6. สภาพแวดล้อมเวลาดำเนินงาน

การทำงานของโปรแกรมมักเกี่ยวข้องกับตัวแปรซึ่งมีหลายชนิด ตัวแปรบางตัวถูกกำหนดให้ใช้ได้ตั้งแต่เริ่มต้นจนจบการทำงานของโปรแกรม ตัวแปรบางตัวใช้ในระหว่างที่ฟังก์ชันหรือส่วนย่อยของโปรแกรมทำงานเท่านั้น ตัวแปลภาษาต้องจัดการเนื้อที่ในหน่วยความจำให้ตัวแปรเหล่านี้อย่างเหมาะสม หัวข้อนี้อธิบายการจัดการหน่วยความจำในโปรแกรมสำหรับตัวแปรชนิดต่างๆ หัวข้อ 6.1 อธิบายตัวแปรแบบต่าง ๆ ที่ใช้ในภาษาโปรแกรม หัวข้อ 6.2 อธิบายการจัดหน่วยความจำสำหรับตัวแปรชนิดต่าง ๆ หัวข้อ 6.3 อธิบายคำสั่งที่ตัวแปลภาษาต้องเพิ่มในรหัสกลางหรือรหัสเครื่องเพื่อจัดการเกี่ยวกับฟังก์ชัน

6.1 ชนิดของตัวแปรในภาษาโปรแกรม

ตัวแปรในภาษาโปรแกรมแบ่งตามการผูก (binding) กับหน่วยความจำและช่วงชีวิต (lifetime) ของตัวแปรได้เป็น 4 ชนิดคือ ตัวแปรแบบสถิต (static variable) ตัวแปรในสแตคแบบพลวัต (stack-dynamic variable) ตัวแปรในฮีปแบบกำหนดโดยปริยาย (implicit heap dynamic variable) และ ตัวแปรในฮีปแบบกำหนดชัดแจ้ง (explicit heap dynamic variable)

6.1.1 ตัวแปรแบบสถิต (static variable)

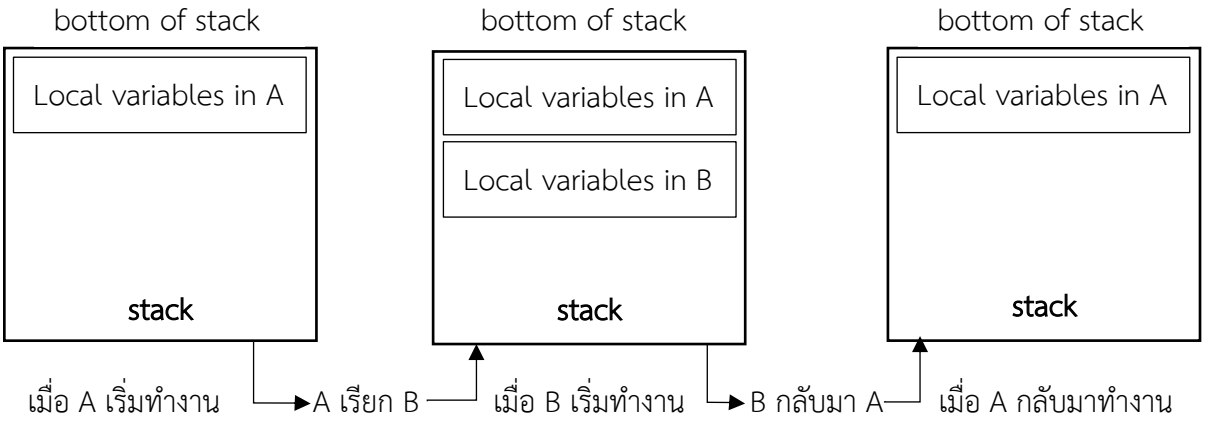
ตัวแปรแบบสถิตเป็นตัวแปรที่มีช่วงชีวิตยาวตลอดการทำงานของโปรแกรม ตัวแปรในภาษาซีที่ประกาศโดยมีคำว่า `static` กำกับเป็นตัวแปรแบบสถิต เช่น `static int x` ตัวแปลภาษาต้องกำหนดที่อยู่ในหน่วยความจำให้ตัวแปรชนิดนี้ตั้งแต่โปรแกรมเริ่มทำงานและตัวแปรนั้นจะผูกกับเลขที่อยู่ในหน่วยความจำที่ตำแหน่งนั้นตลอดการทำงาน ดังนั้นจึงสามารถอ้างถึงตัวแปรนี้ในรหัสเครื่องด้วยเลขที่อยู่นั้นตลอดการทำงานของโปรแกรม เช่น ถ้าประกาศตัวแปร `x` ดังที่ยกตัวอย่างไปแล้ว ตัวแปลภาษากำหนดเลขที่อยู่ในหน่วยความจำที่เก็บตัวแปร `x` ตั้งแต่โปรแกรมเริ่มทำงาน ในที่นี้สมมุติให้เป็นเลขที่อยู่ 0326 จากนั้นจะใช้เลขที่อยู่ 0326 นี้อ้างถึงตัวแปร `x` ในโปรแกรมได้เสมอ

