บทที่ 7
I/O ในภาษา C

ในภาษา C นั้นเราจะไม่มีคำสั่ง input หรือ output ใช้เช่นในภาษาเบสิค แปลว่าไม่มีคำสั่ง PRINT หรือ INPUT หรืออื่น ๆ การรับข้อมูล (input) หรือแสดงข้อมูลในภาษา C จะใช้ในลักษณะของการเรียกใช้ฟังก์ชัน อาจเป็น Compiler library หรือฟังก์ชันซึ่งผู้เขียนโปรแกรมสร้างขึ้นใช้เองซึ่งยืดหยุ่นไปได้มากมายตามความสามารถของผู้เขียนโปรแกรม เรื่องของ definition function หรือ ฟังก์ชันที่เราเขียนขึ้นเองเพื่อเรียกใช้ในลักษณะ modular approach นั้นได้กล่าวถึงมาแล้วละเอียดพอควร คราวนี้ลองมาศึกษา library function คือฟังก์ชันในคอมไพเลอร์ โดยเฉพาะในส่วนที่เป็น I/O ดูบ้าง

ฟังก์ชัน I/O ในภาษา C นั้นอาจวัดได้เป็น 2 พวกคือ character I/O ซึ่งใช้กับเทอร์มินัลเช่น getchar ( ), putchar ( ) และ formatted I/O ซึ่งใช้กับ disk file หรือหากมองในรูปการรับและแสดงข้อมูลก็อาจจัดเป็น 2 พวกคือ พวกที่รับและแสดงข้อมูลทีละอักษรเรียกว่า buffered routine และพวกที่รับแสดงข้อมูลเป็นไฟล์เรียกว่า Unbuffered routine

7.1 preprocessor

preprocesson คือ โปรแกรมที่ใช้จัดการกับคำสั่งพิเศษบางคำสั่งก่อน เพื่อเปลี่ยนให้เป็นรหัสที่คอมไพเลอร์อ่านได้ คำสั่งที่นิยมใช้กันมากมี 2 คำสั่งคือ # define และ # include [1]

1) คำสั่ง # define

define เป็นคอมมานด์ที่เราใช้สำหรับสร้าง symbolic constant ขึ้นมาใช้ในโปรแกรม เพื่อให้เกิดความคล่องตัวและสะดวกสบายในการแก้โปรแกรม โดยเราไม่ต้องตรวจแก้โปรแกรมมาก อาจเปลี่ยนค่าของ symbolic constant เพียงเล็กน้อยก็เกิดเงื่อนไขใหม่ที่ใช้ได้ตลอดโปรแกรม เช่น # define MAX 80 เพื่อกำหนดจำนวนอักขระสูงที่สุดในแต่ละบรรทัดให้เท่ากับ 80 อักขระ หรือ # define CLEAR "\014" เพื่อ

[1] ยังมีคำสั่งใน preprocessor อีกหลายคำสั่งเช่น #indef, # if, # ifdef, # ifndef # else, # endif และ # line ซึ่งจะกล่าวถึงต่อไป

เพื่อกำหนดให้ล้างจอภาพ [1/] และคำสั่งอื่น ๆ ที่พบมากในไฟล์ stdio.h (ย่อจากคำว่า standard I/O . overhead)

รูปแบบของคำสั่ง # define คือ

```
# define NAME xxxxx
```

คำว่า NAME หมายถึง symbolic ที่เราต้องการเรียกใช้ และ xxxxx หมายถึง string of character หากมีสตริงยาวมากจนต้องเขียนต่อในบรรทัดต่อไปให้ระบุท้ายบรรทัดแรกนั้นด้วย back slashคือ \ โดยปกติเราจะต้องกำหนด # define ไว้ที่ส่วนต้นของโปรแกรม นอกจากนี้ # define ยังสามารถมีอาร์กิวเมนต์ที่ส่งไปยังสตริง xxxxx และสามารถอ้างอิงถึง # define ที่เคยนิยามไว้แล้ว กล่าวคือ

กรณีที่มีอาร์กิวเมนต์ รูปแบบคำสั่งคือ

```
# define NAME (arg1,arg 2, . ..)
```

เช่นเรานิยามว่า # define PLUSONE (x) x+1 ดังนั้น PLUSONE(5) จะถูกแปลเป็น 5+1 หรือ PLUSONE (y) จะถูกแปลเป็น y+1

สำหรับกรณีอ้างอิง # defineเดิมให้ดูตัวอย่างต่อไปนี้ ถ้าคำสั่งเดิมนิยามว่า

[1/] ขอให้สังเกตว่า \014 อยู่ภายในเครื่องหมายคำพูดคู่คือ " " แสดงว่า \014 เป็น string constant ที่เก็บอยู่ในส่วนความจำในรูป '\014' กับ '\0' สำหรับ '\014' เรียกว่า character constant ใช้แทนข้อความที่เรานับว่าเป็นอักขระเพียงอักขระ เดียว กรณีนี้จะไม่ใช้ null terminator คือ '\0' string constant ใช้ %sเป็น conversion character ใน printf( ) ขณะที่ character constant ใช้ %c โดย %s จะพิมพ์ข้อความทั้งปวงและหยุดเมื่อพบ '\0' ขณะที่ % c จะพิมพ์เพียงอักขระเดียว

```
# define ONE 12
```

และเราต้องการอ้างถึงนิยามนี้ด้วย # define TWO ว่า # define TWO ONE+ONE ค่าของ TWO จะเป็น 12+12 ดังนี้เป็นต้น

