```
procedure Sigma(value A : array; value n : integer;
                                  var Sum : real);
    var i : integer;
    begin
          Sum := 0;
          for i := 1 to n do
               sum := Sum t A[i]
    end;
```

ตามกฎทั่วไป **value** สมนัยกับ อินพุทพารามิเตอร์อย่างเข้มงวด และ **var** สมนัยกับเอ้าพุทพารามิเตอร์อย่างเข้มงวด ข้อยกเว้น (an exception) ปกติเกิดขึ้นกับ แถวลำดับอินพุท ซึ่งบ่อยครั้ง ประกาศเป็น พารามิเตอร์ **var** เพื่อหลีกเลี่ยง การทำสำเนาที่ไม่จำเป็นของแถว-ลำดับขนาดใหญ่ ใน ตำแหน่งชั่วคราว ซึ่งสูญเปล่าทั้งเวลา และเนื้อที่

ดังนั้น การเรียก

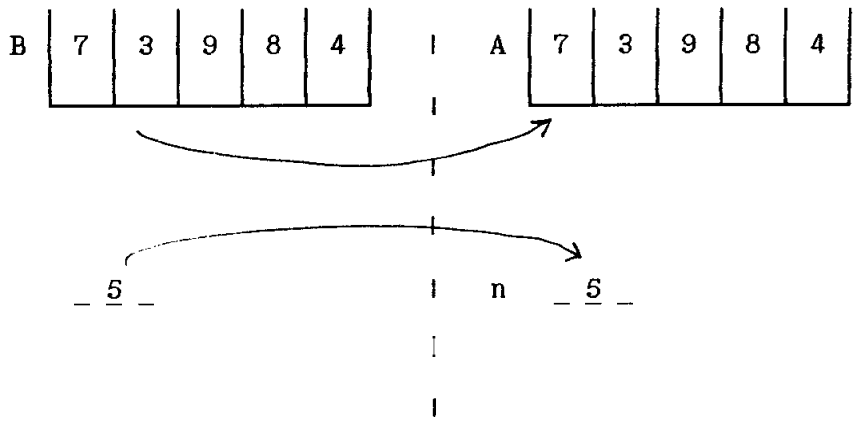
Sigma(B, 5 Bsum)

เมื่อ B มีค่า

7	3	9	8	4

จะให้ผลลัพธ์ ในการโยงพารามิเตอร์ ดังนี้

(i) ค่าต่างๆ ของ B และ 5 จะถูกทำสำเนา ในเนื้อที่เฉพาะที่ เกี่ยวข้องกับพารามิเตอร์ A และ n



(ii) พารามิเตอร์ Sum มีความหมายเหมือนกับ อาร์กิวเม้นต์ Bsum

Bsum - - - - - <———> Sum

ขณะนี้โปรซีเดอร์ถูกปฏิบัติการ และผลกระทบทั้งหมดบนพารามิเตอร์ Sum หมายถึง กำลังเกิดขึ้นจริงภายในตำแหน่งหน่วยเก็บ Bsum เพราะมันคือสิ่งเดียวกัน

Bsum ~~7~~ ~~10~~ ~~19~~ ~~27~~ 31 <———> Sum

ในที่สุด ตัวแปรเฉพาะที่ทั้งหมด รวมทั้งเนื้อที่หน่วยเก็บที่สร้างขึ้น สำหรับ A และ n ถูกทำลาย และการควบคุม ย้อนกลับ ไปยัง โปรซีเดอร์เรียก ด้วย ผลลัพธ์ ซึ่งอยู่ใน Bsum

จากตัวอย่าง จะเห็นชัดว่า เอ้าพุทพารามิเตอร์ต้องเป็น called by reference ถ้าเป็น called by value ผลลัพธ์จากการคำนวณของมัน จะทิ้งในเนื้อที่หน่วยเก็บเฉพาะที่ ซึ่ง มันจะหายไป เมื่อส่งกลับจาก โปรซีเดอร์

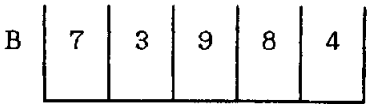
ตัวอย่าง แสดงให้เห็น call by name จงพิจารณา โปรซีเดอร์ Sigma ข้างล่างนี้

