บทที่ 10
แถวลำดับของตัวชี้
(Arrays of Pointers)

ในบทก่อน เราได้ศึกษาเกี่ยวกับแถวลำดับ ทั้งแถวลำดับมิติเดียว สองมิติ หรือหลายมิติของจำนวนเต็ม จำนวนจริงหรืออักขระ ซึ่งเป็นข้อมูลประเภทที่เราค่อนข้างคุ้นเคย แต่ในบทนี้จะกล่าวถึงแถวลำดับของข้อมูลประเภทอื่น ๆ เช่น ตัวชี้ (pointers) และตัวชี้ที่ชี้ไปยังฟังก์ชัน (pointers to functions) เป็นต้น

10.1 แถวลำดับของตัวชี้

ภาษา C ได้อำนวยความสะดวกให้ผู้ใช้สามารถนำตัวชี้ไปจัดเก็บไว้ในแถวลำดับได้เช่นเดียวกันกับข้อมูลประเภทอื่น รูปแบบของการประกาศตัวแปรประเภทแถวลำดับของตัวชี้ คือ

```
type  *var_name [size];
```

เช่น ถ้าต้องการประกาศตัวแปรแถวลำดับชื่อ '*ptr' เพื่อเก็บตัวชี้ไปยังจำนวนเต็ม 10 จำนวน สามารถกำหนดได้ ดังนี้

```
int  *ptr [10]
```

ในที่นี้ ptr เป็นชื่อของแถวลำดับที่จัดเก็บตัวชี้

ptr[0], ptr[1], … , ptr[9] เป็นตัวชี้ที่ชี้ไปยังตำแหน่งต่าง ๆ ซึ่งเป็นเลขที่อยู่ของจำนวนเต็ม ที่บรรจุอยู่ที่เลขที่ตำแหน่งนั้น

สมมติ x เป็นตัวแปรชนิดจำนวนเต็มซึ่งถูกชี้โดยสมาชิกตัวที่สองในแถวลำดับ 'ptr' เราจะเขียนได้ว่า

```
ptr[2]  =  &x;        /* &x  คือ เลขที่อยู่ของตัวแปร x*/
```

หรือ

```
*ptr[2] = x;
```

10.1.1 การส่งผ่านแถวลำดับของตัวชี้ไปยังฟังก์ชัน

การส่งผ่านแถวลำดับของตัวชี้ไปยังฟังก์ชันนั้น สามารถดำเนินการได้เช่นเดียวกันกับการส่งผ่านแถวลำดับของข้อมูลประเภทอื่น ๆ นั่นคือ ส่งชื่อของแถวลำดับโดยไม่ระบุขนาดหรือดัชนีให้เป็นพารามิเตอร์ของฟังก์ชัน ดังตัวอย่างที่ 10.1

ตัวอย่างที่ 10.1 ฟังก์ชัน display_array เป็นตัวอย่างฟังก์ชันรับค่าของแถวลำดับ 'ptr' ของตัวชี้ แล้วพิมพ์ค่าออกทางอุปกรณ์ส่งออกมาตรฐาน

```
⋮
int             *ptr[10];
void            display_array (ptr);
⋮
void display_arrary (int *recarray[ ])
{
    int i ;
    for (i = 0; i <10, i + +)
            printf ("%d", *recarray[i]);
}
```

ข้อควรระวัง ตัวแปร recarray ไม่ใช่ตัวชี้ไปยังจำนวนเต็ม แต่เป็นตัวชี้ไปยังแถวลำดับของตัวชี้

ดังนั้น ในนิยามของฟังก์ชัน 'display_array' จึงต้องกำหนดด้วยรูปแบบของอาร์กิวเมนต์ด้วย int *recarray[ ] ไม่ใช่ int recarray

10.1.2 แถวลำดับของตัวชี้ที่ชี้ไปยังสายอักขระ

นอกจากจะกำหนดตัวชี้ไปยังจำนวนเต็มแล้ว เรายังสามารถกำหนดตัวชี้ให้ชี้ไปยังสายอักขระได้เช่นกัน ดังตัวอย่างที่ 10.2 ต่อไปนี้

ตัวอย่างที่ 10.2 แสดงโปรแกรมการกำหนดตัวชี้ที่ชี้ไปยังสายอักขระและการพิมพ์สายอักขระนั้น โดยใช้ตัวชี้ในการเข้าถึงตัวอักขระแต่ละตัวแทนการใช้ดัชนีของแถวลำดับ

```
#include <stdio.h>
main ( )
{
    char  *text_ptr  = "Hello! Every body";
    for ( ;  *text_ptr   != '\0'; ++text_ptr)
         printf ("%c", *text_ptr);
     return 0;
}
```

