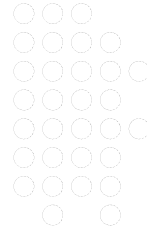


# 第十章 指標

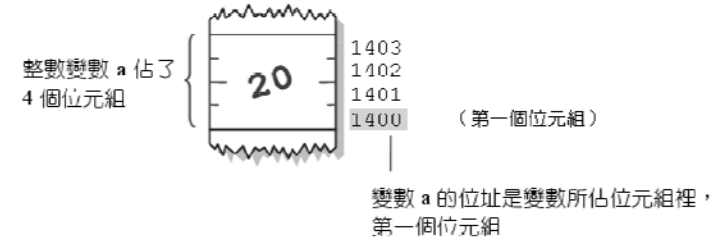
記憶體位址與指標  
間接參考運算子的用法  
指標與函數參數  
指標與陣列之間的關係



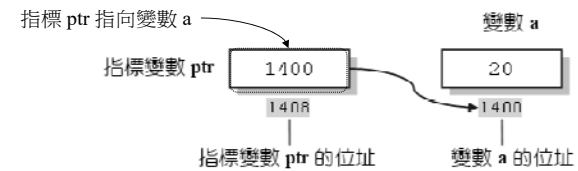
## 10.1 指標概述

### 變數的位址

- 變數的位址是它所佔位元組裡，第一個位元組的位址：



- 指標變數是用來存放變數在記憶體中的位址



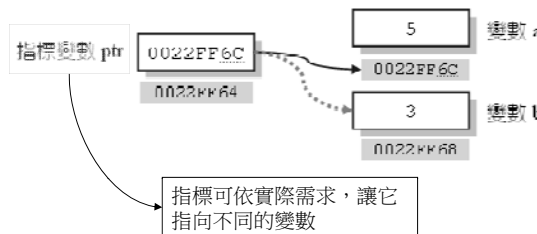
### 為什麼要用指標？

- 利用指標可以使得函數在傳遞陣列時更有效率

```
int showArray(int arr[]) {  
    ...  
}
```

傳遞陣列的位址，而非整個陣列，因而執行速度較快

- 可隨時改變所指之對象，因而可彈性地
  - 處理不同的變數
  - 呼叫不同的函數



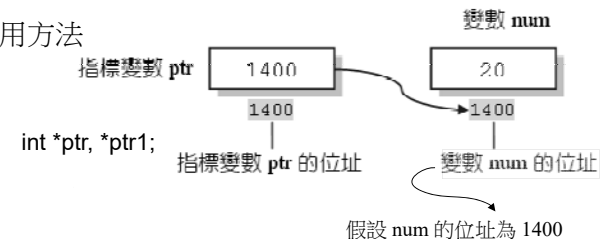
## 10.2 使用指標變數

### 指標變數的宣告

資料型態 \* 指標變數;  
或  
資料型態\* 指標變數;

- 指標變數的使用方法

- int num=20;
- int \*ptr;
- ptr=&num;

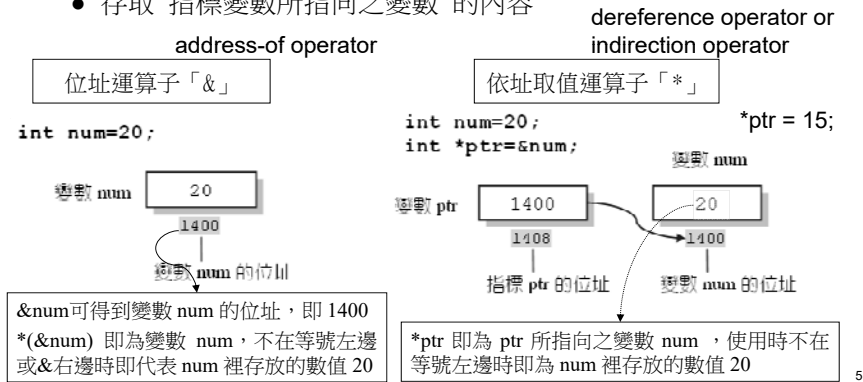


第二行與第三行敘述也可以合併運用「變數的初始化」語法寫成：  
int \*ptr=&num;

# 指標變數的使用

● 指標變數常見的用法有兩種：

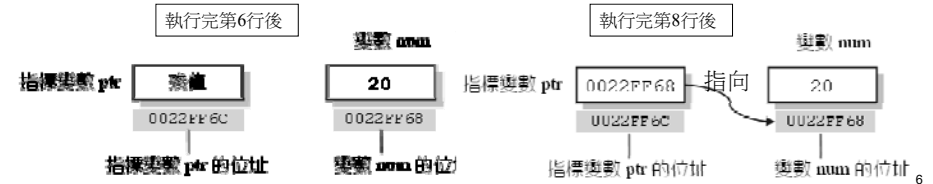
- 存取 指標變數裡 某一變數的位址
- 存取 指標變數所指向之變數 的內容



# 指標變數的使用範例 (1/2)

```

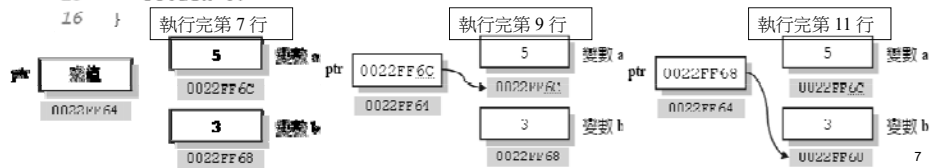
01 /* prog10_2, 指標變數的宣告 */ /* prog10_2 OUTPUT-----
02 #include <stdio.h>
03 #include <stdlib.h>
04 int main(void)
05 {
06     int *ptr, num=20;
07
08     ptr=&num;
09     printf("num=%d, &num=%p\n", num, &num);
10     printf("^ptr=%d, ptr=%p, *ptr=%p\n", *ptr, ptr, *ptr);
11     system("pause");
12     return 0;
13 }
    
```



