# 第3章　最简单的C程序设计——顺序程序设计

有了前两章的基础，现在可以开始由浅入深地学习C语言程序设计了。

为了能编写出C语言程序，必须具备以下的知识和能力。

(1) 要有正确的解题思路，即学会设计算法，否则无从入手。

(2) 掌握C语言的语法，知道怎样使用C语言所提供的功能编写出一个完整的、正确的程序。也就是在设计好算法之后，能用C语言正确表示此算法。

(3) 在写算法和编写程序时，要采用结构化程序设计方法，编写出结构化的程序。

算法的种类很多，不可能等到把所有算法都学透以后再来学习编程序。C语言的语法规定很多，很烦琐，孤立地学习语法不但枯燥乏味，而且即使倒背如流，也不一定能写出一个好的程序，必须找到一种有效的学习方法。

本书的做法是：以程序设计为主线，把算法和语法紧密结合起来，引导读者由易及难地学会编写C程序。对于简单的程序，算法比较简单，程序中涉及的语法现象也比较简单(一般只用到简单的变量、简单的输出格式)。对于比较复杂的算法，程序中用到的语法现象也比较复杂(例如要使用数组、指针和结构体等)。

本章先从简单的程序开始，介绍简单的算法，同时介绍最基本的语法现象，使读者具有编写简单的程序的能力。在此基础上，逐步介绍复杂一些的程序，介绍比较复杂的算法，同时介绍较深入的语法现象，把算法与语法有机地结合起来，由浅入深，由简单到复杂，使读者很自然地、循序渐进地学会编写程序。

## 3.1　顺序程序设计举例

**【例3.1】** 有人用温度计测量出用华氏法表示的温度(如64°F)，今要求把它转换为以摄氏法表示的温度(如17.8℃)。

**解题思路**：这个问题的算法很简单，关键在于找到二者之间的转换公式。根据物理学知识，知道以下转换公式：

$$c=\frac{5}{9}(f-32)$$

其中，f代表华氏温度，c代表摄氏温度。据此可以用N-S图表示算法，见图3.1。

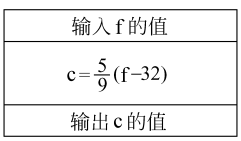


图　3.1

算法由3个步骤组成，这是一个简单的顺序结构。

**编写程序**：有了N-S图，很容易用C语言表示，写出求此问题的C程序。

```
#include <stdio.h>
int main()
```

```
{
  float f,c;                                  //定义 f 和 c 为单精度浮点型变量
  f=64.0;                                     //指定 f 的值
  c=(5.0/9)*(f-32);                           //利用公式计算 c 的值
  printf("f=%f\nc=%f\n",f,c);                 //输出 c 的值
  return 0;
}
```

**运行结果:**

```
f=64.000000
c=17.777778
```

读者应能看懂这个简单的程序。

**【例 3.2】** 计算存款利息。有 1000 元,想存一年。有 3 种方法可选:(1)活期,年利率为 r1;(2)一年定期,年利率为 r2;(3)存两次半年定期,年利率为 r3。请分别计算出一年后按 3 种方法所得到的本息和。

**解题思路:** 关键是确定计算本息和的公式。从数学知识可知,若存款额为 p0,则:

活期存款一年后本息和为 p1=p0(1+r1)。

一年定期存款,一年后本息和为 p2=p0(1+r2)。

两次半年定期存款,一年后本息和为 $p3=p0\left(1+\frac{r3}{2}\right)\left(1+\frac{r3}{2}\right)$。

画出 N-S 流程图,见图 3.2。

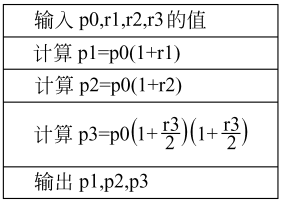


图 3.2

**编写程序:** 按照 N-S 图所表示的算法,很容易写出 C 程序。

```
#include <stdio.h>
int main()
{ float p0=1000, r1=0.0036, r2=0.0225, r3=0.0198, p1, p2, p3;
                                              //定义变量
  p1=p0*(1+r1);                               //计算活期本息和
  p2=p0*(1+r2);                               //计算一年定期本息和
  p3=p0*(1+r3/2)*(1+r3/2);                    //计算存两次半年定期的本息和
  printf("p1=%f\np2=%f\np3=%f\n",p1, p2, p3);          //输出结果
  return 0;
}
```

**运行结果:**

```
p1=1003.599976
p2=1022.500000
p3=1019.898010
```

第 1 行是活期存款一年后本息和,第 2 行是一年定期存款一年后本息和,第 3 行是两次半年定期存款一年后本息和。

**程序分析:** 第 4 行,在定义实型变量 p0,p1,p2,p3,r1,r2,r3 的同时,对变量 p0,r1,r2,r3 赋予初值。

第 8 行,在输出 p1,p2 和 p3 的值之后,用\n 使输出换行。

**注意:** 在 Visual C++ 6.0 系统中对以上两个程序进行编译时,会显示出"警告"信息。这是因为编译系统把所有实数都作为双精度数处理。因此提醒用户:把双精度常量转换成 float 型会造成精度损失。对这类"警告",用户知道是怎么回事就可以了。承认此现实,让程序继续进行

连接和运行，不影响运行结果。如果用 GCC 编译系统，则不会出现此“警告”信息。

## 3.2　数据的表现形式及其运算

有了以上写程序的基础，本节对程序中最基本的成分作必要的介绍。

**说明**：本节介绍的主要是 C 语言的一些语法规定，在编程序时会用到这些知识，因此不知道是不行的，所以本书作了简单的介绍。但是，不需要死记硬背，这样既枯燥又难以奏效，教师也不必在课堂中一一讲授。建议学习本节时采取“浏览”的方法，大致知道有这些因素就可以了，这样在遇到有关问题时就不会茫然。在后续的章节中，通过阅读程序和分析程序对这些内容会具体掌握的，必要时再回头查阅一下即可。

### 3.2.1　常量和变量

在计算机高级语言中，数据有两种表现形式：常量和变量。

#### 1. 常量

在程序运行过程中，其值不能被改变的量称为常量。如例 3.1 程序中的 5，9，32 以及例 3.2 程序中的 1000，0.0036，0.0225，0.0198 是常量。数值常量就是数学中的常数。

常用的常量有以下几类。

（1）**整型常量**。如 1000，12345，0，－345 等都是整型常量。

（2）**实型常量**。有两种表示形式。

① 十进制小数形式，由数字和小数点组成。如 123.456，0.345，－56.79，0.0，12.0 等。

② 指数形式，如 12.34e3（代表 $12.34\times10^{3}$），－346.87e－25（代表 $-346.87\times10^{-25}$），0.145E－25（代表 $0.145\times10^{-25}$）等。由于在计算机输入或输出时无法表示上角或下角，故规定以字母 e 或 E 代表以 10 为底的指数。但应注意：e 或 E 之前必须有数字，且 e 或 E 后面必须为整数。如不能写成 e4，12e2.5。

（3）**字符常量**。有两种形式的字符常量。

① **普通字符**，用单撇号括起来的一个字符，如'a'，'Z'，'3'，'?'，'＃'。不能把多个字符放在一对撇号里，例如'ab'或'12'是不正确的。请注意：单撇号只是界限符，字符常量只能是一个字符，不包括单撇号。'a'和'A'是不同的字符常量。字符常量存储在计算机存储单元中时，并不是存储字符（如 a，z，＃等）本身，而是以其代码（一般采用 ASCII 码）存储的，例如字符'a'的 ASCII 码是 97，因此，在存储单元中存放的是 97（以二进制形式存放）。ASCII 字符与代码对照表见附录 A[①]。

