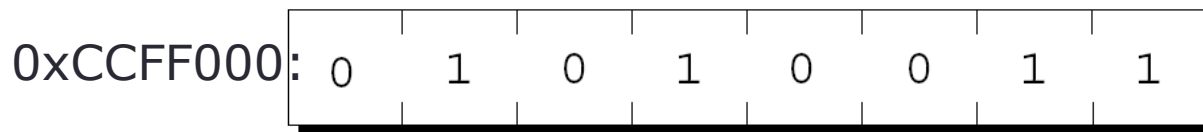


CHAPTER 7 - POINTERS

7.1 指標(Pointers)簡介

- 指標
 - 記憶體裡每個byte都有位置



- 可以用來模擬傳址(call-by-reference)的參數傳遞法
- 與陣列和字串有緊密的關係

指標簡介 (cont'd)

- 一般變數的內容存放的是某個特定數值

count

7

- 指標變數(Pointer variables)
 - 內容是記憶體位置
 - 某個變數的位置

countPtr

count

0x00000002

Address	Contents
0	01010011
1	01110101
2	01110011
3	01100001
4	01101110
	⋮
n-1	01000011

變數count

{ 2
3

7.2 宣告指標變數

- 指標變數宣告(Pointer declarations)

- 宣告一個指向整數的指標變數

- 加上*
- `int *myPtr;`

- 宣告多個指向整數的指標變數

- `int *myPtr1, *myPtr2;`

- 任意型態都可以宣告指標變數

- 即是此指標變數的內容是某個型態的變數的位置

`double *p;`

`char *r;`

* 指標變數的位元組(無論型態) <
 64位元: 8位元組
 32位元: 4位元組

- 初始化

- 某個位置
- 0 或 **NULL** – points to nothing (**NULL preferred**)
 - `myPtr1 = NULL;`

Why pointers in C?

- **To simulate call-by-reference**

```
int x;  
scanf("%d", &x); // pointers as arguments  
// scanf could change the value of x
```

- **To allocate memory space at run-time**

// allocate a space for 100 integers at run-time.

```
int *p = (int *) malloc(sizeof(int)*100);
```

// release the allocated memory space

```
free(p);
```

- **To make linked data structures**

- Linked lists
- Trees
- Graphs

```
struct listNode {  
    int data;  
    struct listNode *link;  
}
```

```
struct treeNode {  
    int data;  
    struct treeNode *leftChild, *rightChild;  
}
```

7.3 Pointer Operators

- **&** (address operator)

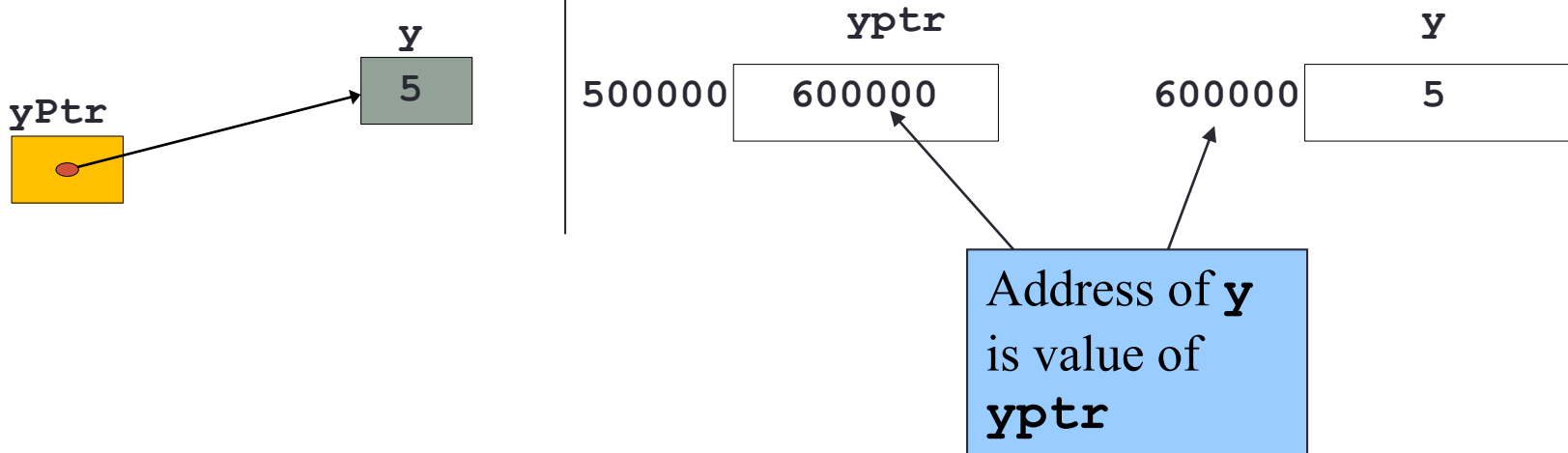
- 得到某變數的位置

```
int y = 5;
```

```
int *yPtr;
```

```
yPtr = &y;           // yPtr gets address of y
```

```
// We may say that yPtr "points to" y
```



7.3 Pointer Operators

- ***** (indirection/dereferencing operator)

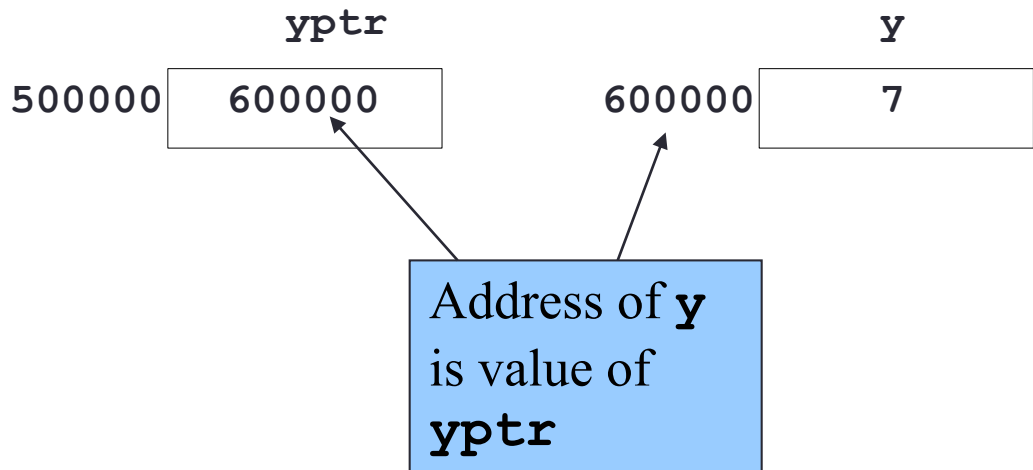
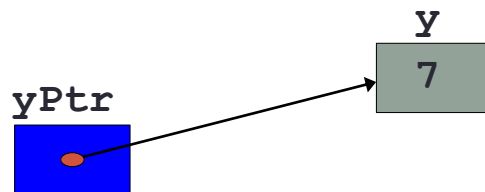
- 使用指標變數所指的那個變數

