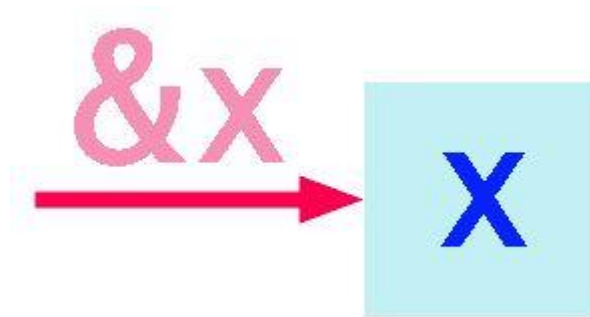


ตัวแปรพอยน์เตอร์



ตัวแปรพอยน์เตอร์คืออะไร

- เป็นชนิดข้อมูลประเภทหนึ่งและจัดเป็นชนิดข้อมูลขั้นสูง
- ตัวชี้ไม่ได้เก็บข้อมูล (Value) โดยตรง แต่เก็บที่อยู่ (Address) ของข้อมูลแทน
- โดยปกติแล้วตัวแปร (Variable) ทุกตัวอยู่ในหน่วยความจำเครื่อง (RAM) และมีที่อยู่กำกับไว้คล้ายเลขที่บ้าน



- ที่อยู่ของตัวแปร x กับ ค่าของ x เป็นของคู่กันตลอด แต่ก่อนเราสนใจแต่ค่าของตัวแปร ไม่ได้สนใจที่อยู่ของ ตัวแปร
- การที่รู้ที่อยู่ของ x จะทำให้เราอ่านหรือเปลี่ยนค่าของ x ได้

เราใช้ pointer หรือที่อยู่ของข้อมูลทำอะไร?

- ใช้เพื่อข้ามขีดจำกัดของการรับ – ส่งพารามิเตอร์ของฟังก์ชัน
 - พารามิเตอร์ที่ส่งไปให้ฟังก์ชัน โดยปกติแล้วจะเป็นตัวแปรทั่วไป ไม่ใช่ตัวชี้
 - เมื่อฟังก์ชันแก้ค่าพารามิเตอร์ในตัวแปรทั่วไป ค่าที่เปลี่ยนไปนั้นจะมีผลอยู่เฉพาะในฟังก์ชัน (เพราะเป็นแค่สำเนาของค่าที่ส่งไป)
 - คำถาม ?? ถ้าเราอยากให้ฟังก์ชันแก้ค่าตัวแปรต้นฉบับ ทำอย่างไร (เราใช้ที่อยู่ของตัวแปรกับคำสั่ง scanf() มาตลอด)
- Pointer กับ Array
 - ✓ อาเรย์ เป็นตัวแปร pointer แบบหนึ่ง เพราะอาเรย์จะชี้ไปที่ข้อมูลตัวแรกเสมอ

การประกาศตัวแปรพอยน์เตอร์

ประเภทของข้อมูล* ชื่อตัวแปร

`int* pt_x;` สร้างตัวแปรพอยน์เตอร์ชนิด `int` ทำให้ `pt_x`
 ใช้เก็บตำแหน่งที่อยู่ของตัวแปรชนิด `int` เท่านั้น

`float* pt_num;` สร้างตัวแปรพอยน์เตอร์ชนิด `float` ทำให้ `pt_num`
 ใช้เก็บตำแหน่งที่อยู่ของตัวแปร ชนิด `float` เท่านั้น

`char* pt_ch;` สร้างตัวแปรพอยน์เตอร์ชนิด `char` ทำให้ `pt_ch`
 ใช้เก็บตำแหน่งที่อยู่ของตัวแปร ชนิด `char` เท่านั้น

ข้อควรจำ : ตัวชี้ (pointer) จริง ๆ แล้วเป็นชนิดข้อมูล (data type) ขึ้นอยู่กับชนิดหนึ่ง

