互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

# 第2章 词法元素

本章描述C语言的词法结构，即C源文件中可以出现的字符以及它们是如何形成词法单元（或称为标记）的。

## 2.1 字 符 集

C源文件就是从一个字符集中所选择的字符序列。C程序是使用下面这些字符编写的，这些字符是在ISO/IEC 10646的基本拉丁文（Basic Latin）部分定义的：

1. 52个大写和小写拉丁字母：

```
A B C D E F G H I J K L M N O P Q R S T
U V W X Y Z a b c d e f g h i j k l m
n o p q r s t u v w x y z
```

2. 10个数字：

```
0 1 2 3 4 5 6 7 8 9
```

3. 空格。

4. 水平制表符（HT）、垂直制表符（VT）和换页符（FF）等格式控制符。

5. 29个图形字符以及它们的官方名称（如表2-1所示）。

另外，还必须有办法把源程序划分为多个代码行。这个任务可以由一个字符或一个字符序列来完成，或者使用源字符集之外的其他机制（例如记录尾指示符）来完成。

表2-1 图形字符

| 字符 | 名称 | 字符 | 名称 | 字符 | 名称 |
|---|---|---|---|---|---|
| ! | 感叹号 | + | 加号 | " | 双引号 |
| # | 序号字符 | = | 等号 | { | 左花括号 |
| % | 百分号 | ~ | 波浪符 | } | 右花括号 |
| ^ | 折音符 | [ | 左方括号 | , | 逗号 |
| & | 和号 | ] | 右方括号 | . | 句号、点号 |
| * | 星号 | ’ | 撇号、单引号 | < | 小于号 |
| ( | 左括号 | \| | 竖杠 | > | 大于号 |
| _ | 下划线 | \ | 反斜杠 | / | 正斜杠 |
| ) | 右括号 | ; | 分号 | ? | 问号 |
| - | 连字符、减号 | : | 冒号 | | |

有些使用本国字符集的国家并没有包含表2-1中的所有图形字符。C89（修正案1）定义了三字符组（trigraph）和标记重拼（token respelling），允许用ISO 646—1083不变性代码集（Invariant Code Set）编写C程序。

C源程序有时候还使用其他字符，包括：

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



1. 像退格符（BS）和回车符（CR）这样的格式字符。

2. 其他的基本拉丁字符，包括$符号（美元符号）、@（商用的AT）和`（重音）。

格式字符被看成是空格，除此之外不会影响源程序。其他图形字符只可能出现在注释、字符常量、字符串常量以及文件名中。

参考：基本拉丁文 第2.9节；字符常量 第2.7.3节；注释 第2.2节；字符编码 第2.1.3节；字符转义码 第2.7.6节；执行字符集 第2.2.1节；字符串常量 第2.7.4节；标记重拼 第2.4节；三字符组 第2.1.4节。

### 2.1.1 执行字符集

在执行C程序时，字符集的解释方式并不一定和编写C程序时所使用的字符集相同。执行字符集中的字符是用源字符集中的对应字符表示的，或者用以反斜杠（\）开头的特殊的字符转义序列表示。

除了前面所提到的标准字符之外，执行字符集还必须包括：

1. null字符，它必须被编码为0这个值。

2. 换行符，表示行末标志。

3. 警告、退格和回车字符。

null字符用于标志字符串的结束；换行符用于在输入/输出时把字符流分割为不同的行（如果换行符事实上就存在于执行环境的文本流中，对程序员无疑很有吸引力。但是，运行时函数库的实现可以采用自己的方式对它进行模拟。例如，换行符可以被转换为输出中的记录尾指示符，而记录尾指示符也可以被转换为输入中的换行符）。

和源字符集一样，在执行字符集中包含格式字符如退格符、水平制表符、垂直制表符、换页符、回车符也很常见。另外，它还可以使用特殊的转义序列表示源程序中的这些字符。

当C程序在同一台计算机上编译和执行时，源字符集和执行字符集是相同的。但有时候C程序是跨机器编译的，也就是在一台计算机（宿主计算机）上编译，但在另一台计算机（目标计算机）上执行。当编译器计算一个涉及字符的常量表达式的编译时值时，它必须使用目标计算机的编码方式，而不是采用最自然（对于编译器编写者而言）的源字符集编码方式。

参考：字符常量 第2.7.3节；注释 第2.2节；字符编码 第2.1.3节；字符集 第2.1节；执行字符集 第2.2.1节；常量表达式 第7.11节；转义字符 第2.7.5节；文本流 第15章。

### 2.1.2 空白字符和行终止符

在C源程序中，空格、行末符、垂直制表符、换页符和水平制表符（如果存在）合称为空白字符（稍后讨论的注释也属于空白）。这些字符将被忽略，除非它们用于分隔相邻的标记，或者出现在字符常量、字符串常量和#include文件名中。空白字符可以用于设置C程序的布局，使它更容易阅读。

行末字符（或字符序列）标志着一行源程序的结束。在有些编译器中，回车符、换页符和垂直制表符等格式字符都可以终止源代码行，因此它们合称为行分隔符。为了确定预处理器所处理的行，行终止符是非常重要的。紧随行分隔符之后的那个字符被认为是下一行的第1个字符。如果第1个字符就是行分隔符，这一行就结束（表示空行），接下来以此类推。

为了把一个源代码行延续到下一行，可以在第1行的末尾使用反斜杠字符（\），或者使用标

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



准C的三字符组??/。这时，反斜杠和行末标志符将会被删除，并创建一个更长的逻辑源代码行。这个约定在预处理器命令以及字符串常量中总是有效的，这也是它最常用的场合，并且具有很好的移植性。标准C和许多非标准的编译器对它进行了扩展，使它可以作用于任何源程序行。从概念上说，源代码行的延续出现在预处理和C程序的词法分析之前，但在三字符组的处理以及任何多字节字符序列到源字符集的转换之后。

例子

在标准C中，甚至是标记也可以跨越多行。下面这两行

```
if (a==b) x=1; el\
se x=2;
```

相当于下面这一行代码

```
if (a==b) x=1; else x=2;
```

如果一个编译器把所有的非标准源字符都看成是空格或行分隔符，它分别应该像处理空白字符和行末标志字符一样处理它们。标准C建议编译器在第一次读取源程序时把所有这类字符转换为某种规范的表示形式，以便进行上述的处理。但是，程序员应该意识到依赖这种做法可能造成的结果。例如，可以预料到出现在反斜杠后面的换页符将会被去除。

大多数C编译器对行延续符之前和之后的源代码行的最大长度施加了限制。C89要求编译器允许逻辑源代码行的长度至少为509个字符；C99则要求至少允许4095个字符。

参考：字符常量 第2.7.3节；预处理器词法转换 第3.2节；源字符集 第2.1.1节；字符串常量 第2.7.4节；标记 第2.3节；三字符组 第2.1.4节。

### 2.1.3 字符编码

计算机的（执行）字符集中的每个字符都具有某种编码约定，也就是在计算机中具有一种数值表示形式。这种编码是非常重要的，因为C把字符转换为整数，这个整数的值就是该字符的编码约定。前面所列出的所有标准字符必须都具有唯一的、非负的整数编码形式。

一种常见的C编程错误就是以为计算机所使用的是某种编码方案，但实际上它所使用的是另一种编码方案。

例子

C表达式'z' - 'A'+ 1计算Z和A的编码值之差加上1的结果，也就是字母表中的字符数量。事实上，在ASCII字符集编码方案中，这个结果是26。但是，在EBCDIC编码方案中，字母并不是连续编码的，这个计算的结果是41。