6.1.2 ตัวแปรในสแตคแบบพลวัต (stack-dynamic variable)

ในภาษาโปรแกรมมีตัวแปรที่ใช้ได้เฉพาะในฟังก์ชันและมีช่วงชีวิตอยู่ในช่วงที่ฟังก์ชันนั้นทำงานเท่านั้น ตัวอย่างของตัวแปรประเภทนี้ ได้แก่ ตัวแปรเฉพาะที่ (local variable) ซึ่งประกาศไว้ในฟังก์ชันในภาษาซีและภาษาจาวา เนื่องจากตัวแปรประเภทนี้มีชีวิตอยู่เมื่อฟังก์ชันนั้นทำงาน แต่ฟังก์ชันหนึ่งอาจเรียกอีกฟังก์ชันหนึ่งทำงาน เช่น ฟังก์ชัน A เรียกฟังก์ชัน B ให้ทำงานซึ่งทำให้ฟังก์ชัน A ต้องหยุดการทำงานไปเมื่อฟังก์ชัน B เริ่มทำงาน ฟังก์ชัน A จะกลับมาทำงานอีกครั้งเมื่อฟังก์ชัน B ทำงานเสร็จ ดังนั้นตัวแปรในฟังก์ชัน A ต้องถูกเก็บไว้ เมื่อฟังก์ชัน B ทำงานเสร็จ ฟังก์ชัน A จะกลับมาทำงานต่อโดยต้องนำตัวแปรในฟังก์ชัน A กลับมาใช้ได้โดยไม่มีการเปลี่ยนค่าไปจากตอนที่ฟังก์ชัน A หยุดทำงาน

เนื่องจากการเรียก (call) ฟังก์ชันและการกลับ (return) จากฟังก์ชันมีลักษณะเหมือนการทำงานของสแตค คือ ฟังก์ชันที่ถูกเรียกหลังจะจบการทำงานก่อนในลักษณะเข้าหลัง-ออกก่อน (last-in, first-out) ตัวแปลภาษาจึงสามารถจัดการที่เก็บตัวแปรที่ใช้ในฟังก์ชันแบบสแตคได้เช่นกัน ตัวแปลภาษาจัดแบ่งที่ใน

หน่วยความจำไว้ใช้เป็นสแตคที่เก็บตัวแปรเหล่านี้ จากรูปข้างล่างนี้ ตัวแปลภาษาจะใช้ที่ว่างบนสแตคเพื่อเก็บตัวแปรในฟังก์ชัน A เมื่อฟังก์ชัน A เริ่มทำงาน หากฟังก์ชัน A มีการเรียกใช้ฟังก์ชัน B ต่อไปก็จะใช้ที่ถัดไปบนสแตคเพื่อเก็บตัวแปรในฟังก์ชัน B ดังนั้นหากฟังก์ชัน A ยังทำงานไม่เสร็จตัวแปรของฟังก์ชัน A จะยังถูกเก็บไว้ในสแตค ในขณะที่ด้านบนเป็นที่เก็บตัวแปรของฟังก์ชัน B เมื่อฟังก์ชัน B ทำงานเสร็จแล้ว ตัวแปรของฟังก์ชัน B ที่อยู่ด้านบนก็จะถูกนำออกไปจากสแตคซึ่งทำให้ตัวแปรของฟังก์ชัน A ปรากฏอยู่บนสุดของสแตคแทนและถูกนำกลับมาใช้ในฟังก์ชัน A ได้โดยมีค่าเดียวกับตอนที่ฟังก์ชันหยุดทำงาน ดังนั้นเลขที่อยู่ในหน่วยความจำที่ผูกกับตัวแปรจะไม่เปลี่ยนไประหว่างการทำงาน



6.1.3 ตัวแปรในฮีปแบบกำหนดชัดแจ้ง (explicit heap dynamic variable)

นอกจากตัวแปรสองชนิดที่ได้กล่าวไปแล้ว ภาษาโปรแกรมยังมีตัวแปรที่ผู้เขียนโปรแกรมสามารถระบุให้สร้างขึ้นมาหรือให้ลบตัวแปรนั้นทิ้งในขณะที่โปรแกรมทำงานได้ เช่น ในภาษาจาวา ผู้เขียนโปรแกรมสามารถใช้คำสั่ง `new` เพื่อขอเนื้อที่ในหน่วยความจำสำหรับตัวแปรหนึ่งได้ และใช้คำสั่ง `free` เพื่อเลิกใช้หน่วยความจำสำหรับตัวแปรนั้น ดังนั้นช่วงชีวิตของตัวแปรนั้นถูกกำหนดโดยผู้เขียนโปรแกรมซึ่งทำให้ตัวแปลภาษาไม่สามารถกำหนดช่วงชีวิตของตัวแปรนั้นได้ ตัวแปลภาษาจะแบ่งเนื้อที่ในหน่วยความจำสำหรับตัวแปรชนิดนี้แยกไว้ เมื่อมีคำสั่งขอใช้ที่สำหรับตัวแปรแบบนี้ก็จะขอเนื้อที่ในหน่วยความจำส่วนนี้มาใช้ พื้นที่ในหน่วยความจำส่วนนี้ เรียกว่า ฮีป (heap)

