指標

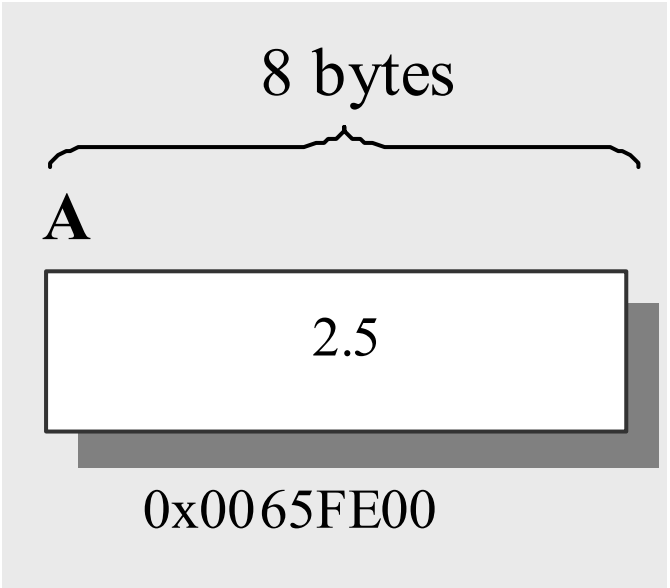
- 8.1　記憶體位址與指標
- 8.2　指標與參照
- 8.3　陣列與指標的代數計算
- 8.4　指標參數
- 8.5　函數指標
- 8.6　動態記憶體配置

在記憶體內資料的四個相關特性

- 1. 資料名稱。
- 2. 資料型態。
- 3. 資料內容。
- 4. 該資料在記憶體內的位址。

變數 A 的內容和位址

```
double A = 2.5
// 定義 double變數A並存入 2.5
```



範例程式 Variable.cpp

```
// Variable.cpp
#include <iostream>
using namespace std;
// ----- 主程式 -----------------------
int main()
{
   double A = 2.5;
   cout << "A 的值為："      << A << endl;
   cout << "A 所佔用的記憶體大小為："
        << sizeof(A)        << " bytes 或是 "
        << sizeof(double) << " bytes" << endl;
   cout << "A 所在位址為："
        << “0x” <<   &A   << endl;
   return 0;
}
```

操作結果

A 的值為：2.5

A 所佔用的記憶體大小為：

8 bytes 或是 8 bytes

A 所在位址為：

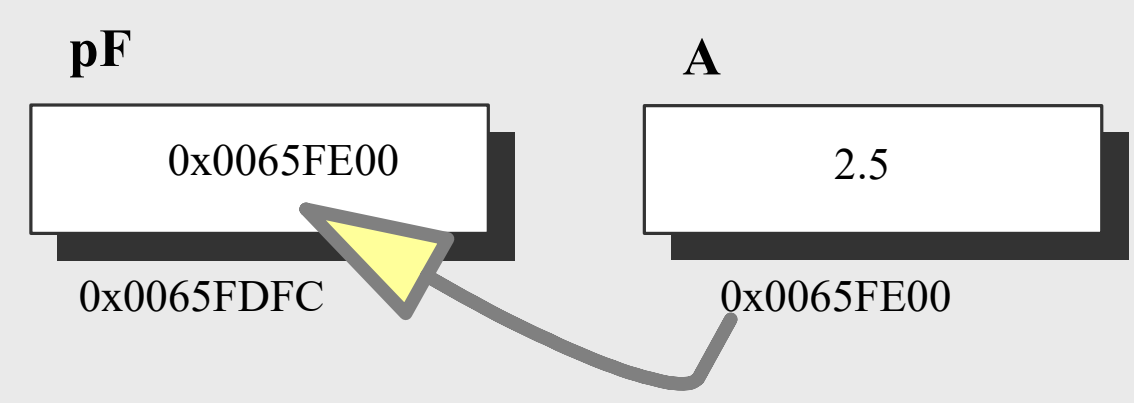
0x0012FF84

指標

- 指標是用來儲存其他資料的位址的特殊代號。例如，我們可以使用以下敘述來定義一個名叫 pF的指標：

```
double A = 2.5;
double* pF;          // 定義指標 pF
pF = &A;             // 將資料 A 的位址存入指標 pF
```

