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

```
//例如：
#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。

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

编译器预处理的时候

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

```
#define foo() \
do{ \
fun1(); \
fun2(); \
}while(0)
//对于下面的语句
if(0) foo();
//编译后的执行如下：
if(0) do{
        fun1();
        fun2();
        }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

```
if(0) foo();
//编译后的执行如下：
if(0) {
        fun1();
        fun2();
        }
;//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();},加上大括号{}也不会有问题。

3. 避免部分goto语句的使用

```
//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：
bool foo(){
  int *p = (int*)malloc(5*sizeof(int));
```

```
  bool bOk = true;
  //执行并处理错误
  bOk = fun1();
  if(!bOk){
    free(p);
    p=NULL;
    return false;
  }
  bOk = fun2();
  if(!bOk){
    free(p);
    p=NULL;
    return false;
  }
  bOk = fun3();
  if(!bOk){
    free(p);
    p=NULL;
    return false;
  }
//......
//执行成功，释放资源并返回ture
  free(p);
  p = NULL;
  return true;
}
```

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：

```
bool foo(){
  int *p = (int*)malloc(5*sizeof(int));
  bool bOk = true;
//执行并处理错误
  if(!fun1()) goto errorlable;
  if(!fun2()) goto errorlable;
  if(!fun3()) goto errorlable;
//......
//执行成功，释放资源并返回ture
  free(p);
  p = NULL;
  return true;
//冗余部分的，错误返回代码
errorlable:
  free(p);
  p = NULL;
 return false;
}
```

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：

```
bool foo(){
//分配资源
  int *p = (int*)malloc(5*sizeof(int));
  bool bOk = true;
//执行并处理错误
do{
  bOk = fun1();
  if(!bOk)break;
  bOk = fun2();
  if(!bOk)break;
  bOk = fun3();
  if(!bOk)break;
  //......
}while(0);

//释放资源并返回bOk
  free(p);
  p = NULL;
  return bOk;
}
```