# 指標變數的使用範例 (2/2)

```

01 /* prog10_3, 指標變數的使用 */
02 #include <stdio.h> /* prog10_3 OUTPUT
03 #include <stdlib.h>
04 int main(void)
05 {
06     int a=5, b=3;
07     int *ptr;
08
09     ptr=&a; /* 將 a 的位址設給指標 ptr 存放 */
10     printf("&a=%p, &ptr=%p, ptr=%p, *ptr=%d\n", &a, &ptr, ptr, *ptr);
11     ptr=&b; /* 將 b 的位址設給指標 ptr 存放 */
12     printf("&b=%p, &ptr=%p, ptr=%p, *ptr=%d\n", &b, &ptr, ptr, *ptr);
13
14     system("pause");
15     return 0;
16 }
    
```



# 指標變數所佔的位元組

● 查詢指標變數所佔的位元組：/\* prog10\_4 OUTPUT

```

01 /* prog10_4, 指標變數的大小 */
02 #include <stdio.h>
03 #include <stdlib.h>
04 int main(void)
05 {
06     int *ptri; /* 宣告指向整數的指標 ptri */
07     char *ptrc; /* 宣告指向字元的指標 ptrc */
08
09     printf("sizeof(ptri)=%d\n", sizeof(ptri));
10     printf("sizeof(ptrc)=%d\n", sizeof(ptrc));
11     printf("sizeof(*ptri)=%d\n", sizeof(*ptri));
12     printf("sizeof(*ptrc)=%d\n", sizeof(*ptrc));
13
14     system("pause");
15     return 0;
16 }
    
```

sizeof(ptri)=4 } 指標變數皆佔了 4 個位元組  
sizeof(ptrc)=4 }  
sizeof(\*ptri)=4  
sizeof(\*ptrc)=1

## 指標操作的練習 (1/2)

```

01  /* prog10_5, 指標的操作練習 */          /* prog10_5 OUTPUT-----
02  #include <stdio.h>                        a=64, b=69, *ptr1=64, *ptr2=64
03  #include <stdlib.h>                      ptr1=0022FF6C, ptr2=0022FF6C
04  int main(void)                            -----*/
05  {
06      int a=b, b=10;
07      int *ptr1, *ptr2;
08      ptr1=&a;                               /* 將 ptr1 設為 a 的地址 */
09      ptr2=&b;                               /* 將 ptr2 設為 b 的地址 */
10      *ptr1=7;                              /* 將 ptr1 指向的內容設為 7 */
11      *ptr2=32;                             /* 將 ptr2 指向的內容設為 32 */
12      a=17;                                  /* 設定 a 為 17 */
13      ptr1=ptr2;                             /* 設定 ptr1=ptr2 */
14      *ptr1=9;                              /* 將 ptr1 指向的內容設為 9 */
15      ptr1=&a;                               /* 將 ptr1 設為 a 的地址 */
16      a=64;                                  /* 設定 a 為 64 */
17      *ptr2=*ptr1+5;                        /* 將 ptr2 指向的內容設為 *ptr1+5 */
18      ptr2=&a;                               /* 將 ptr2 設為 a 的地址 */
19      printf("a=%2d, b=%2d, *ptr1=%2d, *ptr2=%2d\n", a, b, *ptr1, *ptr2);
20      printf("ptr1=%p, ptr2=%p\n", ptr1, ptr2);
21      system("pause");
22      return 0;
23  }

```

9

## 指標操作的練習 (2/2)

執行6~18行時，變數變化的情形 (&amp;a=FF6C, &amp;b= FF68)

行號	程式碼	a	b	ptr1	*ptr1	ptr2	*ptr2
06	int a=5, b=10;	5	10				
07	int *ptr1, *ptr2;	5	10	殘值	殘值	殘值	殘值
08	ptr1=&a;	5	10	FF6C	5	殘值	殘值
09	ptr2=&b;	5	10	FF6C	5	FF68	10
10	*ptr1=7;	7	10	FF6C	7	FF68	10
11	*ptr2=32;	7	32	FF6C	7	FF68	32
12	a=17;	17	32	FF6C	17	FF68	32
13	ptr1=ptr2;	17	32	FF68	32	FF68	32
14	*ptr1=9;	17	9	FF68	9	FF68	9
15	ptr1=&a;	17	9	FF6C	17	FF68	9
16	a=64;	64	9	FF6C	64	FF68	9
17	*ptr2=*ptr1+5;	64	69	FF6C	64	FF68	69
18	ptr2=&a;	64	69	FF6C	64	FF6C	64

