Chapter 6. Pointers

- Address and pointers
- Pointers and Reference
- Array and Pointers
- Pointers and Function
- Dynamic Memory Allocation

1. Address and pointers

- Pointers（指標）
  - ➢Store address of variables.
  - ➢Permit the construction of dynamically link list.
  - ➢Access directly the address of variables and manipulate them
    - ☞add, subtract, compare….
  - ➢ float A=2.5;

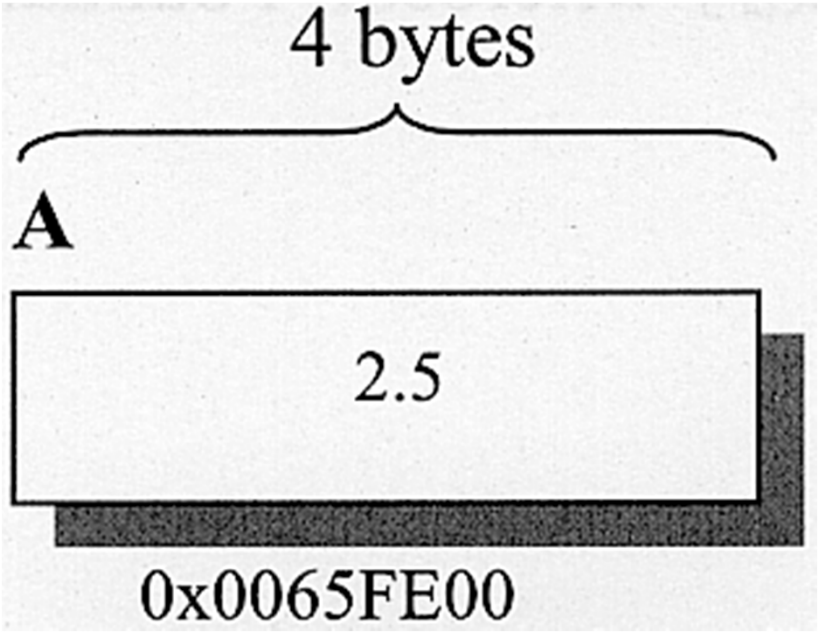
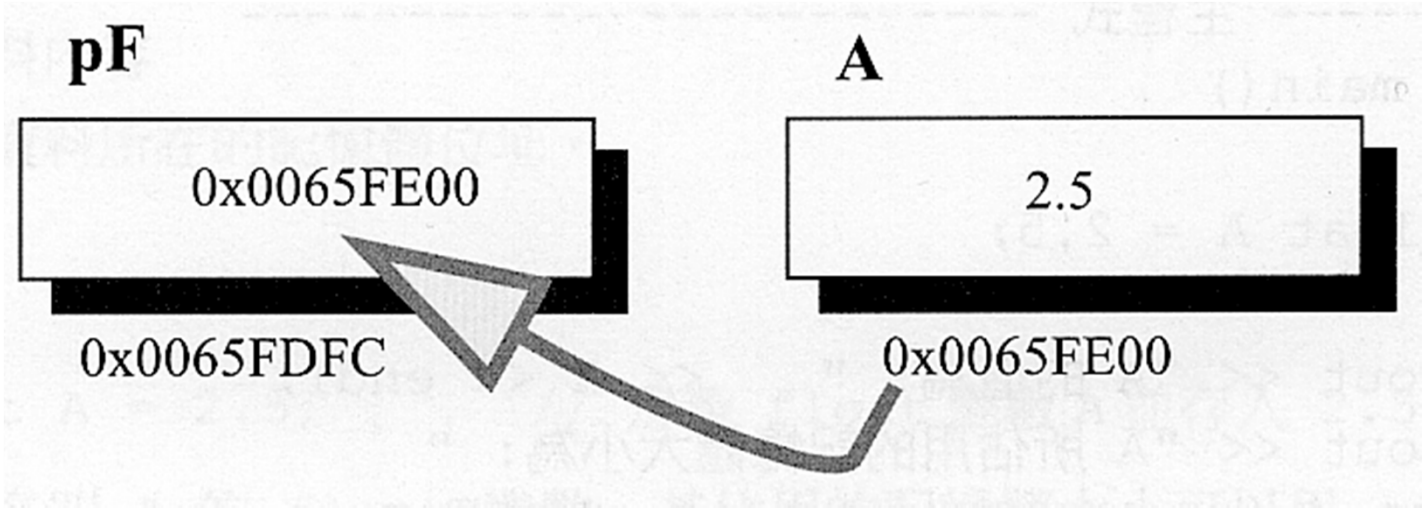


