第3章　基本数据类型、运算符与表达式


微课视频

◇ 学习意义

学习C语言的最终目的就是能编写程序来解决实际问题，那么什么是程序呢？从功能上来讲，程序是解决某种问题的一组指令的有序集合；从结构上来讲，一个程序应包括对数据的描述和对数据处理的描述。著名计算机科学家沃思(Nikiklaus Wirth)提出一个公式：**程序＝数据结构＋算法**，这里的数据结构其实就是指对数据的描述。数据结构是计算机学科的核心课程之一，有许多专门著作论述，本书就不再赘述。在C语言中，系统提供的数据结构是以数据类型的形式体现出来的。而算法其实就是对数据处理的描述，算法是为解决一个问题而采取的方法和步骤，是程序的灵魂。

因此，要学好C语言并用C语言来编写程序，首先必须十分了解和熟练掌握C语言中的数据类型描述以及运算符与表达式，这是学习C语言的重要基础，后续章节的内容从某种意义上来讲都是在此基础上展开的。

◇ 学习目标

(1) 掌握变量和常量的概念；

(2) 理解各种类型的数据在内存中的存放形式；

(3) 掌握各种类型数据的常量的使用方法；

(4) 掌握各种整型、字符型、浮点型变量的定义和引用方法；

(5) 了解调用printf函数输出各种类型数据的方法；

(6) 掌握数据类型转换的规则以及强制数据类型转换的方法；

(7) 掌握赋值运算符、算术运算符、位运算符、逗号运算符、sizeof运算符的使用方法；

(8) 理解运算符的优先级和结合性的概念，记住所学的各种运算符的优先级关系和结合性。

◇ 难点提示

(1) 数据在内存中的表示；

(2) 有符号数与无符号数；

(3) 数据类型的自动转换与强制类型转换。

微课视频

3.1 C 语言的数据类型

C 语言程序在处理数据之前，要求数据必须具有明确的数据类型。因此，C 语言是一种强类型语言。所谓数据类型是按被说明量的性质、表示形式、占据存储空间的多少及构造特点来划分的。在 C 语言中，数据类型可分为基本数据类型、构造数据类型、指针类型、空类型四大类。

1. 基本数据类型

基本数据类型最主要的特点是其值不可以再分解为其他类型。也就是说，基本数据类型是自我说明的。在 C 语言中，基本数据类型主要有整型（短整型、长整型）、字符型、实型（单精度实型、双精度实型）三种。这是本章讨论的重点。

2. 构造数据类型

构造数据类型也叫复杂数据类型，是根据已定义的一个或多个数据类型用构造的方法来定义的。也就是说，一个构造数据类型的值可以分解成若干个“成员”或“元素”。每个“成员”都是一个基本数据类型或又是一个构造数据类型。在 C 语言中，构造数据类型主要有：数组类型、结构类型、联合体类型（共用体类型）、位域、枚举类型五种。关于构造数据类型将在第 7 章和第 11 章进行详细讨论。

3. 指针类型

指针是一种特殊的同时又是具有重要作用的数据类型，其值用来表示某个量在内存中的地址。虽然指针变量的取值类似于整型量，但这是两个类型完全不同的量，因此不能混为一谈。关于指针类型将在第 9 章进行详细讨论。

4. 空类型

空类型是从语法完整性的角度给出的一种数据类型，表示该处不需要具体的数据值，因而没有数据类型。其类型说明符为 void。关于空类型将在第 8 章进行详细介绍。

图 3-1 为 C 语言数据类型层次图。

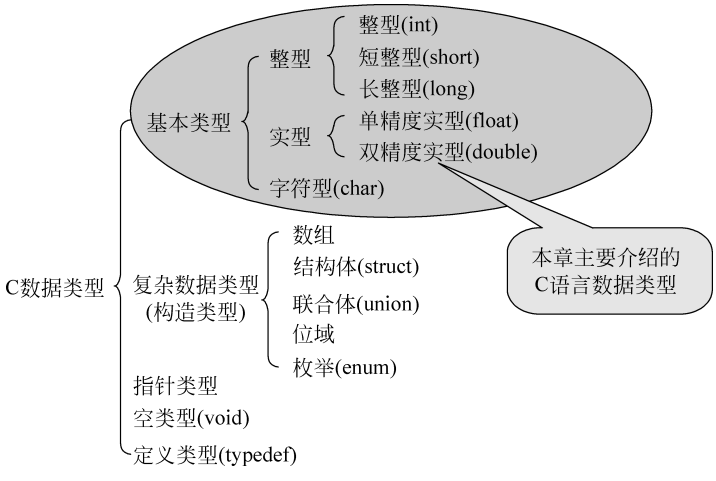


图 3-1　C 语言数据类型层次图

微课视频

3.2 常量、变量和标识符

C语言中存在着两种表征数据的形式：常量和变量。常量用来表示数据的值，变量不仅可以用来表示数据的值，而且可以用来存放数据。因为变量对应着一定的内存单元。但变量甚至常量以及后面要介绍的函数常常需要一个名字(即标识符)来表示才能使用，所以首先介绍标识符及其命名规则。

1. 标识符

1) 标识符的定义

用来标识变量、常量、函数等的字符序列。

2) 标识符的命名规则

(1) 有效字符：只能由字母、数字、下画线组成，且第一个字母必须是字母或下画线。

(2) 有效长度：随系统而异，在 TC 2.0 及 BC 3.1 中，变量名(标识符)的有效长度为1～32个字符，默认值为32，但在 VC 6.0、VC 2010、CB 17.12 中其长度可达到255。

(3) C语言的关键字(又叫保留字)不能用作变量名(关键字见附录C)。

(4) C语言对英文字母的大小写敏感，即同一字母的大小写，被认为是两个不同的字符。

【思考题 3-1】 在C语言中，变量名 total 与变量名 TOTAL、ToTaL、tOtAl 等是同一个变量吗？

3) 标识符的命名习惯

(1) 变量名和函数名中的英文字母一般用小写，以增加可读性。

(2) 见名知意，即通过变量名就知道变量值的含义。通常应选择能表示数据含义的英文单词(或缩写)作为变量名，或汉语拼音字头作为变量名。例如，name/xm(姓名)、sex/xb(性别)、age/nl(年龄)、salary/gz(工资)。

(3) 字符不易混淆。如数字 1 与字母 l、数字 0 与字母 O 等。

【思考题 3-2】 判断下列标识符的合法性。

```
sum   Sum   M.D.John   day   Date   3days   student_name
#33   lotus_1_2_3   char   a>b   _above   $123   main
```

2. 常量

1) 常量的定义

程序运行时其值不能改变的量称为常量(即常数)。

2) 常量的分类

(1) 直接常量：又称值常量，主要包括整型常量(如 10，15，－10，－30)、实型常量(如12.5，30.0，－1.5)、字符常量(如'A'，'b'，'c')、字符串常量(如"sum"，"A"，"123")。

（2）符号常量：用标识符来代表常量。定义一个符号常量需要使用一条预处理命令#define，其定义格式为：

```
#define    符号常量    常量
```

例如：

```
#define  NUM   20
#define  PI    3.1415926
```

有了上面的两行文本，NUM 的值就是 20，PI 的值就是 3.1415926。定义一个符号常量，实际上就是为一个值常量起个名字。关于预处理命令#define 的详细用法见第 10 章。

【例 3-1】　符号常量举例。

```
1   #include<stdio.h>
2   #define  PRICE  30
3
4   int main( )
5   {
6     int num, total;
7
8     num=10;
9     total=num * PRICE;      //PRICE 是符号常量，它代表常量 30
10    printf("total=%d", total);
11    return 0;
12  }
```

运行结果：

```
total=300
```

⚠注意：

- 在用#define 定义符号常量时，行尾一般不能有分号，除非特意这样做。
- define 前面的#标志着 define 是一个预处理命令而不是 C 语句。
- 符号常量名最好使用大写字符来命名。
- 符号常量名最好有意义，这样可提高程序的可读性。

使用符号常量的好处如下：

- 含义清楚。在例 3-1 的程序中，从字面上就可知道 PRICE 代表价格。因此定义符号常量名时应考虑“见名知意”。
- 在需要改变一个常量值时能做到“一改全改”。例如，在一个程序中多处用到物品的价格，若价格用常数表示，则在价格调整时，就需要在程序中进行多处修改，很不方便，且易出错。若用符号常量 PRICE 代表价格，只需改动一处即可。如例 3-1 中的第 2 行，如果将 30 改为 50，则在程序中所有以 PRICE 代表的价格就会一律自动改为 50。

3. 变量

1) 变量的定义

在程序运行过程中,其值可以被改变的量称为变量。

2) 变量的两个要素

(1) 变量名:变量的名字。变量命名遵循标识符命名规则。

(2) 变量值:变量存储在内存中的值。在程序中,通过变量名来引用变量的值。

3) 变量的定义与初始化

在C语言中,要求对所有用到的变量,必须先定义、后使用;在定义变量的同时进行赋初值的操作称为变量初始化。

(1) 变量定义的一般格式:

```
[存储类型]    数据类型    变量名1[,变量名2,变量名3,…,变量名n];
```

格式说明如下:

- 存储类型是指所定义的变量占用内存空间的方式,也称为存储方式。详细情况将在第8章进行介绍。[ ]表示该项可以省略。
- 数据类型是指所定义变量的数据类型,它决定变量所占内存空间的大小。数据类型可以是基本数据类型(如int、float、char等),也可是复杂数据类型(或构造数据类型)或指针等,后面的章节将会一一介绍。
- 变量名必须是合法的标识符,定义多个变量时变量名之间用逗号(,)分隔。
- 存储类型、数据类型及变量名之间至少有一个空格。
- 最后必须以分号(;)结尾。

例如:

```
int x, y, z;                    //定义3个整型变量x、y、z
float radius, length, area;     //定义3个单精度浮点型变量radius、length、area
```

(2) 变量初始化的一般格式:

```
[存储类型]    数据类型    变量名1[=初值1][,变量名2[=初值2] …];
```

例如:

```
int x=2, y=3, z=4;              //定义3个整型变量x、y、z,并分别赋初始值2、3、4
float radius=2.5, length=4;     //定义2个单精度浮点型变量radius、length,并分别赋初始值2.5、4
```

⚠注意:

- 变量和符号常量必须先定义后引用,否则就会出错。
- 变量定义以后,计算机会在程序运行时给该变量分配一定大小的内存单元,具体大小随变量定义的数据类型而定,后面的内容中会详细介绍。
- 变量定义的位置一般放在函数开头。
- 对变量进行赋值实际上就是将数据写到变量所对应的内存单元中。

3.3 简单数据类型与表示范围

通过 3.2 节的学习，已经知道对于基本数据类型量来说，根据其取值是否可改变分为常量和变量两种。但如果与数据类型结合起来分类，则又可分为整型常量、整型变量、浮点常量、浮点变量、字符常量、字符变量等。在本节中，将重点讨论 C 语言中的简单数据类型所表示的常量、变量以及变量在内存中的表示和数据表示范围。


微课视频

3.3.1 整型数据

1. 整型常量

整型常量即整常数，在 C 语言中整型常量可用三种形式表示。

1）十进制整型常量

由数字 0～9 和正负号表示。以下各数是合法的十进制整型常量：237，－568，65535，1627；以下各数不是合法的十进制整型常量：023（不能有前导 0），23D（含有非十进制数码）。

2）八进制整型常量

由数字 0 开头，后跟数字 0～7 来表示。以下各数是合法的八进制整型常量：015（十进制为 13），0101（十进制为 65），－012（十进制为－10）；以下各数不是合法的八进制整型常量：256（无前缀 0），03A2（包含非八进制数码），O127（前缀不能是字母 O）。

3）十六进制整型常量

由 0x 或 0X 开头，后跟 0～9，a～f 或 A～F 来表示。以下各数是合法的十六进制整型常量：0X2A（十进制为 42），－0xA0（十进制为－160），0XFFFF（十进制为 65535）；以下各数不是合法的十六进制整型常量：5A（无前缀 0X 或 0x），0X3H（含有非十六进制数码）。

有了上面三种整型常量表示方法，我们可以这样定义整数的符号常量：

```
#define    NUM1    20      //十进制数
#define    NUM2    020     //八进制数
#define    NUM3    0x2a    //十六进制数
```

其中，符号常量 NUM1 的值是十进制 20，符号常量 NUM2 的值是十进制 16，符号常量 NUM3 的值是十进制 42。

【思考题 3-3】 下列哪些整型常量是合法的？

012　oX7A　00　078　0x5Ac　－0xFFFF　0034　7B

注意：

- 八进制整数和十六进制整数都是以数字 0 开头的，不要写成字母 O 或 o。
- 定义十六进制整数时，0x 或 0X 后面的 a～f 或 A～F 大小写可以交错。例如，0x6Ac 和 0XFFbC 都是合法的。

