第八章

指標與動態記憶體

前言

- 在上一章的函式呼叫時，我們曾經介紹一種特殊的資料存取方法－『指標』。
- 指標與記憶體位址息息相關，並且是C語言的一大特色。
- 由於C語言允許程式設計師透過指標存取資料，因此，可以進行更低階的記憶體存取動作。
- 但程式設計師在使用指標時必須特別小心，否則將會造成無法預期的後果。

大綱

- 8.1 指標與記憶體位址
  - 8.1.1 存取記憶體內容
  - 8.1.2 指標變數
  - 8.1.3 宣告指標變數
  - 8.1.4 取址運算子與指標運算子
  - 8.1.5 指標變數的大小
- 8.2 指標運算
  - 8.2.1 指標的指定運算
  - 8.2.2 指標變數的加減運算
  - 8.2.3 指標變數的比較運算
  - 8.2.4 指標變數的差值運算
- 8.3 函式的傳指標呼叫

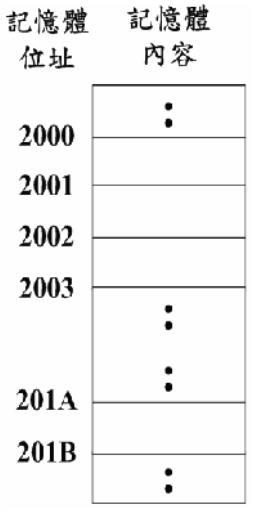
大綱

- 8.4『指標』、『陣列』、『字串』的關係
  - 8.4.1　指標與陣列
  - 8.4.2　指標與字串
- 8.5 指標函式回傳值
- 8.6『指標』的『指標』
- 8.7 動態記憶體配置
  - 8.7.1　配置記憶體函式－malloc( )
  - 8.7.2　釋放記憶體函式－free( )
- 8.8 本章回顧

8.1 指標與記憶體位址

8.1.1 存取記憶體內容

- 所有要被中央處理器處理的資料，都必須先存放在記憶體中。
- 這些記憶體被劃分為一個個的小單位，並且賦予每一個單位一個位址，這個動作稱之為『記憶體空間的定址』。
- 當記憶體被定址之後，作業系統或應用程式才能夠控制記憶體的內容。
- 如圖所示，由於每一個記憶體空間單位都被配置了一個虛擬的記憶體位址，透過這些記憶體位址，我們就可以取得記憶體的內容。

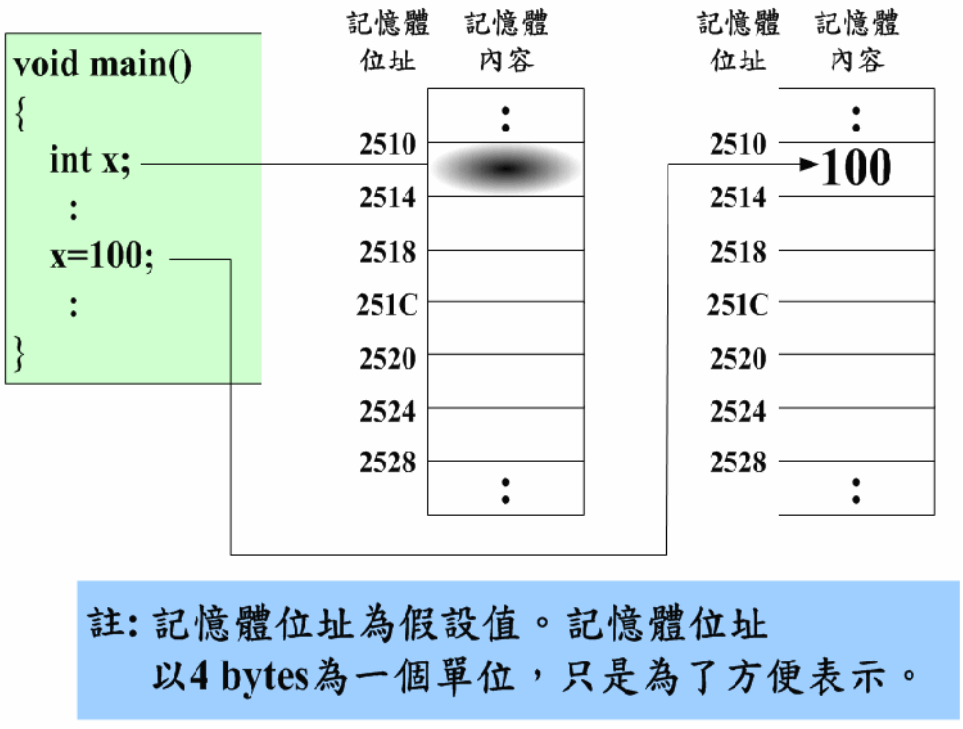


記憶體位址

8.1.1 存取記憶體內容

- 假設我們宣告變數x為整數型態，系統將空間來存放x的內容，而它所在的記憶體位址為（2510~2513），當我們透過敘述『x=100;』修改x變數的內容時，實際上在記憶體位址2510~2513的內容也就變成了100。

- 換句話說，普通變數佔用的記憶單位內存放的是該變數的數值。



修改變數值的記憶體內容變化

8.1.1 存取記憶體內容

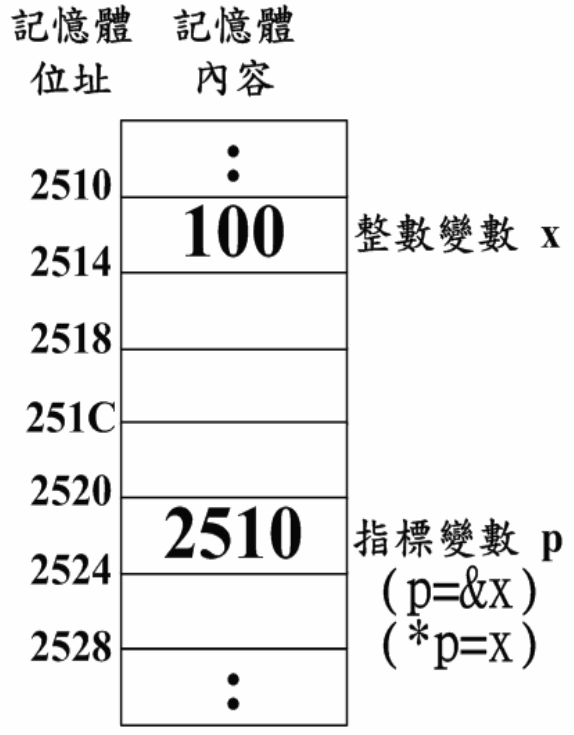
- 如果我們在上述範例中，再加入『x=x+1;』敘述，則執行該敘述時，2510~2513位址的資料將被取出，放到CPU中執行『+1』的運算，得到結果『101』，然後再回存到位址2510~2513之中。

- 換句話說，屆時2510~2513位址的內容將會是『101』。

- 如果我們用之前的取址運算子&取出x的變數位址時（也就是『&x;』），將會得到0x2510。

8.1.2 指標變數

- 普通變數意味著佔用某一塊記憶體空間，該空間內則存放變數資料。
- 『指標變數』與普通變數差不多，只不過指標變數的內容，是另一個變數的記憶體位址，換句話說，在記憶體空間內存放的是另一個變數所佔用的記憶體位址（如圖所示）。
- 在圖中，指標變數p的內容為整數變數x的記憶體位址，
- C語言允許指標變數內容為記憶體的任何位址，並且可以透過提領運算子『*』來改變指標所指向記憶體位址的內容。



指標變數的內容

8.1.2 指標變數

- 由於上述指標特性，因此使得C語言可以做到更為低階的記憶體處理行為，這通常是其他高階語言無法做到的。

- 不過也因為指標的功能強大，且指標觀念較難理解以及不易發現錯誤，因此若指標使用不當（例如恰好指向作業系統佔用的記憶體區塊），將可能會導致不夠穩定的作業系統當機（如DOS作業系統）或被作業系統拒絕執行（如Linux作業系統將產生Segment Fault）。

8.1.3 宣告指標變數

- 就和普通變數一樣，在使用指標變數之前，我們要先『宣告指標變數』，宣告指標的語法格式如下：
  - ANIS C語言指標的宣告語法：

```
資料型態 *指標變數名稱;
```

  - C++（大多數C++編譯器支援）新增的指標宣告語法：

```
資料型態* 指標變數名稱;
```

  - 【語法說明】：
  - 資料型態代表該指標變數所指向(point to)的是何種資料型態。

8.1.3 宣告指標變數

- 【範例】：合法的指標宣告
  - ANSI C　　C++
  - int *pSum;　　相當於　　int* pSum;
  - char *Name;　　相當於　　char* Name;
  - float *pAvg;　　相當於　　float* pAvg;

- 【說明】：
  - pSum是一個指向整數變數的指標變數，Name是一個指向字元變數的指標變數，pAvg是一個指向浮點型態變數的指標變數。

8.1.4 取址運算子與指標運算子

- 指標與記憶體位址有很大的關係，因此取址運算子「&」可以與指標搭配使用。
- 另外，如果要實際應用指標，則不可避免地，必須使用到提領運算子「*」。
- 『&』取址運算子
  - 『&』稱為取址運算子或位址運算子，主要是用來取出某變數的記憶體位址，換句話說，當我們在變數前面加上「&」符號時，即可取得該變數的記憶體位址。
  - 取址運算子的使用方法如下：

```
&變數名稱
```

8.1.4 取址運算子與指標運算子

- 【範例】：假設x是整數型態的普通變數（內容為20），p是一個整數指標變數，下列片段程式將可以使得指標p指向x所在的記憶體位址（也就是p的內容為存放x的記憶體位址）。

```
int x=20;
int *p;
p=&x;
 /*  使用取址運算子取出變數x的記憶體位址並指定給指標變數p  */
```

透過取址運算子取出變數的記憶體位址並將之設定為指標變數的內容

8.1.4 取址運算子與指標運算子

- 『*』提領運算子（指標運算子）
  - 『*』除了可以作為乘法符號外，在C語言中，也可以做為提領運算子（Dereference），我們可以藉由提領運算子存取指標變數所指向記憶體位址的變數內容。
  - 指標運算子的使用方法如下：

```
*指標變數名稱
```

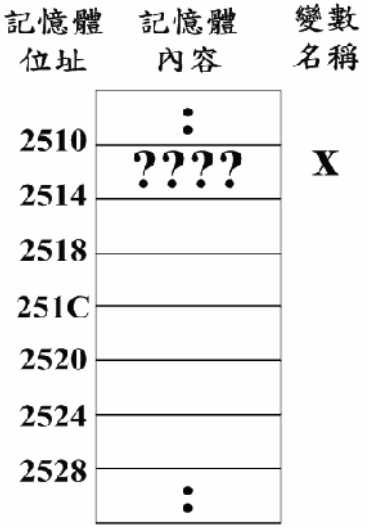
  - 【範例】：

```
1  int x;
2  int *p;
3  p=&x;/* 使用取址運算子取出變數x的記憶體位址並指定給指標變數p */
4  *p=50;/*  透過指標運算子設定x = 50  */
```

8.1.4 取址運算子與指標運算子

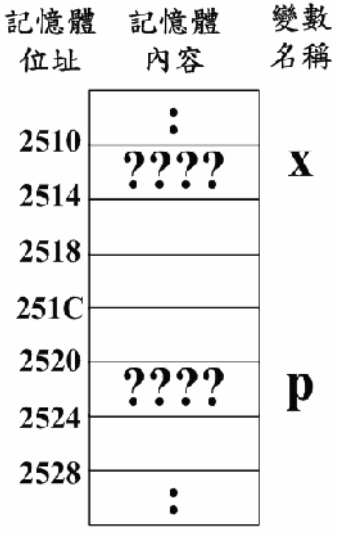
- 這範例的四個敘述的分解動作如下：
  - 第一行敘述執行完畢：

```
int x;
```



