**The C Programming Language**

中文版序（节选）

**C程序设计语言最早是由Dennis Ritchie于1973年设计的并实现的。从那时起，C语言已经从其位于贝尔实验室的发源地传播到了世界各地。它已经成为全球程序员的公共语言，并由此诞生了两个新的主流语言C++和Java——它们都建立在C语言的语法和基本结构的基础上。现在，世界上的许多软件都是在C语言及其衍生的各种语言的基础上开发出来的。**

引言（节选）

C语言是一种通用的程序设计语言。它同UNIX系统之间具有非常密切的联系——C语言是在UNIX系统上开发的。但是，C语言并不受限于任何一种操作系统或机器。由于它很适合用来编写编译器和操作系统，因此被成为“系统编程语言”，但它同样适合于编写不同领域中的大多数程序。

C语言的很多重要概念来源于由Martin Richards开发的BCPL语言。BCPL语言对C语言的影响间接地来自于B语言，它是由Ken Thompson为第一个UNIX系统而于1970年在DEC PDP-7计算机上开发的。

**（省略部分关于语言特性的论述，我们在之后会慢慢学习，现在看了也不懂）**

1983年，美国国家标准协会（ANSI）成立了一个委员会以制定一个现代的、全面的C语言定义。最后的结果就是1988年完成的ANSI标准，即“ANSI C”。该标准的大部分特性已被当前的编译器所支持。

**（省略部分关于新旧标准的讨论）**

本书是按照下列结构编排的：第1章将对C语言的核心部分进行简要介绍，其目的是让读者能尽快开始编写C语言程序，因为我们深信，实际编写程序才是学习一种新语言的好方法。这部分的内容的介绍假定的这对程序设计的基本元素有一定的了解。我们在这部分内容中没有解释计算机、编译等概念，也没有解释诸如n=n+1这样的表达式。**（如果大家对任何地方有疑问，都可以课下和我讨论）**我们将尽量在合适的地方介绍一些实用的程序设计技术，但是，本书的中心目的并不是介绍数据结构和算法。在篇幅有限的情况下，我们将专注于讲解语言本身。

从第2章到第6章将更详细地讨论C语言的各种特性，所采用的方式将比第1章更加形式化一些。其中的重点将放在一些完整的程序例子上，而并不仅仅是一些孤立的程序段。第2章介绍基本的数据结构、运算符和表达式。第3章介绍控制流，如if-else、switch、while和for等。第4章介绍函数和程序结构——外部变量、作用域规则和多源文件等，同时还会讲述一些预处理器的知识。第5章介绍指针和地址运算。第6章介绍结构和联合。

第7章介绍标准库。标准库提供了一个于操作系统交互的公用接口。这个函数库是由ANSI标准定义的，这就意味着所有支持C语言的机器都会支持它，因此，使用这个库执行输入、输出或其他访问操作系统的操作的程序可以不加修改地运行在不同机器上。

第8章介绍C语言程序和UNIX操作系统之间的接口，我们将重点放在输入/输出、文件系统和存储分配上。尽管本章中的某些内容是针对UNIX系统所写的，但是使用其他系统的程序员仍然会从中获益，比如深入了解如何实现标准库以及有关可移植性方面的一些建议。**（尽管作者这么说，我们还是很可能选择放弃这一章，因为我们的目的不是深入了解，而是简单了解一下）**

附录A是一个语言参考手册。虽然C语言的语法和语义的官方证实定义是ANSI标准本身，但是，ANSI标准的文档首先是写给编译器的编写者看的，因此，对程序员来说不一定最合适。本书中的参考手册采用了一种很不严格的形式，更简洁地对C语言的定义进行了介绍。附录B是对标准库的一个总结，它同样是为程序员而非编译器实现者准备的。附录C对标准C语言相对最初的C语言版本所做出的变更做了一个简短的小结。

# 第1章 导言

在本书的开篇，我们首先概要地介绍 C 语言，主要是通过实际的程序引入 C 语言的基本元素，至于其中的具体细节、规则以及一些例外情况，在此暂时不多做讨论。因此，本章不准备完整、详细地讨论 C 语言中的一些技术（当然，这里所举的所有例子都是正确的）。我们是希望读者能尽快地编写出有用的程序，为此，本章将重点介绍一些基本概念，比如变量与常量、算术运算、控制流、函数、基本输入／输出等。而对于编写较大型程序所涉及到的一些重要特性，比如指针、结构、C 语言中十分丰富的运算符集合、部分控制流语句以及标准库等，本章将暂不做讨论。

