陣列3

6.6 陣列的排序

- 排序 (Sorting) 資料 (也就是照特定的順序放置資料，例如遞增或遞減順序) 是電腦最重要的應用之一。
- 圖6.15的程式以遞增順序，為擁有10個元素的陣列**a** (第10行) 中的所有元素值進行排序。我們所使用的技術稱為**氣泡排序** (**bubble sort**或**sinking sort**)，因為較小的數值將會如「氣泡」浮出水面一樣，慢慢地上升至陣列的頂點，而較大的數值則會沉到陣列的尾端。

```
1  // Fig. 6.15: fig06_15.c
2  // Sorting an array's values into ascending order.
3  #include <stdio.h>
4  #define SIZE 10
5
6  // function main begins program execution
7  int main( void )
8  {
9     // initialize a
10    int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
11    int pass; // passes counter
12    size_t i; // comparisons counter
13    int hold; // temporary location used to swap array elements
```



```
14
15 puts( "Data items in original order" );
16
17 // output original array
18 for ( i = 0; i < SIZE; ++i ) {
19    printf( "%4d", a[ i ] );
20 } // end for
21
22 // bubble sort
23 // loop to control number of passes
24 for ( pass = 1; pass < SIZE; ++pass ) {
25
26    // loop to control number of comparisons per pass
27    for ( i = 0; i < SIZE - 1; ++i ) {
28
29       // compare adjacent elements and swap them if first
30       // element is greater than second element
31       if ( a[ i ] > a[ i + 1 ] ) {
32          hold = a[ i ];
33          a[ i ] = a[ i + 1 ];
34          a[ i + 1 ] = hold;
35       } // end if
36    } // end inner for
37 } // end outer for
```

```
38
39     puts( "\nData items in ascending order" );
40
41     // output sorted array
42     for ( i = 0; i < SIZE; ++i ) {
43        printf( "%4d", a[ i ] );
44     } // end for
45
46     puts( "" );
47  } // end main
```

```
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
```


- 氣泡排序的優點是它很容易撰寫。但氣泡排序執行得相當慢，因為每次的交換只能朝元素的最終位置前進一步。
- 可試試別的排序法~~~作業~~

6.8 搜尋陣列

- 將來你常會碰到存放在陣列裡的大量資料。有時候可能需要知道陣列中是否有一個符合某個**關鍵值 (key value)** 的數值。找出陣列中某個元素的過程稱為**搜尋 (searching)**。
- 使用線性搜尋來搜尋陣列
  - 線性搜尋 (見圖6.18) 用**搜尋關鍵值 (search key)** 來比較陣列中的每個元素。由於陣列中並沒有任何特別的順序，所以有可能在第一次比較就找到，也有可能要到最後一個元素才能找到。平均上，程式需要一半的陣列元素與搜尋鍵比較。
- 使用二元搜尋來搜尋陣列
  - 對於小型的陣列或未排序過的陣列而言，線性搜尋可以表現的很好。但是將線性搜尋用在大型陣列上，就很沒有效率了。如果陣列已經排序過了，則我們可以用速度很快的二元搜尋法。

```
1  // Fig. 6.18: fig06_18.c
2  // Linear search of an array.
3  #include <stdio.h>
4  #define SIZE 100
5
6  // function prototype
7  size_t linearSearch( const int array[], int key, size_t size );
8
9  // function main begins program execution
10 int main( void )
11 {
12    int a[ SIZE ]; // create array a
13    size_t x; // counter for initializing elements 0-99 of array a
14    int searchKey; // value to locate in array a
15    size_t element; // variable to hold location of searchKey or -1
16
17    // create some data
18    for ( x = 0; x < SIZE; ++x ) {
19       a[ x ] = 2 * x;
20    } // end for
21
22    puts( "Enter integer search key:" );
23    scanf( "%d", &searchKey );
24
25    // attempt to locate searchKey in array a
26    element = linearSearch( a, searchKey, SIZE );
27
```

```
28    // display results
29    if ( element != -1 ) {
30       printf( "Found value in element %d\n", element );
31    } // end if
32    else {
33       puts( "Value not found" );
34    } // end else
35 } // end main
36
37 // compare key to every element of array until the location is found
38 // or until the end of array is reached; return subscript of element
39 // if key is found or -1 if key is not found
40 size_t linearSearch( const int array[], int key, size_t size )
41 {
42    size_t n; // counter
43
44    // loop through array
45    for ( n = 0; n < size; ++n ) {
46
47       if ( array[ n ] == key ) {
48          return n; // return location of key
49       } // end if
50    } // end for
51
52    return -1; // key not found
53 } // end function linearSearch
```



```
Enter integer search key:
36
Found value in element 18
```

```
Enter integer search key:
37
Value not found
```

