前置處理器－#define 和#include

一般的 C 語言中，程式指令是要給機器看然後來執行的，所以會需要經過「編譯」這個動作。而由#開頭的程式碼並不會編譯給機器執行，而是在編譯的過程中給編譯器看的，我們稱他們爲「前置處理器」。

先說我們平常用的#include，爲什麼每次我們都要先加那兩行呢？其實 stdio.h 和 stdlib.h 是兩個「函數庫」，也就是已經放了很多函數進去裡面所以可以直接使用那些函數。像 printf()和 scanf 這些基本的函數，就是被定義在 stdio.h 裡面的！至於 stdlib.h 則是我們寫讓他暫停的那行－system(“Pause”);system()所定義的地方。所以如果你沒有 include 好的話，程式執行上就會有一些問題。(不幸的是 DevC++有一點天兵，你就算沒寫它也不會怎樣，但實際上是去好一點的編譯器編譯是會怎樣的。)

再來是新的東西－#define(巨集指令)，他的格式如下：

```
#define  識別名稱  代換標記
```

記得最後面不用加分號。

所謂的識別名稱就是替換內容的縮寫，一般來說較喜歡用大寫標示較易辨認；代換標記的部份可以是常數、字串或是函數。下面舉些簡單的合法例子：

```
#define PI 3.14
#define SAY “CKEFGISC”
#define POWER(X) (X)*(X)*(X)
```

這些都是合法的定義。來看用這些寫成的例子：

```
#include<stdio.h>
#include<stdlib.h>
#define PI 3.14
#define SAY "CKEFGISC"
#define POWER(X) (X)*(X)*(X)
int main(){
	printf("圓半徑是 2，圓面積爲%f\n",2*2*PI);
	printf("We are %s\n",SAY);
	printf("2 的三次方=%d\n",POWER(2));

	system("Pause");
	return 0;
}
```

```
圓半徑是2，圓面積爲12.560000
We are CKEFGISC
2的三次方=8
請按任意鍵繼續 . . .
```

第三個定義的巨集其實就是一個簡單的小函數，不過要注意的是我們在後面每一個 X 都加上括號，爲什麼呢？來看一下有沒有加括號的差別：

```
#include<stdio.h>
#include<stdlib.h>
#define POWER1(X) X*X*X
#define POWER2(X) (X)*(X)*(X)
int main(){
	int i=2;
	printf("[POWER1(%d)=%d]",i,POWER1(i));
	printf("[POWER2(%d)=%d]\n",i,POWER2(i));

	printf("[POWER1(%d)=%d]",i+1,POWER1(i+1));
	printf("[POWER2(%d)=%d]\n",i+1,POWER2(i+1));

	system("Pause");
	return 0; }
```

```
[POWER1(2)=8][POWER2(2)=8]
[POWER1(3)=7][POWER2(3)=27]
請按任意鍵繼續 . . .
```

有沒有發現了？沒有加括號的，在處理 i+1 的時候產生了錯誤。

因爲對他來說那只是一個代換，等於說現在我把 i+1 丟進去的時候，沒加括號的那一個會變成：i+1*i+1*i+1，而數學上又是先乘除後加減，所以就會變成 7。也是因爲這樣所以最好在定義巨集時要注意到這個問題，妥當運用括號。

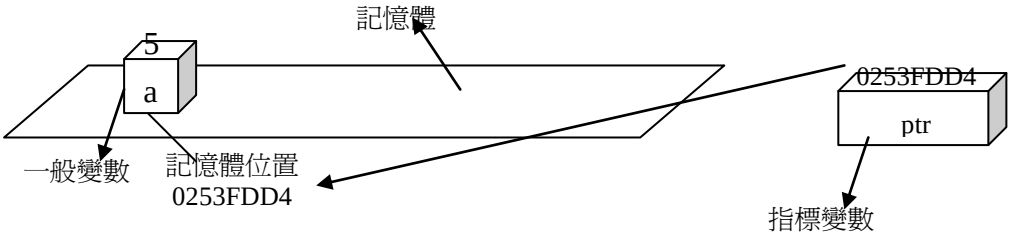
指標

指標，對於程式的學習是一個很大的關鍵，大型的程式、真正開始運用的許多東西都得運用到指標，而指標的概念說簡單，也不簡單，說難，也沒有那麼難(笑)。那麼，我們就正式來介紹一下指標吧！