2. 整数在内存中的表示

通过第 1 章的学习,读者已经了解到一个实际的数(即真值)在计算机中以二进制形式存放,其机内表示形式(即机器数)通常有三种:原码、补码和反码。根据它们的运算规则,补码运算最为简单,可连同符号位一起参与运算,也就是说,对于补码运算,符号位和数据位一起只看成一个二进制数值。所以计算机系统有这样的规定:**整数的数值在内存中以补码的形式存放**。

如何求得某个整数所对应的补码的方法在第 1 章中已经介绍过,下面用一种比较简单的办法来求得一个整数的补码表示(假设用 n 个二进制位的内存单元来表示它):

- 如果是正整数,采用符号-绝对值表示,即最高有效位(符号位)为 0 表示正,数的其余部分则表示数的绝对值。
- 如果是负整数,则先写出与该负数相对应的正数的补码表示,然后将其按位求反,最后在末位(最低位)加 1。
- 然后将上述求得的补码的低 n 位存放于内存单元之中,就得到了该整数在内存中的表示,内存单元的最高位是符号位(0 表示正,1 表示负)。

在 TC 2.0 或 BC 3.1 下,一个整数默认情况下需要 2 字节(16 位)的内存单元存放;而在 VC 6.0、VC 2010 或 CB 下,则需要 4 字节(32 位)。下面来具体看一看几种不同整数在内存中的存放形式。

1) +14

对于 16 位的内存单元来说,$(14)_{补}$ = 0000 0000 0000 1110,最前面的 0 为符号位,表示是正数,其表示形式如图 3-2 所示。

图 3-2 十进制数+14 的 2 字节补码表示形式

其在内存中的实际存放形式如图 3-3 所示。

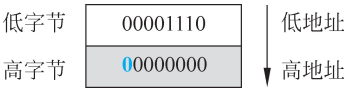


图 3-3 十进制数+14 的 2 字节内存实际存放形式

对于 32 位的内存单元来说,$(14)_{补}$ = 0000 0000 0000 0000 0000 0000 0000 1110,其表示形式如图 3-4 所示。

其在内存中的实际存放形式如图 3-5 所示。

⚠**记住**:数据在内存中的存放位置是高字节放在高地址的存储单元中,低字节放在低地址的存储单元中。

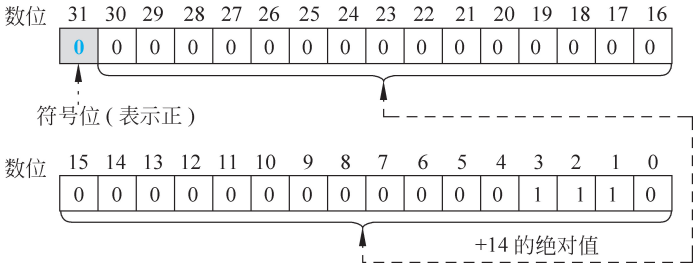


图 3-4　十进制数＋14 的 4 字节补码表示形式

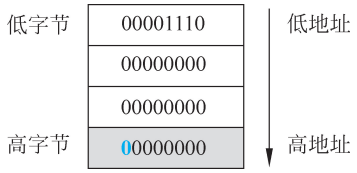


图 3-5　十进制数＋14 的 4 字节内存实际存放形式

2）－14

对于 16 位的内存单元来说，先计算$(+14)_{补}$＝ 0000 0000 0000 1110，然后按位求反，末位加 1 得$(-14)_{补}$＝ 1111 1111 1111 0010，最前面的 1 为符号位，表示是负数，其表示形式如图 3-6 所示。

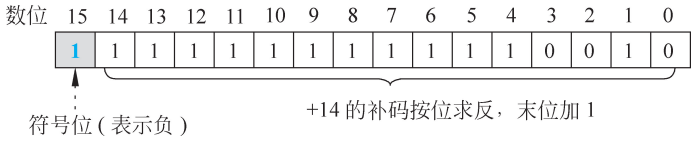


图 3-6　十进制数－14 的 2 字节补码表示形式

其在内存中的实际存放形式可参考图 3-3。

对于 32 位的内存单元来说，同样先计算$(+14)_{补}$＝ 0000 0000 0000 0000 0000 0000 0000 1110，然后按位求反，末位加 1 得$(-14)_{补}$＝ 1111 1111 1111 1111 1111 1111 1111 0010，其表示形式如图 3-7 所示，其在内存中的实际存放形式同样可参考图 3-5。

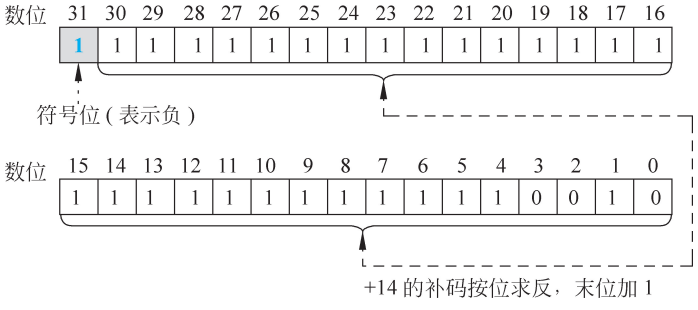


图 3-7　十进制数－14 的 4 字节补码表示形式

3) −65537

对于16位的内存单元来说,先计算$(+65537)_{补}$ = 01 0000 0000 0000 0001,然后按位求反,末位加1得$(-65537)_{补}$ = 10 1111 1111 1111 1111。再将$(-65537)_{补}$的低16位存放于内存之中,所以−65537在内存中实际存放的值为:1111 1111 1111 1111,最前面的1为符号位,表示是负数,其真值为−1,而不是−65537。也就是说,−65537在内存中的值与−1在内存中的值是一样的,这一点读者务必注意!−65537在内存中的实际存放形式如图3-8所示。

图3-8 十进制数−65537的2字节内存实际存放形式

对于32位的内存单元来说,同样先计算$(+65537)_{补}$ = 0000 0000 0000 0001 0000 0000 0000 0001,然后按位求反,末位加1得$(-65537)_{补}$ = 1111 1111 1111 1110 1111 1111 1111 1111。最前面的1为符号位,表示是负数,其真值为−65537,其在内存中的实际存放形式如图3-9所示。

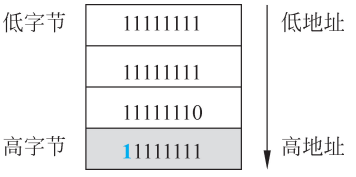


图3-9 十进制数−65537的4字节内存实际存放形式

为什么−65537这个数在16位内存单元中的表示与在32位内存单元中的表示不相同呢?这主要是因为−65537这个数超出了16位内存单元表示数的范围,也就是发生了数据溢出,所以实际存储的值(−1)与要表示的值(−65537)不同,但−32767并没有超出32位内存单元表示数的范围,所以实际存储的值就是其本身。因此,在C语言中对数据处理时必须要注意不同大小内存单元的数据表示范围,以免引起不必要的错误。

【思考题3-4】 请问−0和0在内存中如何表示?如果一个整数用2字节的存储单元存放,问−1和65535在内存中表示的是同一个数吗?−65535在内存中又如何表示呢?

3. 整型变量

1) 整型变量的定义

定义变量的基本格式是“数据类型标识符 变量名;”。整型数据类型标识符是int,因此,定义整型变量的基本格式为:

```
int 变量名1[=初值1][,变量名2[=初值2],…,变量名n[=初值n]];
```

格式说明如下:

- 整型类型名 int 必须小写。
- int 与变量名之间至少要用一个空格分开。
- int 后面可以一次定义多个变量，但变量名之间必须以逗号(,)隔开。
- 可以在变量定义时就对变量赋初值，具体方法是在变量名后面增加“=数值”。
- []表示该项可以省略。
- 最后必须以分号(;)结尾。

例如：

```
int   a;                //定义整型变量 a
int   x, y, z;          //定义 3 个整型变量 x、y、z
int   m=2, n=-3;        //定义 2 个整型变量 m、n,并对 m、n 分别赋初始值 2 和-3
```

当程序中定义了一个变量后，计算机会为这个变量分配一个相应大小的内存单元。因此，这个变量是有值的，它的值就是对应内存单元的值。但这个值程序员是无法预知的。

2) 整型变量的分类

整型变量的基本类型符是 int。C 语言允许程序员在定义整型变量时，在 int 的前面增加两类修饰符：一类控制变量是否有符号，这类修饰符包括 signed(有符号)和 unsigned(无符号)；另一类控制整型变量的值域范围，这类修饰符包括 short 和 long。例如，可以这样定义一个无符号的长整型变量 a：

```
unsigned long int a;
```

unsigned 和 long 都是数据类型修饰符。如果定义变量时，既不指定 signed，也不指定 unsigned，则默认为 signed(有符号)。实际上，signed 修饰符完全可以不写。这样一来就形成了六种整型变量。

(1) 有符号整型(int 或 signed int)。

数据类型符为 signed int，一般直接写成 int，占用字节数由具体的 C 编译系统自行决定，一般占一个机器字。在 TC 2.0 或 BC 3.1 下，这种变量占用 2 字节(16 位)的内存单元，在 VC 6.0、VC 2010 或 CB 17.12 下，这种变量则占用 4 字节(32 位)的内存单元。当该变量所对应的内存表示形式的最高位(即符号位)为 1 时，变量的值是负数，否则就是正数。

例如，**int a=-2**；(定义一个有符号整型变量 a，并赋初值-2)，其内存存放形式如图 3-10 所示(假设是在 VC 6.0、VC 2010 或 CB 17.12 下定义的)。

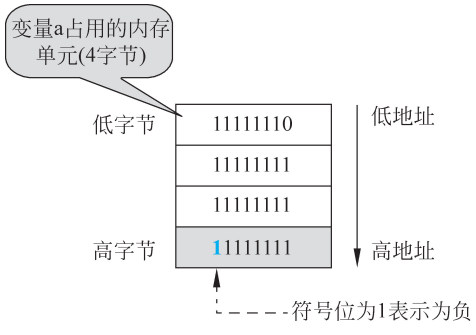


图 3-10　有符号整型变量 a 在内存中的表示形式

(2) 无符号整型(unsigned int 或 unsigned)。

数据类型符为 unsigned int,也可写成 unsigned,占用字节数同 int 类型。但该变量所对应的内存表示形式的最高位不是符号位,而是数据位,参与数值计算。因此,unsigned int 型变量的值是非负数。

例如,**unsigned int b=2;**(定义一个无符号整型变量 b,并赋初值 2),它与 **int b=2;** 在内存中的表示形式是完全相同的,其值均为 2。但对于 **unsigned int b=-2;** 来说,其在内存中的表示形式与图 3-10 是相同的,可 b 的值不是负数-2(因为最高位 1 不是符号位,而是数据位),而是 4294967294(即 $2^{32}-2$),是很大的一个数。

⚠注意:对于有符号数和无符号数,其实在计算机内存中的表示是不加区分的(但运算时有区别,见 3.6 节),都是以其补码形式表示,只是我们怎样看待最高二进制位的问题,如果把最高位当成符号位看待,则为有符号数,如果把最高位当成数据位看待,则变为无符号数。例如:

```
unsigned int a=-2;
printf("%d",a);          //有符号输出,则为-2
printf("%u",a);          //无符号输出,4294967294
```

(3) 有符号短整型(short int 或 short)。

数据类型符为 short int,也可写成 short,占用 2 字节(16 位)的内存单元,最高位为符号位。

例如,**short int a=-2;**(定义一个有符号短整型变量 a,并赋初值-2),其内存存放形式如图 3-11 所示。

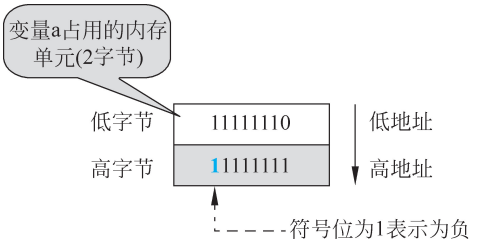


图 3-11 有符号短整型变量 a 在内存中的表示形式

(4) 无符号短整型(unsigned short int 或 unsigned short)。

数据类型符为 unsigned short int,也可写成 unsigned short,占用 2 字节(16 位)的内存单元(同 short 类型),最高位为数据位。