ผลลัพธ์ที่ได้

```
Hello! Every body
```

ในตัวอย่างที่ 10.2 นี้ ตัวแปร text_ptr เป็นตัวชี้ชี้ไปยังสายอักขระ 'Hello! Everybody' และ *text_ptr เป็นค่าของข้อมูล(ตัวอักขระ) ที่ถูกชี้โดย text_ptr และ'\0' หมายถึง อักขระว่าง(null character) เมื่อเพิ่มค่าของ text_ptr (++text_ptr) ขึ้นทีละขั้น จะมีผลทำให้ตัวชี้ text_ptr เลื่อนไปชี้ยังอักขระตัวถัดไป จนกว่าจะพบ '\0' จึงหยุด

รูปแบบ

```
char    * var_name    = string_exp
```

ภาษา C จะเก็บตัวอักขระว่างต่อท้ายสายอักขระ 'string_exp' เสมอ

ถ้ามีสายอักขระหลาย ๆ สาย แต่ละสายมีความยาวไม่เท่ากัน เช่น ข้อมูลของวันในสัปดาห์ หรือ ข้อมูลชื่อของนักเรียนในแต่ละชั้น เป็นต้น ถ้าจะจัดเก็บข้อมูลนี้ในแถวลำดับสองมิติก็สามารถทำได้ โดยกำหนดให้แต่ละแถวของแถวลำดับเก็บชื่อวันในสัปดาห์ดังต่อไปนี้

```
char day_table[][10] =
    {
    {'m', 'o', 'n', 'd', 'a', 'y', '\0'},
        {'t', 'u', 'e', 's', 'd', 'a', 'y', '\0'},
        {'w', 'e', 'd', 'n', 'e', 's', 'd', 'a', 'y', '\0'},
        {'t', 'h', 'u', 'r', 's', 'd', 'a', 'y', '\0'},
        { 'f', 'r', 'i', 'd', 'a', 'y', '\0'},
        {'s', 'a', 't', 'u', 'r', 'd', 'a', 'y', '\0'},
        {'s', 'u', 'n', 'd', 'a', 'y', '\0'}
};
```

ในที่นี้ได้ประกาศให้แต่ละแถวสามารถจัดเก็บสายอักขระที่ยาวที่สุด คือ 10 ตัวอักขระ การใช้แถวลำดับสองมิติจัดเก็บข้อมูลในลักษณะเช่นนี้ จะทำให้บางแถวมีที่ว่างเหลืออยู่ดังในรูปที่ 10.1

m	o	n	d	a	y	\0			
t	u	e	s	d	a	y	\0		
w	e	d	n	e	s	d	a	y	\0
t	h	u	r	s	d	a	y	\0	
f	r	I	d	a	y	\0			
s	a	t	u	r	d	a	y	\0	
s	u	n	d	a	y	\0			

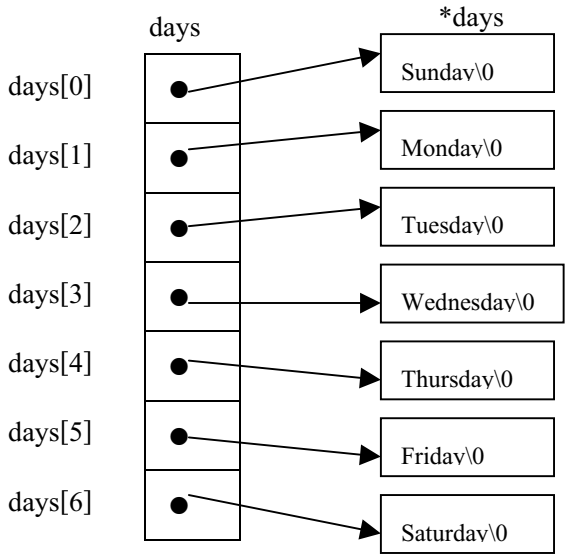
รูปที่ 10.1 แถวลำดับสองมิติของสายอักขระชื่อวันในสัปดาห์

เพื่อให้สามารถใช้เนื้อที่ในการจัดเก็บข้อมูลสายอักขระที่มีความยาวไม่เท่ากันให้เกิดประโยชน์สูงสุด เราสามารถทำได้โดยใช้แถวลำดับที่มีความยาวของแต่ละแถวไม่เท่ากัน ซึ่งเราเรียกแถวลำดับลักษณะเช่นนี้ว่า แถวลำดับขาด(ragged array) ซึ่งการสร้างแถวลำดับขาด สามารถทำได้โดยกำหนดให้เป็นแถวลำดับมิติเดียวของตัวชี้ ไปยังสายอักขระดังการประกาศตัวแปร 'days' ต่อไปนี้

```
char  *days[ ] = {"Sunday", "Monday", "Tuesday", "Wednesday",
                  "Thursday", "Friday", "Saturday"};
```