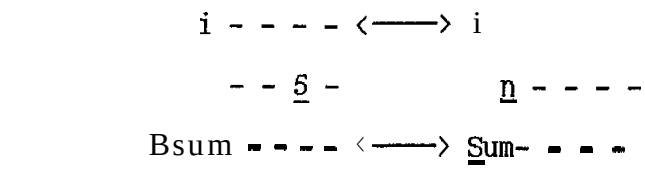
```
procedure Sigma(name A; var i : integer;
                value n : integer; var Sum : real;
    begin
          Sum := 0;
          for i := 1 to n do
            Sum:=SumtA
    end;
```

ขณะนี้ จงพิจารณา การเรียกข้างล่างนี้ ซึ่งออกแบบ ให้มีผลอย่างเดียวกับโปรซีเดอร์ข้างต้น

```
Sigma(B[i], i, 5, Bsum)
```

ในที่นี้ การเปลี่ยนแปลง n และ Sum เหมือนกับตัวอย่างก่อนหน้านี้ แต่ขณะนี้ เรากำลังส่งรายละเอียดของ "สมาชิกทั่วไป" B[i] และ "อาร์กิวเม้นต์ชัดแจ้ง" i ถูกใช้เหมือนดรรชีล่าง การเปลี่ยนแปลงที่สำคัญที่นี้คือ การกำหนดว่า พารามิเตอร์ A เป็น "call by name" ผลกระทบสำหรับการเรียกนี้คือ การแทนที่ หน่วยทั้งหมดของ ชื่อ A ใน text ของโปรซีเดอร์ ด้วยนิพจน์ B[i] จากนั้นปฏิบัติการ text ผลลัพธ์ ดังนั้น การเรียกนี้ เรามี text ผลลัพธ์และการเกี่ยวข้องดังนี้






```
begin
        sum := 0;
        for i := 1 to n do
            Sum := Sum t B[i]
end
```

แต่ power ของ call by name คือทำให้ พารามิเตอร์ กลายเป็น "generic" นั่นคือ โปรซีเดอร์ Sigma ข้างต้น นำไปใช้ได้ดี ในการคำนวณ ผลคูณภายใน (inner product) ของเวกเตอร์ สองชุด

$$\sum_{i=1}^{n} b_i c_i$$

โดยการเรียกดังนี้

```
Sigma(B[i]*C[i], i, 5, Sum)
```

ในที่นี้ text ผลลัพธ์ ของ ตัวโปรซีเดอร์ จะเป็นดังนี้

```
begin
     Sum := 0;
     for i := 1 to n do
          Sum := Sum t B[i]*C[i]
end
```

Call by value-result บางที มีประโยชน์น้อยกว่า กลไกการส่งพารามิเตอร์อื่นๆ เช่นที่เราเห็นในบทที่ 13 ภาษา Ada สนับสนุนสิ่งนี้ โดยวิธีของ การออกแบบ **in-out** ซึ่งเสนอแนะ พารามิเตอร์หนึ่งตัว ซึ่ง ใช้สำหรับอินพุทและเอ้าพุท ทั้งคู่ ในการเรียกหนึ่งครั้ง

โดยทั่วไป ภาษาต่าง ๆ สนับสนุน ยุทธวิธีการโยงพารามิเตอร์ หนึ่ง, สอง หรือมากกว่า ของ 4 วิธีนี้ ตัวอย่างเช่น เรียกโดยการอ้างถึง เป็นการใช้ที่แพร่หลายมากที่สุด และสนับสนุน โดยภาษา Pascal, FORTRAN, COBOL, Pl/1, SNOBOL, APL, LISP, C และ Ada ส่วนการเรียกโดยค่า ไม่สนับสนุนน โดยภาษา FORTRAN, COBOL, SNOBOL หรือ APL การเรียกโดยชื่อ เป็นสิ่งอำนวยความสะดวกเริ่มต้นจากภาษา Algol 60 แต่ ไม่มีความสำคัญคงอยู่ในภาษาปัจจุบัน ภาษา PROLOG มีวิธีของตนเองของการโยงพารามิเตอร์-อาร์กิวเม้นต์ ซึ่งแตกต่างจาก วิธีต่างๆ ทั้งหมด ซึ่งเราจะอภิปรายในบทที่ 11

บางภาษา เช่น *PL/1 อนุญาตให้เลือก ระหว่าง เรียกโดยอ้างอิง กับ เรียกโดยค่า* กระทำโดย **การเรียก** (invocation) ไม่ใช่ การประกาศโปรซีเดอร์ หลักของ PL/1 สำหรับการทำความแตกต่างนี้ ขึ้นอยู่กับ ธรรมชาติ ของ อาร์กิวเม้นต์ a และ พารามิเตอร์ p ในวิธีข้างล่างนี้

