第 2 章　数据的基本类型与基本运算

本章重点介绍 C 语言的基本数据类型（整型、实型、字符型）的概念、详细分类、内存表示及使用方法，以及常量、变量的使用语法、规范。对各种数据类型之间的混合运算的规则给出了详细说明。本章还介绍 C 语言的算术运算、关系运算、逻辑运算、自增自减运算及相应的表达式。通过本章的学习，应能掌握 C 语言数据和运算的基本概念，为以后各章的学习打下基础。

【任务 2.1】计算圆的面积和周长

任务描述：给定圆的半径，求圆的面积和周长。

任务分析：设圆的半径用 radius 表示，圆的面积用 area 表示，圆的周长用 circumference 表示，则面积和周长计算公式可分别表示为：

$$area=\pi \times radius^2 \qquad (2\text{-}1)$$

$$circumference=2\times \pi \times radius \qquad (2\text{-}2)$$

只要给定半径 radius 的值，根据上述两个式子就可以计算出圆的面积和周长值。

程序需要考虑如何解决以下两个主要问题：

（1）在内存中用多大的空间存储 π（假设 π 取值 3.14）和 2 等在运行过程中不变的数据（称为常量），以及 radius、circumference 和 area 等在运行过程中变化的数据（称为变量），如何限定这些数据所能参加的运算。

（2）程序设计语言中如何表示式 2-1 和 2-2。

2.1　基本数据类型

在高级语言的编程中，必须首先在内存中定义数据，才能描述对数据的处理方法。数据在内存中的存储方式、运算方法，被称为数据类型。C 语言提供了丰富的数据类型，如图 2.1 所示。

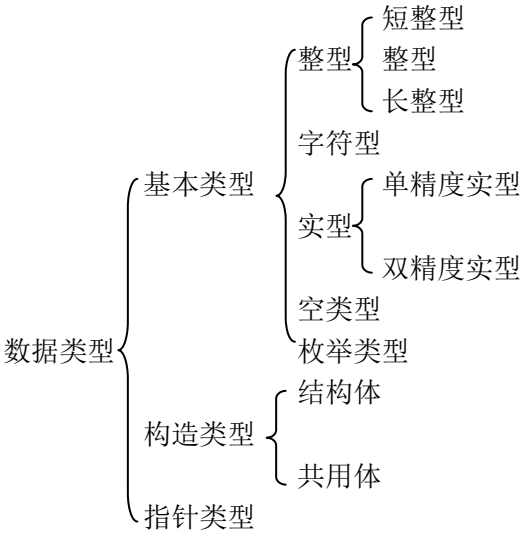


图 2.1　C 语言中的数据类型

C 语言提供了“基本数据类型”标识常见的信息类型，其中的整型、实型用于常见的数值运算；字符型用于非数值运算的应用。为了满足应用中的具体需求，这些基本数据类型可以附加修饰词，用于定义长整型、短整型、有符号整型、无符号整型、有符号字符型、无符号字符型等，详细内容见本章后继章节。

数据类型的本质是确定数据在内存的存储方式。举例而言，在 VC++环境中一个 int 型数占用 4 字节，那么 int 型描述的就是这 32 位是如何表示整数的，以及可表示整数的范围。因此，在学习数据类型时，不仅要关心数据的具体值，更应留意数据在内存的表示形式。

2.1.1 整型

1. 整型的分类

整型的定义符是 int。整型有两个空间大小修饰符 short 和 long。short int 称为短整型，占有 2 个字节；long int 称为长整型，占有 4 个字节。在 VC++环境中，long 是缺省的长度修饰符，所以 int 等价于 long int。

整型有两个符号位修饰符 signed 和 unsigned。signed int 表示有符号整数，unsigned int 表示无符号整数。signed 是缺省的符号位修饰符，所以 int 也等价于 signed int。

由于类型修饰符及默认的书写格式，使得整型的表现形式变化较多，实际上，整型共分为 4 类：有符号的长整型、无符号的长整型、有符号的短整型、无符号的短整型。表 2-1 对整型的分类做了归纳。

表 2-1 整型的分类

整型的分类	类型定义符	占用字节数	取值范围
有符号的长整型	int	4	-2^{31}~2^{31}-1
	long int		
	signed long int		
无符号的长整型	unsigned int	4	0~2^{32}-1
	unsigned long int		
有符号的短整型	short int	2	-2^{15}~2^{15}-1
	signed short int		
无符号的短整型	unsigned short int	2	0~2^{16}-1

