Chapter 6
C Arrays

C How to Program, 8/e

6.2 Arrays

- `int c[ 12 ];`

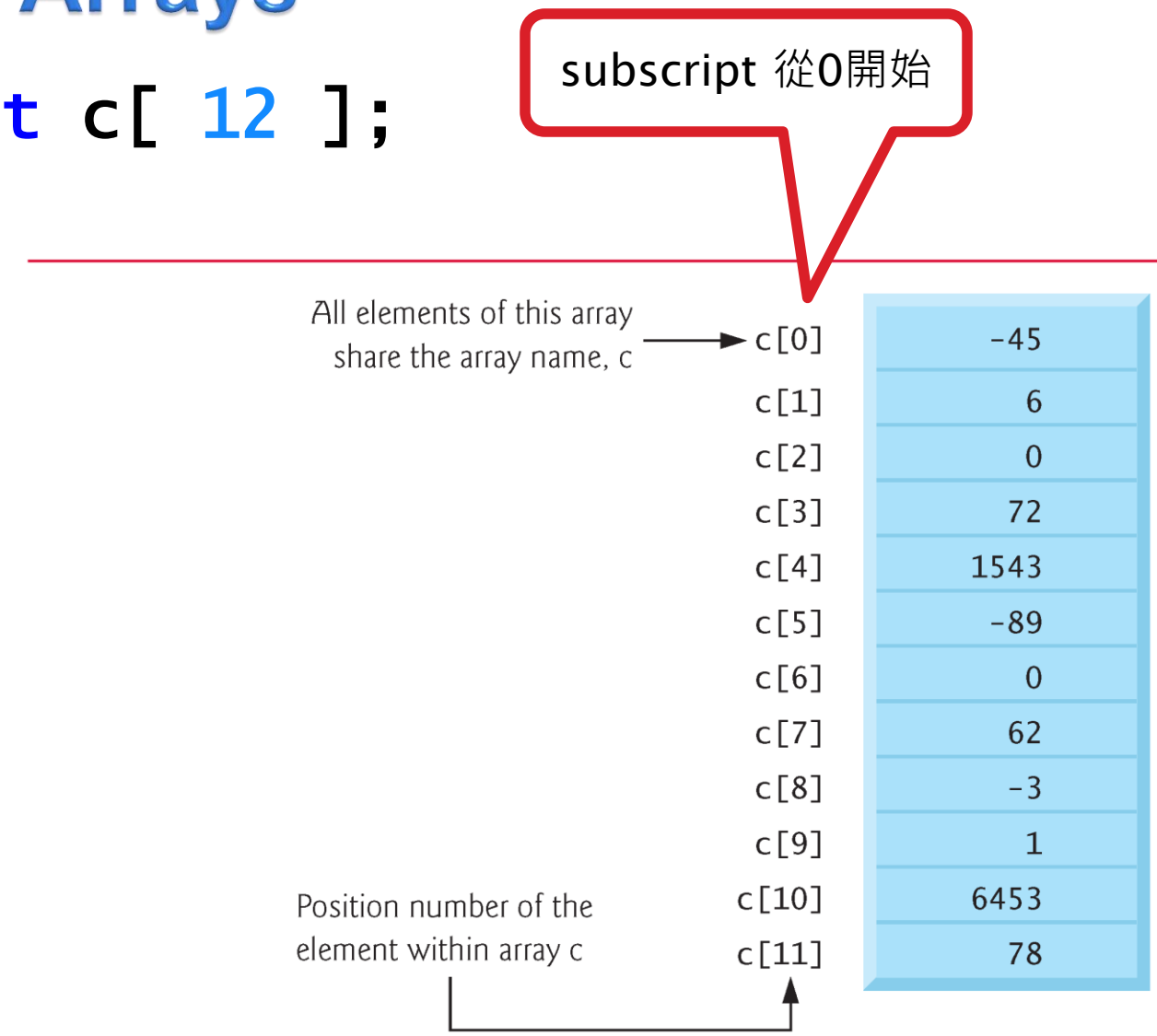


Fig. 6.1 | 12-element array.

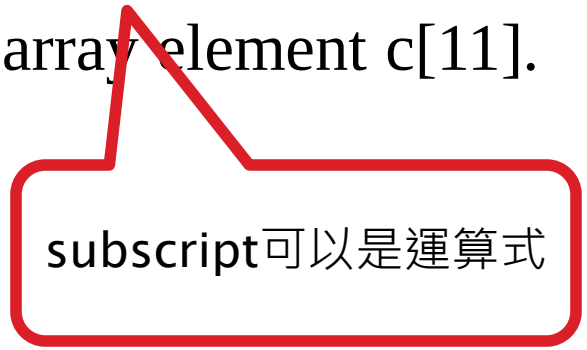
6.2 Arrays (Cont.)

