跟涛哥一起学嵌入式 04：一道面试题，测出你的C语言功底

文档说明	作者	日期
来自微信公众号：宅学部落(armLinuxfun)	wit	2018.7.3
嵌入式视频教程淘宝店：https://wanglitao.taobao.com/		
联系微信：brotau(宅学部落)		

嵌入式C语言面试题中，大家经常会看到宏定义的考题。比如：定义一个宏，求两个数中的最大数。别小看这个考题，虽然简单，但是它却陷阱不断，时刻在考验着你的C语言编程功底！根据你的答案，面试官对你的印象肯定不一样。那下面我们看看各个不同版本的答案吧。

合格

对于学过C语言的同学，写出这个宏基本上不是什么难事，使用条件运算符就能完成：

```
#define  MAX(x,y)  x > y ? x : y
```

这是最基本的C语言语法，如果连这个也写不出来，估计场面会比较尴尬。面试官为了缓解尴尬，一般会对你说：小伙子，你很棒，回去等消息吧，有消息，我们会通知你！这时候，你应该明白：不用再等了，赶紧把这篇文章看完，接着面下家。这个宏能写出来，也不要觉得你很牛X，因为这只能说明你有了C语言的基础，但还有很大的进步空间。比如，我们写一个程序，验证一下我们定义的宏是否正确：

```
#define MAX(x,y) x > y ? x : y
int main(void)
{
    printf("max=%d",MAX(1,2));
    printf("max=%d",MAX(2,1));
    printf("max=%d",MAX(2,2));
    printf("max=%d",MAX(1!=1,1!=2));
    return 0;
}
```

测试程序么，我们肯定要把各种可能出现的情况都测一遍。这不，测试第4行语句，当宏的参数是一个表达式，发现实际运行结果为max=0,跟我们预期结果max=1不一样。这是因为，宏展开后，就变成了这个样子：

```
printf("max=%d",1!=1>1!=2?1!=1:1!=2);
```

因为比较运算符 > 的优先级为6，大于 !=(优先级为7)，所以展开的表达式，运算顺序发生了改变，结果就跟我们的预期不一样了。为了避免这种展开错误，我们可以给宏的参数加一个小括号()来防止展开后，表达式的运算顺序发生变化。这样的宏才能算一个合格的宏：

```
#define MAX(x,y) (x) > (y) ? (x) : (y)
```

中等

上面的宏，只能算合格，但还是存在漏洞。比如，我们使用下面的代码测试：

```
#define MAX(x,y) (x) > (y) ? (x) : (y)
int main(void)
{
    printf("max=%d",3 + MAX(1,2));
    return 0;
}
```

在程序中，我们打印表达式 3 + MAX(1, 2) 的值，预期结果应该是5，但实际运行结果却是1。我们展开后，发现同样有问题：

```
3 + (1) > (2) ? (1) : (2);
```

因为运算符 + 的优先级大于比较运算符 >，所以这个表达式就变为4>2?1:2，最后结果为1也就见怪不怪了。此时我们应该继续修改这个宏：

```
#define MAX(x,y) ((x) > (y) ? (x) : (y))
```

使用小括号将宏定义包起来，这样就避免了当一个表达式同时含有宏定义和其它高优先级运算符时，破坏整个表达式的运算顺序。如果你能写到这一步，说明你比前面那个面试合格的同学强，前面那个同学已经回去等消息了，我们接着面试下一轮。

良好

上面的宏，虽然解决了运算符优先级带来的问题，但是仍存在一定的漏洞。比如，我们使用下面的测试程序来测试我们定义的宏：

```
#define MAX(x,y) ((x) > (y) ? (x) : (y))
int main(void)
{
    int i = 2;
    int j = 6;
    printf("max=%d",MAX(i++,j++));
    return 0;
}
```

在程序中，我们定义两个变量 i 和 j，然后比较两个变量的大小，并作自增运算。实际运行结果发现max = 7，而不是预期结果max = 6。这是因为变量 i 和 j 在宏展开后，做了两次自增运算，导致打印出 i 的值为7。

