บทที่ 5 แถวลำดับ ตัวชี้ สายอักขระ

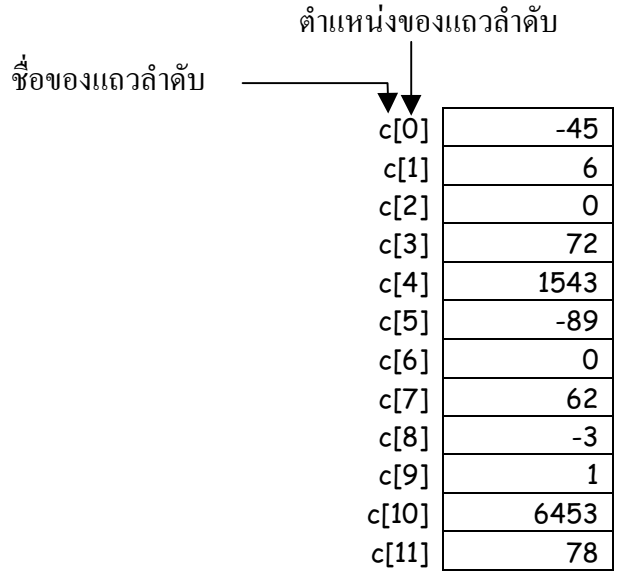
(Arrays Pointers and Strings)

ในบทนี้จะกล่าวถึงโครงสร้างข้อมูลของตัวแปรที่สามารถเก็บข้อมูลได้มากกว่า 1 จำนวน ซึ่งโครงสร้างนี้จะมีความซับซ้อนกว่าตัวแปรที่ได้กล่าวไปแล้วในตอนต้น เช่น int long double และ char โครงสร้างข้อมูลที่จะกล่าวถึงในบทนี้ได้แก่ แถวลำดับ (array) ตัวชี้ (pointer) และสายอักขระ (string)

5.1 แถวลำดับ

แถวลำดับเป็นโครงสร้างข้อมูลที่ประกอบด้วยข้อมูลที่มีชนิดเดียวกัน โดยมีการเก็บข้อมูลในหน่วยความจำเป็นแบบสถิต ซึ่งหมายถึงการใช้เนื้อที่ในหน่วยความจำของแถวลำดับมีขนาดคงที่ตลอดการดำเนินงานของโปรแกรม การอ้างถึงตำแหน่ง หรือสมาชิกของแถวลำดับ ผู้ใช้สามารถกำหนดชื่อของแถวลำดับ และตำแหน่งของสมาชิกในแถวลำดับ

รูปที่ 5.1 แสดงแถวลำดับ c ซึ่งเป็นแถวลำดับของจำนวนเต็ม แถวลำดับนี้มีสมาชิก 12 ตัว การอ้างถึงตำแหน่งหรือสมาชิกในแถวลำดับ ทำได้โดยเรียกชื่อแถวลำดับตามด้วยตำแหน่งของสมาชิก ซึ่งเขียนอยู่ภายในเครื่องหมายวงเล็บก้ามปู ([]) สำหรับสมาชิกตัวแรกของแถวลำดับจะอยู่ที่ตำแหน่งศูนย์ เขียนแทนด้วย c[0] และสมาชิกตัวที่สองของแถวลำดับ c คือ c[1] สมาชิกตัวที่เจ็ดของแถวลำดับ c คือ c[6] โดยทั่วไปสมาชิกตัวที่ i ของแถวลำดับ c นั้นอ้างถึงโดยตำแหน่ง c[i-1]



c[0]	-45
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	78

รูปที่ 5.1 แถวลำดับที่มีสมาชิก 12 ตัว

5.1.1 การประกาศแถวลำดับ

แถวลำดับเป็นโครงสร้างข้อมูลที่ต้องการเนื้อที่ในหน่วยความจำ นักเขียนโปรแกรมจะกำหนดชนิดและขนาดของแถวลำดับไว้ที่ส่วนต้นของโปรแกรมหรือฟังก์ชัน เพื่อจองเนื้อที่หน่วยความจำ เช่น

```
int     c[12];
```

เป็นคำสั่งเพื่อจองที่ให้แถวลำดับ c ไว้เก็บสมาชิกที่เป็นจำนวนเต็ม 12 ตัว

```
int     b[100], x[27];
```

เป็นคำสั่งเพื่อจองที่ให้แถวลำดับ b สำหรับบรรจุสมาชิกที่เป็นจำนวนเต็ม 100 ตัว และแถวลำดับ x สำหรับสมาชิกที่เป็นจำนวนเต็ม 27 ตัว

การประกาศชนิดของตัวแปรและกำหนดค่าเริ่มต้นของสมาชิกในแถวลำดับ สามารถดำเนินการได้พร้อมกันดังนี้

```
int     n[6] = {32, 27, 64, 18, 95, 4};
```

คำสั่งนี้กำหนดให้สมาชิกของแถวลำดับ n ตั้งแต่ตัวที่ 0 ถึง 5 มีค่าเป็น n[0]=32,n[1]= 27,n[2]= 64,n[3]=18, n[4]=95 และ n[5]=4 ตามลำดับ