  - 第二行敘述執行完畢：

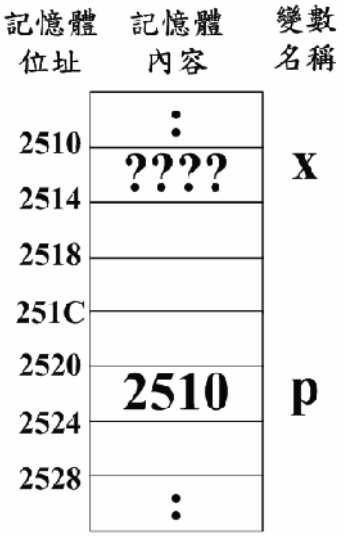
```
int x;
int *p;
```



8.1.4 取址運算子與指標運算子

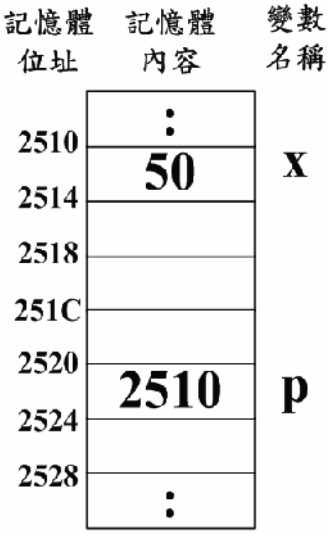
- 第三行敘述執行完畢：

```
int x;
int *p;
p=&x;
```



- 第四行敘述執行完畢：

```
int x;
int *p;
p=&x;
*p=50;
```



8.1.4 取址運算子與指標運算子

- 從上述範例我們可以得知兩件事：
  - p=&x，代表純粹指定指標變數的內容，由於指標變數內容為位址，所以一般搭配『&』取址運算子。
  - *p=50，代表修改指標變數所指向記憶體位址的內容，這就是『*』指標運算子的功能。

8.1.4 取址運算子與指標運算子

- 【觀念範例8-1】：透過取址運算子及指標運算子，將整數變數a的內容指定給整數變數b。
- 範例8-1：ch8_01.c

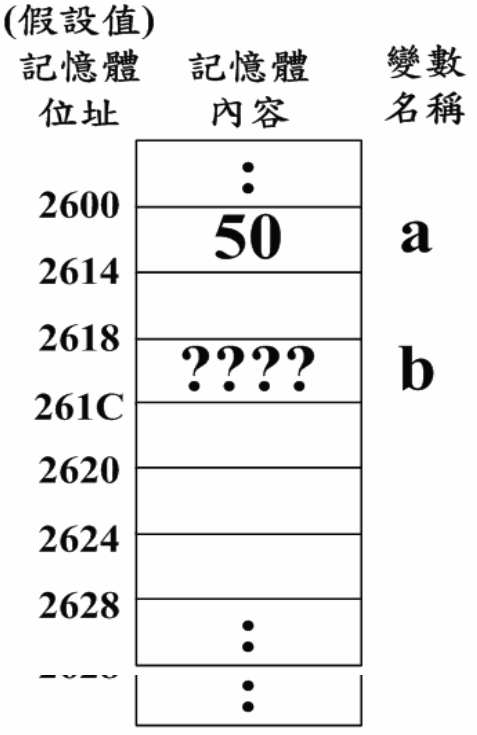
```
1  /*********************************
2      檔名:ch8_01.c
3      功能:取址運算子與指標運算子
4   *********************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  void main(void)
8  {
9   int a=50,b;
10  int *ptr;
11  printf("a=%d\n",a);
12  ptr=&a; /*  將ptr指向a  */
13  printf("*ptr=%d\n",*ptr);
14  b=*ptr; /*  將a的值(ptr指向的值)設定給b  */
15  printf("b=%d\n",b);
16 }
```

- 執行結果：

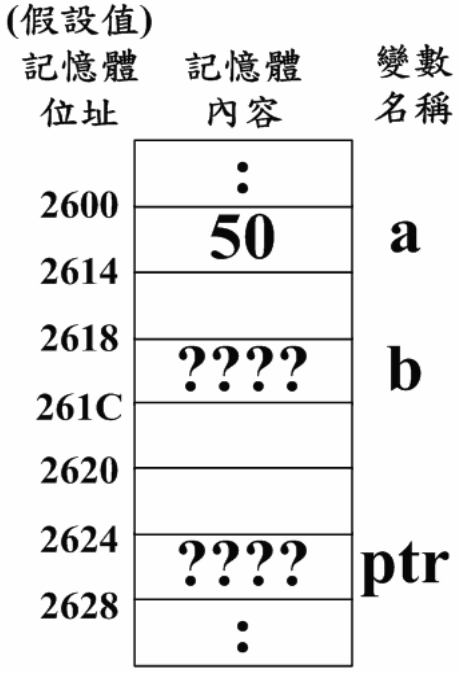
```
a=50
*ptr=50
b=50
```

8.1.4 取址運算子與指標運算子

```
int a=50,b;
```



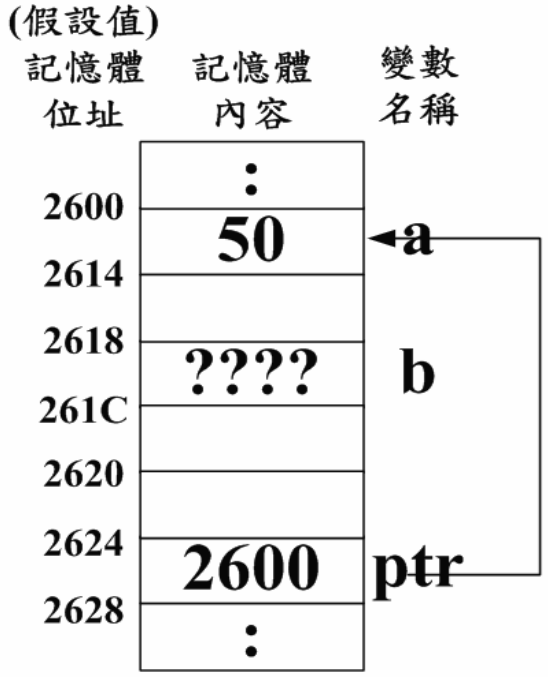

```
int a=50,b;
int *ptr;
```



8.1.4 取址運算子與指標運算子

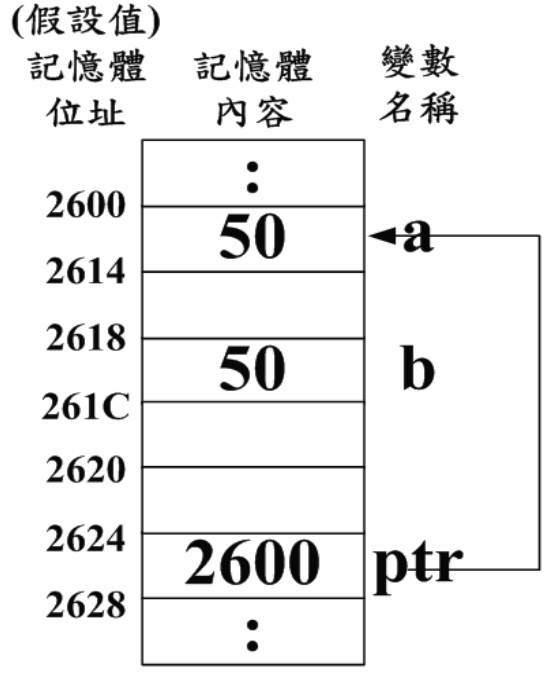
```
int a=50,b;
int *ptr;

ptr=&a;
```




```
int a=50,b;
int *ptr;

ptr=&a;
b=*ptr;
```



8.1.5 指標變數的大小

- 在範例8-1中，我們使用假設的記憶體位址來說明指標變數，事實上，這些假設的記憶體位址與實際的記憶體位址差異頗大，例如我們只用了16個位元（4個16進制位數）來表達記憶體位址。
- 實際上在Windows 2000、XP、2003 Server for 32 bits及Linux Redhat 9等32位元的作業系統環境下，完整的實際位址必須為32位元。
  - 正如上所言，由於實際位址為32位元，因此指標變數若要儲存記憶體位址，也就必須使用4個bytes來加以儲存。
  - 任何一種資料型態的指標變數在這些32位元作業系統的環境下都將佔用4個bytes，因為指標變數存放的是記憶體位址。

8.1.5 指標變數的大小

- 【觀念範例8-2】：觀察指標變數的內容，以及指標變數佔用的記憶體大小。
- 範例8-2：ch8_02.c

```
1  /*******************************************
2      檔名:ch8_02.c
3      功能:指標變數的內容以及指標變數的大小
4   *******************************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7   void main(void)
8  {
9    int a=100;
10   double b=5.5;
11   int *ptr1=&a;    /* 相當於  int *ptr1; 及  ptr1=&a;  */
12   double *ptr2=&b;/* 相當於  double *ptr2; 及  ptr2=&b;  */
13   printf("a=%d\n",a);
     printf("b=%e\n",b);
```

8.1.5 指標變數的大小

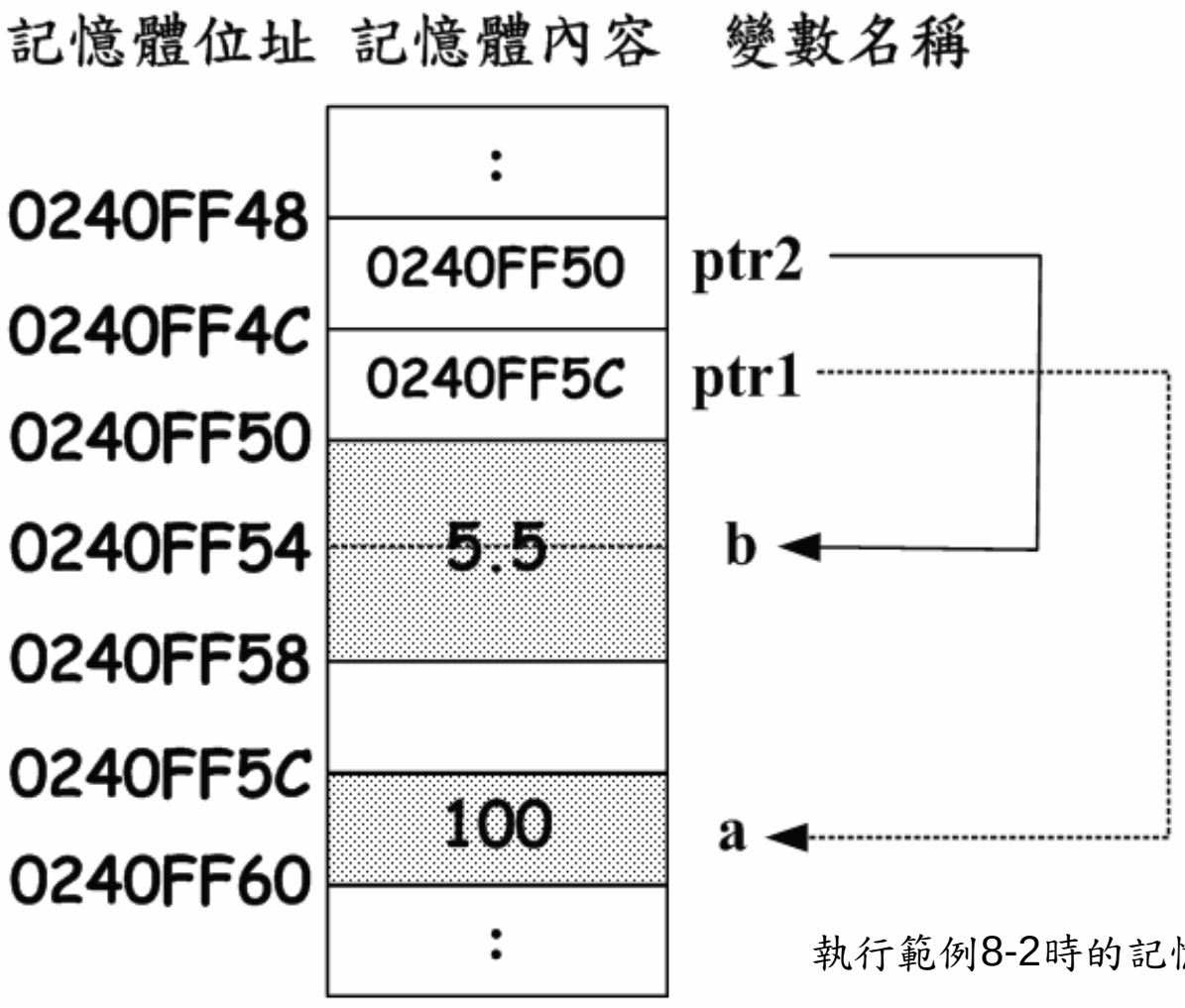
```
14   printf("&a=%p\n",&a);
15   printf("&b=%p\n",&b);
16   printf("*ptr1=%d\n",*ptr1);
17   printf("*ptr2=%e\n",*ptr2);
18   printf("ptr1=%p\n",ptr1);
19   printf("ptr2=%p\n",ptr2);
20   printf("&ptr1=%p\n",&ptr1);
21   printf("&ptr2=%p\n",&ptr2);
22   printf("&*ptr1=%p\n",&*ptr1);/*    &*ptr1相當於&a相當於ptr1   */
23   printf("&*ptr2=%p\n",&*ptr2);/*    &*ptr2相當於&b相當於ptr2   */
24   printf("============================\n");
25   printf("變數a佔用%d個位元組\n",sizeof(a));
26   printf("變數b佔用%d個位元組\n",sizeof(b));
27   printf("============================\n");
28   printf("變數ptr1佔用%d個位元組\n",sizeof(ptr1));
29   printf("變數ptr2佔用%d個位元組\n",sizeof(ptr2));
30   /*  system("pause");  */
31  }
```