例如,**unsigned short b=-2;**(定义一个无符号短整型变量 b,并赋初值-2),其在内存中的表示形式与图 3-11 是相同的,可 b 的值不是负数-2(因为最高位 1 不是符号位,而是数据位),而是 65534(即 $2^{16}-2$),是很大的一个数。

(5) 有符号长整型(long int 或 long)。

数据类型符为 long int,也可写成 long,占用 4 字节(32 位)的内存单元,最高位为符号位。在 VC 6.0、VC 2010 及 CB 17.12 中 long 与 int 类型基本相同,数据处理方法基本一样。

下面给出 long int 类型变量定义的范例:

```
long int x=0xFF00FFCD;
long y=215678;
```

(6) 无符号长整型(unsigned long int 或 unsigned long)。

数据类型符为 unsigned long int,也可写成 unsigned long,占用 4 字节(32 位)的内存单元。但最高位不是符号位,而是数据位。unsigned long 类型与 long 类型的区别可参考 unsigned int 与 int 类型的区别。

下面给出 unsigned long int 类型变量定义的范例:

```
unsigned long int x=4389828;
unsigned long y=0XA5CD99AB;
```

在标准 C 语言程序中,变量的定义必须放在函数的声明部分。下面的例子说明了实际程序中如何定义整型变量(行号不属于程序)。读者只需看懂有阴影的部分即可。

【例 3-2】　各种整型变量的定义。

```
1   #include  <stdio.h>          //文件包含,头文件说明
2
3   #define SUM  65535           //定义符号常量 SUM,值为 65535
4
5   int main( )
6   {
7     int a, b=20;               //定义两个整型变量 a 和 b,b 赋初值 20
8     unsigned int c=0xff;       //定义无符号整型变量 c,并赋初值 0xff
9     short int d;               //定义有符号短整型变量 d
10    long e;                    //定义有符号长整型变量 e
11
12    a=SUM;                     //对 a 赋值为 SUM,这时 a 的值是 65535
13    d=SUM;                     //对 d 赋值为 SUM,这时 d 的值是 65535
14    e=301;                     //对 e 赋值为 301
15
16    printf("a=%d\n", a);       //以有符号十进制形式("%d")显示 a 的值
17    printf("b=%d\n", b);       //以有符号十进制形式("%d")显示 b 的值
18    printf("c=%d\n", c);       //以有符号十进制形式("%d")显示 c 的值
19    printf("d=%d\n", d);       //以有符号十进制形式("%d")显示 d 的值
20    printf("e=%d\n", e);       //以有符号十进制形式("%d")显示 e 的值
21    return 0;
22  }
```

运行结果:

```
a=65535
b=20
c=255
d=-1
e=301
```

对于16位的有符号短整型变量d来说，因65535在内存中的形式为1111111111111111，最高位为1表示负，则其所对应的十进制数就为-1

4. 整型常量的分类

整型变量有六种类型,那么整型常量是否也有不同的类型呢?C 语言根据整型常量的值来决定整型常量的类型,具体规定如下:

(1) 在 VC 6.0、VC 2010 或 CB 17.12 下,C 语言认为整型常量是 int 型常量。

(2) 整型常量后加字母 l 或 L,认为它是 long int 型常量,如 123L、45l、0XAFL。

(3) 无符号数也可用后缀表示,整型常数的无符号数的后缀为 U、u。例如,358u、0x38Au、235Lu 均为无符号数。前缀、后缀可同时使用以表示各种类型的数,如 0XA5Lu 表示十六进制无符号长整数 A5,其十进制为 165。


微课视频

■ 3.3.2 实型数据

1. 实型常量

实型也称为浮点型。实型常量也称为实数或者浮点数。在 C 语言中,实数只采用十进制。它有两种形式:十进制小数形式和十进制指数形式。

(1) 十进制小数形式:由数码 0～9 和小数点组成。例如,0.0、.25、5.789、0.13、5.0、300.、－267.8230 等均为合法的实数。

(2) 十进制指数形式:由十进制数,加阶码标志 e 或 E 以及阶码(只能为整数,可以带符号)组成。其一般形式为 a E n,a 为十进制数,n 为十进制整数,都不可缺少。其可表示为 $a\times10^n$,例如,2.1E5 表示 2.1×10^5,3.7E－2 表示 3.7×10^{-2}。以下不是合法的实数:345(无小数点)、E7(阶码标志 E 之前无数字)、－5(无阶码标志)、53.－E3(负号位置不对)、2.7E(无阶码)。

2. 实数在内存中的存放形式

计算机中的实数是按指数形式存放的。通常一个实数需要 4 字节(32 位)的内存,计算机将这 32 位分成三部分:最高位是尾数的符号位,也就是整个实数的符号位,剩下的 31 位一部分存放实数的尾数部分,另一部分用于存放实数的阶码部分。但到底用多少位来存放尾数,多少位来存放阶码,C 语言没有严格规定,由各自的 C 语言编译器来决定。但按照 IEEE 标准,常用的浮点数的格式见表 3-1。

表 3-1 常用浮点数的格式

	符 号 位	阶 码	尾 数	总 位 数
短浮点数	1	8	23	32
长浮点数	1	11	52	64
临时浮点数	1	15	64	80

下面以短浮点数为例来看看浮点数 68.75 在内存中的存放形式。

$68.75=0.6875\times10^2$,则尾数 $a=(0.6875)_{10}=(0.10110000000000000000000)_2$,阶码 $n=(2)_{10}=(00000010)_2$,其存放形式如图 3-12 所示。

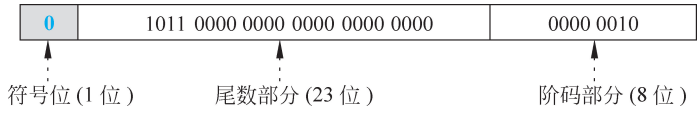


图 3-12　68.75 的存放形式

⚠注意：对于浮点数，小数部分占的位数越多，它所能表示的精度越高，阶码部分占的位数越多，它所能表示的值越大。

3. 实型变量的分类和定义

实型变量的定义格式同整型变量一样，只是数据类型符不同。实型变量有三类：单精度实型、双精度实型和长双精度实型。

1）单精度实型

数据类型符为 float，在 VC 6.0、VC 2010 或 CB 17.12 下，这种变量在内存中均占 4 字节（32 位）的存储单元。例如：

```
float f=3.14, g;        //定义了 2 个单精度浮点型变量 f 和 g,并给 f 赋初始值 3.14
```

2）双精度实型

数据类型符为 double，在 VC 6.0、VC 2010 或 CB 17.12 下，这种变量在内存中均占 8 字节（64 位）的存储单元。例如：

```
double x, y;            //定义了 2 个双精度浮点型变量 x 和 y
```

3）长双精度实型

数据类型符为 long double，在 VC 6.0、VC 2010 下，这种变量在内存中占 8 字节（64 位），与 double 形同；在 CB 17.12 下，这种变量在内存中占 12 字节（96 位）。例如：

```
long double x, y;       //定义了 2 个长双精度浮点型变量 x 和 y
```

⚠注意：

- 三种实数类型中，其精度是 float＜double＜long double。
- long float 实际上就是 double，因此，没有 long float 类型。
- 所有的实型常量按照 double 类型处理。

4. 实型数据的精度

对于一个十进制实数，不一定能用二进制数精确地表示它，这显然就存在某些数位的舍去问题，当然舍去的数位越低越好，这样实数的精度也就越高，但上述三种浮点类型所表示实数的精度是不相同的，表 3-2 列出了它们各自所能精确表示的数字个数（只限于 VC 和 CB）。

表 3-2　几种实型数据所能精确表示的数字个数

类　　型	精确表示的数字个数
float	7～8
double	16～17
long double	17～18

【例 3-3】　实型变量的精度。

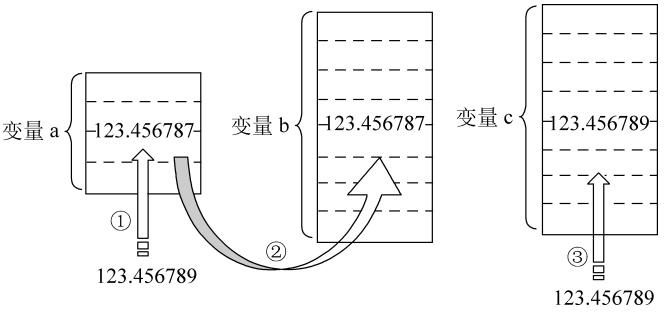
```
1    #include <stdio.h>
2
3    int main( )
4    {
5       float a;                  //定义 float 型变量 a
6       double b, c;              //定义 double 型变量 b 和 c
7
8       a=123.456789;             //对变量 a 赋值为 123.456789
9       b=a;                      //将变量 a 赋给变量 b
10      c=123.456789;             //对变量 c 赋值为 123.456789
11      printf("a=%f   b=%lf   c=%lf\n", a, b, c);
12      return 0;
13   }
```

运行结果：

```
a=123.456787      b=123.456787      c=123.456789
```

程序解释：

- 第 11 行调用 printf 函数以 float 形式(%f)显示变量 a 的值，以 double 形式(%lf)显示变量 b 和 c 的值。
- 由于 float 型变量最多只能精确表示 8 个数字，因此显示 a 的值时，只能有效显示前面 8 个数字即 123.45678，最后追加一位数字 7 是随机的。对于 double 型变量 c 来说，最多能精确表示 17 个数字，所以它能精确地输出 123.456789。那对于变量 b，同样是 double 型变量，为什么输出与 float 型变量 a 相同呢？下面通过图 3-13 看看它们如何赋值以及在内存中的存储情况。



注释：① a=123.456789；② b=a；③ c=123.456789

图 3-13　浮点型变量的赋值及在内存中的存放数据


微课视频

3.3.3　字符型数据和字符串常量

对于一般的数学计算而言，有了整型和实型数据就可以了。但计算机除了数值计算的功能以外，还应当具备对信息（包括文本信息）的处理功能。那么文本信息在计算机中是如何表示和存储的呢？要了解这个问题就必须了解字符型数据，字符型数据就是那些用来表征英文字母、符号、汉字的数据。

字符型数据实际上就是整型数据。但它只占用 1 字节（8 位）的内存单元，用于存放该字符所对应的 ASCII 码的值（参见附录 E）。当然也可把这 1 字节的内存单元用于存放一般的数据，这就和前面介绍的整型数据没多大差别，只是数据表示的范围小一些而已。

1. 字符型常量

字符型常量有以下两种表示方法。

（1）用单引号括起来的一个直接输入的字符。

直接输入的字符是指那些通过键盘输入的字符，如 a、b、C、D、＋、]、8、9 等，可以用单引号括起来表示其常量。例如，'A'、'a'、'3'、'＋'等都是合法的字符常量。

⚠**注意**：字符常量只能用单引号括起来，不能用双引号或其他括号；字符'4'和数值 4 是不相同的，字符'4'的值是其 ASCII 值 0X34。

（2）使用转义字符。

对于那些无法直接输入的字符以及某些特殊字符，需要用单引号括起来的转义字符来表示。转义字符是一种特殊的字符常量。转义字符以反斜线“\”开头，后跟一个或几个字符。转义字符具有特定的含义，不同于字符原有的意义，故称“转义”字符。例如，在前面各例题 printf 函数的格式串中用到的'\n'就是一个转义字符，其意义是“回车换行”。转义字符主要用来表示那些用一般字符不便于表示的控制代码。

⚠**注意**：尽管反斜杠、单引号和双引号都可以直接输入，但如果反斜杠、单引号和双引号本身作为字符常量，则必须使用转义字符：'\\'、'\"'、'\"'，因为转义字符用到了反斜杠，字符常量用到了单引号，字符串用到了双引号。

常用的转义字符及其含义见表 3-3。

表 3-3　常用的转义字符

转义字符	含　义	ASCII 码值
\n	回车换行符。显示该字符时，光标移到下一行的行首	10
\r	回车符。显示该字符时，光标移到当前行的行首	13
\t	制表符。显示该字符时，光标向右移动一个制表位	9
\v	竖向跳格	11
\b	退格	8
\f	走纸换页	12

续表

转义字符	含　义	ASCII码值
\a	鸣铃	7
\\	反斜杠符'\'	92
\'	单引号符	39
\"	双引号符	34
\ddd	1～3位八进制数所代表的字符,d的值可以是0～7的任何数字	
\xhh	1～2位十六进制数所代表的字符,h的值可以是0～f的任何字符	

