第二章　数据对象与计算

计算机程序处理数据，写程序就是描述数据的处理过程，其中必然涉及数据的描述和计算问题。例如，在 C 程序里可以写出下面片段，这是一个表示了某种计算过程的“表达式”，其中包含了一些“数据”，如整数和实数等::

```
-(3.24 * 5 + sin(2,3)) / 4 * 6.24
```

要理解这种表达式，写程序时知道如何写出所需表达式，就必须知道 C 语言对各种数据的写法（术语是*数据的描述*）有什么规定：在表达式里可以写什么？它们表示什么意思？写出的表达式表示了什么计算过程？有关计算的结果是什么？本章首先解决这些问题。

本章将首先讨论 C 语言中各种基本数据的描述，然后介绍如何从基本数据元素出发描述计算，如何写好能求出所需结果的表达式。在这一章里，读者将初步接触到计算机领域的许多重要概念，看到它们在简单程序中的地位和作用。

2.1　基本字符、名字表示、标识符和关键字

一个 C 程序就是 C 语言基本字符的一个符合规定形式的序列。C 语言基本字符包括：

1. 数字字符，0，1，2，3，4，5，6，7，8，9；
2. 大小写拉丁字母：a～z，A～Z；
3. 其他一些可打印（可以显示）的字符（如各种标点符号、运算符号、括号等），包括：

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

现在不必死记这些，随着学习进展，读者将很容易记住这些字符的意义和作用。

4. 还有一些特殊字符，如空格符、换行符、制表符等。

空格符、换行符、制表符等统称为*空白字符*。空白字符在程序中主要用于分隔其他成分。

按规定，C 程序中大部分地方增加空白字符都不影响程序意义。因此人们写程序中常利用这种性质，通过加入一些空白字符，把程序排成适当格式，以增强程序的可读性。例如，在适当地方换行，在适当地方加空格或制表符。这样能使程序的表现形式更好反映其结构和所实现的计算过程。举例说，第 1 章的简单 C 程序也可以写成下面样子：

```
#include <stdio.h>
int main(){printf("Good morning!\n");return 0;}
```

这明显不如前面的写法清晰。对更大的程序情况则会更糟糕。本书后面讨论中还会提出对各种程序成分的较好写法，书中程序示例也反映了这方面的情况。

构成 C 程序的基本成分包括各种名字（如上面出现的 `main`、`printf` 等等），各种数值表示（如 125、3.14 等等），各种运算符和其他符号。

名字（标识符）的构成

程序中有许多需要命名的对象。例如，程序中常常需要定义一些东西，以便在各处使用。为了在定义和使用之间建立联系，表示不同位置用的是同一个对象，基本的方式就是为程序对象命名，通过名字建立起定义与使用间、同一对象的不同使用间的联系。为了这种需要，C 语言规定了名字的书写形式。程序中的名字称为*标识符*。

一个标识符是字母和数字字符的一个连续序列，其中不能有空白字符，而且要求第一个字符必须是字母。为了方便起见，C 语言特别规定将下划线字符“_”也当作字母看待。这就是说，下划线可以出现在标识符中的任何地方，特别是可以作为标识符的第一个字符。下面是一些标识符的例子：

```
abcd    Beijing   C_Programming   _f2048   sin    a3b06
xt386ex    A_great_machine    Small_talk_80    FORTRAN_90
```

以下划线开始的标识符保留给系统使用，在我们编写普通程序时不要使用这种标识符，以免与系统内部的名字冲突造成程序问题。

如果一个字符序列中出现了非字母、非数字、也非下划线的字符，那么它就不是一个标识符了（但有可能其中一部分是个标识符，例如 x3+5+y，其中 x3 和 y 都是标识符，中间的+5+不属于这两个标识符）。下面是一些非标识符的字符序列：

```
+=    3set   a[32]   $$$$   sin(2+5)    ::ab4==
```

C 语言还规定，标识符中同一字母的大写形式和小写形式将看作不同字符，这样，a 和 A 不同，name、Name、NAME、naMe 和 nMAE 是互不相同的标识符。

关键字

C 语言的合法标识符中有一个特殊的小集合，其中的标识符称为关键字。作为关键字的标识符在程序里具有语言预先定义好的特殊意义，因此不能用于其他目的，不能作为普通的名字使用。C 语言关键字共 32 个，列在这里：

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

现在不准备对它们做更多解释。随着书中讨论的进展，读者会一个一个地接触并记住它们。目前只需要了解关键字这一概念。

除了不能使用关键字之外，我们写程序时几乎可以用任何标识符为自己所定义的东西命名，所用的名字可以自由选择。通过长期程序设计实践，人们认识到命名问题并不是一件无关紧要的事情。合理选择程序对象的名字能为人们写程序、读程序提供有益的提示，因此人们倡导采用能说明程序对象内在含义的名字（标识符）。

注读者注意，命名问题并不是 C 语言中特殊的东西，每种程序语言都必须规定程序中名字的形式，在计算机领域中到处都用到名字。例如，计算机里的文件和目录，各种应用程序和系统，图形界面上的图标和按钮，甚至计算机网络中的每台计算机，都需要命名。采用适当命名形式的原则在计算机领域中具有广泛适用性。

2.2 数据与类型

数据是程序处理的对象。C 语言把程序能处理的基本数据对象分成一些集合。属于同一集合的数据对象具有同样性质：采用统一的书写形式，在具体实现中采用同样的编码方式（按同样规则对应到内部二进制编码，采用同样二进制编码位数），对它们能做同样操作等等。语言中具有这样性质的一个数据集合称为一个类型。

从关于计算机基础知识可知，计算机硬件处理的数据也分成一些类型，通常包括字符、整数、浮点数等，CPU 为不同数据类型提供了不同的操作指令。例如，对整数有一套加减乘除指令，对浮点数有另一套加减乘除指令等。程序语言中把数据分成类型与此有密切关系。但类型的意义不仅于此，实际上，类型是计算机科学的核心概念之一。在学习程序设计和程序设计语言的过程中将不断与类型打交道。请读者特别注意这一概念。

C 语言的基本类型包括字符类型、整数类型、实数类型等。请读者特别注意：（1）程序中书写的、执行中处理的每个基本数据都属于某个确定的基本数据类型；（2）类型确定了属于它的数据对象的许多性质，特别是确定了数据的表示范围。在具体 C 语言系统里，基本类型都有确定表示（编码）方式，这就确定了可能表示的数据范围。例如，一个整数类型中的所有整数只是数学中整数的一个子集，其中只包含有限个整数值，存在该类型能表示的最小和最大整数。其他整数在这个类型里没有容身之地，无法在这个类型中表示。