8.1.5 指標變數的大小

- 執行結果：

```
a=100
b=5.500000e+000
&a=0240FF5C
&b=0240FF50
*ptr1=100
*ptr2=5.500000e+000
ptr1=0240FF5C
ptr2=0240FF50
&ptr1=0240FF4C
&ptr2=0240FF48
&*ptr1=0240FF5C
&*ptr2=0240FF50
================================
變數a佔用4個位元組
變數b佔用8個位元組
================================
變數ptr1佔用4個位元組
變數ptr2佔用4個位元組
```

8.1.5 指標變數的大小



執行範例8-2時的記憶體配置狀況

8.2 指標運算

- C語言提供下列四種與指標有關的基本運算功能，讓指標充分展現間接存取資料的優點：
  - (1)指定運算
  - (2)加減運算
  - (3)比較運算
  - (4)差值運算

8.2.1 指標的指定運算

- 【觀念範例8-3】：透過指標變數的指定運算，設定兩個指標指向同一位址，並修改該位址的內容。
- 範例8-3：ch8_03.c

```
1   /*****************************
2       檔名:ch8_03.c
3       功能:指標變數的指定運算
4    *****************************/
5
6   #include <stdio.h>
7   #include <stdlib.h>
8
9   void main(void)
10  {
11   int a=100;
12   int *p,*q;
13
14   printf("============宣告變數時=============\n");
15   printf("&a=%p\t a=%d\n",&a,a);
16   printf("&p=%p\n",&p);
17   printf("&q=%p\n",&q);
```

8.2.1 指標的指定運算

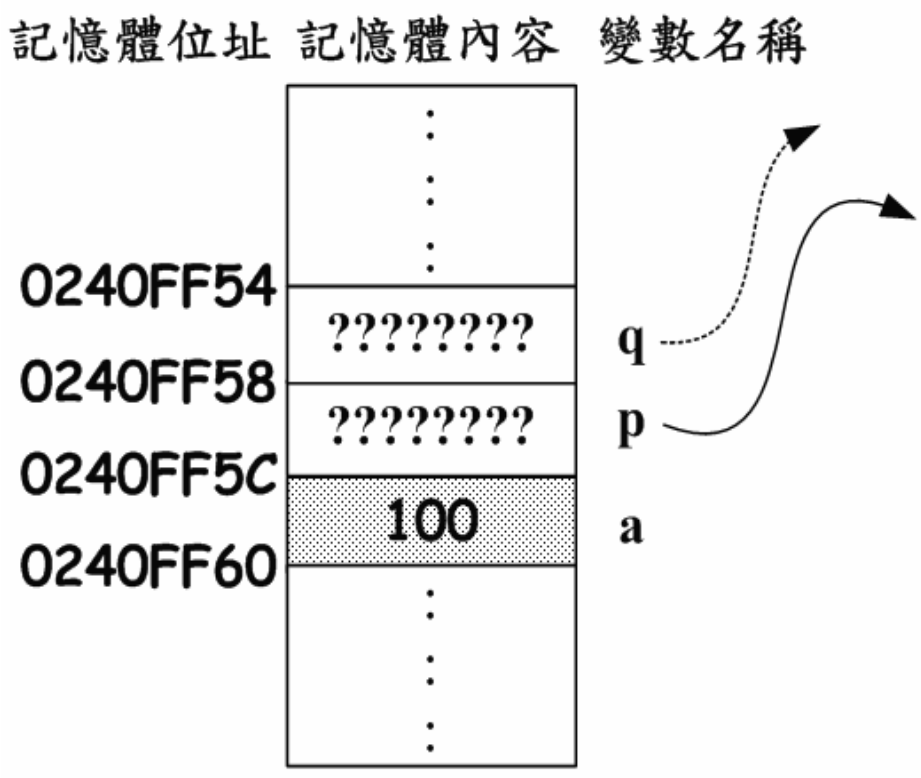
```
18  p=&a;
19  printf("==========設定p=&a後==========\n");
20  printf("p=%p\t *p=%d\n",p,*p);
21  q=p;
22  printf("==========設定q=p後==========\n");
23  printf("q=%p\t *q=%d\n",q,*q);
24   *q=50;
25  printf("==========設定*q=50後==========\n");
26  printf("p=%p\t *p=%d\n",p,*p);
27  printf("q=%p\t *q=%d\n",q,*q);
28  printf("a=%d\n",a);
29  /*   system("pause");   */
30  }
```

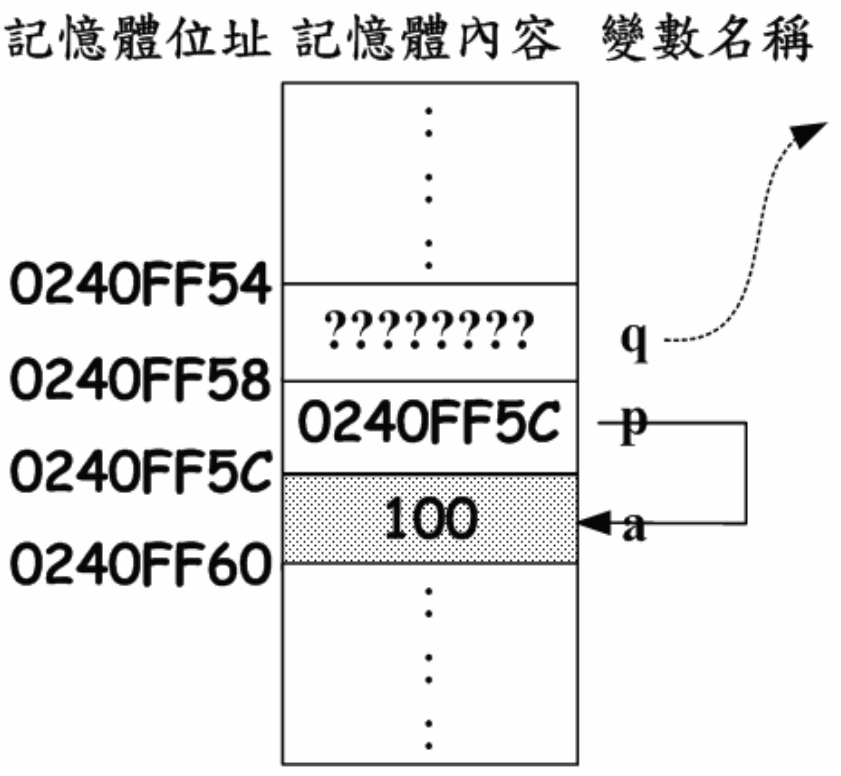
8.2.1 指標的指定運算

- 執行結果：

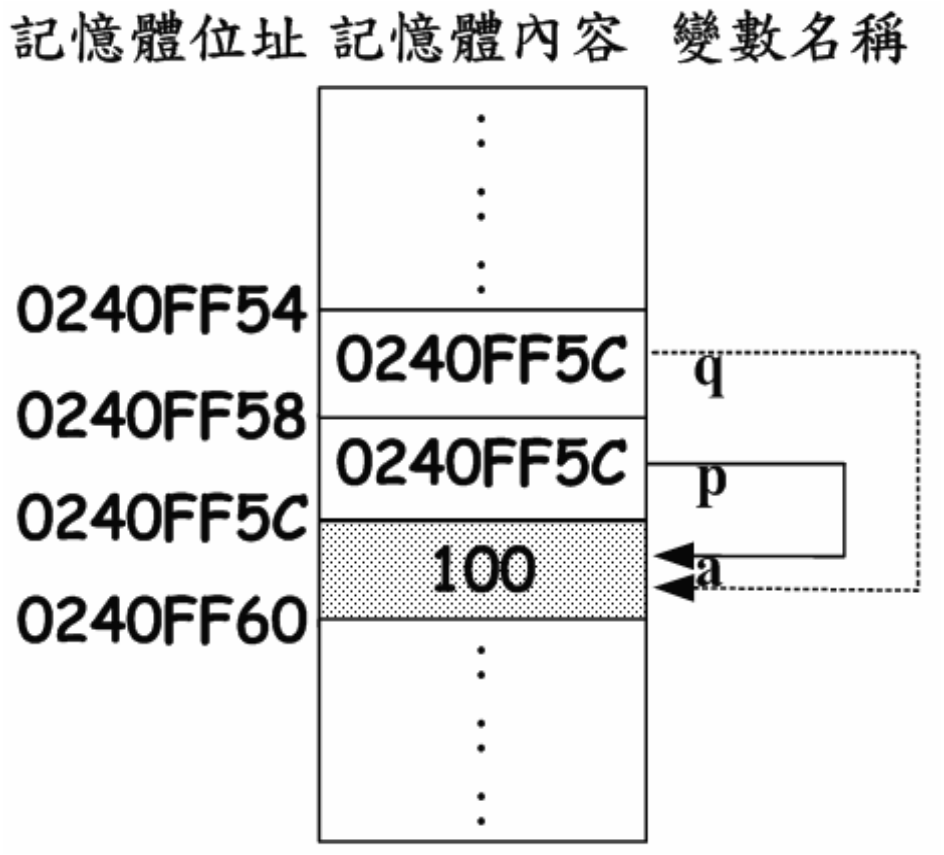
```
============宣告變數時==============
&a=0240FF5C      a=100
&p=0240FF58
&q=0240FF54
============設定p=&a後==============
p=0240FF5C       *p=100
============設定q=p後==============
q=0240FF5C       *q=100
============設定*q=50後============
p=0240FF5C       *p=50
q=0240FF5C       *q=50
a=50
```

