

## แถวลำดับ (Array)

ในงานประยุกต์บางอย่างอาจจะต้องมีการกระทำกับข้อมูลที่มีจำนวนมาก และเป็นข้อมูลประเภทเดียวกัน ตัวอย่างเช่น ถ้าเราต้องการคำนวณหาค่าเฉลี่ยของคะแนนสอบของนักศึกษาจำนวน 30 คน เราจะต้องมีการกำหนดชื่อตัวแปร จำนวน 30 ชื่อ คือ test\_score1, test\_score2, ..., test\_score30 เพื่อเก็บข้อมูลคะแนนสอบของนักศึกษา จำนวน 30 คน ในหน่วยความจำ ในการใช้งานจริง เวลาเราอ้างถึงตัวแปรแต่ละตัวอาจจะยุ่งยาก ในภาษา C ได้กำหนดให้ง่ายโดยใช้เพียงชื่อตัวแปร ตามด้วยดัชนี (Index) หรือตัวบอกลำดับ (subscript) โดยถูกปิดล้อมด้วยเครื่องหมาย [ และ ] เช่น test\_score[30]

โดย test\_score[0] จะแทนคะแนนสอบของนักศึกษาคนที่ 1

test\_score[1] จะแทนคะแนนสอบของนักศึกษาคนที่ 2

: : :

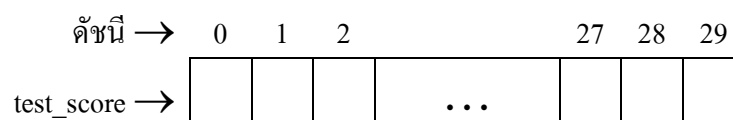
test\_score[29] จะแทนคะแนนสอบของนักศึกษาคนที่ 30

**หมายเหตุ** ในภาษา C ตัวบอกลำดับหรือดัชนีจะเริ่มจาก 0 ซึ่งโครงสร้างข้อมูลแบบนี้ในภาษา C เราจะเรียกว่าแถวลำดับ (array)

### 1. แถวลำดับ (array)

แถวลำดับ (array) คือกลุ่มของข้อมูลที่มีจำนวนที่แน่นอน และมีชนิดของข้อมูลเป็นแบบเดียวกันหมด โดยจะถูกกำหนดเนื้อที่ในหน่วยความจำให้ต่อเนื่องกันไป

เช่น กำหนด int test\_score[30] จะหมายถึง



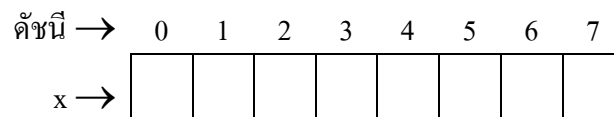
## 2. รูปแบบทั่วไปของการประกาศตัวแปรแถวลำดับ 1 มิติ

storage-class data-type array[expression];

โดย	storage-class	คือ ประเภทหน่วยเก็บ เช่น static, auto เป็นต้น ซึ่งอาจจะมีหรือไม่มีก็ได้  ถ้านิยามภายในฟังก์ชันใด ๆ จะมีประเภทหน่วยเก็บเป็นแบบอัตโนมัติ และถ้านิยามภายนอกฟังก์ชันใด ๆ จะมีประเภทหน่วยเก็บเป็นแบบภายนอก ถ้าภายในแต่ละฟังก์ชันไม่ได้กำหนด จะมี default เป็นแบบอัตโนมัติ
	data-type	คือ ชนิดของข้อมูล เช่น int, char เป็นต้น
	array	คือ ชื่อของตัวแปรแถวลำดับที่ตั้งตามกฎเกณฑ์การตั้งชื่อ
	expression	คือ ค่าจำนวนเต็มบวก ซึ่งอาจจะเป็นค่าคงตัวหรือชื่อคงตัว เพื่อบ่งบอกจำนวนของตัวแปรแถวลำดับ

### ตัวอย่างที่ 1 กำหนด int x[8]

หมายความว่า จะเป็นการประกาศเพื่อบอกให้คอมไพเลอร์เตรียมเนื้อที่ให้กับตัวแปรแถวลำดับ x จำนวน 8 ตัว ให้มีเนื้อที่ต่อเนื่องกันไปโดยแต่ละตัวจะมีข้อมูลเป็นชนิด int ซึ่งแสดงได้ดังรูปที่ 1



รูปที่ 1

ในภาษา C จะกำหนดให้ สมาชิกลำดับที่ 1 มีดัชนีคือ 0  
 สมาชิกลำดับที่ 2 มีดัชนีคือ 1  
 : :  
 สมาชิกลำดับที่ 8 มีดัชนีคือ 7

โดย  $x[0]$  คือ ค่าของสมาชิกในตัวแปรแถวลำดับในลำดับที่ 1  
 $x[1]$  คือ ค่าของสมาชิกในตัวแปรแถวลำดับในลำดับที่ 2  
 : : :  
 $x[7]$  คือ ค่าของสมาชิกในตัวแปรแถวลำดับในลำดับที่ 8

**ตัวอย่างที่ 2** กำหนด `double x[8]` และ

กำหนดค่าเริ่มต้นในตัวแปรแถวลำดับ `x` ดังรูปที่ 2

$x[0]$	$x[1]$	$x[2]$	$x[3]$	$x[4]$	$x[5]$	$x[6]$	$x[7]$
16.0	12.0	6.0	8.0	2.5	12.0	14.0	-54.5

รูปที่ 2

ข้อความสั่ง	ความหมาย
<code>printf(“%.1f”, x[0]);</code>	แสดงค่าของ $x[0]$ บนจอภาพ คือ 16.0
<code>x[3] = 25.0;</code>	นำค่า 25.0 ไปไว้ใน $x[3]$
<code>sum = x[0] + x[1];</code>	หาผลรวมของ $x[0]$ กับ $x[1]$ ไปไว้ในตัวแปร <code>sum</code>
<code>sum += x[2];</code>	หาผลรวมของ $x[2]$ กับ <code>sum</code> ไปไว้ในตัวแปร <code>sum</code>
<code>x[3] += 1.0;</code>	หาผลรวมของ $x[3]$ กับ 1.0 ไปไว้ในตัวแปร $x[3]$
<code>x[2] = x[0] + x[1];</code>	หาผลรวมของ $x[0]$ กับ $x[1]$ ไปไว้ในตัวแปร $x[2]$

เมื่อมีการกระทำการ (execute) ข้อความสั่งทั้งหมดจะได้ผลลัพธ์ ดังรูปที่ 3

$x[0]$	$x[1]$	$x[2]$	$x[3]$	$x[4]$	$x[5]$	$x[6]$	$x[7]$
16.0	12.0	28.0	26.0	2.5	12.0	14.0	-54.5

รูปที่ 3

ตัวอย่างที่ 3 กำหนดค่าเริ่มต้นในตัวแปรแถวลำดับ x ดังรูปที่ 4

X[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	-54.5

รูปที่ 4

และกำหนด `int i = 5;`

ข้อความสั่ง	ความหมาย
<code>printf (“%d %.1f”, 4, x[4] );</code>	แสดงค่า 4 และ 2.5 บนจอภาพ (ค่าของ $x[4] = 2.5$ )
<code>printf (“%d %.1f”, i, x[i] );</code>	แสดงค่า 5 และ 12.0 บนจอภาพ (ค่าของ $x[5] = 12.0$ )
<code>printf (“%.1f”, x[i] + 1 );</code>	แสดงค่า 13.0 บนจอภาพ (นำค่า $x[5]$ บวก 1)
<code>printf (“%.1f”, x[i] + i );</code>	แสดงค่า 17.0 บนจอภาพ (นำค่า $x[5]$ บวก 5)
<code>printf (“%.1f”, x[i + 1] );</code>	แสดงค่า 14.0 บนจอภาพ (นำค่า $x[6]$ บวก 14.0)
<code>printf (“%.1f”, x[i + i] );</code>	ไม่ถูกต้อง เนื่องจาก $x[5 + 5] = x[10]$ ไม่ได้กำหนดไว้
<code>printf (“%.1f”, x[2 * i] );</code>	ไม่ถูกต้อง เนื่องจาก $x[2 * 5] = x[10]$ ไม่ได้กำหนดไว้
<code>printf (“%.1f”, x[2 * i - 3] );</code>	แสดงค่า -54.5 (ค่าของ $x[7] = -54.5$ )
<code>printf (“%.1f”, x[ (int) x [4] ] );</code>	แสดงค่า 6.0 (ค่าของ $x[2] = 6$ )
<code>printf (“%.1f”, x[i++] );</code>	แสดงค่า 12.0 (ค่าของ $x[5] = 12.0$ ) แล้วเพิ่มค่า $i$ ขึ้นอีก 1 ทำให้ $i = 6$
<code>printf (“%.1f”, x[-i] );</code>	แสดงค่า 12.0 (ค่าของ $x[5] = 12.0$ ) ลดค่า $i$ ลง 1 ก่อน แล้วแสดงค่าของ $x[5]$
<code>x[i - 1] = x[i];</code>	กำหนดค่าของ $x[5]$ คือ 12.0 ไปให้ $x[4]$
<code>x[i] = x[i + 1];</code>	กำหนดค่าของ $x[6]$ คือ 14.0 ไปให้ $x[5]$
<code>x[i] - 1 = x[i];</code>	เป็นข้อความสั่งกำหนดค่าไม่ถูกต้อง

### 3. รูปแบบทั่วไปของการกำหนดค่าเริ่มต้นให้กับตัวแปรแถวลำดับ 1 มิติ

```
storage-class data-type array[expression] = {value 1, value 2, ..., value n};
```

โดย Value 1 เป็นการกำหนดค่าเริ่มต้นให้กับสมาชิกลำดับที่ 1 ของตัวแปรแถวลำดับ  
 Value 2 เป็นการกำหนดค่าเริ่มต้นให้กับสมาชิกลำดับที่ 2 ของตัวแปรแถวลำดับ  
 : : :  
 Value n เป็นการกำหนดค่าเริ่มต้นให้กับสมาชิกลำดับที่ n ของตัวแปรแถวลำดับ

**หมายเหตุ** ตัวแปรแถวลำดับที่มีหน่วยเก็บเป็นแบบอัตโนมัติ จะไม่เหมือนกับตัวแปรอัตโนมัติ กล่าวคือ ไม่สามารถกำหนดค่าเริ่มต้นได้ แต่ตัวแปรแถวลำดับที่มีหน่วยเก็บเป็นแบบภายนอก สามารถกำหนดค่าเริ่มต้นได้

**ตัวอย่างที่ 4** กำหนด `int digits[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};`

จะหมายถึงการกำหนดค่าเริ่มต้นให้กับตัวแปรแถวลำดับ `digits` โดย

```
digits[0] = 1
digits[1] = 2
digits[2] = 3
digits[3] = 4
digits[4] = 5
digits[5] = 6
digits[6] = 7
digits[7] = 8
digits[8] = 9
digits[9] = 10
```

ตัวอย่างที่ 5 กำหนด `int digits[10] = {3, 3, 3};`

จะหมายถึงการกำหนดค่าเริ่มต้นให้กับตัวแปรแถวลำดับ `digits` 3 ตัวแรก มีค่าเท่ากับ 3 ส่วนที่เหลือจะถูกกำหนดให้เป็น 0 หมดโดยอัตโนมัติ ดังนี้

```
digits[0] = 3
digits[1] = 3
digits[2] = 3
digits[3] = 0
digits[4] = 0
digits[5] = 0
digits[6] = 0
digits[7] = 0
digits[8] = 0
digits[9] = 0
```

} ถูกกำหนดให้เป็น 0 โดยอัตโนมัติ

ตัวอย่างที่ 6 กำหนด `int digits[] = {1, 2, 3, 4, 5, 6};`

ในตัวอย่างนี้ไม่ได้มีการกำหนดจำนวนตัวให้กับตัวแปรแถวลำดับ ดังนั้นจำนวนตัวของตัวแปรแถวลำดับจะเท่ากับจำนวนที่ได้กำหนดมา

```
digits[0] = 1
digits[1] = 2
digits[2] = 3
digits[3] = 4
digits[4] = 5
digits[5] = 6
```

หมายเหตุ กำหนด `int digits[]` จะไม่ถูกต้อง เนื่องจากไม่ได้กำหนดจำนวนสมาชิก และจะไม่ผ่านการแปลโปรแกรม

ตัวอย่างที่ 7 กำหนด `char color[3] = {'R', 'E', 'D'};`

จะหมายถึงการกำหนดค่าเริ่มต้นให้กับตัวแปรแถวลำดับ `color` โดย

```
color[0] = 'R'
color[1] = 'E'
color[2] = 'D'
```

ตัวอย่างที่ 8 กำหนด `char flag[5] = {'T', 'R', 'U', 'E'};`

จะหมายถึงการกำหนดค่าเริ่มต้นให้กับตัวแปรแถวลำดับ `flag` 4 ตัวแรก และสมาชิก  
ลำดับสุดท้ายจะถูกกำหนดให้มีค่าเท่ากับ `'\0'` โดย

```
flag[0] = 'T'
flag[1] = 'R'
flag[2] = 'U'
flag[3] = 'E'
flag[4] = '\0' ← เป็นอักขระว่าง (null character)
```

ตัวอย่างที่ 9 กำหนด `char color[3] = "RED";`

จะหมายถึง

```
color[0] = 'R'
color[1] = 'E'
color[2] = 'D'
```

ตัวอย่างที่ 10 กำหนด `char color[] = "RED";`

จะหมายถึง

```
color[0] = 'R'
color[1] = 'E'
color[2] = 'D'
color[3] = '\0' ← เป็นอักขระว่าง (null character)
                 จะถูกกำหนดให้โดยอัตโนมัติ
```

หมายเหตุ ตัวอย่างที่ 9 และ 10 จะไม่เหมือนกัน โดย

ตัวอย่างที่ 9 จะไม่ถูกต้อง เนื่องจากจะต้องมีอักขระว่างเติมเข้าไปด้วย

**ตัวอย่างโปรแกรมที่ 11** เป็นโปรแกรมที่มีการประกาศตัวแปรแถวลำดับ array ไว้นอกฟังก์ชัน ซึ่งจะถือว่าเป็นตัวแปรส่วนกลาง และมีค่าเริ่มต้นเป็นศูนย์โดยอัตโนมัติ

```
#include <stdio.h>

int array[3]; ← ตัวแปรส่วนกลาง

main ()
{
    printf ( "contents of array[0] => %d\n", array[0] );
    printf ( "contents of array[1] => %d\n", array[1] );
    printf ( "contents of array[2] => %d\n", array[2] );
}
```

```
contents of array[0] => 0
contents of array[1] => 0
contents of array[2] => 0
```

**ตัวอย่างโปรแกรมที่ 12** เป็นโปรแกรมที่มีลักษณะคล้ายกับโปรแกรมในตัวอย่างที่ 11 เพียงแต่ว่า มีการประกาศตัวแปรแถวลำดับในฟังก์ชันในกรณีนี้จะถือว่าเป็นตัวแปรอัตโนมัติ และไม่มีการกำหนดค่าเริ่มต้นให้

```
#include <stdio.h>

main ()
{
    int array[3]; ← ตัวแปรอัตโนมัติ

    printf ( "contents of array[0] => %d\n", array[0] );
    printf ( "contents of array[1] => %d\n", array[1] );
    printf ( "contents of array[2] => %d\n", array[2] );
}
```

```
contents of array[0] => 0
contents of array[1] => -5672
contents of array[2] => 58 } ← เป็นค่าที่เป็นขยะ
```