1. ถ้า a และ p เป็นข้อมูลชนิดแตกต่างกัน ให้ใช้เรียกโดยค่า
2. ถ้า a เป็นนิพจน์ด้วยตัวปฏิบัติการ ให้ใช้เรียกโดยค่า
3. ถ้า a เป็นค่าคงที่ ให้ใช้เรียกโดยค่า

ส่วนกรณีอื่นๆ ทั้งหมด (ตัวอย่างเช่น a เป็นตัวแปร ชนิดเดียวกับ p) ให้ใช้เรียกโดยอ้างอิง

ดังนั้น การบังคับชนิด (type coercion) กระทำ โดยอัตโนมัติ ใน PL/1 เมื่อใดที่ อาร์กิวเม้นต์ a คือเรียกโดยค่า สิ่งนี้ทั้งห่างจาก ภาษาชนิด strongly type เช่น Pascal

เมื่อ ชนิดการบังคับ ระหว่าง อาร์กิวเม้นต์ a และพารมิเตอร์ p ของมัน ทำไม่ได้ กล่าวคือ a และ p ปกติต้องเป็นชนิดเดียวกัน มิฉะนั้นแล้ว จะเกิด**ข้อผิดพลาดวากยสัมพันธ์** (syntax error)

การทำให้เกิดผลของโปรซีเดอร์ และพารามิเตอร์

(Implementation of Procedures and parameters)

การโยงของ call by reference และ call by value parameters ทำให้เกิดผลในวิธีที่แตกต่างกัน ขึ้นอยู่กับ ภายใต้ ข้อตกลง ซึ่งใช้โดยคอมพิวเตอร์แม่งาน (host machine) สำหรับคอมพิวเตอร์ของเรา และข้อตกลงระเบียนปฏิบัติการ วิธีข้างล่างนี้คือ สิ่งที่กระทำ

1. สำหรับ call by reference พารามิเตอร์ p แต่ละครั้ง เลขที่อยู่ p จะถูกใส่ให้กับตัวแปรเฉพาะที่ ในระเบียนปฏิบัติการ ในตำแหน่งนั้น แอดเดรส ของ อาร์กิวเม้นต์ที่สมนัยกัน ถูกเก็บ ณ เวลาของการเรียก และการอ้างถึงทั้งหมด กับพารามิเตอร์นั้น คือ การอ้างถึงโดยอ้อม (@P) ซึ่งใช้แอดเดรสนั้น

2. สำหรับ call by value พารามิเตอร์ p แต่ละครั้ง ตัวแปรเฉพาะที่ p ถูกสร้างขึ้นในระเบียนปฏิบัติการ และทำสำเนา (copy) ค่าของอาร์กิวเม้นต์ที่สมนัยกัน ในเลขที่อยู่นั้น การอ้างถึงทั้งหมดของ p ภายในตัวโปรซีเดอร์ จะเป็นการอ้างถึง เลขที่อยู่นั้น และค่าสุดท้ายของมัน ณ เวลา RETURN จากโปรซีเดอร์ จะหายไป เพราะว่า ระเบียนปฏิบัติการ ทั้งหมดจะถูกเอาออก จากกองซ้อน

เพื่อแสดงให้เห็นข้อตกลงเหล่านี้ จงพิจารณาโปรซีเดอร์ factorial ข้างล่างนี้

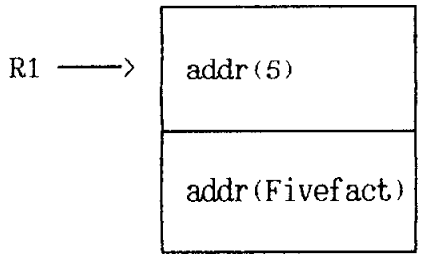
```
procedure factorial(n : integer; var f : integer);
    var i : integer;
    begin
        f := 1;
        for i := 2 to n do
            f := f*i
    end;
```

สมมติให้มีการสร้าง "argument address list" โดยโปรซีเดอร์เรียก และเลขที่อยู่ ของรายการนี้ เก็บในเรจิสเตอร์ R1 เลขที่อยู่ แต่ละชุด ในรายการนี้ ประกอบด้วย เลขที่อยู่ ของ อาร์กิวเม้นต์ ในการเรียก