- The *i*th element of array `c` is referred to as `c[i - 1]`.
  - c[0] is the first array element

- To print the sum of the values contained in the first three elements of array `c`
  - `printf( "%d", c[ 0 ] + c[ 1 ] + c[ 2 ] );`

- If a = 5 and b = 6, then the statement
  - `c[ a + b ] += 2;`

  adds 2 to array element c[11].



6.3 Defining Arrays

- Array declaration
  - `int b[ 100 ], x[ 27 ];`

  reserves 100 elements for integer array `b` and 27 elements for integer array `x`.

- Arrays of other data types
  - float haha[10];
  - char hoho[20];

6.4 Array Examples

```
1  // Fig. 6.3: fig06_03.c
2  // Initializing the elements of an array to zeros.
3  #include <stdio.h>
4
5  // function main begins program execution
6  int main(void)
7  {
8     int n[5]; // n is an array of five integers
9
10    // set elements of array n to 0
11    for (size_t i = 0; i < 5; ++i) {
12       n[i] = 0; // set element at location i to 0
13    }
14
15    printf("%s%13s\n", "Element", "Value");
16
17    // output contents of array n in tabular format
18    for (size_t i = 0; i < 5; ++i) {
19       printf("%7u%13d\n", i, n[i]);
20    }
21 }
```

size_t 就是 unsigned int

初始化

印出表頭

Fig. 6.3 | Initializing the elements of an array to zeros. (Part 1 of 2.)

```
Element         Value
      0             0
      1             0
      2             0
      3             0
      4             0
```

Fig. 6.3 | Initializing the elements of an array to zeros. (Part 2 of 2.)

6.4 Array Examples (Cont.)

```
1  // Fig. 6.4: fig06_04.c
2  // Initializing the elements of an array with an initializer list.
3  #include <stdio.h>
4
5  // function main begins program execution
6  int main(void)
7  {
8     // use initializer list to initialize array n
9     int n[5] = {32, 27, 64, 18, 95};
10
11    printf("%s%13s\n", "Element", "Value");
12
13    // output contents of array in tabular format
14    for (size_t i = 0; i < 5; ++i) {
15       printf("%7u%13d\n", i, n[i]);
16    }
17 }
```

宣告時做初始化

Fig. 6.4 | Initializing the elements of an array with an initializer list. (Part 1 of 2.)

```
Element         Value
      0            32
      1            27
      2            64
      3            18
      4            95
```

Fig. 6.4 | Initializing the elements of an array with an initializer list. (Part 2 of 2.)

6.4 Array Examples (Cont.)

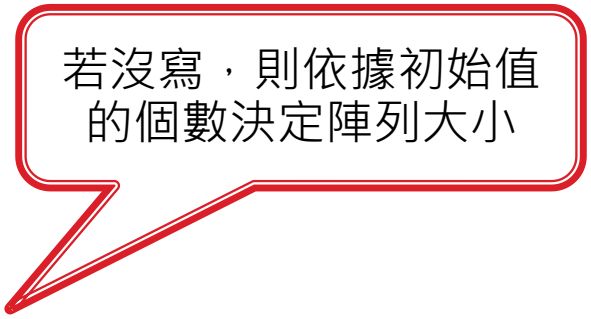
- If there are fewer initializers than elements in the array, the remaining elements are initialized to zero.
  - `int n[ 10 ] = { 0 };`
  - `int a[ 10 ] = { 1 };`

- Arrays are not automatically initialized to zero.

6.4 Array Examples (Cont.)

- syntax error (語法錯誤)
  - `int n[ 5 ] = { 32, 27, 64, 18, 95, 14 };`

6.4 Array Examples (Cont.)



- `int n[] = { 1, 2, 3, 4, 5 };`
  would create a five-element array.

```c
// Fig. 6.5: fig06_05.c
// Initializing the elements of array s to the even integers from 2 to 10.
#include <stdio.h>
#define SIZE 5 // maximum size of array

// function main begins program execution
int main(void)
{
   // symbolic constant SIZE can be used to specify array size
   int s[SIZE]; // array s has SIZE elements

   for (size_t j = 0; j < SIZE; ++j) { // set the va
      s[j] = 2 + 2 * j;
   }

   printf("%s%13s\n", "Element", "Value");

   // output contents of array s in tabular format
   for (size_t j = 0; j < SIZE; ++j) {
      printf("%7u%13d\n", j, s[j]);
   }
}
```

定義常數，沒有分號

宣告Array

Fig. 6.5 | Initializing the elements of array s to the even integers from 2 to 10. (Part 1 of 2.)

```
Element         Value
      0             2
      1             4
      2             6
      3             8
      4            10
```

Fig. 6.5 | Initializing the elements of array `s` to the even integers from 2 to 10. (Part 2 of 2.)

```c
// Fig. 6.6: fig06_06.c
// Computing the sum of the elements of an array.
#include <stdio.h>
#define SIZE 12

// function main begins program execution
int main(void)
{
   // use an initializer list to initialize the array
   int a[SIZE] = {1, 3, 5, 4, 7, 2, 99, 16, 45, 67, 89, 45};
   int total = 0; // sum of array

   // sum contents of array a
   for (size_t i = 0; i < SIZE; ++i) {
      total += a[i];
   }

   printf("Total of array element values is %d\n", total);
}
```

Array element的使用如同普通變數

```
Total of array element values is 383
```

Fig. 6.6 | Computing the sum of the elements of an array.

