CHAPTER 6 - ARRAYS

考慮下面的問題

- 將三個整數由小到大印出來，程式可以寫成下面樣子

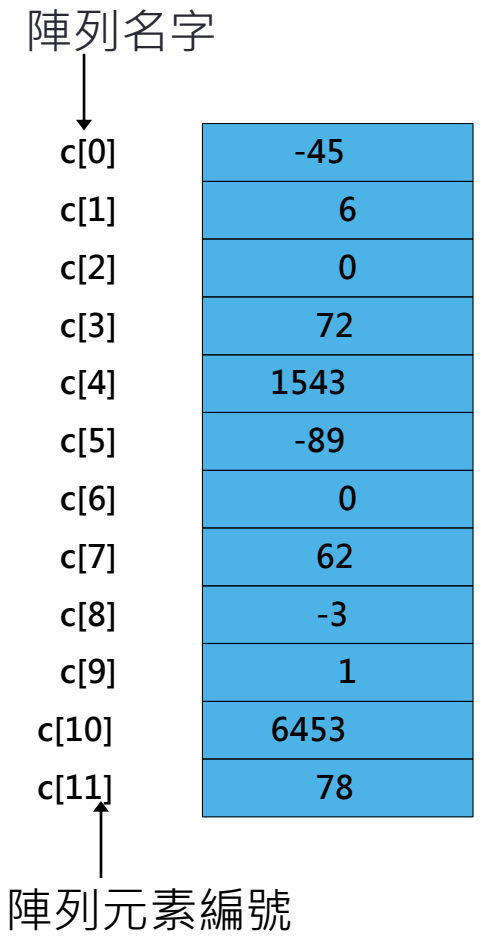
```
int x1,x2,x3;
if (x1 >= x2) {
   if (x1 >= x3) {
      if (x2 >= x3) {
            printf( "%d %d %d\n" ,x3,x2,x1);
      } else {
            printf( "%d %d %d\n" ,x2,x3,x1);
      }
   } else {
      printf( "%d %d %d\n" ,x2,x1,x3);
   }
} else if (x1 >= x3) {
    printf( "%d %d %d\n" ,x3,x1,x2);
} else {
    if (x2 >= x3) {
       printf( "%d %d %d\n" ,x1,x3,x2);
    } else {
       printf( "%d %d %d\n" ,x1,x2,x3);
   }
}
```

- 將三十個整數由小到大印出來，還用上面方式寫嗎？？！！

6.1 Introduction

- 陣列(Arrays)
  - 一塊連續的記憶體空間。
  - 可以想像成連續的格子。

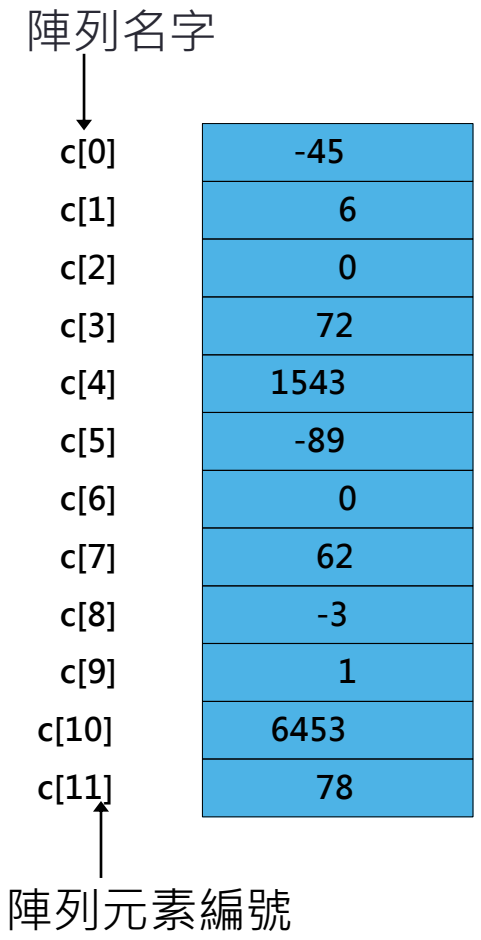
```
int main()
{
    int c[10]; /* an array of size 10 */
    int i;
    for(i=0; i<10;i++) c[i] = 0;
}
```



6.2 Arrays

- 陣列裡的元素
  - 相連的記憶體位置
  - 具有相同的變數名稱與資料型態
  - 可想像成櫃子裡的格子
- 使用陣列裡的元素
  - 陣列名稱
  - 元素在陣列裡的編號
- 語法:

  arrayname[*position number*]
  - **第一個在編號零的位置**(格子)
  - **N個元素的陣列最後一個在n-1那個位置**(格子)
    c[0], c[1]...c[n – 1]



6.2 Arrays

- 陣列裡的元素使用方法與一般變數相同

```
c[ 0 ] =  3;
printf( "%d", c[ 0 ] );
```

  - 陣列元素足標可以是運算式結果
    - **x = 3;**
      c[5 - 2] == c[3] == c[x]

6.3 Declaring Arrays

- 宣告陣列
  - 陣列名字(Name)
  - 陣列型態(Type of array)
  - 幾個元素(Number of elements)

與一般變數宣告方式相同的地方

共有多少元素

```
arrayType arrayName[ numberOfElements ];
```

  - Examples:

```
int c[ 10 ];
float myArray[ 3284 ];
```

- 宣告多個相同型態的陣列
  - Example:

```
int b[ 100 ], x[ 27 ];
```

習慣上會這麼宣告

```
#define C_SIZE 10
int c[C_SIZE];
```

C89宣告時不可以是變數，萬一就是變數時怎麼辦？？？

區域陣列，C99宣告時可以是變數。

常用的程式片段

- 將c的元素設為0
  - for(i=0; i<C_SIZE; ++i) c[i] = 0;
- 將資料讀入陣列c (共C_SIZE個元素)
  - for(i=0; i<C_SIZE; ++i) scanf("%d",&c[i]);
- 將資料讀入陣列c一直到零為止

```
i = 0; scanf("%d",&c[i]);
while(c[i] != 0) {
   i++;
   scanf("%d",&c[i]);
}
```

- 計算陣列c元素總和
  - for(sum=i=0; i<C_SIZE; ++i) sum += c[i];
- 計算陣列c元素a到元素b-1的總和
  - for(sum=0,i=a; i < b; ++i) sum += c[i];
- 找最大的元素
  - for(i=1, max_id = 0; i<C_SIZE; ++i) if (c[i]>c[max_id]) max_id = i;
  - for(i=1, max_val=c[0];i<C_SIZE; ++i) if (c[i]>max_val) max_val=c[i];

Program: reverse a series of numbers

```
#include <stdio.h>

#define N 10

int main(void)
{
  int a[N], i; // in C89, N cannot be a variable, whereas in C99, N may be a variable.

  printf("Enter %d numbers: ", N);
  for (i = 0; i < N; i++) {
    scanf("%d", &a[i]);
   }

  printf("In reverse order:");
  for (i = N - 1; i >= 0; i--) {
    printf(" %d", a[i]);
  }
  printf("\n");

  return 0;
}
```