下面会看到这些基本情况的影响。

2.3 基本类型与数据表示

C 语言提供了一组基本基本类型，并规定了“类型名”。基本类型的名字由一个或几个标识符（关键字）构成，其形式与前面讲的“名字”有所不同。本节将介绍几个最常用的类型。不在这里介绍所有基本类型，是希望能尽快进入讨论的主题——程序与程序设计。这几个基本类型对前几章也足够了。后面章节将对所有基本类型做全面的介绍。

首先应提出**文字量**的概念。文字量就是程序里直接写出的数据。例如，程序里直接写出的整数类型的数据就称为“整型的文字量”。为简单起见，也常把整型文字量简称为“整数”，其他情况也采用类似称呼方式，后面常用这种简称，只在特别需要时才用更严格的说法。C 语言规定了各种基本类型的文字量的书写形式，这也是本节的主要内容。

2.3.1 整数类型和整数的表示

C 语言提供了多个整数类型以适应不同需要。不同整数类型间的差异在于它们可能具有不同的二进制编码位数，因此表示范围可能不同。程序中用的最多是一般整数类型（今后简称为“整数类型”或“整型”）和长整数类型（简称“长整型”）。整数类型的类型名是 `int`；长整型的类型名为 `long int`，可简写为 `long`。`int` 和 `long` 都是关键字。

整数表示

整数（`int` 类型的文字量）有几种书写形式，程序中的整数一般采用十进制写法。用十进制方式写出的一个整数就是普通数字字符组成的一个连续序列，其中不能有空格、换行或其他字符。C 语言规定十进制表示的数字序列的第一个字符不是 0，除非要写的整数本身就是 0。下面是一些整数的例子：

```
123     304     25278     1     0     906
```

由于长整数是另一个不同类型，C 语言为长整数规定了一种专门写法，其特殊之处是在表示数值的数字序列最后附一个字母 `l` 或 `L` 作后缀。由于小写字母 `l` 容易与数字 `1` 混淆，建议读者总采用大写的 `L`。下面是一些长整数的例子：

```
123L     304l     25278L     1l     0L     906L
```

表示范围

C 语言没有规定各种整数类型的表示范围，也就是说，没有规定各种整数的二进制编码长度。对于 `int` 和 `long`，只规定了 `long` 类型的表示范围不小于 `int`，但也允许它们表示范围相同。具体 C 语言系统则会对整型和长整型规定明确表示方式和表示范围。例如，早期微型机的一些 C 系统采用 16 位二进制表示的整数（一个 `int` 占 2 个字节）和 32 位表示的长整数（一个 `long` 占 4 个字节）。这样，整型的表示范围就是-32768 到 32767，即 $-2^{15}\cdots 2^{15}-1$。长整型表示范围是 $-2^{31}\cdots 2^{31}-1$。在许多新的微机 C 语言系统里，整数（`int`）和长整数（`long int`）都采用 32 位的二进制数表示。

C 语言允许在整数的前面写正负号，加上负号的整数就是表示负整数。

整数的八进制书写法和十六进制书写法

整数与长整数都可以采用八进制或十六进制的形式书写。

用八进制形式写出的整数（int 类型的文字量）是由数字 0 开始的连续数字序列，在序列中只允许 0～7 这八个数字。下面是用八进制写法写出的一些整数和长整数：

```
0236      0527      06254      0531      0765432L
```

整数的十六进制形式是由 0x 或 0X 开头的数字序列。由于数字只有 10 个，而在十六进制写法中需要 16 个数字，C 语言采用计算机领域通行的方式，用字母 a～f 或 A～F 表示其余的 6 个十六进制数字，其对应关系是：

字母：	a,A	b,B	c,C	d,D	E,E	f,F
表示的数字：	10	11	12	13	14	15

下面是用十六进制形式写出的一些整数和长整数：

```
0x2073      0xA3B5      0XABCD      0XFFFF      0XF0F00000L
```

请注意：八进制、十进制和十六进制只是整数的不同书写形式，提供多种写法是为了编程方便，使人可以根据需要选择适用的书写方式。无论采用八进制写法还是十六进制写法，写出的仍是某个整数类型的数，并不是新的类型。用八进制、十六进制形式写长整数时，同样需要用后缀 l 或者 L。

日常生活中人们习惯于用十进制的形式书写整数。C 语言提供八进制和十六进制的整数书写方式，也是为了写程序的需要。在写复杂程序时，有些情况下用八进制和十六进制更方便些，后面会看到这方面例子。

2.3.2 实数类型和实数的表示

实数类型

C 语言提供了三个表示实数的类型：单精度浮点数类型，简称*浮点类型*，类型名为 float；双精度浮点数类型，简称*双精度类型*，类型名为 double；长双精度类型，类型名为 long double。这些类型的文字量也分别称作“浮点数”、“双精度数”和“长双精度数”。所有整数类型和实数类型统称为*算术类型*。

实数的计算机内部表示由具体系统规定，其中不少系统采用通行的国际标准（IEEE 标准，IEEE 是电子电器工程师协会，是一个著名的国际性技术组织）：

1. 浮点类型的数用 4 个字节 32 位二进制表示。这样表示的数大约有 7 位十进制有效数字，数值的表示范围约为 $\pm(3.4\times10^{-38}\cdots3.4\times10^{38})$；
2. 双精度类型的数用 8 个字节 64 位二进制表示，双精度数大约有 16 位十进制有效数字，数值的表示范围约为 $\pm(1.7\times10^{-308}\cdots1.7\times10^{308})$；
3. 长双精度类型的数用 10 个字节 80 位二进制表示，大约有 19 位十进制有效数字，其数值的表示范围约为 $\pm(1.2\times10^{-4932}\cdots1.2\times10^{4932})$。

显然，每个实数类型能表示的数也只是数学中实数的一个小子集合，不仅表示范围有限，表

> 要用某计算机上的某个 C 语言系统编程，要做的一件事就是查清该系统里各种整数类型的表示范围。有关情况可以从系统使用手册中查到，或查看介绍该系统的书籍，或查看系统的联机帮助。此外，还可以查看这个 C 系统中名字为 limit.h 的文件。这是每个 C 语言系统都有的一个标准文件，其中列出了各种情况的具体规定。
>
> 对于浮点数也有类似情况。例如，在一些 C 语言系统里，long double 采用与 double 同样的表示方式。有关具体 C 语言系统中浮点数表示的情况，也应查阅系统手册，还可以查阅名为 float.h 的标准文件。

