第2章

基本概念

毫无疑问，学习一门编程语言的基础知识不如编写程序有趣。但是，不知道语言的基础知识会使你在编写程序时缺少乐趣。

2.1 环境

在ANSI C的任何一种实现中，存在两种不同的**环境**。第1种是**翻译环境**(translation environment)，在这个环境里，源代码被转换为可执行的机器指令。第2种是**执行环境**(execution environment)，它用于实际执行代码。标准明确说明，这两种环境不必位于同一台机器上。例如，**交叉编译器**(cross compiler)就是在一台机器上运行，但它所产生的可执行代码运行于不同类型的机器上。操作系统也是如此。标准同时讨论了**独立环境**(freestanding environment)，就是不存在操作系统的环境。你可能在嵌入式系统中（如微波炉控制器）遇到这种类型的环境。

2.1.1 翻译

翻译阶段由几个步骤组成，组成一个程序的每个（有可能有多个）源文件通过编译过程分别转换为**目标代码**(object code)。然后，各个目标文件由**链接器**(linker)捆绑在一起，形成一个单一而完整的可执行程序。链接器同时也会引入标准C函数库中任何被该程序所用到的函数，而且它也可以搜索程序员个人的程序库，将其中需要使用的函数也链接到程序中。图2.1描述了这个过程。

编译过程本身也由几个阶段组成，首先是**预处理器**(preprocessor)处理。在这个阶段，预处理器在源代码上执行一些文本操作。例如，用实际值代替由#define 指令定义的符号以及读入由#include指令包含的文件的内容。

然后，源代码经过**解析**(parse)，判断它的语句的意思。第2个阶段是产生绝大多数错误和警告信息的地方。随后，便产生目标代码。目标代码是机器指令的初步形式，用于实现程序的语句。如果我们在编译程序的命令行中加入了要求进行优化的选项，**优化器**(optimizer)就会对目标代码进一步进行处理，使它效率更高。优化过程需要额外的时间，所以在程序调试完毕并准备生成正式产品之前一般不进行这个过程。至于目标代码是直接产生的，还是先以汇编语言语句的形式存在，然后再经过一个独立的阶段编译成目标文件，对我们来说并不重要。

