##### #define宏的缺点，在c++中如何更好的定义变量和函数型宏？

作者：小林coding  
链接：https://www.zhihu.com/question/366984608/answer/1001746801  
来源：知乎  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

最近正好在看 《Effective C++中文版 第三版》，其中条款 2 就提及你所问的问题。

以下内容是我阅读该书总结的笔记。

## 条款02：尽量以const，enum，inline替换#define

### const 替换 #define

### 问题分析：

首先我们要清楚程序的编译重要的三个阶段： **预处理阶段，编译阶段和链接阶段**

#define 是不被视为语言的一部分，它在程序编译阶段中的**预处理阶段**的作用，就是做简单的替换，将 A 内容替换成 B 内容。

#define PI 3.14

如上述的 PI 宏定义，在程序编译时，编译器在预处理阶段时，会先将源码中所有 PI 宏定义替换成 3.14。

程序编译在预处理阶段后，才进行真正的编译阶段。**在有的编译器，运用了此 PI 常量，如果遇到了编译错误，那么这个错误信息也许会提到 3.14 而不是 PI**，这是会让人困惑的，特别是在项目大的情况下。

### 解决之道：

解决之道是以 const 定义一个常量 替换上述的宏（#define）：

const doule Pi = 3.14;

作为一个语言变量，Pi 肯定会被编译器看到，出错的时候可以很清楚知道，是这个变量导致的问题。

如果是定义常量字符串，则必须要 const 两次，目的是为了防止指针所指内容和指针自身不能被改变：

const char\* const myName = "小林coding";

如果是定义常量 string，则只需要在最前面加一次 const，形式如下：

const std::string myName("小林coding");

### enum 替换 #define

### 问题分析：

还有另外一点宏无法涉及的，就是我们无法利用 #define 创建一个 class 专属常量，因为 #define 并不重视作用域。

对于 class 里定义常量时，我们依然使用 static + const，形式如下：

class Student {private:

static const int num = 10;

int scores[num];};

const int Student::num; // static 成员变量，需要进行声明

### 解决之道：

如果不想外部获取到 class 专属常量的内存地址，可以使用 enum 的方式定义常量，enum 会帮你约束这个条件，因为取一个 enum 的地址是不合法的，形式如下：

class Student {private:

enum { num = 10 };

int scores[num];};

### inline 替换 #define

### 问题分析：

另外一个常见的 #define 误用情况是以它实现宏函数，它不会招致函数调用带来的开销，但是用 #define 编写宏函数容易出错，如下用宏定义写的求最大值的函数：

#define MAX(a, b) ( { (a) > (b) ? (a) : (b); } ) // 求最大值

这般长相的宏有着太的缺点，比如在下面调用例子：

int a = 6, b = 5;int max = MAX(a++, b);

std::cout << max << std::endl;std::cout << a << std::endl;

输出结果（以下结果是错误的）：

7 // 正确的答案是 max 输出 68 // 正确的答案是 a 输出 7

要解释出错的原因很简单，我们把 MAX 宏做简单替换：

int max = ( { (a++) > (b) ? (a++) : (b); } ); // a 被累加了2次！

在上述替换后，可以发现 a 被累加了 2 次。我们可以通过改进 MAX 宏，来解决这个问题：

#define MAX(a, b) ({ \ \_\_typeof(a) \_\_a = (a), \_\_b = (b); \ \_\_a > \_\_b ? \_\_a : \_\_b; \})

简单说明下，上述的 \_\_typeof 可以根据变量的类型来定义一个相同类型的变量。改进后的 MAX 宏，输出的是正确的结果，max 输出 6，a 输出 7。

虽然改进的后 MAX 宏，解决了问题，但是这种宏的长相就让人困惑。

### 解决之道：

用 template inline 的方式，写出短小的函数：

template<typename T>inline T max(const T& a, const T& b){

return a > b? a : b;}

max 是一个真正的函数，它遵循作用域和访问规则，所以不会出现变量被多次累加的现象。

## 请记住

* 对于单纯常量，最好以 const 对象或 enum 替换 #define；
* 对于形式函数的宏，最好改用 inline 函数替换 #define。