- ตัวอย่าง

- `int *p;`

ประกาศตัวแปร p เป็นพอยน์เตอร์

```
int a;  
int *p;  
a = 10;  
p = &a;
```

ใส่ค่า 10 ในตัวแปร a ซึ่งจะอยู่ในหน่วย
ความจำตำแหน่งหนึ่ง

นำค่าแอดเดรสที่เก็บ 10 ไปใส่ในตัวแปร p

ประกาศตัวแปรชื่อ p ชี้ไปยังหน่วยความจำข้อมูลประเภท integer

การใช้ตัวชี้ (Pointer)

- มีอยู่ 2 ลักษณะ คือ
 - 1) ใช้บันทึกค่าที่อยู่ของตัวแปร
 - 2) ใช้อ่านค่าหรือเปลี่ยนค่าตัวแปรที่เราสนใจ
- การใช้งานในแต่ละลักษณะจะมีวิธีเขียนไม่เหมือนกัน ดังนี้
 - ตอนบันทึกค่าที่อยู่ของตัวแปร เราจะใช้ชื่อของตัวชี้ตรง ๆ เช่น

```
int* ptr ;  
int x = 5;  
ptr = &x;
```

- ตอนอ่านหรือเขียนค่าตัวแปรที่เราสนใจในเราใส่ * ไว้หน้าชื่อตัวชี้

```
printf(“%d” , *ptr) ;  
*ptr = 7;  
printf(“%d”,x);
```

ดูตัวอย่างโปรแกรม

```
#include <stdio.h>
void main()
{
    int * ptr;
    int x = 5;
    ptr = &x;

    printf("%d\n" , *ptr);

    *ptr = 7;

    printf("%d\n", x);
}
```

เพราะ ptr เก็บที่อยู่ของ x ไว้เมื่อเราใช้ *ptr
มันจะดึงค่าของ x มาแสดง
ผลลัพธ์ : 5

การใช้เครื่องหมาย * หน้าตัวชี้หมายถึง
การเข้าถึงค่า (Value) ของตัวแปรที่มันชี้
ptr ชี้ใคร ??
ค่าของตัวแปร x เท่ากับ ??

ผลลัพธ์ที่ได้ :
ค่าของตัวแปร x เท่ากับ ??

การประยุกต์ใช้กับฟังก์ชัน

- ทบทวนการใช้คำสั่ง scanf() กับที่อยู่ของตัวแปร

```
int x = 5;  
int* ptr = &x;
```

```
scanf("%d", &x);  
printf("%d\n", x);
```

```
scanf("%d", ptr);  
printf("%d\n", x );
```

คำสั่ง scanf() เราใช้ &x ส่งที่อยู่ของตัวแปร x
สังเกต : scanf() เปลี่ยนค่า x ได้เพราะ มันแก้
ค่าของ x ผ่านที่อยู่ของ x

เราส่งที่อยู่ของ x ในคำสั่ง scanf() ผ่านตัวชี้
แบบนี้ก็ได้ เพราะ

ทบทวนเรื่องการส่งค่าและเปลี่ยนค่าพารามิเตอร์

```
int add_multi (int x , int y){
```

```
    x = x + y;
```

```
    x = x*y;
```

```
    return x;
```

```
}
```

```
void main(){
```

```
    int x ,y ,result ;
```

```
    x = 5; y = 2;
```

```
    result = add_multi(x,y);
```

```
    printf(“%d\n”,x);
```

```
    printf(“%d\n”, result);
```

```
}
```

ความแตกต่างของการส่งค่าให้ฟังก์ชัน scanf()

กับการส่งค่าให้ฟังก์ชัน add_multi

พารามิเตอร์ที่ส่งไปให้ scanf ถูกเปลี่ยนค่าจริง ๆ

แต่พารามิเตอร์ที่ส่งไปให้ add_multi ไม่ถูกเปลี่ยนค่า

ผลลัพธ์ : x =.....

ผลลัพธ์ : result =.....