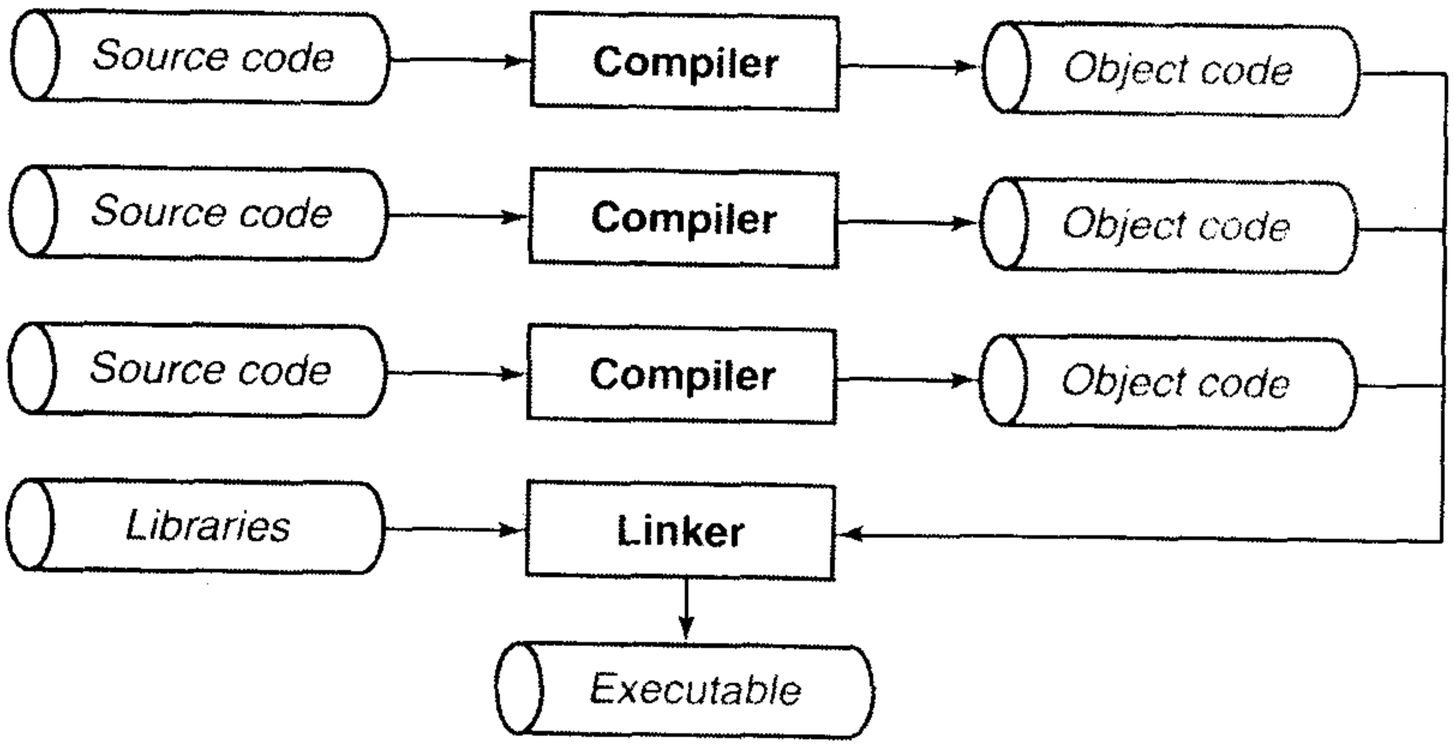

图 2.1　编译过程

一、文件名约定

尽管标准并没有制定文件的取名规则，但大多数环境都存在你必须遵守的文件名命名约定。C 源代码通常保存于以.c 扩展名命名的文件中。由#include 指令包含到 C 源代码的文件被称为头文件，通常具有扩展名.h。

至于目标文件名，不同的环境可能具有不同的约定。例如，在 UNIX 系统中，它们的扩展名是.o，但在 MS-DOS 系统中，它们的扩展名是.obj。

二、编译和链接

用于编译和链接 C 程序的特定命令在不同的系统中各不相同，但许多都和这里所描述的两种系统差不多。在绝大多数 UNIX 系统中，C 编译器被称为 cc，它可以用多种不同的方法来调用。

1．编译并链接一个完全包含于一个源文件的 C 程序：

```
cc program.c
```

这条命令产生一个称为 a.out 的可执行程序。中间会产生一个名为 program.o 的目标文件，但它在链接过程完成后会被删除。

2．编译并链接几个 C 源文件：

```
cc main.c sort.c lookup.c
```

当编译的源文件超过一个时，目标文件便不会被删除。这就允许你对程序进行修改后，只对那些进行过改动的源文件进行重新编译，如下一条命令所示。

3．编译一个 C 源文件，并把它和现存的目标文件链接在一起：

```
cc main.o lookup.o sort.c
```

4．编译单个 C 源文件，并产生一个目标文件（本例中为 program.o），以后再进行链接：

```
cc -c program.c
```

5．编译几个 C 源文件，并为每个文件产生一个目标文件：

```
cc -c main.c sort.c lookup.c
```

6．链接几个目标文件：

```
cc main.o sort.o lookup.o
```

上面那些可以产生可执行程序的命令均可以加上“-o name”这个选项，它可以使链接器把可执行程序保存在“name”文件中,而不是“a.out”。在缺省情况下，链接器在标准 C 函数库中查找。如果在编译时加上“-lname”标志，链接器就会同时在“name”的函数库中进行查找。这个选项应该出现在命令行的最后。除此之外，编译和链接命令还有很多选项，请查阅你所使用的系统的文档。

