第二章(II)：C++的函式與變數

Su - Huang

OUTLINE

1. 函式
2. 引入函式庫與常用函式庫
3. 指標
4. Practice

函式

多載 (1/5) overloading a Function Name

- 多載（overloading），是指相同的函數名稱，如果引數個數不同，或者是引數個數相同、型態不同的話，函數便具有不同的功能
- 以一個簡單的例子說明「函數的多載」之使用

```
01  // prog7_6, 引數型態不同的函數多載
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int add(int,int);                    // 以多載的方式宣告函數原型
06  double add(double,double);

/* prog7_6 OUTPUT---

10+20=30
2.3+3.5=5.8

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

多載 (2/5)

```
                                                    /* prog7_6 OUTPUT---
07  int main(void)                                  10+20=30
08  {                                               2.3+3.5=5.8
09     int a=10,b=20;                               --------------------*/
10     double x=2.3,y=3.5;
11     cout << a << "+" << b << "=" << add(a,b) << endl;
12     cout << x << "+" << y << "=" << add(x,y) << endl;
13     system("pause");
14     return 0;
15  }
16
17  int add(int i,int j)                    // 自訂函數 add()
18  {
19     return i+j;                          // 傳回 i+j 的值
20  }
21
22  double add(double i,double j)           // 自訂函數 add()
23  {
24     return i+j;                          // 傳回 i+j 的值
25  }
```

多載 (3/5)

- 如果只有傳回值型態不同，則不能多載
  - 舉例來說，某個函數的原型如下
  - 上面的函數原型會與下面的原型相衝突而產生錯誤

```
int func(int,int);    // 函數原型，傳回值型態為 int
long func(int,int);   // 函數原型，傳回值型態為 long
```

- 只有傳回值型態不同，則會讓編譯器難以分辨到底該使用哪一個函數

多載 (4/5)

- 接下來再看一個引數個數不同的函數多載

```
01  // prog7_7, 引數個數不同的函數多載
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void print(void);                  // 以多載的方式宣告函數原型
06  void print(int);
07  void print(char,int);
08  int main(void)
09  {
10     cout << "calling print(), ";
11     print();
12     cout << "calling print(8), ";
13     print(8);
14     cout << "calling print('+',3), ";
15     print('+',3);
16     system("pause");
17     return 0;
18  }
19
```

```
/* prog7_7 OUTPUT---------
calling print(), *****
calling print(8), ********
calling print('+',3), +++
-------------------------*/
```

多載 (5/5)

```
20  void print(void)                    // 沒有引數的print()，印出5個*
21  {
22     print(5);                        // 呼叫26~33行的print()，並傳入整數5
23     return;
24  }
25
26  void print(int a)                   // 有一個引數的print()，印出a個*
27  {
28     int i;
29     for(i=0;i<a;i++)
30        cout << "*";
31     cout << endl;
32     return;
33  }
34
35  void print(char ch,int a)           // 有二個引數的print()，印出a個ch
36  {
37     int i;
38     for(i=0;i<a;i++)
39        cout << ch;
40     cout << endl;
41     return;
42  }
```

```
/* prog7_7 OUTPUT---------
calling print(), *****
calling print(8), ********
calling print('+',3), +++
-------------------------*/
```

預設引數 (1/4)

- 未傳入足夠的引數到函數時，預設的引數值就會被使用，這種方式稱為「預設引數」（default argument）
- 要設定預設，可在定義原型時，於引數後面設值給它

```
double circle(double, double pi=3.14);
```

預設引數 (2/4)

- 下面的程式是函數引數預設值的使用範例

```
01  // prog7_8, 引數的預設值
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  double circle(double,double pi=3.14);  // 函數原型,第 2 個引數預設為 3.14
06  int main(void)
07  {
08     cout << "circle(2.0,3.14159)=" << circle(2.0,3.14159) << endl;
09     cout << "circle(2.0)=" << circle(2.0) << endl;
10     system("pause");
11     return 0;
12  }
13
14  double circle(double r, double pi)  // circle()函數的定義，計算圓面積
15  {
16     return (pi*r*r);
17  }
```

```
/* prog7_8 OUTPUT---------
circle(2.0,3.14159)=12.5664
circle(2.0)=12.56
-------------------------*/
```

預設引數 (3/4)

- 沒有使用預設值的引數，要放置在引數列的左邊
  - 舉例來說，函數原型如下

```
void func(int, double, int n=3, char ch='k');
```

  - 下面都是合法的func() 函數呼叫

```
func(5, 1.9);              // n 預設為 3，ch 預設為'k'
func(8, 6.3, 4);           // ch 預設為'k'
func(4, 3.7, 9, 'a');      // 均不使用預設值
```

  - 下列的函數呼叫，會造成編譯時期或是邏輯上的錯誤：

```
func();                    // 最少必須有兩個引數
func(6);                   // 最少必須有兩個引數
func(2, 1.9, 'b');         // 邏輯錯誤的函數呼叫
```

預設引數 (4/4)

- 下面的程式是有加入引數預設值的函數呼叫

```
01  // prog7_9, 引數的預設值
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int sum(int start=1,int end=10,int di=1);  // 函數原型
06  int main(void)
07  {
08     cout << "sum()=" << sum() << endl;
09     cout << "sum(2)=" << sum(2) << endl;
10     cout << "sum(2,8)=" << sum(2,8) << endl;
11     cout << "sum(1,15,3)=" << sum(1,15,3) << endl;
12     system("pause");
13     return 0;
14  }
15
16  int sum(int start,int end,int di)          // 計算數值的累加
17  {
18     int i,total=0;
19     for(i=start;i<=end;i+=di)
20        total+=i;
21     return total;
22  }
```