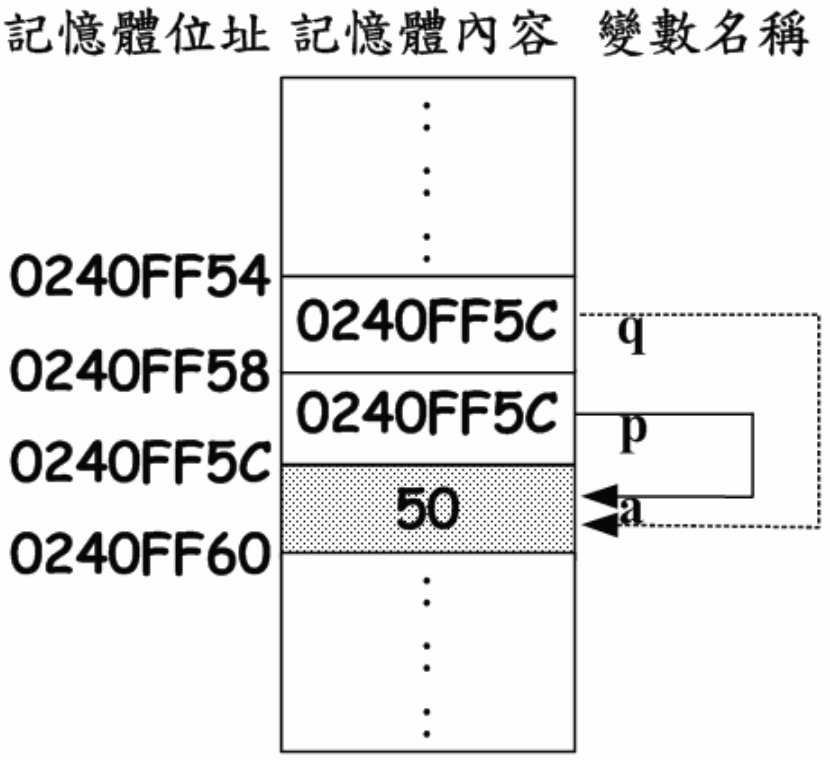
8.2.1 指標的指定運算


記憶體位址 記憶體內容 變數名稱
0240FF54
0240FF58
0240FF5C
0240FF60
????????
????????
100
q
p
a


記憶體位址 記憶體內容 變數名稱
0240FF54
0240FF58
0240FF5C
0240FF60
????????
0240FF5C
100
q
p
a

8.2.1 指標的指定運算


記憶體位址 記憶體內容 變數名稱
0240FF54
0240FF58
0240FF5C
0240FF60
0240FF5C
0240FF5C
100
q
p
a


記憶體位址 記憶體內容 變數名稱
0240FF54
0240FF58
0240FF5C
0240FF60
0240FF5C
0240FF5C
50
q
p
a

8.2.1 指標的指定運算

- 指標變數的初始值設定
  - 指標變數的指定運算與普通變數的指定運算差別不大，只不過指定的值是某一個變數的位址或另一個指標變數值。
  - 在設定初始值方面，指標變數和普通變數可就有很大的不同。
  - 當我們宣告普通變數時，可以將該變數設定一個初值，例如：
    - int a=100;
  - 但對於指標變數來說，我們不可以執行下列初始值設定的動作：
    - int *p=100;　　/*　這是錯誤的敘述　*/

8.2.1 指標的指定運算

- 為了要避免上述這種錯誤，我們可以在宣告指標變數時，同時設定指標指向一個合法的記憶體位址（如下語法），如此一來，不論您如何修改指標所指記憶體位址的內容，都不會發生此種嚴重的錯誤了。
  - int a;
    int *p=&a;
- 上述語法中，『int *p=&a;』將被編譯器分解成『int *p;』與『p=&a;』來執行，因此不會出現錯誤。

8.2.2 指標變數的加減運算

- 指標變數可以做加減運算，由於指標變數的內容為記憶體位址，所以指標變數的加減運算通常是用來增減記憶體位址的位移。

- 換句話說，當某個指標變數做加法時，代表將該指標往後指幾個單位，當某個指標變數做減法時，代表將該指標往前指幾個單位，而移動的單位除了與加減運算的數值有關，也與指標所指的資料型態有關。

8.2.2 指標變數的加減運算

【觀念範例8-4】：指標變數加減法的示範以及資料型態對於指標變數加減法的影響。

範例8-4：ch8_04.c

```
1  /*****************************
2      檔名:ch8_04.c
3      功能:指標變數的加法運算
4   *****************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  void main(void)
8  {
9   int a;
10  short int  *p;
11  int        *q;
12  float      *r;
13  double     *s;
14  p=(short int*) &a;
15  q=&a;
16  r=(float*) &a;
17  s=(double*) &a;
```

```
21
22  printf("p=%p\n",p);
23  printf("q=%p\n",q);
24  printf("r=%p\n",r);
25  printf("s=%p\n",s);
26  printf("======================\n");
27  p=p+1;
28  q=q+1;
29  r=r+1;
30  s=s+1;
31  printf("p=%p\n",p);
32  printf("q=%p\n",q);
33  printf("r=%p\n",r);
34  printf("s=%p\n",s);
35  /*  system("pause");  */
36 }
```

8.2.2 指標變數的加減運算

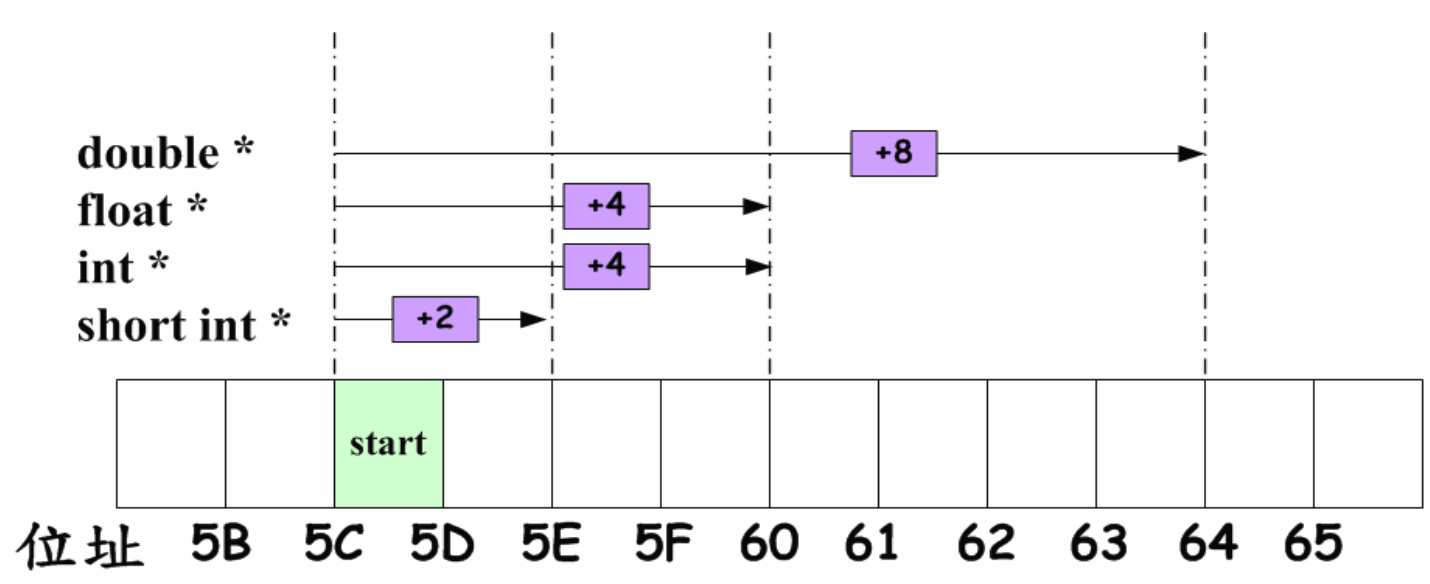
- 執行結果：（使用Dev-C++編譯）

```
p=0240FF5C
q=0240FF5C
r=0240FF5C
s=0240FF5C
================================
p=0240FF5E
q=0240FF60
r=0240FF60
s=0240FF64
```

- 範例說明：
  - 第12~15行：宣告4種指向不同資料型態的指標p,q,r,s。
  - 第17~20行：將變數a的位址指定為指標變數內容，也就是指標要指向的位址。
  - 由於資料型態的不一致，因此必須使用強制型別轉換，來設定正確的指標型態。

8.2.2 指標變數的加減運算

- (3)第27~30行：將指標變數內容『加1』，其實並非只將位址（指標變數的內容）加1而已，它會依據指標指向的資料型態所佔用的記憶體單位大小，來決定移動多少個位址（加1代表加1個單位），以便指向正確的下一筆同樣資料類型的資料。例如：int佔用4個bytes，所以整數指標『加1』，代表位址移動4個bytes。本範例的指標相對位移如下圖。



指標變數的加法示意圖

8.2.2 指標變數的加減運算

- 指標加法只能用來加上『常數值』。不可以做指標變數的相加（如下範例會發生錯誤）因為兩個指標變數的內容相加（位址相加），並不具備任何實際的意義，而且容易使得指標指向不合法的位址。

```
int *p,*q;
p=p+q;    /* 不合法 */
```

8.2.3 指標變數的比較運算

- 『相同型態』的指標變數可以做比較運算，藉由比較運算，我們可以得知記憶體位址的先後關係。
- 【觀念範例8-5】：觀察指標變數內容，得知Windows作業系統會將Dev-C++要求的變數記憶體配置，由高位址開始往低位址配置。
- 範例8-5：ch8_05.c

```
1  /*****************************
2       檔名:ch8_05.c
3       功能:指標變數的比較運算
4   *****************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7   void main(void)
8  {
9   int a,b;
10  int *p=&a,*q=&b;
```

8.2.3 指標變數的比較運算

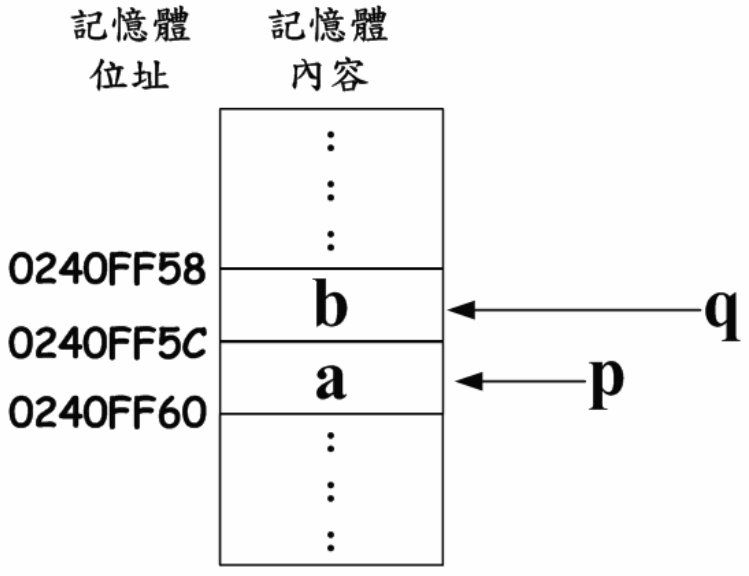
```
11   printf("指標p指向記憶體位址%p\n",p);
12   printf("指標q指向記憶體位址%p\n",q);
13   if(p>q)
14     printf("變數a的記憶體位址高於變數b的記憶體位址\n");
15   else
16     printf("變數b的記憶體位址高於變數a的記憶體位址\n");
17   /*  system("pause");  */
18 }
```

- 執行結果：

```
指標p指向記憶體位址0240FF5C
指標q指向記憶體位址0240FF58
變數a的記憶體位址高於變數b的記憶體位址
```

8.2.3 指標變數的比較運算

- 範例說明：
  - 記憶體配置如下圖。



較早配置記憶體的變數位於較高的記憶體位址

8.2.4 指標變數的差值運算

- 雖然兩個指標變數無法做加法運算，但兩個相同資料型態的指標變數卻可以做減法運算，稱之為『差值運算』。
- 指標變數差值運算所得的結果代表兩個記憶體位址之間的可存放多少個該資料型態的資料。
- 【實用範例8-6】：計算二維陣列（陣列大小為8*15）元素[2][6]~[6][10]共佔用多少位元組。
- 範例8-6：ch8_06.c

```
1  /*****************************
2      檔名:ch8_06.c
3      功能:計算相隔元素個數
4   *****************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7   void main(void)
8  {
9   double array[8][15];
10  double *p,*q;
11  int blocksize,count;
```

8.2.4 指標變數的差值運算

```
12   p=&array[2][6];
13   q=&array[6][11];
14   count=q-p;
15   blocksize=count*sizeof(double);
16   printf("p=%p\t q=%p\n",p,q);
17   printf("元素[2][6](含)~[6][10](含)之間共有%d個元素\n",count);
18   printf("元素[2][6](含)~[6][10](含)之間的記憶體區塊大小為");
19   printf("%d位元組\n",blocksize);
20   /*  system("pause");  */
21  }
```

- 執行結果：

```
p=0240FCC0          q=0240FEC8
元素[2][6](含)~[6][10](含)之間共有65個元素
元素[2][6](含)~[6][10](含)之間的記憶體區塊大小為520位元組
```

8.3 函式的傳指標呼叫

- 在上一章中，我們曾經提及，C語言提供了傳指標呼叫以符合程式語言的傳址呼叫方式。
- 【實用範例8-7】：實作整數交換函式swap( )。
- 範例8-7：ch8_07.c

```
1  /*********************************
2      檔名:ch8_07.c
3      功能:傳指標呼叫實作swap( )
4   *********************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  void swap(int *a,int *b)
8  {
9   int temp;
10  temp=*a;
11  *a=*b;
12  *b=temp;
13 }
```