2. 整型数据的内存表示

整型数据的内存表示分为两类：

（1）无符号整数，只能表示 0 和正整数；

（2）有符号整数，可以表示负数、0 和正数。

为便于计算，以下以 2 个字节的短整型数据为例，加以说明。

无符号整型数在内存中采用二进制数形式存储，如图 2.2 所示。显然，2 个字节的无符号整数可以表示的整数范围是 0~2^{16}-1，4 个字节的无符号整数可以表示的整数范围是 0~2^{32}-1。

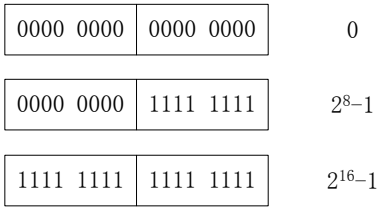


图 2.2　2 个字节无符号短整型数的表示

有符号整型数在内存中采用补码形式存储。在补码规则中，为区别正数、负数的表示，设最高位为符号位，其余位是数据位。若符号位为 1 则表示负数，否则表示 0 或正数。正整数的补码比较简单，是对应整数的二进制数形式，示例如图 2.3（a）所示。

负整数的补码规则是：①将符号位设为 1；②计算负整数绝对值的二进制形式；③对所有数据位取反，再加 1，示例如图 2.3（b）所示。

（a）0 和正整数的表示　　（b）负整数的表示

图 2.3　有符号短整型数的表示

显然，2 个字节的有符号短整数可以表示的范围是-2^{15}~$2^{15}-1$，4 个字节的有符号长整数可以表示的范围是-2^{31}~$2^{31}-1$。

无论哪种整型单元，当试图存储超过范围的数据时，实际只能存储部分数据，造成数据的失真，这种错误称为“溢出”。在程序运行时，“溢出”将导致严重的逻辑错误。

2.1.2　实型

单精度实型的定义符是 float，占用 4 个字节；双精度实型的定义符是 double，占用 8 个字节。在内存中，实型数据被分成符号位（1 位）、小数部分、指数部分，按指数形式存储。其中小数部分和指数部分的位数由具体的编译系统决定。显然，小数部分位数越多，数据精度就越高；指数部分位数越多，数据范围就越大。

表 2-2 给出了 VC++环境中实型数的取值范围。

表 2-2　实型的分类

实型的分类	类型定义符	占用位数			取值范围	精度
		符号位	小数部分	指数部分		
单精度实型	float	1	23	8	10^{-38}~10^{38}	7~8 位十进制有效数字
双精度实型	double	1	48	15	10^{-308}~10^{308}	15~16 位十进制有效数字

2.1.3 字符型

字符型的定义符是 char，字符数据在内存中占用 1 个字节，存储的是字符的 ASCII 值。如'a'~'z'的码值是 97~122，'A'~'Z'的码值是 65~90，'0'~'9'的码值是 48~57。常见字符与 ASCII 代码对照表参见附录 A。

和整型相似，字符类型的修饰符有 signed、unsigned，具体类型分为 signed char（有符号字符型）和 unsigned char（无符号字符型）。signed 是缺省修饰符，所以 char 等价于 signed char。

unsigned char 型采用二进制数据形式存储字符码值，码值范围是 0~255。signed char 型采用补码形式存储字符的码值，将最高位作为符号位，码值范围是-128~127。实际上，标准 ASCII 码值的范围是 0~127，所以就简单字符处理而言，unsigned char 与 signed char 基本没有差别。图 2.4 是 char 型数据'a'、'A'、'0'的内存表示，它们的符号位都是 0。

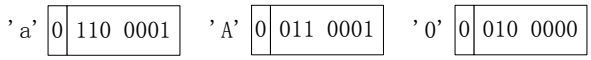


图 2.4　字符型数据的存储表示

2.2 常量

2.2.1 字面常量

常量是程序运行过程中其值不能改变的数据。它有两种表现形式：字面常量和符号常量。字面常量以字面格式直接写在程序之中。字面常量用于不同数据类型的表示时，有不同的格式要求。

1. 整型常量

在 C 语言中，整型数据有 3 种字面格式：

（1）十进制整数：书写格式与通常整数写法无异，如-111、0、111 等

（2）八进制整数：书写格式是在通常八进制数的前面加一个数字 0，如-0111、0111，分别表示的十进制数是-73、73。

（3）十六进制整数：书写格式是在通常十六进制数的前面加一个数字 0x，如-0x111、0x0、0x111，分别表示的十进制数是-273、0、273。

在 VC++环境中，整数的字面常量的表示范围是-2^{31}~2^{31}-1。对于超出范围的字面常量，编译器常常并不提示语法错误，可相应内存单元实际只存储了部分数字。这种情形称为“溢出”，可能造成严重的逻辑错误。

