用C语言指针作为函数返回值

<上一节 下一节> 关注我们： 微信公众号 新浪微博 QQ交流群：588321099

C语言允许函数的返回值是一个指针（地址），我们将这样的函数称为指针函数。下面的例子定义了一个函数 strlong()，用来返回两个字符串中较长的一个：

```
01.  #include <stdio.h>
02.  #include <string.h>
03.
04.  char *strlong(char *str1, char *str2){
05.      if(strlen(str1) >= strlen(str2)){
06.          return str1;
07.      }else{
08.          return str2;
09.      }
10.  }
11.
12.  int main(){
13.      char str1[30], str2[30], *str;
14.      gets(str1);
15.      gets(str2);
16.      str = strlong(str1, str2);
17.      printf("Longer string: %s\n", str);
18.
19.      return 0;
20.  }
```

运行结果：

```
C Language↙
c.biancheng.net↙
Longer string: c.biancheng.net
```

用指针作为函数返回值时需要注意的一点是，函数运行结束后会销毁在它内部定义的所有局部数据，包括局部变量、局部数组和形式参数，函数返回的指针请尽量不要指向这些数据，C语言没有任何机制来保证这些数据会一直有效，它们在后续使用过程中可能会引发运行时错误。请看下面的例子：

```
01.  #include <stdio.h>
02.
03.  int *func(){
04.      int n = 100;
05.      return &n;
06.  }
07.
08.  int main(){
09.      int *p = func(), n;
10.      n = *p;
11.      printf("value = %d\n", n);
12.      return 0;
13.  }
```

运行结果：

```
value = 100
```

n 是 func() 内部的局部变量，func() 返回了指向 n 的指针，根据上面的观点，func() 运行结束后 n 将被销毁，使用 *p 应该获取不到 n 的值。但是从运行结果来看，我们的推理好像是错误的，func() 运行结束后 *p 依然可以获取局部变量 n 的值，这个上面的观点不是相悖吗？

为了进一步看清问题的本质，不妨将上面的代码稍作修改，在第9~10行之间增加一个函数调用，看看会有什么效果：

```
01.  #include <stdio.h>
02.
03.  int *func(){
04.      int n = 100;
05.      return &n;
06.  }
07.
08.  int main(){
09.      int *p = func(), n;
10.      printf("c.biancheng.net\n");
11.      n = *p;
12.      printf("value = %d\n", n);
13.      return 0;
14.  }
```

运行结果：

```
c.biancheng.net
value = -2
```

可以看到，现在 p 指向的数据已经不是原来 n 的值了，它变成了一个毫无意义的甚至有些怪异的值。与前面的代码相比，该段代码仅仅是在 *p 之前增加了一个函数调用，这一细节的不同却导致运行结果有天壤之别，究竟是为什么呢？

前面我们说函数运行结束后会销毁所有的局部数据，这个观点并没错，大部分C语言教材也都强调了这一点。但是，这里所谓的销毁并不是将局部数据所占用的内存全部抹掉，而是程序放弃对它的使用权限，弃之不理，后面的代码可以随意使用这块内存。对于上面的两个例子，func() 运行结束后 n 的内存依然保持原样，值还是 100，如果使用及时也能够得到正确的数据，如果有其它函数被调用就会覆盖这块内存，得到的数据就失去了意义。

> 关于函数调用的原理以及函数如何占用内存的更多细节，我们将在《C语言和内存》专题中深入探讨，相信你必将有所顿悟，解开心中的谜团。

第一个例子在调用其他函数之前使用 *p 抢先获得了 n 的值并将它保存起来，第二个例子显然没有抓住机会，有其他函数被调用后才使用 *p 获取数据，这个时候已经晚了，内存已经被后来的函数覆盖了，而覆盖它的究竟是一份什么样的数据我们无从推断（一般是一个没有意义甚至有些怪异的值）。