

# 計算機程式設計

## C語言 Pointer II

郭忠義

[jykuo@ntut.edu.tw](mailto:jykuo@ntut.edu.tw)

臺北科技大學資訊工程系

# 指標陣列(array of pointers)

❑ 指標陣列：由指標組成的陣列

○ 宣告：`int *x[10];` // 一維陣列

○ 運用：`int var=5; x[2]=&var;`

○ 傳送給函數：利用陣列名稱傳送整個指標陣列

○ **trap 的 x**: 指向一維整數指標陣列的指標

```
int trap(int * x[ ]) {  
    return (*x[0] + *x[1])* (*x[2])/2;  
}  
  
void test() {  
    int *x[3];  
    int base1=3, base2=5, height=4, area=0;  
    x[0] = &base1;           // *(x+0) == x[0]  
    x[1] = &base2;           // *(x+1) == x[1]  
    x[2] = &height;  
    area=trap(x);  
    printf(" %d", area); // 4*(3+5)/2 = 16  
}
```

trap

值	變數名	記憶體位址 (假設)
100D	x	1031

test

值	變數名	記憶體位址 (假設)
3	base1	1001
5	base2	1005
4	height	1009
1001	x[0]	100D
1005	x[1]	1011
1009	x[2]	1015
100D	x	

# 指標陣列(array of pointers)

❑ `int *x[3][2];` // 二維陣列

○ 傳送給函數：利用陣列名稱傳送整個指標陣列

○ `x[0][0]`的type是 `int*`

○ `x[0]`的type是 `int**` 或 `int *[2]`

➢ 指向2個元素的一維陣列

○ `x`的type是 `int *[3][2]`

➢ 指向2維陣列的指標

此處記憶體空間須要有code配置

變數名	記憶體位址 (假設)	值 (記憶體位址)
	2001	
	2005	
	2009	
	200D	
	2011	

此處記憶體空間須要有code配置

變數名	記憶體位址 (假設)	值 (記憶體位址)
	2101	
	2105	
	2109	
	210D	
	2111	

變數名	記憶體位址 (假設)	值 (記憶體位址)
<code>x[0][0]</code>	1001	
<code>x[0][1]</code>	1005	
<code>x[1][0]</code>	1009	
<code>x[1][1]</code>	100D	
<code>x[2][0]</code>	1011	
<code>x[2][1]</code>	1015	

# 二維指標動態記憶體配置

```
#define ROW 2
#define COL 4
int main() {
    int i, j;
    int **ptr2 = NULL;
    ptr2 = (int**)malloc(sizeof(int*)*ROW);
    for(i=0; i<ROW; i++){
        ptr2[i] = (int*)malloc(sizeof(int)*COL);
    }
    for(i=0; i<ROW; i++){
        for(j=0; j<COL; j++) ptr2[i][j] = i*j;
    }
    // ptr2[0] == *(ptr2+0)
    // ptr2[0][0] == (*(ptr2+0)+0)
    // *(ptr2[1]+1) == ptr2[1][1]
    // *ptr2[1]+1 ==
    // *ptr2[1] = ptr2[1][0]
```

```
for(i=0; i<ROW; i++){
    for(j=0; j<COL; j++) {
        printf("%d*%d=%d\t", i, j, ptr2[i][j]);
    }
    printf("\n");
}

for(i=0; i<ROW; i++) free(ptr2[i]);
free(ptr2);
return 0;
}
```

# Exercise

## □ 指標++，寫下程式輸出

```
#include <stdio.h>
void print(int a[], int n) {
    int i=0;
    for (i=0; i<n; i++) {
        printf("(%x - %d)", &a[i], a[i]);
        if (i%3==2) printf("\n");
    }
}
void f1() {
    int i=0, a[]={6,5,4,3,2,1};
    int *p = a;
    *(p++) = 0;
    print(a, 6);
    printf("%x, %d\n", p, *p);
    *(++p) = -1;
    print(a, 6);
    printf("%x, %d\n", p, *p);
    a[0]=(*p++);
```

```
    print(a, 6);
    printf("%x, %d\n", p, *p);
}
int main() {
    f1();
    return 0;
}
```

```
(60fee0 - 0)(60fee4 - 5)(60fee8 - 4)
(60feec - 3)(60fef0 - 2)(60fef4 - 1)
60fee4, 5
(60fee0 - 0)(60fee4 - 5)(60fee8 - -1)
(60feec - 3)(60fef0 - 2)(60fef4 - 1)
60fee8, -1
(60fee0 - -1)(60fee4 - 5)(60fee8 - -1)
(60feec - 3)(60fef0 - 2)(60fef4 - 1)
60feec, 3
```