② **转义字符**，除了以上形式的字符常量外，C 语言还允许用一种特殊形式的字符常量，就是以字符“\”开头的字符序列。例如，前面已经遇到过的，在 printf 函数中的'\n'代表一

① C 语言并没有指定使用哪一种字符集，由各编译系统自行决定采用哪一种字符集。C 语言只是规定：基本字符集中的每个字符必须用 1 字节表示；空字符也占 1 字节，它的所有二进制位都是 0；对数字 0～9 字符的代码，后面一个数字的代码应比前一个数字的代码大 1（如在 ASCII 字符集中，数字'2'的代码是 50，数字'3'的代码是 51，后者比前者的代码大 1，符合要求）。绝大多数计算机系统采用 ASCII 字符集，ASCII 是 American Standard Code for Information Interchange（美国标准信息交换代码）的缩写。

个“换行”符。'\t'代表将输出的位置跳到下一个 Tab 位置(制表位置),一个 Tab 位置为 8 列。这是一种在屏幕上无法显示的“控制字符”,在程序中也无法用一个一般形式的字符来表示,只能采用这样的特殊形式来表示。

常用的以“\”开头的特殊字符见表 3.1。

**表 3.1 转义字符及其作用**

| 转义字符 | 字符值 | 输出结果 |
|---|---|---|
| \' | 一个单撇号(') | 输出单撇号字符' |
| \" | 一个双撇号(") | 输出双撇号字符" |
| \? | 一个问号(?) | 输出问号字符? |
| \\ | 一个反斜线(\) | 输出反斜线字符\ |
| \a | 警告(alert) | 产生声音或视觉信号 |
| \b | 退格(backspace) | 将光标当前位置后退一个字符 |
| \f | 换页(form feed) | 将光标当前位置移到下一页的开头 |
| \n | 换行 | 将光标当前位置移到下一行的开头 |
| \r | 回车(carriage return) | 将光标当前位置移到本行的开头 |
| \t | 水平制表符 | 将光标当前位置移到下一个 Tab 位置 |
| \v | 垂直制表符 | 将光标当前位置移到下一个垂直制表对齐点 |
| \o、\oo 或\ooo<br>其中 o 代表一个八进制数字 | 与该八进制码对应的 ASCII 字符 | 与该八进制码对应的字符 |
| \xh[h…]<br>其中 h 代表一个十六进制数字 | 与该十六进制码对应的 ASCII 字符 | 与该十六进制码对应的字符 |

表 3.1 中列出的字符称为**转义字符**,意思是将“\”后面的字符转换成另外的意义。如“\n”中的“n”不代表字母 n,而是作为“换行”符。

表 3.1 中倒数第 2 行是一个以八进制数表示的字符,例如'\101'代表八进制数 101 的 ASCII 字符,即'A'(八进制数 101 相当于十进制数 65,从附录 A 可以看到 ASCII 码(十进制数)为 65 的字符是大写字母'A')。'\012'代表八进制数 12(即十进制数的 10)的 ASCII 码所对应的字符“换行”符。表 3.1 中倒数第 1 行是一个以十六进制数表示的 ASCII 字符,如'\x41'代表十六进制数 41 的 ASCII 字符,也是'A'(十六进制数 41 相当于十进制数 65)。用表 3.1 中的方法可以表示任何可显示的字母字符、数字字符、专用字符、图形字符和控制字符。如'\033'或'\x1B'代表 ASCII 码为 27 的字符,即 ESC 控制符。'\0'或'\000'是代表 ASCII 码为 0 的控制字符,即“空操作”字符,它常用在字符串中。

(4) **字符串常量**。如"boy","123"等,用双撇号把若干字符括起来,字符串常量是双撇号中的全部字符(但不包括双撇号本身)。注意不能错写成'CHINA','boy','123'。单撇号内只能包含一个字符,双撇号内可以包含一个字符串。

**说明**:从其字面形式上即可识别的常量称为“字面常量”或“直接常量”。字面常量

是没有名字的不变量。

(5) **符号常量**。用 # define 指令，指定用一个符号名称代表一个常量。例如：

```
#define  PI  3.1416                    //注意行末没有分号
```

经过以上的指定后，本文件中从此行开始所有的 PI 都代表 3.1416。在对程序进行编译前，预处理器先对 PI 进行处理，把所有 PI 全部置换为 3.1416。这种用一个符号名代表一个常量的，称为**符号常量**。在预编译后，符号常量已全部变成字面常量(3.1416)。使用符号常量有以下好处。

① 含义清楚。看程序时从 PI 就可大致知道它代表圆周率。在定义符号常量名时应考虑“见名知义”。在一个规范的程序中不提倡使用很多的常数，如 sum＝15 * 30 * 23.5 * 43，在检查程序时搞不清各个常数究竟代表什么。应尽量使用“见名知义”的变量名和符号常量。

② 在需要改变程序中多处用到的同一个常量时，能做到“一改全改”。例如在程序中多处用到某物品的价格，如果价格用一个常数 30 表示，则在价格调整为 40 时，就需要在程序中作多处修改，若用符号常量 PRICE 代表价格，只须改动一处即可：

```
#define PRICE 40
```

**注意**：要区分符号常量和变量，不要把符号常量误认为变量。符号常量不占内存，只是一个临时符号，代表一个值，在预编译后这个符号就不存在了，故不能对符号常量赋新值。符号常量是一种“宏”，即编译预处理时的替换，为与变量名相区别，习惯上符号常量用大写表示，如 PI、PRICE 等。

### 2. 变量

如例 3.1 程序中的 c，f 和例 3.2 程序中的 p0，p1，p2，p3，r1，r2，r3 等是变量。变量代表一个有名字的、具有特定属性的一个存储单元。它用来存放数据，也就是存放变量的值。在程序运行期间，变量的值是可以改变的。

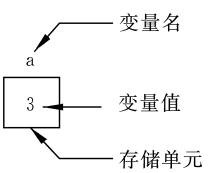

图　3.3

变量必须**先定义，后使用**[①]。在定义时指定该变量的类型和名字。一个变量应该有一个名字，以便被引用。请注意区分**变量名**和**变量值**这两个不同的概念，图 3.3 中 a 是变量名，3 是变量 a 的值，即存放在变量 a 的内存单元中的数据。变量名实际上是以一个名字代表的一个存储地址。在对程序编译连接时由编译系统给每一个变量名分配对应的内存地址。从变量中取值，实际上是通过变量名找到相应的内存地址，从该存储单元中读取数据。

### 3. 常变量

C99 允许使用**常变量**，方法是在定义变量时，前面加一个关键字 const，例如：

```
const int a=3;
```

定义 a 为一个整型变量，指定其值为 3，而且在变量存在期间其值不能改变。

---

① 定义变量的位置：一般在函数开头的声明部分中定义变量，也可以在函数外定义变量(即外部变量、全局变量，见第 7 章)。C89/90 允许在函数中的复合语句(用一对花括号包起来)中定义变量。C99 进一步允许变量定义出现在其他可执行语句之后(但必须在使用这个变量之前)，也允许在 for 循环初始条件中定义变量。