示的精度（数的有效数字位数）也有限，请读者注意这些情况。

实数的写法

C语言中最基本的实数类型是双精度类型。双精度数的书写形式中的基本部分是一个数字序列，在序列中或者包含了一个表示小数点的圆点“.”（可以是第一个或最后一个字符），或者在表示数值的数字序列后面有一个指数部分。指数部分是以 e 或 E 开头的另一（可以包括正负号的）数字序列，指数以10为底，这种形式称为科学记数法。也可以既有小数点，又有指数部分。下面是一些双精度数的例子：

```
3.2    3.    2E-3    2.45e17    0.038    105.4E-10    304.24E8
```

下面是其中一些双精度类型类型的文字量（双精度数）与它们所表示的实数的对照表：

双精度数	所表示的实数值
2E-3	0.002
105.4E-10	0.00000001054
2.45e17	24,5000,0000,0000,0000.0
304.24E8	304,2400,0000.0

浮点数（float）类型数的写法在与双精度数类似，只是在数最后应附后缀字符 f 或者 F。表示长双精度数的后缀是 l 和 L。下面是一些浮点数类型和长双精度类型数的例子：

```
13.2F     1.7853E-2F     24.68700f     .32F     0.337f
12.869L     3.417E34L     .05L     5.E88L     1.L
```

负实数同样通过在数前加负号表示。

2.3.3 字符类型和字符的表示

字符类型数据主要用于程序的输入输出。此外，文字处理也是计算机的一个重要应用领域，该应用领域的应用程序必须能使用和处理字符形式的数据。由于大部分程序都需要与人打交道，需要接收人的输入信息（例如人给程序发的命令，或者提供的数据），还需要给人输出信息，因此字符类型的数据在程序中的使用很广泛。

最常用的字符类型的类型名是 char。字符类型的数据值包括本计算机所用编码字符集中的所有字符。目前微机和工作站常用ASCII字符集，其中的字符包括所有大小写英文字母、数字、各种标点符号字符，还有一些控制字符，一共128个。扩展的ASCII字符集包括256个字符。字符集的所有字符都是字符类型的值。在程序执行时，其中的字符就用对应的编码表示，一个字符通常占用一个字节。

字符文字量的书写形式是用单引号括起的单个字符，例如'1'、'a'、'D'等。一些特殊字符无法用这样写出，例如换行字符等。C语言为它们规定了特殊写法。下面是几个最常用的特殊字符的写法：

```
换行字符      '\n'          双引号字符      '\"'
单引号字符    '\''          反斜线字符      '\\'
```

这里的写法都是在单引号里面先写一个反斜线字符（\），后面再写一个字符。在这种写法中，反斜线字符的作用就是表明它后面的字符不取原来意义。这样连续的两个字符（或更多几个字符）称为一个*换意序列*，用于表示无法写出的字符。反斜线字符在其中起着特殊作用，它也被称为*换意字符*。还有些特殊字符也有特别规定的写法，附录B列出了所有特殊字符的写法，还说明了采用八进制和十六进制编码形式写字符的方式。

这里需要强调两点。

1. 字符数据与标识符不同。例如 x 和'x'是两种完全不同的东西，后者表示一个数据项，是程序处理的对象；前者则是程序描述中所用的一个名字，它可能代表程序里的某个东

西。显然它们不在同一个层次上。

2. 数字字符和数不同。例如 1 和'1'，前者是一个整型文字量，是一个 int 类型的数据对象，其存储要占据 int 所规定的那么多单元。在常见的微机 C 系统里，它可能占了 2 个或 4 个字节，其中存着整数 1 的二进制编码。而'1'是个 char 类型的数据，其存储通常占一个字节，其中存着字符'1'的编码（在 ASCII 码中'1'的编码是 49）。

C 语言的一个特殊规定是把字符看作一种特别短的整数，允许程序中直接用字符的值参与算术运算，这方面的情况将在后面讨论。

字符串

字符串是 C 程序里可以直接写出来的另一类数据，其形式是用双引号括起的一系列字符。下面是几个字符串的例子：

```
"CHINA"     "Beijing"     "Daxue"     "Welcome\n"
```

在字符串里的特殊字符也用换意序列的形式书写，例如上面第四个字符串的最后是一个换意序列，表示了一个换行字符。

程序中的字符串主要用于输入输出，在第一章的简单 C 程序里有下面一行：

```
printf("Good morning!\n");
```

圆括号里就是一个字符串。

C 语言规定程序不能在字符串中间换行，否则编译会出错。

2.3.4 数据的外部表示、内部表示与转换

数据的外部表示指的是人在写出的 C 语言源程序中写数据所用的形式、或者人给正在运行中的程序提供数据时所用的形式、或者人从程序中得到的输出所具有的形式。内部表示指程序运行中，各种数据在计算机内部的二进制编码形式，也就是计算机内部存储和处理数据时所用的形式。显然这是两种不同形式。例如：

	外部形式	内部形式
整数	十进制（或其他进制）的数字字符序列	整数的二进制编码
字符	字符本身	相应的二进制编码（常用的 ASCII 编码）

举例来说，如果我们在源程序里写整数 123，而在程序运行时实际使用的则是存在内存某（几）个单元里的二进制形式 1111011。如果在程序里写字符'a'，程序运行中将会把字符 a 的编码的二进制形式保存在计算机里的某个地方。

由于同样数据在外部和内部具有不同表示形式，在源程序编译时，或者在程序运行中执行输入输出时，都需要做两种不同形式间的转换。编译时的转换由编译程序完成，C 语言对程序中各种数据的书写形式都做出了严格规定（参见前面讨论的文字量），只要写程序的人按规定形式写数据，编译过程中就能将它们正确转换为内部形式，供程序内部使用。在 C 程序的输入输出过程中的数据形式转换，则需要在程序里明确写出来。

现在先讨论输出中的数据转换，介绍如何描述输出动作。由于每个程序都需要输出，写程序时要解决的一个问题就是写输出操作，指定在输出过程中数据的转换方式。

C 语言定义了标准库，每个具体的 C 系统都提供了标准库功能。标准库提供了许多常用函数，供在写程序的人使用。标准函数里有一个名字是 printf 的常用输出函数（函数名也是标识符），在第一章的简单程序里就用到它。函数 printf 的功能是把一些信息送到标准输出，一般是送到显示器。printf 的使用形式是：

```
printf(格式描述串, 其他参数);
```

