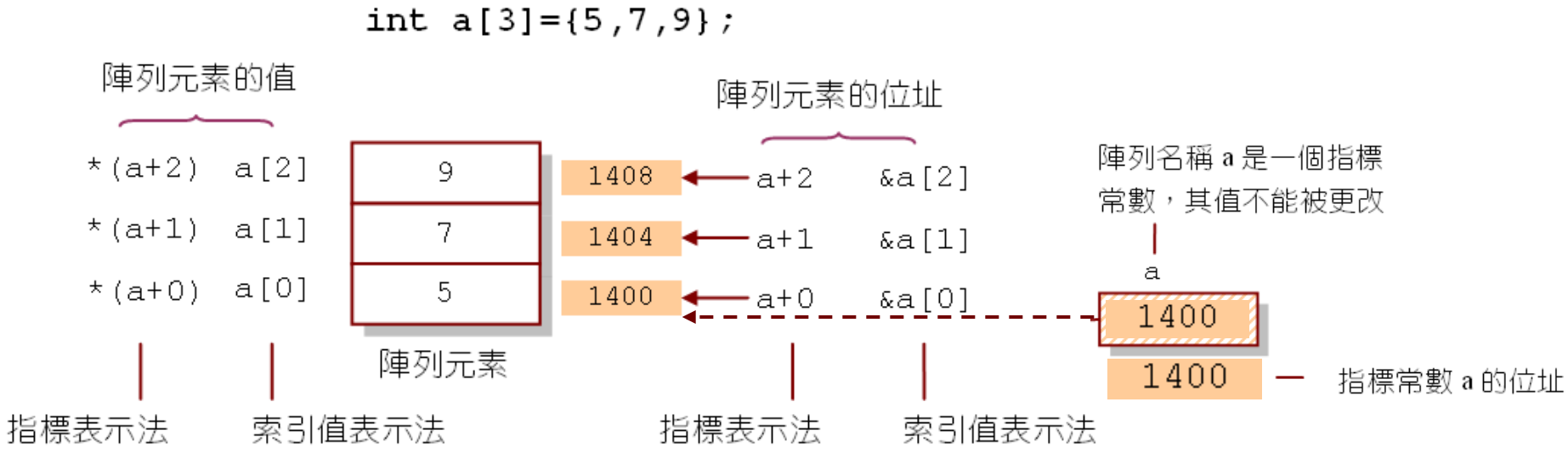
第十章
再談指標

- 學習指標變數的使用
- 認識動態記憶體配置
- 瞭解指標與參照的不同
- 認識指標與參照在函數之間的傳遞方式

陣列與指標

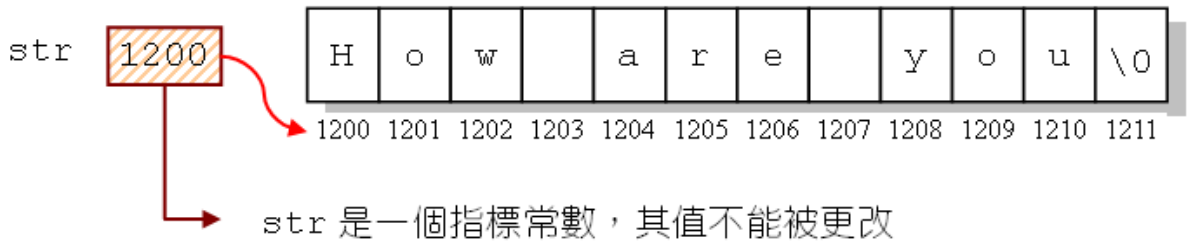
- 陣列a於記憶體中的配置



字串指標變數字串字面值 (1/2)

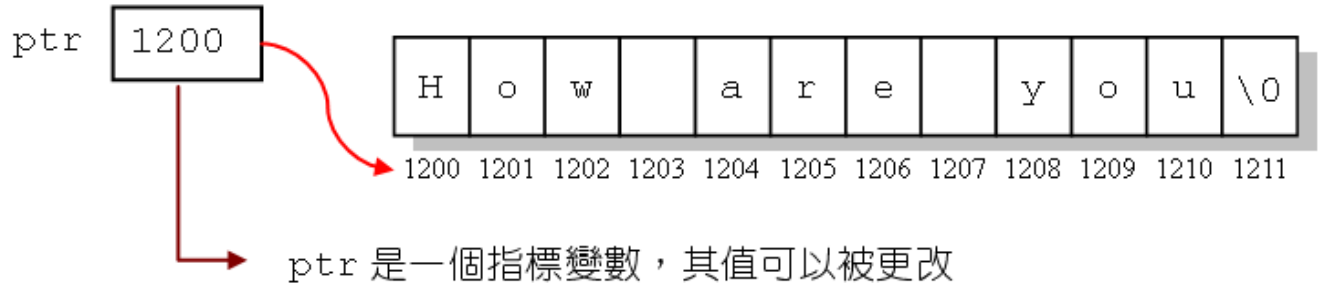
- 字面值 "How are you?" 可以利用字元陣列來儲存

```
char str[]="How are you?";
```



- 字面值可以利用指向字元的指標ptr指向它

```
char *ptr="How are you?";
```



以指標變數指向字串 (2/2)

- 下面是利用指標來指向字串的練習

```
01  // prog9_20, 以指標變數指向字串
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07     char name[20];
08     char *ptr="How are you?";          // 將指標指向字串"How are you?"
09
10     cout << "What's your name? ";
11     cin.getline(name,20);                                  // 輸入字串
12     cout << "Hi, " << name << ", " << ptr << endl;          // 輸出
13
14     system("pause");
15     return 0;
16  }
```

想想看，如果在程式碼的第 13 行分別加上

```
cout << (++name) << endl;
cout << (++ptr) << endl;
```