6.4 Array Examples (Cont.)

- Consider the problem statement.
  - Forty students were asked to rate the quality of the food in the student cafeteria on a scale of 1 to 10 (1 means awful and 10 means excellent). Place the 40 responses in an integer array and summarize the results of the poll.

```c
// Fig. 6.7: fig06_07.c
// Analyzing a student poll.
#include <stdio.h>
#define RESPONSES_SIZE 40 // define array sizes
#define FREQUENCY_SIZE 11

// function main begins program execution
int main(void)
{
   // initialize frequency counters to 0
   int frequency[FREQUENCY_SIZE] = {0};

   // place the survey responses in the responses array
   int responses[RESPONSES_SIZE] = {1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
        1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
        5, 6, 7, 5, 6, 4, 8, 6, 8, 10};

   // for each answer, select value of an element of array responses
   // and use that value as an index in array frequency to
   // determine element to increment
   for (size_t answer = 0; answer < RESPONSES_SIZE; ++answer) {
      ++frequency[responses[answer]];
   }

```

Fig. 6.7 | Analyzing a student poll. (Part 1 of 2.)

```
25      // display results
26      printf("%s%17s\n", "Rating", "Frequency");
27
28      // output the frequencies in a tabular format
29      for (size_t rating = 1; rating < FREQUENCY_SIZE; ++rating) {
30         printf("%6d%17d\n", rating, frequency[rating]);
31      }
32   }
```

```
Rating        Frequency
     1                2
     2                2
     3                2
     4                2
     5                5
     6               11
     7                5
     8                7
     9                1
    10                3
```

Fig. 6.7 | Analyzing a student poll. (Part 2 of 2.)

6.4 Array Examples (Cont.)

- The key statement in the loop is line 25
  - `++frequency[ responses[ answer ] ];`

- 當answer : 0 => responses[0] : 1
- => ++frequency[1]

6.4 Array Examples (Cont.)

- *C has no array bounds checking to prevent the program from referring to an element that does not exist.*

6.4 Array Examples (Cont.)

- Example 2:
  - Roll a single six-sided die 6000 times to test whether the random number generator actually produces random numbers.

```c
// Fig. 6.9: fig06_09.c
// Roll a six-sided die 60,000,000 times
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define SIZE 7

// function main begins program execution
int main(void)
{
   unsigned int frequency[SIZE] = {0}; // clear counts

   srand(time(NULL)); // seed random number generator

   // roll die 60,000,000 times
   for (unsigned int roll = 1; roll <= 60000000; ++roll) {
      size_t face = 1 + rand() % 6;
      ++frequency[face]; // replaces entire switch of Fig. 5.12
   }

```

Fig. 6.9 | Roll a six-sided die 60,000,000 times. (Part 1 of 2.)

```
21      printf("%s%17s\n", "Face", "Frequency");
22
23      // output frequency elements 1-6 in tabular format
24      for (size_t face = 1; face < SIZE; ++face) {
25         printf("%4d%17d\n", face, frequency[face]);
26      }
27   }
```

```
Face          Frequency
   1            9997167
   2           10003506
   3           10001940
   4            9995833
   5           10000843
   6           10000711
```

Fig. 6.9 | Roll a six-sided die 60,000,000 times. (Part 2 of 2.)

6.4 Array Examples (Cont.)

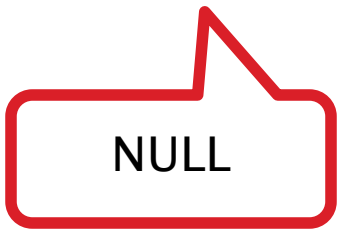
- A character array can be initialized using a string literal.

```
char string1[] = "first";
```

- The string `"first"` contains five characters plus a special string-termination character called the **null character**.
  - array `string1` actually contains **six** elements.