这种讲解方式也有缺点。应当提请注意的是，在本章的内容中无法找到任何特定语言特性的完整说明，并且，由于比较简略，可能会使读者产生一些误解；再者，由于所举的例子并没有用到 C 语言的所有强大功能，因此，这些例子也许并不简洁、精炼。虽然我们已经尽力将这些问题的影响降到最低，但问题肯定还是存在。另一个不足之处在于，本章所讲的某些内容在后续相关章节还必须再次讲述。我们希望这种重复给读者带来的帮助效果远远超过它的负面影响。

**（这就好比参观一个城市，在短短的几个小时里，你可能会匆匆游览几个主要的经典、听到一些有趣的传说或故事，但仅靠这一段旅程，你无法真正了解这个城市，甚至对听到和看到的东西一知半解。要想认识并融入一座城市，必须在其中生活很多年。不过，这样的游览之后你会对城市的总体情况有一些了解，知道城市的特殊之处，并且对某些方面产生兴趣）**

## 1.1.入门

对于所有语言的初学者来说， 编写的第一个程序几乎都是相同的，即：

在控制台打印出下列内容：**hello, world**

**“1.1.c”**

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7 | #include <stdio.h>  int main()  {  printf("hello, world\n");  } | 执行结果：  C:\Windows\system32\cmd.exe  （这个黑框框就叫cmd控制台） |

下面对程序本身做些说明。一个C语言程序，无论其大小如何，都是由函数和变量组成的。函数中包含一些语句，以指定所要执行的计算操作；变量则用于存储计算过程中使用的值。在本例中，函数的名字为main。通常情况下，函数的命名没有限制，但main是一个特殊的函数名——每个程序都从main函数的起点开始执行，这意味着每个程序都必须在某个位置包含一个main函数。

main 函数通常会调用其它函数来帮助完成某些工作，被调用的函数可以是程序设计人员自己编写的，也可以来自于函数库。上述程序段中的第一行语句

**#include <stdio.h>**

用于告诉编译器在本程序中包含标准输入／输出库的信息。许多C语言源程序的开始处都包含这一行语句。

函数之间进行数据交换的一种方法是调用函数向被调用函数提供一个值（称为参数）列表。函数名后面的一对圆括号将参数列表括起来。在本例中，main函数不需要任何参数，因此用空参数表()表示。