2. 实型常量

实型常量只能使用十进制表示，字面格式有两种：

（1）一般格式，如-12.345、0.0、12.345 等。有时小数点前后的“0”可以被省略，如-.12、0.、.12、12.等等。

（2）指数格式，如-1.2345e2、12.34e-4 等，由尾数、字母 e 或 E、指数部分组成。所有实型常量无论大小，都是按照双精度类型进行存储的。实型常量可以表示的最小实数是-1.79e308，最大实数是 1.79e308。

3. 字符型常量

字符常量就是用单引号括起来的单个字符，如'a'、'X'、'5'、','、' '。注意' '（两个单引号之间有一个空格）表示空格字符，''（两个单引号之间没有空格）是非法的字符常量。若单引号中包含多个字符，在编译时，一般编译器会忽略多余的字符，并产生警告错误。

在 ASCII 字符集中，有一些不可打印的控制字符以及一些特殊字符，无法采用上述方式表示，C 语言使用转义字符表示它们。转义字符是以“\”开头的字符序列。常用的转义字符及其含义参见表 2-3。

表 2-3　常用的转义字符及其作用

字符形式	含　　义
\n	换行，将当前位置移到下一行开头（ASCII 码值 10）
\t	水平制表，跳到下一个 Tab 位置（ASCII 码值 9）
\r	回车，将当前位置移到本行开头（ASCII 码值 13）
\b	退格，将当前位置移到前一列（ASCII 码值 8）
\\	代表一个反斜杠字符 \
\'	代表一个单撇号字符 '
\"	代表一个双撇号字符 "
\ddd	1~3 位八进制数所代表代表的字符
\xhh	1~2 位十六进制数所代表的字符

在表 2-3 中可以看到，因为单引号（'）、双引号（"）、反斜杠（\）字符在程序中的特殊用途，所以必须设置专门的转义字符序列表示它们。\ddd 和\xhh 是字符的两种通用表示方法，可用于表示任意字符，但一般用于控制字符的情形较多。

4. 字符串常量

字符串常量是字符序列，它是用双引号"括起来的若干字符，如"abc123"、"x,Y,z"。其中包含的字符数可长可短，甚至可以为 0。如""(两个连续的双引号)表示一个空的字符串，请留意其与' '（空格字符）的区别。

在内存中，字符串常量由字符的表示组成。其占用的字节数取决于字符串中的字符数。若字符串中有 n 个字符，则需要 n+1 个字节。除了存储每个字符的 ASCII 码之外，在末尾字符之后，另需要 1 个字节存储空字符'\0'，以标记字符串的结束。如图 2.5 所示，是 3 个字符串的内存表示。其中空字符串""，虽然不包含任何字符，可存储字符串结束符的'\0'的字节依然不能缺少。

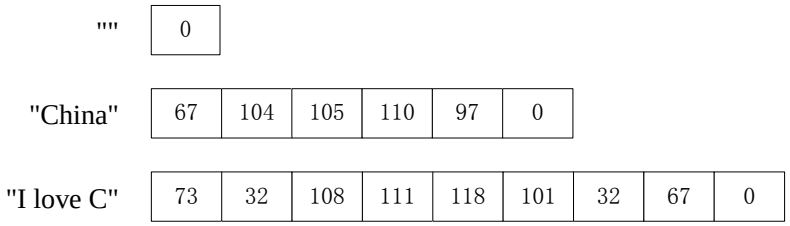


图 2.5　字符串常量的存储表示

2.2.2　符号常量

在同一程序中，某些字面常量可能使用多次，为了提高程序的可读性，方便程序员维护代码，常常采用预处理命令#define 定义符号常量，格式如下：

#define 符号常量 常量值

例如以下命令定义符号常量 PI 为 3.14。

#define PI 3.14

#define 命令的功用发生在程序被编译之前的预处理阶段。在此阶段，编译器根据符号常量的定义，将程序中的所有符号常量，统一替换为相应的字面常量。替换之后，再对程序代码进行编译。

程序中语句为：

s=PI*radius*radius;

预处理后结果为：

S=3.14*radius*radius;

一个程序可以根据需要定义多个符号常量。不同程序中的符号常量互不影响。

在使用#define 命令时，有以下注意事项：

（1）符号常量是一种标识符，应遵守标识符的命名规则，详见 2.3.1 小节。

（2）符号常量的名字不能与程序中的其他标识符重名。否则，在预处理阶段中，常量替换将造成代码的混乱。许多看似正确的代码，被替换后可能错误百出，而源头其实只是符号常量的重名问题。