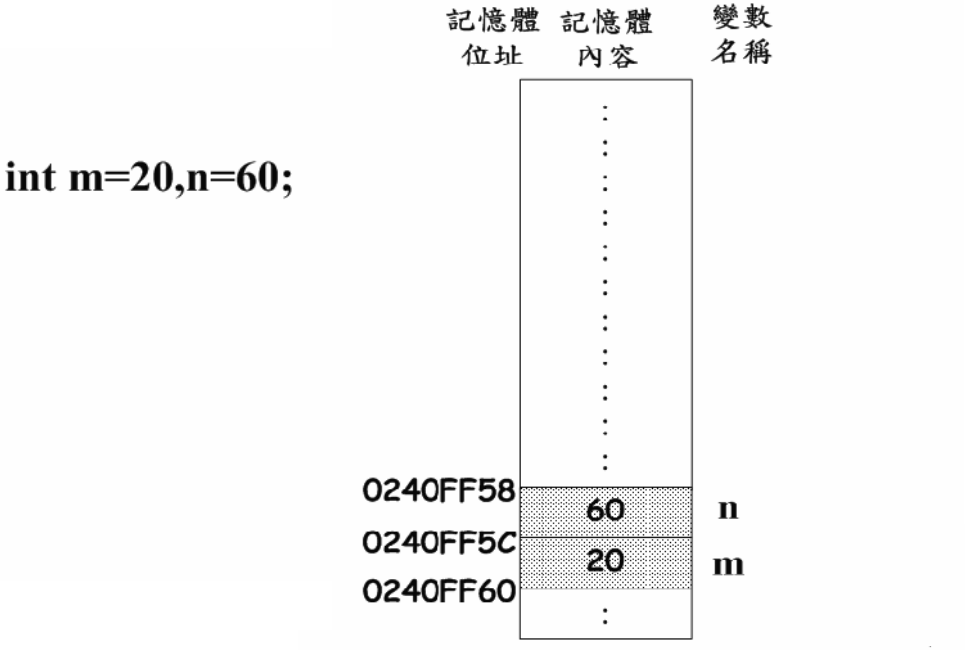
8.3 函式的傳指標呼叫

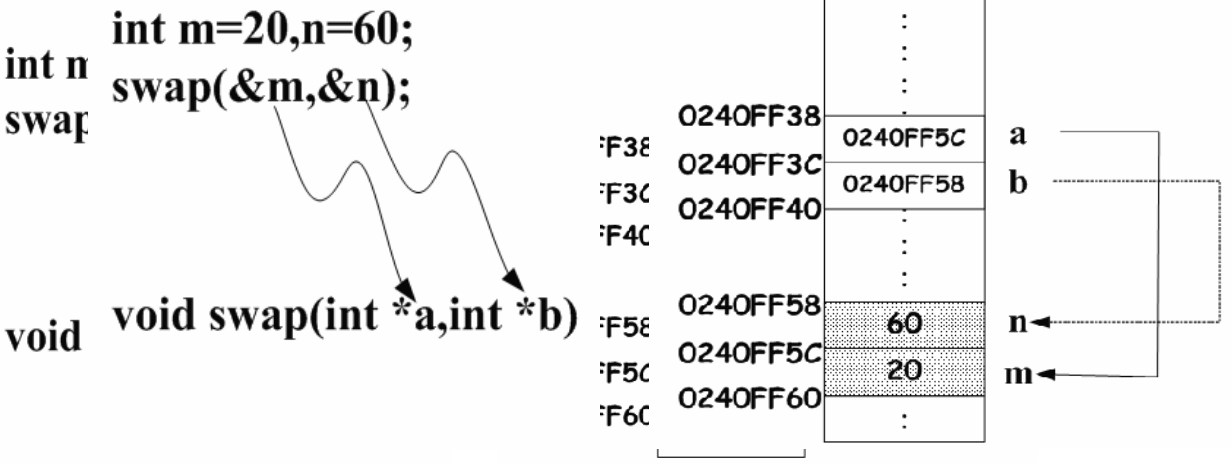
```
14 void main(void)
15 {
16  int m=20,n=60;
17  printf("變換前(m,n)=(%d,%d)\n",m,n);
18  swap(&m,&n);
19  printf("變換後(m,n)=(%d,%d)\n",m,n);
20  /*  system("pause");  */
21 }
```

- 執行結果：

```
變換前(m,n)=(20,60)
變換後(m,n)=(60,20)
```

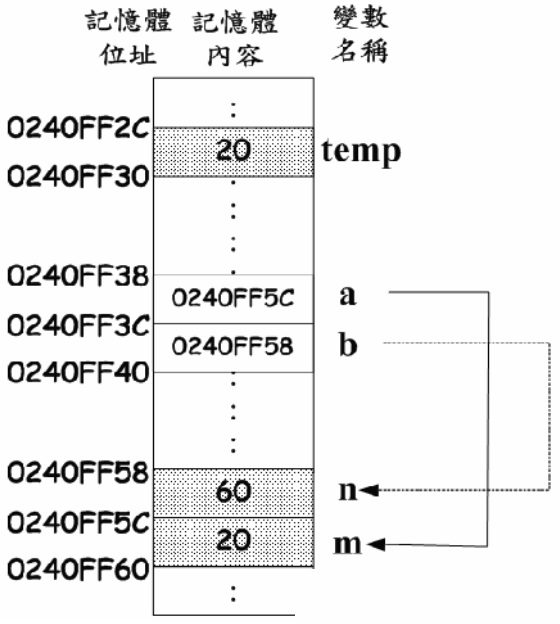
8.3 函式的傳指標呼叫


記憶體 位址
記憶體 內容
變數 名稱
int m=20,n=60;
0240FF58
60
n
0240FF5C
20
m
0240FF60

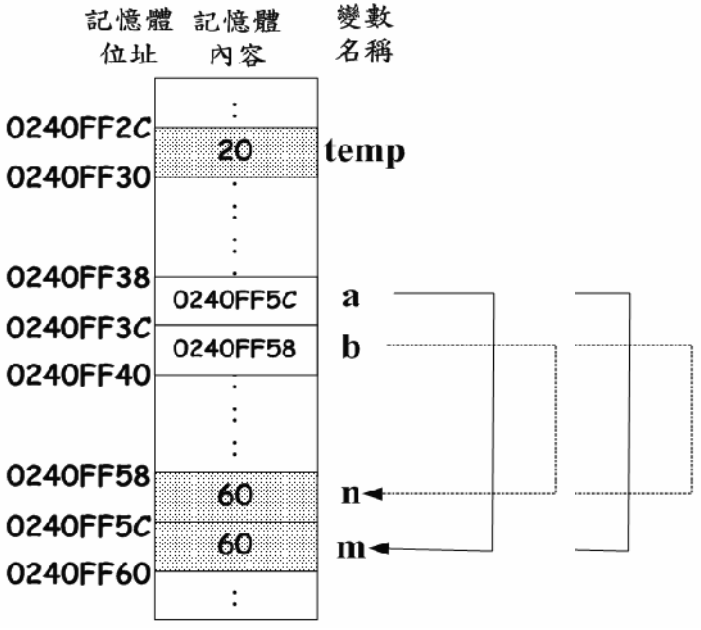

int m=20,n=60;
swap(&m,&n);
void swap(int *a,int *b)
記憶體 位址
記憶體 內容
變數 名稱
0240FF38
0240FF5C
a
0240FF3C
0240FF58
b
0240FF40
0240FF58
60
n
0240FF5C
20
m
0240FF60

8.3 函式的傳指標呼叫

```
int temp;
temp=*a;
```

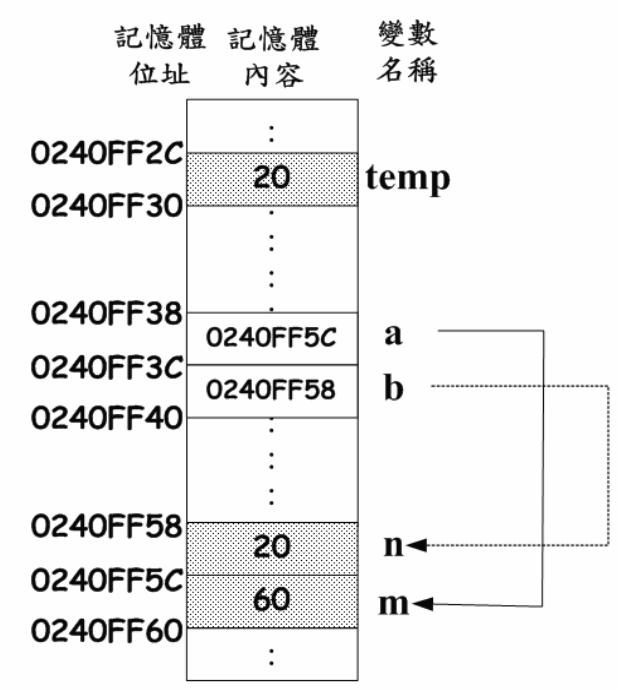


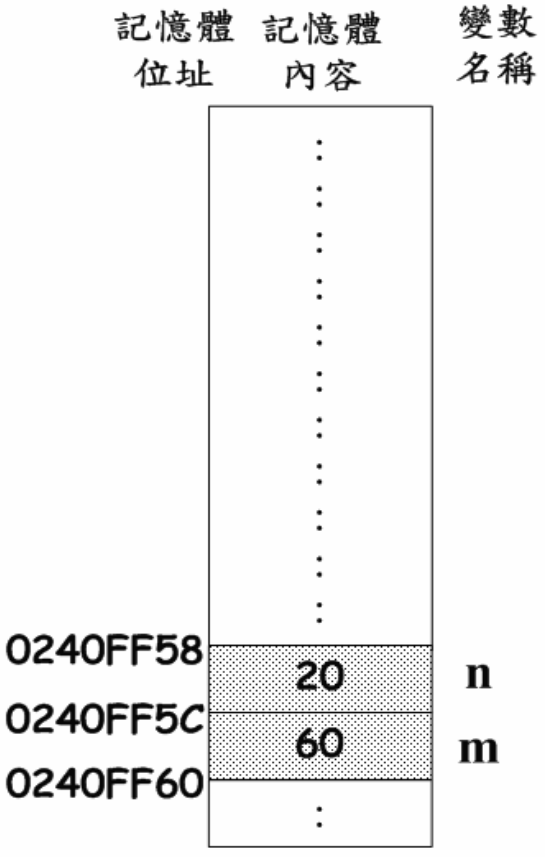

```
int temp;
temp=*a;
*a=*b;
```



8.3 函式的傳指標呼叫

```
int temp;
temp=*a;
*a=*b;
*b=temp;
```





8.4 『指標』、『陣列』、『字串』的關係

- 在C語言中，字串是一種特殊的字元陣列（此種字串稱之為字元字串character string）。

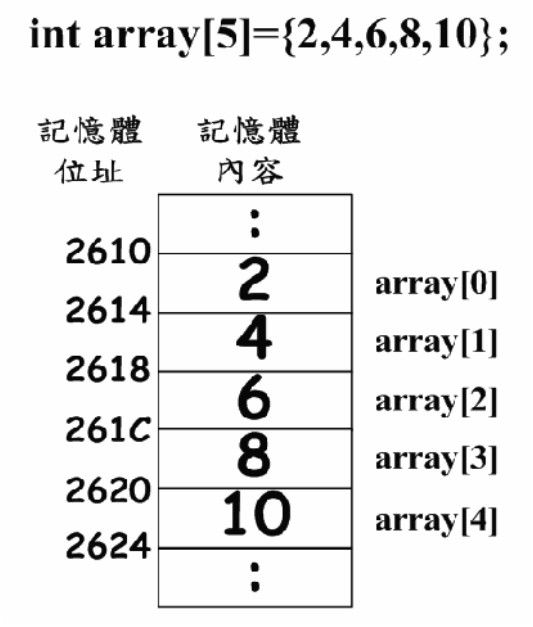
- 陣列名稱也是一個記憶體位址，因此，指標、陣列、字串三者之間有著耐人尋味的關係，在本節中，我們將深入說明此三者的關係，並說明使用指標的優點。