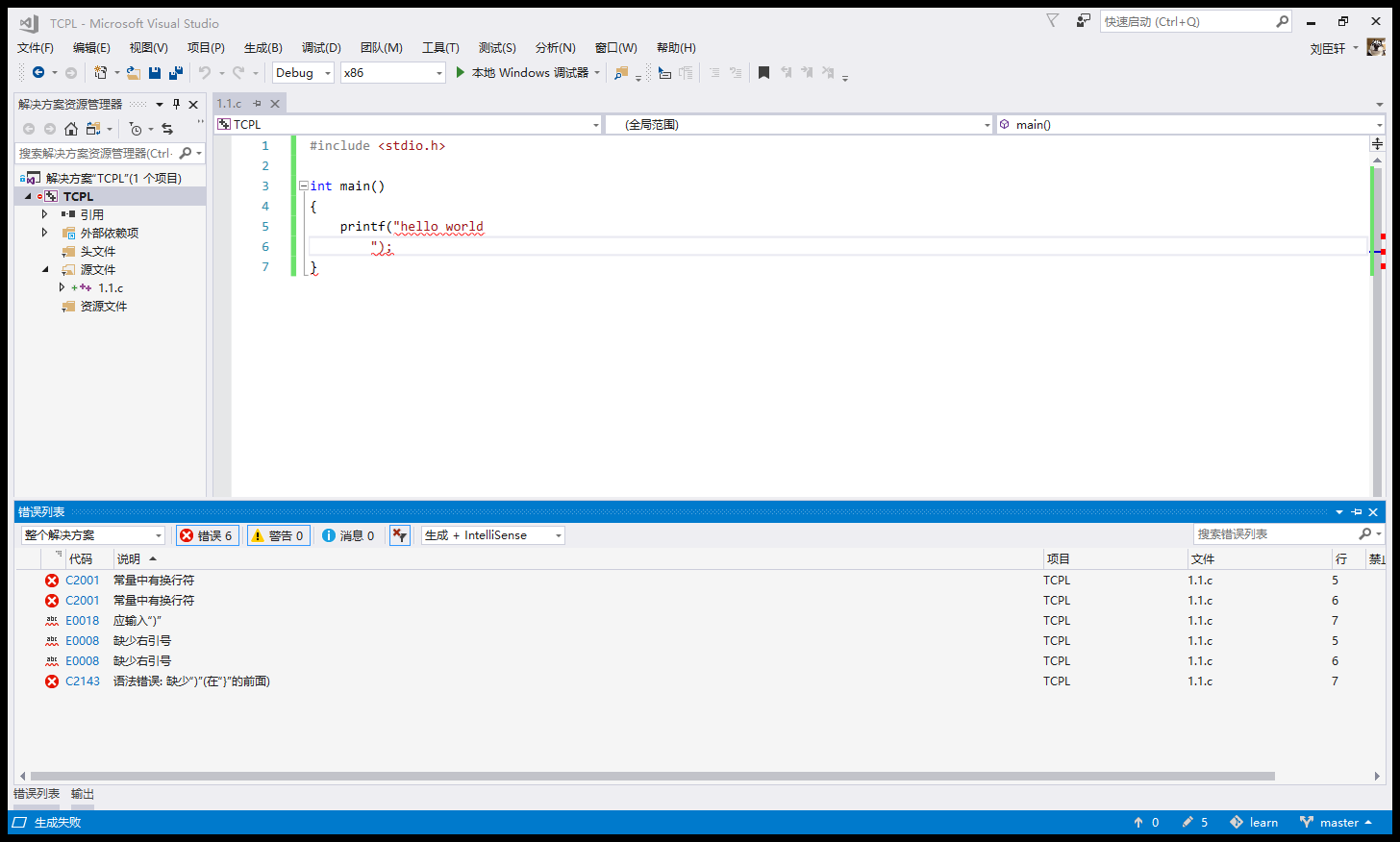
函数中的语句用一对花括号{}括起来。本例中的main函数仅包含下面一条语句：

**printf("hello, world\n");**

调用函数时，只需要使用函数名加上用圆括号括起来的参数表即可。上面这条语句将"hello,world\n"作为参数调用printf函数。printf 是一个用于打印输出的库函数，在此处， 它打印双引号中间的字符串。

用双引号括起来的字符序列称为字符串或字符串常量，如"hello, world\n"就是一个字符串。目前我们仅使用字符串作为printf及其它函数的参数。

在 C 语言中，字符序列\n 表示换行符，在打印中遇到它时，输出打印将换行，从下一行的左端行首开始。如果去掉字符串中的\n（这是个值得一做的练习），即使输出打印完成后也不会换行。在 printf 函数的参数中，只能用\n 表示换行符。如果用程序的换行代替\n，例如：

**printf("hello, world**

**");**

编译器将会产生一条错误信息。

**（错误信息来自visual studio 2017，不同编译器给出的错误信息可能表述方式不一样，但是其内在含义是一样的）**

printf函数永远不会自动换行，这样我们可以多次调用该函数以分阶段得到一个长的输出行。上面给出的第一个程序也可以改写成下列形式：

|  |
| --- |
| #include <stdio.h>  int main()  {  printf("hello, ");  printf("world");  printf("\n");  } |

（这段程序与前面的程序的输出相同）

请注意，\n只代表一个字符。类似于\n的转义字符序列为表示无法输入的字符或不可见字符提供了一种通用的可扩充的机制。除此之外，C语言提供的转义字符序列还包括：\t 表示制表符；\b表示回退符；\"表示双引号；\\表示反斜杠符本身。2.3节将给出转义字符序列的完整列表。

**练习1-1** 在你自己的系统中运行“hello, world"程序。再有意去掉程序中的部分内容，看看会得到什么出错信息。

**练习1-2** 做个实验，当printf函数的参数字符串中包含\c（其中c是上面的转义字符序列中未曾列出的某一个字符）时，观察一下会出现什么情况。

解析1-2 可以编译通过，但运行结果可能与你想象的不同，因为\c不是一个转义字符。

运行结果：**hello, worldc**

## 1.2.变量与算术表达式

例子：使用公式打印右侧的华氏温度与摄氏温度对照表：

**“1.2\_1.c”**

|  |  |
| --- | --- |
| #include <stdio.h>  /\*  当 fahr = 0, 20, ..., 300时，  分别打印华氏温度与摄氏温度对照表  \*/  int main()  {  int fahr, celsius;  int lower, upper, step;  lower = 0;  upper = 300;  step = 20;  fahr = lower;  while (fahr <= upper) {  celsius = 5\*(fahr-32)/9;  printf("%d\t%d\n", fahr, celsius);  fahr = fahr + step;  }  } | C:\Windows\system32\cmd.exe |