圖 8.1.1 變數 A 的內容和位址

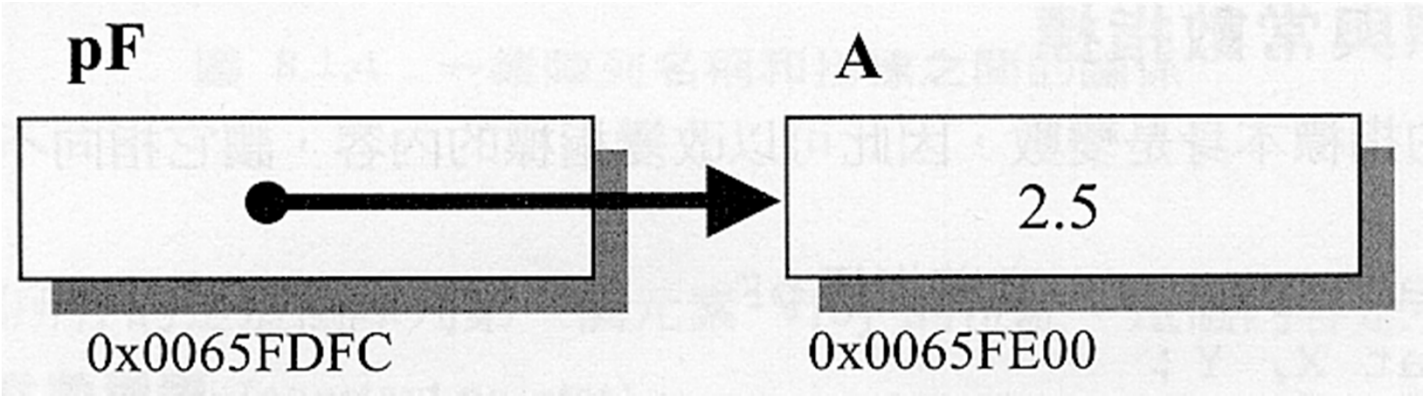
- float A=2.5;

  float *pF; ⟵ pointer

  pF=&A; ⟵ 將A的位置存入指標pF




```
名稱:          pF
資料型態:      float *
資料內容:      0x0065FE00
記憶體位址:    0x0065FDFC。
```



Program 6.1.1

```
#include <iostream>
using namespace std;

void main()
{int a=2, *p;

 cout<<"&a="<<&a<<", a="<<a<<endl;
 p=&a;
 cout<<"&p="<<&p <<", *p="<< *p<<", p="<<p<<endl;
}
```

```
&a=0012FF7C, a=2
&p=0012FF78, *p=2, p=0012FF7C
```

- 指標存放的型態要一致
  - wrong example !!

```
float a=2.5;
int  *p;

p=&a;
```

- float* p; //或 float *p;
  - float* p1, p2;
    - p1被宣告為pointer 但 p2是float
  - float *p1, *p2;
    - p1及p2被宣告為pointer
- wrong example !!

```
int  *p;

*p=20;
```

→ 尚未給定所指之位置!! Compile不會出錯, 但執行程式會出問題

- Example

```
float *p,a=2.5,x, *p1;
p=&a;
x=*p;
*p=5.5;
p=&x;
p1=&x;
```

p指向a

將指標p所指記憶體內的資料存到變數x

將指標p所指記憶體內的資料改變為5.5

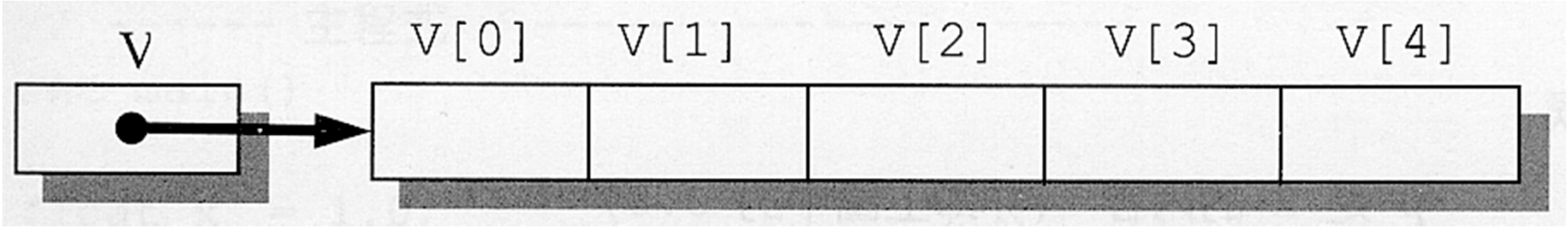
也就是a的值也被改變為5.5

將p指向x

將p1指向x

- array本身也是指標且是常數指標(constant pointer)

double v[5];