พารามิเตอร์ที่เป็นที่อยู่ของข้อมูล

- พารามิเตอร์ที่รับที่อยู่ของตัวแปรหรือตัวชี้ได้ต้องมีชนิดข้อมูลเป็นตัวชี้
- ถ้าเราต้องการแก้ไขพารามิเตอร์ x ในฟังก์ชัน `add_multi` เป็นแบบตัวชี้เราก็ต้องใช้เครื่องหมาย `*` ตามหลังชนิดข้อมูลของ x

```
int add_multi (int x , int y){  
    x = x + y;    x = x*y;    return x;  
}
```

แบบเดิม


```
int add_multi (int* x , int y){  
    *x = *x + y;  
    *x = *x*y;  
    return *x;  
}
```

แบบตัวชี้


เปรียบเทียบผลลัพธ์

```
void main() {  
    int x, y, result;  
    x = 5;  y = 2;  
  
    result = add_mult(x, y);  
    printf("%d\n", x);  
    printf("%d\n\n", result);  
}
```

แบบเดิมไม่เปลี่ยนค่า x



แบบนี้ค่า x ที่ส่งไปจะได้รับการแก้ไขภายใน
add_mult_ptr และมีผลกับ x ใน main ด้วย



```
result = add_mult_ptr(&x, y);  
printf("%d\n", x);  
printf("%d\n\n", result);  
}
```

ตัวชี้และอาร์กิวเมนต์ของฟังก์ชัน (Pointer and Function Arguments)

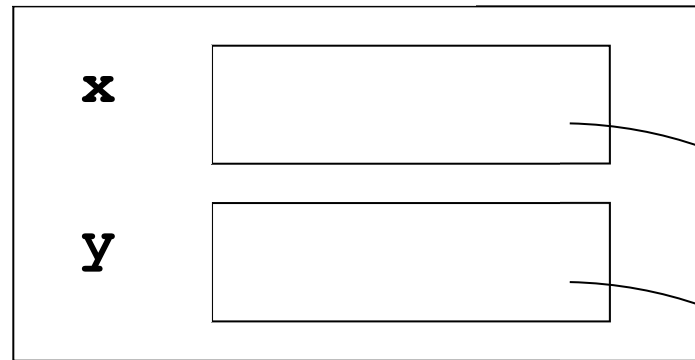
เนื่องจากภาษาซีมีการส่งอาร์กิวเมนต์ให้กับฟังก์ชันแบบ By Value และฟังก์ชันสามารถคืนค่า (return) ค่าได้เพียงหนึ่งค่า หากต้องการให้ฟังก์ชันมีการเปลี่ยนแปลงค่าและคืนค่ากลับมายังฟังก์ชันที่เรียกใช้มากกว่าหนึ่งค่าจะต้องนำพอยน์เตอร์เข้ามาช่วย

โปรแกรมตัวอย่างการสลับค่าตัวแปร 2 ตัวโดยผ่านฟังก์ชัน จะแสดงการส่งอาร์กิวเมนต์ให้เป็นพอยน์เตอร์

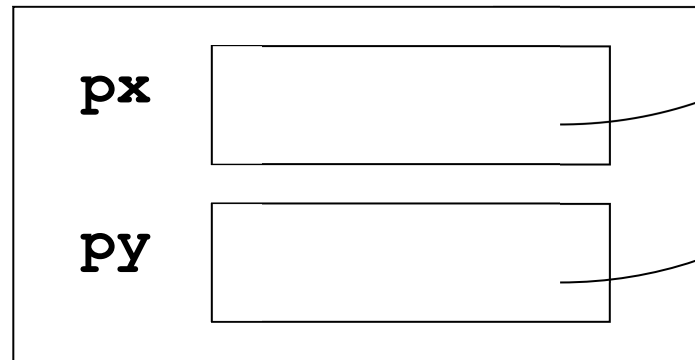
```
#include <stdio.h>
void swap (int *, int *);
void main ( )
{
    int x = 5, y = 10;
    printf("Before swap : x = %d , y = %d\n",x,y);
    swap ( &x, &y);
    printf("After swap : x = %d , y = %d\n",x,y);
}
```

```
void swap (int *px, int *py)
{
    int temp;
    temp = *px;
    *px = *py;
    *py = temp;
}
```

