บทที่ 5 Pointer

ภาษา C ได้ชื่อว่าเป็นภาษาการเขียนโปรแกรมที่มีคุณลักษณะเป็น Low-level Language[1] ซึ่งเป็นระดับที่ใกล้กับภาษาเครื่อง สาเหตุประการหนึ่งก็คือ ความสามารถในการอ้างอิงถึงเลขที่อยู่ในหน่วยความจำ โดยใช้ชนิดข้อมูลที่มีชื่อเรียกว่า Pointer ในภาษา C ความสามารถนี้นอกจากทำให้ประสิทธิภาพของโปรแกรมสูงขึ้นแล้ว ยังเพิ่มความยืดหยุ่นในการเขียนโปรแกรมได้อีกด้วย

5.1 Pointer

Pointer หมายถึงชนิดข้อมูลสำหรับเก็บเลขที่อยู่ (address) ของตัวแปร กล่าวคือ ตำแหน่งที่อยู่ในหน่วยความจำของตัวแปร ซึ่งแตกต่างจากชนิดข้อมูลที่กล่าวถึงในบทที่ 7 ตรงที่ชนิดข้อมูลเช่น char, int, float, double ใช้เก็บตัวอักษร จำนวนเต็ม หรือเลขทศนิยม แต่ Pointer ใช้เก็บเฉพาะเลขที่อยู่เท่านั้น

เมื่อเราประกาศตัวแปรตัวหนึ่ง เช่น

```
float x ;
```

เมื่อเราทำการคอมไพล์ คอมไพเลอร์จะทำการรวมเนื้อที่สำหรับเก็บตัวแปรนี้ไว้ในโปรแกรม เมื่อทำการรันโปรแกรม ไม่ว่าเราจะใช้หรือไม่ใช้ตัวแปรนั้นๆในการคำนวณ จะต้องมีการจองเนื้อที่นี้ (จึงมีชื่อเรียกว่า *Static Allocation*) เมื่อเรารันโปรแกรม โปรแกรมจะทำการจองที่ในหน่วยความจำสำหรับบันทึกค่าของตัวแปร x สมมุติให้จองเนื้อที่ในหน่วยความจำที่เลขที่อยู่ 0x1000[2] สำหรับตัวแปร x การที่เราอ้างอิงถึงค่าของ x ก็คือการที่เราอ้างอิงถึงเลขที่อยู่ที่ 0x1000 ในหน่วยความจำนั่นเอง อย่างเช่น ถ้าเรากำหนดให้ค่า x เท่ากับ 0

```
x = 0 ;
```

ก็จะมีความหมายว่า เรานำค่า 0 ไปเก็บไว้ในเลขที่อยู่ที่ 0x1000 ในหน่วยความจำ ในกรณีนี้ เราเรียก 0x1000 ว่าเป็นเลขที่อยู่ของ x หรือใช้สำนวนว่าเป็น *pointer* to x

การประกาศตัวแปร pointer ทำได้ดังนี้

```
data type   *variable_name ;
```

ยกตัวอย่าง เช่น

```
float *a_pointer ; /* define a pointer to integer type variable */
```

<u>หมายเหตุ</u> ข้อสังเกตเกี่ยวกับตัวแปร Pointer

- การประกาศ pointer จะต้องกำหนดชนิดข้อมูลก็เพราะแต่ละชนิดข้อมูลมีความยาวไม่เท่ากัน เช่น int ใช้ 2 หรือ 4 ไบต์ float ใช้ 4 ไบต์ เป็นต้น ดังนั้น เวลาอ้างอิงถึงตัวแปรนั้น จะต้องรู้ว่า ต้องอ้างอิงถึงกี่ไบต์ เช่น การอ้างอิงถึง x ในตัวอย่างข้างต้น จะต้องอ่านข้อมูลจำนวน 4 ไบต์นับจากเลขที่อยู่ที่ 0x1000 เป็นต้น
- หน่วยความจำที่ใช้ในการเก็บตัวแปร Pointer แต่ละตัวขึ้นกับโปรเซสเซอร์ที่ใช้ เนื่องจากคอมพิวเตอร์ส่วนใหญ่ในปัจจุบันใช้โปรเซสเซอร์ขนาด 32 บิต จึงใช้ 4 ไบต์ในการแสดงเลขที่อยู่ ดังนั้น ไม่ว่าจะเป็นชนิดข้อมูลใด ตัวแปร Pointer ทุกตัวจะใช้หน่วยความจำ 4 ไบต์ในการเก็บเลขที่อยู่

---

[1] Low-level Language หมายถึง High-level Language ที่มีคุณสมบัติที่ใกล้เคียงกับภาษาเครื่อง (Machine Code) กล่าวคือ มีรูปแบบเหมือน High-level Language แต่สามารถเขียนโปรแกรมในลักษณะ Machine โค้ดได้ เช่น การอ้างอิงถึงเลขที่อยู่การคำนวณแบบ Bitwise เป็นต้น

[2] ในความเป็นจริง จะจองเนื้อที่ 0x1000 - 0x1003 เนื่องจากข้อมูลชนิด float 1 ตัวใช้เนื้อที่ในหน่วยความจำ 4 ไบต์

5.2 ตัวดำเนินการ Pointer (Pointer Operator)

