國立屏東大學 資訊工程學系 程式設計(二)

16. 指標與陣列

在C語言中，因為陣列是記憶體中連續的一塊空間，因此我們也可以透過指標來存取儲存在陣列中的資料。本章將說明如何使用指標來存取陣列中的元素，並進一步探討指標與陣列的關係。

16.1 指標運算與陣列

考慮以下的程式碼：

```
int data[10]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

int i;

printf("data at %p.\n", data);
for(i=0;i<10;i++)
{
   printf("data[%d] at %p. \n", i, &data[i]);
}
```

其執行結果為：

```
data at 0x7fffb0aad1e0.
data[0] at 0x7fffb0aad1e0.
data[1] at 0x7fffb0aad1e4.
data[2] at 0x7fffb0aad1e8.
data[3] at 0x7fffb0aad1ec.
data[4] at 0x7fffb0aad1f0.
data[5] at 0x7fffb0aad1f4.
data[6] at 0x7fffb0aad1f8.
data[7] at 0x7fffb0aad1fc.
data[8] at 0x7fffb0aad200.
data[9] at 0x7fffb0aad204.
```

上述的程式碼宣告並在記憶體內配置了一塊連續十個整數的陣列，並且將其位址配置印出。由於陣列是連續的空間配置，在上面的例子中，data位於0x7fffb0aad1e0，那麼其第一筆資料(也就是data[0])就是位於同一個位址，或者更詳細的說，是從0x7fffb0aad1e0~3這四個記憶體位址。由於一個位址代表一個byte，四個bytes就剛好表示一個32位元的整數。其後的每筆資料也都剛好間隔4個bytes。

我們可以用一個指標指向陣列的第一筆資料：

```
int *p;

p = &data[0];
```

或是

```
p = data;
```

或者

```
p = &data;  //這行會有編譯的警告，因為p應該要指向一個整數，但data其實只是一個陣列所在的記憶體位址

p = (int *)&data; //這樣就沒問題了
```

上述的程式碼宣告了一個整數指標p指向陣列所在之處，如figure 1。

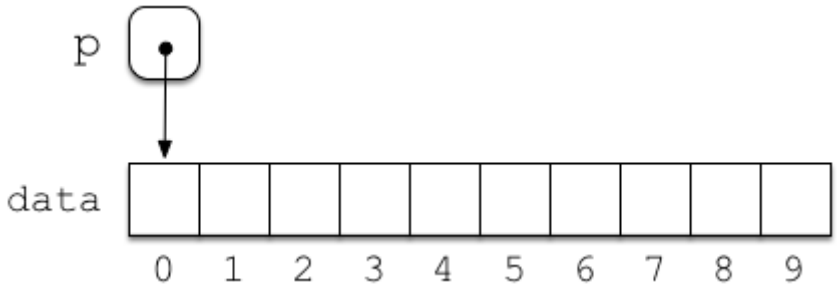

Fig. 1

我們可以透過*p來存取data[0]的數值資料。　如果我們要使用指標p來存取陣列中別的元素，例如第3筆資料(也就是data[2])則可以用：

```
printf("%d\n", *(p+2) ); //這行程式中的*(p+2)就等於 data[2];
```

考慮到p目前指向的是0x7fffb0aad1e0也就是data[0]所在的位址p+2並不會等於0x7fffb0aad1e0+2而是等於0x7fffb0aad1e0 + 2*sizeof(int)因為操作的對象p是一個指向整數的指標，因此若對它進行加法的運算，其運算單位是以整數的大小為依據，所以p+2會等於0x7fffb0aad1e8這也就是data[2]所在的位址。

指標除了可以進行加法的運算外，也可以進行減法的運算：

```
int *p;
int *q;

p=&data[5];
q = p-2;  // q現在指向data[3]
p -= 3;   // p現在指向data[2]
```

前面提到，指標的運算是以其參考型態的大小為依據。當運算元皆是指標時，其運算結果也是會轉換為其參考型態的大小間的差距：

```
int i;
p = &data[5];
q = &data[2];

i = p - q; // i的值為3
i = q - p; // i的值為-3
```

