

資料結構



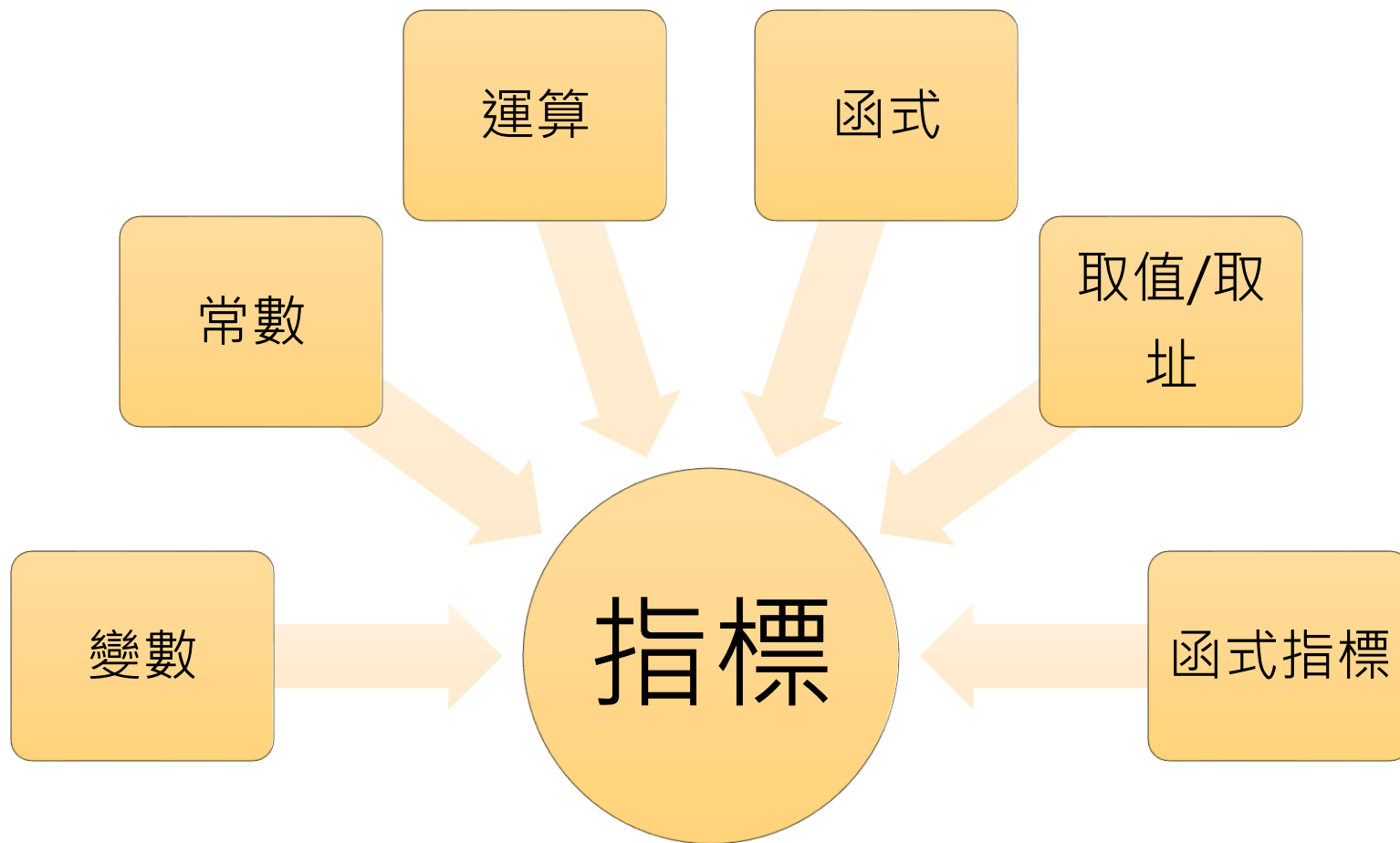
演算法

指標與陣列  