ผลจาก # define จึงปรากฏในรูปของการแทนที่ค่าของ symbolic ด้วยค่าคงที่ที่กำหนดเป็น string characterไปตลอดโปรแกรม การแก้โปรแกรมจึงแก้เพียง string character ซึ่งมีผลเสมือนแก้ค่าคงที่นั้นตลอดโปรแกรม ซึ่งเราทำได้ง่ายมาก เนื่องจากเรานิยมกำหนดให้ # define นิยามไว้ต้นโปรแกรมดังที่กล่าวมาแล้ว

ลองดูตัวอย่างต่อไปนี้

```
for(i=0; (c=getchar( ) )! = '\n' && i<MAX-1; + +i)
    name [i] = c ;
mane [i] = '\0' ;
```

[1/]

จะเห็นว่า for loop นี้เริ่มที่ i = 0 และจบที่ (c = getchar ( )) ! = '\n' && i < MAX - 1 ซึ่งเป็นปลาย loop ที่ผสมทั้ง i และ c เข้าด้วยกัน เนื่องจากเรากำหนดให้ # define MAX 80 ดังนั้นปลาย loop ก็คือ (c = getchar ( )) ! = '\n' && i < 79 โดยที่เราสงวนที่ท้ายสตริงไว้สำหรับ terminater คือ '\0' ซึ่งใช้แสดงจุด

[1/] (c = getchar ( )) ! = '\n' && i < MAX-1 ก็คือ (c = getchar ( )) = = '\n' || i < MAX-1 && หมายถึง A N D และ || หมายถึง O R

ปิดท้ายสตริง 1/ จะเห็นได้ว่าตามตัวอย่างนี้ ถ้าเรานิยามค่า MAX ไว้แล้วและประสงค์จะเปลี่ยนค่าของ MAX เป็นอย่างอื่นเราก็กระทำได้โดยง่ายโดยเพียงแต่เปลี่ยนค่าของ ××××× ใน # define ซึ่งมีผลเสมือนแก้โปรแกรมทั้งโปรแกรม หากไม่ใช้ # define เราจะต้องตรวจโปรแกรมโดยตลอดแล้วค่อย ๆ แก้ทีละจุด

2) คำสั่ง # include

คำสั่ง # include ใช้สำหรับอ่านคำสั่งในไฟล์แล้วผนวกเข้ากับโปรแกรมโดยจัดให้ไฟล์นั้นเป็นส่วนหนึ่งของโปรแกรม การเรียกใช้ให้เรียกใช้ในลักษณะของฟังก์ชัน คำสั่งนี้ตรงกับคำสั่ง APPEND ในภาษาเบสิคบางรุ่น

ลองดูตัวอย่างต่อไปนี้ซึ่งแสดงการผนวกฟังก์ชันชื่อ inputs (s) ซึ่งเราเก็บ (save) ลงในดิสค์เป็นไฟล์ชื่อ inputs.c เข้ากับโปรแกรมโดยเรียกใช้เป็น inputs (s) ;

```
    /* function that aceept a string from key board */
inputs (s)
char s [ 1;
{
     int i ;
     for(i=0; (s[i] = getchar( )! = '\n' && i<MAX-1; ++i)

     s [i] = '\0' ;
}
```

1/ ประโยชน์ null terminator ประการหนึ่งก็คือช่วยให้ฟังก์ชัน strlen ( ) นับจำนวนอักขระไม่ผิดเพราะ strlen ( ) จะนับจำนวนอักขระของสตริงไปเรื่อย ๆ เมื่อพบ '\0' ก็จะหยุดนับแล้วแจ้งจำนวนอักขระออกมาเป็น int ดูตัวอย่างการใช้ประโยชน์ลักษณะอื่นในตอน 7.2

เมื่อเก็บฟังก์ชัน inputs (s) ลงดิสค์ในชื่อ inputs.c เราสามารถผนวกไฟล์นี้เข้ากับโปรแกรมดังนี้ (ขอให้สังเกตว่า loop body คือ ; ซึ่งแสดงว่าไม่ทำอะไร แต่ต้องมีไว้เพราะรูปไวยากรณ์ของ for loop บังคับดูข้อสังเกตหน้าถัดไป)

```
# mclude"stdio.h" /*read in the standard I/O file */
# define MAX 80   /* maximum length of name */
# define CLEAR"\014"  /*ASCII 12 clear my screen */
# include "inputs.c" /* include the input stringfunction */
main ( )
i
     char s[MAX] ;
     printf (CLEAR) ;
     printf ("enter your name") ;
     inputs (s) ;
     printf ("\n\n your name is : %s", s) ;
{
```

โปรแกรมนี้มีผลเหมือนกับโปรแกรมต่อไปนี้ซึ่งเขียนคำสั่งสำหรับรับอักขระเข้ามาในลักษณะของส่วนหนึ่งของโปรแกรม วิธีทั้งสองนับได้ว่าเป็นวิธีที่ดี แต่วิธีที่ดีที่สุดก็คือ การเรียกใช้ฟังก์ชันในลักษณะของ module

```
# include" stdio.h"
# define MAX 80
# define CLEAR '\014'
main ( )
{
     char name [MAX] ;
     int c, i;
     printf (CLEAR);
```

```
    printf ("enteryour name") ;
    for (i=0; c=getchar (1)! = '\n' && i<MAX-1; ++)
         name [i] = c ;
    name [i] = '\0' ;
    printf ("\n\n your name is %s", name) ;
}
```

ขอให้สังเกตว่า # define และ # include ไม่มีเครื่องหมายอัฒภาค (;) ต่อท้าย เหมือนคำสั่งอื่น ๆ รูปแบบของ # include คือ

```
# include " filename"
```