`in main ()`



`in swap ()`



รูปแสดงความสัมพันธ์ของการส่งอาร์กิวเมนต์แบบพอยน์เตอร์กับฟังก์ชัน

การใช้ตัวชี้กับอาร์เรย์

การทำงานใด ๆ ของอาร์เรย์สามารถใช้พอยน์เตอร์เข้ามาช่วย ซึ่งจะทำให้มีความเร็วในการทำงานสูงขึ้น สมมติว่ามีอาร์เรย์ `a` และพอยน์เตอร์ `pa` ดังนี้

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

```
int *pa;
```

กำหนดให้พอยน์เตอร์ `pa` ชี้ไปยังอาร์เรย์ `a` ด้วยคำสั่ง

```
pa = &a[0]; /* หรือใช้คำสั่ง pa = a; */
```

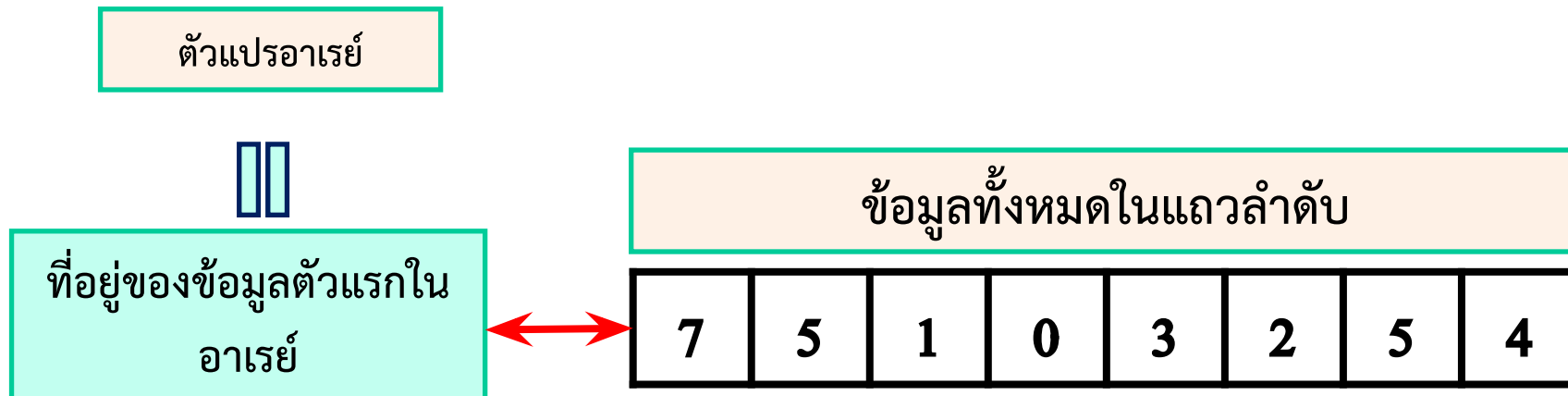
`pa` จะเก็บค่าแอดเดรสเริ่มต้นของอาร์เรย์ `a`