6.4 Examples Using Arrays

- 初始化

  int n[5] = { 1, 2, 3, 4, 5 };

  - 若初始化個數少於陣列大小，剩下的補零

    int n[5] = { 0 }

    - n的所有元素初始值為0
    - This method of initializing the array elements to 0 is performed at compile time for static arrays and at run time for automatic arrays

```
void staticArrayInit() {
   static int array1[3];
}
void dynamicArrayInit() {
   int array2[3]={1,2,3};
}
```

    - 若初始化個數多於陣列大小會產生語法錯誤

- 如果陣列宣告沒寫大小，就由初始值個數決定

  int n[ ] = { 1, 2, 3, 4, 5 };

  - 5 個初始值, 因此陣列大小為5

練習

下面程式會輸出什麼？

```
#include<stdio.h>
void main()
{
        int a[10] = {9,8,7,6,5,4,3,2,1,0};

        printf( “%d\n” ,a[0]);
        printf( “%d\n” ,a[a[0]]);
        printf( “%d\n” ,a[a[a[0]]]);
        printf( “%d\n” ,a[a[a[a[0]]]]);
        printf( “%d\n” ,a[a[a[a[a[0]]]]]);
}
```

6.4 Examples Using Arrays

- 字元陣列(Character arrays)
  - 字元陣列可以用字串來給初始值

    char string1[] = "first";
    - 編譯器會自動加一個Null character **‘\0’ 在最後面**
    - **String1其實有六個元素**
      - 你可以這樣宣告，會有相同效果

```
char string1[] = { 'f', 'i', 'r', 's', 't', '\0' };
```

  - 你可以拿到每個字元

    string1[3] is character ‘s’
  - 陣列的名字就是代表陣列的開始位置

```
scanf( "%s", string2 );
scanf( “%s”,&string2[0]);
```

    - 當輸入的字元數目比你宣告的陣列大時，會發生什麼事呢？

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

int main()
{
   char string1[ 20 ], string2[] = "string literal";
   int i;

   printf(" Enter a string: ");
   scanf( "%s", string1 );
   printf( "string1 is: %s\nstring2: is %s\n"
           "string1 with spaces between characters is:\n",
           string1, string2 );

   for ( i = 0; string1[ i ] != '\0'; i++ )
      printf( "%c ", string1[ i ] );

   printf( "\n" );
   return 0;
}
```

```
Enter a string: Hello there
string1 is: Hello
string2 is: string literal
string1 with spaces between characters is:
H e l l o
```

1. Initialize strings

2. Print strings

2.1 Define loop

2.2 Print characters individually

2.3 Input string

3. Print string

Program Output

```c
/* Fig. 6.8: fig06_08.c
   Histogram printing program */
#include <stdio.h>
#define SIZE 10

int main()
{
   int n[ SIZE ] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };
   int i, j;

   printf( "%s%13s%17s\n", "Element", "Value", "Histogram" );

   for ( i = 0; i < SIZE; i++ ) {
      printf( "%7d%13d        ", i, n[ i ]) ;

      for ( j = 1; j <= n[ i ]; j++ )   /* print one bar */
         printf( "%c", '*' );

      printf( "\n" );
   }

   return 0;
}
```

```
Element        Value        Histogram
      0           19        *******************
      1            3        ***
      2           15        ***************
      3            7        *******
      4           11        ***********
      5            9        *********
      6           13        *************
      7            5        *****
      8           17        *****************
      9            1        *
```

```
#include<stdio.h>
#define C_SIZE 11
#define score_SIZE 90
int main(void)
{
  int N, count[C_SIZE]={0},  score[score_SIZE];
  printf("Enter %d numbers: ", N);
  for (i = 0; i < N; i++) {
   scanf("%d", &score[i]); // score[i] is between 0 and 100
   }
```

```
for(i = 0; i < N; i++) {
    if (score[i] < 10) {
       count[0]++; // count the number of scores less than 10
    } else if (score[i] < 20) {
       count[1]++; // count the number of scores in [10,20)
    } else if  (score[i] < 30) {
       count[2]++; // count the number of scores in [20,30)
    } else if (score[i] < 40) {
       count[3]++; // count the number of scores in [30,40)
    } else if (score[i] < 50) {
       count[4]++; // count the number of scores in [40,50)
    } else if (score[i] < 60) {
       count[5]++; // count the number of scores in [50,60)
    } else if (score[i] < 70) {
       count[6]++; // count the number of scores in [60,70)
    } else if (score[i] < 80) {
       count[7]++; // count the number of scores in [70,80)
    } else if (score[i] < 90) {
       count[8]++; // count the number of scores in [80,90)
    } else if (score[i] < 100) {
       count[9]++; // count the number of scores in [90,100)
    } else {
       count[10]++; // count the number of scores equal to100
    }
  }
```

Ugly

```
}
```

```
for(i = 0; i < N; i++) {
   for(j = 0; j <= 10; ++j)
      if (score[i] >= j*10 && score[i] < (j+1)*10) {
         count[j]++; // count the number of scores in [j*10,j*10+10)
      }
   }
}
```

More pretty but still inefficient

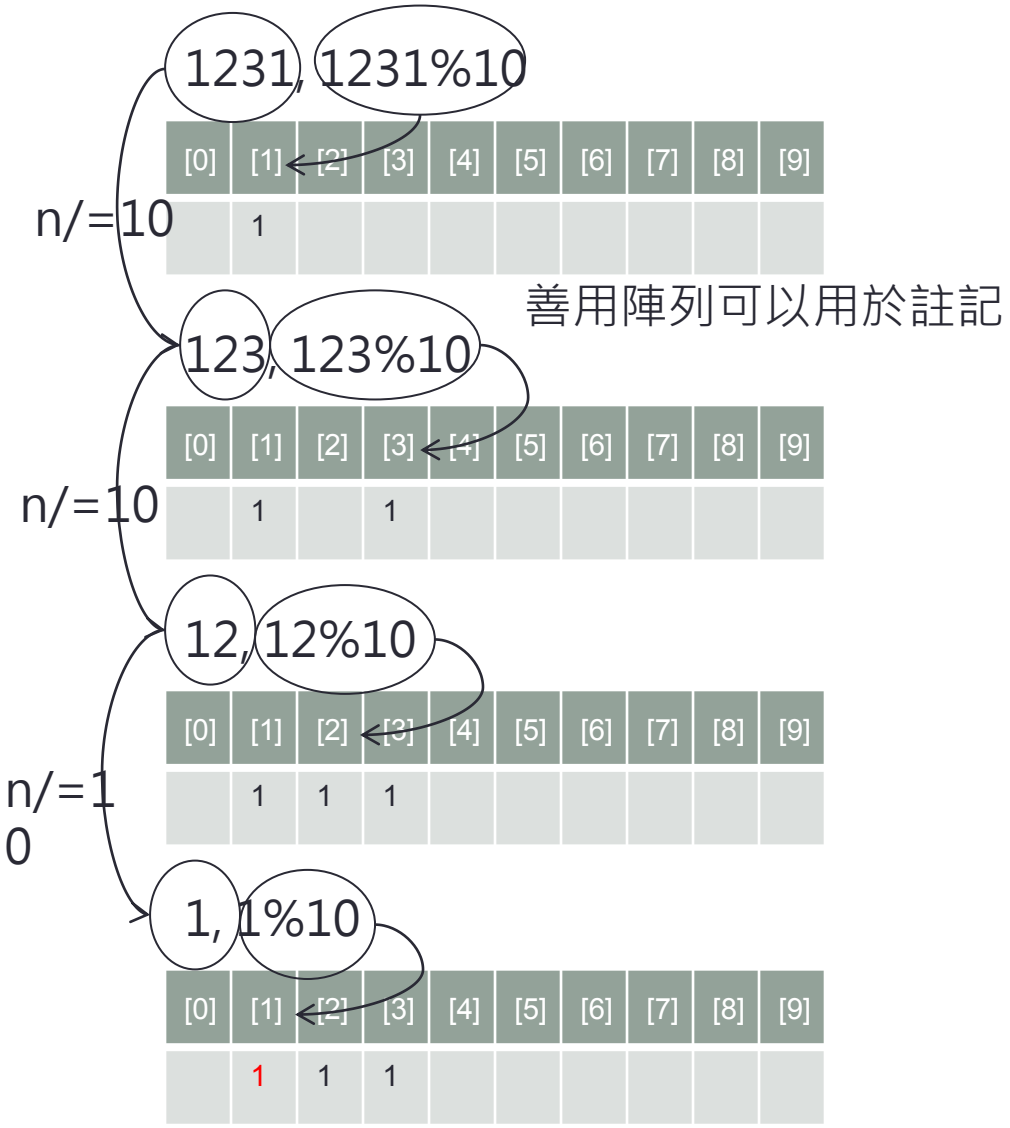
```
for(i = 0; i < N; i++) {
      count[score[i]/10]++;
}
```

Much better

善用陣列可以簡化程式

範例: check a number for repeated digits

