บทที่ 7
ฟังก์ชันในภาษา C
(Function in C)

ฟังก์ชันในภาษา C จะเป็นการเขียนข้อความสั่งไว้เป็นกลุ่มที่อิสระ เพื่อที่จะทำงานอย่างใดอย่างหนึ่ง โดยกลุ่มของข้อความสั่งจะอยู่ภายในเครื่องหมาย { และ } โดยจะมีการกำหนดชื่อให้กับกลุ่มของข้อความสั่งเพื่อระบุในการใช้งาน

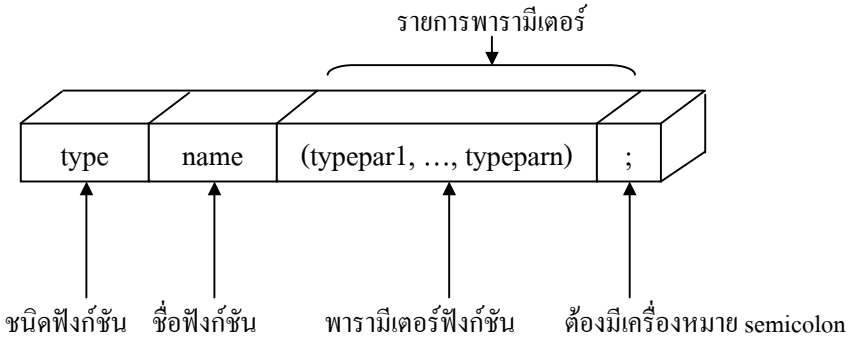
ในการเขียนโปรแกรมขนาดใหญ่ จะมีการแบ่งงานออกเป็นส่วน ๆ โดยแต่ละส่วนก็จะมอบหมายให้แต่ละคนไปทำ ส่วนย่อย ๆ ดังกล่าวก็คือฟังก์ชัน หรือในงานบางงานอาจจะมีการเรียกใช้บ่อย ๆ เราก็สามารถกำหนดงานส่วนนี้เป็นฟังก์ชัน

ทุก ๆ โปรแกรมในภาษา C จะต้องมีฟังก์ชันอย่างน้อย 1 ฟังก์ชันเสมอ คือ ฟังก์ชัน main ()

ฟังก์ชันในภาษา C จะถูกแบ่งออกเป็น 2 แบบ คือ

1. ฟังก์ชันที่กำหนดมาให้แล้วแต่อยู่ในไลบรารี เช่น ฟังก์ชัน printf (), scanf () เป็นต้น
2. ฟังก์ชันที่ถูกนิยามโดยผู้ใช้ เพื่อใช้ในการแก้ปัญหาบางสิ่งบางอย่าง

7.1 รูปแบบทั่วไปของฟังก์ชันต้นแบบ (General of a Function Prototype)



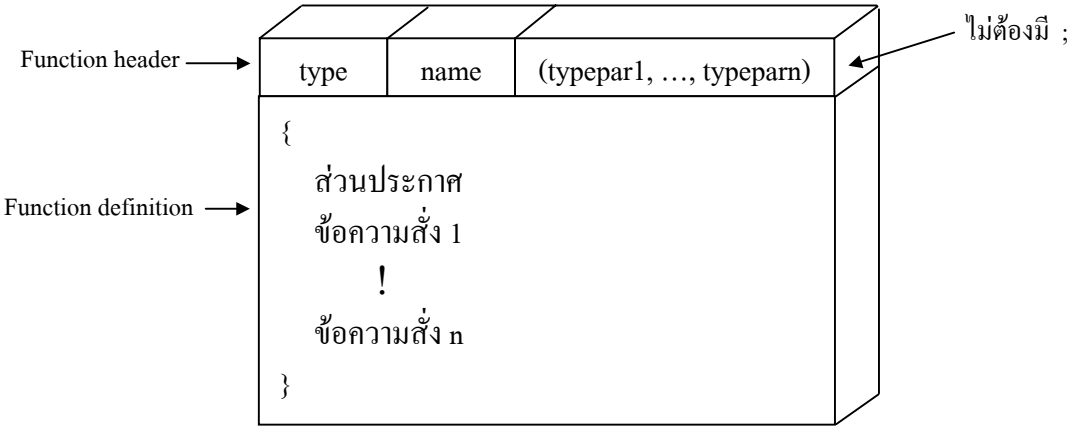
โดยที่ **type** คือ ชนิดของข้อมูลฟังก์ชันที่ส่งกลับคืนมา เช่น int, char เป็นต้น แต่ถ้าไม่มีการส่งค่ากลับคืนมา เราจะระบุโดยใช้คำสงวน void นำหน้า และถ้าไม่มีการระบุชนิดข้อมูล จะมี default เป็น int

name คือ ชื่อของฟังก์ชันที่ตั้งขึ้นมาโดยอาศัยกฎเกณฑ์การตั้งชื่อ

typepar1, …, typeparn คือ รายการของพารามิเตอร์ พร้อมระบุชนิดข้อมูล ถ้ามีพารามิเตอร์มากกว่า 1 ตัว ให้เขียนแยกกันโดยใช้เครื่องหมายคอมม่า (,) และปิดล้อมด้วยเครื่องหมาย (และ) และตามด้วยเครื่องหมาย (;) ถ้าไม่มีพารามิเตอร์เลยให้ระบุเป็น (void) ; หรือ ( ) ; ก็ได้

แต่ละนิยามของฟังก์ชันต้นแบบจะถูกเขียนไว้เหนือฟังก์ชัน main และมีเครื่องหมาย (;) ปิดท้าย ซึ่งโดยปกติแล้วทุก ๆ ฟังก์ชันในภาษา C ยกเว้นฟังก์ชัน main จะต้องมีการประกาศก่อน เพื่อบอกให้คอมไพเลอร์รู้ว่าในโปรแกรมนี้จะมีการเรียกใช้ฟังก์ชันนี้และจองเนื้อที่ในหน่วยความจำหลักให้กับฟังก์ชันนี้ด้วย

7.2 รูปแบบของการนิยามฟังก์ชัน



โดย Function header คือส่วนหัวของฟังก์ชัน จะประกอบด้วย

type คือ ชนิดของข้อมูลฟังก์ชันที่ส่งค่ากลับคืน

name คือ ชื่อของฟังก์ชันที่เหมือนกับฟังก์ชันต้นแบบ

typepar1, …, typeparn คือ รายการของ formal parameter ที่เหมือนกับฟังก์ชันต้นแบบ

และ **Function definition** คือ นิยามของฟังก์ชันซึ่งจะประกอบไปด้วยส่วนประกาศและข้อความสั่งต่าง ๆ

7.3 การเรียกใช้ฟังก์ชัน

ฟังก์ชันที่ถูกนิยามไว้ในโปรแกรม สามารถถูกเรียกใช้โดยการระบุชื่อของฟังก์ชันตามด้วยรายการของพารามิเตอร์ โดยพารามิเตอร์ส่วนนี้จะถูกเรียกว่า actual parameter หรือบางครั้งก็จะเรียกว่า อาร์กิวเมนต์ (argument) เนื่องจากว่ามันเป็นค่าจริง ๆ ที่ถูกส่งออกไป พารามิเตอร์จะต้องถูกปิดด้วยเครื่องหมาย (และ) ถ้ามีอาร์กิวเมนต์มากกว่า 1 ตัว จะต้องคั่นด้วยเครื่องหมาย , และถ้าไม่มีอาร์กิวเมนต์เลย อาจจะระบุโดยใช้คำสงวน void หรือเว้นว่างไว้ก็ได้ ถ้ามีการเรียกใช้ฟังก์ชันที่ต้องการ โปรแกรมก็จะส่งการควบคุมการทำงานไปกระทำการที่ฟังก์ชันถูกเรียก เมื่อสิ้นสุดก็จะส่งการควบคุมกลับมาที่ฟังก์ชันเรียกใช้

7.4 ข้อความสั่ง return

รูปแบบ return (expression) ;

โดย return เป็นคำสงวน

และ expression เป็นนิพจน์ที่จะส่งค่ากลับคืน และจะต้องมีชนิดข้อมูลสอดคล้องกับฟังก์ชันที่ถูกส่ง

ถ้าไม่มีการส่งค่ากลับคืน เราจะสามารถเขียนได้เป็น

return ;

return จะเป็นข้อความสั่งที่ให้หยุดกระทำการ (execute) ของฟังก์ชัน และส่งการควบคุมกลับไปยังฟังก์ชันที่เรียกได้

หมายเหตุ return จะส่งค่ากลับได้เพียง 1 ค่าเท่านั้น

ถ้าข้อมูลของฟังก์ชันที่ถูกส่งกลับไม่ได้ระบุไว้ จะมี default เป็น int

ถ้าฟังก์ชันที่เรียกใช้ ไม่ต้องการค่ากลับคืนมา ให้ระบุคำสงวน void นำหน้าชื่อฟังก์ชัน และในส่วนข้อความสั่ง return ไม่ต้องมีนิพจน์ โดยเขียนเป็นแบบง่าย ๆ คือ return ;

แต่ถ้าในฟังก์ชันถูกเรียกไม่มีการระบุข้อความสั่ง return จะหมายความว่า เมื่อสิ้นสุดการกระทำการ ที่ฟังก์ชันถูกเรียกโดยพบเครื่องหมาย } ซึ่งจะเป็นการบอกว่าสิ้นสุดฟังก์ชันก็จะส่งการควบคุมกลับไปยังฟังก์ชันที่เรียกใช้ และไม่มีการส่งค่ากลับไปให้ฟังก์ชันเรียกใช้

ใน 1 ฟังก์ชัน อาจจะมีหลายข้อความสั่ง return ก็ได้

เช่น

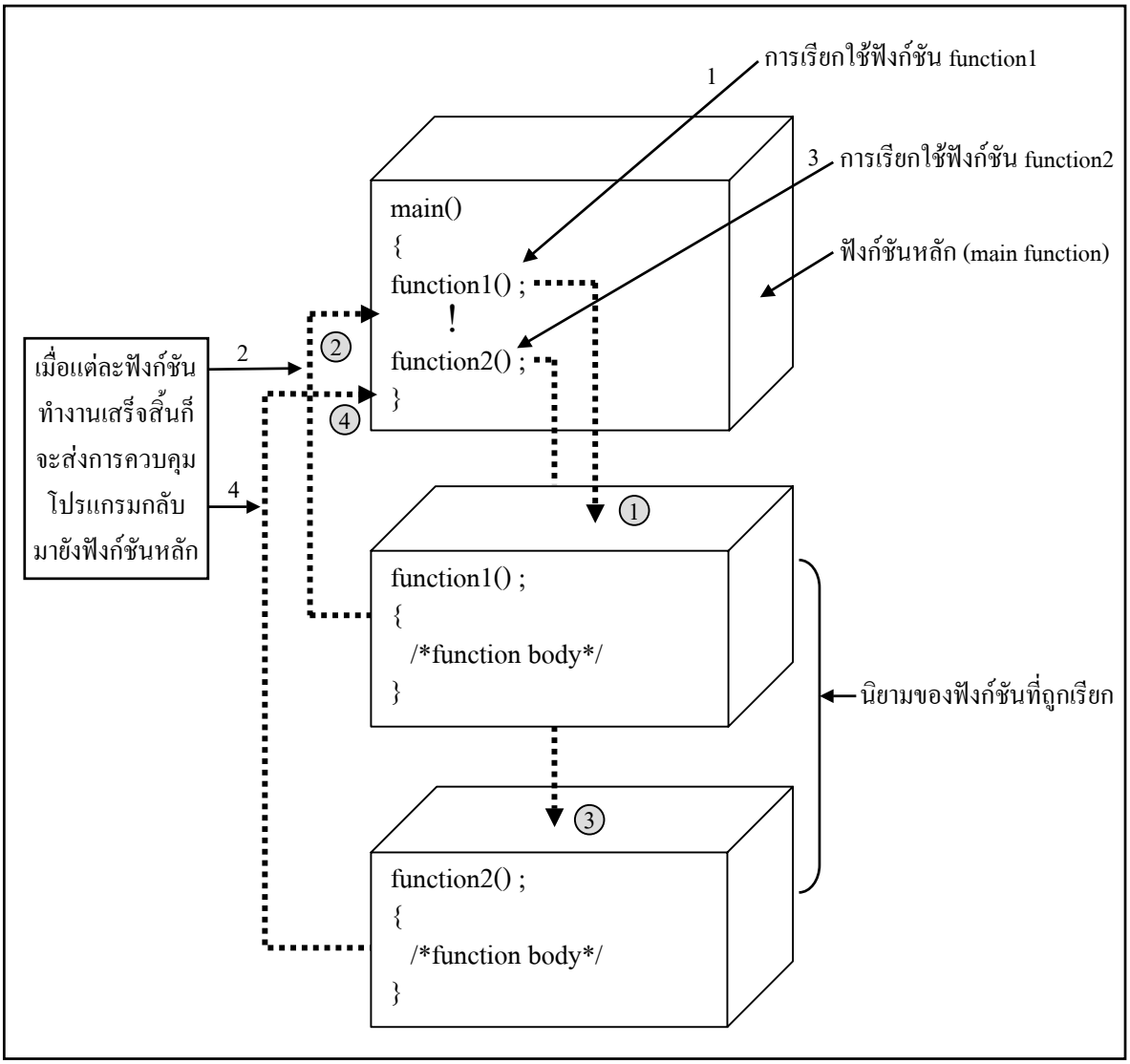
```
int cal (int x, int y)
{
    if (x > 2)
        return (2 * x) ;
    else
        return (2 * x – 1) ;
}
```

แต่ถ้าจะให้ดี ควรจะแก้ไขโปรแกรมนี้ใหม่เป็น

```
int cal (int x, int y)
{
    int   y = 0 ;
        if (x > 2)
            y = 2 * x ;
        else
            y = 2* x – 1 ;
    return (y) ;
}
```

เพราะว่า ต้องการให้มีการส่งค่ากลับคืนออกมาเพียงจุดเดียว แทนที่จะมีการส่งค่ากลับคืนมาหลายจุด เพราะว่าอาจจะทำให้การตรวจสอบโปรแกรมยุ่งยากและเสียเวลา ถ้าในกรณีโปรแกรมมีขนาดใหญ่

