ACPS C++ 教學講義 07

C++ 陣列

C++ 支持陣列資料結構，它可以儲存一個固定大小的相同類型元素的順序集合。陣列是用來儲存一系列資料，但它往往被認為是一系列相同類型的變數。

陣列的宣告並不是宣告一個個單獨的變數，而是宣告一個陣列變數，比如 numbers，然後使用 numbers[0]、numbers[1]、...、numbers[99] 來代表一個個單獨的變數。

numbers

numbers[0]	numbers[1]	. . . . .	numbers[99]

陣列中的特定元素可以通過索引訪問。

所有的陣列都是由連續的記憶體位置組成。最低的地址對應第一個元素，最高的地址對應最後一個元素。

宣告陣列

在 C++ 中要宣告陣列，需要指定元素的類型和元素的數量，如下所示：

```
type arrayName [ arraySize ];
```

這叫做一維陣列。

教講者：華祖彬

arraySize 必須是一個大於零的整數常數，type 可以是任意有效的 C++ 資料類型。

例如，要宣告一個類型為 double 的包含 10 個元素的陣列 balance，宣告語句如下：

```
double balance[10];
```

現在 balance 是一個可用的陣列，可以容納 10 個 double 的數字。

初始化陣列

在 C++ 中，您可以逐個初始化陣列，也可以使用一個初始化語句，如下所示：

```
double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
```

大括號 { } 之間的值的數目不能大於我們在陣列宣告時在方括號 [ ] 中指定的元素數目。

如果您省略掉了陣列的大小，陣列的大小則為初始化時元素的個數。因此，如果：

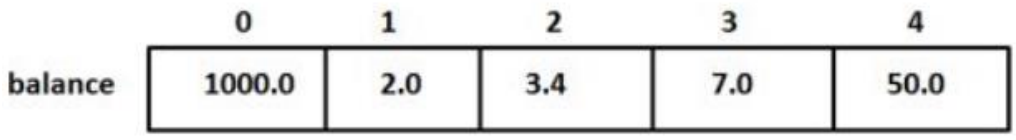
```
double balance[] = {1000.0, 2.0, 3.4, 17.0, 50.0};
```

您將創建一個陣列，它與前一個實例中所創建的陣列是完全相同的。下面是一個為陣列中某個元素賦值的實例：

```
balance[4] = 50.0;
```

上述的語句把陣列中第五個元素的值賦為 50.0。所有的陣列都是以 0 作為它們第一個元素的索引，也被稱為基索引，陣列的最後一個索引是陣列的總大小減去 1。

以下是上面所討論的陣列的的圖形表示：



訪問陣列元素

陣列元素可以通過陣列名稱加索引進行訪問。

元素的索引是放在方括號內，跟在陣列名稱的後邊。例如：

```
double salary = balance[9];
```

上面的語句將把陣列中第 10 個元素的值賦給 salary 變數。

下面的實例使用了上述的三個概念，即，宣告陣列、陣列賦值、訪問陣列：

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

#include <iomanip>
using std::setw;

int main ()
{
   int n[ 10 ]; // n 是一個包含 10 個整數的陣列

   // 初始化陣列元素
   for ( int i = 0; i < 10; i++ )
   {
      n[ i ] = i + 100; // 設置元素 i 為 i + 100
   }
   cout << "Element" << setw( 13 ) << "Value" << endl;

   // 輸出陣列中每個元素的值
   for ( int j = 0; j < 10; j++ )
   {
      cout << setw( 7 )<< j << setw( 13 ) << n[ j ] <<
endl;
   }

   return 0;
}
```

上面的程序使用了 setw() 函數來格式化輸出。

當上面的程式碼被編譯和執行時，它會產生下列結果：

```
Element       Value
      0         100
      1         101
      2         102
      3         103
      4         104
      5         105
      6         106
      7         107
      8         108
      9         109
```

多維陣列

C++ 支持多維陣列。多維陣列宣告的一般形式如下：

```
type name[size1][size2]...[sizeN];
```

例如，下面的宣告創建了一個三維 5 . 10 . 4 整數型陣列：

```
int threedim[5][10][4];
```

二維陣列

多維陣列最簡單的形式是二維陣列。一個二維陣列，在本質上，是一個一維陣列的列表。

宣告一個 x 行 y 列的二維整數型陣列，形式如下：