常变量与常量的异同是：常变量具有变量的基本属性：有类型，占存储单元，只是不允许改变其值。可以说，常变量是有名字的不变量，而常量是没有名字的不变量。有名字就便于在程序中被引用。

**请思考**：常变量与符号常量有什么不同？例如：

```
#define Pi 3.1415926                    //定义符号常量
const float pi=3.1415926;               //定义常变量
```

符号常量 Pi 和常变量 pi 都代表 3.1415926，在程序中都能使用。但二者性质不同：定义符号常量用#define 指令，它是预编译指令，它只是用符号常量代表一个字符串，在预编译时仅进行字符替换，在预编译后，符号常量就不存在了(全置换成 3.1415926 了)，对符号常量的名字是不分配存储单元的。而常变量要占用存储单元，有变量值，有变量类型，只是该值不可以改变而已。从使用的角度看，常变量具有许多符号常量的优点，而且使用更方便。有了常变量以后，可以不必多用符号常量。

**说明**：实现 C99 或更新标准的编译系统才能使用常变量。

**4. 标识符**

在计算机高级语言中，用来对变量、符号常量名、函数、数组、类型等命名的有效字符序列统称为**标识符**(identifier)。简单地说，标识符就是一个对象的名字。前面用到的变量名 p1，p2，c，f，符号常量名 PI，PRICE，函数名 printf 等都是标识符。

C 语言规定标识符只能由字母、数字和下画线 3 种字符组成，且第 1 个字符必须为字母或下画线。下面列出的是合法的标识符，可以作为变量名：

sum，average，_total，Class，day，month，Student_name，lotus_1_2_3，BASIC，li_ling。

下面是不合法的标识符和变量名：

M.D.John，￥123，#33，3D64，a>b

**注意**：编译系统认为大写字母和小写字母是两个不同的字符。因此，sum 和 SUM 是两个不同的变量名，同样，Class 和 class 也是两个不同的变量名。一般而言，变量名用小写字母表示，与人们日常习惯一致，以提高可读性。

### 3.2.2 数据类型

在例 3.1 和例 3.2 中可以看到，在定义变量时需要指定变量的类型。如例 3.1 中变量 f 和 c 被定义为单精度(float)型。C 语言要求在定义所有的变量时都要指定变量的类型。常量也是区分类型的。

为什么在用计算机运算时要指定数据的类型呢？在数学中，数值是不分类型的，数值的运算是绝对准确的，例如，78 与 97 之和为 175，1/3 的值是 0.33333333…(循环小数)。数学是一门研究抽象问题的学科，数和数的运算都是抽象的。而在计算机中，数据是存放在存储单元中的，它是具体存在的。而且，存储单元是由有限的字节构成的，每个存储单元中存放数据的范围是有限的，不可能存放“无穷大”的数，也不能存放循环小数。例如用 C 程序计算和输出 1/3：

```
printf("%f",1.0/3.0);
```

得到的结果是 0.333333，只能得到 6 位小数，而不是无穷位的小数。

**注意**：用计算机进行的计算不是抽象的理论值的计算，而是用工程的方法实现的计算，在许多情况下无法得到绝对精确的结果。

所谓类型，就是对数据分配存储单元的安排，包括存储单元的长度（占多少字节）以及数据的存储形式。不同的类型分配不同的长度和存储形式。

C 语言允许使用的类型见图 3.4，图中有 * 的是 C99 所增加的。

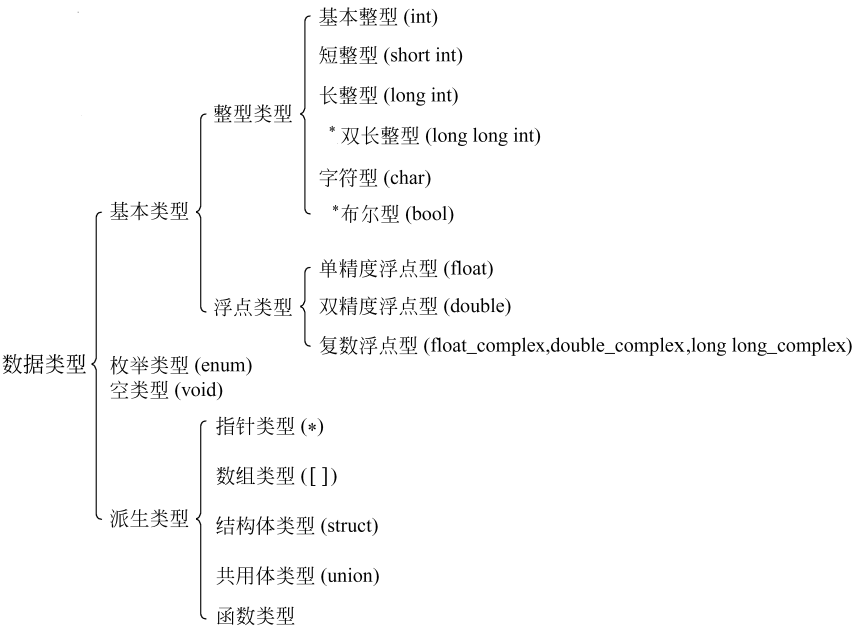


图　3.4

其中，基本类型（包括整型和浮点型）和枚举类型变量的值都是数值，统称为算术类型（arithmetic type）。算术类型和指针类型统称为纯量类型（scalar type），因为其变量的值是以数字来表示的。枚举类型是程序中用户定义的整数类型。数组类型和结构体类型统称为组合类型（aggregate type），共用体类型不属于组合类型，因为在同一时间内只有一个成员具有值。函数类型用来定义函数，描述一个函数的接口，包括函数返回值的数据类型和参数的类型。

不同类型的数据在内存中占用的存储单元长度是不同的，例如，Visual C++ 为 char 型（字符型）数据分配 1 字节，为 int 型（基本整型）数据分配 4 字节，存储不同类型数据的方法也是不同的。

本书不孤立地、枯燥地叙述以上各种类型的规则，而是结合编程介绍怎样使用各种数据类型。本章及第 4、5 章介绍基本数据类型的应用，第 6 章介绍数组，第 7 章介绍函数，第 8 章介绍指针，第 9 章介绍结构体类型、共用体类型和枚举类型。

### 3.2.3 整型数据

#### 1. 整型数据的分类

本节介绍最基本的整型类型。

(1) **基本整型**(int 型)。

编译系统分配给 int 型数据 2 字节或 4 字节(由具体的 C 编译系统自行决定)。如 Turbo C 2.0 为每个整型数据分配 2 字节(16 位),而 Visual C++ 为每个整型数据分配 4 字节(32 位)。在存储单元中的存储方式是:用整数的补码(complement)形式存放。一个正数的补码是此数的二进制形式,如 5 的二进制形式是 101,如果用 2 字节存放一个整数,则在存储单元中数据形式如图 3.5 所示。如果是一个负数,则应先求出负数的补码。求负数的补码的方法是:先将此数的绝对值写成二进制形式,然后对其所有二进制位按位取反,再加 1,如−5 的补码见图 3.6。

