cassert 头文件，也最好不要为了其他目的使用 assert。很多头文件都包含了 cassert，这就意味着即使你没有直接包含 cassert，它也很有可能通过其他途径包含在你的程序中。

assert 宏常用于检查“不能发生”的条件。例如，一个对输入文本进行操作的程序可能要求所有给定单词的长度都大于某个阈值。此时，程序可以包含一条如下所示的语句：

```
assert(word.size() > threshold);
```

NDEBUG 预处理变量

assert 的行为依赖于一个名为 NDEBUG 的预处理变量的状态。如果定义了 NDEBUG，则 assert 什么也不做。默认状态下没有定义 NDEBUG，此时 assert 将执行运行时检查。

我们可以使用一个#define 语句定义 NDEBUG，从而关闭调试状态。同时，很多编译器都提供了一个命令行选项使我们可以定义预处理变量：

```
$ CC -D NDEBUG main.C # use /D with the Microsoft compiler
```

这条命令的作用等价于在 main.c 文件的一开始写#define NDEBUG。

定义 NDEBUG 能避免检查各种条件所需的运行时开销，当然此时根本就不会执行运行时检查。因此，assert 应该仅用于验证那些确实不可能发生的事情。我们可以把 assert 当成调试程序的一种辅助手段，但是不能用它替代真正的运行时逻辑检查，也不能替代程序本身应该包含的错误检查。

除了用于 assert 外，也可以使用 NDEBUG 编写自己的条件调试代码。如果 NDEBUG 未定义，将执行#ifndef 和#endif 之间的代码；如果定义了 NDEBUG，这些代码将被忽略掉：

242

```
void print(const int ia[], size_t size)
{
#ifndef NDEBUG
     // __func__ 是编译器定义的一个局部静态变量，用于存放函数的名字
     cerr << __func__ << ": array size is " << size << endl;
#endif
// ...
```

在这段代码中，我们使用变量__func__输出当前调试的函数的名字。编译器为每个函数都定义了__func__，它是 const char 的一个静态数组，用于存放函数的名字。

除了 C++编译器定义的__func__之外，预处理器还定义了另外 4 个对于程序调试很有用的名字：

__FILE__ 存放文件名的字符串字面值。

__LINE__ 存放当前行号的整型字面值。

__TIME__ 存放文件编译时间的字符串字面值。

__DATE__ 存放文件编译日期的字符串字面值。

可以使用这些常量在错误消息中提供更多信息：

```
if (word.size() < threshold)
    cerr << "Error: " << __FILE__
         << " : in function " << __func__
```