5.1.2 การกำหนดตัวคงที่ภายในโปรแกรม

คำสั่ง #define เป็นคำสั่งที่เขียนที่ส่วนต้นของโปรแกรมหรือฟังก์ชัน เพื่อกำหนดค่าให้กับตัวคงที่ โดยตัวคงที่นี้จะไม่เปลี่ยนค่าตลอดการดำเนินงานของโปรแกรม

เช่น #define SIZE 10

เป็นการกำหนดให้ตัวคงที่ SIZE มีค่าเป็น 10

ตัวอย่างที่ 5.1 โปรแกรมคำนวณผลรวมของมูลค่าที่อยู่ในแถวลำดับ a ที่มีสมาชิก 9 ตัวโดยใช้คำสั่งวนซ้ำ for

```
/* Compute the sum of the elements of the array */
#include <stdio.h>
#define SIZE 9
main()
{
        int a[SIZE] = {1, 3, 5, 4, 7,  2, 99, 16, 45};
        int i, total = 0;
        for(i = 0; i<= SIZE –1; i++)
                total += a[i];
        printf("Total of array element value is %d\n", total);
        return 0;
}
```

ผลลัพธ์ที่ได้

```
Total of array element value is 182
```

5.2 สายอักขระ

สายอักขร(string) คือ ชุดของตัวอักขระ ซึ่งอาจประกอบด้วยตัวอักษร ตัวเลข หรือตัวอักขระพิเศษอื่น ๆ ได้ แก่ +, -, *, /, $ สำหรับตัวคงที่สายอักขระ (String literal; String constant) ในภาษา C เขียนอยู่ภายใต้เครื่องหมายคำพูด เช่น

“Bang Bau”	(ชื่อ)
“111 Phaholyotin Road “	(บ้านเลขที่)
“Bangkok”	(จังหวัด)
“(662) 5791023”	(เบอร์โทรศัพท์)

สายอักขระในภาษา C คือ ตัวอักขระตัวสุดท้ายของแถวลำดับคือ ‘\0’ เรียกว่า อักขระว่าง(null) การเข้าถึงสายอักขระทำได้โดยใช้ตัวชี้ชี้ไปยังตัวอักขระตัวแรกของสายอักขระ

การประกาศสายอักขระสามารถทำได้ 2 วิธี คือ

1. กำหนดให้เป็นแถวลำดับของตัวอักขระ เช่น

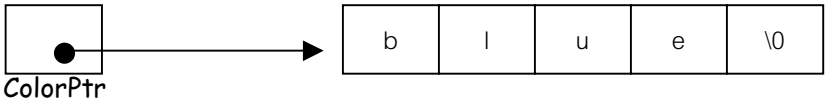
```
char    color[] = “blue”;
```

คำสั่งนี้สร้างแถวลำดับ color ที่มีสมาชิกเป็นตัวอักขระ 5 ตัว คือ ‘b’, ‘l’, ‘u’, ‘e’ และ ‘\0’

2. กำหนดให้เป็นตัวชี้ชี้ไปยังสายอักขระ เช่น

```
char    *colorPtr = “blue”;
```

คำสั่งนี้สร้างตัวชี้ colorPtr ซึ่งชี้ไปที่สายอักขระ ‘blue’ ที่เก็บไว้ในหน่วยความจำ ดังรูปที่ 5.2



รูปที่ 5.2 แสดงตัวชี้ ColorPtr ที่ชี้ไปยังสายอักขระ blue

5.2.1 การสร้างสายอักขระจากข้อมูลนำเข้า

ในการเขียนโปรแกรม นักเขียนโปรแกรมมักจะสร้างสายอักขระจากข้อมูลนำเข้า ในตัวอย่างนี้ ฟังก์ชัน getline เป็นฟังก์ชันสำหรับอ่านข้อมูลนำเข้าทีละ 1 บรรทัด

ตัวอย่างที่ 5.2 แสดงฟังก์ชัน getline ที่ใช้สายอักขระเป็นที่เก็บข้อมูลนำเข้า

```
/*
* Line number its input, one line at a time.
*/
#include <stdio.h>
#define MAXLEN 80                                   /* longest line */
main()
{
        char line[MAXLEN +1];                       /* input line (plus NULL) */
        unsigned long lines;                        /* line count */

        int getline(char *buf, int bufsize);
        lines = 0L;
        while (getline(line, MAXLEN) != -1)
                printf("%6lu %s\n", ++lines, line);
                return 0;
} /* End main */
int getline(char line[], int max)
{
        int c,                                      /* current character */
            i;                                      /* character count */
        i = 0;
        while ((c = getchar()) != '\n' && c != EOF)
                if (i < max)
                        line[i++] = c;
                line[i] = '\0';                     /* terminate with null */
                return (c == EOF) ? -1 : i;
} /* End int geetline */
```

ผลลัพธ์ที่ได้

```
352.65
     1 352.65
489.1
     2 489.1
56.03
     3 56.03
10
     4 10
1056
     5 1056
```