Program 6.1.2

```
#include <iostream>
using namespace std;

void main()
{int a=2,x,b[4]={1,2,3,16}, *p;

 cout<<"&a="<<&a<<", a="<<a<<endl;
 p=&a;
 cout<<"&p="<<&p<<", p="<<p<<endl;
 x=*p;
 *p=5;
 cout<<"&a="<<&a<<", a="<<a<<endl;
 cout<<"&x="<<&x<<", x="<<a<<endl;
 p=&x;
 cout<<"&p="<<&p<<", p="<<p<<endl;
 cout<<"&b="<<&b<<", b="<<b<<endl;
}
```

```
&a=0012FF7C, a=2
&p=0012FF64, p=0012FF7C
&a=0012FF7C, a=5
&x=0012FF78, x=2
&p=0012FF64, p=0012FF78
&b=0012FF68, b=0012FF68
```

2. Pointers and Reference

- 取址運算子(address operator) &
  - float y,z;
    float &x=y; //宣告x是y的參照,即x所對應的記憶體位址與y所對應的記憶體位址相同
  - 不可單獨宣告
    float &x; 錯誤的宣告!!
  - 不能更改參照的對象
    float y,z;
    float &x=y;
    float &x=z; 錯誤的宣告!! 不能更改參照的對象
  - Pointer宣告並給初始值
    int a;
    int* p=&a; // 或 int *p=&a;

Program 6.2.1

```
#include <iostream>
using namespace std;

void main()
{
 float x  = 1.0;
 float &y = x,*p = &x;

 cout << "x 原來的值是 " << x << endl;
 *p=5.0;
 cout << "執行 *p= 5.0; 後\n";
 cout << "x 的值是     " << x << endl;
 y = 7.3;
 cout << "執行 y = 7.3; 後\n";
 cout << "x 的值是     " << x << endl;
}
```

```
x 原來的值是 1
執行 *p= 5.0; 後
x 的值是     5
執行 y = 7.3; 後
x 的值是     7.3
```

3. Array and Pointers

- 陣列元素的位址
  - const int m=5;

    double A[m];
    - ☞ 第k+1個元素(A[k])的位址為

      &A[0]+k*sizeof(double)
  - 二維陣列

    const int m=20;

    const int n=60;

    double B[m][n];
    - ☞ 元素B[i][j]的位址為

      &B[0][0]+(i*n+j)*sizeof(double)

- 三維陣列 20×60×80

```
const int m=20;
const int n=60;
const int p=80;
double C[m][n][p];
```

  - 元素C[i][j][k]的位址為
    &C[0][0][0]+(i*n*p+j*p+k)*sizeof(double)

- 用pointer來存取陣列的元素
  - 指標的內容可以接受加減乘除四則運算以及關係運算
    - int A[10];
      int *p;
      - p=&A[0];或p=A;都可將A[0]的位址存到指標變數p
      - *(p+i) 相當於A[i]

指標與取值運算子、增減運算子常結合在一起，寫成很精簡的敘述。
我們將這些操作的語法歸納成下表：

*++p	相當於	*(++p)	先增加指標再取用元素
*p++	相當於	*(p++)	先取用元素再增加指標
*--p	相當於	*(--p)	先減少指標再取用元素
*p--	相當於	*(p--)	先取用元素再減少指標

- 二維陣列的指標
  - const int m=2;
    const int n=3;
    double B[m][n];
  - 二維陣列可看成是由“列”為單位元素所組成的向量，期中各“列”本身又是由很多元素組成的向量