```
type arrayName [ x ][ y ];
```

其中，type 可以是任意有效的 C++ 資料類型，arrayName 是一個有效的 C++ 標識符號。

一個二維陣列可以被認為是一個帶有 x 行和 y 列的表格。下面是一個二維陣列，包含 3 行和 4 列：

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

因此，陣列中的每個元素是使用形式為 a[i , j] 的元素名稱來標識的，其中 a 是陣列名稱，i 和 j 是唯一標識 a 中每個元素的下標。

初始化二維陣列

多維陣列可以通過在括號內為每行指定值來進行初始化。

下面是一個帶有 3 行 4 列的陣列。

```
int a[3][4] = {
 {0, 1, 2, 3} ,   /*  初始化索引號為 0 的行 */
 {4, 5, 6, 7} ,   /*  初始化索引號為 1 的行 */
 {8, 9, 10, 11}   /*  初始化索引號為 2 的行 */
};
```

內部巢狀的括號是可選的，下面的初始化與上面是等同的：

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

訪問二維陣列元素

二維陣列中的元素是通過使用下標（即陣列的行索引和列索引）來訪問的。例如：

```
int val = a[2][3];
```

上面的語句將獲取陣列中第 3 行第 4 個元素。

讓我們來看看下面的程序，我們將使用巢狀循環來處理二維陣列：

```
#include <iostream>
```

```
using namespace std;

int main ()
{
   // 一個帶有 5 行 2 列的陣列
   int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};

   // 輸出陣列中每個元素的值
   for ( int i = 0; i < 5; i++ )
      for ( int j = 0; j < 2; j++ )
      {
         cout << "a[" << i << "][" << j << "]: ";
         cout << a[i][j]<< endl;
      }

   return 0;
}
```

當上面的程式碼被編譯和執行時，它會產生下列結果：

```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8
```

如上所述，您可以創建任意維度的陣列，但是一般情況下，我們創建的陣列是一維陣列和二維陣列

指向陣列的指標

陣列名是一個指向陣列中第一個元素的常數指標。因此，在下面的宣告中：

```
double balance[50];
```

balance 是一個指向 &balance[0] 的指標，即陣列 balance 的第一個元素的地址。

因此，下面的程序片段把 p 賦值為 balance 的第一個元素的地址：

```
double *p;
double balance[10];

p = balance;
```

使用陣列名作為常數指標是合法的，反之亦然。因此，

***(balance + 4) 是一種訪問 balance[4] 數據的合法方式。**

一旦您把第一個元素的地址儲存在 p 中，

您就可以使用 *p、*(p+1)、*(p+2) 等來訪問陣列元素。

下面的實例演示了上面討論到的這些概念：

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

int main ()
{
   // 帶有 5 個元素的整數型陣列
   double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
   double *p;

   p = balance;

   // 輸出陣列中每個元素的值
   cout << "使用指標的陣列值 " << endl;
   for ( int i = 0; i < 5; i++ )
   {
       cout << "*(p + " << i << ") : ";
       cout << *(p + i) << endl;
   }

   cout << "使用 balance 作為地址的陣列值 " << endl;
   for ( int i = 0; i < 5; i++ )
```

```
   {
      cout << "*(balance + " << i << ") : ";
      cout << *(balance + i) << endl;
   }

   return 0;
}



#include <stdio.h>

int main ()
{
   /* 帶有 5 個元素的整數型陣列 */
   double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
   double *p;
   int i;

   p = balance;

   /* 輸出陣列中每個元素的值 */
   printf( "使用指標的陣列值\n");
   for ( i = 0; i < 5; i++ )
   {
      printf("*(p + %d) : %f\n",  i, *(p + i) );
   }

   printf( "使用 balance 作為地址的陣列值\n");
   for ( i = 0; i < 5; i++ )
   {
      printf("*(balance + %d) : %f\n",  i, *(balance +
i) );
   }

   return 0;
}
```

當上面的程式碼被編譯和執行時，它會產生下列結果：

使用指標的陣列值

```
*(p + 0) : 1000
*(p + 1) : 2
*(p + 2) : 3.4
*(p + 3) : 17
*(p + 4) : 50
```