Pointer & Array

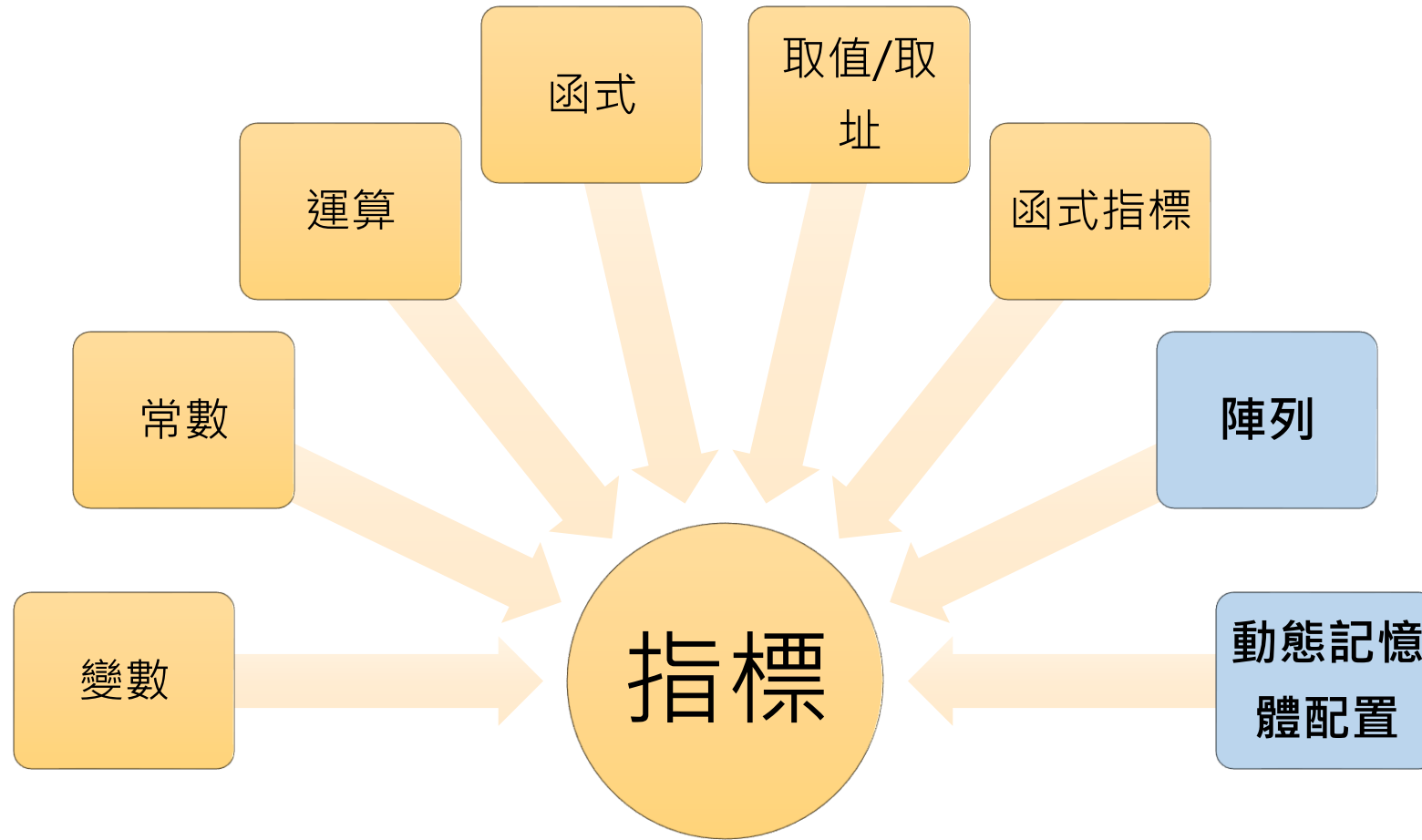
指標與陣列

[十八豆教育科技](#)