โปรดสังเกตว่าเราไม่ได้ระบุจำนวนสมาชิกของแถวลำดับ 'days' เพราะคอมไพเลอร์ภาษา C สามารถที่จะจัดสรรเนื้อที่ให้เพียงพอที่จะจัดเก็บข้อมูลที่มีอยู่ในคำสั่งประกาศตัวแปรแถวลำดับได้

แผนภาพแสดงการจัดเก็บข้อมูล ปรากฏในรูปที่ 10.2 ดังนี้



รูปที่ 10.2 แสดงโครงสร้างแถวลำดับ days

10.1.3 การเข้าถึงสมาชิกในแถวลำดับของตัวชี้

การเข้าถึงสมาชิกในแถวลำดับของตัวชี้นั้นสามารถทำได้เช่นเดียวกันกับการเข้าถึงสมาชิกในแถวลำดับของข้อมูลประเภทอื่น ๆ

ตัวอย่างที่ 10.3 โปรแกรม 'prdays.c' ต่อไปนี้ เป็นโปรแกรมพิมพ์ชื่อวันซึ่งในหนึ่งบรรทัดพิมพ์หนึ่งชื่อ โดยเรียกใช้ฟังก์ชัน 'print_strings' ด้วยคำสั่ง

```
print_strings(days, 7);
```

ซึ่งอาจจะเขียนได้อีกแบบหนึ่ง คือ

```
print_strings (days, sizeof(days)/size of(char*));
```

ในกรณีนี้คอมไพเลอร์จะทำการคำนวณจำนวนสมาชิกของตัวแปรแถวลำดับ days ให้ซึ่งจะได้ผลลัพธ์เป็น 7 เช่นกัน

```
/* Program : prdays.c
 *                  Main program that prints a table of days
 */
#include <stdio.h>
char *days[ ] =
{
                "monday","tuesday",
                "wednesday","thursday","friday","saturday","sunday"
};
main()
{
                void      print_strings(char *table[ ], int n);
                print_strings(days, sizeof(days)/sizeof(char *));
                return 0;
}
```

ฟังก์ชัน print_strings ที่ใช้ในการพิมพ์ชื่อวันนั้น สามารถเขียนได้หลายวิธี ดังปรากฏในโปรแกรมต่าง ๆ ดังนี้

1. prstr1.c
2. prstr2.c
3. prstr3.c

ตัวอย่างที่ 10.4 แสดงการทำงานของฟังก์ชัน prt_strings โดยใช้รูปแบบ '%s' ในการพิมพ์สาย อักขระ

```
 /* Program : prstr1.c
  *                  Print a table of strings, one per line
 */
#include <stdio.h>
void print_strings(char *table[], int n)

int                 i;
for ( i = 0; i < n; i++)
                    printf("\n%s", table[i]);
```

ผลลัพธ์ที่ได้ คือ

```
monday
tuesday
wednesday
thursday
friday
saturday
sunday
```

ฟังก์ชัน print_strings ในโปรแกรม 'prstr1.c' นี้ ใช้รูปแบบการพิมพ์ในคำสั่ง printf ด้วยรูปแบบ '%s' ซึ่งจะพิมพ์สายอักขระทั้งสายทีละตัวไปเรื่อย ๆ จนพบ '\0' จึงหยุด

ตัวอย่างที่ 10.5 แสดงการทำงานของฟังก์ชัน prt_strings โดยใช้ฟังก์ชัน putchar ในการพิมพ์สายอักขระ

```
/* Program : prstr2.c
 *                    Print a table of character strings, one per line
 */
#include <stdio.h>
void                print_strings(char *table[], int n)
{
                    int   i, j;
                    for (i=0; i < n; i++)
                    {
                          for (j = 0; table[i][j] != '\0'; j++)
                                putchar(table[i][j]);
                          putchar('\n');
                    }
}
```

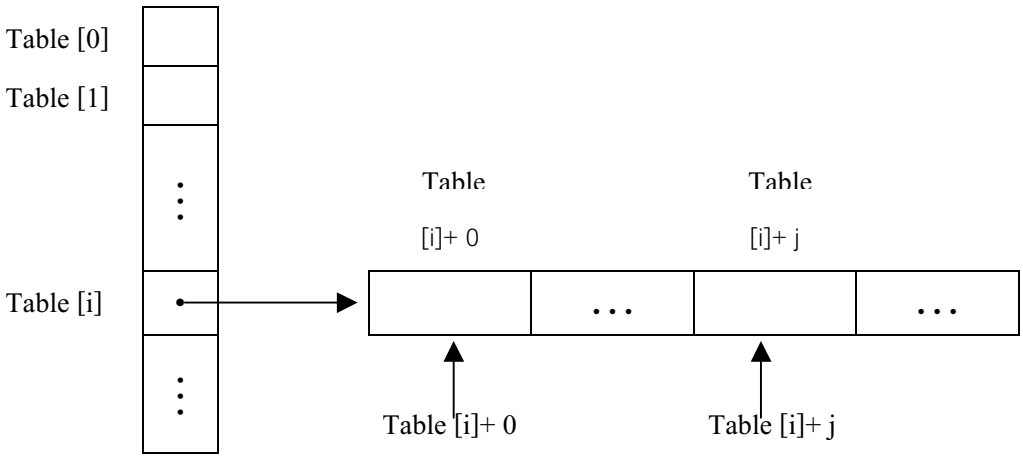
ผลลัพธ์ที่ได้เหมือนผลลัพธ์ในตัวอย่างที่ 10.4