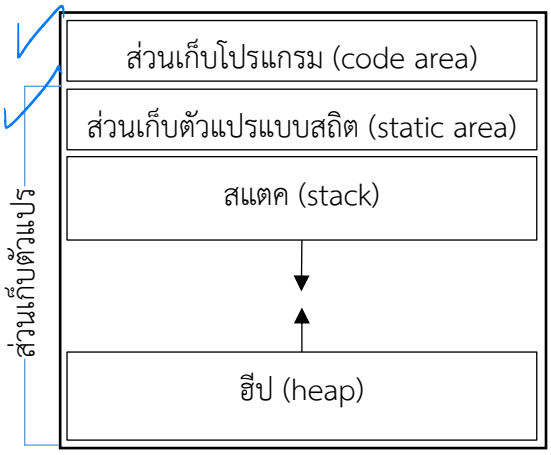
6.1.4 ตัวแปรในฮีปแบบกำหนดโดยปริยาย (implicit heap dynamic variable)

ในภาษาโปรแกรมที่อนุญาตให้ตัวแปรเปลี่ยนชนิดในขณะที่โปรแกรมทำงานได้ เมื่อชนิดของตัวแปรเปลี่ยนไป ขนาดของหน่วยความจำที่ต้องใช้ในการเก็บตัวแปรนั้นอาจเปลี่ยนไปด้วย เช่น เมื่อตัวแปร `x` มีชนิดเป็นจำนวนเต็มด้วยคำสั่ง `x=0` ตัวแปร `x` ใช้เนื้อที่ 4 ไบท์ ต่อมาเมื่อใช้คำสั่ง `x=[1,2,3,4]` `x` เป็นตัวแปรแถวลำดับที่เก็บจำนวนเต็ม 4 ค่าและต้องใช้เนื้อที่ 16 ไบท์ ดังนั้นตัวแปลภาษาต้องจัดหาเนื้อที่ในฮีปให้ใหม่เพื่อเก็บตัวแปร `x` การขอที่ในฮีปของตัวแปรแบบนี้ต่างกับตัวแปรในฮีปแบบกำหนดชัดแจ้ง คือ ผู้เขียนโปรแกรมไม่ได้กำหนดการขอใช้เนื้อที่หน่วยความจำ แต่ตัวแปลภาษาขอใช้เนื้อที่ให้โดยปริยายเมื่อมีการเปลี่ยนชนิดของตัวแปร

หัวข้อต่อไปอธิบายการจัดการหน่วยความจำสำหรับตัวแปรที่ใช้ในขณะที่โปรแกรมทำงาน

6.2 การจัดหน่วยความจำสำหรับตัวแปร

เนื้อที่ในหน่วยความจำที่ใช้ระหว่างการทำงานของโปรแกรมแบ่งเป็นส่วนที่เก็บโปรแกรมและส่วนที่เก็บตัวแปร ตัวแปลภาษาสามารถบอกได้ว่าเนื้อที่ส่วนเก็บโปรแกรมต้องมีขนาดเท่าไรเพราะส่วนนี้คือรหัสเครื่องที่สร้างโดยตัวแปลภาษาเอง แต่ตัวแปลภาษาไม่สามารถบอกได้ว่าหน่วยความจำส่วนที่ใช้เก็บตัวแปรของโปรแกรมต้องมีขนาดเท่าไรเพราะตัวแปรภายในฟังก์ชันจะถูกสร้างขึ้นเมื่อฟังก์ชันถูกเรียกใช้และตัวแปลภาษาไม่อาจบอกได้ก่อนโปรแกรมเริ่มทำงานว่า ฟังก์ชันใดจะถูกเรียกใช้กี่ครั้ง แต่ตัวแปลภาษาบอกได้ว่าตัวแปรแบบสถิตในโปรแกรมต้องการเนื้อที่ในหน่วยความจำเท่าไร ดังนั้นตัวแปลภาษาจะจัดเนื้อที่ในหน่วยความจำสำหรับโปรแกรมหนึ่งดังแสดงในรูปข้างล่างนี้