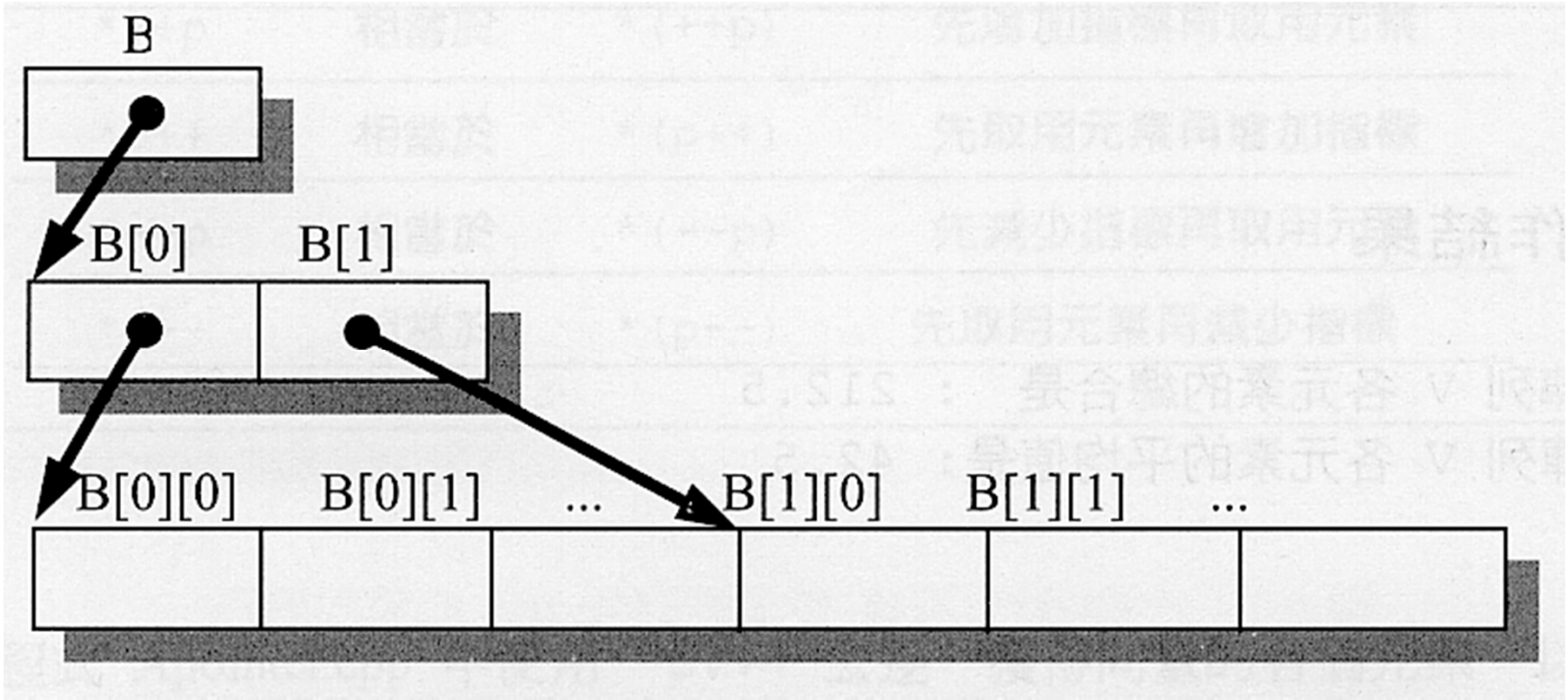


表 8.3.1 二維陣列的元素表示法

下標表示法	指標表示法 (1)	指標表示法 (2)
B[0][0]	*B[0]	*(*B)
B[0][1]	*(B[0]+1)	*(*B+1)
B[0][2]	*(B[0]+2)	*(*B+2)
B[1][0]	*B[1]	*(*(B+1))
B[1][1]	*(B[1]+1)	*(*(B+1)+1)
B[1][2]	*(B[1]+2)	*(*(B+1)+2)

通式:

B[i][j] : *(B[i]+j) 或 *(*(B+i)+j)

4. Pointers and Function

- Call function
  - X=fun(a,b);
    - fun 的 prototype 可能是下列兩種之一
      - float fun(float, float);
      - float fun(float &, float &);
  - X=fun(&a,&b); //呼叫fun,參數用a跟b的位址
    - fun 的 prototype
      - float fun(float *, float *);

  - float v, p[5];

    v=average(p);

- ☞ average 的 prototype
  - ■ float average(float []);
  - ■ float average(float *);

- Function pointers
  - ➢ function本身就是常數指標,指向函式自己
    - ☞ double func(int); //func的prototype
      double (*p)(int); //定義函式指標p
      p=func; //將函式指標p指向函式func
      - ■ 上面兩式也可合併為
        double (*p)(int)=func;//定義函式指標p並指向函式func
      - ■ cout<<p(6)<<endl;

Program 6.4.1

```
#include <iostream>
using namespace std;
int test(int ,int);

void main()
{int i,j;
 int (*p)(int,int);
 int *a=&i;

 i=1;j=2;
 p=test;
 *a=p(i,j);
 cout<<i<<endl;
}

int test(int a,int b)
{
 return a*b+2*b;
}
```