ฟังก์ชัน print_string ในโปรแกรม prstr2.c นี้ ใช้ putchar เพื่อพิมพ์ตัวอักขระแต่ละตัว ไปเรื่อยๆ จนพบ '\0' จึงหยุดขึ้นบรรทัดใหม่ เพื่อพิมพ์ชื่อวันใหม่ต่อไป

การอ้างถึงอักขระแต่ละตัวในสายอักขระนั้น อาจใช้ตัวชี้ของแถวลำดับสองมิติ table[i][j] ซึ่งบอกว่าเป็นอักขระตัวที่ j ในแถวที่ i สาเหตุที่เราใช้ดัชนีของแถวลำดับสองมิติ เข้าถึงอักขระตัวที่ j ที่ถูกชี้โดยตัวชี้ table[i] ได้ก็เพราะว่า table[i] คือตัวชี้ไปยังอักขระตัวแรกของแถวที่ i และ table [i] + j จะหมายถึงตัวชี้ไปยังอักขระตัวที่ j ของแถวที่ i

นั่นคือ

*(table[i] + j) จึงเป็นอักขระตัวที่ j ของแถวที่ i ซึ่งสามารถอ้างถึงได้ด้วย table[i][j] ดังแสดงในรูปที่ 10.3 ต่อไปนี้



รูปที่ 10.3 แสดงการเข้าถึงอักขระตัวที่ j ของแถวที่ i

ตัวอย่างที่ 10.6 แสดงการทำงานของฟังก์ชัน prt_strings โดยใช้ตัวชี้

```
/* Program : prstr3.c
 *                    Print a table of character strings, one per line,
 *                    this time using pointers
 */


#include <stdio.h>

void                print_strings(char *table[], int n)
{
                    int   i;
                    char          *ptr;

                    for (i=0; i < n; i++)
                    {
                          for (ptr = table[i] ; *ptr !='\0'; ptr++)
                                putchar(*ptr);
                          putchar('\n');
                    }
}
```

ผลลัพธ์ที่ได้เหมือนผลลัพธ์ในตัวอย่างที่ 10.4

โปรแกรม 'prstr3.c' นี้ ใช้ตัวชี้ 'ptr' ซึ่งเป็นตัวชี้ไปยังตัวอักขระ เมื่อเริ่มต้น กำหนดให้

```
ptr      =   table[i]
```

ptr เป็นตัวชี้ที่ชี้ไปยังตัวอักขระตัวแรกของแต่ละแถว หลังจากนั้นจะท่องผ่านเข้าไปที่อักขระแต่ละตัวในสายอักขระนั้น ๆ โดยเพิ่มค่าของ 'ptr' ทีละหนึ่งด้วยคำสั่ง 'ptr++' แล้วท่องต่อไปเรื่อย ๆ จนพบ '\0' ซึ่งแสดงว่าจบสายอักขระ แล้วจึงกำหนดให้ 'ptr' เป็นตัวชี้ไปยังตัวอักขระแรกของแถวถัดไป

10. 2 ตัวชี้กับแถวลำดับของตัวชี้

การท่องเข้าไปในแถวลำดับของตัวชี้ไม่เพียงแต่จะท่องเข้าไปโดยใช้ดัชนีเท่านั้น เราสามารถท่องเข้าไปโดยใช้ตัวชี้ได้เช่นกัน

ตัวอย่างที่ 10.7 ต่อไปนี้เป็นการกำหนดฟังก์ชัน 'print_strings' โดยค่าพารามิเตอร์ที่รับมาจากตัวเรียก คือ ตัวชี้ที่ชี้ไปยังสมาชิกตัวแรกของแถวลำดับ