ฟังก์ชัน getline อ่านตัวอักขระทีละตัวและนำไปเก็บไว้ในสายอักขระ line จนกระทั่งอ่านพบตัวอักขระ '\n' ซึ่งเป็นตัวบอกจุดจบบรรทัด หรือพบว่าสายอักขระเต็มจึงหยุดอ่านข้อมูล

ฟังก์ชัน getline นี้ใส่อักขระว่าง(null) ที่ตำแหน่งสุดท้ายของสายอักขระลงไปในสายอักขระ line เพื่อบอกการจบสายอักขระ และฟังก์ชัน getline จะคำนวณหาความยาวของสายอักขระ และคืนค่าเป็นความยาวของสายอักขระ หรือคืนค่าเป็น –1 เมื่ออ่านพบจุดสุดท้ายของแฟ้มข้อมูล

5.2.2 การเรียกใช้ฟังก์ชันสายอักขระจากคลังมาตรฐาน

ในคลังฟังก์ชันมาตรฐานของภาษา C จะมีฟังก์ชันที่สามารถดำเนินการกับสายอักขระหลายตัว โดยการเรียกใช้ฟังก์ชันเหล่านี้ต้องใช้คำสั่ง #include<string.h> ไว้ที่ส่วนต้นของโปรแกรม รายชื่อของฟังก์ชันสายอักขระและหน้าที่ของการทำงาน แสดงในตารางที่ 5.1 ดังนี้

ชื่อ	การทำงาน
strcat(s1, s2)	เชื่อมสายอักขระโดยนำสายอักขระ s2 ไปต่อท้ายสายอักขระ s1
strncat(s1, s2, n)	เชื่อมสายอักขระโดยนำสายอักขระ s2 ในส่วนที่เป็น n ตัวแรกไปต่อท้ายสายอักขระ s1
strcpy(s1, s2)	คัดลอกสายอักขระ s2 ไปยังสายอักขระ s1
strncpy(s1, s2, n)	คัดลอกสายอักขระ s2 ในส่วน n ตัวแรกไปยังสายอักขระ s1
strcmp(s1, s2)	เปรียบเทียบสายอักขระ s1 และ s2 ว่าเหมือนกันหรือไม่ โดยเปรียบเทียบตัวอักขระทั้งสองทีละคู่ตามลำดับ และคืนค่าเป็นลบเมื่อ s1<s2 คืนค่าเป็นศูนย์เมื่อ s1=s2 คืนค่าเป็นบวกเมื่อ s1>s2
strncmp(s1, s2, n)	เปรียบเทียบตัวสายอักขระ n ตัวแรกของสายอักขระ s1 และ s2 ว่าเหมือนกันหรือไม่
strlen(s)	คืนค่าของจำนวนตัวอักขระในสายอักขระ s
strchr(s, c)	คืนค่าตัวชี้ที่ที่ชี้ไปยังอักขระ c ตัวแรกที่ปรากฏในสายอักขระ s
strrchr(s, c)	คืนค่าตัวชี้ที่ที่ชี้ไปยังอักขระ c ตัวสุดท้ายที่ปรากฏในสายอักขระ s

ตารางที่ 5. 1 ฟังก์ชันสายอักขระมาตรฐานในภาษา C

ในภาษา C นักเขียนโปรแกรมไม่สามารถใช้คำสั่งกำหนดค่าของสายอักขระ 1 สาย ให้กับตัวแปรประเภทแถวลำดับของตัวอักขระได้ ทั้งนี้เพราะในภาษา c ไม่มีตัวดำเนินการที่ทำงานกับสายอักขระได้โดยตรง ดังนั้นนักเขียนโปรแกรมจึงกำหนดค่าให้สายอักขระทีละ 1 ตัวอักขระ จนจบสายอักขระ เช่น

เมื่อมีการประกาศให้ *s เป็นตัวชี้ที่ชี้ไปยังสายอักขระ “Bangkok Thailand” และให้ t เป็นสายอักขระที่มี 100 ตัวดังนี้

```
char *s = “Bangkok Thailand”;
char t[100];
```

ในบทนี้จะกล่าวถึงรายละเอียดของฟังก์ชันสายอักขระที่มีการใช้งานบ่อย ได้แก่ ฟังก์ชัน strcpy ฟังก์ชัน strcmp และฟังก์ชัน strrchr

1. ฟังก์ชัน strcpy

รูปแบบ strcpy(t,s);

ฟังก์ชันนี้ทำหน้าที่คัดลอกข้อมูลจากสายอักขระ s ไปยังสายอักขระ t ซึ่งลำดับการเขียนคำสั่งคล้ายกับคำสั่งกำหนดค่า t = s และมีอาร์กิวเมนต์ 2 ตัว โดยอาร์กิวเมนต์ตัวแรก คือ สายอักขระปลายทางที่รับข้อมูล ส่วนอาร์กิวเมนต์ตัวที่ 2 คือ สายอักขระต้นแบบ