- 具體表示如下：



指標的四個特性

■對於指標 pF而言，它的

名稱： pF

資料型態： double *

資料內容： 0x0065FE00

記憶體位址： 0x0065FDFC

■ 宣告時所指定的目標資料的資料型態無法改變

譬如，pF 已宣告為

```
double* pF;     // 定義指標 pF
```

假如

```
int X = 12;
```

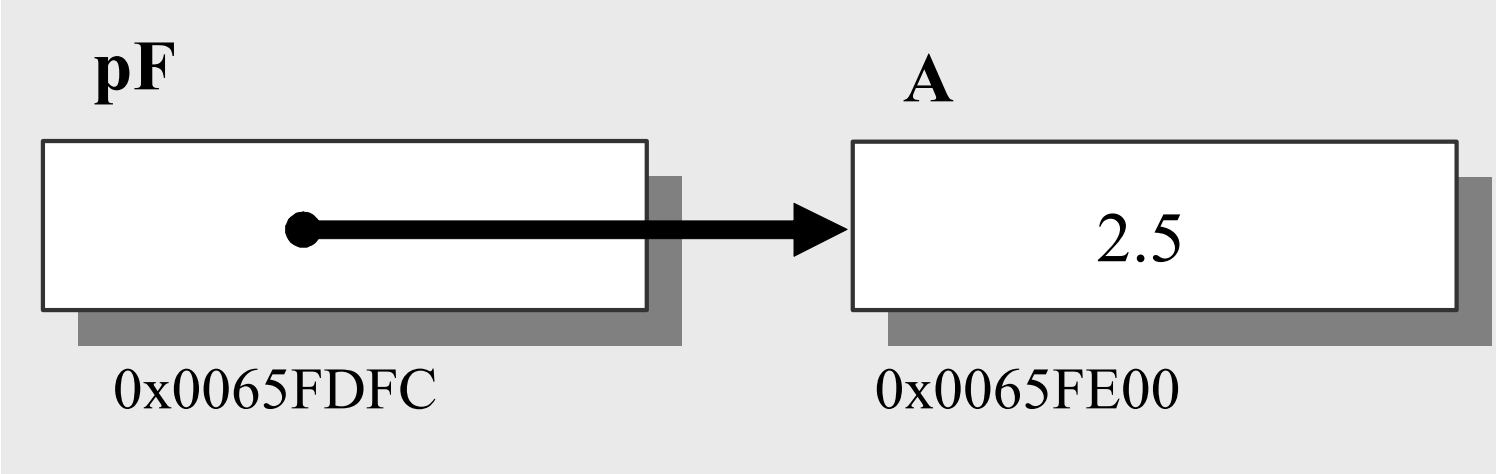
則敘述

```
pF = &X;        // 錯誤! 指標 pF 必需存放 double 資料的位址
```

必需配合目標資料的資料型態改成

```
int* pF;
```

指標 pF 指向變數 A



指標pF和變數A之間的關係

指標的定義

- 指標的定義敘述有兩種寫法：

```
double*  pF;          // 定義指標 pF (較常用)
double  *pF;          // 定義指標 pF (較少用)
```

- 同時定義兩個指標：

```
double *p1, *p2;    // 同時定義p1, p2兩個指標
```

- 在定義pF的同時給予初值：

```
double*  pF = &A;   // 定義指標pF，同時存入資料 A 的位址
```

取值運算子 (value-of operator)

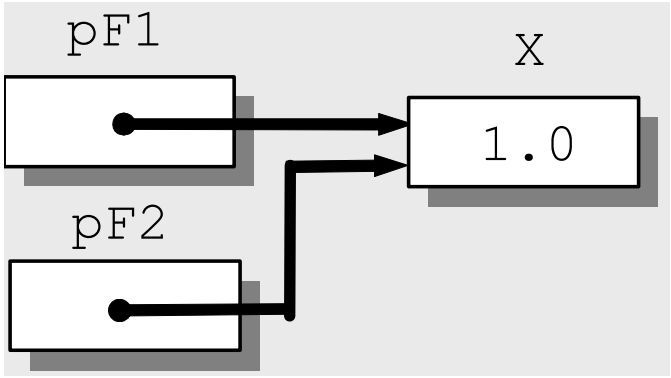
```
double x;          // 定義 double 變數 x

x = *pF;  // 將指標 pF所指處的資料存到變數 x
```

表示的是「將存在指標pF所指之處的內容儲存到x裏面」。

變數指標

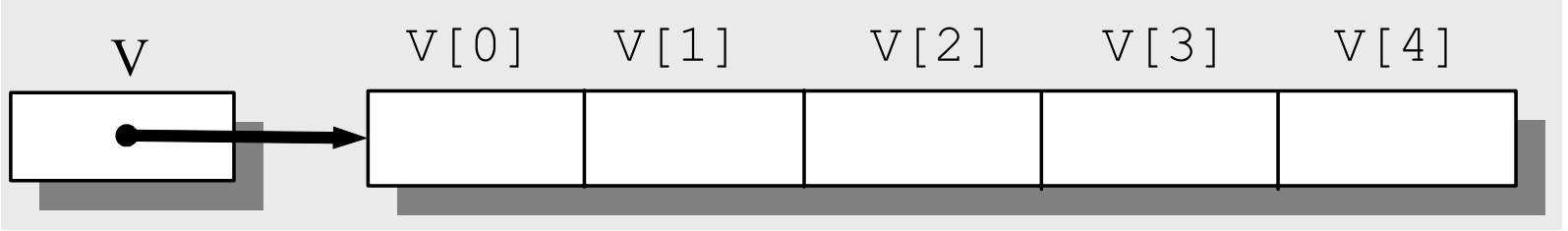
```
double* pF;     // 定義指標pF
double X, Y;
pF = &X;  // pF 指向 X
pF = &Y;  // pF 改指向 Y
```