广义地讲,C语言字符集中的任何一个字符均可用转义字符来表示。表3-3中的\ddd和\xhh正是为此而提出的。ddd和hh分别为八进制和十六进制的ASCII代码。如'\101'的值是65,即$(101)_8$,不是十进制101,表示'A'。'\134'表示反斜线,'\X0A'表示换行等。另外,反斜杠后面跟单个字符(除0～7及特殊控制符外,如a、x)表示的就是反斜杠后面的字符,如'\G'表示的就是'G','\9'表示的就是'9'。

【例3-4】　转义字符的应用。

```
#include <stdio.h>

int main()
{
    printf("\101 \x42 C\n");
    printf("I say:\"How are you?\"\n");
    printf("\\C Program\\\n");
    printf("Visual \'C\'\n");
    printf("Y\b=\n");
    return 0;
}
```

运行结果:

```
A B C
I say:"How are you?"
\C Program\
Visual 'C'
=
```

2. 字符型变量

字符型数据类型符是char(英文character的缩写)。在内存中占1字节(8位),由于字符型数据也可以参与运算,因此,C语言将字符型数据分为有符号字符和无符号字符。默认情况下为有符号字符。符号位为该字节的最高位。

无符号字符的数据类型符是unsigned char。下面给出定义字符型变量范例。

```
char          ch;                  //定义有符号字符型变量ch
unsigned char C='B';//定义无符号字符变量C,并赋初始值'B'(实际为其ASCII码值0X42)
```

⚠**记住**：short、int、long 及 char 型变量其实都是整型变量，它们均存在有符号和无符号之分。只是所占内存大小不同而已：short 是 2 字节整型，int 和 long 是 4 字节整型，char 是 1 字节整型。

3. 字符串常量

字符串常量是由一对双引号括起来的字符序列。例如，"CHINA"、"C program"、"$12.5"等都是合法的字符串常量。

字符串常量和字符常量是不同的量。它们之间主要有以下区别。

（1）字符常量由单引号括起来，字符串常量由双引号括起来。

（2）字符常量只能是单个字符，字符串常量则可以含一个或多个字符。

（3）可以把一个字符常量赋予一个字符变量，但不能把一个字符串常量赋予一个字符变量。**在 C 语言中没有相应的字符串变量**，但是可以用一个字符数组来存放一个字符串常量，这在数组一章会进行介绍。

（4）字符常量占 1 字节的内存空间。字符串常量占的内存字节数等于字符串中字节数加 1。增加的一个字节中存放字符'\0'（ASCII 码为 0），这是字符串结束的标志。例如，字符串"HELLO"在内存中占 6 字节，其存放形式如图 3-14 所示。

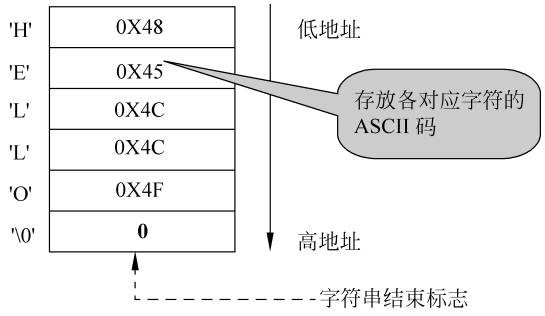


图 3-14　字符串"HELLO"在内存中的存放形式

❓【思考题 3-5】　请问字符常量'A'与字符串常量"A"在内存中的表示有何不同？只包含一个空格字符的字符串常量" "与不包含任何字符的字符串常量""两者之间又有何不同？

■ 3.3.4　简单数据类型的表示范围

微课视频

前面已经介绍了 C 语言中的基本数据类型，这是 C 语言对数据处理的基础，也是广大读者必须熟练掌握的基本内容。C 语言中，不同数据类型的变量在内存中所占存储单元的大小也不完全相同，但不管怎样，它们所占的存储单元是有限的，因此在一定的存储单元内表示数据绝对不是无限的，这就涉及在一定存储空间内数据的表示范围的问题，也就是 C 语言中的每种数据类型都有一定的数据表示范围，这一点是许多 C 语言学习者极易忽视但又非常重要的内容。千万不要以为用 C 语言的基本数据类型来处理数据要多大就有多大，

不了解这一点,编写的程序有时候会出现意想不到的结果。下面先来看一个简单的C语言程序例子,运行的结果是否有点意外呢?

【例 3-5】 变量的存储范围。

```
#include <stdio.h>

int main()
{
    char ch;
    int x;

    ch=80+50;
    x=80+50;
    printf("ch=%d\n", ch);
    printf("x=%d\n", x);
    return 0;
}
```

运行结果:

```
ch=-126
x=130
```

为什么会出现两个正数相加,得出的却是一个负数呢?这就是不同数据类型所表示的数值范围不同引起的。为了理解C语言中不同数据类型所表示的范围,以一个16位short类型为例来加以讨论。其他的数据类型可采用同样的方法得到。

对于一个16位的short型的数据,不管它是有符号(signed)还是无符号(unsigned),在内存中都是以其二进制数补码的形式存放的,当最高位(第15位)被看作符号位时它就表示有符号数,0表示+,1表示-;当最高位(第15位)被看作数据位时它就表示无符号数。图3-15以图示的形式来说明其表示范围。

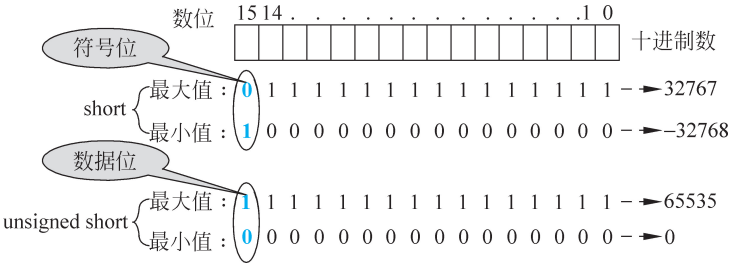


图 3-15 16位整型数所表示的数据范围

因此,16位short(有符号)型变量所表示的数据范围为-32768~32767,而unsigned short(无符号)型变量所表示的数据范围为0~65535。其他类型的数据可根据其所占内存单元的大小以同样的方法求得。表3-4给出了C语言基本数据类型的表示范围。

表 3-4　基本数据类型的表示范围

类型	符号	关　键　字	占字节数	数的表示范围
整型	有	(signed)　int　在 16 位系统下	2	−32768～32767
		(signed)　　在 32 位系统下	4	−2147483648～2147483647
		(signed)　short	2	−32768～32767
		(signed)　long	4	−2147483648～2147483647
	无	unsigned　int 在 16 位系统下	2	0～65535
		unsigned　int 在 32 位系统下	4	0～4294967295
		unsigned　short	2	0～65535
		unsigned　long	4	0～4294967295
实型	有	float	4	0 以及绝对值 1.2×10^{-38}～3.4×10^{38}
	有	double	8	0 以及绝对值 2.3×10^{-308}～1.7×10^{308}
	有	long double	8	0 以及绝对值 2.3×10^{-308}～1.7×10^{308}
			12	0 以及绝对值 3.4×10^{-4932}～1.2×10^{4932}
字符型	有	char	1	−128～127
	无	unsigned　char	1	0～255

下面再回过头来看看例 3-5 中变量 ch 的输出值为什么是−126 呢？其实只要了解 ch 的数据表示范围及数据的计算过程就会明白，如图 3-16 所示。

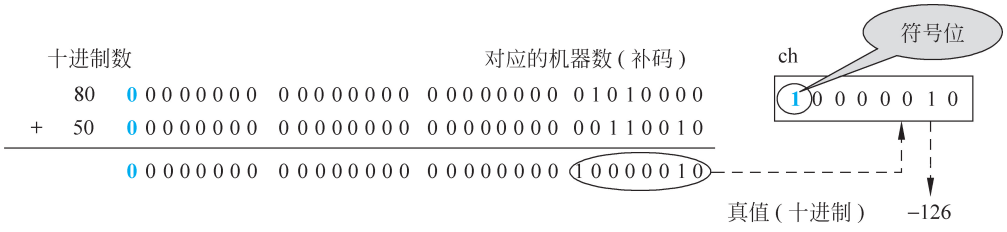


图 3-16　变量 ch 的内存数据处理过程

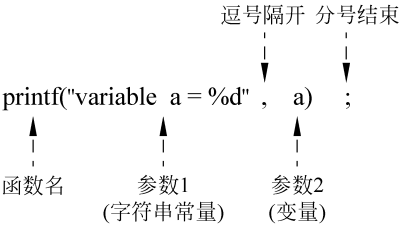
3.3.5　数据的简单输出


微课视频

C 语言中没有用于输出的语句，只能通过标准库函数的调用来完成数据的输出任务。库函数的一般调用格式为：

```
函数名(参数 1,参数 2,…,参数 n);
```

C 语言的标准库函数很多，这里只介绍格式化输出函数 printf 的基本用法，其他的库函数会在后面的章节中陆续介绍。printf 函数的功能是在计算机屏幕上以文本的方式格式化输出程序运行的结果，它可以带多个参数，这里只介绍一个参数和两个参数的用法。printf 函数的第一个参数必须是字符串常量，第二个参数可以是某个变量。例如：



如果调用 printf 函数时只提供第一个参数，printf 函数就将第一个参数的字符串显示输出。例如，下面一行的执行结果显示：How are you!。

```
printf("How are you!");
```

也就是说，printf 函数将第一个参数的字符串原样显示了。如果要显示：variable a=100，那么可这样调用 printf 函数：

```
int a=100;
printf("variable a=%d",a);
```

其中，"variable a="会原样输出，%d 的位置上会显示 a 的值。实际上，%d 是一种格式控制字符，它不能原样显示，在它的位置上会显示第二个参数的值。数据输出的常用格式控制字符主要有以下几个：

%d：用于显示有符号整型数据，如 int、short 型数据。

%u：用于显示无符号整型数据，如 unsigned int、unsigned short 型数据。

%f：用于显示实型数据，如 float 型数据。

%c：用于显示字符型数据，如 char 型数据。

%s：用于显示字符串数据。

【例 3-6】 数据的简单输出。

```
#include <stdio.h>

int main()
{
    int a, b;
    unsigned short c;
    unsigned int u;
    long d, e;
    char ch;
    float f;

    a=200;
    b=-1;           //b 的值为-1,实际存储为-1 的补码,即 0xffffffff
    c=b;            //c 的值为 b 的低 16 位,即 0xffff
    u=b;            //u 的值为 0xffffffff
    d=c;            //d 的值为 0x0000ffff
    e=u;            //e 的值为 0xffffffff
    ch='A';         //ch 的值为'A'的 ASCII 码,即十进制 65
    f=32.17;

    printf("a=%d\t", a);
```

```
22      printf("b=%d\t", b);
23      printf("c=%d\n", c);
24      printf("u=%u\t", u);
25      printf("d=%ld\t", d);
26      printf("e=%ld\n", e);
27      printf("f=%f\n", f);
28      printf("ch is %c and value is %d\n", ch, ch);
29      printf("I    love C language!\rYou\n");      //I后有三个空格
30      return 0;
31  }
```

运行结果：

```
a=200   b=-1   c=65535
u=4294967295   d=65535   e=-1
f=32.169998
ch is A and value is 65
You love C language!
```

程序解释：

- 第21、22、24、25行的printf函数调用没有输出\n，因此，光标没有换行，所以第23行的printf的输出与第21、22行的printf的输出位于同一行上。同样，第26行的printf的输出与第24、25行的printf的输出位于同一行上。
- \n表示回车换行，它的作用是使得光标回到当前行的下一行的行首。\t表示制表符。
- b占4字节(32位)内存单元，赋值为-1，其32位补码为0xffffffff；当它赋值给c时(第14行)，由于c是无符号短整型，占2字节(16位)，因此，只能将b的低16位赋值给c，因此c实际存放的值为0xffff。第23行将c的值以4字节有符号十进制整型形式输出(%d)，因此c以4字节整型表示为0x0000ffff(因c为无符号，高位补0)，其对应的十进制整数为65535。
- 当b赋值给u时(第15行)，由于u是无符号整型，占4字节(32位)，因此u实际存放的值为0xffffffff，第24行将u的值以无符号十进制整型形式输出(%u)，故输出4294967295。但要注意，一定要用%u。
- d、e是有符号长整型，均占4字节内存单元，当把c赋值给d时(第16行)，d的高2字节为全0(因c无符号)，故d的值为0x0000ffff，第25行再将d的值以有符号十进制长整型形式输出(%ld)，即为65535。
- 当把u赋值给e时(第17行)，e实际存放的值为0xffffffff(-1的补码)，第26行再将e的值以有符号十进制长整型形式输出(%ld)，即为-1。
- ch是字符型变量，可当成一个字节的整型变量看待，第28行当用%d输出字符时，输出的是该字符'A'的ASCII码值65。
- 为什么没有输出“I love C language!”呢？因为第29行的printf函数的第一个参数中有个回车符\r。当显示到\r时，光标又回到了行首，后面的“You”继续显示，覆盖

