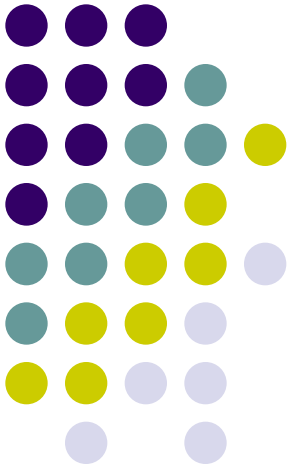
第九章
指　標

認識指標的概念及指標變數的使用

學習指標運算子的運算方式

熟悉指標的運算

利用函數傳遞指標與陣列

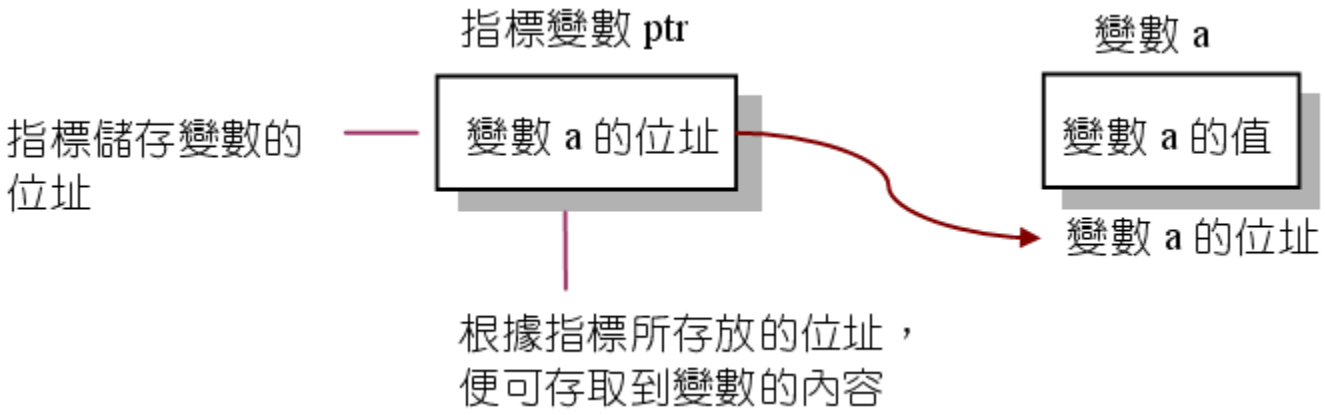


什麼是指標 (1/2)

- 指標（pointer）是用來存放變數在記憶體中的位址
- 如果指標ptr存放變數a的位址，則

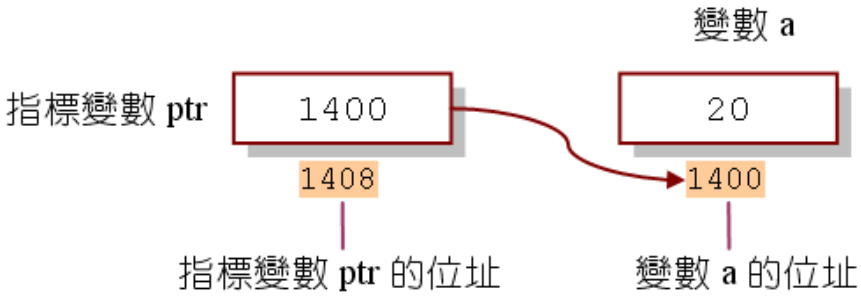
" 指標 ptr 指向變數 a "

- 下面是指標ptr指向變數a的示意圖

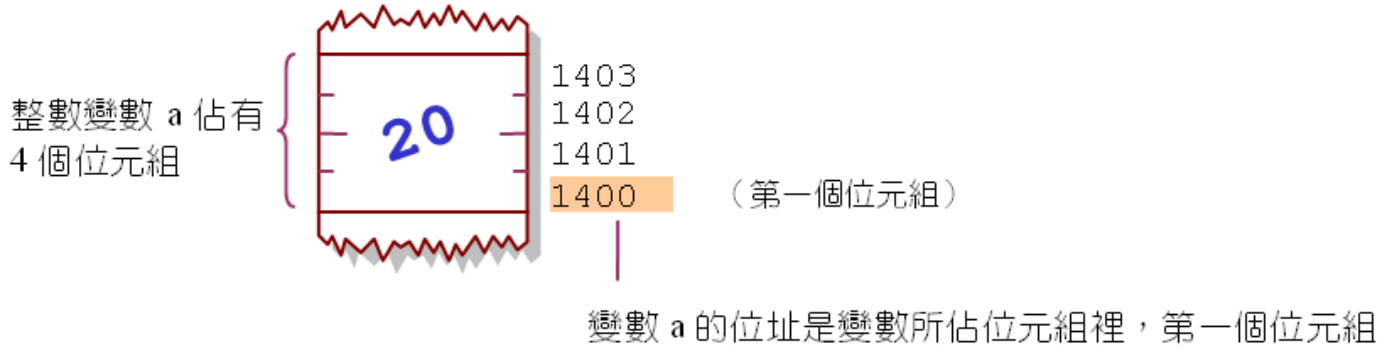


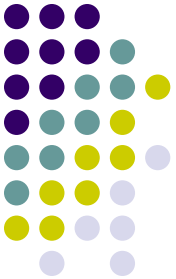
什麼是指標 (2/2)

- 變數a與指標變數ptr的配置可由下圖來表示



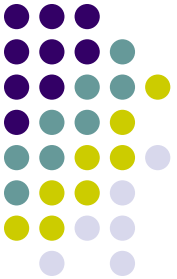
- 通常編譯器是採「位元組定址法」決定變數的位址：





為什麼要用指標？

- 更有效率
- 較複雜的資料結構，需要指標才能將資料鏈結在一起
- 許多函數必須利用指標來傳達記憶體的訊息



記憶體的位址 (1/2)

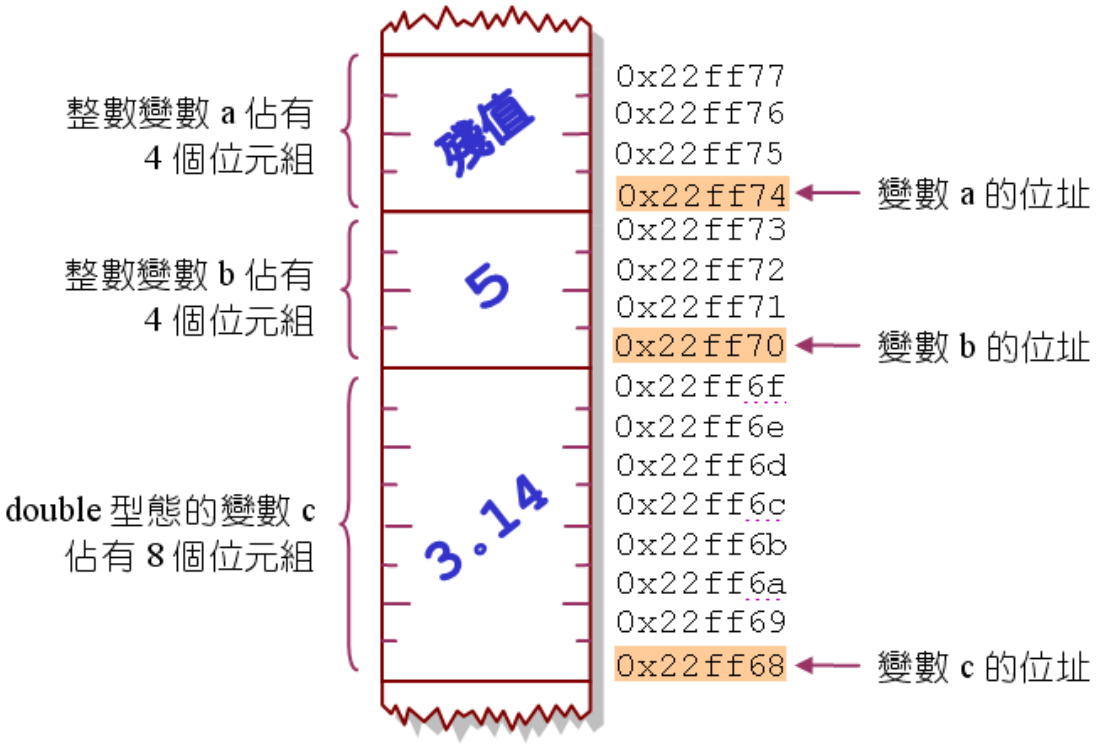
- 下面的程式印出變數的值、記憶體的大小，與變數的位址

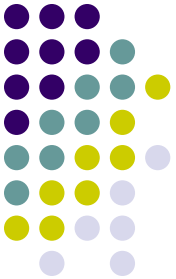
```
01  // prog9_1, 印出變數於記憶體內的位址
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07     int a,b=5;                  // 宣告變數 a 與 b，但變數 a 沒有設定初值
08     double c=3.14;
09
10     cout << "a=" << a << ", sizeof(a)=" << sizeof(a);
11     cout << ", 位址為" << &a << endl;
12     cout << "b=" << b << ", sizeof(b)=" << sizeof(b);
13     cout << ", 位址為" << &b << endl;
14     cout << "c=" << c << ", sizeof(c)=" << sizeof(c);
15     cout << ", 位址為" << &c << endl;
16
17     system("pause");
18     return 0;
19  }
```

```
/* prog9_1 OUTPUT------------------
a=2, sizeof(a)=4, 位址為 0x22ff74
b=5, sizeof(b)=4, 位址為 0x22ff70
c=3.14, sizeof(c)=8, 位址為 0x22ff68
------------------------------------*/
```

記憶體的位址 (2/2)

- 下圖是prog9_1中，變數於記憶體內配置的情形：





指標變數的宣告 (1/2)

