

C语言中do{...}while(0)的妙用

原创

单片机嵌入式爱好者

于 2018-03-22 13:19:59 发布

3642

今天看了些有关do{...}while(0)妙用的文章，觉得写的很好，这里总结下分享给大家。

这里分享的有3个用法，分别是：

- 1.避免空的宏定义在编译时出现warning。
- 2.多个语句一起，定义成一个宏时，增加代码适应(特别是条件语句调用这类宏要注意)
- 3.避免部分goto语句的使用

1. 避免空的宏定义在编译时出现warning

```
1 //例如：
2 #define foo() do{}while(0)
```

2. 多个语句一起，定义成一个宏时，增加代码适应(特别是条件语句调用这类宏要注意)，以下if(0)和if(1)在实际应用时是if(表达式)，表示表达式假和真。

```
1 //例如：一个宏包含以下两个语句，
2 #define foo() \
3 fun1();\
4 fun2;
```

编译器预处理的时候

```
1 if(1)foo();
2 //此时就相当于下面的语句，
3 if(0)fun1();
4     fun2();//逻辑上多执行的代码，会导致系统BUG
5     ;//逻辑上多执行的代码
```

如果使用do{...}while(0)就可以解决上面的问题

```
1 #define foo() \
2 do{ \
3 fun1(); \
4 fun2(); \
5 }while(0)
6 //对于下面的语句
7 if(0) foo();
8 //编译后的执行如下：
9 if(0) do{
10     fun1();
11     fun2();
12 }while(0);
```

这样就不会出现上面那种有逻辑上不该执行的代码被执行的问题。当然这里也可以用其他方法避免这个问题，比如加大括号{}

```
#define foo() {fun1();fun2;}
```

编译器会预处理下面语句

```
1 if(0) foo();
2 //编译后的执行如下：
3 if(0) {
4     fun1();
5     fun2();
6 }
7 ;//会多个;号，但是也没有逻辑上的问题
```

语句块宏定义时注意的就是这些，另外在写if语句时，尽量后面要加大括号，避免出错，例如上面的if(0){foo();}.加大括号{}也不会有问题。

3. 避免部分goto语句的使用

```
1 //例如：如果一个函数要分配一些资源，然后中途遇到错误，要退出函数，退出前要释放资源，代码结构可能如下：
2 bool foo(){
3     int *p = (int*)malloc(5*sizeof(int));
```

```

4 |     bool bOk = true; 5 | //执行并处理错误
6 |     bOk = fun1();
7 |     if(!bOk){
8 |         free(p);
9 |         p=NULL;
10 |        return false;
11 |    }
12 |    bOk = fun2();
13 |    if(!bOk){
14 |        free(p);
15 |        p=NULL;
16 |        return false;
17 |    }
18 |    bOk = fun3();
19 |    if(!bOk){
20 |        free(p);
21 |        p=NULL;
22 |        return false;
23 |    }
24 | //.....
25 | //执行成功，释放资源并返回ture
26 |     free(p);
27 |     p = NULL;
28 |     return true;
29 | }

```

这里就觉得很多代码冗余，然而使用沟通可以很好的解决冗余的部分，代码如下：

```

1 | bool foo(){
2 |     int *p = (int*)malloc(5*sizeof(int));
3 |     bool bOk = true;
4 |     //执行并处理错误
5 |     if(!fun1()) goto errorlable;
6 |     if(!fun2()) goto errorlable;
7 |     if(!fun3()) goto errorlable;
8 |     //.....
9 |     //执行成功，释放资源并返回ture
10 |    free(p);
11 |    p = NULL;
12 |    return true;
13 | //冗余部分的，错误返回代码
14 | errorlable:
15 |     free(p);
16 |     p = NULL;
17 |     return false;
18 | }

```

然后C语言中过多的使用goto语句会提高程序的灵活性，繁杂点的程序会让程序员捉摸不定，程序跳来跳出，难以捉摸，容易逻辑上产生混淆从而出现BUG。对于上面的这种情况使用do {...}while(0)就可以很好的解决这些跳来跳出的问题，代码结构如下：

```

1 | bool foo(){
2 | //分配资源
3 |     int *p = (int*)malloc(5*sizeof(int));
4 |     bool bOk = true;
5 | //执行并处理错误
6 | do{
7 |     bOk = fun1();
8 |     if(!bOk)break;
9 |     bOk = fun2();
10 |    if(!bOk)break;
11 |    bOk = fun3();
12 |    if(!bOk)break;
13 |    //.....
14 | }while(0);
15 |
16 | //释放资源并返回bOk
17 |     free(p);
18 |     p = NULL;
19 |     return bOk;
20 | }

```