参考：源字符集和执行字符集 第2.1.1节。

### 2.1.4 三字符组

标准C增加了一组三字符组，使C程序可以只使用ISO 646-1083不变性字符集来编写。这个字符集是7位的ASCII码的一个子集，它在许多非英语国家的字符集中也极为常用。表2-2列出了由两个连续的问号字符所引入的三字符组。标准C还提供了一些标记的重拼（第2.4节）和<iso64.h>头文件，后者定义了一些可用于替换一些操作符的宏。但是，和三字符组不同，这些宏并不会被看成是字符串和字符常量。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



表2-2 ISO三字符组

| 三字符组 | 替换字符 | 三字符组 | 替换字符 |
|---|---|---|---|
| ??( | [ | ??) | ] |
| ??< | { | ??> | } |
| ??/ | \ | ??! | \| |
| ??' | ^ | ??- | ~ |
| ??= | # | | |

在源程序中，三字符组的替换出现在词法分析（标记化）以及字符串和字符串常量中的转义字符被确认之前。C编译器一共认识9个三字符组，所有其他字符序列（例如??&）都不会被当成是三字符组而进行转换。另外，一个新的转义字符\?可以防止对类似三字符组的字符序列进行转换。

例子

如果需要在一个字符串中包含一个三字符序列，但这三个字符在正常情况下将被解释为一个三字符组，必须使用反斜杠转义字符引用这三个字符中的至少一个字符。因此，字符串常量“What?\?!”实际上所表示的是字符串常量“What??!”。

如果需要在一个字符串中包含一个反斜杠字符，必须使用连续2个反斜杠（第1个引用第2个）。另外，每个反斜杠可以转换为对应的三字符组。因此，字符串“??/??/”表示一个包含了单个\字符的字符串。

参考：字符集 第2.1节；转义字符 第2.7.5节；iso646.h 第11.9节；字符串连接 第2.7.4节；标记重拼 第2.4节。

### 2.1.5 多字节字符和宽字符

为了容纳可能包含大量字符的非英语字母，标准C引入了宽字符和宽字符串的概念。为了表示宽字符以及外部世界中面向字节的字符串，标准C还引入了多字节字符的概念。C89修正案1提供了扩展，以处理宽字符和多字节字符。

宽字符和宽字符串 宽字符是扩展字符集（extended character set）中的元素的二进制表示形式。它的类型是整数类型wchar_t，这个类型是在头文件stddef.h中声明的。C89修正案1添加了整数类型wint_t，它必须可以表示wchar_t类型的所有值，并且可以表示一个额外的、可以区分的非宽字符值，用WEOF表示。标准C并没有指定任何宽字符编码方案，但0这个值被保留为“空的宽字符值”。宽字符常量可以用一种特殊的常量语法来表示（第2.7.3节）。

例子

一般情况下，宽字符占据16位。因此，在32位的计算机上，wchar_t可以用short或unsigned short来表示。如果wchar_t用short表示，并且-1并不是合法的宽字符，则wint_t可以用short表示，WEOF可以用-1表示。但是，更典型的方法是用int或unsigned int表示wint_t。

如果编译器厂商选择不支持扩展字符集（美国的C编译器厂商经常采用这种做法），wchar_t可以被定义为char，“扩展字符集”就与常规的字符集相同。

宽字符串（wide string）是以一个null宽字符结尾的连续的宽字符序列。null宽字符就是表示形式为0的宽字符。除了null宽字符以及另外的WEOF之外，标准C并没有指定扩展字符集的编码

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



方案。宽字符常量可以用特殊的字符串常量来指定（第2.7.4节）。

多字节字符 在C程序中，宽字符可以作为一个单独的单元进行操作，但大多数外部媒介（例如文件）和C源程序是基于单个字节长度的字符的。熟悉扩展字符集的程序员设计了多字节编码（multibyte encoding），这是一种区域特定的方法，在字节长度的字符序列和宽字符序列之间进行映射。

多字节字符是源字符集或执行字符集中宽字符的表示形式（它们可能具有不同的编码形式）。因此，多字节字符串是常规的C字符串，但是这些字符可以被解释为一系列的多字节字符。多字节字符的形式以及多字节字符和宽字符之间的映射取决于定义的实现方式。在编译时，这种映射是针对宽字符常量和宽字符串常量执行的，标准函数库提供了一些函数，可以在运行时执行这种映射。

多节字字符可以使用依赖状态的编码（state-depedent encoding），即多字节字符的解释可能取决于前一个出现的多字节字符。一般情况下，这种编码利用了转移字符。这是一种控制字符，属于多字节字符的一部分，用于更改当前以及后续字符的解释。在一个多字节字符序列中，当前解释被称为编码的转换状态（conversion state），或称转移状态（shift state）。开始对一个多字节字符序列进行转换时，总是使用一个可区分的初始转换（转移）状态，并且这个状态经常在转换结束时被返回。

例子

A编码（我们在这个例子中所使用的一种假设编码）是一种取决于状态的编码方案，它具有两个转移状态：“ 上 ”和“ 下 ”。↑字符把转移状态修改为“ 上 ”，↓字符把转移状态修改为“ 下 ”。“ 下 ”状态是初始状态，所有的非转移字符具有常规的解释。在“ 上 ”状态中，每个多字节字符由一对字母数字字符组成，它们以一种我们并未指定的方式定义了一个宽字符。

在下面这些字符序列中，每个序列包含了3个A编码形式的多字节字符，都从初始的转换状态开始。

```
abc    ab↑e3    ↑ab↓b↑23    ↓a↓b↓c
```

最后一个字符串包含了并非严格必需的转移字符。如果允许冗余的转移序列，多字节字符可以变成任意长度（例如↓↓…↓x）。除非知道转换状态位于多字节字符序列的开始，否则就无法对诸如abcdef这样的序列进行解析，它既可以表示3个也可以表示6位宽字符。

ab|?x这个序列在A编码形式下是非法的，因为在“ 上 ”转移状态中出现了非字母数字字符。a↑b这个序列是非法的，因为最后一个多字节字符过早结束。

多字节字符也可以使用状态无关编码（state-independent encoding），即一个多字节字符的解释并不依赖于前一个多字节字符（尽管可能需要在一个多字节序列中从头开始寻找一个在字符串中间开始的多字节字符）。例如，C的转义字符的语法（第2.7.5节）表示一种char类型的状态独立编码，因为反斜杠字符（\）改变了它后面的一个或多个字符的解释，形成一个char类型的值。

例子

B编码是另一种假设的编码方案，它是一种状态无关编码，并使用了一个特殊的字符，用▽表示，用于更改它后面的非null字符的含义。在B编码方案中，下面这些字符序列各自包含了3个多字节字符：

```
abc    ▽a▽b▽c    ▽▽▽▽▽▽    a ▽bc
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



在B编码方案中，▽▽▽这个序列是非法的，因为它的最后并不是一个非null字符。

标准C对多字节字符设置了一些限制：

1. 标准字符集中的所有字符都必须出现在编码中。

2. 在初始的转移状态中，标准字符集中的所有单字节字符都保留正常的解释，并不影响转换状态。

3. 包含全0的字符不管是什么转移状态都被认为是null字符。多字节字符不能使用全0的字节作为它的第二个或后续的字符。

总而言之，这些规则保证多字节序列能够作为正常的C字符串（即中间不会嵌入null字符）处理。并且，没有特殊多字节编码的C字符和多字节序列一样可以得到预期的解释。

多字节字符的源代码和执行用法 多字节字符可以出现在注释、标识符、预处理器头文件名、字符串常量和字符常量中。每个注释、标识符、预处理器头文件名、字符串常量和字符常量必须以初始转移状态开始并结束，并且必须由合法的多字节字符序列组成。源代码的物理表示形式中的多字节字符在任何词法分析、预处理或甚至延续行的连接之前就被确认。

例子

一个日文文本编辑程序允许在字符串常量和注释中使用日文字符。如果文本被写入到一个字节流文件中，则日文字符将被转换为多字节序列，这对于字符串常量而言是可以接受的，可以被标准C编辑器所理解。

在预处理期间，字符串和字符常量中的字符被转换为执行字符集，然后才被解释为多字节序列。因此，在形成多字节字符时，转义序列能使用到。在这个阶段之前，程序中的注释已经被删除，因此多字节注释中的转义序列可能没有什么意义。

例子

如果源字符集和执行字符集相同，并且如果'a'的值在执行字符集为$141_8$，那么字符串常量"▽aa"所包含的两个多字节字符与"▽\141\141"相同（B编码方案）。

参考：字符常量 第2.7.3节；注释 第2.2节；多字节转换工具 第11.7、11.8节；字符串常量 第2.7.4节；wchar_t、WEOF第11.1节；宽字符 2.7.3节；宽字符串 第2.7.4节；wint_t 第11.1节。

## 2.2 注　释

在标准C中，可以使用两种方法编写注释。传统的注释从两个字符/*开始，以接下来出现的第1对*/结束。注释可以包含任意数量的字符，并且总是被当成空白看待。

从C99开始，注释也可以从//字符开始，并延续到下一个行分隔符（但并不包括这个行分隔符）。这有可能会破坏旧式C程序的代码（不过可能性极小）。作为练习，读者可以判断在什么条件下会出现这种情况。

如果注释位于字符串或字符常量内部，或者位于其他注释的内部，它并不会被编译器当做注释。除了能够认识注释中的多字节字符和行分隔符之外，C编译器并不会检查注释的内容。

例子

下面这个程序包含了4个合法的C注释：