```
/* prog7_9 OUTPUT---
sum()=55
sum(2)=54
sum(2,8)=35
sum(1,15,3)=35
-------------------*/
```

OUTLINE

1. 函數
2. 引入函式庫與常用函式庫
3. 指標
4. Practice

引入函式庫與常用函式庫

Predefined Functions

- Libraries full of functions for our use!
- Two types:
  - Those that return a value
  - Those that do not (void)
- Must "#include" appropriate library
  - e.g.,
    - <cmath>, <cstdlib> (Original "C" libraries)
    - <iostream> (for cout, cin)

Using Predefined Functions

- Math functions very plentiful
  - Found in library <cmath.h>
  - Most return a value (the "answer")
- Example: theRoot = sqrt(9.0);
  - Components:
    sqrt = name of library function
    theRoot = variable used to assign "answer" to
    9.0 = argument or "starting input" for function
  - In I-P-O:
    - I = 9.0
    - P = "compute the square root"
    - O = 3, which is returned & assigned to theRoot

The Function Call

- Back to this assignment:

  theRoot = sqrt(9.0);

  - The expression "sqrt(9.0)" is known as a function *call*, or function *invocation*
  - The argument in a function call (9.0) can be a literal, a variable, or an expression
  - The call itself can be part of an expression:
    - bonus = sqrt(sales)/10;
    - A function call is allowed wherever it's legal to use an expression of the function's return type

A Larger Example:
Display 3.1 A Predefined Function That Returns a Value (1 of 2)

Display 3.1 A Predefined Function That Returns a Value

```
1  //Computes the size of a doghouse that can be purchased
2  //given the user's budget.
3  #include <iostream>
4  #include <cmath>
5  using namespace std;

6  int main( )
7  {
8      const double COST_PER_SQ_FT = 10.50;
9      double budget, area, lengthSide;

10     cout << "Enter the amount budgeted for your doghouse $";
11     cin >> budget;

12     area = budget/COST_PER_SQ_FT;
13     lengthSide = sqrt(area);
```

A Larger Example:
Display 3.1 A Predefined Function That Returns a Value (2 of 2)

```
14      cout.setf(ios::fixed);
15      cout.setf(ios::showpoint);
16      cout.precision(2);
17         cout << "For a price of $" << budget << endl
18           << "I can build you a luxurious square doghouse\n"
19           << "that is " << lengthSide
20           << " feet on each side.\n";

21      return 0;
22  }
```

SAMPLE DIALOGUE

```
Enter the amount budgeted for your doghouse $25.00
For a price of $25.00
I can build you a luxurious square doghouse
that is 1.54 feet on each side.
```

More Predefined Functions

- #include <cstdlib>
  - Library contains functions like:
    - abs() // Returns absolute value of an int
    - labs() // Returns absolute value of a long int
    - *fabs() // Returns absolute value of a float
  - *fabs() is actually in library <cmath>!
    - Can be confusing
    - Remember: libraries were added after C++ was "born," in incremental phases
    - Refer to appendices/manuals for details

More Math Functions

- pow(x, y)
  - Returns x to the power y
    double result, x = 3.0, y = 2.0;
    result = pow(x, y);
    cout << result;
    - Here 9.0 is displayed since $3.0^{2.0} = 9.0$
- Notice this function receives two arguments
  - A function can have any number of arguments, of varying data types

Even More Math Functions: **Display 3.2** Some Predefined Functions (1 of 2)

Display 3.2 Some Predefined Functions

NAME	DESCRIPTION	TYPE OF ARGUMENTS	TYPE OF VALUE RETURNED	EXAMPLE	VALUE	LIBRARY HEADER
sqrt	Square root	double	double	sqrt(4.0)	2.0	cmath
pow	Powers	double	double	pow(2.0,3.0)	8.0	cmath
abs	Absolute value for int	int	int	abs(-7) abs(7)	7 7	cstdlib
labs	Absolute value for long	long	long	labs(-70000) labs(70000)	70000 70000	cstdlib
fabs	Absolute value for double	double	double	fabs(-7.5) fabs(7.5)	7.5 7.5	cmath

Even More Math Functions: **Display 3.2** Some Predefined Functions (2 of 2)

ceil	Ceiling (round up)	double	double	ceil(3.2) ceil(3.9)	4.0 4.0	cmath
floor	Floor (round down)	double	double	floor(3.2) floor(3.9)	3.0 3.0	cmath
exit	End pro-gram	int	void	exit(1);	None	cstdlib
rand	Random number	None	int	rand( )	Varies	cstdlib
srand	Set seed for rand	unsigned int	void	srand(42);	None	cstdlib

Predefined Void Functions

- No returned value
- Performs an action, but sends no "answer"
- When called, it’s a statement itself
  - exit(1); // No return value, so not assigned
    - This call terminates program
    - void functions can still have arguments
- All aspects same as functions that "return a value"
  - They just don’t return a value!

Random Number Generator

- Return "randomly chosen" number
- Used for simulations, games
  - rand()
    - Takes no arguments
    - Returns value between 0 & RAND_MAX
  - Scaling
    - Squeezes random number into smaller range
      rand() % 6
    - Returns random value between 0 & 5
  - Shifting
    rand() % 6 + 1
    - Shifts range between 1 & 6 (e.g., die roll)

Random Number Seed

- Pseudorandom numbers
  - Calls to rand() produce given "sequence" of random numbers
- Use "seed" to alter sequence
  srand(seed_value);
  - void function
  - Receives one argument, the "seed"
  - Can use any seed value, including system time: srand(time(0));
  - time() returns system time as numeric value
  - Library <time> contains time() functions

Random Examples

- Random double between 0.0 & 1.0:
  (RAND_MAX – rand())/static_cast<double>(RAND_MAX)
  - Type cast used to force double-precision division
- Random int between 1 & 6:
  rand() % 6 + 1
  - "%" is modulus operator (remainder)
- Random int between 10 & 20:
  rand() % 10 + 10

使用 random number 的例子 (1)