10

## 當指標指向錯誤的型態時

- 指標指向不正確的型態所產生的錯誤：

```

01  /* prog10_6, 錯誤的指標型態 */          /* prog10_6 OUTPUT
02  #include <stdio.h>                        sizeof(a1)-4
03  #include <stdlib.h>                      sizeof(a2)-4
04  int main(void)                          a1=100, *ptr1=-1717986918
05  {                                        a2=3.2, *ptrf=0.0
06      int a1=100, *ptr1;
07      float a2=3.2f, *ptrf;
08      ptr1=&a2; /* 錯誤，將 int 型態的指標指向 float 型態的變數 */
09      ptrf=&a1; /* 錯誤，將 float 型態的指標指向 int 型態的變數 */
10      printf("sizeof(a1)=%d\n", sizeof(a1));
11      printf("sizeof(a2)=%d\n", sizeof(a2));
12      printf("a1=%d, *ptr1=%d\n", a1, *ptr1);
13      printf("a2=%.1f, *ptrf=%.1f\n", a2, *ptrf);
14      system("pause");
15      return 0;
16  }

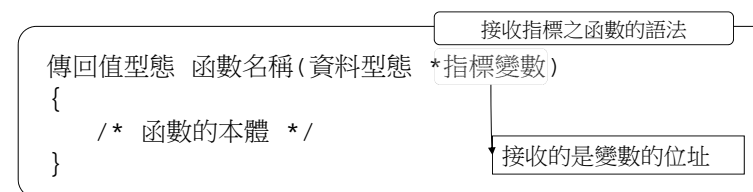
```

incompatible pointer assignment:  
dev C++ 4.9.9.2: warnings only  
vc2010, dev C++ 5.8.3 and later: errors

11

## 傳遞指標到函數 (1/3)

- 接收指標的函數：



```

int main(void)
{
    int num=5;
    func(&num);
    ...
}

void func(int *ptr)
{
    /* 函數的本體 */
}

```

接收位址的變數 ptr

傳遞 num 變數的地址

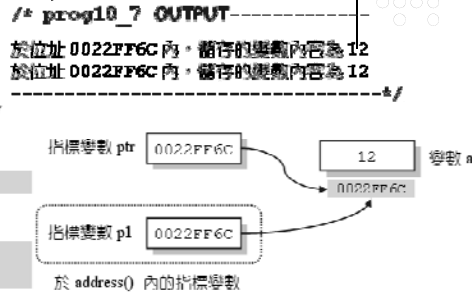
12

# 傳遞指標到函數 (2/3)

## ● 傳遞指標到函數

```

01 /* prog10_7, 傳遞指標到函數裡 */
02 #include <stdio.h>
03 #include <stdlib.h>
04 void address(int *);
05 int main(void)
06 {
07     int a=12;
08     int *ptr=&a;
09
10     address(&a); /* 將 a 的地址傳入 address() 函數中 */
11     address(ptr); /* 將 ptr 傳入 address() 函數中 */
12
13     system("pause");
14     return 0;
15 }
16 void address(int *p1)
17 {
18     printf("於位址%p內，儲存的變數內容為%d\n",p1,*p1);
19 }
    
```



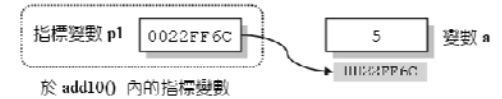
# 傳遞指標到函數 (3/3)

## ● 傳遞指標的應用：

```

01 /* prog10_8, 傳遞指標的應用 */
02 #include <stdio.h>
03 #include <stdlib.h>
04 void add10(int *);
05 int main(void)
06 {
07     int a=5;
08     printf("呼叫 add10 () 之前,a=%d\n",a);
09     add10(&a); /* 呼叫 add10() 函數 */
10     printf("呼叫 add10 () 之後,a=%d\n",a);
11     system("pause");
12     return 0;
13 }
14 void add10(int *p1)
15 {
16     *p1=*p1+10; // 透過指標變數p1間接存取變數 a
17 }
    
```

/\* prog10\_8 OUTPUT---  
呼叫 add10 () 之前, a=5  
呼叫 add10 () 之後, a=15  
-----\*/



# 變數值的互換 (錯誤)

```

01 /* prog10_9, 將 a 與 b 值互換 (錯誤示範) */
02 #include <stdio.h>
03 #include <stdlib.h>
04 void swap(int *,int *);
05 int main(void)
06 {
07     int a=5,b=20;
08     printf("交換前... ");
09     printf("a=%d,b=%d\n",a,b);
10     swap(a,b);
11     printf("交換後... ");
12     printf("a=%d,b=%d\n",a,b);
13
14     system("pause");
15     return 0;
16 }
17
18 void swap(int *x,int *y)
19 {
20     int tmp=x;
21     x=y;
22     y=tmp;
23 }
    
```

/\* prog10\_9 OUTPUT---  
交換前... a=5,b=20  
交換後... a=5,b=20  
-----\*/

a	b	x	y	tmp
5	20	/	/	/
5	20	5	20	??
5	20	5	20	5
5	20	20	20	5
5	20	20	5	5

# 變數值的互換 (正確)

```

01 /* prog10_10, 將 a 與 b 值互換 (正確範例) */
02 #include <stdio.h>
03 #include <stdlib.h>
04 void swap(int *,int *);
05 int main(void)
06 {
07     int a=5,b=20;
08     printf("交換前... ");
09     printf("a=%d,b=%d\n",a,b);
10     swap(&a,&b);
11     printf("交換後... ");
12     printf("a=%d,b=%d\n",a,b);
13
14     system("pause");
15     return 0;
16 }
17
18 void swap(int *p1,int *p2)
19 {
20     int tmp=*p1;
21     *p1=*p2;
22     *p2=tmp;
23 }
    
```