ตัวอย่างที่ 5.3 โปรแกรมแสดงฟังก์ชันคัดลอกสายอักขระ

```
/*
 A program and a function to copy one string to anotther.
*/
#include <stdio.h>
#define MAXLEN 80
main()
{
        char *from;
        char to[MAXLEN +1];
        void strcpy(char *dest, char *src);
        from = “copied string”;
        to[0] = ‘\0’;
        printf (“Before copy: from=\”%s\”, to=\”%s\”\n”, from, to);
        strcpy(to, from);
        printf (“After copy: from=\”%s\”, to=\”%s\”\n”, from, to);
        return 0;
}
void strcpy(char dest[], char source[])
{
        int i;
        for (i = 0; (dest[i] = source[i]) != ‘\0’; i++);
}
```

ผลลัพธ์ที่ได้

```
Before copy: from="copied string", to=""
After copy:  from="copied string", to="copied string"
```

2. ฟังก์ชัน strcmp

รูปแบบ strcmp(s1,s2);

การทำงานของฟังก์ชัน strcmp จะมีการเปรียบเทียบตัวอักขระของสายอักขระ 2 สาย ทีละคู่(s1[i],s2[i]) ตามลำดับ การทำงานนี้จบลงเมื่อพบตัวอักขระที่ต่างกัน ซึ่งหมายความว่าสายอักขระนั้นไม่เท่ากัน เนื่องจากสายอักขระสายหนึ่งสั้นกว่าหรือน้อยกว่าอีกสายหนึ่ง strcmp จะคืนค่าที่เป็นความแตกต่างของตัวอักขระ เมื่อสายอักขระไม่เท่ากัน กรณีที่สายอักขระเท่ากัน strcmp จะให้ค่าเป็นศูนย์ เช่น

X1	X2	ผลการเปรียบเทียบ	หมายเหตุ
meat	mean	+	เพราะ x1 [3] > x2[3] จึงทำให้ x1>x2 (ในที่นี้คืนค่าเป็น 6)
net	Set	-	เพราะ x1 [0] < x2[0] จึงทำให้ x1<x2 (ในที่นี้คืนค่าเป็น -5)
book	book	0	เพราะ x1 =x2 (ในที่นี้คืนค่าเป็น 0)

ตารางที่ 5.2 แสดงการเปรียบเทียบการทำงานของฟังก์ชัน strcmp

ตัวอย่างที่ 5.4 โปรแกรมแสดงการเปรียบเทียบสายอักขระ

```
/*
* A program and a function to compare one string to another.
*/
#include <stdio.h>
#define MAXLEN 80
main()
{
        char *str1;
        char *str2;
        int strcmp(char *s1, char *s2);
        str1 = "alex", str2 = "tony";
        printf("Comparing %s with %s, result %d.\n",
                str1, str2, strcmp(str1, str2));
        printf("Comparing %s with %s, result %d.\n",
                str2, str1, strcmp(str2, str1));
        printf("Comparing %s with %s, result %d.\n",
                str1, str1, strcmp(str1, str1));
        return 0;
}
/*
* Compare two strings, returning:
*       0 if they are the same
*       negative value if s1 < s2
*       positive value if s1 > s2
*/
int strcmp(char s1[], char s2[])
{
        int i;
        for (i = 0; s1[i] == s2[i] && s1[i] != '\0'; i++);
        return s1[i] -s2[i];       /* return difference between chars */
}
```

ผลลัพธ์ที่ได้

```
Comparing alex with tony, result -19.
Comparing tony with alex, result 19.
Comparing alex with alex, result 0.
```

3.ฟังก์ชัน strrchr

รูปแบบ strrchr(s,c)

การทำงานของฟังก์ชัน strrchr จะมีการเปรียบเทียบตัวอักขระ c ในสายอักขระ s โดยคืนค่าตัวชี้ที่ชี้ไปยังอักขระ c ตัวสุดท้ายที่ปรากฏในสายอักขระ s

ตัวอย่างที่ 5.5 โปรแกรมแสดงการทำงานของฟังก์ชัน strrchr จากคลังมาตรฐาน

```
/* Using strrchr */
#include <stdio.h>
#include <string.h>
main()
{
        char *string1 = "A zoo has many animals including zebra";
        int c = 'z';
        printf("%s\n%s'%c'%s\"%s\"\n",
                "The remainder of string1 begining with the",
                "last occurrence of character ", c,
                " is: ", strrchr(string1, c));
        return 0;
} /* End main */
```

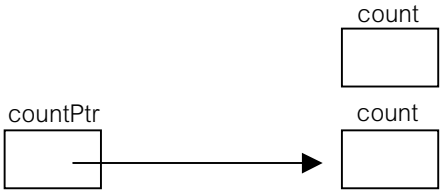
ผลลัพธ์ที่ได้

```
The remainder of string1 begining with the
last occurrence of character 'z' is: "zebra"
```

โปรแกรมนี้ค้นหาตัวอักษร 'z' ตัวสุดท้าย ที่ปรากฏอยู่ในสายอักขระ "A zoo has many animals including zebras" เป็นตัวสุดท้าย แล้วคืนค่าเป็นตัวชี้ของสายอักขระที่ตัวอักษร z ตัวสุดท้ายปรากฏอยู่