宣告test這函式的prototpye

宣告p為函式指標

宣告a為pointer並指向i

將函式指標p指向test函式

將p函式指標執行的輸出結果存入到指標a,也就是存到i

- 函式以函式指標的方式傳遞
  - double func1(float); //函式func1的原型
    double func2(float); //函式func2的原型
    double test(double (*func)(float),float); //函式test的原型

    a=test(func1,b); //call test函式且其對應的函式是func1

Program 6.4.2

```
#include <iostream>
using namespace std;
int test1(int ,int);
int test2(int,int);
int test(int (*fun)(int, int),int,int);

void main()
{int i,j;
 int (*p)(int,int);
 int *a=&i;

 i=1;j=2;
 p=test1;
 *a=test(test1,i,j);
 cout<<i<<endl;
 *a=test(p,i,j);
 cout<<i<<endl;
}
```

```
int test1(int a,int b)
{return a*b+2*b;}

int test2(int a,int b)
{return a*(a+b);}

int test(int (*fun)(int,int),int a,int b)
{int i=5;
 return i+fun(a,b);
}
```

```
11
31
```

5. Dynamic Memory Allocation

- “new” and “delete” for one dimensional array
  - ➢ int Size=100;
    float *pV=new float[Size]; ←——— 配置一塊記憶體
    - ☞ float *pV;
      pV=new float[Size];
    - ☞ pV[i]可以用*(pV+i)表示,其意義相同
  - ➢Const int N;
    float V[N];
    - ☞ pV是變數指標,可以改變指向的目標,而V是常數指標,不能更動.
    - ☞陣列V的大小固定,而pV目前所指向的陣列大小可由Size的臨時決定.整數N必須是常數,而Size可以是變數.

☞V和&V是一樣的,都表示向量第一個元素的位址.而pV才存有新定義的向量起始位址,&p是指標變數pV的位址.

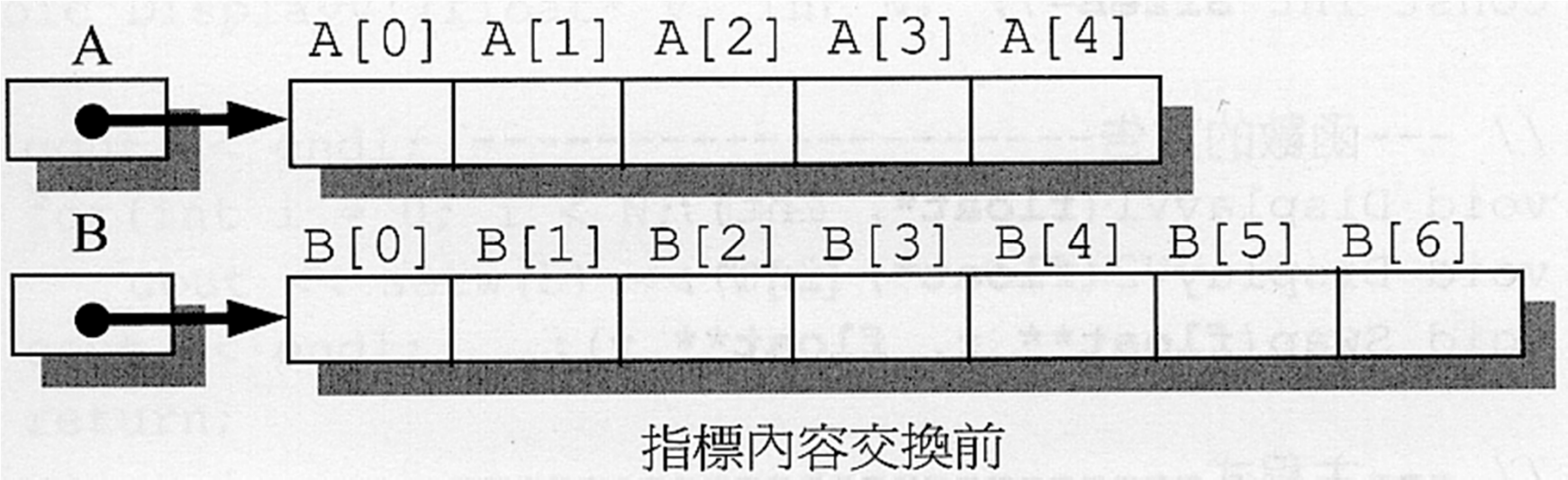
☞由new所配置的記憶體空間不再使用時,可用下列任一個敘述收回:

delete [] pV; 或 delete pV;

■ 這個回收指令並沒有將指標pV一起清除, 只回收他所指向的記憶體空間.

➢指標交換

☞ 若需要對調兩個陣列,只需將他們的指標內容(內部儲存的位址)對調就可以,不需要對調每個元素.