# Exercise

## □ 指標++，寫下程式輸出

```
#include <stdio.h>
#include <stdlib.h>
void test01(int a[]) {
    int *p=a;
    *(p++) = 99;
}
void test02(int a[]) {
    int *p=a;
    *(++p) = 99;
}
void print(int a[], int size) {
    int i=0;
    for (i=0; i<size; i++) {
        printf("%d,", a[i]);
    }
    printf("\n");
}
```

```
void test001(){
    int a[]={0,1,2,3,4,5};
    int b[]={0,1,2,3,4,5};
    test01(a);print(a,6);
    test02(b);print(b,6);
}
int main() {
    test001();
    return 0;
}
```

```
99,1,2,3,4,5,
0,99,2,3,4,5,
```

# Exercise

## □ 指標++，寫下程式輸出

```
#include <stdio.h>
void print(int a[], int size) {
    int i=0;
    for (i=0; i<size; i++) {
        printf("%d,", a[i]);
    }
    printf("\n");
}
void test002() {
    int a[]={5,4,3,2,1,0};
    int b[]={0, 0, 0};
    int *p;
    p = a;
    b[1] = (*(p++))++;
    printf("%d, %d\n", b[1], *p);
    print(a,6);
}
```

```
void test003() {
    int a[]={5,4,3,2,1,0}, b[]={0, 0, 0};
    int *p;
    p = a;
    b[1] = ++(*(p++));
    printf("%d, %d\n", b[1], *p);
    print(a,6);
}
int main() {
    test002();
    test003();
    return 0;
}
```

```
5, 4
6,4,3,2,1,0,
6, 4
6,4,3,2,1,0,
```

# Exercise

## □ 指標++，寫下程式輸出

```
#include <stdio.h>
void test004() {
    int a[]={5,4,3,2,1,0};
    int b[]={0, 0, 0};
    int k=0;
    int *p;
    p = a;
    k = *(p++);
    b[1] = (k)++;
    printf("%d, %d, %d\n", b[1], *p, k);
    print(a,6);
}
int main() {
    test004();
    return 0;
}
```

5, 4, 6  
5,4,3,2,1,0,



# 指向函式的指標

## ❑ 函式指標宣告

- 資料型別 (\*變數名稱)(參數資料型別 參數變數名稱, ...);
- `int (*fptr) (int );`
- `int (*p)(const char*, const char*);`

## ❑ 函式指標指向函式的位址

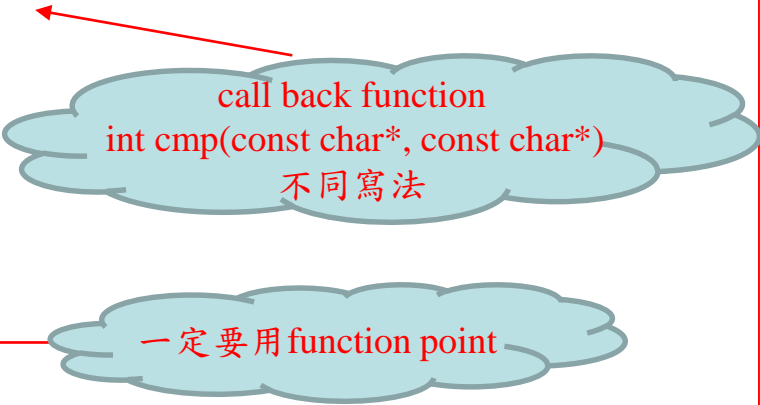
- 函式位址是函式入口點。
- 利用函式名稱獲得函式位址。(與陣列名稱同)

## ❑ 把函式當成參數傳給函式

- 函式的參數宣告成函式指標，把函式名稱當成參數傳入

# 指向函數的指標

```
#include <stdio.h>
#include <string.h>      // 宣告 int strcmp( const char *lhs, const char *rhs );
void check (char *a, char *b, int (*cmp)(const char*, const char*)) {
    if (!(*cmp) (a,b)) printf("equal");
    else printf("not equal");
}
void test() {
    char s1[80], s2[80];
    int (*p)(const char*, const char*);
    p = strcmp;
    gets(s1);
    gets(s2);
    check(s1, s2, p);
}
int main() {
    test();
    return 0;
}
```



call back function  
int cmp(const char\*, const char\*)  
不同寫法

一定要用function point

# 指向函數的指標

```
int square(int len){  
    return len*len;  
}  
int main() {  
    int (*ptr) (int);  
    ptr = square  
    int area = (*ptr)(20);  
    printf("area=%d", area);  
    return 0;  
}
```