为使符号常量区别于一般标识符，C 程序的编程习惯是将符号常量一律定义为大写单词。这是一种良好的编程风格，能增强代码的可读性。

（3）#define 命令不是 C 语言的语句，其后不应添加分号等不必要的符号。多余的符号在常量替换过程中，极可能产生意想不到的混乱后果。

2.3 变量

2.3.1 变量的概念与命名

变量是程序代码中最常见的标识符，它用于表示程序运行中可以发生变化的数据。变量的本质是一个被命名的内存单元，单元中数据的类型固定不变，值可以变化。

变量命名由一个标识符来表示，称为变量名。设定一个标识符时有以下规则：

（1）标识符只能由字母（A～Z，a～z）、数字（0～9）、下划线（_）组成，第一个字符必须是字母或下划线。例如：ab、_ab、_12、_a12b 都是合法的标识符。下划线作为一个特殊字符，当命名较长的标识符时，常常用到它来提高标识符的可读性。

（2）在标识符中，大、小写字母是有区别的，例如：book 与 Book 是两个不同的名字。

（3）if、else、int、float 等关键字是系统保留的名字，不能将它们定义为标识符。注意，main、define 等虽然在程序中常见，但它们不是关键字。

虽然有如上规则的约束，标识符的命名还是有很大的自由度。在程序设计中，我们应培养良好的命名习惯，应使得标识符的名字尽可能地达到“见名知义”的效果，以增加程序的可读性。例如我们常常将 i、j、k 作为循环变量的名字，用 count、sum 来表示计数和求和的变量。

2.3.2 变量的定义和初始化

1. 变量定义语句

在程序中使用任何变量，都必须先定义再使用。

变量定义语句格式如下：

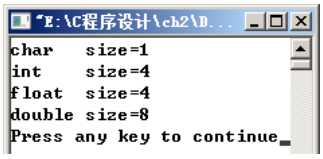
类型定义符 变量名，变量名,……，变量名;

一条变量定义语句可以定义若干个同类型的变量。操作系统将为每个变量分配一个连续的内存空间，空间的大小取决于变量的类型。

【例 2.1】 定义变量，并观察变量所占内存空间的字节数。

```
#include<stdio.h>

int main()
{
    char c;                                  // 将 c 定义为 char 型的变量
    int i;                                   // 将 i 定义为 int 型的变量
    float x;                                 // 将 x 定义为 float 型的变量
    double z;                                // 将 z 定义为 double 型的变量

    printf("char     size=%d\n",sizeof(c));
    printf("int      size=%d\n",sizeof(i));
    printf("float    size=%d\n",sizeof(x));
    printf("double size=%d\n",sizeof(z));

    return 0;
}
```

程序运行结果截图如下：



程序分析：

以上程序中，定义了 4 种基本类型的变量。使用了一个运算符 sizeof。sizeof 可以计算各种数据类型所占空间的字节数。sizeof 的使用形式有两种：

（1）sizeof(类型名)：计算指定数据类型所占的字节数

（2）sizeof(变量名)：计算指定变量所占的字节数

根据以上输出，可以绘制程序运行时的内存结构图，如图 2.6 所示，图中每个方框表示一个字节。

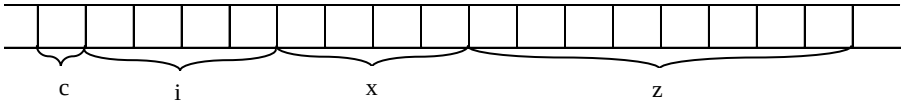


图 2.6 例 2.1 中定义的变量的内存结构图

需要注意的是，变量定义语句虽然为变量分配了各自的内存空间，但并未明确指定内存单元的数据状态。对于动态变量，其值为随机值；对于静态变量，其值为 0。这将在后续章节中详述。

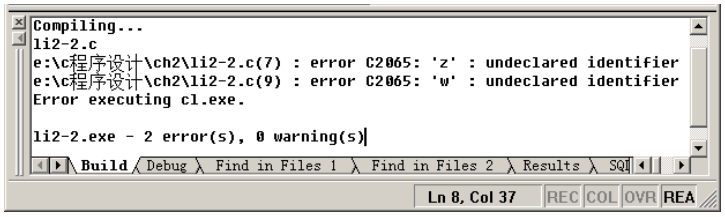
【例 2.2】 任何变量在使用前必须定义，否则将造成语法错误。

```
#include<stdio.h>

int main()
```

```
{
    int x,y;

    z=3;        // 语句①
    y=5;
    w=x+y;   // 语句②
    printf("w=%d\n",w);

    return 0;
}
```