```
enum _bool   {false, true };
typedef enum _bool bool;
#include <stdio.h>

int main(void)
{
  bool digit_seen[10] = {false};
  int digit;
  long n;

  printf("Enter a number: ");
  scanf("%ld", &n);

  while (n > 0) {
    digit = n % 10;
    if (digit_seen[digit])
      break;
    digit_seen[digit] = true;
    n /= 10;
  }

  if (n > 0)
    printf("Repeated digit\n");
  else
    printf("No repeated digit\n");

  return 0;
}
```



例子: 求質數 (The sieve method)

```
#include  <stdio.h>

#define   MAXSIZE   200
#define   DELETED   1
#define   KEPT      0

int main(void)
{
    int  sieve[MAXSIZE+1]={0};   /* the sieve array   sieve[i] is a flag for 2i+3      */
    int  prime;
    int  i, k, count = 1;


    for (i = 0; i <= MAXSIZE; i++){ /* for each i, it corresponds to 2i+3     */
        if (sieve[i] == KEPT) {/* if it is not sieved, 2i+3 is a prime number*/
            prime = i + i + 3; /* prime=2i+3.       */
            count++;
           for (k = prime + i; k <= MAXSIZE; k += prime){
               sieve[k] = DELETED; /* screen multiple*/
           }
        }
     }
    printf("There are %d prime numbers between 2 and %d\n", count,MAXSIZE*2+3);
    printf("\n%6d", 2);      /* output prime numbers.       */
    for (i = 0, k = 2; i <= MAXSIZE; i++) {
        if (sieve[i] == KEPT) {
            if (k > 10) {
                printf("\n");
                k = 1;
            }
            printf("%6d", 2*i+3);
            k++;
        }
    }
   return 0;
}
```

```
There are 79 prime numbers between 2 and 403

     2     3     5     7    11    13    17    19    23    29
    31    37    41    43    47    53    59    61    67    71
    73    79    83    89    97   101   103   107   109   113
   127   131   137   139   149   151   157   163   167   173
   179   181   191   193   197   199   211   223   227   229
   233   239   241   251   257   263   269   271   277   281
   283   293   307   311   313   317   331   337   347   349
   353   359   367   373   379   383   389   397   401
```

解說: 從最簡單的想法開始

char sieve[MAXSIZE]={0}; //sieve[u]註記u是否為質數。若u是，sieve[u]==0

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
0	0	0	0	0	0	0	0	0	0	0	0	0

從2開始將2的倍數標記他們不是質數 for(u=4; u<MAXSIZE; u+=2) sieve[u]=DELETED;

[0]	[1]	**[2]**	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
0	0	0	0	1	0	1	0	1	0	1	0	1

接下來將3的倍數標記他們不是質數for(u=6; u<MAXSIZE; u+=3) sieve[u]=DELETED;

[0]	[1]	[2]	**[3]**	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
0	0	0	0	1	0	1	0	1	1	1	0	1

接下來因為4不是質數因為他被他的因數標示過了，所以不必處理。

[0]	[1]	[2]	[3]	**[4]**	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
0	0	0	0	1	0	1	0	1	1	1	0	1

接下來將5的倍數標記他們不是質數for(u=10; u<MAXSIZE; u+=5) sieve[u]=DELETED;

[0]	[1]	[2]	[3]	[4]	**[5]**	[6]	[7]	[8]	[9]	[10]	[11]	[12]
0	0	0	0	1	0	1	0	1	1	1	0	1

將上述想法寫成程式

```
for(p=2; p < MAXSIZE; ++p) {
  if (sieve[p]==KEPT) {
    for(u=p+p; u <MAXSIZE; u+=p) {
      sieve[u] = DELETED;
    }
  }
}
```

將p的倍數標記他們不是質數的迴圈

由於0,1不是質數，所有偶數只有2是質數。上述程式可以再精進成陣列sieve只用於標註>=3的奇數是否為質數，質數2當成特例處理即可。
所以，可以做下面改變。

char sieve[MAXSIZE]={0}; //sieve[p]表示2p+3這個數字是否為質數的註記

根據這個方法的原則若2p+3為質數，那麼它的倍數皆不是質數，其中只有奇數倍數在sieve裡要被標註。

2p+3的奇數倍數為3*(2p+3), 5*(2p+3), 7*(2p+3),...。如果將他們整理成2k+3的形式，那麼他們在sieve的足標就是k。

2p+3的奇數倍數可以整理為
2(p+(2p+3))+3,2(p+2(2p+3))+3,2(p+3(2p+3))+3,...
2p+3的奇數倍數的足標為p+(2p+3),p+2(2p+3),p+3(2p+3),...

程式因此可以寫成

```
for(p=0; p < MAXSIZE; ++p) {
  if (sieve[p]==KEPT) {
     int prime=2*p+3;
     for(u=p+prime; u <MAXSIZE; u+=prime) {
        sieve[u] = DELETED;
     }
  }
}
```

檢查陣列範圍