Wireless Access Technologies & Software Engineering

```c
// Fig. 6.19: fig06_19.c
// Binary search of a sorted array.
#include <stdio.h>
#define SIZE 15

// function prototypes
size_t binarySearch(const int b[], int searchKey, size_t low, size_t high);
void printHeader( void );
void printRow( const int b[], size_t low, size_t mid, size_t high );

// function main begins program execution
int main( void )
{
   int a[ SIZE ]; // create array a
   size_t i; // counter for initializing elements of array a
   int key; // value to locate in array a
   size_t result; // variable to hold location of key or -1

   // create data
   for ( i = 0; i < SIZE; ++i ) {
      a[ i ] = 2 * i;
   } // end for

   printf( "%s", "Enter a number between 0 and 28: " );
   scanf( "%d", &key );
```


```c

   printHeader();

   // search for key in array a
   result = binarySearch( a, key, 0, SIZE - 1 );

   // display results
   if ( result != -1 ) {
      printf( "\n%d found in array element %d\n", key, result );
   } // end if
   else {
      printf( "\n%d not found\n", key );
   } // end else
} // end main

// function to perform binary search of an array
size_t binarySearch(const int b[], int searchKey, size_t low, size_t high)
{
   int middle; // variable to hold middle element of array

   // loop until low subscript is greater than high subscript
   while ( low <= high ) {

      // determine middle element of subarray being searched
      middle = ( low + high ) / 2;

```

```
52          // display subarray used in this loop iteration
53          printRow( b, low, middle, high );
54
55          // if searchKey matched middle element, return middle
56          if ( searchKey == b[ middle ] ) {
57             return middle;
58          } // end if
59
60          // if searchKey less than middle element, set new high
61          else if ( searchKey < b[ middle ] ) {
62             high = middle - 1; // search low end of array
63          } // end else if
64
65          // if searchKey greater than middle element, set new low
66          else {
67             low = middle + 1; // search high end of array
68          } // end else
69       } // end while
70
71       return -1; // searchKey not found
72    } // end function binarySearch
73
```

» 圖6.19 在已排序的陣列中進行二元搜尋(3/7)


```
74  // Print a header for the output
75  void printHeader( void )
76  {
77     unsigned int i; // counter
78
79     puts( "\nSubscripts:" );
80
81     // output column head
82     for ( i = 0; i < SIZE; ++i ) {
83        printf( "%3u ", i );
84     } // end for
85
86     puts( "" ); // start new line of output
87
88     // output line of - characters
89     for ( i = 1; i <= 4 * SIZE; ++i ) {
90        printf( "%s", "-" );
91     } // end for
92
93     puts( "" ); // start new line of output
94  } // end function printHeader
95
```

```
96  // Print one row of output showing the current
97  // part of the array being processed.
98  void printRow( const int b[], size_t low, size_t mid, size_t high )
99  {
100    size_t i; // counter for iterating through array b
101
102    // loop through entire array
103    for ( i = 0; i < SIZE; ++i ) {
104
105       // display spaces if outside current subarray range
106       if ( i < low || i > high ) {
107          printf( "%s", "   " );
108       } // end if
109       else if ( i == mid ) { // display middle element
110          printf( "%3d*", b[ i ] ); // mark middle value
111       } // end else if
112       else { // display other elements in subarray
113          printf( "%3d ", b[ i ] );
114       } // end else
115    } // end for
116
117    puts( "" ); // start new line of output
118 } // end function printRow
```



```
Enter a number between 0 and 28: 25

Subscripts:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
---------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
                                 16  18  20  22* 24  26  28
                                                 24  26* 28
                                                 24*

25 not found
```

```
Enter a number between 0 and 28: 8

Subscripts:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
---------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
  0   2   4   6*  8  10  12
                  8  10* 12
                  8*

8 found in array element 4
```