8.4 『指標』、『陣列』、『字串』的關係

- 8.4.1 指標與陣列
  - 有許多C語言的入門書籍提到『陣列』其實就是『指標』，這幾乎是正確的。
  - 但更精確的說法應該是，陣列名稱可以視為一個常數指標（不會變動內容的指標）來加以操作，它指向陣列在記憶體中開始的位址。

8.4.1 指標與陣列

int array[5]={2,4,6,8,10};



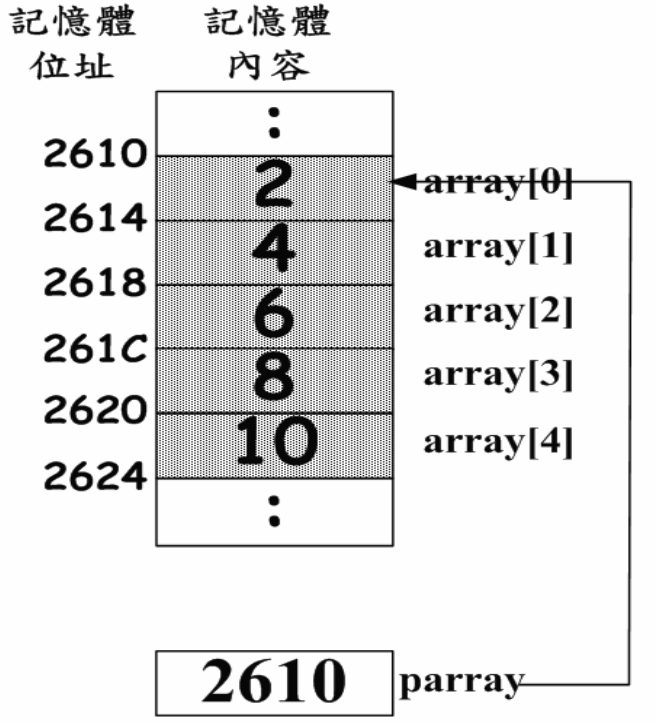
宣告陣列後，記憶體配置連續記憶體

- 事實上，要取出陣列元素也可以透過指標來完成，首先，我們先宣告一個指標parray，並將之指定為a[0]的位址，如下語法：

```
int array[5]={2,4,6,8,10};
int *parray;
parray=&array[0];
```

8.4.1 指標與陣列

- 此時將有一個指標parray指向陣列的第一個元素array[0]，如下圖：



將指標parray指向陣列的第一個元素

8.4.1 指標與陣列

- 經過上述的動作之後，如果您想要取出陣列第四個元素8放入value變數，可以透過『value=*(parray+3);』來取出元素值8，因為實際上*(parray+3)，代表的是*(2610+4*3)，它使用的原理是指標加法代表位址的位移，因此所謂+3代表移動3個單位。

- 在上述語法中，您會發現，執行『value=*(parray+3);』並不會改變parray指標的值。

- 在C語言中，所謂『指標』變數，代表著一個變數儲存著某一個記憶體位址。而陣列名稱實際上也是一個記憶體位址（該位址為陣列第一個元素的記憶體位址），

8.4.1 指標與陣列

- 因此我們可以使用下列語法取得陣列的第一個元素的記憶體位址位址：
  - parray=array;/* 效力等同於 parray = &array[0] */

- C語言提供了一個簡便的方法，以節省指標變數。也就是直接將陣列名稱當做指標來使用。因此下列語法也是相等的。
  - value=*(array+3); 等同於 value=array[3];

- 上述語法看起來不同，但實際上對於編譯器而言則會將之翻譯成相同的目的碼，並且根據C語法發明人K & R在其著作The C Programming Language中宣稱，在計算array[i]時，C語言會將之轉換為*(array+i)來加以運作。

8.4.1 指標與陣列

- 我們將陣列名稱的獨特性整理如下：
  - (1)陣列名稱是一個記憶體位址，該位址一定是陣列第一個元素在記憶體中的位址。
  - (2)陣列名稱所代表的記憶體位址可以被讀取，但不可被更改。因此陣列名稱具有唯讀性。
  - (3)您可以將陣列名稱當做一個唯讀的指標來運作，但不可以改變其值。

- 在瞭解了陣列名稱與指標變數的差別之後，我們重新將陣列元素的存取語法整理如下：
- 存取一維陣列中的第i個元素語法如下：

語法：
陣列表示法：**陣列名稱[i]**
指標表示法：*(陣列名稱+i)

8.4.1 指標與陣列

- 【範例】：
  - array[0]　　存取array陣列的第一個元素
  - *array　　存取array陣列的第一個元素
  - array[2]　　存取array陣列的第三個元素
  - *(array+2)　　存取array陣列的第三個元素

- 【觀念範例8-8】：改寫範例6-2，使用指標方式來存取陣列元素。
- 範例8-8：ch8_08.c

8.4.1 指標與陣列

```
1  /*********************************
2  檔名:ch8_08.c
3  功能:使用指標存取陣列元素
4   *********************************/
5
6   #include <stdio.h>
7  #include <stdlib.h>
8   void main(void)
9  {
10   float Temper[12],sum=0,average;
11   int i;
12    for(i=0;i<12;i++)
13   {
14       printf("%d月的平均溫度:",i+1);
15       scanf("%f",(Temper+i));
16       sum=sum+*(Temper+i);
17   }
18   average=sum/12;
19   printf("=============================\n");
20   printf("年度平均溫度:%f\n",average);
21   /*  system("pause");  */
22  }
```

□ 執行結果：

```
1月的平均溫度:15.6
2月的平均溫度:17.3
3月的平均溫度:24.2
4月的平均溫度:26.7
5月的平均溫度:28.4
6月的平均溫度:30.2
7月的平均溫度:29.6
8月的平均溫度:30.5
9月的平均溫度:29.2
10月的平均溫度:28.6
11月的平均溫度:25.4
12月的平均溫度:22.9
=============================
年度平均溫度:25.716667
```

8.4.2 指標與字串

- 陣列可以視為指標常數（不會變動內容的指標），它代表著陣列在記憶體中開始的位址（也就是陣列第一個元素的記憶體位址）。
- 字串既然是一種特殊的字元陣列。

8.4.2 指標與字串

- 【觀念範例8-9】：釐清普通字元陣列與字元字串的差異。
- 範例8-9：ch8_09.c

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```

```
/*********************************
    檔名:ch8_09.c
    功能:普通字元陣列與字元字串
 *********************************/
 #include <stdio.h>
#include <stdlib.h>
void main(void)
{
 char s1[]={'W','e','l','c','o','m','e'};
 char s2[]="Welcome";
 printf("s1字元陣列佔用記憶體%d bytes\n",sizeof(s1));
 printf("s2字元字串佔用記憶體%d bytes\n",sizeof(s2));
 /*  system("pause");  */
}
```

8.4.2 指標與字串

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
```

```
/*********************************
    檔名:ch8_09.c
    功能:普通字元陣列與字元字串
 *********************************/
 #include <stdio.h>
#include <stdlib.h>
void main(void)
{
 char s1[]={'W','e','l','c','o','m','e'};
 char s2[]="Welcome";
 printf("s1字元陣列佔用記憶體%d bytes\n",sizeof(s1));
 printf("s2字元字串佔用記憶體%d bytes\n",sizeof(s2));
 /*  system("pause");  */
}
```

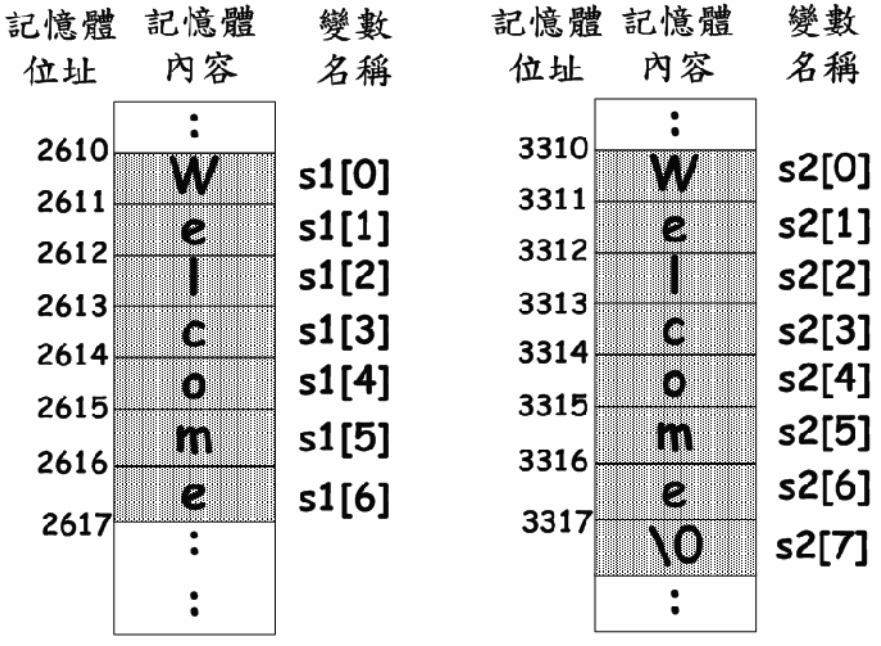
8.4.2 指標與字串

- 執行結果：

```
s1字元陣列佔用記憶體7 bytes
s2字元字串佔用記憶體8 bytes
```

- s1與s2的記憶體配置如下，s2必須存放『'\0'』字串結尾字元，所以大小為8個bytes。

普通字元陣列與字元字串相差一個位元



8.4.2 指標與字串

- 事實上，除了以陣列方式宣告字元字串之外，我們也可以使用指標方式來宣告字串（稱之為指標字串），例如：『char *s3="Welcome"』。
- 【觀念範例8-10】：宣告指標字串。
- 範例8-10：ch8_10.c

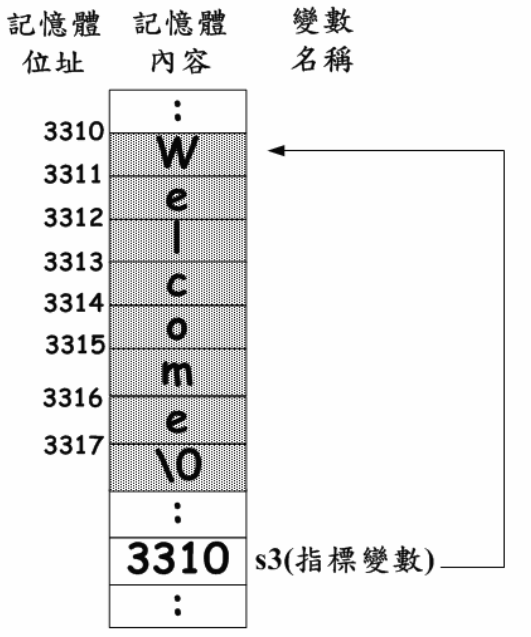
8.4.2 指標與字串

```
1   /*****************************
2       檔名:ch8_10.c
3       功能:指標字串
4    *****************************/
5   #include <stdio.h>
6   #include <stdlib.h>
7   void main(void)
8   {
9    char *s3="Welcome";
10   int i;
11   for(i=0;i<8;i++)
12     if (s3[i] != '\0')
13       printf("s3[%d]=%c\n",i,s3[i]);
14     else
15       printf("s3[%d]='\\0'\n",i);
16   /*  system("pause");  */
17  }
```

8.4.2 指標與字串

- 執行結果：

```
s3[0]=W
s3[1]=e
s3[2]=l
s3[3]=c
s3[4]=o
s3[5]=m
s3[6]=e
s3[7]='\0'
```



- 範例說明：
  - 使用*(s3+i)或s3[i]都可以讀取代表字串的陣列元素。
  - s3的記憶體配置如下，它的大小仍是8個bytes。請特別注意的是，以往我們宣告『int *p=&a；』時，編譯器將之分解為『int *p;』及『p=&a』，同理，當我們宣告『char *s3="Welcome";』時，編譯器也會將之分解為『char *s3;』及『s3=&("Welcome");』。

8.4.2 指標與字串

- 【觀念範例8-11】：釐清指標字串的指標變數與字串陣列的陣列名稱（指標常數）。
- 範例8-11：ch8_11.c

