บทที่ 4
ฟังก์ชัน

4.1 การสร้างฟังก์ชั่น

เนื่องจากภาษา C เป็นภาษาที่ใช้หลักเกณฑ์ของโมดูล (modular - opproach) เป็นหลักในการพัฒนาโปรแกรม โมดูลที่กล่าวถึงคือฟังก์ชั่นซึ่งมี 2 ชนิดคือ ฟังก์ชันที่เราเขียนขึ้นเองเรียกว่า definition function และด้วยเหตุที่ฟังก์ชันแบบนี้เราสามารถเรียกมาใช้งานได้เสมอโดยไม่ต้องเขียนใหม่ เราจึงเรียกฟังก์ชันชนิดนี้ว่า personal library กับฟังก์ชันอีกชนิดหนึ่งซึ่งจัดเตรียมไว้แล้วในคอมไพเลอร์ เรียกว่า compiler library หรือ standard library หน้าที่ของผู้เขียนโปรแกรมในภาษา C ก็คือการเรียกฟังก์ชั่นใน compiler library มาใช้งานตามต้องการโดยเชื่อม-

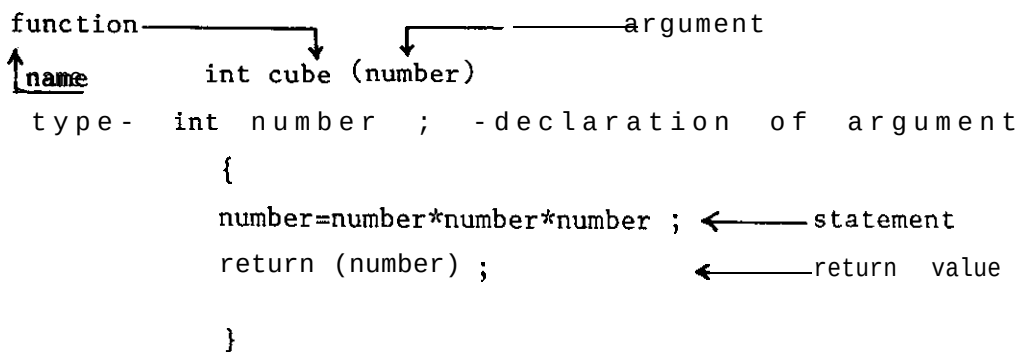
โยงกันด้วยอาร์กิวเมนต์หรือแอดเดรส (ดูเรื่อง pointer ในบทที่ 6) หากไม่มีฟังก์ชั่นใน compiler library ให้ใช้อย่างเพียงพอก็ให้สร้างฟังก์ชั่นขึ้นใช้เอง

รูปไวยากรณ์ (syntax) ของฟังก์ชั่นปรากฏดังนี้

```
type name (argument list)
     argument declaration ;
     }
     declarations of variables ;
     statement (s) ;
     }
```

type หมายถึงชนิดของฟังก์ชั่น ที่จริงแล้วเมื่อพูดถึงฟังก์ชั่นเราควรนึกถึงโปรแกรมย่อยที่เมื่อเราจะใช้โปรแกรมนั้นเราจะเรียก (call) ไปโดยส่งค่าข้อมูลหรือข้อสนเทศ (information) ไปให้ฟังก์ชั่นทำเมื่อเสร็จก็ขอให้ส่งผลลัพธ์คืนกลับไป โดยมากผู้เรียกมาคือ main ( ) ผู้ส่งผลลัพธ์ไม่ให้ main ( )หรือใครก็ได้ที่เรียกมาคือฟังก์ชั่นด้วยเหตุนี้ type ในที่นี้ จึงหมายถึง type ของค่าส่งกลับ (return value) หรือผลลัพธ์ที่ฟังก์ชั่นส่งกลับไปให้ผู้เรียก และข้อมูลหรือข้อสนเทศที่ส่งเข้ามาให้ฟังก์ชั่นทำก็คือ อาร์กิวเมนต์ ถ้าหากสั่งให้ฟังก์ชั่นก็ทำงานที่ต้องใช้อุปกรณ์มากผู้สั่งก็ต้องส่งอุปกรณ์มามาก เสมือนส่งอาร์กิวเมนต์มามาก เรียกว่า argument list ขอให้สังเกตว่ารูปไวยากรณ์ของฟังก์ชั่นไม่มีคำว่า return แสดงว่าเราจะมีคำว่า return หรือไม่ก็ได้ เพราะวงเล็บปีกกาปิดคือ } จะเป็นเครื่องหมายแสดงว่าจบการทำงาน และส่งค่าผลลัพธ์คืนไปยังจุดที่เรียกมาเองโดยอัตโนมัติ

ลองพิจารณาตัวอย่างต่อไปนี้



จากตัวอย่างจะพบว่าฟังก์ชันนี้ชื่อ cube เป็นฟังก์ชันที่รับค่าอาร์กิวเมนต์ ชื่อว่า number จากภายนอกมาทำงานตามความต้องการของผู้ส่ง (call) ซึ่งมักจะเป็น main ( ) การทำงานจะทำโดย นำค่าอาร์กิวเมนต์มายกกำลังสาม เมื่อทำเสร็จก็ส่งอาร์กิวเมนต์ชื่อ-number ซึ่งถูกแทนที่ (assign) ด้วย number*number*number ไปยังผู้เรียกหรือผู้สั่ง ค่าที่ส่งคืนจะต้องเป็น int ตาม type ที่ระบุ ฟังก์ชันนี้มี return ถ้าเราไม่ต้องการให้มีreturn คือตัดคำสั่ง return (number);ทิ้งไป ฟังชัน cube ก็ยังคงส่งค่าชื่อ number คืนไปให้ผู้เรียกอยู่เช่นเดิม

จากรูปไวยากรณ์และตัวอย่างข้างต้นจะเห็นว่าเรามีสิ่งที่ต้องการกระทำหรือรู้จักเกี่ยวกับฟังก์ชันดังนี้

1) type หรือ type specifier

type หรือ type specifier เป็นเครื่องบ่งชี้ชนิดของข้อมูลที่จะส่งคืนออกไปจากฟังก์ชัน โดยปกติคอมไพเลอร์จะถือว่าข้อมูลทุกชนิดที่ส่งออกไป (return value) เป็น int เสมอ (เรียกว่า int คือ default ของค่าส่งกลับ) หากเราต้องการให้ฟังก์ชันส่งค่ากลับคืนในแบบอื่น ๆ เช่นแบบ char, single, double, short, long, float ก็ให้ระบุไปตามความประสงค์ อย่างไรก็ตาม เราสมควรระบุชนิดของ

ค่าส่งกลับให้ชัดเจนอย่าคอยให้เป็นภาระของ default แม้ว่าค่าส่งกลับจะเป็น int ก็ตาม เพราะการไม่ระบุชนิดของค่าส่งกลับให้ชัดเจน ฟังก์ชันอาจส่งค่าไปผิด ๆ เช่นส่งเลข 0 กลับไปก็ได้