```
#include <iostream>
#include <cstdlib>
using namespace std;

int main( )
{
    int month, day;
    cout << "Welcome to your friendly weather program.\n"
         << "Enter today's date as two integers for the month and the day:\n";
    cin >> month;
    cin >> day;
    srand(month*day);
    int prediction;
    char ans;
    cout << "Weather for today:\n";
```

使用 random number 的例子 (2)

```
do
   {
      prediction = rand() % 3;
      switch (prediction)
      {
         case 0:
            cout << "The day will be sunny!!\n";
            break;
         case 1:
            cout << "The day will be cloudy.\n";
            break;
         case 2:
            cout << "The day will be stormy!.\n";
            break;
         default:
            cout << "Weather program is not functioning properly.\n";
      }
      cout << "Want the weather for the next day?(y/n): ";
      cin >> ans;
   }while (ans == 'y' || ans == 'Y');
   cout << "That's it from your 24 hour weather program.\n";
   return 0;
}
```

練習題

- 寫一個四位數字產生器
- Basic: all different digits
- Adv:

OUTLINE

1. 函數
2. 引入函式庫與常用函式庫
3. 指標
4. Practice

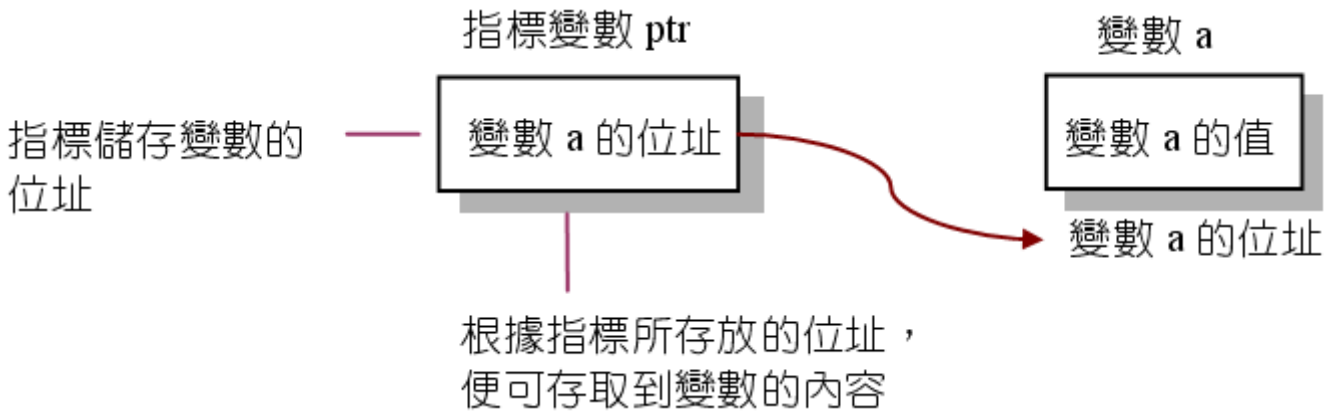
指標

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

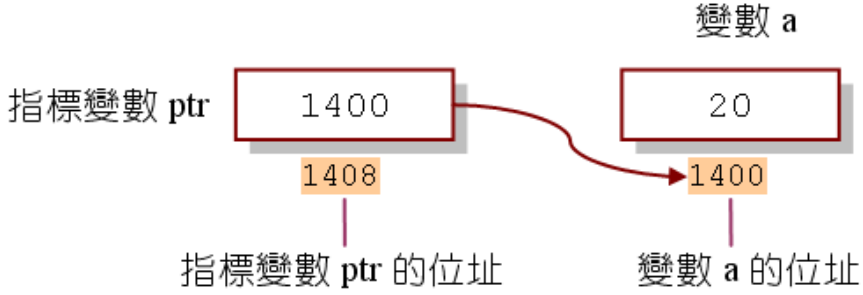
  " 指標 ptr 指向變數 a "

- 下面是指標ptr指向變數a的示意圖

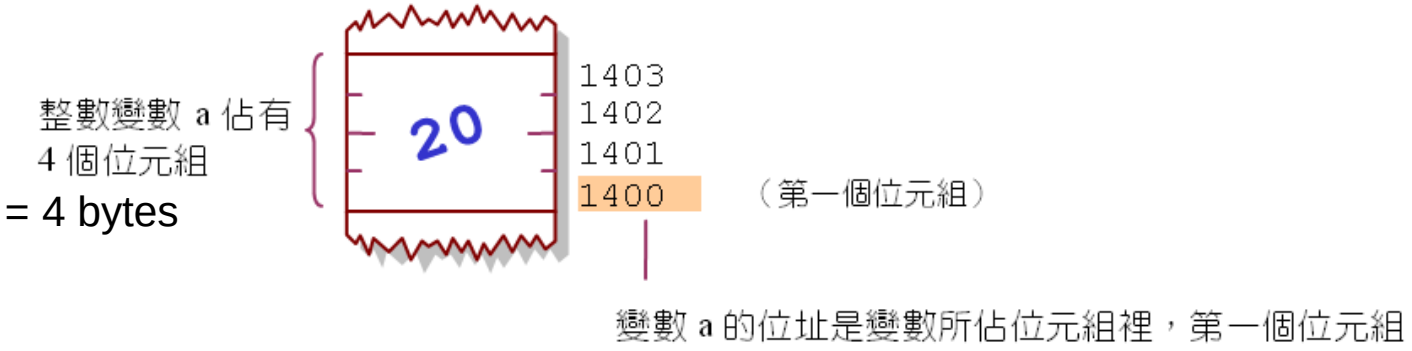


什麼是指標 (2/2)

- 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)

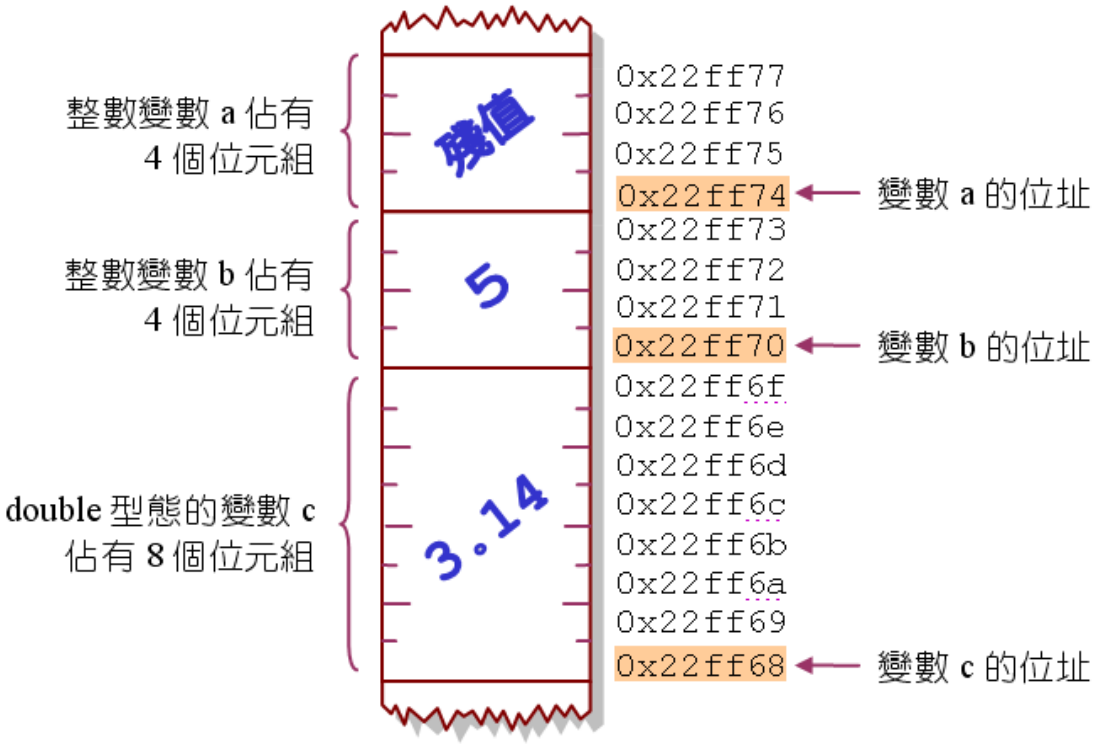
- 下面的程式印出變數的值、記憶體的大小，與變數的位址

