

ตัวชี้

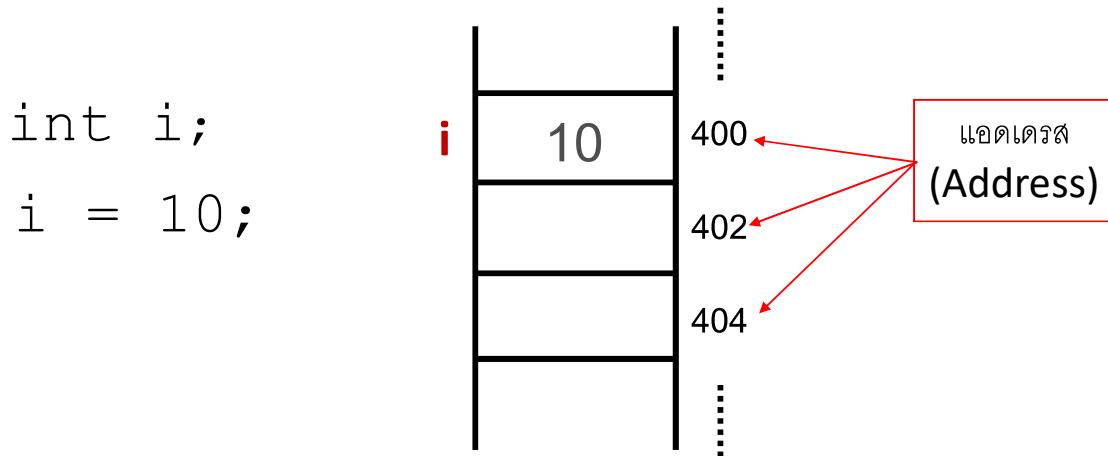
Pointer

ที่มา : อ.ดร.ลือพล พิพานเมฆาภรณ์ ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ
KMUTNB

พอยน์เตอร์หรือตัวชี้

- เป็นชนิดข้อมูลชนิดหนึ่งของภาษาC(int,float,char)
 - มีความเร็วในการทำงานสูง
 - ช่วยประหยัดเนื้อที่ในหน่วยความจำลักษณะประมวลผลเมื่อเทียบกับอาร์เรย์ (array)
 - ใช้ตัวชี้ร่วมกับฟังก์ชันเพื่อเพิ่มประสิทธิภาพการเขียนโปรแกรม

ภาพจำลองการแทนข้อมูลในหน่วยความจำแบบปกติ



แอดเดรส คือ ตำแหน่งที่เก็บข้อมูลในหน่วยความจำ
เสมือนบ้านเลขที่เพื่อใช้ในการอ้างถึงสำหรับนำข้อมูลไปเก็บ
หรือนำออกมาใช้งาน

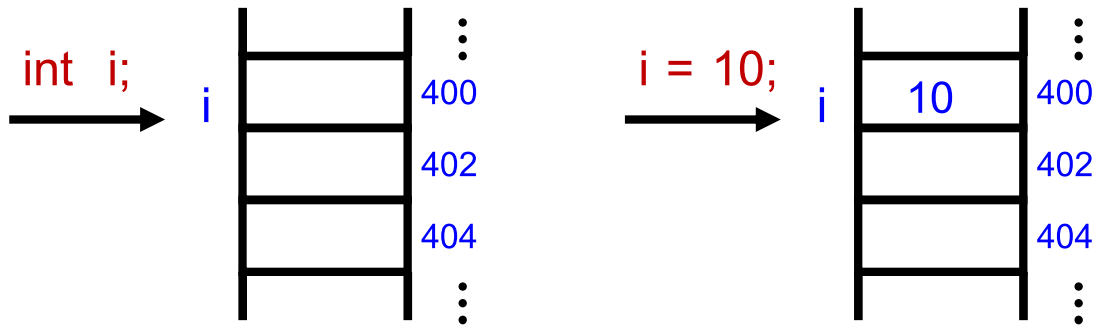
3

ตัวชี้กับแอดเดรส (Pointers and Address)

- วิธีการใช้ตัวชี้หรือพอยน์เตอร์เป็นอีกวิธีที่จะ
เข้าถึงตัวแปรปกติได้
- ตัวแปรชนิดพอยน์เตอร์จะเก็บค่าที่อยู่ของ
หน่วยความจำหลัก ซึ่งต่างกับตัวแปรปกติที่เก็บค่าที่
แท้จริงของข้อมูล
- การใช้ตัวแปรชนิดพอยน์เตอร์จะเป็นการเข้าถึงข้อมูล
หรือเป็นการอ้างถึงตำแหน่งที่เก็บข้อมูล