指標內容交換前

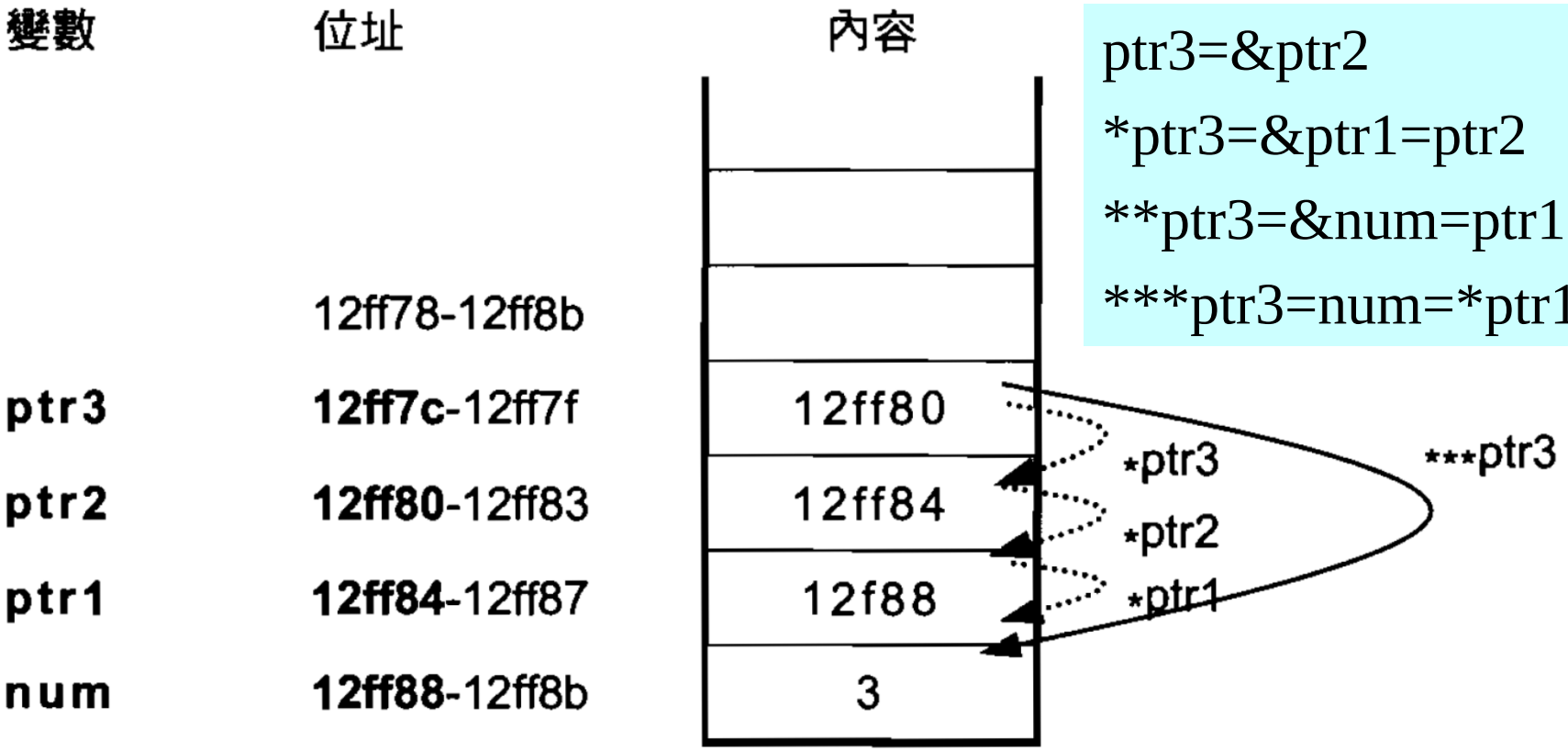
指標內容交換後

➢多重指標

```
int num=3;
int *ptr1=&num;
int **ptr2=&ptr1;
int ***ptr3=&ptr2;
```

宣告ptr1為指標變數且指向num這整數變數

宣告ptr2為雙重指標變數且將ptr1的位置指定給ptr2,ptr2先指向ptr1,再透過ptr1取得num的內容



ptr3=&ptr2

*ptr3=&ptr1=ptr2

**ptr3=&num=ptr1=*ptr2

***ptr3=num=*ptr1=**ptr2

Program 6.5.1

```
#include <iostream>
#include <iomanip>
using namespace std;

const int SizeA=5,SizeB=7;

void DisplayV1(float*, int);
void DisplayV2(float*, int);
void Swap(float** x, float** y);

int main()
{int i;
  float* A = new float [SizeA];
  float* B = new float [SizeB];

   for (i=0; i<SizeA; i++)
      A[i]= 1.0*i;
   for (i=0; i<SizeB; i++)
      B[i]= 3.0*i;
   cout << "執行 Swap() 前, \n";
   cout << "A 是 :\n";
   DisplayV1(A, SizeA);
   cout << "B 是 :\n";
   DisplayV2(B, SizeB);