```
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
```

同理 如何宣告一個指向double的指標變數?

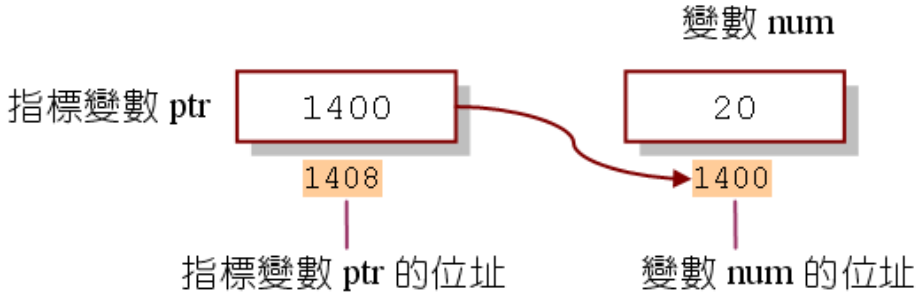
Ans: double *ptr;

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

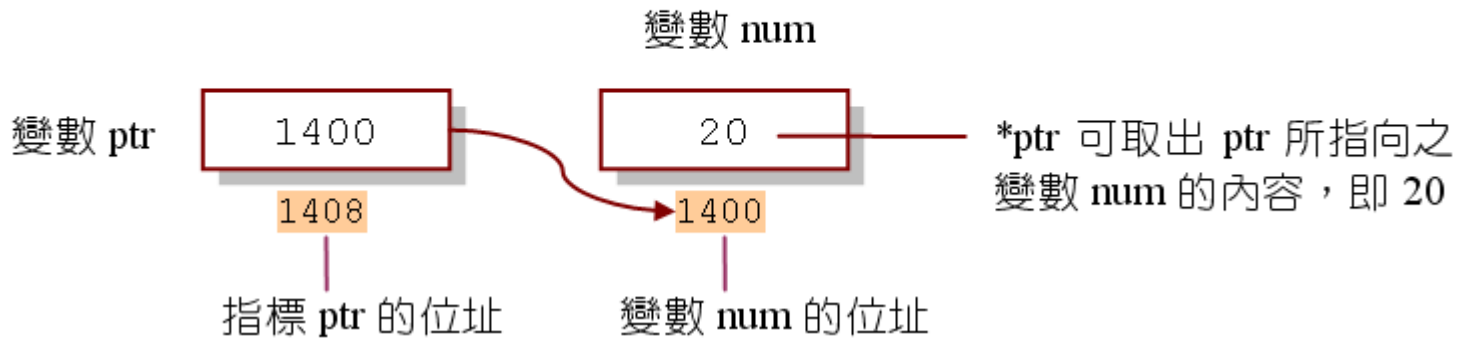
- 位址運算子「&」可用來取得變數的位址
  舉例來說，&num即代表取出num在記憶體中的位址

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

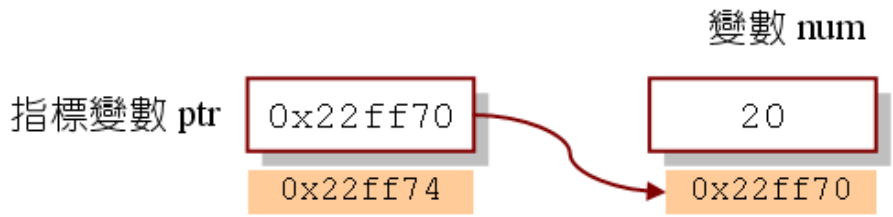
依址取值運算子 (3/3)

- 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