- ***yPtr** 就是 **y**

- ***yPtr = 7;** // changes **y** to 7

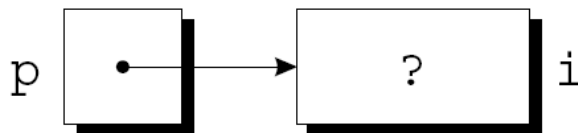
```
int y = 5;  
int *yPtr;  
yPtr = &y;
```

- ***與&互為反運算**

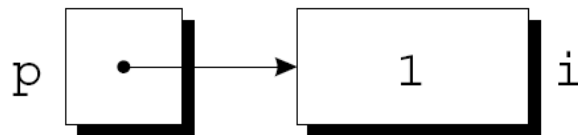


Pointer Operators (cont'd)

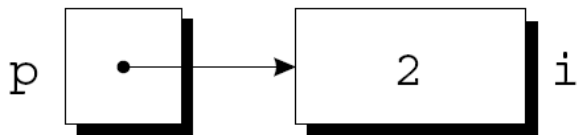
```
p = &i;
```



```
i = 1;
```



```
printf("%d\n", i);      /* prints 1 */  
printf("%d\n", *p);    /* prints 1 */  
*p = 2;
```



```
printf("%d\n", i);      /* prints 2 */  
printf("%d\n", *p);    /* prints 2 */
```

*p如同i的分身一般
*與&互為反運算

Pointer Operators (cont'd)

- 若使用*於一個未初始的指標變數，不知道會有什麼結果:

```
int *p;  
printf("%d", *p);
```

- 一個未初始的指標變數p，若指定值給*p，結果相當危險:

```
int *p;  
*p = 1;  /*** WRONG ***/
```

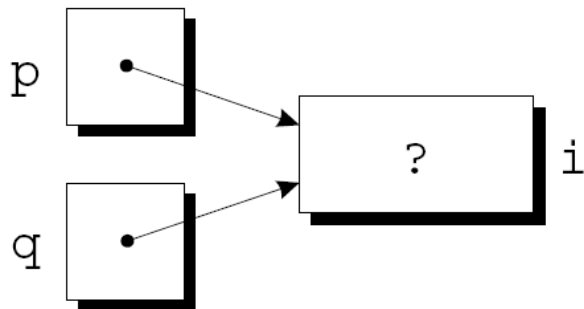
Pointer Assignment

- 相同型態的指標變數可複製其內容。

```
int i, j, *p, *q;
```

```
p = &i; // a pointer assignment
```

```
q = p; // q now points to the same place as p
```

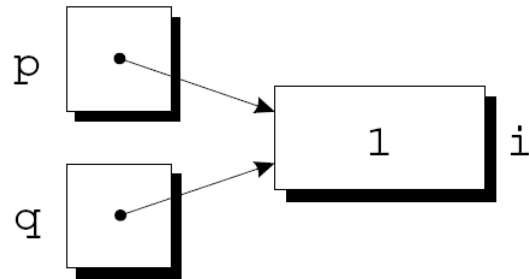


Pointer Assignment (cont'd)

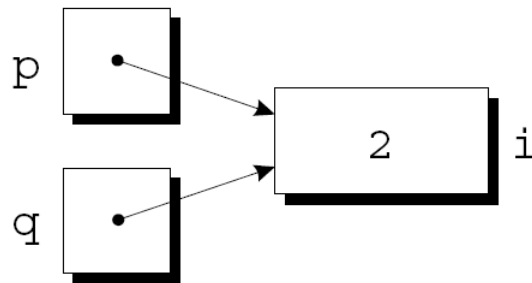
`p=&i;`

`q=&i;` // 改變i變數內容可以透過*`p`或*`q`

`*p = 1;`



`*q = 2;`



- 多個指標變數可以指向同一個變數.

Pointer Assignment (cont'd)

- 不要搞混下面兩者

```
int *p, *q;
```

```
q = p; // pointer assignment
```

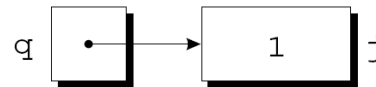
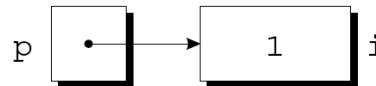
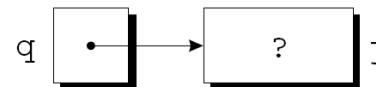
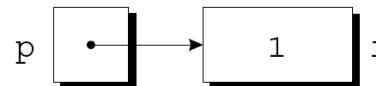
```
*q = *p; // not pointer assignment
```

```
p = &i;
```

```
q = &j;
```

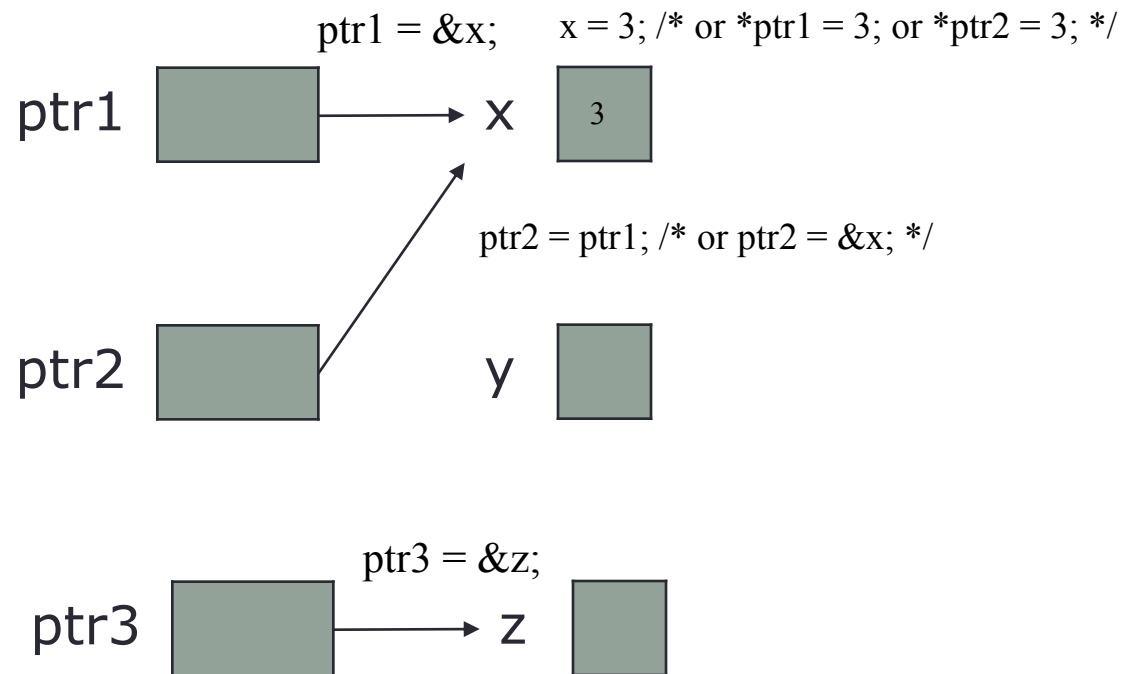
```
i = 1;
```

```
*q = *p;
```



```
int x,y,z;
```

```
int *ptr1, *ptr2, *ptr3;
```



7.4 Calling Functions by Reference

- 使用指標變數模擬傳址呼叫
 - 使用&運算子將變數的位置傳遞給函式
 - 在函式裡你將可以改變變數其內容
 - 陣列不需要使用&，因為陣列名字本質上就是一個指標

```
void Double( int *number ){  
    *number = 2 * ( *number );  
}
```

```
void main() {  
    int x = 5;  
    Double(&x);  
}
```

Notice that the function prototype takes a pointer to an integer (**int ***).