ตัวดำเนินการที่เกี่ยวข้องกับการใช้งานตัวแปร Pointer ได้แก่ ตัวดำเนินการ dereference (*) และตัวดำเนินการ address-of (&) ซึ่งล้วนเป็นตัวดำเนินการประเภทยูนารี ดังนั้น ถึงแม้จะใช้สัญลักษณะเดียวกันกับตัวดำเนินการคูณ ก็ไม่เกิดความสับสน เพราะตัวดำเนินการคูณเป็นตัวดำเนินการประเภทไบนารี

ตารางที่ 5.1 สรุปเกี่ยวกับตัวดำเนินการ Pointer

ตารางที่ 5.1 ตัวดำเนินการ Pointer

ตัวดำเนินการ	ชื่อเรียก	ตัวถูกดำเนินการ	ผลลัพธ์ที่ได้
*	Dereference	ตัวแปร Pointer	ค่าที่เก็บในหน่วยความจำตามเลขที่อยู่นั้น
&	Address-of	ตัวแปรเก็บค่า	เลขที่อยู่ของหน่วยความจำสำหรับตัวแปรนั้น

ตัวอย่าง : ถ้าเราประกาศตัวแปรดังต่อไปนี้

```
int    thing, other ;      /* define a thing of integer type*/
int    *thing_ptr ; /* define a pointer to a thing*/
```

ความหมายของการใช้ Operator จะเป็นดังต่อไปนี้

thing, other เป็นค่า ให้สังเกตว่าในการประกาศ ***thing*** และ **other** ไม่มีเครื่องหมาย * ดังนั้น ตัวแปรทั้งสองนี้ จึงไม่ใช่ pointer ในการใช้ตัวแปรเหล่านี้ ใช้แบบกำหนดค่าทั่วไป เช่น thing = 4 ;

&thing เป็น pointer to thing ***thing*** หมายถึงค่าของตัวแปร ***thing*** ดังนั้น เครื่องหมาย & (address-of operator) จะหมายถึงเลขที่อยู่ของตัวแปร (a pointer) ดังนั้น ***&thing*** เป็น pointer เช่น

```
thing_ptr = &thing ;       /* Point to a thing*/
*thing_ptr = 5 ;           /* Set "thing" to 5 */
```

&other เหมือนกับ &thing

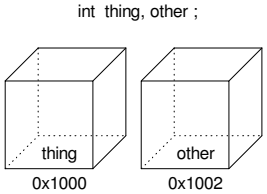
thing_ptr เป็น pointer เครื่องหมาย * ในการประกาศ หมายความว่า ให้ ***thing_ptr*** เป็นตัวแปร pointer

*thing_ptr เป็นค่า เนื่องจาก ***thing_ptr*** เป็น pointer ดังนั้น เครื่องหมาย * (dereference operator) จะเป็นการกำหนดให้ C ไปอ่านค่าที่เก็บอยู่ในเลขที่อยู่ที่ pointer ชี้อยู่ เช่น

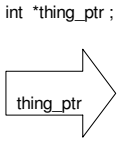
```
*thing_ptr = 5 ;
/* Assign 5 to an integer that thing_ptr points to */
```

สังเกตว่า ***thing_ptr*** ในขณะนี้ ชี้ที่ ***thing*** (***thing_ptr = &thing***) อยู่ ซึ่งเราสามารถเปลี่ยนให้ ***thing_ptr*** ไปชี้ที่ตัวแปรตัวอื่นก็ได้

การประกาศตัวแปร ***thing*** และ ***other*** จะเป็นในลักษณะดังรูปที่ 5.1 โดยสมมุติให้ โปรแกรมจองที่ที่เลขที่อยู่ 0x1000, 0x1002 สำหรับ ***thing*** และ ***other*** ตามลำดับ และเป็นแบบจำนวนเต็มขนาด 2 ไบต์ การประกาศตัวแปร pointer จะเป็นในลักษณะดังรูปที่ 5.2



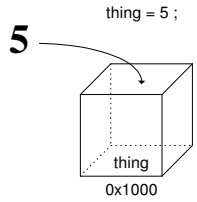
รูปที่ 5.1 การประกาศตัวแปรจำนวนเต็ม



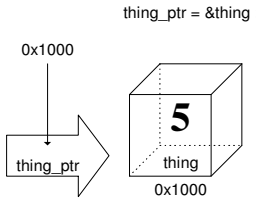
รูปที่ 5.2 การประกาศ Pointer สำหรับจำนวนเต็ม

การกำหนดให้ ***thing*** เท่ากับ 5 จะเป็นดังรูปที่ 5.3 กล่าวคือเหมือนกับการนำค่า *5* เข้าไปเก็บไว้ในกล่องที่ชื่อว่า *thing* และการกำหนดให้ ***thing_ptr*** ชี้ที่ ***thing*** หรือกำหนดให้ค่าของ pointer ***thing_ptr*** เท่ากับเลขที่อยู่ของ ***thing***

จะเป็นดังรูปที่ 5.4 กล่าวคือ นำค่าของเลขที่อยู่ที่เอาไว้สำหรับเก็บค่าตัวแปร *thing* (หรือ เลขที่ของกล่องชื่อ *thing*) ไปเก็บไว้ในตัวแปร Pointer *thing_ptr* หรือเป็นการกำหนดให้ *thing_ptr* ไปชี้ที่ *thing*