/\* prog10\_10 OUTPUT---  
交換前... a=5,b=20  
交換後... a=20,b=5  
-----\*/

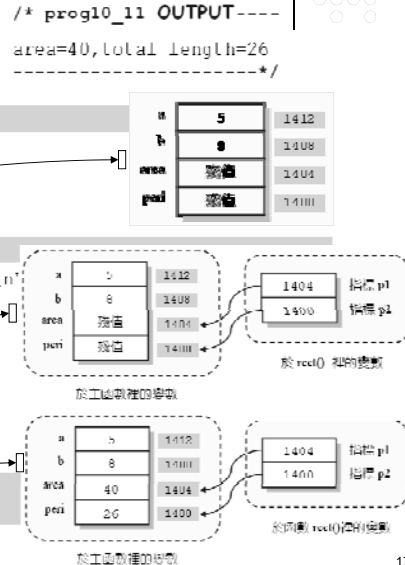
a	b	p1	p2	tmp
5	20	/	/	/
5	20	&a	&b	??
5	20	&a	&b	5
20	20	&a	&b	5
20	5	&a	&b	5

# 傳回多個數值的函數

```

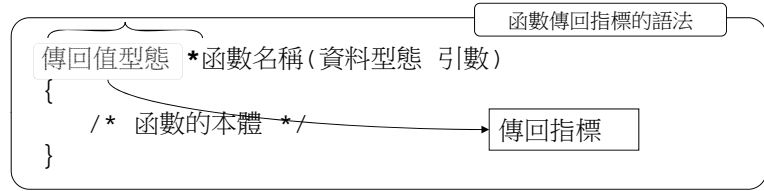
01 /* prog10_11, 傳回多個數值的函數 */
02 #include <stdio.h>
03 #include <stdlib.h>
04 void rect(int, int, int *, int *);
05 int main(void)
06 {
07     int a=5, b=8;
08     int area, peri;
09     rect(a, b, &area, &peri);
10     printf("area=%d, total length=%d\n",
11           area, peri);
12     system("pause");
13     return 0;
14 }
15 void rect(int x, int y, int *p1, int *p2)
16 {
17     *p1=x*y;
18     *p2=2*(x+y);
19 }

```



# 由函數傳回指標 (1/2)

- 傳回指標的函數：



```

int main(void)
{
    int *ptr, num;
    ptr=func(num);
    ...
}

int *func(int num)
{
    /* 函數的本體 */
    return 整數的指標;
    ...
}

```

# 由函數

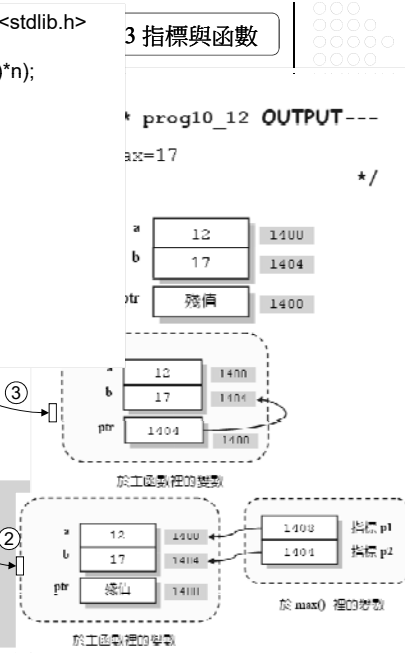
```

#include <malloc.h> // or #include <stdlib.h>
int *fun(int n) {
    int *ptr = (int *) malloc(sizeof(int)*n);
    ...
    return ptr;
}

int main() {
    int i, *array, n=15;
    array = fun(n);
    for (i=0; i<n; i++)
        array[i] = i; // *(array+i) = i;
    if (array) free(array);
}

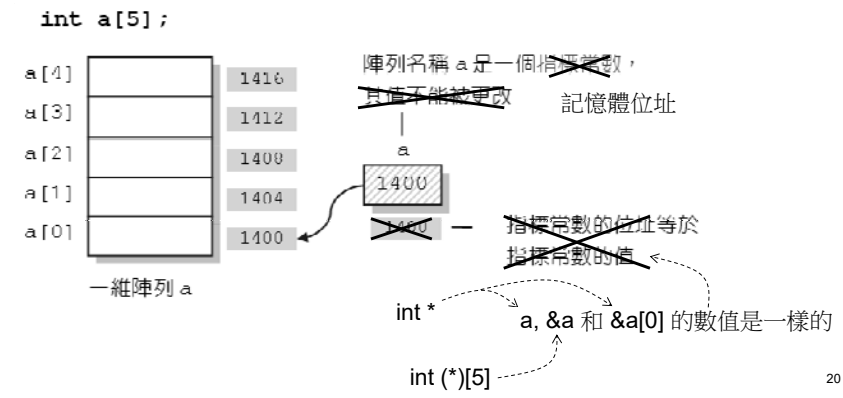
int *max(int *p1, int *p2) {
    if (*p1 > *p2)
        return p1;
    else
        return p2;
}

```



# 指標與一維陣列

- 陣列的名稱是一個記憶體位址，是該陣列第一個元素的記憶體位址，並不是一個指標常數 (內容不能更改的指標變數)