แผนภาพแสดงขั้นตอนการเรียกใช้ฟังก์ชัน ดังรูปที่ 7.1

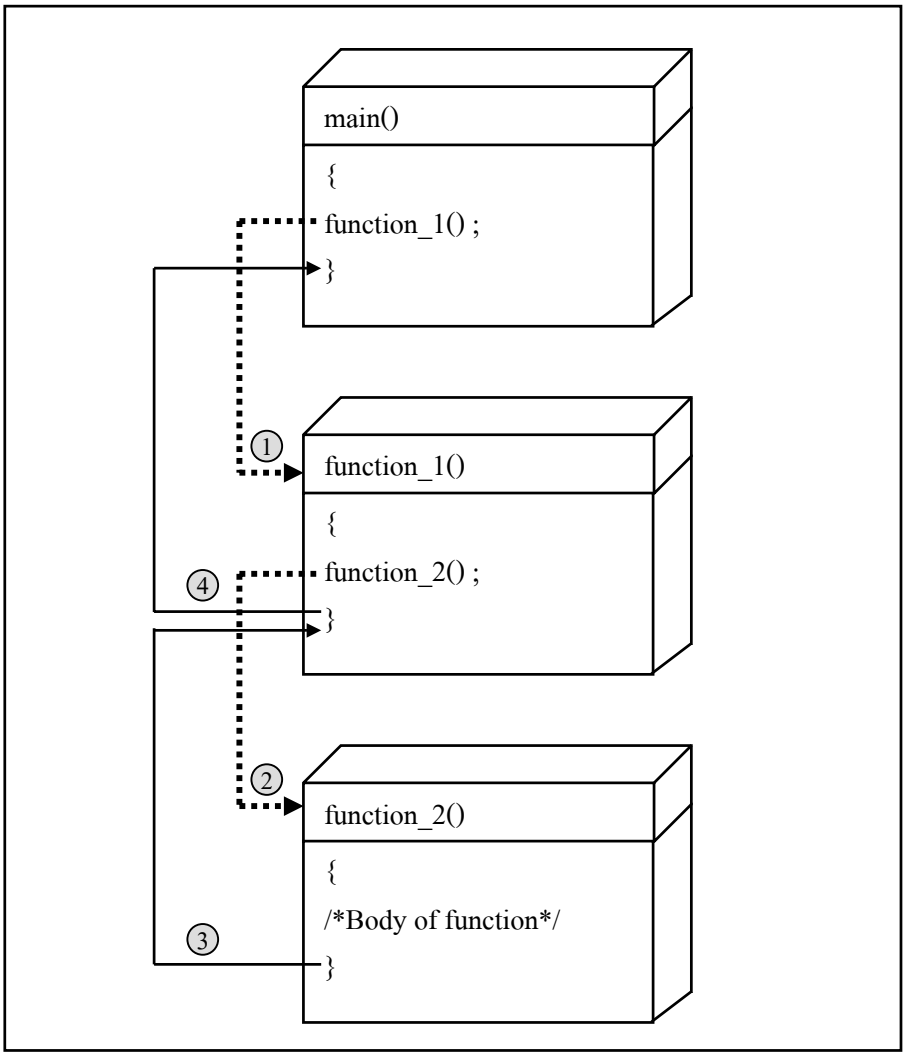


รูปที่ 7.1

จากรูปที่ 7.1 ขั้นตอนการเรียกใช้ฟังก์ชันมีลำดับขั้นตอนดังนี้

เมื่อกระทำการมาพบการเรียกใช้ฟังก์ชัน function1() ก็จะส่งการควบคุมโปรแกรมไปยังฟังก์ชันที่ถูกเรียก คือ ฟังก์ชัน function1() (ตามลูกศรเส้นประ หมายเลข 1) เมื่อทำงานที่ฟังก์ชันถูกเรียก function1() สิ้นสุดก็จะส่งการควบคุมโปรแกรมกลับมายังฟังก์ชันที่เรียกใช้ในข้อความสั่งถัดไป ในฟังก์ชัน main() (ตามลูกศรเส้นประ หมายเลข 2) และกระทำการต่อไปเรื่อย ๆ จนมาพบการเรียกใช้ฟังก์ชัน function2() ก็จะส่งการควบคุมโปรแกรมไปยังฟังก์ชันที่ถูกเรียก คือ

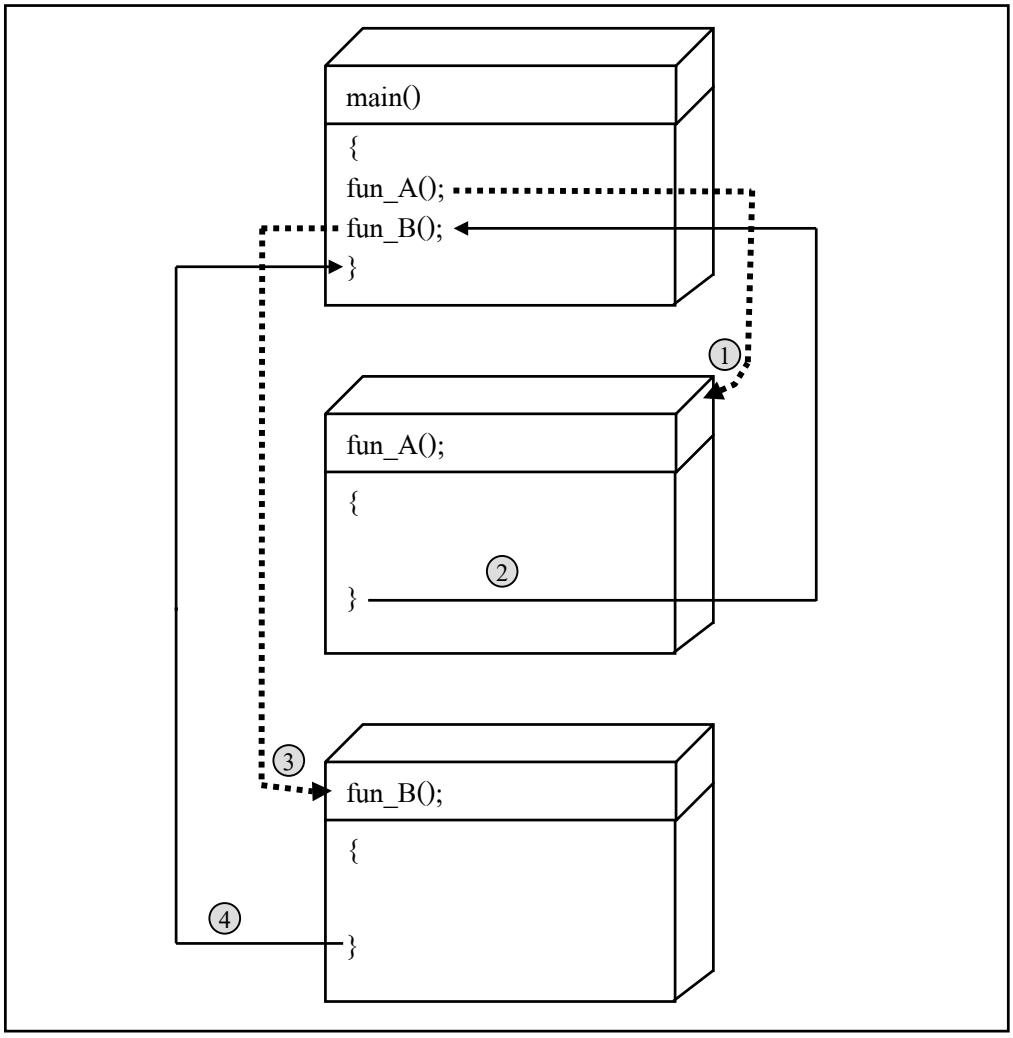
function2() (ตามลูกศรเส้นประ หมายเลข 3) เมื่อทำงานที่ฟังก์ชันถูกเรียก function2() สิ้นสุดก็จะส่งการควบคุมโปรแกรมกลับมายังฟังก์ชันที่เรียกใช้ในข้อความสั่ง ถัดไปในฟังก์ชัน main() (ตามลูกศรเส้นประ หมายเลข 4) แต่เป็นเครื่องหมาย } แสดงว่าสิ้นสุดการกระทำการในฟังก์ชัน main()



รูปที่ 7.2 แสดงการเรียกใช้ฟังก์ชันภายในฟังก์ชันที่ถูกเรียก

จากรูปที่ 7.2 จะพบว่า ภายในฟังก์ชัน main() มีการเรียกใช้ฟังก์ชัน function_1() ก็จะส่งการควบคุมโปรแกรมไปยังฟังก์ชันที่ถูกเรียก function_1() (ตามลูกศรหมายเลข 1) ภายในฟังก์ชันนี้มีการเรียกใช้ function_2() ก็จะส่งการควบคุมโปรแกรมไปยังฟังก์ชันที่ถูกเรียก function_2() (ตามลูกศรหมายเลข 2) เมื่อสิ้นสุดการกระทำการฟังก์ชัน function_2() ก็จะส่งการควบคุมโปรแกรมกลับไปยังในข้อความสั่งถัดไปในฟังก์ชันเรียกใช้ function_2() (ตามลูกศรหมายเลข 3) แต่ข้อความสั่ง

ถัดไปเป็นเครื่องหมาย } แสดงว่า สิ้นสุดการกระทำการฟังก์ชัน function_1() ก็จะส่งการควบคุมโปรแกรมกลับไปยังในข้อความสั่งถัดไปในฟังก์ชันเรียกใช้ main() (ตามลูกศรหมายเลข 4) แต่ข้อความสั่งถัดไปเป็นเครื่องหมาย } แสดงว่าสิ้นสุดโปรแกรม



รูปที่ 7.3 แสดงการเรียกใช้ฟังก์ชัน

จากรูปที่ 7.3 ในฟังก์ชัน main() มีการเรียกใช้ 2 ฟังก์ชัน คือ fun_A() และ fun_B() โดยมีลำดับขั้นตอนดังนี้ คือ

เรียกใช้ฟังก์ชัน fun_A() ก็จะส่งการควบคุมโปรแกรมไปยังฟังก์ชันที่ถูกเรียก fun_A() (ตามลูกศรหมายเลข 1) เมื่อกระทำการที่ฟังก์ชันนี้เรียบร้อยก็จะส่งการควบคุมโปรแกรมกลับไปยังข้อความสั่งถัดไปในฟังก์ชัน main() (ตามลูกศรหมายเลข 2) แต่ข้อความสั่งถัดไปเป็นการเรียกใช้ฟังก์ชัน fun_B() ก็จะส่งการควบคุมโปรแกรมไปยังฟังก์ชัน ถูกเรียก fun_B() (ตามลูกศรหมายเลข 3)

เมื่อกระทำการที่ฟังก์ชันนี้เรียบร้อยก็จะส่งการควบคุมโปรแกรมกลับไปยังข้อความสั่งถัดไปในฟังก์ชัน main() (ตามลูกศรหมายเลข 4) แต่ข้อความสั่งถัดไปเป็นเครื่องหมาย } แสดงว่าสิ้นสุดฟังก์ชัน main()

ตัวอย่างโปรแกรมที่ 7.1 เป็นโปรแกรมแสดงการคำนวณหาค่ากำลังสองของจำนวนที่ป้อนเข้าไปทางแป้นพิมพ์

```
#include <stdio.h>

/* Function prototype. */

float square_it (float number) ;        <-- ฟังก์ชันต้นแบบ
/* This is the function that will square the number. */

main ()
{
    float value ;       /* Number to be squared.       */
    float answer ;      /* The square of the number.   */

    printf ( "Give me a number and I'll square it = >" ) ;
    scanf ( "%f", &value ) ;        ฟังก์ชันที่เรียกใช้
                                    actual parameter
    answer = square_it (value) ;        /* Call the function. */

    printf ( "The square of %f is %f", value, answer ) ;
ชื่อต้องเหมือนกัน
    exit (0) ;              (1)
}                                   formal parameter

float square_it (float number)
{
    float answer ;      (2) (2)     /* The square of the number. */

    answer = number * number ;
                (3)
(4) return (answer) ;
}
```

นิยามของฟังก์ชัน

```
Give me a number and I'll square it = > 3
The square of 3.000000 is 9.000000
```

การทำงานของโปรแกรมนี้สามารถอธิบายได้ดังนี้

ในโปรแกรมมีการประกาศฟังก์ชันต้นแบบ คือ float square_it (float number) ; เพื่อบอกให้คอมไพเลอร์รู้ว่า ในโปรแกรมนี้จะมีการเรียกใช้ฟังก์ชันต้นแบบ และให้เตรียมจองเนื้อที่ในหน่วยความจำ พร้อมทั้งบอกให้รู้ว่าฟังก์ชันนี้มีพารามิเตอร์อะไรบ้าง (ท้ายฟังก์ชันต้นแบบจะมีเครื่องหมาย ; ปิดท้าย) โดยฟังก์ชันต้นแบบจะมี formal parameter คือ number มีชนิดข้อมูลเป็นแบบ float และหน้าฟังก์ชันต้นแบบมีชนิดข้อมูลเป็นแบบ float หมายความว่า ค่าที่ส่งคืนกลับมาจะมีชนิดข้อมูลเป็นแบบ float

ในฟังก์ชัน main () จะมีการเรียกใช้ฟังก์ชัน square_it (value) ; โดยตัวแปร value เราจะเรียกว่า actual parameter เนื่องจากเป็นค่าจริง ๆ ที่ผ่านค่าไปยังฟังก์ชันที่ถูกเรียก เราจะเรียกการผ่านค่าแบบนี้ว่า การผ่านโดยระบุค่า (pass by value) ซึ่งจะมีสำเนาค่าของตัวแปร value ไปยังตัวแปร number ซึ่งเราจะเรียกตัวแปรนี้ว่า formal parameter ชื่อตัวแปรของ actual parameter กับ formal parameter จะเหมือนกันหรือไม่เหมือนกันก็ได้ แต่ต้องมีชนิดข้อมูลเหมือนกัน เมื่อในฟังก์ชัน main () มีการเรียกใช้ฟังก์ชัน square_it ก็จะส่งการควบคุมโปรแกรมไปยังฟังก์ชันที่ถูกเรียกที่มีนิยามของฟังก์ชัน ดังนี้

```
float square_it (float number)
{                                  เป็นตัวแปรเฉพาะที่ (local variable)
    float answer ;   /* The square of the number.    */
    answer = number * number ;
    return (answer) ;
}
```

ให้สังเกตที่ท้ายฟังก์ชันถูกเรียกจะไม่มีเครื่องหมาย semicolon (;) ปิดท้าย เนื่องจากต้องการจะบอกให้คอมไพเลอร์รู้ว่า ส่วนที่ตามมาคือนิยามของฟังก์ชันที่ถูกเรียก ในส่วนของฟังก์ชันที่ถูกเรียกนี้จะมีตัวแปรเฉพาะที่ 2 ตัว คือ answer และ number หมายความว่า เมื่อฟังก์ชันนี้ถูกกระทำการ ตัวแปร 2 ตัวนี้ จะถูกกำหนดเนื้อที่ในหน่วยความจำที่เรียกว่า สแตก (stack) มาให้ เมื่อมีการสำเนาข้อมูลจากตัวแปร value มาให้ number แล้ว ก็จะนำค่านี้ไปคำนวณหาค่าในข้อความสั่ง คือ answer = number * number ; โดยตัวแปรเฉพาะที่ answer จะเก็บผลลัพธ์ที่ได้จากการคำนวณ และข้อความสั่งถัดไปคือ return (answer) ; จะเป็นการส่งค่าของ answer กลับไป

ยังฟังก์ชันที่เรียกใช้ เมื่อสิ้นสุดฟังก์ชันนี้ ตัวแปรเฉพาะที่ 2 ตัวคือ number และ answer ก็จะถูกทำลาย

ในฟังก์ชัน main () ก็จะมีตัวแปรเฉพาะที่ 2 ตัว คือ answer กับ value เราจะพบว่าตัวแปร answer ในฟังก์ชัน main () กับฟังก์ชันที่ถูกเรียกใช้คือ square_it ถึงแม้จะมีชื่อเหมือนกัน แต่ขอให้เข้าใจว่าตัวแปร 2 ตัวนี้ เป็นคนละตัวกัน เปรียบเสมือนกับมีคน 2 คน แต่มีชื่อเหมือนกัน และในโปรแกรมนี้มีข้อความสั่งว่า exit (0) ; หมายความว่าให้ยุติโปรแกรม

สรุป โปรแกรมนี้มี

1. ตัวแปร number เป็น formal parameter
2. ตัวแปร value เป็น actual parameter
3. ฟังก์ชันต้นแบบ คือ square_it (float number)
4. ฟังก์ชันที่เรียกใช้ คือ square_it (value)

เมื่อมีการเรียกฟังก์ชันมาใช้ ก็จะส่งการควบคุมโปรแกรมไปยังฟังก์ชันที่ถูกเรียกใช้ โดยอาจจะมีการส่งข้อมูลไปให้ฟังก์ชันที่เรียกใช้เมื่องานเสร็จสิ้นแล้ว ฟังก์ชันที่ถูกเรียกก็อาจจะส่งข้อมูลที่ได้กลับมายังฟังก์ชันที่เรียกใช้

ในภาษา C มีช่องทางในการส่งค่าไปยังฟังก์ชันที่ถูกเรียกและส่งค่ากลับมายังฟังก์ชันที่เรียกใช้ 3 แบบ คือ

1. ส่งค่าผ่านตัวแปรส่วนกลาง จะใช้ในกรณีที่มีการใช้ตัวแปรร่วมกัน
2. ส่งค่าผ่านทางชื่อฟังก์ชัน
3. ส่งค่าผ่านทางพารามิเตอร์ (parameter)