- 指標變數的宣告格式如下所示

```
資料型態 *指標變數;      // 宣告指標變數
```

- 下面的敘述為指標變數宣告的範例

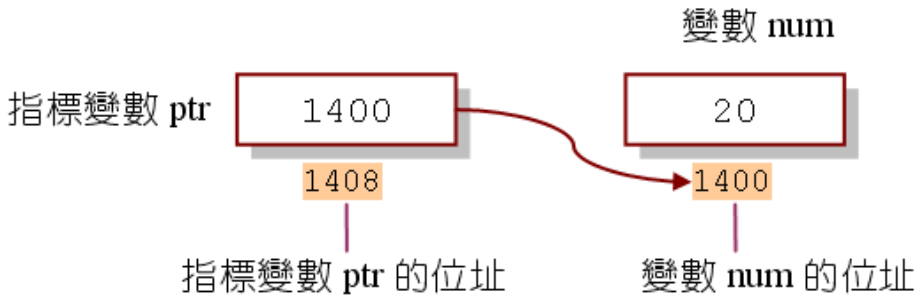
```
int *ptr;          // 宣告指向整數的指標變數 ptr
```

指標變數的宣告 (2/2)

- 把指標ptr指向整數變數num

```
int num=20;     // 宣告整數變數 num，並設值為 20
ptr=&num;       // 把 ptr 設為變數 num 的位址
```

- 下面是設定ptr=&num的示意圖



- 在宣告指標變數時，也可以立即將它指向某個整數

```
int num=20;         // 宣告整數變數 num，並設值為 20
int *ptr=&num;      // 宣告指標變數 ptr，並將它指向變數 num
```

位址運算子

- 位址運算子「&」可用來取得變數的位址

  舉例來說，&num即代表取出num在記憶體中的位址

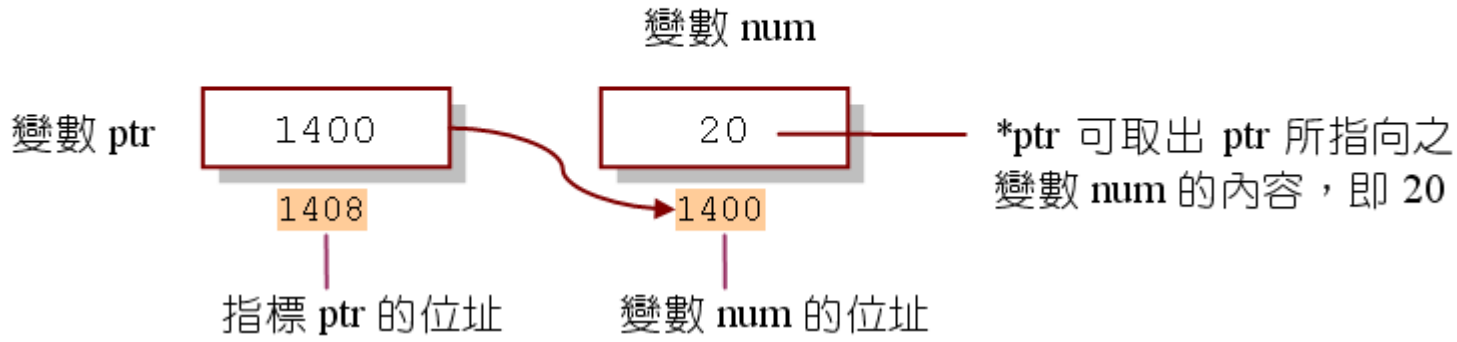
```
int num=20;
```

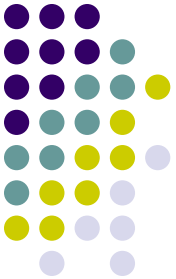
依址取值運算子 (1/3)

- 依址取值運算子「*」可取得指標所指向變數的內容

  舉例來說，*ptr可取得num的值：

```
int num=20;
int *ptr=&num;
```





依址取值運算子 (2/3)

- 下面的範例印出變數的位址與變數值

```
01  // prog9_2, 指標變數的宣告
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07     int *ptr,num=20;                  // 宣告變數 num 與指標變數 ptr
08
09     ptr=&num;                         // 將 num 的位址設給指標 ptr 存放
10     cout << "num=" << num << ", &num=" << &num << endl;
11     cout << "*ptr=" << *ptr << ", ptr=" << ptr;
12     cout << ", &ptr=" << &ptr << endl;
13
14     system("pause");
15     return 0;
16  }
```

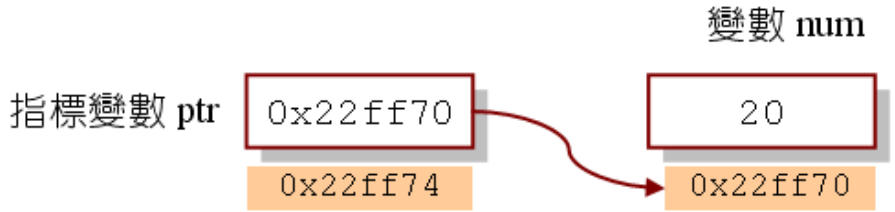
```
/* prog9_2 OUTPUT------------------

num=20, &num=0x22ff70
*ptr=20, ptr=0x22ff70, &ptr=0x22ff74
-----------------------------------*/
```

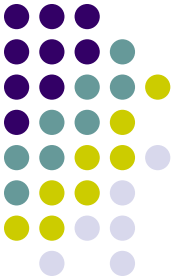
依址取值運算子 (3/3)

- 執行完第7行後，記憶體的配置

```
07      int *ptr,num=20;          // 宣告變數 num 與指標變數 ptr
```

- 執行完第9行後，記憶體的配置





指標變數的使用 (1/2)

- 指標可以重新指向另一個相同型態的變數

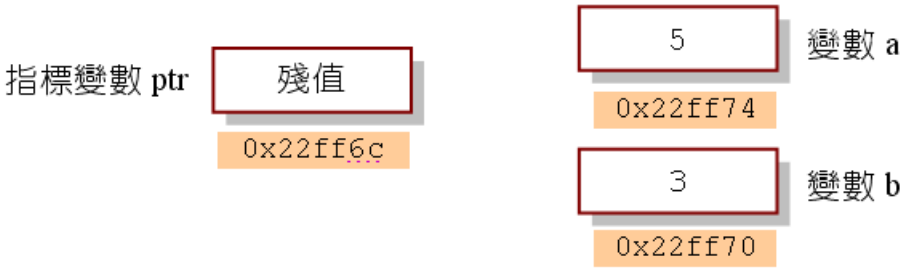
```
01  // prog9_3, 指標變數的使用
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07     int a=5,b=3;
08     int *ptr;                          // 宣告指標變數 ptr
09
10     ptr=&a;                            // 將 a 的位址設給指標 ptr 存放
11     cout << "&a=" << &a << ", &ptr=" << &ptr;
12     cout << ", ptr=" << ptr << ", *ptr=" << *ptr << endl;
13     ptr=&b;                            // 將 b 的位址設給指標 ptr 存放
14     cout << "&b=" << &b << ", &ptr=" << &ptr;
15     cout << ", ptr=" << ptr << ", *ptr=" << *ptr << endl;
16
17     system("pause");
18     return 0;
19  }
```

```
/* prog9_3 OUTPUT------------------------------
&a=0x22ff74, &ptr=0x22ff6c, ptr=0x22ff74, *ptr=5
&b=0x22ff70, &ptr=0x22ff6c, ptr=0x22ff70, *ptr=3
--------------------------------------------*/
```

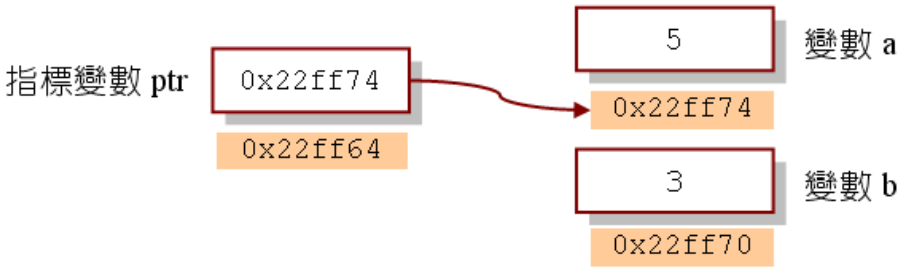
指標變數的使用 (2/2)

- 記憶體的配置情形

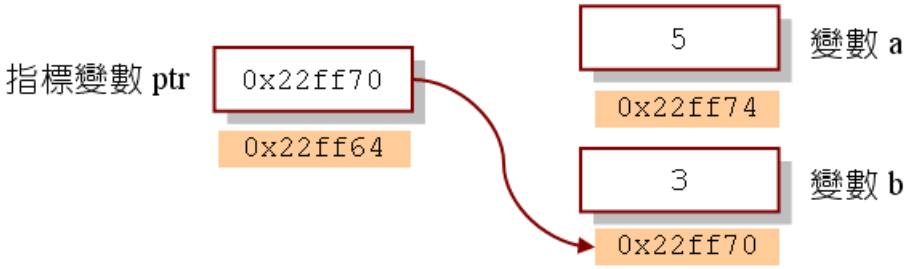
執行完第8行 **`int *ptr;`**

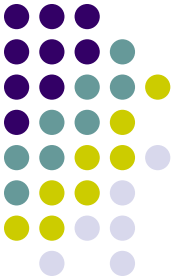


執行完第10行 **`ptr=&a;`**



執行完第13行 **`ptr=&b;`**



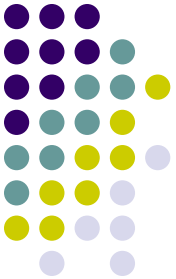


指標變數的大小

- 下面的程式是利用sizeof() 求出指標變數所佔的位元組

```
01  // prog9_4, 指標變數的大小
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07     int *ptri;                  // 宣告指向整數的指標 ptri
08     char *ptrc;                 // 宣告指向字元的指標 ptrc
09
10     cout << "sizeof(ptri)=" << sizeof(ptri) << endl;
11     cout << "sizeof(ptrc)=" << sizeof(ptrc) << endl;
12     cout << "sizeof(*ptri)=" << sizeof(*ptri) << endl;
13     cout << "sizeof(*ptrc)=" << sizeof(*ptrc) << endl;
14
15     system("pause");
16     return 0;
17  }
```

```
/* prog9_4 OUTPUT----
sizeof(ptri)=4   } 指標變數皆佔有 4 個
sizeof(ptrc)=4   } 位元組
sizeof(*ptri)=4
sizeof(*ptrc)=1
--------------------*/
```