指標還可以使用relational operators(包含<, < =, ==, >=, >, !=)進行比較，依指標值其比較結果可以為0(不成立時)或1(條件成立時)。

```
p = &data[5];
q = &data[2];

if( p > q )
   printf("The position of p is after that of q.\n");
if( *p > *q)
   printf("The value of p is larger than that of q.\n");
```

16.2 以指標走訪陣列

下面的程式，配合迴圈的使用，利用指標來將陣列中每個元素都拜訪一次：

```
#include <stdio.h>

int main()
{
  int data[10]={1,2,3,4,5,6,7,8,9,10};

  int i;
  int *p;

  p=&data[0];
  for(i=0;i<10;i++)
    printf("data[%d]=*(p+%d)=%d \n", i, i, *(p+i));

  return 0;
}
```

當然，我們也可以直接用指標來操作:

```
#include <stdio.h>

#define Size 10
```

```
int main()
{
  int data[Size]={1,2,3,4,5,6,7,8,9,10};

  int i;
  int *p;

  p=&data[0];
  for(p=&data[0];p<&data[Size];p++)
    printf("%d \n", *p);

  return 0;
}
```

改用while迴圈，並配合「++」更新指標：

```
 p=&data[0];
  while (p<&data[Size])
  {
    printf("%d \n", *p++);
  }
```

16.3 指標與陣列互相轉換使用

我們也可以直接把陣列當成一個指標來使用：

```
#include <stdio.h>

#define Size 10

int main()
{
  int data[Size]={1,2,3,4,5,6,7,8,9,10};
  int i;

  for(i=0;i<Size;i++)
    printf("%d\n", *(data+i) );

  printf("\n");

  int *p;
  for(p=data;p<data+Size;p++)
    printf("%d \n", *p);

  return 0;
}
```

當然，也可以指標當成陣列來使用：

```
  int data[Size]={1,2,3,4,5,6,7,8,9,10};
  int *p = data;
  int i;

  for(i=0;i<Size;i++)
    printf("%d\n", p[i]);

  printf("\n");

  int *p;
  for(p=data;p<data+Size;p++)
    printf("%d \n", *p);

  return 0;
}
```

16.4 常見的陣列處理

本節以指標操作一些常見的陣列處理：

```
#include <stdio.h>

#define Size 10

int main()
{
  int data[Size]={1,2,3,4,5,6,7,8,9,10};

  int i;
  int *p;
  int sum=0,sum2=0;

  for(p=&data[0];p<&data[Size];p++)
  {
    sum += *p;
  }
  printf("sum = %d\n", sum);

  while(p>=(int *)&data)
  {
    sum2+= *p--;
  }
  printf("sum2 = %d\n", sum2);
```

```
  return 0;
}
```

```
#include <stdio.h>

#define Size 10

int main()
{
  int data[Size]={321,432,343,44,55,66,711,84,19,610};

  int i;
  int *p;
  int max, max2;

  max = *(p = &data[0]);
  for( ; p<&data[Size]; p++)
     max = (max < *p) ? *p : max;
/*  {
    if( max < *p)
      max = *p;
  }*/

  printf("max = %d \n", max);

  return 0;
}
```

```
#include <stdio.h>

#define Size 10

int main()
{
  int data[Size]={3451,25,763,3454,675,256,37,842,3439,510};

  int *p, *q;

  for(p=&data[0];p<&data[Size-1];p++)
    for(q=p+1; q < &data[Size]; q++)
      if(*p<*q)
      {
        int temp = *p;
        *p = *q;
        *q = temp;
      }
```

```
  for(p=&data[0];p<&data[Size];p++)
    printf("%d \n", *p);

  return 0;
}
```

16.5 以陣列做為函式的引數

在函式的設計上，可以陣列做為引數。請參考下面的例子：

```
int sum( int a[], int n)
{
   int i=0, s=0;
   for(;i<n;i++)
   {
      s+=a[i];
   }
   return s;
}
```