» 圖6.19 在已排序的陣列中進行二元搜尋(6/7)

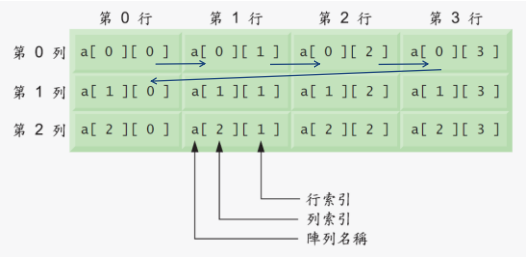
```
Enter a number between 0 and 28: 6

Subscripts:
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
---------------------------------------------------------------
  0   2   4   6   8  10  12  14* 16  18  20  22  24  26  28
  0   2   4   6*  8  10  12

6 found in array element 3
```

» 圖6.19 在已排序的陣列中進行二元搜尋(7/7)

6.9 多維陣列

- C語言的陣列可以有多重下標。**多重下標陣列 (multiple-subscripted arrays)**，標準C語言稱為**多維陣列 (multidimensional arrays)** 經常會用來表示**表格 (tables)**，其數值是依照列 (rows)和行 (columns) 排列的資料所組成。使用兩個下標來指定某個特定元素的陣列，稱為**二維陣列 (double-subscripted arrays)**。多維陣列能夠有兩個以上的索引。

- 圖6.20示範一個二維陣列**a**。



- 多維陣列可以和一維陣列一樣，可以在宣告時指定其初始值。
- 圖6.21表示定義和初始化二維陣列。

```
1  // Fig. 6.21: fig06_21.c
2  // Initializing multidimensional arrays.
3  #include <stdio.h>
4
5  void printArray( int a[][ 3 ] ); // function prototype
6
7  // function main begins program execution
8  int main( void )
9  {
10    // initialize array1, array2, array3
11    int array1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
12    int array2[ 2 ][ 3 ] = { 1, 2, 3, 4, 5 };
13    int array3[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };
14
15    puts( "Values in array1 by row are:" );
16    printArray( array1 );
17
18    puts( "Values in array2 by row are:" );
19    printArray( array2 );
20
21    puts( "Values in array3 by row are:" );
22    printArray( array3 );
23 } // end main
```

```
24
25 // function to output array with two rows and three columns
26 void printArray( int a[][ 3 ] )
27 {
28    size_t i; // row counter
29    size_t j; // column counter
30
31    // loop through rows
32    for ( i = 0; i <= 1; ++i ) {
33
34       // output column values
35       for ( j = 0; j <= 2; ++j ) {
36          printf( "%d ", a[ i ][ j ] );
37       } // end inner for
38
39       printf( "\n" ); // start new line of output
40    } // end outer for
41 } // end function printArray
```

» 圖6.21 初始化多維陣列(2/3)


```
Values in array1 by row are:
1 2 3
4 5 6
Values in array2 by row are:
1 2 3
4 5 0
Values in array3 by row are:
1 2 0
4 0 0
```

» 圖6.21 初始化多維陣列(3/3)