允許好幾個指標指向同一個資料。例如：



陣列名稱也是指標

```
double V[5];                    // 定義陣列 V
```

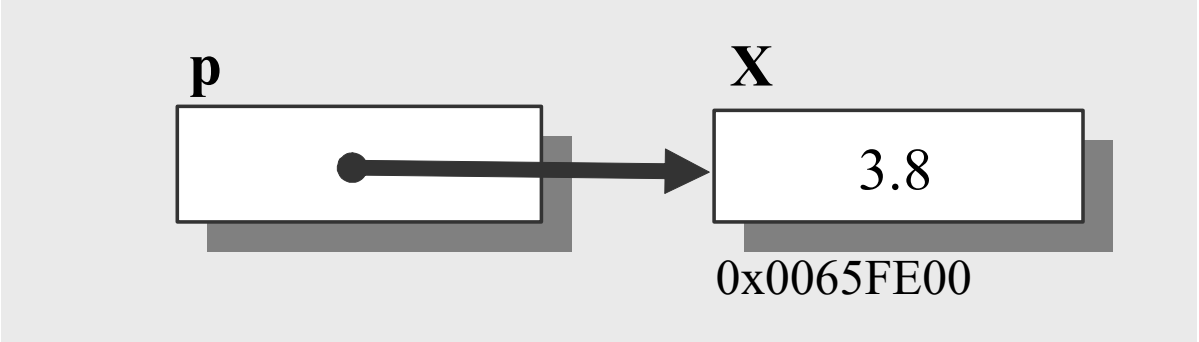


V是一個常數指標 (constant pointer)，
資料型態為double * const。

指標與取址運算子 (address-of operator)「&」

```
double  X = 3.8;      // 宣告和初始化 X
double* p = &X;       // 宣告和初始化 p
```

其關係相當於



*p 是 X 的內容　(亦即上圖中的 3.8)

&X 是 p 的內容　(亦即 X 的位址 0x0065FE00)。

參照

```
double &x = y; // 宣告x 是 y 的參照

double &x;// 錯誤! 必需給 x 參照的對象

double y, z;// 定義 y 和 z 兩個 double 變數

double &x = z; // 錯誤! 不能更改參照的對象
```

範例程式 Ref.cpp 使用參照和指標

```
// Ref.cpp
#include <iostream>
using namespace std;
// ------ 主程式 -----------------------
int main()
{
   double x  = 1.0;
   double &y = x;       // 定義x的參照y
   double *p = &x;      // 定義和初始化指標 p
   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;
   return 0;
}
```

操作結果

```
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
```

使用下標來計算陣列元素的位址

```
const int m = 5;
double A[m];
```

則第k+1個元素，也就是A[k] 的位址為

&A[0] + k x sizeof(double)

使用下標來計算陣列元素的位址

```
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 x n + j) x
  sizeof(double)
```

使用下標來計算陣列元素的位址

```
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

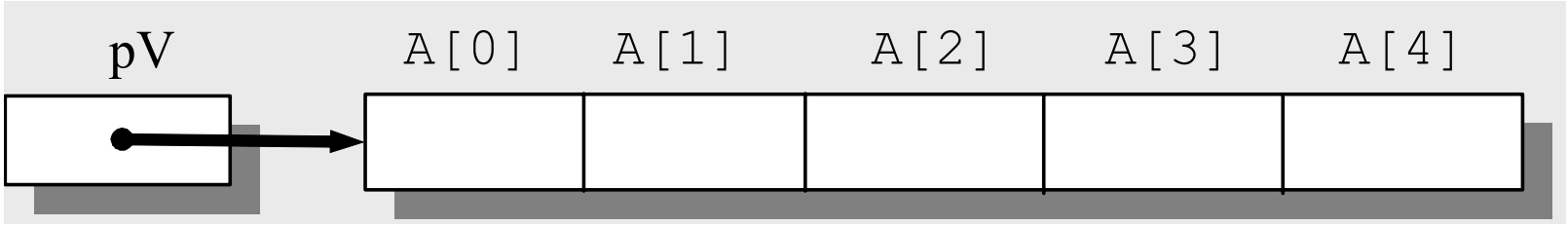
```
&C[0][0][0] + (i x n x p + j x
  p + k) x sizeof(double)
```

使用指標來存取 A[0] 的位址

```
double *pV;
```

則下面兩式都可以將A[0] 的位址存到pV裏面：

```
pV = &A[0];
pV = A;
```



一維陣列(向量)和指標變數pV之間的關係

使用指標來存取陣列元素：指標代數 (pointer algebra)

*pV　　相當於 A[0]

*(pV+i)　　相當於 A[i]

我們可以寫成通式：

A[i] ≡ *(pV+i)

這裏i的範圍為 0 ~ (m-1)。

範例程式 Apointer.cpp (指標代數)

使用*pV++ 逐一獲得向量的各個元素，以求得向量的總合