3) คำสั่งอื่น ๆ

(1) # undef name	ยกเลิกนิยามชื่อ name ที่นิยามตาม # define
(2) # if exp	ทดสอบว่านิพจน์ exp ว่าเป็น non-zero หรือไม่
(3) # ifdef name	ทดสอบว่าชื่อ name ได้เคยนิยามไว้ด้วย #define หรือไม่
(4) # ifndef name	ทดสอบว่าเชื่อ name ได้ยกเลิกไปแล้วหรือยัง
(5) # else	แจ้งให้คอมไพเลอร์แปลคำสั่งบรรทัดต่อไปได้ ถ้าพบว่า if ในข้อ (2)-(4) ไม่จริง
(6) # endif	จบคำสั่ง if ยอมให้คำสั่ง if เป็น nested if
(7) # line constant	แจ้งหมายเลขบรรทัดให้คอมไพเลอร์ทราบ โดยมาก มุ่งหมายเพื่อสั่งพิมพ์ข้อความที่ต้องการในบรรทัดหมายเลขที่ระบุนั้น

ข้อสังเกต จากตัวอย่างและคำอธิบายที่ผ่านมามีสิ่งที่ควรสังเกตดังนี้

1. ฟังก์ชัน getchar ( ) ซึ่งเป็น library function นั้นจะรับอักขระจากแป้นพิมพ์ และเนื่องจากฟังก์ชัน getchar ( )จะต้องส่งค่าเป็น int คือส่งค่าไปเก็บในที่ชื่อ c ที่เรากำหนดให้ c เป็น int (ดูคำสั่ง c = getchar ( )) แสดงว่าเราเตรียมที่ไว้ 2 ไบท์ (เพราะ c เป็น int ใช้ที่ 2 ไบท์) สำหรับเก็บอักขระ (1 อักขระใช้ที่ 1 ไบท์) จากนั้นจึงนำค่าใน c ใส่ลงในอะเรย์ name[ ] ซึ่งเป็น character array (เรากำหนดให้ char name [MAX]) ขอให้สังเกตว่าเราถ่ายค่าจาก getchar ( ) ลง c และถ่ายจาก c ลง name [ ] ทีละอักขระเรื่อยไปจนกว่าจะกดปุ่ม RETURN (↵) หรือ รับข้อมูลเข้าถึงอักขระที่ 79 (MAX-1) จึงหยุดแล้วตัดออกจาก for loop ถ้าเราจะถ่ายค่าจาก getchar ( ) ลงในอะเรย์ name [i]โดยตรง โดยไม่ผ่าน c ก็ได้ดังนี้

```
for(i = 0, (name [i] = getchar ( ))! = '\n' && i<MAX-1; ++i)
name [i] = '\0' ;
```

2. null terminator คือ '\0' เป็น ASCII null มีค่าเป็น 00000000 ในระบบเลขฐาน 2 (เรียกว่า machine zero) เราต้องใช้เป็น '\0' เพื่อชี้ให้เห็นว่ามิใช่ 0 คือ zero ASCII ซึ่งมีค่าเป็น 00110000 ด้วยความเข้าใจในข้อแตกต่างนี้เราสามารถนำความรู้นี้ไปใช้ประโยชน์ได้ เช่นใน while loop เราสามารถเขียนเป็น while (*s) แทนการเขียนว่า while (*s ! = '\0') เพราะถ้าเงื่อนไขจริง คือ * s มิได้ชี้ที่ 00000000 คำสั่งต่อจาก whlie จะถูกเรียกทำงาน

3. ในตัวอย่างข้างต้นที่เราผนวกไฟล์ inputs.c เข้ากับโปรแกรมด้วย-คำสั่ง # inculde จะพบว่าในโปรแกรมเราเรียกใช้ฟังก์ชัน inputs ( )โดยส่งอาร์กิวเมนต์ s ไปให้ ลองดูที่ฟังก์ชัน inputs ( ) จะพบว่าเรากำหนดให้ s เป็นอะเรย์ (character array) คือ s[ ] การเรียกใช้ฟังก์ชัน inputs(s)โดยส่งอาร์กิวเมนต์คืออะเรย์ชื่อ s นั้นมีผลเท่ากับการส่งแอดเดรสแรกของอะเรย์คือแอดเดรสของ s[0] ไปให้ สมมุติ s [0] อยู่ที่แอดเดรส 1000 ก็แสดงว่าฟังก์ชัน input ( ) รับแอดเดรส 1000 ไว้แล้วเริ่มทำงานโดยรับอักขระเข้าด้วยฟังก์ชัน getchar ( ) แล้วเริ่มเก็บไว้ ณ แอดเดรส 1000 เป็นต้นไป ดังนั้นชื่ออะเรย์กับแอดเดรสแรกของอะเรย์จึงเป็นสิ่งเดียวกัน และเราจะเปลี่ยนที่ หรือย้ายที่ของอะเรย์ เช่น การทำสำเนาอะเรย์ไม่ได้ คือจะ increment เป็น ++s หรือ decrement เป็น --s มิได้ เรื่องนี้นับว่าต่างจากตัว-แปรที่มิใช่อะเรย์ เพราะการเรียกใช้ตัวแปรที่มิใช่อะเรย์นั้น ค่าของตัวแปรจะถูกสำเนา (copy) ไปเก็บ ณ แอดเดรสอื่นก่อนแล้วจึงส่งมา/ไปใช้ต่อไป

อนึ่ง เมื่อพิจารณาการทำงานของฟังก์ชัน inputs(s) จะพบว่าฟังก์ชันนี้จะไม่ส่งสิ่งใดคืนไปให้ main ( ) เพราะเรามีความประสงค์ที่จะให้ฟังก์ชัน inputs(s) สร้างสตริงจากข้อมูลที่ส่งเข้าทางแป้นพิมพ์เท่านั้น การที่ฟังก์ชัน inputs(s) จำเป็นต้องรับแอดเดรสเดิมของอะเรย์ s ก็เพื่อให้ฟังก์ชัน printf ( )ทำการพิมพ์ข้อความได้ถูกต้อง มิใช่ไปดึงเอาข้อความในแอดเดรสอื่นที่ไม่เกี่ยวข้องมาพิมพ์

