程式設計第七章

真理大學資工系 洪麗玲
llhung@mail.au.edu.tw

7.5 const修飾詞在指標上的使用

- **const修飾詞 (qualifier)** 讓你能夠告訴編譯器，某個變數的值不應該進行更改。
- 在函式的參數上使用 (或不使用) **const**共有6種可能的方式，其中傳值的參數傳遞有兩種，而傳參考的參數傳遞有四種。如何從這6種可能方式中挑出適合自己使用的呢？你可以用**最小權限原則 (principle of least privilege)** 來做為挑選的準則。

- 傳一個指標給函式共有四種方式：
  指向非常數資料的非常數指標
  (a non-constant pointer to non-constant data)、
  指向非常數資料的常數指標
  (a constant pointer to non-constant data)、
  指向常數資料的非常數指標
  (a non-constant pointer to constant data)以及
  指向常數資料的常數指標
  (a constant pointer to constant data)。
  這四種組合的每一種都提供不同等級的存取權。

7.5.1 使用指向非常數資料的非常數指標將字串轉換成大寫

```
1  // Fig. 7.10: fig07_10.c
2  // Converting a string to uppercase using a
3  // non-constant pointer to non-constant data.
4  #include <stdio.h>
5  #include <ctype.h>
6
7  void convertToUppercase( char *sPtr ); // prototype
8
9  int main( void )
10 {
11    char string[] = "cHaRaCters and $32.98"; // initialize char array
12
13    printf( "The string before conversion is: %s", string );
14    convertToUppercase( string );
15    printf( "\nThe string after conversion is: %s\n", string );
16 } // end main
17
18 // convert string to uppercase letters
19 void convertToUppercase( char *sPtr )
20 {
21    while ( *sPtr != '\0' ) { // current character is not '\0'
22       *sPtr = toupper( *sPtr ); // convert to uppercase
23       ++sPtr; // make sPtr point to the next character
24    } // end while
25 } // end function convertToUppercase
```

使用一個指向常數資料的非常數指標，一次一個字元地印出一個字串

```
1  // Fig. 7.11: fig07_11.c
2  // Printing a string one character at a time using
3  // a non-constant pointer to constant data.
4
5  #include <stdio.h>
6
7  void printCharacters( const char *sPtr );
8
9  int main( void )
10 {
11    // initialize char array
12    char string[] = "print characters of a string";
13
14    puts( "The string is:" );
15    printCharacters( string );
16    puts( "" );
17 } // end main
18
```

```
19 // sPtr cannot modify the character to which it points,
20 // i.e., sPtr is a "read-only" pointer
21 void printCharacters( const char *sPtr )
22 {
23    // loop through entire string
24    for ( ; *sPtr != '\0'; ++sPtr ) { // no initialization
25       printf( "%c", *sPtr );
26    } // end for
27 } // end function printCharacters
```

```
The string is:
print characters of a string
```

接收一個指向常數資料的非常數指標(xPtr)函式嘗試更改指標xPtr所指向的資料

```
1  // Fig. 7.12: fig07_12.c
2  // Attempting to modify data through a
3  // non-constant pointer to constant data.
4  #include <stdio.h>
5  void f( const int *xPtr ); // prototype
6
7  int main( void )
8  {
9     int y; // define y
10
11    f( &y ); // f attempts illegal modification
12 } // end main
13
14 // xPtr cannot be used to modify the
15 // value of the variable to which it points
16 void f( const int *xPtr )
17 {
18    *xPtr = 100; // error: cannot modify a const object
19 } // end function f
```

```
c:\examples\ch07\fig07_12.c(18) : error C2166: l-value specifies const object
```

嘗試更改一個指向非常數資料的常數指標

```
1  // Fig. 7.13: fig07_13.c
2  // Attempting to modify a constant pointer to non-constant data.
3  #include <stdio.h>
4
5  int main( void )
6  {
7     int x; // define x
8     int y; // define y
9
10    // ptr is a constant pointer to an integer that can be modified
11    // through ptr, but ptr always points to the same memory location
12    int * const ptr = &x;
13
14    *ptr = 7; // allowed: *ptr is not const
15    ptr = &y; // error: ptr is const; cannot assign new address
16 } // end main
```

```
c:\examples\ch07\fig07_13.c(15) : error C2166: l-value specifies const object
```

