```
10  int main()
11  {
12      switch(setjmp(buf)) {
13      case 0:
14          foo();
15          break;
16      case 1:
17          printf("Detected an error1 condition in foo\n");
18          break;
19      case 2:
20          printf("Detected an error2 condition in foo\n");
21          break;
22      default:
23          printf("Unknown error condition in foo\n");
24      }
25      exit(0);
26  }
27
28  /* Deeply nested function foo */
29  void foo(void)
30  {
31      if (error1)
32          longjmp(buf, 1);
33      bar();
34  }
35
36  void bar(void)
37  {
38      if (error2)
39          longjmp(buf, 2);
40  }
```

code/ecf/setjmp.c

图 8-43　（续）

longjmp 允许它跳过所有中间调用的特性可能产生意外的后果。例如，如果中间函数调用中分配了某些数据结构，本来预期在函数结尾处释放它们，那么这些释放代码会被跳过，因而会产生内存泄漏。

非本地跳转的另一个重要应用是使一个信号处理程序分支到一个特殊的代码位置，而不是返回到被信号到达中断了的指令的位置。图 8-44 展示了一个简单的程序，说明了这种基本技术。当用户在键盘上键入 Ctrl+C 时，这个程序用信号和非本地跳转来实现软重启。sigsetjmp 和 siglongjmp 函数是 setjmp 和 longjmp 的可以被信号处理程序使用的版本。

在程序第一次启动时，对 sigsetjmp 函数的初始调用保存调用环境和信号的上下文（包括待处理的和被阻塞的信号向量）。随后，主函数进入一个无限处理循环。当用户键入 Ctrl+C 时，内核发送一个 SIGINT 信号给这个进程，该进程捕获这个信号。不是从信号处理程序返回，如果是这样那么信号处理程序会将控制返回给被中断的处理循环，反之，处理程序完成一个非本地跳转，回到 main 函数的开始处。当我们在系统上运行这个程序时，得到以下输出：