   Swap(&A,&B);

   cout << "執行 Swap() 後, \n";
   cout << "A 是 :\n";
   DisplayV1(A, SizeB);
   cout << "B 是 :\n";
   DisplayV2(B, SizeA);

   delete [] A;
   delete B;
   return 0;
}
```

```
void DisplayV1(float* V, int N)
{
  cout << endl;
  for(int i = 0; i < N; i++)
      cout << setw(5) << V[i] << " ";
  cout << endl;
  return;
}

void DisplayV2(float* V, int N)
{
  cout << endl;
  for(int i = 0; i < N; i++)
      cout << setw(5) << *(V+i) << " ";
  cout << endl;
  return;
}
```

```
void Swap(float** x, float** y)
{
   float* Temp;
   Temp = *x;
   *x = *y;
   *y = Temp;
}
```

- “new” and “delete” for two dimensional array
  - 產生二維陣列

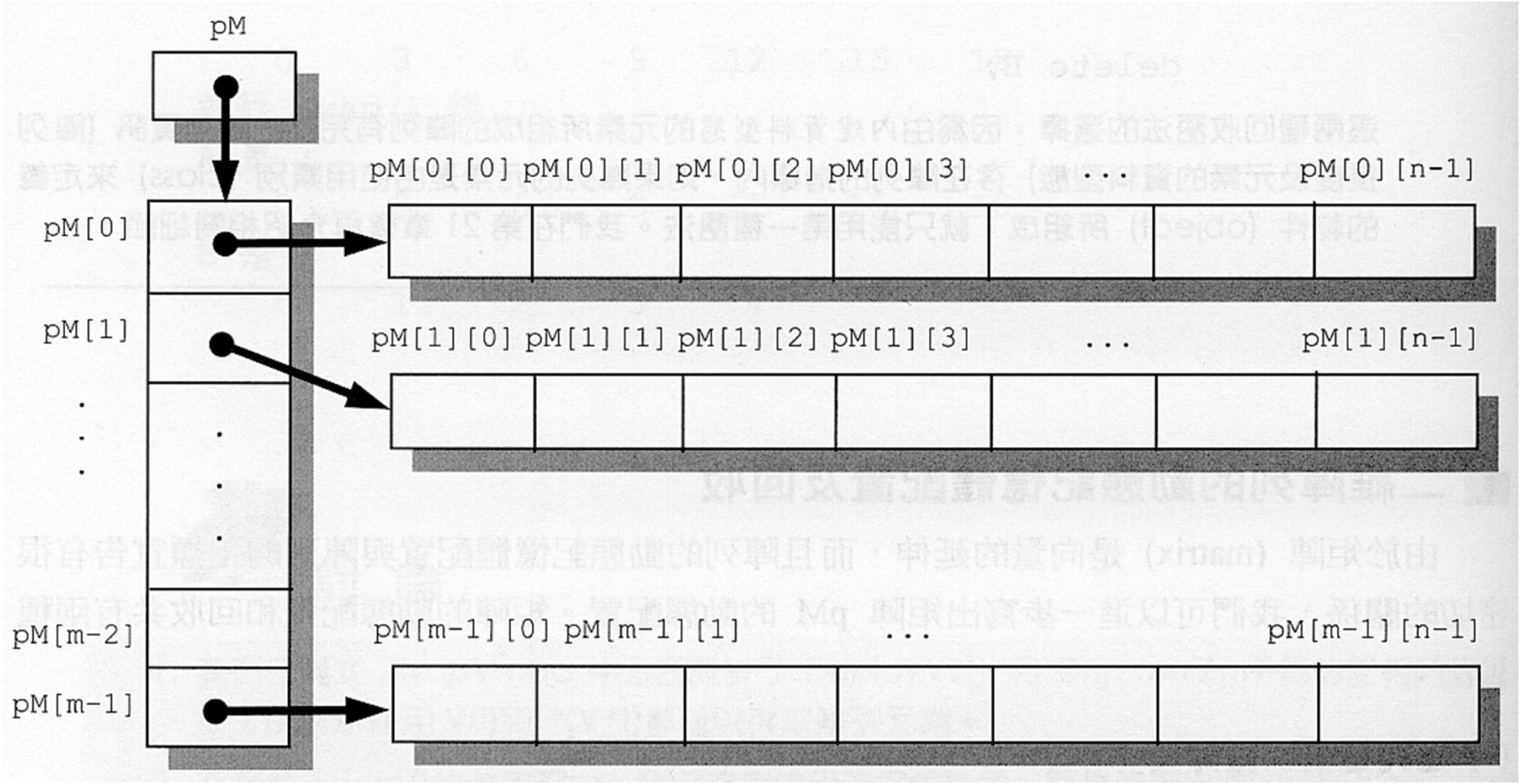
```
int m=20,n=50;  //[20][50]
float **pM=new float *[m];
for (int i=0;i<m;i++)
    pM[i]=new float[n]; //每個pM[i]指向一個新向量
```