輸出結果：  
area=400

# 指向函數的指標

## □ 計算面積(積分)

```
#include <stdio.h>
#include <math.h>
double square(double x) { return x * x;} /* 計算平方 */
double cube(double x) { return x * x * x;} /* 計算三次方 */
/* 計算f()在(x,y)間以n等份逼近的積分數值，使用梯形法 */
double integral(double (*f)(double), int n, double x, double y) {
    int i;
    double gap = (y - x) / n;
    double fy1 = (*f)(x);
    double fy2 = (*f)(x + gap);
    double area = 0;
    for (i = 0; i < n; i++) {
        area += (fy1 + fy2) * gap / 2; // 使用梯形面積公式
        fy1 = fy2;
        fy2 = (*f)(x + gap * (i + 1)); // 下底
    }
    return area;
}
```

# 指向函數的指標

## □ 計算面積(積分)

```
int main() {
    char fun[100];
    int n;
    double x, y;
    double (*f)(double); // f: a pointer to function(double) returning double
    while (scanf("%99s",fun) != EOF) {    // EOF定義於stdio.h內,一般為-1
        if (strcmp(fun,"square")==0) {    f = square;    }
        else if (strcmp(fun,"cube")==0) {    f = cube;    }
        else if (strcmp(fun,"sqrt")==0) {    f = sqrt;    } // sqrt is defined in math.h
        else if (strcmp(fun,"end")==0) {    break;    }
        else {
            printf("Unknown function\n");
            continue;
        }
        scanf("%d%lf%lf", &n, &x, &y);
        printf("Integral of %s from %lf to %lf is: %lf\n", fun, x, y, integral(f, n, x, y));
    }
    return 0;
}
```

# 指向函數的指標

## □ 計算面積(積分)

○ 讓積分算得更快，迴圈內的Code越簡單越好

```
double integral(double (*f)(double), int n, double x, double y) {  
    int i;  
    double area = ((*f)(x) + (*f)(y)) / 2.0L;  
    double gap = (y - x) / n;  
    double next = x;  
    for (i = 1; i < n; i++) {  
        area += (*f)(next += gap);  
    }  
    return area * gap;  
}
```

# Homework I

## □ 計算面積(積分)

- 使用切割面積加總法公式， $T = (h/2)(f(p) + f(q) + 2 f(x_i))$ ， $h = (q-p)/n$ ，求解 $f(x)$ ， $x$ 值從區間起始 $p$ 到區間終點 $q$ 的面積， $n$ 為切割數。
- $f1(x) = \sqrt{(a-x^2)}$ ， $a$  為常數。
- $f2(x) = (ax^3 + 7x)/\sqrt{(a+x)}$ ， $a$  為常數。
- `double area(double (*f)(double x), double a, double p, double q, int n)`，計算 $f(x)$  從 $p$ 到 $q$ 切成 $n$ 塊的面積，即計算面積加總公式 $T$ 值。
- 增加 $n$ 切割數(切割數 $n$ 每次 $*2$ )，精確到小數第 $err$ 位。輸入1求 $f1$ ，2求 $f2$ ，程式要讓使用者重複輸入，直到輸入-1 結束程式。其他顯示錯誤訊息

# Homework I

## □ 計算面積(積分)

○ 1: f1 2:f2 0:exit

○ >1

○ Input a, p, q, err:4 -2 1 9

○ answer=5.054815608319

○ 1: f1 2:f2 0:exit  $\sqrt{(a+x)}$

○ >2

○ Input a, p, q, err:1 0 3 9

○ answer=29.752380952687

○ 1: f1 2:f2 0:exit

○ >0

○ 注意: 答案需輸出到小數點第12位



# 陣列指標(pointers to array)

## □ 陣列指標，指向一個陣列的指標

- 可以作為傳多維陣列的引數
- C語言把多維陣列看成是由低一維的陣列所組成的一維陣列
- 接收n 維陣列的函數的引數，可宣告成指向n-1維陣列的指標

➤ `int x[ ][10][20]; <=> int (*x)[10][20];`

➤ `int x[ ][10]; <=> int (*x)[10];`

➤ `int x[ ]; <=> int *x;`

➤ (X) `int x[ ][ ][20]`

➤ (X) `int (x*)[ ][20]`

➤ `int (*x)[10]; <> int *x[10];`

```
#include <stdio.h>
void trap(int (*x)[3]) { // int x[][3]
    int i=0, j=0;
    for (i=0; i<3; i++) {
        for (j=0; j<3; j++) printf("%d ", x[i][j]);
        printf("\n");
    }
}
int main() {
    int x[3][3]={ {1,2,3}, {4,5,6}, {7,8,9} };
    trap(x);
}
```

