ACPS C++ 教學講義 09

C++ 指標

學習 C++ 的指標既簡單又有趣。通過指標，可以簡化一些 C++ 程式編輯任務的執行，還有一些任務，如動態記憶體分配，沒有指標是無法執行的。所以，想要成為一名優秀的 C++ 程式設計師，學習指標是很有必要的。

正如您所知道的，每一個變數都有一個記憶體位置，每一個記憶體位置都可使用連字號（&）運算符號來取用，它表示了在記憶體中的一個地址。請看下面的實例，它將輸出定義的變數地址：

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

int main ()
{
   int  var1;
   char var2[10];
   cout << "var1 變數的地址： ";
   cout << &var1 << endl;
   cout << "var2 變數的地址： ";
   cout << &var2 << 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：

```
var1 變數的地址： 0xbfebd5c0
var2 變數的地址： 0xbfebd5b6
```

通過上面的實例，我們瞭解了什麼是記憶體地址以及如何取用它。接下來讓我們看看什麼是指標。

什麼是指標？

指標是一個變數，其值為另一個變數的地址，也就是記憶體位置的直接地址。就像其他變數或常數一樣，您必須在使用指標儲存其他變數地址之前，對其進行宣告。指標變數宣告的一般形式為：

```
type *var-name;
```

在這裡，type 是指標的基礎類型，它必須是一個有效的 C++ 資料類型，var-name 是指標變數的名稱。用來宣告指標的星號 * 與乘法中使用的星號是相同的。但是，在這個語句中，星號是用來指定一個變數是指標。以下是有效的指標宣告：

```
int    *ip;    /* 一個整數型的指標 */
double *dp;    /* 一個 double 型的指標 */
float  *fp;    /* 一個浮點型的指標 */
char   *ch     /* 一個字元型的指標 */
```

C++ 中使用指標

使用指標時會頻繁進行以下幾個操作：定義一個指標變數、把變數地址賦值給指標、取用指標變數中可用地址的值。

實例：

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

int main ()
{
```

```
   int  var = 20;   // 實際變數的宣告
   int  *ip;        // 指標變數的宣告
   ip = &var;       // 在指標變數中儲存 var 的地址
   cout << "Value of var variable: ";
   cout << var << endl;
   // 輸出在指標變數中儲存的地址
   cout << "Address stored in ip variable: ";
   cout << ip << endl;
   // 取用指標中地址的值
   cout << "Value of *ip variable: ";
   cout << *ip << 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：

```
Value of var variable: 20
Address stored in ip variable: 0xbfc601ac
Value of *ip variable: 20
```

C++ 指標詳解

在 C++ 中，有很多指標相關的概念，這些概念都很簡單，但是都很重要。下面列出了 C++ 程式設計師必須清楚的一些與指標相關的重要概念：

C++ Null 指標

在變數宣告的時候，如果沒有確切的地址可以賦值，為指標變數賦一個 NULL 值是一個良好的程式編輯習慣。賦為 NULL 值的指標被稱為空指標。

NULL 指標是一個定義在標準庫中的值為零的常數。

請看下面的程序：

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

int main ()
{
   int  *ptr = NULL;
   cout << "ptr 的值是 " << ptr ;

   return 0;
}
```

當上面的程式碼被編譯和執行時，它會產生下列結果：

```
ptr 的值是 0
```

記憶體地址 0 有特別重要的意義，它表明該指標不指向任何記憶體位置。所以如果指標是空值（零值），則假定它不指向任何東西。

而檢查一個空指標，您可以使用 if 語句，如下所示：

```
if(ptr)     /* 如果 p 非空，則完成 */
if(!ptr)    /* 如果 p 為空，則完成 */
```

因此，如果所有未使用的指標都被賦予空值，同時避免使用空指標，就可以防止誤用一個未初始化的指標。很多時候，未初始化的變數存有一些垃圾值，導致程序難以執行。

C++ 指標的算術運算

指標是一個用數值表示的地址。因此，您可以對指標執行算術運算。可以對指標進行四種算術運算：++、--、+、-。

假設 ptr 是一個 4 byte，指向地址 1000 的整數型指標，，我們對該

指標執行下列的算術運算：

```
ptr++
```

在執行完上述的運算之後，ptr 將指向位置 1004，因為 ptr 每增加一次，它都將指向下一個整數位置，即當前位置往後移 4 個字節。

遞增一個指標

我們喜歡在程序中使用指標代替陣列，因為變數指標可以遞增，而陣列不能遞增，因為陣列是一個常數指標。下面的程序遞增變數指標，以便順序取用陣列中的每一個元素：

```
#include <iostream>
using namespace std;
const int MAX = 3;

int main ()
{
   int  var[MAX] = {10, 100, 200};
   int  *ptr;
   // 指標中的陣列地址
   ptr = var;
   for (int i = 0; i < MAX; i++)
   {
      cout << "Address of var[" << i << "] = ";
      cout << ptr << endl;
      cout << "Value of var[" << i << "] = ";
      cout << *ptr << endl;
      // 移動到下一個位置
      ptr++;
   }
   return 0;
}
```

當上面的程式碼被編譯和執行時，它會產生下列結果：

```
Address of var[0] = 0xbfa088b0
Value of var[0] = 10
Address of var[1] = 0xbfa088b4
Value of var[1] = 100
Address of var[2] = 0xbfa088b8
Value of var[2] = 200
Value of var[1] = 10
```

指標的比較

指標可以用關係運算符號進行比較，如 ==、< 和 >。

下面的程序修改了上面的實例，只要變數指標小於或等於陣列的最後一個元素的地址 &var[MAX - 1]，則印出結果並將把變數指標進行遞增：