หน่วยความจำส่วนแรก คือ code area ใช้เก็บโปรแกรมที่จะทำงาน ส่วนถัดมา คือ static area ใช้เก็บตัวแปรแบบสถิต ขนาดของเนื้อที่สำหรับสองส่วนนี้ไม่เปลี่ยนแปลงระหว่างที่โปรแกรมทำงาน ส่วนถัดมาของหน่วยความจำใช้สำหรับตัวแปรในฟังก์ชันที่ถูกเก็บในสแตคซึ่งอาจมีการเพิ่มหรือลดขนาดของสแตคตามการเรียกใช้ฟังก์ชัน ส่วนสุดท้ายเป็นฮีปที่ใช้เก็บตัวแปรที่สร้างระหว่างการทำงานแต่ไม่ได้อยู่ในสแตค เนื่องจากเนื้อที่ของฮีปและสแตคมีการเพิ่มหรือลดระหว่างที่โปรแกรมทำงาน ตัวแปลภาษาจึงจัดให้สแตคและฮีปใช้หน่วยความจำที่ต่อกัน สแตคใช้เนื้อที่เริ่มจากเลขที่อยู่ต่ำและสามารถเพิ่มขนาดไปยังเลขที่อยู่สูงขึ้นได้ ส่วนฮีปใช้เนื้อที่เริ่มจากเลขที่อยู่สูงและสามารถเพิ่มขนาดไปยังเลขที่อยู่ต่ำลง ทั้งนี้ทำให้โปรแกรมใช้เนื้อที่สำหรับสแตคและฮีปร่วมกัน เช่น สแตคอาจใช้หน่วยความจำส่วนนี้ทั้งหมดในช่วงที่โปรแกรมมีการเรียกใช้ฟังก์ชันเวียนเกิดหลายครั้ง แต่เมื่อฟังก์ชันทำงานจบไปแล้วเนื้อที่ในสแตคที่ใช้สำหรับตัวแปรในฟังก์ชันจะถูกคืนกลับไป โปรแกรมอาจขอเนื้อที่สำหรับตัวแปรในฮีปแบบกำหนดชัดแจ้งที่เป็นตัวแปรแถวลำดับขนาดใหญ่โดยขอหน่วยความจำจากฮีปซึ่งอาจเป็นส่วนที่เคยใช้เก็บตัวแปรของฟังก์ชันมาก่อน นั่นคือหน่วยความจำส่วนหนึ่งอาจถูกใช้สำหรับสแตคในเวลาหนึ่งแต่ใช้สำหรับฮีปในอีกเวลาหนึ่งก็ได้

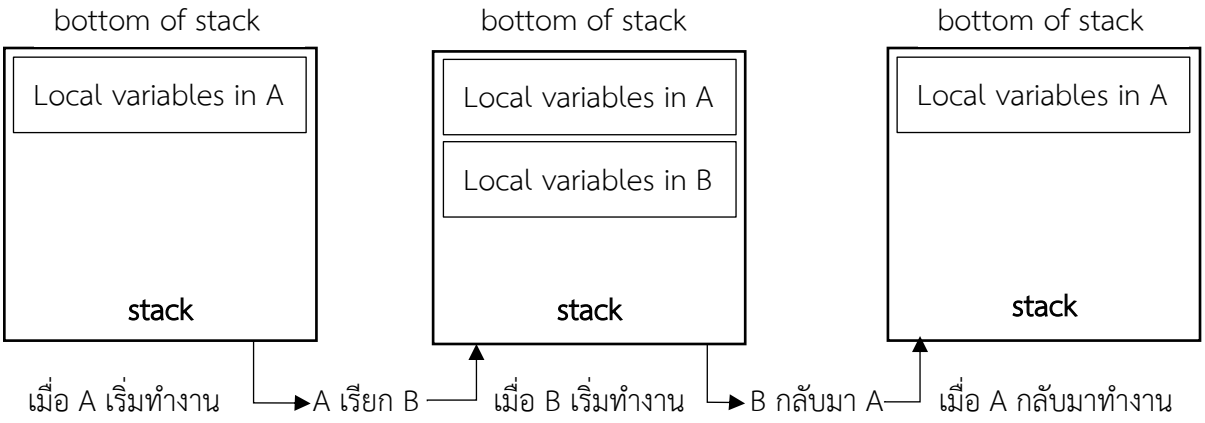
6.2.1 การจัดเก็บตัวแปรแบบสถิต

เนื่องจากตัวแปรแบบสถิตมีช่วงชีวิตตลอดการทำงานของโปรแกรม ตัวแปลภาษาจัดหน่วยความจำในเนื้อที่แบบสถิต (static area) และ กำหนดเลขที่อยู่ให้ตัวแปรแบบสถิตแต่ละตัวได้ตั้งแต่โปรแกรมเริ่มทำงาน

จากนั้นตัวแปลภาษาสามารถใช้เลขที่อยู่นี้อ้างถึงตัวแปรแบบสถิตในรหัสเครื่องตลอดการทำงานของโปรแกรม ดังนั้นตัวแปลภาษาจะเก็บเลขที่อยู่นี้ไว้ในตารางสัญลักษณ์และใช้ในการสร้างรหัสเครื่อง