嘗試更改一個指向常數資料的常數指標

```
1  // Fig. 7.14: fig07_14.c
2  // Attempting to modify a constant pointer to constant data.
3  #include <stdio.h>
4
5  int main( void )
6  {
7     int x = 5; // initialize x
8     int y; // define y
9
10    // ptr is a constant pointer to a constant integer. ptr always
11    // points to the same location; the integer at that location
12    // cannot be modified
13    const int *const ptr = &x; // initialization is OK
14
15    printf( "%d\n", *ptr );
16    *ptr = 7; // error: *ptr is const; cannot assign new value
17    ptr = &y; // error: ptr is const; cannot assign new address
18 } // end main
```

```
c:\examples\ch07\fig07_14.c(16) : error C2166: l-value specifies const object
c:\examples\ch07\fig07_14.c(17) : error C2166: l-value specifies const object
```

7.6 使用傳參考的氣泡排序法

```
1  // Fig. 7.15: fig07_15.c
2  // Putting values into an array, sorting the values into
3  // ascending order and printing the resulting array.
4  #include <stdio.h>
5  #define SIZE 10
6
7  void bubbleSort( int * const array, size_t size ); // prototype
8
9  int main( void )
10 {
11    // initialize array a
12    int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
13
14    size_t i; // counter
15
16    puts( "Data items in original order" );
17
18    // loop through array a
19    for ( i = 0; i < SIZE; ++i ) {
20       printf( "%4d", a[ i ] );
21    } // end for
```

```
22
23     bubbleSort( a, SIZE ); // sort the array
24
25     puts( "\nData items in ascending order" );
26
27     // loop through array a
28     for ( i = 0; i < SIZE; ++i ) {
29        printf( "%4d", a[ i ] );
30     } // end for
31
32     puts( "" );
33  } // end main
34
35  // sort an array of integers using bubble sort algorithm
36  void bubbleSort( int * const array, size_t size )
37  {
38     void swap( int *element1Ptr, int *element2Ptr ); // prototype
39     unsigned int pass; // pass counter
40     size_t j; // comparison counter
```

» 圖7.15　將存在陣列裡的數值以遞增順序排序，印出陣列結果(2/4)


```
41
42     // loop to control passes
43     for ( pass = 0; pass < size - 1; ++pass ) {
44
45        // loop to control comparisons during each pass
46        for ( j = 0; j < size - 1; ++j ) {
47
48           // swap adjacent elements if theyre out of order
49           if ( array[ j ] > array[ j + 1 ] ) {
50              swap( &array[ j ], &array[ j + 1 ] );
51           } // end if
52        } // end inner for
53     } // end outer for
54  } // end function bubbleSort
55
56  // swap values at memory locations to which element1Ptr and
57  // element2Ptr point
58  void swap( int *element1Ptr, int *element2Ptr )
59  {
60     int hold = *element1Ptr;
61     *element1Ptr = *element2Ptr;
62     *element2Ptr = hold;
63  } // end function swap
```

» 圖7.15　將存在陣列裡的數值以遞增順序排序，印出陣列結果(3/4)

```
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
```

» 圖7.15　將存在陣列裡的數值以遞增順序排序，印出陣列結果(4/4)

7.7　sizeof運算子

❖C提供了一個特殊的一元運算子`sizeof`，它可以用來計算出陣列(或任何其他的資料型別)的大小(單位為位元組)。當`sizeof`應用到圖7.16裡的陣列名稱時(第15行)，它會傳回一個型別為size_t的整數，這個整數便是此陣列所佔用的位元組個數。

```c
// Fig. 7.16: fig07_16.c
// Applying sizeof to an array name returns
// the number of bytes in the array.
#include <stdio.h>
#define SIZE 20

size_t getSize( float *ptr ); // prototype

int main( void )
{
   float array[ SIZE ]; // create array

   printf( "The number of bytes in the array is %u"
           "\nThe number of bytes returned by getSize is %u\n",
           sizeof( array ), getSize( array ) );
} // end main

// return size of ptr
size_t getSize( float *ptr )
{
   return sizeof( ptr );
} // end function getSize
```

```
The number of bytes in the array is 80
The number of bytes returned by getSize is 4
```