7.2 ฟังก์ชันรับข้อมูลที่เป็นตัวเลข [1]

ภาษา c เป็นภาษาที่ไม่มีคำสั่งใดสำหรับรับข้อมูลตัวเลข (ภาษาเบสิคใช้คำสั่ง INPUT หรือ READ-DATA) ดังนั้นถ้าเราประสงค์จะรับข้อมูลตัวเลขเข้าเราจึงจำเป็นต้องสร้างฟังก์ชันขึ้นใช้เอง ยกเว้นมี libriary function สำหรับรับข้อมูลตัวเลขอยู่ในคอมไพเลอร์บางรุ่น

ตัวอย่างต่อไปนี้เป็นโปรแกรมที่เรียกใช้ฟังก์ชัน askint ( ) ซึ่งเราสร้างขึ้นเพื่อรับข้อมูลตัวเลขตามความต้องการข้างต้น โปรแกรมนี้จะเรียกใช้ฟังก์ชั่น inputs (s) ที่กล่าวถึงแล้วในตอน 7.1 ขอให้สังเกตว่าเป็นโปรแกรมขั้นแนะนำซึ่งยังไม่ละเอียดมากนัก

```
# include "stdio.h"
# define MAXDIGIT 20 /* set maximum digit to 20 */
main ( )
{
   char s[MAX] ;
   int x ;
   x = askint (s) ;
   printf ("the number is %d", x) ;
   printf ("\n and its square is : %d", x * x) ;
}
/*this routine request and converts a string to an int */
askint (s)
char s[ 1 ;
{
```

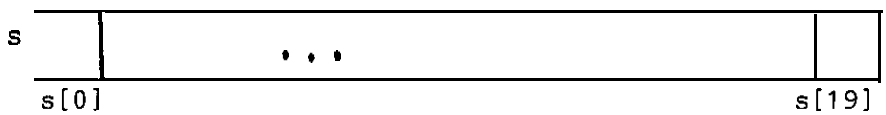
---

[1] ดูฟังก์ชัน getsum ( ) ในตอน 5 . 3

```
int i, sign, num ;
printf ("enter an integer number") ;
inputs (s) ;
i = 0 ;
sign = 1 ;
if (s[i] = = '-') {     /* determineif negative*/
sign = -1 ;             /* assume plus sign not enter ● /
++i ;
}
for (num = 0 ; s[i] >= '0' && s[i] <= '9'; i++)
     num = 10 * num + s[i] - '0' ;
return (num * sign) ;
}
```

จากโปรแกรมจะพบว่า main ( ) เรียกใช้ฟังก์ชัน askint ( ) โดยส่งอะเรย์ s (ซึ่งก็คือแอดเดรสของ s[0]) เป็นอาร์กิวเมนต์แล้วให้ askint ( ) ส่งผลไปเก็บในที่ชื่อ x ซึ่งเป็น int

ฟังก์ชัน askint ( ) ทำงานโดยเรียก characher string จากฟังก์ชันที่รับข้อมูลผ่านแป้นพิมพ์ตามฟังก์ชัน inputs (s) จากนั้นจึงตรวจเครื่องหมายของสมาชิกของอะเรย์ s คือ



ทีละอักขระเรื่อยไปจนครบ 20 อักขระ (ดูคำสั่ง if(x[i] = = '-') { sign = -1 ; ++i ; }) จากนั้นจึงตรวจอักขระต่าง ๆ ทั้ง 20 อักขระ (จำนวนสูงสุดเท่ากับ 20

อักขระ) ทีละตัวว่าเป็นตัวเลขระหว่าง 0 ถึง 9 หรือไม่ ถ้าใช่ให้คำนวณตามสูตร

```
10 * num + s[i]-'0'
```

แล้วเก็บผลลัพธ์ลงในที่ชื่อ num วนเวียนคำนวณและสะสมค่าเรื่อยไปในที่ชื่อ num ดังกล่าวจนครบ 20 อักขระ ได้ผลเท่าไร เอาผลลัพธ์คูณกับ sign คือ (num * sign)แล้วส่งผลลัพธ์คืน main ( ) ในที่ชื่อ x

ลองดูตัวอย่างการทำงานของฟังก์ชัน askint ( ) ดังต่อไปนี้

สมมุติฟังก์ชั่น inputs (s)รับ character string เข้ามาเป็น "-128\0" การทำงานจะเริ่มต้นด้วยการตรวจสตริงนี้ครางละอักขระ

1. เมื่อตรวจที่ s[0] พบเครื่องหมายลบ (-) จึงใส่ค่า -1 ลงในที่ชื่อ sign ตรวจต่อไปจนถึง '\0' จึงเลิกตรวจและไม่พบเครื่องหมายลบอีก เป็นอันว่าตอนนี้ sign = - 1

2. เริ่มตรวจสภาพและสะสมค่าลงในที่ชื่อ num

(1) เมื่อพบ s[1] = '1' ดังนั้น

```
num = 10 * num + s[1] - '0' ;
    = 10 * 0 + 49 -48 ;
      1  (ตอนนี้ในที่ชื่อ num เก็บค่าเท่ากับ 1 เอาไว้)
```

(2) เมื่อ s[2]='2' ดังนั้น

```
num = 10 * num + s[2]-'0' ;
    = 10 * 1 + 50-48 ;
    = 10 + 2 ;
    = 1 2  (ตอนนี้ในที่ชื่อnum เก็บค่าเท่ากับ 12 เอาไว้ )
```

(3) เมื่อ s[3] = '8'

num = **10** * num + s[3]-'0' ;

= 10 * 12 + 56-48 ;

= 120 + 8 ;

