C语言变量的作用域，加深对全局变量和局部变量的理解

< [全局变量和局部变量](http://c.biancheng.net/view/1858.html)[C语言块级变量](http://c.biancheng.net/view/1860.html) >

[C语言中文网推出辅导班啦，包括「C语言辅导班、C++辅导班、算法/数据结构辅导班」，全部都是一对一教学：一对一辅导 + 一对一答疑 + 布置作业 + 项目实践 + 永久学习。QQ在线，随时响应！](http://fudao.biancheng.net/)

所谓**作用域（Scope）**，就是变量的有效范围，就是变量可以在哪个范围以内使用。有些变量可以在所有代码文件中使用，有些变量只能在当前的文件中使用，有些变量只能在函数内部使用，有些变量只能在 [for 循环](http://c.biancheng.net/view/172.html)内部使用。  
  
变量的作用域由变量的定义位置决定，在不同位置定义的变量，它的作用域是不一样的。本节我们只讲解两种变量，一种是只能在函数内部使用的变量，另一种是可以在所有代码文件中使用的变量。

**在函数内部定义的变量（局部变量）**

在函数内部定义的变量，它的作用域也仅限于函数内部，出了函数就不能使用了，我们将这样的变量称为局部变量（Local Variable）。函数的形参也是局部变量，也只能在函数内部使用。请看下面的例子：

1. #include <stdio.h>
2. int sum(int m, int n){
3. int i, sum=0;
4. //m、n、i、sum 都是局部变量，只能在 sum() 内部使用
5. **for**(i=m; i<=n; i++){
6. sum+=i;
7. }
8. **return** sum;
9. }
10. int main(){
11. int begin = 5, end = 86;
12. int result = sum(begin, end);
13. //begin、end、result 也都是局部变量，只能在 main() 内部使用
14. printf("The sum from %d to %d is %d\n", begin, end, result);
15. **return** 0;
16. }

m、n、i、sum 是局部变量，只能在 sum() 内部使用；begin、end、result 也是局部变量，只能在 main() 内部使用。  
  
对局部变量的两点说明：

* main() 也是一个函数，在 main() 内部定义的变量也是局部变量，只能在 main() 函数内部使用。
* 形参也是局部变量，将实参传递给形参的过程，就是用实参给局部变量赋值的过程，它和a=b; sum=m+n;这样的赋值没有什么区别。

**在所有函数外部定义的变量（全局变量）**

C语言允许在所有函数的外部定义变量，这样的变量称为全局变量（Global Variable）。  
  
全局变量的默认作用域是整个程序，也就是所有的代码文件，包括源文件（.c文件）和头文件（.h文件）。如果给全局变量加上 **static** 关键字，它的作用域就变成了当前文件，在其它文件中就无效了。我们目前编写的代码都是在一个源文件中，所以暂时不用考虑 static 关键字，后续我将会在《[C语言多文件编程](http://c.biancheng.net/c/150/" \t "_blank)》专题中详细讲解。  
  
【实例】定义一个函数，根据长方体的长宽高求它的体积以及三个面的面积。

1. #include <stdio.h>
2. //定义三个全局变量，分别表示三个面的面积
3. int s1 = 0, s2 = 0, s3 = 0;
4. int vs(int length, int width, int height){
5. int v; //体积
6. v = length \* width \* height;
7. s1 = length \* width;
8. s2 = width \* height;
9. s3 = length \* height;
10. **return** v;
11. }
12. int main(){
13. int v = 0;
14. v = vs(15, 20, 30);
15. printf("v=%d, s1=%d, s2=%d, s3=%d\n", v, s1, s2, s3);
16. v = vs(5, 17, 8);
17. printf("v=%d, s1=%d, s2=%d, s3=%d\n", v, s1, s2, s3);
18. **return** 0;
19. }

运行结果：  
v=9000, s1=300, s2=600, s3=450  
v=680, s1=85, s2=136, s3=40  
  
根据题意，我们希望借助一个函数得到四份数据：体积 v 以及三个面的面积 s1、s2、s3。遗憾的是，C语言中的函数只能有一个返回值，我们只能将其中的一份数据（也就是体积 v）放到返回值中，其它三份数据（也就是面积 s1、s2、s3）只能保存到全局变量中。  
  