了“I”以及后面的两个空格。

【思考题 3-6】 如果将例 3-6 的第 23 行中的%d 变为%hd 或%u，问 c 的输出分别是多少呢？


微课视频

3.4 C 语言的运算符与表达式

变量用来存放数据，运算符则用来处理数据。用运算符将变量和常量连接起来的符合 C 语言语法规则的式子称为表达式。每个表达式都有值。

根据运算符所带的操作数的数量进行划分，C 语言中的运算符有以下三种类别：

(1) 单目运算符：只带一个操作数的运算符，如++、--运算符。

(2) 双目运算符：带两个操作数的运算符，如+、-运算符。

(3) 三目运算符：带三个操作数的运算符，如？运算符。

C 语言中运算符和表达式数量之多，在高级语言中是少见的。正是丰富的运算符和表达式使 C 语言功能十分完善，这也是 C 语言的主要特点之一。所以要学好和用好 C 语言，务必熟练掌握其运算符的功能、特点及应用。具体学习过程中应把握如下几方面：

(1) 运算符的功能：该运算符主要用于做什么运算。

(2) 与运算量的关系：要求运算量的个数及运算量的类型。

(3) 运算符的优先级：表达式中包含多个不同运算符时运算符运算的先后次序。

(4) 运算符的结合性：同级别运算符的运算顺序(指左结合性还是右结合性)。

(5) 运算结果的类型：表达式运算后最终所得到的值的类型。

下面学习 C 语言中常用运算符的用法。


微课视频

3.4.1 赋值运算符、赋值表达式

1. 赋值运算符

赋值符号“=”就是赋值运算符，它的作用是将一个表达式的值赋给一个变量，实际上是将特定的值写到变量所对应的内存单元中。赋值运算符是双目运算符，因为“=”两边都要有操作数。“=”左边是待赋值的变量，“=”右边是要赋的值。

赋值运算符的一般格式为：

```
变量= 常量或变量或表达式
```

例如：

```
int x, y, z;
x=20;                    //将 20 赋值给变量 x
y=x;                     //将 x 的值赋值给变量 y
z=x+y;                   //将 x 的值加上 y 的值,其和数赋值给变量 z
```

2. 赋值表达式

由赋值运算符或复合赋值运算符(后面即将介绍),将一个变量和一个表达式连接起来的表达式称为赋值表达式。

(1) 赋值表达式的一般格式为:

```
变量　(复合)赋值运算符 表达式
```

(2) 赋值表达式的值。

任何一个表达式都有一个值,赋值表达式也不例外。**被赋值变量的值,就是赋值表达式的值**。例如,a=5 这个赋值表达式,变量 a 的值 5 就是它的值。

3. 赋值语句

按照 C 语言规定,任何表达式在其末尾加上分号就构成为语句。同样,对于赋值表达式来说,在其后面加分号就构成了赋值语句。因此如 **x=8; a=b=c=5;** 都是赋值语句,在前面各例中已大量使用过了。

4. 赋值运算符及赋值表达式的使用

1) 多个变量连续赋值

例如,a=b=c=10。**连续赋值的表达式的运算顺序是从右向左(又称为右结合性)**。其相当于表达式 a=(b=(c=10)),即先对 c 赋值,得到赋值表达式 c=10 的值为 10,然后将该表达式的值 10 赋值给 b,得到表达式 b=c=10 的值 10,最后才对 a 赋值,得到赋值表达式 a=b=c=10 的值为 10。

2) 赋值表达式的嵌套

例如,a=(b=2)+(c=3)。其相当于表达式 a=((b=2)+(c=3)),因为"+"的优先级高于"="的优先级,故不能等同于(a=(b=2))+(c=3)。它将首先对 b 赋值为 2,得赋值表达式 b=2 的值为 2,再对 c 赋值为 3,得赋值表达式 c=3 的值为 3,再将两个赋值表达式的值相加得 5,然后将 5 赋值给 a,最终表达式的值为 5。

⚠**注意:**

- 赋值语句中"="左边必须是变量名或对应某特定内存单元的表达式(后面的章节会遇到这样的表达式),不能是常量或其他表达式。
  例如,**30=a; b+2=5;** 都是错误的。
- 赋值语句中的"="表示赋值,不是代数中相等的意思。要表示相等的意思则应用关系运算符"=="表示,二者切勿混淆!

3.4.2　强制类型转换符

微课视频

C 语言的数据类型是可以相互转换的。转换的方法有两种,一种是自动转换,一种是强

制转换。

1. 自动转换

前面讨论的赋值语句其特点是“=”左右两边的数据类型均是相同的,但是如果“=”左右两边的数据类型不相同,那C语言又如何处理呢?例如,**int a=2.5;**则a的值将是2,而不是2.5。因为C语言首先将“=”右边表达式值的数据类型转换成“=”左边的变量的数据类型,然后再赋值给“=”左边的变量。这种自动改变“=”右边表达式值的数据类型的操作称为数据类型的自动转换。

当然,当“=”左右两边的数据类型是整型或字符型,但两边的数据类型的长度不同时(例如,将int型变量赋值给char变量),C语言也要进行数据类型的自动转换,其自动转换的规则如下。

1)短长度的数据类型→长长度的数据类型

(1)无符号短长度的数据类型→长长度的数据类型。

直接将无符号短长度的数据类型的数据作为长长度的数据类型数据的低位部分,长长度的数据类型数据的高位部分补零,其示意图如图3-17所示。

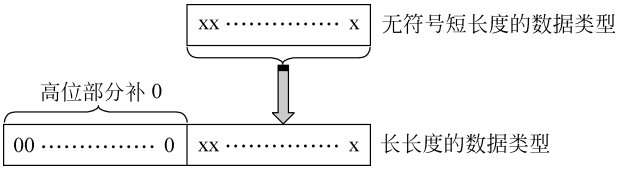


图3-17 无符号短长度的数据类型→长长度的数据类型

例如:

```
unsigned char ch=0xfc;
unsigned short a=0xff00;
short b;
unsigned long u;
b=ch;                    //b的值将是0x00fc
u=a;                     //u的值将是0x0000ff00
```

【思考题3-7】 如果将ch的值赋值为-4,问b的值又是多少呢?

(2)有符号短长度的数据类型→长长度的数据类型。

直接将有符号短长度的数据类型的数据作为长长度的数据类型数据的低位部分,然后将低位部分的最高位(即有符号短长度数据的符号位)向长长度的数据类型数据的高位部分扩展,其示意图如图3-18所示。

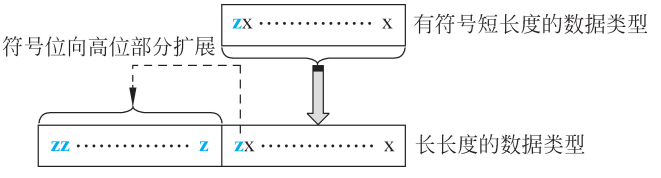


图3-18 有符号短长度的数据类型→长长度的数据类型

例如：

```
char ch=2;
short a=-2;          //a 的值将是-2 的补码,即 0xfffe
short b;
unsigned long u;
b=ch;                //b 的值将是 2
u=a;                 //u 的值将是 0xfffffffe
```

2）长长度的数据类型→短长度的数据类型

直接截取长长度的数据类型数据的低位部分(长度为短长度的数据类型的长度)作为短长度数据类型的数据,其示意图如图 3-19 所示。

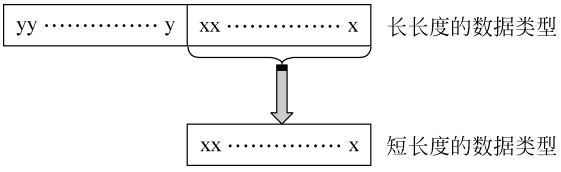


图 3-19　长长度的数据类型→短长度的数据类型

例如：

```
short a=-32768;      //a 的值将是-32768 的补码,即 0x8000
unsigned long b=0xffffaa00;
char ch;
short c;
ch=a;                //ch 的值将是 0
c=b;                 //c 的值将是 0xaa00
```

3）长度相同的数据类型转换

数据按照原样复制即可。

例如：

```
int a=0xffff0000;
unsigned int b=a;    //b 的值将是 0xffff0000
```

2. 强制转换

自动转换的规则其实也适用于强制类型转换。有时候想有意识地改变某个表达式的数据类型,就需要强制类型转换。强制类型转换是通过类型转换运算来实现的。其一般形式为：

```
(类型说明符)(表达式)
```

其功能是把表达式的运算结果强制转换成类型说明符所表示的数据类型。其中,类型说明符是强制类型转换符,它的优先级比较高。例如：

```
float x=3.5, y=2.1;
long a=0xffffa032;
```

这时,表达式(int)x是把x转换为int类型,它的值是3,(int)(x+y)把x+y的结果转换为int类型,它的值是5,而(char)a是把a转换为char类型,它的值是0x32。

⚠注意:在使用强制转换时应注意以下问题。

- 类型说明符和表达式都必须加括号(单个变量可以不加括号)。
  例如,把(int)(x+y)写成(int)x+y则是将x转换成int型之后再与y相加。
- 无论是强制转换还是自动转换,都只是为了本次运算的需要而对变量的数据类型进行的临时性转换,而不改变数据说明时对该变量定义的类型。
  例如,(double)a只是将变量a的值转换成一个double型的中间量,其数据类型并未转换成double型。


微课视频

3.4.3 算术运算符、算术表达式

1. 算术运算符

C语言提供的算术运算符包括五种:加(+)、减(-)、乘(*)、除(/)和取余(%)。它们均是双目运算符。+、-、*、/运算符既可用于整型数据的算术运算,又可用于实型数据的算术运算。而%只能用于整数的运算。

⚠注意:

- C语言规定:两个整数相除,其商为整数,小数部分被舍弃。
  例如,5/2的值是2,不是2.5。要得到2.5,则应写成5.0/2或5/2.0。
- %不能用于浮点型数据,否则会出错。
  例如,5.4%2是非法的,因为%只能用于整型数据的运算。

2. 表达式和算术表达式

1) 表达式的概念

用运算符和括号将运算对象(常量、变量和函数等)连接起来的、符合C语言语法规则的式子,称为表达式。

单个常量、变量或函数,可以看作表达式的一种特例。将单个常量、变量或函数构成的表达式称为简单表达式,其他表达式称为复杂表达式。

2) 算术表达式的概念

如果表达式中的运算符都是算术运算符,则此表达式称为算术表达式。例如,3+6*9、(x+y)/2-1等都是算术表达式。

当一个表达式中存在多个算术运算符时,各个运算符的优先级与常规算术运算相同,即先乘、除和取余,再计算加、减,同级运算符的计算顺序是从左向右,即先计算左边的算术表达式,再进行右边的表达式的计算。当然也可以用圆括号改变表达式计算的先后顺序。

例如，图 3-20 详细地表示了表达式 10＋5＊4－7/3 的求解过程，图中的①～④表示求解过程的先后顺序。

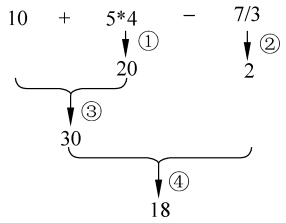


图 3-20　表达式 10＋5＊4－7/3 的求解过程

算术运算符、赋值运算符和类型强制转换运算符的优先级的关系如下：**类型强制转换运算符的优先级＞算术运算符的优先级＞赋值运算符的优先级**。因此，执行下面的语句后，a 的值是 13。

```
int a;
a=(int) 2.5 * 4+5;
```

【思考题 3-8】 执行语句 a=(int)(2.5 * 4)+5 后，问 a 的值又是多少呢？

C 语言中，任何数据类型的数据都有其固定的取值范围。当表达式的值超出了取值范围时，就会丢失数据，这种现象称为数据溢出。例如，两个正数相加，得到的结果比实际的值小，甚至得到一个负数。例 3-5 中的 ch 变量的值即为此例。

算术运算符中，比较容易引起溢出的是乘法运算符。但 C 语言并不对溢出进行检查，这个任务由程序员承担。因此，编写程序时，要特别注意算术表达式的值，使其不要产生数据溢出。一种解决的办法可使用较长数据类型的变量来存放数据。例如，在例 3-5 中的 ch 变量如果用 **int ch;** 来定义，则结果就是 130，不会发生数据溢出现象。