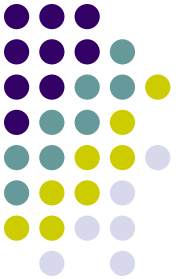


指標的操作練習 (1/3)

- 下面是一個簡單的範例，藉以熟悉指標的操作

```
01  // prog9_5, 指標的操作練習
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07     int a=5,b=10;
08     int *ptr1,*ptr2;
09     ptr1=&a;                  // 將 ptr1 設為 a 的位址
10     ptr2=&b;                  // 將 ptr2 設為 b 的位址
11     *ptr1=7;                  // 將 ptr1 指向的內容設為 7
12     *ptr2=32;                 // 將 ptr2 指向的內容設為 32
13     a=17;                     // 設定 a 為 17
14     ptr1=ptr2;                // 設定 ptr1=ptr2
15     *ptr1=9;                  // 將 ptr1 指向的內容設為 9
16     ptr1=&a;                  // 將 ptr1 設為 a 的位址
17     a=64;                     // 設定 a 為 64
18     *ptr2=*ptr1+5;            // 將 ptr2 指向的內容設為*ptr1+5
19     ptr2=&a;                  // 將 ptr2 設為 a 的位址
```

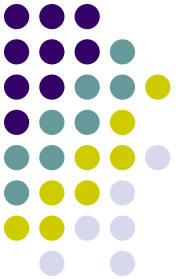
```
/* prog9_5 OUTPUT--------------
a=64, b=69, *ptr1=64, *ptr2=64
ptr1=0x22ff74, ptr2=0x22ff74
------------------------------*/
```



指標的操作練習 (2/3)

```
20
21      cout << "a=" << a << ", b=" << b;
22      cout << ", *ptr1=" << *ptr1 << ", *ptr2=" << *ptr2 << endl;
23      cout << "ptr1=" << ptr1 << ", ptr2=" << ptr2 << endl;
24
25      system("pause");
26      return 0;
27  }
```

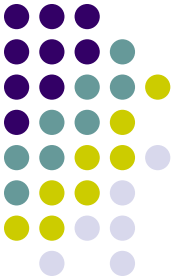
```
/* prog9_5 OUTPUT--------------
a=64, b=69, *ptr1=64, *ptr2=64
ptr1=0x22ff74, ptr2=0x22ff74
--------------------------------*/
```



指標的操作練習 (3/3)

- 下表是變數a、b與指標ptr1、ptr2之值的變化情形

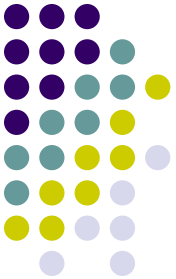
行號	程式碼	a	b	ptr1	*ptr1	ptr2	*ptr2
07	int a=5,b=10;	5	10				
08	int *ptr1,*ptr2;	5	10	殘值	殘值	殘值	殘值
09	ptr1=&a;	5	10	ff74	5	殘值	殘值
10	ptr2=&b;	5	10	ff74	5	ff70	10
11	*ptr1=7;	7	10	ff74	7	ff70	10
12	*ptr2=32;	7	32	ff74	7	ff70	32
13	a=17;	17	32	ff74	17	ff70	32
14	ptr1=ptr2;	17	32	ff70	32	ff70	32
15	*ptr1=9;	17	9	ff70	9	ff70	9
16	ptr1=&a;	17	9	ff74	17	ff70	9
17	a=64;	64	9	ff74	64	ff70	9
18	*ptr2=*ptr1+5;	64	69	ff74	64	ff70	69
19	ptr2=&a;	64	69	ff74	64	ff74	64



指標變數指向之型態 (1/2)

- 下面的程式示範錯誤的指標用法

```
01  // prog9_6, 錯誤的指標型態
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07     int a1=100, *ptri;
08     double a2=3.2, *ptrf;
09     ptri=&a2;          // 錯誤，將 int 型態的指標指向 double 型態的變數
10     ptrf=&a1;          // 錯誤，將 double 型態的指標指向 int 型態的變數
11
12     cout << "sizeof(a1)=" << sizeof(a1) << endl;
13     cout << "sizeof(a2)=" << sizeof(a2) << endl;
14     cout << "a1=" << a1 << ", *ptri=" << *ptri << endl;
15     cout << "a2=" << a2 << ", *ptrf=" << *ptrf << endl;
16
17     system("pause");
18     return 0;
19  }
```



指標變數指向之型態 (2/2)

- 編譯器會在編譯時發出下面的錯誤訊息

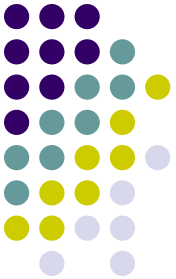
```
cannot convert 'double' to 'int' in assignment
cannot convert 'int' to 'double' in assignment
```

- 第9、10行的程式應修改成

```
09      ptri=&a1;      // 將 int 型態的指標 ptri 指向 int 型態的變數 a1
10      ptrf=&a2;      // 將 double 型態的指標 ptrf 指向 double 型態的變數 a2
```

- prog9_6經過修正、編譯後的執行結果如下

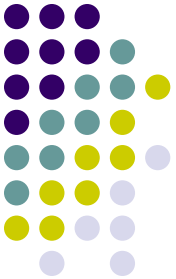
```
/* prog9_6 OUTPUT----
sizeof(a1)=4
sizeof(a2)=8
a1=100, *ptri=100
a2=3.2, *ptrf=3.2
---------------------*/
```



傳遞指標到函數 (1/4)

- 將指標傳入函數裡，可利用如下的語法

```
傳回值型態 函數名稱(資料型態 *指標變數)
{
    // 函數的本體
}
```



傳遞指標到函數 (2/4)

- 設計address()，可接收指向整數的指標，沒有傳回值：

  原型的宣告：

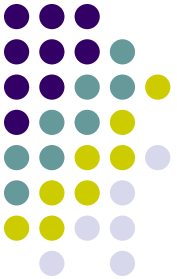
```
void address(int *);          // 宣告函數address()的原型
```

  函數的定義：

```
void address(int *ptr)          // 定義函數 address()
{
   // 函數的內容
}
```

  函數的呼叫：

```
int a=12;
int *ptr=&a;              // 將指標 ptr 指向變數 a
address(&a);              // 傳入 a 的位址
address(ptr);             // 傳入指向整數的指標 ptr
```



傳遞指標到函數 (3/4)

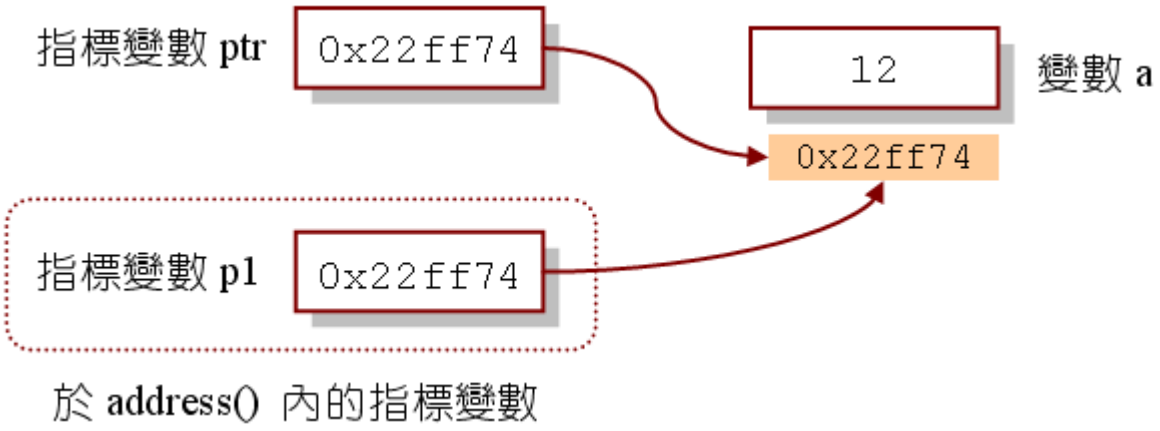
- 下面是函數address() 的完整範例

```
01  // prog9_7, 傳遞指標到函數裡
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void address(int *);              // 宣告 address()函數的原型
06  int main(void)
07  {
08     int a=12;                      // 設定變數 a 的值為 12
09     int *ptr=&a;                   // 將指標 ptr 指向變數 a
10
11     address(&a);                   // 將 a 的位址傳入 address()函數中
12     address(ptr);                  // 將 ptr 傳入 address()函數中
13
14     system("pause");
15     return 0;
16  }
17  void address(int *p1)
18  {
19     cout << "於位址" << p1 << "內，儲存的變數內容為" << *p1 << endl;
20     return;
21  }
```

```
/* prog9_7 OUTPUT----------------
於位址 0x22ff74 內，儲存的變數內容為 12
於位址 0x22ff74 內，儲存的變數內容為 12
--------------------------------*/
```

傳遞指標到函數 (4/4)