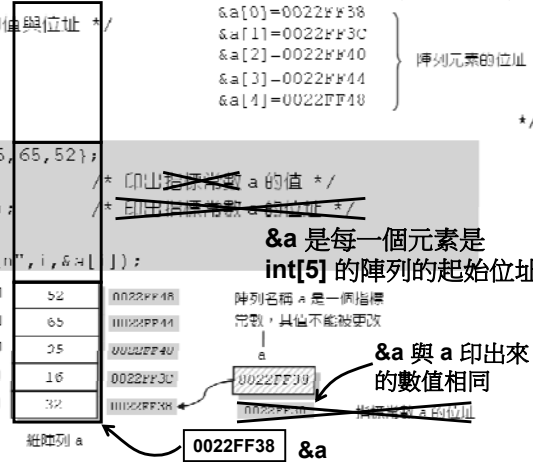


# 陣列名稱即陣列的位址

- 驗證陣列名稱是一個指標常數：

```
01 /* prog10_13, 指標常數的值與位址 */
02 #include <stdio.h>
03 #include <stdlib.h>
04 int main(void)
05 {
06     int i, a[5]={32,16,35,65,52};
07     printf("a=%p\n", a);
08     printf("a=%p\n", &a);
09     for(i=0; i<5; i++)
10         printf("%a[%d]=%p\n", i, &a[i]);
11     system("pause");
12     return 0;
13 }
```

```
/* prog10_13 OUTPUT
a=0022FF38
&a=0022FF38
&a[0]=0022FF38
&a[1]=0022FF3C
&a[2]=0022FF40
&a[3]=0022FF44
&a[4]=0022FF48
```



a 是陣列裡第一個元素的記憶體位址，不是一個變數，&a 不是那個變數的位址，&a 是右邊藍色陣列第一個元素的位址

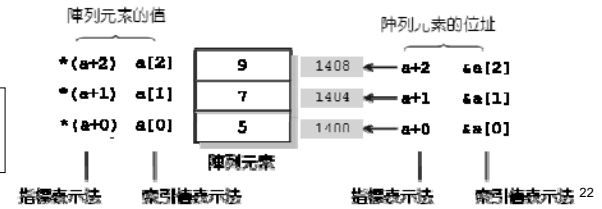
# 指標的算術運算 (1/3)

- 利用指標存取陣列的內容

```
01 /* prog10_14, 利用指標常數來存取陣列的內容 */
02 #include <stdio.h>
03 #include <stdlib.h>
04 int main(void)
05 {
06     int a[3]={5,7,9};
07     printf("a[0]=%d, *(a+0)=%d\n", a[0], *(a+0));
08     printf("a[1]=%d, *(a+1)=%d\n", a[1], *(a+1));
09     printf("a[2]=%d, *(a+2)=%d\n", a[2], *(a+2));
10     system("pause");
11     return 0;
12 }
```

double \*ptr; ptr+5 時將存放在變數 ptr 裡的記憶體位址加上 5 \* sizeof(double) \*(ptr+5) 和 ptr[5] 一模一樣

如果指標 a 指向某一個陣列，則 a+i 指向陣列裡索引值為 i 的元素。



# 指標的算術運算 (2/3)

- 利用指標計算一維陣列內所有元素的和

```
01 /* prog10_15, 利用指標求陣列元素和 */
02 #include <stdio.h>
03 #include <stdlib.h>
04 int main(void)
05 {
06     int a[3]={5,7,9};
07     int i, sum=0;
08     for(i=0; i<3; i++)
09         sum+=*(a+i);
10     printf("sum=%d\n", sum);
11     system("pause");
12     return 0;
13 }
```

Funny thought:  
a[i] 等效於 \*(a+i)  
\*(a+i) 等效於 \*(i+a)  
那麼 i[a] 是什麼東西?

```
/* prog10_15 OUTPUT--
```

sum=21

\*/

# 指標的算術運算 (3/3)

- 改以指標變數 ptr 來指向陣列 a，並計算總和：

```
01 /* prog10_16, 利用指標變數 ptr 來指向陣列 a，並計算總和 */
02 #include <stdio.h>
03 #include <stdlib.h>
04 int main(void)
05 {
06     int a[3]={5,7,9};
07     int i;
08     int *ptr=a;
09     for(i=0; i<3; i++)
10         sum+=*(ptr++);
11     printf("sum=%d\n", sum);
12     system("pause");
13     return 0;
14 }
```

a++ a--	Suffix/postfix Increment and decrement
type() type{}	Functional cast
a()	Function call
a[i]	Subscript
. ->	Member access
++a --a	Prefix increment and decrement
+a -a	Unary plus and minus
! ~	Logical NOT and bitwise NOT
(type)	C-style cast
*a	Indirection (dereference)
&a	Address of

codes like ptr++ are not recommended

for (i=0; i<3; i++) sum+=a[i];

for (i=0; i<3; i++) sum+=\*ptr++;

for (i=0; i<3; i++) sum+=\*(ptr)++;

```
/* prog10_16 OUTPUT--
```

sum=21

-----\*/

## 傳遞一維陣列到函數

```

01 /* prog10_17, 將陣列第 n 個元素的值
02 #include <stdio.h>
03 #include <stdlib.h>
04 void replace(int *,int,int);
05 int main(void)
06 {
07     int a[5]={13,32,67,14,95};
08     int i,num=24;
09
10     replace(a,4,num); /* 呼叫函數 replace() */
11     printf("置換後,陣列的內容為");
12     for(i=0;i<5;i++) /* 印出陣列的內容 */
13         printf("%3d",a[i]);
14     printf("\n"); /* prog10_17 OUTPUT-----
15     system("pause"); 置換後,陣列的內容為 13 32 67 24 95
16     return 0; -----*/
17 }
18 void replace(int *ptr,int n,int num)
19 {
20     *(ptr+n-1)=num; /* 將陣列第 n 個元素設值為 num */
21 }