这个程序仍然只包括一个名为main的函数，但是引入了一些新的概念，包括注释、声明、变量、算术表达式、循环以及格式化输出。

其中/\*注释\*/称为注释，此处，它简单地解释，该程序是做什么用的。包含在/\*与\*/之间的字符序列将被编译器忽略。注释可以自由地运用在程序中，使得程序更易于理解。（注意，这种注释不可以嵌套，例如/\*comment/\*comment\*/content\*/，是非法的）

在C语言中，所有变量都必须先声明后使用。声明通常放在函数起始处**（英文版的原著时1988年出版的，这个思想在现在已经过时了，现在通常什么时候使用变量，就什么时候声明，并且声明即初始化，以免造成未定义的结果）**，在任何可执行语句之前。声明用于说明变量的属性，它由一个类型名和一个变量表组成，例如：

**int fahr, celsius;**

其中，类型int表示其后所列变量为整型（integer），与之相对应的，double表示所列变量为浮点数（即可以带有小数部分的数）。int与double类型的取值范围**取决于具体的机器**（CPU的位数、操作系统的位数、编译器的位数等等）。

C:\Users\14121\AppData\Roaming\Tencent\Users\1412167533\TIM\WinTemp\RichOle\FOY8MH6J6R(9BB[KTG1(I{M.pngC:\Users\14121\AppData\Roaming\Tencent\Users\1412167533\TIM\WinTemp\RichOle\8L~%3`%_5VCUD50QFO0KPKX.pngchar 字符 ‘a’

short int 短整型 1234

C:\Users\14121\AppData\Roaming\Tencent\Users\1412167533\TIM\WinTemp\RichOle\F9(G_JJ4[N1UC{}[ZYAIFA7.pngint 整形 12345

C:\Users\14121\AppData\Roaming\Tencent\Users\1412167533\TIM\WinTemp\RichOle\F9(G_JJ4[N1UC{}[ZYAIFA7.pnglong int 长整型 1234567

C:\Users\14121\AppData\Roaming\Tencent\Users\1412167533\TIM\WinTemp\RichOle\S(U{042565LV90$5~SQACAD.pngdouble 双精度浮点型 3.1415926

**（在原书中，此处使用float单精度浮点型进行举例的，但是现在很少有人用float，基本都用double，原因是为了获得更精确的计算结果，而不考虑多占用的一点点空间，毕竟现在谁家的电脑内存不是16GB，32GB的）**

这些数据类型对象在计算机中占用的内存大小也**取决于具体的机器**，如果是32位系统，char占一个字节，其它数据类型相对于char的比例如上图所示。后续相应章节中会分别介绍这些数据类型。

在上面的温度转换程序中，最开始执行的计算是下列4个赋值语句：

**lower = 0;**

**upper = 300;**

**step = 20;**

**fahr = lower;**

它们为变量设置初值。各条语句均以分号结束。

温度转换表中的各行计算方式相同，因此可以用循环语句重复输出各行。这是while循环语句的用途：

**while (fahr <= upper) {**

**...**

**}**

while 循环语句的执行方式是这样的：首先测试圆括号中的条件，如果条件成立，则执行循环体（括在花括号中的3条语句），然后再重新测试圆括号中的条件，如果为真，则再次执行循环体；当圆括号中的条件测试结果为假(fahr>upper)时，循环结束，并继续执行跟在 while 循环语句之后的下一条语句。在本程序中，循环语句后没有其它语句，因此整个程序的执行终止。

while 语句的循环体可以是用花括号括起来的一条或多条语句（如上面的温度转换程序），也可以是不用花括号包括的单条语句，例如：

**while (i < j)**

**i = 2 \* i;**

在这两种情况下，我们总是把由while控制的语句缩进一个制表位（Tab键），这样就可以很容易地看出循环语句中包含哪些语句。这种缩进方式突出了程序的逻辑结构。正确的缩进以及保留适当空格的程序设计风格对程序的易读性非常重要。（**大家应该选择适合自己的一种风格，并养成一直使用这种风格的好习惯）**