4

ตัวชี้กับแอดเดรส (Pointers and Address)

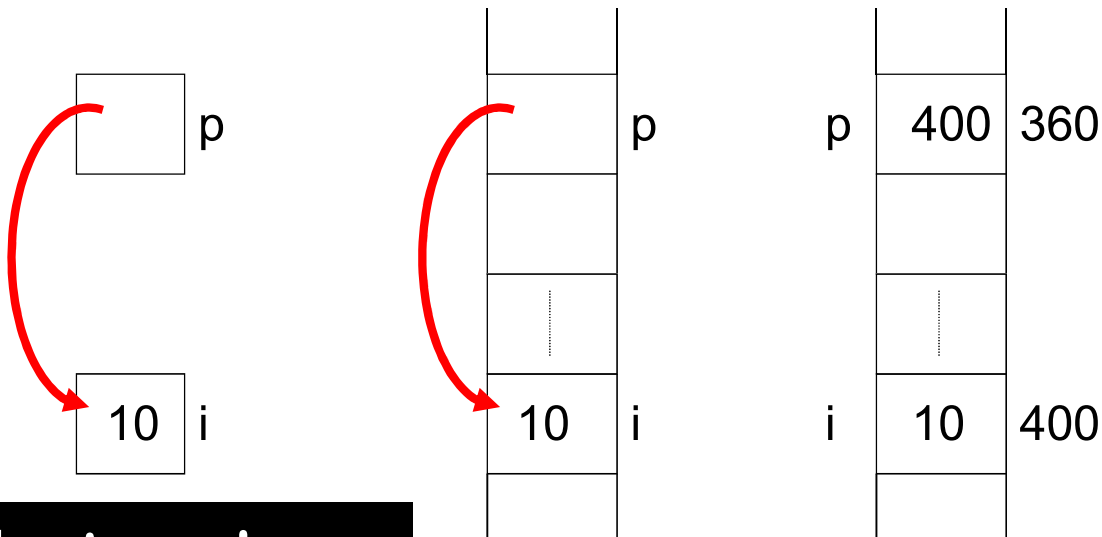


```
int i;  
i = 10;
```

การแทนข้อมูลในหน่วยความจำของตัวแปรประเภทพื้นฐาน

5

ตัวชี้กับแอดเดรส (Pointers and Address)



```
int *p;  
p = &i;
```

การแทนข้อมูลในหน่วยความจำของตัวแปรประเภทตัวชี้

6

การประกาศตัวแปรประเภทตัวชี้

- การประกาศตัวแปรประเภทพอยน์เตอร์จะใช้ Unary Operator * ซึ่งมีชื่อเรียกว่า Indirection หรือ Dereferencing Operator
- โดยจะต้องประกาศประเภทของตัวแปรพอยน์เตอร์ให้สอดคล้องกับประเภทของตัวแปรที่เราต้องการ (ยกเว้นตัวแปรพอยน์เตอร์ประเภท void ที่สามารถชี้ไปยังตัวแปรประเภทใดก็ได้)

7

การประกาศตัวแปรประเภทตัวชี้

```
int *ip;
```

- เป็นการประกาศตัวแปร ip ให้เป็นตัวแปรพอยน์เตอร์ที่ชี้ไปยังตัวแปรประเภท int

```
double *dp, atof(char *);
```

- เป็นการประกาศตัวแปร dp เป็นตัวแปรพอยน์เตอร์ที่ชี้ไปยังตัวแปรประเภท double และประกาศฟังก์ชัน atof มีพารามิเตอร์เป็นตัวแปรพอยน์เตอร์ประเภท char

8

การกำหนดค่าและการอ่านค่าตัวแปรประเภทตัวชี้

- การกำหนดค่าให้กับตัวแปรพอยน์เตอร์จะเป็นการกำหนดแอดเดรสของตัวแปรที่มีประเภทสอดคล้องกับประเภทของตัวแปรพอยน์เตอร์เท่านั้น
- โดยการใช้ Unary Operator **&** เป็นโอเปอเรเตอร์ที่อ้างถึงแอดเดรสของออปเจ็ค (Object) ไต ๆ

9

การกำหนดค่าและการอ่านค่าตัวแปรตัวชี้

```
int    x = 1, y = 2;
int    *ip, *iq;
ip     =  &x;
y      =  *ip;
*ip    =  0;
y      =  5;
ip     =  &y;
*ip    =  3;
iq     =  ip;
```