具有这种形式的程序片段被称为一个语句。每个语句最后有一个分号，这是语句的一部分，不能缺少。按上面形式写出的语句可看作是一个输出语句，语句中使用了标准库的输出函数

printf，函数名后面括号里面的描述称为函数的实际参数。

每个函数都明确规定了自己的使用方式，只有符合规定的使用才能得到正确结果。如果不遵循函数的使用规定，写出的程序或者是编译时不能通过，编译程序指出程序有错误，或者编译结果产生的程序不能正确执行。

要使用函数 printf，除了写出函数名外，还要写一对圆括号，其中给出函数所要求的参数。格式描述串是一个字符串（双引号括起来的字符序列），其他参数应与格式描述相匹配（下面介绍），可以有一个或几个，也可以没有。不同参数间用逗号分隔。

如果格式描述串中没有特殊字符%，那么该输出语句里就不该有其他参数，也不需要表示分隔的逗号，这是使用 printf 的最简单形式。这样一段函数名、括号及其中的实际参数表示了对函数 printf 的一次使用。这种形式的输出语句的作用就是输出格式描述串本身。第一章例子写了语句“printf("Good morning\n");”，它的执行将输出：

```
Good morning
```

作为另一个例子，程序在执行下面语句时：

```
printf("Welcome\nto\nBeijing!\n");
```

将输出三行字符：

```
Welcome
to
Beijing!
```

请注意，上面格式描述串里包含了三个换行字符，它们也都被输出了。

我们常说使用 printf 的语句实施的是对函数 printf 的一次调用。下面简单程序里三次调用函数 printf，它产生的输出与上面一个例子相同：

```
#include <stdio.h>

int main() {
    printf("Welcome\n");
    printf("to\n");
    printf("Beijing!\n");
    return 0;
}
```

在格式描述串中起特殊作用的是以百分号字符（%）开始的，若干字符构成的序列。这种序列称为转换描述，其作用就是指明与之相对应的其他参数的转换和输出方式。函数 printf 的一般调用形式是：

printf(格式描述串, 其他参数$_1$, ..., 其他参数$_k$);

它可以有任意多个其他参数，但要求格式描述串中转换描述的个数与其他参数的个数互相匹配（简单情况是个数相同，更详细的情况见第八章的有关讨论）。这样，每个转换描述对应于一个其他参数，说明该参数的输出形式（转换方式）。下表列出了程序中最常用的几个转换描述，它们所指定的转换，以及与其对应的其他参数应该具有的类型：

转换描述	实现的转换	对应参数的类型
%d	将参数按整数形式转换输出	对应参数应是 int 类型
%ld	将参数按长整数形式转换输出	对应参数应是 long 类型
%f	将参数按带小数点数形式转换输出	对应参数应是 double 类型
%Lf	将参数按带小数点数形式转换输出	对应参数应是 long double 类型
%c	输出一个字符	对应参数应表示字符的编码
%s	输出一个字符串	对应参数应该是一个字符串

一个格式描述串通常由一些普通字符和几个转换描述按某种所需顺序构成，所有普通字符形成了实际输出的框架，这些字符都将被正常输出，只是在那些原来写着转换描述的地方，实际输出时将用其他参数经过转换得到的结果替代，形成最终输出。前面例子里的

“printf("Welcome\n");”不含转换描述，因此将输出 Welcome 和一个换行。

语句：

```
printf("%d + %d = %d\n", 2, 3, 5);
```

的格式描述中包含转换描述，执行它时将形成一行输出：

```
2 + 3 = 5
```

这一输出的框架由"%d + %d = %d\n"提供，它说明输出的形式：首先输出对后面的第一个参数按整数的形式转换而得到的东西（这里就是 2），然后是一个空格、一个加号、另一个空格，然后是第二个参数按整数形式转换而得到的东西、……。这样，当程序执行到这个语句时，就产生了上述输出。

下面是另一个例子：

```
printf("len:%f, width:%f, area:%f\n", 2.2, 3.5, 7.7);
```

这个语句的执行将输出：

```
len:2.200000, width:3.500000, area:7.700000
```

注意，参数和转换描述间按位置一一对应，分别转换。这里需要特别强调在格式描述串中的转换描述和它所对应的参数在类型上的一致性。如果这两者在类型上不一致，就无法保证得到正确的输出结果，甚至还可能引起更严重的程序运行错误。另一个问题是要保证格式描述串里转换描述个数和其他参数个数之间的一致性，否则也是一种错误。这种错误引起的后果是无法预料的（这一说法意味着可能产生很严重的后果）。

初学者常常遇到这种情况：写好了一个程序，但程序的输出结果却不正确。仔细检查程序中的计算过程，怎么也找不出错误。经过许多努力，最后发现是有关输出的语句中格式转换描述与对应参数不匹配。这种情况值得读者特别注意。

如果在程序里使用 printf，程序在最前面就应当包括行：

```
#include <stdio.h>
```

这个行的作用是告诉编译程序，在本程序中使用了 C 标准库里的输入输出函数，要求编译程序正确处理输出函数的使用。这种行的细节意义将在后面章节里讨论。

> 上面讨论中提到标准输出，请读者回忆一下计算机基础知识中关于操作系统标准输入和标准输出的概念，在 DOS、UNIX 等各种操作系统中都有这些概念，送到标准输出的信息通常将被自动送到计算机的显示屏幕，使我们可以在屏幕上看到这些输出。在 Windows 等图形用户界面的系统里，程序送到标准输出的信息可能被显示在一个窗口中。此外，许多操作系统都允许将送到标准输出的信息重新进行定向（DOS、UNIX 系统都支持用户这样做）。因此可以通过这种机制把送到标准输出的信息转送到任何其他地方，例如送到某个文件里，直接送到打印机等等。

2.4 运算符、表达式与计算

了解了基本数据的描述后，现在就可以讨论计算过程的描述问题了。在 C 语言程序里，描述计算的最基本结构是*表达式*，表达式由被计算的对象（例如文字量，后面将会介绍更多的基本计算对象）和表示运算的特殊符号按照一定的规则构造而成。

描述数据运算的特殊符号称为*运算符*，C 语言里的运算符大都由一个或两个特殊字符表示（有个别例外）。本节将讨论各种算术运算符的形式和意义，介绍如何用它们构造算术表达式。讨论中还要介绍一些与运算符、表达式和表达式所描述的计算有关的重要问题，理解这些问题，对于正确描述所需计算的表达式都非常重要。

2.4.1 算术运算符

算术运算符一共有 5 个，它们是：

运算符	使用形式	意义
+	一元和二元运算符	一元表示正号，二元表示加法运算
-	一元和二元运算符	一元表示负号，二元表示减法运算
*	二元运算符	乘法运算
/	二元运算符	除法运算
%	二元运算符	取模运算（求余数）