ตัวอย่าง การเรียก

factorial(5, Fivefact)

ใน R1 จะประกอบด้วยเลขที่อยู่ของรายการต่อไปนี้



ดังนั้น ค่า 5 กำหนดโดย การอ้าง โดยอ้อมคู่ @@R1 และเลขที่อยู่ของตัวแปร Fivefact กำหนดโดย การอ้างโดยอ้อม @(R1+4) รหัสเครื่อง สำหรับโปรซีเดอร์ factorial จะเป็นดังนี้

```
I D S   L .    ; I is an long integer value (4 bytes)
FACT PUSHALL RO, R15     ; save registers from invocation
   PUSH I                ; local variable
   PUSH @ (R1+4)         ; addr of var parameter f
   PUSH @@ R1            ; copy of parameter value n
; New all references to parameter n below are given by STACK, all
; references to parameter f are given by @ (STACK-4), and all re-
; ferences to i by STACK-8

   MOV #1, @(STACK-4)    ; f := 1
   MOV #2, STACK-8       ; i := 1
```

```
$L0001 CMP  STACK-8, STACK     ;if  i>n
       B     G        $L0002 ; then exit  the  loop
       MOV @(STACK-4), R2
       MUL  R2, STACK-8
       MOV R2,  @(STACK-4)     ; f := f*i
       MOV  STACK-R,  R3
       ADD  #1, R3
       MOV  R3,  STACK-8        ; i := i t 1
       B    $L0001
; New begin the return sequence
$L0002  POP                     ; destroy n
       POP                      ; destroy @f
       POP                      ; destroy local variable i
       POPALL  R0, R15          ; restore registers
       RETURN
```

(ในที่นี้ สมมติว่า มี additional macros PUSHALL และ POPALL อยู่ สำหรับความสะดวกของการเก็บ (saving) และการปล่อย (restoring) เรจิสเตอร์ทั้งหมด ใน หนึ่งบรรทัดของรหัส ภาษาแอสเซมบลีส่วนใหญ่ มีคำสั่งชนิดนี้ ใน หนึ่งแบบ หรือ รูปแบบอันอื่น)

รหัส ซึ่งกำหนดในที่นี้ สะท้อน การทำสำเนาโดยตรง ของ แต่ละคำสั่งใน โปรซีเดอร์ Pascal เริ่มต้น โดยไม่มีการสังเกตศักยภาพ ความเหมาะสม ซึ่งจะทำให้สำเร็จ สิ่งนี้คล้ายกับเอ้าพุทของตัวแปลชุดคำสั่งซึ่งไม่เล็งผลเลิศ (nonoptimizing compiler) แน่นอน ถ้า รหัสภาษาเครื่องถูกกระทำ โดยตรงด้วยมือ หรือโดยตัวแปลชุดคำสั่งแบบเล็งผลเลิศ (optimizing compiler) การย้ายซ้ำ (redundant moving) ของ i และ f ในและนอกเรจิสเตอร์ R2 และ R3 จะถูกขจัดออกไป

การโยงสำหรับโปรซีเดอร์เวียนบังเกิด

(Linkage for Recursive Procedure)

ข้อตกลงข้างต้น พอเพียงสำหรับการทำให้เกิดผลของ โปรซีเดอร์เวียนบังเกิด เช่นเดียวกัน เนื่องจาก ค่าของ ตัวแปรเฉพาะที่แต่ละตัวสำหรับการเรียกแต่ละครั้ง เก็บใน ระเบียนปฏิบัติการแยกต่างหากจากกัน บนกองซ้อนเวลาดำเนินงาน เพื่อแสดงให้เห็น จะพิจารณาเวอร์ชันการเวียนบังเกิดของ ฟังก์ชัน factorial ข้างล่างนี้

```
function factorial(n : integer) : integer;
    begin
        if  n <= 1  then  factorial := 1
        else  factorial := n * factorial(n-1))
    end;
```