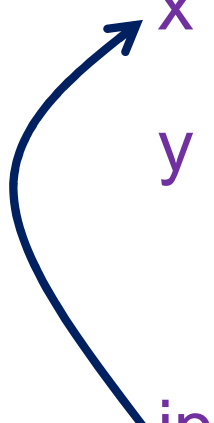
10

x	1	400
y	2	402
	⋮	
ip		500
iq		504

```
int x = 1, y = 2;
int *ip, *iq;
```

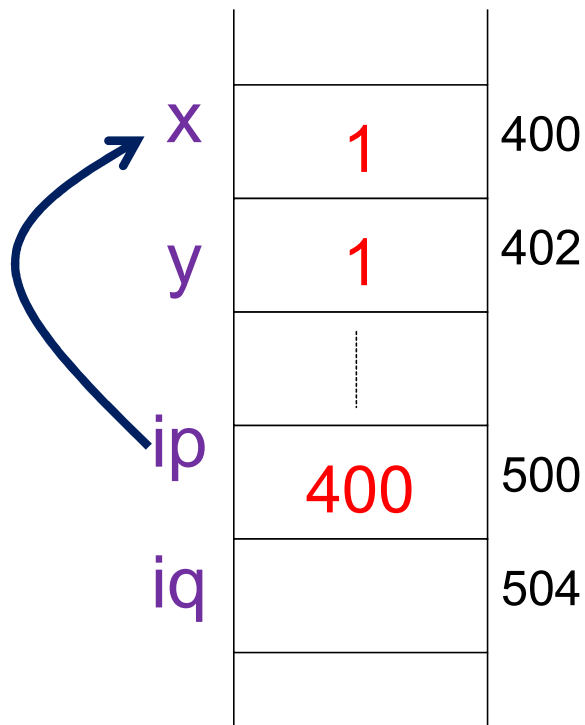
11

x	1	400
y	2	402
	⋮	
ip	400	500
iq		504

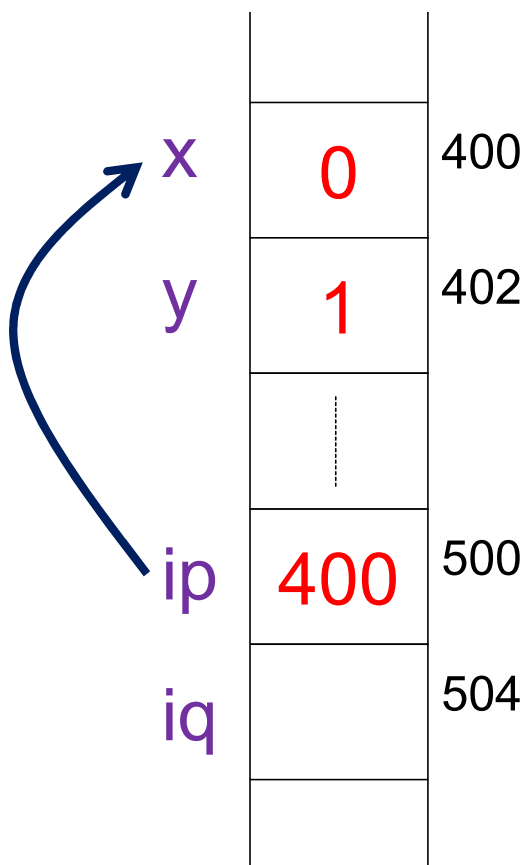


```
ip = &x;
```

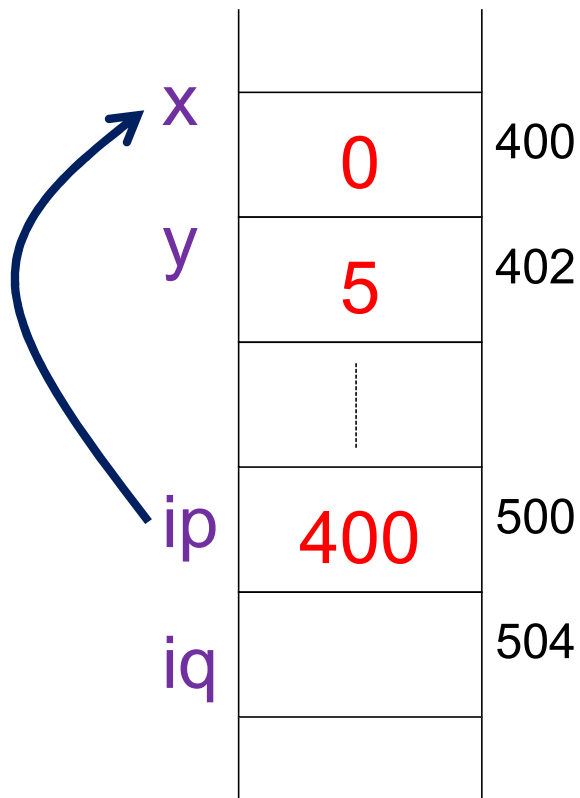
12



`y = *ip;`

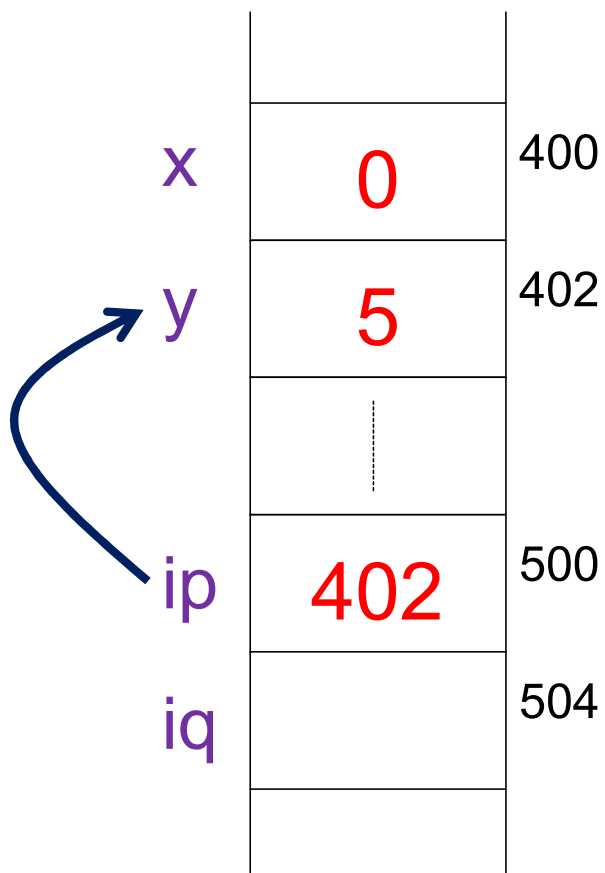


`*ip = 0;`



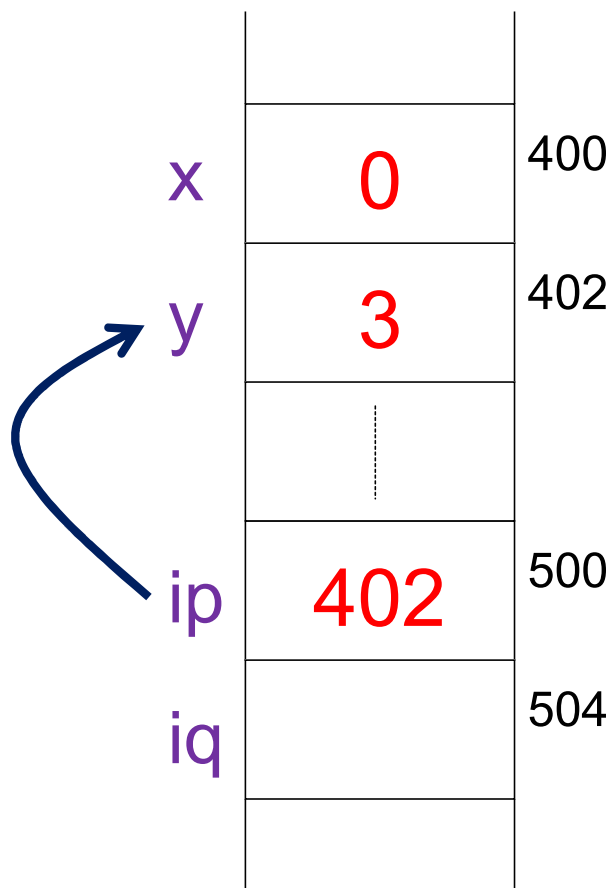
`y = 5;`

15



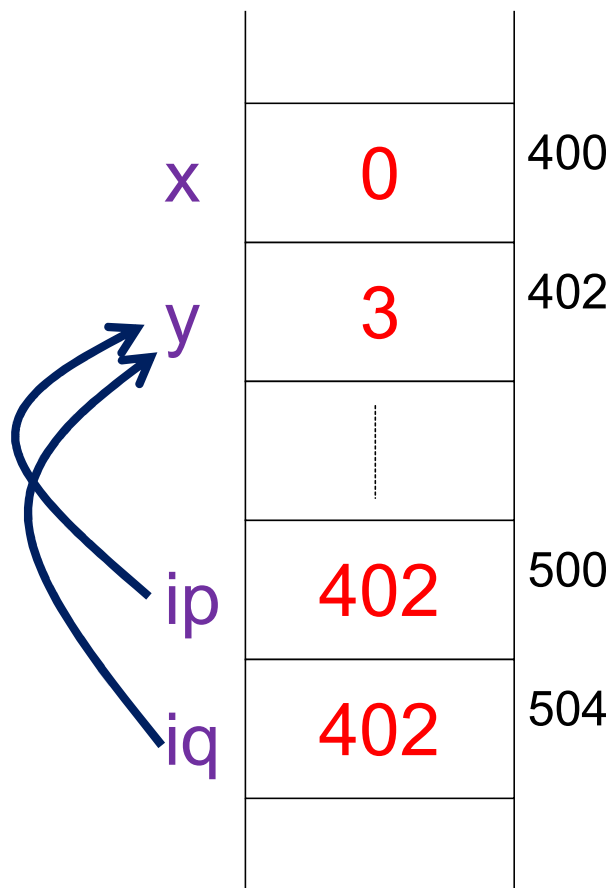
`ip = &y;`

16



`*ip = 3;`

17



`iq = ip;`

18

ตัวชี้กับฟังก์ชัน

- ปัญหาของฟังก์ชันในภาษาซี