C语言代码从前往后依次执行，变量在使用之前必须定义或者声明，全局变量 s1、s2、s3 定义在程序开头，所以在 vs() 和 main() 中都有效。  
  
在 vs() 中将求得的面积放到 s1、s2、s3 中，在 main() 中能够顺利取得它们的值，这说明：在一个函数内部修改全局变量的值会影响其它函数，全局变量的值在函数内部被修改后并不会自动恢复，它会一直保留该值，直到下次被修改。  
  
全局变量也是变量，变量只能保存一份数据，一旦数据被修改了，原来的数据就被冲刷掉了，再也无法恢复了，所以不管是全局变量还是局部变量，一旦它的值被修改，这种影响都会一直持续下去，直到再次被修改。

**关于变量的命名**

每一段可运行的C语言代码都包含了多个作用域，即使最简单的C语言代码也是如此。

1. int main(){
2. **return** 0;
3. }

这就是最简单的、可运行的C语言代码，它包含了两个作用域，一个是 main() 函数内部的局部作用域，一个是 main() 函数外部的全局作用域。  
  
C语言规定，在同一个作用域中不能出现两个名字相同的变量，否则会产生命名冲突；但是在不同的作用域中，允许出现名字相同的变量，它们的作用范围不同，彼此之间不会产生冲突。这句话有两层含义：

* 不同函数内部可以出现同名的变量，不同函数是不同的局部作用域；
* 函数内部和外部可以出现同名的变量，函数内部是局部作用域，函数外部是全局作用域。

1) 不同函数内部的同名变量是两个完全独立的变量，它们之间没有任何关联，也不会相互影响。请看下面的代码：

1. #include <stdio.h>
2. void func\_a(){
3. int n = 100;
4. printf("func\_a: n = %d\n", n);
5. n = 86;
6. printf("func\_a: n = %d\n", n);
7. }
8. void func\_b(){
9. int n = 29;
10. printf("func\_b: n = %d\n", n);
11. func\_a(); //调用func\_a()
12. printf("func\_b: n = %d\n", n);
13. }
14. int main(){
15. func\_b();
16. **return** 0;
17. }

运行结果：  
func\_b: n = 29  
func\_a: n = 100  
func\_a: n = 86  
func\_b: n = 29  
  
func\_a() 和 func\_b() 内部都定义了一个变量 n，在 func\_b() 中，n 的初始值是 29，调用 func\_a() 后，n 值还是 29，这说明 func\_b() 内部的 n 并没有影响 func\_a() 内部的 n。这两个 n 是完全不同的变量，彼此之间根本“不认识”，只是起了个相同的名字而已，这就好像明星撞衫，北京和云南都有叫李红的，赶巧了而已。  
  
2) 函数内部的局部变量和函数外部的全局变量同名时，在当前函数这个局部作用域中，全局变量会被“屏蔽”，不再起作用。也就是说，在函数内部使用的是局部变量，而不是全局变量。  
  
变量的使用遵循就近原则，如果在当前的局部作用域中找到了同名变量，就不会再去更大的全局作用域中查找。另外，只能从小的作用域向大的作用域中去寻找变量，而不能反过来，使用更小的作用域中的变量。  
  
下面我们通过一个具体的例子来说明：

1. #include <stdio.h>
2. int n = 10; //全局变量
3. void func1(){
4. int n = 20; //局部变量
5. printf("func1 n: %d\n", n);
6. }
7. void func2(int n){
8. printf("func2 n: %d\n", n);
9. }
10. void func3(){
11. printf("func3 n: %d\n", n);
12. }
13. int main(){
14. int n = 30; //局部变量
15. func1();
16. func2(n);
17. func3();
18. printf("main n: %d\n", n);
20. **return** 0;
21. }

运行结果：  
func1 n: 20  
func2 n: 30  
func3 n: 10  
main n: 30  
  
代码中虽然定义了多个同名变量 n，但它们的作用域不同，所有不会产生命名冲突。  
  
下面是对输出结果的分析：

* 对于 func1()，输出结果为 20，显然使用的是 func1() 内部的 n，而不是外部的 n。
* 调用 func2() 时，会把 main() 中的实参 n 传递给 func2() 中的形参 n，此时形参 n 的值变为 30。形参 n 也是局部变量，所以就使用它了。
* func3() 输出 10，使用的是全局变量，因为在 func3() 中不存在局部变量 n，所以编译器只能到函数外部，也就是全局作用域中去寻找变量 n。
* main() 中 printf() 语句输出 30，说明使用的是 main() 中的 n，而不是外部的 n。