陣列2

洪麗玲

使用字元陣列來儲存以及操作字串

```
1  // Fig. 6.10: fig06_10.c
2  // Treating character arrays as strings.
3  #include <stdio.h>
4  #define SIZE 20
5
6  // function main begins program execution
7  int main( void )
8  {
9     char string1[ SIZE ]; // reserves 20 characters
10    char string2[] = "string literal"; // reserves 15 characters
11    size_t i; // counter
12
13    // read string from user into array string1
14    printf( "%s", "Enter a string (no longer than 19 characters): " );
15    scanf( "%19s", string1 ); // input no more than 19 characters
16
```

```
17      // output strings
18      printf( "string1 is: %s\nstring2 is: %s\n"
19              "string1 with spaces between characters is:\n",
20              string1, string2 );
21
22      // output characters until null character is reached
23      for ( i = 0; i < SIZE && string1[ i ] != '\0'; ++i ) {
24         printf( "%c ", string1[ i ] );
25      } // end for
26
27      puts( "" );
28   } // end main
```

```
Enter a string (no longer than 19 characters): Hello there
string1 is: Hello
string2 is: string literal
string1 with spaces between characters is:
H e l l o
```

圖6.10　將字元陣列視為字串(2/2)

- 靜態區域陣列以及自動區域陣列
  - 圖6.11的程式示範含有宣告**static**區域陣列 (第24行) 的**staticArrayInit**函式 (第21-40行)，以及含有自動區域陣列 (第46行) 的**automaticArrayInit**函式 (第43-62行)。程式呼叫兩次**staticArrayInit**函式 (第12和16行)。此函式中的**static**區域陣列的初始值在程式開始時設為零 (第24行)。

Static陣列

```
1  // Fig. 6.11: fig06_11.c
2  // Static arrays are initialized to zero if not explicitly initializ
3  #include <stdio.h>
4
5  void staticArrayInit( void ); // function prototype
6  void automaticArrayInit( void ); // function prototype
7
8  // function main begins program execution
9  int main( void )
10 {
11    puts( "First call to each function:" );
12    staticArrayInit();
13    automaticArrayInit();
14
15    puts( "\n\nSecond call to each function:" );
16    staticArrayInit();
17    automaticArrayInit();
18 } // end main
```

```
20 // function to demonstrate a static local array
21 void staticArrayInit( void )
22 {
23    // initializes elements to 0 first time function is called
24    static int array1[ 3 ];
25    size_t i; // counter
26
27    puts( "\nValues on entering staticArrayInit:" );
28
29    // output contents of array1
30    for ( i = 0; i <= 2; ++i ) {
31       printf( "array1[ %u ] = %d  ", i, array1[ i ] );
32    } // end for
33
34    puts( "\nValues on exiting staticArrayInit:" );
35
36    // modify and output contents of array1
37    for ( i = 0; i <= 2; ++i ) {
38       printf( "array1[ %u ] = %d  ", i, array1[ i ] += 5 );
39    } // end for
40 } // end function staticArrayInit
```

```
42  // function to demonstrate an automatic local array
43  void automaticArrayInit( void )
44  {
45     // initializes elements each time function is called
46     int array2[ 3 ] = { 1, 2, 3 };
47     size_t i; // counter
48
49     puts( "\n\nValues on entering automaticArrayInit:" );
50
51     // output contents of array2
52     for ( i = 0; i <= 2; ++i ) {
53        printf("array2[ %u ] = %d  ", i, array2[ i ] );
54     } // end for
55
56     puts( "\nValues on exiting automaticArrayInit:" );
57
58     // modify and output contents of array2
59     for ( i = 0; i <= 2; ++i ) {
60        printf( "array2[ %u ] = %d  ", i, array2[ i ] += 5 );
61     } // end for
62  } // end function automaticArrayInit
```

```
First call to each function:

Values on entering staticArrayInit:
array1[ 0 ] = 0  array1[ 1 ] = 0  array1[ 2 ] = 0
Values on exiting staticArrayInit:
array1[ 0 ] = 5  array1[ 1 ] = 5  array1[ 2 ] = 5

Values on entering automaticArrayInit:
array2[ 0 ] = 1  array2[ 1 ] = 2  array2[ 2 ] = 3
Values on exiting automaticArrayInit:
array2[ 0 ] = 6  array2[ 1 ] = 7  array2[ 2 ] = 8
```

```
Second call to each function:

Values on entering staticArrayInit:
array1[ 0 ] = 5  array1[ 1 ] = 5  array1[ 2 ] = 5
Values on exiting staticArrayInit:
array1[ 0 ] = 10  array1[ 1 ] = 10  array1[ 2 ] = 10

Values on entering automaticArrayInit:
array2[ 0 ] = 1  array2[ 1 ] = 2  array2[ 2 ] = 3
Values on exiting automaticArrayInit:
array2[ 0 ] = 6  array2[ 1 ] = 7  array2[ 2 ] = 8
```