圖7.16　sizeof運算子應用到陣列名稱時，程式將傳回此陣列所佔用的位元組個數

❖ 算出標準型別、陣列和指標的大小

- **sizeof**運算子可以用在任何的變數名稱、型別，或值上 (包含運算式的值)。當應用在變數名稱 (這裡不包括陣列名稱) 或常數時。將會傳回用來存放此變數或常數之型別的位元組個數。
- 圖7.17的程式計算了PC上的每一種標準資料型別所佔用的位元組個數。但結果需視實作而定，而且在不同的平台所得到的結果可能不同，不同編譯器在同一平台上的結果有時也不同。

```
1  // Fig. 7.17: fig07_17.c
2  // Using operator sizeof to determine standard data type sizes.
3  #include <stdio.h>
4
5  int main( void )
6  {
7     char c;
8     short s;
9     int i;
10    long l;
11    long long ll;
12    float f;
13    double d;
14    long double ld;
15    int array[ 20 ]; // create array of 20 int elements
16    int *ptr = array; // create pointer to array
17
18    printf( "    sizeof c = %u\tsizeof(char)  = %u"
19            "\n     sizeof s = %u\tsizeof(short) = %u"
20            "\n     sizeof i = %u\tsizeof(int) = %u"
21            "\n     sizeof l = %u\tsizeof(long) = %u"
22            "\n    sizeof ll = %u\tsizeof(long long) = %u"
23            "\n     sizeof f = %u\tsizeof(float) = %u"
24            "\n     sizeof d = %u\tsizeof(double) = %u"
25            "\n    sizeof ld = %u\tsizeof(long double) = %u"
26            "\n sizeof array = %u"
27            "\n   sizeof ptr = %u\n",
```

圖7.17　使用sizeof運算子來算出標準資料型別的大小(1/2)


```
28            sizeof c, sizeof( char ), sizeof s, sizeof( short ), sizeof i,
29            sizeof( int ), sizeof l, sizeof( long ), sizeof ll,
30            sizeof( long long ), sizeof f, sizeof( float ), sizeof d,
31            sizeof( double ), sizeof ld, sizeof( long double ),
32            sizeof array, sizeof ptr );
33 } // end main
```

```
    sizeof c = 1        sizeof(char)  = 1
    sizeof s = 2        sizeof(short) = 2
    sizeof i = 4        sizeof(int) = 4
    sizeof l = 4        sizeof(long) = 4
   sizeof ll = 8        sizeof(long long) = 8
    sizeof f = 4        sizeof(float) = 4
    sizeof d = 8        sizeof(double) = 8
   sizeof ld = 8        sizeof(long double) = 8
sizeof array = 80
  sizeof ptr = 4
```

圖7.17　使用sizeof運算子來算出標準資料型別的大小(2/2)

7.8 指標運算式和指標的算術運算

- 指標是算術運算式、指定運算式，以及比較運算式的合法運算元。不過，並非這些運算式中的所有運算子都能夠與指標變數一起使用。
- 假設陣列**int v[5]**已經定義過，它的第一個元素在記憶體中的位置為**3000**。假設指標**vPtr**設定成指向**v[0]**，亦即**vPtr**的值為**3000**。圖7.18描述了上述的情形。我們可以用以下兩個敘述式之一，將**vPtr**設定成指向陣列**v**。

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

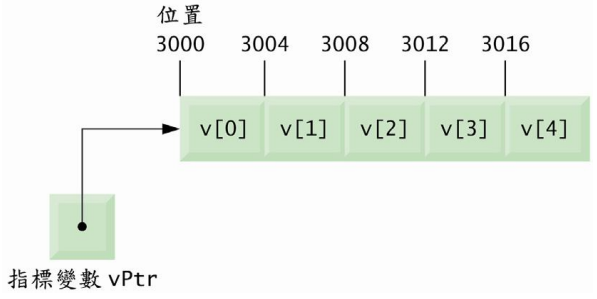



圖7.18 陣列v及一個指向v的指標變數vPtr

- 當某個指標要加上或減去某個整數時，指標並不只是加上或減去這個整數值而已，而是加上或減去這個整數乘以指標所指向之物件的大小。需要加減的數目取決於物件的資料型別。例如，下面的敘述式

- 將會產生**3008 (3000 + 2 * 4)**，假設整數為4個位元組。而陣列**v**中的**vPtr**此時所指向的是**v[2]** (見圖7.19)。

```
vPtr += 2;
```