用于 MS-DOS 和 Windows 的 Borland C/C++ 5.0 有两种用户界面，你可以分别选用。Windows 集成开发环境是一个完整的独立编程工具，它包括源代码编辑器、调试器和编译器。它的具体使用不在本书的范围之内。MS-DOS 命令行界面则与 UNIX 编译器差不太多，只是有下面几点不同：

1．它的名字是 bcc。

2．目标文件的名字是 file.obj。

3．当单个源文件被编译并链接时，编译器并不删除目标文件。

4．在缺省情况下，可执行文件以命令行中第一个源或目标文件名命名，不过你可以使用“-ename”选项把可执行程序文件命名为“name.exe”。

2.1.2　执行

程序的执行过程也需要经历几个阶段。首先，程序必须载入到内存中。在宿主环境中（也就是具有操作系统的环境），这个任务由操作系统完成。那些不是存储在堆栈中的尚未初始化的变量将在这个时候得到初始值。在独立环境中，程序的载入必须由手工安排，也可能是通过把可执行代码置入只读内存(ROM)来完成。

然后，程序的执行便开始。在宿主环境中，通常一个小型的启动程序与程序链接在一起。它负责处理一系列日常事务，如收集命名行参数以便使程序能够访问它们。接着，便调用 main 函数。

现在，便开始执行程序代码。在绝大多数机器里，程序将使用一个运行时**堆栈**(stack)，它用于存储函数的局部变量和返回地址。程序同时也可以使用**静态**(static)内存，存储于静态内存中的变量在程序的整个执行过程中将一直保留它们的值。

程序执行的最后一个阶段就是程序的终止，它可以由多种不同的原因引起。“正常”终止就是 main 函数返回[1]。有些执行环境允许程序返回一个代码，提示程序为什么停止执行。在宿主环境中，启动程序将再次取得控制权，并可能执行各种不同的日常任务，如关闭那些程序可能使用过但并未显式关闭的任何文件。除此之外，程序也可能是由于用户按下 break 键或者电话连接的挂起而终止，另外也可能是由于在执行过程中出现错误而自行中断。

2.2　词法规则

词法规则就像英语中的拼写规则，决定你在源程序中如何形成单独的字符片段，也就是**标记**(token)。

一个 ANSI C 程序由声明和函数组成。函数定义了需要执行的工作，而声明则描述了函数和（或）函数将要操作的数据类型（有时候是数据本身）。注释可以散布于源文件的各个地方。

---

[1] 或当有些程序执行了 exit，将在第 16 章描述。

2.2.1 字符

标准并没有规定 C 环境必须使用哪种特定的字符集，但它规定字符集必须包括英语所有的大写和小写字母，数字 0 到 9，以及下面这些符号：

```
!  "  #  %  '  (  )  *  +  ,  -  .  /  :
;  <  >  =  ?  [  ]  \  ^  _  {  }  |  ~
```

换行符用于标志源代码每一行的结束，当正在执行的程序的字符输入就绪时，它也用于标志每个输入行的末尾。如果运行时环境需要，换行符也可以是一串字符，但它们被当作单个字符处理。字符集还必须包括空格、水平制表符、垂直制表符和格式反馈字符。这些字符加上换行符，通常被称作空白字符，因为当它们被打印出来时，在页面上出现的是空白而不是各种记号。

标准还定义了几个**三字母词(trigrph)**，三字母词就是几个字符的序列，合起来表示另一个字符。三字母词使 C 环境可以在某些缺少一些必需字符的字符集上实现。这里列出了一些三字母词以及它们所代表的字符。

```
??(  [        ??<  {        ??=  #
??)  ]        ??>  }        ??/  \
??!  |        ??'  ^        ??-  ~
```