```
1  /*****************************
2      檔名:ch8_11.c
3      功能:指標字串與字元字串
4   *****************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  void main(void)
8  {
9   char s2[]="Welcome";
10  char *s3="Welcome";
11  char *s4;
12  char *s5="Good morning";;
13   /*  s2=s5;  *//*  此行不合法 */
14  s3=s5;
15  s4=s2;
16  printf("s2=%s\n",s2);
17  printf("s3=%s\n",s3);
18  printf("s4=%s\n",s4);
19 }
```

- 執行結果：

```
s2=Welcome
s3=Good morning
s4=Welcome
```

8.4.2 指標與字串

- 範例說明：
  - s3、s4、s5是指標變數，所以可以指定指向其他的記憶體位址（例如第17行，指標s3將指向s5字串的開始位址、第18行指標s4將指向s2字串的開始位址）。
  - s2是陣列名稱（唯讀的位址），所以無法指向其他記憶體位址（若將第16行的註解取消，將無法通過編譯）。

8.4.2 指標與字串

- 剖析陣列名稱（指標常數）與指標變數
  - 在前面的內容中，我們一直強調陣列名稱與指標變數是不同的，指標變數一定會佔用一個記憶體位址，但陣列名稱卻不會。覺得奇怪嗎？那我們就來看看下面的四個小程式。

(1)

```
char s[]="abc";
s[2]='d';
```

(2)

```
char s[]="abc";
*(s+2)='d';
```

(3)

```
char *s="abc";
*(s+2)='d';
```

(4)

```
char *s="abc";
s[2]='d';
```

8.4.2 指標與字串

- 上面的四個小程式都可以將s字串修改為“abd”。
- (1)(2)將字串s宣告為以陣列為主的字元字串，而(3)(4)將字串s宣告為指標字串。
- 其中(1)與(3)的使用上很單純，陣列型態的字元字串使用陣列來存取字元。

8.4.2 指標與字串

- 關於『字串常數』、『陣列名稱』與『指標變數』的觀念，假設我們有下列片段程式：

```
char s1[] = "good1";            /* array */
char *s2 = "good2";      /* pointer */
char *s3;                /* pointer */
s3="good3";
```

  - 上述的“good1”、“good2”、“good3”在編譯時，都會將之編譯為字串常數而儲存在資料段中。
  - 但s1卻不會如此，意即下列語法無法通過編譯：

```
char s1[];
s1="good1";      /*   無法通過編譯   */
```

8.4.2 指標與字串

- 多維陣列字串與指標
  - 除了指標是否唯讀的差異之外，乍看之下，用一維陣列或指標宣告字串並沒有太大的不同，但這只僅限於對於一維陣列的字串。對於二維以上的字串陣列而言，兩種宣告方式就有很大的差異。
  - 以往我們要儲存一群字串時，通常會使用二維陣列來儲存這些字串。例如：我們想要將一週的英文單字存入陣列，則會如下宣告2維陣列：

```
char Week[7][10]
={"Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday"};
```

8.4.2 指標與字串

- 由於要存放最長的字串，所以二維部分的寬度必須宣告為10，經由上述宣告後，記憶體配置如下圖：

```
char Week[7][10] ={"Monday","Tuesday","Wednesday","Thursday",
                   "Friday","Saturday","Sunday"};
```

0x2000	M	o	n	d	a	y	\0			
0x200a	T	u	e	s	d	a	y	\0		
0x2014	W	e	d	n	e	s	d	a	y	\0
0x201e	T	h	u	r	s	d	a	y	\0	
0x2028	F	r	i	d	a	y	\0			
0x2032	S	a	t	u	r	d	a	y	\0	
0x203c	S	u	n	d	a	y	\0			

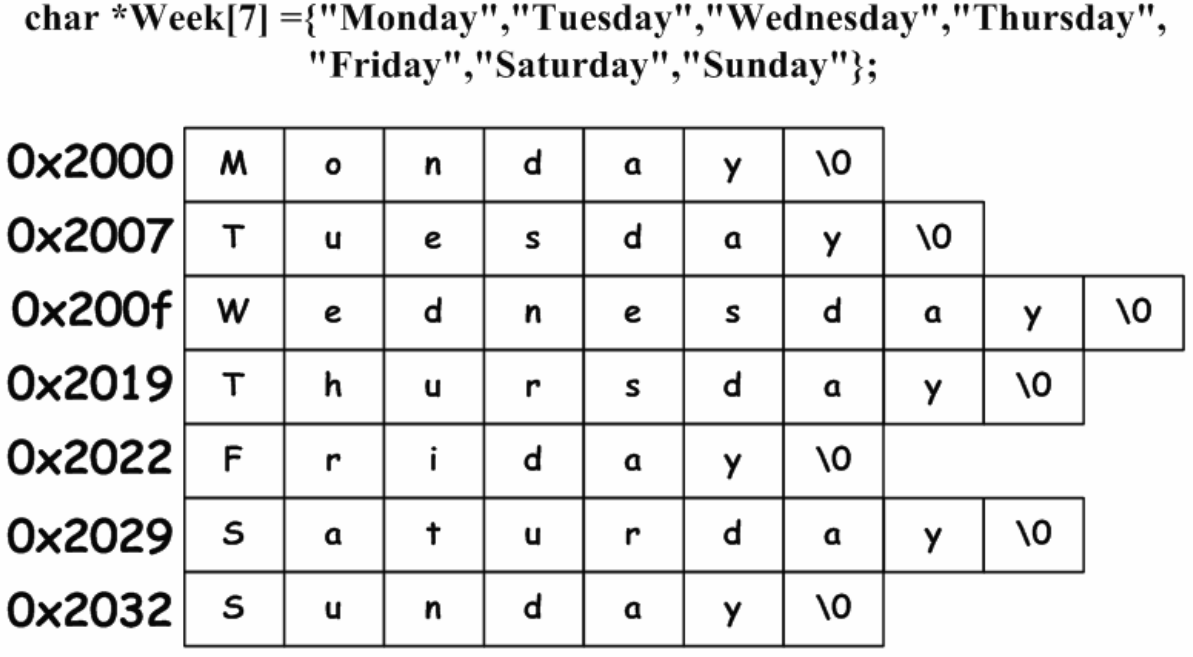
使用二維陣列宣告方式儲存字串群

8.4.2 指標與字串

- 上圖中，黑底的記憶體空間明顯地被浪費了，如果使用指標陣列來存放這些字串，宣告方式如下：

```
char *Week[7]
={"Monday","Tuesday","Wednesday","Thursday","Friday","Saturday","Sunday"};
```

- 經由上述宣告後，記憶體配置如下圖：



8.4.2 指標與字串

- 【觀念範例8-12】：取出指標陣列內的字串。
- 範例8-12：ch8_12.c

```
1  /*************************
2      檔名:ch8_12.c
3      功能:指標陣列與字串陣列
4   *************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7  void main(void)
8  {
9   int i;
10  char
11 *Week[7]={"Monday","Tuesday","Wednesday","Thursday","Friday","Saturday
12 ","Sunday"};
13        for(i=0;i<=6;i++)
14     printf("Week[%d]=%s\n",i,Week[i]);
15  /*  system("pause");  */
   }
```

8.4.2 指標與字串

- 執行結果：

```
Week[0]=Monday
Week[1]=Tuesday
Week[2]=Wednesday
Week[3]=Thursday
Week[4]=Friday
Week[5]=Saturday
Week[6]=Sunday
```

- 範例說明：
  - 請注意第16行使用的是Week[i]，而非*Week[i]。
  - 讀者可以將範例修改為*Week[i]，看看會有什麼情況發生。

8.5 指標函式回傳值

- 由於函式最多只能回傳一個回傳值，所以一般來說，大多回傳『數值』即可，不過，在某些狀況下，我們會希望回傳一個指標，例如：想要回傳字串時。
  - 回傳指標必須在函式宣告與定義時，將回傳值宣告為指標型態，例如想要回傳一個字串指標可以宣告如下語法：

```
char *func1(引數串列);
```

  - 當然，如果您想要接收這個回傳的指標，也必須使用一個指標型態的指標變數來加以接收，我們直接使用一個範例來加以示範。

8.5 指標函式回傳值

- 【觀念與實用範例8-13】：設計一個反轉字串的函式，並且將反轉結果以字串指標回傳。
- 範例8-13：ch8_13.c

```
1  /*******************************************
2     檔名:ch8_13.c
3     功能:回傳字串指標(設計反轉字串函式)
4   *******************************************/
5   #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8  char *inverse(char *dest,char *src);
9   char *inverse(char *dest,char *src)
10 {
11  int i,len;
12   len=strlen(src);
13  for(i=len-1;i>=0;i--)
14     *(dest+len-1-i)=*(src+i);
15  *(dest+len)='\0';
16  return dest;
17 }
18 void main(void)
19 {
20  char *s1="Welcome";
21  char *s2;
22  s2=inverse(s2,s1);
23   printf("s1=%s\n",s1);
24  printf("s2=%s\n",s2);
25 }
```

8.5 指標函式回傳值

- 執行結果：

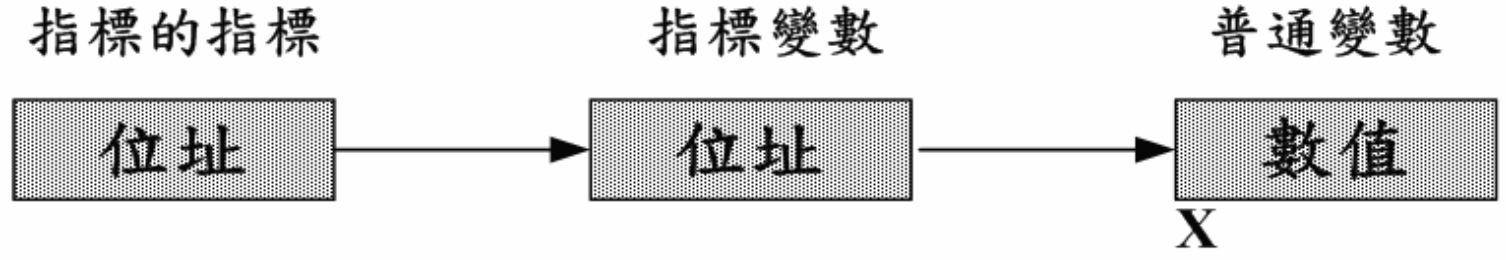
```
s1=Welcome
s2=emocleW
```

- 範例說明：
  - 第10行的函式宣告，將回傳值的型態宣告為指標型態（char *inverse）。第12行的函式定義也是如此。而我們模擬了標準函式庫提供的strcpy( )的引數宣告方式，將回傳值的dest也當做引數一起傳遞過去，您可以試著將引數dest刪除，再重新編譯看看，會發生什麼結果，您就知道為何strcpy( )的引數與回傳值需要重複了。
  - 第20行，使用return回傳dest指標。
  - 第27行，使用s2指標來接收dest指標。

8.6 『指標』的『指標』

- 指標除了可以指向普通變數，其實也可以指向指標變數，而在這種情況下，我們稱之為『指標的指標』，
- 例如指標pp指向指標p，而指標p指向資料X，則我們可以透過指標的指標，間接地存取『指標pp所指向的指標p再指向的內容』X（如下圖所示）。
- 除了指標的指標（兩層的指標變數）之外，我們

pp　　p



指標的指標示意圖

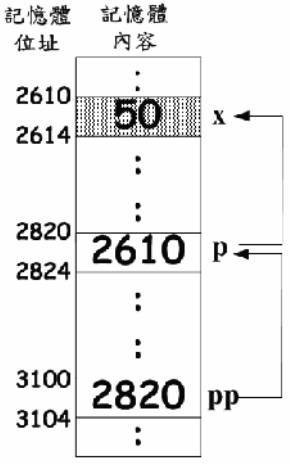
8.6 『指標』的『指標』

- 宣告多重指標，只需要在指標的「*」符號前面再加上一個以上的「*」即可，語法如下：

```
int *指標名稱;      /*  指標  */
int **指標名稱;     /*  指向指標的指標  */
int ***指標名稱;    /*  三層指標變數  */
```

- 假設我們使用了2重指標（指標的指標）來運算，則您還必須將第2重指標的位址指向第一重指標的位址，如下範例：