Notice how the address of **number** is given - **cubeByReference** expects a pointer (an address of a variable).

Inside **cubeByReference**, ***nPtr** is used (***nPtr** is **number**).

```
1  /* Fig. 7.7: fig07_07.c
2     Cube a variable using call
3     with a pointer argument */
4
5  #include <stdio.h>
6
7  void cubeByReference( int * ); /*
8
9  int main()
10 {
11     int number = 5;
12
13     printf( "The original value of number is %d", number );
14     cubeByReference( &number );
15     printf( "\nThe new value of number is %d\n", number );
16
17     return 0;
18 }
19
20 void cubeByReference( int *nPtr )
21 {
22     *nPtr = *nPtr * *nPtr * *nPtr; /* cube number in main */
23 }
```

The original value of number is 5
The new value of number is 125

```

#include<stdio.h>
void func(int* ptr)
{
    *ptr += 10;
}
int main()
{
    int    y;
    int    *iptr,*jptr;

    y      = 5;
    iptr   = &y;
    jptr   = iptr;

    printf("&y=%p iptr=%p\n",&y,iptr);
    printf(" y=%d *iptr=%d\n",y, *iptr);
    printf(" y=%d *&y=%d\n",y,*&y);
    printf("iptr=%p *&iptr=%p\n",iptr,*&iptr);

    printf("jptr=%p iptr=%p\n",jptr,iptr);

    func(iptr);

    printf("y=%d\n",y);

    func(jptr);

    printf("y=%d\n",y);

    func(&y);

    printf("y=%d\n",y);
}

```

&y=0012FF74 iptr=0012FF74

y=5 *iptr=5

y=5 *&y=5

iptr=0012FF74 *&iptr=0012FF74

jptr=0012FF74 iptr=0012FF74

y=15

y=25

y=35

Pointers as arguments

```
void decompose(double x, long *int_part, double *frac_part)
{
    *int_part = (long) x;
    *frac_part = x - *int_part;
}
```

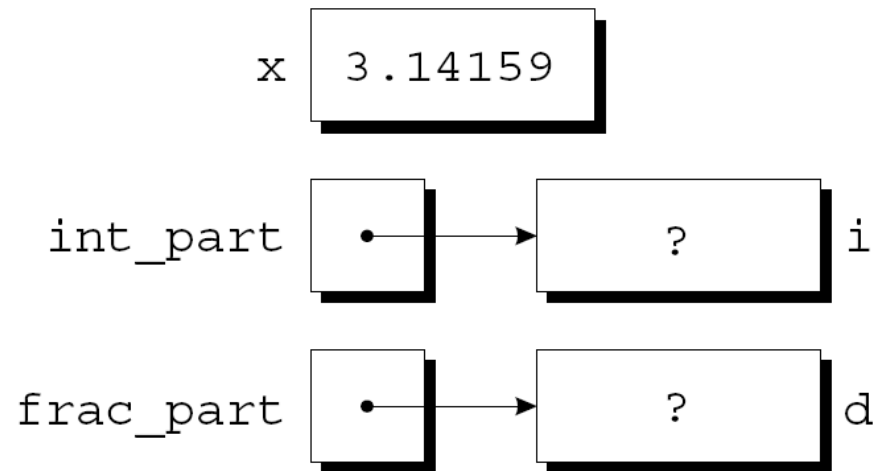
- prototypes for decompose:

```
void decompose(double x, long *int_part, double *frac_part);
void decompose(double, long *, double *);
```

Pointers as arguments (cont'd)

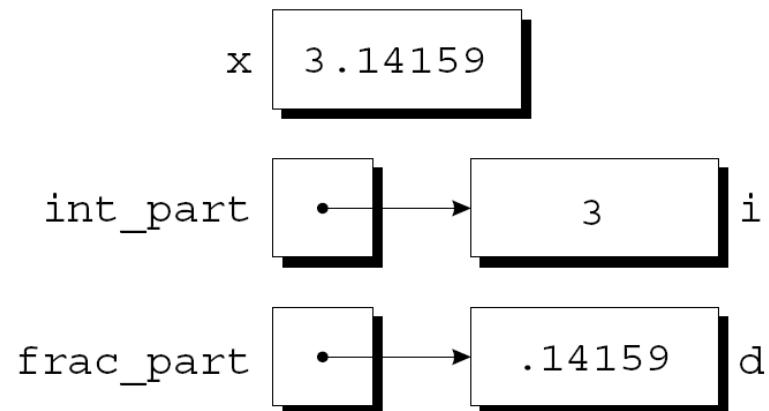
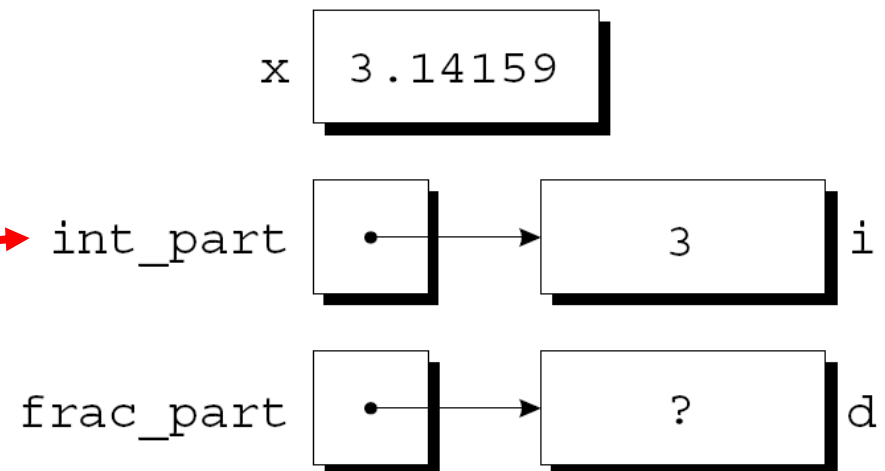
- 呼叫 decompose:
decompose(3.14159, &i, &d);
- 結果, int_part points to i and frac_part points to d:

```
void decompose(double x,  
               long *int_part,  
               double *frac_part)  
{  
    *int_part = (long) x;  
    *frac_part = x -  
    *int_part;  
}
```



Pointers as arguments (cont'd)

```
void decompose(double x,  
               long *int_part,  
               double *frac_part)  
{  
    *int_part = (long) x;  
    *frac_part = x - *int_part;  
}
```



Pointers as arguments (cont'd)

- 呼叫 `scanf` 的參數是指標:

```
int i;
```

```
...
```

```
scanf("%d", &i);
```

若在*i*之前沒有`&`。scanf得到的不是*i*的位置，而是目前*i*的內容。

```
int i, *p;
```

```
...
```

```
p = &i;
```

```
scanf("%d", p);
```

```
scanf("%d", &p);  /** WRONG **/
```

Pointers as arguments (cont'd)

- 若沒正確的值給指標參數，程式執行會發生嚴重錯誤

例如`decompose(3.14159, i, d);`

- `*int_part` 與 `*frac_part` 會分別指向 `i` 與 `d` 變數內容的那個位置，而不是 `i` 與 `d` 的位置。
- 若呼叫 `decompose` 前已宣告 `decompose` 的 `prototype`，則編譯器會幫你把這種錯誤挑出。
- 但是 `scanf` 這個函式卻行不通!! Why??