在该程序中，绝大部分工作都是在循环体中完成的。循环体中的赋值语句

**celsius = 5\*(fahr-32) / 9;**

用于计算与指定华氏温度相对应的摄氏温度值，并将结果赋值给变量celsius。在该语句中,之所以把表达式写成先乘5然后再除以9而不是直接写成5 / 9,其原因是在C语言及许多其它语言中,整数除法操作将执行舍位，结果中的任何小数部分都会被舍弃。由于5和 9 都是整数，5 / 9相除后经截取所得的结果为0,因此这样求得的所有摄氏温度都将为0。

从该例子中也可以看出 printf 函数的一些功能。printf是一个通用输出格式化函数，第7章将对此做详细介绍。该函数的第一个参数是待打印的字符串，其中的每个百分号表示其它的参数（第二个、第三个、……参数）之一进行替换的位置，并指定打印格式。例如，**%d**指定一个整型参数，因此语句

**printf(" %d\t%d\n", fahr, celsius);**

用于打印两个整数fahr与celsius的值，并在两者之间留一个制表符的空间。

printf函数的第一个参数中的各个%分别对应于第二个、第三个、……参数,它们在数目和类型上都必须匹配，否则将出现错误的结果。

顺便指出，printf函数并不是C语言本身的一部分，C语言本身并没有定义输入／输出功能。printf仅仅是标准库函数中一个有用的函数而已，这些标准序函数在C语言程序中通常都可以使用。但是，ANSI标准定义了printf函数的行为，因此，对每个符合该标准的编译器和库来说，该函数的属性都是相同的。

**（第7章前不对输入输出做更多介绍，特别将格式化输入推后到第7章讲解）**

上述的温度转换程序存在两个问题。比较简单的问题是，由于输出的数不是右对齐的， 所以输出的结果不是很美观。这个问题比较容易解决：如果在printf 语句的第一个参数的%d中指明打印宽度，则打印的数字会在打印区域内右对齐。例如，可以用语句

**printf(" %3d %6d\n", fahr, celsius);**

打印 fahr与celsius的值，fahr的值占3个数字宽，celsius的值占6个数字宽，输出结果就如本节开始一样，但是，由于我们使用的是整形算术运算，得到的结果不太精确，例如，0℉应该对应-17.8℃，但是在转换为整形是发生了截断（不是四舍五入），为了获得更精确的结果，应该用浮点数运算代替上面的整形算术运算。

第二个版本的程序如下：

**“1.2\_2.c”**

|  |  |
| --- | --- |
| #include <stdio.h>  /\*  当 fahr = 0, 20, ..., 300时，  分别打印华氏温度与摄氏温度对照表  \*/  int main()  {  double fahr, celsius;  int lower, upper, step;  lower = 0;  upper = 300;  step = 20;  fahr = lower;  while (fahr <= upper) {  celsius = (5.0/9.0)\*(fahr-32.0);  printf("%3.0f\t%6.1f\n", fahr, celsius);  fahr = fahr + step;  }  } | C:\Windows\system32\cmd.exe |

这个程序与前一个程序基本相同，不同的是，它把fahr与celsius声明为double类型，转换公式的表述方式也更自然一些。在前一个程序中，之所以不能使用5 / 9的形式，是因为按整型除法的计算规则。但是，常数中的小数点表明该常数是一个浮点数，因此5.0/9.0是两个浮点数相除，结果将不被截断。

如果某个算术运算符的所有操作数均为整型，则执行整型运算。但是，如果某个算术运算符有一个浮点型操作数和一个整型操作数，则在开始运算之前整型操作数将会被转换为浮点型。

在这里需要注意，赋值语句

**fahr = lower;**

与条件测试语句

**while (fahr <= upper)**

也都是按照这种方式执行的，即在运算之前先把 int 类型的操作数转换为 float 类型的操作数。

printf中的转换说明**%3.0f**表明待打印的浮点数至少占3个字符宽，且不带小数点和小数部分；**%6.1f**表明另一个待打印的数至少占6个字符宽，且小数点后面有1位数字。