哪一個可以正確的編譯？

```
/* prog9_20 OUTPUT-----------
What's your name? Tippi Hong
Hi, Tippi Hong, How are you?
-----------------------------*/
```

指標陣列 (1/2)

- 一維指標陣列的宣告格式：

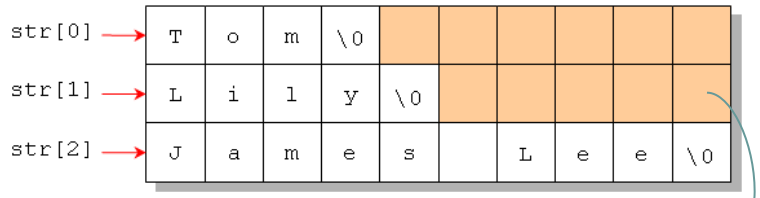
宣告指標陣列

資料型態* 陣列名稱[元素個數];

資料型態 *陣列名稱[元素個數];

- 以二維的字元陣列來儲存字串陣列：

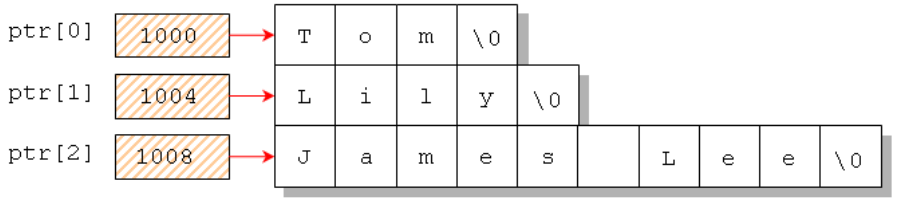
```
char str[3][10]={"Tom", "Lily", "James Lee"};
```



浪費掉的空間

- 以指標陣列的方式來撰寫：

```
char *ptr[3]={"Tom", "Lily", "James Lee"};
```



空間不浪費

二維字元陣列的儲存方式 (3/3)

- 下面的範例是使用指標陣列的練習

```
01  // prog9_21, 指標陣列
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07     char *ptr[3]={"Tom", "Lily", "James Lee"};
08     for(int i=0;i<3;i++)
09        cout << ptr[i] << endl;
10
11     system("pause");
12     return 0;
13  }
```

```
/* prog9_21 OUTPUT---
Tom
Lily
James Lee
----------------------*/
```

整理：字串儲存方式

http://www.cplusplus.com/reference/cstring/

	字元陣列	字元指標	字串型態
宣告方式	char s1[] = "text1";	char* s2 = "text2";	string s3 = "text3“
#include	<string.h>	<string.h>	<string>

字串設定	s1 = s1; // error
	s1 = s2; // error
	s1 = s3; // error
		s2 = s1;
		s2 = s2;
		s2 = s3; // error
			s3 = s1;
			s3 = s2;
			s3 = s3;

字串串接	s1 + s1; // error
	s1 + s2; // error
	s1 + s3;
		s2 + s1; // error
		s2 + s2; // error
		s2 + s3;
			s3 + s1;
			s3 + s2;
			s3 + s3;

存取字元	s1[index]	*(s2+index)	s3.at(index); s3[index]
字串長度	strlen(s1)	strlen(s2)	s3.length()
判斷相等	strcmp(s1, s1)	strcmp(s2, s2)	s3 == s3

取子字串	s1+1; //ext1
	s1++; //error
		s2+2; //xt2
		s2+=2; //xt2
			s3+3; //error
			s3.substr(3); //t3

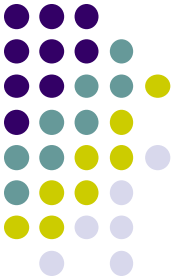
http://www.cplusplus.com/reference/string/string/

雙重指標 (1/2)

- 指向指標的指標（pointer to pointer），稱為雙重指標
- 雙重指標內所存放的是某個指標變數的位址

- 雙重指標變數的宣告格式如下所示：

```
資料型態 **雙重指標;
```



雙重指標 (2/2)

- 下面的敘述為宣告雙重指標的範例

```
int **ptri;          // 宣告一個指向整數的雙重指標 ptri
double **ptrf;       // 宣告一個指向倍精度浮點數的雙重指標 ptrf
```

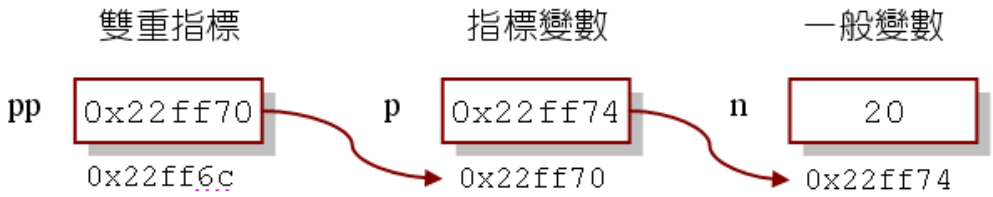
- 也可以在兩個指標符號之間加上括號

```
int *(*ptri);        // 宣告一個指向整數的雙重指標 ptri
double *(*ptrf);     // 宣告一個指向倍精度浮點數的雙重指標 ptrf
```

雙重指標的使用 (1/2)