```
09      ptr=&num;                 // 將 num 的位址設給指標 ptr 存放
```



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

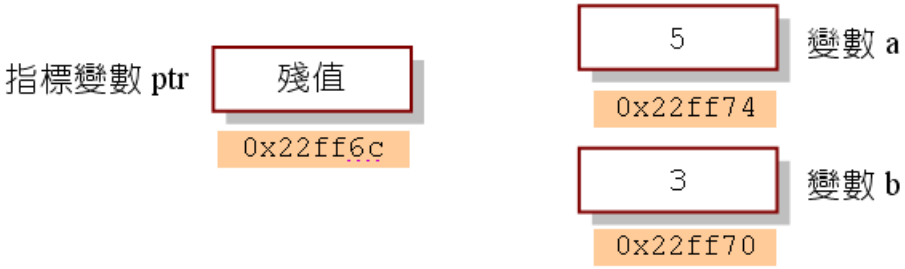
```
01  // prog9_3, 指標變數的使用
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  int main(void)
06  {
07     int a=5,b=3;
08     int *ptr;                          // 宣告指標變數 ptr
09
10     ptr=&a;                            // 將 a 的位址設給指標 ptr 存放
11     cout << "&a=" << &a << ", &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
-------------------------------------------*/
```

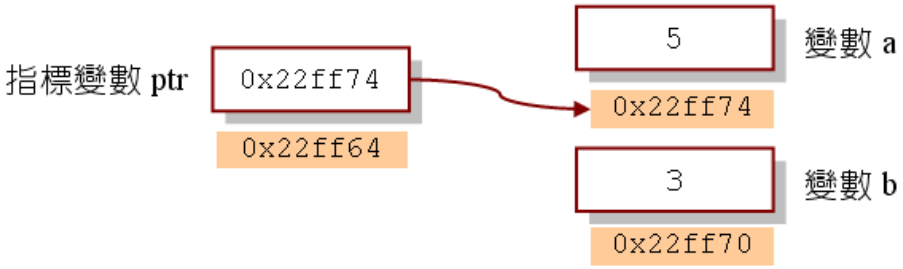
指標變數的使用 (2/2)

- 記憶體的配置情形

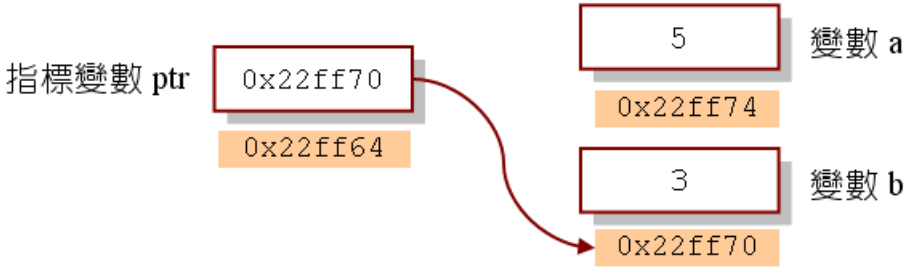
執行完第8行 `int *ptr;`



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
}
```

Ex:

```
int Sum(int *a, int *b){
        int sum = *a + *b;
        return sum;
}
```

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)