2) **name**

name หมายถึงชื่อของฟังก์ชั่น การตั้งชื่อของฟังก์ชันนี้เรายังคงใช้กฎเกณฑ์เดียวกันกับการตั้งชื่อตัวแปรคือ จะต้องเริ่มต้นด้วยตัวอักษรหรือขีด จากนั้นจึงเป็นตัวเลขหรืออักขระใด ๆ ที่เราชอบหรือสื่อความหมายได้ดีในสายงานของเรา และควรตั้งชื่อให้ยาวไม่เกิน 8 อักขระแต่จะยาวกว่า 8 อักขระก็ได้ซึ่งก็มีผลหรือมีความหมายต่อคอมไพเลอร์เพียง 8 อักขระแรก และเพื่อให้คอมไพเลอร์พบข้อแตกต่างระหว่างชื่อตัวแปรและชื่อฟังก์ชั่นมิให้ส่งค่า-รับค่าผิด เราจึงใช้วงเล็บเปิด-ปิดต่อท้ายชื่อฟังก์ชัน สิ่งที่อยู่ในวงเล็บคือ อาร์กิวเมนต์หรือข่าวสารที่จะส่งเข้ามาจากภายนอก หากฟังก์ชันใดไม่จำเป็นต้องรับข่าวสารจากภายนอกก็ไม่ต้องใส่ชื่ออาร์กิวเมนต์ เมื่อมีเครื่องจำแนกเช่นนี้คอมไพเลอร์ก็จะไม่เกิดความสับสน เช่น เมื่อเขียน main ก็ทราบว่านี่คือชื่อตัวแปร แต่ถ้าเขียนเป็น main ( ) ก็จะทราบว่านี่คือชื่อฟังก์ชั่น เขียนว่า nothing ก็จะทราบว่านี่คือชื่อตัวแปร ถ้าเขียนเป็น nothing ( ) ก็จะทราบว่านี่คือชื่อฟังก์ชั่น หรือเขียนว่า double faddone ก็จะทราบว่านี่คือตัวแปรชื่อ faddone เป็นตัวแปรชนิด double แต่ถ้าเป็น double faddone (fin) ก็จะทราบว่านี่คือ ฟังก์ชั่นชื่อ faddone ที่มีอาร์-กิวเมนต์ชื่อ fin และฟังก์ชั่นนี้จะส่งค่ากลับคืนไปในแบบของ double เป็นต้น

ขอให้สังเกตรูปไวยากรณ์ว่าเราไม่มีเครื่องหมายอัฒภาค (;) ท้ายชื่อฟังก์-ชัน ถ้าเราใส่เครื่องหมายนี้เครื่องจะแจ้งให้ทราบว่าผิดไวยากรณ์ (syntax error)

3) argument list

อาร์กิวเมนต์ก็คือ ชื่อตัวแปรที่นำเอาข้อมูลข้อสนเทศ จากภายนอกเข้ามาในฟังก์ชันเพื่อฟังก์ชันจะได้ทำงานตามที่สั่งมาเฉพาะคราวเฉพาะงานได้ argument list เป็นชื่อรวม ๆ หมายถึงตัวแปรหมู่หนึ่งหรือกลุ่มหนึ่ง ซึ่งอาจประกอบด้วยตัวแปร 1, 2, 3, ... หรือกี่ตัวก็ได้หรือแม้แต่ไม่มีเลยก็ได้เช่น main ( ) หรือ nothing ( ) หรือ inv ( ) เป็นต้น ทั้งนี้ขึ้นอยู่กับตัวฟังก์ชันก์เองว่ามีลักษณะที่เพรียบพร้อมเพียงใด ฟังก์ชันใดต้องร้องขอ "สิ่งของ" (หมายถึงข้อมูลข้อสนเทศ) จากภายนอกมากก็มีอาร์กิวเมนต์มาก ฟังก์ชันใดสมบูรณ์บริบูรณ์แล้วไม่ต้องให้ใครหยิบยื่นสิ่งของมาให้จึงจะทำงานให้ได้ก็จะมีอาร์กิวเมนต์น้อยหรือไม่ต้องมีเลย

4) argument declaration

หมายถึงการกำหนดลักษณะหรือชนิดของอาร์กิวเมนต์ การกำหนดลักษณะของอาร์กิวเมนต์ก็ด้วยวัตถุประสงค์ที่จะแจ้งให้ฟังก์ชันเทราบว่ากำลังรับหรือจะรับข้อมูลแบบใดที่ส่งหรือ หยิบยื่นเข้ามาจากภายนอกเราควรกำหนดลักษณะตัวแปร หรืออาร์กิวเมนต์ให้เป็นนิสัย

โดยกำหนดไว้ก่อนเขียนตัวฟังก์ชัน (function body) คือกำหนดไว้ก่อนวงเล็บปีกกาเปิด เพื่อแสดงให้ทราบว่าสิ่งที่มีลักษณะต่าง ๆ เหล่านั้นคือสิ่งที่หยิบยื่นหรือส่งมาจากภายนอก เพื่อนำมาเป็นวัตถุดิบเพื่อผลิตออกมาโดยสิ่งที่อยู่ภายในวงเล็บปีกกา (คือตัวฟังก์ชัน) แล้วส่งคืนไปให้ตามที่ขอมาหรือสั่งมา การไม่กำหนดลักษณะตัวแปรหรืออาร์กิวเมนต์ไว้ก่อนให้ชัดเจน คอมไพเลอร์จะไม่ทราบและไม่ทราบว่าจะอ้างอิงหรือเกี่ยวข้องกับตัวแปรหรืออาร์กิวเมนต์นั้น อย่างไร ทั้งไม่ทราบชนิดของตัวแปรและไม่ทราบว่าจะหาแอดเดรสของตัวแปรหรืออาร์กิวเมนต์นั้นได้อย่างไรด้วย

พิจารณาตัวอย่างต่อไปนี้

```
addone (innum)
int innum ;
{
int outnum ;
outnum = innum + 1 ;
return outnum ;
 }
```

แสดงว่าสิ่งที่ส่งเข้ามาจากภายนอกคือตัวแปร (อาร์กิวเมนต์) ชื่อ innum เป็นตัวแปรแบบ int เมื่อเราส่ง innum เข้าไปในตัวฟังก์ชัน การผลิตหรือทำงานจะเริ่มขึ้นโดยค่าของ innum + 1 จะถูกแทนลงในที่ของตัวแปรชื่อ outnum แล้วฟังก์ชันก็จะส่งค่าตัวแปรชื่อ outnum ไปให้ผู้สั่งหรือผู้เรียก

หรือในตัวอย่างฟังก์ชันชื่อ cube และ valume คือ

```
int  cube  (number)
int  number  ;
 {
number = number * number * number
return  number  ;
  }
```

และ

```
int volume (l, h,  w)
int l, h,  w ;
 {
volume = l*h*w ;
return  volume  ;
  }
```

จะสังเกตได้ว่าเราไม่เขียนเครื่องหมายอัฒภาค (;) หลังชื่อฟังก์ชันและจะต้องเขียนเครื่องหมายอัฒภาคหลังชื่อตัวแปร (อาร์กิวเมนต์) และหลังคำสั่งทุกบรรทัด ขอให้สังเกตว่าฟังก์ชันทั้ง 3 นี้ต้องคอยรับข่าวสารหรือสิ่งที่ต้องหยิบยื่นส่งเข้ามาจากภายนอกจึงจะทำงานได้ ที่ท้ายฟังก์ชั่นยังมีคำสั่ง return ลองดูตัวอย่างต่อไปนี้เปรียบเทียบกัน

```
pause _ l( )
 {
   int c ;
   c = getchar ( ) ;
   return ;
  }
```

และ

```
pause_ 2.( )
  {
   int c ;
   c = getchar ( ) ;
   }
```

ฟังก์ชั่น pause_1 ( ) และ pause 2 ( ) คือฟังก์ชั่นที่สั่งคอมพิวเตอร์ให้หยุดทำงานเพื่อคอยให้ผู้ใช้กดอักขระอะไรก็ได้บนแป้นพิมพ์ถ้าประสงค์จะทำงานต่อ (ฟังก์ชั่น getchar ( ) เป็นฟังก์ชั่นใน compiler library ที่ใช้สำหรับรับอักขระ(เพียงอักขระเดียว) เข้าทางแป้นพิมพ์) จะเห็นว่าเป็นฟังก์ชั่นที่ไม่ต้องการสิ่งใดจากภายนอกคือไม่มีอาร์กิว-เมนต์ อนึ่งฟังก์ชั่น pause-1 ( ) และ pause-2 ( ) เป็นฟังก์ชั่นที่เรียกเอาฟังก์ชั่น-getchar ( ) มาใช้ ขอให้สังเกตรูปไวยากรณ์ c = getchar ( ) การเรียกใช้แบบนี้จึงเป็นการเรียกจากภายในฟังก์ชั่น เราจะได้ศึกษาเรื่องนี้อีกครั้ง ในบทที่ 6 ที่ว่าด้วย pointer และจะเห็นได้อีกว่าฟังก์ชั่น pause_1 ( ) มีคำสั่ง return ขณะที่ฟังก์ชั่น pause_2 ( ) ไม่มีคำสั่ง return แต่ทั้งสองฟังก์ชันก็ทำงานเหมือนกันคือสั่งให้เครื่องทำงานต่อไปเมื่อผู้ใช้กดแคร่อักขระใด ๆ บนแป้นพิมพ์ ทั้งนี้เพราะคอมไพเลอร์จะทราบว่าต้อง return ได้เองเมื่ออ่านพบวงเล็บปีกกาปิด