使用 balance 作為地址的陣列值

```
*(balance + 0) : 1000
*(balance + 1) : 2
*(balance + 2) : 3.4
*(balance + 3) : 17
*(balance + 4) : 50
```

在上面的實例中，p 是一個指向 double 型的指標，這意味著它可以儲存一個 double 類型的變數。

一旦我們有了 p 中的地址，***p 將給出儲存在 p 中相應地址的值**。

傳遞陣列給函數

C++ 不允許向函數傳遞一個完整的陣列作為參數，但是，您可以通過指定不帶索引的陣列名來傳遞一個指向陣列的指標。

如果您想要在函數中傳遞一個一維陣列作為參數，您必須以下面三種方式來宣告函數形式參數，**這三種宣告方式的結果是一樣的**，因為每種方式都會告訴編譯器將要接收一個整數型指標。

同樣地，您也可以傳遞一個多維陣列作爲形式參數。

方式 I

形式參數是一個指標：

```
void myFunction(int *param)
{
.
.
.
}
```

方式 2

形式參數是一個已定義大小的陣列：

```
void myFunction(int param[10])
{
.
.
.
}
```

方式 3

形式參數是一個未定義大小的陣列：

```
void myFunction(int param[])
{
.
.
.
}
```

實例

現在，讓我們來看下面這個函數，它把陣列作為參數，同時還傳遞了另一個參數，根據所傳的參數，會返回陣列中各元素的平均值：

```
double getAverage(int arr[], int size)
{
  int    i, sum = 0;
  double avg;
```

```
  for (i = 0; i < size; ++i)
  {
    sum += arr[i];
   }

  avg = double(sum) / size;

  return avg;
}
```

現在，讓我們呼叫上面的函數，如下所示：

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

// 函數宣告
double getAverage(int arr[], int size);

int main ()
{
   // 帶有 5 個元素的整數型陣列
   int balance[5] = {1000, 2, 3, 17, 50};
   double avg;

   // 傳遞一個指向陣列的指標作為參數
   avg = getAverage( balance, 5 ) ;

   // 輸出返回值
   cout << "平均值是：" << avg << endl;

   return 0;
}
```

當上面的程式碼被編譯和執行時，它會產生下列結果：

```
平均值是： 214.4
```

您可以看到，就函數而言，陣列的長度是無關緊要的，因為 C++ **不會對形式參數執行邊界檢查**。

從函數返回陣列

C++ 不**允許返回一個完整的陣列作為函數的參數**。但是，您可以通過指定不帶索引的陣列名來返回一個指向陣列的指標。

如果您想要從函數返回一個一維陣列，您必須宣告一個返回指標的函數。

實例：

```
int * myFunction()
{
.
.
.
}
```

另外，C++ 不支持在函數外返回區域變數的地址，除非定義區域變數為 **static** 變數。

現在，讓我們來看下面的函數，它會生成 10 個隨機數，並使用陣列來返回它們，具體如下：

```
#include <iostream>
#include <ctime>

using namespace std;

// 要生成和返回隨機數的函數
int * getRandom( )
{
  static int  r[10];

  // 設置種子
  srand( (unsigned)time( NULL ) );
  for (int i = 0; i < 10; ++i)
  {
    r[i] = rand();
    cout << r[i] << endl;
  }
```

```
  return r;
}

// 要呼叫上面定義函數的主函數
int main ()
{
   // 一個指向整數的指標
   int *p;

   p = getRandom();
   for ( int i = 0; i < 10; i++ )
   {
       cout << "*(p + " << i << ") : ";
       cout << *(p + i) << endl;
   }

   return 0;
}
```

當上面的程式碼被編譯和執行時，它會產生下列結果：

```
624723190
1468735695
807113585
976495677
613357504
1377296355
1530315259
1778906708
1820354158
667126415
*(p + 0) : 624723190
*(p + 1) : 1468735695
*(p + 2) : 807113585
*(p + 3) : 976495677
*(p + 4) : 613357504
*(p + 5) : 1377296355
*(p + 6) : 1530315259
*(p + 7) : 1778906708
*(p + 8) : 1820354158
*(p + 9) : 667126415
```