6.4 Array Examples (Cont.)

- The preceding definition is equivalent to
  - ```
    char string1[] =
        { 'f', 'i', 'r', 's', 't', '\0' };
    ```



- `string1[0]` is the character `'f'` and `string1[3]` is the character `'s'`.
    ```

6.4 Array Examples (Cont.)

- Input string using scanf

```
char string2[ 20 ];
scanf( "%19s", string2 );
```

19 characters and a terminating null character.

注意: 沒有&

不超出array範圍

- The name of the array is passed to `scanf` without the preceding & used with nonstring variables.

6.4 Array Examples (Cont.)

- Print a string
  - `Printf( "%s\n", string2 );`

- The characters of the string are **printed until a terminating null character** is encountered.

陣列名稱實際代表陣列開頭位址

```c
// Fig. 6.12: fig06_12.c
// Array name is the same as the address of the array's first element.
#include <stdio.h>

// function main begins program execution
int main(void)
{
   char array[5]; // define an array of size 5

   printf("    array = %p\n&array[0] = %p\n   &array = %p\n",
      array, &array[0], &array);
}
```

%p 列印位址

```
    array = 0031F930
&array[0] = 0031F930
   &array = 0031F930
```

Fig. 6.12 | Array name is the same as the address of the array's first element.

```c
// Fig. 6.10: fig06_10.c
// Treating character arrays as strings.
#include <stdio.h>
#define SIZE 20

// function main begins program execution
int main(void)
{
   char string1[SIZE]; // reserves 20 characters
   char string2[] = "string literal"; // reserves 15 characters

   // read string from user into array string1
   printf("%s", "Enter a string (no longer than 19 characters): ");
   scanf("%19s", string1); // input no more than 19 characters

   // output strings
   printf("string1 is: %s\nstring2 is: %s\n"
          "string1 with spaces between characters is:\n",
          string1, string2);
```

Fig. 6.10 | Treating character arrays as strings. (Part 1 of 2.)

```
20
21      // output characters until null character is reached
22      for (size_t i = 0; i < SIZE && string1[i] != '\0'; ++i) {
23         printf("%c ", string1[i]);
24      }
25
26      puts("");
27   }
```

%c: print個別文字

```
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
```

Fig. 6.10 | Treating character arrays as strings. (Part 2 of 2.)

```c
// Fig. 6.11: fig06_11.c
// Static arrays are initialized to zero if not explicitly initialized.
#include <stdio.h>

void staticArrayInit(void); // function prototype
void automaticArrayInit(void); // function prototype

// function main begins program execution
int main(void)
{
   puts("First call to each function:");
   staticArrayInit();
   automaticArrayInit();

   puts("\n\nSecond call to each function:");
   staticArrayInit();
   automaticArrayInit();
}

```

Fig. 6.11 | Static arrays are initialized to zero if not explicitly initialized. (Part 1 of 4.)

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

宣告static Array 在compile時就已分配空間，Function 結束array1並不會消失

修改array1的值，下次進入function時array1中的值還會留存

Fig. 6.11 | Static arrays are initialized to zero if not explicitly initialized. (Part 2 of 4.)

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

每次進入Function
系統都重新分配
array2空間
結束時array2空間
還給系統

Fig. 6.11 | Static arrays are initialized to zero if not explicitly initialized. (Part 3 of 4.)

```
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 preserved from last call
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 reinitialized after last call
Values on exiting automaticArrayInit:
array2[0] = 6  array2[1] = 7  array2[2] = 8
```

Fig. 6.11 | Static arrays are initialized to zero if not explicitly initialized. (Part 4 of 4.)

6.5 Passing Arrays to Functions

- Define a function to receive an array argument

```
void modifyArray( int b[], int size )
{
        …
}
```

6.5 Passing Arrays to Functions

- To pass an array argument to a function, specify the name of the array without any brackets.
  - `int hourlyTemperatures[ 24 ];`

  - `modifyArray( hourlyTemperatures, 24 );`
    - passes array `hourlyTemperatures` and its size to function `modifyArray`.

```c
// Fig. 6.13: fig06_13.c
// Passing arrays and individual array elements to functions.
#include <stdio.h>
#define SIZE 5

// function prototypes
void modifyArray(int b[], size_t size);
void modifyElement(int e);

// function main begins program execution
int main(void)
{
   int a[SIZE] = {0, 1, 2, 3, 4}; // initialize array a

   puts("Effects of passing entire array by reference:\n\nThe "
      "values of the original array are:");

   // output original array
   for (size_t i = 0; i < SIZE; ++i) {
      printf("%3d", a[i]);
   }

   puts(""); // outputs a newline

```

Fig. 6.13 | Passing arrays and individual array elements to functions. (Part 1 of 4.)


```
25      modifyArray(a, SIZE); // pass array a to modifyArray by reference
26      puts("The values of the modified array are:");
27
28      // output modified array
29      for (size_t i = 0; i < SIZE; ++i) {
30         printf("%3d", a[i]);
31      }
32
33      // output value of a[3]
34      printf("\n\n\nEffects of passing array element "
35         "by value:\n\nThe value of a[3] is %d\n", a[3]);
36
37      modifyElement(a[3]); // pass array element a[3] by value
38
39      // output value of a[3]
40      printf("The value of a[3] is %d\n", a[3]);
41   }
42
```

Fig. 6.13 | Passing arrays and individual array elements to functions. (Part 2 of 4.)