6.2.2 การจัดเก็บตัวแปรในสแตคแบบพลวัต

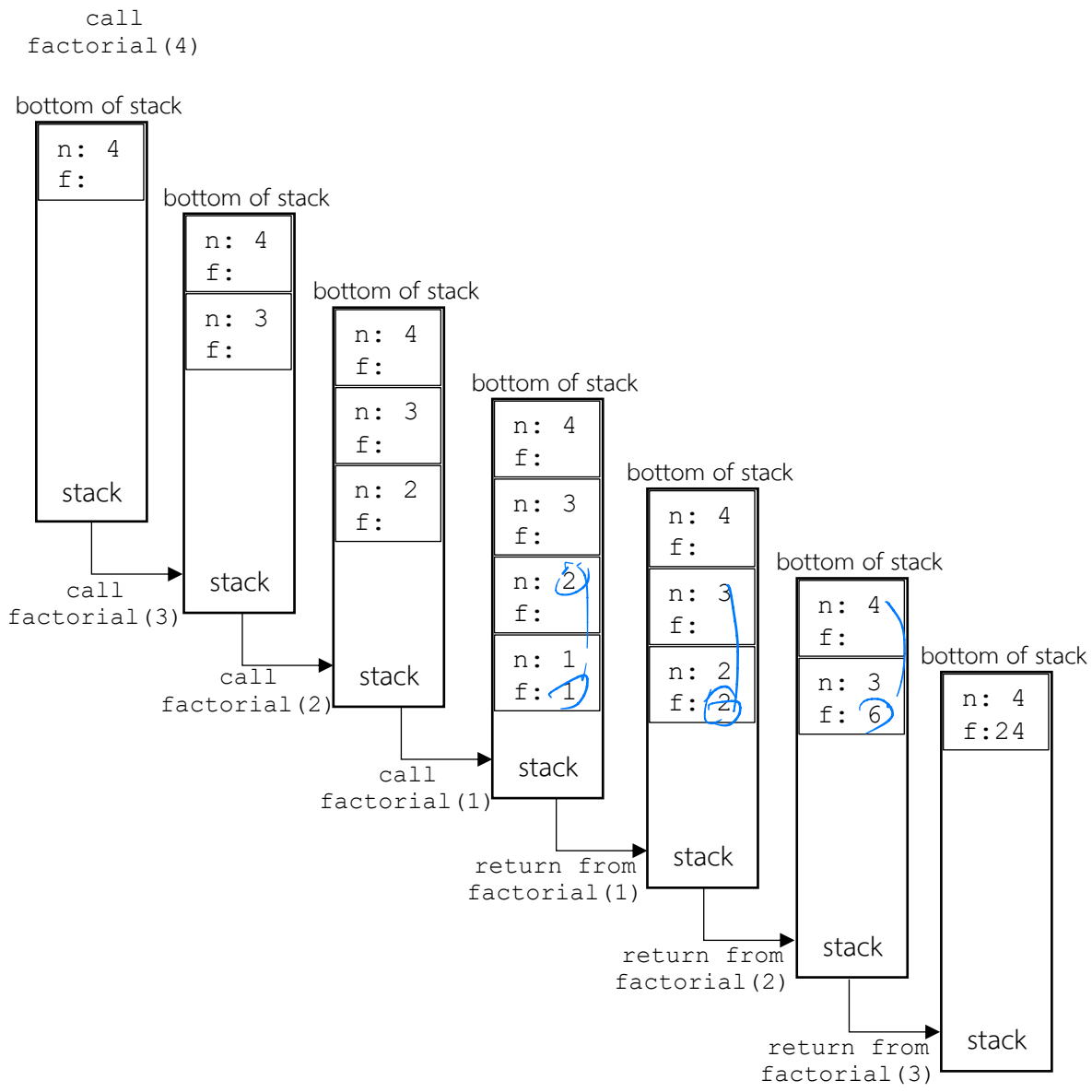
ตัวแปรเฉพาะที่ในฟังก์ชันถูกสร้างขึ้นเมื่อมีการเรียกใช้ฟังก์ชันและถูกเก็บไว้จนกระทั่งฟังก์ชันทำงานจบ ดังนั้นตัวแปรเหล่านี้ถูกเก็บไว้ในสแตค สมมติให้มีฟังก์ชัน A และ B ซึ่งฟังก์ชัน A เรียกใช้ฟังก์ชัน B เราเรียก A เป็นตัวเรียก (caller) และ B เป็นตัวถูกเรียก (callee) เมื่อฟังก์ชัน A ทำงาน ตัวแปรเฉพาะที่ของฟังก์ชัน A จะถูกเก็บไว้ในสแตค เมื่อฟังก์ชัน A เรียกใช้ฟังก์ชัน B ตัวแปรเฉพาะที่ของฟังก์ชัน A จะยังถูกเก็บไว้และตัวแปรของฟังก์ชัน B จะถูกสร้างเก็บไว้ด้านบนของสแตคดังแสดงนรูปข้างล่าง เมื่อฟังก์ชัน B จบการทำงานและย้อนกลับมาทำงานของฟังก์ชัน A ต่อ ตัวแปรเฉพาะที่ของฟังก์ชัน B ที่อยู่บนสุดในสแตคจะถูกป๊อบ (pop) ทิ้งและทำให้ตัวแปรเฉพาะที่ของฟังก์ชัน A กลับมาอยู่บนสุดในสแตคดังแสดงในรูปข้างล่าง พารามิเตอร์ (parameter) ของฟังก์ชันเป็นเหมือนตัวแปรเฉพาะที่ซึ่งต้องมีค่าส่งเข้ามาหรือนำค่าส่งออกไป ดังนั้นจึงต้องเก็บในสแตคในลักษณะเดียวกัน