```
#define TABLE_SIZE 100

int table[TABLE_SIZE];

size_t index; // unsigned index
//...
if (index < TABLE_SIZE) {
   x = table[index];
} else {
   // error handling
}
```

```
int index;
//...
if (index>=0&& index <
TABLE_SIZE) {
   x = table[index];
} else {
   // error handling
}
```

6.5 Passing Arrays to Functions

- 將陣列為函式參數時
  - 只要寫陣列名稱

    int myArray[24];

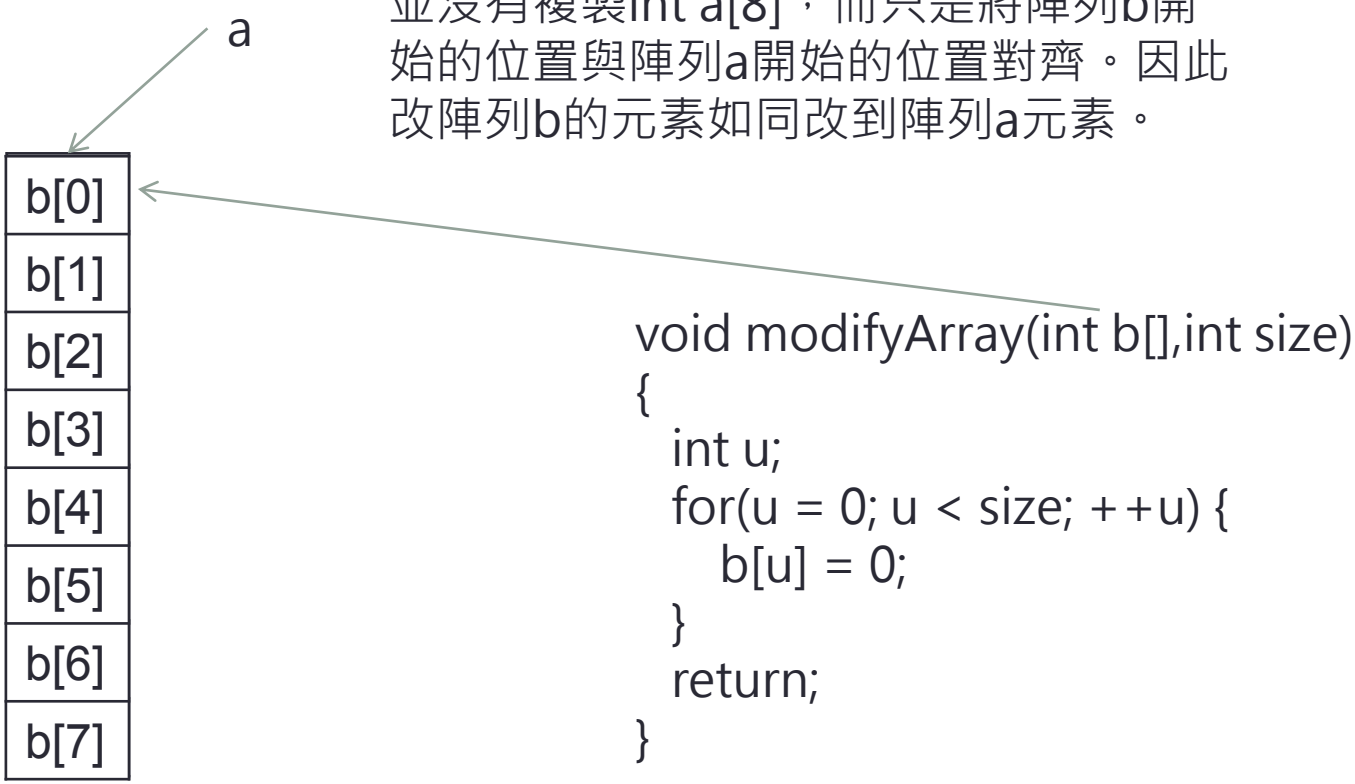
    myFunction(myArray, 24);
  - 陣列的啟始位置會被傳遞給函式
  - 24告知myFunction，myArray共24元素
- 將陣列某個元素傳遞給函式時
  - Passed by call-by-value
  - Pass subscripted name (i.e., **myArray[3]**) to function
- 函式原型(Function prototype)可以宣告如下：

  void modifyArray(int b[], int arraySize);
  - 參數名字可寫可不寫 **void modifyArray(int [], int);**
    - **int b[]** could be written **int []**
    - **int arraySize** could be simply **int**

```
/* Fig. 6.12: fig06_12.c
   The name of an array is the same as &array[ 0
] */
#include <stdio.h>

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

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

   return 0; /* indicates successful termination */

} /* end main */
```

```
int a[8];

modifyArray(a,8);
```

並沒有複製int a[8]，而只是將陣列b開始的位置與陣列a開始的位置對齊。因此改陣列b的元素如同改到陣列a元素。




```
void modifyArray(int b[],int size)
{
   int u;
   for(u = 0; u < size; ++u) {
      b[u] = 0;
   }
   return;
}
```

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

void modifyArray( int [], int );  /* appears strange */
void modifyElement( int );

int main()
{
   int a[ SIZE ] = { 0, 1, 2, 3, 4 }, i;

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

   for ( i = 0; i < SIZE; i++ )
      printf( "%3d", a[ i ] );

   printf( "\n" );
   modifyArray( a, SIZE );  /* passed call by reference */
   printf( "The values of the modified array are:\n" );

   for ( i = 0; i < SIZE; i++ )
      printf( "%3d", a[ i ] );

   printf( "\n\n\nEffects of passing array element call "
           "by value:\n\nThe value of a[3] is %d\n", a[ 3 ] );
   modifyElement( a[ 3 ] );
   printf( "The value of a[ 3 ] is %d\n", a[ 3 ] );
   return 0;
}
```

1. Function definitions

2. Pass array to a function

2.1 Pass array element to a function

3. Print

Entire arrays passed call-by-reference, and can be modified

Array elements passed call-by-value, and cannot be modified

```
33
34 void modifyArray( int b[], int size )
35 {
36    int j;
37
38    for ( j = 0; j < size; j++ )
39       b[ j ] *= 2;
40 }
41
42 void modifyElement( int e )
43 {
44    printf( "Value in modifyElement is %d\n", e *= 2 );
45 }
```

3.1 Function definitions

Program Output

```
Effects of passing entire array call 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 call by value:


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

6.6 資料排序(1/3) : Bubble Sort

- **Bubble sort (sinking sort)**
  - 由足標0開始比較相鄰兩元素大小
    - 若足標小元素的值較大, 交換兩元素
  - 重複上述輪回至整輪回沒有資料交換為止