一元运算符就是只有一个运算对象的运算符，运算对象写在运算符后面。二元运算符有两个运算对象，分别写在运算符两边。在上面的算术运算符中，+ 和 - 同时作为一元和二元运算符使用，其他都是二元运算符。对表达式里的某个 + 或 - 运算符，根据其出现位置的上下文总可以确定它是作为哪种运算符使用的。取模就是求余数，例如 17 对 5 求余数的结果是 2。取模运算符只能用于各种整数类型，其余运算符可用于所有算术类型。

2.4.2 算术表达式

算术表达式由计算对象（例如数值的文字量等）、算术运算符及圆括号构成，其基本形式与数学上的算术表达式类似。下面是两个表达式的例子：

```
-(28 + 32) + (16 * 7 - 4)
25 * (3 - 6) + 234
```

对属于同一类型（int、long、float、double 或 long double）的一个或两个数据使用算术运算符，计算结果仍然是该类型的值。例如：3 + 5 算出整数类型的 8，3L + 5L 计算出长整数值 8，而 3.2 + 2.88 计算出一个双精度值。在写表达式时，为清晰起见，可以在运算对象和运算符之间适当地加一个空格。这种空格并不影响程序的意义。

例：写程序计算半径为 6.5 厘米的圆球的体积。

根据前面有关简单 C 程序的说明，表达式的写法，以及 printf 的使用形式，我们很容易写出下面的简单程序：

```
#include <stdio.h>

int main() {
    printf("V = %fcm^3\n",
           (3.1416 * 6.5 * 6.5 * 6.5) * 4.0 / 3.0 );
    return 0;
}
```

这个程序经过加工之后，运行时将输出计算结果：

```
V = 1150.349200cm^3
```

从这个例子里也可以可以看到格式描述串的作用，它给出了输出的框架。程序最前面写了 #include<stdio.h>，表示程序里使用了标准输入输出函数 printf。

这一程序示例也表示了一种最简单的计算程序的模式，将其中的表达式换成其他表达式，就可以完成各种算术表达式的计算了。还可以根据需要写格式描述串，描述输出的形式。写这种程序时，请特别注意表达式计算出的结果的类型（例如，是 int 还是 double），相应格式串中的转换描述必须与之对应，否则就是程序错误。例如，下面程序里就有一个错误，我们无法保证这一程序能给出什么结果，甚至不知道它是否会导致系统崩溃。请设法找出这个程序里的错误，但不要去试验这个错误程序：

```
#include <stdio.h>

int main() {
    printf("Factorial of %d is %f\n", 7, 1*2*3*4*5*6*7);
    return 0;
}
```

2.4.3 表达式的求值

表达式的计算过程又称*表达式求值*。表达式的意义就是它所求出的值。一个表达式可能很复杂，其中可能有多个运算符，这时该表达式将确定什么样的计算过程？或者说，其中的运算符将按照怎样的顺序计算？程序语言必须对此做出明确规定。C 语言对表达式求值的规定包括几个方面：运算符优先级的规定，运算符的结合方式的规定，运算对象求值顺序的规定，以及括号的意义。下面分别介绍这几个问题。

优先级

小学生学算术就知道先乘除后加减，也就是说，乘除运算符具有更高*优先级*，在计算中应先做。C 语言里有很多运算符，它为每个运算符规定了一个优先级。当不同的运算符在表达式里相邻出现时，具有较高优先级的运算符应比具有较低优先级的运算符先行计算。算术运算符被放在三个不同的优先级上：

运算符：	一元 + 和 –	* / %	二元 + 和 –
优先级：	高	中	低

这样，在下面表达式里加法将会最后做：

```
5 / 3 + 4 * 6 / 2
```

这与数学中的规定相符。

结合方式

仅靠优先级，上面例子里子表达式 `4 * 6 / 2` 的计算方式仍没有确定，因为其中相邻的乘除运算符具有相同优先级。*结合方式*解决这类问题，它确定具有相同优先级的运算符相邻出现时表达式的计算方式。C 语言规定一元算术运算符自右向左结合；二元算术运算符自左向右结合，优先级相同时左边的运算符先计算。这样，在上面的例子里就将先计算 `4 * 6`，而后再用它们的计算结果去除另一个运算对象 `2`。此外，`-+-8` 还是计算出 8。这一规定也符合数学习惯。

括号

括号供人明确地描述表达式中的计算顺序。如果用括号括起表达式中的某个部分，括号里面的表达式将先行计算，得到的结果再参与括号外面的其他计算。例如，下面表达式里各步骤的计算顺序都已经完全确定了：

```
-(((2 + 6) * 4) / (3 + 5))
```

括号使人用于控制计算过程的一种手段。如果直接写出的表达式所产生的计算顺序不符合需要，就可以通过加括号的方式，强制程序去执行所需要的特定计算顺序。

运算对象的求值顺序

虽然有了上述规定，计算过程中仍有些事情没有完全确定。请看下面表达式：

```
(5 + 8) * (6 + 4)
```

显然，子表达式`(5 + 8)`和`(6 + 4)`的计算应先完成，然后才能去做乘法。但两个运算对象`(5 + 8)`和`(6 + 4)`中哪个先算呢？这也就是问乘法运算符的两个运算对象（更一般性的问题是各种二元运算符的两个运算对象）的计算顺序问题。C 语言对算术运算符的这一问题没有明确规定。这样，有的 C 系统可能先算左边对象，另一 C 系统可能先算右边对象，甚至可以有这样的 C 系统，其中有时先算左边的对象，有时先算右边的对象。

应当这样理解 C 语言的规定：在写程序时，不应写那种依赖于特殊计算顺序的表达式，因为我们无法保证它在各种系统里都能算出所希望的结果。显然，在上面例子中，运算对象的求值顺序不会影响结果。但是读者不久就会看到一些对求值顺序敏感的表达式。根据上面的讨论，程序里不应该写对求值顺序敏感的表达式。

运算对象的求值顺序问题是程序语言中的特殊问题，在数学里不存在这种问题。从这里读者也应该看到计算机与数学的不同。

了解了上面这些规定，在需要描述计算过程时，我们就可以写出正确的表达式了。还应提出一点：如果表达式很复杂，有时加入一些括号对读表达式的人有利，即使这些括号并不必要。此外，如果一个表达式很长，一行无法写完时也可以换行。多行书写的表达式也应采取某种对齐方式，以利于人的理解，出现错误也容易发现和改正。