5.3 ตัวชี้

ตัวชี้ (pointer) คือ ประเภทของตัวแปรที่ใช้เก็บเลขที่อยู่ (address) ของตัวแปรที่มันชี้อยู่ โดยทั่วไปชื่อตัวแปรชื่อตัวแปรจะบอกถึงมูลค่า แต่ตัวชี้จะบอกตำแหน่งที่อยู่ ดังรูปที่ 5.3



count อ้างถึง ตัวแปรที่มีค่าภายในเป็น 7

countPtr อ้างถึง ตัวแปรที่มีค่าภายในเป็นเลขที่อยู่ของตัวแปร count

รูปที่ 5.3 การอ้างถึงตัวแปรและการอ้างถึงที่อยู่ของตัวแปร

รูปแบบการประกาศตัวชี้ในโปรแกรมทำได้ดังนี้

type *name;

เช่น int *countPtr, count;

คำสั่งนี้เป็นการประกาศตัวแปร countPtr ให้มีชนิดเป็นตัวชี้ชี้ไปยังจำนวนเต็ม และ count เป็นตัวแปรชนิดจำนวนเต็ม เมื่อตัวอักขระ * ปรากฏในส่วนการประกาศตัวแปรของโปรแกรม เป็นการบอกว่าตัวแปรได้รับการประกาศให้เป็นตัวชี้

การกำหนดค่าเริ่มต้นของตัวชี้ทำได้โดยกำหนดค่าว่าง(null) หรือ 0 ให้กับตัวชี้ ซึ่งหมายถึงตัวชี้ที่ไม่ได้ชี้ไปที่ใด เนื่องจาก null เป็นคำที่ถูกกำหนดไว้ในคลังมาตราฐาน stddef.h โปรแกรมที่เรียกใช้ null จึงต้องกำหนดคำสั่งตัวประมวลผลก่อน c โดยประกาศ #include <stddef.h> ไว้ที่ส่วนต้นของโปรแกรม

5.3.1 ตัวดำเนินการของตัวชี้

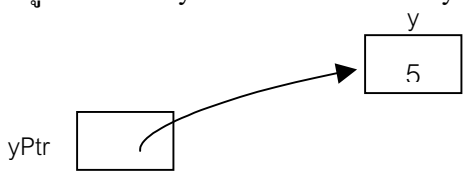
ในตัวดำเนินการของตัวชี้(pointer operator) เราใช้ตัวดำเนินการของเลขที่อยู่ (address operator) หรือ & ซึ่งเป็นตัวดำเนินการแบบเอกภาพที่คืนค่าเป็นที่อยู่ของตัวถูกดำเนินการ เช่น

int y = 5;

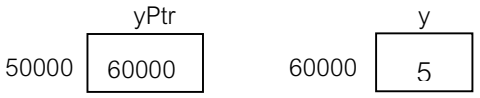
int *yPtr;

คำสั่ง yPtr = &y;

กำหนดให้ตัวชี้ yPtr เก็บค่าที่อยู่ของตัวแปร y อีกนัยหนึ่งคือตัวแปร yPtr ชี้ไปที่ y



รูปที่ 5.4 แสดงตัวชี้ yPtr ชี้ไปยังตัวแปร y ที่เก็บจำนวนเต็มในหน่วยความจำ



รูปที่ 5.5 แสดงค่าภายในของ y และ yPtr

5.3.2 การเข้าถึงมูลค่าที่ตัวชี้ชี้ไป

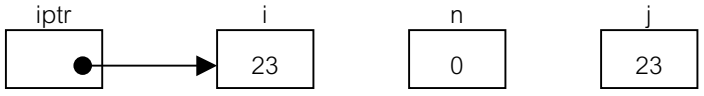
การเข้าถึงมูลค่าที่ตัวชี้ชี้ไปทำได้โดยเรียกใช้ตัวดำเนินการ * และตัวชี้ ผลที่ได้คือมูลค่าที่ตำแหน่งนั้น จากรูปที่ 5.4 ถ้าต้องการแสดงมูลค่าของ y ที่ yptr ชี้ โดย *yPtr จะได้ *yPtr =5 และจากรูปที่ 5.5 จะได้ yPtr = 60000 จะเห็นได้ว่า *yPtr คือ 5 นั้นคือมูลค่าของ y เพราะว่า yPtr คือตัวชี้ชี้ไปที่จำนวนเต็ม เราจึงสามารถใช้ *yPtr เหมือนกับตัวแปรโดยทั่วไป และนำไปใช้ในคำสั่งกำหนดค่า

ส่วนของคำสั่งต่อไปนี้แสดงการเข้าถึงข้อมูลโดยใช้ตัวชี้ และแสดงค่าภายในของตัวแปรต่างๆ ดังรูปที่ 5.6

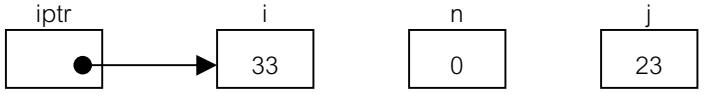
```
int     *iptr;          /* iptr คือ ตัวชี้ชี้ไปที่จำนวนเต็ม */
int     i, j, n ;       /* i,j,n เป็นจำนวนเต็ม */
i = 0;
iptr = &i;              /* จะได้ *iptr = 0 */
n = *iptr;              /* n ได้ค่าของ *iptr ที่ชี้ไป ซึ่งมีค่าเป็นศูนย์ */
*iptr = j;              /* ค่าภายในของ address ที่ *iptr ชี้ไปมีค่าเท่ากับค่าของตัวแปร j  */
*iptr = *iptr + 10;     /* เพิ่มค่าให้มูลค่าที่ *ptr ชี้ไปอีก 10 */
```