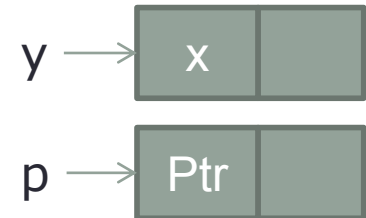
Pointers as arguments (cont'd)

```
int *Ptr, x;  
void assign1(int *p, int y)  
{  
    p = &y;  
}
```

Will Ptr point to x after calling assign1(Ptr, x)?

// correct version

```
void assign2(int **p, int* y)  
{  
    *p = y;  
}
```



Ptr will point to x after calling assign2(&Ptr, &x).

Pointers as arguments (cont'd)

```
int *Ptr, x;  
int* assign3(int y)  
{  
    return &y;  
}
```

Will Ptr point to x after executing `Ptr=assign3(x)`?

```
// correct version  
int* assign4(int* y)  
{  
    return y;  
}
```

Ptr will point to x after calling `Ptr=assign4(&x)`.

範例：交換變數內容的函式

```
void swap(int* a,int *b)
{
    int x;
    x = *a;
    *a=*b;
    *b=x;
}
```


小結論(1/2)

- 在C語言裡，你若要宣告一個變數其內容是一個整數，你就會宣告一個int變數。若是要宣告一個變數其內容是實數你就會宣告float變數。若是要宣告一個變數其內容為位址，那你就宣告**指標變數**。
- 若是指標變數內容所指的那個位址是一個int變數，那麼此指標變數型態為int*；若是指標變數內容所指的那個位址是一個float變數，那麼此指標變數型態為float*。若是指標變數內容所指的那個位址是一個整數指標變數呢(就是int*)？那麼此指標變數型態為**int****。若是那個位址是一個函式其型態如int func(int,float)的函式呢？此指標變數型態就是int (*)(int,float)(稍後會講解)。
- 若是你要使用指標變數所指的那個變數時，就在指標變數前加上*即可。
- C使用指標變數**模擬call-by-reference**，也就是說，當你呼叫函式所傳的參數需要被此函式修改時，你必須將此參數的位址傳給這個函式。在C程式語言，哪種變數的內容可以是位址呢？就是指標變數啦！因此需要被修改的參數在那個函式參數列裡必須是被宣告為指標變數型態。

小結論(2/2)

例子:

要讓函式f改變參數int x的值，在C語言必須將x的位置傳入f，所以必須這樣呼叫f(&x)。

函式f對此參數必須宣告為整數指標變數

```
void f(int *p)
{
    /* ... */
    *p *= 2;
    return;
}
```



在函式f裡，若要存取x。那麼必須透過指標p，語法就是*p。

$p == \&*p == \&x$

7.5 Using the `const` Qualifier with Pointers

- **const** qualifier
 - 變數內容不可被更改
 - 當一個函式不會改變一個變數的內容時（只會讀他的內容）那你就可以使用 `const`
 - 企圖去改變一個 `const` variable 產生語法錯誤
- **const** pointers
 - Point to a constant memory location
 - 宣告時就必須給定初值
- `int *const myPtr = &x;`
 - `Type int *const` – constant pointer to an `int`
- `const int *myPtr = &x;`
 - Regular pointer to a `const int`
- `const int *const Ptr = &x;`
 - `const` pointer to a `const int`
 - `x` can be changed, but not `*Ptr`

```

1  /* Fig. 7.13: fig07_13.c
2     Attempting to modify a constant pointer to
3     non-constant data */
4
5  #include <stdio.h>
6
7  int main()
8  {
9     int x, y;
10
11     int * const ptr = &x; /* ptr is a constant pointer to an
12                            integer. An integer can be modified
13                            through ptr, but ptr always points
14                            to the same memory location. */
15     *ptr = 7;
16     ptr = &y;
17
18     return 0;
19 }

```

Changing ***ptr** is allowed – **x** is not a constant.

Changing **ptr** is an error – **ptr** is a constant pointer.

```

FIG07_13.c:
Error E2024 FIG07_13.c 16: Cannot modify a const object in
function main
*** 1 errors in Compile ***

```

7.6 Bubble Sort Using Call-by-reference

- 使用指標來製作bubblesort
 - 交換兩個變數的內容(**swap two elements**)

- Psuedocode

- Initialize array*

- print data in original order*

- Call function bubblesort*

- print sorted array*

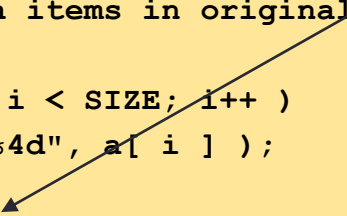
- Define bubblesort*

```

1  /* Fig. 7.15: fig07_15.c
2     This program puts values into an array, sorts the values into
3     ascending order, and prints the resulting array. */
4  #include <stdio.h>
5  #define SIZE 10
6  void bubbleSort( int *, const int );
7
8  int main()
9  {
10
11     int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
12     int i;
13
14     printf( "Data items in original\n" );
15
16     for ( i = 0; i < SIZE; i++ )
17         printf( "%4d", a[ i ] );
18
19     bubbleSort( a, SIZE );           /* sort the array */
20     printf( "\nData items in ascending order\n" );
21
22     for ( i = 0; i < SIZE; i++ )
23         printf( "%4d", a[ i ] );
24
25     printf( "\n" );
26
27     return 0;
28 }
29
30 void bubbleSort( int *array, const int size )
31 {
32     void swap( int *, int * );

```

Bubblesort gets passed the address of array elements (pointers). The name of an array is a pointer.



```

33     int pass, j;
34     for ( pass = 0; pass < size - 1; pass++ )
35
36         for ( j = 0; j < size - 1; j++ )
37
38             if ( array[ j ] > array[ j + 1 ] )
39                 swap( &array[ j ], &array[ j + 1 ] );
40 }
41
42 void swap( int *element1Ptr, int *element2Ptr )
43 {
44     int hold = *element1Ptr;
45     *element1Ptr = *element2Ptr;
46     *element2Ptr = hold;
47 }