> C 语言里不规定表达式中两个运算对象求值顺序的定义，是想给 C 语言系统的实现者提供一些方便。这样，做编译程序的人就可以根据自己的认识和需要，自由地确定某种方式，这样可能提高系统实现的效率，也可能简化系统的实现。

2.4.4 计算和类型

由于参与计算的数据都有类型，计算过程中自然会出现许多与类型有关的问题。下面介绍其中的一些重要问题。

类型对计算的限制

两个 int 类型的数据经过计算，得到的还是 int 类型的结果。对长整数类型、各种实数类型，情况也一样。这个事实将带来许多后果。

首先，int 类型（以及 long 类型）数据的除法是整除，计算得到的商仍为整数，余数将被丢掉。这种情况有时会带来一些容易迷惑人的结果，例如下面两个表达式：

```
1 / 3 * 3   和   1 * 3 / 3
```

计算得到的结果不同，前一个表达式算出的值是 0。对所有整数类型都有这个问题，在写程序时必须注意。下面程序也不能正确算出平均值：

```
#include <stdio.h>

int main() {
    printf("Average of %d and %d is %d\n", 68, 39, (68 + 39) / 2);
    return 0;
}
```

算术类型的计算中还有一个共同问题。由于每个类型都有确定取值范围，超出这一范围的值在该类型中就无法表示了。前面说，两个同样类型计算对象的计算结果仍为这个类型的值。但这种计算结果也有可能超出相应类型的表示范围，而超范围的值又无法在该类型里表示，这样得到的结果就没有任何保证了。运行中出现这种情况称为“溢出”。

C 程序通常不对计算过程中发生溢出的情况做任何事情，不会报告错误，计算将继续下去。这里对溢出时得到的结果也没有任何规定。但无论如何，发生溢出后，得到的结果已不可能是我们所希望的东西了，随后的计算也不可能再有任何价值。例如，如果某 C 系统里的 int 类型由 16 位二进制表示，下面表达式就是不正确的：

```
32766 + 5
```

因为计算结果超出了 int 类型的最大值 32767，得到的结果不再有意义。写程序时应特别注意这种问题。如果认为可能出现溢出，那么就应该考虑换一个表示范围更大的类型，例如对上面例子，用：

```
32766L + 5L
```

就不会出问题。实数类型的计算中同样可能发生溢出。

混合类型计算和类型转换

当某个运算符的运算对象具有不同类型时，就出现了混合类型计算。例如表达式：

```
3.27 + 201
```

这里的一个运算对象是 double 类型，而另一个运算对象是 int 类型。混合类型计算中出现了一些新问题。

在 C 语言里，存在有 int 类型的加法运算，有 long 类型的加法运算，也有 double 类型的加法运算等等，所有这些运算都用 + 运算符表示。对表达式中出现的一个加运算符，编译程序需要根据运算对象的类型情况确定究竟怎样完成计算。例如，对于下面两个表达式里的加运算符，很容易确定应该用哪种加法：

```
3 + 2          应当用 int 类型的加法运算
3.0 + 2.0      应当用 double 类型的加法运算
```

但 C 语言里没有混合类型的（运算对象类型不同的）算术运算。当表达式计算中遇到混合类型计算时，处理方式是转换某个（或两个）运算对象的值，先从计算对象转换出相同类型的值，然后再做实际计算。这种由混合类型计算引起的类型转换称为算术运算中的**自动类型转换**。“自动”的意思就是说这种转换不需要在程序里明确写出。

自动类型转换的基本原则是把表示范围小的类型的值转换到表示范围大的类型的值。按规定，几个算术类型转换的排列顺序从小到大是：

```
int    long    float    double    long double
```

long double 是表示范围最大的类型。如果两个运算对象类型不同，就把位于位于左边的类型（小类型）的值转换到另一个类型（较大的类型）的值，然后用这个新值参与计算。例如，对表达式：

```
32767 + 2L
```

由于其中的一个运算对象是长整数，所以整数 32767 将被转换，产生对应的 long 类型值，然后用这个新值参与计算，得到的是 long 类型的 32769。

再看另一个混合计算表达式的例子：

```
2L + 3 * 4.5
```

计算这个表达式时，int 类型的 3 先被转换，产生一个 double 类型的值，然后用这个值与 4.5 计算，得到的结果仍是 double 类型的。下一步，long 类型的 2L 也被转换，得到对应 double 类型的值，参与计算，最后得到所需结果。图 2.1 形象地描述了这一计算过程。

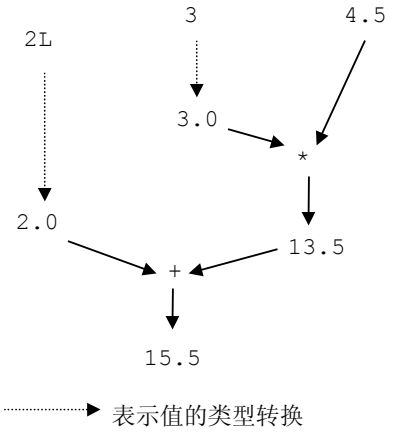

图 2.1　表达式 2L+3*4.5 的计算过程

注意，如图 2.1 中所示，在混合类型计算过程中自动地插入了一些隐含的数值转换动作，这种动作由具有某类型的原值出发，转换产生出一个所需类型的新值，用这个新值参与随后的计算。

下面程序计算这个表达式：

```
#include <stdio.h>

int main() {
    printf("%f", 2L + 3 * 4.5);
    return 0;
}
```

在写这个 printf 的格式描述串时，同样需要注意表达式的结果类型。

由上面例子可以看出，无论是写程序还是读程序，都需要特别注意表达式中计算对象的类型以及计算结果的类型，看清楚哪些地方将要发生类型转换，各是从什么类型的值向什么类型的值的转换。只有这样，才能清楚地理解一个表达式所描述的计算过程。

显式要求类型转换

如果表达式自然形成的计算过程不符合需要，可以通过适当加括号的方式，强制性地要

求某种特定计算顺序。与此类似，如果自动类型转换不能满足需要，语言也提供了显式要求做特定类型转换的描述形式。显式要求的类型转换也被称为*强制转换*或者*类型强制*，其形式是在被转换表达式前面写一对括号，括号里写一个类型名。这表示要求把表达式的计算结果转换到指定类型。例如，表达式：

```
(int)(3.6 * 15.8) + 4
```

就是要求把 3.6 * 15.8 计算的结果（一个 double 值）首先转换为 int 值，而后再用这个 int 值参与加法运算。实数类型值到整型值的转换方式是直接丢掉小数部分。