```
43  // in function modifyArray, "b" points to the original array "a"
44  // in memory
45  void modifyArray(int b[], size_t size)
46  {
47     // multiply each array element by 2
48     for (size_t j = 0; j < size; ++j) {
49        b[j] *= 2; // actually modifies original array
50     }
51  }
52
53  // in function modifyElement, "e" is a local copy of array element
54  // a[3] passed from main
55  void modifyElement(int e)
56  {
57     // multiply parameter by 2
58     printf("Value in modifyElement is %d\n", e *= 2);
59  }
```

Fig. 6.13 | Passing arrays and individual array elements to functions. (Part 3 of 4.)

```
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
```

Fig. 6.13 | Passing arrays and individual array elements to functions. (Part 4 of 4.)

```c
// Fig. 6.14: fig06_14.c
// Using the const type qualifier with arrays.
#include <stdio.h>

void tryToModifyArray( const int b[] ); // function prototype

// function main begins program execution
int main( void )
{
   int a[] = { 10, 20, 30 }; // initialize array a

   tryToModifyArray( a );

   printf("%d %d %d\n", a[ 0 ], a[ 1 ], a[ 2 ] );
} // end main

// in function tryToModifyArray, array b is const, so it cannot be
// used to modify the original array a in main.
void tryToModifyArray( const int b[] )
{
   b[ 0 ] /= 2; // error
   b[ 1 ] /= 2; // error
   b[ 2 ] /= 2; // error
} // end function tryToModifyArray
```

宣告const，傳入的array不可在function中變更儲存的值

Fig. 6.14 | Using the const type qualifier with arrays. (Part 1 of 2.)

```
fig06_14.c(21) : error C2166: l-value specifies const object
fig06_14.c(22) : error C2166: l-value specifies const object
fig06_14.c(23) : error C2166: l-value specifies const object
```

Fig. 6.14 | Using the `const` type qualifier with arrays. (Part 2 of 2.)

6.6 Sorting Arrays

```
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
12    puts("Data items in original order");
13
14    // output original array
15    for (size_t i = 0; i < SIZE; ++i) {
16       printf("%4d", a[i]);
17    }
18
```

Fig. 6.15 | Sorting an array's values into ascending order. (Part 1 of 3.)

```
19      // bubble sort
20      // loop to control number of passes
21      for (unsigned int pass = 1; pass < SIZE; ++pass) {
22
23         // loop to control number of comparisons per pass
24         for (size_t i = 0; i < SIZE - 1; ++i) {
25
26            // compare adjacent elements and swap them if first
27            // element is greater than second element
28            if (a[i] > a[i + 1]) {
29               int hold = a[i];
30               a[i] = a[i + 1];
31               a[i + 1] = hold;
32            }
33         }
34      }
35
```

Fig. 6.15 | Sorting an array's values into ascending order. (Part 2 of 3.)

```
36      puts("\nData items in ascending order");
37
38      // output sorted array
39      for (size_t i = 0; i < SIZE; ++i) {
40         printf("%4d", a[i]);
41      }
42
43      puts("");
44   }
```

```
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
```

Fig. 6.15 | Sorting an array's values into ascending order. (Part 3 of 3.)

6.8 Searching Arrays

- It may be necessary to determine whether an array contains a value that matches a certain key value.

- linear search
- binary search

```c
// Fig. 6.18: fig06_18.c
// Linear search of an array.
#include <stdio.h>
#define SIZE 100

// function prototype
size_t linearSearch(const int array[], int key, size_t size);

// function main begins program execution
int main(void)
{
   int a[SIZE]; // create array a

   // create some data
   for (size_t x = 0; x < SIZE; ++x) {
      a[x] = 2 * x;
   }

   printf("Enter integer search key: ");
   int searchKey; // value to locate in array a
   scanf("%d", &searchKey);

```

Fig. 6.18 | Linear search of an array. (Part 1 of 3.)

```
23      // attempt to locate searchKey in array a
24      size_t index = linearSearch(a, searchKey, SIZE);
25
26      // display results
27      if (index != -1) {
28         printf("Found value at index %d\n", index);
29      }
30      else {
31         puts("Value not found");
32      }
33   }
34
```

Fig. 6.18 | Linear search of an array. (Part 2 of 3.)