```

ptr[n-1]=num;

25

## 函數傳回指標

```

01 /* prog10_18, 函數傳回值為指標 */
02 #include <stdio.h>
03 #include <stdlib.h>
04 #define SIZE 5
05 int *maximum(int *); /* 宣告 maximum() 函數的原型 */
06 int main(void)
07 {
08     int a[SIZE]={0,1,7,2,6}; /* prog10_18 OUTPUT--
09     int i,*ptr; array n=3 1 7 2 6
10     printf("array a="); maximum=7
11     for(i=0;i<SIZE;i++) -----*/
12         printf("%3d",a[i]);
13     ptr=maximum(a); /* 呼叫 maximum() 函數, 並傳入陣列 a */
14     printf("\nmaximum=%d\n",*ptr);
15     system("pause");
16     return 0;
17 }
18
19 int *maximum(int *arr) /* 定義 maximum() 函數 */
20 {
21     int i,*max;
22     max=arr; /* 設定指標 max 指向陣列的第 0 個元素 */
23     for(i=1;i<SIZE;i++)
24         if(*max < *(arr+i))
25             max=arr+i;
26     return max; /* 傳回最大值之元素的位址 */
27 }

```

26

## 以指標變數指向字串 (1/2)

- 利用字元陣列來儲存字串：

```
char str[]="How are you";
```



str 是一個指標常數, 其值不能被更改

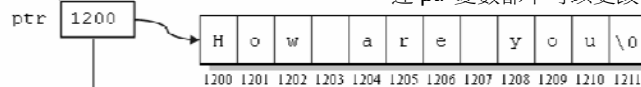
字元變數 str[i] 可以更改

- 利用字元常數的指標指向字串：

```
const char *ptr="How are you";
```

```
const char *const ptr="How are you";
```

連 ptr 變數都不可以更改



ptr 是一個指標變數, 其值可以被更改 常數字元變數 ptr[i] 不可以更改

27

## 以指標變數指向字串 (2/2)

- 以指標變數指向字串的範例：

```

01 /* prog10_19, 以指標變數指向字串 */
02 #include <stdio.h>
03 #include <stdlib.h>
04 int main(void)
05 {
06     char name[20];
07     const char *ptr="How are you?"; /* 將指標指向字串 "How are you?" */
08     printf("what's your name? ");
09     gets(name); /* 由鍵盤讀入字串 */
10     printf("Hi, %s, ",name); /* 印出字串陣列 name 的內容 */
11     puts(ptr); /* 印出由 ptr 所指向的字串 */
12
13     system("pause"); /* prog10_19 OUTPUT---
14     return 0; what's your name? Wien
15 } li, Wien, How are you?
-----*/

```

28

## 指標陣列 (1/3)

- 一維指標陣列的宣告格式：

宣告指標陣列

資料型態 \*陣列名稱 [元素個數];

- 以二維的字元陣列來儲存字串陣列：

```
char str[3][10]={"Tom", "Lily", "James Lee"};
```



浪費掉的空間

- 以指標陣列的方式來撰寫：

```
char *ptr[3]={"Tom", "Lily", "James Lee"};
```



空間不浪費

29

## 指標陣列 (2/3)

- 指標陣列的範例：

```

/* prog10_20 OUTPUT--
Tom
Lily
James Lee
-----*/

01 /* prog10_20, 指標陣列 */
02 #include <stdio.h>
03 #include <stdlib.h>
04 int main(void)
05 {
06     int i;
07     char *ptr[3]={"Tom", "Lily", "James Lee"};
08     for(i=0; i<3; i++)
09         puts(ptr[i]); /* 印出指標 ptr[i] 所指向的字串 */
10
11     system("pause");
12     return 0;
13 }

```

30

## 指標陣列 (3/3)

```
#include <stdlib.h> // qsort
#include <string.h> // strcmp
```

```
char flowers[][20] = {"tulip", "marigold", "petunia", "rose", "daisy"};
qsort(flowers, 5, 20, (int (*)(const void *, const void *))strcmp);
```

```

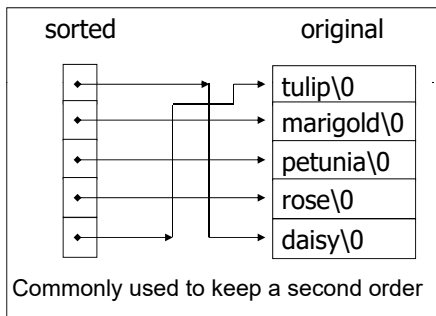
char *flowerSorted[5];
for (i=0; i<5; i++)
    flowerSorted[i] = &flowers[i][0]; // flowers[i]
qsort(flowerSorted, 5, sizeof(char *), compare);

```

```

int compare(const void *a, const void *b) {
    return strcmp(*(char **)a, *(char **)b);
}

```



Commonly used to keep a second order

31

## 指向指標的指標 (1/3)

- 雙重指標存放指標變數的位址



雙重指標宣告的格式

資料型態 \*\*雙重指標;

- 宣告雙重指標的範例：

- int \*\*ptri;
- char \*\*ptrc;