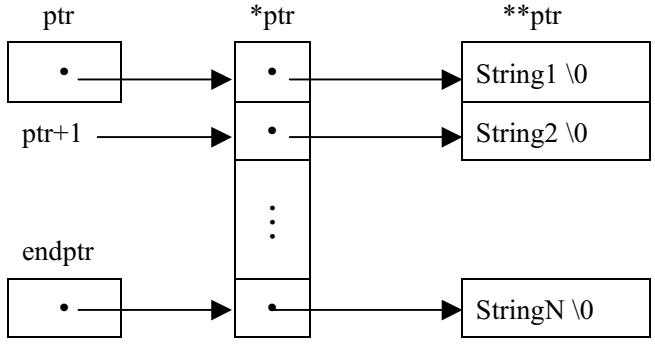
ตัวอย่างที่ 10.7 แสดงการทำงานของฟังก์ชัน prt_strings โดยใช้ตัวชี้ชี้ไปยังตัวชี้

```
/* Program : prstr4.c
 *                    Print a table of character strings, one per line
 */
#include <stdio.h>
void                print_strings(char **ptr, int n)
{
                    char          **endptr = ptr + n - 1;
                    for ( ; ptr <= endptr; ptr++)
                          printf("%s\n", *ptr);
}
```

ผลลัพธ์ที่ได้เหมือนผลลัพธ์ในตัวอย่างที่ 10.4

ในที่นี้ตัวอย่างเช่น char **ptr ptr หมายถึงตัวชี้ที่ชี้ไปยังตัวชี้ที่ชี้ไปยังตัวอักขระ

โปรแกรม prstr4.c นี้กำหนดให้ตัวแปร 'ptr' เป็นตัวชี้ที่ชี้ไปยังสมาชิกแรกของแถวลำดับของตัวชี้ แล้วเพิ่มค่าของ 'ptr' ขึ้นทีละหนึ่ง ด้วยคำสั่ง 'ptr++' จนกระทั่งถึงสมาชิกตัวสุดท้ายในแถวลำดับดังปรากฏในรูปที่ 10.4



รูปที่ 10.4 แสดงการใช้ตัวชี้ท่องเข้าไปในแถวลำดับของตัวชี้

แต่ละรอบในคำสั่งวงวนนั้น คำสั่ง 'printf' จะได้รับค่าตัวชี้ไปยังสมาชิกตัวแรกของสายอักขระซึ่งในการพิมพ์สายอักขระที่ลงท้ายด้วย '\0' โดยใช้รูปแบบการพิมพ์ '%s' นั้น สายอักขระทั้งสายจะถูกพิมพ์ทั้งหมด ดังคำสั่ง printf ("%s\n", *ptr); หลังจากพิมพ์สายอักขระเสร็จแล้วจะขึ้นบรรทัดใหม่เพิ่มค่าตัวชี้ 'ptr' เพื่อพิมพ์บรรทัดต่อไป

ตัวอย่างที่ 10.8 แสดงการท่องเข้าไปในแถวลำดับของตัวชี้ โดยใช้ตัวชี้ดังในตัวอย่างที่ 10.7

```
/* Program : prstr5.c
 *          Print a table of character strings, one per line
 */
#include <stdio.h>
void    print_strings(char **ptr, int n)
{
        int     j;
        char    **endptr = ptr + n - 1;
        for ( ; ptr <= endptr; ptr++)
        {
                for (j = 0; (*ptr)[j] !='\0'; j++)
                        putchar((*ptr)[j]);
                putchar('\n');
        }
}
```

ผลลัพธ์ที่ได้เหมือนผลลัพธ์ในตัวอย่างที่ 10.4

โปรแกรม 'prstr5.c' ต่างจาก 'prstr4.c' ตรงที่โปรแกรมนี้ใช้คำสั่ง 'putchar( )' ในการพิมพ์อักขระทีละตัว และเข้าถึงสมาชิกตัวที่ j ในแต่ละแถวด้วยคำสั่ง

(*ptr)[j]

เมื่อ *ptr เป็นตัวชี้ที่ชี้ไปยังอักขระแรกของแต่ละแถว

10.3 อาร์กิวเมนต์แถวคำสั่ง

อาร์กิวเมนต์แถวคำสั่ง(command-line arguments) เป็นวิธีการสำหรับส่งผ่านอาร์กิวเมนต์ให้กับฟังก์ชันหลัก 'main' หรือให้กับโปรแกรมเมื่อเริ่มต้นกระทำการโปรแกรม

ตัวอย่างของอาร์กิวเมนต์แถวคำสั่งใน 'DOS' เมื่อต้องการใช้โปรแกรม 'tcc' คอมไพล์โปรแกรม inout.c สามารถกำหนดได้ดังนี้

```
C:\>tcc inout.c
```

จากอาร์กิวเมนต์แถวคำสั่งข้างต้น 'tcc' คือคำสั่งหรือโปรแกรมที่สามารถกระทำการได้ และ 'inout.c' เป็นอาร์กิวเมนต์หรือพารามิเตอร์ที่จะส่งไปให้โปรแกรม 'tcc'