两个问号开头再尾随一个字符一般不会出现在其他表达形式中，所以把三字母词用这种形式来表示，这样就不致引起误解。

警告：

尽管三字母词在某些环境中很有用，但对于那些用不着它的人而言，它实在是个令人讨厌的小东西。之所以选择??这个序列作为每个三字母词的开始是因为它们出现的形式很不自然，但它们仍然隐藏着危险。你的脑子里一般不会有三字母词这个概念，因为它们极少出现。所以，当你偶尔书写了一个三字母词时，如下所示：

```
printf("Delete file (are you really sure??): " );
```

结果输出中将产生]字符，这无疑会令你大吃一惊。

当你编写某些 C 源代码时，你在一些上下文环境里想使用某个特定的字符，却可能无法如愿，因为该字符在这个环境里有特别的意义。例如，双引号 " 用于定界字符串常量，你如何在一个字符串常量内部包含一个双引号呢？K&R C 定义了几个**转义序列**(escape sequence)或**字符转义**(character escape)，用于克服这个难题。ANSI C 在它的基础上又增加了几个转义序列。转义序列由一个反斜杠 \加上一或多个其他字符组成。下面列出的每个转义序列代表反斜杠后面的那个字符，但并未给这个字符增加特别的意义。

\? 在书写连续多个问号时使用，防止它们被解释为三字母词。

\" 用于表示一个字符串常量内部的双引号。

\' 用于表示字符常量 '。

\\ 用于表示一个反斜杠，防止它被解释为一个转义序列符。

有许多字符并不在源代码中出现，但它们在格式化程序输出或操纵终端显示屏时非常有用。C 语言也提供了一些这方面的转义符，方便你在程序中包含它们。在选择这些转义符的字符时，特地

考虑了它们是否有助于记忆它们代表的字符的功能。

K&R C：

下面的转义符中，有些标以“ †”符号，表示它们是 ANSI C 新增的，在 K&R C 中并未实现。

\a　† 警告字符。它将奏响终端铃声或产生其他一些可听见或可看见的信号。

\b　退格键。

\f　进纸字符。

\n　换行符。

\r　回车符。

\t　水平制表符。

\v　† 垂直制表符。

\ddd　ddd 表示 1～3 个八进制数字。这个转义符表示的字符就是给定的八进制数值所代表的字符。

\xddd †　与上例类似，只是八进制数换成了十六进制数。

注意，任何十六进制数都有可能包含在\xddd 序列中，但如果结果值的大小超出了表示字符的范围，其结果就是未定义的。

2.2.2　注释

C 语言的注释以字符/*开始，以字符*/结束，中间可以包含除*/之外的任何字符。在源代码中，一个注释可能跨越多行，但它不能嵌套于另一个注释中。注意，/*或*/如果出现在字符串字面值内部，就不再起注释定界符的作用。

所有的注释都会被预处理器拿掉，取而代之的是一个空格。因此，注释可以出现于任何空格可以出现的地方。

警告：

注释从注释起始符/*开始，到注释终止符*/结束，其间的所有东西均作为注释的内容。这个规则看上去一目了然，但对于编写了下面这段看上去很无辜的代码的学生而言，情况就不一定如此了。你能看出来为什么只有第 1 个变量才被初始化吗？

```
x1=0;      /************************
x2=0;      **Initialize the       **
x3=0;      **counter variables.  **
x4=0;      ************************/
```

警告：

注意中止注释用的是*/而不是*？。如果你击键速度太快或者按住 shift 键的时间太长，就可能误输入为后者。这个错误在指出来以后是一目了然，但在现实的程序中这种错误却很难被发现。

2.2.3　自由形式的源代码

C 是一种自由形式的语言，也就是说并没有规则规定什么地方可以书写语句，一行中可以出现

多少条语句，什么地方应该留下空白以及应该出现多少空白等[1]。唯一的规则就是相邻的标记之间必须出现一至多个空白字符(或注释)，不然它们可能被解释为单个标记。因此，下列语句是等价的：

```
y=x+1;

y = x + 1;

y = x
+
1
```