```
35 // compare key to every element of array until the location is found
36 // or until the end of array is reached; return index of element
37 // if key is found or -1 if key is not found
38 size_t linearSearch(const int array[], int key, size_t size)
39 {
40    // loop through array
41    for (size_t n = 0; n < size; ++n) {
42
43       if (array[n] == key) {
44          return n; // return location of key
45       }
46    }
47
48    return -1; // key not found
49 }
```

```
Enter integer search key: 36
Found value at index 18
```

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

Fig. 6.18 | Linear search of an array. (Part 3 of 3.)

Binary search

```
1  // Fig. 6.19: fig06_19.c
2  // Binary search of a sorted array.
3  #include <stdio.h>
4  #define SIZE 15
5
6  // function prototypes
7  size_t binarySearch(const int b[], int searchKey, size_t low, size_t high);
8  void printHeader(void);
9  void printRow(const int b[], size_t low, size_t mid, size_t high);
10
11 // function main begins program execution
12 int main(void)
13 {
14    int a[SIZE]; // create array a
15
16    // create data
17    for (size_t i = 0; i < SIZE; ++i) {
18       a[i] = 2 * i;
19    }
20
21    printf("%s", "Enter a number between 0 and 28: ");
22    int key; // value to locate in array a
23    scanf("%d", &key);
24
```

Fig. 6.19 | Binary search of a sorted array. (Part 1 of 7.)

```
25      printHeader();
26
27      // search for key in array a
28      size_t result = binarySearch(a, key, 0, SIZE - 1);
29
30      // display results
31      if (result != -1) {
32         printf("\n%d found at index %d\n", key, result);
33      }
34      else {
35         printf("\n%d not found\n", key);
36      }
37   }
38
39   // function to perform binary search of an array
40   size_t binarySearch(const int b[], int searchKey, size_t low, size_t high)
41   {
42      // loop until low index is greater than high index
43      while (low <= high) {
44
45         // determine middle element of subarray being searched
46         size_t middle = (low + high) / 2;
47
```

Fig. 6.19 | Binary search of a sorted array. (Part 2 of 7.)

```
48          // display subarray used in this loop iteration
49          printRow(b, low, middle, high);
50
51          // if searchKey matched middle element, return middle
52          if (searchKey == b[middle]) {
53             return middle;
54          }
55
56          // if searchKey is less than middle element, set new high
57          else if (searchKey < b[middle]) {
58             high = middle - 1; // search low end of array
59          }  if
60
61          // if searchKey is greater than middle element, set new low
62          else {
63             low = middle + 1; // search high end of array
64          }
65       } // end while
66
67       return -1; // searchKey not found
68    }
69
```

Fig. 6.19 | Binary search of a sorted array. (Part 3 of 7.)

```
70  // Print a header for the output
71  void printHeader(void)
72  {
73     puts("\nIndices:");
74
75     // output column head
76     for (unsigned int i = 0; i < SIZE; ++i) {
77        printf("%3u ", i);
78     }
79
80     puts(""); // start new line of output
81
82     // output line of - characters
83     for (unsigned int i = 1; i <= 4 * SIZE; ++i) {
84        printf("%s", "-");
85     }
86
87     puts(""); // start new line of output
88  }
89
```

Fig. 6.19 | Binary search of a sorted array. (Part 4 of 7.)

```
90   // Print one row of output showing the current
91   // part of the array being processed.
92   void printRow(const int b[], size_t low, size_t mid, size_t high)
93   {
94      // loop through entire array
95      for (size_t i = 0; i < SIZE; ++i) {
96
97         // display spaces if outside current subarray range
98         if (i < low || i > high) {
99            printf("%s", "    ");
100        }
101        else if (i == mid) { // display middle element
102           printf("%3d*", b[i]); // mark middle value
103        }
104        else { // display other elements in subarray
105           printf("%3d ", b[i]);
106        }
107     }
108
109     puts(""); // start new line of output
110  }
```

Fig. 6.19 | Binary search of a sorted array. (Part 5 of 7.)

```
90   // Print one row of output showing the current
91   // part of the array being processed.
92   void printRow(const int b[], size_t low, size_t mid, size_t high)
93   {
94      // loop through entire array
95      for (size_t i = 0; i < SIZE; ++i) {
96
97         // display spaces if outside current subarray range
98         if (i < low || i > high) {
99            printf("%s", "   ");
100        }
101        else if (i == mid) { // display middle element
102           printf("%3d*", b[i]); // mark middle value
103        }
104        else { // display other elements in subarray
105           printf("%3d ", b[i]);
106        }
107     }
108
109     puts(""); // start new line of output
110  }
```

Fig. 6.19 | Binary search of a sorted array. (Part 5 of 7.)

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

Indices:
  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
```

Fig. 6.19 | Binary search of a sorted array. (Part 6 of 7.)

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

Indices:
  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 at index 4
```

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

Indices:
  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 at index 3
```

Fig. 6.19 | Binary search of a sorted array. (Part 7 of 7.)