4.2 การเรียกฟังก์ชั่น

รูปไวยากรณ์ของการเรียกฟังก์ชั่นปรากฏดังนี้ ขอให้สังเกตว่าถ้าเราจะ-เรียกฟังก์ชั่นใดก็อ้างชื่อฟังก์ชันนั้นพร้อมค่าอาร์กิวเมนต์ (อาจเรียกว่า actual parameter)หรือไม่ต้องระบุค่าอาร์กิวเมนต์ก็ได้คือ

```
name (argument list) ;
```

หรือเราจะตั้งที่รับไว้ด้วยก็ได้ เช่น

```
faddone ( ) ;
double faddone ( ) ;
answer = addone (5) ;
answer = addone (4) + 5 ;
```

ฟังก์ชันเรียกนั้นโดยมากจะอยู่ใน main ( ) หรือในฟังก์ชันเรียกใด ๆ ก็ได้ ตามตัวอย่างข้างต้นเราเรียกให้ฟังก์ชัน faddone( )ให้ส่วนผลลัพธ์ที่ต้องการไปให้ โดยไม่ส่งอาร์กิวเมนต์มาให้ ส่วน double faddone ก็คือเราเรียกฟังก์ชัน faddone( )ให้ส่งค่าที่เป็น doubleมายัง main ( ) หรือโปรแกรมเรียก answer = addone (5) ; หมายถึงเราเรียกฟังก์ชัน addone( )โดยยื่นค่าอาร์กิวเมนต์เท่ากับ 5 ไปให้แล้วขอให้ฟังก์ชันaddone()ส่งผลลัพธ์ไปให้โดยใส่ผลลัพธ์นั้นลงในที่ชื่อ answer นั่นคือจากฟังก์ชัน addone ซึ่งมีรายละเอียดดังนี้

```
addone (innum)
int innum ;
{
int outnum ;
outnum = innum + 1 ;
return outnum ;

}
```

ถ้าโปรแกรม main ( ) เรียกมาเป็น addone(5) ; ฟังก์ชัน addone( ) จะรับค่าอาร์กิวเมนต์คือ 5 เข้ามาในฟังก์ชันทำหน้าที่เป็น innum ผลลัพธ์ที่ได้คือ outnum ซึ่งมีค่าเท่ากับ 6 ก็จะถูกส่งออกจากaddone()ไปยังmain( ) ถ้าใน main ( ) เรียกว่า answer = addone (5) ; ฟังก์ชันaddone()จะส่งค่าคือ 6 ไปให้โดยใส่ค่าเท่ากับ 6 ลงในที่ชื่อ answer หรือในกรณีที่เรียกมาเป็น answer = addone (4) + 5 ; จะเห็นว่าอาร์กิวเมนต์ที่ยื่นเข้ามาจากภายนอกฟังก์ชันaddone()คือ 4 ฟังก์ชันจะคำนวณค่าoutnum ได้ 5 เมื่อบวกเข้ากับ 5 เป็น 10 ก็ส่งไปเก็บใน main ( ) ในที่ชื่อanswerดังนี้เป็นต้น

ตัวอย่างฟังก์ชันในภาษาอื่นปรากฏดังนี้

BASIC

```
5  DEF FN AD(IN)=IN+1                          function definition
10 AN=FN AD(5)                                 function call
```

FORTRAN

```
INTEGER FUNCTION ADDONE(INNUM)                 function definition
INTEGER INNUM
ADDONE=INNUM+1
RETURN
END

ANSWER=ADDONE(5)                               function call
```

PASCAL

```
FUNCTION ADDONE(INNUM:INTEGER): INTEGER;       function definition
BEGIN
      ADDONE := INNUM+1
END;

ANSWER := ADDONE(5);                           function call
```

PL/I

```
ADDONE: PROCEDURE(INNUM) RETURNS               function definition
  (FIXED(15,0));
DECLARE INNUM FIXED(15,0);
DECLARE OUTNUM FIXED(15,0);
BEGIN;
       OUTNUM=INNUM+1;
       RETURN (OUTNUM);
END ADDONE;

ANSWER = ADDONE(5);                            function call
```

COBOL

Nothing equivalent

การเรียกฟังก์ชั่นเรากระทำได้ 2 วิธี

วิธีที่ 1 Call by value หมายถึงเรียกใช้ฟังก์ชั่นด้วยค่าอาร์กิวเมนต์ โดยโปรแกรม nain ( ) หรือฟังก์ชันเรียกหรือโปรแกรมเรียกจะส่งค่าอาร์กิวเมนต์ไปยังฟังก์ชัน เมื่อฟังก์ชันรับอาร์กิวเมนต์ก็จะนำค่าอาร์กิวเมนต์ไปใช้เพื่อผลิตสิ่งที่ผู้เรียกต้องการ ผลิตเสร็จที่ส่งคืนไปให้ หากผู้เรียกหรือ main ( ) ระบุที่เก็บของไว้ให้ฟังก์ชั่นก็จะส่งของไปเก็บไว้ในที่ ๆ ระบุ เช่นฟังก์ชันคือ

```
addone (innum)
int innum ;
{
innum+ = 1 ;
return innum ;
 }
```

ใน main ( ) หรือในโปรแกรมเรียกใด ๆ หากมีคำสั่งและเรียกดังนี้

```
int i = 5 ;
int j ;

j  = addone (i) ;
```

ซึ่งเป็นการเรียกให้ฟังก์ชั่น addone ทำงานโดยส่งอาร์กิวเมนต์คือ i ซึ่งเรากำหนดค่าเริ่มต้นให้ว่าเท่ากับ 5 ไปให้แล้วให้ส่งผลลัพธ์มาเก็บในที่ชื่อ j ดังนี้เป็นต้น

ขอให้สังเกตว่า innum+ = 1 ; เป็นรูปย่อ รูปเต็มคือ innum=innum + 1 (อ่านบทที่ 2 เรื่องตัวดำเนินการกำหนดค่า คือ f op = g)

วิธีที่ 2 call by reference หมายถึงการเรียกใช้ฟังก์ชั่นโดยmain ( ) หรือฟังก์ชั่นเรียก จะเรียกให้ฟังก์ชันทำงานโดยแจ้งแอดเดรสของตัวแปร (อาร์กิว-เมนต์ที่ส่งไปคือแอดเดรส) ให้ฟังก์ชันตามไปเอามูลค่าตามที่อยู่ (แอดเดรส) ที่บอกให้นั้นไปใช้ผลิตสิ่งที่ main ( ) ต้องการ

การเรียกแบบนี้จะต้องมี operator 2 ตัว ตัวหนึ่งคือ & operator หรือ address operator ทำหน้าที่ชี้แอดเดรสของอาร์กิวเมนต์และ * operator ใช้ทำหน้าที่รับค่าแอดเดรสไว้แล้วหยิบเอามูลค่าที่บรรจุในแอดเดรสนี้ไปใช้ประโยชน์หรือผลิตในสิ่งที่ main ( ) หรือโปรแกรมเรียกต้องการ (อ่านเรื่อง pointer ในบทที่ 6) โดยเราจะเติมโอเปอร์เตอร์ & และ * หน้าอากิวเมนต์ เช่นใน main ( ) หรือโปรแกรมเรียกเรียกใช้ฟังก์ชั่น increase( ) ดังนี้

```
int out = 5 ;

increase (& out) ;
```