# 先複習一下指標相關概念



# 即將加入的



# 指標與 一維陣列



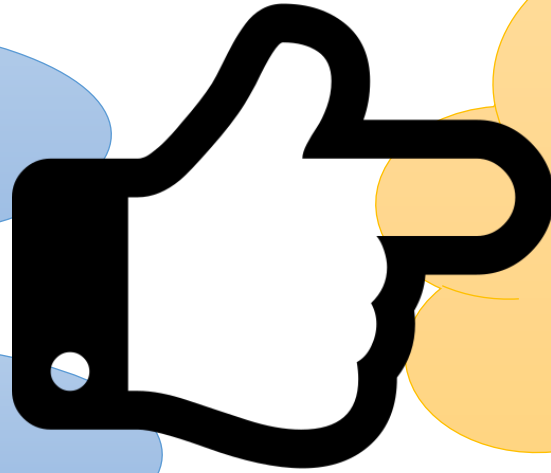
# 指向陣列的指標

```
int a[10]  
int *ptr
```

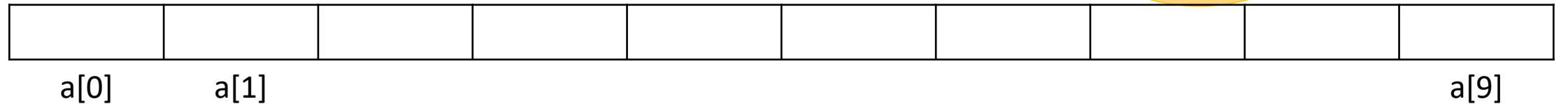
ptr

ptr = &a[0]