- prog9_7內，指標ptr與p1均指向同一個變數

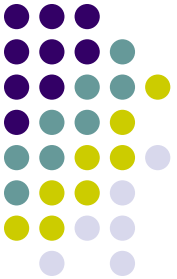


傳遞指標的應用

- 下面的程式是透過位址來改變呼叫端變數的內容

```
01  // prog9_8, 傳遞指標的應用
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void add10(int *);          // add10()函數的原型
06  int main(void)
07  {
08     int a=5;                 // 設定變數 a 的值為 5
09
10     cout << "呼叫 add10()之前, a=" << a << endl;
11     add10(&a);
12     cout << "呼叫 add10()之後, a=" << a << endl;
13
14     system("pause");
15     return 0;
16  }
17  void add10(int *p1)
18  {
19     *p1=*p1+10;
20     return;
21  }
```

```
/* prog9_8 OUTPUT------
呼叫 add10()之前, a=5
呼叫 add10()之後, a=15
----------------------*/
```



錯誤的示範 (1/4)

- 有些運算必須透過指標傳遞才能達成，下面為錯誤的示範

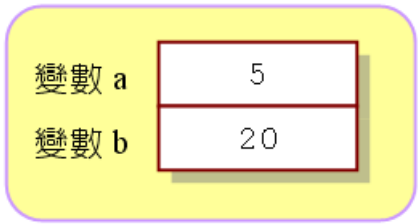
```
01  // prog9_9, 將 a 與 b 值互換(錯誤示範)
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void swap(int,int);
06  int main(void)
07  {
08     int a=5,b=20;
09     cout << "交換前... a=" << a << ", b=" << b << endl;
10     swap(a,b);
11     cout << "交換後... a=" << a << ", b=" << b << endl;
12
13     system("pause");
14     return 0;
15  }
16  void swap(int x,int y)          // 定義 swap()函數
17  {
18     int tmp=x;
19     x=y;
20     y=tmp;
21     return;
22  }
```

```
/* prog9_9 OUTPUT---
交換前... a=5, b=20
交換後... a=5, b=20
--------------------*/
```

錯誤的示範 (2/4)

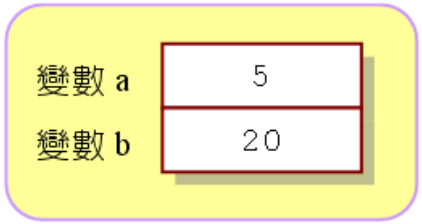
- 記憶體的配置情形

執行完第8行後，記憶體的配置情形

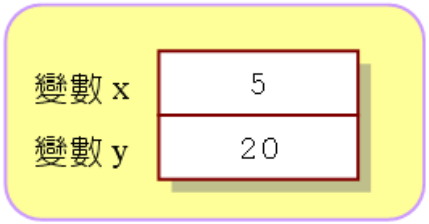


於主函數裡的變數

進入swap() 函數時，記憶體配置的情形



於主函數裡的變數

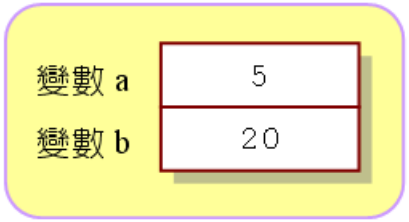


於 swap() 裡的變數

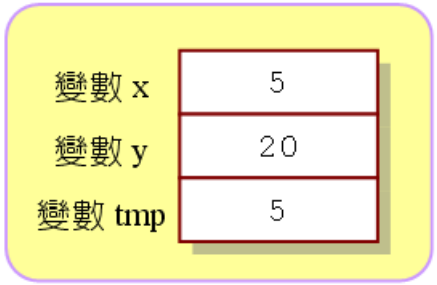
```
01  // prog9_9, 將a與b值...
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void swap(int,int);
06  int main(void)
07  {
08     int a=5,b=20;
09     cout << "交換前...;
10     swap(a,b);
11     cout << "交換後...;
12
13     system("pause");
14     return 0;
15  }
16  void swap(int x,int y)
17  {
18     int tmp=x;
19     x=y;
20     y=tmp;
21     return;
22  }
```

錯誤的示範 (3/4)

執行完第18行後，記憶體配置的情形

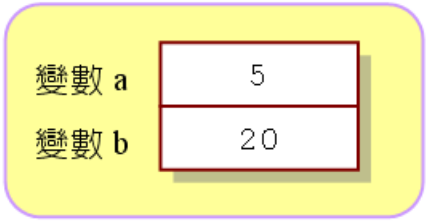


於主函數裡的變數

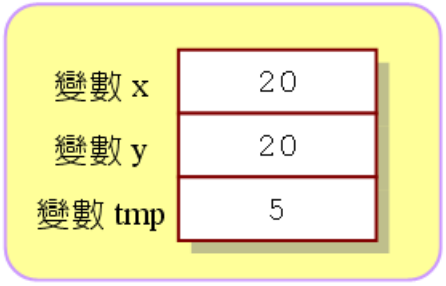


於 swap() 裡的變數

執行完第19行後，記憶體配置的情形



於主函數裡的變數

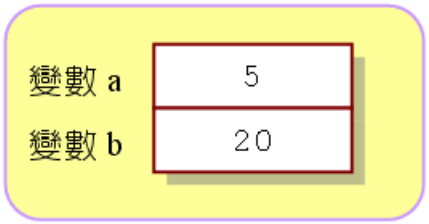


於 swap() 裡的變數

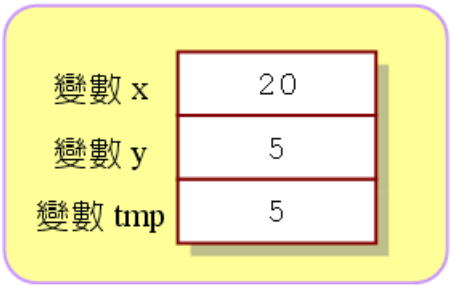
```
01  // prog9_9, 將a與b值...
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void swap(int,int);
06  int main(void)
07  {
08     int a=5,b=20;
09     cout << "交換前...;
10     swap(a,b);
11     cout << "交換後...;
12
13     system("pause");
14     return 0;
15  }
16  void swap(int x,int y)
17  {
18     int tmp=x;
19     x=y;
20     y=tmp;
21     return;
22  }
```

錯誤的示範 (4/4)

執行完第20行後，記憶體配置的情形

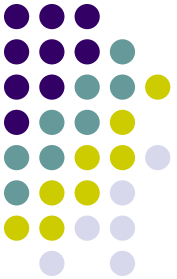


於主函數裡的變數



於 swap() 裡的變數

```
01  // prog9_9, 將a與b值...
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void swap(int,int);
06  int main(void)
07  {
08     int a=5,b=20;
09     cout << "交換前...;
10     swap(a,b);
11     cout << "交換後...;
12
13     system("pause");
14     return 0;
15  }
16  void swap(int x,int y)
17  {
18     int tmp=x;
19     x=y;
20     y=tmp;
21     return;
22  }
```



正確的範例 (1/4)

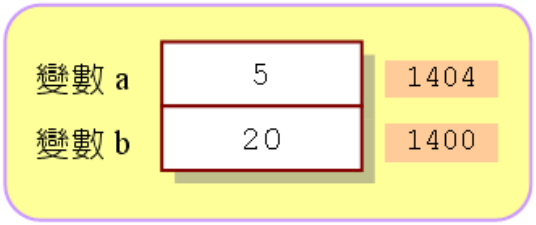
```
01  // prog9_10, 將 a 與 b 值互換(正確範例)
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void swap(int *,int *);          // 函數 swap()原型的宣告
06  int main(void)
07  {
08     int a=5,b=20;
09     cout << "交換前... a=" << a << ", b=" << b << endl;
10     swap(&a,&b);                  // 呼叫 swap()函數,並傳入 a 與 b 的位址
11     cout << "交換後... a=" << a << ", b=" << b << endl;
12
13     system("pause");
14     return 0;
15  }
16  void swap(int *p1,int *p2)       // swap()函數的定義
17  {
18     int tmp=*p1;
19     *p1=*p2;
20     *p2=tmp;
21     return;
22  }
```

- 將程式prog9_9修改成prog9_10

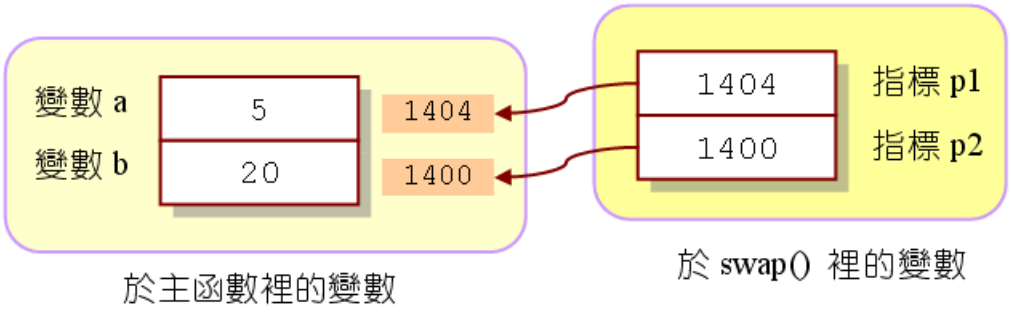
正確的範例 (2/4)

