ACPS 進階
C++ 教學講義 05

C++ 函數與遞迴

函數是一組一起執行一個任務的語句。

每個 C++ 程序都至少有一個函數，即主函數 main()。您可以把程式碼劃分到不同的函數中，如何劃分程式碼到不同的函數中是由您來決定的，

但在邏輯上，劃分通常是根據每個函數執行一個特定的任務來進行的。**函數宣告**告訴編譯器函數的名稱、返回類型和參數。**函數定義**提供了函數的實際主體。

C++ 標準庫提供了大量的程序可以呼叫的內建函數。

例如，函數 strcat() 用來連接兩個字串，函數 memcpy() 用來複製記憶體到另一個位置。

函數還有很多叫法，比如函式、方法、例程或子程序，等等。

定義函數

C++ 中的函數定義的一般形式如下：

```
return_type function_name( parameter list )
{
   body of the function
}
```

在 C++ 中，函數由一個函數頭和一個函數主體組成。

教講者：華祖彬

下面列出一個函數的所有組成部分：

返回類型：	一個函數可以返回一個值。 return_type 是函數返回的值的資料類型。 有些函數執行所需的操作而不返回值，在這種情況下， return_type 是關鍵字 void。
函數名稱：	這是函數的實際名稱。 函數名和參數列表一起構成了函數簽名。
參數：	參數就像是佔位符號。當函數被呼叫時，您向參數傳遞一個值，這個值被稱為實際參數。 參數列表包括函數參數的類型、順序、數量。參數是可選的，也就是說，函數可能不包含參數。
函數主體：	函數主體包含一組定義函數執行任務的語句。

實例

以下是 max() 函數的程式碼。

該函數有兩個參數 num1 和 num2，會返回這兩個數中較大的那個數：

```
// 函數返回兩個數中較大的那個數

int max(int num1, int num2)
{
   // 區域變數宣告
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

函數宣告

函數宣告會告訴編譯器函數名稱及如何呼叫函數。

函數的實際主體可以單獨定義。

函數宣告包括以下幾個部分：

```
return_type function_name( parameter list );
```

針對上面定義的函數 max()，以下是函數宣告：

```
int max(int num1, int num2);
```

在函數宣告中，參數的名稱並不重要，只有參數的類型是必需的，因此下面也是有效的宣告：

```
int max(int, int);
```

當您在一個源文件中定義函數且在另一個文件中呼叫函數時，函數宣告是必需的。在這種情況下，您應該在呼叫函數的文件頂部宣告函數。

呼叫函數

創建 C++ 函數時，會定義函數做什麼，然後通過呼叫函數來完成已定義的任務。

當程序呼叫函數時，程序控制權會轉移給被呼叫的函數。被呼叫的函數執行已定義的任務，當函數的返回語句被執行時，或到達函數的結束括號時，會把程序控制權交還給主程序。

呼叫函數時，傳遞所需參數，如果函數返回一個值，則可以儲存返回值。

實例：

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

```
// 函數宣告
int max(int num1, int num2);

int main ()
{
   // 區域變數宣告
   int a = 100;
   int b = 200;
   int ret;

   // 呼叫函數來獲取最大值
   ret = max(a, b);

   cout << "Max value is : " << ret << endl;
    return 0; }   // 函數返回兩個數中較大的那個數
int max(int num1, int num2)  {    // 區域變數宣告
    int result;
    if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

把 max() 函數和 main() 函數放一塊，編譯程式碼。當運行最後的可執行文件時，會產生下列結果：

```
Max value is : 200
```

函數參數

如果函數要使用參數，則必須宣告接受參數值的變數。這些變數稱為函數的形式參數。

形式參數就像函數內的其他區域變數，在進入函數時被創建，退出函數時被銷毀。

當呼叫函數時，有三種向函數傳遞參數的方式：

呼叫類型	描述
傳值呼叫	該方法把參數的實際值複製給函數的形式參數。在這種情況下，修改函數內的形式參數對實際參數沒有影響。
傳址呼叫	該方法把參數的地址複製給形式參數。 在函數內，該地址用於訪問呼叫中要用到的實際參數。 這意味著，修改形式參數會影響實際參數。
引用呼叫	該方法把參數的引用複製給形式參數。 在函數內，該引用用於訪問呼叫中要用到的實際參數。 這意味著，修改形式參數會影響實際參數。

預設情況下，C++ 使用傳值呼叫來傳遞參數。一般來說，這意味著函數內的程式碼不能改變用於呼叫函數的參數。

C++ 傳值呼叫

向函數傳遞參數的傳值呼叫方法，把參數的實際值複製給函數的形式參數。在這種情況下，修改函數內的形式參數不會影響實際參數。

預設情況下，C++ 使用傳值呼叫方法來傳遞參數。

實例：

```
// 函數定義
void swap(int x, int y)
{
   int temp;

   temp = x; /* 保存 x 的值 */
   x = y;    /* 把 y 賦值給 x */
```

```
   y = temp; /* 把 x 賦值給 y */

   return;
}
```