 - การส่งอาร์กิวเมนต์ให้กับฟังก์ชันแบบ **Pass by Value** และ
 - ฟังก์ชันสามารถคืนค่า (return) ค่าได้เพียงหนึ่งค่าเท่านั้น
- **พอยน์เตอร์**สามารถช่วยแก้ปัญหานี้
 - ส่งค่าแบบ **Pass by Reference** (ส่งแอดเดรสของตัวแปรไปให้พอยน์เตอร์)
 - **แก้ไขค่าที่ต้องการได้เลย** โดยมีต้องส่งค่ากลับ

19

ตัวอย่าง

```
#include <stdio.h>

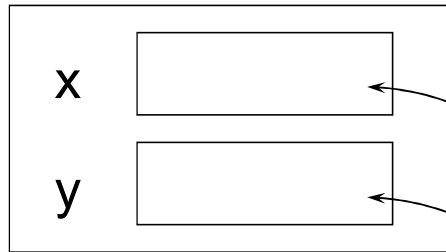
void swap(int *, int *);

void main () {
    int x = 5, y = 10;
    printf("Before x=%d y=%d\n", x, y);
    swap (&x, &y);
    printf("After x=%d y=%d\n", x, y);
}

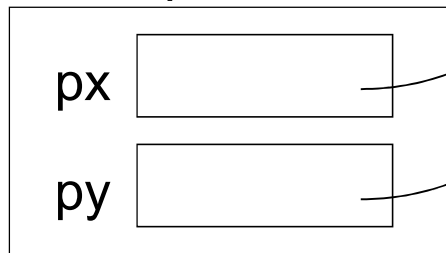
void swap (int *px, int *py)
{ int temp;
  temp = *px;
  *px = *py;
  *py = temp;
}
```

ความสัมพันธ์ของการส่งอาร์กิวเมนต์แบบพอยน์เตอร์กับฟังก์ชัน

in main ()



in swap ()



21

พอยน์เตอร์กับอาร์เรย์

- พอยน์เตอร์มีการทำงานที่ใกล้เคียงกับอาร์เรย์มาก จนกระทั่งเราสามารถเปลี่ยนรูปแบบของอาร์เรย์เป็นพอยต์เตอร์ได้เลย

```
#include <stdio.h>
```

```
void main ( ) {
```

```
    int i, x[5] = {10, 20, 30, 40, 50};
```

```
    for (i=0; i<5; i++)
```

```
        printf("element %d data is%d\n",i+1,*(x+i) );
```

```
}
```

10	20	30	40	50
x[0]	x[1]	x[2]	x[3]	x[4]
*x	*(x+1)	*(x+2)	*(x+3)	*(x+4)