- 記憶體的配置情形

執行完第8行後，記憶體配置的情形



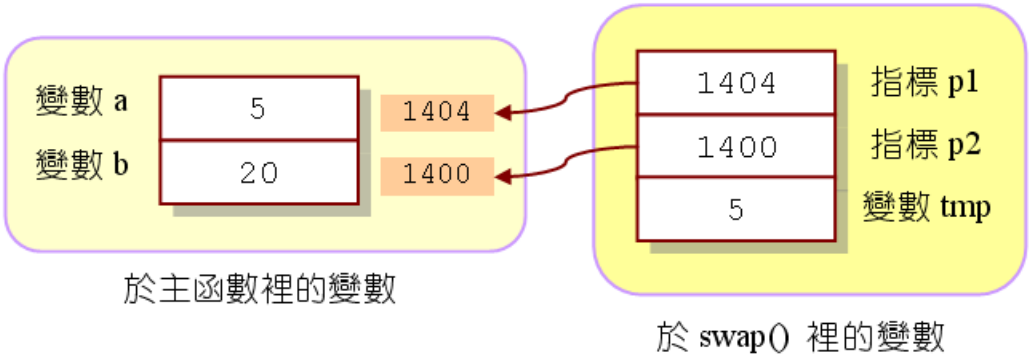
進入swap() 函數時，記憶體配置的情形



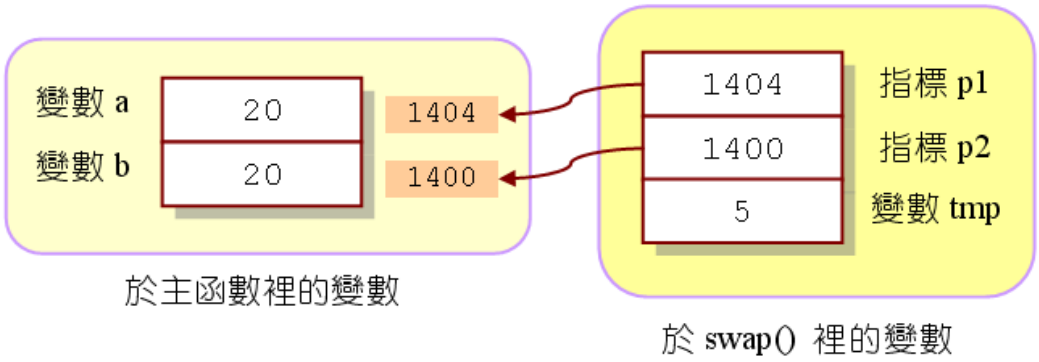

```
01  // prog9_10, 將 a 與 b 值互換
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void swap(int *,int *);
06  int main(void)
07  {
08     int a=5,b=20;
09     cout << "交換前...;
10     swap(&a,&b);
11     cout << "交換後...;
12
13     system("pause");
14     return 0;
15  }
16  void swap(int *p1,int *p2)
17  {
18     int tmp=*p1;
19     *p1=*p2;
20     *p2=tmp;
21     return;
22  }
```

正確的範例 (3/4)

執行完第18行後，記憶體配置的情形



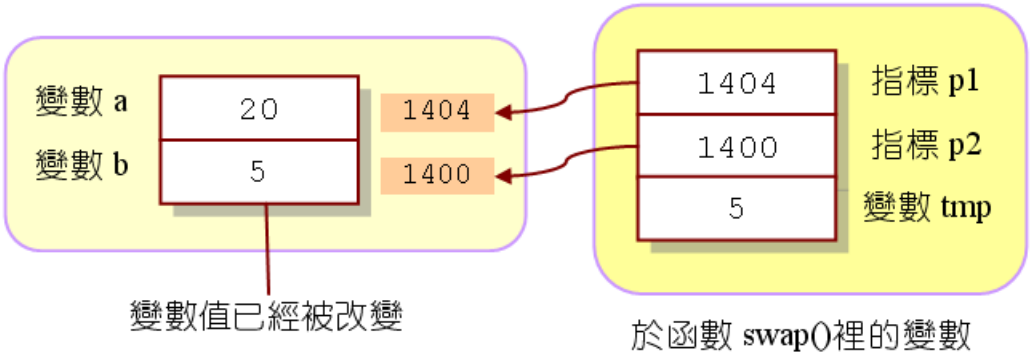
執行完第19行後，記憶體配置的情形



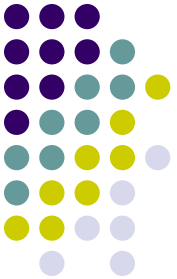

```
01  // prog9_10, 將 a 與 b 值互換
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void swap(int *,int *);
06  int main(void)
07  {
08     int a=5,b=20;
09     cout << "交換前...;
10     swap(&a,&b);
11     cout << "交換後...;
12
13     system("pause");
14     return 0;
15  }
16  void swap(int *p1,int *p2)
17  {
18     int tmp=*p1;
19     *p1=*p2;
20     *p2=tmp;
21     return;
22  }
```

正確的範例 (4/4)

執行完第20行後，記憶體配置的情形



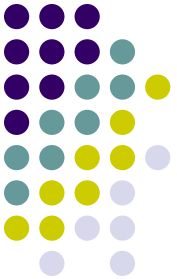

```
01  // prog9_10, 將 a 與 b 值互換
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void swap(int *,int *);
06  int main(void)
07  {
08     int a=5,b=20;
09     cout << "交換前...;
10     swap(&a,&b);
11     cout << "交換後...;
12
13     system("pause");
14     return 0;
15  }
16  void swap(int *p1,int *p2)
17  {
18     int tmp=*p1;
19     *p1=*p2;
20     *p2=tmp;
21     return;
22  }
```



傳回值為指標的函數 (1/4)

- 從函數傳回指標的格式

```
傳回值型態 *函數名稱(資料型態  引數)
{
    // 函數的本體
}
```



傳回值為指標的函數 (2/4)

- 下面的範例說明如何從函數傳回指標

```
01  // prog9_11, 由函數傳回指標
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int *max(int *,int *);          // 宣告函數 max()的原型
06  int main(void)
07  {
08     int a=12,b=17,*ptr;
09     ptr=max(&a,&b);
10     cout << "max=" << *ptr << endl;
11
12     system("pause");
13     return 0;
14  }
15  int *max(int *p1, int *p2)
16  {
17     if(*p1>*p2)
18        return p1;
19     else
20        return p2;
21  }
```

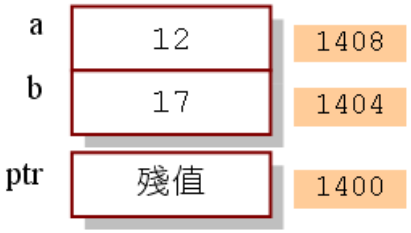
傳回 p1 與 p2 所指向之整數中，數值較大之整數的位址

```
/* prog9_11 OUTPUT---
max=17
--------------------*/
```

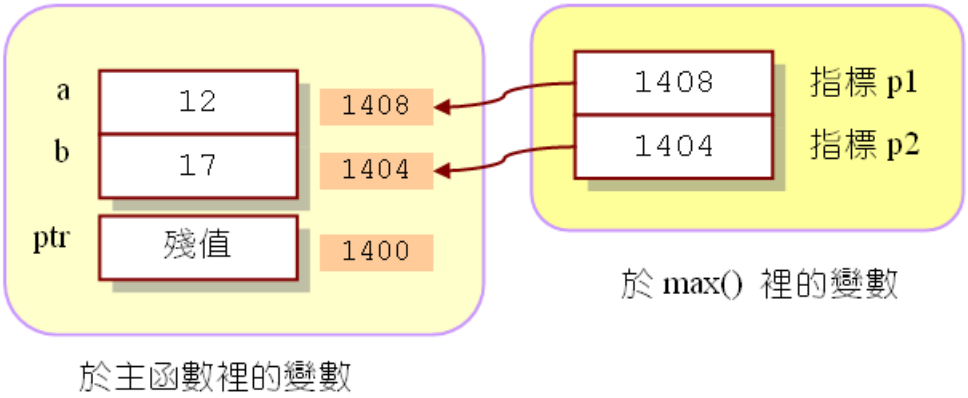
傳回值為指標的函數 (3/4)

- 記憶體的配置情形

執行完第8行後，記憶體配置的情形



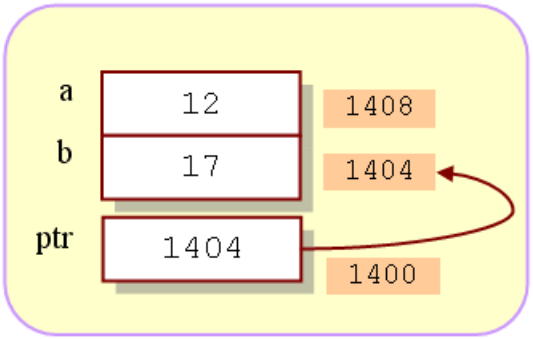
進入max() 函數時，記憶體配置的情形




```
01  // prog9_11, 由函數傳回指標
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int *max(int *,int *);
06  int main(void)
07  {
08     int a=12,b=17,*ptr;
09     ptr=max(&a,&b);
10     cout << "max=" << ...;
11
12     system("pause");
13     return 0;
14  }
15  int *max(int *p1, int *p2)
16  {
17     if(*p1>*p2)
18        return p1;
19     else
20        return p2;
21  }
```

傳回值為指標的函數 (4/4)