```
/* loop to control number of passes */
 for (pass = 1,exchange=1; exchange==1&&pass < SIZE; pass++ ) {
   /* loop to control number of comparisons per pass */
   for (exchange = i = 0; i < SIZE - pass; i++ ) {
     /* compare adjacent elements and swap them if first
     element is greater than second element */
     if (a[ i ] > a[ i + 1 ] ) {
       hold = a[ i ];
       a[ i ] = a[ i + 1 ];
       a[ i + 1 ] = hold;
        exchange = 1;
     } /* end if */
   } /* end inner for */
 }
```

Pass=1

a[0]	a[1]	a[2]	a[3]	a[4]
6	5	3	2	4

Compare a[0],a[1]

a[0]	a[1]	a[2]	a[3]	a[4]
5	6	3	2	4

Compare a[1],a[2]

a[0]	a[1]	a[2]	a[3]	a[4]
5	3	6	2	4

Compare a[2],a[3]

a[0]	a[1]	a[2]	a[3]	a[4]
5	3	2	6	4

Compare a[3],a[4]

a[0]	a[1]	a[2]	a[3]	a[4]
5	3	2	4	6

Pass=2

a[0]	a[1]	a[2]	a[3]	a[4]
5	3	2	4	6

a[0]	a[1]	a[2]	a[3]	a[4]
3	5	2	4	6

a[0]	a[1]	a[2]	a[3]	a[4]
3	2	5	4	6

a[0]	a[1]	a[2]	a[3]	a[4]
3	2	4	5	6

Pass=3

a[0]	a[1]	a[2]	a[3]	a[4]
3	2	4	5	6

a[0]	a[1]	a[2]	a[3]	a[4]
2	3	4	5	6

a[0]	a[1]	a[2]	a[3]	a[4]
2	3	4	5	6

```
for (p= 1,ch=1; ch==1&&
                        p < SIZE; p++ ) {
    for (ch = i = 0; i < SIZE - p; i++ ) {
      if ( a[ i ] > a[ i + 1 ] ) {
        hold = a[ i ];
        a[ i ] = a[ i + 1 ];
        a[ i + 1 ] = hold;
        ch= 1;
      } /* end if */
    } /* end inner for */
  }
```

	a[i+1]						
a[i]		[0]	[1]	[2]	…	[n-pass-1]	[n-pass]
	[0]						
	[1]						
	[2]						
	…						
	[n-pass-1]						

```
for(i=0;i<n-pass; i++) {
    if (a[ i ] > a[ i + 1 ] ) {
        …..
    }
}
```

6.6 資料排序(2/3): Selection Sort

- **Selection sort**
  - 選擇最小的元素，與足標0元素交換內容
  - 在剩下SIZE-1資料選擇最小的元素，與足標1元素交換內容
  - 在剩下SIZE-2資料選擇最小的元素，與足標2元素交換內容
  - 直到剩下1個資料。

```
//Selection sort
for (p = 0; p < SIZE-1; p++ ) {
  for (smallest=p, i = p+1; i < SIZE; i++ ) {
       if (a[ i ] < a[ smallest ] ) {
          smallest=i;
        } /* end if */
  } /* end inner for */
   hold           = a[p];
   a[p]           = a[smallest];
   a[smallest]  = hold;
}
```

P=0

a[0]	a[1]	a[2]	a[3]	a[4]
6	5	3	2	4

a[3] is the smallest

a[0]	a[1]	a[2]	a[3]	a[4]
6	5	3	2	4

a[0]	a[1]	a[2]	a[3]	a[4]
2	5	3	6	4

P=1

a[0]	a[1]	a[2]	a[3]	a[4]
2	5	3	6	4

a[2] is the smallest

a[0]	a[1]	a[2]	a[3]	a[4]
2	5	3	6	4

a[0]	a[1]	a[2]	a[3]	a[4]
2	3	5	6	4

P=2

a[0]	a[1]	a[2]	a[3]	a[4]
2	3	5	6	4

a[4] is the smallest

a[0]	a[1]	a[2]	a[3]	a[4]
2	3	5	6	4

a[0]	a[1]	a[2]	a[3]	a[4]
2	3	4	6	5

```
//Selection sort
for (p = 0; p < SIZE-1; p++ ) {
  for (j=p, i = p+1; i < SIZE; i++ ) {
        if (a[ i ] < a[ j ] ) {
           j=i;
         } /* end if */
 } /* end inner for */
 hold = a[p];
 a[p] = a[j];
 a[j]  = hold;
}
```

6.6 資料排序(3/3): Insertion Sort

- Insertion sort
  - 由足標p=1開始，維持資料由小到大順序將a[p]插入a[0],...,a[p-1]