ptr = a

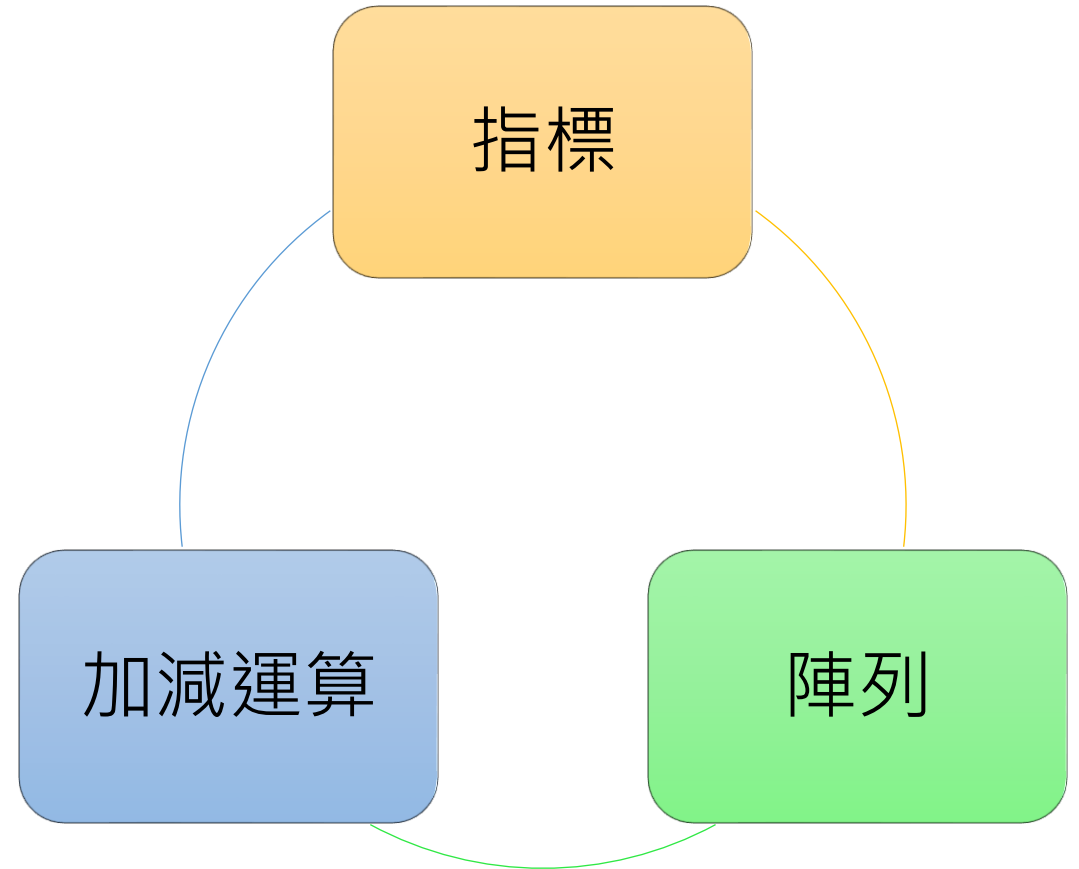


陣列名稱相當於是指向陣列第一個元素的指標常數（值無法更改），因此直接設定指標 `ptr = a` 或是設定 `ptr = &a[0]` 都可以將指標指向陣列。



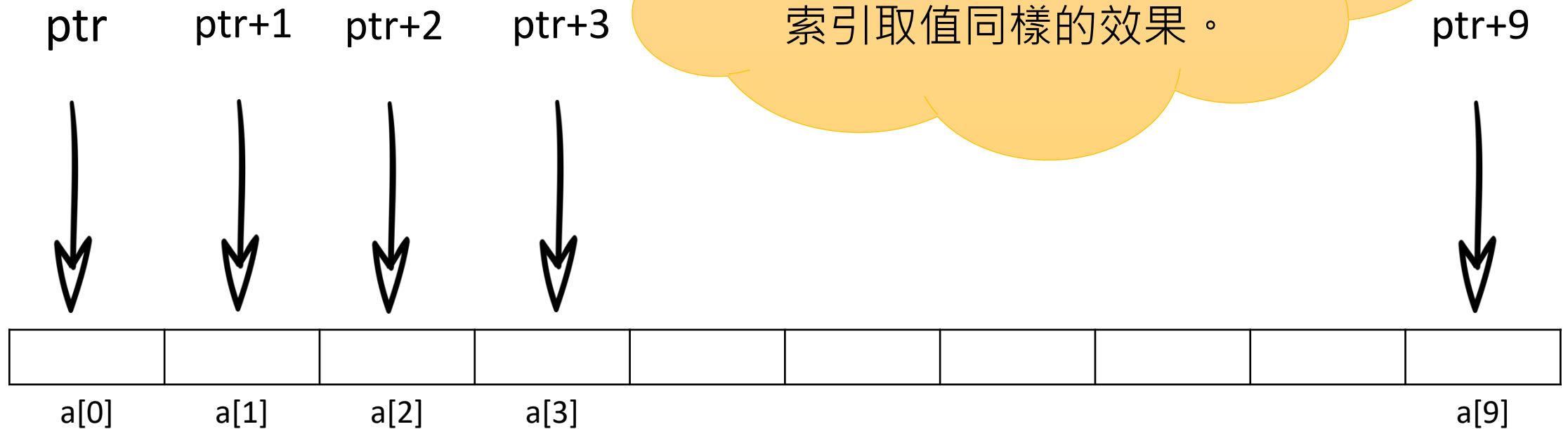
# 指標的運算

- 前面有提過，單純指標的加減法運算是沒有意義的！
- 只有指向陣列的指標進行加減運算才有意義！
  - 陣列在記憶體中的配置是連續的空間，因此可透過指標運算取得前後陣列元素位址。



# 指標的運算

透過指標的加減運算，可以將指標指向陣列內不同索引位置，再搭配取值(\*)與取址(&)運算達到與陣列索引取值同樣的效果。



# 陣列取址

```
int a[10]  
int *ptr = a
```

| 陣列 a | 使用陣列索引取址 | 使用指標取址 |
|------|----------|--------|
| 0    | &a[0]    | ptr    |
| 1    | &a[1]    | ptr+1  |
| 2    | &a[2]    | ptr+2  |
| 3    | &a[3]    | ptr+3  |
| 4    | &a[4]    | ptr+4  |
| 5    | &a[5]    | ptr+5  |
| 6    | &a[6]    | ptr+6  |
| 7    | &a[7]    | ptr+7  |
| 8    | &a[8]    | ptr+8  |
| 9    | &a[9]    | ptr+9  |