32



## 指向指標的指標 (2/3)

- 雙重指標的範例：

```

01 /* prog10_21, 雙重指標的範例 */
02 #include <stdio.h>
03 #include <stdlib.h>
04 int main(void)
05 {
06     int n=20, *p, **pp;
07     p=&n;
08     pp=&p;
09     printf("n=%d, &n=%p, *p=%d, p=%p, &p=%p\n", n, &n, *p, p, &p);
10     printf("^ ^ pp=%p, ^ pp=%p, pp=%p, &pp=%p\n", ^ ^ pp, ^ pp, pp, &pp);
11
12     system("pause"); /* prog10_21 OUTPUT-----
13     return 0;
14 }

```

雙重指標: pp (0022FF68) at 0022FF64  
 指標變數: p (0022FF6C) at 0022FF68  
 般變數: n (20) at 0022FF6C

```

n=20, &n=0022FF6C, *p=20, p=0022FF6C, &p=0022FF68
**pp=20, *pp=0022FF6C, pp=0022FF68, &pp=0022FF64
-----*/

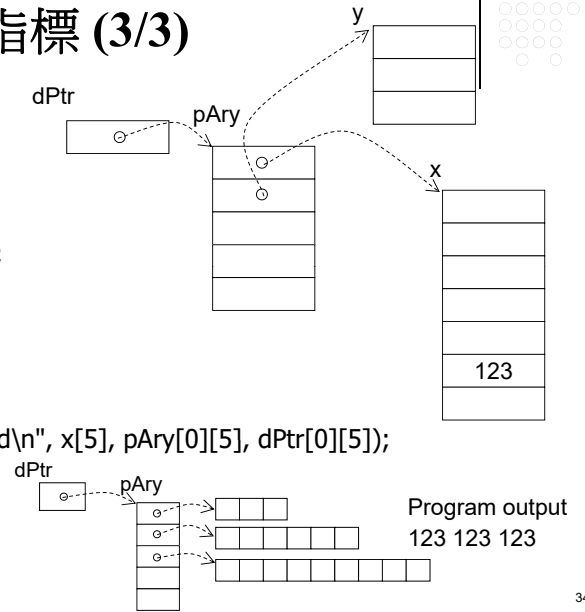
```

## 指向指標的指標 (3/3)

```

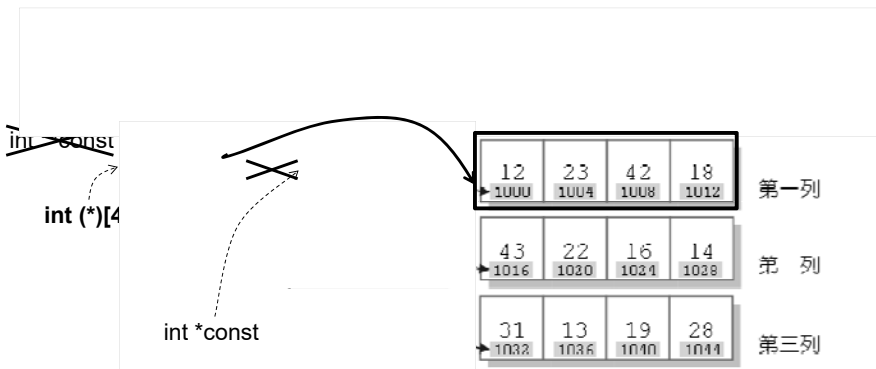
01 #include <stdio.h>
02 #include <stdlib.h>
03 int main(void) {
04     int *pAry[5];
05     int **dPtr = pAry;
06     int x[7], y[3];
07     pAry[0] = x;
08     pAry[1] = y;
09     x[5] = 123;
10     printf("%d %d %d\n", x[5], pAry[0][5], dPtr[0][5]);
11     system("pause");
12     return 0;
13 }

```



## 二維陣列與雙重指標的關係

`int num[3][4];`  
 • 由 3 個一維陣列所組成，每個一維陣列裡各有 4 個整數  
 • `num[0]`、`num[1]` 與 `num[2]` 為整數指標常數 `int *const`，它們分別指向這 3 個一維陣列，型態是 `int[4]`  
 • `num` 是 3 個元素的陣列，每個元素是 4 個整數的陣列，`num` 或是 `&num[0]` 是第一個元素的記憶體位址



## 驗證二維陣列與指標的關係

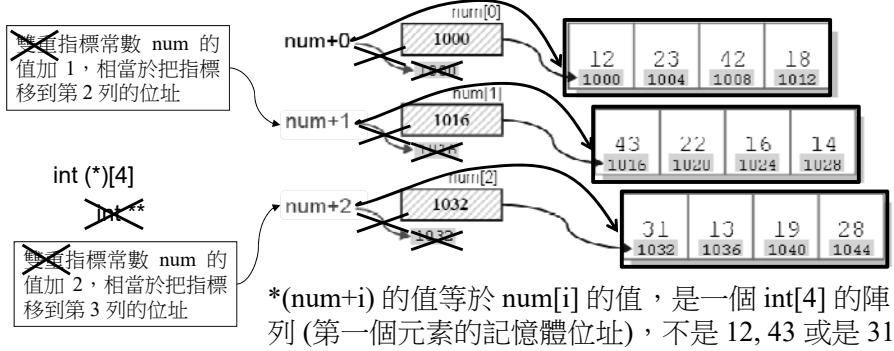
```

01 /* prog10_22, 印出陣列的位址 */
02 #include <stdio.h>
03 #include <stdlib.h>
04 int main(void)
05 {
06     int num[3][4];
07     printf("num=%p\n", num);
08     printf("&num=%p\n", &num);
09     printf("*num=%p\n", *num);
10
11     printf("num[0] %p\n", num[0]);
12     printf("num[1] %p\n", num[1]);
13     printf("num[2] %p\n", num[2]);
14
15     printf("&num[0] %p\n", &num[0]);
16     printf("&num[1] %p\n", &num[1]);
17     printf("&num[2] %p\n", &num[2]);
18     system("pause");
19     return 0;
20 }