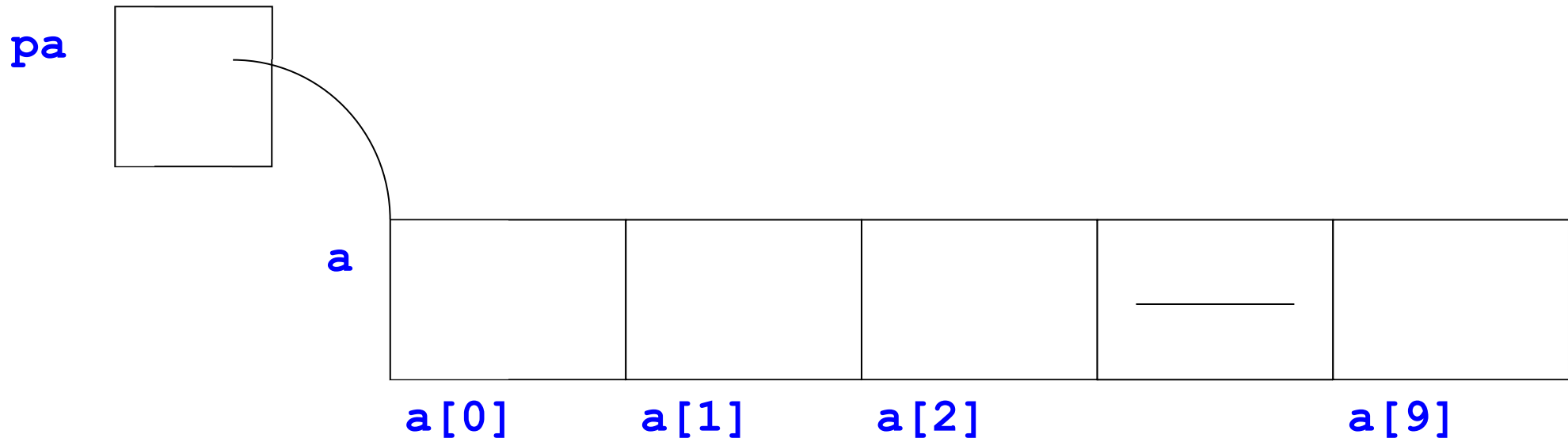
ตัวชี้กับแถวลำดับ

- แถวลำดับแท้จริงเป็นตัวชี้ในรูปแบบหนึ่ง การรู้จักวิธีส่งผ่านแถวลำดับไปเป็นพารามิเตอร์ของฟังก์ชันจะช่วยให้เราได้อย่างมาก

สิ่งที่ต้องเข้าใจ : ตัวแปรแถวลำดับ, ที่อยู่ของข้อมูล, และ ข้อมูลในแถวลำดับ

- ค่าตัวแปรแถวลำดับที่จริงเก็บที่อยู่ของข้อมูลตัวแรกในแถวลำดับไว้
- ตัวแปรแถวลำดับทำให้รูปแบบการอ้างถึงข้อมูลดูเข้าใจง่ายขึ้น





รูป แสดงตัวชี้ชี้ไปยังแอดเดรสเริ่มต้นของอาร์เรย์

การนำไปใช้งานจะสามารถอ่านค่าอาร์เรย์ผ่านพอยน์เตอร์ได้ดังนี้

$x = *pa;$

จะเป็นการกำหนดค่าให้ x มีค่าเท่ากับ $a[0]$ การเลื่อนไปอ่านค่าสมาชิกตำแหน่งต่าง ๆ ของอาร์เรย์ผ่านทางพอยน์เตอร์สามารถทำได้โดยการเพิ่มค่าพอยน์เตอร์ขึ้น 1 เพื่อเลื่อนไปยังตำแหน่งถัดไป หรือเพิ่มค่าขึ้น N เพื่อเลื่อนไป N ตำแหน่ง หรืออาจจะลดค่าเพื่อเลื่อนตำแหน่งลง

ทดลองใช้ตัวแปรแถวลำดับในฐานะตัวชี้

- จำไว้ว่าการอ่านเขียนข้อมูลที่ตัวชี้อ้างอิงอยู่ต้องใช้เครื่องหมาย * นำหน้า
- บ้านเลขที่มักเป็นเลขติดต่อกันไป ที่อยู่ของข้อมูลในแถวลำดับก็เป็นแบบนั้น

```
int A[3] = {6, 7, 8};  
printf("array = %d %d %d\n",  
      A[0], A[1], A[2]);
```

การเรียงตำแหน่งอาเรย์
ตามลำดับ 0, 1, 2

```
printf("array = %d %d %d\n",  
      *(A+0), *(A+1), *(A+2));
```

คือการเรียง
ตามที่อยู่ของข้อมูลในช่องถัดไปนั่นเอง

- printf ทั้งสองให้ผลลัพธ์เหมือนกันทุกประการ เพราะแท้จริงการเขียนว่า
A[0], A[1], A[2], ..., A[i] ก็คือ *(A+0), *(A+1), *(A+2), ..., *(A+i)