ตามข้อตกลง สมมติว่า ผลลัพธ์ ส่งกลับโดย ฟังก์ชันอยู่ในเรจิสเตอร์ R0 ดังนั้นการปฏิบัติการแต่ละครั้งของแฟกทอเรียล ต้องทิ้งผลลัพธ์ใน R_0 ทันที ก่อนการออกไป และไม่มีปฏิบัติการใดๆ กับ R0 ในเวลานั้น

```
TEMP      DS  L              ; temporary value of n-l
TEMPADDR  DS  A              ; address of TEMP
FACT      PUSHALL R1, R15 ; save all but R0
          PUSH  @@R1         ; value of parameter n
          MOV #1, R0         ; assume result = 1 (n <= 1)
          CMP STACK, #1
          BLE $L0001
          MOV  STACK, R2     ; n > 1, prepare for call
          SUB #1, R2         ; compute  n-l
          MOV R2, TEMP
```

```
                MOV  @TEMP, TEMPADDR
                MOV  @TEMPADDR, R1      ; argument linkage
                CALL FACT
                MUL  STACK, R0          ; n * factorial(n-1)
$L0001          POP                     ; return sequence
                POPALL R1, R15
                RETURN
```

ไม่เหมือน R0, มูลค่าของ R1 และ R2 เก็บและเอาออก ณ แต่ละระดับของการเวียนบังเกิด ดังนั้น การเรียกแต่ละครั้ง ไม่จำเป็นต้องเกี่ยวกับการทำลายค่าของอีกอันหนึ่ง ในเรจิสเตอร์เหล่านี้ ผู้อ่านควรจะตามรอย (trace) การปฏิบัติการของรหัสนี้ พูดอีกอย่างหนึ่งคือ แฟกทอเรียล 5 เพื่อดู การเปลี่ยนแปลงของการเรียกเวียนบังเกิด กับ กองซ้อนเวลาดำเนินงาน

แบบฝึกหัด

1. สำหรับแต่ละคำสั่งข้างล่างนี้ จงอภิปราย การบังคับ (coercions) ที่เป็นไปได้ซึ่งสามารถเกิดขึ้น

   (a) DO 10 I=1, N (ภาษา FORTRAN)

   (b) READ EMPLOYEE-REC FROM EMPLOYEE-FILE (ภาษา COBOL)

   (c) X <— Y + 3 (ภาษา APL)

   (d) DO I = 1 TO N BY .5; (ภาษา PL/1)

   (e) X'HI' = ' BYE' (ภาษา SNOBOL)

   (f) (setq x (car Y)) (ภาษา LISP)

2. จงเขียนรหัสเครื่องสำหรับข้อย่อย (c) ของคำถามข้อ 1 โดยสมมติว่า

   (a) X และ Y เป็นตัวแปรชนิด scalar real

   (b) X และ Y เป็นแถวลำดับ หนึ่งมิติของตัวแปรชนิด real

3. ทำเช่นเดียวกัน กับ ข้อย่อย (e) ของคำถามข้อ 1 โดยสมมติว่า X เป็นสายอักขระ ความยาวสูงสุด 100 การเก็บใช้ข้อตกลงที่อธิบายไปแล้วในบทนี้

4. สำหรับแถวลำดับ A, B และ C ซึ่งประกาศ ในหัวข้อ 10.3

   (a) จงใช้สูตรคำนวณเลขที่อยู่ ของ A[3], B[2,3] และ [5,3,2]

   (b) จงใช้สูตร สำหรับการคำนวณเลขที่อยู่ของ A[i], B[i,j] และ C[i,j,k] สมมติว่ามีการเก็บในอันดับแบบสดมภ์ เช่น ภาษา FORTRAN

   (c) ทำให้เกิดผลในรหัสเครื่อง สำหรับการคำนวณเลขที่อยู่ของ B[i,j) สมมติว่าเก็บในอันดับแบบสดมภ์

5. สำหรับ machine implementation ที่กำหนดให้ ใน เวอร์ชันไม่เวียนบังเกิดของโปรซีเดอร์ factorial จงตามรอย (trace) มูลค่าของกองซ้อนเวลาดำเนินงาน, ตัวแปรเฉพาะที่ และอาร์กิวเม้นต์ เมื่อคำนวณ factorial ของ 5

6. จงเขียนรหัสเครื่อง ซึ่ง จะเรียกโปรซีเดอร์ factorial อย่างถูกต้อง โดยใช้ 5 เป็นอาร์กิวเม้นต์ และข้อตกลงการโยงที่ได้อธิบายในหัวข้อ 10.5

7. จงตามรอย การปฏิบัติการของ เวอร์ชันเวียนบังเกิด ของฟังก์ชัน factorial ด้วยอาร์-กิวเม้นต์ 5 โดยการแสดงให้เห็นลำดับของระเบียนปฏิบัติการ, ตัวแปรเฉพาะที่ และมูลค่าของเรจิสเตอร์ทั้งหมด