```
// Program to compute the squares of
// the first 10 integers
#include <stdio.h>
```

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



```
void Squares( /* no arguments */ )
{
    int i;
    /*
       Loop from 1 to 10,
       printing out the squares
    */
    for (i=1; i<=10; i++)
       printf("%d //squared// is %d\n",i,i*i);
}
```

在预处理之前，编译器会删除注释，因此注释内部的预处理器命令将无法被认识，注释内部的行分隔符也不会终止预处理器命令。

例子

下面这两个#define命令具有相同的效果：

```
#define ten (2*5)

#define ten /* ten:
               one greater than nine
            */ (2*5)
```

标准C规定，所有的注释被一个空格字符所代替，以便于对C程序作进一步的转换。但是，有些旧式的实现并不会插入任何空白字符。这将影响预处理器的行为，详情将在第3.3.9节讨论。

有一些非标准的C编译器实现了“可嵌套注释”，即注释中的每一对/*必须存在对应的*/。这并不是标准的实现，程序员不应该依赖这个特性。为了使程序保持兼容，注释的内部不应该包含/*字符。

例子

为了使编译器忽略一个C程序的很大一部分代码，最好使用预处理器命令封装这块代码：

```
#if 0
…
#endif
```

这个方法优于在文本前后插入/*和*/。如果采用后面这种方法，如果这段代码内部也存在注释，就不会担心出现/*-风格问题。

参考：#if预处理器命令 第3.5.1节；预处理器词法约定 第3.2节；空白 第2.1节。

## 2.3 标 记

根据本章剩余部分所讨论的规则，组成C程序的字符被收集为词法标记。标记一共分为5种类型：操作符、分隔符、标识符、关键字和常量。

当编译器按照从左向右的顺序收集字符时，它总是尽可能形成最长可能的标记，即使这样做所形成的并不是合法的C程序。相邻的标记可以用空白字符或注释进行分隔。为了防止混淆，标识符、关键字、整型常量、浮点型常量必须与后面的标识符、关键字、整型常量或浮点型常量分隔。

预处理器采用了略有不同的标记约定。具体地说，标准C预处理器把#和##看成是标记，这在传统C中是非法的。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



例子

| 字符 | C标记 |
|---|---|
| forwhile | forwhile |
| b>x | b、>、x |
| b->x | b、->、x |
| b--x | b、--、x |
| b---x | b、--、-、x |

在第4个例子中，b--x这个字符序列是非法的C语法。如果分解为b、-、-、x这些标记，虽然在语法上是合法的，但编译器并不允许进行这样的标记化。

参考：注释 第2.2节；常量 第2.7节；标识符 第2.5节；预处理器标记 第3.2节；关键字 第2.6节；标记合并 第3.3.9节；空白字符 第2.1节。

## 2.4 操作符和分隔符

表2-3列出了C标记中的操作符和分隔符（标点符号）。为了帮助程序员使用没有美国英语字符的I/O设备，C提供了替代性拼写<%、%>、<:、:>、%:和%:%，它们分别等于标点符号{、}、[、]、#、##。除了这些重拼之外，头文件iso64.h定义了对某些操作符进行扩展的宏。

在传统C中，复合赋值操作符被看成是两个独立的标记，也就是1个操作符和1个等号，可以用空格进行分隔。在标准C中，复合赋值操作符被看成是一个单独的标记。

参考：复合赋值操作符 第7.9.2节；iso64.h 第11.9节；预处理器标记 第3.2节；三字符组 第2.1.4节。

表2-3 操作符和分隔符

| 标记类型 | 标 记 |
|---|---|
| 简单操作符 | ! % ^ & * - + = ~ \| . < > / ? |
| 复合赋值操作符 | += -= *= /= %= <<= >>= &= ^= \|= |
| 其他复合操作符 | -> ++ -- << >> <= >= == != && \|\| |
| 分隔符 | ( ) [ ] { } , ; : ... |
| 替代型标记重拼 | <% %> <: :> %: %:% |

## 2.5 标 识 符

标识符（或名称）由大小写拉丁字母、数字和下划线字符序列组成。标识符不能以数字开头，并且必须与现有的关键字不同。

从C99开始，标识符也可以包含统一字符名称（第2.9节）和其他因编译器而异的多字节字符。在使用统一字符名称时，不能把数字放在标识符的开始处，并且必须使用“类似字母”的字符，不能使用标点符号。C99标准（ISO/IEC 9899:1999，附录D）和ISO/IEC TR 10176-1998提供了一个准确的统一字符名称列表。

标识符：

标识符-非数字

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



标识符 标识符-非数字
标识符 数字

标识符-非数字:

非数字
统一字符名称
其他因编译器而异的字符

非数字：下列之一

```
A  B  C  D  E  F  G  H  I  J  K  L  M
N  O  P  Q  R  S  T  U  V  W  X  Y  Z
a  b  c  d  e  f  g  h  i  j  k  l  m
n  o  p  q  r  s  t  u  v  w  x  y  z
_
```

数字：下列之一

```
0  1  2  3  4  5  6  7  8  9
```

当两个标识符具有相同的拼写时（包括所有字母的大小写也必须相同），它们就是相同的。也就是说，标识符abc和aBc是不同的。

除了避免使标识符的拼写与关键字相同之外，C程序员还必须避免自己所使用的标识符意外与标准函数库中的名称相同，不论是当前的标准函数库还是语言标准所描述的“未来函数库指南”部分。标准C保留了所有以下划线开头并且后面跟一个大写字母或另一个下划线的标识符。程序员应该避免使用这类标识符。C编译器有时会使用这些标识符，对标准C进行扩展（或者由于其他目的）。

C89要求编译器至少能够区分标识符的前31个字符，C99把这个要求增加到63个。在考虑这个要求时，每个统一字符名称或多字节字符都被看成是1个字符。

例子

在标准C之前的编译器中，标识符的长度被限制在8个字符之内。因此，标识符countless和countlessone被认为是相同的标识符。一般而言，较长的名称可以提高程序的清晰性，减少发生错误的机会。使用下划线以及混合使用大小写字母可以提高较长标识符的可读性：