โดยฟังก์ชัน increase มีลักษณะดังนี้คือ

```
increase (innum)
int * innum ;
    {
    (* innum) ++ ;
       return ;
    }
```

หมายความว่า main ( ) หรือโปรแกรมเรียกจะเรียกใช้ฟังก์ชั่นชื่อ increase ทำงานขยายค่าโดยแจ้งให้ increase ตามไปรับข้อมูล ณ ที่อยู่ (แอดเดรส) ของตัวแปรชื่อ out ฟังก์ชั่นชื่อ increase จะตามไปรับมูลค่าที่อยู่ในแอดเดรสนั้น (คือ 5) มาทำการ

ขยายค่า [(* innum)++; หมายถึง *innum = * innum+1;] แล้วส่งค่าคืนสู่ main ( ) หรือโปรแกรมเรียกต่อไป

ตัวอย่างฟังก์ชันเรียกในภาษาอื่นปรากฏดังนี้

BASIC

```
5  OU=5
10 GOSUB 50                                   subroutine call
        . . .
50 OU=OU+1                                    subroutine definition
60 RETURN
```

FORTRAN

```
SUBROUTINE INCREASE(INNUM)                    subroutine
INTEGER INNUM
INNUM=INNUM+1
RETURN
END

CALL INCREASE (OUT)                           subroutine call
```

PASCAL

```
PROCEDURE INCREASE(VAR INNUM:INTEGER);        subroutine
BEGIN
      INNUM := INNUM+1
END;

INCREASE(OUT);                                subroutine call
```

PL/I

```
INCREASE: PROCEDURE (INNUM);                  subroutine
DECLARE INNUM FIXED(15,0);
INNUM=INNUM+1;
RETURN;
END INCREASE;

CALL INCREASE(OUT);                           subroutine call
```

COBOL

```
IDENTIFICATION DIVISION.                      subroutine
PROG-ID.
        INCREASE.
        . . .
ENVIRONMENT DIVISION.
DATA DIVISION.
LINKAGE SECTION.
        . . .
01 INNUM PICTURE 99999.
        . . .
PROCEDURE DIVISION USING INNUM.
        COMPUTE INNUM = INNUM + 1.
        EXIT PROGRAM.
        . . . .

        CALL'INCREASE' USING OUT.             subroutine call
```

ตัวอย่างฟังก์ชั่นในภาษา C ปรากฏดังนี้ ฟังก์ชั่นเหล่านี้เป็นเพียงตัวอย่างส่วนน้อย ฟังก์ชั่นส่วนใหญ่ปรากฏใน compiler library ซึ่งจะกล่าวถึงต่อไปในเรื่อง I/O

1) ฟังก์ชั่น chklim เป็นฟังก์ชันที่ใช้ทดสอบว่าค่าที่สนใจปรากฏอยู่ในระหว่างค่า 2 ค่าที่กำหนดให้หรือไม่

```
chklim(num,low,high)
/*  returns 1 if num between low and high
    else returns 0 */
int num; /* number to check */
int low; /* low limit */
int high; /* high limit */
    {
    int ret=0;
    if (num>=low)
          {
          if (num<=high) ret=1;
          else ret=0;
          }
    else ret=0;
    return ret;
    }
```

This could be also written as:

```
chklim(num,low,high)
        /*  returns 1 if num is between low and high
            else returns 0 */
        int num; /* number to check */
        int low; /* low limit */
        int high; /* high limit */
            {
            return ( (num>=low) && (num<=high));
            }
```

2) ฟังก์ชัน power ใช้สำหรับยกกำลังจำนวนเต็มบวกให้แก่จำนวนใด ๆ ในรูป d^n

power

This routine raises a floating number to an integer power

```
double power(d,n)
/* raises number to an integer power */
double d; /* number to raise to power */
int n;      /* power to raise it to */
    {
    double ret;
    ret=1.;
    /* it power is negative, use 1/d */
    if (n<0)
        {
        n=-n;
        d=1./d;
        }
    while (n--)
        {
        ret *= d;
        }
    return ret;
    }
```

3) ฟังก์ชัน isdigit (chr) ใช้ส่งค่าที่ไม่เป็น 0 ถ้า chr เป็นตัวเลข และส่งค่าเท่ากับ 0 ถ้า chr ไม่เป็นตัวเลขไปยัง main ( ) หรือโปรแกรมเรียก

```
isdigit(chr)
/* returns 0  if chr is not a digit
     non-zero if chr is a digit */
int chr; /* character to test */
        {
        int ret;
        if ((chr>='0')&&(chr<='9')) ret=1;
        else ret = 0;
        return ret;
        }
```

or

```
isdigit(chr)
int chr;
     {
     return ((chr>='0')&&(chr<='9'));
     }
```

4) ฟังก์ชัน toupper (chr) ใช้เปลี่ยนค่าของ character constant ของตัวอักษรตัวเล็ก (a-z ซึ่งมีรหัสแอสกี้เท่ากับ 97-122) เป็น charactor constant ของตัวใหญ่ (A-Z ซึ่งมีรหัสแอสกี้เท่ากับ 65-90)

```
toupper(chr)
/* returns upper case character it chr is lower case, else returns chr */
int chr; /* character to test */
    {
    int sub;
    if ((chr> ='a')&&(chr< ='z')) sub ='a' – 'A';
    else sub=0;
    return   chr-sub;
    }


toupper(chr)
int chr;
    {
    return ( (chr> ='a') && (chr<='z') ?
          chr – ('a' -'A') : chr);
    }
```

4.3 การออกแบบโปรแกรมภาษา C

การเขียนโปรแกรมภาษา C นั้นเรากระทำได้โดย พียงแต่เชื่อมโยงฟังก์ชั่นเข้าด้วยกันด้วยอาร์กิวเมนต์หรือตัวแปรภายนอก (external variable) โดยต้องมีโปรแกรมหลักคือ main ( ) และจากโปรแกรมหลักให้เรียกฟังก์ชันต่าง ๆ มาใช้ตามความจำเป็น การร่างโครงโปรแกรมเรานิยมใช้ pseudocode [1]

ตัวอย่างเช่นเราจะเขียนโปรแกรมเพื่อคำนวณหาเลขยกกำลังสอง และเลขยกกำลังสามเราอาจกำหนดโครงโปรแกรม (outline) ได้ดังนี้คือ

1. รับเลขจำนวนเต็มที่จะยกกำลัง 2 และยกกำลัง 3
2. พิมพ์ค่าเลขจำนวนเต็มดังกล่าว
3. ยกกำลัง 2

ยกกำลัง 2 ได้หรือไม่ ? (หมายถึง overflow หรือไม่?)

(1) ถ้ายกกำลัง 2 ได้ให้ยกกำลัง 2

ส่งผลลัพธ์คืนให้แก่โปรแกรมเรียกหรือ main ( )

(2) ถ้าไม่สามารถยกกำลัง 2 ได้ให้บอกว่าเลขโตเกินไป

ส่ง 0 คืนให้แก่โปรแกรมเรียกหรือ main ( )

4. พิมพ์ผลลัพธ์คือเลขยกกำลัง 2
5. ยกกำลัง 3

ยกกำลัง 3 ได้หรือไม่? (หมายถึง overflow หรือไม่?)