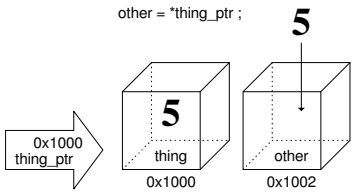


<u>รูปที่ 5.3</u> thing = 5

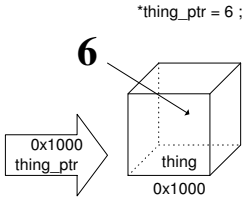


<u>รูปที่ 5.4</u> thing_ptr = &thing

ถ้ากำหนดให้ ***other*** เท่ากับค่าที่อยู่ในเลขที่อยู่ที่ ***thing_ptr*** ชี้อยู่ จะเป็นดังรูปที่ 5.5 และในการเปลี่ยนแปลงค่าในเลขที่อยู่ที่ ***thing_ptr*** จะเป็นดังรูปที่ 5.6



<u>รูปที่ 5.5</u> other = *thing_ptr



<u>รูปที่ 5.6</u> *thing = 6

ตัวอย่างต่อไปนี้ เป็นตัวอย่างแสดง pointer Operation ที่ไม่ถูกต้อง

*thing ไม่ถูกต้อง เพราะเป็นการสั่งให้ C อ่านค่าในเลขที่อยู่ที่ ***thing*** ชี้อยู่ แต่เนื่องจาก ***thing*** ไม่ใช่ pointer ไม่สามารถเก็บเลขที่อยู่ได้ *thing จึงไม่ถูกต้อง

&thing_ptr ไม่ผิด แต่ความหมายจะเปลี่ยนไปจากปกติ กล่าวคือ ***thing_ptr*** เป็น pointer และเครื่องหมาย & (address of operator) เป็นการอ้างอิงถึงเลขที่อยู่ที่ pointer กำลังชี้อยู่ ซึ่งในที่นี้สิ่งที่กำลังชี้อยู่ ก็คือ ***thing_ptr*** ซึ่งเป็น pointer ดังนั้น ผลที่ได้จาก &thing_ptr จะเป็น <u>pointer to pointer</u>

<u>ตัวอย่างที่ 5.1</u> : ตัวอย่างการใช้ตัวแปร Pointer

```
/* This program demonstrates the basic usage of pointer variable */
#include <stdio.h>
int main()
{
  int   thing_var ;/* define a variable for thing */
  int   *thing_ptr ;/* define a pointer to thing */
  thing_var = 2 ;/* assigning a value to thing */
  printf("Value of thing_var is %d\n", thing_var) ;
  printf("Address of thing_var is %X\n", &thing_var) ;
  thing_ptr = &thing_var ;/* make the pointer point to thing */
  printf("Value of thing_ptr is %X\n", thing_ptr) ;
  *thing_ptr = 5 ;
  /* thing_ptr points to thing_var so thing_var changes to 5 */
  printf("Value of thing_var is changed to %d\n", thing_var) ;
  printf("Value of the location where thing_ptr points to is %d\n",
        *thing_ptr) ;
  return 0;
}
```

<u>ผลการรันโปรแกรม</u>

```
Value of thing_var is 2
Address of thing_var is EFFFFBEC
Value of thing_ptr is EFFFFBEC
Value of thing_var is changed to 5
Value of the location where thing_ptr points to is 5
```

NULL

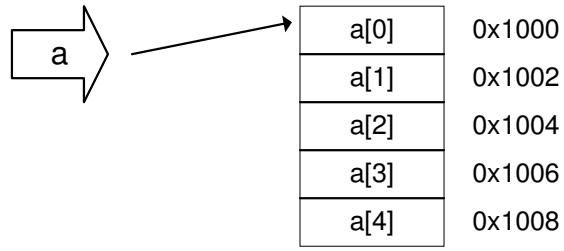
NULL เป็น pointer พิเศษ ที่ไม่ได้ชี้ที่เลขที่อยู่ไหนเลย (ใน C คอมไพเลอร์ทั่วไป จะกำหนดให้มีค่าเท่ากับ 0) กล่าวคือ เป็น Pointer ที่ไม่ได้เก็บค่าเลขที่อยู่ใดๆไว้ โดยทั่วไป NULL Pointer มักจะใช้สำหรับการแสดงเงื่อนไขในการจองเนื้อที่ในหน่วยความจำ ยกตัวอย่างฟังก์ชัน malloc ที่ใช้จองเนื้อที่ในหน่วยความจำ ถ้าจองไม่ได้เนื่องจากมีเนื้อที่ไม่เพียงพอ ฟังก์ชันนี้จะคืนค่าเป็น NULL Pointer

5.3 Pointer กับอาร์เรย์

เวลาเราประกาศอาร์เรย์ ยกตัวอย่างเช่น

```
int    a[5] = { 1, 2, 3, 4, 5 } ;
```

ชื่ออาร์เรย์ *a* จะหมายถึงค่าคงที่ที่บันทึกค่าเลขที่อยู่ของหน่วยความจำที่ใช้เก็บตัวประกอบตัวแรกของอาร์เรย์นั้น ซึ่งก็คือ a[0] ดังรูปที่ 5.7 ดังนั้น a กับ &a[0] จะมีค่าเดียวกัน และนิพจน์ a[0] มีค่าเท่ากับ *a หรือเท่ากับ 1