จากตัวอย่างนี้ ผลลัพธ์ของตัวแปรแถวลำดับ array[0] ถึง array[2] จะเป็นค่าที่เกิดจากการตกค้างของหน่วยความจำก่อนหน้าที่เราเรียกว่า ค่าที่เป็นขยะ (garbage value)



**หมายเหตุ** ตัวอย่างโปรแกรมที่ 12 ตัวแปรแถวลำดับ array จะถูกประกาศไว้ภายในฟังก์ชัน เราจะถือว่าเป็นตัวแปรอัตโนมัติหรือตัวแปรเฉพาะที่ และจะไม่มีกำหนดค่าเริ่มต้นเป็น 0 ถ้าต้องการกำหนดค่าเริ่มต้นเป็น 0 ให้กำหนดประเภทหน่วยเก็บเป็นแบบ static ดังรูปที่ 5

```
main ()
{
    static int array[3];
    .
    .
}
```

รูปที่ 5

**ตัวอย่างโปรแกรมที่ 13** เป็นโปรแกรมที่มีการกำหนดค่าเริ่มต้นให้กับตัวแปรภายในฟังก์ชัน

```
#include <stdio.h>

main ()
{
    int array[3];

    array[0] = 10;
    array[1] = 20;
    array[2] = 30;

    printf ( "contents of array[0] => %d\n", array[0] );
    printf ( "contents of array[1] => %d\n", array[1] );
    printf ( "contents of array[2] => %d\n", array[2] );
}
```

```
contents of array[0] => 10
contents of array[1] => 20
contents of array[2] => 30
```

**ตัวอย่างโปรแกรมที่ 14** เป็นโปรแกรมที่มีการกำหนดค่าเริ่มต้นให้กับตัวแปรแถวลำดับ และใช้ชื่อคำสั่ง `for` พิมพ์ค่าตัวแปรแถวลำดับแต่ละตัวออกสู่จอภาพ

```
#include <stdio.h>

main ()
{
    int number_array[7];
    int index ;

    /* Place values in the array. */

    number_array[0] = 0 ;
    number_array[1] = 2 ;
    number_array[2] = 4 ;
    number_array[3] = 8 ;
    number_array[4] = 16 ;
    number_array[5] = 32 ;
    number_array[6] = 64 ;

    for (index = 1 ; index <= 5 ; index++)
    {
        printf ( "number_array[%d] = %d", index, number_array[index] );
        printf ( "number_array[%d + 1] = %d", index, number_array[index + 1] );
        printf ( "number_array[%d - 1] = %d\n", index, number_array[index - 1] );
    }
}
```

<pre>number_array[1] = 2 number_array[1 + 1] = 4 number_array[1 - 1] = 0 number_array[2] = 4 number_array[2 + 1] = 8 number_array[2 - 1] = 2 number_array[3] = 8 number_array[3 + 1] = 16 number_array[3 - 1] = 4 number_array[4] = 16 number_array[4 + 1] = 32 number_array[4 - 1] = 8 number_array[5] = 32 number_array[5 + 1] = 64 number_array[5 - 1] = 16</pre>
--

**ตัวอย่างโปรแกรมที่ 15** เป็นโปรแกรมพิมพ์ตัวอักษรออกสู่จอภาพ

```
#include <stdio.h>

main ()
{
    static char array[7] = {'H', 'e', 'l', 'l', 'o'};

    printf ( "%s", array );
}
```




Hello

**ตัวอย่างโปรแกรมที่ 16** เป็นโปรแกรมที่มีการกำหนดค่าเริ่มต้นภายในฟังก์ชัน เป็นแบบ static และพิมพ์ค่าแต่ละตัวออกสู่จอภาพ

```
#include <stdio.h>

main ()
{
    static int array[3] = {10, 20, 30};
    int index;

    for (index = 0; index < 3; index++)
    {
        printf ( "array[%d] = %d\n", index, array[index] );
    }
}
```



```
array[0] = 10
array[1] = 20
array[2] = 30
```

ตัวอย่างโปรแกรมที่ 17 เป็นโปรแกรมที่มีการกำหนดค่าเริ่มต้นภายในฟังก์ชัน และพิมพ์ค่าแต่ละตัวออกสู่จอภาพ

```
#include <stdio.h>

main ()
{
    int values[10];
    int index ;

    values[0] = 197 ;
    values[2] = -100 ;
    values[5] = 350 ;
    values[3] = values[0] + values[5] ;
    values[9] = values[5] / 10 ;
    --values[2];

    for ( index = 0 ; index < 10 ; ++index )
        printf ( "values[%d] = %d\n", index, values[index] );
}
```

```
values[0] = 197
values[1] = 0
values[2] = -101
values[3] = 547
values[4] = 0
values[5] = 350
values[6] = 0
values[7] = 0
values[8] = 0
values[9] = 35
```

**ตัวอย่างโปรแกรมที่ 18** เป็นโปรแกรมแสดงลำดับ ฟิโบนักชี (Fibonacci) จำนวน 15 พจน์แรก

```
/* Program to generate the first 15 Fibonacci numbers */


#include <stdio.h>

main ()
{
    int Fibonacci[15], i ;

    Fibonacci[0] = 0 ; /* by definition */
    Fibonacci[1] = 1 ; /* ditto */

    for ( i = 2 ; i < 15 ; ++i )
        Fibonacci[i] = Fibonacci[i - 2] + Fibonacci[i - 1] ;

    for ( i = 0 ; i < 15 ; ++i )
        printf ( "%d\n", Fibonacci[i] ) ;
}
```



```
0
1
1
2
3
5
8
13
21
34
55
89
144
233
377
```

ตัวอย่างโปรแกรมที่ 19 เป็นโปรแกรมแสดงจำนวนเฉพาะที่อยู่ในช่วงตั้งแต่ 1 ถึง 50

```
/* Modified program to generate prime numbers */

#include <stdio.h>

main ()
{
    int p, is_prime, i, prime[50], prime_index = 2 ;

    prime[0] = 2 ;
    prime[1] = 3 ;

    for ( p = 5 ; p <= 50 ; p = p + 2 )
    {
        is_prime = 1 ;

        for ( i = 1 ; is_prime &&
                p / primes[i] >= primes[i] ; ++i )
            if ( p % prime[i] == 0 )
                is_prime = 0 ;

        if ( is_primes )
        {
            primes[prime_index] = p ;
            ++prime_index ;
        }
    }
    for ( i = 0 ; i < prime_index ; ++i )
        printf ( "%d ", primes[i] );

    printf ( "\n" );
}
```

2	3	5	7	11	13	17	19	23	29	31	37	41	43	47
---	---	---	---	----	----	----	----	----	----	----	----	----	----	----

**ตัวอย่างโปรแกรมที่ 20** เป็นโปรแกรมแสดงข้อความ “Hello!” โดยตัวแปรแถวลำดับ word

```
#include <stdio.h>

main ()
{
    static char word[ ] = { 'H', 'e', 'l', 'l', 'o', '!' }
    int i;

    for (i = 0; i < 6; ++i)
        printf ("%c", word[i]);

    printf ("\n");
}
```

```
Hello!
```

**ตัวอย่างโปรแกรมที่ 21** เป็นโปรแกรมที่มีการกำหนดค่าเริ่มต้นให้กับตัวแปรแถวลำดับ 5 จำนวนแรก ส่วน 5 จำนวนหลัง ได้จากการคำนวณและพิมพ์ค่าทุกตัวออกสู่จอภาพ

```
#include <stdio.h>

main ()
{
    static int array_values[10] = { 0, 1, 4, 9, 16 };
    int i;

    for (i = 5; i < 10; ++i)
        array_values[i] = i * i;

    for (i = 0; i < 10; ++i)
        printf ("array_values[%d] = %d\n", i, array_values[i]);
}
```

```
array_values[0] = 0
array_values[1] = 1
array_values[2] = 4
array_values[3] = 9
array_values[4] = 16
array_values[5] = 25
array_values[6] = 36
array_values[7] = 49
array_values[8] = 64
array_values[9] = 81
```

**ตัวอย่างโปรแกรมที่ 22** เป็นโปรแกรมการแปลงเลขฐานสิบไปยังเลขฐานตามที่ต้องการโดยให้ผู้ใช้ป้อนเลขฐานที่ต้องการแปลง

```

/* Program to convert a positive integer to another base */

#include <stdio.h>

main ()
{
    static char base_digits[16] =
        { '0', '1', '2', '3', '4', '5', '6', '7',
          '8', '9', 'A', 'B', 'C', 'D', 'E', 'F' };
    int converted_number[64];
    long int number_to_convert;
    int next_digit, base, index = 0;

    /* get the number and the base */

    printf ( "Number to be converted? " );
    scanf ( "%ld", &number_to_convert );
    printf ( "Base? " );
    scanf ( "%d", &base );

    /* convert to the indicated base */

    do
    {
        converted_number[index] = number_to_convert % base;
        ++index;
        number_to_convert = number_to_convert / base;
    }
    while ( number_to_convert != 0 );

    /* display the results in reverse order */

    printf ( "Converted number = " );

    for ( --index; index >= 0; --index )
    {
        next_digit = converted_number[index];
        printf ( "%c", base_digits[next_digit] );
    }

    printf ( "\n" );
}

```

```

Number to be converted? 10
Base? 2
Converted number = 1010

```

```

Number to be converted? 128362
Base? 16
Converted number = 1F56A

```



**ตัวอย่างโปรแกรมที่ 23** เป็นโปรแกรมที่มีการกำหนดจำนวนตัวในตัวแปรแถวลำดับ โดยใช้

```
#define SIZE 10

/* Initialize the elements of array s to the even integers from 2 to 20 */
#include <stdio.h>
#define SIZE 10

/* function main begins program execution */
int main ()
{
    /* symbolic constant SIZE can be used to specify array size */
    int s[SIZE]; /* array s has 10 elements */
    int j; /* counter */

    for (j = 0 ; j < SIZE ; j++) { /* set the values */
        s[j] = 2 + 2 * j;
    } /* end for */

    printf( "%s%13s\n", "Element", "Value" );

    /* output contents of array s in tabular format */
    for (j = 0 ; j < SIZE ; j++) {
        printf( "%7d%13d\n", j, s[j] );
    } /* end for */

    return 0 ; /* indicates successful termination */

} /* end main */
```

Element	Value
0	2
1	4
2	6
3	8
4	10
5	12
6	14
7	16
8	18
9	20

**ตัวอย่างโปรแกรมที่ 24** เป็นโปรแกรมที่หาผลรวมของค่าในตัวแปรแถวลำดับทุกตัว

```
/* Compute the sum of the elements of the array */
#include <stdio.h>
#define SIZE 12

/* function main begins program execution */
int main ()
{
    /* use initializer list to initialize array */
    int a[SIZE] = { 1, 3, 5, 4, 7, 2, 99, 16, 45, 67, 89, 45 };
    int i ; /* counter */
    int total = 0 ; /* sum of array */

    /* sum contents of array a */
    for ( i = 0 ; i < SIZE ; i++ ) {
        total += a[i] ;
    } /* end for */

    printf ( "Total of array element values is %d\n", total ) ;

    return 0 ; /* indicates successful termination */

} /* end main */
```

```
Total of array element values is 383
```

ตัวอย่างโปรแกรมที่ 25 เป็นโปรแกรมที่นำข้อมูลในตัวแปรแถวลำดับมาแสดงเป็นแผนภาพ  
ฮิสโตแกรม

```

/* Histogram printing program */
#include <stdio.h>
#define SIZE 10

/* function main begins program execution */
int main ()
{
    /* use initializer list to initialize array */
    int n[SIZE] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
    int i; /* outer for counter for array elements */
    int j; /* inner for counter counts *s in each histogram bar */

    printf( "%s%13s%17s\n", "Element", "Value", "Histogram" );

    /* for each element of array n, output a bar of the histogram */
    for ( i = 0 ; i < SIZE ; i++ ) {
        printf( "%7d%13d", i, n[i] );

        for ( j = 1 ; j <= n[i] ; j++ ) { /* print one bar */
            printf( "%c", '*' );
        } /* end inner for */

        printf( "\n" ); /* end a histogram bar */
    } /* end outer for */

    return 0 ; /* indicate successful termination */

} /* end main */

```

Element	Value	Histogram
0	19	*****
1	3	***
2	15	*****
3	7	*****
4	11	*****
5	9	*****
6	13	*****
7	5	*****
8	17	*****
9	1	*

**ตัวอย่างโปรแกรมที่ 26** เป็นโปรแกรมที่มีการระบุให้ตัวแปรแถวลำดับ days ในฟังก์ชัน main เป็นแบบ extern

```
#include <stdio.h>
#define MONTHS 12
int days[MONTHS] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
int main (void)
{
    int index ;
    extern int days[ ] ; /* optional declaration */

    for (index = 0 ; index < MONTHS ; index++)
        printf ( "Month %d has %d days. /n", index +1, days[index] ) ;
    return 0 ;
}
```

```
Month 1 has 31 days.
Month 2 has 28 days.
Month 3 has 31 days.
Month 4 has 30 days.
Month 5 has 31 days.
Month 6 has 30 days.
Month 7 has 31 days.
Month 8 has 31 days.
Month 9 has 30 days.
Month 10 has 31 days.
Month 11 has 30 days.
Month 12 has 31 days.
```

**ตัวอย่างโปรแกรมที่ 27** เป็นโปรแกรมที่มีลักษณะคล้ายกับตัวอย่างโปรแกรม 26 โดยมีการกำหนด ค่าเริ่มต้นให้กับตัวแปรแถวลำดับ เพียง 10 ตัวแรก ส่วนที่เหลือจะถูกกำหนดให้เป็น 0

```
#include <stdio.h>
#define MONTHS 12
int days[MONTHS] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31 };
int main (void)
{
    int index ;
    extern int days[ ] ; /* optional declaration */

    for (index = 0 ; index < MONTHS ; index++)
        printf ( "Month %d has %d days. \n", index +1, days[index] ) ;
    return 0 ;
}
```

```
Month 1 has 31 days.
Month 2 has 28 days.
Month 3 has 31 days.
Month 4 has 30 days.
Month 5 has 31 days.
Month 6 has 30 days.
Month 7 has 31 days.
Month 8 has 31 days.
Month 9 has 30 days.
Month 10 has 31 days.
Month 11 has 0 days.
Month 12 has 0 days.
```

**ตัวอย่างโปรแกรมที่ 28** เป็นโปรแกรมที่มีลักษณะคล้ายกับตัวอย่างโปรแกรมที่ 27 โดยไม่ได้ มีการกำหนดจำนวนในตัวแปรแถวลำดับ days ภายในฟังก์ชัน main

```
#include <stdio.h>
int days[ ] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31 };
int main (void)
{
    int index ;
    extern int days[ ] ; /* optional declaration */

    for (index = 0 ; index < sizeof days / sizeof (int) ; index++)
        printf ( "Month %d has %d days. \n", index +1, days[index] );
    return 0 ;
}
```

```
Month 1 has 31 days.
Month 2 has 28 days.
Month 3 has 31 days.
Month 4 has 30 days.
Month 5 has 31 days.
Month 6 has 30 days.
Month 7 has 31 days.
Month 8 has 31 days.
Month 9 has 30 days.
Month 10 has 31 days.
```

