國立屏東大學 資訊工程學系 程式設計(二)

13. 指標(Pointers)

指標(Pointers)往往是許多同學在C語言的學習過程中，所遇到的第一個巨大的障礙。其實，只要將觀念建構清楚，指標其實一點都不困難。本章將就指標的基本觀念與相關語法進行說明。

13.1 基本觀念

要學習指標就必須先瞭解記憶體與變數的關係。在開始之前，先讓我們回顧一下在第5章所談到的變數與記憶體位址的概念：

<note>

摘錄第5章部份內容

事實上，電腦系統在執行程式時，所有的資料都是存放在記憶體當中。… 可是記憶體空間是由作業系統負責管理的，每當有程式要求執行時，作業系統的<html>動態記憶體配置(或稱動態記憶體管理)模組</html>會動態地分配一塊空間給該程式使用，但在絕大多數的情況下其所分配到的位置都不相同(當然也有相同的可能，只是機率比較低)。… 具體的做法是，先告訴電腦我們需要一個記憶體位置來存放一個整數，並且為了方便管理程式中還可能會需要的其它記憶體位址，我們必須為這個(需要空間來存放的)整數取個名字，例如以下的宣告：

```
int n;
```

上述的程式碼，稱為<html>變數宣告(variable declaration)</html>表達了我們需要一個記憶體空間來存放一個稱為n的整數。在程式設計的術語裡，這個整數n被稱為<html>變數(variable)</html>

換句話說，以上的宣告會要到一塊記憶體空間來存放變數n當程式執行時，會產生一個稱為symbol table的表格，用來記錄所有的變數所分配到的記憶體空間，例如

symbol	type	address
n	int	100

當變數宣告完成後，我們可以使用&運算子來表達變數所在的記憶體位址，例如變數n所存放的位置，可以在程式碼中以&n來表示。在前述的例子中，變數n的記憶體位置為100，意即&n為100。所以當我們以scanf("%d", &n)來取得一個整數時，該整數的值(value)將會被存放到記憶體編號100的位置。假設使用者輸入的整數是3，那麼：從程式的角度來看，存放在記憶體位址100的變數n的值為3，也就是n=3&n=100...

再強調一次，在程式碼中n代表一個變數&n則代表這個變數的數值所存放的記憶體位址。

</note>

再回顧一下，在第6章中，我們針對變數與記憶體的關係，所進行的說明。

<note>

摘錄第6章部份內容

由於程式設計的需要，我們必須在程式執行的階段利用記憶體空間來存放一些資料，並且可以對這些資料進行運算與處理。我們將這些在程式執行階段所使用到的資料項目稱為變數(variable)。C語言要求所有的變數必須先宣告後才能使用。假設我們需要在程式中處理一個整數的運算，因此我們必須先進行整數變數的宣告：

```
int x;
```

上述的程式碼宣告了一個名稱為x的變數，在程式執行時，作業系統會分配足以放置整數的記憶體空間供它使用。所謂的足夠的空間指得是存放一個整數所需的空間，其大小視作業系統而定，通常是32bits。也就是4個bytes。

<note tip>若要知道您所在的作業系統，使用多少空間來存放一個整數，可以使用sizeof(int)。它會傳回一個整數所佔的記憶體空間大小(其單位為byte)。</note>

讓我們假設程式開始執行，並且變數x所需的記憶體空間被配置在0x7ffff34fff00記憶體位址，那麼程式的symbol table內容為：

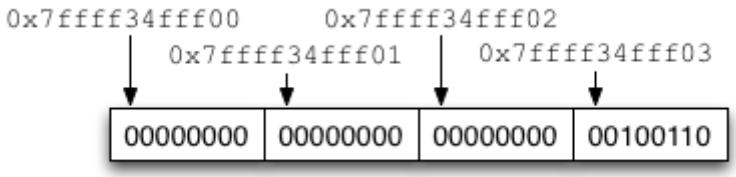
symbol	**type**	**address**
x	int	0x7ffff34fff00

當中記載了一個名為x的符號，其型態為int。並且存放於0x7ffff34fff00位址起始的連續四個bytes(因為已經記載了其型態為int。所以系統可以知道其佔了四個bytes)。接著假設我們執行以下的程式碼：

```
x=38;
```

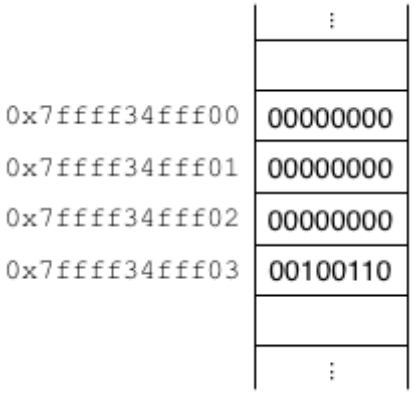
系統會檢視符號x所在的記憶體空間與其型態，然後將數值38放入對應的記憶體空間內，請參考Fig 1。

Fig 1.