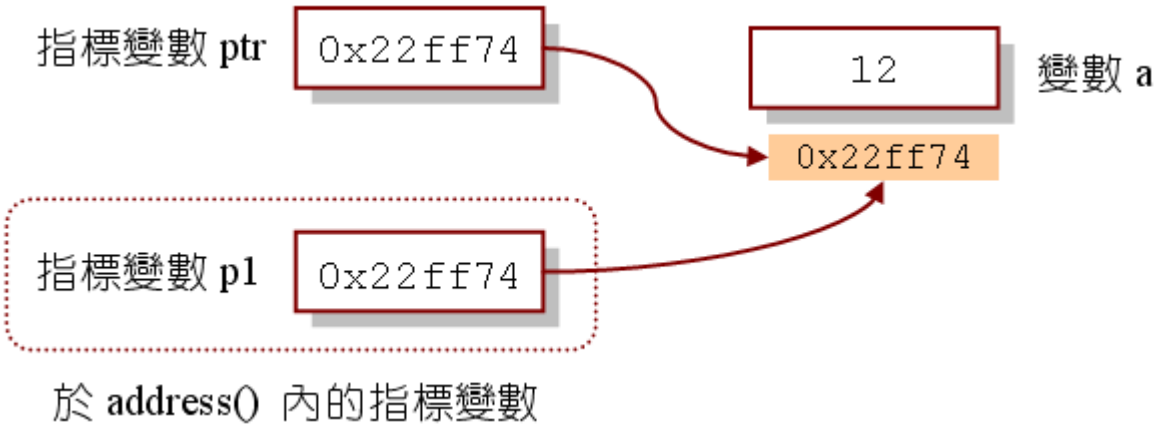
- 下面是函數address() 的完整範例

```
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

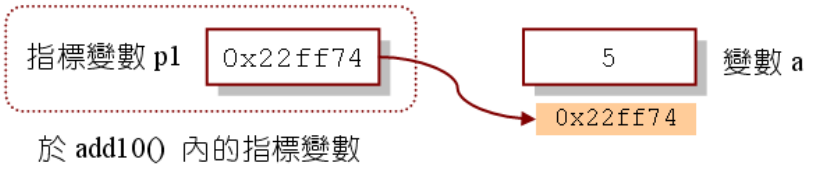


傳遞指標的應用

```
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  }
```

- 下面的程式是透過位址來改變呼叫端變數的內容

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



錯誤的示範 (1/4)

```
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  }
```

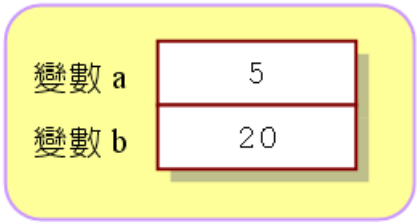
- 有些運算必須透過指標傳遞才能達成，下面為錯誤的示範

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

錯誤的示範 (2/4)

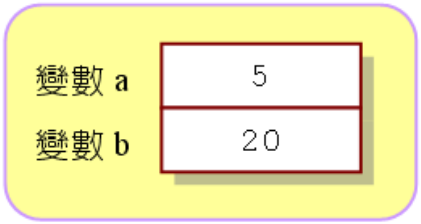
- 記憶體的配置情形

執行完第8行後，記憶體的配置情形

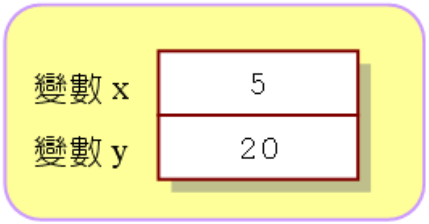


於主函數裡的變數

進入swap() 函數時，記憶體配置的情形



於主函數裡的變數

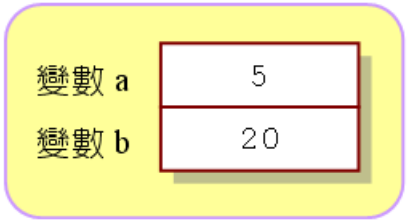


於 swap() 裡的變數

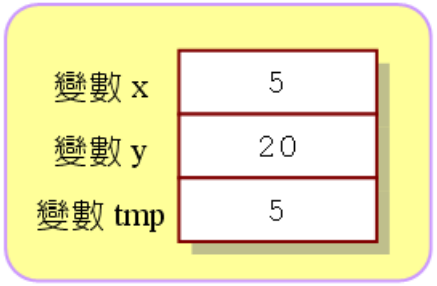
```
01  // prog9_9, 將a與b值...
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void swap(int,int);
06  int main(void)
07  {
08     int a=5,b=20;
09     cout << "交換前...;
10     swap(a,b);
11     cout << "交換後...;
12
13     system("pause");
14     return 0;
15  }
16  void swap(int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

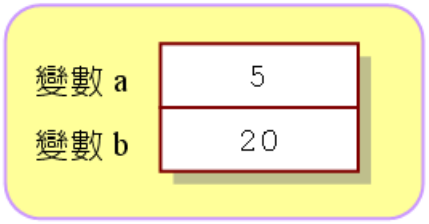


於主函數裡的變數

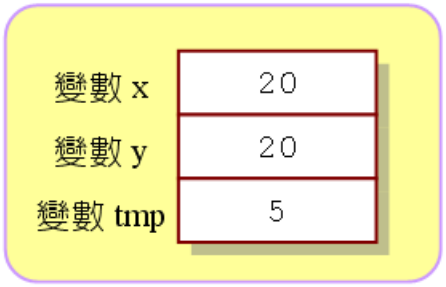


於 swap() 裡的變數

執行完第19行後，記憶體配置的情形



於主函數裡的變數

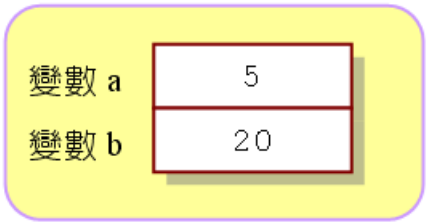


於 swap() 裡的變數

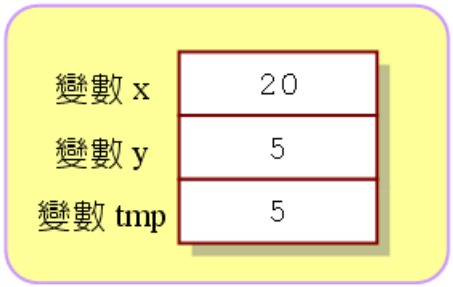
```
01  // prog9_9, 將a與b值...
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void swap(int,int);
06  int main(void)
07  {
08     int a=5,b=20;
09     cout << "交換前...;
10     swap(a,b);
11     cout << "交換後...;
12
13     system("pause");
14     return 0;
15  }
16  void swap(int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

當離開 swap 副程式後，於該副程式所更改的值屬於local variable，因此並不會影響主程式裡的變數