หมายเหตุ 1) ควรหลีกเลี่ยงการผ่านค่าทางตัวแปรส่วนกลางไปยังฟังก์ชันถูกเรียก แต่ควรจะใช้พารามิเตอร์ส่งค่าไปยังฟังก์ชันถูกเรียก และส่งค่ากลับมายังชื่อของฟังก์ชันดีกว่า

2) เนื่องจากทุก ๆ ฟังก์ชันสามารถเรียกใช้ตัวแปรส่วนกลางได้ทุกฟังก์ชัน ดังนั้น ค่าที่เราคาดหวังไว้อาจจะมีการเปลี่ยนแปลงและผิดไป ซึ่งทำให้เราต้องเสียเวลาตรวจสอบว่าที่ฟังก์ชันใดทำให้เกิดผลที่ผิดไป ถ้าเป็นไปได้ควรจะพยายามใช้ตัวแปรเฉพาะที่ให้มากที่สุด

7.5 การส่งค่าระหว่างฟังก์ชันที่เรียกใช้ กับฟังก์ชันที่ถูกเรียก

การส่งค่าระหว่าง actual parameter กับ formal parameter ในภาษา C จะแบ่งออกเป็น 2 แบบ คือ

7.5.1 การผ่านโดยระบุค่า (pass by value)

7.5.2 การผ่านโดยการอ้างอิง (pass by reference)

7.5.1 การผ่านโดยระบุค่า จะเป็นการสำเนาข้อมูลจาก actual parameter ไปยัง formal parameter โดยค่าของ formal parameter ที่มีการเปลี่ยนแปลงภายในฟังก์ชันที่ถูกเรียกใช้ จะไม่มีผลกระทบต่อข้อมูลของ actual parameter ในบางครั้งเราจะเรียกว่าเป็นการส่งข้อมูลแบบทางเดียว

7.5.2 การผ่านโดยการอ้างอิง จะเป็นการส่งค่าที่อยู่ของ actual parameter ไปยัง formal parameter นั่นคือ ทั้ง actual parameter กับ formal parameter จะมีที่อยู่ของข้อมูลอยู่ที่ตำแหน่งเดียวกัน ดังนั้นถ้าข้อมูลของ formal parameter มีการเปลี่ยนแปลงค่าภายในฟังก์ชันที่ถูกเรียกใช้ก็จะมีผลกระทบต่อข้อมูลของ actual parameter ในบางครั้งเราจะเรียกว่า เป็นการส่งข้อมูลแบบสองทาง ดังนั้น ถ้าเราต้องการส่งข้อมูลกลับมากกว่า 1 ค่า เราจะใช้วิธีการผ่านโดยการอ้างอิง

กฎเกณฑ์ในการส่ง actual parameter ไปยัง formal parameter คือ

1. จำนวน actual parameter กับจำนวน formal parameter ต้องเท่ากัน
2. การส่งค่าระหว่างพารามิเตอร์ทั้งสอง จะกระทำแบบหนึ่งต่อหนึ่ง กล่าวคือ

   actual parameter ตัวที่ 1 จะจับคู่กับ formal parameter ตัวที่ 1

   actual parameter ตัวที่ 2 จะจับคู่กับ formal parameter ตัวที่ 2

   ! ! !

   actual parameter ตัวที่ n จะจับคู่กับ formal parameter ตัวที่ n

3. ชนิดของข้อมูลระหว่าง actual parameter กับ formal parameter ในแต่ละคู่ต้องเหมือนกัน
4. ชื่อของ actual parameter กับ formal parameter ในแต่ละคู่จะเหมือนหรือต่างกันก็ได้

กฎเกณฑ์ในการส่งพารามิเตอร์โดยวิธีการผ่านโดยระบุค่า

1. actual parameter กับ formal parameter ในแต่ละคู่ จะต้องมีชนิดข้อมูลเหมือนกัน
2. formal parameter แต่ละตัว ห้ามเป็นตัวชี้ (pointer)
3. actual parameter อาจจะเป็นค่าคงตัว, ตัวแปร หรือ นิพจน์
4. แต่ละครั้งที่การควบคุมโปรแกรมถูกส่งไปให้ฟังก์ชันที่ถูกเรียกใช้ จะมีการสำเนาค่าของแต่ละ actual parameter ไปยัง formal parameter ที่สมนัยกัน การเปลี่ยนแปลงค่าใด ๆ ที่ถูกสำเนามาในฟังก์ชันที่ถูกเรียกใช้ จะไม่มีผลกระทบต่อ actual parameter ที่สมนัยกัน ดังนั้นการผ่านค่าแบบนี้จะใช้เฉพาะส่งค่าไปยังฟังก์ชันที่ถูกเรียกเท่านั้น

กฎเกณฑ์ในการส่งพารามิเตอร์โดยวิธีการผ่านโดยการอ้างอิง

1. actual parameter กับ formal parameter ในแต่ละคู่จะต้องมีชนิดข้อมูลเหมือนกัน
2. actual parameter จะต้องเป็นที่อยู่ของตัวแปร โดยจะมีเครื่องหมาย ampersand (&) เติมหน้า
3. formal parameter แต่ละตัว ต้องมีเครื่องหมาย * เติมหน้า
4. ในส่วนของฟังก์ชันต้นแบบ formal parameter แต่ละตัวจะต้องมีเครื่องหมาย * เติมหน้า
5. การเข้าถึงข้อมูลในตำแหน่งหน่วยความจำของ actual parameter เราจะต้องใช้ชื่อที่สมนัยกับ formal parameter เติมหน้าด้วยเครื่องหมาย * โดยเราจะเรียกเครื่องหมาย * นี้ว่าตัวดำเนินการเข้าถึงโดยอ้อม (indirection operator)

7.6 ขอบเขตของตัวแปร (scope of variable)

ในโปรแกรมภาษา C ก่อนที่เราจะใช้ตัวแปรในโปรแกรมได้ เราจะต้องมีการประกาศตัวแปรเหล่านั้นก่อน เพื่อกำหนดบริเวณที่แน่นอนในโปรแกรม บริเวณดังกล่าวเราจะเรียกว่าขอบเขตของตัวแปร

ขอบเขตของตัวแปรใด ๆ จะเริ่มต้นที่จุดประกาศตัวแปร และจุดสิ้นสุดของตัวแปรจะอยู่ที่ใดนั้นขึ้นอยู่ว่า ตัวแปรดังกล่าวถูกประกาศไว้ที่ใด และประกาศอย่างไร ในโครงสร้างแบบ block ของภาษา C จุดเริ่มต้นของ block เริ่มที่เครื่องหมาย { และจุดสิ้นสุดของ block จะอยู่ที่เครื่องหมาย }

ขอบเขตของตัวแปรส่วนกลาง จะมีจุดเริ่มต้นที่ตัวแปรถูกประกาศ และสิ้นสุดเมื่อจบโปรแกรม ดังนั้นเราจะเรียกตัวแปรส่วนกลางนี้ว่า ขอบเขตของแฟ้ม (file scope)

ขอบเขตของตัวแปรเฉพาะที่ จะมีจุดเริ่มต้นที่ตัวแปรถูกประกาศ และมีจุดสิ้นสุดที่ block ของฟังก์ชันที่ตัวแปรนี้ประกาศอยู่ ดังนั้นเราจะเรียกตัวแปรเฉพาะที่นี้ว่า ขอบเขตของ block (block scope) หรือ ขอบเขตเฉพาะที่ (local scope)

formal parameter ก็เป็นตัวแปรเฉพาะที่เช่นกัน เพียงแต่ว่าเป็นตัวแปรที่ถูกประกาศไว้ในรายการของฟังก์ชันที่ถูกเรียกใช้ ดังนั้นจุดเริ่มต้นของตัวแปรเฉพาะที่จะเริ่มต้นที่รายการนี้ และมีจุดสิ้นสุดที่ block ของฟังก์ชัน ดังนั้นเราจะเรียกตัวแปรเฉพาะที่ของ formal parameter นี้ว่า ขอบเขต parameter (parameter scope)

7.7 ความมองเห็นได้ (visibility)

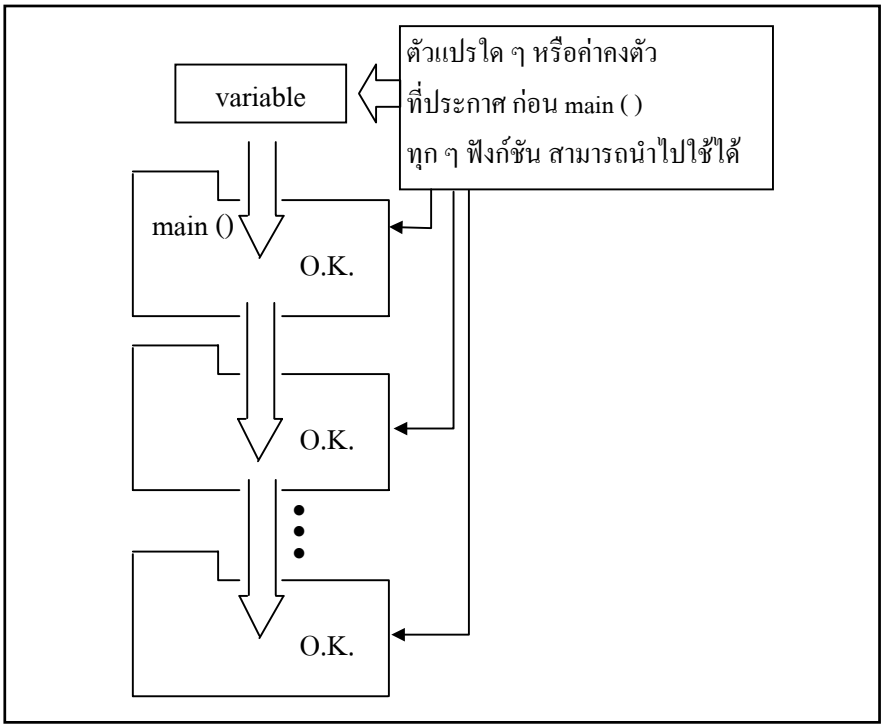
ความมองเห็นได้ของตัวแปร คือ ความสามารถของโปรแกรมในการเข้าถึงตำแหน่งหน่วยความจำของตัวแปร

7.8 ช่วงชีวิตของตัวแปร (lifetime)

ช่วงชีวิตของตัวแปร เป็นเวลาระหว่างที่ตำแหน่งหน่วยความจำของตัวแปรยังอยู่ในช่วงเวลาดำเนินการ (run time)

7.9 ตัวแปรส่วนกลาง (global variable)

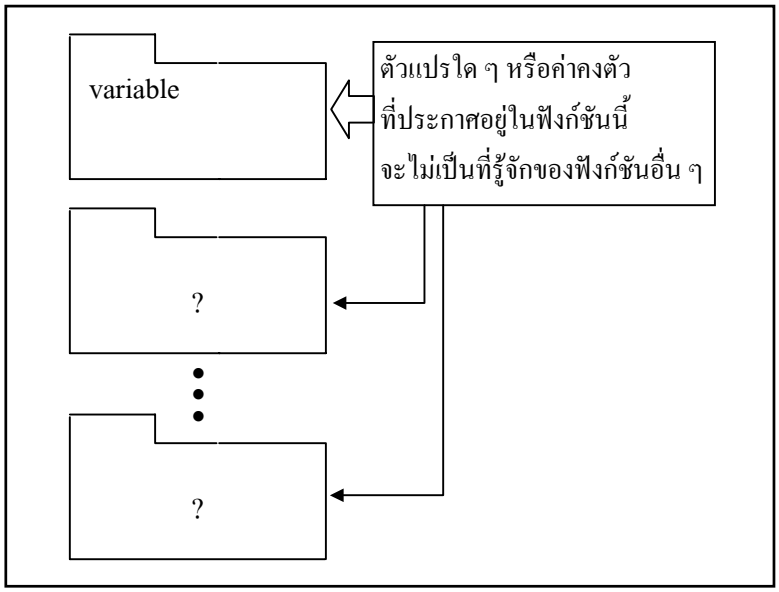
ตัวแปรส่วนกลาง เป็นตัวแปรที่ทุก ๆ ฟังก์ชันในโปรแกรมสามารถเรียกใช้ได้ โดยตัวแปรนี้จะมีช่วงชีวิต (lifetime) อยู่ตลอดไป จนกระทั่งยุติโปรแกรมจึงจะถูกทำลาย ตัวแปรนี้จะต้องถูกประกาศภายนอกฟังก์ชัน และก่อนฟังก์ชัน main () แสดงได้ดังรูปที่ 7.4



รูปที่ 7.4 แสดงขอบเขตของตัวแปรส่วนกลาง (Scope of global variable)

7.10 ตัวแปรเฉพาะที่ (local variable)

ตัวแปรเฉพาะที่ จะเป็นตัวแปรที่ถูกใช้เฉพาะภายในฟังก์ชันของตัวมันเอง และไม่เป็นที่รู้จักของฟังก์ชันอื่น ๆ เมื่อฟังก์ชันนี้ถูกเรียกใช้ ตัวแปรเฉพาะที่ทุกตัวที่อยู่ภายในฟังก์ชันนี้ จะถูกกำหนดเนื้อที่ให้กับตัวแปรแต่ละตัว โดยใช้หน่วยความจำที่เรียกว่า สแตก (stack) และช่วงชีวิตของตัวแปรเหล่านี้จะสิ้นสุดและถูกทำลายเมื่อสิ้นสุดฟังก์ชันนี้ ตัวแปรนี้จะต้องถูกประกาศอยู่ภายในแต่ละฟังก์ชัน และ formal parameter ก็จะถือว่าเป็นตัวแปรเฉพาะที่ด้วย แสดงได้ดังรูปที่ 7.5



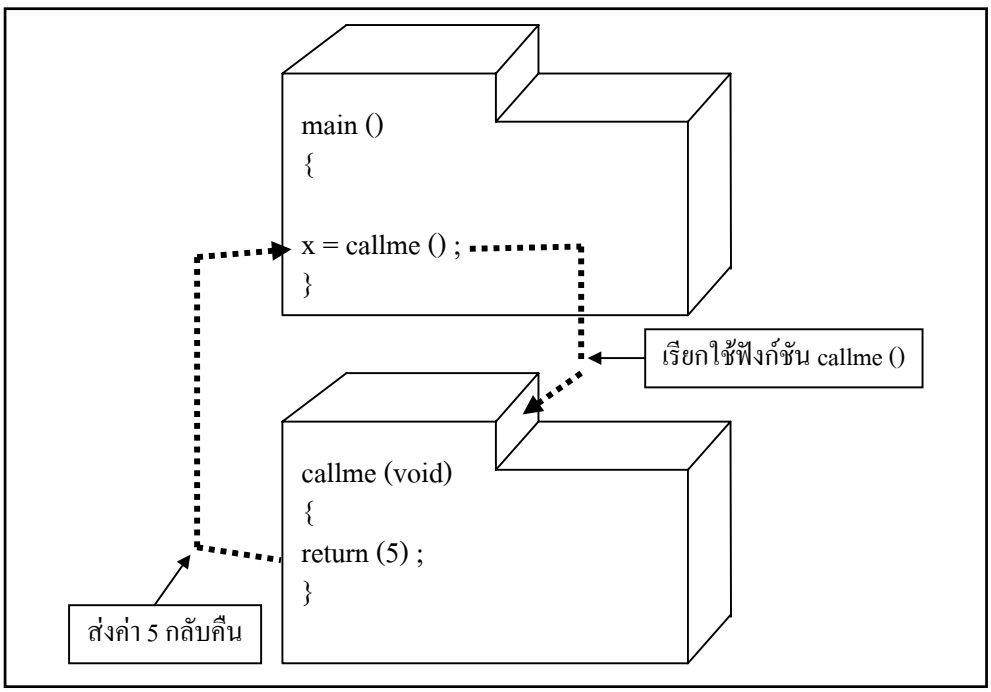
รูปที่ 7.5 แสดงขอบเขตของตัวแปรเฉพาะที่ (Scope of local variable)