由於一個整數假設為32bits。也就是四個bytes。所以我們會將0x7ffff34fff00~0x7ffff34fff03的記憶體空間，以38的二進位100110來儲存。通常描述記憶體空間的圖有兩種畫法，其一為figure ##的橫式，或是Fig 2的直式，兩者的意義相同。

Fig 2.



<note tip>記憶體位址的單位為byte且通常以16進位來表示0x7fff34fff00開頭處的0x就表示數值為16進位 </note>

除卻這些細節，通常我們在設計C語言程式時，只需要抽像化地將變數x表達為記憶體中的某塊位置，不用特別去注意其所在的記憶體位置為何Fig 3就是我們常用的思考方式。

Fig 3.



如第5章所說明的，如果我們想要知道一個變數到底存放在哪個記憶體位址，可以使用&運算子來取得。請參考以下的程式，它宣告了一個整數變數，並將其值與記憶體位址加以輸出。

h variableAndAddress.c

```
#include <stdio.h>

int main()
{
  int x;

  x=38;

  printf("The value of x is %d.\n",x);
  printf("The memory address of x is %p\n", &x);
  return 0;
}
```

<note tip> 在printf()函式中，如要輸出記憶體位址，其format specifier為%p </note>

</note>

13.2 指標變數(Pointer Variables)

指標變數，顧名思義即為一個變數，但其所儲存的不是數值(value)而是某個記憶體的位址(memory address)，或者說其所儲存的值為某個記憶體的位址。其宣告方法等同一般的變數宣告，但在變數名稱前，須加入一個 “ * ” 號。

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

以上這三個宣告的結果都相同，都會建立一個 “ 應該會 ” 指向某一個儲存整數的記憶體位址，C語言也允許我們混合一般的變數宣告與指標變數宣告，請參考下面的這些宣告:

```
int *x, y;      // x是指標變數, y是一般變數
int i, *j;      // i是一般變數, j是指標變數
int* i,j, k;    // i是指標變數, j與k則是一般變數
```

在前述的例子中，指標變數都被宣告為 “ 應該會 ” 指向儲存整數(int)的記憶體位址，所以上述的宣告可視為宣告了一些 “ 整數的指標變數 ” 。一般而言，我們常將指標變數的型態稱為參考型態(referenced type)，C語言允許我們將指標宣告為各種型態，換言之，C語言在指標的參考型態上並無限制，例如下面這些都是可行的宣告:

```
double *p;
char *p;
float *p;
```

請先執行下列程式，以確認你的系統上的記憶體使用情形：

```
int main()
{
  printf("The size of an int is %d.\n", sizeof(int));
  printf("The size of an int-pointer is %d.\n", sizeof(int *));
  return 0;
}
```

假設我們所得到的執行結果分別為4與8，由於其單位為byte，表示一個int的整數佔用32位元的記憶體空間，而一個整數指標佔用64位元。

<note tip> 以64位元進行定址的系統，其可使用的記憶體空間最大支援16EB (註1000GB = 1TB, 1000TB = 1PB, 1000PB = 1EB) (EB = Exabyte, 唸做艾克薩 - 拜) </note>

圖figure 1顯示了一個int整數x與整數指標p(它是由 int *p加以宣告)，假設整數變數x是儲存於記憶體位址中的0x7ffff34fff00 - 0x7ffff34fff03 (32位元的整數佔四個bytes)，其值為38，且假設整數指標p儲存於0x7ffff34fff68-0x7ffff34fff6f (8 bytes=64 bits)，其值為0x7ffff34fff00。