```
#include <iostream>
using namespace std;
const int MAX = 3;

int main ()
{
   int  var[MAX] = {10, 100, 200};
   int  *ptr;
   // 指標中第一個元素的地址
   ptr = var;
   int i = 0;
   while ( ptr <= &var[MAX - 1] )
   {
      cout << "Address of var[" << i << "] = ";
      cout << ptr << endl;
      cout << "Value of var[" << i << "] = ";
      cout << *ptr << endl;
```

```
      // 指向上一個位置
      ptr++;
      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：

```
Address of var[0] = 0xbfce42d0
Value of var[0] = 10
Address of var[1] = 0xbfce42d4
Value of var[1] = 100
Address of var[2] = 0xbfce42d8
Value of var[2] = 200
```

C++ 指標 vs 陣列

指標和陣列是密切相關的。事實上，指標和陣列在很多情況下是可以互換的。請看下面的程序：

```
#include <iostream>
using namespace std;
const int MAX = 3;

int main ()
{
   int  var[MAX] = {10, 100, 200};
   int  *ptr;
   // 指標中的陣列地址
   ptr = var;
   for (int i = 0; i < MAX; i++)
```

```
   {
      cout << "Address of var[" << i << "] = ";
      cout << ptr << endl;
      cout << "Value of var[" << i << "] = ";
      cout << *ptr << endl;
      // 移動到下一個位置
      ptr++;
   }
   return 0;
}
```

當上面的程式碼被編譯和執行時，它會產生下列結果：

```
Address of var[0] = 0xbfa088b0
Value of var[0] = 10
Address of var[1] = 0xbfa088b4
Value of var[1] = 100
Address of var[2] = 0xbfa088b8
Value of var[2] = 200
```

然而，指標和陣列並不是完全互換的。

```
#include <iostream>
using namespace std;
const int MAX = 3;

int main ()
{
   int  var[MAX] = {10, 100, 200};
   for (int i = 0; i < MAX; i++)
   {
      *var = i;    // 這是正確的語法
      var++;       // 這是不正確的
   }
```

```
   return 0;
}
```

把指標運算符號 * 應用到 var 上是完全可以接受的，但修改 var 的值是非法的。這是因為 var 是一個指向陣列開頭的常數，不能作為左值。

由於一個陣列名對應一個指標常數，只要不改變陣列的值，仍然可以用指標形式的表達式。例如，下面是一個有效的語句，把 var[2] 賦值為 500：

```
*(var + 2) = 500;
```

上面的語句是有效的，且能成功編譯，因為 var 未改變。

C++ 指標陣列

在我們講解指標陣列的概念之前，先讓我們來看一個實例，它用到了一個由 3 個整數構成的陣列：

```
#include <iostream>
using namespace std;
const int MAX = 3;

int main ()
{
   int  var[MAX] = {10, 100, 200};
   for (int i = 0; i < MAX; i++)
   {
      cout << "Value of var[" << i << "] = ";
      cout << var[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：

```
Value of var[0] = 10
```

```
Value of var[1] = 100
Value of var[2] = 200
```

可能有一種情況，我們想要讓陣列儲存指標。下面是一個指向整數的指標陣列的宣告：

```
int *ptr[MAX];
```

在這裡，把 ptr 宣告為一個陣列，ptr 中的每個元素，都是一個指向 int 值的指標。

```
#include <iostream>

using namespace std;
const int MAX = 3;

int main ()
{
   int  var[MAX] = {10, 100, 200};
   int *ptr[MAX];

   for (int i = 0; i < MAX; i++)
      ptr[i] = &var[i]; // 賦值為整數的地址
   for (int i = 0; i < MAX; i++)
   {
      cout << "Value of var[" << i << "] = ";
      cout << *ptr[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：

```
Value of var[0] = 10
Value of var[1] = 100
```

```
Value of var[2] = 200
```

您也可以用一個指向字元的指標陣列來儲存一個字元串列表，如下：

```
#include <iostream>
using namespace std;
const int MAX = 4;

int main ()
{
 const char *names[MAX] = {
                "Zara Ali",
                "Hina Ali",
                "Nuha Ali",
                "Sara Ali",
   };
   for (int i = 0; i < MAX; i++)
   {
      cout << "Value of names[" << i << "] = ";
      cout << names[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：

```
Value of names[0] = Zara Ali
Value of names[1] = Hina Ali
Value of names[2] = Nuha Ali
Value of names[3] = Sara Ali
```

C++ 傳遞指標給函數

C++ 允許您傳遞指標給函數，只需要簡單地宣告函數參數為指標類型即可。

下面的實例中，我們傳遞一個無符號的 long 型指標給函數，並在函數內改變這個值：

```
#include <iostream>
#include <ctime>
using namespace std;
void getSeconds(unsigned long *par);

int main ()
{
   unsigned long sec;
   getSeconds( &sec );
   // 輸出實際值
   cout << "Number of seconds :" << sec << endl;

   return 0;
}

void getSeconds(unsigned long *par)
{
   // 獲取當前的秒數
   *par = time( NULL );
   return;
}
```

當上面的程式碼被編譯和執行時，它會產生下列結果：

```
Number of seconds :1294450468
```

函數能接受指標作為參數，而指標可存取陣列，所以函數也能接受陣列作為

參數：

```
#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 << "Average value is: " << avg << endl;
   return 0;
}

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;
}
```

當上面的程式碼被編譯和執行時，它會產生下列結果：

```
Average value is: 214.4
```

C++ 從函數返回指標

C++ 允許您從函數返回指標。為了做到這點，您必須宣告一個返回指標的函數，另外，C++ 不支持在函數外返回局部變數的地址，除非定義局部變數為 static 變數。

現在，讓我們來看下面的函數，它會生成 10 個隨機數，並使用表示指標的陣列名（即第一個陣列元素的地址）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
```