ในโปรแกรมนี้ มีการใช้ตัวดำเนินการ sizeof หมายถึง การหาขนาดของตัวแปร เช่น sizeof days จะหมายถึง หาค่าขนาดของตัวแปรแถวลำดับ days มีขนาด 20 ไบต์ เนื่องจาก days เป็นตัวแปรแถวลำดับ มีข้อมูล 10 ตัว แต่ละตัวกินเนื้อที่ 2 ไบต์ เพราะว่าเป็นข้อมูลชนิด int ส่วน sizeof (int) หมายถึง ขนาดของข้อมูลชนิด int จะกินเนื้อที่ 2 ไบต์

ดังนั้น ในโปรแกรมนี้จะแสดงข้อมูลทั้งหมดเท่ากับจำนวนเนื้อที่ของตัวแปรแถวลำดับ days หารด้วยขนาดของข้อมูลชนิด int

นั่นคือ  $20 \div 2 = 10$  ตัว

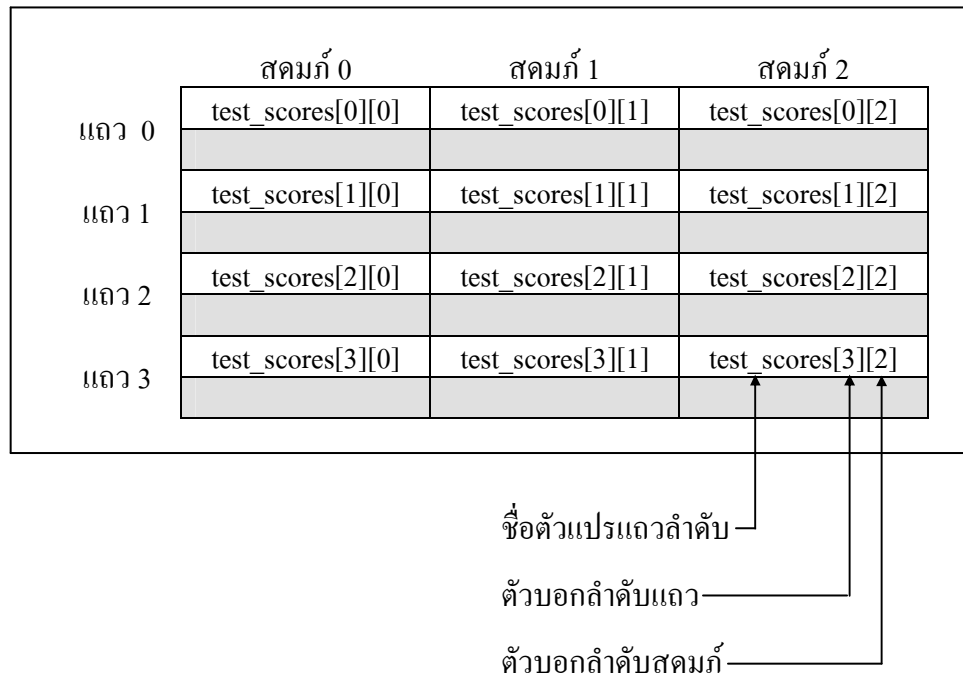
#### 4. รูปแบบทั่วไปของการประกาศตัวแปรแวลค่าดับ 2 มิติ

```
storage-class data-type array[expression 1][expression 2];
```

โดย	storage-class	คือ ประเภทหน่วยเก็บ เช่น static, auto เป็นต้น ซึ่งอาจจะมีหรือไม่มีก็ได้  ถ้านิยามภายในฟังก์ชันใด ๆ จะมีประเภทหน่วยเก็บเป็นแบบอัตโนมัติ และถ้านิยามภายนอกฟังก์ชันใด ๆ จะมีประเภทหน่วยเก็บเป็นแบบภายนอก ถ้าภายในแต่ละฟังก์ชันไม่ได้กำหนด จะมี default เป็นแบบอัตโนมัติ
	data-type	คือ ชนิดของข้อมูล เช่น int, char เป็นต้น
	array	คือ ชื่อของตัวแปรแวลค่าดับที่ตั้งตามกฎเกณฑ์การตั้งชื่อ
	expression 1	คือ ค่าจำนวนเต็มบวก ซึ่งอาจจะเป็นค่าคงตัวหรือชื่อคงตัว เพื่อบ่งบอกจำนวนของตัวแปรแวลค่าดับ
	expression 2	คือ ค่าจำนวนเต็มบวก ซึ่งอาจจะเป็นค่าคงตัวหรือชื่อคงตัว เพื่อบ่งบอกจำนวนสดมภ์ของตัวแปรแวลค่าดับ

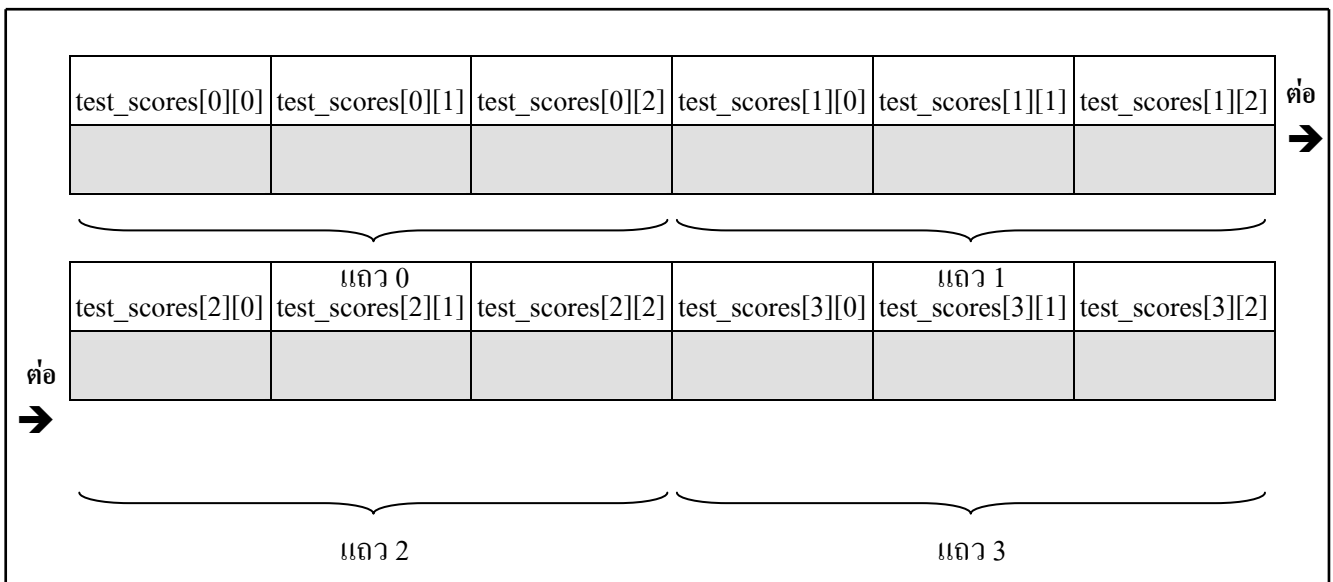
**ตัวอย่างที่ 29** กำหนด `int test_scores[4][3];`

หมายความว่า จะเป็นการประกาศเพื่อบอกให้คอมพิวเตอร์เตรียมเนื้อที่ให้ตัวแปรแวลค่าดับ `test_scores` ให้มีเนื้อที่ต่อเนื่องกันไปจำนวน  $4 \times 3 = 12$  ตัว โดยแต่ละตัวจะมีข้อมูลเป็นแบบ `int` ซึ่งแสดงได้ดังรูปที่ 6



รูปที่ 6

และจากรูป 6 จะถูกเก็บในหน่วยความจำแบบอันดับโรว์เมเจอร์ (row-major order) กล่าวคือ จะจัดเก็บข้อมูลเรียงติดต่อกันไปที่ละแถวในหน่วยความจำ ดังรูปที่ 7



รูปที่ 7



**ตัวอย่างที่ 30** กำหนด `int test_scores[4][3] = { {95, 80, 78}, {69, 75, 81},  
 {100, 98, 100}, {98, 85, 87} } ;`

จะหมายถึง การกำหนดค่าเริ่มต้นให้กับตัวแปรแถวลำดับ `test_scores` และแสดงได้ดัง

รูปที่ 8

	สดมภ์ 0	สดมภ์ 1	สดมภ์ 2
แถว 0	test_scores[0][0] 95	Test_scores[0][1] 80	test_scores[0][2] 78
แถว 1	test_scores[1][0] 69	Test_scores[1][1] 75	test_scores[1][2] 81
แถว 2	test_scores[2][0] 100	Test_scores[2][1] 98	test_scores[2][2] 100
แถว 3	test_scores[3][0] 98	Test_scores[3][1] 85	test_scores[3][2] 87

คำของตัวแปร  
ชื่อที่ใช้ในการอ้างถึง

รูปที่ 8

โดย `test_scores[1][1] = 75` และ `test_scores[3][2] = 87`

**ตัวอย่างที่ 31** กำหนด `int test_scores[4][3] = { 95, 80, 78, 69, 75, 81,  
 100, 98, 100, 98, 85, 87 } ;`

จะมีลักษณะการกำหนดค่าเริ่มต้นให้กับตัวแปรแถวลำดับ `test_scores` เหมือนกับรูปที่ 8

ในตัวอย่าง 30

**ตัวอย่างที่ 32** กำหนด `int test_scores[ ][3] = { {95, 80, 78}, {69, 75, 81},  
 {100, 98, 100}, {98, 85, 87} } ;`

ในตัวอย่างนี้ ไม่ได้กำหนดจำนวนแถว แต่กำหนดจำนวนสดมภ์ จำนวน 3 สดมภ์ ซึ่งในภาษา C จะยินยอม และจะมีการเก็บข้อมูลเหมือนกับรูปที่ 8 ในตัวอย่างที่ 30

**ตัวอย่างที่ 33** กำหนด `int test_scores[4][ ] = { {95, 80, 78}, {69, 75, 81},  
{100, 98, 100}, {98, 85, 87} } ;`

ในตัวอย่างนี้ ไม่ได้กำหนดจำนวนสมาชิก ซึ่งในภาษา C จะไม่ยินยอม ซึ่งจะทำให้แปลโปรแกรมไม่ได้

**ตัวอย่างที่ 34** กำหนด `int test_scores[ ][ ] = { {95, 80, 78}, {69, 75, 81},  
{100, 98, 100}, {98, 85, 87} } ;`

ในตัวอย่างนี้ ไม่ได้กำหนดจำนวนแถวและจำนวนสมาชิก ซึ่งในภาษา C จะถือว่าผิดไวยากรณ์ (syntax error)

**ตัวอย่างที่ 35** กำหนด `int test_scores[4][3] = {95, 80, 78, 69, 75, 81,  
100, 98, 100, 98, 85, 87, 65} ;`

ในตัวอย่างนี้ มีการกำหนดจำนวนสมาชิกให้มีค่าเริ่มต้นมากกว่าจำนวนสมาชิกของตัวแปรแถวลำดับ กล่าวคือ มีจำนวนสมาชิกในตัวแปรแถวลำดับ จำนวน 12 ตัว แต่มีการกำหนดค่าเริ่มต้นมาให้ จำนวน 13 ตัว ในกรณีนี้ในภาษา C จะไม่ยินยอม

**ตัวอย่างที่ 36** กำหนด `int test_scores[4][3] = {95} ;`

ในตัวอย่างนี้ มีการกำหนดค่าเริ่มต้นจำนวน 1 ตัว น้อยกว่าจำนวนของตัวแปรแถวลำดับทั้งหมด 12 ตัว ในกรณีนี้จะทำให้ `test_scores[0][0]` มีค่าเท่ากับ 95 และตัวแปรแถวลำดับที่เหลือจะมีค่าเท่ากับ 0 ทั้งหมดโดยอัตโนมัติและแสดงค่าเริ่มต้นได้ ดังรูปที่ 9

	สดมภ์ 0	สดมภ์ 1	สดมภ์ 2
แถว 0	test_scores[0][0]	test_scores[0][1]	test_scores[0][2]
	95	0	0
แถว 1	test_scores[1][0]	test_scores[1][1]	test_scores[1][2]
	0	0	0
แถว 2	test_scores[2][0]	test_scores[2][1]	test_scores[2][2]
	0	0	0
แถว 3	test_scores[3][0]	test_scores[3][1]	test_scores[3][2]
	0	0	0

รูปที่ 9

ตัวอย่างที่ 37 กำหนด `int test_scores[4][3] = {95, 80, 78, 69} ;`

ในตัวอย่างนี้ จะมีการกำหนดค่าเริ่มต้นให้กับตัวแปรแถวลำดับ 4 ตัวแรกเป็น 95, 80, 78 และ 69 ส่วนที่เหลือจะถูกกำหนดเป็น 0 โดยอัตโนมัติและแสดงค่าเริ่มต้น ได้ดังรูปที่ 10

	สดมภ์ 0	สดมภ์ 1	สดมภ์ 2
แถว 0	test_scores[0][0]	test_scores[0][1]	test_scores[0][2]
	95	80	78
แถว 1	test_scores[1][0]	test_scores[1][1]	test_scores[1][2]
	69	0	0
แถว 2	test_scores[2][0]	test_scores[2][1]	test_scores[2][2]
	0	0	0
แถว 3	test_scores[3][0]	test_scores[3][1]	test_scores[3][2]
	0	0	0

รูปที่ 10

**ตัวอย่างที่ 38** กำหนด `int test_scores[4][3] = { {95}, {69}, {100}, {98} } ;`

ในตัวอย่างนี้ จะมีการกำหนดค่าเริ่มต้นให้กับตัวแปรแถวลำดับ ดังรูปที่ 11

	สดมภ์ 0	สดมภ์ 1	สดมภ์ 2
แถว 0	test_scores[0][0] 95	test_scores[0][1] 0	test_scores[0][2] 0
แถว 1	test_scores[1][0] 69	test_scores[1][1] 0	test_scores[1][2] 0
แถว 2	test_scores[2][0] 100	test_scores[2][1] 0	test_scores[2][2] 0
แถว 3	test_scores[3][0] 98	test_scores[3][1] 0	test_scores[3][2] 0

รูปที่ 11

**ตัวอย่างที่ 39** กำหนด `int test_scores[4][3] ;`

ถ้าเราต้องการกำหนดค่าเริ่มต้นให้กับตัวแปรแถวลำดับ `test_scores` ทุกตัวมีค่าเริ่มต้นเป็น 0 หมด เราสามารถเขียนส่วนของโปรแกรมที่เขียนโดยใช้ข้อความสั่ง `for` ได้ 2 แบบ ดังนี้

**แบบที่ 1** กระทำไปที่ละแถว แสดงได้ดังรูปที่ 12

```
for (i = 0 ; i < 4 ; i++)
    for (j = 0 ; j < 3 ; j++)
        test_scores[i][j] = 0 ;
    /* end for */
/* end for */
```

