C. 说明如何修改这段代码好让它能可靠地工作。将测试语句改成：

```
return strlen(s) > strlen(t);
```

2.27　这个函数是对确定无符号加法是否溢出的规则的直接实现。

```
/* Determine whether arguments can be added without overflow */
int uadd_ok(unsigned x, unsigned y) {
    unsigned sum = x+y;
    return sum >= x;
}
```

2.28　本题是对算术模 16 的简单示范。最容易的解决方法是将十六进制模式转换成它的无符号十进制值。对于非零的 x 值，我们必须有 $(-_4^u x)+x=16$。然后，我们就可以将取补后的值转换回十六进制。

x		$-_4^u x$	
十六进制	十进制	十进制	十六进制
0	0	0	0
5	5	11	B
8	8	8	8
D	13	3	3
F	15	1	1

2.29　本题的目的是确保你理解了补码加法。

x	y	$x+y$	$x+_5^t y$	情况
−12 [10100]	−15 [10001]	−27 [100101]	5 [00101]	1
−8 [11000]	−8 [11000]	−16 [110000]	−16 [10000]	2
−9 [10111]	8 [01000]	−1 [111111]	−1 [11111]	2
2 [00010]	5 [00101]	7 [000111]	7 [00111]	3
12 [01100]	4 [00100]	16 [010000]	−16 [10000]	4

2.30　这个函数是对确定补码加法是否溢出的规则的直接实现。

```
/* Determine whether arguments can be added without overflow */
int tadd_ok(int x, int y) {
    int sum = x+y;
    int neg_over = x <  0 && y <  0 && sum >= 0;
    int pos_over = x >= 0 && y >= 0 && sum <  0;
    return !neg_over && !pos_over;
}
```

2.31　通过学习 2.3.2 节，你的同事可能已经学到补码加会形成一个阿贝尔群，因此表达式 (x+y)-x 求值得到 y，无论加法是否溢出，而 (x+y)-y 总是会求值得到 x。

2.32　这个函数会给出正确的值，除了当 y 等于 *TMin* 时。在这个情况下，我们有 -y 也等于 *TMin*，因此函数 tadd_ok 会认为只要 x 是负数时，就会溢出，而 x 为非负数时，不会溢出。实际上，情况恰恰相反：当 x 为负数时，tsub_ok(x, *TMin*)为 1；而当 x 为非负时，它为 0。

这个练习说明，在函数的任何测试过程中，*TMin* 都应该作为一种测试情况。

2.33　本题使用非常小的字长来帮助你理解补码的非。

对于 $w=4$，我们有 $TMin_4=-8$。因此 −8 是它自己的加法逆元，而其他数值是通过整数非来取非的。