指標(Pointer)其實是一種特殊的變數，用來存放的是***變數在記憶體中的位置***。我們在宣告變數的時候，記憶體上會產生一塊足夠的空間給這個變數；那麼我們要怎麼知道這個變數在哪？當然就是利用記憶體位址啦！位址，就好像我們的住家住址的感覺一樣，他是獨一無二。系統中也可以依位置來存取變數，就像郵差可以依住址來送信一樣。

利用指標變數，我們可以把變數在記憶體內的位址存入指標中，當我們需要用到這個變數時，便可以利用指標先找到該變數的位址，再由該位址取出位址內所儲存的變數值。這種方式我們稱之爲「***間接定址取值法***」。

畫個簡單的圖示來說：



等於說我們可以用指標變數裡存放的記憶體位置，找到他所指向的那個一般變數裡的數值。

指標變數的宣告

指標的變數所存放的是某個資料在記憶體中的位址，不是像數值、文字等資料內容，所以有特定的宣告方式：

資料型態 *指標變數名稱;

最重要的就是多了*這個符號，即是指標符號，這些變數有了指標符號就會變成指標變數，像是：

```
int *ptri;        /*宣告一個名爲 ptri 的整數型態指標變數*/
char *ptrch;    /*宣告一個名爲 ptrch 的字元型態指標變數*/
```

只要是 C 語言的資料型態，通通可以宣告成指標。

指標變數的使用

使用指標變數時，不是存取放在指標裡的位置，就是用指向位置內的那個資

料，這兩樣工作需要經由下列兩種指標運算子來完成：

1. ***位址運算子&***：用來求取變數的位置，像我們平常宣告變數 a，&a 就是他的記憶體位址。
2. ***依址取值運算子****：用來取得指標指向的位置內所放的資料。假設變數 a 的值是 100，有個指標 ptra 指向 a，*ptra 就是 a 的值 100。

像上面的例子，要怎樣把指標和變數連結起來呢？這時候就是要用位址運算子把記憶體位置存到指標內去了，如這樣的敘述：prta=&a;

化成程式來看看：

```
#include<stdio.h>
#include<stdlib.h>
int main(void){
	int a=100;
	int *ptra;
	ptra=&a;
	printf("a=%d,address of a=%p\n",a,&a);
	printf("*ptra=%d,ptra=%p\n",*ptra,ptra);
	system("Pause");
	return 0;
}
```

```
a=100,address of a=0022FF74
*ptra=100,ptra=0022FF74
請按任意鍵繼續 . . .
```

//剛剛沒有提到的是記憶體位置輸出的格式是%p。

ptra=&a 將 a 的記憶體位置放入 ptra，(&a 是 a 的記憶體位置)，*ptra 則是將 ptra 裡面指向的位置裡面的值取出。

剛剛所提到的兩個運算子，要注意的是因爲&是取址，所以像是&100 或是&(i++)都是不合法的！而*(依址取值運算子)跟在一開始宣告時的意義(指標符號)是有不同的意思唷！不要把兩個搞混了。

另外要注意的是，***指標指向的變數型態要跟他自己的型態一致***，不然並無法正確印出其內容。

指標的運算

C 語言中指標提供了三種運算：設定運算、加減法運算、差值運算。陷在來介紹一下：

1. ***設定運算***

   其實跟變數的指派很像，也一樣是使用了=(這裡稱爲設定運算子)。

   像 ptra=&a;這樣就是設定。而我們也可以假設宣告了兩個指標變數－ptra、ptrb;並且設定 ptra=&a;

   當我們寫 ptrb=ptra;時，ptrb 所指向的也是 a。而如果我們使用了*ptrb=*ptra 會發生什麼事呢？

   程式會先找到兩個的記憶體位置，並且將 ptrb 所指向的變數 b 裡面換成 ptra 所指向的變數 a 的值。

小重點：因爲指標的使用非常靈活，而且是直接控制到了記憶體，不小心可能會造成不可預期的情況發生，所以在宣告之後最好能馬上指向正確的變數，如果不行我們可以先將其設成 NULL。

Ex:ptr=NULL;。

把剛剛所敘述的化爲程式碼：

```
#include<stdio.h>
#include<stdlib.h>
int main(void){
	int a=100,b=200;
	int *ptra,*ptrb;
	ptra=&a;
	ptrb=&b;
	printf("一開始的 a=%d,b=%d\n",a,b);
	*ptrb=*ptra;
	printf("現在的 a=%d,b=%d\n",a,b);
	system("Pause");
	return 0;
}
```

```
一開始的a=100,b=200
現在的a=100,b=100
請按任意鍵繼續 . . .
```