遇到这种情况，那该怎么办呢？ 这时候，语句表达式就该上场了。我们可以使用语句表达式来定义这个宏，在语句表达式中定义两个临时变量，分别来暂储 i 和 j 的值，然后进行比较，这样就避免了两次自增、自减问题。

```
#define MAX(x,y)({     \
    int _x = x;        \
    int _y = y;        \
    _x > _y ? _x : _y; \
})
int main(void)
{
    int i = 2;
    int j = 6;
    printf("max=%d",MAX(i++,j++));
    return 0;
}
```

在语句表达式中，我们定义了2个局部变量*x、*y来存储宏参数 x 和 y 的值，然后使用 _x 和 _y 来比较大小，这样就避免了 i 和 j 带来的2次自增运算问题。

你能坚持到了这一关，并写出这样自带BGM的宏，面试官心里可能已经有了给你offer的意愿了。但此时此刻，千万不要骄傲！为了彻底打消面试官的心理顾虑，我们需要对这个宏继续优化。

优秀

在上面这个宏中，我们定义的两个临时变量数据类型是int型，只能比较两个整型的数据。那对于其它类型的数据，就需要重新再定义一个宏了，这样太麻烦了！我们可以基于上面的宏继续修改，让它可以支持任意类型的数据比较大小：

```
#define MAX(type,x,y)({     \
    type _x = x;        \
    type _y = y;        \
    _x > _y ? _x : _y; \
})
int main(void)
{
    int i = 2;
    int j = 6;
    printf("max=%d\n",MAX(int,i++,j++));
    printf("max=%f\n",MAX(float,3.14,3.15));
    return 0;
}
```

在这个宏中，我们添加一个参数：type，用来指定临时变量 _x 和 _y 的类型。这样，我们在比较两个数的大小时，只要将2个数据的类型作为参数传给宏，就可以比较任意类型的数据了。如果你能在面试中，写出这样的宏，面试官肯定会非常高兴，他一般会跟你说：小伙子，稍等，待会HR会跟你谈待遇问题。

还能不能更牛逼？

如果你想薪水拿得高一点，待遇好一点，此时不应该骄傲，你应该大手一挥：且慢，我还可以更牛逼！

上面的宏定义中，我们增加了一个type类型参数，来兼容不同的数据类型，此时此刻，为了薪水，我们应该把这个也省去。如何做到？使用typeof就可以了，typeof是GNU C新增的一个关键字，用来获取数据类型，我们不用传参进去，让typeof直接获取！

```
#define max(x, y) ({    \
    typeof(x) _x = (x); \
    typeof(y) _y = (y); \
    (void) (&_x == &_y);\
    _x > _y ? _x : _y; })
```

在这个宏定义中，使用了typeof关键字用来获取宏的两个参数类型。干货在(void) (&*x* == &y);这句话，简直是天才般的设计！一是用来给用户提示一个警告，对于不同类型的指针比较，编译器会给一个警告，提示两种数据类型不同；二是，当两个值比较，比较的结果没有用到，有些编译器可能会给出一个warning，加个(void)后，就可以消除这个警告！

此刻，面试官看到你的这个宏，估计会倒吸一口气：乖乖，果然是后生可畏，这家伙比我还牛逼！你等着，HR待会过来跟你谈薪水！

恭喜你，拿到offer了！

此时的你是不是心满意足了？不！还有继续 优化的空间！

打造一个趋近完美的宏

以上的宏解决了自增自减运算符 ++/-- 带来的一系列问题。但也不是十全十美，还是有漏洞：在宏内部的语句表达中，我们定义了2个临时变量 _比如当我们使用下面的代码时：

```
max(x, _x)
```

当宏展开后，第二个参数就与宏内部定义的临时变量同名了，这就影响宏最后的结果。因此，为了防止用户传入的参数跟宏内部的临时变量产生同名冲突，我们可以将宏内部的临时变量尽量定义得复杂一些，降低同名的概率，比如Linux 内核中max宏的定义：

```
#define max(x, y) ({                \
    typeof(x) _max1 = (x);          \
    typeof(y) _max2 = (y);          \
    (void) (&_max1 == &_max2);      \
    _max1 > _max2 ? _max1 : _max2; })
```