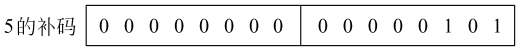


图 3.5

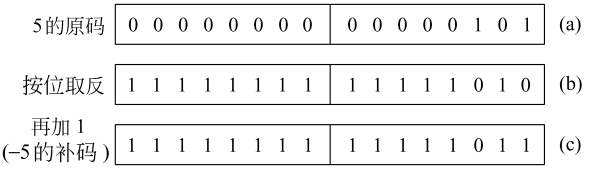


图 3.6

在存放整数的存储单元中,最左面一位是用来表示符号的。如果该位为 0,表示数值为正;如果该位为 1,表示数值为负。

有关补码的知识不属本书范围,在此不深入介绍,如需进一步了解,可参考有关计算机原理的书籍。

**说明**:早期的 C 语言编译器如果给整型变量分配 2 字节,则存储单元中能存放的最大值为 0111111111111111,第 1 位为 0 代表正数,后面 15 位为全 1,此数值是$(2^{15}-1)$,即十进制数 32 767。最小值为 1000000000000000,此数是$-2^{15}$,即−32 768。因此一个整型变量的值的范围是−32 768～32 767。超过此范围,就出现数值的“溢出”,输出的结果显然不正确。多数 64 位系统中的 C 语言编译器给整型变量分配 4 字节,其能容纳的数值范围为$-2^{31}$～$(2^{31}-1)$,即−2 147 483 648～2 147 483 647。

(2) **短整型**(short int)。

类型名为 short int 或 short。多数编译系统分配给 int 型数据 4 字节,短整型数据 2 字节。存储方式与 int 型相同。一个短整型变量的值的范围是−32 768～32 767。

(3) **长整型**(long int)。

类型名为 long int 或 long。Visual C++ 给一个 long 型数据分配 4 字节(即 32 位),因此 long int 型变量的值的范围是$-2^{31}$～$(2^{31}-1)$,即−2 147 483 648～2 147 483 647。多数其他支持 64 位系统的 C 语言编译器给 long 型数据分配 8 字节(即 64 位),因此在这些系统

中 long 型变量的值的范围是$-2^{63}$～$(2^{63}-1)$。

(4) **双长整型**(long long int)。

类型名为 long long int 或 long long,分配 8 字节。这是 C99 新增的类型,但许多 C 编译系统中 long 已经是 8 字节,所以 long long 与 long 没有区别。

**说明**:C 标准没有具体规定各种类型数据所占用存储单元的长度,这是由各编译系统自行决定的。C 标准只要求 long 型数据长度不短于 int 型,short 型不长于 int 型。即

sizeof(short)≤sizeof(int)≤sizeof(long)≤sizeof(long long)

sizeof 是测量类型或变量长度的运算符。在 Turbo C 2.0 中,int 型和 short 型数据都是2 字节(16 位),而 long 型数据是 4 字节(32 位)。在 Visual C++ 中,short 数据的长度为 2 字节,int 型数据的长度为 4 字节,long 型数据的长度为 4 字节。通常的做法是:把 long 型定为 32 位,把 short 型定为 16 位,而 int 型可以是 16 位,也可以是 32 位,由编译系统决定。读者应了解所用系统的规定。在将一个程序从 A 系统移到 B 系统时,需要注意这个区别。例如,在 A 系统,整型数据占 4 字节,程序中将整数 50000 赋给整型变量 price 是合法的、可行的。但在 B 系统,整型数据占 2 字节,将整数 50000 赋给整型变量 price 就超过整型数据的范围,出现"溢出"。这时应当把 int 型变量改为 long 型,才能得到正确的结果。

### 2. 整型变量的符号属性

以上介绍的几种类型,变量值在存储单元中都是以补码形式存储的,存储单元中的第 1 个二进制位代表符号。整型变量的值的范围包括负数到正数(见表 3.2)。

**表 3.2　整型数据常见的存储空间和值的范围(以 Microsoft Visual C++ 为例)**

| 类　　型 | 字节数 | 取 值 范 围 |
|---|---|---|
| int(基本整型) | 4 | −2 147 483 648～2 147 483 647,即$-2^{31}$～$(2^{31}-1)$ |
| unsigned int(无符号基本整型) | 4 | 0～4 294 967 295,即 0～$(2^{32}-1)$ |
| short(短整型) | 2 | −32 768～32 767,即$-2^{15}$～$(2^{15}-1)$ |
| unsigned short(无符号短整型) | 2 | 0～65 535,即 0～$(2^{16}-1)$ |
| long(长整型) | 4 | −2 147 483 648～2 147 483 647,即$-2^{31}$～$(2^{31}-1)$ |
| unsigned long(无符号长整型) | 4 | 0～4 294 967 295,即 0～$(2^{32}-1)$ |
| long long(双长型) | 8 | −9 223 372 036 854 775 808～9 223 372 036 854 775 807<br>即$-2^{63}$～$(2^{63}-1)$ |
| unsigned long long<br>(无符号双长整型) | 8 | 0～18 446 744 073 709 551 615,即 0～$(2^{64}-1)$ |

在实际应用中,有的数据的范围常常只有正值(如学号、年龄、库存量、存款额等)。为了充分利用变量的值的范围,可以将变量定义为"无符号"类型。可以在类型符号前面加上修饰符 unsigned,表示指定该变量是"无符号整数"类型。如果加上修饰符 signed,则是"有符号类型"。因此,在以上 4 种整型数据的基础上可以扩展为以下 8 种整型数据:

| | |
|---|---|
| 有符号基本整型 | [signed] int |
| 无符号基本整型 | unsigned int |
| 有符号短整型 | [signed] short [int] |
| 无符号短整型 | unsigned short [int] |
| 有符号长整型 | [signed] long [int] |
| 无符号长整型 | unsigned long [int] |
| 有符号双长整型* | [signed] long long [int] |
| 无符号双长整型* | unsigned long long [int] |

以上有"*"的是 C99 增加的,方括号表示其中的内容是可选的,既可以有,也可以没有。如果既未指定为 signed 也未指定为 unsigned 的,默认为"有符号类型"。如 signed int a 和 int a 等价。

有符号整型数据存储单元中最高位代表数值的符号(0 为正,1 为负)。如果指定 unsigned(为无符号)型,存储单元中全部二进制位(b)都用作存放数值本身,而没有符号。无符号型变量只能存放不带符号的整数,如 123,4687 等,而不能存放负数,如-123,-3。由于左面最高位不再用来表示符号,而用来表示数值,因此无符号整型变量中可以存放的正数的范围比一般整型变量中正数的范围扩大一倍。如果在程序中定义 a 和 b 两个短整型变量(占 2 字节),其中 b 为无符号短整型:

```
short a;                          //a 为有符号短整型变量
unsigned short b;                 //b 为无符号短整型变量
```

则变量 a 的数值范围为-32 768～32 767,而变量 b 的数值范围为 0～65 535。图 3.7(a)表示有符号整型变量 a 的最大值(32 767),图 3.7(b)表示无符号整型变量 b 的最大值(65 535)。

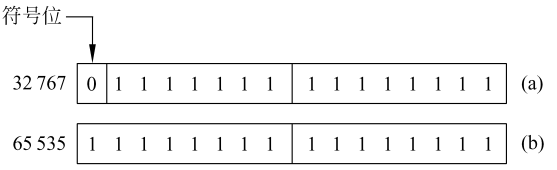


图 3.7

**说明:**

(1) 只有整型(包括字符型)数据可以加 signed 或 unsigned 修饰符,实型数据不能加。

(2) 对无符号整型数据用"%u"格式输出。%u 表示用无符号十进制数的格式输出。例如:

```
unsigned short price = 50;                //定义 price 为无符号短整型变量
printf("%u\n",price);                     //指定用无符号十进制数的格式输出
```