```
// Apointer.cpp
#include <iostream>
using namespace std;
// ----- 主程式 ---------------------------------
int main()
{
    const int Size = 5;
    double V[Size]  = {48.4, 39.8, 40.5, 42.6, 41.2};
    double Sum = 0.0, Average;
    double *pV = V;
    int i;
    for (i=0; i<Size; i++)
                Sum += *pV++;
    cout  << "陣列 V 各元素的總合是  : "
                << Sum << endl;
    cout  << "陣列 V 各元素的平均值是: "
           << Sum / double(Size) << endl;
    return 0;
}
```

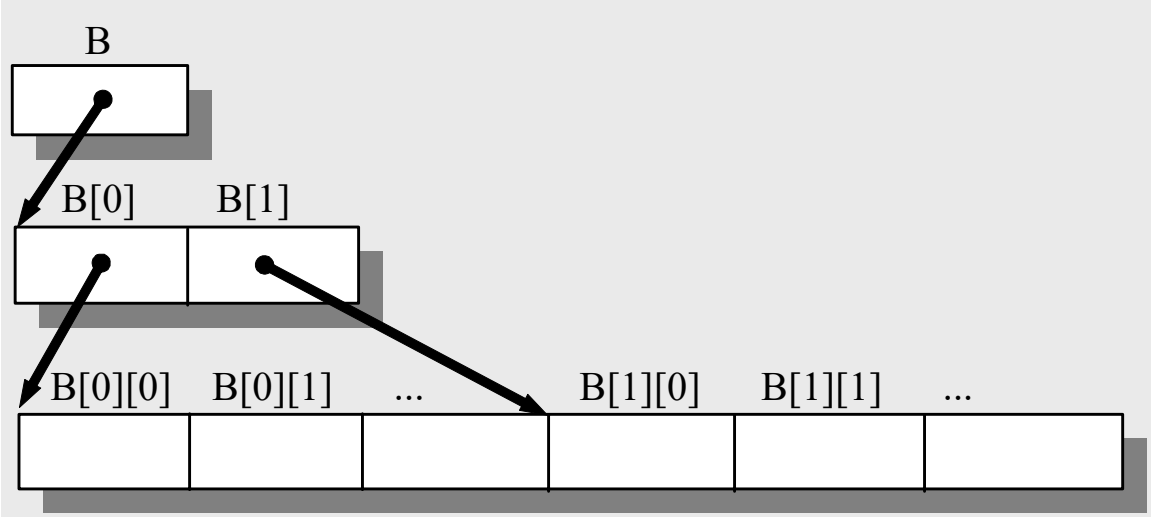
操作結果

陣列 V 各元素的總合是　：212.5

陣列 V 各元素的平均值是：42.5

二維陣列的指標

```
const int m = 2;
const int n = 3;
double B[m][n];
```



二維陣列B的儲存方式及相關的指標

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) ≡ *(*(B+i) + j)

範例程式 ShowMatrix.cpp (指標表示法)

```
#include <iomanip>
#include <iostream>
using namespace std;
// ------------------------------------
int main()
{
  const int Row = 2;     const int Col = 3;
  double B[Row][Col]={ 1.8, 4.9, 6.8, 6.2, 2.1,
3.4};
  int i, j;
  cout  << "(1)陣列 B 是: " << endl;
  for (int i=0; i< Row; i++)
    { for (int j=0; j< Col; j++)
          cout << setw(5) << B[i][j];
       cout << endl;
    }
```

```
cout << endl;
  cout  << "(2)陣列 B 是: " << endl;
  for (int i=0; i< Row; i++)
    { for (int j=0; j< Col; j++)
          cout << setw(5) << *(B[i] + j);
       cout << endl;
    }
  cout << endl;
  cout  << "(3)陣列 B 是: " << endl;
  for (int i=0; i< Row; i++)
    { for (int j=0; j< Col; j++)
          cout << setw(5) << *(*(B+i) + j);
       cout << endl;
    }
  return 0;
}
```

操作結果

(1)陣列 B 是：

1.8 4.9 6.8

6.2 2.1 3.4

(2)陣列 B 是：

1.8 4.9 6.8

6.2 2.1 3.4

(3)陣列 B 是：

1.8 4.9 6.8

6.2 2.1 3.4

指標參數 (Pointers as Parameters)

- 使用參照時，從呼叫端無從辨別參數有沒有使用參照

  例如，如果在呼叫這一側的敘述寫成

```
X = Fnc(a, b);
  // 呼叫函數Fnc()
```

  被呼叫側的函數其原型可能是下列兩者中的任一個：

```
double Fnc(double, double);
double Fnc(double&, double&);
```

傳址（Passing Addresses）

在呼叫端可以直接將位址當成參數：

```
X = Fnc(&a, &b);          // 呼叫函數Fnc()，
                          //   參數用a 和 b 的位址
```

而在接收端就可以使用指標來接受這個位址：

```
double Fnc(double*, double*);     //參數列是指標
```

範例程式 Swap2F.cpp

使用傳址的方式來交換變數的值

```
// Swap2F.cpp
#include <iostream>
using namespace std;
// --- 函數 SwapF () 的宣告 ----------------
void SwapF(double* x, double* y);
// --- 主程式 ------------------------------
int main()
{
   double a = 2, b = 5;
   cout << "執行 SwapF() 前, \n";
   cout << "  a 是: " << a  << "  b 是: " << b <<
endl;
   SwapF(&a, &b);
   cout << "執行 SwapF() 後, \n";
   cout << "  a 是: " << a  << "  b 是: " << b;
   return 0;
}
```

```
// --- 函數 SwapF () 的定義 -----------------
void SwapF(double* x, double* y)
{
  double Temp;
  Temp = *x;
  *x = *y;
  *y = Temp;
  return;
}
```

操作結果

執行 SwapF() 前,

a 是：2 b 是：5

執行 SwapF() 後,

a 是：5 b 是：2

傳遞指標以存取陣列

將陣列當成函數的參數傳遞