# 陣列指標(pointers to array)

## □ 陣列指標，指向一個陣列的指標

- `int (*x)[3][2];`
- `x` 的 type 是 `int(*)[3][2]`
- `*x` 的 type 是 `int[3][2]` 或 `int(*)[2]`
- `(*x)[0]` 的 type 是 `int*` 或 `int[2]`
- `(*x)[0][0]` 的 type 是 `int`

`x`

<code>*(x+0)</code>	<code>(*x)[0]</code> <code>(*x)[0][0]</code> <code>(*x)[0][1]</code>	<code>(*x)[1]</code> <code>(*x)[1][0]</code> <code>(*x)[1][1]</code>	<code>(*x)[2]</code> <code>(*x)[2][0]</code> <code>(*x)[2][1]</code>
<code>*(x+1)</code>	<code>(*x+1)[0]</code> <code>(*x+1)[0][0]</code> <code>(*x+1)[0][1]</code>	<code>(*x+1)[1]</code> <code>(*x+1)[1][0]</code> <code>(*x+1)[1][1]</code>	<code>(*x+1)[2]</code> <code>(*x+1)[2][0]</code> <code>(*x+1)[2][1]</code>
<code>*(x+2)</code>	<code>(*x+2)[0]</code> <code>(*x+2)[0][0]</code> <code>(*x+2)[0][1]</code>	<code>(*x+2)[1]</code> <code>(*x+2)[1][0]</code> <code>(*x+2)[1][1]</code>	<code>(*x+2)[2]</code> <code>(*x+2)[2][0]</code> <code>(*x+2)[2][1]</code>

永遠只有一維陣列  
一個\*換一組[]

# 陣列指標(pointers to array)

```
void test() {  
    // a是一個陣列，有兩個元素，每個元素是一個陣列  
    int a[2][2] = {55,66,77,88};  
    // a[0]是一個陣列，有兩個元素，每個元素是一個整數  
    int b[2]={11,22};  
    //p是指標，指向一個陣列(的記憶體)，該陣列有兩元素，P 是指標的指標  
    int (*p)[2];  
    //p指向一個陣列 b  
    p=&b;  
    printf("(00)=>%d\n",**p);    //印出 b[0], *(p+*(p+0)), *p = b, **p=*(b+0)=b[0]  
    printf("(01)=>%d\n",*(*p+1)); // 印出 b[1], *(b+1) = b[1]  
    printf("(02)=>%d\n",(*p)[0]); // 印出 b[0]  
    printf("(03)=>%d\n",(*p)[1]); // 印出 b[1]  
    printf("(04)=>%d\n", p[0][0]); // 印出 b[0]  
    printf("(05)=>%d\n", p[0][1]); // 印出 b[1]  
    printf("(06)=>%d\n",*(p[0])); // 印出 b[0]  
    printf("(07)=>%d\n\n",*(p[0]+1)); // 印出 b[1]  
    printf("(08)=>%d\n",***(p+1)); // 錯誤指向，超出範圍  
    printf("(09)=>%d\n",p[1][0]); // 錯誤指向，超出範圍  
    printf("(0a)=>%d\n\n",*(p[1])); // 錯誤指向，超出範圍
```

# 陣列指標(pointers to array)

## □ 陣列指標

```
p=&a[0];  
// a[0]是一個陣列，每個陣列有兩個元素，每個元素是整數  
// p指向一個陣列 a[0]  
printf("(10)=>%d\n",**p);    // 印出 a[0][0]  
printf("(11)=>%d\n",*(p+1)); // 印出 a[0][1]  
printf("(12)=>%d\n",(*p)[0]); // 印出 a[0][0]  
printf("(13)=>%d\n",(*p)[1]); // 印出 a[0][1]  
printf("(14)=>%d\n", p[0][0]); // 印出 a[0][0]  
printf("(15)=>%d\n", p[0][1]); // 印出 a[0][1]  
printf("(16)=>%d\n",*(p[0])); // 印出 a[0][0]  
printf("(17)=>%d\n\n",*(p[0]+1)); // 印出 a[0][1]  
  
printf("(18)=>%d\n", **(p+1)); // 錯誤指向，超出範圍a[0][0]  
printf("(19)=>%d\n", p[1][0]); // 錯誤指向，超出範圍a[0][0]  
printf("(1a)=>%d\n\n", *(p[1])); // 錯誤指向，超出範圍a[0][0]
```