```
averylongidentifier
AVeryLongIdentifier
a_very_long_identifier
```

外部标识符（用存储类别extern声明的标识符）可能具有额外的拼写限制。这种标识符必须由其他软件（例如调试器或链接器）进行处理，在处理时可能会有更多的限制。C89所要求的最小容量是6个字符，并且不考虑大小写。C99把这个限制增加到31个字符，并区分大小写。但是，C99允许把统一字符名称当成是6个字符（最大为\U0000FFFF）或10个字符（\U00010000或更高）。即使在C99之前，大多数编译器允许外部名字至少为31个字符。

例子

当一个C编译器允许较长的外部标识符，但目标计算机却要求更短的外部名称时，可以使用预处理器隐藏这些较短的名称。在下面的代码中，一个外部错误处理函数具有较短（并因此比较难看明白）的名称eh73，但这个函数却用一个更具描述性的名称error_handler来表示。这是通

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



过声明一个展开结果为eh73的预处理宏error_handler实现的。

```
#define error_handler eh73
extern void error_handler();
…
error_handler("nil pointer error");
```

有些编译器允许在标识符中使用前面所指定字符的之外的其他字符。标识符中常常允许使用美元符号（$），使程序可以访问其他计算机系统所提供的特殊的非C库函数。

参考：#define命令 第3.3节；外部名称 第4.2.9节；关键字 第2.6节；多字节序列 第2.1.5节；保留的函数库标识符 第10.1.1节；统一字符名称 第2.9节。

## 2.6 关 键 字

表2-4所列出的标识符是标准C中的关键字，它们不能作为普通的标识符使用。它们可以作为宏名，因为所有的预处理都是在确认这些关键字之前进行的。关键字_Bool、_Complex、_Imaginary、inline和restrict是C99新增的。

表2-4 C99的关键字

| | | | |
|---|---|---|---|
| auto | _Bool[a] | break | case |
| char | _Complex[a] | const | continue |
| default | restrict[a] | do | double |
| else | enum | extern | float |
| for | goto | if | _Imaginary[a] |
| inline[a] | int | long | register |
| return | short | signed | sizeof |
| static | struct | switch | typedef |
| union | unsigned | void | volatile |
| while | | | |

a 这些关键字是C99新增的，在C++中并未保留。

除了表2-4所列出的关键字之外，标识符asm和fortran是常见的语言扩展。程序员还可能希望把头文件iso646.h所定义的宏（and、and_eq、bitand、bitor、compl、not、not_eq、or、or_eq、xor和xor_eq）作为保留标识符。这些标识符在C++中被保留。

例子

下面的代码是少数适合使用与关键字相同的名称作为宏名的例子之一。这个定义允许非标准编译器生成那些使用了void关键字的程序。

```
#ifndef __STDC__
#define void int
#endif
```

参考：_Bool 第5.1.5节；C++关键字 第2.8节；_Complex 第5.2.1节；#define命令 第3.3节；identifiers 第2.5节#ifndef命令 第3.5节；inline 第9.10节；<iso646.h>头文件 第11.5节；restrict 第4.4.6节；__STDC__ 第11.3节；void类型指定符 第5.9节；_Imaginary 第6.2.4节。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



预定义标识符

C99引入了预定义标识符（predefined identifier）的概念，并定义了一个预定义标识符：__func__。它的性质和关键字相似，尽管它本身并不是关键字。和预定义宏不同，预定义标识符可以遵循正常的代码块作用域规则。和关键字一样，程序员无法自己定义预定义标识符。

如果下面这个声明出现在每个函数定义的左花括号之后，C99编译器就会隐式地声明标识符__func__：

```
static const char __func__[] = "function-name";
```

这个标识符可以由调试工具使用，打印出外层函数的名称，如下所示：

```
if (failed) printf("Function %s failed\n", __func__);
```

把C程序移植到内存限制较为严格的目标计算机时，如果程序在运行时并不需要这些字符串，C编译器必须仔细地将它们去除。

参考：函数定义 第9.1节；预定义宏 第3.3.4节；作用域 第4.2.1节。

## 2.7 常 量

常量的词法类型包括4种不同类型的常量：整数、浮点数、字符和字符串：

常量：

整数常量
浮点数常量
字符常量
字符串常量

这些标记在其他语言中称为字面值，以便与其他一些值为常量（即不会变化）但又不属于这些词法类型的对象进行区分。在C中，后面这类对象的一个例子是枚举常量，它属于标识符这个词法类型。在本节中，我们使用编译C的术语常量同时表示这两种情况。

每个常量都有值和类型。下面各节描述了不同类型的常量的格式。

参考：字符常量 第2.7.3节；枚举常量 第5.5节；浮点数常量 第2.7.2节；整数常量 第2.7.1节；字符串常量 第2.7.4节；标记 第2.3节；值 第7.3节。

### 2.7.1 整数常量

整数常量可以用十进制、八进制或十六进制记法来指定：

整数常量：

十进制常量 整数后缀$_{opt}$
八进制常量 整数后缀$_{opt}$
十六进制常量 整数后缀$_{opt}$

十进制常量：

非零数字
十进制常量数字

八进制常量：

0

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



八进制常量 八进制数字

十六进制常量：

0x 十六进制数字
0X 十六进制数字
十六进制常量 十六进制数字

数字：下列之一

0 1 2 3 4 5 6 7 8 9

非零数字：下列之一

1 2 3 4 5 6 7 8 9

八进制数字：下列之一

0 1 2 3 4 5 6 7

十六进制数字：下列之一

0 1 2 3 4 5 6 7 8 9
A B C D F F a b c d e f

整数后缀：

long后缀 unsigned后缀$_{opt}$
long long后缀 unsigned后缀$_{opt}$ （C99）
unsigned后缀 long后缀$_{opt}$
unsigned后缀 long long后缀$_{opt}$ （C99）

long后缀：下列之一

l L

long long后缀:下列之一

ll LL

unsigned后缀：下列之一

u U

下面是确定整数常量的基数的规则：

1. 如果一个整数常量以字母0x或0X开始，那么它就是十六进制常量，其中a-f（或A-F）分别表示10-15。

2. 否则，如果它以数字0开始，那么它就是八进制常量。

3. 否则，它就是十进制常量。

整数常量后面可以紧随后缀字母，指定了它的类型的最小长度：

• 字母l或L表示long类型的常量
• 字母ll或LL表示long long类型的常量（C99）
• 字母u或U表示unsigned类型（int、long或long long）的常量

unsigned后缀可以按照任何顺序与long或long long后缀组合使用。小写字母l很容易与数字1混淆，因此应该避免作为后缀使用。

如果没有溢出，整数常量的值总是非负的。如果它的前面有个负号，它是应用于这个常量的单目操作符，它本身并不是常量的一部分。

整数常量的实际类型取决于它的长度、基数、后缀字母，类型表示形式取决于C编译器。确定整数常量类型的规则非常复杂，并且在非标准C、C89和C99中是不相同的。表2-5显示了所有的规则。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



如果某种类型的整数常量的值超出了表2-5中它自己那组中最后一个类型的最大可表示整数，其结果是未定义的。在C99中，编译器可以把这些大常量赋值为一种扩展整数类型，然后再加上表中的符号约定（如果所有的标准选择都是有符号的，则扩展类型也必须是有符号的。如果所有的标准选择都是无符号的，则扩展类型也必须是无符号的。在其他情况下，有符号和无符号都是可以接受的）。在C89中，和整数类型的表示形式有关的信息是由头文件limits.h提供的。在C99中，头文件stdint.h和inttypes.h包含了额外的信息。

为了说明整数常量的一些微妙之处，假设int类型使用16位的补码表示形式，long类型使用32位的补码表示形式，long long类型使用64位的补码表示形式。我们在表2-6中列出了一些有趣的整数常量、它们的真实数学值、它们的类型（在常规情况下以及在标准C规则下），以及用于存储常量的实际C表示形式。

在这张表中，一个值得注意的有趣之处是$2^{15}$和$2^{16}-1$之间的整数在写成十进制常量时为正值，但在写成八进制或十六进制常量时（被转换为int类型）为负值。尽管存在这种异常的情况，但程序员很少受到整数常量值的困扰，因为常量的表示形式即使在出现问题时仍然是相同的。

C99通过INTN_C、UINTN_C和UINTMAX_C这些宏对整数常量的长度和类型实行可移植的控制。这些宏是在stdint.h中定义的。

表2-5 整数常量的类型

| 常量 | 最初的C[a] | C89[a] | C99[a,b] |
|---|---|---|---|
| dd...d | int<br>long | int<br>long<br>unsigned long | int<br>long<br>long long |
| 0dd...d<br>0Xdd...d | unsigned<br>long | int<br>unsigned<br>long<br>unsigned long | int<br>unsigned<br>long<br>unsigned long<br>long long<br>unsigned long long |
| dd...dU<br>0dd...dU<br>0Xdd...dU | 无应用 | unsigned<br>unsinged long | unsigned int<br>unsigned long<br>unsigned long long |
| dd...dL | long | long<br>unsigned long | long<br>long long |
| 0dd...dL<br>0Xdd...dL | long | long<br>unsigned long | long<br>unsigned long<br>long long<br>unsigned long long |
| dd...dUL<br>0dd...dL<br>0Xdd...dL | 无应用 | unsigned long | unsigned long<br>unsigned long long |
| dd...dLL | 无应用 | 无应用 | long long |
| 0dd...dLL<br>0Xdd...dLL | 无应用 | 无应用 | long long<br>unsigned long long |

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



（续）

| 常量 | 最初的C[a] | C89[b] | C99[a,b] |
|---|---|---|---|
| dd...dULL<br>0dd...dULL<br>0xdd...dULL | 无应用 | 无应用 | unsigned long long |

a 所选择的类型是对应的组中能够不溢出地表示常量值的第1个类型。
b 如果所列出的类型都不够大，并且存在扩展类型，就可以使用这种扩展类型。

表2-6 整数常量的类型赋值

| C常量记法 | 真实值 | 传统类型 | 标准C类型 | 实际表示形式 |
|---|---|---|---|---|
| 0 | 0 | int | int | 0 |
| 32767 | $2^{15}-1$ | int | int | 0x7FFF |
| 077777 | $2^{15}-1$ | unsigned | int | 0x7FFF |
| 32768 | $2^{15}$ | long | long | 0x00008000 |
| 010000 | $2^{15}$ | unsigned | unsigned | 0x8000 |
| 65535 | $2^{16}-1$ | long | long | 0x0000FFFF |
| 0xFFFF | $2^{16}-1$ | unsigned | unsigned | 0xFFFF |
| 65536 | $2^{16}$ | long | long | 0x00010000 |
| 0x10000 | $2^{16}$ | long | long | 0x00010000 |
| 2147483647 | $2^{31}-1$ | long | long | 0x7FFFFFFF |
| 0x7FFFFFFF | $2^{31}-1$ | long | long | 0x7FFFFFFF |
| 2147483648 | $2^{31}$ | long[a] | unsigned long<br>C99:long long | 0x80000000 |
| 0x80000000 | $2^{31}$ | long[a] | unsigned long | 0x80000000 |
| 4294967295 | $2^{32}-1$ | long[a] | unsigned long<br>C99:long long | 0xFFFFFFFF<br>0x00000000FFFFFFFF |
| 0xFFFFFFFF | $2^{32}-1$ | long[a] | unsigned long | 0xFFFFFFFF |
| 4294967296 | $2^{32}$ | 未定义 | 未定义<br>C99：long long | 0x0<br>0x0000000100000000 |
| 0x100000000 | $2^{32}$ | 未定义 | 未定义<br>C99：long long | 0x0<br>0x0000000100000000 |

a 该类型无法准确地表示值。

例子

如果long类型采用32位的补码表示形式，下面这个程序可以判断实际使用的规则：