在上面的宏定义中，虽然临时变量 _max1 和 *max2 比我们上面的* _x 和 _y 好点，也只是更进一步降低跟用户的传参同名冲突的概率，但是还是不能完全杜绝。极端一点，我们可以把这两个变量定义得无比长、无比奇葩，只要不超过C标准规定以的标识符最大长度j就可以：

```
________tmp_________for________max______
```

再奇葩的程序员，再猪一样的队友，哪怕是团队毒瘤、代码杀手，估计也不会定义这样的变量吧！这样同名冲突的概率就大大降低了，但是还是不能完全杜绝，算是Linux内核的一个小漏洞吧。下载新版本的Linux内核，发现已经堵住了这个漏洞：

```
#define __max(t1, t2, max1, max2, x, y) ({              \
    t1 max1 = (x);                                      \
    t2 max2 = (y);                                      \
    (void) (&max1 == &max2);                            \
    max1 < max2 ? max1 : max2; })

#define ___PASTE(a,b) a##b
#define __PASTE(a,b) ___PASTE(a,b)

#define __UNIQUE_ID(prefix) __PASTE(__PASTE(__UNIQUE_ID_, prefix), __COUNTER__)

#define max(x, y)                                           \
    __max(typeof(x), typeof(y),                             \
          __UNIQUE_ID(max1_), __UNIQUE_ID(max2_),           \
          x, y)
```

在新版的宏中，内部的临时变量不再由程序员自己定义，而是让编译器生成一个独一无二的变量，这样就避免了同名冲突的风险。宏__UNIQUE_ID的作用就是生成了一个独一无二的变量，确保了临时变量的唯一性。关于它的使用，可以参考下面的文章，写的很好：

Linux kernel中的min和max宏gaomf.cn

是不是已经完美了？

新版本Linux内核堵住了临时变量可能带来的同名冲突的漏洞，但是是不是就完美了呢？还是不一定！针对Linux内核中宏的新版本，最近又引发各种争论，比如针对常量、变长数组问题等，看看他们提交的各种更新的版本吧：

Variable-length arrays and the max() messlwn.netThe joy of max()lwn.net

还有这种更加复杂的max宏的实现：

```
#define __typecheck(x, y) \
      (!!(sizeof((typeof(x)*)1 == (typeof(y)*)1)))

  #define __is_constant(x) \
  (sizeof(int) == sizeof(*(1 ? ((void*)((long)(x) * 0l)) : (int*)1)))

  #define __no_side_effects(x, y) \
      (__is_constant(x) && __is_constant(y))

  #define __safe_cmp(x, y) \
      (__typecheck(x, y) && __no_side_effects(x, y))

  #define __cmp(x, y, op) ((x) op (y) ? (x) : (y))

  #define __cmp_once(x, y, op) ({ \
      typeof(x) __x = (x);    \
      typeof(y) __y = (y);    \
      __cmp(__x, __y, op); })

  #define __careful_cmp(x, y, op)         \
      __builtin_choose_expr(__safe_cmp(x, y), \
                  __cmp(x, y, op), __cmp_once(x, y, op))

  #define max(x, y)   __careful_cmp(x, y, >)
```

小结：

上面以一个宏为例子，意在说明，对一门语言的掌握是永无止境的，就算你把当前所有的C语言知识点、编程技能都掌握了，C语言也是不断更新的、C标准也是不断更新变化的。编程技巧、编程技能也是不断进步的。

而自学往往是最有效的学习方法，但是前提是你要有好的学习资料、学习方法、学习目标，再加上刻意练习和实时反馈。否则，就是两眼一抹黑，不知道自己学得怎么样、学到什么水平了、学了有什么用、学得对不对。其实还有一种比较有效的学习方法，找个行业内的工程师带一带、参考优秀的书籍、教程学一学、再结合几个项目练一练，就知道什么该学、要学到什么程度，而且可以大大提高学习效率。

本文根据《C语言嵌入式Linux高级编程》第5期：C标准及Linux内核中的C语法扩展部分视频改编。《跟涛哥一起学嵌入式》，会持续跟大家分享嵌入式相关技术、学习方法、学习路线、求职面试等

专注嵌入式、Linux精品教程：https://wanglitao.taobao.com/
嵌入式技术教程博客：http://zhaixue.cc/
联系 QQ：3284757626
嵌入式技术交流QQ群：475504428
微信公众号：宅学部落(armlinuxfun)