22

การใช้พอยน์เตอร์กับอาร์เรย์

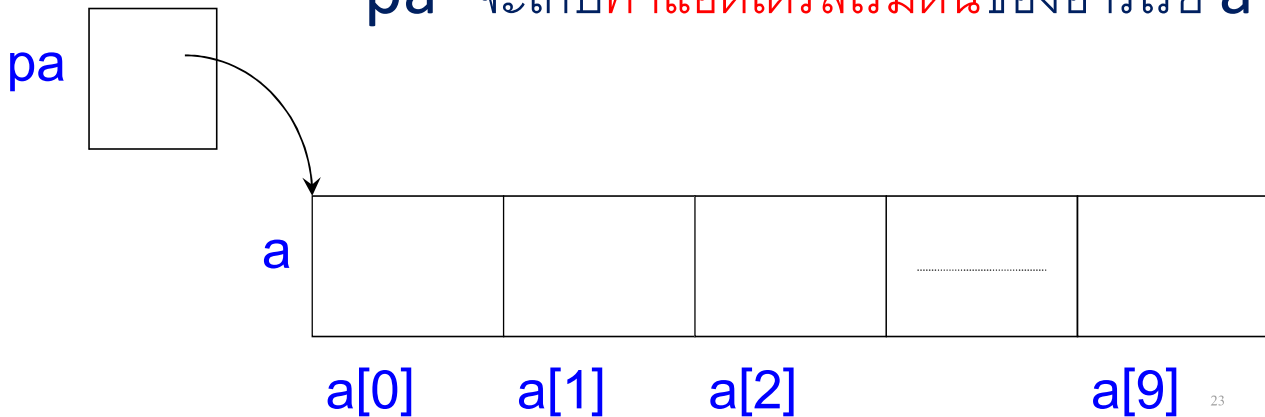
```
int    a[10];
```

```
int    *pa;
```

กำหนดให้พอยน์เตอร์ pa ชี้ไปยังอาร์เรย์ a ด้วยคำสั่ง

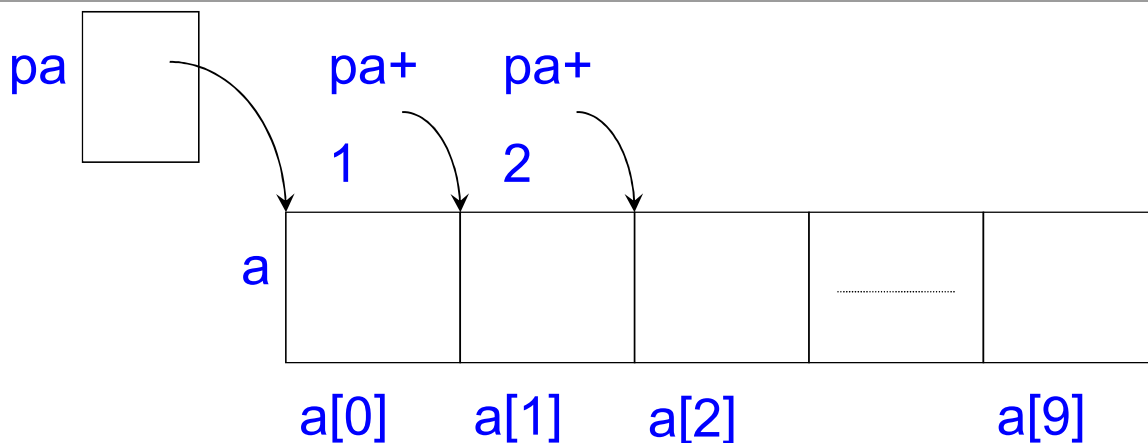
$pa = \&a[0];$ หรือใช้คำสั่ง $pa = a;$

pa จะเก็บค่าแอดเดรสเริ่มต้นของอาร์เรย์ a



23

การใช้พอยน์เตอร์กับอาร์เรย์



การอ้างอิงตำแหน่งในอาร์เรย์ผ่านตัวชี้

เช่น ต้องการคำนวณตำแหน่ง $a[2]$ ผ่านพอยน์เตอร์ก็คือ $pa + 2$

การเข้าถึงสมาชิกในอาร์เรย์กระทำได้โดยผ่านตัวกระทำ $*pa$

เช่น $x = *(pa+2)$

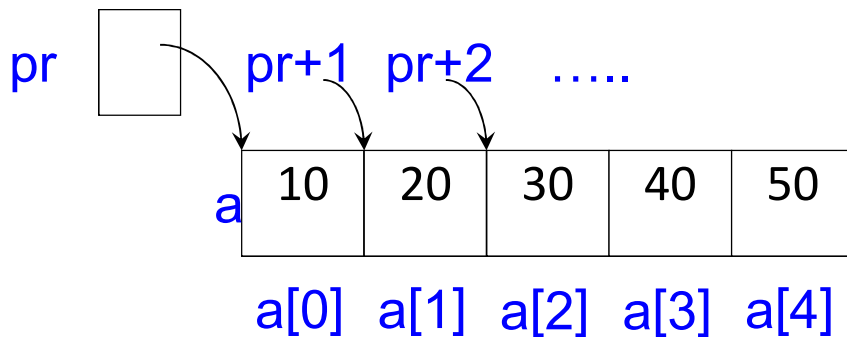
24

การอ้างอิงตำแหน่งในอาร์เรย์ผ่านตัวชี้

```
#include <stdio.h>
```

```
void main () {  
    int i, a[5] = {10, 20, 30, 40, 50},  
    int *pr;  
    pr = a;          // pr = &a[0];  
    for (i=0; i<5; i++)  
        printf("element %d data is %d\n", i+1, *(pr+i));  
}
```

element 1 data is 10
element 2 data is 20
element 3 data is 30
element 4 data is 40
element 5 data is 50