รูปที่ 12

**แบบที่ 2** กระทำไปที่ละสดมภ์ แสดงได้ดังรูปที่ 13

```
for (j = 0 ; j < 3 ; j++)
    for (i = 0 ; i < 4 ; i++)
        test_scores[i][j] = 0 ;
    /* end for */
/* end for */
```

รูปที่ 13

**ตัวอย่างที่ 40** ถ้าเราต้องการนำข้อมูลในตัวอย่าง 30 มาแสดงผลที่จอภาพ สามารถเขียนส่วนของโปรแกรมได้ดังนี้

```
int i, j

for (i = 0; i < 4; i++) {
    printf( "ROW %d OF test_scores : ", i );

    for (j = 0; j < 3; j++)
        printf( "%d ", test_scores[i][j] );
    /* end for */

    printf( "\n" );
} /* end for */
```

```
ROW 0 OF test_scores : 95  80  78
ROW 1 OF test_scores : 69  75  81
ROW 3 OF test_scores : 100 98  100
ROW 4 OF test_scores : 98  85  87
```

**ตัวอย่างที่ 41** กำหนด char colors[3][6] = {  
 {'R', 'E', 'D'},  
 {'G', 'R', 'E', 'E', 'N'},  
 {'B', 'L', 'U', 'E'}  
 };

จะมีการกำหนดค่าเริ่มต้นให้กับตัวแปรแถวลำดับ colors แสดงได้ดังรูปที่ 14

	สดมภ์ 0	สดมภ์ 1	สดมภ์ 2	สดมภ์ 3	สดมภ์ 4	สดมภ์ 5
แถว 0	colors[0][0]	colors[0][1]	colors[0][2]	colors[0][3]	colors[0][4]	colors[0][5]
	'R'	'E'	'D'	'\0'	'\0'	'\0'
แถว 1	colors[1][0]	colors[1][1]	colors[1][2]	colors[1][3]	colors[1][4]	colors[1][5]
	'G'	'R'	'E'	'E'	'N'	'\0'
แถว 2	colors[2][0]	colors[2][1]	colors[2][2]	colors[2][3]	colors[2][4]	colors[2][5]
	'B'	'L'	'U'	'E'	'\0'	'\0'

รูปที่ 14

**ตัวอย่างโปรแกรมที่ 42** เป็นโปรแกรมที่แสดงการกำหนดค่าเริ่มต้นให้กับตัวแปรแถวลำดับ 2 มิติ และนำค่าออกมาแสดงที่จอภาพ

```
#include <stdio.h>

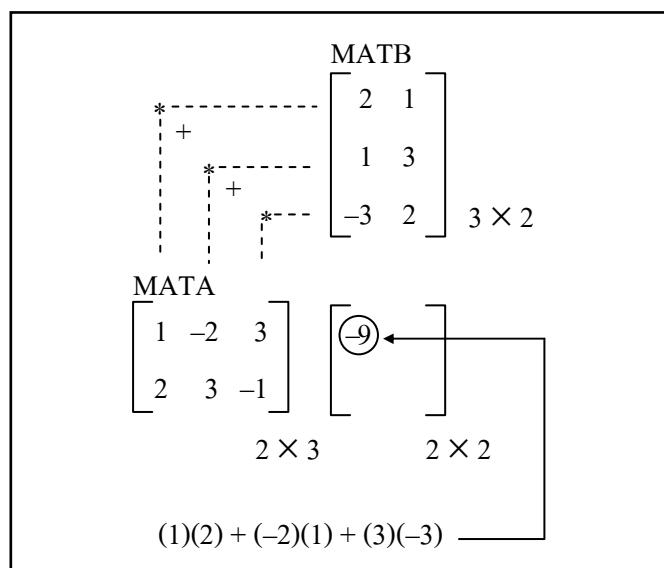
main ()
{
    static int array[2][3] = {
        {10, 20, 30},
        {11, 21, 31},
    };

    int row ;
    int column ;

    for (row = 0 ; row < 2 : row++)
    {
        for (column = 0 ; column < 3 ; column++)
            printf ( "%5d", array[row][column] ) ;
        printf ( "\n\n" ) ;
    }
}
```

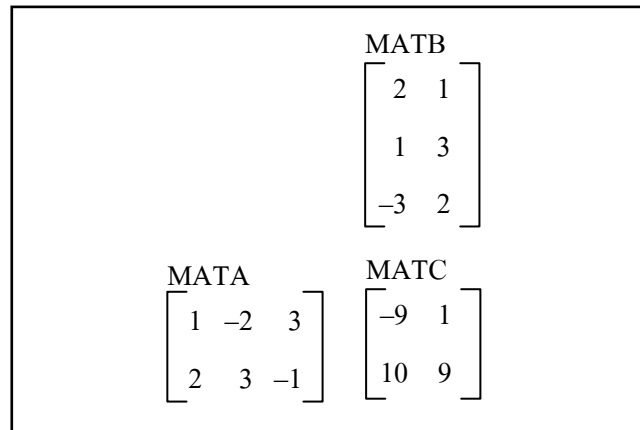
10	20	30
11	21	31

**ตัวอย่างโปรแกรมที่ 43** เป็นโปรแกรมที่แสดงการคำนวณหาผลคูณของเมทริกซ์ MATA ขนาด  $2 \times 3$  และเมทริกซ์ MATB ขนาด  $3 \times 2$  โดยสามารถแสดงการคูณได้ดังรูปที่ 15



รูปที่ 15

รูปที่ 15 เป็นการคำนวณหาสมาชิกแถว 0 และสดมภ์ 0 และผลลัพธ์สุดท้ายจะได้ เมทริกซ์ MATC ดังรูปที่ 16



รูปที่ 16

```
#include <stdio.h>

main ()
{
    int mata[2][3] = { { 1, -2, 3 },
                      { 2, 3, -1 } };
    int matb[3][2] = { { 2, 1 },
                      { 1, 3 },
                      { -3, 2 } };

    int matc[2][2];
    int r, c, v, psum ;

    for (r = 0 ; r < 2 ; r++)
        for (c = 0 ; c < 2 ; c++)
            {
                psum = 0 ;
                for (v = 0 ; v < 3 ; v++)
                    psum += mata[r][v] * matb[v][c] ;
                matc[r][c] = psum ;
            }
    printf ( "The array product is : \n" );
    for (r = 0 ; r < 2 ; r++)
        {
            for (c = 0 ; c < 2 ; c++)
                printf ( " %5d", matc[r][c] );
            printf ( "\n" );
        }
}
```

**ตัวอย่างโปรแกรมที่ 44** เป็นโปรแกรมแสดงการคำนวณหาจัตุรัสกอล (magic square) ซึ่งจัตุรัสกอล จะเป็นเมทริกซ์จัตุรัส ที่มีผลบวกในแต่ละแถว, แต่ละสดมภ์ และตามแนวทแยงมุมหลัก เท่ากันหมด (ในกรณีนี้ให้ป้อนขนาดของเมทริกซ์เป็นเลขคี่)

```
#include <stdio.h>

main ()
{
    static int magic[9][9];
    int row, col, k, n, x, y;

    printf ( "Enter size of magic square => " );
    scanf ( "%d", &n );
    if ( 0 == n % 2 )
    {
        printf ( "Sorry, you must enter an odd number." );
        exit ( 0 );
    }
    k = 2;
    row = 0;
    col = ( n - 1 ) / 2;
    magic[row][col] = 1;
    while ( k <= n * n )
    {
        x = ( row - 1 < 0 ) ? n - 1 : row - 1;
        y = ( col - 1 < 0 ) ? n - 1 : col - 1;
        if ( magic[x][y] != 0 )
        {
            x = ( row + 1 < n ) ? row + 1 : 0;
            y = col;
        }
        magic[x][y] = k;
        row = x;
        col = y;
        k++;
    }
    for ( row = 0; row < n; row++ )
    {
        for ( col = 0; col < n; col++ )
            printf ( "\t%d", magic[row][col] );
        printf ( "\n" );
    }
}
```

Enter size of magic square => 3

6	1	8
7	5	3
2	9	4

Enter size of magic square => 4

Sorry, you must enter and odd number.

Enter size of magic square => 5

15	8	1	24	17
16	14	7	5	23
22	20	13	6	4
3	21	19	12	10
9	2	25	18	11



## 5. การใช้ตัวแปรแถวลำดับผ่านค่าไปให้ฟังก์ชัน

ในภาษา C การผ่านค่าตัวแปรแถวลำดับ ซึ่งเป็น actual parameter ไปยัง formal parameter นั้น ให้ actual parameter เป็นชื่อของตัวแปรแถวลำดับเท่านั้น โดยไม่ต้องมีเครื่องหมาย [ ] ตาม และในส่วนของ formal parameter มีแต่เครื่องหมาย [ ] โดยไม่ต้องระบุจำนวนตัว ซึ่งการผ่านค่าแบบนี้เราก็จะเรียกว่า การผ่านค่าโดยการอ้างอิง (call by reference)

ตัวอย่างโปรแกรมที่ 45 เป็นโปรแกรมแสดงการผ่านค่าตัวแปรแถวลำดับไปให้ฟังก์ชัน

```
#include <stdio.h>
void function (int this [ ] );

main ()
{
    int array[3] ;

    array[0] = 10 ;
    array[1] = 20 ;
    array[2] = 30 ;
    function1(array) ;
}

void function1 (int this[ ])
{
    printf ( "contents of array[0] => %d\n", this[0] ) ;
    printf ( "contents of array[1] => %d\n", this[1] ) ;
    printf ( "contents of array[2] => %d\n", this[2] ) ;
}

contents of array[0] => 10
contents of array[1] => 20
contents of array[2] => 30
```

ไม่ต้องระบุชื่อจำนวนสมาชิก

เป็นชื่อของตัวแปรแถวลำดับเท่านั้น ไม่ต้องมีเครื่องหมาย [ ]

เป็นชื่อของตัวแปรแถวลำดับ และมีเครื่องหมาย [ ] ด้วย

ในโปรแกรมนี้ มีการประกาศฟังก์ชันต้นแบบ คือ void function1 (int this [ ] ) ; โดยมี formal argument คือ this[ ] โดยกรณีนี้ไม่ต้องมีตัวดำเนินการโดยอ้อม ( \* ) มาใช้ เนื่องจาก this[ ] เป็นตัวชี้ (pointer) ที่ชี้ไปยังสมาชิกลำดับแรกของตัวแปรแถวลำดับ ทำให้ตัวแปรแถวลำดับทั้งสองชี้ไปยังที่อยู่เดียวกัน ในฟังก์ชัน main () มีการกำหนดชนิดข้อมูลและกำหนดค่าของตัวแปรแถวลำดับ ดังนี้

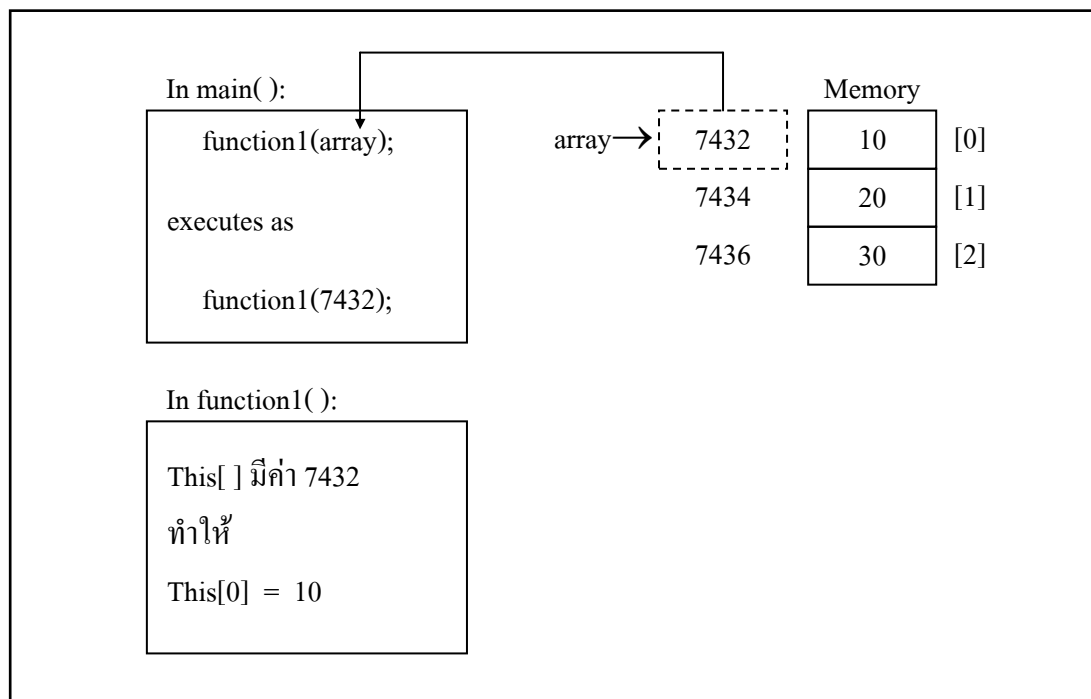
```
int array[3];

array[0] = 10;
array[1] = 20;
array[2] = 30;
```

หลังจากนั้นมีการเรียกใช้ฟังก์ชัน `function1(array)`; โดยมี `array` เป็น actual argument โดยไม่ต้องมีเครื่องหมาย `&` นำหน้า เนื่องจากตัวแปรแถวลำดับ `array` จะเก็บที่อยู่ของตัวแปรแถวลำดับ หรืออาจจะใช้ `&array[0]` เป็น actual argument ก็ได้ เมื่อฟังก์ชัน `function1` ได้รับค่าที่อยู่ของสมาชิกของตัวแรกของตัวแปรแถวลำดับ ก็จะแสดงค่าของสมาชิก 3 ตัว โดยใช้

```
printf("contents of array[0] => %d\n", this[0]);
printf("contents of array[1] => %d\n", this[1]);
printf("contents of array[2] => %d\n", this[2]);
```

และสามารถแสดงการผ่านค่าที่อยู่ไปยังฟังก์ชันถูกเรียกได้ดังรูปที่ 17



รูปที่ 17

**หมายเหตุ** ตัวเลข 7432 เป็นตัวเลขที่สมมติขึ้นมาแทนที่อยู่ลำดับแรกของตัวแปรแถวลำดับ `array` เนื่องจากตัวแปรแถวลำดับ `array` มีชนิดข้อมูลเป็น `int` ข้อมูลแต่ละตัวจะกินเนื้อที่ 2 ไบต์ ดังนั้นที่อยู่ของตัวแปรแถวลำดับถัดไป คือ  $7432 + 2 = 7434$