(1) ถ้ายกกำลัง 3 ได้ ให้ยกกำลัง 3

ส่งผลลัพธ์คืนให้แก่โปรแกรมเรียกหรือ main ( )

(2) ถ้ายกกำลัง 3 ไม่ได้ให้บอกว่าเลขโตเกินไป

ส่ง 0 คืนให้แก่โปรแกรมเรียกหรือ main ( )

6. พิมพ์ผลลัพธ์คือเลขยกกำลัง 3
7. จบโปรแกรม

จากโครงโปรแกรมเราสามารถเขียนโครงโปรแกรมเป็น pseudocode ดังนี้

```
main ( )
 I
  declare  working  variable;
  initialize variable;
  print ( ) its original value ;
  call ( ) and assign square  ; /*need a square function*/
  print ( ) its squared value ;
  call ( ) and assign cube ; /* need a cube function */
  print ( ) its cubed value ;
 }
```

จาก pseudocode เราสามารถเขียนโปรแกรมได้ดังนี้

```
/*take aninteger and print its equare and cube*/
#define MAX-SQR 181 /*number>181 overflow square*/
#define MAX-CUBE 32 /*number>=32 overflow cube */
main ( )
  {
     int i, x ;
     x=3 ;
     printf ("\n the value being used is %d", x) ;
     i=sq (x) ;
     printf ("\n the value squared is %d", i) ;
     i=cube (i, x) ;
     printf ("\n the value cubed is %d", i) ;
   }
sq (i)
int i ;
   {
    if(i<=MAX-SQR) /*MAX=181 for squared integer */
       i=i*i ;
```

คำอธิบายประกอบ

หมายเหตุ

สร้าง symbolic constant ขึ้นใช้ในโปรแกรม

โปรแกรมหลัก

เริ่มต้นโปรแกรม

กำหนดให้ i และ x เป็น int

กำหนดค่าเริ่มต้นให้ x

สั่งพิมพ์ด้วยฟังก์ชัน printf ( )

เรียกฟังก์ชัน sq ( )

สั่งพิมพ์ด้วยฟังก์ชัน printf ( )

เรียกฟังก์ชัน cube ( )

สั่งพิมพ์ด้วยฟังก์ชัน printf ( )

จบโปรแกรม

ฟังก์ชัน sq ( )

กำหนดให้ i เป็น int

เริ่มฟังก์ชัน

คำสั่งย้ายตามเงื่อนไข

```
    else {                                                    if-else
       i=0
       printf ("\n number too large for integer square");  สั่งพิมพ์ข้อความด้วยฟังก์ชัน
         }                                                    printf ( )
                          จบเงื่อนไขคำสั่ง if-else
       return (i) ;
                          ส่งค่าของ i ไป main ( )
}
                          จบฟังก์ชัน sq ( )
cube (i, x)
                          ฟังก์ชัน cube อาร์กิวเมนต์คือ i และ x
int i, x ;
                          กำหนดให้ i และ x เป็น int
{
                          เริ่มฟังก์ชัน
    if(x<MAX-CUBE) /*MAX=32 for cubed integer */    คำสั่งย้ายตามเงื่อนไข
        i=i*x ;                                      if-else
    else{
        i=0 ;
        printf ("\n number too lai-ge for integer cube.")
    }                                    สั่งฟังก์ชันด้วยฟังก์ชัน printf ( )
    return (i) ;                         จบเงื่อนไขคำสั่ง  if-else
  }                                      ส่งค่า i คืน main ( )
                                         จบฟังก์ชัน cube ( )
```

อีกตัวอย่างหนึ่งเป็นโปรแกรมยกกำลังเช่นกัน เป็นโปรแกรมที่เรียกฟังก์ชัน power ในตอน 4.2 มาใช้ดังนี้

```
                                                    คำอธิบายเพิ่มเติม
main ( )
/* raise a number to a power */          โปรแกรมหลัก
   {                                      หมายเหตุ
    double power ( ) ;                    เริ่มโปรแกรมหลัก
    float fin ;                           เรียกฟังก์ชัน power ให้ส่งค่ามาให้ในรูป double
    int iin ;                             กำหนดให้ fin เป็น float
    float result ;                        กำหนดให้ iin เป็น int
                                          กำหนดให้ result เป็น float
```

```
printf ("\n input the float.") ; สั่งพิมพ์ด้วย prinf ( )
scanf ("%f", & fin) ; เรียกฟังก์ชัน scanf ( ) โดยส่งค่าที่แอดเดรสของ fin ไปให้
printf ("\n input the power.") ; สั่งพิมพ์ข้อความด้วย printf
scanf ("%d ", & iin) ; เรียกฟังก์ชัน scanf ( ) โดยส่งค่าที่แอดเดรสของ iin
result=power (fin, iin) ; ให้เรียกฟังก์ชัน power ให้ส่งผลลัพธ์มาเก็บใน result
printf ("\n result is % f", result) ; สั่งพิมพ์
exit (0)                                  end
}                                         จบโปรแกรมหลัก
double power (d, n)       ฟังก์ชัน power มีอาร์กิวเมนต์เป็น d และ n
                          และฟังก์ชันนี้จะส่งค่าให้ main ในรูป double
```

```
*/ raise number to a" integer power */      หมายเหตุ
double d; /*number raise to power */        กำหนดให้ฐานคือ d เป็น double
int n ; /*power to raise it to */           กำหนดให้กำลังเป็น int
    {                                       เริ่มฟังก์ชัน power
    double ret ;                            กำหนดให้ ret เป็น double
    ret=1 ;                                 กำหนดให้ค่าเริ่มต้นของ ret เท่ากับ 1
    /* if power is negative use l/d */      ถ้า n < 0
    if (n<0)                                บล็อคของ if
          {                                 แทน-n ใน n
          n=-n ;                            แทน 1/d ใน d
          d=1 /d ;                          จบบล็อค
          }
    while (n--)                             while n=n-1
          {                                 บล็อคของ while
          ret* = d ;                        ret = ret*d
          }                                 จบบล็อค
    return ret ;                            ส่งค่า retไป main ( )
    }                                       จบฟังก์ชัน
```

หมายเหตุ ฟังก์ชั่น scanf ( ) ใช้ทำหน้าที่ read หรือ input หรือ accept โดยปกติจะต้องระบุรูปแบบไว้ด้วย รูปไวยากรณ์ของ scanf ( ) ปรากฏดังนั้น

```
scanf ("control string", variable address . ..) ;
```

จะกล่าวถึงเรื่องนี้อีกครั้งในเรื่อง I/O ในบทที่ 8

4.4 ตัวแปร

ดังที่ได้กล่าวมาแล้วว่าการเขียนโปรแกรมในภาษา C นั้นเราใช้วิธีโยงฟังก์ชั่นต่าง ๆ ทั้งใน compiler library และ definition function [1] เข้าด้วยกันด้วยอาร์กิวเมนต์และตัวแปรภายนอก ในที่นี้เราจึงสมควรรู้จักตัวแปรที่จำเป็นอีกครั้งหนึ่ง ตัวอย่างโปรแกรมที่เชื่อมโยงด้วยตัวแปรภายนอก จะแสดงไว้ท้ายตอน (ดูเรื่อง storage class ในตอน 2.3)

4.4.1 ตัวแปรภายนอก (externalvariable)

ตัวแปรภายนอกก็คือที่เก็บข้อมูลภายนอกฟังก์ชั่นที่เราสามารถเรียกไปใช้ในฟังก์ชั่นได้ (ขอให้สังเกตว่าตัวแปรภายนอกทำหน้าที่คล้ายอาร์กิวเมนต์) เหตุที่ต้องมีตัวแปรภายนอกก็เพราะภาษา C นั้นไม่สามารถทำฟังก์ชันซ้อน (nested function) ได้ การเรียกใช้ตัวแปรภายนอกนั้นเราต้องพิจารณาว่าตัวแปรนั้นมีภาระกิจที่จะสามารถเอื้อเฟื้อให้เรียกใช้ได้หรือไม่ ถ้าตัวแปรภายนอกใดถูกนิยามไว้ต้นโปรแกรมก่อน main ( ) ตัวแปรนั้นก็จะมี

---

[1] กรณี definition function เราสามารถเขียนขึ้นแล้วส่งเก็บ (save) ลงดิสค์ เมื่อประสงค์จะเรียกใช้ให้เรียกใช้เช่นเดียวกับฟังก์ชันในคอมไพเลอร์ (compiler library)