```
#define K 0xFFFFFFFF /* -1 in 32-bit, 2's compl. */
#include <stdio.h>
int main()
{
    if (0<K) printf("K is unsigned (Standard C)\n");
    else printf("K is signed (traditional C)\n");
    return 0;
}
```

参考：整数类型的转换 第6.2.3节；扩展整数类型 第5.1.4节；整数类型 第5.1节；

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



INTMAX_C 第21.5节；INTN_C 第21.3节；limits.h 第5.1.1节；溢出 第7.2.2节；stdint.h 第21章；单目负号操作符 第 7.5.3节；无符号整数 第5.1.2节。

### 2.7.2 浮点数常量

浮点常量可以写成带小数点的形式，也可以写成带符号的指数形式，也可以两者皆采用。标准C允许使用一个后缀字母（浮点后缀）指定float和long double类型的常量。如果不使用后缀，浮点常量的类型默认为double：

浮点常量：

十进制浮点常量
十六进制浮点常量 （C99）

十进制浮点常量：

数字序列 指数 浮点后缀$_{opt}$
带小数点的数字 指数$_{opt}$ 浮点后缀$_{opt}$

数字序列：

数字
数字序列 数字

带小数点的数字：

数字序列 .
数字序列 . 数字序列
. 数字序列

数字：下列之一

0 1 2 3 4 5 6 7 8 9

指数：

e 符号部分$_{opt}$ 数字序列
E 符号部分$_{opt}$ 数字序列

符号部分：下列之一

+ -

浮点后缀：下列之一

f F l L

如果没有溢出，浮点常量的值总是非负的。如果浮点常量的前面有个负号，它是作用于这个常量的单目操作符，它本身并不是浮点常量的一部分。如果无法准确地表示一个浮点常量，编译器可以选择最接近的表示值V，或者略大小或略小于V的接近值。如果浮点常量的数值由于太大或太小而难以表示，其结果就是不可预测的。有些编译器会向程序员警告这个问题，但大多数编译器会悄然地换用其他可表示的值。在标准C中，浮点限制是在float.h头文件中指定的。像无限和NaN（不是数）这样的特殊浮点常量是在math.h中定义的。

在C99中，复数浮点常量被写成带有虚数常量_Complext_I（或I）的浮点常量表达式。虚数常量是在complex.h中定义的。

例子

下面这些是合法的十进制浮点常量：0.、3e1、3.14159、.0、1.0E-3、1e-3、1.0、.00034、

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



2e+9。下面这些浮点常量在标准C中是合法的：1.0f、1.0e67L、0E1L。

C99复数常量的一个例子是1.0 + 1.0 * I（如果已经包含了complex.h）。

C99允许用十六进制记法表示浮点常量。以前版本的C只允许十进制浮点常量。十六进制格式使用字母p分隔小数部分和指数部分，这是因为惯用的表示指数的字母e会与十六进制数字混淆。二进制指数是一种有符号十进制数，表示2的幂（并不是十进制常量所使用的10的幂，也不是有些人所猜想的16的幂）。

十六进制浮点常量： （C99）

十六进制前缀 带小数点的十六进制数字 二进制指数 浮点后缀$_{opt}$
十六进制前缀 十六进制数字序列 二进制指数 浮点后缀$_{opt}$

十六进制前缀：

0x
0X

带小数点的十六进制数字：

十六进制数字序列
十六进制数字序列 . 十六进制数字序列
. 十六进制数字序列

十六进制数字序列

十六进制数字
十六进制数字序列 十六进制数字

二进制指数

p 符号部分$_{opt}$ 数字序列
P 符号部分$_{opt}$ 数字序列

如果FLT_RADIX（float.h）不等于2，可能无法精确地表示十六进制浮点常量。如果无法精确地表示，它所指定的值必须是最接近的可表示值。

参考：complex.h 第23.2节；double类型 第5.2节；float.h 第5.2节；上溢和下溢 第7.2.2节；浮点类型的长度 第5.2节；单目负号操作符 第7.5.3节。

### 2.7.3 字符常量

字符常量写成一对单引号内的一个或多个字符。有一种特殊的转义机制可以表示那些不方便或无法在源程序中直接输入的字符值或数值。标准C允许在字符常量前面添加前缀L，表示宽字符常量。

字符常量：

'c字符序列'
L'c字符序列' （C99）

c字符序列：

c字符
c字符序列 c字符

c字符：

除了单引号（'）、反斜杠（\）和换行符之外的所有源字符

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



转义字符
统一字符名称 （C99）

可以通过使用转义字符在字符常量中包含单引号、反斜杠和换行符，如第2.7.5节所述。用转义字符来表示那些在源程序中难以阅读的字符（例如格式字符）是种很好的做法。C99还允许在字符常量中使用统一字符名称（第2.9节）。

如果字符常量前面没有前缀L，它的类型就是int。这种字符常量一般用于表示一个单独的字符或转义码（第2.7.7节），常量的值是执行字符集中对应的字符的整数编码。如果一个字符常量是从一个char类型的对象转换而来，就会计算它的整数值。例如，如果char类型是一种8位的有符号类型，字符常量'\377'就具有符号位，它的值是 - 1。如果符合下列条件，字符常量的值是由编译器所定义：

1. 在执行字符集中不存在对应的字符。
2. 在常量中出现了超过1个的执行字符。
3. 数值转义符的值在执行字符集中无法表示。

例子

下面是一些单字符常量的例子，以及它们在ASCII编码中的十进制值。

| 字符 | 值 | 字符 | 值 |
|---|---|---|---|
| 'a' | 97 | 'A' | 65 |
| ' ' | 32 | '?' | 63 |
| '\r' | 13 | '\0' | 0 |
| '"' | 34 | '\377' | 255 |
| '%' | 37 | '\23' | 19 |
| '8' | 56 | '\\' | 92 |

标准C的宽字符常量由前缀字母L所指定，它的类型是wchar_t，这是stddef.h头文件所定义的一种整数类型。宽字符常量的目的是允许C程序员表示那些无法用char类型表示的字符（例如日文字符）。宽字符常量一般由一系列的字符和转义码组成，一起形成了一个单独的多字节字符。多字节字符与对应的宽字符之间的映射是由编译器定义的，对应于mbtowc函数，这个函数在运行时执行多字节字符到宽字符的转换。如果多字节字符使用了转移状态编码，则宽字符常量必须以初始转移状态开始和结束。如果一个宽字符常量包含了超过1个的宽字符，它的值是由编译器所定义的。

多字符常量 整数常量和宽字符常量可以包含一系列的字符。把这个字符序列映射到执行字符集之后，仍然可能存在超过1个的执行字符。这种常量的含义是由编译器决定的。

旧式编译器所使用的一种约定是把4字节的整数常量表示了为一个4字符常量，例如'gR8t'。这种用法是不可移植的，因为有些计算机可能不允许这种做法，而且不同的计算机的整数类型的长度以及“字符顺序”（即字符包装为字的顺序）可能不同。

例子

在一种4字节整数并且字节从左向右包装的ASCII实现中，'ABCD'的值将是$41424344_{16}$（'A'的值是0x42，'B'的值是0x42，接下来以此类推）。但是，如果使用了从右向左的包装，'ABCD'的值将是$44434241_{16}$。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



参考：ASCII字符 附录A；字节顺序 第6.1.2节；字符码 第2.1节；char类型 第5.1.3节；转义字符 第2.7.5节；格式字符 第2.1节；mbtowc工具函数 第11.7节；多字节字符 第2.1.5节；wchar_t 第11.1节。

### 2.7.4 字符串常量

字符串常量是位于一对双引号内部的字符序列（可以为空）。程序员可以使用和字符常量相同的转义机制表示字符串中的字符。标准C允许在字符串常量前面添加前缀L，指定宽字符串常量。

字符串常量：

" s字符序列$_{opt}$ "
L " s字符序列$_{opt}$ " （C89）

s字符序列：

s字符
s字符序列 s字符

s字符：

除了双引号（"）、反斜杠（\）和换行符之外的所有源字符
转义字符
统一字符名称 （C99）

双引号、反斜杠和换行符可以通过使用转义字符包含在字符串常量中，如第2.7.5节所述。用转义字符来表示那些在源程序中难以阅读的字符（例如格式字符）是种很好的做法。C99允许在字符串常量中使用统一字符名称（第2.9节）。

例子

下面列出了5个字符串常量。