= 128 (ตอนนี้ในที่ชื่อ num เก็บค่าเท่ากับ 128 เอาไว้)

ดังนั้นค่าที่ส่งออกจากฟังก์ชั่น askint ( ) ไปไว้ในที่ชื่อ x คือ num * sign ซึ่งก็คือ 128* (-1) = -128

คำถามก็คือเมื่อถึง s[4] คือ '\0' ไม่ทำอะไรหรือ คำตอบก็คือไม่ต้องอะไรเพราะ'\0' เป็น null terminator และเป็นอักขระที่มิใช่ตัวเลข 0-9 ที่กำหนดใน for loop จึงตัดออกจาก loop

ฟังก์ชัน askint ( ) นี้ทำงานเหมือนฟังก์ชั่น atoi ( ) (หมายถึง ASCII to integer) ในคอมไพเลอร์ ลองตรวจดูคอมไพเลอร์ของท่านเพื่อศึกษาการเรียกใช้และลักษณะการใช้งาน โดยปกติ atoi ( ) จะต้องการตัวแปร pointer เสมอ เพื่อใช้ไปที่ตำแหน่งที่สตริงปรากฏอยู่แล้วทำการเปลี่ยนรหัสแอสกี้เป็น int แล้วส่งค่าเป็น int ไปยังจุดเรียก (คอมไพเลอร์บางรุ่นจะแปลง ASCII string เป็น floating point)

สำหรับฟังก์ชันรับข้อมูลเข้า (input function) ที่เป็น compiler library นอกเหนือไปจาก getchar ( ) ที่เราคุ้นเคยดีแล้วยังมีฟังก์ชันอีกฟังก์ชั่นหนึ่งที่ถือว่าเกรดสูงกว่า getchar ( ) ฟังก์ชันดังกล่าวคือ scanf ( )ฟังก์ชัน scanf ( ) เป็นฟังก์ชั่นที่สามารถรับข้อมูลทุกชนิดไม่ว่าจะเป็น character, string, decimal integerหรืออื่น ๆ แต่คอมไพเลอร์บางรุ่นอาจไม่มีฟังก์ชันนี้ซึ่งอาจทำให้เรารู้สึกอึดอัดอยู่

บ้างเพราะขาดเครื่องมือเครื่องใช้ แต่ถ้าเรารู้จักสร้าง input function ขึ้นใช้เอง ได้ดังที่ศึกษาผ่านมาแล้วก็คงจะมิใช่ปัญหาและคงหายอึดอัดไปได้บ้าง

ฟังก์ชัน scanf ( ) มีรูปแบบ (format) ดังนี้

```
scanf (" control string ", argument 1, argument 2, . ) ;
```

control string คือ สิ่งที่ใช้ระบุรูปแบบของข้อมูลนำเข้า ขอให้สังเกตว่า control string จะต้องอยู่ภายในเครื่องหมายคำพูดคู่คือ " " และจะต้องมีเครื่องหมายเปอร์เซ็นต์คือ % เรียกว่า conversion character เขียนอยู่ข้างหน้าตัวอักษรที่แสดงรูปแบบของข้อมูลนำเข้าดังนี้

```
d  =  decimal integer
0  =  octal integer
x  =  hexadecimal integer
h  =  short integer
c  =  single character
s  =  character string
f  =  float-point number
```

argument ก็คืออาร์กิวเมนต์ที่เราส่งไปให้ scanf ( ) ซึ่งสามารถส่งไปได้มากกว่า 1 อาร์กิวเมนต์ ทั้งนี้ขึ้นอยู่กับเราว่าจะรับข้อมูลนำเข้ามากรายการเพียงใด สิ่งที่สำคัญที่ต้องระลึกไว้เสมอก็คืออาร์กิวเมนต์จะต้องเป็น pointer เสมอ (เพื่อชี้ชนิดของตัวแปร ณ แอดเดรสที่เราประสงค์จะให้ scanf ( ) เอาข้อมูลไปเก็บไว้) ดังนั้นข้างหน้าอาร์กิวเมนต์จึงต้องเขียนเครื่องหมายแอมเพอร์แซนด์คือ & กำกับไว้เสมอ ยกเว้นอาร์กิวเมนต์ที่เป็นสตริงหรืออะเรย์เท่านั้นที่ไม่ต้องมีเครื่องหมาย & เพราะชื่ออะเรย์เป็น pointerอยู่

แล้ว และสิ่งที่ต้องระมัดระวังให้มากสำหรับกรณีที่อาร์กิวเมนต์เป็นอะเรย์ก็คือ เราจะต้องเผื่อที่หรือขนาดของอะเรย์ไว้ 1 สำหรับ null terminator คือ \0 แปลว่าถ้าอะเรย์นั้นเตรียมไว้สำหรับข้อมูลที่ใช้ที่สูงสุด 10 ที่เราจะต้องกำหนดขนาดของอะเรย์ไว้เท่ากับ 11

ตัวอย่างเช่น

```
main ( )
{
     char adr [15] ;
     printf ("enter address") ;
     scanf ("enter adress % s ", adr) ;
     printf ("address is\ n\n % s ", adr);

  }
```

ขอให้สังเกตที่คำสั่ง scanf ("enter address % s ", adr); จะพบว่าเรากำหนดให้ข้อมูลนำเข้าเป็นแบบสตริง และอาร์กิวเมนต์ที่ส่งไปจาก main ( ) คืออะเรย์ชื่อ adr ขอให้สังเกตว่าไม่มีเครื่องหมาย & หน้า adr และกรณีนี้เป็นกรณีที่เราเตรียมไว้แล้วว่าที่อยู่ของบุคคลที่มีอยู่นั้นใช้ที่สูงสุดไม่เกิน 14 อักขระ เริ่มได้ตั้งแต่แอดเดรสของ adr [0] ถึงแอดเดรสของ adr [13] ส่วน adr [14] จะเก็บ null terminator (กรณีชื่อยาวที่สุด)