在将一个变量定义为无符号整型后,不应向它赋予一个负值,否则会得到错误的结果。例如:

```
unsigned short price = -1;                //不能把一个负整数存储在无符号变量中
printf("%d\n",price);
```

得到结果为 65535。显然与原意不符。

**请思考**:这是为什么?

原因是：系统对－1 先转换成补码形式，就是全部二进位都是 1(见图 3.8)，然后把它存入变量 price 中。由于 price 是无符号短整型变量，其左面第一位不代表符号，按“%d”格式输出，就是 65535。

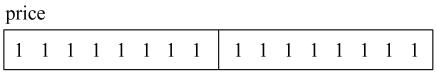


图　3.8

对以上补码的表示有初步了解即可，暂时可不细究。

**说明**：在程序中经常会对各种类型的数据进行操作，使用 C 语言编程时应当对数据在计算机内部的存储情况有一些基本的了解。否则，对运行时出现的问题会感到莫名其妙，无从分析。

### 3.2.4　字符型数据

由于字符是按其代码(整数)形式存储的，因此 C 把字符型数据作为整数类型的一种。但是，字符型数据在使用上有自己的特点，与前面介绍的 short，int，long 和 long long 型数据都不同，因此把它单独列为一节来介绍。

#### 1. 字符与字符代码

字符与字符代码并不是任意写一个字符，程序都能识别的。例如，代表圆周率的 π 在程序中是不能识别的，只能使用系统的字符集中的字符，目前大多数系统采用 ASCII 字符集。各种字符集(包括 ASCII 字符集)的基本集都包括了 127 个字符。其中包括：

- 字母：大写英文字母 A～Z，小写英文字母 a～z。
- 数字：0～9。
- 专门符号：29 个，包括

  ! " # & ' ( ) * + , － . / : ; ＜ ＝ ＞ ? [ \ ] ^ _ ` { | } ～
- 空格符：空格、水平制表符(tab)、垂直制表符、换行、换页(form feed)。
- 不能显示的字符：空(null)字符(以'\0'表示)、警告(以'\a'表示)、退格(以'\b'表示)、回车(以'\r'表示)等。

详见附录 A(ASCII 字符表)。这些字符用来写英文文章、材料或编程序基本够用了。

前已说明，字符是以整数形式(字符的 ASCII 码)存放在内存单元中的。例如：

大写字母'A'的 ASCII 码是十进制数 65，二进制形式为 1000001。

小写字母'a'的 ASCII 码是十进制数 97，二进制形式为 1100001。

数字字符'1'的 ASCII 码是十进制数 49，二进制形式为 0110001。

空格字符' '的 ASCII 码是十进制数 32，二进制形式为 0100000。

专用字符'%'的 ASCII 码是十进制数 37，二进制形式为 0100101。

转义字符'\n'的 ASCII 码是十进制数 10，二进制形式为 0001010。

可以看到，以上字符的 ASCII 码最多用 7 个二进制位就可以表示。所有 127 个字符都可以用 7 个二进制位表示(ASCII 码为 127 时，二进制形式为 1111111，7 位全 1)。所以在 C 语言中，指定用 1 字节(8 位)存储一个字符(所有系统都不例外)。此时，字节中的第 1 位置为 0。

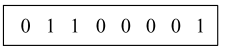


图 3.9

小写字母'a'在内存中的存储情况见图 3.9('a'的 ASCII 码是十进制数 97,二进制数为 01100001)。

**注意**:字符'1'和整数 1 是不同的概念。字符'1'只是代表一个形状为'1'的符号,在需要时按原样输出,在内存中以 ASCII 码形式存储,占 1 字节,见图 3.10(a);而整数 1 是以整数存储方式(二进制补码方式)存储的,占 2 或 4 字节,见图 3.10(b)。

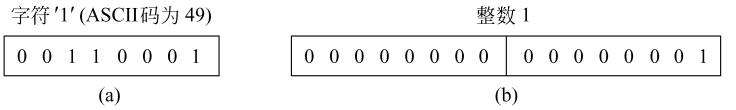


图 3.10

整数运算 1+1 等于整数 2,而字符'1'+'1'并不等于整数 2 或字符'2'。

**2. 字符变量**

字符变量是用类型符 char 定义字符变量。char 是英文 character(字符)的缩写,见名即可知义。例如:

```
char c='?';
```

定义 c 为字符型变量并使初值为字符'?'。'?'的 ASCII 码是 63,系统把整数 63 赋给变量 c。

c 是字符变量,实质上是 1 字节的整型变量,由于它常用来存放字符,所以称为字符变量。可以把 0～127 的整数赋给一个字符变量。

在输出字符变量的值时,可以选择以十进制整数形式输出,或以字符形式输出。例如:

```
printf("%d  %c\n",c,c);
```

输出结果是

```
63  ?
```

**说明**:用"%d"格式输出十进制整数 63,用"%c"格式输出字符'?'。

前面介绍了整型变量可以用 signed 和 unsigned 修饰符表示符号属性。字符类型也属于整型,也可以用 signed 和 unsigned 修饰符。

字符型数据的存储空间和值的范围见表 3.3。

**表 3.3 字符型数据的存储空间和值的范围**

| 类 型 | 字 节 数 | 取 值 范 围 |
|---|---|---|
| signed char(有符号字符型) | 1 | −128～127,即 $-2^7$～$(2^7-1)$ |
| unsigned char(无符号字符型) | 1 | 0～255,即 0～$(2^8-1)$ |

**说明**:在使用有符号字符型变量时,允许存储的值为−128～127,但字符的代码不

可能为负值，所以在存储字符时实际上只用到 0～127 这一部分，其第 1 位都是 0[①]。

C 语言是全面支持包括中文在内的多语种文字的，C95 开始支持多字节类型，wchar_t 在 Windows 上一般是 2 字节，16 个二进制位，在 UNIX/Linux 系统中则一般是 4 字节，32 个二进制位；C99 明确了支持 Unicode。C11 支持 UTF-16/UTF-32 类型 char16_t 和 char32_t；C23 进一步增加了支持 UTF-8 的类型 char8_t。这 3 种新类型都是无符号整型的别称。其中，char32_t 可以用于存储一个完整的 Unicode 字符，而单个 char16_t 和 char8_t 不足以表示所有 Unicode 字符，因此有可能需要两个 char16_t 或者多达 4 个 char8_t 来表达一个完整的高位 Unicode 字符。

### 3.2.5　浮点型数据

浮点型数据是用来表示具有小数点的实数的。为什么在 C 中把实数称为浮点数呢？在 C 语言中，实数是以指数形式存放在存储单元中的。一个实数表示为指数可以有不止一种形式，如 3.14159 可以表示为 $3.14159\times10^{0}$，$0.314159\times10^{1}$，$0.0314159\times10^{2}$，$31.4159\times10^{-1}$，$314.159\times10^{-2}$ 等，它们代表同一个值。可以看到，小数点的位置是可以在 314159 几个数字之间、之前或之后（加 0）浮动的，只要在小数点位置浮动的同时改变指数的值，就可以保证它的值不会改变。由于小数点位置可以浮动，所以实数的指数形式称为**浮点数**。