```
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)

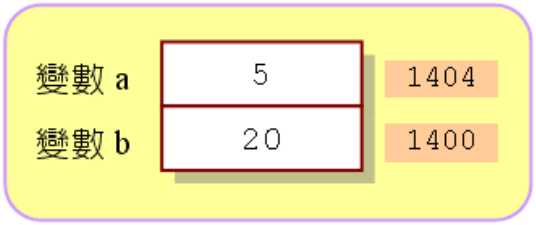
```
01  // prog9_10, 將 a 與 b 值互換(正確範例)
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void swap(int *,int *);          // 函數 swap()原型的宣告
06  int main(void)
07  {
08     int a=5,b=20;
09     cout << "交換前... a=" << a << ", b=" << b << endl;
10     swap(&a,&b);                  // 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

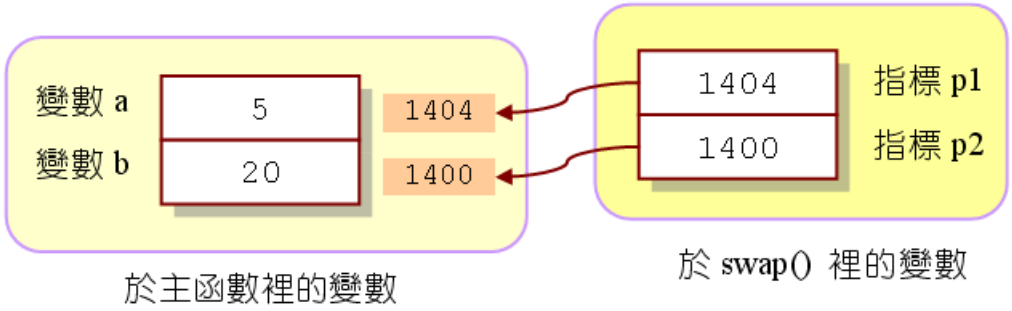
正確的範例 (2/4)

- 記憶體的配置情形

  執行完第8行後，記憶體配置的情形



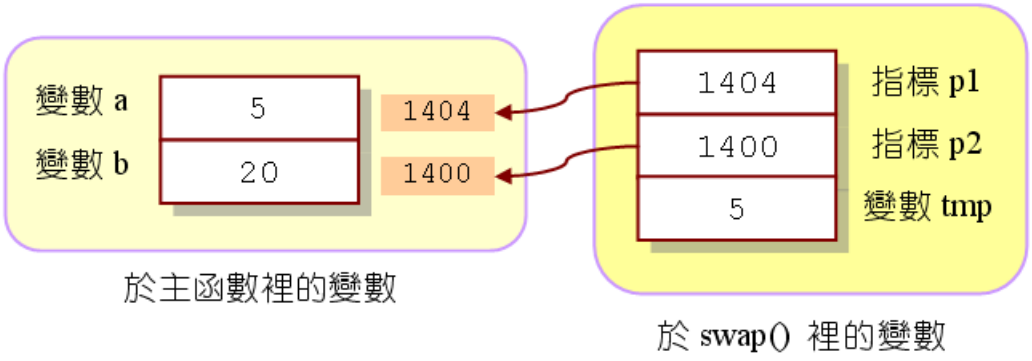
  進入swap() 函數時，記憶體配置的情形



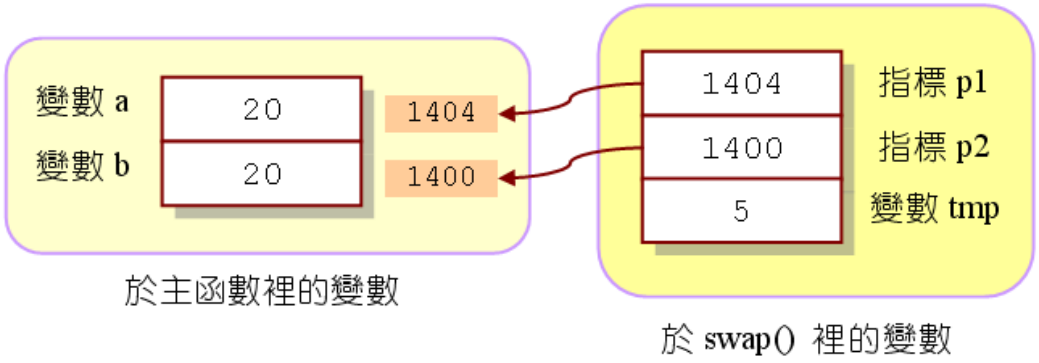

```
01  // prog9_10, 將 a 與 b 值互換
02  #include <iostream>
03  #include <cstdlib>
04  using namespace std;
05  void swap(int *,int *);
06  int main(void)
07  {
08     int a=5,b=20;
09     cout << "交換前...;
10     swap(&a,&b);
11     cout << "交換後...;
12
13     system("pause");
14     return 0;
15  }
16 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