- 下面是雙重指標的使用範例

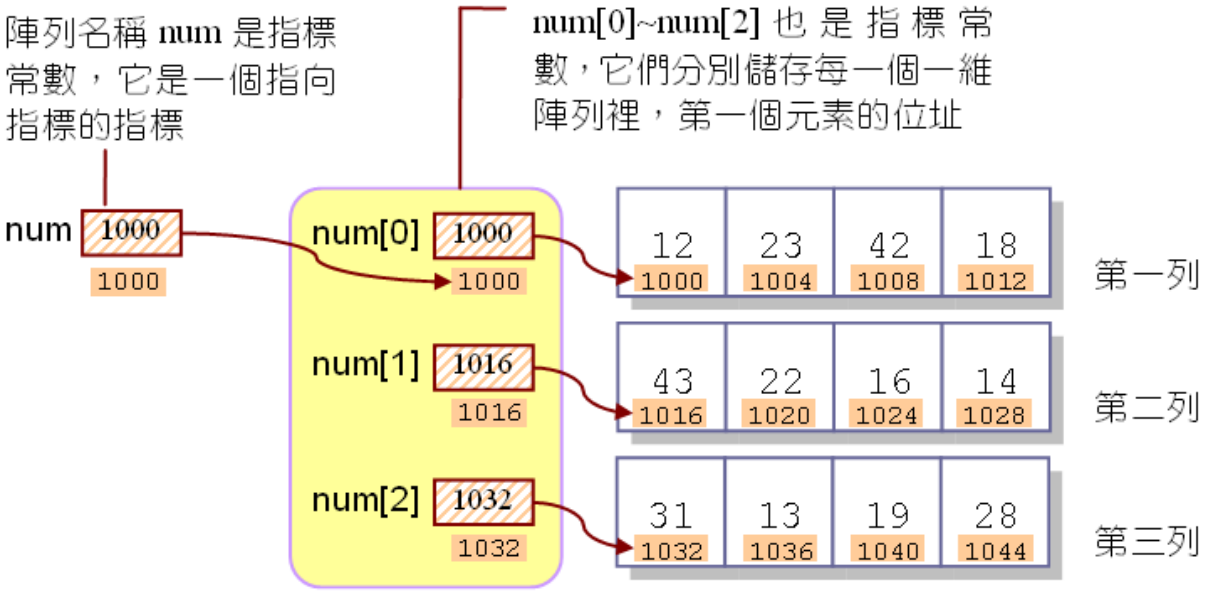
```
01  // prog10_1, 雙重指標的範例
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07     int n=20,*p,**pp;
08     p=&n;
09     pp=&p;
10     cout << "n=" << n << ", &n=" << &n << ", *p=";
11     cout << *p << ", p=" << p << ", &p=" << &p << endl;
12     cout << "**pp=" << **pp << ", *pp=" << *pp;
13     cout << ", pp=" << pp << ", &pp=" << &pp << endl;
14
15     system("pause");
16     return 0;
17  }
```




```
/* prog10_1 OUTPUT------------------------------
n=20, &n=0x22ff74, *p=20, p=0x22ff74, &p=0x22ff70
**pp=20, *pp=0x22ff74, pp=0x22ff70, &pp=0x22ff6c
---------------------------------------------*/
```

雙重指標的使用 (2/2)

- 二維陣列與雙重指標之間的關係
  - 畫繪製出二維陣列num的示意圖



印出陣列的位址 (1/2)

- 下面的程式用來印出二維陣列的位址

```
01  // prog10_2, 印出陣列的位址
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07      int num[3][4];                              // 宣告 3×4 的二維陣列 num
08
09      cout << "num=" << num << endl;              // 印出雙重指標 num 的值
10      cout << "&num=" << &num << endl;            // 印出雙重指標 num 的位址
11      cout << "*num=" << *num << endl;  //印出雙重指標 num 所指向之指標的值
12
13      cout << "num[0]=" << num[0] << endl; //印出指標常數 num[0]的值
14      cout << "num[1]=" << num[1] << endl; //印出指標常數 num[1]的值
15      cout << "num[2]=" << num[2] << endl; //印出指標常數 num[2]的值
16
17      cout << "&num[0]=" << &num[0] << endl; //印出指標常數 num[0]的位址
18      cout << "&num[1]=" << &num[1] << endl; //印出指標常數 num[1]的位址
19      cout << "&num[2]=" << &num[2] << endl; //印出指標常數 num[2]的位址
20
21      system("pause");
22      return 0;
23  }
```

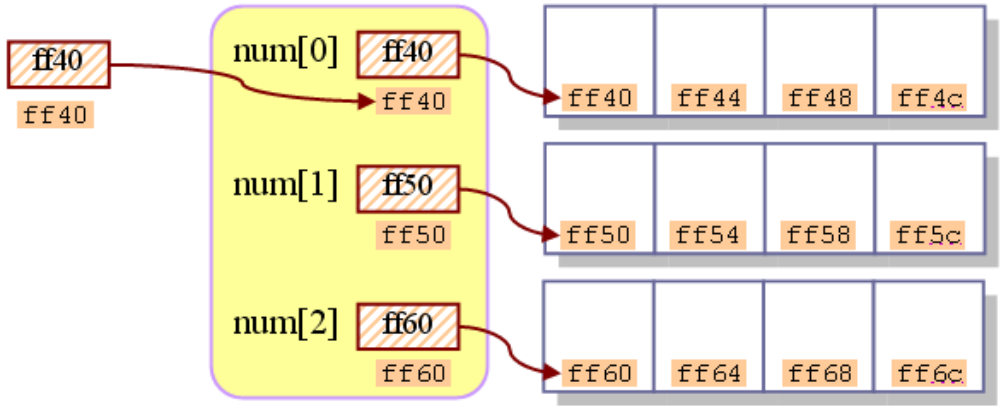
印出陣列的位址 (2/2)

- prog10_2執行結果的說明

```
/* prog10_2 OUTPUT-----------------
num=0x22ff40
&num=0x22ff40
*num=0x22ff40
num[0]=0x22ff40
num[1]=0x22ff50
num[2]=0x22ff60
&num[0]=0x22ff40
&num[1]=0x22ff50
&num[2]=0x22ff60
--------------------------------*/
```