หรือในตัวอย่างต่อไปนี้

```
char name [15] ;
int num, new-num,  old-num;
scanf ("%s%d%*d%d", name,  & num,  & new-num. & old-num);
```

ตัวอย่างนี้มี control string ที่แปลกอยู่รายการหนึ่งคือ %*d ดอกจัน (*) ที่เขียนไว้หน้า d เป็นสัญลักษณ์ที่บ่งบอกว่า "ไม่ต้องสนใจ" ลองดูฟังก์ชัน scanf ( ) ข้างบนอีกครั้งแล้วลองตีความกันดู

```
scanf ("%s%d%*d%d", name, & num, & new-num, & old-num) ;
```

แสดงว่าเราระบุว่าข้อมูลของอะเรย์ชื่อ name เป็น string character ให้รับข้อมูลไปเก็บไว้ ณ แอดเดรสของ name [0] เป็นต้นไป ข้อมูลของ num เป็นจำนวนในระบบเลขฐาน 10 ให้เก็บข้อมูลนี้ ณ แอดเดรสชื่อ num ข้อมูลของ new-num เป็นจำนวนในระบบเลขฐาน 10 ไม่ต้องสนใจ และข้อมูลของ old-numเป็นจำนวนในระบบเลขฐาน 10 ให้เอาไปเก็บไว้ ณ แอดเดรสของ old-num คำว่า "ไม่ต้องสนใจ" หมายความว่าเรา จัด format ของข้อมูลเข้าไว้ในรูปทั่วไป แต่ประสงค์จะให้เก็บหรืออ่านข้อมูล แล้วเก็บเฉพาะบางส่วนที่เราสนใจเท่านั้น กรณีนี้จึงมีประโยชน์มาก เพราะเราจะได้ไม่ต้องจัด-format ของข้อมูลเฉพาะแบบเฉพาะกรณี ทำให้เราเจาะจงเก็บหรือไม่เก็บข้อมูลได้ โดย format เดิมไม่เสียหรือไม่ต้องกำหนด format ใหม่

นอกจากนี้ เราอาจเพิ่มตัวเลขต่อท้ายเครื่องหมาย % เป็นตัวเลขอะไรก็ได้-เพื่อเป็นจำนวนแสดงขนาดของรายการข้อมูล (width of input field) เช่น

```
scanf ("%3d", & num) ;
```

แปลว่าเรากำหนดให้รับข้อมูลนำเข้าที่เป็นเลขในระบบฐาน 10 ขนาดไม่เกินหลักร้อย(เลข 3 หลัก) แล้วเอาไปเก็บไว้ ณ แอดเดรสของ num (ดูตอน 7.3)

ตัวอย่างต่อไปนี้เป็นตัวอย่างที่เคยแสดงไว้แล้วในบทที่ 1 ขอให้ผู้อ่านลองทำความเข้าใจตามที่ได้อธิบายผ่านมาแล้ว อาจทำให้เห็นภาพต่าง ๆ ในชัดเจนมากขึ้นกว่าที่

เคยเข้าใจเมื่อพบครั้งแรก

```
main ( )
/* outputs twice the larger number entered */
{
   float numone, numtwo, outnum ;
   printf ("enter two numbers") ;
   scanf ("%f%f", & numone, & numtwo) ;
   if (numone > numtwo)
        outnum = numone * 2 ;
        else
        outnum = numtwo * 2 ;
   printf ("double the largest number is % f", outnum) ;
   exit (0)
}
```

ขอให้สังเกต control string ในฟังก์ชัน printf ( ) ซึ่งจะได้กล่าวถึงต่อไป

7.3 ฟังก์ชันแสดงผล (Output function)

ในภาษา C เรามีฟังก์ชัน (library function) ที่ใช้เสนอผลอยู่ 3 แบบ ซึ่งต่างก็มีขอบเขตความสามารถแตกต่างกันดังนี้

1) putchar (c) ใช้แสดงผลลัพธ์เป็นอักขระเดี่ยว (single character) บนจอภาพ เช่น putchar (CLEAR); putchar (BELL) ในที่นี้ CLEAR และ BELL เป็นอักขระเดี่ยวตามนิยาม #define CLEAR '\014' และ # define BELL '\007'

2) puts (s) ใช้แสดงผลลัพธ์เป็นสตริงบนจอภาพ s ใน ( ) คืออาร์-กิวเมนต์ ซึ่งเป็นสตริง เช่น puts ("ERROR"), puts ("enter something on sereen"), puts ("my name is")

ขอให้สังเกตลักษณะของอารกิวเมนต์ของฟังก์ชัน putchar ( ) และ puts ( )

สำหรับตัวอย่างต่อไปนี้เป็นตัวอย่างแสดงการใช้ putchar ( )และ getchar ( ) ร่วมกัน

```
while ((c = getchar ( ) ) ! = EOF)
    {
      putchar (toupper (c) );
    }
```

เป็นตัวอย่างแสดงการรับอักขระจากแป้นพิมพ์ แล้วแปลงเป็นอักษรตัวใหญ่ ด้วยฟังก์ชัน - toupper ( ) แล้วแสดงผลทางจอภาพด้วยฟังก์ชัน putchar ( ) ทั้งนี้ให้กระทำเช่นนี้เรื่อยไปตราบเท่าที่เรายังไม่กด EOF (end of file)

ตัวอย่างต่อไปนี้ แสดงการควบคุมตำแหน่งของเคอเซอร์ ด้วยฟังก์ชันชื่อ - cursor ( )

```
# define ESCAPE 27
# define OFFSET 32
cursor (row, column) /*position the cursor */
int row ;
int column ;
    {
```

```
        putchar (ESCAPE) ;
        putchar (' = ') :
        putchar (row+OFFSET) ;
        putchar (column+OFFSET) ;
        return ;
    }
```