```
""
"\""
"Total expenditures: "
"Copyright 2000 \
Texas Instruments. "
"Comments begin with '/*'.\n"
```

第4个字符串相当于"Copyright 2000 Texas Instruments."，在0和T之间并没有换行符。

对于包含了n个字符的非宽字符串，在运行时将会静态分配一块可以容纳n+1个字符的内存，其中前n个字符是该字符串中的字符，最后一个字符是null字符'\0'。这块内存相当于这个字符串常量的值，它的类型是char [n+1]。类似，宽字符串常量由n个宽字符以及后面的一个null宽字符组成，它的类型是wchar_t [n +1]。

例子

sizeof操作符返回它的操作数的长度，而strlen函数（第13.4节）返回一个字符串的字符数量。因此，sizeof("abcdef")是7而不是6，sizeof("")是1而不是0，strlen("abcdef")是6，strlen("")是0。

如果一个字符串常量不是&操作符的参数，也不是sizeof操作符的参数，也不是作为字符数组的初始化值，就可以应用寻常数组转换，把这个字符串转换为由一个指针所指向的字符数组（指针指向字符串中的第1个字符）。

例子

char *p = "abcdef" 这个声明产生一个指针p，它被初始化为指向一块存储了7个字符的内存。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



这7个字符分别是'a'、'b'、'c'、'd'、'e'、'f'和'\0'。

单字符的字符串常量的值和字符常量的值是不同的。int X = (int) "A"；这个声明使X被初始化为一个指向存储了2个字符的内存块（包含'A'和'\0'）的指针的整数值（前提是这个指针可以用int类型来表示）。但是，int Y = (int) 'A'；这个声明使Y被初始化'A'的字符码（在ISO 646编码中为0x41）。

字符串常量的存储 存储了字符串常量的字符的内存是无法修改的，因为它是只读的，也就是存在对修改的物理保护。有些函数（例如mktemp）要求向它传递指向字符串的指针，以便对字符串进行修改。因此，不要向这类函数传递字符串常量。反之，初始化一个包含了字符串常量的所有字符的字符数组（非const），并把这个数组的第1个元素的地址传递给这类函数。

例子

考虑下面这3个声明：

```
char p1[]= "Always writable";
char *p2 = "Possibly not writable";
const char p3[] = "Never writable"; /* Standard C only */
```

p1、p2和p3的值都是指向字符数组的指针，但它们在写入能力方面各不相同。p1[0] = 'x'这个赋值总是可行。p2[0] = 'x'可能成功，也可能导致一个运行时错误。p3[0] = 'x'总是会导致一个编译错误，这是由于const的含义所致。

不要以为所有的字符串常量会存储在不同的地址。标准C允许编译器为两个包含相同字符的字符串常量使用相同的存储地址。

例子

下面是一个简单的程序，显示了各种不同的字符串实现的区别。如果字符串常量是在只读内存中分配的，对string1[0]的赋值将会导致运行时错误。