執行完第9行後，記憶體配置的情形

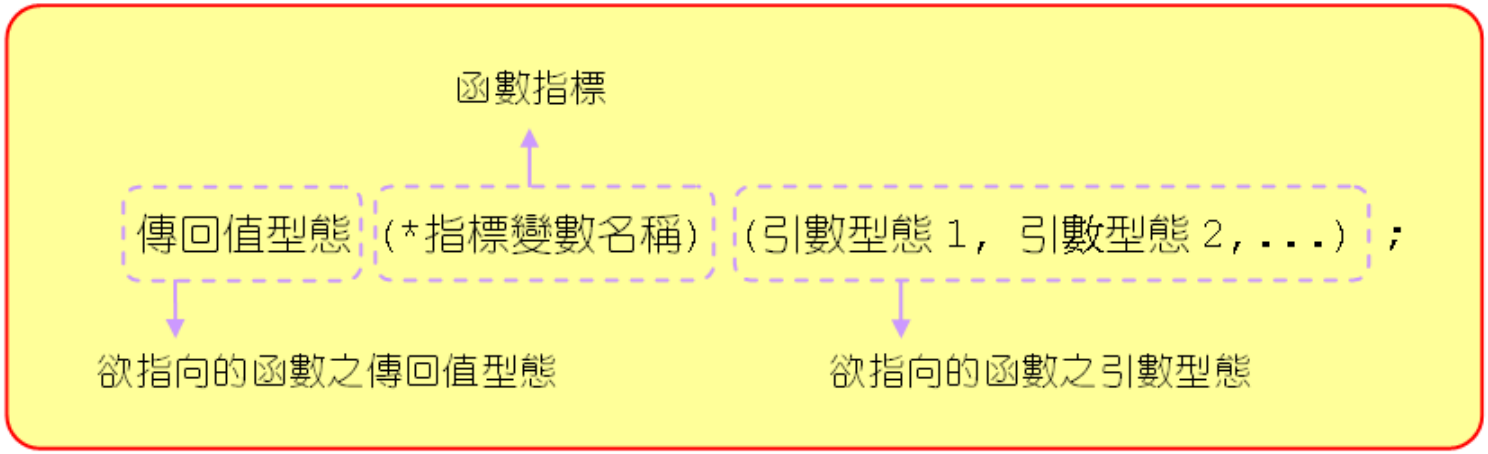


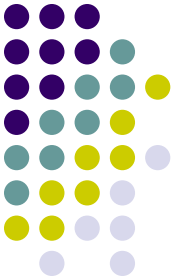
於主函數裡的變數

```
01  // prog9_11, 由函數傳回指標
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int *max(int *,int *);
06  int main(void)
07  {
08     int a=12,b=17,*ptr;
09     ptr=max(&a,&b);
10     cout << "max=" << ...;
11
12     system("pause");
13     return 0;
14  }
15  int *max(int *p1, int *p2)
16  {
17     if(*p1>*p2)
18        return p1;
19     else
20        return p2;
21  }
```

函數指標

- 函數名稱本身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()，原型定義如下

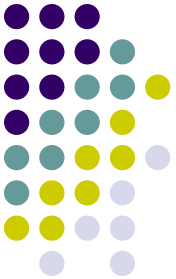
```
int square(int);        // 定義 square()函數的原型
```

- 想把函數指標pf指向square()函數，可寫成下面的敘述

```
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()
```

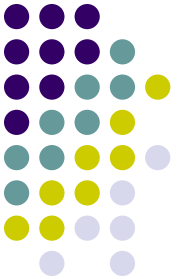


函數指標的用法 (2/2)

- 下面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
----------------------*/
```



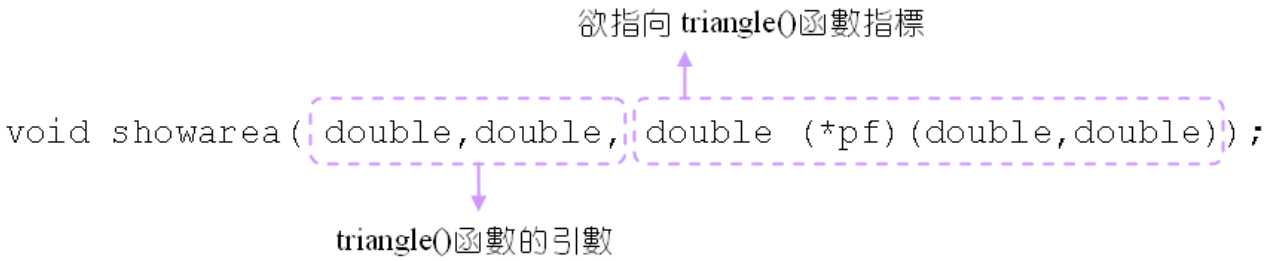
傳遞函數到其它函數中 (1/3)

- 下面的步驟說明如何傳遞函數到其它函數中

  設triangle() 函數可傳回三角形的面積，其原型如下

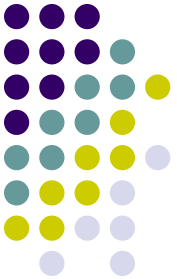
```
double triangle(double,double);
```

  現希望撰寫showarea() 函數，可傳入triangle() 函數，並傳回triangle() 的傳回值，則showarea() 可定義如下



showarea() 函數的定義如下

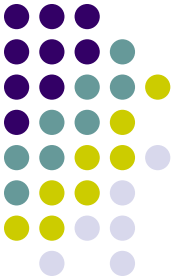
```
void showarea(double x,double y,double (*pf)(double,double))
{
   cout << (*pf)(x,y) << endl;          // 呼叫函數指標所指向的函數
}
```



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

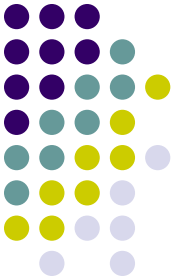
```
01  // prog9_13, 傳遞函數到其它函數中
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  double triangle(double,double),rectangle(double,double);
06  void showarea(double,double,double (*pf)(double,double));
07  int main(void)
08  {
09     cout << "triangle(6,3.2)=";
10     showarea(6,3.2,triangle);        // 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");                 /* prog9_13 OUTPUT-----
15     return 0;                        triangle(6,3.2)=9.6
16  }                                   rectangle(4,6.1)=24.4
                                        ----------------------*/
```



傳遞函數到其它函數中 (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
----------------------*/
```



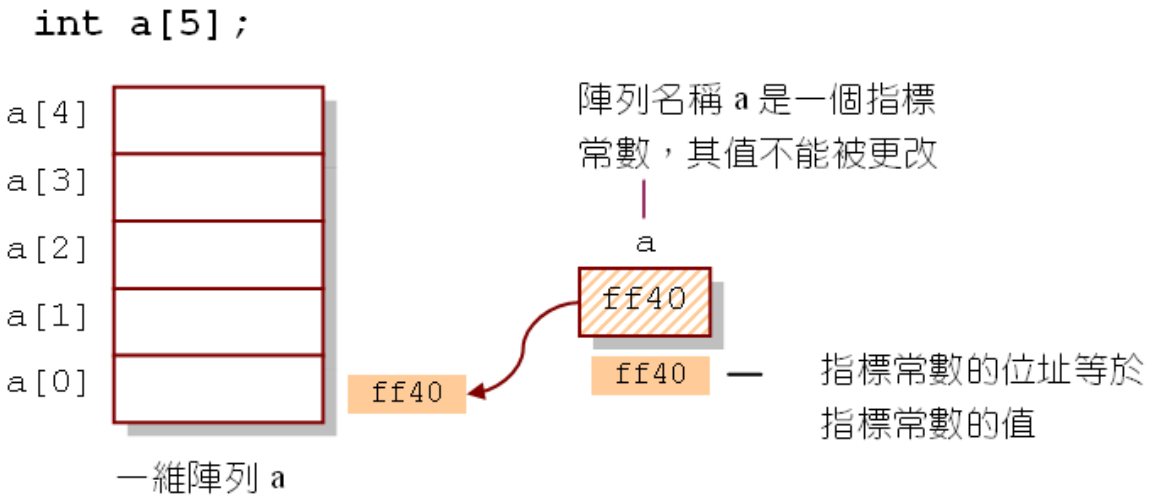
指標的算術運算 (1/6)

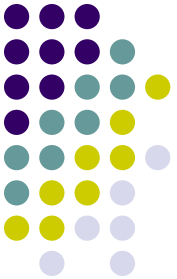
- 指標的算術運算（arithmetic operation），是指標內所存放的位址做加法或減法運算
- 指標做加法或減法運算時，是針對它所指向之資料型態的大小來處理
- 指標的算術運算多半是用在存取陣列元素的操作

指標的算術運算 (2/6)

- 陣列有個巧妙的設計，也就是

陣列名稱本身是一個存放位址的「指標常數」，它指向陣列的位址





指標的算術運算 (3/6)