```

Data items in original order

2 6 4 8 10 12 89 68 45 37

Data items in ascending order

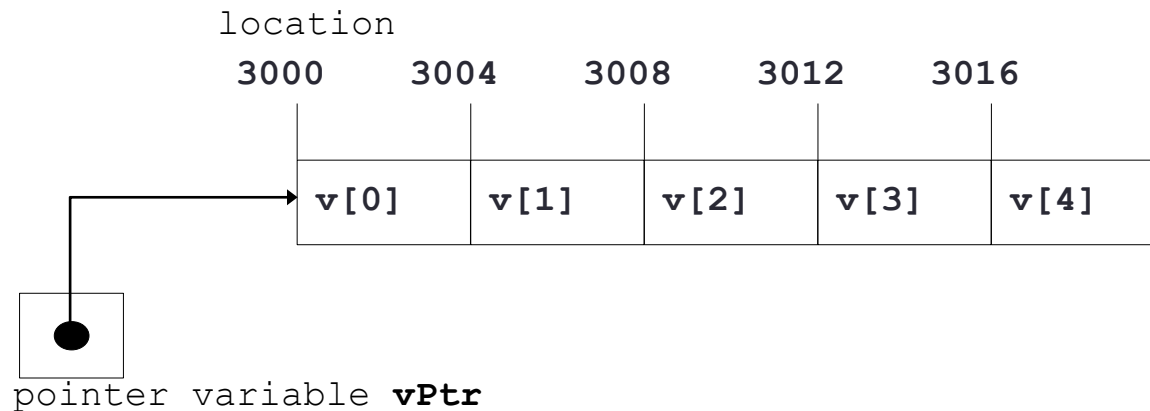
2 4 6 8 10 12 37 45

7.7 Pointer Expressions and Pointer Arithmetic

- 指標變數可進行的運算
 - Increment/decrement pointer (`++` or `--`)
 - 加減一個整數(`+` or `+=` , `-` or `-=`)
 - 指標變數互減
 - 基本上必須對陣列操作才有意義

7.7 Pointer Expressions and Pointer Arithmetic

- 5 個元素的 `int` array (假設`sizeof(int)`為4)
 - `vPtr = &v[0] /* vPtr = v */`
 - at location 3000 (`vPtr` 內容為 3000)
 - `vPtr += 2; /* vPtr指向v[2], vPtr 內容為 3008 */`
 - 讀成往後位移兩個元素 (`vPtr`內容由3000變為3008)



7.7 Pointer Expressions and Pointer Arithmetic

- 指標互減

- 得到它們之間有幾個元素

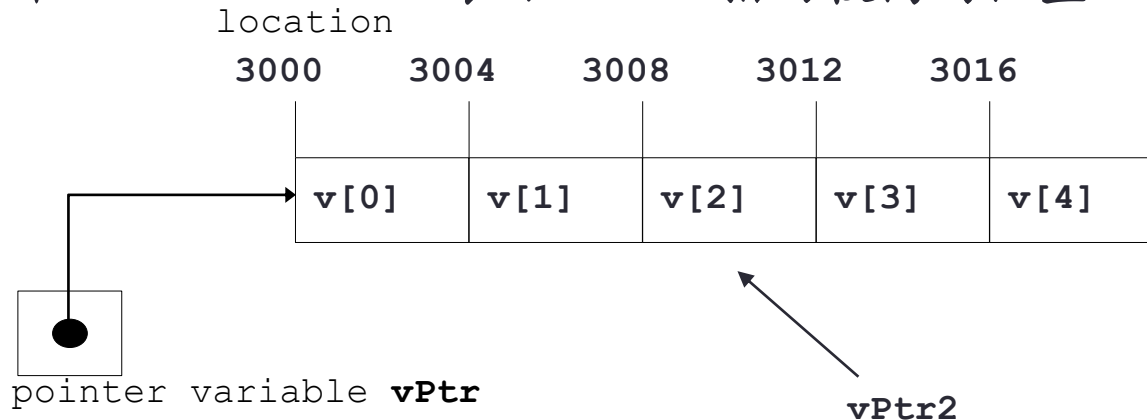
```
vPtr2=&v[ 2 ];
```

```
vPtr=&v[ 0 ];
```

- `vPtr2 - vPtr` 得到 2

- 指標比較 (`<`, `==`, `>`)

- 如果 `vPtr2 > vPtr` 表示 `vPtr2` 指向較高的位置



7.7 Pointer Expressions and Pointer Arithmetic

- 型態相同的指標變數可以互相給值，如果型態不同就必須進行類型轉換

```
int *x,*y;  
char *z;  
x = y; /* correct */  
x = z; /* incorrect */  
x = (int *) z; /*correct */
```

- 例外: pointer to **void** (type **void ***)

```
int *x;  
void *y;  
  
y = x; /* correct */  
x = y; /* incorrect */  
x = (int*) y; /* correct */  
*y = 5; /* incorrect */
```

- void pointers 不可使用* (why ???)**

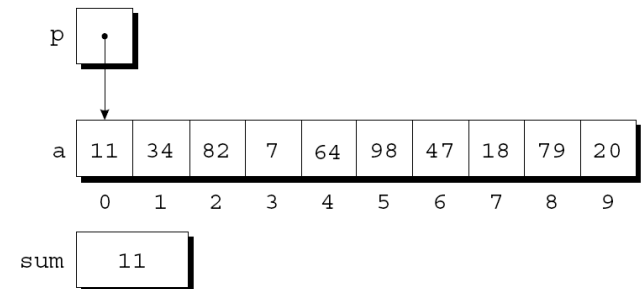
Using Pointers for Array Processing

- 我們可以使用指標來處理陣列裡的每一個元素

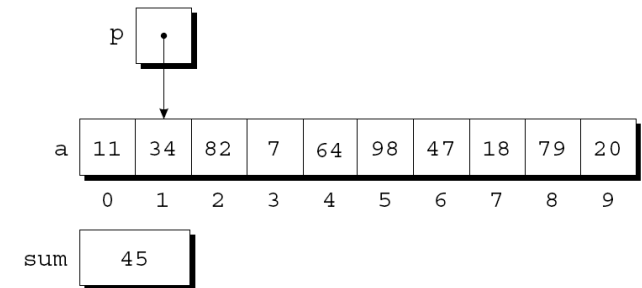
```
#define N 10
...
int a[N], sum, *p;
...
sum = 0;
for (p = &a[0]; p < &a[N]; p++)
    sum += *p;
```

Using Pointers for Array Processing

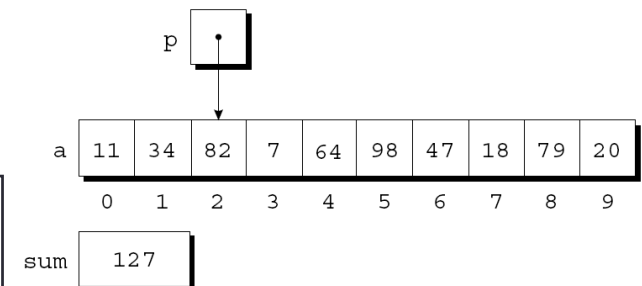
At the end of the first iteration:



At the end of the second iteration:



At the end of the third iteration:



Pointer arithmetic may save execution time. However, some C compilers produce better code for loops that rely on subscripting.