```
char *string1, *string2;
int main() {
    string1 = "abcd"; string2 = "abcd";
    if (string1==string2) printf("Strings are shared.\n");
    else printf("Strings are not shared.\n");
    string1[0] = '1'; /* RUN-TIME ERROR POSSIBLE */
    if (*string1=='1') printf("Strings writable\n");
    else printf("Strings are not writable\n");
    return 0;
}
```

字符串的延续 一个字符串常量一般出现在源程序文本的一行中。如果字符串太长，不方便写在一行中，可以在包含这个字符串的所有文本行（除最后一行外）后面加上一个反斜杠字符\。在这种情况下，反斜杠字符和行末符号将被忽略。这就允许把一个字符串常量分在多行书写。有些旧式的编译器会删除延续行的前导空白字符，但这是不正确的做法。

标准C会自动连接相邻的字符串常量和相邻的宽字符串常量，并在最后一个字符串的末尾添加一个null字符。因此，在标准C中，\延续机制的一种替代方法就是把长字符串分隔为几个字符串。在C99中，宽字符串和常规的字符串也可以按照这种方式进行连接，产生一个宽字符串常量。在C89中，这种做法是不允许的。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



例子

字符串s1的初始化在标准C和标准C之前的编译器中是允许的，但字符串s2的初始化则只被标准C所允许：

```
char s1[] = "This long string is acc\
eptable to all C compilers.";
char s2[] = "This long string is permissible "
            "in Standard C.";
```

换行符（即执行字符集中的行末符）可以通过在字符中常量中增加一个转义序列\n插入到这个字符串中。这应该不会与字符串常量所使用的行延续符混淆。

宽字符串 在标准C中，字符串常量前面如果有一个前缀字母L，它就是个宽字符串常量，它的类型是“ wchar_t的数组 ”。它表示来自一个扩展执行字符集的宽字符序列，可能为诸如日语这样的语言所使用。宽字符串常量中的字符是多字节字符串，将按照一种取决于编译器的方式映射到一个宽字符序列（mbstowcs函数在运行时执行类似的任务）。如果多字符字符使用了一种转换状态编码，这个宽字符串常量必须以起始转移状态开始和结束。

参考：数组类型 第5.4节；const类型指定符 第4.4.4节；数组类型的版本 第6.2.7节；转义字符 第2.7.5节；初始值 第4.6节；mbstowcs工具函数 第11.8节；mktemp工具函数 第15.16节；多字节字符 第2.1.5节；指针类型 第5.3节；预处理器词法约定 第3.2节；sizeof操作符 第7.5.2节；strlen工具函数 第13.4节；空白字符 第2.1节；普通单目转换 第6.3.3节；wchar_t 第11.1节；统一字符名称 第2.9节。

### 2.7.5 转义字符

转义字符可以在字符常量和字符串常量中表示难以或无法直接在源程序中表示的字符。转义字符可以分为两种类型，一种是“ 字符转义 ”，可以用于表示一些特殊的格式和特殊的字符，另一种是“ 数值转义 ”，允许用一个数值编码来指定一个字符。C99还增加了统一字符名称作为转义字符。

转义字符：

```
\ 转义码
统一字符名称                    (C99)
```

转义码

```
字符转义码
八进制转义码
十六进制转义码                  (C89)
```

字符转义码：下列之一

```
n   t   b   r   f
v   \   '   ''
a   ?                           (C89)
```

八进制转义码：

```
八进制数字
八进制数字   八进制数字
八进制数字   八进制数字   八进制数字
```

十六进制转义码：

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



x 十六进制数字
十六进制转义码 十六进制数字 （C89）

这些转义字符的含义将在下面各节中讨论。

如果反斜杠后面的字符既不是个八进制数字，也不是字母x，也不是前面所列出的转义码之一，其结果就是未定义的（在传统C中，这种反斜本将被忽略）。在标准C中，反斜杠后面的所有小写字母被保留，用于将来的语言扩展。大写字母可以用于由编译器所定义的特定扩展。

参考：统一字符名称 第2.9节。

### 2.7.6 字符转义码

字符转义码用于表示一些常用的特殊字符，并且与目标计算机的字符集无关。表2-7列出了可以出现在反斜杠后面的字符以及它们的含义。

表2-7 字符转义码

| 转义码 | 转换 | 转义码 | 转换 |
|---|---|---|---|
| a[a] | 警告（即响铃） | v | 垂直制表符 |
| b | 退格 | \ | 反斜杠 |
| f | 换页 | ' | 单引号 |
| n | 换行 | " | 双引号 |
| r | 回车 | ?[a] | 问号 |
| t | 水平制表符 | | |

a 标准C所增加。

转义码\a一般映射到“响铃”或输出设备上的其他声音信号（例如ASCII的control-G，它的值是7）。转义码\?用于在较罕见的情况下表示问号（在这些情况下如果直接使用问号，可能会被误认为是三字符组的一部分）。

引用标记（"）可能出现在没有前导反斜杠的字符常量中，撇号（'）可能出现在没有反斜杠的字符串常量中。

例子

为了说明如何使用字符转义码，下面是一个小程序，对输入的行数（实际上是换行符的数量）进行计数。getchar函数返回下一个输入字符，直到到达了输入的结束，此时getchar返回EOF宏的值，后者是在stdio.h中定义的。

```
#include <stdio.h>
int main(void) /* Count the number of lines in the input. */
{
    int next_char;
    int num_lines = 0;
    while ((next_char = getchar()) != EOF)
        if (next_char == '\n')
            ++num_lines;
    printf("%d lines read.\n", num_lines);
    return 0;
}
```

参考：字符常量 第2.7.3节；EOF 第15.1节；getchar工具函数 第15.6节；stdio.h 第15.1节；

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



字符串常量 第2.7.4节；三字符组 第2.1.4节。

### 2.7.7 数值转义码

数值转义码允许直接用八进制（在标准C中，还允许十六进制）编码值表示执行字符集中的一个字符。它最多允许使用3个八进制数字以及任意数量的十六进制数字。但是，标准C禁止使用unsigned char范围之外的值表示正常的字符常量，并禁止使用wchar_t范围之外的值表示宽字符常量。例如，在ASCII编码中，字符'a'可以被写成'141'或'x61'，字符'?'可以写成'\77'或'x3F'。用于终止字符串的null字符总是写成\0。如果一个数值转义码的值并不对应于执行字符集中的某个字符，其结果由编译器决定。

例子

下面这段简短的代码显示了数值转义码的用法。变量inchar的类型为int。

```
for (;;) {
   inchar = receive();
   if (inchar == '\0') continue;        /* Ignore */
   if (inchar == '\004') break;         /* Quit */
   if (inchar == '\006') reply('\006'); /* ACK */
   else reply('\025');                  /* NAK */
   }
```

在使用数值转义码时，程序员必须小心，原因有二。首先，数值转义码可能依赖于字符编码，因此是不可移植的。用宏定义来隐藏转义码是一种更好的做法，这样可以方便地对它们进行修改：

```
#define NUL '\0'
#define EOT '\004'
#define ACK '\006'
#define NAK '\025'
```

其次，数值转义码的语法是脆弱的。当八进制数字达到3个或者遇到第1个字符不是八进制数字的字符时，八进制转义符就会终止。因此，字符串"\0111"由2个字符组成，分别是\011和1。字符串"\090"由3个字符组成，分别是\0、9和0。十六进制转义序列也会遇到这种终止问题，尤其是因为它们的长度可以是任意的。为了在一个字符串中停止一个标准C的十六进制转义码，可以分割字符串：

```
"\xabc"     /* This string contains one character. */
"\xab" "c"  /* This string contains two characters. */
```

有些非标准的C编译器所提供的十六进制转义序列和八进制转义序列一样，只允许固定数量的十六进制数字。

参考：字符常量 第2.7.3节；#define 第3.3节；宏定义 第3.3节；null字符 第2.1节；字符串常量 第2.7.4节；执行字符集 第2.1节。

## 2.8 C++兼容性

本节列出了C和C++的一些词法区别。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



### 2.8.1 字符集

标准C的标记重拼和三字符组也是C++标准的一部分，但它们在标准C++之前的编译器中并不常见。C和C++都允许统一字符名称，它们所使用的语法相同，但只有C明确允许在标识符中使用其他由编译器所定义的字符（个人希望C++编译器以扩展的形式提供这个功能）。

### 2.8.2 注释

C99的注释可以为C++所接受，反之亦然。在C99之前，标准C并不使用//作为注释起始符，因此字符序列//*在C和C++中的解释并不相同（它的细节作为练习留给读者）。

### 2.8.3 操作符

C++增加了3个新的复合操作符：