การกำหนดฟังก์ชันหลัก

การกำหนดฟังก์ชันหลัก (main) ในโปรแกรมเพื่อให้รับอาร์กิวเมนต์จากแถวคำสั่งเมื่อเริ่มต้นกระทำการ มีรูปแบบดังนี้

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

หรืออีกรูปแบบหนึ่งคือ

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

ในที่นี้

- argc คือ จำนวนอาร์กิวเมนต์ในประโยคแถวคำสั่ง รวมทั้งชื่อโปรแกรม
- argv คือ แถวลำดับของตัวชี้ไปยังสายอักขระ

โปรแกรม 'echo1.c' ในตัวอย่างที่ 10.9 ต่อไปนี้ เป็นการสร้างคำสั่ง 'echo' เพื่อให้คอมพิวเตอร์พิมพ์สายอักขระที่อยู่หลังคำสั่งออกทางอุปกรณ์ส่งข้อมูลออก

ตัวอย่างที่ 10.9 โปรแกรมสร้างคำสั่ง 'echo'

```
/* Program : echo1.c
 *          Echo arguments (using array subscripting).
 */
#include <stdio.h>


int   main(int argc, char *argv[])
{
        int       next;                    /* index to next argument */
        for ( next = 1; next < argc; next++)
                printf("%s %c", argv[next], (next < argc - 1)? ' ': '\n');
        return 0;
}
```

ผลลัพธ์ที่ได้คือ

```
C:\TC\BIN>echo C language is very wonderful
C language is very wonderful
```

จะสังเกตเห็นว่า โปรแกรม 'echo1.c' นี้ใช้ดัชนีในการเข้าถึงสมาชิกแต่ละตัวในแถวลำดับของตัวชี้ และใช้รูปแบบ '%s' ในการพิมพ์สายอักขระทั้งชุดดังในคำสั่ง

```
printf ("%s%c", argv[next], (next < argc – 1) ? ' ' : '\n');
```

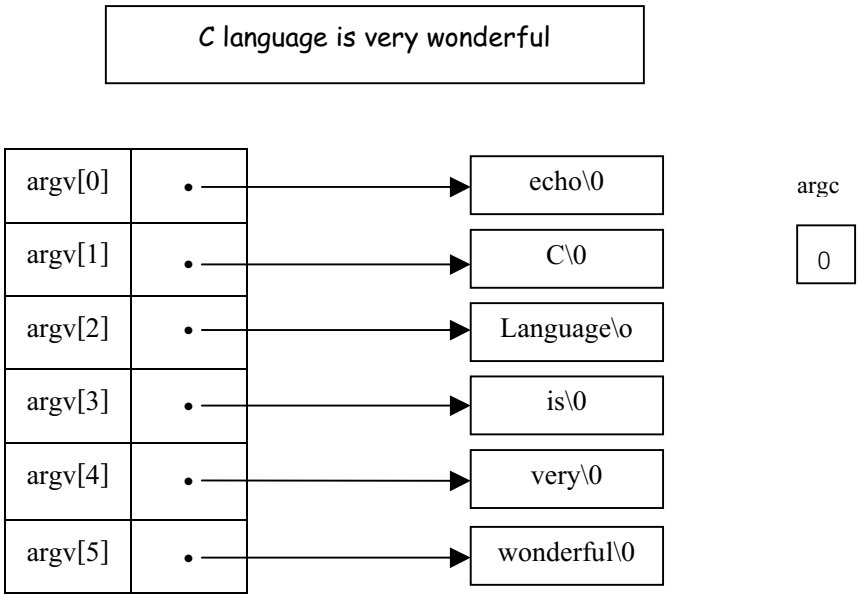
หลังจากสร้างโปรแกรมเรียบร้อยแล้ว ให้แปลโปรแกรมเพื่อให้ได้คำสั่งหรือแฟ้มคำสั่งชื่อ 'echo' รูปแบบการแปลโปรแกรมด้วย 'tcc'

```
C:\>tcc  echo1.c -o echo
```

แล้วทดลองกระทำการ ด้วยคำสั่ง echo

```
C:\> echo C language is very wonderful
```

ผลลัพธ์ที่ได้คือ



รูปที่ 10.5 แสดงค่าเริ่มต้นของ argc กับ argv ในคำสั่ง echo

จากรูปที่ 10.5 เมื่อฟังก์ชัน 'main' ถูกเรียกนั้น

- argc มีค่าเป็น 6
- argv เป็น แถวลำดับของตัวชี้ที่ชี้ไปยังสายอักขระต่างๆ

ตัวอย่างที่ 10.10 โปรแกรม 'echo2.c' แสดงการสร้างคำสั่ง 'echo' โดยใช้ตัวชี้ท่องเข้าไปในแถวลำดับของตัวชี้ argv แทนการใช้ดัชนี ดังในตัวอย่างที่ 10.9

```
/* Program : echo2.c
 *          Echo arguments (using pointer indexing).
 */
#include <stdio.h>
int main(int argc, char *argv[])
{
        char    **argptr        = argv + 1;             /* ptr to first argument */
        char    **endptr        = argv + argc - 1;      /* ptr to last argument */

        for ( ; argptr <= endptr; argptr++)
                printf("%s %c", *argptr, (argptr < endptr)? ' ': '\n');
        return 0;
}
```