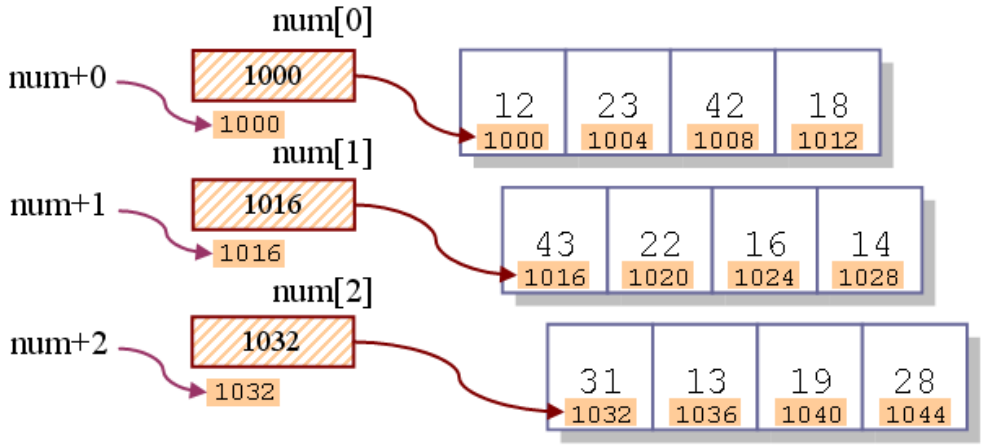
指標常數的值

指標常數的位址

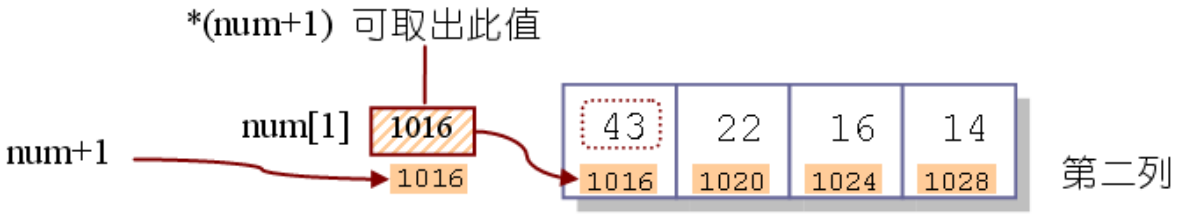


二維陣列的指標表示方式 (1/2)

- num+m的值代表第m+1列的位址，可以從下圖驗證

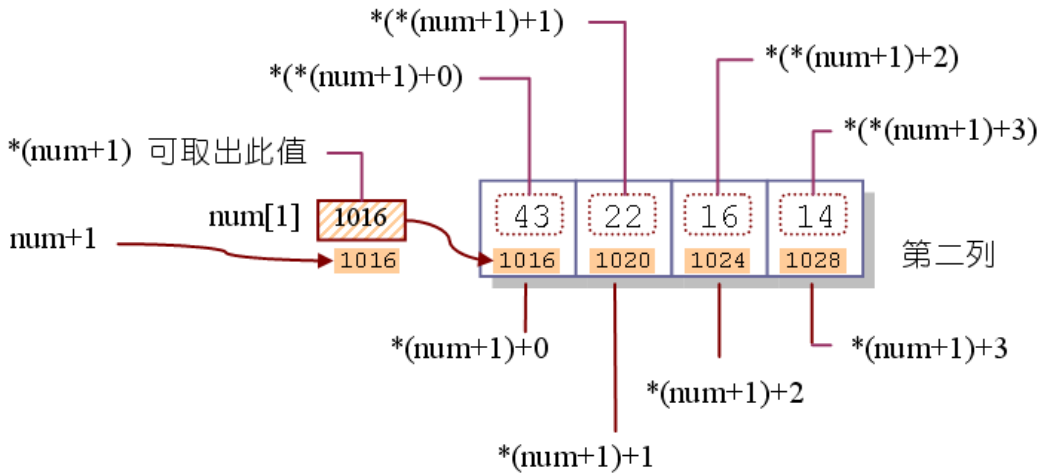


- *(num+1) 可取得num[1]的內容



二維陣列的指標表示方式 (2/2)

- *(num+m)+n代表第m+1列，第n+1行的位址



- 要取出第m+1列，第n+1行的各零時，可用下列的語法

```
*(*(num+m)+n);    // 用指標表示陣列元素 num[m][n]
```

用指標印出陣列的位址 (1/2)

```
01  // prog10_3, 印出陣列的位址
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07     int num[3][4]={{12,23,42,18},
08                    {43,22,16,14},
09                    {31,13,19,28}};
10     int m,n;
11     for(m=0;m<3;m++)
12        for(n=0;n<4;n++)
13        {
14           cout << "num[" << m << "][" << n << "]=" << *(*(num+m)+n);
15           cout << ", 位址=" << *(num+m)+n << endl;
16        }
17
18     cout << "**num=" << **num << endl;
19
20     system("pause");
21     return 0;
22  }
```

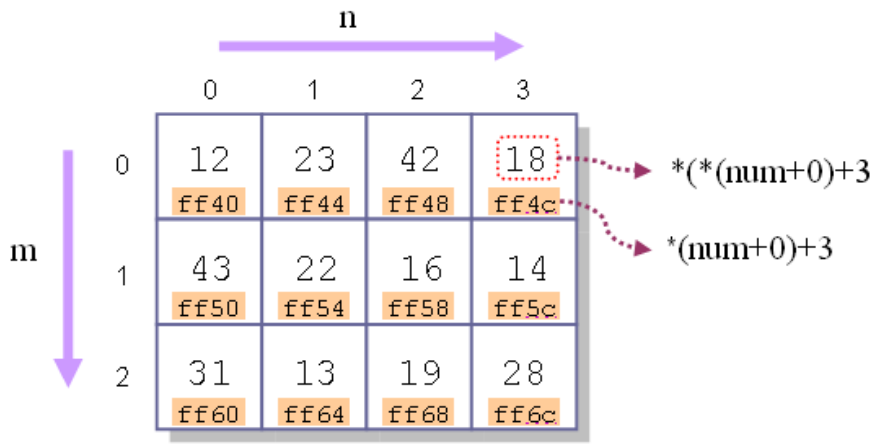
((num+m)+n) : num[m][n] 的值

*(num+m)+n : num[m][n] 的位址

- 下面的程式是利用指標印出陣列的位址及指向的變數值

印指標和陣列的位址 (2/2)