25

การส่งผ่านอาร์เรย์แบบ pass by reference

```
#include <stdio.h>
```

```
void print(int *pr)  
{ for (i=0; i<5; i++)  
    printf("element %d data is  
    %d\n", i+1, *(pr+i));  
}
```

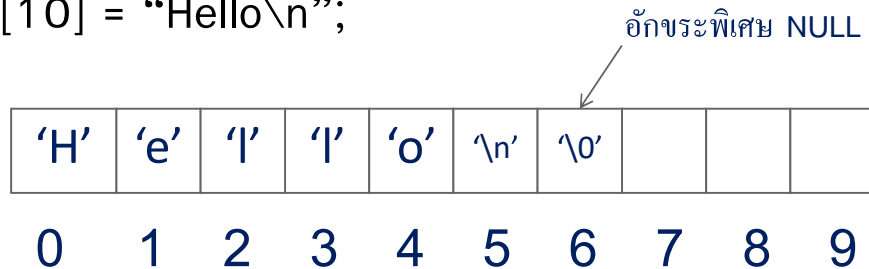
```
void main () {  
    int i, data[5] = {10, 20, 30, 40, 50};  
    print(data);          //  
    &data[0]  
}
```

26

สายอักขระ (String)

- อาร์เรย์ของอักขระ (character) ที่เรียงต่อกันเป็นคำหรือเป็นข้อความจะถูกเรียกว่า **สตริง (string)**

เช่น `char text[10] = "Hello\n";`



- ยิ่งไปกว่านั้น เรา**ไม่จำเป็นต้องระบุขนาดของอาร์เรย์**โดยการกำหนดค่าเริ่มต้น

`char text[] = "Hello\n";`

27

สายอักขระ (String)

- เนื่องจากอาร์เรย์ก็เปรียบเสมือน **pointer** ดังนั้นสตริงก็คือ **พอยน์เตอร์** ที่มีชนิดเป็น **character**

`char *text = "Hello\n";`

`char *name;`

- ข้อสังเกตค่า `char i='s'` และ `char *i="s"` มีความแตกต่างกัน เนื่องจาก 's' จะใช้ 1 byte แต่ "s" จะใช้ 2 bytes

28

ฟังก์ชันสำคัญ ๆ เกี่ยวกับสตริง

strlen(s)	คำนวณความยาวของสตริง s
strcpy(s1, s2)	คัดลอกสตริง s2 ไปที่สตริง s1
strcmp(s1, s2)	เปรียบเทียบ s1 และ s2 เท่ากันหรือไม่
strlwr(s)	เปลี่ยนอักขระในสตริง s ให้เป็นตัวเล็ก
strupr(s)	เปลี่ยนอักขระในสตริง s ให้เป็นตัวใหญ่

29

การอ้างถึงตำแหน่งในอาร์เรย์ผ่านตัวชี้

- เมื่อมีการส่งชื่อของอาร์เรย์ให้แก่ฟังก์ชัน จะเป็นการส่งตำแหน่งแอดเดรสของสมาชิกตัวแรกของอาร์เรย์ให้แก่ฟังก์ชัน ดังนั้นพารามิเตอร์ในฟังก์ชันนั้นจะเป็นตัวแปรประเภทพอยน์เตอร์