3.4.4　自增自减运算符、负号运算符


微课视频

C 语言中，减号(－)既是一个算术运算符，又是一个负号运算符。负号运算符是单目运算符。例如，a＝2，那么－a 的值就是－2。负号运算符的优先级比较高，与强制类型转换符是同一个级别。

C 语言还提供了另外两个用于算术运算的单目运算符：自增(＋＋)和自减(－－)。

1. 用法

自增运算(＋＋)使单个变量的值增 1，自减运算(－－)使单个变量的值减 1。

2. 用法与运算规则

自增、自减运算符都有以下两种用法。

(1) 前置运算——运算符放在变量之前：＋＋变量、－－变量。先使变量的值增(或减)1，然后再以变化后的值参与其他运算，即先增减、后运算。

(2) 后置运算——运算符放在变量之后：变量＋＋、变量－－。变量先参与其他运算，然后再使变量的值增(或减)1，即先运算、后增减。

【例 3-7】 自增、自减运算符的用法与运算规则。

```
1    #include <stdio.h>
2
3    int main( )
4    {
```

```
5       int a=2, b=4;
6       int c, d;
7
8       c=a++;      //等价于 c=a; 和 a=a+1; 两条语句
9       d=--b;      //等价于 b=b - 1; 和 d=b; 两条语句
10      printf("a=%d, b=%d\n", a, b);
11      printf("c=%d, d=%d\n", c, d);
12      return 0;
13  }
```

运行结果：

```
a=3,  b=3
c=2,  d=3
```

⚠注意：

- ++和--运算符只能用于变量，不能用于常量和表达式。因为++和--蕴含着赋值操作。如5++、--(a+b)都是非法的表达式。
- 负号运算符、++、--和强制类型转换运算符的优先级相同，当这些运算符连用时，按照从右向左的顺序计算，即具有右结合性。
- 两个+和两个-之间不能有空格。
- 在表达式中，连续使同一变量进行自增或自减运算时，很容易出错，所以最好避免这种用法。如++i++是非法的。
- 自增、自减运算常用于循环语句中，使循环控制变量加(或减)1，以及指针变量中，使指针指向下(或上)一个地址。

假设在VC 6.0或VC 2010下有定义int p, i=2, j=3；现分几种情况来讨论一下++和--的使用方法。

- 情况一：p=-i++；相当于p=-(i++)；即先把-i的值赋给p，然后i增1。执行后p的值为-2，i的值为3。
- 情况二：p=i+++j；相当于p=(i++)+j；即先将i+j的值赋给p，然后i增1。执行后p的值为5，i的值为3，j的值不变。
- 情况三：p=i+--j；相当于p=i+(--j)；即先将j减1，然后把i+j的值赋给p。执行后p的值为4，i的值不变，j的值为2。
- 情况四：p=i+++--j；相当于p=(i++)+(--j)；即先将j减1，然后把i+j的值赋给p，再把i的值增1。执行后p的值为4，i的值为3，j的值为2。
- 情况五：p=i+++i++；相当于p=(i++)+(i++)；即先将i+i的值赋给p，再把i的值增1两次。执行后p的值为4，i的值为4。
- 情况六：p=++i+(++i)；相当于p=(++i)+(++i)；即先将i的值增1两次。然后把i+i的值赋给p，执行后p的值为8，i的值为4。注意：该式不能写成p=++i+++i。

⚠注意：

不同C语言编译系统对++、--运算的解释是有差别的，所以含有++、--运算的算术表达式在不同的编译环境下其运行结果可能有差异。像情况五：p=i+++i++；在VC 6.0及VC 2010下运行p的值为4，i的值为4。但在CB 17.12下运行p的值为5，i的值为4，其解释为表达式第一个i++，则先取i的值2，然后执行i增1操作，i的值变为3，第二个i++，此时i的值为3，故p的值为2+3=5，然后执行i增1操作，i的值最后也是4。

？【思考题 3-9】　p=++i+(++i)能否写成p=++i+++i？能否写成p=++i++++i？p=++i--++i和p=++i+--i合法吗？

■ 3.4.5　算术运算中数据类型转换规则

微课视频

在C语言中，整型、实型和字符型数据间可以混合运算（因为字符数据与整型数据可以通用）。如果一个运算符两侧的操作数的数据类型不同，则系统按**“先转换、后运算”**的原则，首先将数据自动转换成同一类型，然后在同一类型数据间进行运算。转换规则如图3-21所示。

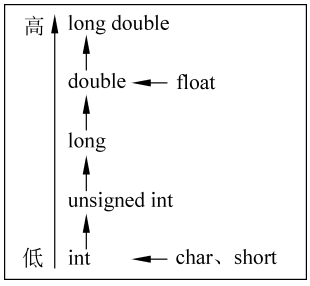


图 3-21　数据类型转换规则

图中横向向左的箭头，表示必需的转换。char和short型必须转换成int型，float型必须转换成double型。

纵向向上的箭头，表示不同类型的转换方向。例如，int型与double型数据进行混合运算，则先将int型数据转换成double型，然后在两个同类型的数据间进行运算，结果为double型。

⚠注意：图3-21中箭头方向只表示数据类型由低向高转换，不要理解为int型先转换成unsigned型，再转换成long型，最后转换成double型。

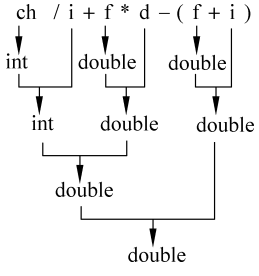


图 3-22　计算表达式ch/i+f*d-(f+i)的数据类型转换示意图

下面以一个实例来看看算术表达式中不同数据类型间的转换。

假设变量的定义为：

```
char ch;
int i;
float f;
double d;
```

现计算ch/i+f*d-(f+i)的值，则其类型转换如图3-22所示。

【例 3-8】　不同类型数据间的算术运算。

```
#include <stdio.h>

int main( )
{
    float a, b, c;

    a=7/2;          //计算 7/2 得 int 型值 3,因此 a 的值为 3.0
    b=7/2*1.0;      //计算 7/2 得 int 型值 3,再与 1.0 相乘,因此 b 的值为 3.0
    c=1.0*7/2;      //先计算 1.0*7 得 double 型的结果 7.0,然后再计算 7.0/2,因此 c 的值是 3.5
    printf("a=%f, b=%f, c=%f", a, b, c);
    return 0;
}
```

运行结果：

```
a=3.000000,  b=3.000000,  c=3.500000
```

微课视频

3.4.6 位运算符、位运算表达式

C语言可以通过位运算符对二进制数进行按位运算，这些位运算符包括：按位与(&)、按位或(|)、按位取反(~)、按位异或(^)、左移(<<)、右移(>>)六种。除~是单目运算符外，其余均是双目运算符。位运算符只能作用于整型数据(包括int、short、long和char型)，它们的具体含义和操作细节在第1章中已经介绍过了，读者需记住的是这些位运算符的符号。下面主要介绍一下移位运算符。

1. 左移运算

左移运算(<<)实现将某变量所对应的二进制数往左移位，溢出的最高位被丢掉，空出的低位用零填补。其一般格式为：

```
返回整型值的表达式 << 返回整型值的表达式
```

例如，int a=3；则表达式 a<<2 将 a 所对应的二进制数左移两位，该表达式的值为12。而表达式 2<<a 将 2 所对应的二进制数左移三位(a的值)，该表达式的值为16。

2. 右移运算

右移运算(>>)实现将某变量所对应的二进制数往右移位，溢出的最低位被丢掉，如果变量是无符号数，空出的高位用零填补，如果变量是有符号数，空出的高位用原来的符号位填补(即负数填1，正数填0)。其一般格式为：

```
返回整型值的表达式 >> 返回整型值的表达式
```

例如，int a=8；则表达式 a >> 2 将 a 所对应的二进制数右移两位，该表达式的值为 2。

【例 3-9】　将 short 类型数据的高、低位字节互换。

```
1    #include <stdio.h>
2
3    int main( )
4    {
5        short a=0xf245, b, c;
6
7        b=a << 8;          //将 a 的低 8 位移到高 8 位赋值给 b,b 的值为 0x4500
8        c=a >> 8;          //将 a 的高 8 位移到低 8 位赋值给 c,c 的值为 0xfff2
9        c=c & 0x00ff;      //将 c 的高 8 位清 0 后赋值给 c,c 的值为 0x00f2
10       a=b+c;             //将 b 和 c 的值相加赋值给 a,a 的值为 0x45f2
11       printf("a=%x", a);
12       return 0;
13   }
```

运行结果：

```
a=45f2
```

程序说明：

该程序的第 8 行和第 9 行也可改为一行，即改为 c=(unsigned short)a >> 8；同样也行，因为 a 为有符号数，加了强制类型转换符(unsigned short)后就把 a 当成无符号数实现右移，则高 8 位补 0，c 的值就为 0x00f2。

位运算符之间不是等优先级的。它们的优先级关系为：

按位取反(~) >移位(<<,>>) >按位与(&) > 按位异或(^) >按位或(|)

⚠注意：

- 每左移一位相当于“<<”左边的值乘以 2，每右移一位相当于“>>”左边的值除以 2。
- 如同表达式 a+2 不能改变 a 的值一样，a << 2 或 a&0x00ff 也不能改变 a 的值。
- 移位运算符的两个<和>中间不能有空格。
- 移位运算符<<和>>两边必须都是整型数，否则非法。如 a << 2.0 是错误的。

■ 3.4.7　逗号运算符、逗号表达式

微课视频

C 语言提供了一种特殊的运算符：逗号运算符(,)。逗号运算符可以将多个表达式连接起来，用逗号连接起来的表达式称为逗号表达式。逗号表达式的一般形式为：

```
表达式 1,表达式 2,…,表达式 n
```

例如：a+3，b=4，b++。

逗号运算符的优先级是最低的,并且具有左结合性。逗号表达式的求值顺序是从左向右依次计算用逗号分隔的各表达式的值,最后一个表达式的值就是整个逗号表达式的值。

下面通过几个例子来说明逗号运算符的应用:

(1) a=4, b=a+5, b++的值为 9。

(2) a=4, b=a+5,++b 的值为 10。

(3) x=a=3, 6 * a 表达式的值为 18,x 的值为 3。

对于逗号表达式还要说明以下几点:

(1) 程序中使用逗号表达式,通常是要分别求逗号表达式内各表达式的值,并不一定要求整个逗号表达式的值。

(2) 并不是在所有出现逗号的地方都组成逗号表达式。如在变量说明中,函数参数表中的逗号只是用作各变量之间的间隔符。

(3) 逗号表达式在 C 语言程序中用途较少,通常只用在 for 循环语句中。

微课视频

3.4.8 sizeof 运算符、复合赋值运算符

1. sizeof 运算符

C 语言中提供了一个能获取变量和数据类型所占内存大小(字节数)的运算符:sizeof。其使用格式为:

```
sizeof 表达式
sizeof(数据类型名或表达式)
```

例如,sizeof(short)的值是 2,sizeof(int)、sizeof(long)的值是 4,sizeof(10L)的值也是 4。又如,如果有 unsigned long a=2,那么 sizeof(a)的值是 4。

sizeof 运算符的优先级比较高,与++、--是同一个优先级。

⚠注意: sizeof 仅提供后面表达式或数据类型所占内存字节数,并不完成对表达式的计算。例如:

```
int a=2, b=3, c;
c=sizeof(b=a++);
printf("a=%d, b=%d, c=%d\n", a, b, c); //输出:a=2, b=3, c=4,a、b 的值没变化
```

2. 复合赋值运算符

C 语言除了提供赋值运算符“=”以外,还提供了各种复合赋值运算符。将算术运算符、位运算符与赋值运算符组合在一起就构成了复合赋值运算符。复合赋值运算符既包含算术运算或位运算,又包含赋值操作。

复合赋值运算符具体有如下几种:+=、-=、*=、/=、%=、&=、|=、^=、<<=、>>=。

其含义为：

```
exp1 op=exp2
```

等价于：

```
exp1=exp1 op exp2
```

例如：

a+=3 等价于 a=a+3。

x *=y+8 等价于 x=x*(y+8)。

x &=y=3 等价于 y=3 和 x=x & y。

复合赋值运算符与赋值运算符是同一个优先级，具有右结合性。


微课视频

3.5 运算符的优先级和结合性

C 语言中不同的运算符具有不同的优先级。在计算表达式的值时，先操作优先级比较高的运算符，如果表达式中运算符的优先级相同，还要按照运算符的结合性确定计算的先后次序。完整的运算符的优先级及结合性请参见附录 D。表 3-5 只是列出本章所介绍的运算符的优先级和结合性(通常单目运算符比双目运算符的优先级高)。