ในการเรียกฟังก์ชันแบบเวียนเกิด (recursive function) ซ้ำ ตัวแปรเฉพาะที่ของฟังก์ชันนั้นต้องถูกสร้างขึ้นใหม่ทุกครั้งด้วย ตัวอย่าง เช่น ฟังก์ชัน `factorial` ข้างล่างนี้มีตัวแปรเฉพาะที่ `f`

```
int factorial(int n) {
   int f;
   if (n==0 or n==1) f=n;
   else f=n*factorial(n-1);
   return f;
}
```

หากโปรแกรมหลักเรียกฟังก์ชัน `factorial(4)` ตัวแปร `f` และพารามิเตอร์ `n` ของฟังก์ชัน `factorial` ต้องถูกเก็บไว้สำหรับการเรียกแบบเวียนเกิดทุกครั้งดังแสดงในรูปข้างล่างนี้



จากตัวอย่างที่กล่าวมานี้จะเห็นว่าเมื่อตัวแปลภาษาอ้างถึงตัวแปรเฉพาะที่ในการเรียกฟังก์ชันแต่ละครั้ง เลขที่อยู่ของตัวแปรเฉพาะที่นั้นจะต่างกัน เพราะอ้างถึงเนื้อที่ในสแตคที่สร้างขึ้นในการเรียกฟังก์ชันแต่ละครั้ง

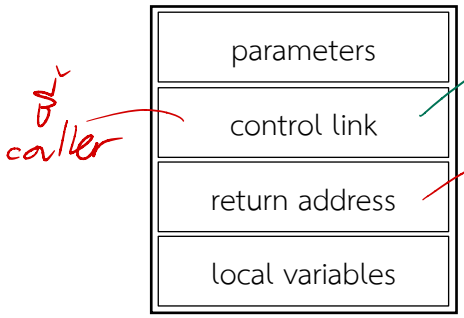
6.2.3 การจัดเก็บตัวแปรในฮีป

เมื่อโปรแกรมขอเนื้อที่ในฮีปทั้งแบบชัดแจ้งและโดยปริยาย ตัวแปลภาษาต้องตรวจสอบว่ามีเนื้อที่ว่างในฮีปหรือไม่และจัดเนื้อที่ให้หากมี ตัวแปลภาษาจะหาหน่วยความจำในส่วนฮีปที่ว่างและใช้เลขที่อยู่นี้สำหรับตัวแปรนั้น ดังนั้นตัวแปรนี้จะมีเลขที่อยู่จริงเมื่อโปรแกรมทำงานไปแล้ว

6.3 คำสั่งที่ใช้จัดการสแตคการเรียกและเลขที่อยู่ของตัวแปรเฉพาะที่

ในการเรียกใช้ฟังก์ชัน ตัวแปลภาษาต้องเตรียมเนื้อที่สำหรับตัวแปรเฉพาะที่และพารามิเตอร์ของฟังก์ชันดังที่กล่าวไปแล้ว นอกจากนั้นตัวแปลภาษายังต้องจัดให้เก็บเลขที่อยู่หลังคำสั่งเรียกใช้ฟังก์ชันที่เรียกว่า

เลขที่อยู่กลับ (return address) เพื่อให้สามารถกระโดดกลับมาทำงานที่คำสั่งนั้นเมื่อฟังก์ชันนั้นทำงานจบ ตัวแปลภาษาจะเก็บตัวชี้ที่เรียกว่า ลิงค์ควบคุม (control link) ที่ชี้ไปยังระเบียนการทำงานของฟังก์ชันที่เป็นตัวเรียก (caller) ด้วย ข้อมูลเกี่ยวกับฟังก์ชันที่กล่าวมานี้รวมเรียกว่า ระเบียนการทำงาน (activation record) หรือ เฟรม (frame) ดังแสดงในรูปข้างล่าง



ระเบียนการทำงาน (activation record)

6.3.1 คำสั่งจัดการเรียกฟังก์ชันและกลับจากฟังก์ชัน

เมื่อโปรแกรมเรียกใช้ฟังก์ชัน จะต้องมีการเก็บระเบียนการทำงานในสแตค เรียกว่า สแตคการเรียก (call stack) ตัวแปลภาษาสร้างคำสั่งที่เก็บระเบียนการทำงานในสแตคการเรียกโดยใช้ตัวชี้ของเฟรม FP (frame pointer) เพื่อบอกตำแหน่งของระเบียนการทำงานและตัวชี้ของสแตค SP (stack pointer) เพื่อบอกตำแหน่งบนสุดของสแตคการเรียก คำสั่งชุดนี้เรียกว่าลำดับการเรียก (calling sequence) ซึ่งประกอบด้วยการทำงานต่อไปนี้