ผลลัพธ์ที่ได้เหมือนผลลัพธ์ในตัวอย่างที่ 10.9

โปรแกรมนี้กำหนดค่าเริ่มต้นของตัวชี้ 'argptr' ให้ชี้ไปยังสายอักขระอันแรกที่อยู่ต่อจากคำสั่ง โดยกำหนดให้ค่าเท่ากับ argv + 1 ซึ่งเป็นการกำหนดให้ข้ามชื่อของโปรแกรมหรือคำสั่ง

ตัวอย่างที่ 10.11 ต่อไปนี้เป็นอีกวิธีหนึ่งในการท่องเข้าไปในแถวลำดับของตัวชี้ argv เพียงแต่ว่าการควบคุมการทำงานให้สามารถท่องเข้าไปในทุก ๆ สมาชิกนั้น ใช้ค่าของ argc (จำนวนอาร์กิวเมนต์) เป็นตัวควบคุม คือให้พิมพ์สายอักขระทุกสายยกเว้นสมาชิกที่มีดัชนีเป็น '0'

ตัวอย่างที่ 10.11 แสดงการใช้ค่าของ argc การควบคุมการทำงาน

```
/* Program : echo3.c
 *           Echo command line arguments; another version
 */
#include <stdio.h>


int    main(int argc, char *argv[])
{
         while (--argc > 0)
                  printf("%s %s", *++argv, (argc > 1)? " ": " ");
         printf("\n");
         return 0;
}
```

ผลลัพธ์ที่ได้เหมือนผลลัพธ์ในตัวอย่างที่ 10.9

10.4 ตัวชี้ไปยังฟังก์ชัน

ถึงแม้ว่าฟังก์ชันจะไม่ใช่ตัวแปร แต่เราสามารถเก็บชื่อของฟังก์ชันในส่วนของหน่วยงานความจำได้เช่นเดียวกันกับตัวแปร นั่นคือ เราสามารถกำหนดตัวชี้ไปยังเลขที่อยู่ของฟังก์ชัน และใช้ตัวชี้นี้เรียกใช้งานฟังก์ชันได้

โดยทั่ว ๆ ไป เราสามารถอ้างถึงเลขที่อยู่ของฟังก์ชันได้ โดยใช้ชื่อของฟังก์ชัน แล้วตามด้วยอาร์กิวเมนต์ที่อยู่ภายในวงเล็บหลังชื่อของฟังก์ชัน

ตัวอย่างที่ 10.12 โปรแกรมแสดงการใช้ตัวชี้ไปยังฟังก์ชัน

```
/* prtTofunc1.c
 *    The program with pointer to function
 */
#include        <stdio.h>
#include        <string.h>
int check(char *a, char *b, int  (*cmp)( ));
main(void )
{
        char    s1[80], s2[80];
        int     (*p)( );
        p       = strcmp;

        gets(s1);
        gets(s2);
        check(s1, s2, p);
        return 0;
}

int  check(char  *a, char  *b, int  (*cmp)())
{
        printf("testing for equality \n");
        if      (! (*cmp)(a, b))
                printf("equal\n");
        else
                printf ("not equal\n");
        return 0;
}
```

ผลลัพธ์ที่ได้จากการรันโดยใส่ค่า 234567 และ 234567

```
234567
234567
testing for equality
equal
```

ผลลัพธ์ที่ได้จากการรันโดยใส่ค่า Naree และ Navee

```
Naree
Navee
testing for equality
not equal
```

ในโปรแกรม ptrTofuncl.c นี้ ได้ประกาศให้ตัวแปร 'p' เป็นตัวชี้ที่ชี้ไปยังฟังก์ชัน ซึ่งรูปแบบของการประกาศ คือ

```
int    (*p)( );
```

หลังจากนั้นกำหนดให้ p = strcmp และกำหนดให้ p เป็นพารามิเตอร์ของฟังก์ชัน check ร่วมกับตัวแปร s1, s2 ซึ่งเป็นตัวชี้ไปยังสายอักขระ s1, s2 ดังนี้

```
check (s1, s2, p);
```