表 3-5　运算符的优先级及其结合性

优先级	运　算　符	需要操作数的个数	结合性
高	( )		从左向右
↑	～　++　--　-(负号运算符)　sizeof　(类型)	1(单目运算符)	从右向左
	*　/　%	2(双目运算符)	从左向右
	+　-(减法)	2(双目运算符)	从左向右
	<<　>>	2(双目运算符)	从左向右
	&	2(双目运算符)	从左向右
	^	2(双目运算符)	从左向右
	|	2(双目运算符)	从左向右
	=　+=　-=　*=　/=　%= >>=　<<=　&=　^=　|=	2(双目运算符)	从右向左
低	,		从左向右

有了优先级的知识，就可以准确地判断表达式 0XF0F0 & 0X1010+0X0A0A <<5/2 的值。该表达式等价于(0XF0F0 & ((0X1010+0X0A0A)<<(5/2)))，因为/的优先级最高，所以先计算 5/2 的值是 2，表达式变为 0XF0F0 & 0X1010+0X0A0A << 2，+的优先级比 & 和<<高，因此先计算 0X1010+0X0A0A 的值是 0X1A1A，此时表达式变为 0XF0F0 & 0X1A1A << 2，<<的优先级比 & 高，因此先计算 0X1A1A << 2 的值是 0X6868，最后计算 0XF0F0 & 0X6868 的值是 0X6060。

微课视频

3.6 有符号数与无符号数之间的运算问题

尽管在C语言中整数分为有符号数与无符号数,但其实它们都是以其补码的形式存储在内存中的,从内存表示形式上二者并无差异。例如:

```
short   a=-2;
unsigned short   b=-2;
```

变量a和变量b在内存中的二进制补码表示均为:1111111 11111110,如果要输出变量a或b的值,此时与变量a、b的有符号还是无符号定义无关,关键在于如何重新看待最高二进制位的问题,当把最高位看作符号位时,输出的就是有符号数,当把最高位看作数据位时,输出的就是无符号数。例如:

```
printf("%hu\n", a);   //将变量a的值以无符号短整型形式输出(最高为1看作数据位)
```

输出的值为65534,与printf("%hu\n", b);的输出值相同。

```
printf("%hd\n", b);   //将变量b的值以有符号短整型形式输出(最高为1看作符号位)
```

输出的值为-2,与printf("%hd\n", a);的输出值相同。

由此是否可以得出有符号短整型变量a与无符号短整型变量b是完全等价的呢?答案是否定的。上面的例子只是说明有符号数和无符号数在没有参与其他运算(只是简单的输出)的情况下并无什么差异,但一旦它们参与到其他运算的表达式之中,二者就会完全不同了,输出的结果也是大相径庭。

当表达式中存在有符号类型和无符号类型时,按照3.4.5节中算术运算数据类型转换规则(见图3-21),所有的操作数都自动转换为无符号类型。因此,从这个意义上讲,无符号数的运算优先级要高于有符号数,这一点对于频繁用到无符号数据类型的嵌入式系统来说是非常重要的。

首先看一个实例,分别定义一个signed int型数据和unsigned int型数据,然后进行大小比较:

```
unsigned int a=30;
int b=-130;
```

a>b?还是b>a?实验证明b>a,也就是说-130>30,为什么会出现这样的结果呢?这是因为在C语言操作中,如果遇到无符号数与有符号数之间的操作,编译器会自动转换为无符号数来进行处理,因此a=30,b=4294967166,这样比较下去当然b>a了。

比较下面程序一、程序二的运行结果。

程　序　一	程　序　二
#include <stdio.h> int main( )	#include <stdio.h> int main( )

{
unsigned int a=30;
int b=-130, c;
 	
c=(a+b)/2;
printf("c=%d\n", c);
return 0;
}
	{
	int b=-130, c;
	
	c=(30+b)/2;
	printf("c=%d\n", c);
	return 0;
	}

运行结果： c=2147483598	运行结果： c=-50

为什么程序一的输出结果与程序二的输出结果有如此大的差异呢？

因为在程序一中对于表达式(a+b)/2，在运算之前，考虑到变量 a 为无符号整型，因此 b 必须被转换为无符号整型，即 b 被转换为 4294967166(-130 的 4 字节的补码，最高位为数据位)，所以(a+b)/2 其实就是(30+4294967166)/2=2147483598。

在程序二中对于表达式(30+b)/2 的计算，因整常数 30 为有符号数，b 与整常数之间操作时不影响 b 的类型，运算结果仍然为 int 型，所以(30+b)/2 其实就是(30-130)/2=-50。

减法和乘法的运算结果类似。

而对于浮点数来说，浮点数(float，double)实际上都是有符号数，unsigned 和 signed 前缀不能加在 float 和 double 之前，当然就不存在有符号数与无符号数之间转换的问题了。

【例 3-10】　有符号数与无符号数之间的运算。

```
1    #include <stdio.h>
2
3    int main( )
4    {
5        unsigned int a=6;
6        int b=-20;
7
8        printf(" a+b=%d\n", a+b);
9        printf("(a+b) > 6? %s\n",(a+b) > 6 ? "Yes" : "No");
10       printf("(6+b) > 6? %s\n",(6+b) > 6 ? "Yes" : "No");
11       return 0;
12   }
```

运行结果：

```
a+b=-14
(a+b) > 6? Yes
(6+b) > 6? No
```

程序说明：

- 第8行计算a+b之前先将b转换为无符号整数4294967276(－20的4字节的补码,最高位为数据位),然后计算a＋b得4294867282(十六进制内存中表示为0XFFFFFFF2),由于输出时是按有符号十进制整型数输出(%d),因此将其计算的结果的最高位(为1,表示负)看成符号位,所以输出的结果均为－14。
- 第9行(a+b)＞6？"Yes"："No"是条件表达式(具体细节将在5.2.3节中介绍),其含义是如果(a+b)＞6为真,则该表达式的值为"Yes",否则为"No"。因为计算(a+b)的结果为4294967282,大于6,所以输出为Yes。
- 第10行因为计算(6+b)时,b与整常数之间操作时不影响b的类型,运算结果仍然为int型,其结果为－14,小于6,所以输出为No。


微课视频

3.7 本章小结及常见错误列举

本章所介绍的主要内容是整型数据、实型数据和字符型数据的常量表示法和变量定义格式,以及可以作用于这些数据类型的运算符。虽然本章的内容比较繁杂,学起来也许比较枯燥,但本章的内容是学好C语言的基础,每个C语言程序员必须熟练掌握本章内容。现在回忆一下本章有哪些内容值得特别留意和必须深刻领会的呢?

- 变量的含义;
- 数据在内存中的表示形式;
- 不同类型的数据在内存中的表示范围;
- 转义字符;
- 有符号数与无符号数的区别;
- 数据类型的自动转换与强制类型转换;
- 各种运算符、运算符的优先级和结合性。

从这一章开始读者可以试着编写程序了。但目前只能编写只有一个main函数的简单程序,因此,主要任务就是编写main函数。但编程并非总是非常顺利的,总会出现一些这样或那样的错误,特别是对初学者来说更是如此。下面就列举一些在编程过程中初学者极易犯的错误,希望能给读者编程带来一定的帮助。

(1) 书写标识符时,忽略了大小写字母的区别。

```
int main( )
{
   int a=5;
   printf("%d", A);
   return 0;
}
```

编译程序把a和A认为是两个不同的变量名,而显示出错信息。C语言认为大写字母和小写字母是两个不同的字符。习惯上,符号常量名用大写,变量名用小写,以增加可读性。

(2) 忽略了变量的类型,进行了不合法的运算。

```
int main( )
{
   float a,b;
   printf("%d", a%b);
   return 0;
}
```

%是求余运算，得到 a/b 的整余数。整型变量 a 和 b 可以进行求余运算，而实型变量则不允许进行“求余”运算。

(3) 将字符常量与字符串常量混淆。

```
char c;
c="a";
```

在这里就混淆了字符常量与字符串常量，字符常量是由一对单引号括起来的单个字符，而字符串常量是由一对双引号括起来的字符序列。C 语言规定以'\0'作为字符串结束标志，它是由系统自动加上的，所以字符串"a"实际上包含两个字符：'a'和'\0'，而把它赋给一个字符变量是不行的。

(4) 忘记加分号。

分号是 C 语句中不可缺少的一部分，语句末尾必须有分号。

```
a=1
b=2
```

编译时，编译程序在“a=1”后面没发现分号，就把下一行“b=2”也作为上一行语句的一部分，这就会出现语法错误。改错时，有时在被指出有错的一行中未发现错误，就需要看一看上一行是否漏掉了分号。

(5) 多加分号。

对于一个复合语句，如：

```
{
   z=x+y;
   t=z/100;
   printf("%f", t);
};
```

复合语句的花括号后不应再加分号，否则将是画蛇添足。

(6) 将 C 语言语句写在{ }的外面了。

C 语言规定，任何 C 语句都必须位于函数中。下面程序的错误就是把语句写在 main 函数的外面了。

```
int main( )
{
   int a;
   a=20;
   return 0;
}
printf("a=%d", a);          //位于函数外面了
```

(7) 变量未定义就使用。

C 语言规定,变量必须先定义后使用。下面的程序包含语法错误。

```
int mian( )
{
  //应在这里定义变量 a: int  a;
  a=20;
  a++;
  printf("a=%d", a);
  return 0;
}
```

(8) 在执行部分定义变量。

在标准 C 语言程序中,函数由声明部分和执行部分组成,并且这两部分不能有交叉,也就是说,不能在 C 语句中间定义变量。下面的程序中,变量 b 的定义放到了执行部分。

```
void main( )
{
  int a;
  a=10;
  int b;          //应将该行放在 a=10; 的前面
  b=a+20;
}
```

但是要记住,在 C++程序中没有这样的规定,只要变量定义在前,使用在后就行。所以上述程序在 C++程序中是正确的。读者在编程时一定要注意自己所使用的开发环境。

(9) 给变量赋值时忽视了变量的表示范围。

C 语言中定义的任何变量都有其数据的表示范围,在给变量赋值时一定要注意数的大小不要超出变量所表示的范围。尽管 C 编译时不会带来任何语法错误,但会带来意想不到的结果。这种错误往往比一般语法错误更难发现,甚至感到不可思议。例如下面的程序:

```
int main( )
{
  char ch=130;              //ch 的表示范围是-128~+127,130 超出了 ch 的表示范围
  printf("ch=%d", ch);      //输出 ch 的值将是-126
  return 0;
}
```

(10) 定义多个变量时,变量名之间用空格或分号分隔。

```
int a  b;                   //应改为: int a, b;
int x; y;                   //应改为: int x, y;
```

(11) 输入字符常量时漏掉单引号,认为 A、B 就是'A'、'B'。

```
char ch=A;                  //应该是 char ch='A';
char c;
c=B;                        //应该是 c='B';
```

(12) C 语句末尾的分号用了中文的分号(;)而不是西文的分号(;)。

```
A=x+y;                    //应改为 A=x+y;
```

(13) 误将字母 o 当成数字零(0)。

```
int a=o;                  //应改为 int a=0;
```

(14) 编程过程中经常漏掉}、)、'、"。

在 C 语言程序中,所有的{与}、(与)、'与'、"与"都是配对出现的,千万不可多了一个或少了一个。

```
x=((a+b)*c+d;             //应改为 x=((a+b)*c+d );
```

其实避免这种情况的发生很简单,就是在输入时同时成对输入,然后在中间填入其他内容即可。

(15) 定义变量时数据类型关键字与变量名之间无空格。

如果像这样定义变量:

```
inta;                     //应改为 int a;
```

C 编译程序将认为 inta 是一个标识符,但前面没有数据类型符,因此认为有语法错误。

(16) 对于 float 型变量使用%运算符。

C 语言规定,%只能用于整型变量(包括 int、short、long、char)。例如,下面的语句是错误的写法。

```
int a;
a=15.3 % 4;               //应改为 a=15 % 4;
```

(17) 对表达式进行强制类型转换时漏掉了( )。

像这样对一个表达式进行强制类型转换是错误的:

```
int(3.2+a)
```

应改为:

```
(int)(3.2+a)
```

(18) 赋值运算符“=”的左边使用表达式。

C 语言规定,不能对表达式或常量赋值,因为表达式或常量不对应内存单元。下面的做法是错误的。

```
int a, b;
a+b=30;                   //错误
30=a+b;                   //错误
```