# 陣列指標(pointers to array)

```
p=a;                // p指向一個陣列 a
printf("(20)=>%d\n",**p);    // 印出 a[0][0]
printf("(21)=>%d\n",*(p+1)); // 印出 a[0][1]
printf("(22)=>%d\n",(*p)[0]); // 印出 a[0][0]
printf("(23)=>%d\n",(*p)[1]); // 印出 a[0][1]
printf("(24)=>%d\n", p[0][0]); // 印出 a[0][0]
printf("(25)=>%d\n", p[0][1]); // 印出 a[0][1]
printf("(26)=>%d\n",*(p[0]));  // 印出 a[0][0]
printf("(27)=>%d\n\n",*(p[0]+1)); // 印出 a[0][1]

printf("(30)=>%d\n",**(p+1)); // 印出 a[1][0], *(p+1) ==p[1]=a[1]
printf("(31)=>%d\n",*(*(p+1)+1)); // 印出 a[1][1]
printf("(32)=>%d\n",(* (p+1))[0]); // 印出 a[1][0]
printf("(33)=>%d\n",(* (p+1))[1]); // 印出 a[1][1]
printf("(34)=>%d\n", p[1][0]); // 印出 a[1][0]
printf("(35)=>%d\n", p[1][1]); // 印出 a[1][1]
printf("(36)=>%d\n",*(p[1]));  // 印出 a[1][0]
printf("(37)=>%d\n\n",*(p[1]+1)); // 印出 a[1][1]
}
```

# 指標型別

- `()[]` 是第一優先權左結合，`*` 是第二優先權右結合
  - `[]` — array[] of
  - `*` — pointer to
  - 變數後面的 `()` — function() returning

```
char *x;      // x: a pointer to char
char x[3];    // x: an array[3] of char
char x();     // x: a function() returning char
char *x[3];   // x: an array[3] of pointer to char
char (*x)[3]; // x: a pointer to array[3] of char
char **x;     // x: a pointer to pointer to char
char *x();    // x: a function() returning pointer to char
char *x()[3]; // x: a function() returning array[3] of pointer to char
char (*x[])(); // x: an array[] of pointer to function() returning char
char (*x())(); // x: a function() returning pointer to function() returning char
char (*(x[]))(int, int); // x: a pointer to array[] of pointer to function(int,int) returning char
```

# 指標型別

## □ 表達式的資料型別辨識原則

### ○ 觀察表達式前面的資料型別

```
char *x;      // x: a pointer to char
*x : a char
char *x[3];   // x: an array[3] of pointer to char
x[0] : a pointer to char
char **x;     // x: a pointer to pointer to char
*x : a pointer to char
char *x();    // x: a function() returning pointer to char
x() : a pointer to char
char *x()[3]; // x: a function() returning array[3] of pointer to char
x()[1] : a pointer to char
```

# Exercise

- 表達式的資料型別辨識原則
  - 觀察表達式前面的資料型別

```
char **argv;  
int (*daytab)[13];  
int *daytab[13];  
void *comp();  
void (*comp)();  
char ((*x())[])();  
char ((*x[3])())[5];
```



# 不定長參數的函數

□ 處理不定長參數要用<stdarg.h>。

○ va\_list 資料型別存取每個參數

```
#include <stdarg.h>
// or #include <sys/varargs.h>
void minprintf(char *fmt, ...) {
    va_list ap; /* pointer to each unnamed arg in turn */
    char *p, *sval;
    int ival;
    double dval;
    va_start(ap, fmt); /* make ap point to 1st unnamed arg */
    for (p = fmt; *p; p++) { // check each character
        if (*p != '%') { // 不是特殊字元,直接輸出即可
            putchar(*p);
            continue;
        }
    }
}
```

# 不定長參數的函數

```
switch(*++p) { // 檢查%的下一個字元是甚麼
    case 'd':
        ival = va_arg(ap, int);
        printf("%d", ival);
        break;
    case 'f':
        dval = va_arg(ap, double);
        printf("%f", dval);
        break;
    case 's':
        for (sval = va_arg(ap, char *); *sval; sval++) { // 印出sval所指到的所有字元
            putchar(*sval);
        }
        break;
    default :
        putchar(*p);
        break;
}
va_end(ap); // clean up when done */
}
```