ภารกิจ (scope) ที่สามารถเอื้อเฟื้อต่อคำร้องขอหรือเรียกใช้ได้ตลอดไม่เลือกหน้าผู้เรียก เรียกว่า global variable หมายความว่าทุกฟังก์ชั่นในโปแกรมสามารถเรียกใช้ตัวแปรนี้ได้แม้จะมิได้กำหนดตัวแปรชื่อเดียวกันนี้ไว้ในฟังก์ชั่นต่าง ๆ ก็ไม่เป็นปัญหา ค่าของตัวแปรที่ใช้ลักษณะนี้ทำหน้าที่เหมือนกับอาร์กิวเมนต์ ดังในตัวอย่างต่อไปนี้

```
int x ;
main ( )
{
    x=1 ;
    printf ("x=%d\n", x ;
    func 1 ( ) ;
    printf ("x=%d\n", x) ;
    func 2 ( ) ;
    printf ("x=%d\n", x) ;
    func 3 ( ) ;
    printf ("x=%d\n", x) ;
 }
    func 1 ( ) /* multiply x by1 */
    {
      x=x*1 ;
    }
    func 2 ( ) /* multiply x by2 */
    {
      x=x*2 ;
    }
    func 3 ( ) /* multiply x by3 */
    {
      x=x*3 ;
    }
}
```

จะเห็นว่าเรากำหนดตัวแปร x ให้เป็น int โดยกำหนดไว้ต้นโปรแกรมก่อน main ( ) แสดงว่า func 1 ( ), func 2 ( ) และ func 3 ( ) สามารถเรียกใช้ x ได้ การกิจของ x จึงรับใช้ตั้งแต่ต้นจนจบโปรแกรม หมายความว่าค่า x ถูกเรียกใช้โดยไม่จำเป็นต้องมีการส่งค่าเข้าสู่ฟังก์ชันต่าง ๆ ในลักษณะอาร์กิวเมนต์

อนึ่งตัวแปรภายนอกนั้นจะใช้ได้เฉพาะไฟล์ (file) ถ้ามีไฟล์มากกว่า 1 ไฟล์ (แฟ้ม)ที่เราต้องการผสมเข้าเป็นโปรแกรมเดียวกัน ตัวแปรภายนอกของไฟล์ใดในโปรแกรมก็จะมีภารกิจให้เรียกใช้ได้เฉพาะไฟล์นั้น หากประสงค์จะใช้ตัวแปรภายนอกในไฟล์ใด ๆ - เป็นตัวแปรภายนอกของไฟล์อื่น ๆ ในโปรแกรมด้วยเราสามารถกระทำได้ โดยเติมคำว่า extern หน้าชื่อตัวแปรนั้น ซึ่งจะมีผลทำให้ตัวแปร ซึ่งเป็นตัวแปรภายนอกของไฟล์หนึ่งใดในโปรแกรมกลายเป็นตัวแปรภายนอกของไฟล์นั้นด้วย (จบภารกิจเมื่อถึง end of file) เช่น ในโปรแกรมหลักคือ main ( ) หากเราระบุว่า extern int x; คอมไพเลอร์จะทราบโดยส่งให้ linker ไปเรียกหาตัวแปร x ที่นิยามไว้เป็นตัวแปรภายนอก ณ ที่ใดที่หนึ่งในไฟล์ได้ ไฟล์หนึ่งในโปรแกรมมาทำหน้าที่เป็นตัวแปรภายนอกของโปรแกรมหลัก และไฟล์ใดหากต้องการใช้ตัวแปรภายนอกของไฟล์อื่น เช่น x ก็ให้ระบุตัวแปรว่า extern x ไว้ในไฟล์นั้น ตัวอย่างเช่น เรามีไฟล์อยู่ 3 ไฟล์คือไฟล์ที่ 1 ไฟล์ที่ 2 และไฟล์ที่ 3 โดยไฟล์ที่ 1 ประกอบด้วย main ( ) และ func 1 ( ) func 1 ( ) มีตัวแปร x เป็นตัวแปรภายใน (เรียกว่า internal variable หรือ private variable) ไฟล์ที่ 2 ประกอบด้วย func 2 ( ) และ func 3 ( ) ไฟล์นี้มีตัวแปร x เป็นตัวแปรภายนอกและไฟล์ที่ 3 ประกอบด้วย func 4 ( ) เราสามารถผสม (combine) ไฟล์ทั้ง 3 เข้าด้วยกันด้วยการเรียกตัวแปรภายนอกของไฟล์ที่ 2 ไปใช้ในไฟล์อื่น อนึ่งเรื่องผสมไฟล์นี้มีกฎอยู่ข้อหนึ่งว่า "เมื่อเราผสมไฟล์หลายไฟล์เข้าด้วยกัน ด้วยตัวแปรภายนอกภารกิจของตัวแปรภายนอก ในรูป - extern จะเริ่ม ณ จุดที่กำหนด (point of definition) เรื่อยไปและจบภารกิจ ณ จุด

จบหรือท้ายไฟล์ (end of file, EOF)"

ขอให้สังเกตโครงสร้างต่อไปนี้

ไฟล์ที่ 1

```
main ( )
 {
   extern int x ;
 }
func 1 ( )
   {
   int x ;
   }
```

ไฟล์ที่ 2

```
int x ;
func 2 ( )
   {
   }
func 3 ( )
   }
   {
```

ไฟล์ที่ 3

```
func 4 ( )
   {
   extern int x ;
   }
```

จะเห็นได้ว่าไฟล์ที่ 1 เรียกตัวแปร x ภายนอกจากไฟล์ที่ 2 ไปร่วมใช้ใน main ( ) โดยเรียกเป็น extern int x ภารกิจของตัวแปรภายนอกตัวนี้จะมีตั้งแต่เริ่มนิยามไปจนจบไฟล์ที่ 1 อนึ่งขอให้สังเกตว่าใน func 1 ( ) มีตัวแปร x อยู่ ตัวแปรนี้ถือว่าเป็นตัวแปรภายในหรือตัวแปรส่วนตัวของ func 1 ( ) ไม่เกี่ยวกับ x ใน main ( ) ที่เรียกมาใช้แบบ extern การทำงานของ x ใน func 1 ( ) จึงไม่เกี่ยวข้องหรือกระทบกระเทือนกับextern

สำหรับไฟล์ที่ 2 เรานิยามตัวแปร x เป็นตัวแปรภายนอกอยู่แล้ว ตัวแปร x นี้จึงใช้ได้ทั้งใน `func 2 ( )` และ `func 3 ( )` ภารกิจของ x ในไฟล์ที่ 2 จึงมีอยู่ตลอดไฟล์

ไฟล์ที่ 3 หากเราต้องการผสมกับไฟล์ที่ 1 และ 2 ให้กำหนดตัวแปร x เป็น `extern int x;` ไว้ในไฟล์ที่ 3 ภารกิจของ x ในไฟล์ที่ 3 จะมีอยู่ ณ จุดที่นิยามไปจนจบไฟล์ที่ 3

ตัวแปรภายนอกในภาษาอื่นปรากฏดังนี้

BASIC

All variables in a program are global.

MBASIC

Common variables in chained programs are global among programs

FORTRAN

COMMON OUTSIDE

PASCAL

All identifiers declared in the main program are global.
All identifiers declared within a procedure are hidden from external procedures, but are globally known by procedures nested within it.

PL/I

DECLARE OUTSIDE EXTERNAL FIXED BIN(15,0);
All identifiers declared within a procedure are hidden from external procedures, but are globally known by procedures nested within it.

COBOL

LINKAGE SECTION.