6.9 Multiple-Subscripted Arrays

- int a[3][4];
- All array elements are **stored consecutively** in memory regardless of the number of subscripts.

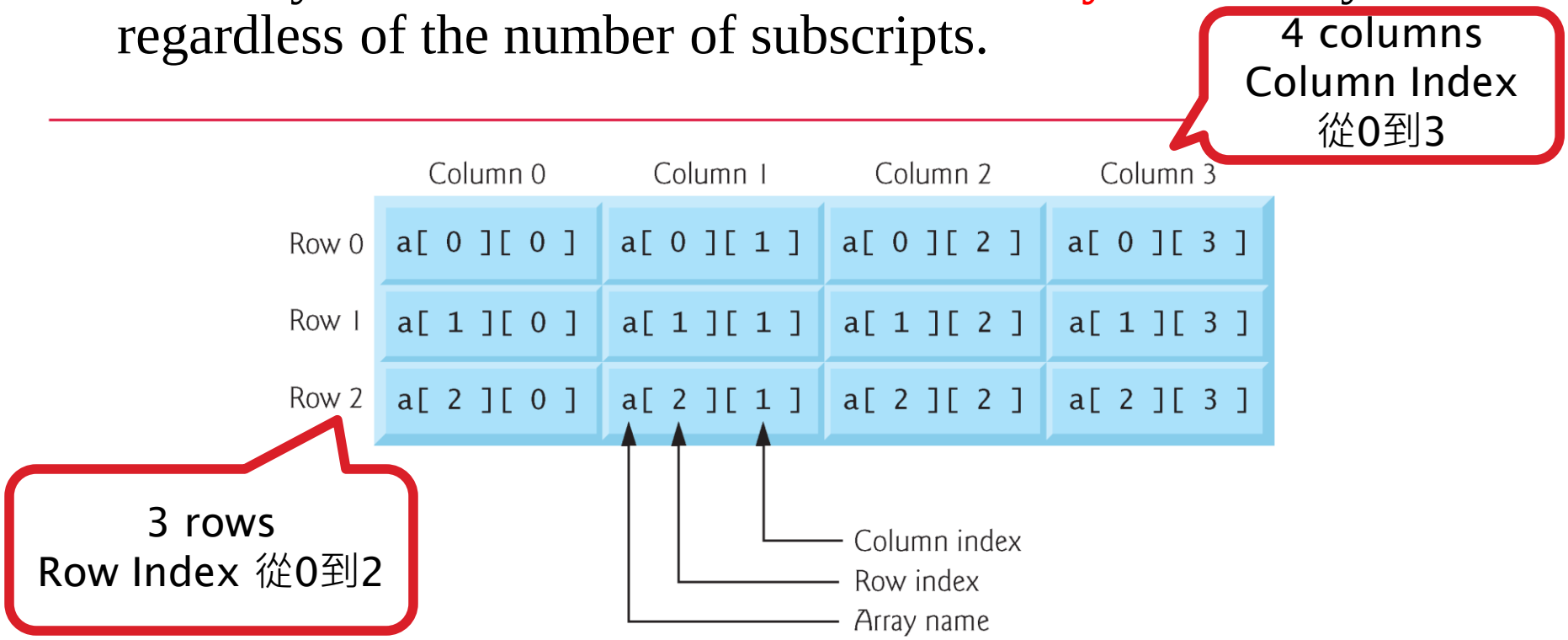


Fig. 6.20 | Two-dimensional array with three rows and four columns.

6.9 Multiple-Subscripted Arrays (Cont.)

- Initialization
  - `int b[ 2 ][ 2 ] = { { 1, 2 }, { 3, 4 } };`

- The values in the first set of braces initialize row 0 and the values in the second set of braces initialize row 1.
  - The values `1` and `2` initialize elements `b[0][0]` and `b[0][1]`, respectively
  - The values `3` and `4` initialize elements `b[1][0]` and `b[1][1]`, respectively.

```c
// Fig. 6.21: fig06_21.c
// Initializing multidimensional arrays.
#include <stdio.h>

void printArray(int a[][3]); // function prototype

// function main begins program execution
int main(void)
{
   int array1[2][3] = {{1, 2, 3}, {4, 5, 6}};
   puts("Values in array1 by row are:");
   printArray(array1);

   int array2[2][3] = {1, 2, 3, 4, 5};
   puts("Values in array2 by row are:");
   printArray(array2);

   int array3[2][3] = {{1, 2}, {4}};
   puts("Values in array3 by row are:");
   printArray(array3);
}
```

先初始化row 0
再初始化 row1
不足的初始化為0

不足的初始化
為0

Fig. 6.21 | Initializing multidimensional arrays. (Part 1 of 2.)

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

傳入function第一個下標可省，其他下標必須要有

```
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
```

Fig. 6.21 | Initializing multidimensional arrays. (Part 2 of 2.)