与类型转换有关的还有一些问题：

1. 类型转换中可能丢失信息。上面例子里要求的显式类型转换显然会丢失信息，从双精度值转换为整数时，原数的小数部分丢掉了。算术运算的自动转换有时也可能丢失信息，一个典型例子是长整数类型的数据转换到 float 类型。通常 float 类型的表示精度位数比长整数少，因此就可能产生丢失长整数低位信息的情况。
2. 如果被转换的值在给定结果类型里无法表示，那么转换的结果没有定义。这种情况在算术运算引起的自动类型转换里不会出现，写强制类型转换时必须注意。
3. 按照 C 语言规定，任何数值类型间的都可以互相转换。其他允许的转换在后面讨论。
4. 在 C 语言里，显式类型转换作为一元运算符看待，具有与其他一元运算符同样的优先级和结合方式。如果上面表达式里不写括号，其意义就会不同。请读者自己分析。
5. 类型转换是值的转换，是从一个值出发得到另一个不同类型的值的操作。类型转换不改变原来的值，而是产生一个新值。这一点也非常重要。

了解了这些情况后，我们已经可以用 C 语言写出各种计算算术表达式值的程序了。这类程序都可以采用前面示例的框架，程序里只用一个 printf 语句产生输出，被计算表达式写在相应的参数位置，在这些参数的前面用格式串描述所需要的输出格式。要特别注意的是格式串中的转换描述和对应参数之间的类型匹配问题。

2.5 数学函数库及其使用

2.5.1 函数、函数调用

读者已经看到过函数，前面程序中多次使用的 printf 就是一个函数。每个函数实现一个计算过程，printf 完成的是内部数据到外部表现形式的转换和输出。C 语言标准库还提供了许多其他函数，可供我们在程序里使用。标准库里有一组*数学函数*，它们可以计算出常用数学函数的函数值。了解了这方面情况后，我们就能写出更多程序了。

要使用一个函数，只需要知道：

1. 该函数的名字，
2. 该函数的使用方式，
3. 该函数完成什么计算，能给出什么结果。

完全不必关心这一函数的功能是通过什么样的计算过程实现的。C 语言提供标准库的目的就是为写程序的人提供方便，使人可以方便地使用这些函数。

举例说，标准库提供了一个名为 sin 的函数，其功能是从一个双精度数值出发，把这个值看成弧度，算出与之对应的正弦函数值，得到的结果也是个双精度数值。假设我们希望计算弧度 2.4 的正弦函数值的两倍，应该写的表达式就是：

```
2.0 * sin(2.4)
```

乘号后面的写法表示要求用函数 sin，送给函数去算的数值是 2.4。而后再用该函数计算得到的结果参与乘法运算，得到最后结果。这个表达式正好表达了所需要的计算。

送给函数作为计算对象（或表达式）称为函数的*实际参数*，简称*实参*。实际参数是具体函数计算的出发点。函数计算得到的值称为*计算结果*或*函数返回值*（简称*返回值*）。例如，下面表达式里两次使用 sin 函数，分别要求它从不同的实际参数出发做计算，表达式的最终值是这两次函数计算得到的结果的乘积：

```
sin(2.4) * sin(3.98)
```

使用一个函数的专门术语是*函数调用*，我们说在上面表达式里两次调用了 sin 函数。

在表达式中使用函数的一般形式是：

函数名(*实际参数*)

函数名(*实际参数*, *实际参数*)

……

函数名之后写括号，其中写实参表达式。如果一个函数调用需要多个实参，就用逗号分隔它们。上面给出了一个和两个实参的形式。特定函数通常对所要求的实参个数有明确规定。

这样，如果要计算两个边长分别为 3.5 和 4.72，两边夹角为 37 度的三角形的面积，表达式就可以写为：

```
3.5 * 4.72 * sin(37.0 / 180.0 * 3.1416) / 2.0
```

函数的实际参数可以是表达式（如上面这个例子）。显然，对一个函数调用而言，函数所指定的计算过程只能在作为它实际参数的表达式计算出值之后才能开始进行。在前面使用 printf 的程序示例中，已经多次用到比较复杂的实参表达式。

2.5.2 数学函数及其使用

标准库的每个数学函数都规定了参数个数，规定了所要求的实际参数的类型，都规定返回值类型。标准数学函数大都要求一个 double 参数，其返回值也是 double 类型。例如 sin 函数的情况就是这样。为方便起见，可以把 sin 函数的类型特征表述为：

```
double sin(double)
```

用这种方式说明函数名是 sin，它要求一个双精度参数（用写在括号里的一个 double 表示），返回双精度值（用写在函数名前面的 double 表示）。这种表述方式简洁明了。后面会看到这种表述方式在 C 语言程序里的用途。

标准库数学函数主要包括三角函数等：

三角函数	sin	cos	tan
反三角函数	asin	acos	atan
双曲函数	sinh	cosh	tanh

指数和对数函数：

以 e 为底的指数函数	exp
自然对数函数	log
以 10 为底的对数函数	log10

其他函数包括：

平方根	sqrt
绝对值	fabs
乘幂，第一个参数作为底，第二个是指数	double pow(double, double)
实数的余数，两个参数分别是被除数和除数	double fmod(double, double)

上面所有没给出类型特征的函数都要求一个参数，其参数与返回值都是 double 类型。最后两个函数要求两个 double 参数，返回 double 类型的值。pow 是求乘幂的函数，其第一个参数是底，第二个是指数。表达式：

```
pow(2.5, 3.4)
```

将计算出 $2.5^{3.4}$。当底为负数时，pow 要求指数参数必须是整数。fmod 求实数除法的余数（近似值），它的两个参数分别是被除数和除数。表达式：

```
fmod(235.74, 3.14159265)
```

求出 235.74 除以 3.14159265 的余数。余数的符号总与被除数相同。

如果程序里要用标准库里的数学函数，程序最前面要另写一行：

```
#include <math.h>
```

例：写程序求两邻边长度分别为 3.5 和 4.72 米，两边夹角为 37 度的三角形的面积。

根据前面的经验及函数调用的写法，这个程序可以写为：

```
#include <stdio.h>
#include <math.h>

int main () {
    printf("Area of the triangle: %fm^2\n",
           3.5 * 4.72 * sin(37.0 / 180 * 3.1416) / 2);
    return 0;
}
```