浮点数类型包括 float（单精度浮点型）、double（双精度浮点型）、long double（长双精度浮点型）。

（1）**float 型**（单精度浮点型）。编译系统为每一个 float 型变量分配 4 字节，数值以规范化的二进制数指数形式存放在存储单元中。在存储时，系统将实型数据分成小数部分和指数部分两部分，分别存放。小数部分的小数点前面的数为 0。如 3.14159 在内存中的存放形式可以用图 3.11 表示。

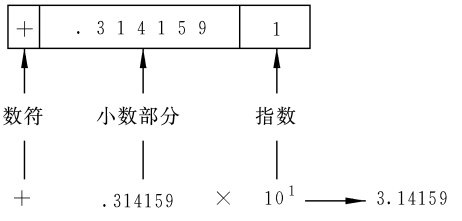

图　3.11

图 3.11 是用十进制数来示意的，实际上在计算机中是用二进制数来表示小数部分以及用 2 的幂次来表示指数部分的。在 4 字节（32 位）中，究竟用多少位来表示小数部分，多少位来表示指数部分，C 标准并无具体规定，由各 C 语言

---

① 前面已介绍：127 个基本字符用 7 个二进制位存储，如果系统只提供 127 个字符，那么就将 char 型变量的第 1 个二进制位设置为 0，用后面 7 位存放 127 个字符的代码。在这种情况下，系统提供的 char 型相当于 signed char。但是在实际应用中，往往觉得 127 个字符不够用，希望能多提供一些可用的字符。根据此需要，有的系统提供了扩展的字符集。把可用的字符由 127 个扩展为 255 个，即扩大了一倍。怎么解决这个问题呢？就是把本来不用的第一位用起来。把 char 变量改为 unsigned char，即第一位并不固定设为 0，而是把 8 位都用来存放字符代码。这样，可以存放 $2^{8}-1$ 即 255 个字符代码。附录 A 中 ASCII 码的 128～255 部分就是某系统扩展的 ASCII 字符，它并不适用于所有的系统。

读者可以用以下语句检查 ASCII 码 128～255 部分的扩展字符。

```
unsigned char c=128;                    //定义 c 为无符号字符变量
printf("%d:%c\n",c,c);                  //输出 ASCII 码为 128 的字符
```

观察是否输出附录 A 中代码为 128 的字符。可以用类似方法检查其他扩展字符。

中文有成千上万个汉字，即使用 255 个代号也是无法完整编码所有汉字的，因此，中文操作系统是多字节操作系统，ASCII 码为 127 以后的部分被在多数中文操作系统中与代表中文字符的代码重叠，故不会显示出附录 A 中的扩展字符。

编译系统自定。有的C语言编译系统以24位表示小数部分(包括符号),以8位表示指数部分(包括指数的符号)。由于用二进制形式表示一个实数以及存储单元的长度是有限的,因此不可能得到完全精确的值,只能存储成有限的精确度。而且与整数不同,用二进制形式表示小数与十进制小数并完全一一对应,因此有些十进制有限小数依然无法用二进制形式精确表示,会有精度损失。小数部分占的位(bit)数越多,数的有效数字越多,精度也就越高。指数部分占的位数越多,则能表示的数值范围越大。float型数据能得到6位有效数字,数值范围为$-3.4\times10^{-38}\sim3.4\times10^{38}$。

(2) **double型**(双精度浮点型)。为了扩大能表示的数值范围,用8字节存储一个double型数据,可以得到15位有效数字,数值范围为$-1.7\times10^{-308}\sim1.7\times10^{308}$。为了提高运算精度,在C语言中进行浮点数的算术运算时,将float型数据都自动转换为double型,然后进行运算。

(3) **long double型**(长双精度)型,不同的编译系统对long double型的处理方法不同,GCC对long double型分配16字节。而Visual C++则对long double型和double型一样处理,分配8字节。请读者在使用不同的编译系统时注意其差别。

表3.4列出实型数据的有关情况(Visual C++环境下)。

**表3.4 实型数据的有关情况**

| 类 型 | 字 节 数 | 有效数字 | 数值范围(绝对值) |
|---|---|---|---|
| float | 4 | 6 | 0以及$1.2\times10^{-38}\sim3.4\times10^{38}$ |
| double | 8 | 15 | 0以及$2.3\times10^{-308}\sim1.7\times10^{308}$ |
| long double | 8 | 15 | 0以及$2.3\times10^{-308}\sim1.7\times10^{308}$ |
| | 16 | 19 | 0以及$3.4\times10^{-4932}\sim1.1\times10^{4932}$ |

**说明**:用有限的存储单元不可能完全精确地存储一个实数,例如float型变量能存储的最小正数为$1.2\times10^{-38}$,不能存放绝对值小于此值的数,如$10^{-40}$。float型变量能存储的范围见图3.12。即数值可以在3个范围内:(1)$-3.4\times10^{38}\sim-1.2\times10^{-38}$;(2)0;(3)$1.2\times10^{-38}\sim3.4\times10^{38}$。

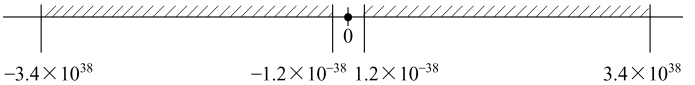


图 3.12

### 3.2.6 怎样确定常量的类型

在C语言中,不仅变量有类型,常量也有类型。为什么要把常量分为不同的类型呢?在程序中出现的常量是要存放在计算机中的存储单元中的。这就必须确定分配给它多少字节,按什么方式存储。例如,程序中有整数12,在Visual C++中会分配给它4字节,按补码方式存储。

怎样确定常量的类型呢?从常量的表示形式即可以判定其类型。对于字符常量很简单,只要看到由单撇号括起来的单个字符或转义字符就可以知道它是字符常量。对于数值

常量按以下规律判断。

**整型常量**。不带小数点的数值是整型常量，但应注意其有效范围。在早期的 C 语言编译器(如 Turbo C)中，系统为整型数据分配 2 字节，其表值范围为－32 768～32 767，如果在程序中出现数值常量 23 456，系统把它作为 int 型处理，用 2 字节存放。如果出现 49 875，由于超过 32 767，2 字节放不下，系统会把它作为长整型(long int)处理，分配 4 字节。在较新的 C 语言编译器(如 GCC 和 Visual C++ )中，在范围－2 147 483 648～2 147 483 647 的不带小数点的数都作为 int 型，分配 4 字节，在此范围外，而又在 long long 型数的范围内的整数，作为 long long 型处理。

在一个整数的末尾加大写字母 L 或小写字母 l，表示它是长整型(long int)。例如 123L，234l 等。GCC 会分配 8 字节用于存储长整型数据，但在 Visual C++ 中由于对 int 和 long int 型数据都分配 4 字节，因此如果需要存储更大的整数需要用 long long 型，而用 long int 型没有实际作用。

**浮点型常量**。凡以小数形式或指数形式出现的实数均是浮点型常量，在内存中都以指数形式存储。例如，10 是整型常量，10.0 是浮点型常量。那么对浮点型常量是按单精度处理还是按双精度处理呢？C 语言编译系统把浮点型常量都按双精度处理，分配 8 字节。