# 陣列取值

```
int a[10]
int *ptr = a
```

| 陣列 a | 使用陣列索引取值 | 使用指標取值   |
|------|----------|----------|
| 0    | a[0]     | *ptr     |
| 1    | a[1]     | *(ptr+1) |
| 2    | a[2]     | *(ptr+2) |
| 3    | a[3]     | *(ptr+3) |
| 4    | a[4]     | *(ptr+4) |
| 5    | a[5]     | *(ptr+5) |
| 6    | a[6]     | *(ptr+6) |
| 7    | a[7]     | *(ptr+7) |
| 8    | a[8]     | *(ptr+8) |
| 9    | a[9]     | *(ptr+9) |

# 用指標逐一檢視陣列

## 指標加法

```
#include <stdio.h>
int main()
{
    int size = 5, i;
    int list[size];
    int *ptr = list;
    for (i = 0; i < size; i++)
    {
        list[i] = i * 5;
        printf("list[%d]=%d\n", i, *(ptr+i));
    }
    return 0;
}
```

## 指標遞增

```
#include <stdio.h>
int main()
{
    int size = 5, i;
    int list[size];
    int *ptr = list;
    for (i = 0; i < size; i++)
    {
        list[i] = i * 5;
        printf("list[%d]=%d\n", i, *ptr++);
    }
    return 0;
}
```

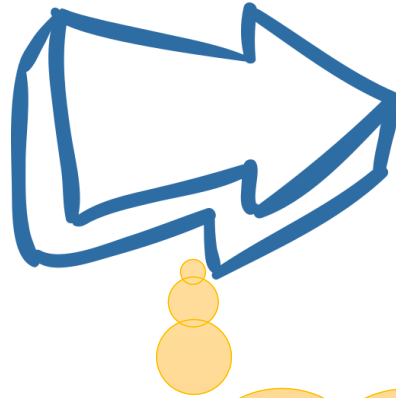
18dice

# 指標與 多維陣列



# 多維陣列記憶體位置

| <b>a[3][3]</b> | <b>0</b> | <b>1</b> | <b>2</b> |
|----------------|----------|----------|----------|
| <b>0</b>       | a[0][0]  | a[0][1]  | a[0][2]  |
| <b>1</b>       | a[1][0]  | a[1][1]  | a[1][2]  |
| <b>2</b>       | a[1][0]  | a[1][1]  | a[1][2]  |



從最小維度開始，  
由小到大對應到  
記憶體位址

|             | 陣列索引    | 記憶體位址      |
|-------------|---------|------------|
| <b>a[0]</b> | a[0][0] | 0x0022FDD0 |
|             | a[0][1] | 0x0022FDD4 |
|             | a[0][2] | 0x0022FDD8 |
| <b>a[1]</b> | a[1][0] | 0x0022FDDC |
|             | a[1][1] | 0x0022FDE0 |
|             | a[1][2] | 0x0022FDE4 |
| <b>a[2]</b> | a[2][0] | 0x0022FDE8 |
|             | a[2][1] | 0x0022FDEC |
|             | a[2][2] | 0x0022FDE0 |

# 指標存取多維陣列

陣列變數[列數][行數]

元素在第幾列  
(第幾行從 0  
開始計算)

元素在第幾行  
(第幾行從 0 開  
始計算)

$\text{ptr} + (\text{第幾列} * \text{一列有幾行}) + \text{第幾行}$

陣列變數宣告時的  
「行數」值  
(從 1 開頭)

# 多維陣列取址

```
int a[3][3]
int *ptr = a
```

## 使用陣列索引取址

| a[3][3] | 0        | 1        | 2        |
|---------|----------|----------|----------|
| 0       | &a[0][0] | &a[0][1] | &a[0][2] |
| 1       | &a[1][0] | &a[1][1] | &a[1][2] |
| 2       | &a[1][0] | &a[1][1] | &a[1][2] |

| a[3][3] | 0       | 1       | 2       |
|---------|---------|---------|---------|
| 0       | a[0][0] | a[0][1] | a[0][2] |
| 1       | a[1][0] | a[1][1] | a[1][2] |
| 2       | a[1][0] | a[1][1] | a[1][2] |