程序分析：

以上程序中，语句①②由于变量 z 和 w 未定义，在 VC++编译器中会给出如下图所示的错误提示：



2. 变量初始化

变量的初始化是指在定义变量的同时，给变量赋予初值。如以下语句在定义 x、y 的同时，给 x 赋初值 3，给 y 赋初值 5。

```
int x=3, y=5;
```

在变量定义语句中，也可以只初始化部分变量。如以下语句定义了 3 个变量，只对其中的变量 x、z 进行了初始化。

```
int x=3, y, z=5;
```

变量初始化的优点有以下两方面：

（1）可以减少几条赋值语句，能适当提高代码效率，使程序风格显得简洁；

（2）避免了变量取随机值的现象，保证了程序执行状态的确定性，降低了程序出错的机会。

由于在变量定义语句中，可以初始化变量，也可以不初始化变量，所以应警惕那些未初始化的变量的使用，避免引用尚是随机值的变量单元。

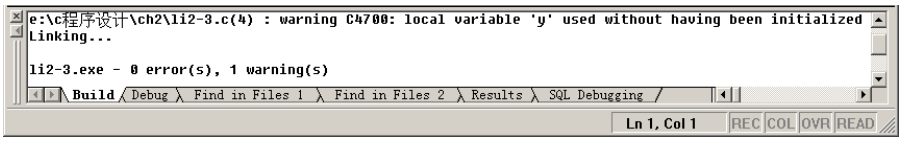
【例 2.3】 使用未初始化的变量，将导致警告性的语法错误。

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

    return 0;
}
```

程序分析：

以上程序中， 变量定义语句，初始化了变量 x，但变量 y 和 z 尚是随机值，在“z=x+y;”

中，却使用了 y 值来计算 z 值，显然这里隐藏了逻辑错误。在实际使用 VC++进行编译时会产生如下图所示的警告：



2.3.3 变量的赋值

程序中，通过为变量赋值可以修改存储变量单元的内容。变量赋值语句的格式如下：

变量名=表达式;

其中，=是赋值运算符。赋值语句的功能是先计算表达式的值，然后将该值存储到变量单元中（赋值语句将在 4.2.1 节中详细介绍）。例如：

int a=2;

表示变量 a 的初始值为 2，在执行以下赋值语句：

a=a+1;

则变量 a 的值被修改为 3。

2.4 数据的基本运算

2.4.1 C 语言运算符简介

C 语言的运算符种类较多，除了控制语句和输入输出以外的几乎所有的基本操作都是由各种类型的运算符完成的。C 语言运算符参见表 2-4。

表 2-4 C 语言运算符的分类

分类	运算符
算术运算符	+ - * / % ++ --
关系运算符	> < == >= <= !=
逻辑运算符	! && ‖
位运算符	<< >> ~ | ^ &
赋值运算符	=
条件运算符	?:
逗号运算符	,
指针运算符	* &
求字节数运算符	sizeof
强制类型转换运算符	(类型定义符)
分量运算符	. ->
下标运算符	[ ]
其他运算符	如，函数调用运算符( )

本章只介绍算术运算符、关系运算符、逻辑运算符和自增自减运算符，其他运算符在后续章节中介绍。

2.4.2 算术运算符与算术表达式

1. 算术运算符

基本的算术运算符有以下几种：。

（1）加运算+，示例：3+5, a+b+c。

（2）减运算-，示例：5-3, a-b-c。

（3）乘运算*，示例：3*5, a*b*c。

（4）除运算/，示例：5/3, a/b/c。

（5）模运算%，示例：5%3, a%b。模运算的两个运算数必须是整型数据，否则在 VC++ 中编译时，会造成语法错误。

2. 算术表达式

用算术运算符将运算数连接起来，并符合 C 语法规则的式子，称为 C 的算术表达式。其中运算数可以包括常量、变量、函数等。

将数学表达式转换为算术表达式时，初学者应注意一些格式上的转变，如：$x=\frac{-b+\sqrt{b^2-4ac}}{2a}$ 对应的表达式是 x=(-b+sqrt(b*b-4*a*c))/(2*a)。

sqrt 是求平方根的系统库函数，使用到它时，应在程序首部添加#include <math.h>。

将数学表达式转换为算术表达式时，有一些技巧可以使提高计算的效率。如计算 $y=x^3-5x^2+16x-80$，可以将算术表达式写成 y=x*x*x-5*x*x+16*x-80; 也可以将其写作 y=((x-5)*x+16)*x-80; 显然，后者的计算效率高于前者。

2.4.3 关系运算符与关系表达式

1. 关系运算符

在 C 语言中有以下关系运算符：

（1）< 小于

（2）<= 小于或等于

（3）> 大于

（4）>= 大于或等于

（5）== 等于

（6）!= 不等于

关系运算符都是双目运算符，其结合性均为左结合。关系运算符的优先级低于算术运算符，高于赋值运算符。在六个关系运算符中，<、<=、>、>=的优先级相同，高于==和!=，==和!=的优先级相同。

2. 关系表达式

关系表达式的一般形式为：