**注意**：C 程序中的实型常量都作为双精度浮点型常量。

如果有

```
float a=3.14159;
```

在进行编译时，对 float 变量分配 4 字节，但对于浮点型常量 3.14159，则按双精度处理，分配 8 字节。编译系统会发出“警告”(warning：truncation from 'const double' to 'float')。意为“把一个双精度常量转换为 float 型”，提醒用户注意这种转换可能损失精度。这样的“警告”，一般不会影响程序运行结果的正确性，它用于提醒用户可能会影响程序运行结果的精确度的情况。

可以在常量的末尾加专用字符，强制指定常量的类型。如在 3.14159 后面加字母 F 或 f，就表示是 float 型常量，分配 4 字节。如果在实型常量后面加大写或小写的 L，则指定此常量为 long double 型。例如：

```
float a=3.14159f;              //把此 3.14159 按单精度浮点常量处理，编译时不出现“警告”
long double a = 1.23L;         //把此 1.23 作为 long double 型处理
```

**注意**：要区分**类型**与**变量**。

有些读者容易弄不清类型和变量的关系，往往把它们混为一谈。应当看到它们是既有联系又有区别的两个概念。每个变量都属于一个确定的类型，类型是变量的一个重要的属性。变量是占用存储单元的，是具体存在的实体，在其占用的存储单元中可以存放数据。而类型是变量的共性，是抽象的，不占用存储单元，不能用来存放数据。

例如，“大学生”是一个抽象的名词，它代表所有大学生共有的属性(在高等学校学习的、具有正式学籍的学生)，而张方章、李四元、王建则是具体存在的大学生，他们有姓名、家庭、成绩等。可以输出张方章的成绩，但不能输出“大学生”的成绩。同理，可以对一个变量赋值，但不能向一个类型赋值。例如：

```
int a; a=3;                    //正确。对整型变量 a 赋值
```

```
int=3;                          //错误。不能对类型赋值
```

## 3.3 运算符和表达式

几乎每个程序都需要进行运算,对数据进行加工处理,否则程序就没有意义了。要进行运算,就需规定可以使用的运算符。C 语言的运算符范围很宽,把除了控制语句和输入输出以外几乎所有的基本操作都作为运算符处理,例如将赋值符“=”作为赋值运算符、方括号作为下标运算符等。

### 3.3.1 C 运算符

C 语言提供了以下运算符:

(1) 算术运算符 (+ - * / % ++ --)
(2) 关系运算符 (> < == >= <= !=)
(3) 逻辑运算符 (!&& ||)
(4) 位运算符 (<< >> ~ | ^ &)
(5) 赋值运算符 (=及其扩展赋值运算符)
(6) 条件运算符 (?:)
(7) 逗号运算符 (,)
(8) 指针运算符 (* 和 &)
(9) 求字节数运算符 (sizeof)
(10) 强制类型转换运算符 ((类型))
(11) 成员运算符 (.->)
(12) 下标运算符 ([])
(13) 其他 (如函数调用运算符())

本章先介绍算术运算符和赋值运算符,其余的在以后各章中陆续介绍。

### 3.3.2 基本的算术运算符

最常用的算术运算符见表 3.5。

**表 3.5 最常用的算术运算符**

| 运 算 符 | 含 义 | 举 例 | 结 果 |
|---|---|---|---|
| + | 正号运算符(单目运算符) | +a | a 的值 |
| - | 负号运算符(单目运算符) | -a | a 的算术负值 |
| * | 乘法运算符 | a * b | a 和 b 的乘积 |
| / | 除法运算符 | a/b | a 除以 b 的商 |
| % | 求余运算符 | a%b | a 除以 b 的余数 |
| + | 加法运算符 | a+b | a 和 b 的和 |
| - | 减法运算符 | a-b | a 和 b 的差 |

**说明：**

- 由于键盘无×号，运算符×以 * 代替。
- 由于键盘无÷号，运算符÷以/代替。两个实数相除的结果是双精度实数，两个整数相除的结果为整数，如 5/3 的结果值为 1，舍去小数部分。但是，如果除数或被除数中有一个为负值，则舍入的方向取决于编译器，C99 以后，商向 0 取型（余数与被除数符号一致）。例如，－5/3，在支持 C99 及更新标准的编译器里，采取“向零取整”的方法，即 5/3＝1，－5/3＝－1，取整后向 0 靠拢。
- %运算符要求参加运算的运算对象（即操作数）为整数，结果也是整数。如 8%3，结果为 2。与商向 0 靠拢一致，余数的符号与被除数相同。
- 除%以外的运算符的操作数都可以是任何算术类型。

### 3.3.3　自增（＋＋）、自减（－－）运算符

自增（＋＋）、自减（－－）运算符的作用是使变量的值加 1 或减 1，例如：

```
++i,--i  (在使用 i 之前，先使 i 的值加(减)1)
i++,i--  (在使用 i 之后，使 i 的值加(减)1)
```

粗略地看，＋＋i 和 i＋＋的作用相当于 i＝i＋1。但＋＋i 和 i＋＋的不同之处在于：＋＋i 是先执行 i＝i＋1，再使用 i 的值；而 i＋＋是先使用 i 的值，再执行 i＝i＋1。如果 i 的原值等于 3，请分析下面的赋值语句：

```
① j=++i;  (i 的值先变成 4，再赋给 j，j 的值为 4)
② j=i++;  (先将 i 的值 3 赋给 j，j 的值为 3，然后 i 变为 4)
```

又例如：

```
i=3;
printf("%d",++i);
```

输出 4。若改为

```
printf("%d\n",i++);
```

则输出 3。

自增（减）运算符常用于循环语句中，使循环变量自动加 1；也用于指针变量，使指针指向下一个地址。这些将在以后的章节中介绍。

有些专业人员喜欢在使用＋＋或－－运算符时采用一些技巧，但是初学者如果试图模仿，往往会出现意想不到的副作用，例如 i＋＋＋j，是理解为（i＋＋）＋j 还是 i＋（＋＋j）呢？初学时勿过多使用技巧，应当更注意清晰易读，不致引起歧义。建议谨慎使用＋＋和－－运算符，只用最简单的形式，即 i＋＋，i－－。而且把它们作为单独的表达式，而不要在一个复杂的表达式中使用＋＋或－－运算符。

### 3.3.4　算术表达式和运算符的优先级与结合性

用算术运算符和括号将运算对象（也称操作数）连接起来的、符合 C 语法规则的式子称为**C 算术表达式**。运算对象包括常量、变量、函数等。例如，下面是一个合法的 C 算术表达式：

a * b/c−1.5+'a'

C 语言规定了运算符的优先级(如先乘除后加减),还规定了运算符的**结合性**。

在表达式求值时,先按运算符的优先级顺序执行,如表达式 a−b * c,b 的左侧为减号,右侧为乘号,而乘号优先级高于减号,因此,相当于 a−(b * c)。

如果在一个运算对象两侧的运算符的优先级相同,如 a−b+c,则按规定的"结合方向"处理。C 语言规定了各种运算符的结合方向(结合性),算术运算符的结合方向都是"自左至右",即先左后右,因此 b 先与减号结合,执行 a−b 的运算,然后再执行加 c 的运算。"自左至右的结合方向"又称"左结合性",即运算对象先与左面的运算符结合。以后可以看到有些运算符的结合方向为"自右至左",即右结合性(例如,赋值运算符,若有 a=b=c,按从右到左顺序,先把变量 c 的值赋给变量 b,然后把变量 b 的值赋给变量 a)。关于"结合性"的概念在其他一些高级语言中是没有的,是 C 语言的特点之一,希望能弄清楚。附录 C 列出了所有运算符以及它们的优先级别和结合性。

