```
1  /* bar1.c */
2  int main()
3  {
4      return 0;
5  }
```

在这个情况中，链接器将生成一条错误信息，因为强符号 main 被定义了多次(规则 1)：

```
linux> gcc foo1.c bar1.c
/tmp/ccq2Uxnd.o: In function `main':
bar1.c:(.text+0x0): multiple definition of `main'
```

相似地，链接器对于下面的模块也会生成一条错误信息，因为强符号 x 被定义了两次(规则 1)：

```
1  /* foo2.c */
2  int x = 15213;
3
4  int main()
5  {
6      return 0;
7  }
```

```
1  /* bar2.c */
2  int x = 15213;
3
4  void f()
5  {
6  }
```

然而，如果在一个模块里 x 未被初始化，那么链接器将安静地选择在另一个模块中定义的强符号(规则 2)：

```
1  /* foo3.c */
2  #include <stdio.h>
3  void f(void);
4
5  int x = 15213;
6
7  int main()
8  {
9      f();
10     printf("x = %d\n", x);
11     return 0;
12 }
```

```
1  /* bar3.c */
2  int x;
3
4  void f()
5  {
6      x = 15212;
7  }
```

在运行时，函数 f 将 x 的值由 15213 改为 15212，这会给 main 函数的作者带来不受欢迎的意外！注意，链接器通常不会表明它检测到多个 x 的定义：