รูปที่ 5.7 pointer กับอาร์เรย์

นอกจากนี้ ในภาษา C เราสามารถทำการคำนวณโดยใช้ pointer ได้ เช่น a + 1 จะหมายถึง หน่วยความจำที่ถัดจาก a ไป 1 Cell ซึ่ง Cell ในที่นี้ จะขึ้นอยู่กับจำนวนไบต์ที่ใช้ในการแสดงชนิดข้อมูลที่เป็นตัวประกอบของอาร์เรย์อยู่ ดังนั้น ถ้าชนิดข้อมูลของตัวประกอบของ a เป็น int ขนาด 2 ไบต์ตามที่ได้แสดงในรูปที่ 12.7 a + 1 จะมีค่าเท่ากับ 1000 + 2 = 1002 ซึ่งเป็นเลขที่อยู่ของหน่วยความจำที่ใช้เก็บ a[1] ทำให้ Expression *(a + 1) มีค่าเท่ากับ 2 ตารางที่ 12.2 เป็นตัวอย่างการคำนวณแบบ Pointer ในกรณีที่เป็นอาร์เรย์ในรูปที่ 12.7 และ b เป็นตัวแปร Pointer สำหรับชี้ไปที่ข้อมูลชนิด int

ตารางที่ 5.2 ตัวอย่างการคำนวณแบบ Pointer ในกรณีที่เป็นอาร์เรย์ในรูปที่ 5.7

นิพจน์	ชนิดของค่า	ค่า/เลขที่อยู่
a	Address	1000
*a	Value	1
a + 1	Address	1002
*(a + 1)	Value	2
a + 3	Address	1006
*(a + 3)	Value	4
หลัง b = a + 2;		
b	Address	1004
*b	Value	3
หลัง b++;		
b	Address	1006
*b	Value	4
b - 2	Address	1002
*(b - 2)	Value	2
หลัง b--;		
b	Address	1004
*b	Value	3

ตัวอย่าง 5.2 การใช้ตัวแปร Pointer ในการแสดงอาร์เรย์

```
/* This program demonstrates how to use pointer to represent array */
#include <stdio.h>
int main()
```

```
{
       int    a[5] = {0, 1, 2, 3, 4} ;
       int    *a_ptr ;
       a_ptr = a ;                /* a_ptr points to a[0] */
       printf("value of *a_ptr %d\n", *a_ptr) ;       /* TEST */
       a_ptr = &a[0] ;            /* the same as above */
       printf("value of *a_ptr %d\n", *a_ptr) ;       /* TEST */
       a_ptr++ ;           /* increment a_ptr so a_ptr points to a[1] */
       printf("value of *a_ptr %d\n", *a_ptr) ;
       a_ptr = a + 3 ;            /* a_ptr points to a[3] */
       printf("value of *a_ptr %d\n", *a_ptr) ;
       printf("value of *a_ptr %d\n", *(a_ptr++)) ;
       printf("value of *a_ptr %d\n", *a_ptr) ;
       return 0;
}
```

ผลการรันโปรแกรม

```
value of *a_ptr 0
value of *a_ptr 0
value of *a_ptr 1
value of *a_ptr 3
value of *a_ptr 3
value of *a_ptr 4
```

5.4 Pointer กับสตริง

ดังที่ได้กล่าวไว้ข้างต้นแล้วว่า สตริงก็คืออาร์เรย์ชนิดหนึ่งที่มีชนิดข้อมูลของตัวประกอบเป็นตัวอักษร ดังนั้น เราก็สามารถใช้ pointer แสดงสตริงได้เฉกเช่นเดียวกับอาร์เรย์ทั่วไป การกำหนดค่าเริ่มต้นในกรณีที่ใช้ pointer ทำได้ดังนี้

```
char          *p = "xyz" ;
```

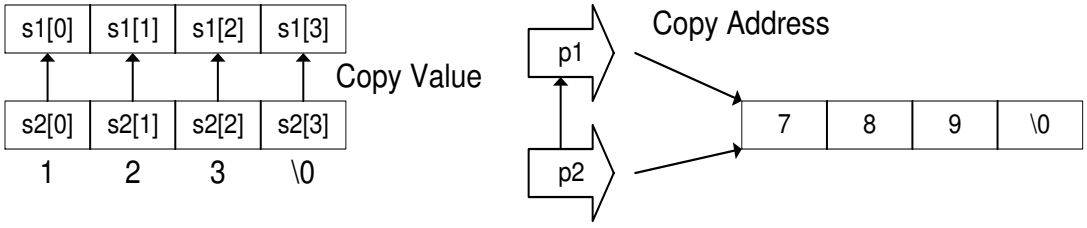
ในกรณีนี้ p จะชี้ไปที่เลขที่อยู่เริ่มต้นของหน่วยความจำที่ใช้ในการบันทึกข้อมูล xyz

ตัวอย่างที่ 5.3 การเปรียบเทียบการใช้ Pointer กับการใช้ฟังก์ชันสตริง

```
#include <string.h>
#include <stdio.h>
int main()
{
       char   s1[10], s2[] = "123" ;
       /* notice that it's necessary to indicate the dimension of s1 */
       char   *p1, *p2 = "789" ;
       /* ประกาศ pointer2 p1 and p2, then initialize p2 to point to
       “789” */
       strcpy(s1, s2) ;    /* copy string s2 to s1 */
       p1 = p2 ;           /* Make p1 point to p2 */
       printf(" s1 : %s    p1 points to %s\n", s1, p1) ;
       return 0;
}
```