Combining the * and ++ Operators

- 若 `p=&a[i];` 那麼下面效果一樣

`a[i++] = j; // a[i]=j; i++;`

`*p++ = j; // *p = j; p++;`

`*(p++) = j; // *p = j; p++;`

- *與 ++ 可能的組合與意義:

<i>Expression</i>	<i>Meaning</i>
<code>*p++ or *(p++)</code>	Value of expression is *p before increment; increment p later
<code>(*p)++</code>	Value of expression is *p before increment; increment *p later
<code>*++p or *(++p)</code>	Increment p first; value of expression is *p after increment
<code>++*p or ++(*p)</code>	Increment *p first; value of expression is *p after increment

- * 與 -- 組合的意義與 *和 ++ 相同

Combining the * and ++ Operators

- *與 ++最常見的組合為*p++

```
for (p = &a[0]; p < &a[N]; p++)
```

```
    sum += *p;
```

//上面迴圈可改寫為

```
p = &a[0];
```

```
while (p < &a[N])
```

```
    sum += *p++;
```

7.8 The Relationship Between Pointers and Arrays

- C語言裡你可以將指標變數當陣列使用
 - `int *p,a[10];`
 - `p=a;`
 - `p[0]= p[1]+p[2];`
- C語言裡單獨使用陣列的名字，將會視他為一個常數指標變數(不能再將他指向其他位置)
 - `int b[10];`
 - `int *bPtr;`
 - `b = bPtr; /* incorrect */`
 - `bPtr = b; / * correct */`
 - `bPtr[1] = 2;`
 - `bPtr = b + 2;`
 - `bPtr[0] = 2; /* b[2]被改了*/`

Original loop:

```
for (sum=0,p = &a[0]; p < &a[N]; p++) sum += *p;
```

Another version:

```
for (sum=0,p = a; p < a + N; p++) sum += *p;
```

```
int a[10], b;  
*a      = 5;  
a[0]    = 5;  
*(a+i)  =4; // CORRECT!!  
a[i]    =4;
```

```
a++;    // WRONG!!  
a = &b; // WRONG!!
```


7.8 The Relationship Between Pointers and Arrays

- `bPtr = b;`
- `b[3]` 可以是
 - `*(bPtr + 3)`
 - `bptr[3]`
 - `*(b + 3)`
- 下面是不對的

```
int a[10];
while (*a != 0)
    a++;
```

/*** WRONG ***/
- 要改寫為

```
int a[10], *p;
p = a;
while (*p != 0)
    p++;
```

Array Arguments (Revisited)

- Example:

```
int find_largest(int a[], int n)
{
    int i, max;
    max = a[0];
    for (i = 1; i < n; i++)
        if (a[i] > max)
            max = a[i];
    return max;
}
```

- 呼叫 `find_largest`: 將陣列 `b` 當參數。 `b` 其實是指標，將指標 `b` 的內容複製給指標 `a`，因此指標 `a` 的內容為陣列 `b` 的位置。

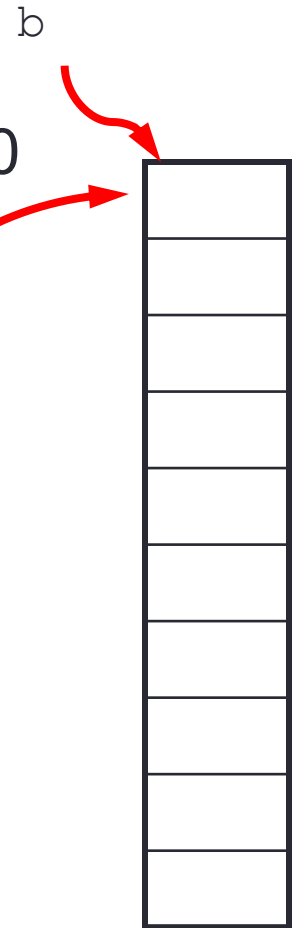
```
#define N 10
int b[N];
largest = find_largest(b, N);
```

- 沒有複製整個陣列。

Array Arguments (Revisited)

```
#define N 10  
int b[N];  
store_zeros(b, N); // 會將陣列b的元素全改為0
```

```
void store_zeros(int a[], int n)  
{  
    int i;  
  
    for (i = 0; i < n; i++)  
        a[i] = 0;  
}
```



Array Arguments (Revisited)

- 若要避免陣列元素被函式改了，請加const

```
int find_largest(const int a[], int n)
{
    a[5] = 3; //WRONG!!! A compiler-time error
}
```

- 代表陣列a的元素為常數整數，因此在 find_largest裡若對a的元素有assignment statement，編譯器會挑出錯誤。

Array Arguments (Revisited)

- 陣列參數可用指標來宣告

```
int find_largest(int *a, int n)
{
    int i;
    a = &i;
}
```

與下面完全一樣嗎？？？

```
int find_largest(int a[], int n)
{
    int i;
    a = &i; // ????
}
```

Array Arguments (Revisited)

- 下面的敘述有什麼不同??

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

```
int *a;
```

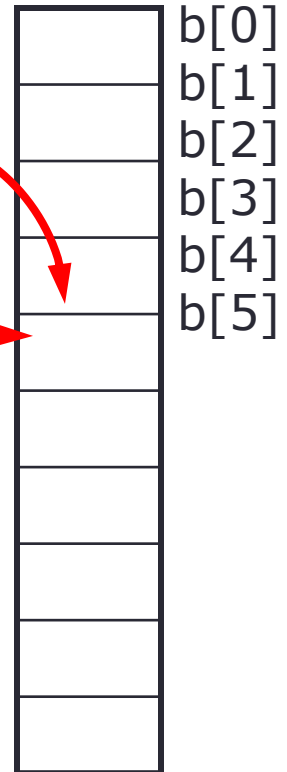
Array Arguments (Revisited)

```
largest = find_largest(&b[5], 10);
```

```
int find_largest(int a[], int n)
{
    int i, max;

    max = a[0];
    for (i = 1; i < n; i++)
        if (a[i] > max)
            max = a[i];
    return max;
}
```

&b[5]



Using a Pointer as an Array Name

- 雖然`p`是指標，我們依然可以用陣列`[]`的語法來拿元素

```
#define N 10
```

```
...
```

```
int a[N], i, sum = 0, *p = a;
```

```
...
```

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

```
    sum += p[i];
```

編譯器會把 `p[i]` 當做 `*(p+i)`。

sizeof與陣列

- **sizeof(x)**
 - 會得到x有多少個bytes
 - x可為
 - Variable names
 - Type name
 - Constant values
- 若x是陣列: $\text{sizeof}(x) = \text{sizeof}(\text{element}) * \text{number of elements}$
 - 如果 `sizeof(int) == 4`

```
int myArray[ 10 ];  
printf( "%d", sizeof( myArray ) );
```