現在，讓我們通過傳遞實際參數來呼叫函數 swap()：

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

// 函數宣告
void swap(int x, int y);

int main ()
{
   // 區域變數宣告
   int a = 100;
   int b = 200;

   cout << "交換前，a 的值：" << a << endl;
   cout << "交換前，b 的值：" << b << endl;

   // 呼叫函數來交換值
   swap(a, b);

   cout << "交換後，a 的值：" << a << endl;
   cout << "交換後，b 的值：" << b << 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：

```
交換前，a 的值： 100
交換前，b 的值： 200
交換後，a 的值： 100
交換後，b 的值： 200
```

上面的實例表明了，雖然在函數內改變了 a 和 b 的值，但是**實際上 a 和**

b 的值沒有發生變化。

傳址呼叫

向函數傳遞參數的指標呼叫方法，把參數的地址複製給形式參數。

在函數內，該地址用於訪問呼叫中要用到的實際參數。

這意味著，修改形式參數會影響實際參數。

按指標傳遞值，參數指標被傳遞給函數，就像傳遞其他值給函數一樣。因此相應地，在下面的函數 swap() 中，您需要宣告函數參數為指標類型，該函數用於交換參數所指向的兩個整數變數的值。

實例：

```
// 函數定義
void swap(int *x, int *y)
{
   int temp;
   temp = *x;/* 保存地址 x 的值 */
   *x = *y;      /* 把 y 賦值給 x */
   *y = temp;/* 把 x 賦值給 y */

   return;
}
```

現在，讓我們通過指標傳值來呼叫函數 swap()：

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

// 函數宣告
void swap(int *x, int *y);

int main ()
{
   // 區域變數宣告
   int a = 100;
```

```
   int b = 200;

   cout << "交換前，a 的值：" << a << endl;
   cout << "交換前，b 的值：" << b << endl;

   /* 呼叫函數來交換值
    * &a 表示指向 a 的指標，即變數 a 的地址
    * &b 表示指向 b 的指標，即變數 b 的地址
    */
   swap(&a, &b);

   cout << "交換後，a 的值：" << a << endl;
   cout << "交換後，b 的值：" << b << 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：

```
交換前，a 的值： 100
交換前，b 的值： 200
交換後，a 的值： 200
交換後，b 的值： 100
```

引用呼叫

向函數傳遞參數的引用呼叫方法，把參數的地址複製給形式參數。

在函數內，該引用用於訪問呼叫中要用到的實際參數。

這意味著，修改形式參數會影響實際參數。

按引用傳遞值，參數引用被傳遞給函數，就像傳遞其他值給函數一樣。因此相應地，在下面的函數 swap() 中，您需要宣告函數參數為引用類型，該函數用於交換參數所指向的兩個整數變數的值。

教講者：華祖彬

```
// 函數定義
void swap(int &x, int &y)
{
   int temp;
   temp = x; /* 保存地址 x 的值 */
   x = y;    /* 把 y 賦值給 x */
   y = temp; /* 把 x 賦值給 y  */

   return;
}
```

現在，讓我們通過引用傳值來呼叫函數 swap()：

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

// 函數宣告
void swap(int &x, int &y);

int main ()
{
   // 區域變數宣告
   int a = 100;
   int b = 200;

   cout << "交換前，a 的值：" << a << endl;
   cout << "交換前，b 的值：" << b << endl;

   /* 呼叫函數來交換值 */
   swap(a, b);

   cout << "交換後，a 的值：" << a << endl;
   cout << "交換後，b 的值：" << b << 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：

```
交換前，a 的值： 100
交換前，b 的值： 200
交換後，a 的值： 200
交換後，b 的值： 100
```

參數的預設值

當您定義一個函數，您可以為參數列表中後邊的每一個參數指定預設值。當呼叫函數時，如果實際參數的值留空，則使用這個預設值。

這是通過在函數定義中使用賦值運算符號來為參數賦值的。呼叫函數時，如果未傳遞參數的值，則會使用預設值，如果指定了值，則會忽略預設值，使用傳遞的值。

實例：

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

int sum(int a, int b=20)
{
  int result;

  result = a + b;

  return (result);
}

int main ()
{
   // 區域變數宣告
   int a = 100;
   int b = 200;
   int result;
```

```
   // 呼叫函數來添加值
   result = sum(a, b);
    cout << "Total value is :" << result << endl;     //
再次呼叫函數
    result = sum(a);
    cout << "Total value is :" << result << 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：

```
Total value is :300
Total value is :120
```