ผลการรันโปรแกรม

```
s1 : 123       p1 points to 789
```



รูปที่ 5.8 เปรียบเทียบการใช้ฟังก์ชัน strcpy กับการใช้ pointer

5.5 อาร์เรย์ของ pointer

ในภาษา C เราสามารถสร้างอาร์เรย์ที่มีตัวประกอบเป็น pointer ได้ ซึ่งสามารถใช้ประโยชน์ในการสร้างอาร์เรย์หลายมิติ อาร์เรย์ของ pointer มักจะใช้กันมากในการสร้างอาร์เรย์ของสตริง โดย pointer แต่ละตัวในอาร์เรย์จะชี้ไปที่สตริงแต่ละอัน ซึ่งมีผลทำให้ประหยัดเนื้อที่ในหน่วยความจำได้

วิธีการประกาศอาร์เรย์ของ Pointer จะมีรูปแบบดังนี้

```
data type   *variable_name[dimension] ;
```

ยกตัวอย่าง เช่น

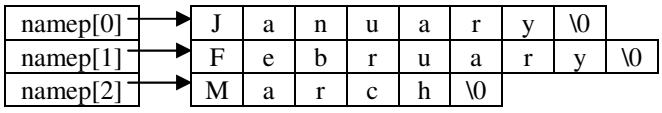
```
char        *c_ptr[3] ;
```

เปรียบเทียบการประกาศ 2 ตัวอย่างต่อไปนี้

```
(1) char            namev[3][10] = { "January", "February", "March" } ;
(2) char            *namep[3] = { "January", "February", "March" } ;
```

(1) เป็นการใช้อาร์เรย์ 2 มิติ และ (2) เป็นการใช้อาร์เรย์ของ pointer โดยจะแตกต่างกันดังแสดงในรูปที่ 5.9

J	a	n	u	a	r	y	\0		
F	e	b	r	u	a	r	y	\0	
M	a	r	c	h	\0				



รูปที่ 5.9อาร์เรย์2 มิติ (ซ้าย) กับอาร์เรย์ของ pointer (ขวา)

5.6 Command-line argument

ในกรณีที่สั่งงานคำสั่งที่ command prompt ของ windows หรือที่ command line ของ Linux (หรือ Unix) เช่น

```
COPY  A.TXT  A:
```

เราจะเรียก A.TXT และ A: ว่าเป็นอาร์กิวเมนต์บรรทัดคำสั่ง (Command-line Argument) ในทำนองเดียวกัน ถ้าเราเขียนโปรแกรมหนึ่ง แล้วคอมไพล์ออกมาเป็นไฟล์สั่งงาน (Executable File) สมมุติว่าตั้งชื่อเป็น pro.exe ใน DOS หรือ pro ที่สามารถสั่งงานได้ใน UNIX แล้วต้องการให้มีการรับค่า (ซึ่งก็คือ อาร์กิวเมนต์) ตอนที่สั่งรัน เช่น

```
pro  a b
```

เราจะต้องกำหนดให้ฟังก์ชัน main มีอาร์กิวเมนต์ ซึ่งอาร์กิวเมนต์ที่ฟังก์ชัน main จะรับได้มีอยู่ 2 อันคือ argc (argument count : จำนวนอาร์กิวเมนต์ ใน Command-line ซึ่งจะนับรวมชื่อโปรแกรมด้วย) และ argv (argument vector : ค่าของ argument แต่ละตัว ซึ่งเป็น string) โดยทำในลักษณะดังต่อไปนี้

ANSI C		TRADITIONAL C

`void main(int argc, char *argv[])`	or	`void main(argc, argv)`
		`int  argc ;`
		`char  *argv[] ;`

ตัวอย่าง 5.4 : การใช้ Command-line arguments (ใช้อาร์เรย์ของ pointer (*argv[]) version)

```
#include <stdio.h>
int main(int argc, char *argv[])
{
      int   i ;
      for (i = 0 ; i < argc ; i++)
            printf("argv[%d] = %s\n", i, argv[i]) ;
      return 0;
}
```

ผลการรันโปรแกรม เมื่อป้อนคำสั่ง ***a.out a b 1 2*** ที่ Command Line แล้วจะได้ผลดังนี้

```
leibniz:~/program$ a.out a b 1 2
argv[0] = a.out
argv[1] = a
argv[2] = b
argv[3] = 1
argv[4] = 2
```

5.7 Pointer to pointer

pointer หมายถึงตัวแปรที่สำหรับบันทึกเลขที่อยู่ของตัวแปรต่าง ๆ ในทำนองเดียวกัน pointer to pointer ก็คือ ตัวแปรที่สำหรับบันทึก pointer หรือเลขที่อยู่ของตัวแปร pointer หรือว่าชี้ไปยัง pointer อีกตัวหนึ่ง ดังได้กล่าวข้างต้นแล้วว่า ตัวแปร pointer มักใช้แสดงอาร์เรย์ 1 มิติ ในทำนองเดียวกัน pointer to pointer ใช้แสดงอาร์เรย์หลายมิติหรืออาร์เรย์ของ Pointer