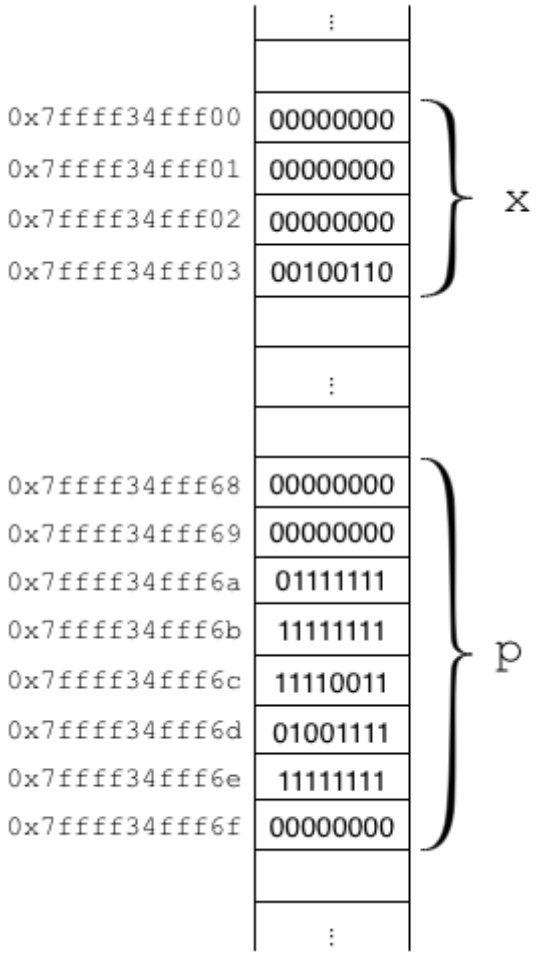


Fig. 1

若不考慮記憶體位址的細節，我們可以將figure 1表示為figure 2。



Fig. 2

在圖中p是一個指向0x7ffff34fff00位址的指標，因為當初我們有宣告p的參考型態為int，所以C語言的compiler會知道，p所指向的是位於0x7ffff34fff00 - 0x7ffff34fff03的四個bytes，從抽象角度來看，由於p所儲存的值，正好是x所在的位址，因此我們更常用figure 3來加以表達這樣的一個關係。

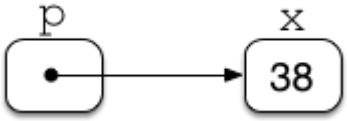


Fig. 3

還記得我們在前面的章節中，曾提過一個變數可以使用“ & “ 運算子來取得其所再的記憶體位址嗎？我們可以透過下面的程式碼，來將整數變數x的位址，指定給整數指標p，如此一來，就如同figure 3一樣，p成為了指向整數變數x的指標。

```
int x=38;
int *p;

p=&x;
```

13.3 記憶體位址與間接存取運算子

C語言提供 ” & “ 運算子，來取得變數所在的記憶體位址，以及 ” * ” 來間接存取指標變數所指向的記憶體位址中的值。例如,

```
int x=38;
int *p;

printf("x is located at %p.\n", &x);
```

其結果會輸出x所在的記憶體位址，且下面的程式碼，會讓p指向x所在的位址，並且將其值輸出。

```

p=&x;  // 使p指向x所在的位址

printf("The value of *p is %d.\n", *p);
```

當我們以 p = &x; 將x的所在位址指派給p時，p就好比是x的別名一樣，我們可以在程式碼中使用x或是透過p來存取同一個記憶體位址的值。思考下面的程式碼片段，其執行結果為何？

```
int x, *p;

p = &x;

x = 6;
printf("%d %d\n", x, *p);

*p = 8;
printf("%d %d\n", x, *p);
```

再思考下列的程式碼：

```
int x,y=5;

x = *&y;

y = &x;

x = *y;
```

13.4 指標指派(Pointer Assignment)

C語言允許兩個相同參考型態的指標，彼此間進行值的指派，當然其值所代表的是記憶體位址。考慮下面的程式碼，其執行結果為何？

```
int x=5, y=8, *p, *q;

p = &x;
q = &y;

printf("*p=%d *q=%d\n", *p, *q);
```

再考慮以下的程式，其執行結果又為何？

```
int x=5, y=8, *p, *q;

p = &x;
q = &y;

printf("x=%d y=%d *p=%d *q=%d\n", x, y, *p, *q);

*p = *q;

printf("x=%d y=%d *p=%d *q=%d\n", x, y, *p, *q);
```