C 语言编程习惯如下所述:

- 一行只放一条语句。尽管 C 语言允许在一行文本中放置多条语句,但一行只放一条语句有利于程序的调试。
- 养成随时给程序加注释的习惯。注释主要是帮助程序员自己或别人日后对程序的快速理解,因为对于一个稍微复杂一点的程序,如果没有加注释,时间长了即使是编写者自己也可能很难看懂,更何况他人呢。

- 程序的书写要有层次感,该缩进的一定要缩进。下面的程序中,左边的没有层次感,阅读起来比较别扭,右边的就好多了(在变量定义和执行部分的行首增加一些空格)。

```
int main()
{
int i,sum;
sum=0;
for(i=1;i<=100;i++)
if(i%2==0) sum+=i;
printf("sum=%d",sum);
return 0;
}
```

```
int main( )
{
   int i, sum;

   sum=0;
   for(i=1; i<=100; i++)
      if(i % 2==0)
         sum+=i;
   printf("sum=%d", sum);
   return 0;
}
```

- 编写函数时,变量定义部分和函数的执行部分之间增加一空行,或者在程序的执行部分按照完成的功能块增加相应的空行,会增加程序的易读性。
- 为变量起有意义的名字,既可以帮助程序员读懂程序,也可以避免变量的重复乱用,导致程序的逻辑错误。
- 在运算符和赋值符的两边加上一个空格会增加程序的易读性。

习题 3

1. 填空题

(1) 在 C 语言中,基本数据类型主要有_____、_____、_____三种。

(2) 根据 C 语言标识符的命名规则,标识符只能由_____、_____、_____组成,而且第一个字符必须是_____或_____。

(3) C 语言中的常量分为_____常量和_____常量两种。定义_____常量需要使用预处理命令 # define。

(4) 在 C 语言中,八进制整型常量以_____作前缀,十六进制整型常量以_____作前缀。

(5) 在 C 语言中,一个 char 型数据在内存中所占的字节数为_____;一个 short 型数据在内存中所占的字节数为_____。

(6) 在 C 语言中,一个 float 型数据在内存中所占的字节数为_____;一个 double 型数据在内存中所占的字节数为_____。

(7) C 语言中,short 型数据的取值范围为_____。

(8) 已知 int m=5, y=2; 则计算表达式 y+=y--m*=y 后的 y 值是_____。

(9) 语句: x++;++x; x=x+1; x=1+x; 执行后都使变量 x 中的值增 1,请写出一条同一功能的赋值语句(不得与列举的相同)_____。

(10) 若 a 为整型变量,则表达式"(a=4*5, a*2), a+6"的值为_____。

(11) 假设 m 是一个三位数,从左到右用 a, b, c 表示各位的数字,则从左到右各个数字是 bac 的三位数的表达式是_____。

(12) 若有以下定义语句: int u=010, v=0x10, w=10; printf("%d,%d,%d\n",u,v,w); 则输出结果是_____。

(13) 求解赋值表达式 a=(b=10)%(c=6),a、b、c 的值依次为 _____。

(14) 设 float x=2.5, y=4.7; int a=7; 则表达式 x+a%3*(int)(x+y) % 2/4 的值为_____。

(15) 若有定义: int a=2, b=3; float x=3.5, y=2.5;则表达式(float)(a+b)/2+(int)x%(int)y 的值为_____。

(16) 表达式 8/4*(int)2.5/(int)(1.25*(3.7+2.3))值的数据类型为_____。

(17) 已知: int a=5; 则执行 a+=a--a*a; 语句后,a 的值为_____。

(18) 设有 int a,b; a=100; b=20; a+=200; b*=a-100; 则 a=_____, b=_____。

(19) 在 C 语言源程序中,一个变量代表_____。

(20) 若 t 为 double 型变量,表达式 t=1, t+5, t++的值是_____。

2. 选择题

(1) 在 C 语言系统中,double、long、int、char 类型数据所占字节数分别为(　　)。

A. 8,2,4,1　　B. 2,8,4,1　　C. 4,2,8,1　　D. 8,4,4,1

(2) 下面四个选项中,均是不合法的用户标识符的选项是(　　)。

A. A　P_0　do　　B. float　la0　_A

C. b-a　sizeof　int　　D. _123　temp　int

(3) 下面四个选项中,均是合法整型常量的选项是(　　)。

A. 160　-0xffff　011　　B. -0xcdf　01a　0xe

C. -01　986,012　0668　　D. -0x48a　2e5　0x

(4) 下面四个选项中,均是不合法的浮点数的选项是(　　)。

A. 160.　0.12　e3　　B. 123　2e4.2　.e5

C. -.18　123e4　0.0　　D. -e3　.234　1e3

(5) 下面四个选项中,均是不合法的转义字符的选项是(　　)。

A. '\"'　'\\'　'\xf'　　B. '\1011'　'\'　'\ab'

C. '\011'　'\f'　'\}'　　D. '\abc'　'\101'　'xlf'

(6) 下面四个选项中,均是正确的数值常量或字符常量的选项是(　　)。

A. 0.0　0f　8.9e　'&'　　B. "a"　3.9e-2.5　1e1　'\"'

C. '3'　011　0xff00　0a　　D. +001 0xabcd　2e2　50.

(7) 下面程序段的输出结果是(　　)。

```
int i=5, k;
k=(++i)+(++i)+(i++);
printf("%d,%d", k, i);
```

A. 24,8　　B. 21,8　　C. 21,7　　D. 24,7

(8) 下面程序段的输出结果是(　　)。

```
short int i=32769;
printf("%d\n", i);
```

A. 32769　　B. 32767

C. −32767　　D. 输出不是确定的数

(9) 若有说明语句：char c='\72'；则变量 c(　　)。

A. 包含 1 个字符　　B. 包含 2 个字符

C. 包含 3 个字符　　D. 说明不合法，c 的值不确定

(10) 若有定义：int a=7；float x=2.5，y=4.7；则表达式 x+a%3*(int)(x+y)%2/4 的值是(　　)。

A. 2.500000　　B. 2.750000　　C. 3.500000　　D. 0.000000

(11) 设变量 a 是整型，f 是实型，i 是双精度型，则表达式 10+'a'+i*f 值的数据类型为(　　)。

A. int　　B. float　　C. double　　D. 不确定

(12) sizeof(float)是(　　)。

A. 一个双精度型表达式　　B. 一个整型表达式

C. 一种函数调用　　D. 一个不合法的表达式

(13) 设变量 n 为 float 类型，m 为 int 类型，则以下能实现将 n 中的数值保留小数点后两位，第三位进行四舍五入运算的表达式是(　　)。

A. n=(n*100+0.5)/100.0　　B. m=n*100+0.5，n=m/100.0

C. n=n*100+0.5/100.0　　D. n=(n/100+0.5)*100.0

(14) 在 C 语言中，要求运算数必须是整型的运算符是(　　)。

A. /　　B. ++　　C. !=　　D. %

(15) 若变量已正确定义并赋值，下面符合 C 语言语法的表达式是(　　)。

A. a:=b+1　　B. a=b=c+2　　C. int 18.5%3　　D. a=a+7=c+b

(16) 若有定义：int k=7,x=12；则能使值为 3 的表达式是(　　)。

A. x%=(k%=5)　　B. x%=(k-k%5)

C. x%=k-k%5　　D. (x%=k)-(k%=5)

(17) 若变量 a、i 已正确定义，且 i 已正确赋值，合法的语句是(　　)。

A. a==1　　B. ++i;　　C. a=a++=5;　　D. a=int(i);

(18) 若有如下程序：

```
int main( )
{
  int y=3, x=3, z=1;
  printf("%d %d\n",(++x, y++), z+2);
```

```
    return 0;
}
```

运行该程序的输出结果是(　　)。

A. 3　4　　B. 4　2　　C. 4　3　　D. 3　3

(19) 下面正确的字符常量是(　　)。

A. "c"　　B. '\\'　　C. 'W'　　D. "

(20) 在C语言中,5种基本数据类型的存储空间长度的排列顺序为(　　)。

A. char<int<=long int<=float<double

B. char=int<=long int<=float<double

C. char<int<=long int=float=double

D. char=int=long int<=float<double

(21) 假设所有变量均为整型,则表达式(a=2, b=5, b++, a+b)的值是(　　)。

A. 7　　B. 8　　C. 6　　D. 2

(22) 以下正确的叙述是(　　)。

A. 在C程序中,每行中只能写一条语句

B. 若a是实型变量,C程序中允许赋值a=10,因此实型变量中允许存放整型数

C. 在C程序中,无论是整数还是实数,都能被准确无误地表示

D. 在C程序中,%是只能用于整数运算的运算符

(23) 假定x和y为double型,则表达式x=2, y=x+3/2的值是(　　)。

A. 3.500000　　B. 3　　C. 2.000000　　D. 3.000000

(24) 下面程序的输出结果是(　　)。

```
int main( )
{
   int a=3;
   printf("%d\n",(a+=a-=a*a) );
   return 0;
}
```

A. -6　　B. 12　　C. 0　　D. -12

(25) 已知各变量的类型说明如下:int k, a, b; unsigned long w=5; double x=1.42;则以下不符合C语言语法的表达式是(　　)。

A. x%(-3)　　B. w+=-2

C. k=(a=2,b=3,a+b)　　D. a+=a-=(b=4)*(a=3)

(26) 若变量a是int类型,并执行了语句:a='A'+1.6;则正确的叙述是(　　)。

A. a的值是字符C

B. a的值是浮点型

C. 不允许字符型和浮点型相加

D. a的值是字符'A'的ASCII值加上1

(27) 语句printf("a\bre\'hi\'y\\\bou\n");的输出结果是(　　)。

A. a\bre\'hi\'y\\\bou　　B. a\bre\'hi\'y\bou

C. re'hi'you　　D. abre'hi'y\bou

(28) 下面程序的输出结果是(　　)。

```
int main( )
{ int x='f';  printf("%c \n", 'A'+(x-'a'+1)); return 0; }
```

A. G　　B. H　　C. I　　D. J

(29) 下面程序的输出结果是(　　)。

```
int main( )
{ char x=0xFFFF;  printf("%d \n", x--);  return 0; }
```

A. -32767　　B. FFFE　　C. -1　　D. -32768

(30) 已知：int x=1，y=-1；则语句 printf("%d\n",(x-- &++y))；的输出结果是(　　)。

A. 1　　B. 0　　C. -1　　D. 2

3. 阅读题

(1) 下面程序的输出结果是_____。

```
int main( )
{
  int x=5, y;
  y=++x*++x;
  printf("y=%d\n", y);
  return 0;
}
```

(2) 下面程序的输出结果是_____。

```
int main( )
{
  float a=1, b;
  b=++a*++a;
  printf("%f\n", b);
  return 0;
}
```

(3) 下面程序的输出结果是_____。

```
int main( )
{
  short int x=-32769;
  printf("%d\n", x);
  return 0;
}
```

(4) 下面程序的输出结果是_____。

```
int main( )
{
```

```
    unsigned short a=65536;
    int b;
    printf("%d\n", b=a);
    return 0;
}
```

(5) 下面程序的输出结果是_____。

```
int main( )
{
    unsigned char x, y, z;
    x=0x3; y=x | 0x8; z=x<<1;
    printf("%d, %d\n", y, z);
    return 0;
}
```

(6) 下面程序的输出结果是_____。

```
int main( )
{
    char a=0x95, b, c;
    b=(a&0xf)<<4;
    c=(a&0xf0)>>4;
    a=b|c;
    printf("%x\n", a);
    return 0;
}
```

(7) 下面程序的输出结果是_____。

```
int main( )
{
    unsigned short a, b;
    a=0x9a;
    b=~a;
    printf("a=%x  b=%x\n", a, b);
    return 0;
}
```

(8) 下面程序的输出结果是_____。

```
int main( )
{
    int k=10;
    float a=3.5, b=6.7, c;
    c=a+k%3*(int)(a+b)%2/4;
    printf("%f\n", c);
    return 0;
}
```

(9) 下面程序的输出结果是_____。

```
int main( )
```

```
{
   char ch;
   short int a=-32768, c;
   unsigned long b=0xffffaa00;
   ch=a;
   c=b;
   printf("ch=%d, c=%hx\n",ch,c);
   return 0;
}
```

(10) 下面程序的输出结果是_____。

```
int main( )
{
   char a='\x41';
   printf("%c, %c, %d, %d", a, a+1, a, a+1);
   return 0;
}
```