 - 將印出 40
- 注意:

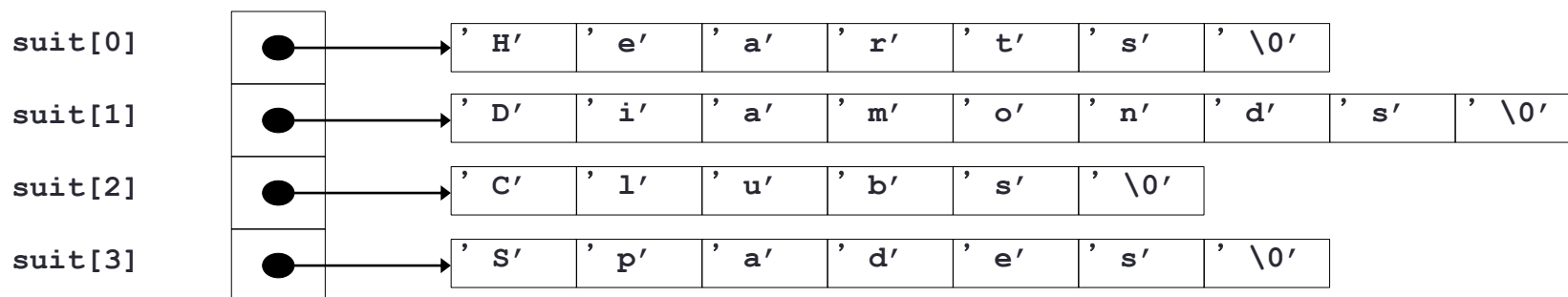
```
void function(int a[4]) {  
    //sizeof(a) is equal to sizeof(int*)  
}
```

7.9 指標變數陣列(Arrays of Pointers)

- 我們可以宣告一個陣列其內容是指標

```
char *suit[ 4 ] = { "Hearts", "Diamonds",  
                  "Clubs", "Spades" };
```

- suit[0]指向字串的第一個字元位置
- char * suit[4]**每個元素都是char指標



•

7.10 Case Study: A Card Shuffling and Dealing Simulation

- 發牌：

- `int desk[4][13]={0}`陣列，表示4種花色，每個花色有13種點數。52張牌發牌順序。
 - 例如，`desk[3][2]`為♠3發牌順序。如果`desk[3][2]`值是1表示♠3是第一張發出的牌。
- 對1~52發牌順序，指定給使用亂數產生器所挑選的一張牌

```
for(card = 1; card <= 52; ++card) {
    do {
        row = rand()%4;
        col = rand()%13;
    } while(desk[row][col] != 0);
    desk[row][col] = card;
}
```

這個範例用的技巧不是很好！
你能指出來並提出方法改進嗎？

- 使用指標陣列指向牌的花色與點數名稱字串

[illegible]

```

1  /* Fig. 7.24: fig07 24.c
2     Card shuffling dealing program */
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <time.h>
6
7  void shuffle( int [][] [ 13 ] );
8  void deal( const int [][] [ 13 ], const char *[], const char *[] );
9
10 int main()
11 {
12     const char *suit[ 4 ] =
13         { "Hearts", "Diamonds", "Clubs", "Spades" };
14     const char *face[ 13 ] =
15         { "Ace", "Deuce", "Three", "Four",
16           "Five", "Six", "Seven", "Eight",
17           "Nine", "Ten", "Jack", "Queen", "King" };
18     int deck[ 4 ][ 13 ] = { 0 };
19
20     srand( time( 0 ) );
21
22     shuffle( deck );
23     deal( deck, face, suit );
24
25     return 0;
26 }
27
28 void shuffle( int wDeck[][ 13 ] )
29 {
30     int row, column, card;
31
32     for ( card = 1; card <= 52; card++ ) {

```

```

33 do {
34     row = rand() % 4;
35     column = rand() % 13;
36 } while( wDeck[ row ][ column ] != 0 );
37
38 wDeck[ row ][ column ] = card;
39 }
40 }

```

The numbers 1-52 are randomly placed into the **deck** array.

```

41
42 void deal( const int wDeck[][ 13 ], const char *wFace[],
43           const char *wSuit[] )
44 {
45     int card, row, column;
46
47     for ( card = 1; card <= 52; card++ )
48
49         for ( row = 0; row <= 3; row++ )
50
51             for ( column = 0; column <= 12; column++ )
52
53                 if ( wDeck[ row ][ column ] == card )
54                     printf( "%5s of %-8s%c",
55                           wFace[ column ], wSuit[ row ],
56                           card % 2 == 0 ? '\n' : '\t' );
57 }

```

Searches **deck** for the **card** number, then prints the **face** and **suit**.

Six of Clubs	Seven of Diamonds
Ace of Spades	Ace of Diamonds
Ace of Hearts	Queen of Diamonds
Queen of Clubs	Seven of Hearts
Ten of Hearts	Deuce of Clubs
Ten of Spades	Three of Spades
Ten of Diamonds	Four of Spades
Four of Diamonds	Ten of Clubs
Six of Diamonds	Six of Spades
Eight of Hearts	Three of Diamonds
Nine of Hearts	Three of Hearts
Deuce of Spades	Six of Hearts
Five of Clubs	Eight of Clubs
Deuce of Diamonds	Eight of Spades
Five of Spades	King of Clubs
King of Diamonds	Jack of Spades
Deuce of Hearts	Queen of Hearts
Ace of Clubs	King of Spades
Three of Clubs	King of Hearts
Nine of Clubs	Nine of Spades
Four of Hearts	Queen of Spades
Eight of Diamonds	Nine of Diamonds
Jack of Diamonds	Seven of Clubs
Five of Hearts	Five of Diamonds
Four of Clubs	Jack of Hearts
Jack of Clubs	Seven of Spades

7.11 Pointers to Functions

- 指向函式的指標變數(Pointer to function)
 - 內容是一個函式的位置
 - 函式名稱為函式的位置
- 指向函式的指標變數可以有下面用途
 - 當成參數傳遞(Passed to functions)
 - 存在陣列裡(Stored in arrays)
 - 指定給其他指標變數(Assigned to other function pointers)

7.11 指向函式的指標(Pointers to Functions)

- Example: bubblesort

- **Bubblesort**有一個參數為指向函式的指標，宣告語法如下

```
void bubble( int work[], const int size, bool  
    ( *compare )( int, int ) )
```