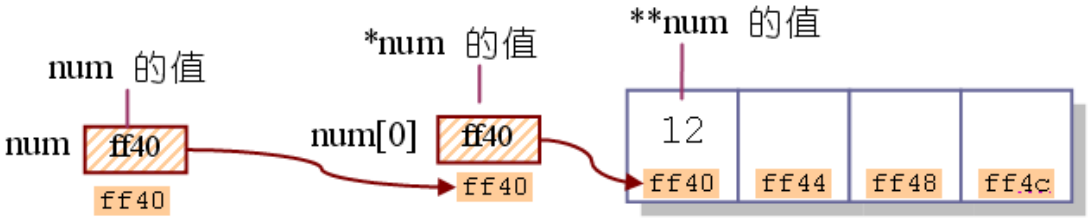
```
/* prog10_3 OUTPUT------------
num[0][0]=12, 位址=0x22ff40
num[0][1]=23, 位址=0x22ff44
num[0][2]=42, 位址=0x22ff48
num[0][3]=18, 位址=0x22ff4c
num[1][0]=43, 位址=0x22ff50
num[1][1]=22, 位址=0x22ff54
num[1][2]=16, 位址=0x22ff58
num[1][3]=14, 位址=0x22ff5c
num[2][0]=31, 位址=0x22ff60
num[2][1]=13, 位址=0x22ff64
num[2][2]=19, 位址=0x22ff68
num[2][3]=28, 位址=0x22ff6c
**num=12
-----------------------------*/
```



*(num+m)+n 代表陣列元素 num[m][n] 的位址

((num+m)+n) 代表陣列元素 num[m][n] 的值

- num是雙重指標，所以**num的值即為num[0][0]的值



雙重指標的範例

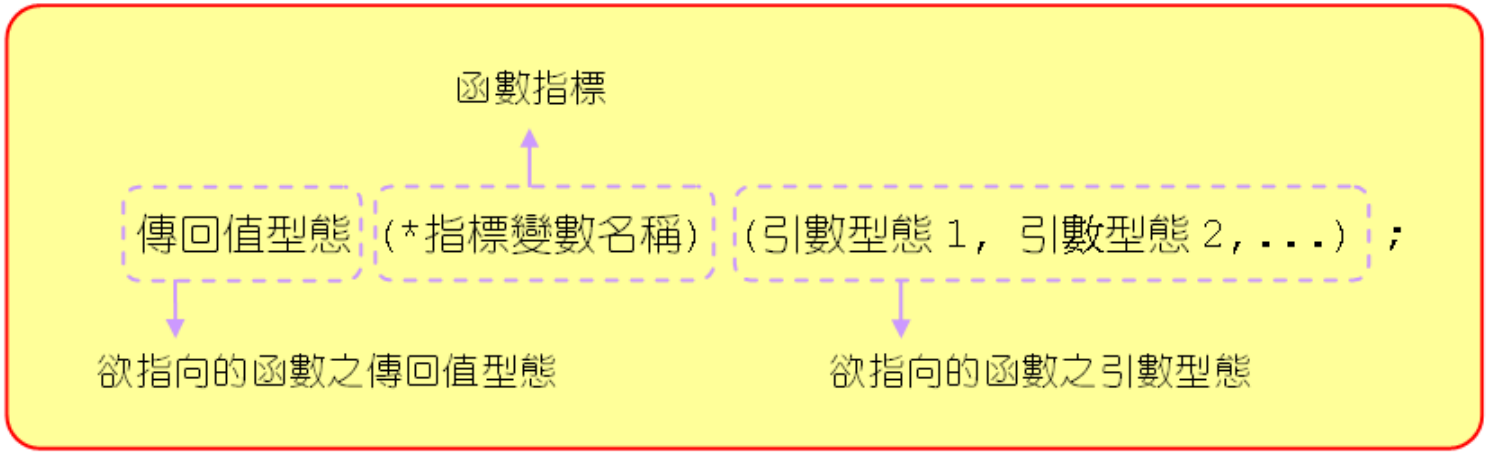
```
01  // prog10_4, 利用指標將大於 40 的陣列元素設值為 40
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07     int num[3][4]={{12,23,42,18},
08                    {43,22,16,14},
09                    {31,13,19,28}};
10     int m,n;
11     for(m=0;m<3;m++)
12     {
13        for(n=0;n<4;n++)
14        {
15           if(*(*(num+m)+n)>40)          // 判別 num[m][n]的值是否大於 40
16              *(*(num+m)+n)=40;          // 如果是，則將元素值設為 40
17           cout << *(*(num+m)+n) << " ";   // 印出元素 num[m][n]的值
18        }
19        cout << endl;
20     }
21
22     system("pause");
23     return 0;
24  }
```

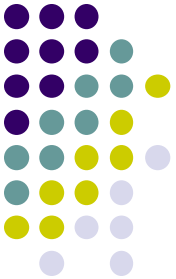
- 下面的程式可將陣列中，大於40的元素值均以40來取代

```
/* prog10_4 OUTPUT---
 12 23 40 18
 40 22 16 14
 31 13 19 28
--------------------*/
```

函數指標

- 函數名稱主要記錄著函數的起始位址
- 函數指標（function pointer）的定義格式如下





函數指標的用法 (1/2)

- 設有一個計算平方值的函數square()，先對它做宣告

```
int square(int);      // 定義 square()函數的原型
```

- 想把函數指標pf指向 square()函數，可這麼做宣告的敘述

```
int (*pf)(int);       // 定義函數指標 pf
```

- 把函數指標pf指向 函數square()

```
pf=square;            // 使函數指標 pf 指向 square()
```

函數指標的用法 (2/2)