## 使用指標取址

| a[3][3] | 0           | 1           | 2           |
|---------|-------------|-------------|-------------|
| 0       | ptr+(0*3)+0 | ptr+(0*3)+1 | ptr+(0*3)+2 |
| 1       | ptr+(1*3)+0 | ptr+(1*3)+1 | ptr+(2*3)+2 |
| 2       | ptr+(2*3)+0 | ptr+(2*3)+1 | ptr+(2*3)+2 |

# 多維陣列取址

```
int a[3][3]
int *ptr = a
```

## 使用陣列索引取址

| a[3][3] | 0        | 1        | 2        |
|---------|----------|----------|----------|
| 0       | &a[0][0] | &a[0][1] | &a[0][2] |
| 1       | &a[1][0] | &a[1][1] | &a[1][2] |
| 2       | &a[1][0] | &a[1][1] | &a[1][2] |

## 使用指標取址

| a[3][3] | 0           | 1           | 2           |
|---------|-------------|-------------|-------------|
| 0       | ptr+(0*3)+0 | ptr+(0*3)+1 | ptr+(0*3)+2 |
| 1       | ptr+(1*3)+0 | ptr+(1*3)+1 | ptr+(2*3)+2 |
| 2       | ptr+(2*3)+0 | ptr+(2*3)+1 | ptr+(2*3)+2 |

# 多維陣列取值

```
int a[3][3]
int *ptr = a
```

使用陣列索引取值

| a[3][3] | 0       | 1       | 2       |
|---------|---------|---------|---------|
| 0       | a[0][0] | a[0][1] | a[0][2] |
| 1       | a[1][0] | a[1][1] | a[1][2] |
| 2       | a[1][0] | a[1][1] | a[1][2] |

使用指標取值

| a[3][3] | 0              | 1              | 2              |
|---------|----------------|----------------|----------------|
| 0       | *(ptr+(0*3)+0) | *(ptr+(0*3)+1) | *(ptr+(0*3)+2) |
| 1       | *(ptr+(1*3)+0) | *(ptr+(1*3)+1) | *(ptr+(2*3)+2) |
| 2       | *(ptr+(2*3)+0) | *(ptr+(2*3)+1) | *(ptr+(2*3)+2) |



# 用指標操作多維陣列

```
#include <stdio.h>
int main()
{
    int size = 3, i, j;
    int list[size][size];
    int *ptr = list, *qtr;
    for (i = 0; i < size; i++) {
        qtr = list[i];
        for (j = 0; j < size; j++) {
            list[i][j] = i * size + j ;
            printf("[%d][%d]\n", i, j);
            printf("ptr(%p): %d\n", ptr+(i*size)+j, *(ptr+(i*size)+j));
            printf("qtr(%p): %d\n", qtr+j, *(qtr+j));
        }
    }
    return 0;
}
```



# 指標與函式



# 傳遞一維陣列給函式

```
#include <stdio.h>
int show(int *list, int size) {
    int i = 0;
    for (i = 0; i < size; i++) {
        printf("list[%d](%p): %d\n", i, list+i, *(list+i));
    }
    return size;
}
int main()
{
    int size = 3, i;
    int list[size];
    for (i = 0; i < size; i++) {
        list[i] = i;
        printf("list[%d](%p): %d\n", i, &list[i], list[i]);
    }
    printf("-----\n");
    show(list, size);
    return 0;
}
```

陣列名稱等同於常數指標，因此函式接收陣列參數時，可直接使用指標變數接收。

執行結果

```
dice - pointerANDarray $ ./04.exe
list[0](000000000022FDF0): 0
list[1](000000000022FDF4): 1
list[2](000000000022FDF8): 2
-----
list[0](000000000022FDF0): 0
list[1](000000000022FDF4): 1
list[2](000000000022FDF8): 2
```