1. จองที่ในสแตคสำหรับพารามิเตอร์ที่ส่งคืน (return parameter) และนำค่าของพารามิเตอร์ที่ส่งเข้ามาเก็บในสแตคการเรียก แล้วเลื่อน SP
2. เก็บลิงค์ควบคุมซึ่งได้จาก FP ที่ชี้ไปยังระเบียนการทำงานของฟังก์ชันเดิมลงในสแตค แล้วเลื่อน SP
3. เลื่อน FP มาชี้ที่เดียวกับ SP
4. เก็บเลขที่อยู่กลับลงในสแตคการเรียก แล้วเลื่อน SP
5. จองที่ในสแตคสำหรับตัวแปรเฉพาะที่ แล้วเลื่อน SP
6. กระโดดไปทำงานที่ฟังก์ชันที่ถูกเรียก

ตัวแปลภาษาต้องเพิ่มคำสั่งที่ทำงานตามลำดับการเรียกเมื่อมีการเรียกฟังก์ชัน ตัวอย่างต่อไปนี้แสดงคำสั่งที่จัดการลำดับการเรียกใน 3-address code

ตัวอย่าง พิจารณาลำดับการเรียกของ `factorial(x)` ที่แสดงในโปรแกรมด้วยตัวหนาในโปรแกรมภาษาซีต่อไปนี้

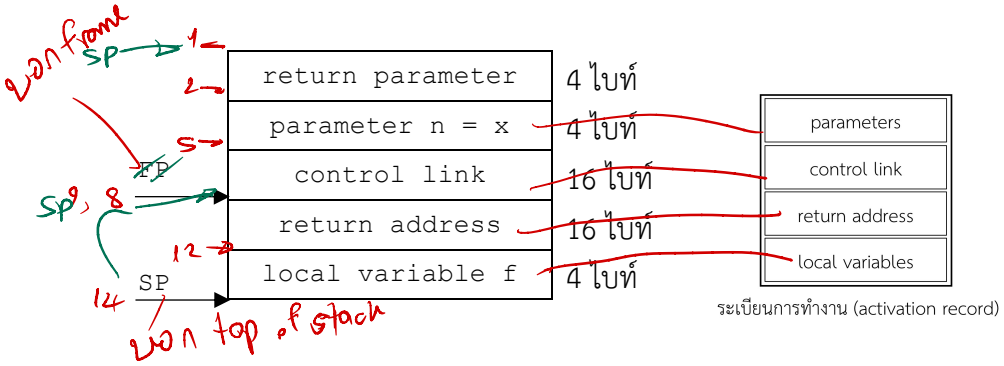
```
int factorial(int n){
    int f;
    if (n==0) f=1;
    else      f=factorial(n-1)*n;
    return f;
}
```

```
int main(void) {
    int x, y;
    x=3;
    …
    y=factorial(x);
}
```

ฟังก์ชัน `factorial` มี `n` เป็น input parameter และส่งคืน return parameter ที่กำหนดเป็น `int` ในหัวฟังก์ชัน ตัวแปลภาษาเพิ่มคำสั่งของ 3-address code เพื่อจัดการเรียกฟังก์ชัน `factorial(x)` ดังนี้

```
T   = SP+4        // make space for return parameter
SP  = T           // update SP
*SP= x            // push x as parameter n
T   = SP+4        // size of integer is 4
SP  = T           // update SP
*SP= FP           // push control link
T   = SP+16       // size of address is 16
SP  = T           // update SP
FP  = SP          // update FP
*SP= RTA          // push return address
T   = SP+16       // size of address is 16
SP  = T           // update SP
T   = SP+4        // make space for local variable f
SP  = T           // update SP
goto factorial    // jump to function
label RTA         // set return address
```

คำสั่งชุดนี้ทำให้ได้ระเบียนการทำงานข้างล่างนี้อยู่ในสแตคการเรียก



เมื่อการทำงานของฟังก์ชันจบลงและมีคำสั่ง return เพื่อให้ย้อนกลับมาทำงานต่อจากคำสั่งเรียกใช้ฟังก์ชันนั้น ตัวแปลภาษาต้องสร้างคำสั่งที่ทำให้ย้อนกลับมาทำงานต่อจากคำสั่งเรียกฟังก์ชัน พร้อมทั้งส่งพารามิเตอร์กลับ และคืนค่าตัวแปรเฉพาะที่ของฟังก์ชันนั้นกลับมา คำสั่งชุดนี้เรียกว่าลำดับการกลับ (return sequence) ซึ่งประกอบด้วยการทำงานต่อไปนี้

1. เก็บเลขที่อยู่กลับจากระเบียนการทำงานเพื่อกระโดดไปทำงานที่ฟังก์ชันที่เรียกตาม
2. เลื่อน SP ไปชี้ที่เดียวกับ FP (ป๊อปตัวแปรเฉพาะที่และเลขที่อยู่กลับทิ้ง)
3. เลื่อน FP ไปชี้ที่ระเบียนการทำงานของฟังก์ชันเดิม โดยนำลิงค์ควบคุมไปเก็บใน FP
4. เลื่อน SP ไปชี้ที่พารามิเตอร์ที่ส่งคืน
5. กระโดดไปทำงานคำสั่งที่เลขที่อยู่กลับ