ส่งอาเรย์ไปเป็นพารามิเตอร์ของฟังก์ชัน (1)

```
double average(int A[], int n) {  
    double sum = 0;  
    int i;  
    for(i = 0; i < n; ++i) {  
        sum += A[i];  
    }  
    return sum / n;  
}  
  
void main() {  
    int A[4] = {1, 2, 3, 4};  
    double avg = average(A, 4);  
    printf("%lf\n", avg);  
}
```

ส่งอาร์เรย์ไปเป็นพารามิเตอร์ของฟังก์ชัน (2)

```
double average(int* A, int n) {  
    double sum = 0;  
    int i;  
    for(i = 0; i < n; ++i) {  
        sum += A[i];  
    }  
    return sum / n;  
}
```

ถึงจะเขียนไว้ข้างบนว่าเป็นตัวชี้ แต่เราก็
ใช้มันในรูปแบบอาร์เรย์ได้เหมือนกัน

```
void main() {  
    int A[4] = {1, 2, 3, 4};  
    double avg = average(A, 4);  
    printf("%lf\n", avg);  
}
```

ตัวอย่าง

ฟังก์ชันที่รับพารามิเตอร์เป็นพอยน์เตอร์ โดยอาร์กิวเมนต์ที่ส่งมาเป็นอาร์เรย์

```
int  strlen (char *s)
{
    int  n;
    for ( n = 0; *s != '\0' ; s++ )
        n++;
    return n;
}
```

```
char  s[ ]
```

จะเห็นว่า s เป็นพอยน์เตอร์ ในฟังก์ชันจะมีการตรวจสอบข้อมูลว่ามีค่าเท่ากับ '\0' หรือไม่ และมีการเลื่อนตำแหน่งทีละ 1 ค่า (s++) (นับว่าข้อมูลมีความยาวเพิ่มขึ้นทีละ1) โดยใช้ n++

การเรียกใช้ฟังก์ชัน strlen สามารถทำได้หลายลักษณะ

```
strlen ("hello world"); /* string constant */
```

```
strlen (array);          /* char array[10] */
```

```
strlen (ptr);             /* char *ptr;    */
```


`f (&a[2])`

หรือ `f (a+2)`

เป็นการส่งแอดเดรสของสมาชิก `a[2]` ให้กับฟังก์ชัน `f`
การประกาศฟังก์ชัน `f` สามารถทำได้โดยการประกาศ

`f (int arr[]) { }`

หรือ `f (int *arr) { }`

ตัวอย่าง

ฟังก์ชัน strlen() ปรับปรุงให้กระชับขึ้น

```
int  strlen (char *s)
{
    char  *p = s;
    while  (*p != '\0')
        p++;
    return  p-s;
}
```

ตัวชี้ตัวอักษรและฟังก์ชัน (Character Pointer and Function)

การทำงานกับข้อความหรือที่เรียกว่า สตริง (String) เป็นการ
ใช้ข้อมูลตัวอักษรหลาย ๆ ตัว หรืออาร์เรย์ของข้อมูลประเภท
char หรืออาจจะใช้พอยน์เตอร์ชี้ไปยังข้อมูลประเภท char การ
ทำงานกับค่าคงที่สตริง (String Constant) สามารถเขียนภายใน
เครื่องหมาย “ ”

“I am a string”

เมื่อมีการใช้ค่าคงที่สตริงจะมีการพื้นที่ในหน่วยความจำเท่ากับความยาวของค่าคงที่สตริงบวกด้วย 1 เนื่องจากลักษณะการเก็บข้อมูลประเภทข้อความในหน่วยความจำจะมีการปะตัวอักษร null หรือ ‘\0’ ต่อท้ายเสมอเพื่อให้รู้ว่าเป็นจุดสิ้นสุดของข้อมูล

การจองพื้นที่ดังกล่าวจะเหมือนการจองพื้นที่ของข้อมูลประเภทอาร์เรย์เป็นอาร์เรย์ของ char

I		a	m		a		s	t	r	i	n	g	\0
---	--	---	---	--	---	--	---	---	---	---	---	---	----

แสดงแบบจำลองการเก็บข้อมูลประเภทสตริงในหน่วยความจำ

ค่าคงที่สตริงที่พบเห็นได้เสมอได้แก่ข้อความที่ใช้ในฟังก์ชัน printf ()

```
printf ( “Hello, world\n” );
```