至于下面这组语句，前 3 条语句是等价的，但第 4 条语句却是非法的：

```
int
x;

int    x;

int/*comment*/x;

intx;
```

这种代码书写的极度自由有利有弊。很快你就将听到一些关于这个话题的肥皂盒哲学。

2.2.4 标识符

标识符(identifier)就是变量、函数、类型等的名字。它们由大小写字母、数字和下划线组成，但不能以数字开头。C 是一种大小写敏感的语言，所以 abc、Abc、abC 和 ABC 是 4 个不同的标识符。标识符的长度没有限制，但标准允许编译器忽略第 31 个字符以后的字符。标准同时允许编译器对用于表示外部名字（也就是由链接器操纵的名字）的标识符进行限制，只识别前六位不区分大小写的字符。

下列 C 语言关键字是被保留的，它们不能作为标识符使用：

```
auto      do        goto          signed    unsigned
break     double    if            sizeof    void
case      else      int           static    volatile
char      enum      long          struct    while
const     extern    register      switch
continue  float     return        typedef
default   for       short         union
```

2.2.5 程序的形式

一个 C 程序可能保存于一个或多个源文件中。虽然一个源文件可以包含超过一个的函数，但每个函数都必须完整地出现于同一个源文件中[2]。标准并没有明确规定，但一个 C 程序的源文件应该包含一组相关的函数，这才是较为合理的组织形式。这种做法还有一个额外的优点，就是它使实现抽象数据类型成为可能。

---

[1] 预处理指令是个例外，第 14 章将对此进行描述，它是以行定位的。

[2] 从技术上说，使用#include 指令，一个函数可以分在两个源文件中定义，只要把其中一个包含到另一个就行，但这个方法可不是#include 指令的合理用法。

2.3　程序风格

这里按顺序列出了一些有关编程风格的评论。像 C 这种自由形式的语言很容易产生邋遢的程序，就是那种写起来很快很容易但以后很难阅读和理解的程序。人们一般凭借视觉线索进行阅读，所以你的源代码如果井然有序，将有助于别人以后阅读（阅读的人很可能就是你自己）。程序 2.1 就是一个例子，虽然有些极端，但它说明了这个问题。这是一个可以运行的程序，执行一些多少有点用处的功能。问题是，你能明白它是干什么的吗[1]？更糟的是，如果你要修改这个程序，该从何处着手呢？尽管，如果时间充裕，经验丰富的程序员能够推断出它的意思，但恐怕很少会有人乐意这么干。把它扔在一边，自己从头写一个要方便快速得多。

```
#include <stdio.h>
main(t,_,a)
char *a;
{return!0<t?t<3?main(-79,-13,a+main(-87,1-_,
main(-86, 0, a+1 )+a)):1,t<_?main(t+1, _, a ):3,main ( -94, -27+t, a
)&&t == 2 ?_<13 ?main ( 2, _+1, "%s %d %d\n" ):9:16:t<0?t<-72? main(_,
t,"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l,+,/n{n+\
,/+#n+,/#;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l q#'+d'K#!/\
+k#;q#'r}eKK#}w'r} eKK{nl}'/#;#q#n'}{}#}w'}{}{nl}'/+#n';d}rw' i;# ){n\
l}!/n{n#'; r{#w'r nc{nl}'/#{l,+'K {rw' iK{;[{nl}'/w#q#\
n'wk nw' iwk{KK{nl}!/w{%'l##w#' i; :{nl}'/*{q#'ld;r'} {nlwb!/*de}'c \
;;{nl'-{}rw}'/+,} ##'*}#nc,',#nw]'/+kd'+e}+;\
#'rdq#w! nr'/ ') }+}{rl#'{n' '}# }'+}##(!!/")
:t<-50?_==*a ?putchar(a[31]):main(-65,_,a+1):main((*a == '/')+t,_,a\
+1 ):0<t?main ( 2, 2 , "%s"): *a=='/'|| main(0, main(-61,*a, "!ek;dc \
i@bK'(q)-[w]*%n+r3#l,{} :\nuwloca-O; m .vpbks,fxntdCeghiry"),a+1);}
```