(a) i = 0; iptr = &i; n = 17; j = 23;

(b) n = * iptr;



(c) * iptr = j;



(d) * iptr = *iptr + 10;

รูปที่ 5.6 แสดงค่าภายในของตัวแปรต่างๆ ตามลำดับ

5.3.2 การท่องไปในแถวลำดับโดยใช้ตัวชี้

ในภาษา C แถวลำดับและตัวชี้นั้นสามารถใช้แทนกันได้ เมื่อเราประกาศแถวลำดับ ตัวแปลภาษาจะจองเนื้อที่จำนวนหนึ่งสำหรับเก็บแถวลำดับ และกำหนดชื่อแถวลำดับให้เป็นตัวชี้ชนิดคงที่ที่ชี้ไปยังสมาชิกอันดับที่ 0

เช่น กำหนด

```
int a[100];
```

ตัวแปลภาษาจะจองเนื้อที่ 100 ตำแหน่ง แต่ละตำแหน่งจะเก็บค่าของจำนวนเต็ม 1 จำนวนเท่านั้น เนื่องจากดัชนีของแถวลำดับเริ่มที่ตำแหน่งศูนย์ a จึงมีค่าเท่ากับ &a[0] แถวลำดับนี้จึงเริ่มจาก a[0], a[1], ….., a[99]

การประมวลผลแถวลำดับมี 2 แบบคือ การเข้าถึงโดยตรงทางดัชนีของแถวลำดับ ซึ่งเรียกใช้ชื่อแถวลำดับและดัชนีที่เขียนอยู่ภายใต้เครื่องหมายวงเล็บก้ามปู และค่าเริ่มต้นของดัชนีแถวลำดับคือศูนย์ ดังที่กล่าวไปแล้ว

ตัวอย่างที่ 5.6 โปรแกรมแสดงการเข้าถึงสมาชิกของแถวลำดับโดยใช้ดัชนี

```
/* Print first "n" elements using array subsc */
0    lude <stdio.h>
1    fine MAX 20
2    ()
3
4
5        int table [MAX];
6        int i;
7        void print_table (int a[], int n);
8        for (i = 0; i < MAX; i++)          /* provid */
9                table[i] = i;
10       print_table(table, MAX);           /* print t */
11       return 0;
12   End main */
13
14   print_table(int a[], int n)
15   {
16           int i;
17           for (i = 0; i < n; i++)
18                   printf("%d\n", a[i]);
19   } /* End void print_table */
```

ผลลัพธ์ที่ได้

การเข้าถึงสมาชิกของแถวลำดับโดยวิธีนี้เป็นวิธีแบบดั้งเดิม ซึ่งเข้าใจได้ง่ายและมีรูปแบบคล้ายกับการทำงานในภาษาอื่น อย่างไรก็ตามการท่องไปในแถวลำดับอย่างมีประสิทธิภาพนั้นสามารถทำได้อีกวิธีหนึ่งโดยการใช้ตัวชี้เข้าถึงสมาชิกของแถวลำดับ โดยวิธีนี้เราสามารถบวกหรือลบจำนวนเต็มกับตัวชี้ได้

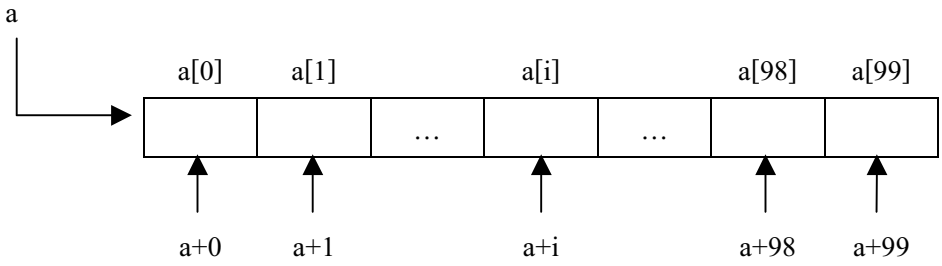
ตัวอย่างที่ 5.7 โปรแกรมแสดงการเข้าถึงสมาชิกของแถวลำดับโดยใช้ตัวชี้

```
/*
    nt first "n" elements using pointer indexing.

    lude <stdio.h>
    fine MAX 20
    ()

        int table [MAX];
        int i;
        void print_table(int a[], int n);
        for (i = 0; i < MAX; i++) /* provide some initial values */
                table[i] = i;
        print_table(table, MAX);        /* print the array */
        return 0;
    End main */
void print_table(int a[], int n)
{
        int i;
        int *ptr;
        for (ptr = a, i = 0; i < n; ptr++, i++)
                printf("%d\n", *ptr);
} /* End void print_table */
```