วิธีการประกาศ Pointer to Pointer จะมีรูปแบบดังนี้

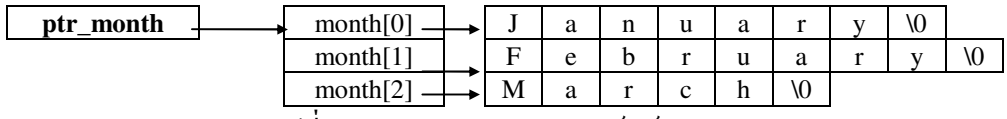
```
data type   **pointer_name ;
```

ยกตัวอย่าง เช่น

```
int         **ptr_ptr ;
```

ตัวอย่างที่ 5.5 : ตัวอย่างแสดงความสัมพันธ์ระหว่าง Pointer to Pointer กับอาร์เรย์ของ Pointer

```
#include <stdio.h>
int main()
{
       char   **ptr_month ; /* pointer to pointer declaration */
       char   *month[3] = {"January", "February", "March"} ;
                                  /*อาร์เรย์of pointer declaration */
       int    i ;
       ptr_month = month ; /* ptr_month points to month */
       for (i = 0 ; i < 3 ; i++)
              printf("%s  ", *(ptr_month + i)) ;
       printf("\n") ;
       return 0;
}
```



รูปที่ 5.10 Pointer to Pointer กับอาร์เรย์ของ Pointer

ผลการรันโปรแกรม

```
January  February  March
```

ตัวอย่างที่ 5.6 : ตัวอย่างการใช้งาน Pointer, Pointer to Pointer และอาร์เรย์ 2 มิติ

```
#include <stdio.h>
int main()
{
       int    thing[3][5] =       /* define multi-dimensional array*/
              {{0, 1, 2, 3, 4}, {5, 6, 7, 8, 9}, {10, 11, 12, 13, 14}};
       int    *ptr[3] ;           /* define array of pointer */
       int    **ptr_ptr ;   /* define pointer to pointer */
       int    i ;
       ptr[0] = thing[0] ; /* ptr[0] points to thing[0] */
       for (i = 0 ; i < 3 ; i++)
              printf("value of *(ptr[0]+%d) = %d \n", i, *(ptr[0] + i)) ;
              /* *(ptr[0] + i) means the value in address ptr[0] + i */
       printf("\n") ;
       ptr[1] = thing[2] ;
       for (i = 0 ; i < 3 ; i++)
              printf("value of *(ptr[1]+%d) = %d \n", i, *(ptr[1] + i)) ;
       printf("\n") ;
       ptr[2] = thing[1] ;
       ptr_ptr = ptr ;            /* ptr_ptr points to ptr */
       for (i = 0 ; i < 3 ; i++)
              printf("value of *(ptr_ptr+%d)[0] = %d\n",i,*(ptr_ptr + i)[0]);
       printf("\n") ;
       for (i = 0 ; i < 3 ; i++)
          printf("value of *(*(ptr_ptr+%d)+3) = %d\n",i,*(*(ptr_ptr + i)+3));
       return 0;
}
```

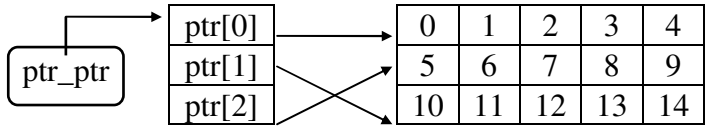
ผลการรันโปรแกรม

```
value of *(ptr[0]+0) = 0
value of *(ptr[0]+1) = 1
value of *(ptr[0]+2) = 2

value of *(ptr[1]+0) = 10
value of *(ptr[1]+1) = 11
value of *(ptr[1]+2) = 12

value of *(ptr_ptr+0)[0] = 0
value of *(ptr_ptr+1)[0] = 10
value of *(ptr_ptr+2)[0] = 5

value of *(*(ptr_ptr+0)+3) = 3
value of *(*(ptr_ptr+1)+3) = 13
value of *(*(ptr_ptr+2)+3) = 8
```



รูปที่ 5.11 Pointer, Pointer to Pointer และอาร์เรย์ 2 มิติ (ตัวอย่าง 5.6)

ตัวอย่างที่ 5.7 : Command-line Arguments (ใช้ pointer to pointer (**argv))

```
#include <stdio.h>
int main(int argc, char **argv)/* use **argv instead of *argv[] */
{
	int	i ;
	for (i = 0 ; i < argc ; i++)
		printf("argv[%d] = %s\n", *(argv + i)) ;
	return 0;
}
```

5.8 Command-line argument conversion function

ฟังก์ชันที่เกี่ยวข้องกับการแปลงอาร์กิวเมนต์บรรทัดคำสั่งที่ใช้บ่อยครั้ง ได้แก่

```
atoi  convert argument (string) to int type data
atof  convert argument (string) to float type data.
```

ข้อควรระวัง function atoi และ atof ถูกประกาศในไฟล์ stdlib.h ดังนั้น เวลาใช้ จะต้องทำการรวมไฟล์นี้เข้ามาด้วย

ตัวอย่างที่ 5.8 : Usage of atoi and atof

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
	int	i ;
	for (i = 1 ; i < argc ; i++)
		printf("argv[%d] = %s -> %d %f\n",
			i, argv[i], atoi(argv[i]), atof(argv[i])) ;
	return 0;
}
```

ผลการรันโปรแกรม เมื่อป้อนคำสั่ง ***a.out 1 2 3*** ที่ Command Line แล้วจะได้ผลดังนี้

```
leibniz:~/program$ a.out 1 2 3
argv[1] = 1 -> 1 1.000000
argv[2] = 2 -> 2 2.000000
argv[3] = 3 -> 3 3.000000
```

5.9 Memory Allocation

ในกรณีที่เราประกาศอาร์เรย์เช่น