- 由使用者透過函式指標傳入比較大小的函式，Bubblesort就可以由小到大或由大到小排序。

- 如果語法寫成下面的樣子

```
bool *compare( int, int )
```


- 你是宣告一個接受兩個整數參數及回傳一個指向bool的指標的函式


```

1  /* Fig. 7.26: fig07_26.c
2     Multipurpose sorting program using function pointers */
3  #include <stdio.h>
4  #define SIZE 10
5  void bubble( int [], const int, int (*)( int, int ) );
6  int ascending( int, int );
7  int descending( int, int );
8
9  int main()
10 {
11
12     int order,
13         counter,
14         a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
15
16     printf( "Enter 1 to sort in ascending order,\n"
17            "Enter 2 to sort in descending order: " );
18     scanf( "%d", &order );
19     printf( "\nData items in original order\n" );
20
21     for ( counter = 0; counter < SIZE; counter++ )
22         printf( "%5d", a[ counter ] );
23
24     if ( order == 1 ) {
25         bubble( a, SIZE, ascending );
26         printf( "\nData items in ascending order\n" );
27     }
28     else {
29         bubble( a, SIZE, descending );
30         printf( "\nData items in descending order\n" );
31     }
32

```

Notice the function pointer parameter.



```

33  for ( counter = 0; counter < SIZE; counter++ )
34      printf( "%5d", a[ counter ] );
35
36  printf( "\n" );
37
38  return 0;
39 }
40
41 void bubble( int work[], const int size,
42             int (*compare)( int, int ) )
43 {
44     int pass, count;
45
46     void swap( int *, int * );
47
48     for ( pass = 1; pass < size; pass++ )
49
50         for ( count = 0; count < size - 1; count++ )
51
52             if ( (*compare)( work[ count ], work[ count + 1 ] ) )
53                 swap( &work[ count ], &work[ count + 1 ] );
54 }
55
56 void swap( int *element1Ptr, int *element2Ptr )
57 {
58     int temp;
59
60     temp = *element1Ptr;
61     *element1Ptr = *element2Ptr;
62     *element2Ptr = temp;
63 }
64

```

ascending and descending return **true** or **false**. **bubble** calls **swap** if the function call returns **true**.

Notice how function pointers are called using the dereferencing operator. The ***** is not required, but emphasizes that **compare** is a function pointer and not a function.

```

65 int ascending( int a, int b )
66 {
67     return b < a;    /* swap if b is less than a */
68 }
69
70 int descending( int a, int b )
71 {
72     return b > a;    /* swap if b is greater than a */
73 }

```

```

Enter 1 to sort in ascending order,
Enter 2 to sort in descending order: 1

```

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

```

```

Enter 1 to sort in ascending order,
Enter 2 to sort in descending order: 2

```

Data items in original order

```

  2   6   4   8  10  12  89  68  45  37

```

Data items in descending order

```

 89  68  45  37  12  10   8   6   4   2

```

qsort與bsearch (stdlib.h)

```
void qsort (void* base, size_t num, size_t size,  
            int (*compar)(const void*,const void*));
```

```
void* bsearch (const void* key, const void* base,  
               size_t num, size_t size,  
               int (*compar)(const void*,const void*));
```

```
/* qsort example */  
#include <stdio.h>      /* printf */  
#include <stdlib.h>     /* qsort */  
  
int values[] = { 40, 10, 100, 90, 20, 25 };  
  
int compare (const void * a, const void * b)  
{  
    return ( *(int*)a - *(int*)b );  
}  
  
int main ()  
{  
    int n;  
    qsort (values, 6, sizeof(int), compare);  
    for (n=0; n<6; n++)  
        printf ("%d ",values[n]);  
    return 0;  
}
```

<http://cplusplus.com/reference/cstdlib/qsort/>

```
/* bsearch example */  
#include <stdio.h>      /* printf */  
#include <stdlib.h>     /* qsort, bsearch, NULL */  
  
int compareints (const void * a, const void * b)  
{  
    return ( *(int*)a - *(int*)b );  
}  
  
int values[] = { 50, 20, 60, 40, 10, 30 };  
  
int main ()  
{  
    int * pItem;  
    int key = 40;  
    qsort (values, 6, sizeof (int), compareints);  
    pItem = (int*) bsearch (&key, values, 6, sizeof (int), compareints);  
    if (pItem!=NULL)  
        printf ("%d is in the array.\n",*pItem);  
    else  
        printf ("%d is not in the array.\n",key);  
    return 0;  
}
```

<http://cplusplus.com/reference/cstdlib/bsearch/>

常用技巧：使用函式指標陣列

- **Function prototype**要一樣

```
void function1( int a );  
void function2( int b );  
void function3( int c );
```

- 宣告函式指標陣列

```
/* initialize array of 3 pointer to functions that each take an  
   int argument and return void */
```

```
void (*f[ 3 ])( int ) = { function1, function2, function3 };
```

- 函式指標陣列常用於選單

```
printf( "Enter a number between 0 and 2, 3 to end: ");  
scanf( "%d", &choice );  
while ( choice >= 0 && choice < 3 ) {
```

```
    /* 呼叫選擇的功能 */  
    (*f[ choice ])( choice );
```

```
    printf( "Enter a number between 0 and 2, 3 to end: ");  
    scanf( "%d", &choice );  
} /* end while */
```

```
/* Fig. 7.28: fig07_28.c
```

```
    Demonstrating an array of pointers to functions */
```

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

```
/* prototypes */
```

```
void function1( int a );
```

```
void function2( int b );
```

```
void function3( int c );
```

```
int main()
```

```
{
```

```
    /* initialize array of 3 pointer to functions that each take an  
       int argument and return void */
```

```
    void (*f[ 3 ])( int ) = { function1, function2, function3 };
```

```
    int choice; /* variable to hold user's choice */
```

```
    printf( "Enter a number between 0 and 2, 3 to end: " );
```

```
    scanf( "%d", &choice );
```

```
    /* process user's choice */
```

```
    while ( choice >= 0 && choice < 3 ) {
```

```
        /* invoke function at location choice in array f and pass  
           choice as an argument */
```

```
        (*f[ choice ])( choice );
```

```
        printf( "Enter a number between 0 and 2, 3 to end: " );
```

```
        scanf( "%d", &choice );
```

```
    } /* end while */
```

```
    printf( "Program execution completed.\n" );
```

```
    return 0; /* indicates successful termination */
```

```
} /* end main */
```

```
void function1( int a )
```

```
{
```

```
    printf( "You entered %d so function1 was called\n\n", a );
```

```
} /* end function1 */
```

```
void function2( int b )
```

```
{
```

```
    printf( "You entered %d so function2 was called\n\n", b );
```

```
} /* end function2 */
```

```
void function3( int c )
```

```
{
```

```
    printf( "You entered %d so function3 was called\n\n", c );
```

```
} /* end function3 */
```

一個困難的函式指標例子

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

```
int (*f)(float (*)(long),char*))(double);
```

```
int f0(double a)
{
    return a+1;
}
```

```
float f1(long a)
{
    return a+1;
}
```

```
//return type is a pointer to function
int(* fx(float (*xx)(long),char*yy))(double)
{
    printf("%f\n",xx(5));
    return f0;
}
int main()
{
    f = fx;
    printf("%d\n", ((*f)(f1,"hi"))(6));
    return 0;
}
```

f 是一個指向函式的指標，這個函式的
第一個參數:一個函式(型態為接受一個long傳回float的函式)指標，
第二個參數:字元指標
傳回: 函式指標(型態為接受一個double傳回int的函式)