表达式 关系运算符 表达式

例如：

a+b>c-d

x>1

'a'+1<'b'

i==j+1

都是合法的关系表达式。字符变量是以它对应的 ASCII 码参与运算。

由于表达式也可以又是关系表达式，因此允许出现嵌套的情况。例如：

a>(b>c)

a!=(b==c)

等。

关系表达式的结果是“真”和“假”，分别用值“1”和“0”表示。例如：

5>0

为“真”，值为 1。

(a=1)>(b=2)

由于 1>2 不成立，故为“假”，其值为 0。

注意：

在使用关系运算符时，应避免对实数作相等或不等的判断。例如对于关系表达式 1.0/3.0*3.0==1.0 的结果为假（值为 0），实际编程中可改写为：

fabs(1.0/3.0*3.0-1.0)<1e-6

2.4.4 逻辑运算符与逻辑表达式

1. 逻辑运算符及其优先次序

C 语言中提供了 3 种逻辑运算符：

&& 与运算

|| 或运算

! 非运算

与运算符&&和或运算符||均为双目运算符，具有左结合性。非运算符!为单目运算符，具有右结合性。

2. 逻辑表达式

逻辑表达式的一般形式为：

表达式 逻辑运算符 表达式

例如：

5>0 && 4>2

5>0||5>8

!(5>0)

逻辑表达式中的表达式可以又是逻辑表达式，从而组成了嵌套的情形。

例如：

(a&&b)&&c

上式也可写为：

a&&b&&c

逻辑运算的结果也为“真”和“假”两种，用值“1”和“0”来表示。

其求值规则如下：

（1）与运算 &&。参与运算的两个量都为真时，结果才为真，否则为假。例如：

5>0 && 4>2

由于 5>0 为真，4>2 也为真，相与的结果也为真。

（2）或运算||。参与运算的两个量只要有一个为真，结果就为真。两个量都为假时，结果为假。例如：

5>0||5>8

由于 5>0 为真，相或的结果也就为真。

（3）非运算!。参与运算量为真时，结果为假；参与运算量为假时，结果为真。例如：

!(5>0)

的结果为假。

虽然 C 编译在给出逻辑运算值时，以“1”代表“真”，“0”代表“假”。但在判断一个量是为“真”还是为“假”时，以“0”代表“假”，以非“0”的数值作为“真”。例如：

由于 5 和 3 均为非“0”，因此 5&&3 的值为“真”，即为 1。

又如：

5||0

的值为“真”，即为 1。

注意：

（1）对于在数学中常见的表达式-1<x<1，在 C 语言程序中必须表示成：

x>-1&& x<1

如果错误地在 C 语言程序中写成-1<x<1，其执行过程是，先判断-1<x，得到一个逻辑值 0 或 1，再拿这个值与 1 作比较，结果当然不是预期的。

（2）在逻辑表达式求值的过程中，并不是所有的逻辑运算符都被执行，通常称之为“短路”运算。例如：

在求解 a&&b&&c 的值时，只在 a 为真时，才判别 b 的值；只在 a、b 都为真时，才判别 c 的值。如果 a 为假，则不会判别 b 和 c，因为整个表达式的结果已经确定为假了。

在求解 a||b||c 的值时，只在 a 为假时，才判别 b 的值；只在 a、b 都为假时，才判别 c 的值，如果 a 为真，就不会判别 b 和 c，因为整个表达式的结果已经确定为真了。

看一个具体的例子：

a=1; b=2; c=3; d=4; m=1; n=1;

(m=a>b)&&(n=c>d)的结果将使得 m 的值为 0，但 n 的值仍为 1。因为先计算 m=a>b，则 m 的值为 0(因 a<b)，则整个表达式的值就为 0，&&后面的表达式根本就不计算，所以 n 的值不变，仍为 1。

2.4.5 自增、自减运算符

自增运算符++的作用是使变量的值增 1，自减运算符--的作用是使变量的值减 1。例如：设有“int a=5,b=3;”，执行“a++;”之后，a 的值是 6；执行“b--;”之后，b 的值是 2。

显然，自增、自减运算符只能用于变量。常量、表达式的自增、自减是非法的，如：5++，(a+b)++，将造成编译时语法错误。这是因为常量不可以改变，而表达式的值并无明确的空间存储，自增自减后的结果无处可存。

根据自增、自减运算符的位置，常常将其分为“前++”、“前--”和“后++”、“后--”两类。孤立地看：

i++和++i 都等价 i=i+1;

i--和--i 都等价于 i=i-1;

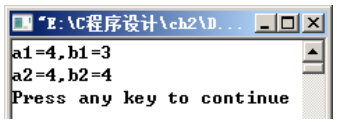
当将自增、自减运算符运用于较复杂的表达式时，其间的区别就显现出来了。以“前++”和“后++”为例说明。

【例 2.4】 “前++”和“后++”的差别。

#include<stdio.h>