01 OUTSIDE PICTURE 9(5).

4.4.2 automatic variable

ตัวแปร auto คือตัวแปรที่มีขอบข่ายภารกิจ อยู่เฉพาะในฟังก์ชันที่กำหนดลักษณะตัวแปรนั้นไว้เท่านั้น (internal หรือ local) โดยเราสามารถกำหนดค่าให้ตัวแปรดังกล่าวได้ตามความประสงค์เพราะเราถือว่าเป็นตัวแปรภายในหรือตัวแปรส่วนตัวของเฉพาะฟังก์ชัน การเคลื่อนไหวค่าของตัวแปรนี้จึงเป็นเรื่องภายใน (local)มิได้กระทบกระเทือนถึงตัวแปรในฟังก์ชันอื่นแม้ว่าจะมีชื่อซ้ำกันก็ตาม และหากมีชื่อซ้ำกับตัวแปรภายนอกก็ไม่เป็นปัญหา เพราะจะข้ามผ่านตัวแปรภายนอกนั้นไป ตัวอย่างเช่น int x; ใน func 1 ( ) ข้างต้น

ขอให้พิจารณาตัวอย่างต่อไปนี้

```
/*show the effect of automatic storage class variable */
main ( )
{
     int i, x ;
     for (i=1, x=0; i<10; i++, x++) {
     printf ("\n the value of x in main : %d", x);
     prove_it( ) ;
    }
{
prove_it ( )
  {
     int x ;
     printf ("the value of x in prove-it : % d", x) ;
  }
```

จากโปรแกรมตัวแปร x เป็นตัวแปร auto ตลอดทั้งใน main ( ) และในฟังก์ชัน prove_it ( ) เนื่องจากค่าของตัวแปร auto นั้นมีค่าเท่ากับค่าที่ค้างอยู่เดิม (garbage) ในการทำงานคราวก่อน ค่าของตัวแปร x ในฟังก์ชัน prove-it ( ) จึงเท่ากับค่าของ x

ใน main ( ) ลองวิ่งโปรแกรมข้างบนดูจะพบว่าเนื่องจาก prove-it( ) ใน main ( ) ถูกเรียกภายหลังฟังก์ชัน printf ( ) ค่าของ x ใน printf ("the value of x in main : %d", x) , จึงทำงานก่อน ค่าของ x จะค้างอยู่ (garbage) แล้วถูกส่งไปพิมพ์ในฟังก์ชัน printf ("the value of x prove-it : %d", x) ; ของฟังก์ชัน prove-it ( )

ลองตัด int x; ใน prove-it ( ) ออกแล้วดูผลลัพธ์ จากนั้นลองนำ int x ; ใน main ( ) ไว้เป็นตัวแปรภายนอกโปรแกรม main ( ) แล้วดูผลลัพธ์

สิ่งที่น่าสังเกตไว้ก็คือตัวแปรภายนอกจะส่งค่าเดิมไปตลอดโปรแกรม ขณะที่ตัวแปรภายในคือ automatic variable จะมีค่าเปลี่ยนไปเมื่อฟังก์ชันทำงานจบลงแล้วโดยมีค่าที่คำนวณครั้งสุดท้ายเก็บไว้ในส่วนความจำและเปลี่ยนไปเก็บค่าใหม่ (เป็นขยะกองใหม่) ค้างไว้ในส่วนความจำนั้น หากประสงค์จะให้ตัวแปร auto มีค่าเป็น 0 หรือมีค่าเท่ากับปริมาณใด ๆ ให้กำหนดค่าเริ่มต้นใหม่เสมอไป

4.4.3 static variable

ตัวแปร static สามารถทำหน้าที่ทั้งในฐานะตัวแปรภายใน (internal หรือ private หรือ local variable) ของฟังก์ชันและในฐานะตัวแปรภายนอก ตัวแปร static ต่างกันกับตัวแปรแบบ auto ตรงที่ค่าของตัวแปร static จะคงที่อยู่เช่นเดิมเสมอ เช่นในฟังก์ชัน prove_it ( ) ตัวแปร x เป็นตัวแปร auto ค่าของ x ในฟังก์ชันซึ่งแปรค่าไปตามค่าของ x ที่ค้าง (ขยะ) มาจาก main ( ) ถ้าเราประสงค์จะให้ค่าของ x ในฟังก์ชัน prove_it ( ) คงที่อยู่เสมอไป โดยไม่เปลี่ยนแปลงเราสามารถกำหนดตัวแปร x ให้เป็น static ดังนี้คือ

```
static int x ;
```

ซึ่งจะทำให้ค่าของ x ในฟังก์ชัน prove it ( ) คงที่เสมอตราบใดที่ยังมิได้ถูกกำหนดค่าเริ่มต้นให้เป็นอย่างอื่น

เราอาจกำหนดให้ตัวแปรหรือฟังก์ชันเป็น static ได้เช่นกันกับที่ถ้าเราประสงค์จะให้ตัวแปร static หรือฟังก์ชัน static เป็นตัวแปรภายนอก (กำหนดเป็น external static variable) หรือฟังก์ชันภายนอก (กำหนดเป็น external static function) การกระทำดังนี้จะมีผลให้ตัวแปร external static variable สามารถใช้งานร่วมลักษณะ global ได้ตลอดไฟล์ของตน (ไม่เกี่ยวกับไฟล์อื่น) และทำให้ฟังก์ชัน external static function สามารถใช้งานร่วมลักษณะ golbal ตลอด source file ของตน ด้วยเหตุดังกล่าวแม้เราจะตั้งชื่อตัวแปรหรือชื่อฟังก์ชันในไฟล์หลายไฟล์ไว้ซ้ำกัน การกำหนดให้เป็น static จะช่วยป้องกันความสับสนได้ ทั้งนี้เพราะตัวแปร หรือฟังก์ชัน static จะเป็นตัวแปรภายในของฟังก์ชัน (internal variable หรือ local variable) หรือฟังก์ชันภายในของไฟล์ (internal function) อยู่แล้วโดยธรรมชาติ การที่เป็น internalแปลว่าภารกิจจำกัดอยู่เฉพาะเรื่องภายใน หากแม้นเราขยายภารกิจของ static ให้มีขอบข่ายกว้างขึ้นก็ยังคงจำกัดอยู่เฉพาะเรื่องภายใน คือภายใน file ที่ตนสังกัดหรือภายใน source file ที่ตนสังกัด การขยายภารกิจ ก็คือ การระบุให้เป็น external static variableหรือ external static function ตามลำดับนั่นเอง

4.4.4 register variable

การกำหนดพื้นที่แบบ register storage class ก็คือการแจ้งให้คอมไพเลอร์สำรองรีจิสเตอร์ใน CPU ไว้ให้แก่ตัวแปร การกำหนดให้ตัวแปรเป็นแบบ register จึงหมายถึงการกำหนดให้ตัวแปรเก็บค่าไว้ในรีจิสเตอร์ของ CPU เหตุผลของ register storage class ก็คือเพื่อต้องการความรวดเร็วในการทำคำสั่ง เพราะการรับส่งข้อมูล (data manipulation) ในรีจีสเตอร์นั้นรวดเร็วกว่าในส่วนความจำ โดยปกติเรานิยมใช้ register variable ในกรณีที่ตัวแปรนั้นต้องทำงานนาน เช่น loop counter ซึ่งจะทำงานด้วยระยะเวลาที่สั้นลงถ้าถูกกำหนดให้นั้น register

ผู้ใช้คอมพิวเตอร์ควรตรวจดูว่าคอมไพเลอร์ (ดูจาก compiler document) ว่าสามารถจัดที่เก็บข้อมูลแบบ external, static และ register ได้หรือไม่นอกเหนือจากแบบ automatic ซึ่งต้องมีอยู่แล้วเพราะโดยทั่วไปถ้าเป็นเครื่องระดับไมโครคอมพิวเตอร์คอมไพเลอร์จะไม่จัดที่เก็บข้อมูลแบบ static และ register ไว้ให้เพราะมีจำนวนรีจิสเตอร์ใน CPU ค่อนข้างจำกัด อย่างไรก็ตาม คอมไพเลอร์ที่ดี (โดยมากมักมีราคาสูง) มักจัดที่เก็บแบบ register ไว้ให้