inline 內嵌函數

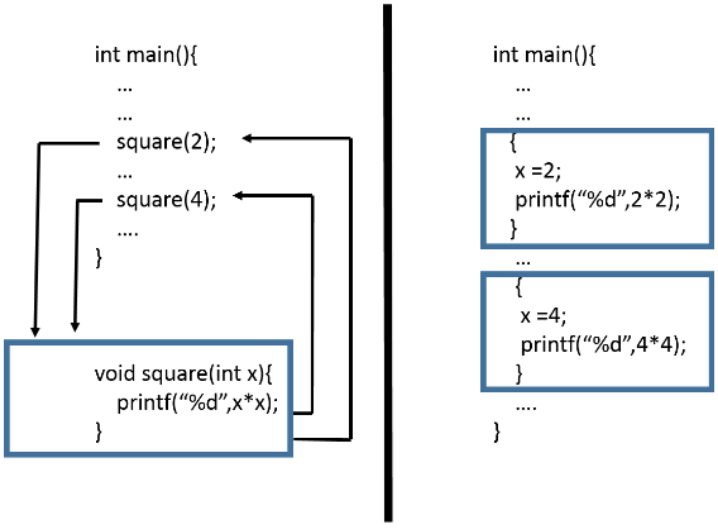
內嵌函數是在一般的函數前面，加上 inline 這個關鍵字，例如以下的例子，我要做一個計算平方的函數

```
int square(int x) { return x*x; }
```

此時只要把 inline 這個關鍵字加上去變成

```
inline int square(int x) { return x*x; }
```

inline 內嵌函數的執行過程



一般而言，當我們撰寫函數，並呼叫使用，電腦的機器語言指令會紀錄目前工作階段的記憶體位址，然後跳至函數的記憶體位置處理完程序後，並回到原先的位址上，而這樣來回會造成時間上的額外負擔。

C++於是提供這種內嵌函數，當我們加入關鍵字時，在編譯時便會把函數中的程式直接展開，使用內嵌函數可以省去較多的呼叫函數的時間，並且如果常常會使用到此函數，使用內嵌的效率也會比較好

所以如果你的函數中的程式碼或處理較短，並且常呼叫使用的話，可以給他加上個 inline

另外如果是遞迴的程式，不要使用 inline，編譯器可能也不會理你（因為裡面的不斷呼叫運算會使編譯器判斷為此函數中程式的行可能會過大）

挑戰：

一次性輸入不同 3 個半徑，求取圓周長。
提示：圓周長公式：2R x PI，PI 為圓周率 3.14159
圓周長計算需寫成函數呼叫，可使用內嵌函數 inline，
輸出結果格式取至小數 2 位

遞迴（Recursion）

遞迴（Recursion）是在函式中呼叫自身同名函式，而呼叫者本身會先被置入記憶體堆疊中，等到被呼叫者執行完畢之後，再從堆疊中取出之前被置入的函式繼續執行。

「堆疊」（Stack）是一種「先進後出」的資料結構，就好比您將書本置入箱中，最先放入的書會最後才取出。

C++支援函式的遞迴呼叫，遞迴的概念較抽象，但實際應用很多，舉個例子來說，求最大公因數就可以使用遞迴來求，下面的程式是使用遞迴來求最大公因數的一個例子。

實例：

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

int gcd(int, int);

int main() {
    int m = 0;
    int n = 0;

    cout << "輸入兩數：";
    cin >> m >> n;

    cout << "GCD: "
         << gcd(m, n) << endl;

    return 0;
}

int gcd(int m, int n) {
    if(n == 0)
        return m;
    else
        return gcd(n, m % n);
}
```

執行結果：

```
輸入兩數：10 45
GCD: 5
```

上面的程式是使用輾轉相除法來求最大公因數；遞迴具有重複執行的特性，而可以使用遞迴求解的程式，實際上也可以使用迴圈來求解。

那麼使用遞迴好還是使用迴圈求解好？

這並沒有一定的答案。不過通常由於遞迴本身有重複執行與記憶體堆疊的特性，所以若在求解時需要使用到堆疊特性的資料結構時，使用遞迴在設計時的邏輯會比較**容易理解**，程式碼設計出來也會比較**簡潔**，然而遞迴會有函式呼叫的負擔，因而有時會比使用迴圈求解時來得**沒有效率**，若迴圈求解時使用到堆疊時，通常在程式碼上會比較**複雜**。

挑戰：

1. 利用遞迴作法，求解 12!。
   提示：12! = 1 x 2 x 3 x 4 x . . . . . x 10 x 11 x 12

```
12!=479001600
```

2. 利用遞迴作法，將輸入的數字反轉處理。
   提示：數字不包括 0

```
設定正整數：159827
反向整數值：728951
```