```
int main()
{
    int a1,a2,b1,b2;

    a1=3;
    b1=a1++;
    printf("a1=%d,b1=%d\n",a1,b1);

    a2=3;
    b2=++a2;
    printf("a2=%d,b2=%d\n",a2,b2);

    return 0;
}
```

程序运行结果截图如下：



程序运行结果分析：

以上程序中，执行语句“b1=a1++;”的流程是：首先将 a1 的值赋给 b1，因此 b1 的值是 3；然后 a1 自增 1，值为 4。执行语句“b2=++a2;”的流程是：首先 a2 的值自增 1，值是 4；然后将 a2 的值赋给 b2，因此 b2 的值是 4。

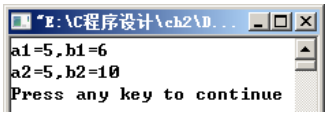
可见“a++”和“++a”的差别在于自增操作的发生时机。“a--”和“--a”的差别也与此类似。

在 C 语言标准中，对表达式中子表达式的求值次序并未详细规定。若一个表达式中包含多个前++、后++运算，那么这些子表达式的执行次序，其实并无标准可言。在不同的编译器中，表达式的计算结果可能不同。

【例 2.5】 表达式中子表达式的求值次序。

```
#include<stdio.h>

int main()
{
    int a1,a2,b1,b2;

    a1=3;
    b1=(a1++)+(a1++);     // 语句①
    printf("a1=%d,b1=%d\n",a1,b1);

    a2=3;
    b2=(++a2)+(++a2);     // 语句②
    printf("a2=%d,b2=%d\n",a2,b2);

    return 0;
}
```

在 VC++环境中，程序运行结果截图如下：



程序运行结果分析：

以上程序中，语句①等价于“b1=a1+a1; a1=a1+1; a1=a1+1;”因此，首先 b1 被赋值 6，之后 a1 的值被增加到 5。语句②等价于“a2=a2+1; a2=a2+1; b2=a2+a2;”因此，首先 a2 的值被增加到 5，之后 b2 被赋值 10。

也有编译器可能将语句①等价于“a1=a1+1; b1=a1+a1; a1=a1+1;”。因此，本例在不同的编译器中，其输出可能是不确定的。

C 语言提供自增、自减运算符的本意是，为了在编译过程中，对于一些常用的、简单的运算进行代码优化，以提高程序的效率。初学者应领会此意，避免编写可能具有歧义的代码。

2.4.6 运算符的优先级与结合性

C 语言语法规定了运算符的优先级和结合性。在表达式求值过程中，先按照运算符的优先级别的高低次序执行，例如先乘除后加减。如“a+b*c”，其中*的优先级高于+的优先级。如果运算数两侧的运算符号的优先级相同，如“a+b-c”，则按照规定的“结合性”处理。算法运算符的结合方向是“自左至右”。在“a+b-c”中 b 先与左侧的运算符结合，即先执行加法再执行减法。

表 2-5 仅给出了本章介绍的运算符的优先级和结合性，完整的运算符优先级和结核性参见附录 B。

表 2-5 运算符的优先级和结合性

运算符	优先性	优先级
!、++、--	1	自右至左
*、/、%	2	自左至右
+、-	3	自左至右
<、<=、>、>=	4	自左至右
==、!=	5	自左至右
&&、‖	6	自左至右

【例 2.6】已知有声明：int x=5,y=8; 计算++x>5&&4/5<0||7%9+1==y 的值。

解：表达式中各运算符的优先级别如图 2.7 所示。