6.5 傳遞陣列給函式

- 若我們想傳遞陣列引數給某個函式的話，只需要指定陣列名稱即可，不必加任何的中括號。
- 先前提過C語言中所有引數都是傳值方式，C會自動以傳參考的方式，來將陣列傳給函式。
- 圖6.12的程式利用 %p轉換指定詞 (一個用來列印位址的特殊轉換指定詞) 印出**array**、**&array[0]**和**&array**，來驗證陣列名稱確實是此陣列第一個元素所在的位址。

```
1  // Fig. 6.12: fig06_12.c
2  // Array name is the same as the address of the arrays first element.
3  #include <stdio.h>
4
5  // function main begins program execution
6  int main( void )
7  {
8     char array[ 5 ]; // define an array of size 5
9
10    printf( "    array = %p\n&array[0] = %p\n   &array = %p\n",
11       array, &array[ 0 ], &array );
12 } // end main
```

```
    array = 0012FF78
&array[0] = 0012FF78
   &array = 0012FF78
```

陣列的名稱和陣列第一個元素的位址是相同的

- 傳遞整個陣列與傳遞陣列元素的差異
  - 圖6.13的程式示範傳遞整個陣列和傳遞一個陣列元素之間的差異。

```
1  // Fig. 6.13: fig06_13.c
2  // Passing arrays and individual array elements to functions.
3  #include <stdio.h>
4  #define SIZE 5
5
6  // function prototypes
7  void modifyArray( int b[], size_t size );
8  void modifyElement( int e );
9
```

```
11 int main( void )
12 {
13    int a[ SIZE ] = { 0, 1, 2, 3, 4 }; // initialize array a
14    size_t i; // counter
15
16    puts( "Effects of passing entire array by reference:\n\nThe "
17       "values of the original array are:" );
18
19    // output original array
20    for ( i = 0; i < SIZE; ++i ) {
21       printf( "%3d", a[ i ] );
22    } // end for
23
24    puts( "" );
25
26    // pass array a to modifyArray by reference
27    modifyArray( a, SIZE );
28
29    puts( "The values of the modified array are:" );
30
31    // output modified array
32    for ( i = 0; i < SIZE; ++i ) {
33       printf( "%3d", a[ i ] );
34    } // end for
```

```
36     // output value of a[ 3 ]
37     printf( "\n\n\nEffects of passing array element "
38        "by value:\n\nThe value of a[3] is %d\n", a[ 3 ] );
39
40     modifyElement( a[ 3 ] ); // pass array element a[ 3 ] by value
41
42     // output value of a[ 3 ]
43     printf( "The value of a[ 3 ] is %d\n", a[ 3 ] );
44  } // end main
45
46  // in function modifyArray, "b" points to the original array "a"
47  // in memory
48  void modifyArray( int b[], size_t size )
49  {
50     size_t j; // counter
51
52     // multiply each array element by 2
53     for ( j = 0; j < size; ++j ) {
54        b[ j ] *= 2; // actually modifies original array
55     } // end for
56  } // end function modifyArray
```

```
58  // in function modifyElement, "e" is a local copy of array element
59  // a[ 3 ] passed from main
60  void modifyElement( int e )
61  {
62     // multiply parameter by 2
63     printf( "Value in modifyElement is %d\n", e *= 2 );
64  } // end function modifyElement
```

```
Effects of passing entire array by reference:

The values of the original array are:
  0  1  2  3  4
The values of the modified array are:
  0  2  4  6  8

Effects of passing array element by value:

The value of a[3] is 6
Value in modifyElement is 12
The value of a[ 3 ] is 6
```

Array的傳遞一定會被改值??

- 陣列參數使用const修飾詞
  - 圖6.14解釋**const**修飾詞的用法。函式**tryToModifyArray** (第20行) 和參數**const int b[]** 一起定義，這項定義指出陣列**b**為常數，並且不能夠更改。輸出的結果是編譯器所產生的錯誤訊息——使用不同的編譯器會產生不同的錯誤結果。

```
1  // Fig. 6.14: fig06_14.c
2  // Using the const type qualifier with arrays.
3  #include <stdio.h>
4
5  void tryToModifyArray( const int b[] ); // function prototype
6
7  // function main begins program execution
8  int main( void )
9  {
10    int a[] = { 10, 20, 30 }; // initialize array a
11
12    tryToModifyArray( a );
13
14    printf("%d %d %d\n", a[ 0 ], a[ 1 ], a[ 2 ] );
15 } // end main
16
17 // in function tryToModifyArray, array b is const, so it cannot be
18 // used to modify the original array a in main.
19 void tryToModifyArray( const int b[] )
20 {
21    b[ 0 ] /= 2; // error
22    b[ 1 ] /= 2; // error
23    b[ 2 ] /= 2; // error
24 } // end function tryToModifyArray
```

本週作業 (星期一中午前交)

- **1.改寫6.13，由使用者輸入陣列(個數不限)，呼叫修改每項的內容值是原本的值與前一項值得和。**

  例如：

  1 3 2 9 6 8 ➔ 1 4 5 11 15 14

- **2.改寫 1.，由使用者輸入陣列(個數不限)，呼叫修改每項的內容值是由前面各項累加所得到的值。**

  1 3 2 9 6 8 ➔ 1 4 6 15 21 29