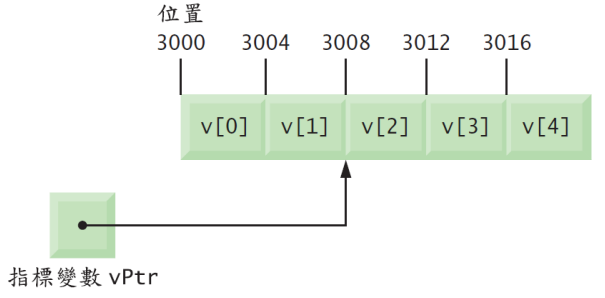
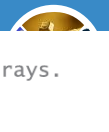


圖7.19　經過指標算術運算之後的vPtr指標。

7.9 指標與陣列的關係

- 圖7.20的程式使用以上所討論的四種方法來參考陣列中的元素。這四種方法是：陣列下標法、以陣列名稱做為指標的指標／位移法、**指標下標 (pointer subscripting)**，以及使用真正指標的指標／位移法。此程式分別以這四種方法，印出了整數陣列**b**的四個元素。



```
1  // Fig. 7.20: fig07_20.cpp
2  // Using subscripting and pointer notations with arrays.
3  #include <stdio.h>
4  #define ARRAY_SIZE 4
5
6  int main( void )
7  {
8     int b[] = { 10, 20, 30, 40 }; // create and initialize arra
9     int *bPtr = b; // create bPtr and point it to array b
10    size_t i; // counter
11    size_t offset; // counter
12
13    // output array b using array subscript notation
14    puts( "Array b printed with:\nArray subscript notation" );
15
16     // loop through array b
17     for ( i = 0; i < ARRAY_SIZE; ++i ) {
18        printf( "b[ %u ] = %d\n", i, b[ i ] );
19     } // end for
20
```

圖7.20　使用四種方法來參考陣列元素(1/3)

```
21      // output array b using array name and pointer/offset notation
22      puts( "\nPointer/offset notation where\n"
23            "the pointer is the array name" );
24
25      // loop through array b
26      for ( offset = 0; offset < ARRAY_SIZE; ++offset ) {
27         printf( "*( b + %u ) = %d\n", offset, *( b + offset ) );
28      } // end for
29
30      // output array b using bPtr and array subscript notation
31      puts( "\nPointer subscript notation" );
32
33      // loop through array b
34      for ( i = 0; i < ARRAY_SIZE; ++i ) {
35         printf( "bPtr[ %u ] = %d\n", i, bPtr[ i ] );
36      } // end for
37
38      // output array b using bPtr and pointer/offset notation
39      puts( "\nPointer/offset notation" );
40
41      // loop through array b
42      for ( offset = 0; offset < ARRAY_SIZE; ++offset ) {
43         printf( "*( bPtr + %u ) = %d\n", offset, *( bPtr + offset ) );
44      } // end for
45   } // end main
```

圖7.20　使用四種方法來參考陣列元素(2/3)

```
Array b printed with:
Array subscript notation
b[ 0 ] = 10
b[ 1 ] = 20
b[ 2 ] = 30
b[ 3 ] = 40

Pointer/offset notation where
the pointer is the array name
*( b + 0 ) = 10
*( b + 1 ) = 20
*( b + 2 ) = 30
*( b + 3 ) = 40
```

```
Pointer subscript notation
bPtr[ 0 ] = 10
bPtr[ 1 ] = 20
bPtr[ 2 ] = 30
bPtr[ 3 ] = 40

Pointer/offset notation
*( bPtr + 0 ) = 10
*( bPtr + 1 ) = 20
*( bPtr + 2 ) = 30
*( bPtr + 3 ) = 40
```

圖7.20　使用四種方法來參考陣列元素(3/3)

❖ 用陣列和指標複製字串

- 為了更進一步說明陣列與指標的可交換性，讓我們來看看圖7.21的兩個字串複製函式——**copy1**和**copy2**。這兩個函式都會將一個字串複製到一個字元陣列。我們可看到**copy1**和**copy2**的函式原型是一樣的，但他們的製作方式並不相同。