ผลลัพธ์ที่ได้

5.3.3 การดำเนินงานทางคณิตศาสตร์ของตัวชี้

การดำเนินงานทางคณิตศาสตร์ของตัวชี้ทำได้โดยอัตโนมัติ เช่น การบวก 1 กับตัวชี้ของแถวลำดับ จะทำให้ตัวชี้ชี้ไปยังสมาชิกช่องถัดไป โดยไม่คำนึงถึงว่าประเภทข้อมูลของแถวลำดับนั้นจะเป็นอย่างไร



รูปที่ 5.7 แสดงแถวลำดับ a ประกอบด้วยจำนวนเต็ม 100 ตัว

การกำหนดแถวลำดับในรูปที่ 5.7 โดย a เป็นตัวชี้ชี้ไปที่สมาชิกตัวแรก (&a[0])

เมื่อเขียน a+1 คือตัวชี้ที่ชี้ไปยังสมาชิกตัวที่สอง (&a[1])

a+2 คือตัวชี้ที่ชี้ไปยังสมาชิกตัวที่สาม (&a[2])

:

a+i คือตัวชี้ที่ชี้ไปยังสมาชิกตัวที่ i+1(&a[i])

เนื่องจาก a+i คือที่อยู่ของ a[i] ดังนั้น *(a+i) มีค่าเท่ากับ a[i] เราจึงสามารถกำหนดค่าเริ่มต้นของ a[3] ให้เป็น 0 โดยการใช้ดัชนี

```
a[3] = 0;
```

หรืออีกวิธีการหนึ่งโดยใช้ตัวชี้ดังนี้

```
*(a+3) = 0;
```

สำหรับวิธีการที่ 2 นั้นตัวแปลภาษาเปลี่ยนดัชนีของแถวลำดับไปเป็นตัวชี้ a[i] คือ *(a+i) การเขียนวิธีที่ 2 นี้อาจจะซับซ้อนกว่าการเขียนวิธีแรก แต่เมื่อนำมาใช้ในโปรแกรมจะทำให้การทำงานมีประสิทธิภาพกว่าวิธีการเข้าถึงสมาชิกของแถวลำดับแบบโดยตรง การเข้าถึงสมาชิกในแถวลำดับทำได้ดังตัวอย่างโปรแกรมที่ 5.7

5.3.4 การเปรียบเทียบตัวชี้

ในภาษา C นอกจากจะนำตัวชี้มาบวกลบได้แล้ว ยังสามารถใช้ตัวดำเนินการสัมพันธ์ (relational operator) เพื่อเปรียบเทียบตัวชี้ ตัวดำเนินการเหล่านี้ได้แก่

==	เท่ากัน (equal)
!=	ไม่เท่ากัน (not equal)
<	น้อยกว่า (less than)
<=	น้อยกว่าหรือเท่ากับ (less than or equal)
>	มากกว่า (greater than)
>=	มากกว่าหรือเท่ากับ (greater than or equal)

ตัวชี้ 2 ตัวเท่ากันก็ต่อเมื่อตัวชี้ทั้ง 2 ชี้ไปยังตำแหน่งเดียวกัน และไม่เท่ากันต่อเมื่อชี้ไปที่ตำแหน่งต่างกัน ตัวชี้ตัวหนึ่งมีค่าน้อยกว่าอีกตัวหนึ่งถ้าชี้ไปที่ตำแหน่งที่ต่ำกว่าในหน่วยความจำ เช่น &a[3] น้อยกว่า &a[5]

หมายเหตุ การเปรียบเทียบตัวชี้นั้นกระทำกับตัวชี้ของแถวลำดับเดียวกันเท่านั้น

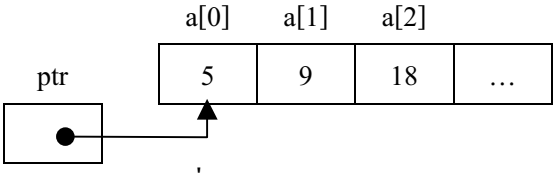
ในตัวอย่างโปรแกรมที่ 5.7 ใช้การเปรียบเทียบตัวชี้ในการพิมพ์ตารางและใช้การวนซ้ำที่มีประสิทธิภาพ เพราะไม่มีการทดสอบตัวนับเมื่อท่องไปในแถวลำดับ แต่ใช้การเปรียบเทียบดัชนีตัวชี้ (ptr) ว่าเป็นสมาชิกตัวสุดท้ายของแถวลำดับ (endptr) หรือไม่ วิธีนี้จะทำให้เวลาในการทำงานลดลง

อีกกรณีหนึ่งสามารถเขียนการวนซ้ำเพื่อพิมพ์มูลค่าจากตารางได้ดังนี้

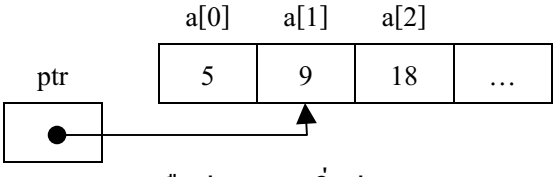
```
endptr = (ptr = table) +n –1;
while (ptr <= endptr)
        printf (”%d\n”, *ptr++);
```

ในที่นี้ *ptr++ หมายถึง ค่าที่ ptr ชี้ไป แล้วคืนค่านั้น ต่อมาเพิ่มค่าให้ตัวชี้

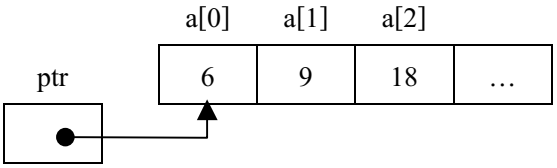
(*ptr)++ หมายถึง เพิ่มค่าให้มูลค่าที่ตัวชี้ชี้ไป แล้วคืนค่าเป็นค่าเดิมก่อนการเพิ่มค่า



(a) กำหนดค่าเริ่มต้นให้ ptr = &a[0].