```
	int	a[10] ;
```

คอมไพเลอร์ ก็จะจัดการสั่งให้โปรแกรมจองเนื้อที่ในหน่วยความจำให้เพียงพอที่จะใช้บันทึกอาร์เรย์อันนี้ ในกรณีนี้ จะจองเนื้อที่ไว้สำหรับ 10 จำนวนเต็มหรือหน่วยความจำ 40 ไบต์ที่อยู่ติดกัน (ยกตัวอย่าง เช่นเลขที่อยู่ที่ 0x1000 - 0x1019) แต่ในกรณีที่เราใช้ pointer ในการแสดงอาร์เรย์ ถ้าเพียงแต่เรา ประกาศ ตัวแปร pointer เช่น

```
int   *ptr_a ;
```

คอมไพเลอร์ ก็เพียงแต่จองเนื้อที่สำหรับตัวแปร *ptr_a* เท่านั้น (กล่าวคือ 4 ไบต์) ไม่ได้จองเนื้อที่สำหรับอาร์เรย์ ดังนั้น เราจะต้องสั่งให้โปรแกรมทำการจองเนื้อที่ที่จะใช้ในการบันทึกอาร์เรย์แล้วให้ pointer ptr_a ไปชี้ที่เลขที่อยู่แรกสุดของเนื้อที่ส่วนที่ทำการจอง จึงจะมีค่าเท่ากับการ ประกาศอาร์เรย์ ซึ่งการจองเนื้อที่ในหน่วยความจำนี้ เราเรียกว่า *Memory Allocation*

ในภาษา C มี Function ที่ใช้เกี่ยวกับการจองเนื้อที่ในหน่วยความจำ ดังนี้ คือ

- *malloc(size)* จะทำหน้าที่จองเนื้อที่ในหน่วยความจำขนาดเท่ากับ *size* (หน่วยเป็น Byte)
- *calloc(n,size)* คล้ายกับ *malloc* แต่จะจองเนื้อที่เท่ากับ *n* Block ซึ่งแต่ละ Block มีขนาดเท่ากับ *size*
- *free(ptr)* จะทำหน้าที่ยกเลิกการจองเนื้อที่ในหน่วยความจำนั้น เพื่อที่สามารถนำไปใช้ในงานอื่นได้
- *realloc(ptr,size)* คล้ายกับ *malloc* คือจะรับค่า pointer ของเนื้อที่ที่จองไว้ก่อนแล้ว แล้วเปลี่ยนเนื้อที่การจองให้เท่ากับ *size*

และมีตัวดำเนินการที่เกี่ยวข้อง คือ

- *sizeof*(*type*) or *sizeof*(*variable*) จะทำหน้าที่คำนวณขนาดของชนิดของข้อมูล หรือขนาดของตัวแปร

ในการใช้ function เหล่านี้ ต้องทำการ include <stdlib.h> หรือ <alloc.h> ด้วย

ยกตัวอย่าง เช่น

```
int_ptr = (int *)malloc(100 * sizeof(int))
```

หมายถึง ให้โปรแกรมจองเนื้อที่ในหน่วยความจำ 100 × ขนาดของ int (= 4 ไบต์) = 400 ไบต์ แล้วกำหนดให้ int_ptr เท่ากับเลขที่อยู่แรกสุดของเนื้อที่ที่จอง

ตัวอย่างที่ 5.9 การคูณเมตริกซ์โดยจองเนื้อที่สำหรับเมตริกซ์ด้วย Memory Allocation

```
#include <stdio.h>
#include <alloc.h>
#include <stdlib.h>/* function exit() is declared in <stdlib.h> */
int main()
{
       int    *matrixa, *matrixb, *matrixc ;
       int    i, j, k, m, n ;
       matrixa = (int *) malloc(100 * sizeof(int)) ;
       matrixb = (int *) malloc(100 * sizeof(int)) ;
       matrixc = (int *) malloc(100 * sizeof(int)) ;
       if ((matrixa == NULL) || (matrixb == NULL) ||
             (matrixc == NULL)) {
             printf("Insufficient Memory !\n") ;
             exit(1) ;
       }             /* Notice how to use NULL Pointer */
       printf("Enter Row :") ;           scanf("%d", &m) ;
       printf("Enter Column :") ;        scanf("%d", &n) ;
       for (i = 0 ; i < m*n ; i++)  /* Input Matrix a */
             scanf("%d", matrixa + i) ;
       for (i = 0 ; i < m*n ; i++)  /* Input Matrix b */
             scanf("%d", matrixb + i) ;
       for (i = 0 ; i < m*n ; i++)  /* Calculate Matrix a + b */
             *(matrixc + i) = *(matrixa + i) + *(matrixb + i) ;
       /* Display the result */
       for (i = 0 ; i < m ; i++) {
             printf("|") ;
             for (j = 0 ; j < n ; j++) {
```

```
                k = i*n + j ;
                printf(" %2d ", *(matrixa + k)) ;
        }
        if (i == 0)
                printf("| + |") ;
        else
                printf("|   |") ;
        for (j = 0 ; j < n ; j++) {
                k = i*n + j ;
                printf(" %2d ", *(matrixb + k)) ;
        }
        if (i == 0)
                printf("| = |") ;
        else
                printf("|   |") ;
        for (j = 0 ; j < n ; j++) {
                k = i*n + j ;
                printf(" %2d ", *(matrixc + k)) ;
        }
        printf("|\n") ;
    }
    return 0;
}
```

ผลการรันโปรแกรม