```
double VecAvg,    P[5];    // P是長度為5
```

的一維陣列

```
VecAvg = Average(P);     // 呼叫函數
```

Average()

而被呼叫函數的原型可寫成：

```
double Average(double []);
double Average(double *);
```

範例程式　檔案 ArrayPFnc.cpp

```
// ArrayPFnc.cpp
#include <iostream>
using namespace std;
// ---函數 Average()和MaxElem()的宣告
double Average(double *);
double MaxElem(double *);
const int Size =  5;
// ---主程式-----------------------
int main()
{
    double P[Size]  = {48.4, 39.8, 40.5, 42.6, 41.2};
    cout  << "陣列 P 的平均值是: " << Average(P) <<
endl;
    cout   << "陣列 P 的最大值是: " << MaxElem(P) <<
endl;
    return 0;
}
```

```
// ---函數 Average() 的定義----------
double Average(double* V)
{
  double Sum = 0;
  for (int i = 0; i < Size; i++)
     Sum += V[i];
  return Sum/double(Size);
}
// ---函數 MaxElem() 的定義----------
double MaxElem(double* V)
{
  double MaxE;     MaxE = V[0];
  for (int i = 1; i < Size; i++ )
    if (MaxE < V[i]) MaxE = V[i];
  return MaxE;
}
```

操作結果

陣列 P 的平均值是：42.5

陣列 P 的最大值是：48.4

函數指標 (Function Pointers)

函數名稱本身就是一種常數指標 (constant pointer)。

```
double Func(int); // 函數 Func( )的原型
```

可以使用下列的語法定義函數指標pF並指向函數 Func()：

```
double(*pF)(int);        // 定義函數指標pF
pF = Func;    // 將函數指標pF指向函數 Func( )
```

上面兩式可以合併成一個具初始化功能的定義敘述：

```
double(*pF)(int) = Func;
```

函數指標的使用方式

```
cont << pF(6) << endl;
```

函數指標的使用

例如，把數值積分的演算法寫成函數 Integ()：

```
double Integ(double (*Fnc)(double));
```

對於不同的數學函數，譬如

```
double Fnc1(double);          // 數學函數Fnc1() 的原型
double Fnc2(double);          // 數學函數Fnc2() 的原型
```

這兩個數學函數的積分值：

```
double x1, x2;                // 宣告實數 x1 和 x2
x1 = Integ(Fnc1);     // 計算Fnc1() 的積分值
x2 = Integ(Fnc2);     // 計算Fnc2() 的積分值
```

範例程式 FncPoint.cpp 函數指標的使用

```
// FncPoint.cpp
#include <iostream>
using namespace std;
// --- 函數 F1()， F2()和Twice() 的宣告 --
double F1(int);
double F2(int);
double Twice(double (*)(int), int);
// --- 主程式 ---------------------------
int main()
{
    int A = 3;
    int B = 5;
    cout   << "Twice(F1, A)的值是: "  <<
Twice(F1, A) << endl;
    cout   << "Twice(F2, B)的值是: "  <<
Twice(F2, B) << endl;
    return 0;
}
```

