```
linux> gcc -o foobar3 foo3.c bar3.c
linux> ./foobar3
x = 15212
```

如果 x 有两个弱定义，也会发生相同的事情(规则 3)：

```
1  /* foo4.c */
2  #include <stdio.h>
3  void f(void);
4
5  int x;
6
7  int main()
8  {
9      x = 15213;
10     f();
11     printf("x = %d\n", x);
12     return 0;
13 }
```

```
1  /* bar4.c */
2  int x;
3
4  void f()
5  {
6      x = 15212;
7  }
```

规则 2 和规则 3 的应用会造成一些不易察觉的运行时错误，对于不警觉的程序员来说，是很难理解的，尤其是如果重复的符号定义还有不同的类型时。考虑下面这个例子，其中 x 不幸地在一个模块中定义为 int，而在另一个模块中定义为 double：

```
1  /* foo5.c */
2  #include <stdio.h>
3  void f(void);
4
5  int y = 15212;
6  int x = 15213;
7
8  int main()
9  {
10     f();
11     printf("x = 0x%x y = 0x%x \n",
12            x, y);
13     return 0;
14 }
```

```
1  /* bar5.c */
2  double x;
3
4  void f()
5  {
6      x = -0.0;
7  }
```