18dice

18dice

# 傳遞二維陣列給函式

二維陣列的參數接收就需要使用雙重指標接收。  
雙重指標的宣告有兩種：

- (1) `int **list`
- (2) `int *list[]`

執行結果

```
#include <stdio.h>
int show(int **list, int size)
{
    int i, j;
    int *ptr = list;
    for (i = 0; i < size; i++) {
        for (j = 0; j < size; j++) {
            printf("list[%d][%d](%p): %d\n", i, j, ptr+(i*size)+j, *(ptr+(i*size)+j));
        }
    }
    return size;
}
int main()
{
    int size = 3, i, j;
    int list[size][size];
    for (i = 0; i < size; i++) {
        for (j = 0; j < size; j++) {
            list[i][j] = i * size + j ;
            printf("[%d][%d](%p): %d\n", i, j, &list[i][j], list[i][j]);
        }
    }
    printf("-----\n");
    show(list, size);
    return 0;
}
```

```
pointerANDarray $ ./05.exe
[0][0](00000000022FDD0): 0
[0][1](00000000022FDD4): 1
[0][2](00000000022FDD8): 2
[1][0](00000000022FDDC): 3
[1][1](00000000022FDE0): 4
[1][2](00000000022FDE4): 5
[2][0](00000000022FDE8): 6
[2][1](00000000022FDEC): 7
[2][2](00000000022FDF0): 8
-----
list[0][0](00000000022FDD0): 0
list[0][1](00000000022FDD4): 1
list[0][2](00000000022FDD8): 2
list[1][0](00000000022FDDC): 3
list[1][1](00000000022FDE0): 4
list[1][2](00000000022FDE4): 5
list[2][0](00000000022FDE8): 6
list[2][1](00000000022FDEC): 7
list[2][2](00000000022FDF0): 8
```





延伸的概念

# 概念1: 命令列參數

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    int i;
    for (i = 0; i < argc; i++)
    {
        printf("第 %d 個參數為: %s\n", i,
argv[i]);
    }
    return 0;
}
```



程式執行時，加上  
參數資料，提供程  
式使用。

執行結果

```
dice - pointerANDarray $ ./06.exe 測試 第?個
第 0 個參數為: D:\18d\D18-資結演算-V1\題目\pointerA
NDarray\06.exe
第 1 個參數為: 測試
第 2 個參數為: 第?個
```



# 概念1: 命令列參數

- 在 C 語言的主執行函式 `main()` 內加上 `int argc` 與 `char *argv[]` 兩個參數，便可在程式內使用程式執行時傳入的參數，

```
int main(int argc, char *argv[])
```

argc 代表傳入的參數個數



argv 儲存參數字串

# 概念1: 命令列參數

```
./a.out 10 20 第三個 測試
```

argc = 5

argv

|         |         |
|---------|---------|
| argv[0] | ./a.out |
|---------|---------|

|         |    |
|---------|----|
| argv[1] | 10 |
|---------|----|

|         |    |
|---------|----|
| argv[2] | 20 |
|---------|----|

|         |     |
|---------|-----|
| argv[3] | 第三個 |
|---------|-----|

|         |    |
|---------|----|
| argv[4] | 測試 |
|---------|----|



執行的程式名稱會是第一個參數，因此 argc 最少也會是 1，不會有 argc = 0 的情況！



# 概念2: 動態記憶體配置

## 指標單元提過... 「沒有指標時的困境(2)」

```
#include <stdio.h>
int main()
{
    int num = 10;
    int num2 = 0;
    int data[num];
    int i = 0;
    scanf("%d", &num2);
    printf("num2: %d\n", num2);
    for (i = 0; i < num; i++)
    {
        data[i] = i;
        printf("data[%d]: %d\n", i, data[i]);
    }
    return 0;
}
```

如果陣列 data 的大小要由 num2 決定呢？

num2 的值會在陣列宣告後才取得，要怎麼才能之後再動態決定陣列大小呢？



# 概念2: 動態記憶體配置

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int *list;
    int size, i;
    scanf("%d", &size);
    list = malloc(size * sizeof(int));
    for (i = 0; i < size; i++)
    {
        *(list + i) = i;
        printf("list[%d]: %d\n", i, *(list+i));
    }
    return 0;
}
```

18dice

使用 **malloc()** 動態  
配置記憶體空間

執行結果

```
dice - pointerANDarray $ ./08.exe
5
list[0]: 0
list[1]: 1
list[2]: 2
list[3]: 3
list[4]: 4
```