```
// ---- 函數F1()的定義 ----------------------
double F1(int N)  { return double (N*N);}
// ---- 函數F2()的定義 ----------------------
double F2(int N)  { return double (N*N*N);}
// ---  函數Twice()的定義 -------------------
double Twice(double (*pF)(int), int N)
  {return 2.0 * double( pF(N) );}
```

操作結果

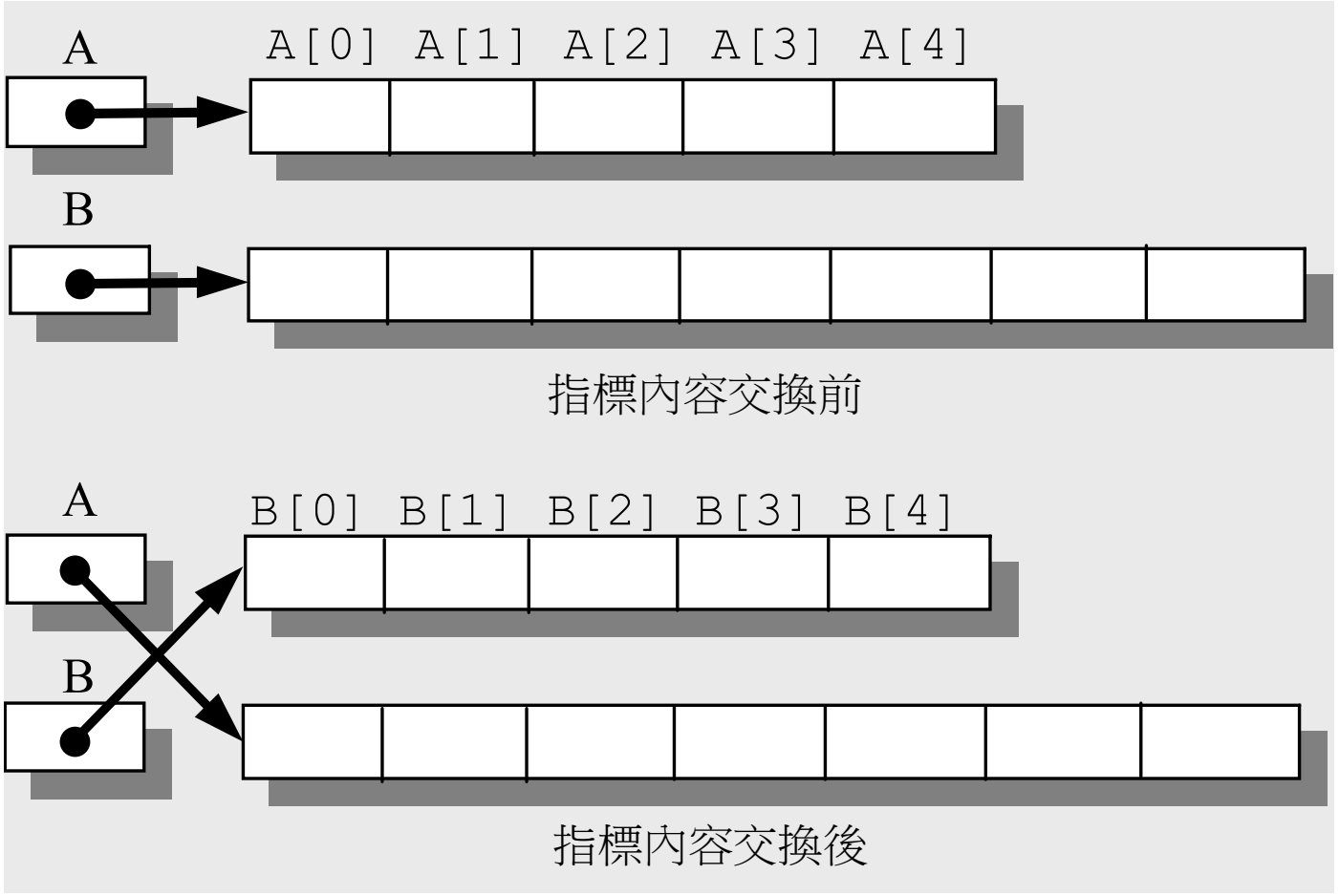
Twice(F1, A)的值是：18

Twice(F2, B)的值是：250

一維陣列的動態記憶體配置及回收

```
int Size = 100;
double *pV = new double[Size];
double *pV;
pV = new double[Size];
delete [] pV;
```

使用動態記憶體配置以避免大量資料的搬移



範例程式 SwapV.cpp 交換指標內容以交換陣列

```
// SwapV.cpp
#include <iostream>
#include <iomanip>
using namespace std;
// ---函數的宣告---------------------
void DisplayV1(double*, int);
void DisplayV2(double*, int);
void Swap(double** x, double** y);
// ---主程式-------------------------
int main()
{
   const int SizeA = 5;
   const int SizeB = 7;
   double* A = new double [SizeA];
   double* B = new double [SizeB];
   for (int i=0; i<SizeA; i++)       A[i]= 1.0*i;
   for (int 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;
    }

    // ------ 函數DisplayV1()的定義 -------------
    void DisplayV1(double* V, int N)
    { cout << endl;
      for(int i = 0; i < N; i++)
          cout << setw(5) << V[i] << " ";
      cout << endl;
      return;
    }
    }
```