ตัวอย่างโปรแกรมที่ 7.2 เป็นโปรแกรมที่แสดงการส่งค่ากลับคืนมายังฟังก์ชันที่เรียกใช้

```
#include <stdio.h>
int callme (void) ;      /* The function to be called by main.     */
main ()
{
    int  x ;             /* Variable to receive a value from the called function.  */
    x = callme () ;
    printf ( “Value of x is : %d”, x ) ;
    exit (0) ;
}
int callme (void)
{
    return (5) ;
}
```

```
Value of x is : 5
```

โปรแกรมนี้จะมีการเรียกใช้ฟังก์ชัน callme () เมื่อโปรแกรมส่งการควบคุมไปยังฟังก์ชัน callme () ก็จะส่งค่า 5 กลับมายังชื่อของฟังก์ชัน callme () ในฟังก์ชัน main () ทำให้ x มีค่าเท่ากับ 5 แสดงได้ดังรูปที่ 7.6



รูปที่ 7.6

ตัวอย่างโปรแกรมที่ 7.3 เป็นโปรแกรมแสดงการผ่านค่าโดยการอ้างอิง

```
#include <stdio.h>

void callme (int *p) ;

main ()
{
    int x ;

    x = 0 ;
    printf ( “The value of x is %d\n”, x ) ;

    callme (&x) ;
    printf ( “The new value of x is %d”, x ) ;

}

void callme (int *p)
{
    *p = 5
}
```

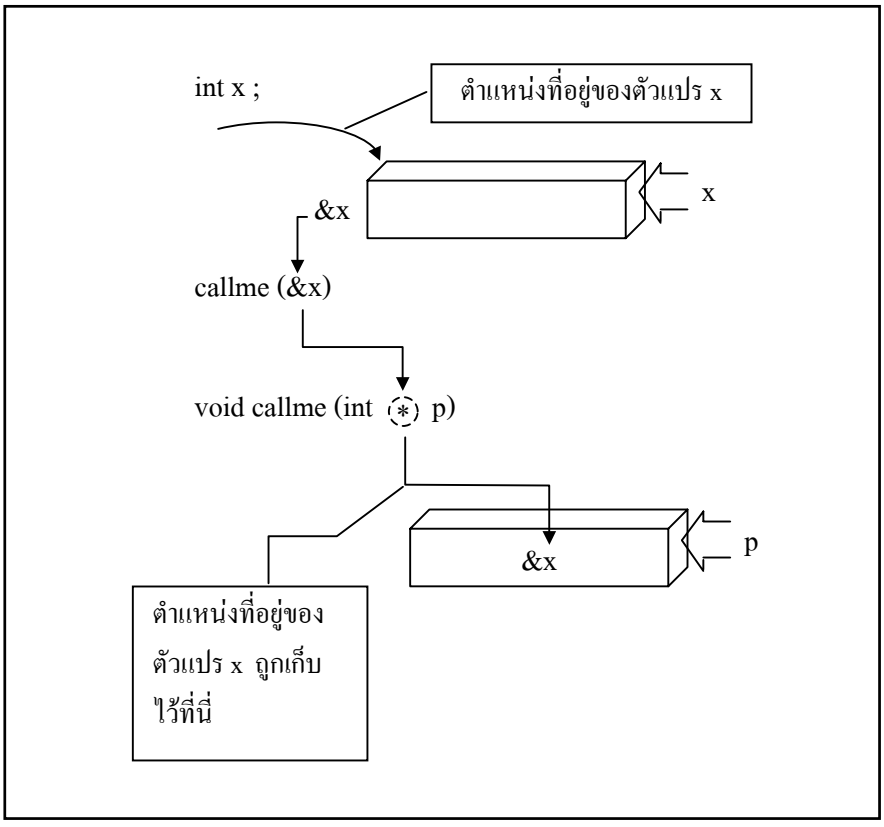
```
The value of x is 0
The new value of x is 5
```

โปรแกรมนี้ในโปรแกรม main () มีการเรียกใช้ฟังก์ชัน callme (&x) ก็จะมีการส่งที่อยู่ของตัวแปร x (&x) ไปยังฟังก์ชันที่ถูกเรียก โดยตัวแปร p จะเก็บที่อยู่ของตัวแปร x เมื่อพบข้อความสั่ง *p = 5 หมายถึง จะนำค่าของ 5 ไปเก็บไว้ยังที่อยู่ที่เก็บไว้ในตัวแปร p นั่นคือ ไปเก็บไว้ในตำแหน่งของตัวแปร x ทำให้ x มีค่าเท่ากับ 5

จากข้อความสั่งต่อไปนี้

```
main ()
{
int x ;

callme (&x) ;
}
```

จะมีความหมายดังรูปที่ 7.7

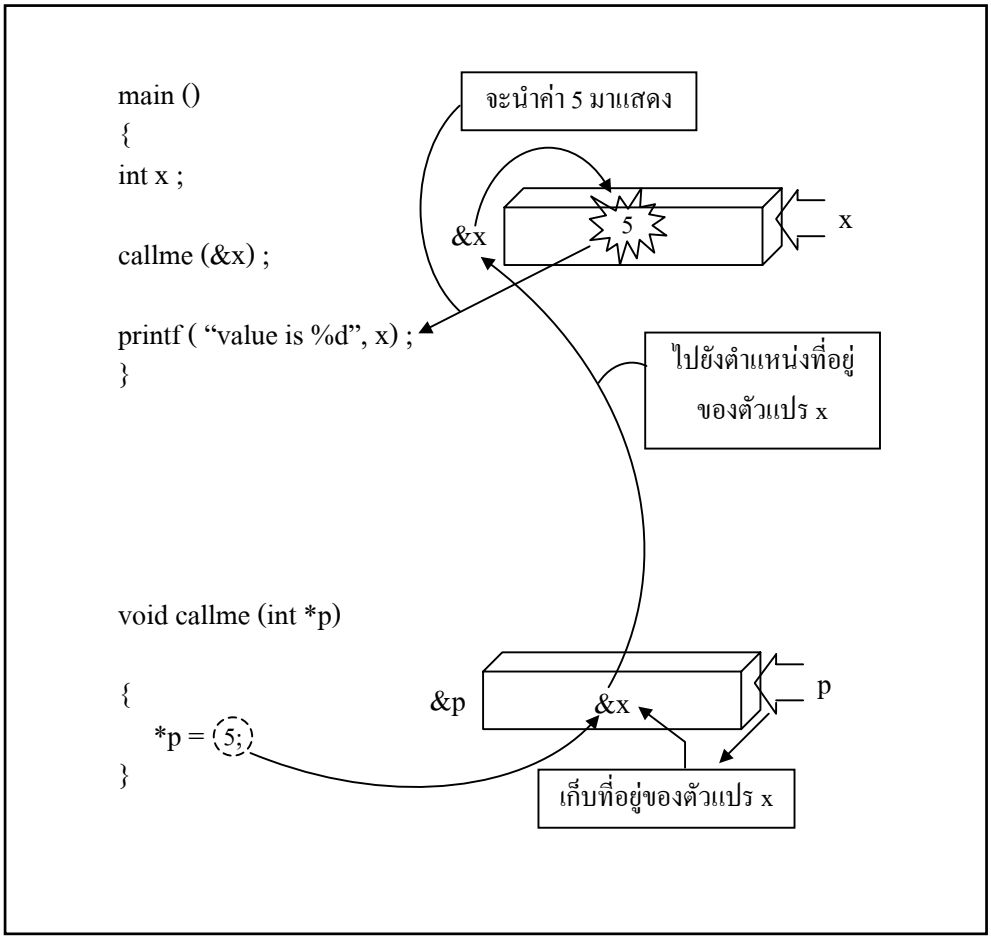


รูปที่ 7.7

จากข้อความสั่งต่อไปนี้

```
void callme (int  *p)
{
    *p = 5
}
```

จะมีความหมายดังรูปที่ 7.8



รูปที่ 7.8

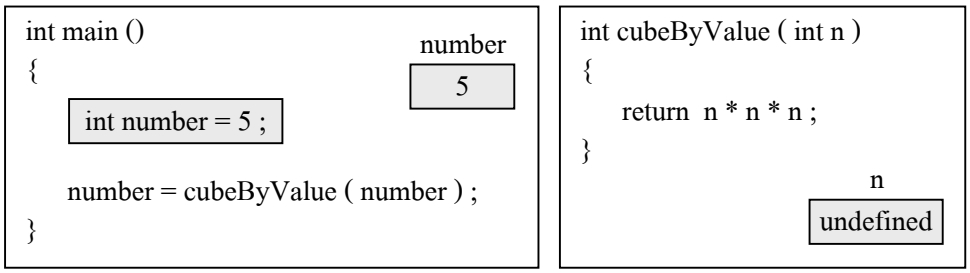
ตัวอย่างโปรแกรมที่ 7.4 เป็นโปรแกรมที่แสดงการผ่านโดยระบุค่า

```
#include <stdio.h>

int cubeByValue (int n) ;   /* prototype */

int main ()
{
    int number = 5 ;  /* initialize number */

    printf ( "The original value of number is %d", number ) ;

    /* pass number by value to cubeByValue */
    number = cubeByValue (number) ;
                                          actual parameter
    printf ( "\nThe new value of number is %d\n", number ) ;

    return 0 ;  /* indicates successful termination */

}   /* end main */

/* calculate and return cube of integer argument */
int cubeByValue (int n)
{                                         formal parameter
    return n * n * n ;     /* cube local variable n and return result */

}   /* end function cubeByValue */
```

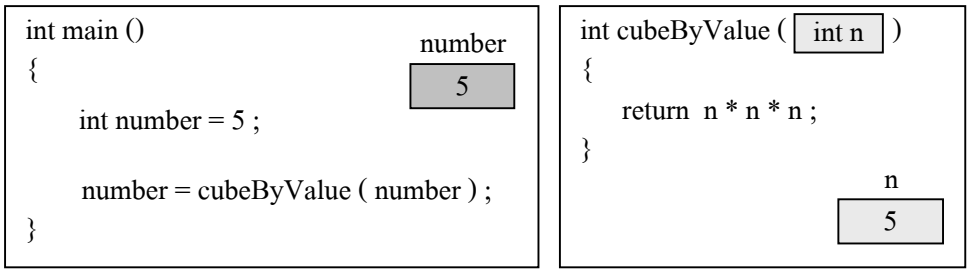
```
The original value of number is 5
The new value of number is 125
```

จากโปรแกรมนี้ สามารถแสดงการผ่านโดยระบุค่าได้ดังนี้

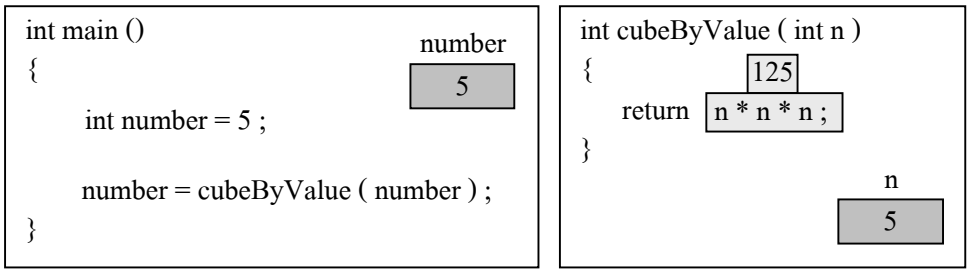
ก่อนการเรียกใช้ฟังก์ชัน cubeByValue



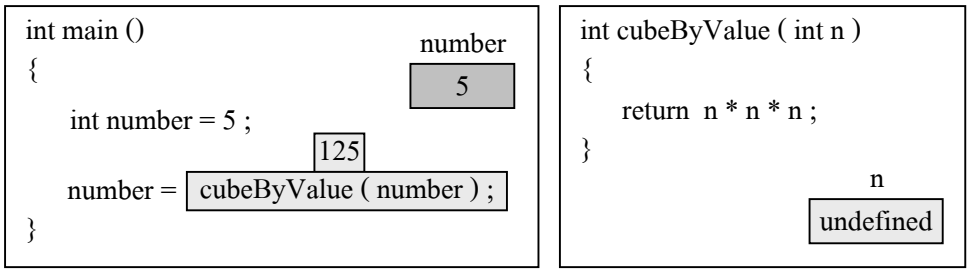
หลังจากฟังก์ชัน cubeByValue ถูกเรียกและได้รับค่าจาก actual parameter



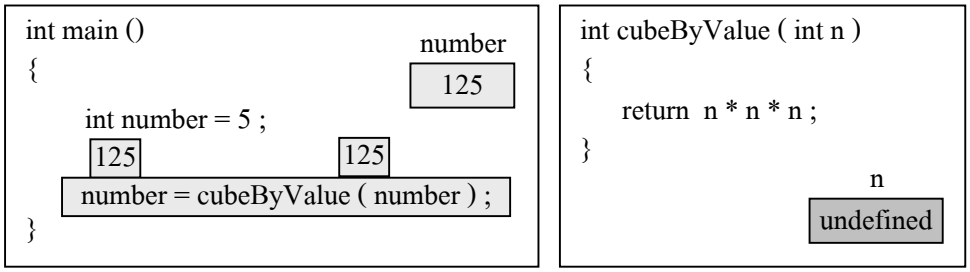
ก่อนฟังก์ชัน cubeByValue จะคืนค่ากลับไปยังฟังก์ชัน main ()



หลังจากฟังก์ชัน cubeByValue คืนค่ากลับไปยังฟังก์ชัน main () และก่อนกำหนดค่าให้กับตัวแปร number



หลังจากกำหนดค่าให้ตัวแปร number และสิ้นสุดฟังก์ชัน main ()



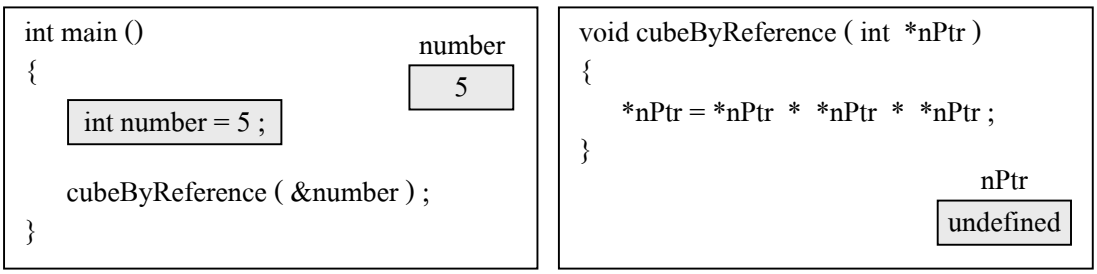
ตัวอย่างโปรแกรมที่ 7.5 เป็นโปรแกรมแสดงการผ่านโดยการอ้างอิง

```
/* Cube a variable using call–by–reference with a pointer argument */

#include <stdio.h>
                                          ในฟังก์ชันต้นแบบต้องมี * เติมหน้า
void cubeByReference ( int *nPtr ) ;  /* Prototype */

int main ()
{
    int number = 5 ;   /* initialize number */

    printf ( “The original value of number is %d”, number ) ;

    /* pass address of number to cubeByReference */
    cubeByReference ( &number ) ;
                          actual parameter ต้องมี & เติมหน้า
    printf ( “\nThe new value of number is %d\n”, number ) ;

    return 0 ;   /* indicates successful termination */

}   /* end main */

/* calculate cube of *nPtr ; modifies variable number in main */
void cubeByReference ( int *nPtr )
{                              formal parameter ต้องมี * เติมหน้า
    *nPtr = *nPtr  *  *nPtr  *  *nPtr ;  /* cube *nPtr */
}   /* end function cubeByReference */
```