**ตัวอย่างโปรแกรมที่ 46** เป็นโปรแกรมที่แสดงการผ่านค่าจากฟังก์ชัน ถูกเรียกกลับมายังฟังก์ชัน ที่เรียกใช้

```
#include <stdio.h>

void function1 (int this[ ]);

main ()
{
    int array[3];

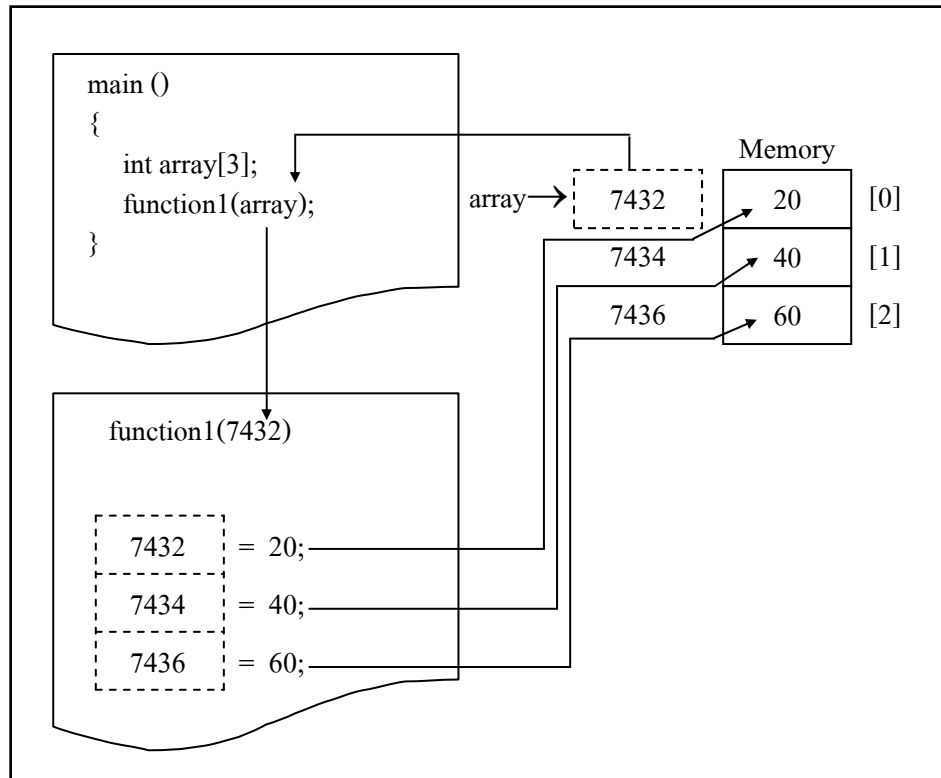
    function1 (array);

    printf ( "contents of array[0] => %d\n", array[0] );
    printf ( "contents of array[1] => %d\n", array[1] );
    printf ( "contents of array[2] => %d\n", array[2] );
}

void function1 (int this[ ])
{
    this[0] = 20;
    this[1] = 40;
    this[2] = 60;
}
```

```
contents of array[0] => 20
contents of array[1] => 40
contents of array[2] => 60
```

ในโปรแกรมนี ในฟังก์ชัน main () ไม่มีการกำหนดค่าเริ่มต้นให้กับตัวแปรแถวลำดับ array เมื่อมีการเรียกใช้ฟังก์ชัน function1 () ก็จะมีการส่งค่าที่อยู่ของตัวแปรแถวลำดับ array ไปยังฟังก์ชันที่ถูกเรียก และภายในฟังก์ชันที่ถูกเรียกมีการกำหนดค่าให้กับตัวแปรแถวลำดับแต่ละตัว เมื่อสิ้นสุดฟังก์ชันที่ถูกเรียก ก็จะทำให้ตัวแปรแถวลำดับ array มีค่าที่ถูกส่งกลับมาให้ เนื่องจากตัวแปร array กับตัวแปร this ซึ่งไปอยู่ที่เดียวกันในตอนที่มีการเรียกใช้ฟังก์ชัน เมื่อสิ้นสุดฟังก์ชัน function1 () ตัวแปรแถวลำดับ this จะถูกทำลายไป และสามารถแสดงการผ่านค่ากลับจากฟังก์ชันที่ถูกเรียกไปยังฟังก์ชันเรียกใช้ได้ดังรูปที่ 18



รูปที่ 18

**ตัวอย่างโปรแกรมที่ 47** เป็นโปรแกรมที่ให้ผู้ใช้ป้อนข้อมูล จำนวน 9 ตัว หลังจากนั้นก็เรียกใช้ฟังก์ชันเพื่อพิมพ์ค่าย้อนกลับ

```
#include <stdio.h>

#define maxnumber 9 /* Maximum number of array elements. */

void run_backwards (int user_array[ ]);

main ()
{
    int number[maxnumber];
    int index;

    printf ( "Give me nine and I'll print them backwards. \n" );

    for (index = 0; index < maxnumber; index++)
    {
        printf ( "Number[%d] = ", index );
        scanf ( "%d", &number[index] );
    }

    printf ( "Thank you... \n" );

    run_backwards (number);
}

void run_backwards (int user_array[ ])
{
    int index;

    printf ( "\n\nHere are the numbers you entered displayed\n" );
    printf ( "in the reverse order of entry : \n" );

    for (index = maxnumber - 1; index >= 0; index --)
        printf ( "number[%d] = %d\n", index, user_array[index] );
}
```

Give me nine numbers and I'll print them backwards.

```
Number[0] = 1
Number[1] = 2
Number[2] = 3
Number[3] = 4
Number[4] = 5
Number[5] = 6
Number[6] = 7
Number[7] = 8
Number[8] = 9
Thank you...
```

Here are the numbers you entered displayed  
in the reverse order of entry:

```
Number[8] = 9
Number[7] = 8
Number[6] = 7
Number[5] = 6
Number[4] = 5
Number[3] = 4
Number[2] = 3
Number[1] = 2
Number[0] = 1
```

**ตัวอย่างโปรแกรมที่ 48** เป็นโปรแกรมที่ให้ผู้ใช้ป้อนตัวเลขจำนวน 9 ตัว หลังจากนั้นเรียกใช้ฟังก์ชันเพื่อให้อาจหาจำนวนน้อยที่สุด

```
#include <stdio.h>

#define maxnumber 9 /* Maximum number of array elements. */

int minimum_value (int user_array[ ]);

main ()
{
    int number[maxnumber];
    int index;

    printf ( "Give me nine numbers and I'll find the minimum value : \n" );
    for (index = 0; index < maxnumber; index++)
    {
        printf ( "Number[%d] = ", index );
        scanf ( "%d", &number[index] );
    }
    printf ( "Thank you... \n" );

    printf ( "The minimum value is %d\n", minimum_value(number) );
}

int minimum_value(int user_array[ ])
{
    register int index;
    int minimum;

    minimum = user_array[0];
    for (index = 0; index < maxnumber; index++)
        if (user_array[index] < minimum)
            minimum = user_array[index];
    return (minimum);
}
```

```
Give me nine numbers and I'll find the minimum value :
Number[0] = 12
Number[1] = 21
Number[2] = 58
Number[3] = 3
Number[4] = 5
Number[5] = 8
Number[6] = 19
Number[7] = 91
Number[8] = 105
Thank you...
The minimum value is 3
```

**ตัวอย่างโปรแกรมที่ 49** เป็นโปรแกรมที่นำค่าในตัวแปรแถวลำดับ array ขนาด  $2 \times 3$  ที่ถูกกำหนดไว้ในฟังก์ชัน main () แล้วเรียกฟังก์ชันมาคำนวณหาผลรวมในแต่ละแถว และในแต่ละสดมภ์

```
#include <stdio.h>

int add_columns (int arrayin[ ][3], int column_value[ ] );
int add_rows (int array[ ][3], int row_value[ ] );

main ()
{
    static int array[2][3] = {
        {10, 20, 30},
        {11, 21, 31}
    };

    int row ;
    int column ;
    int column_value[3] ;
    int row_value[2] ;

    for (row = 0 ; row < 2 ; row++)
    {
        for (column = 0 ; column < 3 ; column++)
            printf ( "%5d", array[row][column] ) ;
        printf ( "\n\n" ) ;

        add_columns(array, column_value) ;

        for (column = 0 ; column < 3 ; column++)
            printf ( "The sum of column %d is %d\n",
                column, column_value[column] ) ;

        add_row(array, row_value) ;

        for (row = 0 ; row < 2 ; row++)
            printf ( "The sum of row %d is %d\n", row, row_value[row] ) ;
    }

    int add_columns (int arrayin[ ][3], int column_value[ ] )
    {
        int row ;
        int column ;

        for (column = 0 ; column < 3 ; column++)
        {
            column_value[column] = 0 ;
            for (row = 0 ; row < 2 ; row++)
                column_value[column] += arrayin[row][column] ;
        }
    }

    int add_rows (int arrayin[ ][3], int row_value[ ] )
    {
```

```
int row ;
int column ;

for (row = 0 ; row < 2 ; row++)
{
    row_value[row] = 0 ;

    for (column = 0 ; column < 3 ; column++)
        row_value[row] += arrayin[row][column] ;
    }
}
```

10	20	30
11	21	31

The sum of column 0 is 21  
The sum of column 1 is 41  
The sum of column 2 is 61  
The sum of row 0 is 60  
The sum of row 1 is 63



**ตัวอย่างโปรแกรมที่ 50** เป็นโปรแกรมที่มีการกำหนดค่าเริ่มต้นให้กับตัวแปรแอมป์ 2 มิติ แล้วเรียกใช้ฟังก์ชันเพื่อนำค่าออกมาพิมพ์

```

/* Initializing multidimensional arrays */
#include <stdio.h>

void printArray( const int a[ ][3] ); /* function prototype */

/* function main begins program execution */
int main ()
{
    /* initialize array1, array2, array3 */
    int array1[2][3] = { {1, 2, 3}, {4, 5, 6} };
    int array2[2][3] = {1, 2, 3, 4, 5};
    int array3[2][3] = { {1, 2}, {4} };

    printf( "Values in array1 by row are : \n" );
    printArray( array1 );

    printf( "Values in array2 by row are : \n" );
    printArray( array2 );

    printf( "Values in array3 by row are : \n" );
    printArray( array3 );

    return 0 ; /* indicates successful termination */
} /* end main */

/* function to output array with two rows and three columns */
void printArray( const int a[ ][3] )
{
    int i ; /* row counter */
    int j ; /* column counter */

    /* loop through rows */
    for ( i = 0 ; i <= 1 ; i++ ) {

        /* output column values */
        for ( j = 0 ; j <= 2 ; j++ ) {
            printf( "%d", a[i][j] );
        } /* end inner for */

        printf( "\n" ); /* start new line of output */
    } /* end outer for */
} /* end function printArray */

```

```

Values in array1 by row are :
1 2 3
4 5 6
Values in array2 by row are :
1 2 3
4 5 0
Values in array3 by row are :
1 2 0
4 0 0

```

## ตัวอย่างโปรแกรมที่ 51 เป็นโปรแกรมพิมพ์ตารางรหัส ASCII

```

#include <stdio.h>

int main (void) {
    char ascii[13][10];
    int row, column ;

    for (row = 0 ; row < 13 ; row++)
        for (column = 0 ; column < 10 ; column++)
            ascii[row][column] = ' ' ;
        /* end for */
    /* end for */

    printf ( "ASCII Character Set \n" ) ;
    printf ( "      |" ) ;

    for (column = 0 ; column < 10 ; column++)
        printf ( "%5d", column ) ;
    /* end for */

    printf ( "\n-----|" ) ;

    for (column = 0 ; column < 10 ; column++)
        printf ( "-----" ) ;
    /* end for */

    for (row = 3 ; row < 13 ; row++)
        for (column = 0 ; column < 10 ; column++)
            if (row == 3 && column < 2)
                continue ;
            else
                ascii[row][column] = 10 * row + column ;
            /* end if */
        /* end for */
    /* end for */

    for (row = 3 ; row < 13 ; row++ ) {
        printf ( "\n%3d |", row ) ;

        for (column = 0 ; column < 10 ; column++)
            if (row == 12 && column > 6)
                break ;
            else
                printf ( "   %c", ascii[row][column] ) ;
            /* end if */
        /* end for */
    } /* end for */
    return 0 ;
} /* end function main */

```

ASCII Character Set										
	0	1	2	3	4	5	6	7	8	9
3				!	“	#	\$	%	&	'
4	(	)	*	+	,	-	.	/	0	1
5	2	3	4	5	6	7	8	9	:	;
6	<	=	>	?	@	A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z	[	\	]	^	_	'	a	b	c
10	d	e	f	g	h	i	j	k	l	m
11	n	o	p	q	r	s	t	u	v	w
12	x	y	z	{		}	~			

จากโปรแกรม มีการประกาศ `char ascii[13][10]` ; หมายความว่า `ascii` เป็นตัวแปรแถวลำดับ 2 มิติ ขนาด 13 แถวและ 10 สดมภ์ โดยข้อความสั่ง `for` แบบซ้อน จะมีการกำหนดค่าเริ่มต้นให้กับตัวแปรแถวลำดับ `ascii` เป็นช่องว่าง (space)

ข้อความสั่ง `for` แบบซ้อนถัดมา จะเป็นการสร้างตารางเก็บค่ารหัส ASCII ไว้ในตัวแปรแถวลำดับ `ascii` โดยอาศัยนิพจน์

$$\text{ascii}[\text{row}][\text{column}] = 10 * \text{row} + \text{column} ;$$

เช่น ถ้าค่าของ `row = 9` และ `column = 7` ค่าของ 97 จะเก็บไว้ในตัวแปรแถวลำดับ `ascii[9][7]` มีค่าเป็น 97

ถ้าเราต้องการรู้ค่าของรหัส ASCII ที่มีค่า 97 คืออะไร ให้ใช้ตัวแปรระบุการแปลงผัน `%c` เพื่อพิมพ์ข้อมูลชนิด `char` ก็จะทำให้รู้ว่าคือ ตัวอักษร `a`

## 6. การเรียงลำดับในแถวลำดับ (Sorting Arrays)

ในงานประยุกต์ในคอมพิวเตอร์ที่เกี่ยวข้องกับการจัดลำดับข้อมูลนั้นมีประโยชน์ช่วยในการค้นหาข้อมูลได้รวดเร็วยิ่งขึ้น การเรียงลำดับเป็นกระบวนการจัดเรียงข้อมูลจากน้อยไปมาก (ascending) หรือมากไปน้อย (descending)

การเรียงลำดับมีหลายวิธีด้วยกัน เช่น Bubble sort, Selection sort, Insertion sort เป็นต้น แต่ในส่วนนี้จะกล่าวถึงเฉพาะการเรียงลำดับแบบ Bubble sort เท่านั้น

### 6.1 การเรียงลำดับแบบ bubble

เป็นการเรียงลำดับข้อมูลโดยการนำข้อมูล 2 ตัว ที่อยู่ติดกันมาเปรียบเทียบกัน

สมมติมีข้อมูลทั้งหมด  $n$  ตัว

ถ้าต้องการเรียงข้อมูลจากน้อยไปมาก จะมีหลักดังนี้

ถ้าข้อมูลตัวแรกมากกว่าตัวที่สอง จะมีการสลับค่ากัน

มิฉะนั้นแล้ว ไม่ต้องสลับค่า โดยจะมีการกระทำทั้งหมด  $n - 1$  รอบ (pass) โดย

#### รอบที่ 1 (pass 1)

จะเปรียบเทียบข้อมูลตัวที่ 1 กับข้อมูลตัวที่ 2, ข้อมูลตัวที่ 2 กับข้อมูลตัวที่ 3, ... ,

ข้อมูลตัวที่  $n - 1$  กับ ข้อมูลตัวที่  $n$

เมื่อสิ้นสุดรอบที่ 1 จะได้ว่า ข้อมูลตัวที่  $n$  จะมีค่ามากที่สุด

#### รอบที่ 2 (pass 2)

จะเปรียบเทียบข้อมูลตัวที่ 1 กับข้อมูลตัวที่ 2, ข้อมูลตัวที่ 2 กับข้อมูลตัวที่ 3, ... ,

ข้อมูลตัวที่  $n - 2$  กับ ข้อมูลตัวที่  $n - 1$

เมื่อสิ้นสุดรอบที่ 2 จะได้ว่า ข้อมูลตัวที่  $n - 1$  จะมีค่ามากที่สุด

!                    !                    !