```
// ------- 函數DisplayV2()的定義 -------------
void DisplayV2(double* V, int N)
{ cout << endl;
  for(int i = 0; i < N; i++)
      cout << setw(5) << *(V+i) << " ";
  cout << endl;
  return;
}
// -------- 函數Swap()的定義 -----------------
void Swap(double** x, double** y)
{  double* Temp;
   Temp = *x;
   *x = *y;
   *y = Temp;
}
```

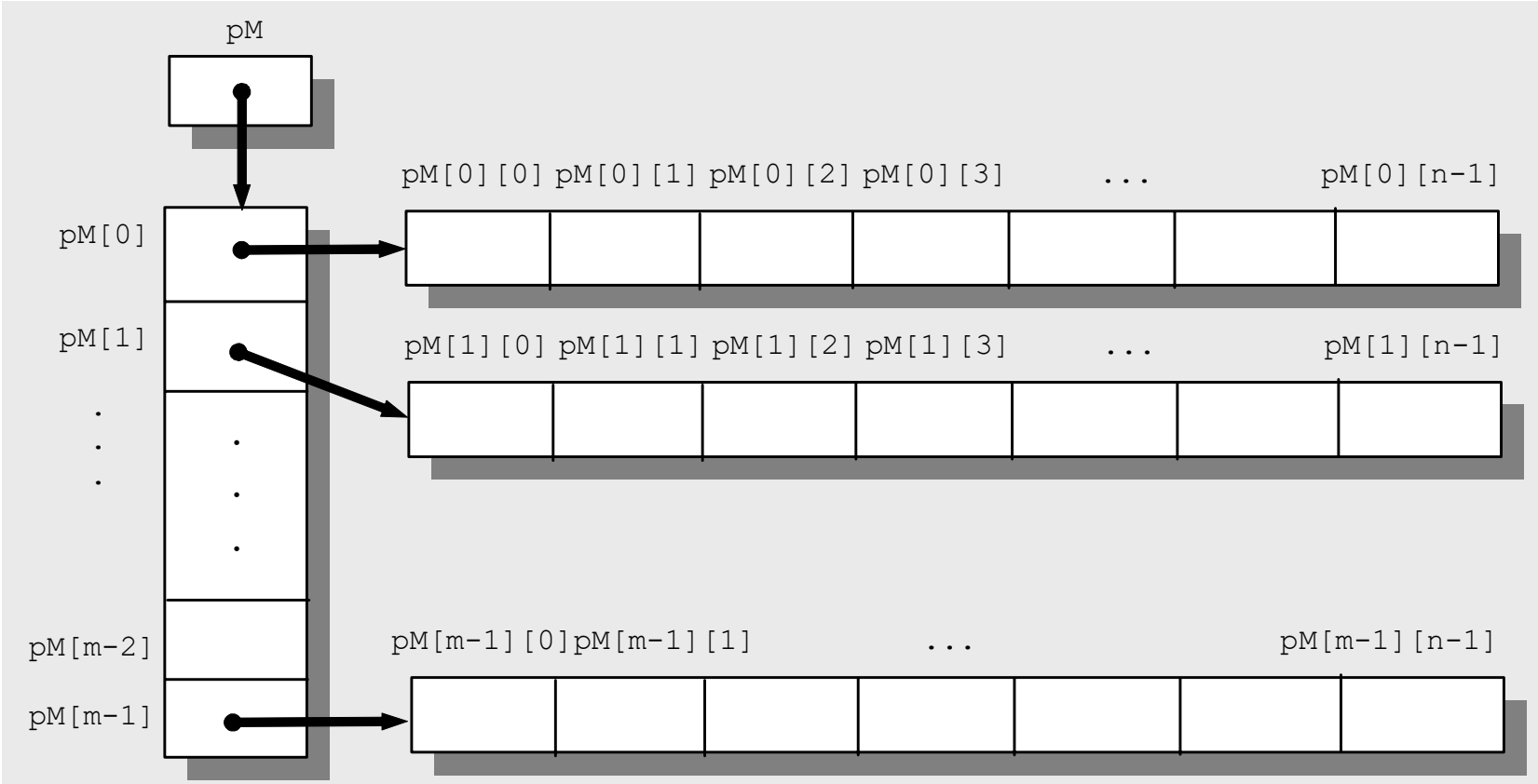
操作結果

```
執行 Swap() 前,
A 是 :
   0     1     2     3     4
B 是 :
   0     3     6     9    12    15    18
執行 Swap() 後,
A 是 :
   0     3     6     9    12    15    18
B 是 :
    0     1     2     3     4
```

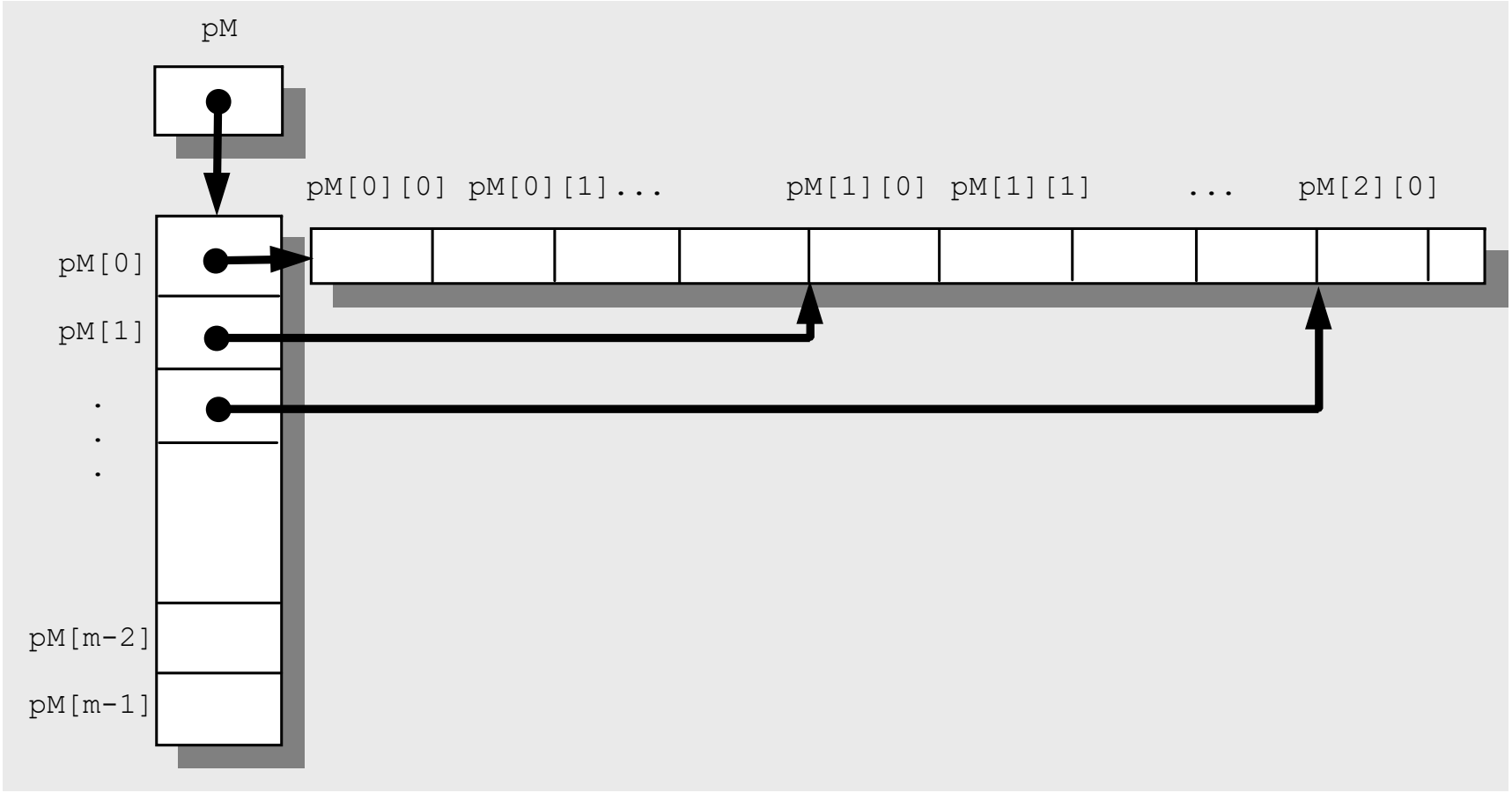
以逐列的方式建構二維陣列