- 二維陣列的處理

陣列的每一列代表一個學生，而每一行則代表學生四次考試成績中的其中一次。陣列的處理是由四個函式來加以執行。

函式**minimum**(第41-60行)會找出所有學生在本學期中的最差成績。

函式maximum(第63-82行)會找出所有學生在本學期中的最高成績。

函式**average**(第85-96行)會算出某位學生本學期的平均成績。

函式**printArray**(第99-118行)會以表列的方式清楚印出這個二維陣列。

```
1  // Fig. 6.22: fig06_22.c
2  // Double-subscripted array manipulations.
3  #include <stdio.h>
4  #define STUDENTS 3
5  #define EXAMS 4
6
7  // function prototypes
8  int minimum( int grades[][ EXAMS ], size_t pupils, size_t tests );
9  int maximum( int grades[][ EXAMS ], size_t pupils, size_t tests );
10 double average( const int setOfGrades[], size_t tests );
11 void printArray( int grades[][ EXAMS ], size_t pupils, size_t tests );
12
```

```
13 // function main begins program execution
14 int main( void )
15 {
16    size_t student; // student counter
17
18    // initialize student grades for three students (rows)
19    int studentGrades[ STUDENTS ][ EXAMS ] =
20       { { 77, 68, 86, 73 },
21         { 96, 87, 89, 78 },
22         { 70, 90, 86, 81 } };
23
24    // output array studentGrades
25    puts( "The array is:" );
26    printArray( studentGrades, STUDENTS, EXAMS );
27
28    // determine smallest and largest grade values
29    printf( "\n\nLowest grade: %d\nHighest grade: %d\n",
30       minimum( studentGrades, STUDENTS, EXAMS ),
31       maximum( studentGrades, STUDENTS, EXAMS ) );
32
33    // calculate average grade for each student
34    for ( student = 0; student < STUDENTS; ++student ) {
35       printf( "The average grade for student %u is %.2f\n",
36          student, average( studentGrades[ student ], EXAMS ) );
37    } // end for
38 } // end main
39
```


```
40 // Find the minimum grade
41 int minimum( int grades[][ EXAMS ], size_t pupils, size_t tests )
42 {
43    size_t i; // student counter
44    size_t j; // exam counter
45    int lowGrade = 100; // initialize to highest possible grade
46
47    // loop through rows of grades
48    for ( i = 0; i < pupils; ++i ) {
49
50       // loop through columns of grades
51       for ( j = 0; j < tests; ++j ) {
52
53          if ( grades[ i ][ j ] < lowGrade ) {
54             lowGrade = grades[ i ][ j ];
55          } // end if
56       } // end inner for
57    } // end outer for
58
59    return lowGrade; // return minimum grade
60 } // end function minimum
61
```

```
62 // Find the maximum grade
63 int maximum( int grades[][ EXAMS ], size_t pupils, size_t tests )
64 {
65    size_t i; // student counter
66    size_t j; // exam counter
67    int highGrade = 0; // initialize to lowest possible grade
68
69    // loop through rows of grades
70    for ( i = 0; i < pupils; ++i ) {
71
72       // loop through columns of grades
73       for ( j = 0; j < tests; ++j ) {
74
75          if ( grades[ i ][ j ] > highGrade ) {
76             highGrade = grades[ i ][ j ];
77          } // end if
78       } // end inner for
79    } // end outer for
80
81    return highGrade; // return maximum grade
82 } // end function maximum
83
```


```
84 // Determine the average grade for a particular student
85 double average( const int setOfGrades[], size_t tests )
86 {
87    size_t i; // exam counter
88    int total = 0; // sum of test grades
89
90    // total all grades for one student
91    for ( i = 0; i < tests; ++i ) {
92       total += setOfGrades[ i ];
93    } // end for
94
95    return ( double ) total / tests; // average
96 } // end function average
97
98 // Print the array
99 void printArray( int grades[][ EXAMS ], size_t pupils, size_t tests )
100 {
101    size_t i; // student counter
102    size_t j; // exam counter
103
104    // output column heads
105    printf( "%s", "                [0]  [1]  [2]  [3]" );
106
107    // output grades in tabular format
108    for ( i = 0; i < pupils; ++i ) {
109
```

```
110         // output label for row
111         printf( "\nstudentGrades[%d] ", i );
112
113         // output grades for one student
114         for ( j = 0; j < tests; ++j ) {
115            printf( "%-5d", grades[ i ][ j ] );
116         } // end inner for
117      } // end outer for
118   } // end function printArray
```

```
The array is:
                [0]  [1]  [2]  [3]
studentGrades[0] 77   68   86   73
studentGrades[1] 96   87   89   78
studentGrades[2] 70   90   86   81

Lowest grade: 68
Highest grade: 96
The average grade for student 0 is 76.00
The average grade for student 1 is 87.50
The average grade for student 2 is 81.75
```