#### รอบที่ $n - 1$ (pass $n - 1$ )

จะเปรียบเทียบข้อมูลตัวที่ 1 กับข้อมูลตัวที่ 2

เมื่อสิ้นสุดรอบที่  $n - 1$  จะได้ว่า ข้อมูลตัวที่ 2 จะมีค่ามากที่สุด

เมื่อกระทำทั้งหมด  $n - 1$  รอบ จะได้ว่า

ข้อมูลทั้งหมด  $n$  ตัว จะถูกเรียงจากน้อยไปมาก

ตัวอย่างที่ 52 สมมติตัวแปรแถวลำดับ `array_to_sort` มีข้อมูล 5 ตัว ดังรูปที่ 19

ดัชนี  $\rightarrow$     0    1    2    3    4  
`array_to_sort`  $\rightarrow$ 

43	22	17	36	16
----	----	----	----	----

รูปที่ 19

ถ้าต้องการเรียงลำดับข้อมูลจากน้อยไปมาก โดยวิธี bubble จะมีการกระทำทั้งหมด 4 pass (เนื่องจากมีข้อมูล 5 ตัว) ดังรูปที่ 20

**การดำเนินการ (Operation)**

**array\_to\_sort**

---

**Pass 1:**

Step 1: 

[0]	[1]	[2]	[3]	[4]
43	22	17	36	16

 → เปรียบเทียบตัวที่ 1 กับตัวที่ 2 และสลับค่า

Step 2: 

22	43	17	36	16
----	----	----	----	----

 → เปรียบเทียบตัวที่ 2 กับตัวที่ 3 และสลับค่า

Step 3: 

22	17	43	36	16
----	----	----	----	----

 → เปรียบเทียบตัวที่ 3 กับตัวที่ 4 และสลับค่า

Step 4: 

22	17	36	43	16
----	----	----	----	----

 → เปรียบเทียบตัวที่ 4 กับตัวที่ 5 และสลับค่า

เมื่อสิ้นสุด Pass 1 จะได้ว่า ข้อมูลตัวที่ 5 มีค่ามากที่สุด

---

**Pass 2:**

Step 1: 

22	17	36	16	43
----	----	----	----	----

 → เปรียบเทียบตัวที่ 1 กับตัวที่ 2 และสลับค่า

Step 2: 

17	22	36	16	43
----	----	----	----	----

 → เปรียบเทียบตัวที่ 2 กับตัวที่ 3 ไม่ต้องสลับค่า

Step 3: 

17	22	36	16	43
----	----	----	----	----

 → เปรียบเทียบตัวที่ 3 กับตัวที่ 4 และสลับค่า

เมื่อสิ้นสุด Pass 2 จะได้ว่า ข้อมูลตัวที่ 4 มีค่ามากที่สุด

---

**Pass 3:**

Step 1: 

17	22	16	36	43
----	----	----	----	----

 → เปรียบเทียบตัวที่ 1 กับตัวที่ 2 ไม่ต้องสลับค่า

Step 2: 

17	22	16	36	43
----	----	----	----	----

 → เปรียบเทียบตัวที่ 2 กับตัวที่ 3 และสลับค่า

เมื่อสิ้นสุด Pass 3 จะได้ว่า ข้อมูลตัวที่ 3 มีค่ามากที่สุด

---

**Pass 4:**

Step 1: 

17	16	22	36	43
----	----	----	----	----

 → เปรียบเทียบตัวที่ 1 กับตัวที่ 2 และสลับค่า

เมื่อสิ้นสุด Pass 4 จะได้ว่า ข้อมูลตัวที่ 2 มีค่ามากที่สุด

---

**Sorted array:** เมื่อกระทำทั้งหมด 4 Pass จะได้ข้อมูลเรียงจากน้อยไปมาก ดังนี้

16	17	22	36	43
----	----	----	----	----

---

ตัวอย่างโปรแกรมที่ 53 เป็นโปรแกรมที่เรียงลำดับข้อมูลจากน้อยไปมาก โดยมีการกำหนดค่าเริ่มต้นไว้ในโปรแกรมทั้งหมด จำนวน 10 ตัว

```

/* This program sorts an array's values into ascending order */
#include <stdio.h>
#define SIZE 10

/* function main begins program execution */
int main ()
{
    /* initialize a */
    int a[SIZE] = {2, 6, 4, 8, 10, 12, 89, 68, 45, 37} ;
    int pass ; /* passes counter */
    int i;     /* comparisons counter */
    int hold ; /* temporary location used to swap array elements */

    printf ( "Data items in original order\n" );

    /* output original array */
    for ( i = 0 ; i < SIZE ; i++ ) {
        printf ( "%4d", a[i] );
    } /* end for */

    /* bubble sort */
    /* loop to control number of passes */
    for ( pass = 1 ; pass < SIZE ; pass++ ) {

        /* loop to control number of comparisons per pass */
        for ( i = 0 ; i < SIZE - 1 ; i++ ) {

            /* compare adjacent elements and swap them if first
            element is greater than second element */
            if ( a[i] > a[i + 1] ) {
                hold = a[i] ;
                a[i] = a[i + 1] ;
                a[i + 1] = hold ;
            } /* end if */

        } /* end inner for */

    } /*end outer for */

    printf ( "\nData items in ascending order\n" );

    /* output sorted array */
    for ( i = 0 ; i < SIZE ; i++ ) {
        printf ( "%4d", a[i] );
    } /* end for */

    printf ( "\n" );

    return 0 ; /* indicates successful termination */
}

```

Data items in original order	2	6	4	8	10	12	89	68	45	37
Data items in ascending order	2	4	6	8	10	12	37	45	68	89

จากโปรแกรมนี้ จะมีการกระทำทั้งหมด 9 pass โดยในแต่ละ pass นั้นจะมีการเปรียบเทียบดังนี้

ใน **pass 1** จะเปรียบเทียบ  $a[0]$  กับ  $a[1]$ ,  $a[1]$  กับ  $a[2]$ , ...,  $a[8]$  กับ  $a[9]$

เมื่อสิ้นสุด pass 1 จะได้ว่า  $a[9]$  จะมีค่ามากที่สุด

**pass 2** จะเปรียบเทียบ  $a[0]$  กับ  $a[1]$ ,  $a[1]$  กับ  $a[2]$ , ...,  $a[7]$  กับ  $a[8]$

เมื่อสิ้นสุด pass 2 จะได้ว่า  $a[8]$  จะมีค่ามากที่สุด

! ! !

**pass 9** จะเปรียบเทียบ  $a[0]$  กับ  $a[1]$

เมื่อสิ้นสุด pass 9 จะได้ว่าข้อมูลเรียงจากน้อยไปมาก

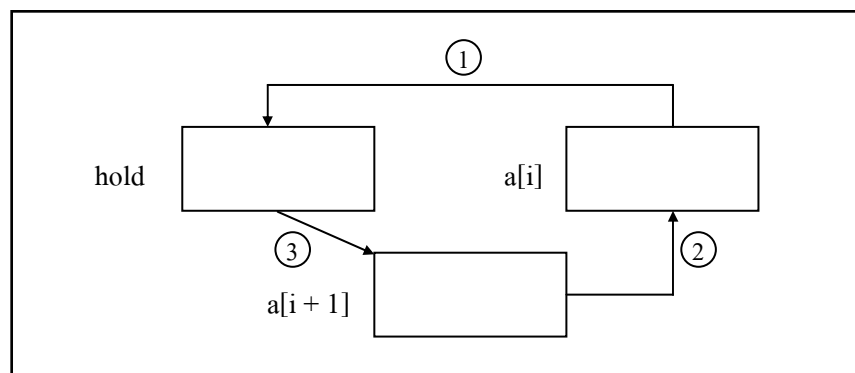
โดยในแต่ละ pass ถ้ามีการสลับค่าจะใช้ 3 ข้อความสั่ง ตามขั้นตอนดังนี้

$hold = a[i];$  ← ①

$a[i] = a[i + 1];$  ← ②

$a[i + 1] = hold;$  ← ③

และแสดงได้ดังรูปที่ 21



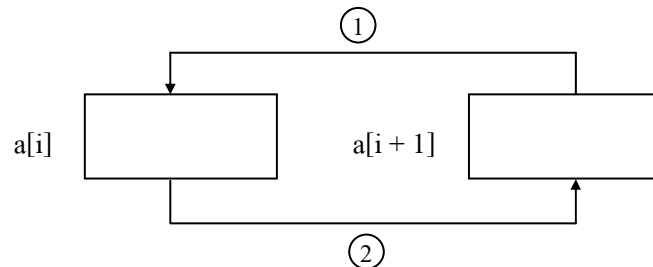
รูปที่ 21

โดย hold เป็นตัวแปรชั่วคราวเพื่อใช้ในการสลับค่า

ถ้าเราใช้ 2 ข้อความสั่งนี้ในการสลับค่า จะทำให้ข้อมูลสูญหายไป

$a[i] = a[i + 1];$  ← ①

$a[i + 1] = a[i];$  ← ②



เช่น ถ้า  $a[i] = 7$  ,  $a[i + 1] = 5$   
 จาก  $a[i] = a[i + 1]$  จะได้ว่า  $a[i] = 5$   
 และ  $a[i + 1] = a[i]$  จะได้ว่า  $a[i + 1] = 5$   
 ซึ่งจะทำให้ค่า 7 สูญหายไป

ดังนั้น ในการสลับค่าเราควรใช้ 3 ขั้นตอน ดังรูปที่ 21

**ตัวอย่างโปรแกรมที่ 54** เป็นโปรแกรมเรียงลำดับข้อมูลจากน้อยไปมาก โดยให้ผู้ใช้ป้อนข้อมูลเข้ามาทางแป้นพิมพ์ จำนวน 9 ตัว

```
#include <stdio.h>

#define maxnumber 9 /* Maximum number of array elements. */

int bubble_sort (int user_array[ ]);
void display_array (int sorted_array[ ]);

main ()
{
    int number[maxnumber];
    int index, compares;

    printf ( "Give me nine numbers and I'll sort them : \n" );
    for (index = 0; index < maxnumber; index++)
    {
        printf ( "Number[%d] = ", index );
        scanf ( "%d", &number[index] );
    }

    compares = bubble_sort (number);

    display_array (number);
    printf ( "\nThe number of comparisons is %d", compares );
}

int bubble_sort (int user_array[ ])
{
    int index;
    int switch_flag;
```



```

int temp_value ;
int valtest = 0 ;

do
{
    switch_flag = 0 ;

    for (index = 0 ; index < maxnumber ; index++)
    {
        valtest++ ;
        if (user_array[index] > user_array[index + 1])
            &&(index != maxnumber - 1) )
        {
            temp_value = user_array[index] ;
            user_array[index] = user_array[index + 1] ;
            user_array[index + 1] = temp_value ;
            switch_flag = 1 ;
        }
    }
} while (switch_flag) ;
return (valtest) ;
}

void display_array (int sorted_array[ ])
{
    int index ;

    printf ( "\n\nThe sorted values are : \n" ) ;

    for (index = 0 ; index < maxnumber ; index++)
        printf ( "Number[%d] = %d\n", index, sorted_array[index] ) ;
}

```

Give me nine numbers and I'll sort them :

```

Number[0] = 6
Number[1] = 7
Number[2] = 5
Number[3] = 8
Number[4] = 4
Number[5] = 9
Number[6] = 3
Number[7] = 0
Number[8] = 2

```

The sorted values are :

```

Number[0] = 0
Number[1] = 2
Number[2] = 3
Number[3] = 4
Number[4] = 5
Number[5] = 6
Number[6] = 7
Number[7] = 8
Number[8] = 9

```

The number of comparisons is 72

จากโปรแกรมนี้ จะมีการกำหนดให้ผู้ใช้ป้อนข้อมูลเข้ามาทั้งหมด 9 จำนวน หลังจากนั้นจะมีการเรียกใช้ฟังก์ชัน bubble\_sort () โดยมี actual parameter คือ ตัวแปรแถวลำดับ number และในฟังก์ชันถูกเรียกจะมี formal parameter คือ user\_array[ ] โดยจะมีการส่งที่อยู่ของ actual parameter ไปยัง formal parameter จึงทำให้ parameter ซ้ำไปยังที่อยู่เดียวกัน โดยในฟังก์ชัน bubble\_sort จะมีการประกาศใช้ตัวแปร 3 ตัว คือ index จะใช้เป็นตัวบอกลำดับของแต่ละตัวในแถวลำดับ

switch\_flag จะใช้บอกให้โปรแกรมรู้ว่า แถวลำดับมีการจัดเรียงลำดับเรียบร้อยแล้ว และ temp\_value จะใช้เป็นตัวแปรชั่วคราวที่ใช้สำหรับสลับค่า

**ตัวอย่างโปรแกรมที่ 55** เป็นโปรแกรมในการเรียงลำดับชื่อจากน้อยไปมาก

```
#include <stdio.h>
#include <string.h>
#define NUM 5

main ()
{
    char name[5][8] = { {'s', 'u', 'e'},
                        {'m', 'i', 'c', 'h', 'e', 'l', 'e'},
                        {'k', 'e', 'n', 'n', 'y'},
                        {'j', 'a', 'm', 'e', 's'},
                        {'k', 'e', 'n'} };

    char swapname[8];
    int j, k;

    printf ( "The original list is : \n" );
    for (j = 0 ; j < NUM ; j++)
        printf ( "%s\n", names[j] );
    for (j = 0 ; j < NUM - 1 ; j++)
        for (k = j + 1 ; k < NUM ; k++)
            if (strcmp(names[j], name[k]) > 0)
            {
                strcpy (swapname, names[j]);
                strcpy (names[j], names[k]);
                strcpy (names[k], swapname);
            }
    printf ( "\nThe sorted list is : \n" );
    for (j = 0 ; j < NUM ; j++)
        printf ( "%s\n", names[j] );
}
```