在上面的例子中，我們設計了一個可以計算陣列中元素和的函式。我們可以下面方式呼叫此函式：

```
int data[10]={12,522,43,3423,23,21,34,22,55,233};
int summation = 0;
...

summation = sum(data,10);
```

由於陣列與指標可互相轉換的特性，前述的函式也可改成：

```
int sum( int *a, int n)
{
   int i=0, s=0;
   for(;i<n;i++)
   {
      s+=a[i];
   }
   return s;
}
```

同樣地，在呼叫時也可以用下列的方法：

```
summation = sum( &data[0], 10);
```

```
或者
summation = sum( (int *)&data, 10);
```

另外，也可以使用

```
summation = sum( &data[3], 5 );
```

來計算從陣列第4筆元素開始，往後5筆的和。

16.6 指標與多維陣列

本節以二維陣列為例，探討指標與多維陣列的關係。請參考以下的例子：

```
#include <stdio.h>

#define ROW 3
#define COL 5

int main()
{
  int data[ROW][COL];

  int i, j;

  for(i=0;i<ROW;i++)
    for(j=0;j<COL;j++)
      data[i][j]= i*COL+j;


  for(i=0;i<ROW;i++)
  {
    for(j=0;j<COL;j++)
    {
      if(j>0)
        printf(", ");
      printf("%3d", data[i][j]);
    }
    printf("\n");
  }

  return 0;
}
```

在這個例子中，我們宣告了一個ROWxCOL(3×5)的二維陣列，給定其初始值後將陣列內容輸出。一般而言，我們可以將二維陣列視為一個二維的表格，如figure 2所示。

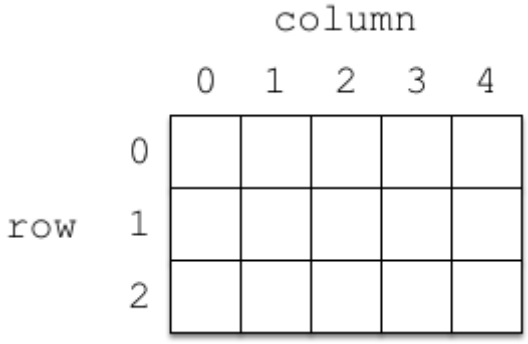

Fig. 2

我們可以宣告一個指標，來存取這個二維陣列，例如：

```
int (*p)[COL];

i = 0;

for(p=&data[0]; p<&data[ROW] ; p++)
{
  for(j=0;j<COL;j++)
  {
    (*p)[j] = i*COL+j;
  }
  i++;
}
```

但其實在記憶體的配置上，仍是以連續的空間進行配置的，如figure 3所示。

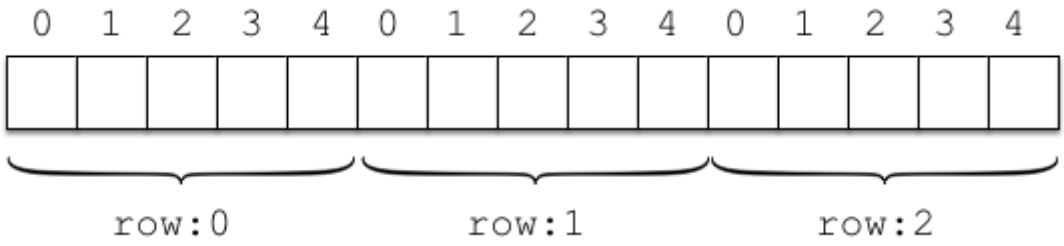

Fig. 3

請試著寫一個程式，輸出一個二維陣列各個元素的記憶體位址，來驗證上述的說法。

所以，同樣的初始值給定的程式碼也可以改寫如下：

```
int *p;

for(i=0, p=&data[0][0]; p<= &data[ROW-1][COL-1];i++, p++)
  *p = i;
```

如果要單獨取出二維陣列中的第i列(row)那麼可以下列程式碼完成：

```
p = &data[i][0];
```

或者

```
p = data[i];
```

但是若要取回二維陣列中的某一行(column)那麼又該如何呢？留給同學做練習吧！

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: 196499

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

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