```

/\* prog10\_22 OUTPUT--  
 num=0022FF38  
 &num=0022FF38  
 \*num=0022FF30  
 num[0]=0022FF38  
 num[1]=0022FF48  
 num[2]=0022FF58  
 &num[0]=0022FF38  
 &num[1]=0022FF48  
 &num[2]=0022FF58

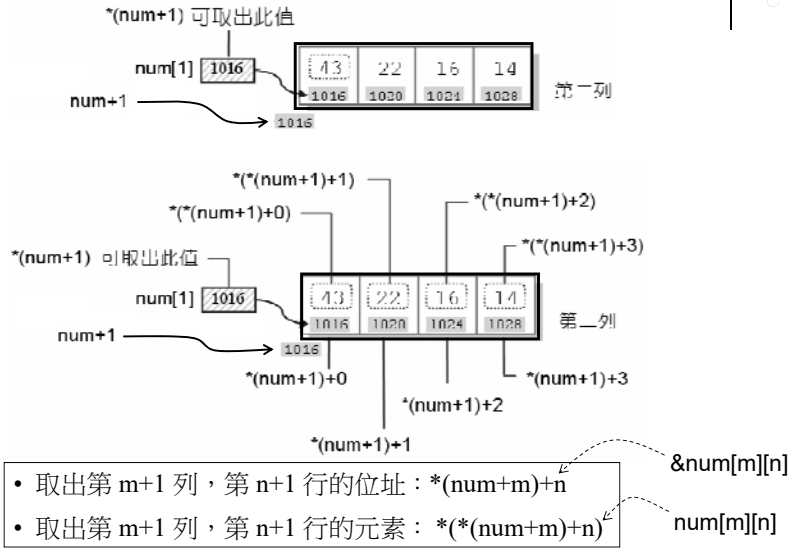
# 二維陣列的指標表示方式



如果 num 的型態是 int \*\*  
則 num+1 時指標只會加 sizeof(int\*)  
因為 num 的型態是 int (\*)[4]  
所以 num+1 時指標會加 sizeof(int [4])

```
int num[3][4];
fun(num);
...
void fun(int (*y)[4]) { ... }
void fun(int **y) { ... }
```

# 取得每一列裡特定的元素



# 以指標操作二維陣列 (1/2)

```
prog10_23 OUTPUT-----
num[0][0]=12, 位址-0022FF38
num[0][1]=23, 位址-0022FF3C
num[0][2]=42, 位址-0022FF40
num[0][3]=18, 位址-0022FF44
num[1][0]=43, 位址-0022FF48
num[1][1]=22, 位址-0022FF4C
num[1][2]=16, 位址-0022FF50
num[1][3]=14, 位址-0022FF54
num[2][0]=31, 位址-0022FF58
num[2][1]=13, 位址-0022FF5C
num[2][2]=19, 位址-0022FF60
num[2][3]=28, 位址-0022FF64
**num=12
```

```
01 /* prog10_23, 印出陣列的位址 */
02 #include <stdio.h>
03 #include <stdlib.h>
04 int main(void)
05 {
06     int num[3][4]={{12,23,42,18},
07                  {43,22,16,14},
08                  {31,13,19,28}};
09     int m,n;
10     for(m=0;m<3;m++)
11         for(n=0;n<4;n++)
12             printf("num[%d][%d]=%d, 位址-%p\n",m,n,*(*(num+m)+n),*(num+m)+n);
13     printf("***num=%d\n",**num);
14     system("pause");
15     return 0;
16 }
```

num 的位址

num 的位址

num 的位址

num 的位址

num[m][n] 的位址

num[m][n] 的位址

\*(num+0)+0

\*(num+0)+3

# 以指標操作二維陣列 (2/2)

```
prog10_24 OUTPUT----
12 23 40 18
40 22 16 14
31 13 19 28
```

```
01 /* prog10_24, 利用指標將人於 40 的陣列元素設置為 40 */
02 #include <stdio.h>
03 #include <stdlib.h>
04 int main(void)
05 {
06     int num[3][4]={{12,23,42,18},
07                  {43,22,16,14},
08                  {31,13,19,28}};
09     int m,n;
10     for(m=0;m<3;m++)
11     {
12         for(n=0;n<4;n++)
13         {
14             if(*(*(num+m)+n)>40) /* 判別 num[m][n] 的值是否人於 40 */
15                 *(*(num+m)+n)-40; /* 如果是，則將元素值設為 40 */
16             printf("%3d",*(*(num+m)+n)); /* 印出元素 num[m][n] 的值 */
17         }
18         printf("\n");
19     }
20     system("pause");
21     return 0;
22 }
```

A last word:  
Now that you know num[m][n] is equivalent to \*(num+m)+n. Do not use this latter form in your program!!