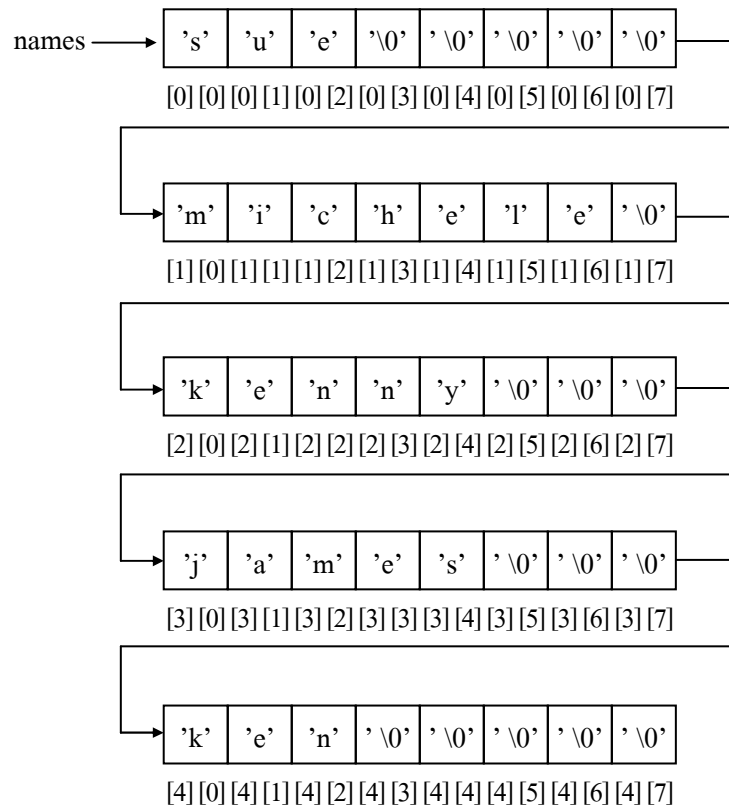
The original list is :

sue  
michele  
kenny  
james  
ken

The sorted list is :

james  
ken  
kenny  
michele  
sue

จากโปรแกรมนี้ names จะเป็นตัวแปรแถวลำดับ ชนิด char มีขนาด  $5 \times 8$  โดยข้อมูลจะถูกเก็บไปที่แถว เรียงกันไป ดังรูปที่ 22



รูปที่ 22

ในการเปรียบเทียบสายอักขระ เราจะใช้ฟังก์ชัน strcmp () ซึ่งจะทำการเปรียบเทียบตัวอักขระในสายอักขระแต่ละชุด โดยฟังก์ชันนี้จะส่งค่ากลับมาเป็นค่าจำนวนเต็ม กล่าวคือ

- 1) ถ้า ค่า = 0 สายอักขระทั้งสองเหมือนกัน
- 2) ถ้า ค่า > 0 สายอักขระแรกมาก่อนสายอักขระที่สอง
- 3) ถ้า ค่า < 0 สายอักขระที่สองมาก่อนสายอักขระที่หนึ่ง

## 7. การค้นหาในแถวลำดับ (searching arrays)

ถ้าเราต้องการค้นหาข้อมูลในแถวลำดับ เราจะต้องมีการกำหนดค่าไว้ใน search key โดย search key จะมีชนิดข้อมูลเหมือนกับชนิดข้อมูลในแถวลำดับ ถ้ากระบวนการค้นหาข้อมูลพบข้อมูลเหมือนค่าของ search key จะถือว่าประสบความสำเร็จ (successful) มิฉะนั้นแล้วถือว่าไม่ประสบความสำเร็จ (unsuccessful)

ในส่วนนี้ จะกล่าวถึงการค้นหา 2 แบบ คือ

7.1 การค้นหาแบบเรียงลำดับ (sequential search)

7.2 การค้นหาแบบไบนารี (binary search)

### 7.1 การค้นหาแบบเรียงลำดับ

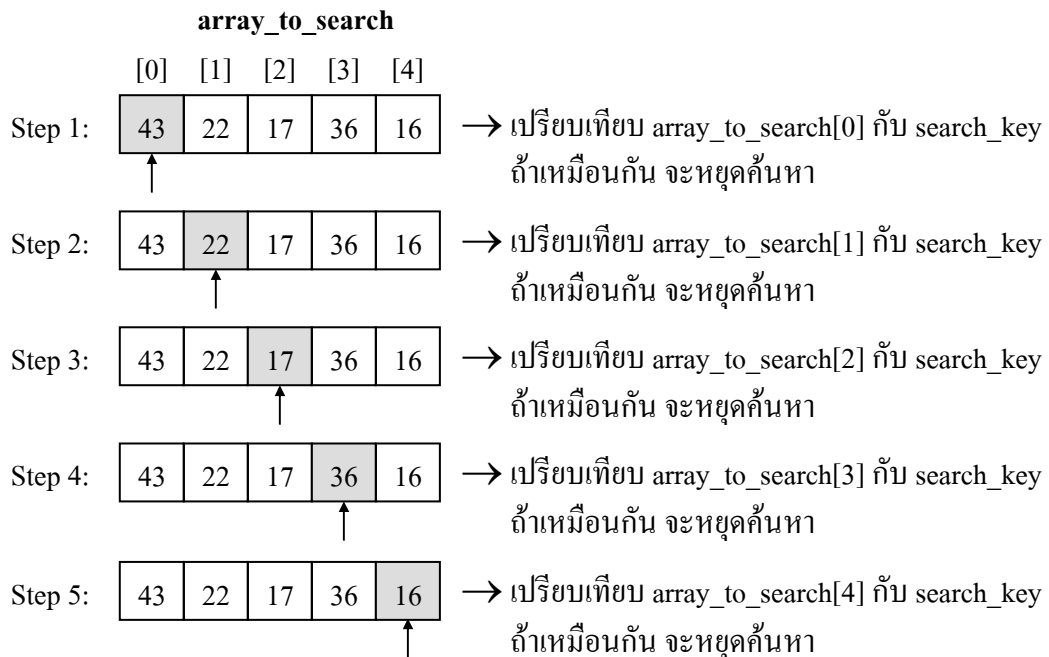
เป็นการค้นหาแบบง่ายที่สุด กล่าวคือ จะมีการค้นหาข้อมูลเริ่มจากข้อมูลตัวแรกเปรียบเทียบกับค่าของ search\_key ถ้ามีค่าเหมือนกัน จะถือว่า การค้นหาประสบความสำเร็จ และถ้าเปรียบเทียบกับกันไปเรื่อย ๆ จนถึงข้อมูลตัวสุดท้าย และมีค่าไม่เหมือนกันเลย จะถือว่าการค้นหาไม่ประสบความสำเร็จ

**ตัวอย่างที่ 56** กำหนดตัวแปรแถวลำดับ มีข้อมูลดังนี้

ดัชนี →	0	1	2	3	4
array_to_search →	43	22	17	36	16

ถ้าต้องการค้นหาแบบเรียงลำดับ โดยกำหนดค่า search\_key จะมีขั้นตอนดังรูปที่ 23

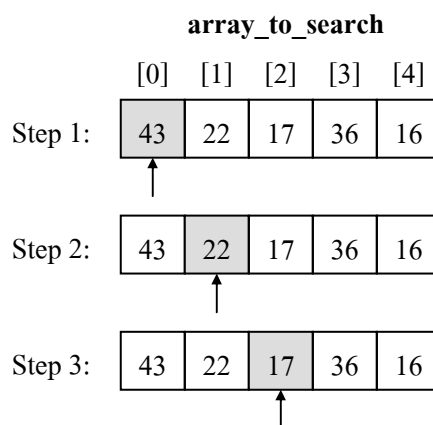
กำหนด  $search\_key = 25$



รูปที่ 23

จากรูปที่ 23 ในแต่ละขั้นตอนจะพบว่า ข้อมูลในแถวลำดับไม่ตรงกับค่า  $search\_key = 25$  เลย ดังนั้น เราจะกล่าวว่า ข้อมูลที่ต้องการค้นหาไม่อยู่ในแถวลำดับนี้

กำหนด  $search\_key = 17$



รูปที่ 24

จากรูปที่ 24 เราจะพบว่า ค่าของ  $array\_to\_search[2] = search\_key$  ในขั้นตอนที่ 3 ดังนั้น เราจะกล่าวว่า ข้อมูลที่ต้องการค้นหาอยู่ในแถวลำดับนี้

ตัวอย่างที่ 57 เราสามารถนำมาเขียนฟังก์ชันในการค้นหาแบบเรียงลำดับได้ดังรูปที่ 25

```

/*****
Function Name : sequential_search_unordered
Purpose       : Searches a one-dimensional unsorted array of integers using
                the sequential search algorithm.
Receives      : search_key, list_size, array_to_search
Returns       : success, index
*****/
void sequential_search_unordered (list_item_type search_key,
                                  int list_size,
                                  const list_item_type
                                  array_to_search[ ],
                                  char success[ ], int *index) {

    /* Local variables : */

    int i;

    /* Function body : */

    strcpy (success, "FALSE");
    *index = -1;

    for (i = 0 ; i < list_size ; i++)
        if (search_key == array_to_search[i]) {
            *index = i;
            strcpy (success, "TRUE");
            break ;
        } /* end if */
    /* end for */
} /* end function sequential_search_unordered */

```

รูปที่ 25 เป็นฟังก์ชันที่ใช้ในการค้นหาข้อมูลในแวลำดับที่ไม่มีการเรียงลำดับ

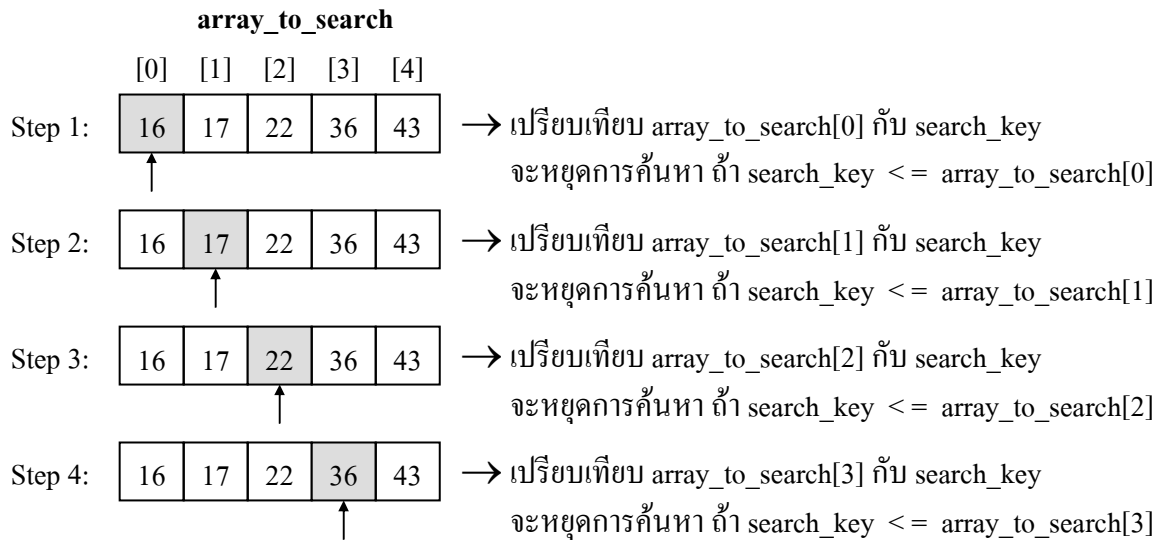
ถ้าเราต้องการให้การค้นหาข้อมูลในแวลำดับมีประสิทธิภาพขึ้น เราควรจะให้ข้อมูลในแวลำดับ มีการเรียงลำดับจากน้อยไปมากก่อน จะทำให้การค้นหามีประสิทธิภาพมากขึ้น ถ้าค่าของ search\_key มากกว่าข้อมูลในแวลำดับตัวที่ใช้เปรียบเทียบก็จะเปรียบเทียบกับข้อมูลในแวลำดับตัวถัดไป

ถ้าค่าของ search\_key เท่ากับข้อมูลในแวลำดับตัวที่ใช้เปรียบเทียบ แสดงว่า พบข้อมูลที่ต้องการ

และถ้าค่าของ search\_key น้อยกว่าข้อมูลในแวลำดับตัวที่ใช้เปรียบเทียบ แสดงว่า ไม่พบข้อมูลที่ต้องการค้นหา

**ตัวอย่างที่ 58** กำหนดข้อมูลในแถวลำดับมีการเรียงข้อมูลจากน้อยไปมาก ถ้าต้องการค้นหาแบบเรียงลำดับ โดยกำหนดค่า `search_key` จะมีขั้นตอนดังรูปที่ 26

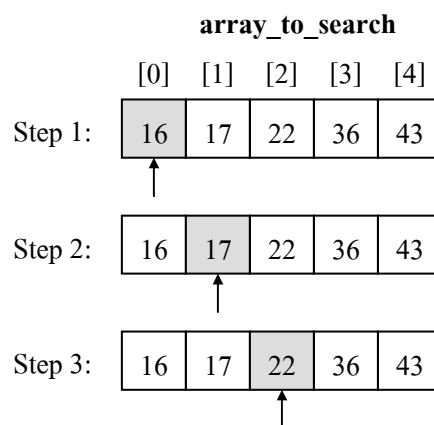
กำหนด `search_key = 25`



รูปที่ 26

จากรูปที่ 26 จะพบว่า ในขั้นตอนที่ 4 ค่าของตัวแปรแถวลำดับตัวที่ 4 มีค่ามากกว่า ค่าของ `search_key` จึงหยุดการค้นหา แสดงว่าข้อมูลที่ต้องการค้นหาไม่อยู่ในแถวลำดับนี้

กำหนด `search_key = 22`



รูปที่ 27

จากรูปที่ 27 จะพบว่าในขั้นตอนที่ 3 ค่าของ `search_key <= array_to_search[2]` ดังนั้นเราจะกล่าวว่าข้อมูลที่ต้องการค้นหาอยู่ในแถวลำดับนี้

จากตัวอย่างที่ 58 เราสามารถนำมาเขียนเป็นฟังก์ชันที่ใช้ในการค้นหาแบบเรียงลำดับ โดยข้อมูลในแถวลำดับต้องมีการเรียงลำดับจากน้อยไปมากได้ดังรูปที่ 28

```

/*****
Function Name : sequential_search_ordered
Purpose       : Searches a one-dimensional sorted array of integers using
                the sequential search algorithm.
Receives      : search_key, list_size, array_to_search
Returns       : success, index
*****/
void sequential_search_ordered (list_item_type search_key,
                               int list_size,
                               const list_item_type array_to_search[ ],
                               char success[ ], int *index) {

    /* Local variables : */

    int i;

    /* Function body : */

    strcpy (success, "FALSE");
    *index = -1;

    for (i = 0; i < list_size; i++)
        if (search_key < array_to_search[i]) {
            break;
        }
        else if (search_key == array_to_search[i]) {
            *index = i;
            strcpy (success, "TRUE");
            break;
        } /* end if */
    /* end for */
} /* end function sequential_search_unordered */

```

รูปที่ 28



**ตัวอย่างโปรแกรมที่ 59** เป็นโปรแกรมการค้นหาค่าข้อมูลแบบเรียงลำดับไปเรื่อย ๆ จนหมดรายการแถวลำดับ

```

/* Linear search of an array */
#include <stdio.h>
#define SIZE 100

/* function prototype */
int linearSearch ( const int array[ ], int key, int size );

/* function main begins program execution */
int main ()
{
    int a[SIZE]; /* create array a */
    int x; /* counter for initializing elements 0–99 of array a */
    int searchKey; /* value to locate in array a */
    int element; /* variable to hold location of searchKey or –1 */

    /* create data */
    for ( x = 0; x < SIZE; x++ ) {
        a[x] = 2 * x;
    } /* end for */

    printf ( “Enter integer search key : \n” );
    scanf ( “%d”, &searchKey );

    /* attempt to locate searchKey in array a */
    element = linearSearch ( a, searchKey, SIZE );

    /* display results */
    if ( element != –1 ) {
        printf ( “Found value in element %d\n”, element );
    } /* end if */
    else {
        printf ( “Value not found\n” );
    } /*end else */

    return 0; /* indicates successful termination */

} /* end main */

/* compare key to every element of array until the location is found
or until the end of array is reached ; return subscript of element
if key or –1 if key is not found */
int linearSearch ( const int array[ ], int key, int size )
{