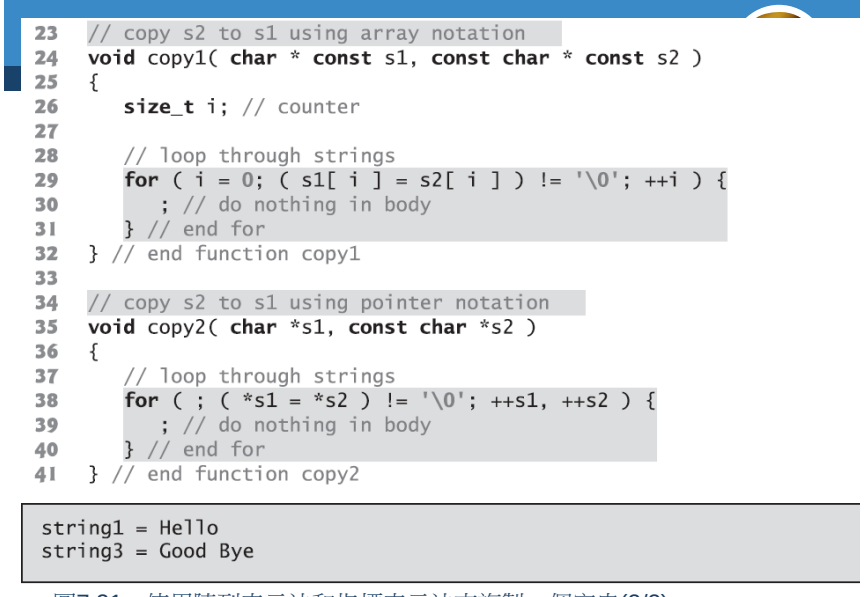
```
1  // Fig. 7.21: fig07_21.c
2  // Copying a string using array notation and pointer notation.
3  #include <stdio.h>
4  #define SIZE 10
5
6  void copy1( char * const s1, const char * const s2 ); // prototype
7  void copy2( char *s1, const char *s2 ); // prototype
8
9  int main( void )
10 {
11    char string1[ SIZE ]; // create array string1
12    char *string2 = "Hello"; // create a pointer to a string
13    char string3[ SIZE ]; // create array string3
14    char string4[] = "Good Bye"; // create a pointer to a string
15
16    copy1( string1, string2 );
17    printf( "string1 = %s\n", string1 );
18
19    copy2( string3, string4 );
20    printf( "string3 = %s\n", string3 );
21 } // end main
22
```

圖7.21　使用陣列表示法和指標表示法來複製一個字串(1/2)

```
23  // copy s2 to s1 using array notation
24  void copy1( char * const s1, const char * const s2 )
25  {
26     size_t i; // counter
27
28     // loop through strings
29     for ( i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; ++i ) {
30        ; // do nothing in body
31     } // end for
32  } // end function copy1
33
34  // copy s2 to s1 using pointer notation
35  void copy2( char *s1, const char *s2 )
36  {
37     // loop through strings
38     for ( ; ( *s1 = *s2 ) != '\0'; ++s1, ++s2 ) {
39        ; // do nothing in body
40     } // end for
41  } // end function copy2
```

```
string1 = Hello
string3 = Good Bye
```

圖7.21　使用陣列表示法和指標表示法來複製一個字串(2/2)



7.10　指標陣列

- 陣列所存放的物件也可以是指標。**指標陣列 (array of pointer)** 常用來建構**字串陣列 (array of strings**，簡稱為**string array)**。在這種情形下，陣列中的每一個項目都是個字串，而C的字串本質上是一個指向字串第一個字元的指標。

- 陣列內的4個值分別為**"Hearts"**、**"Diamonds"**、**"Clubs"**和**"Spades"**。他們都是儲存在記憶體中，以**NULL**為結束符號的字元字串，所以他們的長度都會比引號內的字元個數還多1。這4個字串的長度分別是7、9、6和7個字元長度。雖然看起來好像是這些字串都放到了**suit**陣列裡，但實際上卻只有指標存在陣列中 (見圖7.22)。

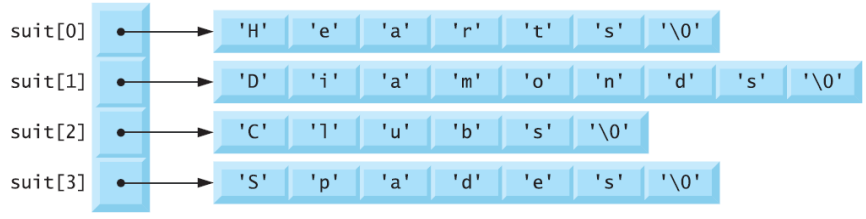
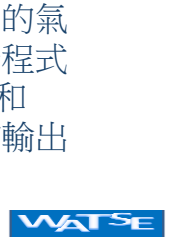