有沒有注意到，因爲*ptrb=*ptra;的這行，b 的值改變了！注意這個改變是真的改變，原本 b 的值就被代換掉了唷！

2. ***指標的加法與減法運算***

在指標中我們一樣可以使用++跟--這兩個運算子，可是要記得它跟你在變數用的有不太一樣的意義－改變的是指向的記憶體位置。

比如說現在有個整數指標 ptra 指向整數變數 a 的記憶體位置(假設是 100)，如果執行 ptra++，那麼原本的 100 就會變成 104(爲什麼是 4！？因爲整數在記憶體中佔了 4 個位址長度，所以往後移一格便是 104。)

舉個例子來看：

```
#include<stdio.h>
#include<stdlib.h>
int main(void){
	int a=100,b=200;
	int *ptra,*ptrb;
	ptra=&a;
	ptrb=&b;
	printf("一開始的*ptra(%p)=%d,*ptrb(%p)=%d\n",ptra,*ptra,ptrb,*ptrb);
	ptra++;
	ptrb--;
	printf("現在的*ptra(%p)=%d,*ptrb(%p)=%d\n",ptra,*ptra,ptrb,*ptrb);
	system("Pause");
	return 0;
}
```

```
一開始的*ptra(0022FF74)=100,*ptrb(0022FF70)=200
現在的*ptra(0022FF78)=2293680,*ptrb(0022FF6C)=2293624
請按任意鍵繼續 . . .
```

有發現嗎？記憶體位置改變之後，因爲並沒有剛好變成對方的記憶體位置，兩個指標變數指向的值就既非 a 也非 b 了。

要注意如果在上面的例子我們想要把 b 的數值加上 1，那我們並不能寫「*ptrb++;」，而是得寫「*ptrb=*ptrb+1;」，前者的意思是去找 b 的下一格記憶體的數值而非把它指向的那格記憶體內的數字加上 1。

3. ***指標的差值運算***

C 語言中指標並不允許直接做加法運算，可是卻允許兩個相同資料型態的指標做差值運算，算出來的意義是什麼？答案是：在記憶體中兩個之間的距離，也就是相差的資料個數。我們來看下面這個例子：

```
#include<stdio.h>
#include<stdlib.h>
int main(void){
	int a=100,b=200;
	int *ptra,*ptrb;
	ptra=&a;
	ptrb=&b;
	printf("*ptra(%p)=%d,*ptrb(%p)=%d\n",ptra,*ptra,ptrb,*ptrb);
	printf("ptra-ptrb=%d\n",ptra-ptrb);
	system("Pause");
	return 0;
}
```

```
*ptra(0022FF74)=100,*ptrb(0022FF70)=200
ptra-ptrb=1
請按任意鍵繼續 . . .
```

差值運算的時候會自動把記憶體位置相減再除以該指標資料型態的記憶體位址長度，所運行的結果可以發現 a,b 之間差一格整數型態記憶體。

指標與函數

之前提過函數可以有傳回值，但是有沒有發現，傳進去的東西可以很多，傳出來卻只能有一個！這種時候我們就可以運用指標來幫我們解決問題囉～

把指標傳入函數，函數宣告時應該要長這樣

資料型態 函數名稱(指標變數型態 *名稱 1….)

來看個例子吧。

```
#include<stdio.h>
#include<stdlib.h>
void fun(int *ptra,int *ptrb){
	int i=*ptrb;
	*ptrb=*ptra+*ptrb;
	*ptra=*ptrb-*ptra;
	return;
}
void list(int *ptra,int *ptrb){
	printf("*ptra(%p)=%d\n",ptra,*ptra);
	printf("*ptrb(%p)=%d\n",ptrb,*ptrb);
	return;
}
int main(void){
	int a=30,b=20;
	int *ptra,*ptrb;
	ptra=&a;
	ptrb=&b;
	list(ptra,ptrb);
	fun(ptra,ptrb);
	printf("處理後....\n");
	list(ptra,ptrb);
	system("Pause");
	return 0;
}
```

```
*ptra(0022FF74)=30
*ptrb(0022FF70)=20
處理後....
*ptra(0022FF74)=20
*ptrb(0022FF70)=50
請按任意鍵繼續 . . .
```

看到了嗎？list()這個函數的主要功用是印出 ptra 和 ptrb 所指向的位址以及其所含的值，而在執行了 fun()之後，我們把 ptra 跟 ptrb 兩個所指向的位址裡面的內容(也就是 a 和 b)都改變了，有沒有很神奇！活用可以很靈活，這就是指標爲什麼強大的原因之一了。