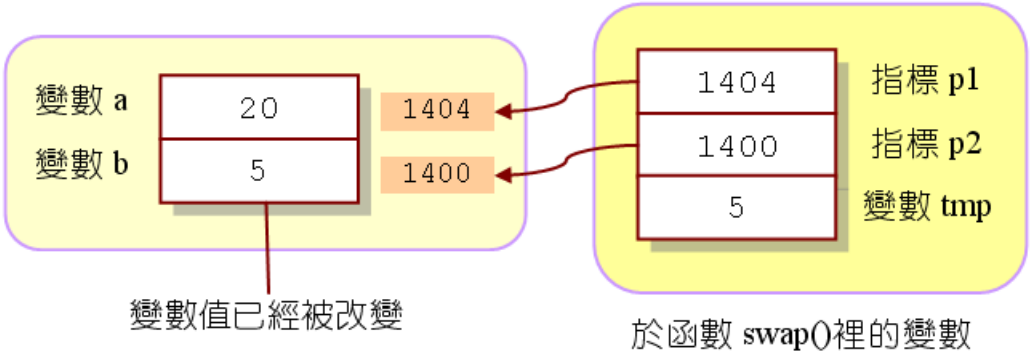
執行完第19行後，記憶體配置的情形




```
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



當離開 swap 副程式後，於該副程式所更改的值是指向主程式變數的指標，因此會影響主程式裡的變數

```
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
}
```

Ex:

```
int *Sum(int *a, int *b){
        *a = *a + *b;
        return a;
}
```

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  }
```

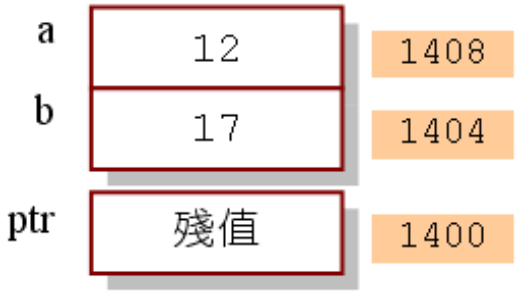
傳回 p1 與 p2 所指向之整數中，數值較大之整數的位址

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

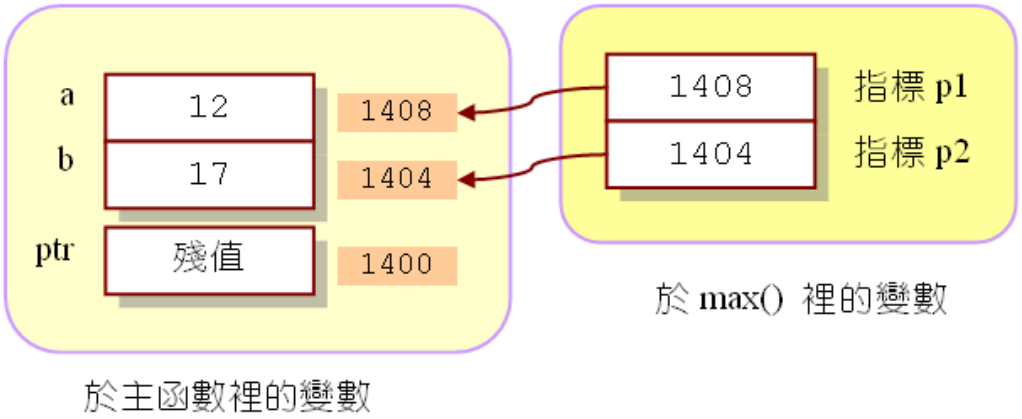
傳回值為指標的函數 (3/4)

- 記憶體的配置情形

  執行完第8行後，記憶體配置的情形



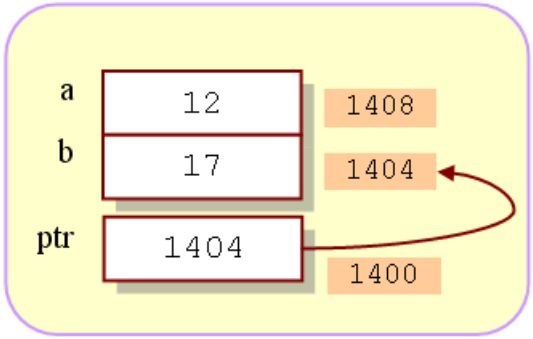
  進入max() 函數時，記憶體配置的情形




```
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  }
```

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
```


* 陣列A若不加上索引符號 [ ]，所代表的即為指向陣列第一個元素的指標

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

OUTLINE

1. 函數
2. 引入函式庫與常用函式庫
3. 指標
4. Practice

practice

1

- 求log

寫一個方便列印log的有用自訂函示

Hint :可使用函式庫 https://en.cppreference.com

- Function prototype:
  - void print_log(int num)

INPUT :

一個數b

OUTPUT:

求出底數為10 的 b 的對數 c

SAMPLE INPUT :

2

SAMPLE OUTPUT:

0.3010

2

- 試寫 power(x,n) 函數的多載，用來計算 x 的 n 次方，n 為 int 型態。當引數 x 型態為 int 時，函數回傳值的形態為 int；當引數 x 的形態為 double 時，函數回傳值的形態為 double。

INPUT :

一個浮點數 a、一個正整數 b 以及他們的指數 c

OUTPUT:

a^c

b^c

SAMPLE INPUT :

2.1 2 2

SAMPLE OUTPUT:

4. 41

2

```
輸入正整數:2
輸入浮點數:2.1
輸入指數:3
8
9.261
Process returned 0 (0x0)   execution time : 6.465 s
Press any key to continue.
```

3

- 建立一個陣列array[3]，其中包含3個random的整數(1~10)。利用一個函數輸入該陣列以及一個指標，在函數中尋找最大值並將最大值存入該指標。
- Function prototype:
  - void find_max(int array[3], int* max)

INPUT :

一個陣列

OUTPUT:

Max

SAMPLE INPUT :

1, 2, 3

SAMPLE OUTPUT:

Max: 3