```
int m = 20, n = 50;
// 動態配置長度為 m 的一維指標陣列
double **pM = new double *[m];
for (int i=0; i<m; i++)
pM[i] = new double[n];
// 回收二維陣列的記憶體
for (int i=0; i<m; i++)
   delete pM[i];
delete pM;
```

二維陣列的動態記憶體配置 (以逐列的方式建構二維陣列)



以連續記憶空間的方式建構二維陣列



以連續記憶空間的方式建構二維陣列

```
int m = 20, n = 50;
// 動態配置長度為 m 的一維指標陣列
double **pM = new double *[m];
pM[0] = new double [m*n];// pM[0]指向新向量的開頭處
for (int i=1; i<m; i++)
    pM[i] = pM[i-1]+n;  // 每個pM[i]指向陣列的特殊
delete pM[0];  // 釋放指標pM[0]所指的向量
delete pM;// 釋放pM所指的指標向量
```

範例程式 DynMatrix.cpp
向量和矩陣用在檢查亂數的平均值

```
// DynMatrix.cpp
#include <iomanip>
#include <iostream>
#include <new>
using namespace std;
const int m = 2;
const int n = 3;
// --- 各函數的宣告 ------------------------
void  ShowMatrix(double **);
double MatrixAvg (double **);
void  Sum(double **, double **, double **);
void  LackMemory()
{
  cerr << "記憶體不足!\n";
  abort();
}
```

```
// --- 主程式 ------------------------------
int main()
{
  // 動態記憶體配置 pMa
  set_new_handler(LackMemory);
  double **pMa = new double *[m];
  for (int i=0; i<m; i++)
      pMa[i] = new double[n];

  for (int i=0; i< m; i++)
      for (int j=0; j< n; j++)
         pMa[i][j]= (i*i+2.0*j)/2.0;
  // 動態記憶體配置 pMb
  double **pMb = new double *[m];
  pMb[0] = new double [m*n];
  for (int i=1; i<m; i++)
      pMb[i] = pMb[i-1]+n;
  for (int i=0; i< m; i++)
    for (int j=0; j< n; j++)
      pMb[i][j] = double(i+j)/2.0;
```

```
// 動態記憶體配置 pMc
  double **pMc = new double *[m];
  pMc[0] = new double [m*n];
  for (int i=1; i<m; i++)
      pMc[i] = pMc[i-1]+n;
// 顯示 pMa 和 pMb
  cout  << "陣列 pMa 是: " << endl;
  ShowMatrix(pMa);
  cout  << "陣列 pMb 是: " << endl;
  ShowMatrix(pMb);
  // 求 pMc
  Sum(pMa, pMb, pMc);
  cout  << "陣列 pMa + pMb 是: "   << endl;
  ShowMatrix(pMc);
  // 求 pMa 的平均值
  cout  << "陣列 pMa 的平均值是: "  << MatrixAvg(pMa) <<
endl;
// 回收 pMa
 for (int i=0; i<m; i++)
  delete [] pMa[i];
 delete [] pMa;
```

```
// 回收 pMb
  delete [] pMb[0];
  delete [] pMb;
  // 回收 pMc
  delete [] pMc[0];
  delete [] pMc;
  return 0;
}
// --- 函數 ShowMatrix() 的定義 ----------
void ShowMatrix(double **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;
 return;
}
```

```
// --- 函數 MatrixAvg() 的定義 ------------
double MatrixAvg(double **M)
{
  double Sum = 0;
  for (int i=0; i< m; i++)
    for (int j=0; j< n; j++)
         Sum+= M[i][j];
  return  Sum / double(m*n);
}
// --- 函數 Sum() 的定義 --------------------
void Sum(double **X, double **Y, double **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;
}
```

程式 DynMatrix.cpp 操作結果

```
陣列pMa是：
  01  2
  0.51.52.5
陣列pMb是：
  00.51
  0.511.5
陣列pMa + pMb是：
  01.53
  12.54
陣列pMa的平均值是：1.25
```