3) printf ( )

ฟังก์ชั่น printf ( )เป็นฟังก์ชันรูปทั่วไปสำหรับใช้แสดงผลเพราะยืดหยุ่นได้มาก เนื่องจากสามารถรับอาร์กิวเมนต์ได้หลายแบบ (option) รูปแบบของ printf( ) ปรากฏดังนี้

```
printf ("control string ", argument 1, argument 2, ... );
```

control string คือสิ่งที่ใช้แสดงรูปแบบ (format) ของข้อมูลที่เราต้องการเสมอผล อาร์กิวเมนต์คือสิ่งที่เราต้องการเสนอ

control string มีรูปทั่วไปดังนี้คือ % - w.plx

- แสดงว่าให้พิมพ์ชิดซ้าย (default คือไม่มีเครื่องหมายแปลว่าชิดขวา)

w ตัวเลขแสดงขนาดของรายการข้อมูล (field width) หรือจำนวนหลักหน้าจุดทศนิยม

. จุดทศนิยม

p จำนวนหลักหลังจุดทศนิยม

l แสดงว่าอาร์กิวเมนต์เป็น long type เช่น long int

x คือรูปแบบของอาร์กิวเมนต์ที่ต้องการเสนอผล ประกอบด้วย

d = decimal signed int

u = decimal unsigned int

x = hexadecimal int

o = octal integer

s = character string

c = single character

f = fixed decimal floating point number

e = exponential floating point number

g = e หรือ f ขึ้นอยู่ว่าใครสั้นกว่า

หมายเหตุ control string ใน scanf ( ) ใช้รูปแบบเป็น %*wlx เครื่องหมาย * หมายความว่า "ไม่ต้องสนใจ" ส่วน wlx ยังคงใช้เหมือนใน printf ( )

รูปแบบ %-w.plx ข้างบนนี้เราไม่จำเป็นต้องใช้พร้อมกันทั้งหมด ทั้งนี้ให้เลือกใช้ตามความจำเป็น และเรายังสามารถใช้ escape sequence ต่อไปนี้ผสมไปด้วยก็ได้

\n = newlin

\t tab

\b backspace

\r = cariage return

\f = form feed

\0 = null

\' = single quote

\\ = backslash

\ xxx = octal bit pattern

ลองดูตัวอย่างต่อไปนี้สมมุติว่า c เก็บคำว่า "thailand\0" เอาไว้ ด้านซ้ายมือคือคำสั่งพิมพ์ด้านขวามือคือผลลัพธ์ที่เสนอ

```
printf (" 12345");              ผลลัพธ์คือ   12345
printf ("\n 12345") ;           ผลลัพธ์คือ
                                             12345
printf ("...\b 12345") ;        ผลลัพธ์คือ    12345
printf("how are you") ;         ผลลัพธ์คือ how are you
printf ("%s", c) ;              ผลลัพธ์คือ  thailand
printf ("%7s", c) ;             ผลลัพธ์คือ        thailand
printf ("%-3s", c) ;            ผลลัพธ์คือ  thaniland
printf ("%5.3 f", 12.67814)     ผลลัพธ์คือ 12.678
printf ("%c" , 'a')             ผลลัพธ์คือ  a a
printf("result is%d", 123)      ผลลัพธ์คือ 123
printf ("x=%f y=%d and z=%2.2f", 14.8, 128, 12.46901)
                                ผลลัพธ์คือ  14.8    128    12.46
printf ("%04d", 5)              ผลลัพธ์คือ 0005
```

ขอให้ผู้อ่านทดลองสั่งพิมพ์โดยใช้รูปแบบที่แตกต่างออกไปจากที่ยกให้เห็นตัวอย่างข้างบนเพื่อให้เกิดความมั่นใจและสามารถสั่งพิมพ์ได้ถูกต้อง

7.4 ตัวอย่างโปรแกรม

recursive

ภาษา C นั้นยอมให้ฟังก์ชันเรียกใช้ตัวเองได้ (เรียกว่า recursive) เช่น ในกรณีอนุกรมการคำนวณหาค่าแฟคทอเรียล (factorial) การ search เช่น binary search การหาค่าดอกเบี้ยทบต้น และอื่น ๆ

ตัวอย่างต่อไปนี้เป็นโปรแกรมและผังควบคุมสำหรับหาค่า factorial ซึ่งจะแสดงให้เห็นทั้งกรณี non-recursive และ recursive และ เพื่อให้มองเห็นภาพของ factorial จะขอยกตัวอย่างการคำนวณหาค่า factorial ให้เห็นพอเข้าใจดังนี้

n! (อ่านว่า n factorial) มีค่าเท่ากับ n.(n-1).(n-2)...3.2.1

(n-1)! มีค่าเท่ากับ (n-1)(n-2)...3.2.1

6! มีค่าเท่ากับ **6.5.4.3.2.1 = 720**

ดังนั้น n! = n(n-1)!