格式说明可以省略宽度与精度，例如，%6f表示待打印的浮点数至少有6个字符宽；%.2f指定待打印的浮点数的小数点后有两位小数，但宽度没有限制；%f则仅仅要求按照浮点数打印该数。

**%d** 按照十进制整型数打印

**%6d** 按照十进制整型数打印，至少6个字符宽

**%f** 按照浮点数打印

**%6f**  按照浮点数打印，至少6个字符宽

**%.2f** 按照浮点数打印，小数点后有两位小数

**%6.2f** 按照浮点数打印，至少6个字符宽，小数点后有两位小数

此外，printf函数还支持下列格式说明：%o表示八进制数；%x表示十六进制数；%c表示字符；%s表示字符串；%%表示百分号本身。

**练习1-3** 修改温度转换程序，使之能在转换表的顶部打印一个标题。

**练习1-4** 编写一个程序打印摄氏温度转换为相应华氏温度的转换表。

解析1-4 见**chapter01\_ex04.c**

## 1.3.for语句

对于某个特定任务我们可以采用多种方法来编写程序。下面这段代码也可以实现前面的温度转换程序的功能：

**“1.3.c”**

|  |
| --- |
| #include <stdio.h>  /\* 打印华氏温度-摄氏温度对照表 \*/  int main()  {  for (int fahr = 0; fahr <= 300; fahr += 20)  printf("%3d %6.1f\n", fahr, (5.0/9.0)\*(fahr-32));  return 0;  } |

**（在原书中，fahr的声明是放在for循环外的，但是C语言新的标准允许我们在初始化部分声明变量，现在更多使用的也是这样一种风格）**

这个程序与上节中介绍的程序执行结果相同，但程序本身却有所不同。最主要的改进在于它去掉了大部分变量，而只使用了一个int类型的变量fahr。在新引入的for语句中，温度的下限、上限和步长都是常量，而计算摄氏温度的表达式现在变成了printf函数的第三个参数，它不再是一个单独的赋值语句。

**for语句**是一种循环语句，它是对while语句的推广。如果将for语句与前面介绍的while语句比较，就会发现for语句的操作更直观一些。圆括号中共包含3个部分，各部分之间用分号隔开。第一部分

**int fahr = 0**

是初始化部分，仅在进入循环前执行一次。第二部分

**fahr <= 300**

是控制循环的测试或条件部分。循环控制将对该条件求值，如果结果值为true，则执行循环体。此后将执行第三部分

**fahr+=20**

以将循环变量 fahr 增加一个步长，并再次对条件求值。如果计算得到的条件值为false，循环将终止执行。与while语句一样，for循环语句的循环体可以只有一条语句，也可以是用花括号括起来的一组语句。初始化部分（第一部分）、条件部分（第二部分）与增加步长部分（第三部分）都可以是任何表达式。

在实际编程过程中，应根据实际情况选择while与for中的一种循环。

**练习1-5** 修改温度转换程序，要求以逆序（即按照从300度到0度的顺序）打印温度转换表。

解析1-5 见**chapter01\_ex05.c**

## 1.4.符号常量

在结束讨论温度转换程序前，我们再来看一下符号常量。在程序中使用300、20等类似的“魔法数字"并不是一个好习惯，它们几乎无法向以后阅读该程序的人提供什么信息，而且使程序的修改变得更加困难。处理这种幻数的一种方法是赋予它们有意义的名字。#define 指令可以把符号名定义为一个特定的字符串：

**#define 名字 替换文本**

在该定义之后，程序中出现的所有在#define中定义的名字（既没有用引号引起来，也不是其它名字的一部分）都将用相应的替换文本替换。其中，名字与普通变量名的形式相同：它们都是以字母打头的字母和数字序列；替换文本可以是任何字符序列，而不仅限于数字。

**“1.4.c”**

|  |
| --- |
| #include <stdio.h>  #define LOWER 0  #define UPPER 300  #define STEP 20  /\* 打印华氏温度-摄氏温度对照表 \*/  int main()  {  for (int fahr = LOWER; fahr <= UPPER; fahr += STEP)  printf("%3d %6.1f\n", fahr, (5.0/9.0)\*(fahr-32));  } |

**(现在我就可以任意修改起始值，中止值和步长而不用在程序中到处寻找魔法数字)**

符号常量名通常用大写字母拼写，这样可以很容易与用小写字母拼写的变量名相区别。**注意，#define 指令行的末尾没有分号。**

## 1.5.字符输入/输出