18dice

## 概念2: 動態記憶體配置: malloc()

- 陣列採用固定的記憶體配置，宣告陣列變數時就會立即提供符合需求的記憶體空間，且此空間是無法變動的！
- 當所需的空間還是未知數時，就需使用動態記憶體配置: malloc()

malloc(大小 \* sizeof(型態))

空間大小



資料型態  
如: sizeof(int)

## 概念2: 動態記憶體配置: malloc()

- malloc() 配置後會回傳配置的記憶體位址，因此需要使用指標變數接收。
- 指標變數接受 malloc() 回傳的記憶體位址後，後續的資料操作就與使用指標操作陣列相同。

```
int *data;  
data = malloc(10 * sizeof(int))
```

```
int list[10];  
int *data = list;
```



# 概念2: 動態記憶體配置: free()

**malloc()** 配置的動態配置記憶體空間可使用 **free()** 將空間歸還。

歸還後位址內存放的值會被清除，此空間也會交由系統重新分配。

因此務必要小心程式不可再使用已經 **free()** 處理後的變數資料！

## 執行結果

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int *list;
    int size, i;
    scanf("%d", &size);
    list = malloc(size * sizeof(int));
    for (i = 0; i < size; i++) {
        *(list + i) = i;
        printf("list[%d](%p): %d\n", i, (list+i), *(list+i));
    }
    free(list);
    printf("-----\n");
    for (i = 0; i < size; i++) {
        printf("list[%d](%p): %d\n", i, (list+i), *(list+i));
    }
    return 0;
}
```

```
dice - pointerANDarray $ ./09.exe
3
list[0](0000000000457EC0): 0
list[1](0000000000457EC4): 1
list[2](0000000000457EC8): 2
-----
list[0](0000000000457EC0): 3504096
list[1](0000000000457EC4): 0
list[2](0000000000457EC8): 3473752
```

18dice

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int *list1, *list2;
    int size1, size2, add, i;
    scanf("%d", &size1);
    list1 = malloc(size1 * sizeof(int));
    for (i = 0; i < size1; i++) {
        *(list1 + i) = i;
        printf("list1[%d](%p): %d\n", i, (list1+i), *(list1+i));
    }
    scanf("%d", &add); // 設定空間要擴大多少
    size2 = size1 + add;
    // 重新配置新空間
    list2 = realloc(list1, size2 * sizeof(int));
    // 只設定新擴增的元素內容值
    for (i = 0; i < add; i++) {
        *(list2 + size1 + i) = 10 + i;
    }
    // 全部顯示
    for (i = 0; i < size2; i++) {
        printf("list2[%d](%p): %d\n", i, (list2+i), *(list2+i));
    }
    return 0;
}
```

malloc() 配置過的空間，  
若需要調整空間大小，可  
使用 realloc() 重新配置

執行結果

```
dice - pointerANDarray $ ./10.exe
2
list1[0](0000000000317EC0): 0
list1[1](0000000000317EC4): 1
3
list2[0](0000000000317EC0): 0
list2[1](0000000000317EC4): 1
list2[2](0000000000317EC8): 10
list2[3](0000000000317ECC): 11
list2[4](0000000000317ED0): 12
```



## 概念2: 動態記憶體配置: realloc()

- realloc() 可重新調整配置的記憶體空間，重新調整後的位址與原位址不一定會相同！
- 若原配置的空間仍有足夠的位置，realloc() 就會使用 malloc() 時得到的位址，若新需求空間遠大於原配置空間，就會變更到新位址。

## 概念2: 動態記憶體配置

使用動態記憶體配置相關函式，需引入 `stdlib.h` 這個 header

```
#include <stdlib.h>
```

```
malloc(需求大小)
```

```
realloc(要重新配置的指標, 新需求大小)
```

```
free(指標)
```





## 重點整理

- 指標與一維陣列
- 指標與多維陣列
- 指標在陣列內的取值與取址
- 指標與函式參數的傳遞接收
- 命令列位址
- 動態記憶體配置