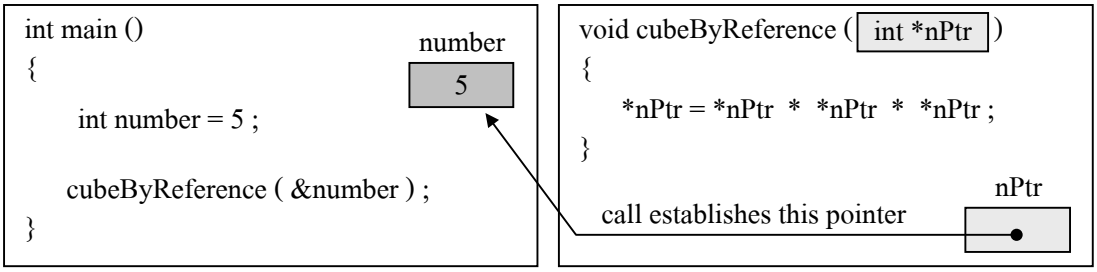
```
The original value of number is 5
The new value of number is 125
```

จากโปรแกรมนี้ สามารถแสดงการผ่านโดยการอ้างอิงได้ดังนี้

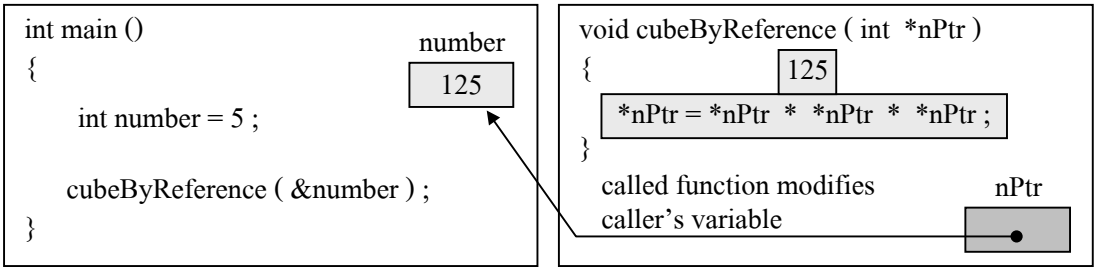
ก่อนเรียกใช้ฟังก์ชัน cubeByReference



หลังจากฟังก์ชัน cubeByReference ถูกเรียกและก่อนการคำนวณกำลัง 3 ของ *nPtr



หลังจากการคำนวณกำลัง 3 ของ *nPtr และก่อนการส่งการควบคุมกลับไปยังฟังก์ชัน main ()



7.11 ประเภทหน่วยเก็บของตัวแปร (Storage class of variables)

ตัวแปรแต่ละตัวในภาษา C จะมีประเภทหน่วยเก็บที่แตกต่างกัน เพื่อที่จะใช้กำหนดความมองเห็นได้ (visibility), ช่วงชีวิต (lifetime) และตำแหน่ง (location)

ในภาษา C จะมีประเภทหน่วยเก็บของตัวแปร 4 แบบ ดังนี้

7.11.1 ตัวแปรอัตโนมัติ (automatic variable)

7.11.2 ตัวแปรสถิต (static variable)

7.11.3 ตัวแปรภายนอก (external variable)

7.11.4 ตัวแปรเรจีสเตอร์ (register variable)

7.11.1 ตัวแปรอัตโนมัติ (automatic variable)

ในภาษา C ตัวแปรใด ๆ ที่ถูกประกาศอยู่ภายในฟังก์ชัน รวมทั้งรายการของ formal parameter จะเป็นตัวแปรเฉพาะที่ (local variable) และจะถูกกำหนดให้เป็นตัวแปรอัตโนมัติ โดยจะมีการระบุโดยใช้คำสงวน auto ถ้าไม่ระบุก็จะมี default เป็น auto และจะมีขอบเขตและช่วงชีวิตอยู่ภายใน block ของฟังก์ชันที่ตัวแปรนี้ถูกประกาศเท่านั้น

ตัวอย่างโปรแกรมที่ 7.6 เป็นโปรแกรมที่มีการใช้คำสงวน auto

```
#include <stdio.h>

main ()
{
    auto int value ;

    value = 5 ;

    printf ( “The value is %d”, value ) ;
}
```

จากตัวอย่างนี้ ตัวแปร value จะเป็นตัวแปรอัตโนมัติ เนื่องจากมีการระบุคำสงวน auto บางครั้งเราไม่จำเป็นต้องเขียน auto ก็ได้ เนื่องจากในภาษา C จะมี default เป็น auto

7.11.2 ตัวแปรสถิต (static variable)

ในภาษา C ตัวแปรสถิตก็เป็นตัวแปรเฉพาะที่เหมือนกัน แต่ไม่เหมือนตัวแปรเฉพาะที่โดยทั่วไปคือ ตัวแปรสถิตจะมีช่วงชีวิตอยู่จนกว่าโปรแกรมจะยุติ ถึงแม้ว่าฟังก์ชันที่ถูกเรียกใช้จะสิ้นสุดไปแล้วก็ตาม แต่ตัวแปรสถิตนี้ยังไม่ถูกทำลาย และยังจำค่าปัจจุบันอยู่ และมีขอบเขตอยู่ภายในฟังก์ชันที่ตัวแปรสถิตนี้อยู่ โดยปกติแล้วเมื่อสิ้นสุดฟังก์ชันที่ตัวแปรเฉพาะที่นี้อยู่ ตัวแปรเฉพาะที่ทุกตัวก็จะถูกทำลายหมด

ตัวอย่างโปรแกรมที่ 7.7 แสดงการใช้ตัวแปรสถิต

```
#include <stdio.h>

void second_function (void) ;

main ()
{
    second_function () ;
    second_function () ;
    second_function () ;
}

void second_function (void)
{
    static int number ;

    number+ + ;
    printf ( “This function has been called %d times. \n”, number ) ;
}
```

```
This function has been called 1 times.
This function has been called 2 times.
This function has been called 3 times.
```

ตัวอย่างโปรแกรมที่ 7.8 เป็นโปรแกรมที่มีการใช้ตัวแปร automatic กับตัวแปร static ในฟังก์ชัน trystat

```
#include <stdio.h>
void trystat (void) ;
int main (void)
{
    int count ;

    for (count = 1 ; count < = 3 ;  count+ +)
    {
        printf ( “Here comes iteration %d : \n”,  count ) ;
        trystat () ;
    }
    return 0 ;
}
void trystat (void)
{
    int fade = 1 ;
    static int stay = 1 ;

    printf ( “fade = %d and stay = %d\n”,  fade+ +,  stay+ + ) ;
}
```

```
Here comes iteration 1 :
fade = 1 and stay = 1
Here comes iteration 2 :
fade = 1 and stay = 2
Here comes iteration 3 :
fade = 1 and stay = 3
```

ตัวอย่างโปรแกรมนี้ ในฟังก์ชัน trystat มีการประกาศตัวแปรสถิตคือ stay และตัวแปรอัตโนมัติ คือ fade ความแตกต่างของทั้ง 2 ตัวแปร คือ เมื่อสิ้นสุดการเรียกใช้ฟังก์ชัน trystat ค่าของตัวแปร fade จะถูกทำลายไป แต่ค่าของตัวแปร stay ยังคงอยู่ เนื่องจากได้กำหนดให้เป็นตัวแปรสถิต แต่ตัวแปรทั้งสองก็ยังคงมีขอบเขตภายในฟังก์ชันนี้เหมือนกัน

7.11.3 ตัวแปรภายนอก (external variable)

ตัวแปรที่ถูกประกาศไว้ภายนอกฟังก์ชัน ในภาษา C จะกำหนดให้เป็นตัวแปรภายนอก และภายในแต่ละฟังก์ชันก็สามารถประกาศให้เป็นตัวแปรภายนอกได้เช่นกัน โดยระบุคำสงวนว่า extern ขอบเขตและช่วงชีวิตของตัวแปรภายนอกนี้ จะเริ่มต้นที่จุดประกาศตัวแปรไปสิ้นสุดที่โปรแกรมยุติ

ตัวอย่างโปรแกรมที่ 7.9 เป็นโปรแกรมที่มีการใช้ตัวแปรภายนอกอยู่ภายในฟังก์ชัน main () และฟังก์ชัน critic ()

```
#include <stdio.h>
int Units ;                    /* an external variable */
void critic (void) ;
int main (void)
{
    extern int Units ;

    printf ( “How many pounds to a firkin of butter? \n” ) ;
    scanf ( “%d”,  &Units ) ;
    while (  Units  ! = 56 )
        critic () ;
    printf ( “You must have looked it up ! \n” ) ;
    return 0 ;
}
void critic (void)
{
    extern int Units ;

    printf ( “No luck,  chummy.  Try again. \n” ) ;
    scanf ( “%d”,  &Units ) ;
}
```

จะเรียกว่าเป็นนิยามการประกาศ

จะเรียกว่าเป็นการอ้างอิงการประกาศ

```
How many pounds to a firkin of butter?
14
No luck,  chummy.  Try again.
56
You must have looked it up !
```

ตัวอย่างโปรแกรมนี้ จะมีตัวแปรประกาศ Units ซ้ำกันอยู่ 3 ที่ คือ อยู่ก่อนฟังก์ชัน main (), อยู่ภายในฟังก์ชัน main () แต่ระบุเป็นตัวแปรภายนอก และอยู่ภายในฟังก์ชัน critic () และระบุเป็นตัวแปรภายนอก ถ้าภายในฟังก์ชัน main () และ critic () ไม่ได้ระบุเป็นตัวแปรภายนอก เราจะถือว่าเป็นตัวแปรเฉพาะที่ แต่การที่เราระบุเป็นตัวแปรภายนอกจุดประสงค์เพื่อจะบอกให้คอมไพเลอร์รู้ว่า เราต้องการอ้างถึงตัวแปร Units ที่ถูกประกาศอยู่ภายนอกฟังก์ชัน หรือบางครั้งอาจจะนอกโปรแกรมก็ได้ ดังนั้น ตัวแปร Units ที่ประกาศอยู่ก่อนฟังก์ชัน main () เราจะเรียกว่าเป็นนิยามการประกาศ (defining declaration) และตัวแปรภายนอกทั้งสอง จะเรียกว่าเป็นการอ้างอิงการประกาศ (referencing declaration)

7.11.4 ตัวแปรเรจีสเตอร์ (register variable)

โดยปกติตัวแปรใด ๆ จะถูกเก็บไว้ในหน่วยความจำหลัก แต่ถ้าเราต้องการให้ตัวแปรไปอยู่หน่วยความจำประเภทเรจีสเตอร์ เพื่อต้องการเข้าถึงตัวแปรเหล่านั้นได้รวดเร็วยิ่งขึ้น เราจะต้องระบุคำสงวน register ไว้หน้าตัวแปร

ตัวอย่างโปรแกรมที่ 7.10 เป็นโปรแกรมที่มีการระบุคำสงวน register เพื่อต้องการที่จะเข้าถึงตัวแปรนี้ได้รวดเร็วยิ่งขึ้น

```
#include <stdio.h>

main ()
{
    register counter ;

    for (counter = 0 ;  counter > 5 ;  counter+ +) ;
}
```

ตัวอย่างโปรแกรมที่ 7.11 จะเป็นการคำนวณหาค่ากำลังสองของค่า x โดยส่งค่า x ไปยังฟังก์ชัน square (int y) ซึ่งทำหน้าที่คำนวณหาค่ากำลังสองของ y แล้วส่งค่ากลับมายังฟังก์ชันเรียกใช้

```
#include <stdio.h>

int square (int y) ;   /* function prototype */

/* function main begins program execution */
int main ()
{
    int x ;  /* counter */

    /* loop 10 times and calculate and output square of x each time */
    for ( x = 1 ;  x < = 10 ; x + + ) {
        printf ( "%d  ", square ( x )  ) ;  /* function call */
    }   /* end for */

    printf ( "\n" ) ;

    return 0 ;  /* indicates successful termination */

}   /* end main */

/* square function definition returns square of parameter */
int square ( int y )   /* y is a copy of argument to function */
{
    return  y * y ;  /* return square of y as an int */

}   /* end function square */
```

```
1   4   9   16   25   36   49   64   81   100
```

จากโปรแกรมนี้ในฟังก์ชัน main () จะมีการเรียกใช้ฟังก์ชัน square (x) จำนวน 10 ครั้ง โดย

รอบที่ 1 จะมีการส่งค่า x = 1 ไปยังฟังก์ชันถูกเรียกโดยสำเนาค่า x ไปให้ตัวแปร y แล้วคำนวณหาค่าของ y * y แล้วส่งค่ากลับมายังฟังก์ชันที่เรียก ซึ่งในกรณีนี้มีค่าเท่ากับ 1 และพิมพ์ผล

รอบที่ 2 จะมีการส่งค่า x = 2 และทำในทำนองเดียวกันกับรอบที่ 1 แต่ค่าที่ส่งกลับมาจะเท่ากับ 4 และพิมพ์ผล

! ! !

รอบที่ 10 จะมีการส่งค่า x = 10 และทำในทำนองเดียวกัน แต่ค่าที่ส่งกลับมาจะเท่ากับ 100 แล้วพิมพ์ผล

ตัวอย่างโปรแกรมที่ 7.12 จะเป็นโปรแกรมที่คำนวณหาค่ามากที่สุดของจำนวนเต็ม 3 จำนวน ที่ถูกป้อนเข้ามาทางแป้นพิมพ์ โดยฟังก์ชันที่ใช้คำนวณหาค่ามากที่สุดของจำนวนเต็ม 3 จำนวน คือ

```
int maximum ( int x,  int y,  int z )
```

```
/* Finding the maximum of three integers */
#include <stdio.h>

int maximum (  int x,  int y,  int z  ) ;   /* function prototype */

/* function main begins program execution */
int main ()
{
    int  number1 ;  /* first integer */
    int  number2 ;  /* second integer */
    int  number3 ;  /* third integer */

    printf ( "Enter three inegers :" ) ;
    scanf ( "%d%d%d", &number1, &number2, &number3 ) ;

    /* number1, number2 and number3 are arguments
       to the maximum function call */
    printf ( "Maximum is : %d\n", maximum ( number1, number2, number3 ) ) ;

    return 0 ;   /* indicates successful termination */

}   /* end main */

/* Function maximum definition */
/* x, y and z are parameters */
int maximum ( int x,  int y,  int z )
{
    int max = x ;          /* assume x is largest */

    if ( y > max )   {    /* if y is larger than max, assign y to max */
        max = y ;
    }   /* end if */

    if ( z > max )   {    /* if z is larger than max, assign z to max */
        max = z ;
    }   /* end if */

    return max ;           /* max is largest value */

}   /* end function maximum */
```

```
Enter three integer : 22  85  17
Maximum is :  85
```

```
Enter three integer : 85  22  17
Maximum is :  85
```

```
Enter three integer : 22  17  85
Maximum is :  85
```