- 下列是函數指標的使用範例

```
01  // prog9_12, 函數指標的使用
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int square(int);                          // 定義 square()函數的原型
06  int main(void)
07  {
08     int (*pf)(int);                        // 定義函數指標 pf
09     pf=square;                             // 使函數指標 pf 指向 square()
10     cout << "square(5)=" << (*pf)(5) << endl; // 印出 square(5)的值
11     system("pause");
12     return 0;
13  }
14
15  int square(int a)                         // 自訂函數 square(), 計算平方值
16  {
17     return (a*a);
18  }
```

```
/* prog9_12 OUTPUT---
square(5)=25
----------------------*/
```

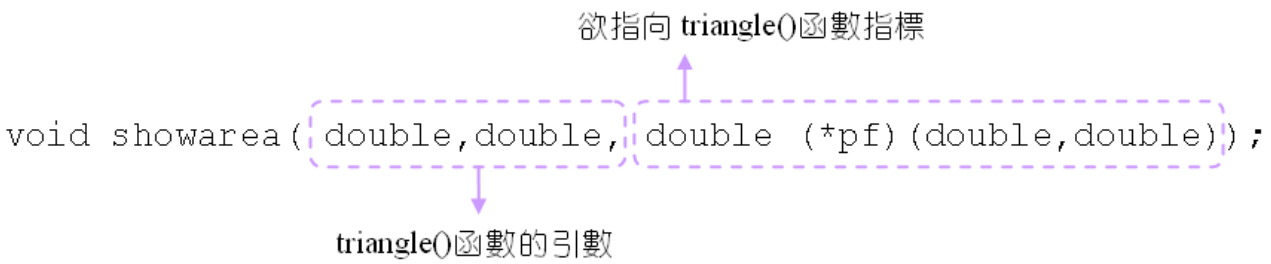
傳遞函數到其中函數 (1/3)

- 以下的步驟說明如何傳遞函數到其中函數

設triangle() 函數可傳回三角形的面積，其原型如下

```
double triangle(double,double);
```

完成上述設定後，可傳入 triangle() 函數，並傳回 triangle() 的傳回值，以showarea() 可定義如下



showarea() 函數的定義如下

```
void showarea(double x,double y,double (*pf)(double,double))
{
   cout << (*pf)(x,y) << endl;          // 呼叫函數指標所指向的函數
}
```

傳遞函數到其它函數中 (2/3)

- 下面的程式示範如何傳遞函數到其它的函數中

```
01  // prog9_13, 傳遞函數到其它函數中
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  double triangle(double,double),rectangle(double,double);
06  void showarea(double,double,double (*pf)(double,double));
07  int main(void)
08  {
09     cout << "triangle(6,3.2)=";
10     showarea(6,3.2,triangle);        // 呼叫 triangle(),並印出其值
11     cout << "rectangle(4,6.1)=";
12     showarea(4,6.1,rectangle);       // 呼叫 rectangle(),並印出其值
13
14     system("pause");
15     return 0;
16  }
```

```
/* prog9_13 OUTPUT-----
triangle(6,3.2)=9.6
rectangle(4,6.1)=24.4
----------------------*/
```

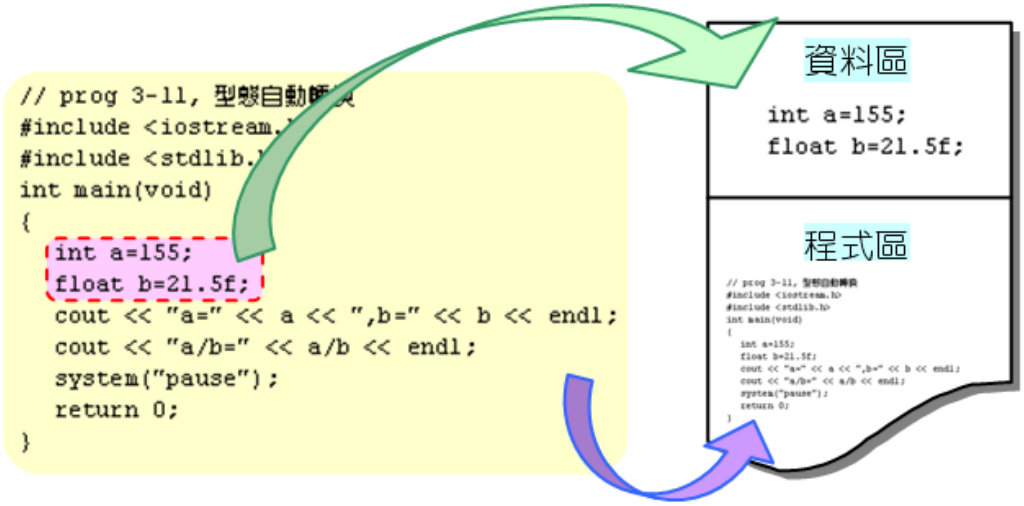
傳遞函數到其中函數 (3/3)

```
17
18  double triangle(double base,double height)    // 計算三角形面積
19  {
20     return (base*height/2);
21  }
22
23  double rectangle(double height,double width)   // 計算長方形面積
24  {
25     return (height*width);
26  }
27
28  void showarea(double x,double y,double (*pf)(double,double))
29  {
30     cout << (*pf)(x,y) << endl;
31     return;
32  }
```

```
/* prog9_13 OUTPUT-----
triangle(6,3.2)=9.6
rectangle(4,6.1)=24.4
----------------------*/
```