```
int x=50;
int *p;
int **pp;
p=&x;
pp=&p;
```



指標的指標記憶體示意圖

8.6 『指標』的『指標』

- 上面這個範例中，x=50、*p=50是毫無疑問的，而要使用『指標的指標』修改x的值，則必須使用兩次的指標提領運算子，換句話說，也就是**pp=50。而p是2610（x的位址），pp則是2820（p的位址）。
  - 多重指標也可以使用指標的基本運算，由於陣列名稱可以視為指標來運算，因此如果我們想要透過指標方式存取二維陣列的元素，則可以透過下列的加法運算完成：

陣列表示法	指標表示法
array[i][j]	*(*(array+i)+j)

8.6 『指標』的『指標』

- 【觀念範例8-14】：使用多重指標（指標的指標）改寫範例6-5，將九九乘法表。
- 範例8-14：ch8_14.c

```
1  /*****************************
2     檔名:ch8_14.c
3     功能:指標的指標
4   *****************************/
5  #include <stdio.h>
6  #include <stdlib.h>
7   void main(void)
8  {
9   int m[9][9];
10  int i,j;
11    for(i=1;i<=9;i++)
12    {
13      for(j=1;j<=9;j++)
14        *(*(m+(i-1))+(j-1))=i*j;
15    }
16  for(i=1;i<=9;i++)
17  {
18    for(j=1;j<=9;j++)
19    {
20      printf("%d*%d=%d\t",i,j,*(*(m+(i-1))+(j-1)));
21    }
22    printf("\n");
23  }
24 }
```

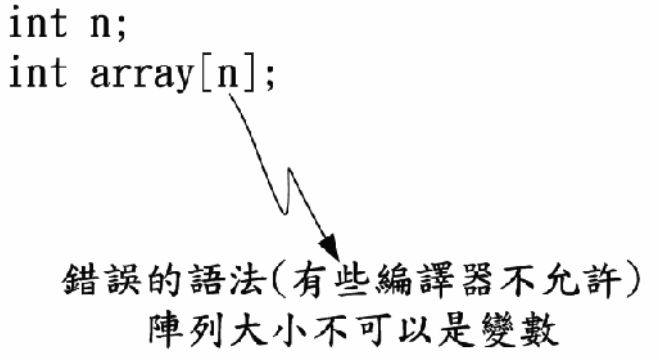
8.6 『指標』的『指標』

- 執行結果：

```
1*1=1  1*2=2  1*3=3  1*4=4  1*5=5  1*6=6  1*7=7  1*8=8  1*9=9
2*1=2  2*2=4  2*3=6  2*4=8  2*5=10 2*6=12 2*7=14 2*8=16 2*9=18
3*1=3  3*2=6  3*3=9  3*4=12 3*5=15 3*6=18 3*7=21 3*8=24 3*9=27
4*1=4  4*2=8  4*3=12 4*4=16 4*5=20 4*6=24 4*7=28 4*8=32 4*9=36
5*1=5  5*2=10 5*3=15 5*4=20 5*5=25 5*6=30 5*7=35 5*8=40 5*9=45
6*1=6  6*2=12 6*3=18 6*4=24 6*5=30 6*6=36 6*7=42 6*8=48 6*9=54
7*1=7  7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49 7*8=56 7*9=63
8*1=8  8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64 8*9=72
9*1=9  9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81
```

8.7 動態記憶體配置

- 讀者如果已經開始練習撰寫程式，相信一定會遇到某些陣列大小無法於事先決定的情況。
- 我們可以宣告一個陣列大小為變數的陣列嗎（例如：int array[n];）？答案是不行



- 有些人會將n先設定為非常大的值，例如200。而如果使用者只要求開出6個球，則剩餘的194個元素將被棄置不用。

8.7 動態記憶體配置

- 其實，最佳的解決辦法應該是利用C語言提供的動態記憶體配置函式來解決這個問題。

- C語言提供的動態記憶體配置malloc( )可以於執行過程中配置一個適當大小的記憶體空間給指標，接著我們就可以藉由這個指標來存取分配到的記憶體空間內的資料。

8.7.1 配置記憶體函式－malloc( )

- 我們可以透過malloc函式向系統要求配置一塊記憶體空間以供使用，語法如下：
  - malloc( )

```
標頭檔：#include <stdlib.h>
        #include <malloc.h>
語法：void *malloc(size_t size);
功能：動態配置記憶體
```

  - 【語法說明】：
    - malloc會配置size位元組的記憶體，並使用指標變數來表示該記憶體的開頭位址，因此必須使用一個指標變數來接收函式回傳指標。
    - 通常我們在計算size大小時，會配合sizeof( )函式來求出記憶體大小，例如您想要一個整數大小的記憶體空間，則size引數可設定為sizeof(int)。而由於各種不同資料型態的長度不盡相同，因此在接收回傳指標之前，常常需要先對回傳指標做資料轉型的動作。

8.7.1 配置記憶體函式－malloc( )

- 【範例】：配置一塊字元陣列（長度為9）的記憶體空間給指標變數。

```
char *ptr;
ptr = (char *)malloc(sizeof(char)*9);
```

- 範例說明：
  - (1)　sizeof(char)的大小為1，所以乘上9之後，恰為我們所要的9個位元組。
  - (2)　(char *)代表將回傳指標轉型為字元指標。

8.7.1 配置記憶體函式－malloc( )

- 【實用及觀念範例8-15】：改寫範例7-25，由於無法事先得知開球數目，因此使用動態記憶體配置，存放開球結果。
- 範例8-15：ch8_15.c

8.7.1 配置記憶體函式－malloc( )

```
1  /*****************************
2      檔名:ch8_15.c
3      功能:動態記憶體配置
4   *****************************/
5   #include <stdio.h>
6  #include <stdlib.h>
7  #include <malloc.h>
8  #include "./lotto.h"
9   /**************main()**************/
10 int main(int argc,char *argv[])
11 {
12  int i,special,ball_qty=6,temp;
13  int *lotto;
14   if(argc>1)
15  {
16     ball_qty=atoi(argv[1]);  /* atoi須引入stdlib.h */
17     if (ball_qty==0)
18     {
19        printf("參數錯誤,例如輸入球數非數字\n");
20        return -1;
21     }
```

8.7.1 配置記憶體函式－malloc( )

```
22      if (!((ball_qty>=1) && (ball_qty<=41)))
23      {
24          printf("參數錯誤,例如輸入球數非1~41\n");
25          return -1;
26      }
27  }
28  lotto=(int*)malloc(sizeof(int)*ball_qty);
29  generate_lotto_sort(&special,lotto,ball_qty);
30
31  printf("樂透號碼如下.....\n");
32  for (i=0;i<ball_qty;i++)
33  {
34      if ((i%6==0) && (i!=0))
35          printf("\n");
36      printf("%d\t",lotto[i]);
37  }
38  printf("\n");
39  printf("特別號:%d\n",special);
40  return 1;
41 }
```

8.7.1 配置記憶體函式－malloc( )

- 執行結果：

```
.....先編譯ch8_15.c，執行檔為ch8_15.exe.....
C:\C_language\ch08>ch8_15
樂透號碼如下.....
1          14         20         23         27         35
特別號:34
C:\C_language\ch08>ch8_15 10
樂透號碼如下.....
1          6          8          10         18         22
26         29         40         42
特別號:3
```

8.7.1 配置記憶體函式－malloc( )

- 範例說明：
  - lotto是一個整數指標變數。
  - 經過動態記憶體配置，lotto指向分配到的「ball_qty」個整數空間的開頭處。
  - 在執行結果中，當使用者輸入6球或未輸入球數參數時，lotto指向的記憶體空間大小為6個整數空間，完全不會浪費記憶體空間。
  - 經由malloc動態記憶體配置的記憶體空間，將會保留到程式執行完畢。
  - 如果要提早釋放這些記憶體空間，則必須使用下面所介紹的free函式。

8.7.2 釋放記憶體函式－free( )

- 我們透過malloc函式取得的記憶體，可以於不需再使用的狀況下，透過free函式將之歸還給系統，語法如下：
  - free( )

```
標頭檔：#include <stdlib.h>
        #include <malloc.h>
語法：void free(void *ptr);
功能：釋放記憶體
```

  - 【語法說明】：
  - free函式會釋放由ptr指標指向的記憶體空間，以便節省記憶體。

8.7.2 釋放記憶體函式－free( )

- 【實用及觀念範例8-16】：將範例8-15改寫，在往後不需要使用記憶體的狀況下，將lotto所指向的記憶體空間釋放。
- 範例8-16：ch8_16.c

8.7.2 釋放記憶體函式－free( )

```
1   /*************************
2       檔名:ch8_16.c
3       功能:釋放記憶體
4    *************************/
5
:   ...同範例8-15的第6~33行...
34   printf("樂透號碼如下.....\n");
35   for (i=0;i<ball_qty;i++)
36   {
37       if ((i%6==0) && (i!=0))
38           printf("\n");
39       printf("%d\t",lotto[i]);
40   }
41   free(lotto);
42   printf("\n");
43   printf("特別號:%d\n",special);
44   return 1;
45  }
```

8.7.2 釋放記憶體函式－free( )

- 範例說明：
  - 我們發現到，當第35~40行執行完畢後，就不再需要lotto指向的連續記憶體，因此將之釋放。換句話說，當您在42行以後想要讀取『lotto陣列』的資料，將再也讀不到剛才所開出的號碼，甚至還可能引起程式發生錯誤。雖然這個範例看不出free的優點，但在需要大量動態記憶體配置的程式中，如何有效管理記憶體空間，將是非常重要的課題，否則記憶體將很容易被一直配置到不夠使用，而發生錯誤。

8.8 本章回顧

- 在本章中，我們認識了C語言的一項特色，也就是允許透過『指標』來存取記憶體內容。本章重點如下：
  - 所有要被中央處理器處理的資料，都必須先存放在記憶體中；這些記憶體被劃分為一個個的小單位，並且賦予每一個單位一個位址，這個動作稱之為『記憶體空間的定址』。
  - 普通變數意味著佔用某一塊記憶體空間，該空間內則存放變數資料。『指標變數』與普通變數差不多，只不過指標變數的內容，是另一個變數的記憶體位址，換句話說，在記憶體空間內存放的是另一個變數所佔用的記憶體位址。

8.8 本章回顧

- 取得變數位址可以使用取址運算子『&』。
- 改變指標所指向記憶體位址的內容，可以使用提領運算子『*』。
- 每一種資料型態的指標變數在32位元的作業系統環境中都佔用4個bytes，因為指標變數存放的是記憶體位址。
- C語言提供下列四種與指標有關的基本運算功能：
  - 1. 指定運算　　2. 加減運算
  - 3. 比較運算　　4. 差值運算

8.8 本章回顧

- 為了避免指標指向不合法的位址而引發錯誤，我們可以在宣告指標變數時，同時設定指標指向一個合法的記憶體位址（如下語法）。下列語法中，『int *p=&a;』將被編譯器分解成『int *p;』與『p=&a;』來執行。

  - int a;
    int *p=&a;

- 程式語言的傳址呼叫在C語言中，以傳指標呼叫來加以實現。所以引數也可以是一個指標，以便達到呼叫者與被呼叫者共用同一塊記憶體的目的。

8.8 本章回顧

- 陣列名稱可以視為一個位址，代表著陣列在記憶體中開始的位址，您可以將之視為唯讀的指標來加以操作。字元字串是一種特殊的字元陣列，而指標字串則提供了更大的彈性，因此我們應該盡量將字串宣告為指標字串，例如：『char *string1;』。

- 在某些函式呼叫中，我們可能需要回傳一個指標，例如：想要回傳字串時。回傳指標必須在函式宣告與定義時，將回傳值宣告為指標型態，例如想要回傳一個字串指標可以宣告如下語法：
  - char *func1(引數串列);

8.8 本章回顧

- 指標也可以指向指標變數，而在這種情況下，我們稱之為『指標的指標』，透過指標的指標，我們就可以間接地存取『指標所指向的指標再指向的記憶體內容』。
- 為了更有效利用記憶體空間，我們可以使用C語言提供的動態記憶體配置函式。
- C語言提供的動態記憶體配置函式有malloc與free。
- 動態記憶體配置函式可以於執行過程中配置一個適當大小的記憶體空間給指標，接著我們就可以藉由這個指標來存取分配到的記憶體空間內的資料。

本章習題