(b) *ptr++ คืนค่า 5 และเพิ่มค่า ptr



(c) (*ptr)++ เพิ่มค่าภายในของ a[0] และคืนค่า 5

รูปที่ 5.8 แสดงความแตกต่างระหว่าง ptr++ และ (*ptr)++

แบบฝึกหัด

1. จงตอบว่าข้อความดังต่อไปนี้ถูกหรือผิด ถ้าผิดจงแสดงเหตุผล
   ก. แถวลำดับสามารถเก็บข้อมูลได้หลายชนิด
   ข. ดัชนีของแถวลำดับนั้นสามารถกำหนดเป็น float
   ค. เมื่อกำหนดค่าเริ่มต้นของแถวลำดับโดยมูลค่าที่จะใส่ในแถวลำดับมีจำนวนน้อยกว่าสมาชิกของแถวลำดับที่กำหนดให้ จะเกิดข้อผิดพลาดขึ้น
   ง. สามารถสร้างแถวลำดับที่มีดัชนีมากกว่า 1 ตัวได้
2. จงบอกว่าส่วนของโปรแกรมต่อไปนี้ถูกหรือผิด ถ้าผิดจงเขียนให้ถูก

   ก. #define size 100;

   ข. size = 10;

   ค. int b[10] = {0}, i;

   ```
   for (i = 0; i <= 10; i++)
           b[i] = 1;
   ```

   ง. #include <stdio.h>;

   จ. int a[2][2] = {{1,2}, {3,4}};

   ```
   a[1,1] = 5;
   ```

3. พนักงานขายของบริษัทแห่งหนึ่งได้รับค่าจ้างสัปดาห์ละ 2,000 บาท และเขาจะได้ค่านายหน้าอีก 9% จากยอดขายในแต่ละสัปดาห์ เช่นเมื่อพนักงานขายขายสินค้าได้ 5,000 บาท ใน 1 สัปดาห์ เขาจะได้รับเงิน 2,000 บาท รวมกับค่านายหน้าของการขายอีก 9% ของ 5,000 บาท คือ 450 บาท รวมทั้งสิ้น 2,450 บาท จงเขียนโปรแกรมโดยใช้แถวลำดับเก็บข้อมูลที่แสดงยอดขาย และเงินที่พนักงานจะได้รับในแต่ละสัปดาห์ เมื่อกำหนดเงินได้ของพนักงานขายจากการขายสินค้า
4. จงเขียนโปรแกรมที่สามารถรับข้อมูลนำเข้าที่เป็นตัวเลขได้ไม่เกิน 5 จำนวน เพื่อคำนวณหาค่าเฉลี่ย ค่าสูงสุด ค่าต่ำสุด ของข้อมูลนำเข้านั้น โดยเก็บข้อมูลไว้ในแถวลำดับ
5. จงพิจารณาว่าส่วนของโปรแกรมต่อไปนี้ผิดหรือถูก ถ้าผิดจงแก้ให้ถูก

   ก. char s[10];

   ```
   strncpy (s, “hello”, 5);
   printf (“%s\n”, s);
   ```

   ข. printf (“%s”, ‘a’);

   ค. char s[12];

strcpy (s, “Welcome Home”);

ง. if (strcmp (string1, string2))

printf (“The strings are equal\n”);

6. จงเขียนโปรแกรมที่อ่านสายอักขระและพิมพ์เฉพาะสายอักขระที่ขึ้นต้นด้วยอักษร “b”
7. จงเขียนโปรแกรมซึ่งรับข้อมูลนำเข้าเป็นหมายเลขโทรศัพท์ ซึ่งเขียนในรูปของสายอักขระ เช่น (662)555-5555 และให้แยกหมายเลขโทรศัพท์ออกเป็น 2 ส่วน คือ รหัสทางไกล กับเบอร์โทรศัพท์
8. จากส่วนของโปรแกรมที่กำหนดให้นี้ จงบอกว่าข้อความนั้นถูกหรือผิด ถ้าผิดให้แก้คำสั่งที่ถูก

```
int *zPtr;
int *aPtr = NULL;
void *sPtr = NULL;
int number, i;
int z[5] = {1,2,3,4,5};
sPtr = z;
```

ก. ++zPtr;

ข. number = zPtr;

ค. number = *zPtr[2];

ง. for (i = 0; i <= 5; i++)

printf (“%d”, zPtr[i]);

จ. number = *sPtr;

ฉ. ++z;