ในฟังก์ชัน 'check' จะมีคำสั่ง

```
(*cmp)(a, b)
```

ซึ่งเรียกใช้ฟังก์ชัน strcmp( ) โดยใช้ตัวชี้ cmp เป็นตัวชี้ที่ชี้ไปยังเลขที่อยู่ของฟังก์ชัน 'strcmp'

หมายเหตุ คือจะต้องใส่วงเล็บล้อมรอบข้อความ *cmp เพื่อแสดงว่า cmp เป็นตัวชี้ไปยังฟังก์ชันที่ต้องการเรียกใช้

สมมติว่าโปรแกรม ptrTofunc1.c ถูกคอมไพล์เป็นแฟ้ม ptrTofunc1 แล้วลองรันจะได้ผลลัพธ์ดังนี้

```
C:\>ptrTofunc1
hello world
hello world
testing for equality
equal
C:\>
```

ตัวอย่างที่ 10.13 โปรแกรม ptrTofunc2.c แสดงการใช้ตัวชี้ไปยังฟังก์ชัน เช่นเดียวกันกับโปรแกรม ptrTofunc1.c แต่เรียกใช้ฟังก์ชันต่างกัน

```
/* prtTofunc2.c
 *    Another program with pointer to function
 */
#include        <stdio.h>
#include        <ctype.h>
#include        <stdlib.h>
#include        <string.h>
int    check(char *a, char *b, int  (*cmp)( ));
int    numcmp(char *a, char *b);
main(void )
{
        char    s1[80], s2[80];
        gets(s1);
        gets(s2);

        if (isalpha(*s1))
                check(s1, s2, strcmp);
        else
                check(s1, s2, numcmp);
        return 0;
}

int  check(char  *a, char  *b, int  (*cmp)())
{
        printf("testing for equality \n");
        if      ( !(*cmp)(a, b))
                printf("equal\n");
        else
                printf ("not equal\n");
        return 0;
}

int numcmp(char *a, char *b)
{
        if (atoi(a) == atoi(b)) return 0;
        else
                return 1;
}
```

ผลลัพธ์ที่ได้ เหมือนกับผลลัพธ์ในตัวอย่างที่ 10.12

แบบฝึกหัด

1. กำหนดให้ numbers เป็นตัวแปรแถวลำดับของจำนวนจริง 'float' จงตอบคำถามต่อไปนี้
   (1) ประกาศตัวแปรแถวลำดับ numbers ที่มีสมาชิก 10 ตัว คือ
   0.1, 1.1, 2.1, …, 9.1
   โดยให้ตัวระบุ SIZE ถูกกำหนดให้มีค่าเป็น 10
   (2) ประกาศตัวแปร nptr เป็นตัวแปรชนิดตัวชี้ที่ชี้ไปยังจำนวนจริง 'float'
   (3) เขียนคำสั่ง 2 คำสั่ง ที่ทำงานอย่างเดียวกัน คือ กำหนดให้ตัวชี้ nptr ชี้ไปยังเลขที่อยู่เริ่มต้นของแถวลำดับ numbers
   (4) เขียนคำสั่งหรือชุดคำสั่ง เพื่อพิมพ์สมาชิกทุกตัวของแถวลำดับ numbers โดยใช้ตัวชี้และดัชนีของแถวลำดับ

2. จงเขียนฟังก์ชัน month_name เพื่อพิมพ์ชื่อเดือนใน 1 ปีโดยใช้ดัชนีและตัวชี้

3. จงเขียนฟังก์ชัน reverse_table เพื่อกลับสายอักขระทุกสายในตาราง table ให้เรียงลำดับตรงกันข้ามกับสายอักขระเดิม (เรียงลำดับตัวอักขระจากขวาไปซ้าย) เมื่อ table เป็นแถวลำดับขาด (Ragged array) ของสายอักขระ

4. จงเขียนโปรแกรมชื่อ compute เพื่อคำนวณนิพจน์ตามแบบหลังลำดับ (postfix notation) จากบรรทัดแถวลำดับคำสั่ง โดยให้แต่ละตัวดำเนินการและตัวถูกดำเนินการเป็นอาร์กิวเมนต์ที่แยกกัน เช่น

   compute 2 3 + จะคำนวณ 2+3
   compute 4 5 * จะคำนวณ 4*5

   เป็นต้น

5. จงเขียนฟังก์ชันเพื่อท่องเข้าไปในแถวลำดับของตัวชี้ไปยังฟังก์ชัน เมื่อแวะผ่านเข้าไปที่สมาชิกใดในแถวลำดับให้เรียกฟังก์ชันที่อยู่ ณ ตำแหน่งนั้น ๆ กระทำการ จนกระทั่งพบฟังก์ชันที่คืนค่าเป็น 0 ให้หยุด หรือ จนกระทั่งแวะผ่านจนครบทุกฟังก์ชัน