```
.*    ->*    ::
```

由于这些标记的组合在标准C程序中是非法的，所以不会影响C程序移植为C++程序。

### 2.8.4 标识符和关键字

表2-8所列出的标识符是C++的关键字，但不是C的关键字。但是，标准C保留了关键字wchar_t，C99还保留了关键字bool、true、false，作为标准函数库的一部分。

表2-8 额外的C++关键字

| | | | |
|---|---|---|---|
| asm | export | private | throw |
| bool | false | protected | true |
| catch | friend | public | try |
| class | mutable | reinterpret_cast | typeid |
| const_cast | namespace | static_cast | typename |
| delete | new | template | using |
| dynamic_cast | operator | this | virtual |
| explict | | | wchar_t |

### 2.8.5 字符常量

在C中，单字符常量的类型是int，但在C++中为char。多字符常量（它们依赖于编译器）在两种语言中都是int类型。在实际使用中，这点差别很少会产生问题，因为在整数上下文环境中所使用的C++字符常量会被隐式地提升为int类型。但是，sizeof('c')在C++中等于sizeof(char)，但在C中等于sizeof(int)。

## 2.9 关于字符集、指令集和编码

在C语言所产生的时代，人们尚未理解国际性的、多语言编程社区的需要。标准C对C语言进行了扩展，以容纳这个社区。本节简单介绍了标准C所解决的一些问题（以及相关的历史），使C语言对于非英语用户更为友好。

指令集和ASCII　每一种文化都根据一种可打印字母或符号的字符指令集进行书面交流。对于美国英语而言，这个指令集包括通常的52个大写和小写字母、十进制数字以及一些标点符号。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



这些字符一共大约100个，它们（由美国英语程序员和计算机生产厂商）通过一种7位的编码（称为ASCII）被赋予特定的二进制值。经过编码的字符出现在标准键盘上，并可以在C语言定义中找到。

遗憾的是，其他文化具有不同的指令集。例如，英国的英语使用者所使用的是£而不是$，但7位的ASCII码并没有包含这个字符。像俄语和希伯来语这样的语言使用和英语完全不同的字母，像汉语、日语和朝鲜语（CJK）这样的语言所使用的指令集由几千个符号组成。当今的程序员常常需要创建要求读者和写入多种语言的文本，包括他们的母语。他们还需要在程序中使用母语注释和母语变量名。他们所编写的程序需要移植到其他文化，至少在其他文件中不会成为非法（除非读者理解梵语并且读者的计算机可以显示梵语字符，否则读者就无法读懂梵语注释）。

这个问题逐渐被人们所认识，人们设计了一些不完整的解决方案，这些解决方案至今仍然得到支持。例如，ISO 646-1083不变性代码集被定义为ASCII的一个子集，在许多非英语的字符集中较为常用。人们又发明了一些方法，替换在更小的字符集中不存在的C字符，例如{、}、[、]和#。

ISO/IEC 10646　字符集的通用解决方案是由ISO/IEC标准10646（加上修正案）定义的，即通用多八位编码字符集（Universal Multiple-Octet Coded Character Set，UCS）。它定义了一种4字节（或4个八位编码）的编码方案UCS-4，可以表示地球上所有文化的所有字符，并保留了充足的空间。UCS-4有一个非常有用的16位子集，称为基本多语言平面（Basic Multilingual Plane），即UCS-2，它由前两个字节为0的UCS-4编码所组成。UCS-2可以表示绝大多数文化的指令集，包括大约2万个CJK象形文字。但是，16位还不足以表示的有的字符，在计算机上操纵不低于32位的长度时显得不便，因此UCS-4是合适的方案。

Unicode字符集标准最初是由Unicode协会（www.unicode.org）所制订的一个16位的编码方案。Unicode 3.0现在完全与ISO/IEC 10646兼容。以前版本的Unicode只与UCS-2兼容。Unicode网站对字符编码提供了非常好的技术介绍。

字符集标准UCS-4、UCS-2和Unicode都与ASCII兼容。高8位为全0的只使用低8位的16位字符是一种扩展的ASCII字符，称为Latin-1。最初版本的7位ASCII字符现在称为Basic Latin，它是前9位均为0的UCS-2字符。

宽字符和多字节字符　字符表示形式大于传统的8位的字符称为宽字符。遗憾的是，8位（或7位）的字符并不容易根除。许多计算机和遗留的应用程序基于8位的字符，人们设计了计多方案，用8位或7位字符序列表示更大的字符指令集和宽字符。它们称为多字节编码或多字节字符。宽字符一般使用固定长度的表示形式，但多字节字符一般使用一个字节表示一些字节，使用两个字节表示另一些字符，使用三个字节表示其他一些字符，接下来以此类推。其中一个或多个8位字符被当成“转义”或“转移”字符，用于开始多字节序列。

我们今天在标准C中所看到的是一些技术的组合：有些方法用于处理明显的ASCII变型（三字符组和两字符组），有些方法用于处理完全现代的宽字符环境，有些方法用于处理I/O期间的多字节字符。最近，有一种方法可以按照一种可移植的方式表示任何文化适应的C程序（统一字符以及标识符中特定于本地文件的字符）。

统一字符名称　C99引入了一个概念，允许在字符常量、字符串常量和标识符中指定任何

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



UCS-2或UCS-4字符。它的语法如下：

统一字符名称：

```
\u 十六进制4位数
\U 十六进制4位数
```

十六进制4位数：

```
十六进制数字  十六进制数字  十六进制数字  十六进制数字
```

每个十六进制4位数包括4个十六进制数字，可以指定一个16位的值。十六进制4位数的值在ISO/IEC 10646中是以统一字符的4位数字和8位数字的“短标识符”指定的。\unnnn所指定的字符与\u0000nnnn所指定的字符相同。

C并不允许短标识符小于00A0的统一字符名称，但0024($)、0040(@)和0060(')例外。C也不允许短标识符在D800到DFFF范围内的统一字符名称。这些是控制字符，包括DELETE以及为UTF-16所保留的字符。使用标记合并创建统一字符名称的结果是未定义的。

参考：标识符和统一字符名称 第2.5节；标记合并 第3.3.9节。

## 2.10 练 习

1. 下面哪些是词法标记？
   （a）关键字
   （b）注释
   （c）空格
   （d）十六进制常量
   （e）三字符组
   （f）宽字符串常量
   （g）括号
2. 假设下面这些由源字符组成的字符串是由C编译器处理的。哪些标记将被确认为一系列的C标记？每个字符串中可以找到几个标识符（不必担心有些标记序列可能无法出现在合法的C程序中）？
   （a）`X++Y`
   （b）`-12uL`
   （c）`1.37E+6L`
   （d）`"String""FOO"""`
   （e）`"String+ \"FOO\""`
   （f）`x**2`
   （g）`"X??/"`
   （h）`B$C`
   （i）`A*=B`
   （j）`while##DO`
3. 删除下面这个C程序片段中的所有注释。

```
/**/*/*"*/*/*"//*//**/*/
```

4. 一个标准C编译器必须在一个输入程序中执行下面每个操作。这些操作是按什么顺序执行的？
   把字符收集为标记
   删除注释
   转换三字符组
   处理行的延续
5. 下面显示了一些比较差的标识符，它们为什么比较差？

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章



(a) `pipesendintake`
(b) `Const`
(c) `l0`
(d) `O77U`
(e) `SYS#input`

6. 编写一些简单的标准C代码码，使它们分别因为下面的原因导致它们在C++中是非法的或者有不同的解释。

(a) C89不存在//风格的注释
(b) 常量类型
(c) 关键字冲突

HZ BOOKS 华章图书