```
//Insertion sort
for (p = 1; p < SIZE; p++ ) {
  hold = a[p];
  for (i = p-1; i >=0; i-- ) {
      if (a[ i ] > hold ) {
         a[i+1] = a[i];
       } else {
         break;
       }
  } /* end inner for */
  a[i+1] = hold;
}
```

P=1

a[0]	a[1]	a[2]	a[3]	a[4]
6	5	3	2	4

Insert 5 into this subsequence

a[0]	a[1]	a[2]	a[3]	a[4]
5	6	3	2	4

P=2

a[0]	a[1]	a[2]	a[3]	a[4]
5	6	3	2	4

Insert 3 into this subsequence

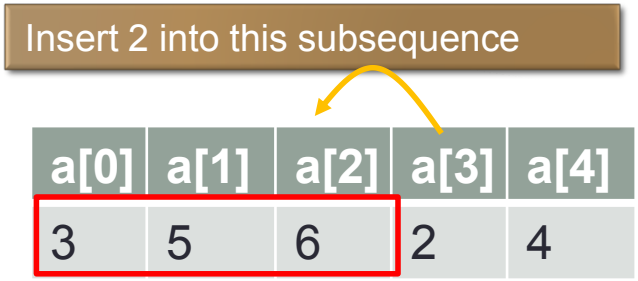
a[0]	a[1]	a[2]	a[3]	a[4]
3	5	6	2	4

P=3

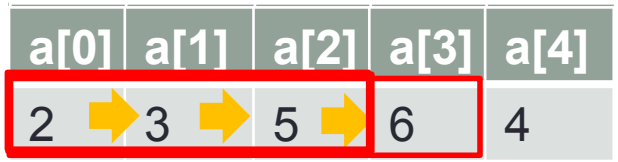
a[0]	a[1]	a[2]	a[3]	a[4]
3	5	6	2	4

Insert 2 into this subsequence

a[0]	a[1]	a[2]	a[3]	a[4]
2	3	5	6	4



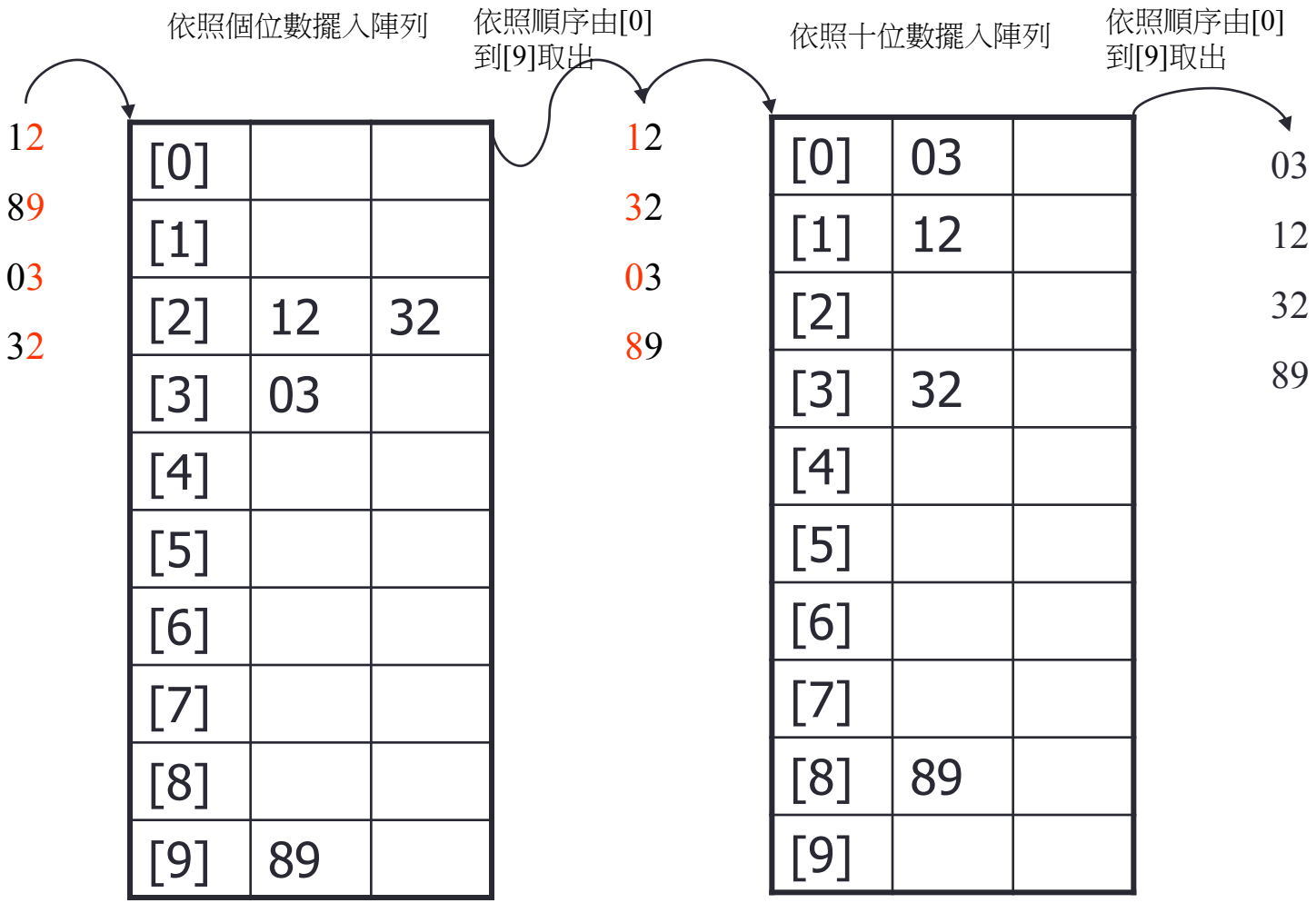
hold=a[3]
2



A longer sorted subsequence

```
//Insertion sort
for (p = 1; p < SIZE; p++ ) {
  hold = a[p];
  for (i = p-1; i >=0; i-- ) {
        if (a[ i ] > hold ) {
           a[i+1] = a[i];
         } else {
           break;
         }
   } /* end inner for */
   a[i+1] = hold;
}
```

Bucket sort


依照個位數擺入陣列
依照順序由[0]到[9]取出
依照十位數擺入陣列
依照順序由[0]到[9]取出
12
89
03
32
[0]
[1]
[2] 12 32
[3] 03
[4]
[5]
[6]
[7]
[8]
[9] 89
12
32
03
89
[0] 03
[1] 12
[2]
[3] 32
[4]
[5]
[6]
[7]
[8] 89
[9]
03
12
32
89

6.7 Case Study: Computing Mean, Median and Mode Using Arrays

- Mean – average
- Median – number in middle of sorted list
  - 1, 2, 3, 4, 5
  - 3 is the median
- Mode – number that occurs most often
  - 1, 1, 1, 2, 3, 3, 4, 5
  - 1 is the mode

平均值與中值

- 解$\text{argmin}_m \frac{1}{n}\sum_{i=1}^{n}(x_i - m)^2$

  答案: m為$x_1, x_2, \ldots, x_n$的平均值$m = \frac{1}{n}\sum_{i=1}^{n} x_i$。

- 解$\text{argmin}_m \frac{1}{n}\sum_{i=1}^{n}|x_i - m|$

  答案: m為$x_1, x_2, \ldots, x_n$的中值。

```c
/* Fig. 6.16: fig06_16.c
   This program introduces the topic of survey data analysis.
   It computes the mean, median, and  mode of the data */
#include <stdio.h>
#define SIZE 99

void mean( const int [] );
void median( int [] );
void mode( int [], const int [] ) ;
void bubbleSort( int [] );
void printArray( const int [] );

int main()
{
   int frequency[ 10 ] = { 0 };
   int response[ SIZE ] =
      { 6, 7, 8, 9, 8, 7, 8, 9, 8, 9,
        7, 8, 9, 5, 9, 8, 7, 8, 7, 8,
        6, 7, 8, 9, 3, 9, 8, 7, 8, 7,
        7, 8, 9, 8, 9, 8, 9, 7, 8, 9,
        6, 7, 8, 7, 8, 7, 9, 8, 9, 2,
        7, 8, 9, 8, 9, 8, 9, 7, 5, 3,
        5, 6, 7, 2, 5, 3, 9, 4, 6, 4,
        7, 8, 9, 6, 8, 7, 8, 9, 7, 8,
        7, 4, 4, 2, 5, 3, 8, 7, 5, 6,
        4, 5, 6, 1, 6, 5, 7, 8, 7 };

   mean( response );
   median( response );
   mode( frequency, response );
   return 0;
}
```

1. Function prototypes

1.1 Initialize array

2. Call functions `mean`, `median`, and `mode`

```
33
34 void mean( const int answer[] )
35 {
36    int j, total = 0;
37
38    printf( "%s\n%s\n%s\n", "********", "  Mean", "********" );
39
40    for ( j = 0; j <= SIZE - 1; j++ )
41       total += answer[  j  ];
42
43    printf( "The mean is the average value of the data\n"
44            "items. The mean is equal to the total of\n"
45            "all the data items divided by the number\n"
46            "of data items ( %d ). The mean value for\n"
47            "this run is: %d / %d = %.4f\n\n",
48            SIZE, total, SIZE, (  double  ) total / SIZE );
49 }
50
51 void median( int answer[] )
52 {
53    printf( "\n%s\n%s\n%s\n%s",
54            "********", " Median", "********",
55            "The unsorted array of responses is" );
56
57    printArray( answer );
58    bubbleSort( answer );
59    printf( "\n\nThe sorted array is" );
60    printArray( answer );
61    printf( "\n\nThe median is element %d of\n"
62            "the sorted %d element array.\n"
63            "For this run the median is %d\n\n",
64            SIZE / 2, SIZE, answer[ SIZE / 2 ] );
```