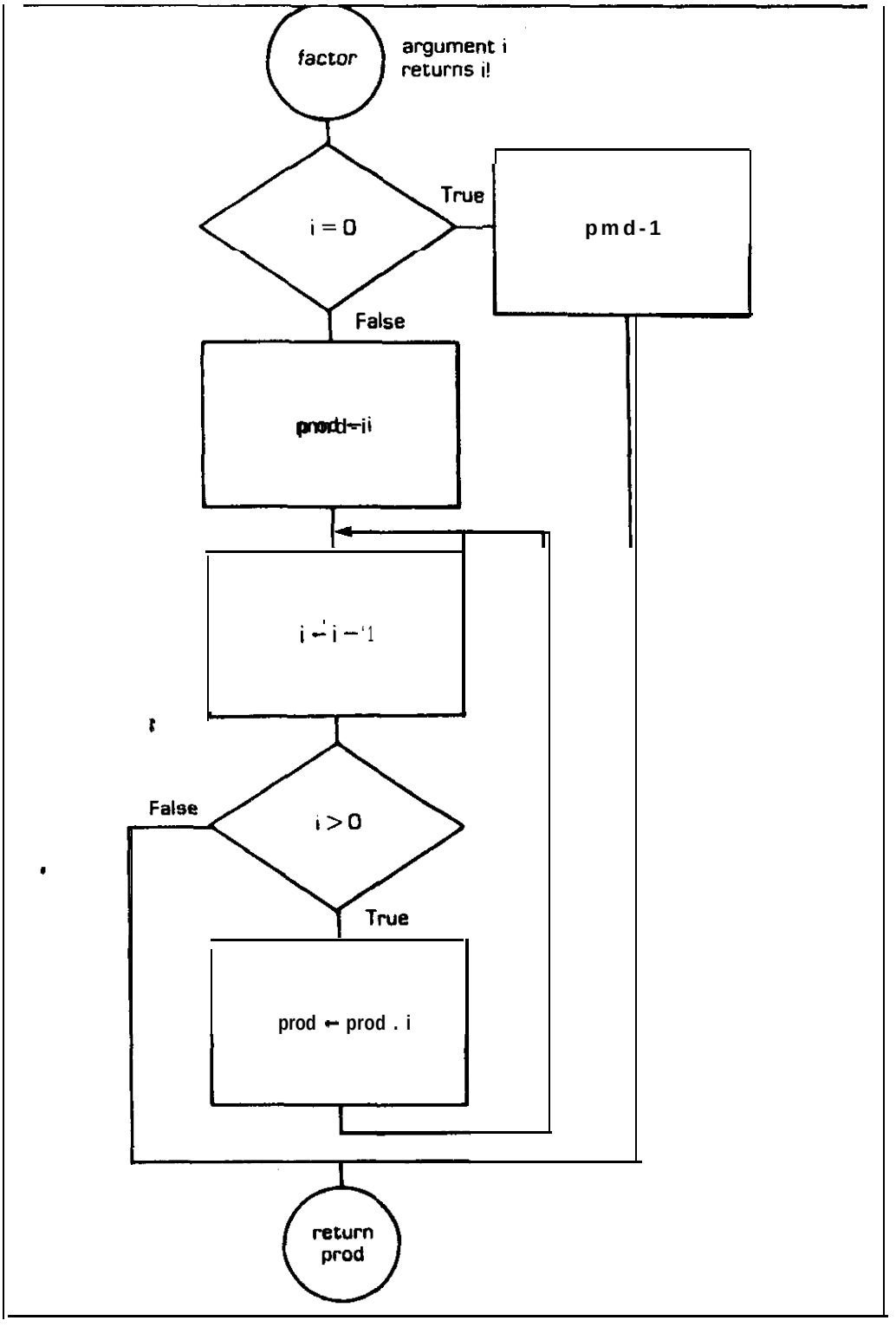
(n-1) = (n-1)(n-2)

6 ! = 6.5! = **6.(5.4.3.2** 1) = 6(120) = 720

จะเห็นได้ว่าค่าของ factorial เกิดจากการนำค่าที่ลดลงไป 1 หน่วยคูณเข้ากับค่าเดิม เรื่อยไปจนถึงค่าที่ลดลงไม่ได้อีกแล้ว (คือ 1) เช่น 6! = 6(6-1)(5-1)(4-1) (3-1)(2-1) ลองดูผังควบคุมและโปรแกรมต่อไปนี้

ก. กรณี non-recursive

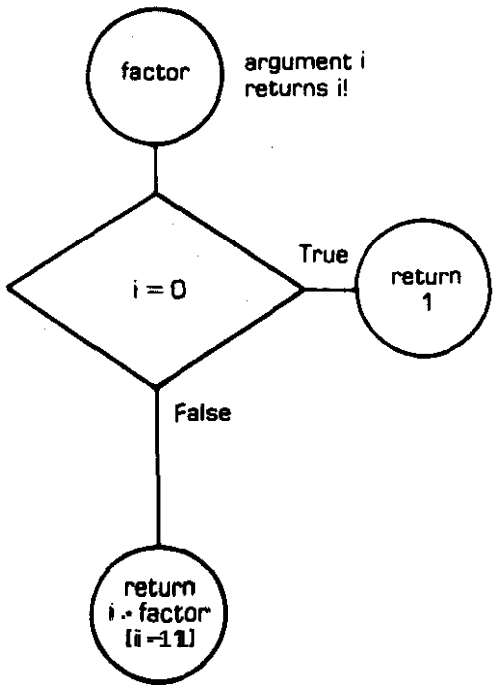
จากผังควบคุมเราสามารถเขียนโปรแกรมได้ดังนี้


factor
argument i
returns i!
i = 0
True
pmd-1
False
prod←ii
i←i−1
i > 0
False
True
prod ← prod . i
return
prod

```
foctor (i) /* compute factorial */
int i ;     /* number to compute factorial */
{
    int prod ;
    if (i = = 0) prod = 1 ;
    else {
            prod = i ;
            wihle (- - i > 0) {
               prod * = i ;
              }
          }
    return prod ;
 }
```

ข. กรณี recursive

จากผังควบคุมเราสามารถเขียนโปรแกรมได้ดังนี้



```
factor (i)  /* compute factorial ● /
int i ;     /* number to compute factorial */
{
    if (i = = 0) return 1 ;
    else  return i*factor (i-1) ;

 }
```