6.9 Multiple-Subscripted Arrays (Cont.)

- The following nested `for` statement determines the total of all the elements in array `a`.

```
total = 0;
for ( row = 0; row <= 2; ++row ) {
   for ( column = 0; column <= 3; ++column )
  {
      total += a[ row ][ column ];
   }
}
```

Example 成績計算

```
1  // Fig. 6.22: fig06_22.c
2  // Two-dimensional array manipulations.
3  #include <stdio.h>
4  #define STUDENTS 3
5  #define EXAMS 4
6
7  // function prototypes
8  int minimum(const int grades[][EXAMS], size_t pupils, size_t tests);
9  int maximum(const int grades[][EXAMS], size_t pupils, size_t tests);
10 double average(const int setOfGrades[], size_t tests);
11 void printArray(const int grades[][EXAMS], size_t pupils, size_t tests);
12
13 // function main begins program execution
14 int main(void)
15 {
16    // initialize student grades for three students (rows)
17    int studentGrades[STUDENTS][EXAMS] =
18       { { 77, 68, 86, 73 },
19         { 96, 87, 89, 78 },
20         { 70, 90, 86, 81 } };
21
22    // output array studentGrades
23    puts("The array is:");
24    printArray(studentGrades, STUDENTS, EXAMS);
```

一row為一個學生
4個考試的成績

Fig. 6.22 | Two-dimensional array manipulations. (Part 1 of 7.)

```
25
26      // determine smallest and largest grade values
27      printf("\n\nLowest grade: %d\nHighest grade: %d\n",
28         minimum(studentGrades, STUDENTS, EXAMS),
29         maximum(studentGrades, STUDENTS, EXAMS));
30
31      // calculate average grade for each student
32      for (size_t student = 0; student < STUDENTS; ++student) {
33         printf("The average grade for student %u is %.2f\n",
34            student, average(studentGrades[student], EXAMS));
35      }
36   }
37
```

Fig. 6.22 | Two-dimensional array manipulations. (Part 2 of 7.)

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

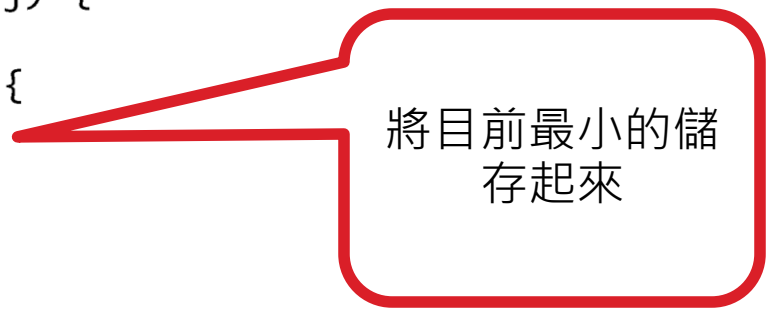


Fig. 6.22 | Two-dimensional array manipulations. (Part 3 of 7.)

```
58  // Find the maximum grade
59  int maximum(const int grades[][EXAMS], size_t pupils, size_t tests)
60  {
61     int highGrade = 0; // initialize to lowest possible grade
62
63     // loop through rows of grades
64     for (size_t i = 0; i < pupils; ++i) {
65
66        // loop through columns of grades
67        for (size_t j = 0; j < tests; ++j) {
68
69           if (grades[i][j] > highGrade) {
70              highGrade = grades[i][j];
71           }
72        }
73     }
74
75     return highGrade; // return maximum grade
76  }
77
```

Fig. 6.22 | Two-dimensional array manipulations. (Part 4 of 7.)

```
78  // Determine the average grade for a particular student
79  double average(const int setOfGrades[], size_t tests)
80  {
81     int total = 0; // sum of test grades
82
83     // total all grades for one student
84     for (size_t i = 0; i < tests; ++i) {
85        total += setOfGrades[i];
86     }
87
88     return (double) total / tests; // average
89  }
90
```

Fig. 6.22 | Two-dimensional array manipulations. (Part 5 of 7.)

```
91  // Print the array
92  void printArray(const int grades[][EXAMS], size_t pupils, size_t tests)
93  {
94     // output column heads
95     printf("%s", "                  [0]  [1]  [2]  [3]");
96
97     // output grades in tabular format
98     for (size_t i = 0; i < pupils; ++i) {
99
100       // output label for row
101       printf("\nstudentGrades[%u] ", i);
102
103       // output grades for one student
104       for (size_t j = 0; j < tests; ++j) {
105          printf("%-5d", grades[i][j]);
106       }
107    }
108 }
```

Fig. 6.22 | Two-dimensional array manipulations. (Part 6 of 7.)

```
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
```

Fig. 6.22 | Two-dimensional array manipulations. (Part 7 of 7.)