```
65 }
66
67 void mode( int freq[], const int answer[] )
68 {
69    int rating, j, h, largest = 0, modeValue = 0;
70
71    printf( "\n%s\n%s\n%s\n",
72            "********", "  Mode", "******** " );
73
74    for ( rating = 1; rating <= 9; rating++ )
75       freq[ rating ] = 0;
76
77    for ( j = 0; j <= SIZE - 1; j++ )
78       ++freq[ answer[ j ] ];
79
80    printf( "%s%11s%19s\n\n%54s\n%54s\n\n",
81            "Response", "Frequency", "Histogram",
82            "1    1    2    2", "5    0    5    0    5" );
83
84    for ( rating = 1; rating <= 9; rating++ ) {
85       printf( "%8d%11d          ", rating, freq[ rating ] );
86
87       if ( freq[ rating ] > largest ) {
88          largest = freq[ rating ];
89          modeValue = rating;
90       }
91
92       for ( h = 1; h <= freq[ rating ]; h++ )
93          printf( "*" );
94
```

Notice how the subscript in **`frequency[]`** is the value of an element in **`response[]`** (**`answer[]`**)

Print stars depending on value of **`frequency[]`**

```
95       printf( "\n" );
96    }
97
98    printf( "The mode is the most frequent value.\n"
99            "For this run the mode is %d which occurred"
100           " %d times.\n", modeValue, largest );
101}
102
103 void bubbleSort( int a[] )
104 {
105   int pass, j, hold;
106
107   for ( pass = 1; pass <= SIZE - 1; pass++ )
108
109      for ( j = 0; j <= SIZE - 2; j++ )
110
111         if ( a[ j ] > a[ j + 1 ] ) {
112            hold = a[ j ];
113            a[ j ] = a[ j + 1 ];
114            a[ j + 1 ] = hold;
115         }
116 }
117
118 void printArray( const int a[] )
119 {
120   int j;
121
122   for ( j = 0; j <= SIZE - 1; j++ ) {
123
124      if ( j % 20 == 0 )
125         printf( "\n" );
```

Bubble sort: if elements out of order, swap them.

```
126
127       printf( "%2d", a[ j ] );
128    }
129 }
```

```
********
 Mean
********
The mean is the average value of the data
items. The mean is equal to the total of
all the data items divided by the number
of data items (99). The mean value for
this run is: 681 / 99 = 6.8788

********
 Median
********
The unsorted array of responses is
7 8 9 8 7 8 9 8 9 7 8 9 5 9 8 7 8 7 8
 6 7 8 9 3 9 8 7 8 7 7 8 9 8 9 8 9 7 8 9
 6 7 8 7 8 7 9 8 9 2 7 8 9 8 9 8 9 7 5 3
 5 6 7 2 5 3 9 4 6 4 7 8 9 6 8 7 8 9 7 8
 7 4 4 2 5 3 8 7 5 6 4 5 6 1 6 5 7 8 7

The sorted array is
 1 2 2 2 3 3 3 3 4 4 4 4 4 5 5 5 5 5 5 5
 5 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7
 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9

The median is element 49 of
the sorted 99 element array.
For this run the median is 7
```

```
********
  Mode
********
Response  Frequency        Histogram

                                  1    1    2    2
                              5   0    5    0    5

       1          1          *
       2          3          ***
       3          4          ****
       4          5          *****
       5          8          ********
       6          9          *********
       7         23          ***********************
       8         27          ***************************
       9         19          *******************
The mode is the most frequent value.
For this run the mode is 8 which occurred 27 times.
```

6.8 Searching Arrays: Linear Search and Binary Search

- 在陣列裡找含key value的資料
  Search an array for a *key value*
- 線性搜尋(Linear search) (fig06_18.c)
  - Simple
  - Compare each element of array with key value
  - Useful for small and unsorted arrays

```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
   size_t x; // counter for initializing elements 0-99 of array a
   int searchKey; // value to locate in array a
   size_t element; // variable to hold location of searchKey or -1

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

   puts( "Enter integer search key:" );
   scanf( "%d", &searchKey );

   // attempt to locate searchKey in array a
   element = linearSearch( a, searchKey, SIZE );

   // display results
   if ( element != -1 ) {
      printf( "Found value in element %d\n", element );
   } // end if
   else {
      puts( "Value not found" );
   } // end else
} // end main
```

```c
// compare key to every element of array until the location is found
// or until the end of array is reached; return subscript of element
// if key is found or -1 if key is not found
size_t linearSearch( const int array[], int key, size_t size )
{
   size_t n; // counter

   // loop through array
   for ( n = 0; n < size; ++n ) {

      if ( array[ n ] == key ) {
         return n; // return location of key
      } // end if
   } // end for

   return -1; // key not found
} // end function linearSearch
```

6.8 Searching Arrays: Linear Search and Binary Search

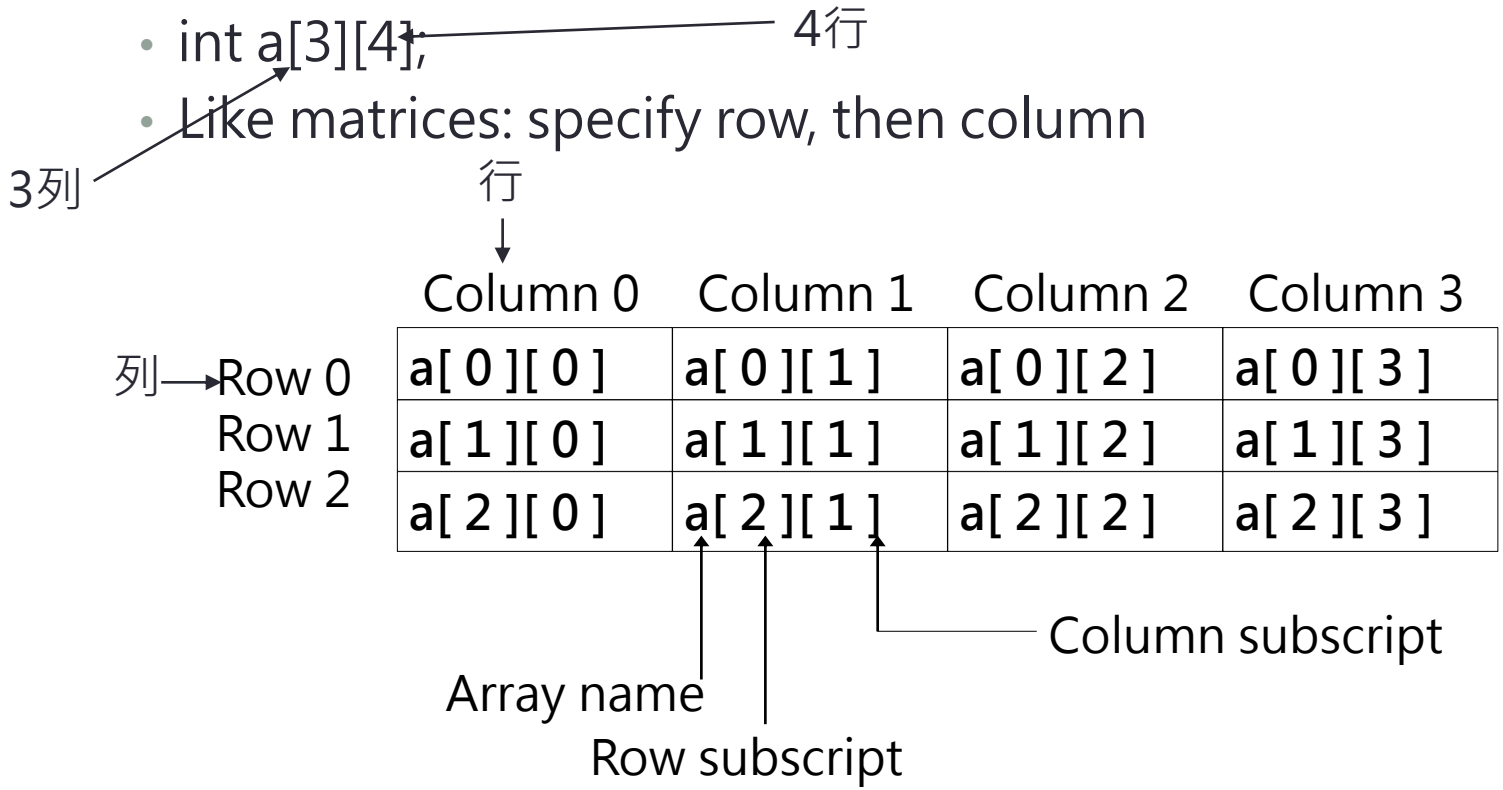
- **二元搜尋(Binary search)** (fig06_19.c)
  - 資料必須排序(For sorted arrays)
  - Compares **middle** element with **key**
    - If equal, match found
    - If **key** < **middle**, looks in first half of array
    - If **key** > **middle**, looks in last half
    - Repeat
  - 非常快：2^n筆資料只要比較n次即可
    at most n steps, where 2^n > number of elements
    - 30 element array takes at most 5 steps
      - 2^5 > 30 so at most 5 steps

binarySearch(data,key,0,n-1);