请注意，C 语言标准库的所有函数都采用完全由小写字母拼写的名字，数学函数也不例外，写函数名时必须注意。

有了数学函数之后，我们写程序的能力得到很大提高，所有能通过这些基本数学函数的复合与算术运算共同描述的计算过程，现在我们都能够写出相应的程序，算出所需要的结果。至少说，现在我们有了相当于普通科学计算器的编程序能力。由于在程序里可以写任意长的、具有任意层次嵌套结构的表达式，我们已经能够解决许多实际需要的计算问题了。本章习题中的程序可以采用这种模式写出来。

2.5.3 函数调用中的类型转换

函数对参数有明确的类型要求，当实参表达式的计算结果类型与函数要求不符时，又出现了类型问题。C 语言规定，在出现这种情况时，先把实参求出的值自动转换为函数所要求类型的值，然后再送给函数去计算。例如，在下面表达式的计算中，会出现两次自动类型转换。在两次调用 sin 时，整型参数值都将先转换到 double 值后才送给 sin：

```
sin(2) * sin(4)
```

假设有另一个函数 f，其类型特征为 int f(int)，在下面表达式里调用 f 时，实参算出的值将从 double 类型（表达式计算结果的类型）转换到 int 类型（f 所要求的实参类型），然后提供给函数使用：

```
4 * f(3.56 * 2.7)
```

例，计算 $\sum_{n=1}^{10} \sin\frac{1}{n}$ 的值。

初学者可能很容易就写出了下面程序：

```
#include <stdio.h>
#include <math.h>
int main () {
    printf("%f\n", sin(1) + sin(1/2) + sin(1/3) + sin (1/4) +
                   sin(1/5) + sin(1/6) + sin(1/7) + sin(1/8) +
                   sin(1/9) + sin(1/10) );
    return 0;
}
```

此后发现这个程序可以正常通过编译，但执行时却得不到正确结果。也就是说，这个程序有语义错误。为什么呢？如果读者在仔细读了这个程序之后还没有发现问题，那就应当复习一

下本章里对数据类型有关问题的讨论了。

例：已知三角形三边的长度分别是 3、5、7 厘米，求该三角形的面积。

首先找出已知三边求三角形面积的公式： $S=\sqrt{s(s-a)(s-b)(s-c)}$，公式中的 $s=\frac{1}{2}(a+b+c)$。根据这个公式，就可以写出下面程序：

```
#include <stdio.h>
#include <math.h>
int main () {
    printf("%f\n", sqrt((3+5+7)/2.0 * ((3+5+7)/2.0 - 3) *
                        ((3+5+7)/2.0 - 5) * ((3+5+7)/2.0 - 7)));
    return 0;
}
```

由于自动类型转换，这个程序能够得到正确的结果。请读者自己分析一下，在这个程序的执行过程中，在哪些地方将发生类型转换，各是从什么类型转换到什么类型。

问题解释:

1. (2.5.3) 程序中的问题在于函数 sin 的参数，虽然参数表达式算出的 int 值能自动转换到双精度值后送给函数，但参数表达式本身却是首先在整数类型里计算的。所以，除第一个函数调用能得到正确结果外，其他调用中实参求出的值都是 0。这可能使人意外。程序的更正很容易，例如把 sin(1/2) 改为 sin(1.0/2.0)，其他类似。

本章讨论的重要概念

基本字符，名字，标识符，关键字，数据，类型，类型名，文字量，整数类型，长整数类型，表示范围，十进制写法，八进制写法，十六进制写法，实数类型，浮点类型，双精度类型，长双精度类型，字符类型，特殊字符，字符串，数据的外部表示，数据的内部表示，表示形式的转换，函数，语句，输出函数 printf，实际参数，格式描述串，函数调用，转换描述，运算符，表达式，计算，一元运算符，二元运算符，运算对象，算术运算符（+、-、*、/、%），算术表达式，优先级，结合方式，求值顺序，溢出，类型转换，类型强制，数学函数，函数名，函数调用时的类型转换。

本章中的有用程序模式

程序模式 2.1：简单计算程序。

```
#include <stdio.h>

int main() {
    printf("... ...", ...); /* 计算表达式写在这里 */
    return 0;
}
```

程序模式 2.2：使用数学函数的简单计算程序采用如下模式：

```
#include <stdio.h>
#include <math.h>

int main() {
    printf("... ...", ...); /* 计算表达式写在这里 */
    return 0;
}
```

练习

1. 指出下面的哪些字符序列不是合法的标识符：

```
_abc        x+-         3x1         Xf_1__4     Eoof___
a$#24       x__x__2     bg--1       ____        I am
```

2. 手工计算下列表达式的值：

```
1）125 + 0125        2）0XAF - 0XFA
3）24 * 3 / 5 + 64）36 + - （5 - 23） / 4
5）35 * 012 + 27 / 4 / 7 * (12 - 4)
```

3. 在下面表达式的计算过程中，在什么地方将发生类型转换，各个转换是从什么类型转换到什么类型，表达式计算的结果是什么？

```
1）3 * (2L + 4.5f) - 012 + 44
2）3 * (int)sqrt(34) - sin(6) * 5 + 0x2AF
3）cos(2.5f + 4) - 6 *27L + 1526 - 2.4L
```

4. 写程序计算第 3 题中各个表达式的值。
5. 写程序计算下面各个表达式的值：

1）$\frac{2.34}{1+257}$　2）$\frac{1065}{24*13}$　3）$\frac{23.582}{7.96/3.67}$　4）$\sqrt{\pi^2+1}$　7）$\ln\ln(10^{2\pi}+1)$

5）$\log_5\sqrt{2\pi-1}$　6）$e^{\sqrt{\pi+1}}$　7）$\arctan(\log_3(e+\pi))$

8）$\sqrt{\frac{13-(2.24-0.24^2)^2}{3.68}}$　9）$1+\frac{2}{3+4/5}$　10）$\ln(2\pi\sqrt{13+e})$

6. 已知铁的比重是 7.86，金的比重是 19.3。写几个简单程序，分别计算出直径 100 毫米和 150 毫米的铁球与金球的重量。
7. 写程序计算 $5x^2+2x+6$ 的两个根，考虑用合适的方式输出。（提示：对这个具体问题上，由于人可以先计算出判别式 b^2-4ac 的值，以此作为已知信息，就可以写出程序了。）
8. 在计算机上试验本章正文中的一些程序。对它们做一些修改，观察程序加工和运行的情况，并对程序的行为做出解释。
9. 在一个能正确工作工作的输出整数结果的程序里，将 `printf` 的相应转换描述改为 `%f` 或者 `%ld`，看看会出现什么问题。在一个能正确工作工作的输出双精度结果的程序里，将 `printf` 的相应转换描述改为 `%d` 或者 `%ld`，看看会出现什么问题。