程序 2.1　神秘程序　　mystery.c

提示：

不良的风格和不良的文档是软件生产和维护代价高昂的两个重要原因。良好的编程风格能够大大提高程序的可读性。良好的编程风格的直接结果就是程序更容易正确运行，间接结果是它们更容易维护，这将节省大笔资金成本。

本书的例子程序使用的风格是通过合理使用空格以强调程序的结构。我在下面列出了这个风格的几个特征，并说明为什么使用它们。

1．空行用于分隔不同的逻辑代码段，它们是按照功能分段的。这样，读者一眼就能看到某个逻辑代码段的结束，而不必仔细阅读每行代码来找出它。

2．if 和相关语句的括号是这些语句的一部分，而不是它们所测试的表达式的一部分。所以，我在括号和表达式之间留下一个空格，使表达式看上去更突出一些。函数的原型也是如此。

---

[1] 不管你相信与否，它打印出歌曲 *The Twelve Days of Christmas* 的歌词。这个程序由 Cambridge Consultants Ltd.的 Ian Phillipps 编写，用于参加国际 C 混乱代码大赛(International Obfuscated C Code Contest, 参见 http:// reality.sgi.com/csp/ioccc)。我在征得同意后把它列于本书中，作了少许修改。

3．在绝大多数操作符的使用中，中间都隔以空格，这可以使表达式的可读性更佳。有时，在复杂的表达式中，我会省略空格，这有助于显示子表达式的分组。

4．嵌套于其他语句的语句将缩进，以显示它们之间的层次。使用 Tab 键而不是空格，你可以很容易地将相关联的语句整齐排列。当整页都是程序代码时，使用足够大的缩进有助于程序匹配部分的定位，只使用两到三个空格是不够的。

有些人避免使用 Tab 键，因为他们认为 Tab 键使语句缩进得太多。在复杂的函数里，嵌套的层次往往很深，使用较大的 Tab 缩进意味着在一行内书写语句的空间就很小了。但是，如果函数确实如此复杂，你最好还是把它分成几个函数，可以使用其他函数来实现原先嵌套太深的部分语句。

5．绝大部分注释都是成块出现的，这样它们从视觉上在代码中很突出。读者可以更容易找到和跳过它们。

6．在函数的定义中，返回类型出现于独立的一行中，而函数的名字则在下一行的起始处。这样，在寻找函数的定义时，你可以在一行的开始处找到函数的名字。

在你研究这些代码例时，你还将看到很多其他特征。其他程序员可以选择他们喜欢的个人风格。你到底采用这种风格还是选择其他风格其实并不重要，关键是要始终如一地坚持使用同一种合理的风格。如果你始终保持如一的风格，任何有一定水平的读者都能较为容易地读懂得你的代码。

2.4 总结

一个 C 程序的源代码保存在一个或多个源文件中，但一个函数只能完整地出现在同一个源文件中。把相关的函数放在同一个文件内是一种好策略。每个源文件都分别编译，产生对应的目标文件。然后，目标文件被链接在一起，形成可执行程序。编译和最终运行程序的机器有可能相同，也可能不同。

程序必须载入到内存中才能执行。在宿主式环境中，这个任务由操作系统完成。在自由式环境中，程序常常永久存储于 ROM 中。经过初始化的静态变量在程序执行前能获得它们的值。你的程序执行的起点是 main 函数。绝大多数环境使用堆栈来存储局部变量和其他数据。

C 编译器所使用的字符集必须包括某些特定的字符。如果你使用的字符集缺少某些字符，可以使用三字母词来代替。转义序列使某些无法打印的字符得以表达，例如在程序中包含某些空白字符。

