55 Macros in C++

1. 预处理阶段

05 How C++ Works
带有#的为*preprocessor statement*，即预处理指令。
该类指令发生在真正的编译之前，当编译器收到一个源文件时，做的第一件事情就是预处理所有预处理指令。

预处理阶段基本上是一个文本编辑阶段，在这个阶段我们可以控制给编译器什么代码，这就是*macro*（宏）的用武之地了。
我们能做的就是写一些宏，它将代码中的文本替换为其它东西，这基本就像遍历我们的代码然后执行查找和替换。
（所以模板会比宏评估得更晚一些）

你使用宏的方式取决于你的个人爱好，如果你用了很多宏，代码可能会比较难理解。不要用太多的C++特性，尤其是当我们进入更高级的特性时，你不需要向所有人炫耀你知道所有的C++特性，用更多的特性也不是写好代码的方式。
之前的章节中我定义过

```C++
#define LOG(x) std::cout << x << std::endl;
```

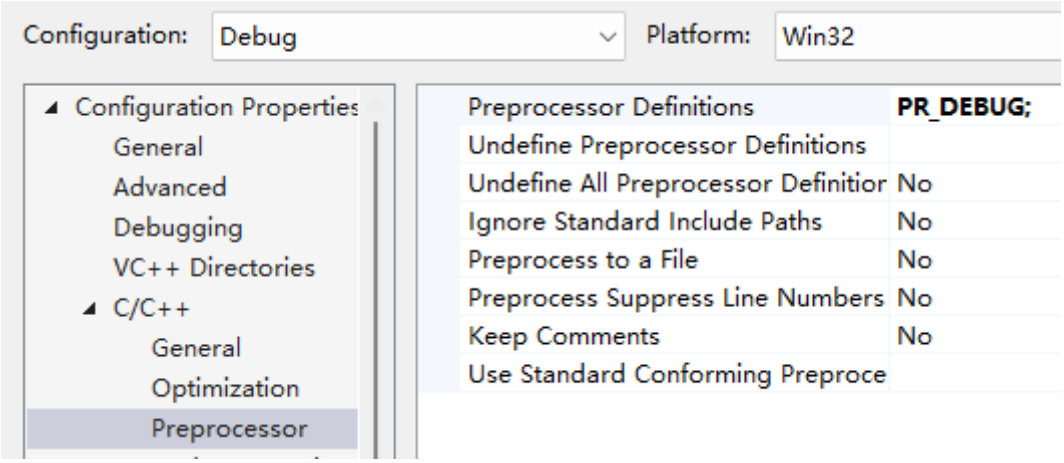
这个就是宏。

2. 宏

```C++
#define WAIT std::cin.get()    // 每次遇到WAIT这个词，就粘贴后面的代码

int main()
{
    WAIT;    // 送入编译器的代码其实是：  std::cin.get();
}
```

我们程序如何运行，编译器看到的以及代码是如何编译的都是一样的，编译器看不到任何区别，因为我们实际上做的只是改变了文本的生成方式。
不过这并不是一个好例子，因为不是你应该使用预处理的方式，很愚蠢，你完全应该写实际的代码，这里主要是理解预处理的工作方式就行了。

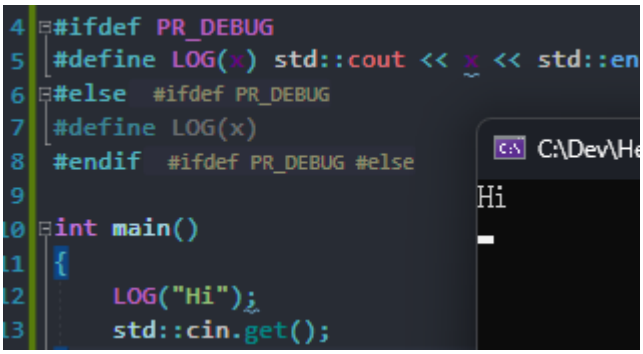
就像是在预编译阶段的函数，宏函数的优点：没有普通函数保存寄存器和参数传递，返回值的开销，展开后的代码效率高，速度快。
包括上面的LOG(x)可能也不会这么用，然而在更加复杂的日志系统中，比如游戏引擎或者应用程序、框架，你可能会在日志系统中使用宏，因为你记录日志的方法可能基于你的设置会发生变化，我们可能希望Debug版本中有日志，而Release版本中去掉日志，而宏可以做到这一点：



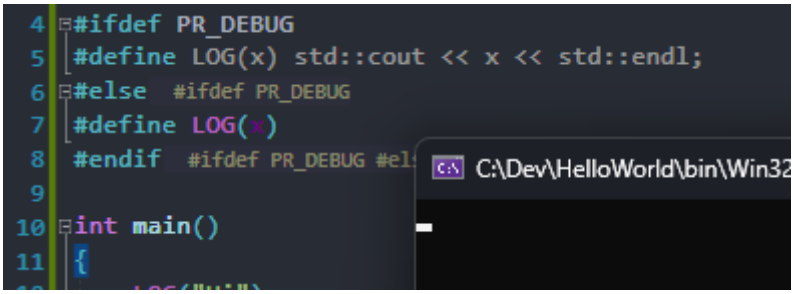
同理给Release下加入PR_RELEASE。

```cpp
#ifdef PR_DEBUG
#define LOG(x) std::cout << x << std::endl;
#else
#define LOG(x)
#endif
```

Debug模式下：



Release模式下：



这就是宏的很实用的用法。

但这里面有一个问题，就是如果仅仅是定义，可能会误解它的意思。一般可以用`#define`来做，而不是`#ifdef`这个很多情况下会更糟一些的东西。

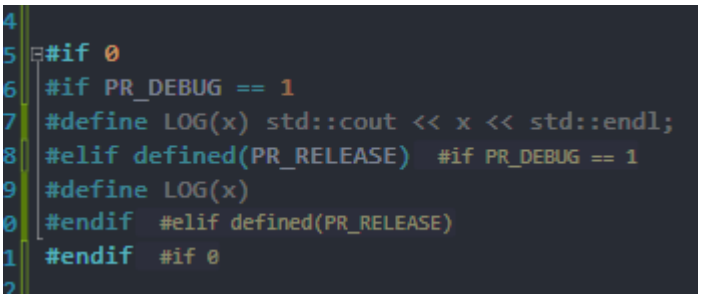
```C++
#define PR_DEBUG 0

#if PR_DEBUG == 1
```

这样你就不用删除它，只用修改即可控制PR_DEBUG，看的更清楚。也可以在properies中改加上PR_DEBUG=1

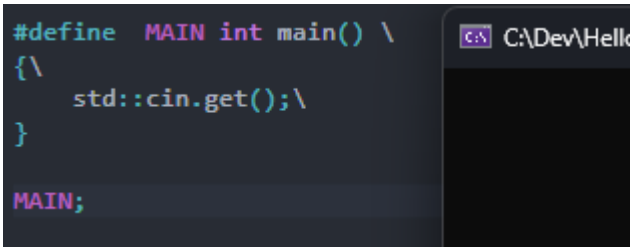
```C++
#if PR_DEBUG == 1
#define LOG(x) std::cout << x << std::endl;
#elif defined(PR_RELEASE)     // 可以用elif
#define LOG(x)
#endif
```

3. 宏的特殊用法

我们还可以利用预处理器和宏来删除特定代码：



宏可以分段写，通过`\`来表示换行（反斜杠是Enter键的转义）：



注意反斜杠后面不要有空格，不然就变成对空格转义了。

如果想追踪内存是什么时候分配的，可以给设置一个`new`的宏，其中记录这是第几行代码和分配了多少。