因为 continue 语句会阻止索引变量 i 被修改，所以这段代码是无限循环。

B. 通用的解决方法是用 goto 语句替代 continue 语句，它会跳过循环体中余下的部分，直接跳到 update 部分：

```
/* Correct translation of for loop into while loop */
long sum = 0;
long i = 0;
while (i < 10) {
    if (i & 1)
        goto update;
    sum += i;
update:
    i++;
}
```

3.30　这个练习给你一个机会，推算出 switch 语句的控制流。要求你将汇编代码中的多处信息综合起来回答这些问题：

- 汇编代码的第 2 行将 x 加上 1，将情况(cases)的下界设置成 0。这就意味着最小的情况标号为－1。
- 当调整过的情况值大于 8 时，第 3 行和第 4 行会导致程序跳转到默认情况。这就意味着最大情况标号为－1＋8＝7。
- 在跳转表中，我们看到第 6 行的表项(情况值 3)与第 9 行的表项(情况值 6)都以第 4 行的跳转指令作为同样的目标(.L2)，表明这是默认的情况行为。因此，在 switch 语句体中缺失了情况标号 3 和－6。
- 在跳转表中，我们看到第 3 行和第 10 行上的表项有相同的目的。这对应于情况标号 0 和 7。
- 在跳转表中，我们看到第 5 行和第 7 行上的表项有相同的目的。这对应于情况标号 2 和 4。

从上述推理，我们得出如下结论：

A. switch 语句体中的情况标号值为－1、0、1、2、4、5 和 7。

B. 目标为 .L5 的情况标号为 0 和 7。

C. 目标为 .L7 的情况标号为 2 和 4。

3.31　逆向工程编译出 switch 语句，关键是将来自汇编代码和跳转表的信息结合起来，理清不同的情况。从 ja 指令(第 3 行)可知，默认情况的代码的标号是 .L2。我们可以看到，跳转表中只有另一个标号重复出现，就是 .L5，因此它一定是情况 C 和 D 的代码。代码在第 8 行落入下面的情况，因而标号 .L7 符合情况 A，标号 .L3 符合情况 B。只剩下标号 .L6，符合情况 E。

原始的 C 代码如下：

```
void switcher(long a, long b, long c, long *dest)
{
    long val;
    switch(a) {
    case 5:
        c = b ^ 15;
        /* Fall through */
    case 0:
        val = c + 112;
        break;
    case 2:
    case 7:
        val = (c + b) << 2;
        break;
    case 4:
        val = a;
        break;
    default:
        val = b;
    }
    *dest = val;
}
```