圖7.22　suit陣列的圖形表示

7.12 函式指標

- **指向函式的指標 (pointer to a function)** 內含有函式在記憶體中的位址。我們曾在第6章提過，陣列名稱實際上是此陣列第一個元素在記憶體內的位址。同樣的，函式的名稱是執行此函式之程式碼的起始位址。指向函式的指標可傳遞給函式，由函式傳回來，存放在陣列中，以及指定給其他的函式指標。
- 為了示範函式指標的使用，我們將圖7.15的氣泡排序程式修改成圖7.26的程式。這個新程式由**main**、**bubble**、**swap**、**ascending**和**descending**等函式所組成。這個程式的輸出結果顯示在圖7.27。



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

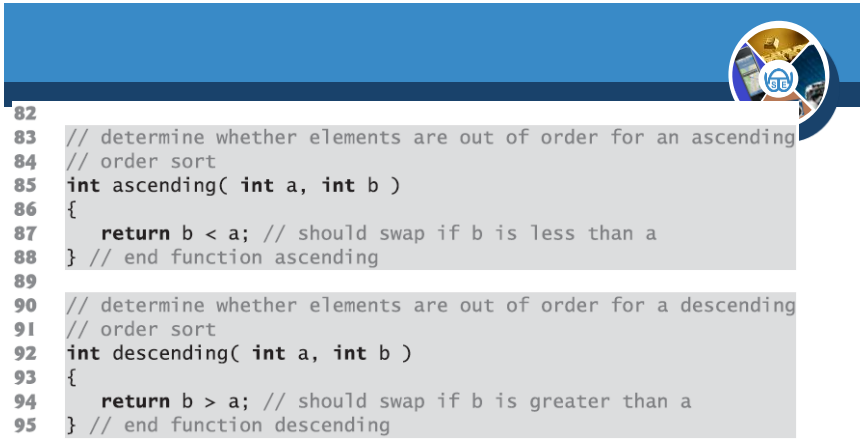
圖7.26 使用函式指標的多功能排序程式(1/4)

```
29
30    // sort array in ascending order; pass function ascending as an
31    // argument to specify ascending sorting order
32    if ( order == 1 ) {
33       bubble( a, SIZE, ascending );
34       puts( "\nData items in ascending order" );
35    } // end if
36    else { // pass function descending
37       bubble( a, SIZE, descending );
38       puts( "\nData items in descending order" );
39    } // end else
40
41    // output sorted array
42    for ( counter = 0; counter < SIZE; ++counter ) {
43       printf( "%5d", a[ counter ] );
44    } // end for
45
46    puts( "\n" );
47 } // end main
48
49 // multipurpose bubble sort; parameter compare is a pointer to
50 // the comparison function that determines sorting order
51 void bubble( int work[], size_t size, int (*compare)( int a, int b ) )
52 {
53    unsigned int pass; // pass counter
54    size_t count; // comparison counter
55
```

圖7.26　使用函式指標的多功能排序程式(2/4)