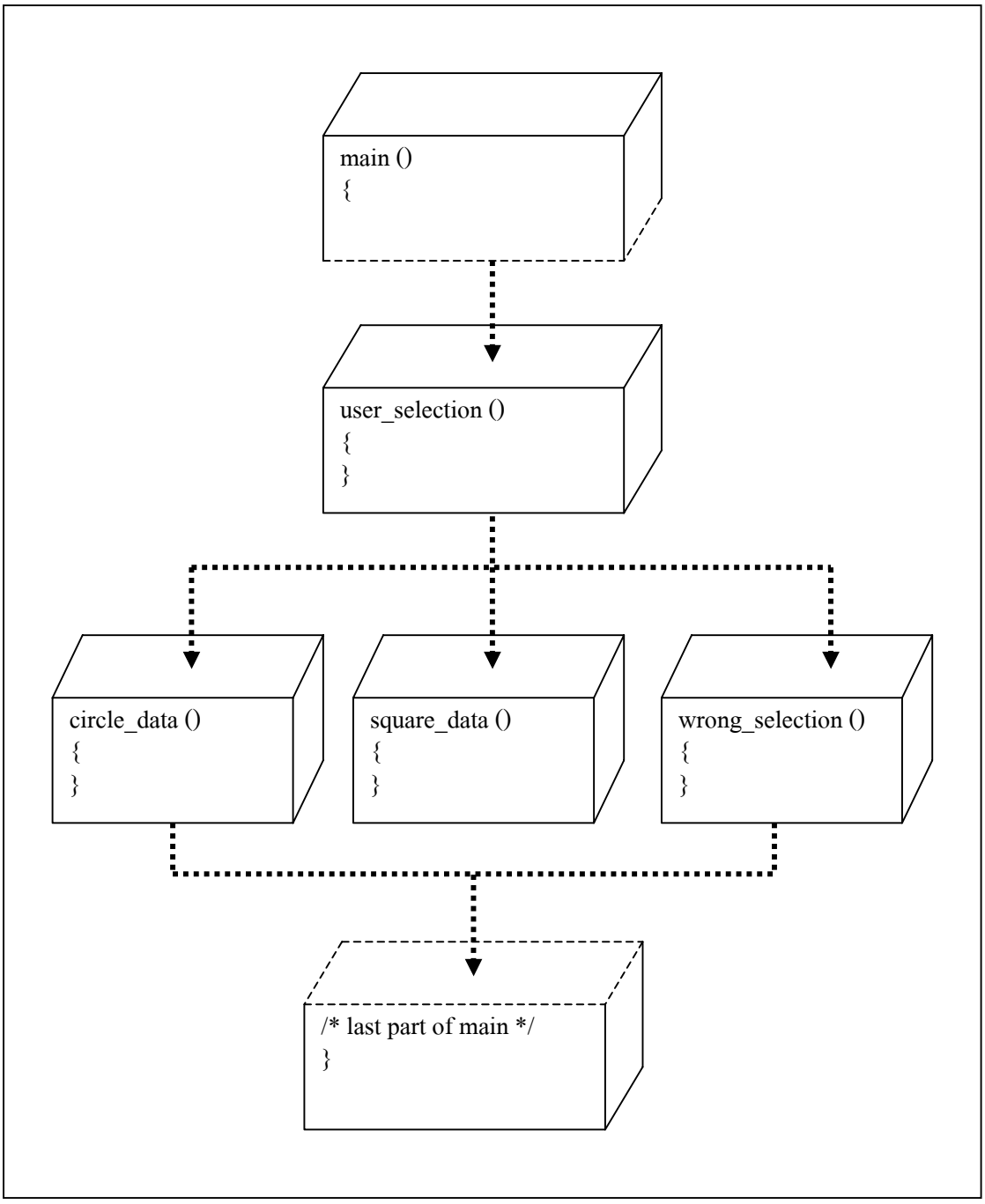
ตัวอย่างโปรแกรมที่ 7.13 เป็นโปรแกรมแสดงรายการ (Menu) ให้คำนวณหาพื้นที่วงกลมและพื้นที่สี่เหลี่ยมจัตุรัส

```
#include <stdio.h>
#define  PI 3.141592              /* The constant pi.     */
#define  square (x)  x * x        /* Area of a square.    */
#define  circle (r)  PI * r * r   /* Area of a circle.    */

void  user_selection (void) ;     /* Get selection from user.          */
void  circle_data (void) ;        /* Get circle radius and compute.    */
void  square_data (void) ;        /* Get square side and compute.      */
void  wrong_selection (void) ;    /* Notify user of wrong selection.   */

main ()
{
    printf ( "\n\nThis program will compute the area of\n" ) ;
    printf ( "a square or the area of a circle. \n" ) ;

    user_selection () ;           /* Get selection from user. */

    printf ( "\n\nThis concludes the program to calculate \n" ) ;
    printf ( "the area of a circle or a square." ) ;

    exit (0) ;
}

void user_selection (void)        /* Get selection from user. */
{
    float selection ;             /* User selection.          */

    printf ( "\nSelect by number : \n" ) ;
    printf ( "1]  Area of circle.  2]  Area of square. \n" ) ;
    printf ("You selection  (1 or 2)  = >  " ) ;
    scanf ( "%f", &selection) ;

    if  (selection = = 1)
        circle_data ( ) ;
    else
    if  (selection = = 2)
        square_data ( ) ;
    else
        wrong_selection ( ) ;
}
```

```
void circle_data (void)          /* Get circle radius and compute.    */
{
    float radius ;               /* Radius of the circle.             */
    float area ;                 /* Circle are in square units.       */

    printf ( “Give me the length of the circle radius = >  ” ) ;
    scanf ( “%f”, &radius ) ;

    area  =  circle (radius) ;

    printf ( “A circle of radius %f has an area of  ”, radius ) ;
    printf ( “%f square units.”, area ) ;
}

void square_data (void)          /* Get square side and compute.      */
{
    float side ;                 /* Side of the square.               */
    float area ;                 /* Area of the square in square units. */

    printf ( “Give me the length of one side of the square = >  ” ) ;
    scanf ( “%f”, &side ) ;

    area  =  square (side) ;

    printf ( “A square of length %f has an area of  ”, side ) ;
    printf ( “%f square units.”, area ) ;
}

void wrong_selection (void)      /* Notify user of wrong selection.   */
{
    printf ( “That was not one of the selections. \n” ) ;
    printf ( “You must run the program again and \n” ) ;
    printf ( “select either a 1 or a 2 . \n” ) ;
}
```

โปรแกรมนี้สามารถแสดงโครงสร้างได้ดังรูปที่ 7.9

ซึ่งจะแบ่งโปรแกรมออกเป็น block ของฟังก์ชัน


main ()
{
user_selection ()
{
}
circle_data ()
{
}
square_data ()
{
}
wrong_selection ()
{
}
/* last part of main */
}

รูปที่ 7.9

ตัวอย่างโปรแกรมที่ 7.14 เป็นโปรแกรมที่ในฟังก์ชัน other_function (void) มีการใช้ตัวแปร a_variable แต่จะแปลโปรแกรมไม่ผ่านเนื่องจากไม่รู้จักตัวแปร a_variable

```
#include <stdio.h>

void other_function (void) ;

main ()
{
    int a_variable ;

    a_variable = 5 ;

    printf ( “The value of a_variable is %d\n”, a_variable ) ;
    other_function () ;
}

void other_function (void)
{
    printf ( “The value of a_variable in this function is %d”, a_variable ) ;
}
```

จากโปรแกรมนี้ในฟังก์ชัน main () มีการกำหนดให้ตัวแปร a_variable เป็นชนิด int มีค่าเริ่มต้นเท่ากับ 5 เมื่อสิ้นสุดฟังก์ชัน main () ตัวแปร a_variable ก็จะถูกทำลาย เนื่องจากเป็นตัวแปรเฉพาะที่ และในฟังก์ชัน main () มีการเรียกใช้ฟังก์ชัน other_function () ก็จะส่งการควบคุมโปรแกรมไปยังฟังก์ชัน other_function () มีการเรียกใช้ตัวแปร a_variable แต่ไม่ได้มีการประกาศ ว่าเป็นตัวแปรชนิดใด ดังนั้นจึงแปลโปรแกรมไม่ผ่าน

ตัวอย่างโปรแกรม 7.15 จะมีลักษณะคล้ายกับโปรแกรมที่ 7.14 เพียงแต่ว่าจะมีการประกาศตัวแปร a_variable เป็นตัวแปรส่วนกลาง ดังนั้นในฟังก์ชัน other_function () ก็จะสามารถเรียกใช้ตัวแปร a_variable ได้แล้ว

```
#include <stdio.h>

void other_function (void) ;
int a_variable ;  <-------- ตัวแปรส่วนกลาง

main ()
{
    a_variable = 5 ;

    printf ( “The value of a_variable is %d\n”, a_variable ) ;
    other_function () ;
}

void other_function ()
{
    printf ( “The value of a_variable in this function is %d”, a_variable ) ;
}
```

```
The value of a_variable is 5
The value of a_variable in this function is 5
```

ตัวอย่างโปรแกรมที่ 7.16 เป็นโปรแกรมแสดงการใช้ตัวแปรส่วนกลาง คือ counter ถ้าเราไม่ระมัดระวังอาจจะทำให้เราเข้าใจคลาดเคลื่อน

```
#include <stdio.h>

void other_function (void) ;
int counter ;

main ()
{
    counter = 0 ;

    while  (counter < 5)
    {
        printf ( “The count is %d\n”, counter ) ;
        counter+ + ;
    }

    other_function () ;
}

void other_function (void)
{
    while  (counter < 6)
    {
        printf ( “The count in this function is %d\n”, counter ) ;
        counter+ + ;
    }
}
```

```
The count is 0
The count is 1
The count is 2
The count is 3
The count is 4
The count in this function is 5
```

จากโปรแกรมนี้ ในฟังก์ชัน other_function (void) ตัวแปร count ก่อนเข้าข้อความสั่งวนซ้ำ while จะมีค่าเท่ากับ 5 เนื่องจากช่วงชีวิตของตัวแปรส่วนกลาง counter ยังไม่สิ้นสุด

ตัวอย่างโปรแกรมที่ 7.17 เป็นโปรแกรมแสดงการคำนวณหาผลบวกของจำนวนเต็มคู่และคี่

```
1    #include <stdio.h>

2    int main (void)  {
3        int limit,  count = 1,  sum_of_odd = 0,  sum_of_all ;

4        printf ( “Enter an integer :   ” ) ;
5        scanf ( “%d”,  &limit ) ;

6        while  (count < = limit)  {
7            if  (count  %  2)
8                sum_of_odd + =  count ;
9            /* end if */

10           count+ + ;
11       }  /* end while */

12       printf ( “%d\n”,  sum_of_odd ) ;

13       {
14           int count = 1,  sum_of_even = 0 ;

15           while  (count < = limit)  {
16               if  (count %2 = = 0)
17                   sum_of_even + = count ;
18               /* end if */

19               count+ + ;
20           }  /* end while */

21           printf ( “%d\n”,  sum_of_even ) ;

22           sum_of_all = sum_of_odd  +  sum_of_even ;
23       }

24       printf ( “%d\n”,  sum_of_all ) ;

25       return 0 ;
26   }  /* end function main */
```

จากโปรแกรมนี้ ตัวแปรทุกตัวจะเป็นแบบเฉพาะที่ โดยบรรทัดที่ 3 จะมีตัวแปรเฉพาะที่ คือ limit, count, sum_of_odd และ sum_of_all เป็นของฟังก์ชัน main () และในบรรทัดที่ 14 จะมีตัวแปรเฉพาะที่คือ count และ sum_of_even เป็นของ block ที่เริ่มตั้งแต่บรรทัดที่ 13 ถึง บรรทัดที่ 23 และเราจะพบว่ามีการประกาศตัวแปรซ้ำกัน คือ count ในบรรทัดที่ 3 และ บรรทัดที่ 14 ในกรณีนี้ ถึงแม้จะมีชื่อตัวแปรเหมือนกัน แต่ตัวแปร 2 ตัวนี้ จะถือว่าเป็นคนละตัวกัน โดยจะเก็บอยู่ในตำแหน่งของหน่วยความจำที่ต่างกัน

ตารางที่ 7.1 แสดงขอบเขตและความมองเห็นได้ของตัวแปรในโปรแกรมนี้

ตัวแปร	ประกาศในบรรทัดที่	ขอบเขต	โซนความมองเห็นได้	โซนความมองเห็นไม่ได้
limit	3	บรรทัด 3–26	บรรทัด 3–26	ไม่มี
count	3	บรรทัด 3–26	บรรทัด 3–12, 24–26	บรรทัด 14–23
sum_of_odd	3	บรรทัด 3–26	บรรทัด 3–26	ไม่มี
sum_of_all	3	บรรทัด 3–26	บรรทัด 3–26	ไม่มี
count	14	บรรทัด 14–23	บรรทัด 14–23	ไม่มี
sum_of_even	14	บรรทัด 14–23	บรรทัด 14–23	ไม่มี

จากตารางที่ 7.1 เราจะพบว่า ตัวแปร limit, sum_of_odd และ sum_of_all มีความมองเห็นได้ในขอบเขตตั้งแต่บรรทัดที่ 3 ถึง 26 และตัวแปร sum_of_even มีความมองเห็นได้ในขอบเขตตั้งแต่บรรทัดที่ 14 ถึง 23

ถ้ารวมตัวแปรที่ประกาศใน block เข้ารวมไว้ในขอบเขตที่มีชื่อตัวแปรเหมือนกันแล้ว ตัวแปร count ก่อนหน้านี้ จะไม่สามารถมองเห็นได้ และจะถูกปิดบังอยู่ภายในขอบเขตก่อนหน้านี้ หลังจากออกจากขอบเขตนี้แล้ว (บรรทัดที่ 13 ถึง 23) ตัวแปรที่ถูกปิดบังอยู่ ก็จะสามารถมองเห็นได้อีกครั้ง กล่าวคือ ตัวแปร count ที่ประกาศในบรรทัดที่ 14 จะทำให้ตัวแปร count ที่ประกาศในบรรทัดที่ 3 ไม่สามารถมองเห็นได้ ตั้งแต่บรรทัดที่ 14 ถึง 23 หลังจากบรรทัดที่ 23 ไปแล้ว ตัวแปร count ที่ประกาศในบรรทัดที่ 3 ก็จะสามารถมองเห็นได้อีกครั้ง จนกระทั่งจบขอบเขต

ตัวอย่างโปรแกรมที่ 7.18 เหมือนกับตัวอย่างโปรแกรมที่ 7.17 เพียงแต่มีการประกาศตัวแปร count เป็นตัวแปรส่วนกลาง (global variable)

```
1    #include <stdio.h>

2    int sum_of_odd_values (int limit) ;
3    int sum_of_even_values (int limit) ;

4    int count ;

5    int main (void)  {
6        int limit,  sum_of_odd,  sum_of_even,  sum_of_all ;

7        printf ( " Enter an integer :   " ) ;
8        scanf ( "%d",  &limit ) ;

9        sum_of_odd = sum_of_odd_values (limit) ;
10       printf ( "%d\n",  sum_of_odd ) ;

11       sum_of_even = sum_of_even_value (limit) ;
12       printf ( "%d\n",  sum_of_even ) ;

13       sum_of_all = sum_of_odd + sum_of_even ;
14       printf ( "%d\n",  sum_of_all ) ;

15       return 0 ;
16   }  /* end function main */

17   int sum_of_odd_values (int limit)  {
18       int count = 1,  sum_of_odd = 0 ;

19       while  (count < = limit)  {
20           if  (count % 2)
21               sum_of_odd + = count ;
22           /* end if */

23           count+ + ;
24       }  /* end while */

25       return sum_of_odd ;
26       }  /* end function sum_of_odd_values */

27   int sum_of_even_values (int limit)  {
28       int count = 1,  sum_of_even = 0 ;

29       while  (count < = limit)  {
30           if  (count %2 = = 0)
31               sum_of_even + = count ;
32           /* end if */

33           count+ + ;
34       }  /* end while */

35       return sum_of_even ;
36   }  /* end function sum_of_even_values */
```

จากโปรแกรมนี้ จะมีการประกาศตัวแปร count ซ้ำกัน 3 ชื่อ โดยบรรทัดที่ 4 ตัวแปร count จะเป็นตัวแปรส่วนกลาง และในบรรทัดที่ 18 ตัวแปร count จะเป็นตัวแปรเฉพาะที่ของฟังก์ชัน sum_of_odd_values และในบรรทัดที่ 28 ตัวแปร count จะเป็นตัวแปรเฉพาะที่ของฟังก์ชัน sum_of_even_values

แม้ว่าขอบเขตของตัวแปรส่วนกลาง count จะขยายไปจนถึงบรรทัดที่ 36 แต่มันจะไม่สามารถมองเห็นได้ในขอบเขตอีก 2 ตัวแปร count ตัวแปร limit มีการประกาศซ้ำกันในบรรทัดที่ 6, 17 และ 27 แต่ตัวแปร limit ทั้ง 3 ตัวนี้ จะแตกต่างกันถึงแม้ว่าจะมีชื่อเหมือนกัน โดยจะไม่มีขอบเขตซ้อนกัน และมันจะสามารถมองเห็นได้ภายในขอบเขตของตัวเอง ในทำนองเดียวกัน ตัวแปร sum_of_odd และตัวแปร sum_of_even ถึงแม้จะมีการประกาศตัวแปรซ้ำกัน 2 ครั้ง แต่ขอบเขตจะไม่มีการซ้อนกัน