```
41 // function to perform binary search of an array
42 size_t binarySearch(const int b[], int searchKey, size_t low, size_t high)
43 {
44    int middle; // variable to hold middle element of array
45
46    // loop until low subscript is greater than high subscript
47    while ( low <= high ) {
48
49       // determine middle element of subarray being searched
50       middle = ( low + high ) / 2;
51
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
```

6.9 Multiple-Subscripted Arrays

- 多維陣列(Multiple subscripted arrays)
  - int a[3][4];
  - Like matrices: specify row, then column



6.9 Multiple-Subscripted Arrays

- 初始化
  - int b[2][2] = { { 1, 2 }, { 3, 4 } };
    - 如果初始值數目不夠則當成0

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

1	0
3	4

b[0][0] b[0][1]

1	2
3	4

b[1][0] b[1][1]

- 存取陣列裡元素[row][column]，先列然後行
  - printf("%d", b[0][1]);

```c
/* Fig. 6.22: fig06_22.c
   Double-subscripted array example */
#include <stdio.h>
#define STUDENTS 3
#define EXAMS 4

int minimum( const int [][ EXAMS ], int, int );
int maximum( const int [][ EXAMS ], int, int );
double average( const int [], int );
void printArray( const int [][ EXAMS ], int, int );

int main()
{
   int student;
   const int studentGrades[ STUDENTS ][ EXAMS ] =
      { { 77, 68, 86, 73 },
        { 96, 87, 89, 78 },
        { 70, 90, 86, 81 } };

   printf( "The array is:\n" );
   printArray( studentGrades, STUDENTS, EXAMS );
   printf( "\n\nLowest grade: %d\nHighest grade: %d\n",
           minimum( studentGrades, STUDENTS, EXAMS ),
           maximum( studentGrades, STUDENTS, EXAMS ) );

   for ( student = 0; student <= STUDENTS - 1; student++ )
      printf( "The average grade for student %d is %.2f\n",
              student,
              average( studentGrades[ student ], EXAMS ) );

   return 0;
}
```

Each row is a particular student, each column is the grades on the exam.

```
33
34 /* Find the minimum grade */
35 int minimum( const int grades[][ EXAMS ],
36              int pupils, int tests )
37 {
38    int i, j, lowGrade = 100;
39
40    for ( i = 0; i <= pupils - 1; i++ )
41       for ( j = 0; j <= tests - 1; j++ )
42          if ( grades[ i ][ j ] < lowGrade )
43             lowGrade = grades[ i ][ j ];
44
45    return lowGrade;
46 }
47
48 /* Find the maximum grade */
49 int maximum( const int grades[][ EXAMS ],
50              int pupils, int tests )
51 {
52    int i, j, highGrade = 0;
53
54    for ( i = 0; i <= pupils - 1; i++ )
55       for ( j = 0; j <= tests - 1; j++ )
56          if ( grades[ i ][ j ] > highGrade )
57             highGrade = grades[ i ][ j ];
58
59    return highGrade;
60 }
61
62 /* Determine the average grade for a particular exam */
63 double average( const int setOfGrades[], int tests )
64 {
```

```
65    int i, total = 0;
66
67    for ( i = 0; i <= tests - 1; i++ )
68       total += setOfGrades[ i ];
69
70    return ( double ) total / tests;
71 }
72
73 /* Print the array */
74 void printArray( const int grades[][ EXAMS ],
75                  int pupils, int tests )
76 {
77    int i, j;
78
79    printf( "                 [0]  [1]  [2]  [3]" );
80
81    for ( i = 0; i <= pupils - 1; i++ ) {
82       printf( "\nstudentGrades[%d] ", i );
83
84       for ( j = 0; j <= tests - 1; j++ )
85          printf( "%-5d", grades[ i ][ j ] );
86    }
87 }
```

```
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
```

多維陣列與指標

```
int a[10][20];
int (*b)[20];
int *c;

c = a[5];
printf("%d\n",&c[0]==&a[5][0]); // output 1

b = a+2;
printf("%d\n",&b[0][0] == &a[2][0]); // output 1
```

等價的操作！ a[1]-a[0] => 差多少單位

```
int a[10][20][2];
int (*b)[20][2];
int (*c)[2];
int *d;
d = a[2][7]+1;
printf("%d\n",&d [0]==&a[2][7][1]); // output 1

c = a[5]+3;
printf("%d\n",&c[0][0]==&a[5][3][0]); // output 1

b = a+2;
printf("%d\n",&b[0][0][0] == &a[2][0][0]); // output 1
```

int *

int (*)[2]

int (*)[20][2]

6.10 Variable-Length Arrays

- C99標準有可變長度陣列，早期版本的C使用記憶體動態配置的方式(malloc)達到類似效果。
- 目前不是所有C編譯器都支援此特徵(如Microsoft Visual C++不支援)。

```c
// Fig. 6.23: figG_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

   // 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
```