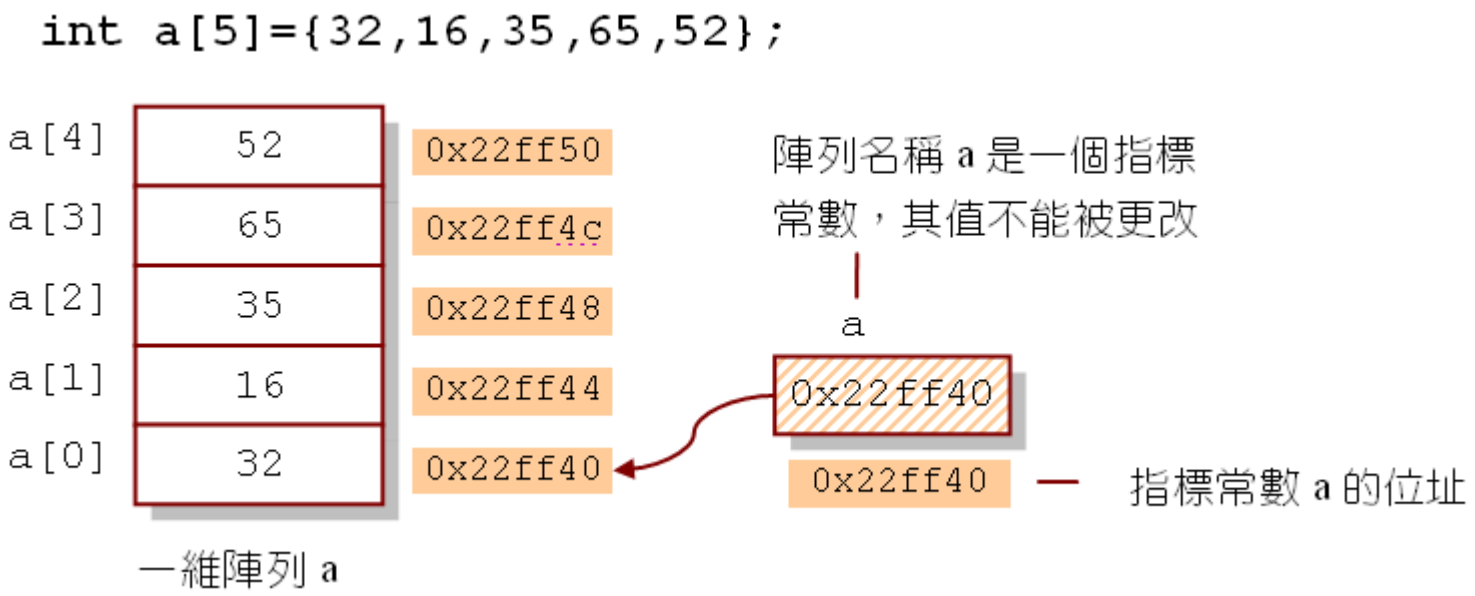
- 下面的程式驗證陣列名稱是一個指向陣列位址的指標

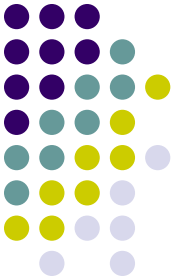
```
01  // prog9_14, 指標常數的值與位址
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07     int i,a[5]={32,16,35,65,52};
08
09     cout << "a=" << a << endl;
10     cout << "&a=" << &a << endl;
11     for(i=0;i<5;i++)
12        cout << "&a[" << i << "]=" << &a[i] << endl;
13
14     system("pause");
15     return 0;
16  }
```

```
/* prog9_14 OUTPUT------------------
a=0x22ff40      ——  指標常數 a 的值
&a=0x22ff40     ——  指標常數 a 的位址
&a[0]=0x22ff40  }
&a[1]=0x22ff44  }
&a[2]=0x22ff48  }   陣列元素的位址
&a[3]=0x22ff4c  }
&a[4]=0x22ff50  }
----------------------------------*/
```

指標的算術運算 (4/6)

- 陣列a於記憶體中的配置圖





指標的算術運算 (5/6)

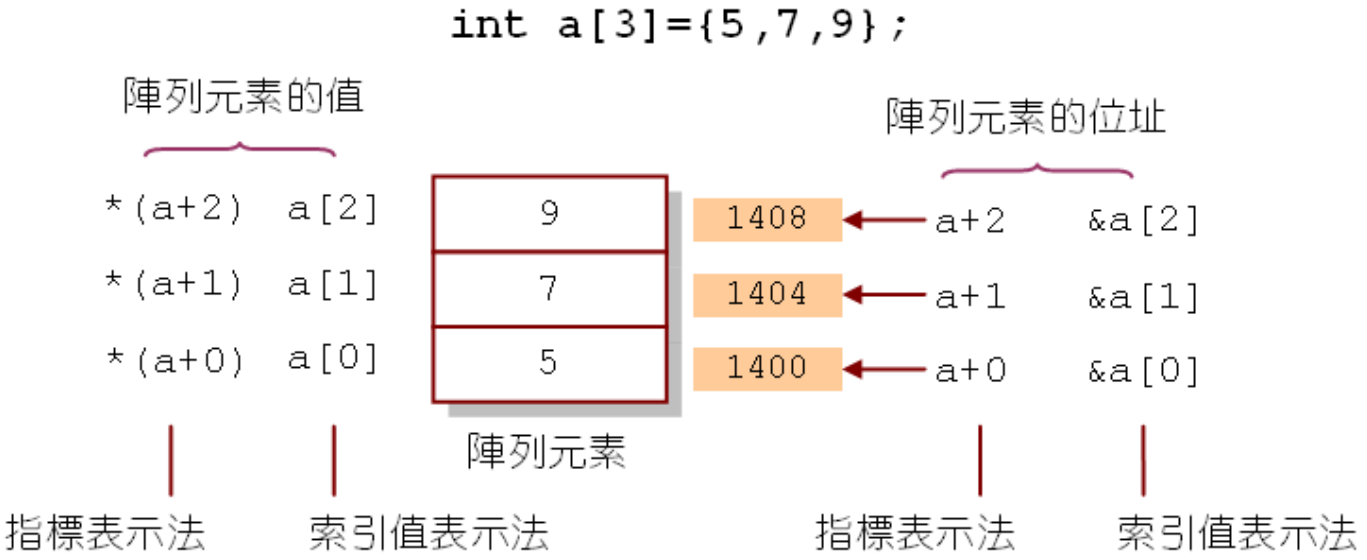
- 下面的範例是利用指標常數來存取陣列的內容

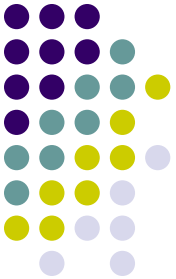
```
01  // prog9_15, 利用指標常數來存取陣列的內容
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07     int a[3]={5,7,9};
08     cout << "a[0]=" << a[0] << ", *(a+0)=" << *(a+0) << endl;
09     cout << "a[1]=" << a[1] << ", *(a+1)=" << *(a+1) << endl;
10     cout << "a[2]=" << a[2] << ", *(a+2)=" << *(a+2) << endl;
11
12     system("pause");
13     return 0;
14  }
```

```
/* prog9_15 OUTPUT---
a[0]=5, *(a+0)=5
a[1]=7, *(a+1)=7
a[2]=9, *(a+2)=9
--------------------*/
```

指標的算術運算 (6/6)

- 陣列a於記憶體中的配置圖





利用指標存取一維陣列的元素

- 下面的程式利用指標計算一維陣列內所有元素的總和

```
01  // prog9_16, 利用指標求陣列元素和
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07     int a[3]={5,7,9};
08     int i,sum=0;
09     for(i=0;i<3;i++)
10        sum+=*(a+i);          // 加總陣列元素的總和
11     cout << "sum=" << sum << endl;
12
13     system("pause");
14     return 0;
15  }
```

```
/* prog9_16 OUTPUT---
sum=21
--------------------*/
```

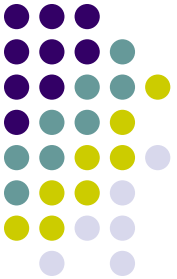
使用指標常數應注意的事項

- 陣列名稱a是一個指標常數，因此不能撰寫如下的敘述：

```
a=a+1;     // 錯誤，陣列名稱 a 是一個指標常數，不能變更它的值
```

- 若是宣告一個指向整數的指標來指向陣列a，則下面的敘述是合語法的：

```
int *ptr=a;              // 宣告指向整數的指標 ptr 來指向陣列 a

ptr=ptr+1;               // 將指標 ptr 指向陣列 a 下一個元素的位址
```



指標常數的使用範例

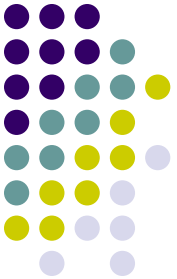
- 下面的範例修改自prog9_16

```
01  // prog9_17, 利用指標求陣列元素和
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07     int a[3]={5,7,9};
08     int i,sum=0;
09     int *ptr=a;                    // 設定指標 ptr 指向陣列 a
10     for(i=0;i<3;i++)
11        sum+=*(ptr++);              // 陣列元素值的累加
12     cout << "sum=" << sum << endl;
13
14     system("pause");
15     return 0;
16  }
```

注意不能把程式的第 11 行寫成這樣的敘述：

```
sum+=*(a++);        // 錯誤，因為 a 是指標常數
```

```
/* prog9_17 OUTPUT---
sum=21
--------------------*/
```



利用指標傳遞一維陣列到函數裡

- 可接收一維陣列之函數的定義格式如下

```
傳回值型態 函數名稱(資料型態 *陣列名稱)
{
    // 函數的內容
}
```

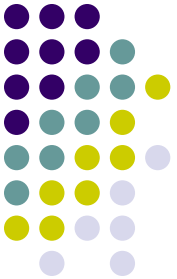
用來接收一維陣列的位址

- 例如func() 的定義可撰寫成如下的格式

```
void func(int *arr)  // 函數 func()，可接收一維的整數陣列
{
    // 函數的內容
}
```

- 在呼叫函數func() 時

```
int A[]={12,43,32,18,98};      // 宣告整數陣列 A，並設定初值
func(A);                       // 呼叫 func 函數，並傳入陣列 A
```

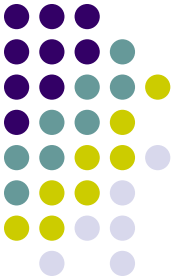


以指標傳遞一維陣列的範例 (1/2)

- 下面的範例說明如何以指標傳遞一維陣列

```
01  // prog9_18, 將陣列第 n 個元素的值取代為 num
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void replace(int *,int,int);                    // 宣告 replace()函數的原型
06  int main(void)
07  {
08     int a[5]={1,2,3,4,5};
09     int i,num=100;
10     cout << "置換前，陣列的內容為 ";
11     for(i=0;i<5;i++)                              // 置換前印出陣列的內容
12        cout << a[i] << " ";
13     cout << endl;
```

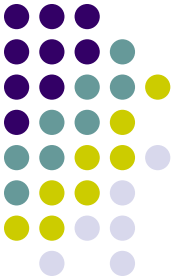
```
/* prog9_18 OUTPUT------------
置換前，陣列的內容為 1 2 3 4 5
置換後，陣列的內容為 1 2 3 100 5
-----------------------------*/
```



以指標傳遞一維陣列的範例 (2/2)

```
14      replace(a,4,num);                          // 呼叫函數 replace()
15      cout << "置換後，陣列的內容為 ";
16      for(i=0;i<5;i++)                           // 置換後印出陣列的內容
17         cout << a[i] << " ";
18      cout << endl;
19
20      system("pause");
21      return 0;
22   }
23   void replace(int *ptr,int n,int num)
24   {
25      *(ptr+n-1)=num;                            // 將陣列第 n 個元素設值為 num
26      return;
27   }
```

```
/* prog9_18 OUTPUT------------
置換前，陣列的內容為 1 2 3 4 5
置換後，陣列的內容為 1 2 3 100 5
-----------------------------*/
```