```

```

int n ; /* counter */

/* loop through array */
for ( n = 0 ; n < size ; ++n ) {

    if ( array[n] == key ) {
        return n ; /* return location of key */
    } /* end if */

} /* end for */

return -1 ; /* key not found */

} /* end function linearSearch */

```

```

Enter integer search key ;
36
Found value in element 18

```

```

Enter integer search key ;
37
Value not found

```

## 7.2 การค้นหาแบบไบนารี

เป็นการค้นหาที่มีประสิทธิภาพดีมากในกรณีมีข้อมูลจำนวนมาก หลักการของวิธีนี้จะต้องมีการเรียงลำดับข้อมูลจากน้อยไปมากก่อน หลังจากนั้นก็จะนำค่าของ `search_key` มาเปรียบเทียบกับค่าของข้อมูลตัวที่อยู่ตรงกลาง ในการเปรียบเทียบนี้จะมี 3 กรณีที่เป็นไปได้ กล่าวคือ

**กรณี 1)** ค่าของ `search_key` เท่ากับค่าของข้อมูลตัวที่อยู่ตรงกลาง ในกรณีนี้จะถือว่าการค้นหาประสบความสำเร็จ และจะยุติการค้นหา

**กรณี 2)** ค่าของ `search_key` มีค่าน้อยกว่าค่าของข้อมูลตัวที่อยู่ตรงกลาง ในกรณีนี้แสดงว่าข้อมูลทุกตัว ตั้งแต่ตัวที่อยู่ตรงกลางขึ้นไป เราจะไม่นำมาพิจารณาเนื่องจากมีค่ามากกว่า `search_key` ทุกจำนวน

**กรณีที่ 3)** ค่าของ `search_key` มีค่ามากกว่าค่าของข้อมูลตัวที่อยู่ตรงกลาง ในกรณีนี้แสดงว่าข้อมูลทุกตัว ตั้งแต่ตัวที่อยู่ตรงกลางลงไป เราจะไม่นำมาพิจารณา เนื่องจากมีค่าน้อยกว่า `search_key` ทุกจำนวน

จาก 3 กรณีนี้ จะเป็นการค้นหาข้อมูลที่ดีคือ จะมีการแบ่งข้อมูลออกเป็น 2 ชุด คือชุดที่มีค่าของทุกจำนวนน้อยกว่าค่าที่อยู่ตรงกลาง กับชุดที่มีค่าของทุกจำนวนมากกว่าค่าที่อยู่ตรงกลางและขนาดของรายการที่ต้องการค้นหา จะลดลงไปที่ละครึ่งเสมอ ซึ่งในการค้นหาแบบไบนารีนี้ จะนำทั้ง 3 กรณีมาพิจารณาจนกระทั่งค้นหาข้อมูลที่ต้องการจนพบ หรือจนกระทั่งรายการของข้อมูลที่ต้องการค้นหาลดลงจนเหลือข้อมูลที่ไม่เหมือนกับค่า `search_key` ซึ่งจะแสดงว่าเป็นการค้นหาที่ไม่ประสบความสำเร็จ

**ตัวอย่างที่ 60** เป็นการแสดงการค้นหาข้อมูลแบบไบนารี

โดยจะกำหนดตัวแปร `first` เก็บตำแหน่งเริ่มต้นของรายการที่ต้องการค้นหา

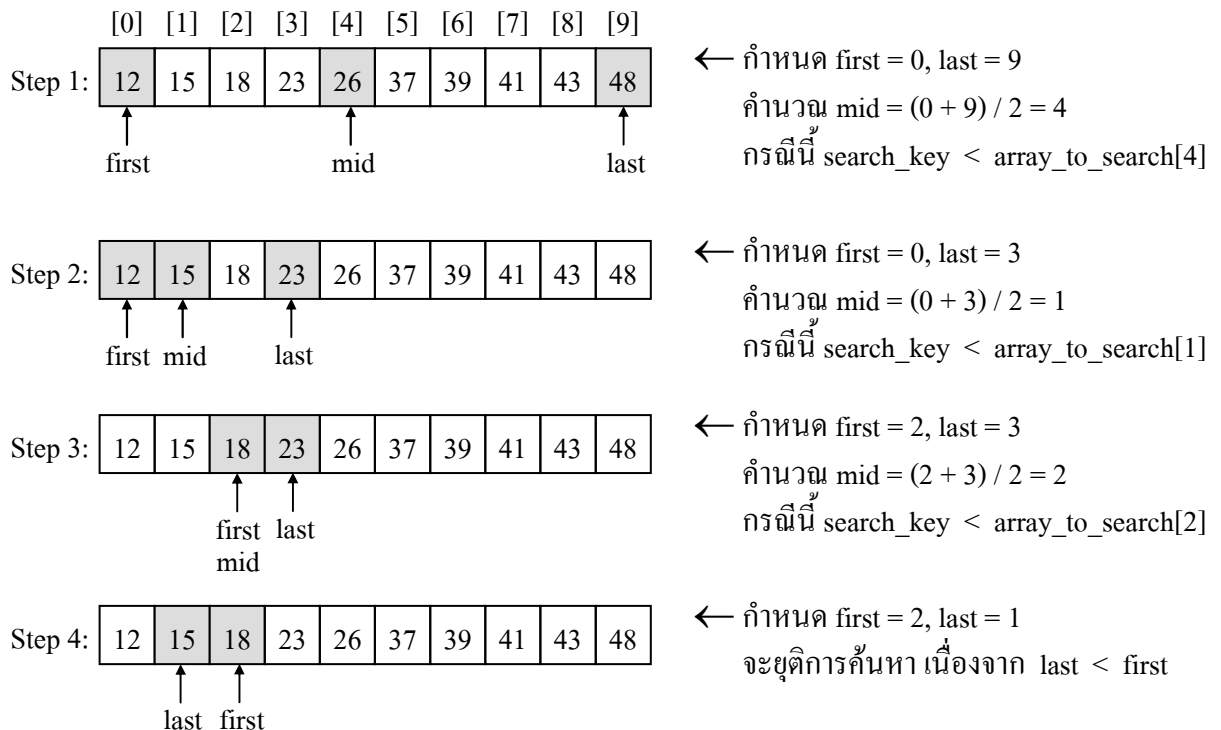
ตัวแปร `last` เก็บตำแหน่งสุดท้ายของรายการที่ต้องการค้นหา และ

ตัวแปร `mid` เก็บตำแหน่งตรงกลางของรายการที่ต้องการค้นหา

โดย  $mid = (first + last) / 2$  ซึ่งสามารถแสดงขั้นตอนได้ดังรูปที่ 29

กำหนด `search_key = 17`

**array\_to\_search**

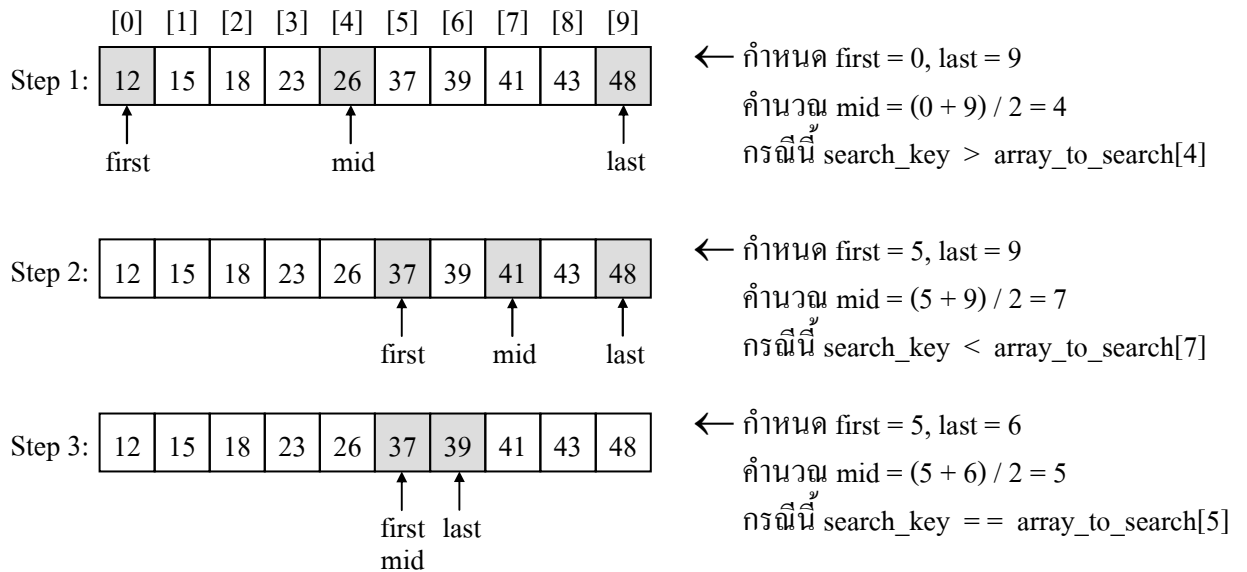


รูปที่ 29

จากรูปที่ 29 จะพบว่า ข้อมูลที่ต้องการค้นหาไม่อยู่ในแถวลำดับ

กำหนด  $search\_key = 37$

**array\_to\_search**



รูปที่ 30

จากรูปที่ 30 เราจะพบว่าในขั้นตอนที่ 3 จะพบข้อมูลที่ต้องการค้นหาอยู่ในลำดับที่ 6

จากตัวอย่างที่ 58 เราสามารถนำมาสรุปเป็นขั้นตอนวิธีในการค้นหาแบบไบนารีได้ดังนี้

- กำหนดให้
- first    เก็บตำแหน่งเริ่มต้นของรายการที่ต้องการค้นหา
  - last    เก็บตำแหน่งสุดท้ายของรายการที่ต้องการค้นหา
  - mid    เก็บตำแหน่งตรงกลางของรายการที่ต้องการค้นหา
- โดย       $mid = (first + last) / 2$

การค้นหาข้อมูลโดยวิธีนี้ จะแบ่งออกเป็น 3 กรณี คือ

- 1) ถ้าค่าของ  $search\_key$  มีค่าเท่ากับค่าข้อมูลตรงกลาง แสดงว่า พบข้อมูลที่ต้องการค้นหา
- 2) ถ้าค่าของ  $search\_key$  มีค่าน้อยกว่าค่าข้อมูลตรงกลาง เราจะกำหนดให้  $first$  มีค่าคงเดิม และ  $last = mid - 1$  ;
- 3) ถ้าค่าของ  $search\_key$  มีค่ามากกว่าค่าข้อมูลตรงกลาง เราจะกำหนดให้  $first = mid + 1$  และ  $last$  มีค่าคงเดิม

**ตัวอย่างโปรแกรมที่ 61** เป็นโปรแกรมแสดงการค้นหาแบบไบนารี โดยจะใช้เครื่องหมาย \* แทนข้อมูลตรงกลาง

```

/* Binary search of an array */
#include <stdio.h>
#define SIZE 15

/* function prototype */
int binarySearch ( const int b[ ], int searchKey, int low, int high );
void printHeader ( void );
void printRow ( const int b[ ], int low, int mid, int high );

/* function main begins program execution */
int main ()
{
    int a[SIZE]; /* create array a */
    int i; /* counter for initializing elements 0-14 or array a */
    int key; /* value to locate in array a */
    int result; /* variable to hold location of key or -1 */

    /* create data */
    for ( i = 0; i < SIZE; i++ ) {
        a[i] = 2 * i;
    } /* end for */

    printf ( "Enter a number between 0 and 28 : " );
    scanf ( "%d", &key );

    printHeader();

    /* search for key in array a */
    result = binarySearch ( a, key, 0, SIZE - 1 );

    /* display results */
    if ( result != -1 ) {
        printf ( "\n%d found in array element %d\n", key, result );
    } /* end if */
    else {
        printf ( "\n%d not found\n", key );
    } /* end else */

    return 0; /* indicates successful termination */
} /* end main */

/* function to perform binary search of an array */
int binarySearch ( const int b[ ], int searchKey, int low, int high )
{
    int middle; /* variable to hold middle element of array */

```

```

/* loop until low subscripts is greater than high subscript */
while ( low <= high ) {

    /*determine middle element of subarray being searched */
    middle = ( low + high ) / 2 ;

    /* display subarray used in this loop iteration */
    printRow ( b, low, middle, high ) ;

    /* if searchKey matched middle element, return middle */
    if ( searchKey == b[middle] ) {
        return middle ;
    } /* end if */

    if searchKey less than middle element, set new high */
    else if ( searchKey < b[middle] ) {
        high = middle - 1 ; /* search low end of array */
    } /* end else if */

    /* if searchKey greater than middle element, set new low */
    else {
        low = middle + 1 ; /* search high end of array */
    } /* end else */
} /* end while */

return -1 ; /* searchKey not found */

} /* end function binarySearch */

/* Print a header for the output */
void printHeader ( void )
{
    int i ; /* counter */

    printf ( "\nSubscripts : \n" ) ;

    /* output column head */
    for ( i = 0 ; i < SIZE ; i++ ) {
        printf ( "%3d", i ) ;
    } /* end for */

    printf ( "\n" ) ; /* start new line of output */
    /* output line of - characters */
    for ( i = 1 ; i <= 4 * SIZE ; i++ ) {
        printf ( "-" ) ;
    } /* end for */

    printf ( "\n" ) ; /* start new line of output */
} /* end function printHeader */

```

```

/* Print one row of output showing the current
   part of the array being processed. */
void printRow ( const int b[ ], int low, int mid, int high )
{
    int i; /* counter for iterating through array b */

    /* loop through entire array */
    for ( i = 0 ; i < SIZE ; i++ ) {

        /* display spaces if outside current subarray range */
        if ( i < low || i > high ) {
            printf( "    " );
        } /* end if */
        else if ( i == mid ) { /* display middle element */
            printf( "%3d*", b[i] ); /* mark middle value */
        } /* end else if */
        else { /* display other elements in subarray */
            printf( "%3d ", b[i] );
        } /* end else */

    } /* end for */

    printf( "\n" ); /* start new line of output */
} /* end function printRow */

```

```

Enter a number between 0 and 28 : 25
Subscripts :
0  1  2  3  4  5  6  7  8  9  10  11  12  13  14
0  2  4  6  8  10  12  14* 16  18  20  22  24  26  28
                                16  18  20  22* 24  26  28
                                    24  26* 28
                                        24*

25 not found

```

```

Enter a number between 0 and 28 : 8
Subscripts :
0  1  2  3  4  5  6  7  8  9  10  11  12  13  14
0  2  4  6  8  10  12  14* 16  18  20  22  24  26  28
0  2  4  6* 8  10  12
                8  10* 12
                    8*

8 found in array element 4

```

```

Enter a number between 0 and 28 : 6
Subscripts :
0  1  2  3  4  5  6  7  8  9  10  11  12  13  14
0  2  4  6  8  10  12  14* 16  18  20  22  24  26  28
0  2  4  6* 8  10  12

6 found in array element 3

```