เมื่อฟังก์ชันทำงานจบและกลับมาทำงานหลังคำสั่งเรียกฟังก์ชัน ตัวแปลภาษาต้องเพิ่มคำสั่งที่ทำงานตามลำดับการกลับ ตัวอย่างต่อไปนี้แสดงคำสั่งที่จัดการลำดับการกลับ

ตัวอย่าง พิจารณาลำดับการกลับของ `factorial(x)` ที่แสดงในโปรแกรมในตัวอย่างที่แล้ว ตัวแปลภาษาเพิ่มคำสั่งของ 3-address code เพื่อจัดการกลับจากฟังก์ชัน `factorial` ดังนี้

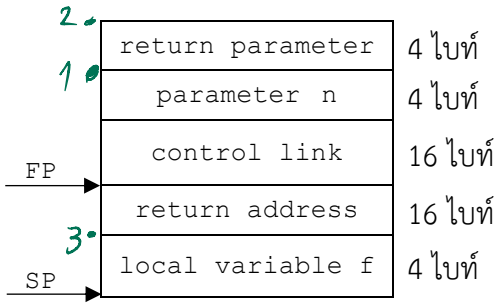
```
RET= *FP          // save return address
SP = FP           // pop local variable f and return address
T  = SP-16        // move SP down to control link
SP = T
FP = *SP          // move FP down to the caller's activation record
T  = SP-4         // pop input parameter
SP = T
T  = SP-4         // move SP down to return parameter
SP = T
goto RET          // jump back
```

6.3.2 การคำนวณเลขที่อยู่ของตัวแปรในฟังก์ชัน

สำหรับตัวแปรเฉพาะที่แต่ละตัว ตัวแปลภาษาจะเก็บตำแหน่งของตัวแปรในระเบียนการทำงานไว้ในตารางสัญลักษณ์เพื่อใช้หาเลขที่อยู่ของตัวแปรนั้น เมื่อต้องการอ้างถึงตัวแปรเฉพาะที่ในฟังก์ชันที่กำลังทำงานอยู่ ตัวแปลภาษาคำนวณเลขที่อยู่ของตัวแปรนั้นโดยนำค่าของตัวชี้ของระเบียนการทำงาน FP บวกกับตำแหน่งของตัวแปรเฉพาะที่จากตารางสัญลักษณ์ ตัวอย่างต่อไปนี้แสดงการคำนวณหาเลขที่อยู่ของตัวแปรเฉพาะที่ในฟังก์ชัน ตัวอย่างต่อไปนี้แสดงการคำนวณเลขที่อยู่ของตัวแปรในฟังก์ชัน

ตัวอย่าง พิจารณาฟังก์ชัน `factorial(x)` และระเบียนการทำงานของฟังก์ชัน `factorial` ข้างล่างนี้

```
int factorial(int n){
    int f;
    if (n==0) f=1;
    else      f=factorial(n-1)*n;
    return f;
}
```



เมื่อฟังก์ชัน `factorial` ทำงาน ระเบียนการทำงานของฟังก์ชัน `factorial` จะอยู่บนสุดของสแตค ดังนั้นเราสามารถคำนวณเลขที่อยู่ของค่าในระเบียนการทำงานได้ดังนี้

1 - เลขที่อยู่ของพารามิเตอร์ `n` คือ `FP-20`
2 - เลขที่อยู่ของพารามิเตอร์ที่ส่งกลับ คือ `FP-24`
3 - เลขที่อยู่ของตัวแปรเฉพาะที่ `f` คือ `FP+16`

ดังนั้น โปรแกรม 3-address code ของฟังก์ชัน factorial เป็นดังนี้

```
label factorial
fAddr = FP+16      // find the address of local variable f
nAddr = FP-16      // find the address of parameter n
c = *nAddr==0
ifFalse c goto Flabel    // if (n==0)
*fAddr = 1               // f=1
goto OUT
label Flabel             // else
p = *nAddr -1            // calculate n-1
// ---- call sequence
T  = SP+4
SP = T
*SP= p                   // set parameter as n-1
T  = SP+4
SP = T
*SP= FP                  // save control link
T  = SP+16
SP = T
FP = SP
*SP= RTA                 // save return address
T  = SP+16
SP = T
T  = SP+4
SP = T
// ---- call factorial
goto factorial
label RTA                // set return address
par = *SP                // get return parameter
N  = *nAddr
*fAddr = par*N           // f=factorial(n-1)*n
Label OUT
T  = FP-20               // return f; store f as return parameter
*T =*fAddr
// ---- return sequence
RET= *FP           // get return address
SP = FP            // pop local variables and return address
T  = SP-16
SP = T
FP = *SP           // move FP to caller's activation record
T  = SP-4
SP = T             // pop input parameter
T  = SP-4
SP = T             // move SP to return parameter
goto RET
```

จากตัวอย่างนี้จะเห็นได้ว่าการอ้างถึงตัวแปรเฉพาะที่ในฟังก์ชันมีขั้นตอนการคำนวนเลขที่อยู่มากกว่าการอ้างถึงตัวแปรแบบสถิต