如果現在我們要寫一個函數把兩個變數裡的數值交換呢？這個時候指標也是好用的啦！先來看如果是本來不用指標，會變成……

```
#include<stdio.h>
#include<stdlib.h>
void swap(int a,int b){
	int temp;
	temp=a;
	a=b;
	b=temp;
	return;
}
int main(void){
	int a=3,b=5;
	printf("執行前：a=%d,b=%d\n",a,b);
	swap(a,b);
	printf("執行後：a=%d,b=%d\n",a,b);
	system("Pause");
	return 0;
}
```

```
執行前：a=3,b=5
執行後：a=3,b=5
請按任意鍵繼續 . . .
```

有沒有發現根本就沒差？用指標應該要這樣寫：

```
#include<stdio.h>
#include<stdlib.h>
void swap(int *x,int *y){
	int temp=*x;
	*x=*y;
	*y=temp;
	return;
}
int main(void){
	int a=3,b=5;
	printf("執行前：a=%d,b=%d\n",a,b);
	swap(&a,&b);
	printf("執行後：a=%d,b=%d\n",a,b);
	system("Pause");
	return 0;
}
```

```
執行前：a=3,b=5
執行後：a=5,b=3
請按任意鍵繼續 . . .
```

記得要傳入的是變數的位置，要用到先前提過的&運算子唷！

指標與陣列

陣列，其實我們可以把它當成是一種指標！不過，陣列是固定長度的記憶體區塊，指標則是一個變數。我們來看看他們之間的關係吧：

```
#include<stdio.h>
#include<stdlib.h>
int main(void){
	int a[3]={3,4,5};
	int i;
	for(i=0;i<3;i++)
		printf("a[%d]=%d,&a[%d]=%p,address of a=%p\n",i,*(a+i),i,&a[i],a+i);
	system("Pause");
	return 0;
}
```

```
a[0]=3,&a[0]=0022FF60,address of a=0022FF60
a[1]=4,&a[1]=0022FF64,address of a=0022FF64
a[2]=5,&a[2]=0022FF68,address of a=0022FF68
請按任意鍵繼續 . . .
```

雖然上面的程式碼有點混亂，可是有沒有發現，我直接寫 a+i(其實就第一次就是 a，第二次是 a+1…)，跟陣列第 i 格的記憶體位置是一樣的——陣列的名稱，其

實在某個角度來說，就是一種指標。而基本上預設的陣列名稱，就是指向陣列第一格的，如果是陣列 a，a 一開始就是指向 a[0]，如果我們寫了 a++，那麼 a 現在就會指向 a[1]。

現在假設我們要用一個函數來對一個陣列來做處理，欸～有一點感覺嗎？沒錯！又是要用指標啦，就是把整個陣列的指標傳進去，然後利用函數去對指標指向的陣列位置作處理，來看個例子吧：(此例子爲找出陣列中的最大值)

```
#include<stdio.h>
#include<stdlib.h>
#define SIZE 5
int *maximum(int *m){
	int i,*max;
	max=m;
	for(i=1;i<SIZE;i++){
		if(*max<*(m+i))
			max=m+i;
	}
	return max;

}
int main(void){
	int a[SIZE]={3,1,7,8,6};
	int i,*ptr;

	printf("array a=");
	for(i=0;i<SIZE;i++)
		printf("%d ",a[i]);
	ptr=maximum(a);
	printf("\nmaximum=%d\n",*ptr);
	system("Pause");
	return 0;
}
```

```
array a=3 1 7 8 6
maximum=8
請按任意鍵繼續 . . .
```

再上面的程式中我們宣告了一個傳回值爲指標的函數 maximum()，記得在呼叫這個函數時，並不需要在前面加上*符號。

字串陣列與指標陣列

一般如果我們宣告了字串陣列，那麼就會有每一個字串都配給一樣的記憶體空間的情況，假使最長的字串有 15 個字元，那麼就算其他的字串都只有 3 個字元，還是會配給到多餘的空間。幾筆資料可能不明顯，如果是數千萬筆，那個空間可就不一樣了！這時候如果使用指標陣列，就可以避免掉這個情況。指標陣列的宣告方式如下：

資料型態 *陣列名稱[個數];

記得陣列名稱前面那個*指標符號很重要。這樣宣告的意義其實就是宣告很多的指向陣列一開始的那格記憶體位置，所以不需要用到兩個中括號[]。

指標的部份大概就談到這了！其實他的運用以及和函數的搭配還有很多，只是現階段可能不太會用到。但是指標真的很重要，大家要盡量把觀念弄懂，對於「指向的概念」能夠越清楚越好唷～