  ☞ **pM 為指標的指標(雙重指標)

  ☞ 把pM[0]到pM[m-1]共m個指標指向這些長度為n的向量

  - 回收二維陣列

```
for (i=0;i<m;i++)
   delete pM[i];   ←—— 釋放pM[0]到pM[m-1]共m個指標所指的向量
delete pM;         ←—— 釋放pM所指的指標向量
```


pM
pM[0]
pM[1]
pM[m-2]
pM[m-1]
pM[0][0] pM[0][1] pM[0][2] pM[0][3] ... pM[0][n-1]
pM[1][0] pM[1][1] pM[1][2] pM[1][3] ... pM[1][n-1]
pM[m-1][0] pM[m-1][1] ... pM[m-1][n-1]

- 以連續記憶體空間的方式建構二維矩陣
  - 產生二維陣列

```
int m=20,n=50;
float **pM=new float *[m];
pM[0]=new float [m*n];
for (i=0;i<m;i++)
    pM[i]=pM[i-1]+n;
```

每個pM[i]指向陣列的特殊位址

  - 回收二維陣列

```
delete pM[0];
```

釋放指標pM[0]所指的向量

```
delete pM;
```

釋放指標pM所指的向量

Program 6.5.2

```
#include <iostream>
#include <iomanip>
using namespace std;

const int m = 2,n=3;
void  ShowMatrix(float **);
float MatrixAvg (float **);
void  Sum(float **, float **, float **);

void main()
{int i,j;
  // 動態記憶體配置 pMa
  float **pMa = new float *[m];
  if (!pMa)
    cout <<"記憶體在配置 pMa 時不足!";
  for (i=0; i<m; i++)
    pMa[i] = new float[n];
for (i=0; i< m; i++)
    for (j=0; j< n; j++)
      pMa[i][j]= (i*i+2.0*j)/2.0;
 // 動態記憶體配置 pMb
  float **pMb = new float *[m];
  pMb[0] = new float [m*n];
  for (i=1; i<m; i++)
    pMb[i] = pMb[i-1]+n;
  for (i=0; i< m; i++)
    for (j=0; j< n; j++)
      pMb[i][j] = float(i+j)/2.0;

  // 動態記憶體配置 pMc
  float **pMc = new float *[m];
    pMc[0] = new float [m*n];
  for (i=1; i<m; i++)
    pMc[i] = pMc[i-1]+n;
```

```
 cout  << "陣列 pMa 是: " << endl;
  ShowMatrix(pMa);
  cout  << "陣列 pMb 是: " << endl;
  ShowMatrix(pMb);

 Sum(pMa, pMb, pMc);
  cout  << "陣列 pMa + pMb 是: " << endl;
  ShowMatrix(pMc);

  cout  << "陣列 pMa 的平均值是: "
            << MatrixAvg(pMa) << endl;

 for (i=0; i<m; i++)
    delete [] pMa[i];
  delete [] pMa;
  delete [] pMb[0];
  delete [] pMb;
  delete [] pMc[0];
  delete [] pMc;
}
```

```
void ShowMatrix(float **M)
{for (int i=0; i< m; i++)
    {for (int j=0; j< n; j++)
        cout << setw(5) << M[i][j];
      cout << endl;
    }
  cout << endl;
}

float MatrixAvg(float **M)
{float Sum = 0;
  for (int i=0; i< m; i++)
    for (int j=0; j< n; j++)
          Sum+= M[i][j];
  return  Sum / float(m*n);
}
```

```
void Sum(float **X, float **Y,
float **Z)
{
  for (int i=0; i< m; i++)
    for (int j=0; j< n; j++)
          Z[i][j]= X[i][j]+Y[i][j];
  return;
}
```