ตารางที่ 7.2 แสดงขอบเขตและความมองเห็นได้ของตัวแปร

ตัวแปร	ประกาศในบรรทัดที่	ขอบเขต	โซนความมองเห็นได้	โซนความมองเห็นไม่ได้
count	4	บรรทัด 4–36	บรรทัด 4–17, 26–27	บรรทัด 18–25, 28–36
limit	6	บรรทัด 6–16	บรรทัด 6–16	ไม่มี
sum_of_odd	6	บรรทัด 6–16	บรรทัด 6–16	ไม่มี
sum_of_even	6	บรรทัด 6–16	บรรทัด 6–16	ไม่มี
sum_of_all	6	บรรทัด 6–16	บรรทัด 6–16	ไม่มี
limit	17	บรรทัด 17–26	บรรทัด 17–26	ไม่มี
count	18	บรรทัด 18–26	บรรทัด 18–26	ไม่มี
sum_of_odd	18	บรรทัด 18–26	บรรทัด 18–26	ไม่มี
limit	27	บรรทัด 27–36	บรรทัด 27–36	ไม่มี
count	28	บรรทัด 28–36	บรรทัด 28–36	ไม่มี
sum_of_odd	28	บรรทัด 28–36	บรรทัด 28–36	ไม่มี

ตัวอย่างโปรแกรมที่ 7.19 เป็นโปรแกรมแสดงขอบเขตของตัวแปรส่วนกลาง, ตัวแปรเฉพาะที่ และตัวแปรสถิต

```
1   /*
2      A scooping example */
3   #include <stdio.h>
4
5   void useLocal ( void ) ;         /* function prototype */
6   void useStaticLocal ( void ) ;   /* function prototype */
7   void useGlobal ( void ) ;        /* function prototype */
8
9   int  x = 1 ;  /* global variable */
10
11  /* function main begins program execution */
12  int main ()
13  {
14      int  x = 5 ;   /* local variable to main */
15
16      printf ( “local x in outer scope of main is %d\n”,  x ) ;
17
18      {   /* start new scope */
19          int  x = 7 ;   /* local variable to new scope */
20
21          printf ( “local x in inner scope of main is %d\n”,  x ) ;
22      }   /* end new scope */
23
24      printf ( “local x in outer scope of main is %d\n”,  x ) ;
25
26      useLocal () ;                /* useLocal has automatic local x */
27      useStaticLocal () ;          /* useStaticLocal has static local x */
28      useGlobal () ;               /* useGlobal uses global x */
29      useLocal () ;                /* useLocal reinitializes automatic local x */
23      useStaticLocal () ;          /* static local x retains its prior value */
31      useGlobal () ;               /* global x also retains its value */
32
33      printf ( “\nlocal x in main is %d\n”,  x ) ;
34
35      return 0 ;   /* indicates successful termination */
36
37  }  /* end main */
38
39  /* useLocal reinitializes local variable x during each call */
40  void useLocal (void)
41  {
42      int  x = 25 ;   /* initialized each time useLocal is called */
43
44      printf ( “\nlocal x in useLocal is %d after entering useLocal\n”,  x ) ;
45      x+ + ;
46      printf ( “local x in useLocal is %d before exiting useLocal\n”,  x ) ;
47  }  /* end function useLocal */
```

```
48
49  /* useStaticLocal initializes static local variable x only the first time
50     the function is called ;  value of x is saved between calls to this
51     function */
52  void useStaticLocal (void)
53  {
54      /* initialized only first time useStaticLocal is called */
55      static int  x = 50 ;
56
57      printf ( "\nlocal static x is %d on entering useStaticLocal\n",  x ) ;
58      x+ + ;
59      printf ( "local static x is %d on exiting useStaticLocal\n",  x ) ;
60  }  /* end function useStaticLocal */
61
62  /* function useGlobal modifies global variable x during each call */
63  void useGlobal (void)
64  {
65      printf ( "\nglobal x is %d on entering useGlobal\n",  x ) ;
66      x  * = 10 ;
67      printf ( "global x is %d on exiting useGlobal\n",  x ) ;
68  }  /* end function useGlobal */
```

```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5

local x in a is 25 after entering useLocal
local x in a is 26 before exiting useLocal

local static x is 50 on entering useStaticLocal
local static x is 51 on exiting useStaticLocal

global x is 1 on entering useGlobal
global x is 10 on exiting useGlobal

local x in a is 25 after entering useLocal
local x in a is 26 before exiting useLocal

local static x 51 on entering useStaticLocal
local static x 52 on exiting useStaticLocal

global x is 10 on entering useGlobal
global x is 100 on exiting useGlobal

local x in main is 5
```

จากโปรแกรมนี้ ตัวแปรส่วนกลาง x ถูกนิยามและกำหนดค่าเริ่มต้นเป็น 1 (ในบรรทัดที่ 9) ตัวแปรส่วนกลาง x นี้ จะถูกปิดบังใน block ใด ๆ หรือฟังก์ชันใด ๆ ที่มีการนิยามชื่อตัวแปร x เหมือนกับตัวแปรส่วนกลาง

ในฟังก์ชัน main () จะมีการนิยามตัวแปรเฉพาะที่ x และกำหนดค่าเริ่มต้นเป็น 5 (ในบรรทัดที่ 14) ตัวแปรเฉพาะที่ x จะถูกพิมพ์ ส่วนตัวแปรส่วนกลาง x ในฟังก์ชัน main () จะถูกปิดบัง

ต่อไป block ใหม่ ในฟังก์ชัน main () จะมีการกำหนดค่าเริ่มต้นให้กับตัวแปรเฉพาะที่ x และกำหนดค่าเริ่มต้นเป็น 7 (ในบรรทัดที่ 19) ตัวแปรนี้จะถูกนำมาพิมพ์ และตัวแปร x นอก block ของฟังก์ชัน main () จะถูกปิดบัง ตัวแปรเฉพาะที่ x มีค่าเท่ากับ 9 จะถูกทำลายเมื่อออกจาก block และตัวแปรเฉพาะที่ x ที่อยู่นอก block แต่อยู่ในฟังก์ชัน main () จะถูกพิมพ์อีกครั้ง และนำมาแสดง ในโปรแกรมมีการนิยมฟังก์ชันขึ้นมา 3 ฟังก์ชัน โดยไม่มีอาร์กิวเมนต์ และไม่มีการส่งค่ากลับ โดยฟังก์ชัน useLocal จะนิยามตัวแปรเฉพาะที่ x และกำหนดค่าเริ่มต้นเป็น 25 (ในบรรทัดที่ 42) เมื่อฟังก์ชัน useLocal ถูกเรียกใช้ ตัวแปรก็จะถูกนำมาพิมพ์, เพิ่มค่า และนำมาพิมพ์อีกครั้ง ก่อนออกจากฟังก์ชัน แต่ละครั้งที่ฟังก์ชัน useLocal ถูกเรียกใช้ ตัวแปรอัตโนมัติ x ก็จะถูกกำหนดค่าเริ่มต้นเป็น 25 อีกครั้ง

ฟังก์ชัน useStaticLocal จะนิยามตัวแปรสถิต x และกำหนดค่าเริ่มต้นเป็น 50 (ในบรรทัดที่ 55) ตัวแปรเฉพาะที่ที่ถูกประกาศเป็นตัวแปรสถิตก็ยังคงจำค่าอยู่ ถึงแม้ว่าจะออกจากฟังก์ชันนี้ เมื่อฟังก์ชัน useStaticLocal ถูกเรียกใช้อีกครั้ง ค่าของ x จะถูกพิมพ์ออกมา, เพิ่มค่าขึ้น 1 และนำมาพิมพ์อีกครั้งก่อนออกจากฟังก์ชัน ถ้ามีการเรียกใช้ฟังก์ชันนี้อีกครั้ง ตัวแปรสถิต x ก็ยังคงมีค่าเท่ากับ 51 ฟังก์ชัน useGlobal ไม่ได้นิยามตัวแปรใด ๆ เลย

ดังนั้น เมื่อมีการอ้างถึงตัวแปร x ซึ่งเป็นตัวแปรส่วนกลาง (ในบรรทัดที่ 9) เมื่อฟังก์ชัน useGlobal นี้ถูกเรียกใช้ ตัวแปรส่วนกลางจะถูกนำพิมพ์ นำมาคูณด้วย 10 และพิมพ์ค่า x อีกครั้ง ก่อนออกจากฟังก์ชัน ต่อไปเมื่อฟังก์ชัน useGlobal ถูกเรียกใช้ ตัวแปรส่วนกลางยังคงมีการเปลี่ยนแปลงค่าโดยการนำ 10 มาคูณ สุดท้ายโปรแกรมก็จะพิมพ์ค่าตัวแปรเฉพาะที่ x ในฟังก์ชัน main () (บรรทัดที่ 33) อีกครั้ง

ตัวอย่างโปรแกรม 7.20 เป็นโปรแกรมแสดงตารางค่าความจริง

```
#include <stdio.h>

void eval (int  a, int  b) ;

main ()
{
    printf ( “Truth Table\n\n” ) ;
    printf ( “    A    B    F\n\n” ) ;
    eval (0, 0) ;
    eval (0, 1) ;
    eval (1, 0) ;
    eval (1, 1) ;
}

void eval (int  a, int  b)
{
    int  f ;      /* Used in Boolean computation.   */
    f = (a  &&  b) || (a  &&  ! b) ;
    printf ( “    %i   %i   %i \n”, a, b, f ) ;
}
```

```
Truth Table

 A   B   F

 0   0   0
 0   1   0
 1   0   1
 1   1   1
```

ตัวอย่างโปรแกรมที่ 7.21 เป็นโปรแกรมที่แปลงเลขฐานสิบเป็นเลขฐานสอง

```
#include <stdio.h>

void tobin (int  n) ;

main ()
{
    int  n ;      /* User input number.  */

    printf ( "Enter an integer from 1 to 255 = >  " ) ;
    scanf ( "%i", &n ) ;
    tobin (n) ;
}

void tobin (int  n)
{
    int  m ;     /* Used in remainder computation.  */

    if  (n  ! = 0)
    {
        m = n  /  2 ;
        tobin (m) ;
        printf ( "%i    ", n – 2 * m ) ;
    }
}
```

```
Enter an integer from 1 to 255 = > 100
1   1   0   0   1   0   0
```

ตัวอย่างโปรแกรมที่ 7.22 เป็นโปรแกรมที่แปลงเลขฐานสิบเป็นเลขฐานสิบหก

```
#include <stdio.h>

void tohex (int  n) ;

main ()
{
    int  n ;          /* User input number.   */
    int  a, b ;       /* Temporary storage.   */

    printf ( "Enter an integer from 0 to 255 = >   " ) ;
    scanf ( "%i", &n ) ;
    a = n  /  16 ;          /* Get upper 4–bit value.   */
    b = n % 16 ;            /* Get lower 4–bit value.   */
    tohex (a) ;
    tohex (b) ;
}

void tohex (int  n)
{
    if  ( (n > = 0)  &&  (n < = 9) )
        printf ( "%i", n ) ;
    else
    {
        switch (n)
        {
            case 10  :  printf ("A") ;  break ;
            case 11  :  printf ("B") ;  break ;
            case 12  :  printf ("C") ;  break ;
            case 13  :  printf ("D") ;  break ;
            case 14  :  printf ("E") ;  break ;
            case 15  :  printf ("F") ;  break ;
            default   :  printf ("Error ! \n") ;
        }
    }
}
```

```
Enter an integer from 0 to 255 = > 100
64
```

```
Enter an integer from 0 to 255 = > 200
C8
```

ตัวอย่างโปรแกรมที่ 7.23 เป็นโปรแกรมแสดงการหา ห.ร.ม. ของเลขจำนวนเต็ม 2 จำนวน โดยวิธียุคลิด

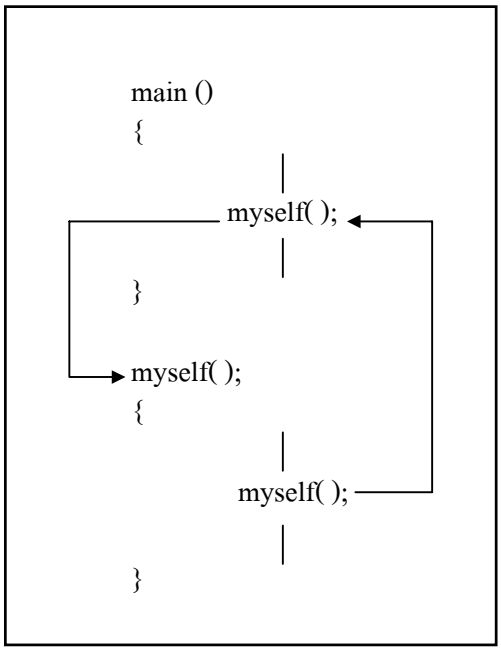
```
#include <stdio.h>

int euclid (int  a,  int  b) ;

main ()
{
    int  m ;     /* First user input number.       */
    int  n ;     /* Second user input number.   */

    printf ( “Enter the first number = >   ” ) ;
    scanf ( “%i”, &m ) ;
    printf ( “Enter the second number = >   ” ) ;
    scanf ( “%i”, &n ) ;
    printf ( “\nThe GCD of %i  and %i  is  %i”, m, n, euclid (m, n) ) ;
}

int euclid (int  a,  int  b)
{
    if  (b = = 0)
        return  a ;
    else
        return euclid (b, a % b) ;
}
```

```
Enter the first number = >  40
Enter the second number = >  100

The GCD of 40 and 100 is 20
```

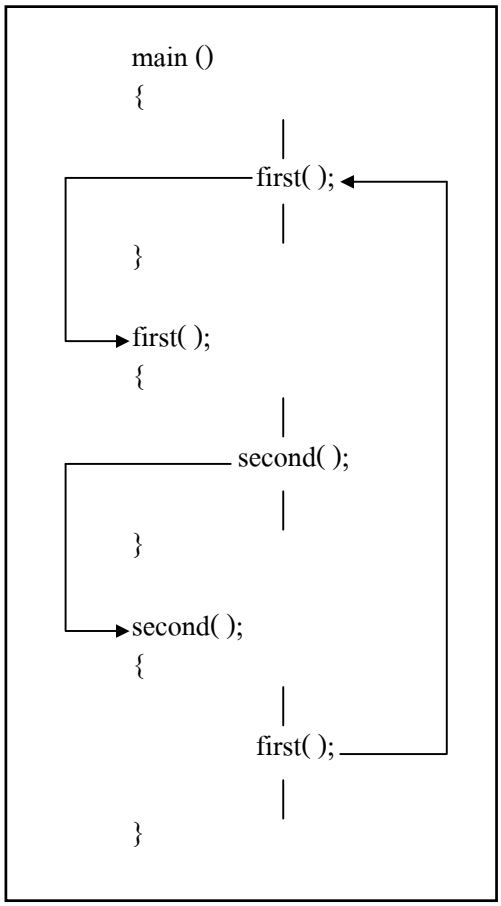
7.12 ฟังก์ชันเรียกซ้ำ (recursive function)

ฟังก์ชันเรียกซ้ำ เป็นฟังก์ชันที่มีการเรียกใช้ฟังก์ชันตัวมันเอง ซึ่งแบ่งออกเป็น 2 แบบ คือ

1. แบบโดยตรง (Direct)
2. แบบโดยอ้อม (Indirect)



a) แบบโดยตรง (Direct)



b) แบบโดยอ้อม (Indirect)

รูปที่ 7.10

จากรูปที่ 7.10 a) เป็นการเรียกซ้ำแบบโดยตรง โดยฟังก์ชัน myself( ) จะเรียกตัวมันเองแบบโดยตรง
จากรูปที่ 7.10 b) เป็นการเรียกซ้ำแบบโดยอ้อม โดยฟังก์ชัน first( ) และ second( ) จะมีการเรียกซ้ำเป็นวง (cycle)