**说明**:不必死记,只要知道:算术运算符是自左至右(左结合性),赋值运算符是自右至左(右结合性),其他复杂的遇到时查一下即可。

### 3.3.5 不同类型数据间的混合运算

在程序中经常会遇到不同类型的数据进行运算,如 5 * 4.5。如果一个运算符两侧的数据类型不同,则先自动进行类型转换,使二者成为同一种类型,然后进行运算。整型、实型、字符型数据之间可以进行混合运算。规律如下。

(1) +、−、*、/运算的两个数中有一个数为 float 或 double 型,结果是 double 型,因为系统将所有 float 型数据都先转换为 double 型,然后进行运算。

(2) 如果 int 型与 float 或 double 型数据进行运算,先把 int 型和 float 型数据转换为 double 型,然后进行运算,结果是 double 型。

(3) 字符(char)型数据与整型数据进行运算,就是把字符的 ASCII 码与整型数据进行运算。如 12+'A',由于字符 A 的 ASCII 码是 65,相当于 12+65,等于 77。如果字符型数据与实型数据进行运算,则将字符的 ASCII 码转换为 double 型数据,然后进行运算。

以上的转换是编译系统自动完成的,用户不必过问。有些读者可能注意到了,自动类型转换会把精度低的数据转换为精度高的类型,从而允许同类型进行运算。自动类型转换是不会造成数据精度损失的。

分析下面的表达式,假设已指定 i 为整型变量,值为 3,f 为 float 型变量,值为 2.5,d 为 double 型变量,值为 7.5。

10+'a'+i * f−d/3

编译时,从左至右扫描,运算次序如下。

① 进行 10+'a'的运算,'a'的值是整数 97,运算结果为 107。

② 由于"*"比"+"优先级高,先进行 i * f 的运算。先将 i 与 f 都转换成 double 型,运算结果为 7.5,double 型。

③ 整数 107 与 i * f 的积相加。先将整数 107 转换成双精度数,相加结果为 114.5,double 型。

④ 进行 d/3 的运算，先将 3 转换成 double 型，d/3 结果为 2.5，double 型。

⑤ 将 10＋'a'＋i＊f 的结果 114.5 与 d/3 的商 2.5 相减，结果为 112.0，double 型。

**【例 3.3】** 给定一个大写字母，要求用小写字母输出。

**解题思路**：前已介绍，字符数据以 ASCII 码存储在内存中，形式与整数的存储形式相同。所以字符型数据和其他算术型数据之间可以互相赋值和运算。

要进行大小写字母之间的转换，就要找到一个字母的大写形式和小写形式之间有什么内在联系。从附录 A 中可以找到其内在规律：同一个字母，用小写表示的字符的 ASCII 码比用大写表示的字符的 ASCII 码大 32。例如字符'a'的 ASCII 码为 97，而'A'的 ASCII 码为 65。将'A'的 ASCII 码加 32，就能得到'a'的 ASCII 码。有此思路就可以编写程序了。

**编写程序：**

```
#include <stdio.h>
int main()
{
  char c1,c2;
  c1='A';                          //将字符'A'的 ASCII 码放到 c1 变量中
  c2=c1+32;                        //得到字符'a'的 ASCII 码，放在 c2 变量中
  printf("%c\n",c2);               //输出 c2 的值，是一个字符
  printf("%d\n",c2);               //输出 c2 的值，是字符'a'的 ASCII 码
  return 0;
}
```

**运行结果：**

```
a
97
```

**程序分析**：程序第 6 行"c2＝c1＋32;"把字符变量 c1 的值(是字符'A'的 ASCII 码)与整数 32 相加。c1＋32 就是'A'＋32，就是 65＋32，其值为 97。将 97 赋给字符变量 c2，在 c2 的存储单元中存放了 97(以二进制形式存储)。

一个字符数据既可以以字符形式输出，也可以以整数形式输出。第 7 行的目的是以字符形式输出 c2，在 printf 函数中指定用"%c"格式，系统会将 c2 变量的值 97 转换成相应字符'a'，然后输出。最后一行的目的是以 ASCII 码(十进制整数)形式输出 c2 的值，故指定用"%d"输出格式，得到 97，见图 3.13。

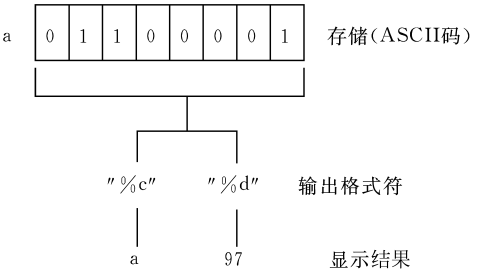


图　3.13

### 3.3.6　强制类型转换运算符

可以利用强制类型转换运算符将一个表达式转换成所需类型。例如：

```
(double)a        (将 a 转换成 double 型)
(int)(x+y)       (将 x+y 的值转换成 int 型)
(float)(5%3)     (将 5%3 的值转换成 float 型)
```

其一般形式为

**(类型名)(表达式)**

注意,如果写成

```
(int)x+y
```

则只将 x 转换成整型,然后与 y 相加。如果要将 x+y 的值转换为整型,则表达式应该用括号括起来(x+y)。

需要说明的是,在强制类型转换时,得到一个所需类型的中间数据,而原来变量的类型未发生变化。例如:

```
a=(int)x
```

如果已定义 x 为 float 型变量,a 为整型变量,进行强制类型运算(int)x 后得到一个 int 类型的临时值,它的值等于 x 的整数部分,把它赋给 a,注意 x 的值和类型都未变化,仍为 float 型。该临时值在赋值后就不再存在了。

由上可知,有两种类型转换。一种是在运算时不必用户干预,系统自动进行的类型转换,如 3+6.5。另一种是强制类型转换。当自动类型转换不能实现目的时,可以用强制类型转换。如%运算符要求其两侧均为整型量,若 x 为 float 型,则 x%3 不合法,必须用(int)x%3。从附录 C 可以查到,强制类型转换运算优先于%运算,因此先进行(int)x 的运算,得到一个整型的中间变量,然后再对 3 求余。此外,在函数调用时,有时为了使实参与形参类型一致,可以用强制类型转换运算符得到一个所需类型的参数。强制类型转换是可能造成数据精度损失的,例如(int)3.5 等于 3,转换后的结果丢掉了小数点后面的部分。

## 3.4 C 语 句

### 3.4.1 C 语句的作用和分类

在前面的例子中可以看到,一个函数包含**声明部分**和**执行部分**,执行部分是由语句组成的,语句的作用是向计算机系统发出操作指令,要求执行相应的操作。一个 C 语句经过编译后产生若干条机器指令。声明部分不是语句,它不产生机器指令,只是对有关数据的声明。

C 程序结构可以用图 3.14 表示。即一个 C 程序可以由若干源程序文件(编译时以文件模块为单位)组成,一个源文件可以由若干函数和预处理指令以及全局变量声明部分组成(关于"全局变量"见第 7 章)。一个函数由数据声明部分和执行语句组成。

C 语句分为以下 5 类。

(1) **控制语句**。控制语句用于完成一定的控制功能。C 语言只有 9 种控制语句,它们的形式如下。

① **if**()…**else**…　　(条件语句)

② **for**()…　　(循环语句)

③ **while**()…　　(循环语句)