再考慮以下的程式，其執行結果又為何？

```
int x=5, y=8, *p, *q;

p = &x;
q = &y;  // 請參考Fig. 4

printf("x=%d y=%d *p=%d *q=%d\n", x, y, *p, *q);

p = q;   // 請參考Fig. 5

printf("x=%d y=%d *p=%d *q=%d\n", x, y, *p, *q);
```

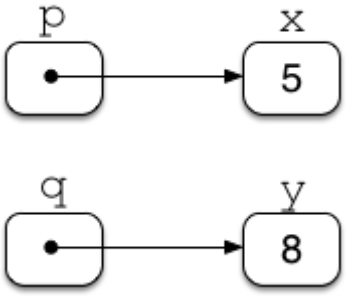


Fig. 4

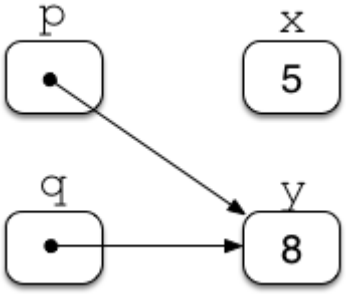


Fig. 5

13.5 指標與函式

13.5.1 以指標做為函式引數

在一般的情況下，一個C語言的函式(function)可以接受多個引數(arguments)並經計算後傳回一個單一的值，但若要讓函式傳回多個數值，就無法做到了。假設我們需要設計一個函式，接受一個double數值做為引數，並將其整數與小數部份分別傳回，則可以考慮以下的做法：

```
void decompose(double val, long *int_part, double *frac_part)
{
   *int_part = (long) val;
   *frac_part = val - ((long)val);
}
```

這個函式decompose並沒有任何的傳回值，可是它接受了兩個指標做為其引數，並在其計算過程中，透過*間接存取運算子進行相關的計算，所以其計算的結果過直接存放在這兩個指標所指向的記憶體位址內。如此一來，就可以做到讓一個函式可以傳回(事實上並沒有傳回的動作了)一個以上的數值。

要注意的是，函式的原型可以使用下列兩者之一進行宣告：

```
void decompose(double val, long *int_part, double *frac_part);
void decompose(double, long *, double *);
```

下面的程式碼展示了呼叫decompose的方法：

```
double d = 3.1415;
int i;
double f;

decompose(d, &i, &f);
```

13.5.2 以指標做為函式的傳回值

除了可以使用指標做為函式的引數，我們也可以使用指標做為函式的傳回值，請參考以下的範例：

```
int *max(int *a, int *b)
{
   if( *a > *b)
      return a;
   else
      return b;
}
```

在呼叫時，要注意必須要以一個整數的指標來接收函式的傳回值:

```
int x, y;
int *p;

p = max( &x, &y);
```

或者，以下列的方法取回函式執行後傳回的指標，並透過間接存取，直接存取該指標所指向的位址裡的值。請參考下面的程式：

```
#include <stdio.h>

int *max(int *a, int *b)
{
  if(*a > *b)
    return a;
  else
    return b;
}

int main()
{
  int x=5, y=10;
  int *p;

  p = max(&x, &y);
  printf("The maximum value is %d.\n", *p);
```

```
  *max(&x, &y)=100;

  printf("The values of x and y are %d and %d.\n", x, y);

  return 0;
}
```

13.6 Call by Value與Call by Address

所謂的Call by Value是指當呼叫函式時，所傳入的是數值；同理，若傳入的是記憶體位址就叫Call by Address下面兩個程式分別就Call by Value與Call by Address做一示範：

```
void swap(int x, int y)
{
   int temp=x;
   x=y;
   y=temp;
}

int main()
{
   int a=5, b=10;
   swap(a,b);
   printf("a=%d b=%d\n", a, b);
}
```

```
void swap(int *x, int *y)
{
   int temp=*x;
   *x=*y;
   *y=temp;
}

int main()
{
   int a=5, b=10;
   swap(&a,&b);
   printf("a=%d b=%d\n", a, b);
}
```

From:
https://junwu.nptu.edu.tw/dokuwiki/ - **Jun Wu的教學網頁**
國立屏東大學資訊工程學系
CSIE, NPTU
Total: 339281

Permanent link:
https://junwu.nptu.edu.tw/dokuwiki/doku.php?id=c:point

Last update: **2019/07/02 15:01**