ในแต่ละครั้งที่ฟังก์ชันเรียกซ้ำเรียกใช้ฟังก์ชันตัวมันเอง จะมีการสำเนาข้อมูลลงในหน่วยความจำที่เรียกว่า สแตก (stack) ซึ่งเป็นโครงสร้างข้อมูลที่ใช้สำหรับเก็บตัวแปร และค่าพารามิเตอร์ของแต่ละฟังก์ชันในโปรแกรม

ตัวอย่างโปรแกรมที่ 7.24 เป็นโปรแกรมที่แสดงการคำนวณหาค่า แฟคทอเรียล ของจำนวนเต็มบวก

```
#include <stdio.h>

int  fact (int) ;

int  main (void)  {
    int  number ;

    printf ( "Enter a nonnegative integer : " ) ;
    scanf ( "%d",  &number ) ;

    printf ( "Factorial of  %d  is  %d . \n",  number,  fact (number) ) ;
    return 0 ;
}  /* end function main */

int  fact (int  num)  {
    int  res ;

    printf ( "- - >  Function call :  fact (%d) \n",  num ) ;

    if  (num = = 0)
        res = 1 ;
    else
        res = num * fact (num – 1) ;
    /* end if */

    printf ( "<- -  %d  returned for fact (%d) . \n",  res,  num ) ;

    return res ;
}  /* end function fact */
```

```
Enter a nonnegative integer :  4
--->  Function  call :  fact (4)
--->  Function  call :  fact (3)
--->  Function  call :  fact (2)
--->  Function  call :  fact (1)
--->  Function  call :  fact (0)
<---  1  returned  for  fact (0) .
<---  1  returned  for  fact (1) .
<---  2  returned  for  fact (2) .
<---  6  returned  for  fact (3) .
<---  24  returned  for  fact (4) .
Factorial  of  4  is  24.
```

โปรแกรมนี้สามารถแสดงขั้นตอนการคำนวณหาค่าของ 4 ! ได้ดังนี้

ขั้นตอนที่ 1 เมื่อมีการเรียกใช้ฟังก์ชัน fact ( ) ก็จะส่งการควบคุมโปรแกรมไปยังฟังก์ชันที่ถูกเรียก โดยการทำสำเนาค่า actual parameter คือ 4 ไปยัง num ซึ่งเป็น formal parameter จึงทำให้ขณะนี้ num มีค่าเท่ากับ 4 เนื่องจาก num $\neq$ 0 ในข้อความสั่ง if ส่วนที่เป็นเท็จก็จะถูกกระทำการ นั่นคือ มีการเรียกใช้ฟังก์ชัน fact (num – 1) เนื่องจาก num = 4 ดังนั้น ฟังก์ชันถูกเรียกจึงเป็น fact (3) เนื่องจากยังไม่ทราบค่าของ fact (3) จึงยังไม่มีการคืนค่าไปยังฟังก์ชัน fact (4) ดังนั้น ค่าของ fact (4) จะถูกเก็บลงในสแตก

ขั้นตอนที่ 2 เมื่อมีการเรียกใช้ฟังก์ชัน fact ( ) ก็จะส่งการควบคุมโปรแกรมไปยังฟังก์ชันที่ถูกเรียก เนื่องจากขณะนี้ค่าของ formal parameter คือ num มีค่าเท่ากับ 3 ซึ่งไม่เท่ากับศูนย์ ในข้อความสั่ง if ส่วนที่เป็นเท็จ ก็จะถูกกระทำการ นั่นคือ มีการเรียกใช้ฟังก์ชัน fact (num – 1) เนื่องจาก num = 3 ดังนั้น ฟังก์ชันถูกเรียกจึงเป็น fact (2) เนื่องจากยังไม่ทราบค่าของ fact (2) จึงยังไม่มีการคืนค่าไปยังฟังก์ชัน fact (3) ดังนั้นค่าของ fact (3) จะถูกเก็บลงในสแตก นั่นคือในสแตกจะมีรายการของ fact (3) และ fact (4)

ขั้นตอนที่ 3 ในทำนองเดียวกัน เมื่อมีการเรียกใช้ฟังก์ชัน fact ( ) ค่าของ fact (2) ก็ยังไม่ทราบค่า ดังนั้นค่าของ fact (2) จะถูกเก็บลงในสแตก นั่นคือในสแตกจะมีรายการของ fact (2), fact (3) และ fact (4)

ขั้นตอนที่ 4 ในทำนองเดียวกัน เมื่อมีการเรียกใช้ฟังก์ชัน fact ( ) ค่าของ fact (1) ก็ยังไม่ทราบค่า ดังนั้นค่าของ fact (1) จะถูกเก็บลงในสแตก นั่นคือในสแตกจะมีรายการของ fact (1), fact (2), fact (3) และ fact (4)

ขั้นตอนที่ 5 ในทำนองเดียวกัน เมื่อมีการเรียกใช้ฟังก์ชัน fact ( ) ขณะนี้ num = 0 ในข้อความสั่ง if ส่วนที่เป็นจริงจะถูกกระทำ นั่นคือทำข้อความสั่ง return1 ;

ขั้นตอนที่ 6 เมื่อสิ้นสุดการคำนวณหาค่า fact (0) ก็จะมีการส่งค่า 1 กลับมายังฟังก์ชัน fact (0) ซึ่งมีค่าเท่ากับ 1 และคำนวณนิพจน์ num * fact (num – 1) ได้ผลลัพธ์ 1 * 1 = 1

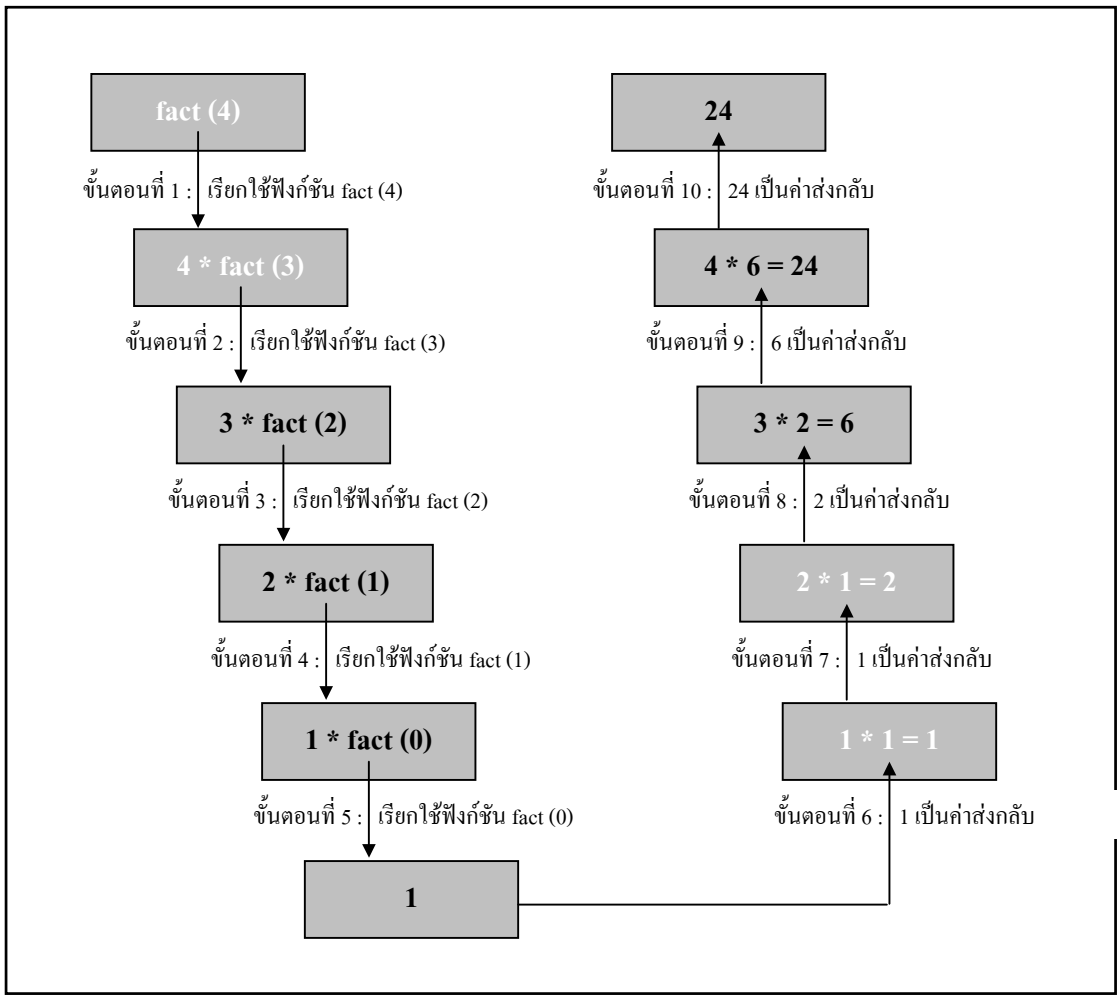
ขั้นตอนที่ 7 จะมีการส่งค่า 1 กลับมายังฟังก์ชัน fact (1) และคำนวณนิพจน์ num * fact (num – 1) ได้ผลลัพธ์ 2 * 1 = 2 และขณะนี้ในสแตกยังคงเหลือ fact (2), fact (3) และ fact (4)

ขั้นตอนที่ 8 จะมีการส่งค่า 2 กลับไปยังฟังก์ชัน fact (2) และคำนวณนิพจน์ num * fact (num – 1) ได้ผลลัพธ์ 3 * 2 = 6 และขณะนี้ในแสตกยังคงเหลือค่าของ fact (3) และ fact (4)

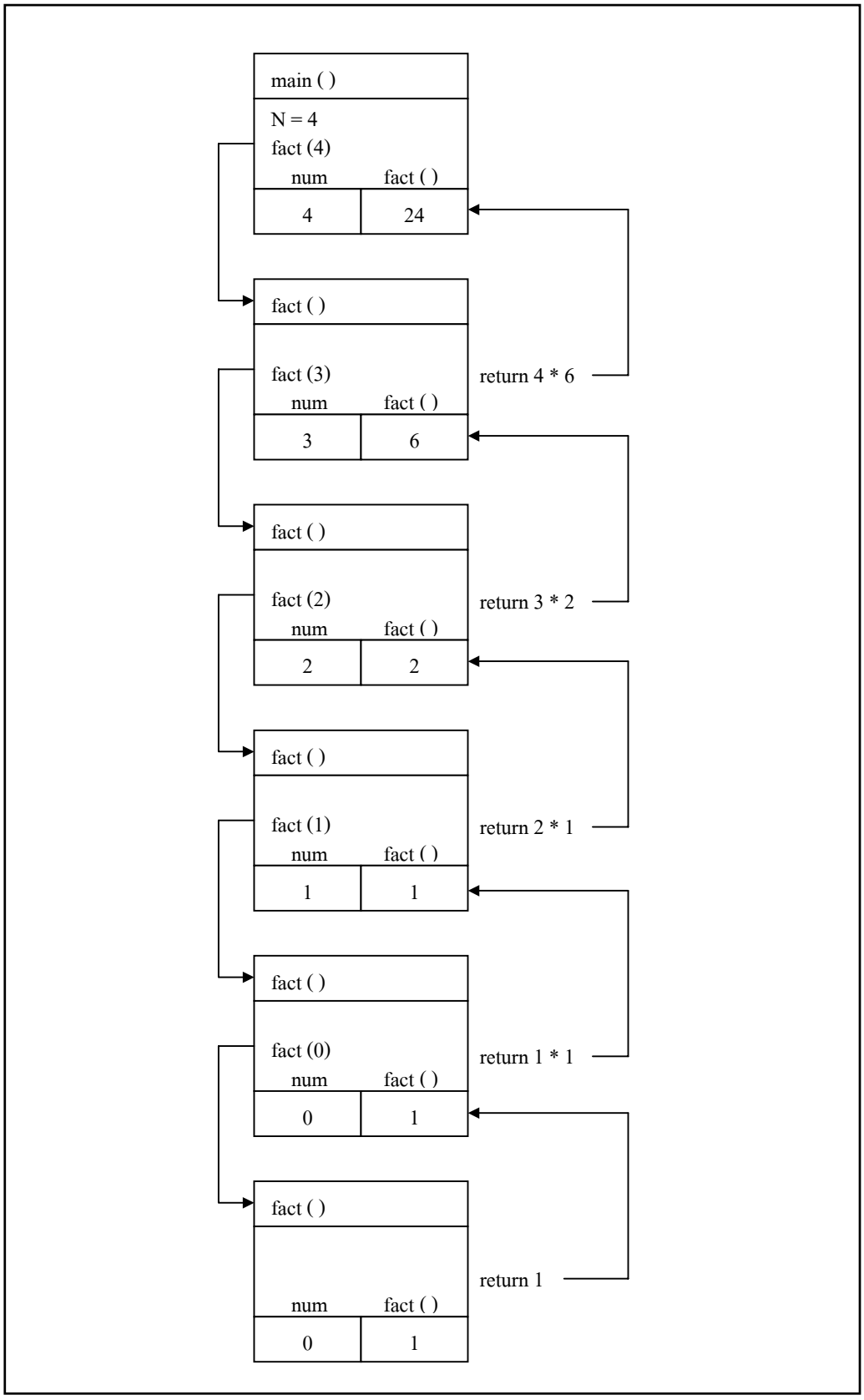
ขั้นตอนที่ 9 จะมีการส่งค่า 6 กลับไปยังฟังก์ชัน fact (3) และคำนวณนิพจน์ num * fact (num – 1) ได้ผลลัพธ์ 4 * 6 = 24 และขณะนี้ในสแตกยังคงเหลือค่าของ fact (4) เท่านั้น

ขั้นตอนที่ 10 จะมีการส่งค่า 24 กลับไปยังฟังก์ชัน fact (4) ดังนั้น fact (4) = 24

แสดงขั้นตอนได้ดังรูปที่ 7.11 และ 7.12



รูปที่ 7.11


main ( )
N = 4
fact (4)
num
fact ( )
4
24
fact ( )
fact (3)
num
fact ( )
3
6
return 4 * 6
fact ( )
fact (2)
num
fact ( )
2
2
return 3 * 2
fact ( )
fact (1)
num
fact ( )
1
1
return 2 * 1
fact ( )
fact (0)
num
fact ( )
0
1
return 1 * 1
fact ( )
num
fact ( )
0
1
return 1

รูปที่ 7.12

ตัวอย่างโปรแกรมที่ 7.15 เป็นโปรแกรมแสดงลำดับฟิโบนักชี (Fibonacci) โดยมีพจน์ที่ 1 มีค่าเท่ากับ 0, พจน์ที่ 2 มีค่าเท่ากับ 1, พจน์ที่ 3 เกิดจากผลบวกของพจน์ที่ 1 และพจน์ที่ 2 และพจน์ที่ n เกิดจากผลบวกของพจน์ที่ (n – 1) กับพจน์ที่ (n – 2) เช่น 0, 1, 1, 2, 3, 5, … โดยนิยามฟังก์ชันดังนี้

fibonacci(1) = 0

fibonacci(2) = 1

และ fibonacci(n) = fibonacci(n – 1) + fibonacci(n – 2)

```
#include <stdio.h>

int recursive_fib (int) ;

int main (void)  {
    int seq_number ;

    printf ( “Enter a positive integer :  ” ) ;
    scanf ( “%d”,  &seq_number ) ;

    printf ( “The  %dth  Fibonacci number is :  %d”,  seq_number,
        recursive_fib (seq_number) ) ;
    return 0 ;
}  /* end function main */

int recursive_fib (int sequence_number)  {
    if  (sequence_number = = 1)
        return 0 ;
    else if (sequence_number = = 2)
        return 1 ;
    else return (recursive_fib (sequence_number – 1) +
        recursive_fib (sequence_number – 2) ) ;
    /* end if */
}  /* end function recursive_fib */
```