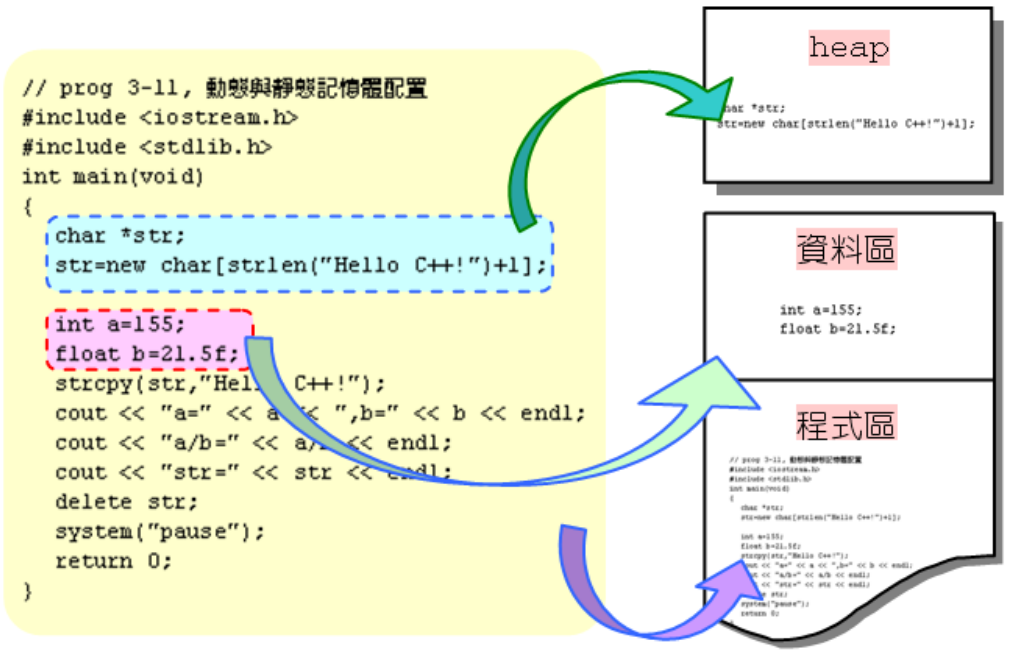
靜態記憶體配置

- 在編譯時期即配置完成記憶體空間，待程式執行時使用，稱為靜態記憶體配置（static memory allocation）
- 下圖為靜態記憶體配置示意圖



動態記憶體配置

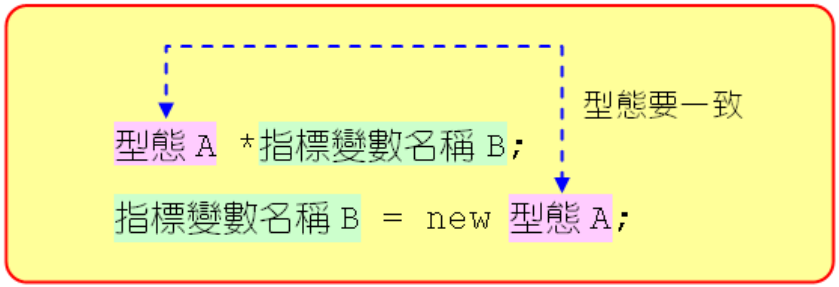
- 在執行時期才配置所需要的記憶體，稱為動態記憶體配置（dynamic memory allocation）
- 在動態記憶體配置時，作業系統會從未被使用的空間中，找尋一塊適合的記憶體區塊供該程式使用



執行時才會從heap中劃分一個區塊給動態記憶體配置的資料使用

使用對主資料型態做動態配置

- 使用 new運算子 來做對主資料型態的動態記憶體配置



- 動態記憶體配置的範例

```
int *ptr;
ptr=new int;
```

- 利用 delete運算子 來釋放所佔用的空間

```
delete 指標變數名稱;
```

- delete運算子的使用範例

```
delete ptr;
delete ptr1,ptr2,ptr3;
```

僅會釋放第一個指標所指向的動態記憶體配置區域

動態記憶體配置的範例

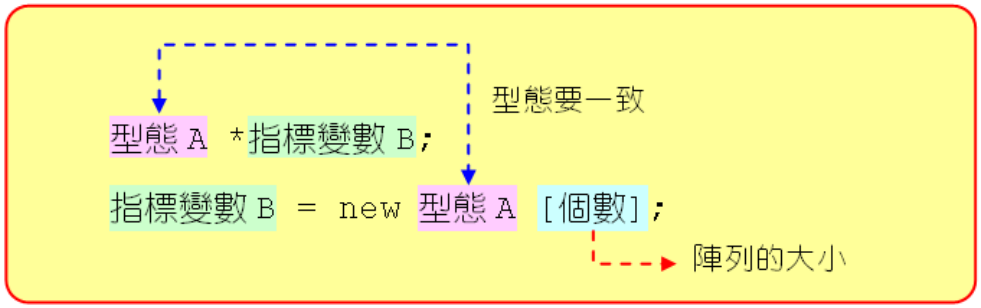
- 下面的程式是動態記憶體配置的範例

```
01  // prog10_5, 基本資料型態之動態記憶體配置
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07      int *a;        // 宣告 int 型態的指標變數 a
08      a=new int;     // 配置 int 型態的動態記憶體, 並將起始位址給指標 a 存放
09      *a=5;          // 將指標 a 所指向的位址之內容設值為 5
10      cout << "*a=" << *a << endl;          // 印出 a 所指向位址的內容
11      cout << *a << "*" << *a << "=" << (*a)*(*a) << endl;
12      delete a;      // 釋放指標 a 所指向的動態記憶體配置區域
13      cout << "*a=" << *a << endl;          // 印出 a 所指向位址的內容
14      a=NULL;                               // 將 a 指向 NULL
15
16      system("pause");
17      return 0;
18  }
```

```
/* prog10_5 OUTPUT---
*a=5
5*5=25
*a=0
--------------------*/
```

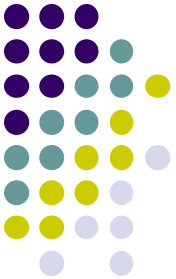
使用陣列做動態配置 (1/4)