6.10 可變長度陣列

- 標準C語言讓使用者可以用可變長度陣列（VALs）來處理未知大小陣列。這並非是大小可變的陣列—這將會影響到記憶體附近區塊的完整性。可變長度陣列是指其長度或大小是在執行階段時定義的。圖6.23的程式碼宣告並印出多個可變長度陣列。

Wireless Access Technologies & Software Engineering

```c
// Fig. 6.23: fig6_14.c
// Using variable-length arrays in C99
#include <stdio.h>

// function prototypes
void print1DArray( int size, int arr[ size ] );
void print2DArray( int row, int col, int arr[ row ][ col ] );

int main( void )
{
   int arraySize; // size of 1-D array
   int row1, col1, row2, col2; // number of rows and columns in 2-D arrays

   printf( "%s", "Enter size of a one-dimensional array: " );
   scanf( "%d", &arraySize );

   printf( "%s", "Enter number of rows and columns in a 2-D array: " );
   scanf( "%d %d", &row1, &col1 );

   printf( "%s",
      "Enter number of rows and columns in another 2-D array: " );
   scanf( "%d %d", &row2, &col2 );

   int array[ arraySize ]; // declare 1-D variable-length array
   int array2D1[ row1 ][ col1 ]; // declare 2-D variable-length array
   int array2D2[ row2 ][ col2 ]; // declare 2-D variable-length array

```


```c
   // test sizeof operator on VLA
   printf( "\nsizeof(array) yields array size of %d bytes\n",
      sizeof( array ) );

   // assign elements of 1-D VLA
   for ( int i = 0; i < arraySize; ++i ) {
      array[ i ] = i * i;
   } // end for

   // assign elements of first 2-D VLA
   for ( int i = 0; i < row1; ++i ) {
      for ( int j = 0; j < col1; ++j ) {
         array2D1[ i ][ j ] = i + j;
      } // end for
   } // end for

   // assign elements of second 2-D VLA
   for ( int i = 0; i < row2; ++i ) {
      for ( int j = 0; j < col2; ++j ) {
         array2D2[ i ][ j ] = i + j;
      } // end for
   } // end for

   puts( "\nOne-dimensional array:" );
   print1DArray( arraySize, array ); // pass 1-D VLA to function

```

```
54     puts( "\nFirst two-dimensional array:" );
55     print2DArray( row1, col1, array2D1 ); // pass 2-D VLA to function
56
57     puts( "\nSecond two-dimensional array:" );
58     print2DArray( row2, col2, array2D2 ); // pass other 2-D VLA to function
59  } // end main
60
61  void print1DArray( int size, int array[ size ] )
62  {
63     // output contents of array
64     for ( int i = 0; i < size; i++ ) {
65        printf( "array[%d] = %d\n", i, array[ i ] );
66     } // end for
67  } // end function print1DArray
68
69  void print2DArray( int row, int col, int arr[ row ][ col ] )
70  {
71     // output contents of array
72     for ( int i = 0; i < row; ++i ) {
73        for ( int j = 0; j < col; ++j ) {
74           printf( "%5d", arr[ i ][ j ] );
75        } // end for
76
77        puts( "" );
78     } // end for
79  } // end function print2DArray
```


```
Enter size of a one-dimensional array: 6
Enter number of rows and columns in a 2-D array: 2 5
Enter number of rows and columns in another 2-D array: 4 3

sizeof(array) yields array size of 24 bytes

One-dimensional array:
array[0] = 0
array[1] = 1
array[2] = 4
array[3] = 9
array[4] = 16
array[5] = 25

First two-dimensional array:
    0    1    2    3    4
    1    2    3    4    5

Second two-dimensional array:
    0    1    2
    1    2    3
    2    3    4
    3    4    5
```

第一次小考循環開始

- 確認組別沒問題
- 題目:請從ITSA題庫中找出可以用陣列解出來的題目並完成解答。
  (建議從簡單的找起)
- 3/17前找定題目(3/18老師助教提供題目意見)
- 3/24前上傳解答(3/25老師助教提供解答意見)
- 4/1考試(小考)