- ฟังก์ชันที่รับพารามิเตอร์เป็นพอยน์เตอร์ โดยอาร์กิวเมนต์ที่ส่งมาเป็นอาร์เรย์

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

main()
{
    char *s="hello";
    printf ("%d", strlen(s));
}
```

30

ฟังก์ชัน strlen() ปรับปรุงให้กระชับขึ้น

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

- เนื่องจาก s ซื่ออยู่ที่ตำแหน่งเริ่มต้น โดยมี p ชี้ไปที่ s เช่นเดียวกัน แต่จะมีการเลื่อน p ไปทีละหนึ่งตำแหน่ง จนกว่าค่าที่ตำแหน่งที่ p ซื่ออยู่จะเท่ากับ '\0' เมื่อนำ p ค่าสุดท้ายมาลบกับ s ที่ตำแหน่งเริ่มต้นก็จะได้ความยาวของข้อมูลที่ส่งเข้ามา

31

ฟังก์ชัน strcpy

- ฟังก์ชัน strcpy () ทำหน้าที่สำเนาข้อความจากตัวแปรหนึ่งไปยังอีกตัวแปรหนึ่งเขียนในลักษณะอาร์เรย์
- ฟังก์ชัน strcpy () เขียนในลักษณะพอยน์เตอร์แบบสั้น

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

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

- ฟังก์ชัน strcpy () เขียนในลักษณะพอยน์เตอร์

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

32

การประกาศตัวแปรชี้ (pointer) ชี้ไปยัง struct

- กรณีการส่งอากิวเมนต์เป็นตัวแปร struct จะไม่เหมาะกับ struct ที่มีขนาดใหญ่ เนื่องจากทุกครั้งที่ส่งตัวแปร struct จะเป็นการสำเนาตัวแปรตัวใหม่ขึ้นมาในฟังก์ชัน ซึ่งจะทำให้ช้าและเปลืองพื้นที่หน่วยความจำ เราจะใช้พอยน์เตอร์เข้ามาช่วยแก้ปัญหานี้
- โดยส่งแอดเดรสของตัวแปร struct มายังฟังก์ชันซึ่งรับอากิวเมนต์ เป็นพอยน์เตอร์ อากิวเมนต์จะชี้ไปยังแอดเดรสเริ่มต้นของตัวแปร struct จะช่วยให้การทำงานเร็วขึ้นและเปลืองหน่วยความจำน้อยลง แต่สิ่งที่ต้องระวังคือ หากมีการเปลี่ยนแปลงค่าที่อากิวเมนต์พอยน์เตอร์ชี้อยู่ ค่าในตัวแปร struct ที่ส่งมายังฟังก์ชันจะเปลี่ยนตามโดยอัตโนมัติ

33

การประกาศตัวแปรชี้ (pointer) ชี้ไปยัง struct

```
struct point{
    int x;
    int y;
};

struct point origin, *pp;

void main() {
    pp = &origin;
    (*pp).x = 10;           // pp->x = 10;
    (*pp).y = 20;           // pp->y = 20;
    printf( "origin is (%d,%d)\n", (*pp).x, (*pp).y );
}
```

34

การประกาศตัวแปรชี้ (pointer) ชี้ไปยัง struct

- การอ้างถึงสมาชิกอาจเขียนอีกลักษณะหนึ่งโดยใช้เครื่องหมาย
-> สมมติ `p` เป็นพอยน์เตอร์ รูปแบบการใช้เป็นดังนี้

`p->memberOfStructure`

จะสามารถแปลงประโยคการใช้พอยน์เตอร์อ้างสมาชิกของ struct จากตัวอย่างข้างบนได้ว่า

```
printf ( "origin is (%d, %d)\n", pp->x, pp->y);
```

35

การประกาศตัวแปรชี้ (pointer) ชี้ไปยัง struct

```
typedef struct {  
    int    day;  
    int    month;  
    int    year;  
} Date;
```

แบบที่ 1

การประกาศแบบ
ข้อมูลโครงสร้าง

```
Date    today;
```

การประกาศตัวแปร
ข้อมูลแบบโครงสร้าง

```
Date    *ptrdate;
```

การประกาศตัวแปร
pointer ชี้ไปยังโครงสร้าง

36

การประกาศตัวแปรชี้ (pointer) ชี้ไปยัง struct

```
struct date {  
    int day;  
    int month;  
    int year;  
} *ptrdate;
```

แบบที่ 2

37

การประกาศตัวแปรชี้ (pointer) ชี้ไปยัง struct

```
typedef struct {  
    int day;  
    int month;  
    int year;  
} Date;  
  
typedef Date *PtrDate;  
  
PtrDate ptrdate;
```

แบบที่ 3

การประกาศแบบ
ข้อมูลโครงสร้าง

การประกาศประเภท
ตัวแปร pointer ชี้ไปยัง
โครงสร้าง

การประกาศตัวแปร
pointer ชี้ไปยัง โครงสร้าง

38

```

#include <stdio.h>
struct date {
    int    day;
    int    month;
    int    year;
};
typedef struct    date    Date;
typedef    Date    *PtrDate;
main ()    {
    Date        today;
    PtrDate    ptrdate;
    ptrdate = &today;
    ptrdate->day        = 27;
    ptrdate->month    = 9;
    ptrdate->year        = 1985;
    printf("Today\'s date is %2d/%2d/%4d\n",
ptrdate->day,ptrdate->month,ptrdate->year );
}

```