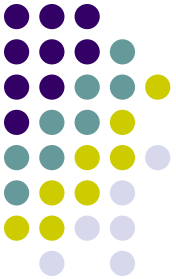


利用函數傳回指標 (1/2)

- 下面的程式是示範如何利用函數傳回指標

```
/* prog9_19 OUTPUT----
陣列的內容為 3 1 7 2 6
最大值為 7
----------------------*/
```

```
01  // prog9_19, 函數傳回值為指標
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int *maximum(int *);              // 宣告 maximum()函數的原型
06  int main(void)
07  {
08     int a[5]={3,1,7,2,6};
09     int i,*ptr;
10     cout << "陣列的內容為 ";
11     for(i=0;i<5;i++)               // 印出陣列的內容
12        cout << a[i] << " ";
13     cout << endl;
14     ptr=maximum(a);                // 呼叫 maximum()函數，並傳入陣列 a
15     cout << "最大值為 " << *ptr << endl;
```



利用函數傳回指標 (2/2)

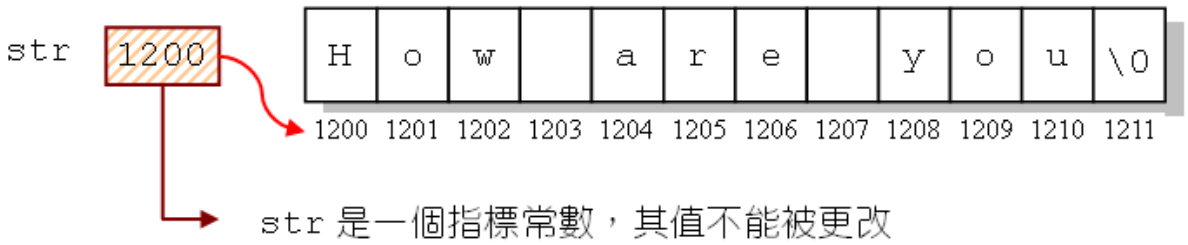
```
16
17      system("pause");
18      return 0;
19  }
20  int *maximum(int *arr)          // 定義 maximum()函數
21  {
22      int i,*max;
23      max=arr;                    // 設定指標 max 指向陣列的第一個元素
24      for(i=1;i<5;i++)
25          if(*max < *(arr+i))
26              max=arr+i;
27      return max;                 // 傳回最大值之元素的位址
28  }
```

```
/* prog9_19 OUTPUT----
陣列的內容為 3 1 7 2 6
最大值為 7
----------------------*/
```

以指標變數指向字串 (1/2)

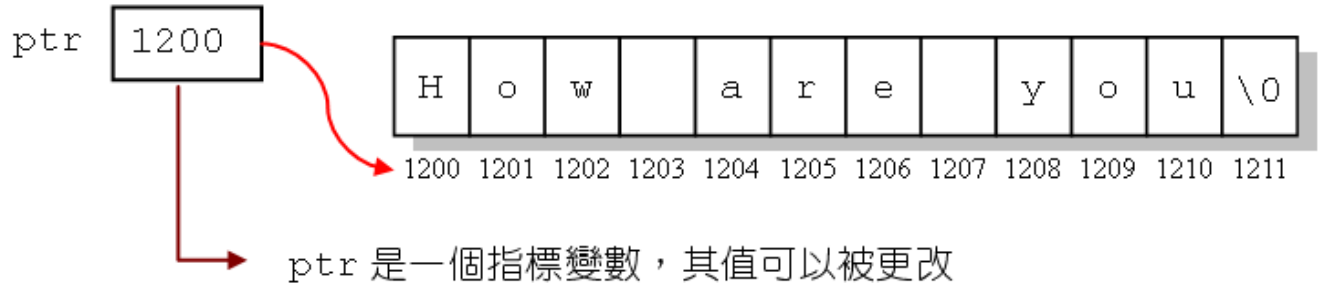
- 字串 "How are you?" 可以利用字元陣列來儲存

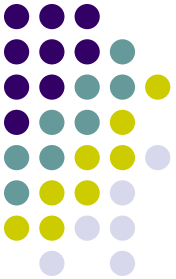
```
char str[]="How are you?";
```



- 字串也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

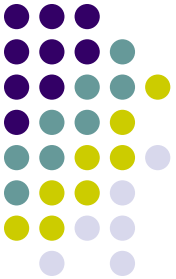
```
01  // prog9_20, 以指標變數指向字串
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07     char name[20];
08     char *ptr="How are you?";          // 將指標指向字串"How are you?"
09
10     cout << "What's your name? ";
11     cin.getline(name,20);               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

- 指標陣列的宣告格式

```
資料型態 *陣列名稱[元素個數];     // 宣告指標陣列
```

若於程式碼裡宣告如下的敘述

```
int *ptr[3];     // 宣告指標陣列 ptr，可存放 3 個指向整數的指標
```

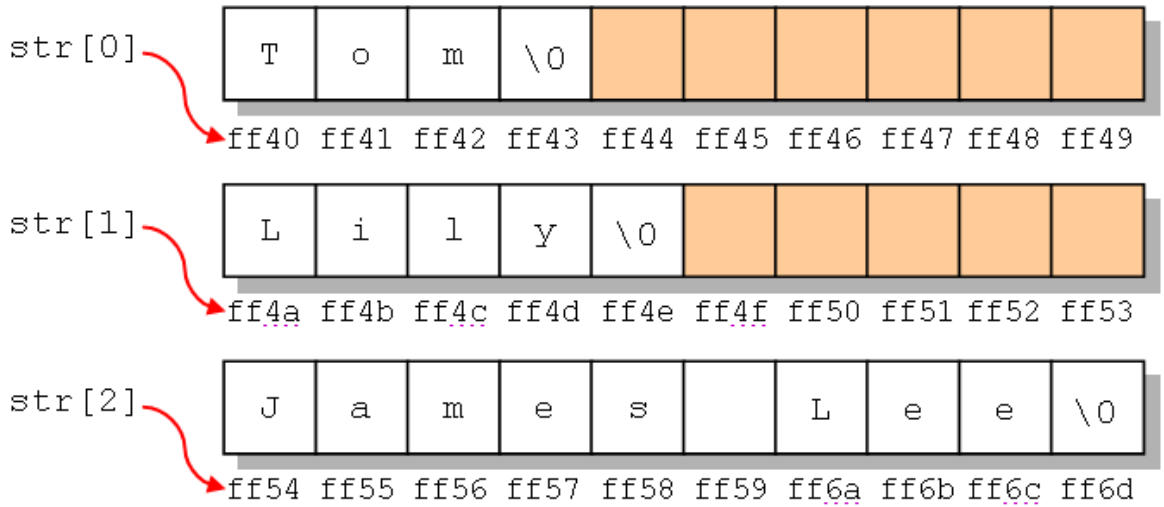
則有三個指向整數的指標可以使用

- 一維的指標陣列常用在字串陣列的儲存

二維字元陣列的儲存方式 (1/3)

- 以二維的字元陣列來儲存字串陣列

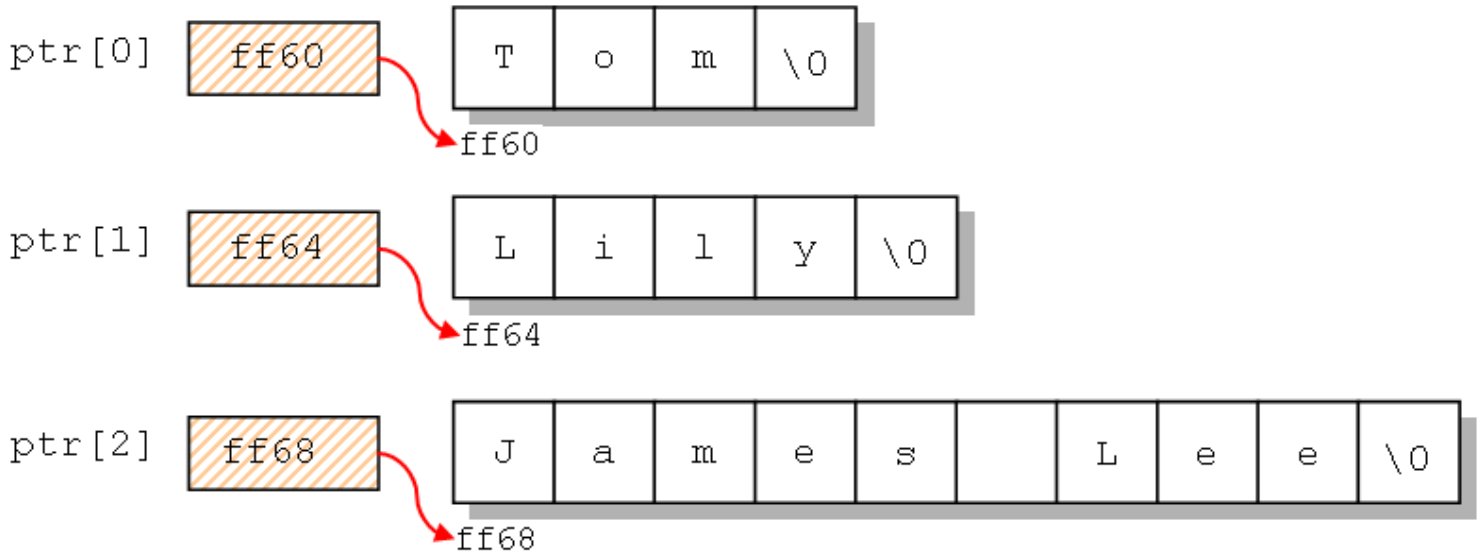
```
char str[3][10]={"Tom", "Lily", "James Lee"};
```

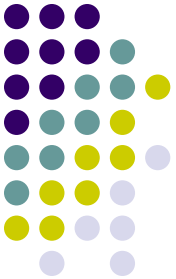


二維字元陣列的儲存方式 (2/3)

- 以指標陣列的方式來儲存字串陣列

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





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
----------------------*/
```


-The End-