```
Enter Row :2
Enter Column :2
1 2 3 4
1 2 0 1
|  1   2 | + |  1   2 | = |  2   4 |
|  3   4 |   |  0   1 |   |  3   5 |
```

แบบฝึกหัด

1. จงหาที่ผิดในโค้ดต่อไปนี้

```
char  var1, ptr1 ;
var1 = 'X' ;
ptr1 = &var1 ;
```

2. จงหาที่ผิดในโค้ดต่อไปนี้

```
char  c = 'A' ;
char  *p ;
p = c ;
```

3. ในโค้ดต่อไปนี้

```
char  c = 'A', d = 'B', e ;
char *p1 = &c, *p2 = &d, *p3 ;
p3 = p1 ;
p1 = p2 ;
p2 = p3 ;
p3 = &e ;
*p3 = *p1 ;
*p1 = *p2 ;
*p2 = *p3 ;
```

สมมุติให้เลขที่อยู่ของหน่วยความจำที่ใช้เก็บค่า c, d และ e เท่ากับ 2530, 4732 และ 6628 ตามลำดับ จงบอกค่าของ Expression ต่อไปนี้ หลังจากที่ทำตามประโยคคำสั่งที่ได้เขียนไว้ทั้งหมด

a) &c b) c c) d d) e e) p2 f) p3 g) *p1 h) *p2 i) p3

4. จงหาที่ผิดในโค้ดต่อไปนี้

```
float val = 99.99 ;
float ptr1, *ptr2 ;
ptr1 = &val ;
ptr2 = &ptr1 ;
```

5. จงหาที่ผิดในโค้ดต่อไปนี้

```
float **p1, *p2 ;
p2 = &p1 ;
```

6. จงหาที่ผิดในโค้ดต่อไปนี้

```
char  c = 'Z' ;
char  **p, *q ;
q = &c ;
p = &q ;
printf(“%c”, *p) ;
```

7. จงหาที่ผิดในโค้ดต่อไปนี้

```
float reals[3] = {789.5, 234.5, 543.2} ;
float *ptr ;
ptr = &reals ;
printf("%f", *ptr) ;
```

8. จงหาที่ผิดในโค้ดต่อไปนี้

```
char  letter[] = {'M', 'u', 'l', 'a', 'n'} ;
char  *p ;
*p = &letter[4] ;
```

9. จงบอกผลที่ได้จากโค้ดต่อไปนี้

```
float reals[10] ;
*(reals + 1) = 555.5 ;
*reals = *(reals + 1) ;
*(reals + 2) = *reals/*(reals + 1) ;
printf(“%f %f”, reals[0], reals[2]) ;
```

10. จากโปรแกรม

```
1    #include <stdio.h>
2    void main()
3    {
4      int  *ptr, **ptr_ptr, a, b[10], i ;
5      scanf("%d", &a) ;
6      b[0] = a + 2 ;
7      b[1] = a - 1 ;
8      for (i=2 ; i<10; i++)
9        b[i] = b[i-2] - b[i-1] ;
10     ptr = &a ;
11     ptr_ptr = &ptr ;
12     ptr = b + 2 ;
13     printf("%d %d \n", *(*ptr_ptr), *ptr) ;
14     for (i=1 ; i<8; i+=2)
15       printf("%d ", *(ptr + i)) ;
16     for (i=0 ; i<8; i+=2)
```

```
17        printf("%d ", *(*ptr_ptr+i)) ;
18    }
```

จงตอบคำถามต่อไปนี้ โดยสมมุติว่าป้อนค่า a ให้เท่ากับ 5

a) หลังบรรทัดที่ 9 b[0], *b, *(b+5) มีค่าเท่ากับเท่าไรบ้าง

b) หลังจากคำสั่งในบรรทัดที่ 10 แล้ว *ptr มีค่าเท่ากับเท่าไร

c) หลังจากคำสั่งในบรรทัดที่ 11 แล้ว *(*ptr_ptr) มีค่าเท่ากับเท่าไร

d) โปรแกรมนี้ จะแสดงผลอะไรออกทางหน้าจอ

11. จงหาค่าของ x, y, z, *a_pt, *b_pt และ *c_pt ที่ปรากฏในโปรแกรมต่อไปนี้

```
int   x = 23, y, z, *a_pt, *b_pt, *c_pt ;
a_pt = &x ; b_pt = &y ; c_pt = &z ;
*c_pt = *a_pt + 17 ; *b_pt = *a_pt + *c_pt ;
```

12. เขียนโปรแกรมสำหรับหาค่ากึ่งกลางของชุดข้อมูล ยกตัวอย่าง เช่น ถ้าชุดข้อมูลเป็น 5 7 2 3 4 9 8 1 6 ค่ากึ่งกลางจะเท่ากับ 5 เป็นต้น โดยแสดงข้อมูลด้วย Pointer

13. เขียนโปรแกรมสำหรับแปลงข้อความที่เป็นตัวพิมพ์ใหญ่ให้เป็นตัวพิมพ์เล็ก โดยใช้ Pointer

14. เขียนโปรแกรมสำหรับเรียงลำดับข้อมูลโดยใช้วิธี Selection Sort

15. เขียนโปรแกรมสำหรับแปลงเลขฐานสิบหกให้เป็นเลขฐานสิบ โดยใช้ Pointer ในการแสดงเลขฐานสิบหก กล่าวคือ ให้แสดงเลขฐานสิบหกในรูปสตริง