4.5 ตัวอย่างโปรแกรม

โปรแกรมต่อไปนี้ เป็นโปรแกรมสำหรับหาค่าเฉลี่ยเคลื่อนที่ (running average) โปรแกรม main ( ) จะเรียกใช้ฟังก์ชัน (ชนิด definition function) 2 ฟังก์ชั่นคือ iniavg ( )และ averag ( ) การเชื่อมโยงฟังก์ชั่นทั้งสองนี้เข้าด้วยกันใช้วิธีเชื่อมโยงด้วยตัวแปรภายนอกชื่อ sum และ count ขอให้สังเกตการทำงานของ do-while loop และคำสั่ง if

```
main ( )
/* this program computes the running average of input numbers */
/* an input of 0 or a bad input terminates the program */
  {
  float f,avg;
  double averag( );
  int ret;
  printf("\nThis program computes running averages");
  iniavg( );

  do
      {
      printf("\nEnter a number. Enter 0 to end: ");
      ret = scanf("%f",&f);
      if ((f! = 0.0)&&(ret = = 1))
          {
          avg = averag(f);
          printf("\nAverage so far is %f",avg);
          }
      }
      while ((f! = 0.0)&&(ret = = 1));
  exit(O);
  }
int count;
double sum;
iniavg( )
/* initializes the averaging for averag( ) */
  {
  sum = 0.0;
  count = 0;
  return;
  }
double averag(f)
/* averages the Input numbers returns average */
double f; /* number to be averaged in */
  {
  sum + = f;
  count + + ;
  return sum/count;
  }
```

4.6 ฟังก์ชันที่มีใช้ทั่วไปในคอมไพเลอร์

ในที่นี้ขอยกเอาฟังก์ชันที่เป็น compiler library มากล่าวสรุปไว้ เพื่อให้เห็นทางใช้ประโยชน์ฟังก์ชั่นที่จะกล่าวถึงนี้โดยปกติจะมีในคอมไพเลอร์ทั่วไปทุกระดับทุกราคา คอมไพเลอร์ที่มีราคาสูงและเป็นคอมไพเลอร์ที่สมบูรณ์แบบ (full feature) จะมีฟังก์ชั่นมากมาย ผู้ใช้สมควรตรวจดูเสียก่อนว่าคอมไพเลอร์ที่ซื้อหามานั้นมีฟังก์ชั่นใดบ้างที่เตรียมไว้ให้เราเรียกใช้งาน ต่อไปนี้เป็นฟังก์ชันบางส่วนที่มีใช้ทั่ว ๆ ไปในทุกรุ่น

ก. ฟังก์ชั่นจัดพื้นที่ส่วนความจำ

1) char * malloc (size)
   int size ;

ใช้จัดสรรบล็อคของส่วนความจำทั้งสิ้น size ไบท์ (sizeเป็นจำนวนเต็ม) malloc จะเป็น pointer ชี้ไปที่แอดเดรสในส่วนความจำ

2) char * calloc (numel, elsize)
   int numel, elsize ;

ใช้จัดสรรบล็อคในส่วนความจำทั้งสิ้น numel × elsize ไบท์ calloc เป็นตัวแปร pointer

3) free (addr)

ใช้เรียกคืน (ปลดปล่อย) พื้นที่หรือบล็อคที่จองไว้ด้วย malloc หรือ calloc ณ แอดเดรสหมายเลข addr ให้เป็นที่ว่างที่สามารถใช้ประโยชน์อย่างอื่นแทน ข้อควรระวังก็คือ อย่าเรียกคืนแอดเดรสที่ยังไม่เคยถูกจองเพราะจะเป็นปัญหา

ข. ฟังก์ชันใช้สอยทั่วไป

1) clearmen (addr, count, chr)

```
char*addr ;
int count ;
char chr ;
```

ใช้เก็บอักขระชื่อ chr ซ้ำ ๆกันจำนวนทั้งสิ้น count ไบท์ตั้งแต่แอดเดรสที่ addr เป็นต้นไป

2) movmem (dest, source, count)

```
char*dest ;
char*source ;
int count ;
```

ใช้ย้ายข้อมูลทั้งสิ้น count ไบท์จากจุดที่เริ่มต้น ณ แอดเดรส source ไปเริ่มต้นใหม่ที่แอดเดรส dest

ค. ฟังก์ชันตรวจสอบเกี่ยวกับอักขระ

1) isdigit (c)

ใช้ตรวจดูว่า c เป็นตัวเลขหรือไม่ ถ้าฟังก์ชันส่งค่าคืนเป็น 0 (แปลว่า false) แสดงว่า c มิใช่ตัวเลข ถ้าส่งค่าคืนเป็น nonzero (แปลว่า true) แสดงว่า c เป็นตัวเลข (digit) คือ 0-9

2) isalpha (c)

ใช้ตรวจดูว่า c เป็นอักษรหรือไม่ ถ้าฟังก์ชันส่งค่าคืนเป็น 0 แสดงว่า c มิใช่ตัวอักษร ถ้าส่งค่าคืนเป็น nonzero แสดงว่า c เป็นตัวอักษร

3) islower (c)

ใช้ตรวจดูว่า c เป็นอักษรตัวเล็ก (lower case character) หรือไม่ ถ้าค่าส่งคืน เป็น 0 แสดงว่า c มิใช่อักษรตัวเล็ก ถ้าค่าส่งคืนเป็น nonzero แสดงว่า c เป็นอักษรตัวเล็ก

4) isupper (c)

ใช้ตรวจดูว่า c เป็นอักษรตัวใหญ่ (upper case character) หรือไม่ ถ้าค่าส่งคืนเป็น 0 แสดงว่า c มิใช่อักษรตัวใหญ่ ถ้าค่าส่งคืนเป็น nonzero แสดงว่า c เป็นอักษรตัวใหญ่

5) isspace (c)

ใช้ตรวจดูว่า c เป็น white space (คือเว้นระยะ ตั้งระยะ เว้นว่าง หรือขึ้นบรรทัดใหม่) หรือไม่ถ้าค่าส่งคืนเป็น 0 แสดงว่า c มิใช่ white space ถ้าค่าส่งคืนเป็น nonzero แสดงว่า c เป็น white space

6) toupper (c)

ใช้เปลี่ยนอักษรตัวเล็กเป็นอักษรตัวใหญ่ ก่อนใช้ฟังก์ชันนี้ เราควรทดสอบด้วยฟังก์ชัน islower (c) เสียก่อน

7) tolower (c)

ใช้เปลี่ยนอักษรตัวใหญ่เป็นอักษรตัวเล็ก ก่อนใช้ควรทดสอบด้วยฟังก์ชัน isupper (c)เสียก่อน

ง. ฟังก์ชันเกี่ยวกับสตริง

1) strlen **(string)**

```
char*string ;
```

ใช้นับจำนวนอักขระ (จำนวนไบท์) ของสตริง หรือเพื่อขอทราบความยาวของสตริง (ไม่นับ null terminator คือ '\0')

2) char *strcpy (dest, source)
   char*dest ;
   char *source. ;

ใช้ย้ายสตริงทั้งชุดจากแอดเดรส sourceไปเริ่มที่แอดเดรส dest กรณีนี้ค่าส่งคืนคือ dest และ '\0' จะถูกย้ายตามสตริงไปด้วย

3) char *strcat (dest, source)
   char *dest ;
   char * source ;

ใช้รวมสตริงที่มีจุดเริ่มที่แอดเดรส source เข้ากับสตริงที่มีจุดเริ่มที่แอดเดรส dest กรณีนี้ค่าส่งคืนคือ dest และ '\0' จะถูกส่งตามไปด้วย

4) strcmp (string 1, string 2)
   char*string 1 ;
   char*string 2 ;

ใช้เปรียบเทียบ string 1 กับ string 2 อักขระต่ออักขระผลการเปรียบเทียบแสดงด้วยค่าส่งคืนดังนี้

ค่าส่งคืนเท่ากับ 0 เมื่อ string 1 กับ string 2 ยาวเท่ากันหรือจับพอดีคู่(match)
ค่าส่งคืนเป็นบวก เมื่อ string 1 ยาวกว่า string 2
ค่าส่งคืนเป็นลบ เมื่อ string 1 สั้นกว่า string 2