注释以/*开始，以*/结束，它不允许嵌套。注释将被预处理器去除。标识符由字母、数字和下划线组成，但不能以数字开头。在标识符中，大写字母和小写字母是不一样的。关键字由系统保留，不能作为标识符使用。C 是一种自由形式的语言。但是，用清楚的风格来编写程序有助于程序的阅读和维护。

2.5 警告的总结

1．字符串常量中的字符被错误地解释为三字母词。
2．编写得糟糕的注释可能会意外地中止语句。
3．注释的不适当结束。

2.6 编程提示的总结

良好的程序风格和文档将使程序更容易阅读和维护。

2.7 问题

1. 在 C 语言中，注释不允许嵌套。在下面的代码段中，用注释来“注释掉”一段语句会导致什么结果？

```
void
squares( int limit )
{
/* Comment out this entire function
        int     i;      /* loop counter */

        /*
        ** Print table of squares
        */
        for( i = 0; i < limit; i += 1 )
                printf( "%d %d0, i, i * i );
End of commented-out code */
}
```

2. 把一个大型程序放入一个单一的源文件中有什么优点？有什么缺点？
3. 你需要用 printf 函数打印出下面这段文本（包括两边的双引号）。你应该使用什么样的字符串常量参数？

```
"Blunder??!??"
```

4. \40 的值是多少？\100、\x40、\x100、\0123、\x0123 的值又分别是多少？
5. 下面这条语句的结果是什么？

```
int   x/*blah blah*/y;
```

6. 下面的声明存在什么错误（如果有的话）？

```
int  Case, If, While, Stop, stop;
```

7. 是非题：因为 C（除了预处理指令之外）是一种自由形式的语言，唯一规定程序应如何编写的规则就是语法规则，所以程序实际看上去的样子无关紧要。
8. 下面程序中的循环是否正确？

```
#include <stdio.h>

int
main( void )
{
int     x, y;

x = 0;
while( x < 10 ){
        y = x * x;
        printf( "%d\t%d\n", x, y );
        x += 1;
}
```

这个程序中的循环是否正确？

```
#include <stdio.h>

int
main( void )
{
        int     x, y;
        x = 0;
        while( x < 10 ){
                y = x * x;
                printf( "%d\t%d\n", x, y );
                x += 1;
}
```

哪个程序更易于检查其正确性？

9. 假定你有一个 C 程序，它的 main 函数位于文件 main.c，它还有一些函数位于文件 list.c 和 report.c。在编译和链接这个程序时，你应该使用什么命令？

10. 接上题，如果你想使程序链接到 parse 函数库，你应该对命令作何修改？

11. 假定你有一个 C 程序，它由几个单独的文件组成，而这几个文件又分别包含了其他文件，如下所示：

文　件	包 含 文 件
main.c	stdio.h, table.h
list.c	list.h
symbol.c	symbol.h
table.c	table.h
table.h	symbol.h, list.h

如果你对 list.c 作了修改，你应该用什么命令进行重新编译？如果是 list.h 或者 table.h 作了修改，又分别应该使用什么命令？

2.8 编程练习

★ 1. 编写一个程序，它由 3 个函数组成，每个函数分别保存在一个单独的源文件中。函数 increment 接受一个整型参数，它的返回值是该参数的值加 1。increment 函数应该位于文件 increment.c 中。第 2 个函数称为 negate，它也接受一个整型参数，它的返回值是该参数的负值（例如，如果参数是 25，函数返回–25；如果参数是–612，函数返回 612）。最后一个函数是 main，保存于文件 main.c 中，它分别用参数 10, 0 和–10 调用另外两个函数，并打印出结果。

★★ 2. 编写一个程序，它从标准输入读取 C 源代码，并验证所有的花括号都正确地成对出现。注意：你不必担心注释内部、字符串常量内部和字符常量形式的花括号。