- 使用 new運算子 做陣列的動態記憶體配置



- 陣列的動態記憶體配置 範例

```
int *ptr;          // 宣告 int 型態的指標 ptr
ptr=new int[5];    // 於執行時配置一個 int 型態的陣列記憶體區塊，
                   其大小為 5，並使指標 ptr 指向它
```



使用陣列做動態配置 (2/4)

- 釋放陣列的動態記憶體配置要用，可以和 delete

```
delete[] 指標變數名稱;
```

- delete運算子的使用範例

```
delete[] ptr;      // 釋放指標 ptr 所指向的記憶體配置區塊
ptr=NULL;          // 使指標 ptr 不指向任何地方
```

使用陣列做動態配置 (3/4)

- 下面的程式使用動態配置的方式，建立一個整數陣列

```
01  // prog10_6, 整數陣列之動態記憶體配置
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07      int i,*a;
08      a=new int[5];          // 為陣列配置動態記憶體, 並使指標 a 指向該記憶體
09      for(i=0;i<5;i++)       // 為陣列元素設值
10          a[i]=i*2;
11      for(i=0;i<5;i++)       // 印出陣列的內容
12          cout << "a[" << i << "]=" << a[i] << "\t";
13      cout << endl;
14      delete[] a;            // 釋放陣列的動態記憶體配置區域
15      a=NULL;                // 使指標 a 不指向任何地方
16
17      system("pause");
18      return 0;
19  }
```

```
/* prog10_6 OUTPUT------------------
a[0]=0  a[1]=2  a[2]=4  a[3]=6  a[4]=8
-----------------------------------*/
```

使用陣列做動態配置 (4/4)

- 下面的範例是使用動態記憶體配置的方式建立字串

```
01  // prog10_7, 動態記憶體配置
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  char *setString(char *);
06  int main(void)
07  {
08     char *str;
09     str=setString("Hello C++!");   // 將欲做動態配置的字串傳入函數
10     cout << str << endl;           // 印出字串內容
11     delete[] str;                  // 釋放字串的動態記憶體配置區域
12
13     system("pause");
14     return 0;
15  }
16
17  char *setString(char *text)
18  {
19     char *ptr;
20     ptr=new char[strlen(text)+1];  // 動態配置後，將位址指定給 ptr 存放
21     strcpy(ptr,text);              // 將 text 的內容複製到 ptr
22     return ptr;
23  }
```

```
/* prog10_7 OUTPUT---
Hello C++!
----------------------*/
```

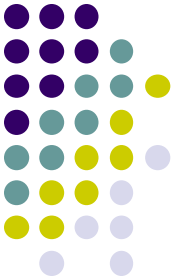
指標與參照 (1/3)

- 指標是利用「位址運算子 &」以及「位址取值運算子 *」取得指向變數的位址及其內容

```
int i=30,*ptr;   // 宣告整數變數 i、整數指標變數 ptr
ptr=&i;          // 使得指標 ptr 指向變數 i
*ptr=*ptr+5;     // 將 ptr 指向變數值加 5
```

- 參照是利用「參照運算子 &」取得欲參考變數的位址，並代替該變數

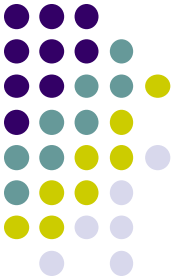
```
int i=15;        // 宣告整數變數 i
int &ref=i       // 宣告 ref 為 i 的參照
```



指標與參照 (2/3)

- 若是函數傳回值為參照，則該函數就可以作為設定敘述的左邊，如下列的敘述：

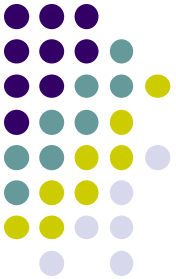
```
int *func1(int *);    // 宣告 func1()函數原型，傳回值為指標
int &func2(int &);    // 宣告 func2()函數原型，傳回值為參照
...
func1(a)=100;         // 錯誤的設定敘述，func1()不能被指定其值
func2(a)=100;         // 合法的設定敘述，可將 func2()傳回的參照設值
```



指標與參照 (3/3)

- 以一個簡單的範例，複習指標與參照的使用方式

```
01  // prog10_8, 指標與參照
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07      int a=10,&ref=a;                    // 宣告變數 a 及其參照 ref
08      int b=15,*ptr;                      // 宣告變數 b 及指標 ptr
09      ptr=&b;                             // 將 ptr 指向 b
10      cout << a << "+" << b << "=";       // 印出 a+b 的結果
11      cout << ref+*ptr << endl;           // 利用指標與參照完成
12
13      system("pause");                    /* prog10_8 OUTPUT---
14      return 0;                           10+15=25
15  }                                       --------------------*/
```



引數的傳遞方式

- 引數在函數之間傳遞的方式分為三種，分別是
  - 傳值（pass by value）
  - 傳址（pass by address）
  - 傳參照（pass by reference）

引數的傳遞方式比較

- 將這幾種傳遞引數的方式整理做整理

函數原型	函數呼叫方式	傳遞方式說明	參考本書範例
int func(int);	func(num);	傳值	prog6_3
int func(int *);	func(ptr1);	傳址(使用指標)	prog9_7
int func(int *);	func(ptr2);	傳址(使用指標)	prog9_18
int func(char *);	func(ptr3);	傳址(使用指標)	習題 9_16
int func(int *);	func(array);	傳址(使用陣列名稱)	prog9_19
int func(int []);	func(array);	傳址(使用陣列名稱)	prog8_7
int func(int []);	func(ptr2);	傳址(使用指標)	prog9_10
int func(int *);	func(&num);	傳址	prog9_8
int func(int &);	func(num);	傳參照	prog7_4
int func(int(*p)(int));	func(func2);	傳址(使用函數指標)	prog9_12


-The End-