ฟังก์ชัน printf () จะรับพารามิเตอร์เป็นพอยน์เตอร์ชี้ไปยังแอดเดรส

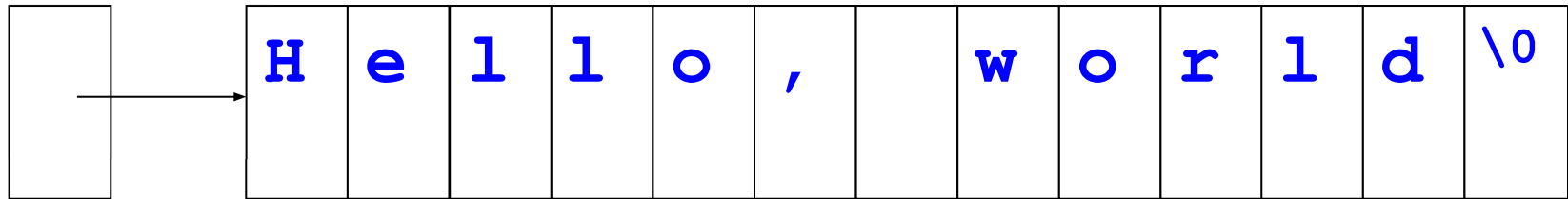
ในการเขียนโปรแกรมจะสามารถใช้พอยน์เตอร์ชี้ไปค่าคงที่ที่สตริงใด ๆ ก็ได้ เช่น

```
char *pmessage = "Hello, world";
```

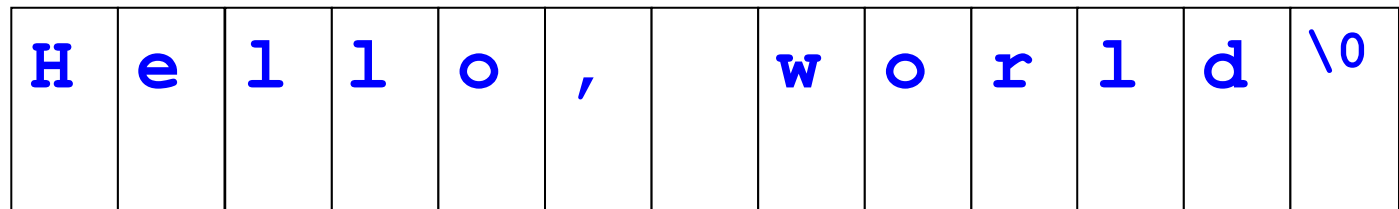
pmessage จะเป็นพอยน์เตอร์ประเภท char ชี้ไปที่อาร์เรย์ของตัวอักษร จะแตกต่างจากการใช้อาร์เรย์ทั่วไปเช่น

```
char amessage[ ] = "Hello, world";
```

pmessage



amessage



การจองพื้นที่ให้กับอาร์เรย์และตัวชี้ชี้ไปยังค่าคงที่สตริง

ตัวอย่าง

ฟังก์ชัน strcpy () ทำหน้าที่สำเนาข้อความจากตัวแปรหนึ่งไปยังอีกตัวแปรหนึ่งเขียนในลักษณะอาร์เรย์

```
void    strcpy ( char  *s,  char  *t )
{
    int  i=0;
    while ( ( s[i] = t[i] )  !=  '\0' )
        i++;
}
```

ตัวอย่าง

ฟังก์ชัน strcpy () เขียนในลักษณะพอยน์เตอร์

```
void strcpy ( char *s, char *t )  
{  
    while ( ( *s = *t ) != '\0' ) {  
        s++;  
        t++;  
    }  
}
```

ตัวอย่าง

ฟังก์ชัน strcpy () เขียนในลักษณะพอยน์เตอร์แบบสั้น

```
void strcpy ( char *s, char *t )  
{  
    while ( ( *s++ = *t++ ) != '\0' ) ;  
}
```