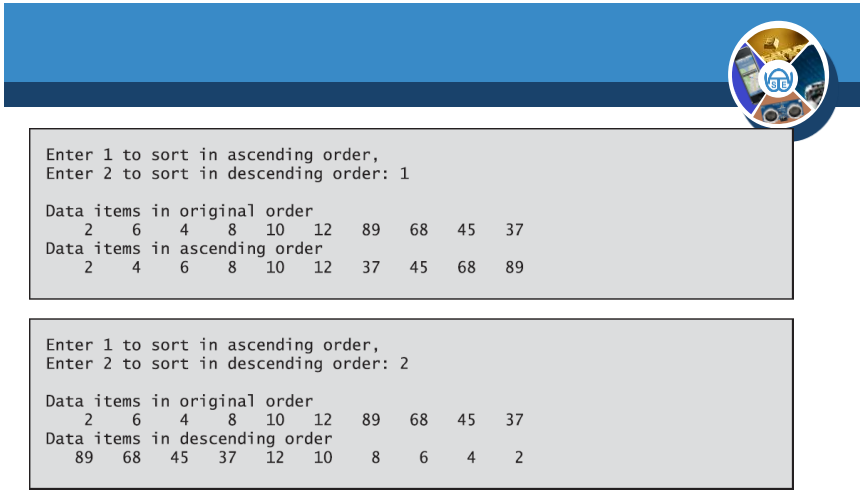
```
56    void swap( int *element1Ptr, int *element2ptr ); // prototype
57
58    // loop to control passes
59    for ( pass = 1; pass < size; ++pass ) {
60
61       // loop to control number of comparisons per pass
62       for ( count = 0; count < size - 1; ++count ) {
63
64          // if adjacent elements are out of order, swap them
65          if ( (*compare)( work[ count ], work[ count + 1 ] ) ) {
66             swap( &work[ count ], &work[ count + 1 ] );
67          } // end if
68       } // end for
69    } // end for
70 } // end function bubble
71
72 // swap values at memory locations to which element1Ptr and
73 // element2Ptr point
74 void swap( int *element1Ptr, int *element2Ptr )
75 {
76    int hold; // temporary holding variable
77
78    hold = *element1Ptr;
79    *element1Ptr = *element2Ptr;
80    *element2Ptr = hold;
81 } // end function swap
```

圖7.26　使用函式指標的多功能排序程式(3/4)



```
82
83 // determine whether elements are out of order for an ascending
84 // order sort
85 int ascending( int a, int b )
86 {
87    return b < a; // should swap if b is less than a
88 } // end function ascending
89
90 // determine whether elements are out of order for a descending
91 // order sort
92 int descending( int a, int b )
93 {
94    return b > a; // should swap if b is greater than a
95 } // end function descending
```

圖7.26　使用函式指標的多功能排序程式(4/4)



```
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
```

圖7.27　圖7.26之氣泡排序程式的輸出

❖在選單驅動的系統上使用函式指標

- 函式指標 **(function pointer)** 常會用在文字型態的選單驅動系統上。在這種系統中，使用者會被要求從選單裡選一個選項 (鍵入1到5)。每一個選項會以不同的函式來加以服務。
- 圖7.28的程式示範了宣告和使用函式指標陣列的方式。此程式定義了三個函式——**function1**、**function2**和**function3**，每個函式都有一個整數引數，而且都不會傳回任何值。

```
1  // Fig. 7.28: fig07_28.c
2  // Demonstrating an array of pointers to functions.
3  #include <stdio.h>
4
5  // prototypes
6  void function1( int a );
7  void function2( int b );
8  void function3( int c );
9
10 int main( void )
11 {
12    // initialize array of 3 pointers to functions that each take an
13    // int argument and return void
14    void (*f[ 3 ])( int ) = { function1, function2, function3 };
15
16    size_t choice; // variable to hold user's choice
```

圖7.28　函式指標陣列的示範(1/3)

```
17
18     printf( "%s", "Enter a number between 0 and 2, 3 to end: " );
19     scanf( "%u", &choice );
20
21     // process user's choice
22     while ( choice >= 0 && choice < 3 ) {
23
24        // invoke function at location choice in array f and pass
25        // choice as an argument
26        (*f[ choice ])( choice );
27
28        printf( "%s", "Enter a number between 0 and 2, 3 to end: " );
29        scanf( "%u", &choice );
30     } // end while
31
32     puts( "Program execution completed." );
33  } // end main
34
35  void function1( int a )
36  {
37     printf( "You entered %d so function1 was called\n\n", a );
38  } // end function1
39
```

圖7.28　函式指標陣列的示範(2/3)

```
40  void function2( int b )
41  {
42     printf( "You entered %d so function2 was called\n\n", b );
43  } // end function2
44
45  void function3( int c )
46  {
47     printf( "You entered %d so function3 was called\n\n", c );
48  } // end function3
```

```
Enter a number between 0 and 2, 3 to end: 0
You entered 0 so function1 was called

Enter a number between 0 and 2, 3 to end: 1
You entered 1 so function2 was called

Enter a number between 0 and 2, 3 to end: 2
You entered 2 so function3 was called

Enter a number between 0 and 2, 3 to end: 3
Program execution completed.
```

圖7.28　函式指標陣列的示範(3/3)