1. （p362）当用户将一个源文件（source file）提交给编译器后，首先对文件进行预处理，即，处理宏（见12.6节）以及将#include指令指定的头文件包含进来（见2.4.1节和15.2.2节）。预处理的结果称为编译单元（translation unit）。编译单元是编译器真正处理的内容，也是C++语言规则所描述的内容。在本书中，仅当需要区分程序员所看到的内容和编译器所处理的内容时才会区分源文件和编译单元。
2. 链接（p363）

除非已显示声明为局部名字，否则函数名、类名、模板名、名字空间名、枚举名以及枚举值名的使用必须跨所有编译单元保持一致。

程序员必须保证每个名字空间、类、函数等必须在其出现的每个编译单元中都正确声明，且对应相同实体的声明是一致的。例如，考虑下面两个文件：

//file1.cpp

int x = 0;

int f() { /\*…\*/ }

//file2.cpp

extern int x;

int f();

void g() { x = f(); }

file2.cpp中的g()使用的x和f()就是file1.cpp中所定义的实体。关键字extern指出file2.cpp中x的声明仅仅是一个声明而已，而非一个定义（见6.3节）。假如x已初始化，extern将会被忽略，因为带初始值的声明总是被看做一个定义。对象在程序中只能定义一次，它可以声明很多次，但类型必须完全一致。例如：

//file1.cpp

int x = 1;

int b = 1;

extern int c;

//file2.cpp

int x; //意味着“int x = 0;”

extern double bl

extern int c;

这个程序有3个错误：x被定义了两次，b的声明类型不一致，c被声明了两次但没有别定义。如果编译器只能同时处理一个文件，就无法检查这些错误（链接错误）。但大多数这种错误能被链接器检查出来。例如，我所知道的所有正确的C++实现都能检测出x的双重定义。但是，流行的C++实现都捕获不到b的声明不一致的问题，遗漏c的定义这一错误通常也只有在c被使用时才能捕获到。

注意，如果全局作用域中或名字空间中的变量定义不带初始值，则该变量会使用默认初始值（见6.3.5.1节）。非static局部变量或创建在自由存储上的对象（见11.2节）则不会使用默认初始值。

在类体外，实体必须先声明后使用（见6.3.4节）。例如：

//file1.cpp:

int g() { return f() + 7; } //错误：f()（尚）未声明

int f() { return x; } //错误：x（尚）未声明

int x;

如果一个名字在其定义处之外的编译单元中也可以使用，我们称其具有外部链接（external linkage）。前一个例子中的所有名字都具有外部链接。如果一个名字只能在其所定义的编译单元中被引用，我们称其具有内部链接（internal linkage）。例如：

static int x1 = 1; //内部链接：其编译单元中不可访问

const char x2 = ‘a’; //内部链接：其编译单元中不可访问

在名字空间作用域（包括全局作用域，见14.2.1节）中使用关键字static（有些不逻辑）辨识“不能再其他源文件中访问”（即内部链接）。如果你希望在其他文件中也能访问x1（“具有外部链接”），就应该去掉static。关键字const暗示默认内部链接，因此如果你希望x2具有外部链接，就需要在其定义前加上extern：

int x1 = 1; //外部链接：其编译单元中可访问

extern const char x2 = ‘a’; //外部链接：其编译单元中可访问

链接器看不到的名字，例如局部变量名，被称为无链接（no linkage）。

inline函数（见12.1.3节和16.2.8节）在其应用的所有编译单元中都必须有完全等价的定义（见15.2.3节）。因此下面这个例子不仅风格糟糕，而且是不合法的：

//file1.cpp

inline int f(int i) { return i; }

//file2.cpp

inline int f(int i) { return i+1; }

不幸的是，C++实现很难捕获这种错误。而下面的例子中外部链接和内部链接的组合虽然完全符合逻辑，但却被禁止的：

//file1.cpp

extern inline int g( int i );

int h(int i) { return g(i); } //错误：此编译单元中无g()定义

//file2.cpp

extern inline int g( int i ) { return i+1; }

//…

我们可以通过使用头文件来保持inline函数的异质性（见15.2.2节），例如：

//h.h

extern inline int next( int i ) { return i+1; }

//file1.cpp

#include “h.h”

int h(int i) { return next(i); } //正确

默认情况下，名字空间中的const对象（见7.5节）constexpr对象（10.4节）、类型别名（6.5节）以及任何声明为static的实体（见6.3.4节）都具有内部链接。因此，下面这个例子是合法的（虽然可能让人困惑）：

//file1.cpp

using T = int;

const int x = 7;

constexpr T c2 = x+1;

//file2.cpp

using T = double;

const int x = 8;

constexpr T c2 = x+9;

为了确保一致性，应该将别名、const对象、constexpr对象和inline函数放置在头文件中（见15.2.2节）