```
++x > 5 && 4/5 <0 || 7 % 9 + 1 == y
①  ④    ⑥ ② ④ ⑥  ②  ③    ⑤
```

图 2.7 表达式中运算符的优先级

具体计算过程如表 2-6 所示。

表 2-6 表达式计算过程

步骤	表达式	操作
1	++x>5&&4/5<0‖7%9+1==y	计算++x 得到结果 6
2	6>5&&4/5<0‖7%9+1==y	自左至右，计算 4/5 和 7%9，得到结果 0 和 7
3	6>5&&0<0‖7+1==y	计算 7+1 得到结果 8
4	6>5&&0<0‖8==y	自左至右，计算 6>5 和 0<0，得到结果 1 和 0

5	1&&0‖8==y	计算 8==y，得到结果 1
6	1&&0‖1	自左至右计算，得到结果 1

【例 2.7】 自增运算符和算术运算符的优先级和结合性。

```
#include<stdio.h>

int main()
{
    int a1,a2,b1,b2,c1,c2;

    a1=5,b1=3;
    c1=a1+b1++;
    printf("%d,%d,%d\n",a1,b1,c1);

    a2=5,b2=3;
    c2=a2+++b2;
    printf("%d,%d,%d\n",a2,b2,c2);

    return 0;
}
```

程序运行结果截图如下：

程序运行结果分析：

以上程序中，执行“c1=a1+b1++”的流程是：先执行“c1=a1+b1”，c1 的值是 8；再执行“b1=b1+1”，b1 的值是 4。执行“c2=a2+++b2;”的流程是：由于++自右至左的结合性，++运算是针对 a2，是 a2 的后++，因此先执行“c2=a2+b2”，c2 的值是 8；再执行“a2=a2+1”，a2 的值是 6。

在实际编程中，要求初学者能读懂类似例 2.7 的代码，但不提倡模仿它的风格。对于大多数程序员而言，在程序代码中，坚持让每个赋值语句只修改一个变量的值，是一种更简洁明了的程序风格。

2.5 数据类型转换

C 语言的表达式总是将相同类型的数据进行运算，但它也允许在一个表达式中，包含整型、实型、字符型等多种类型数据。这是表面上的“宽容”，实际在进行每个运算之前，不同类型的数据要首先转换为同一类型。转换的方式分为自动类型转换、强制类型转换两种。前者是在运算中不需要指定，系统自动进行的转换，程序员应当理解这种自动转换的规则。后者用于当自动转换不能达到目的时，程序员显式指定类型名，对运算对象进行的类型转换。

2.5.1 自动类型转换

自动类型转换的基本原则是空间较小的数据类型自动转换为空间较大的数据类型，如图 2.8 所示，图中的箭头表示了数据类型的转换方向，任何两个类型之间的转换是一次完成的。例如：当 char 型数据和 float 型数据运算时，char 型数据将直接转换为 float 型数据；当 int 型数据和 double 型数据运算时，int 型数据将直接转换为 double 型数据。

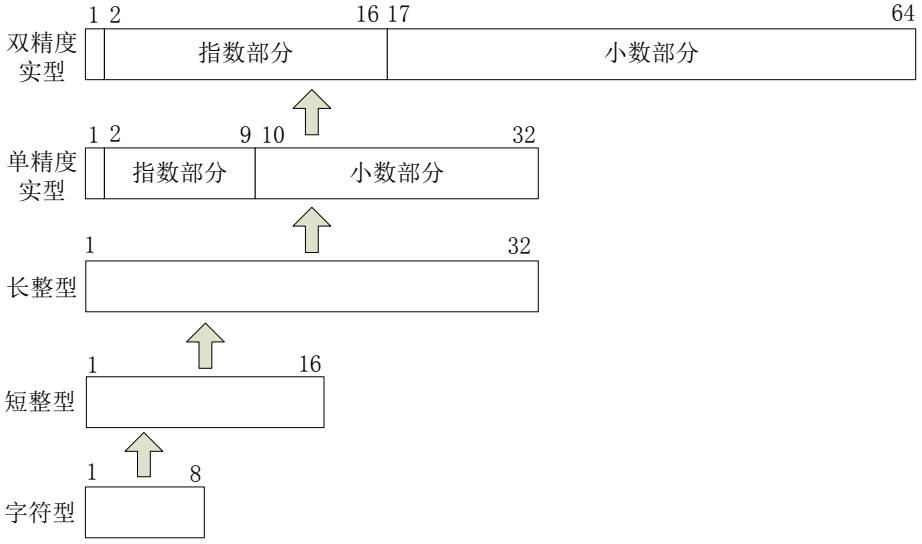


图 2.8　自动类型转换的规则

自动类型转换总是给我们带来“很自然的”转换，但也常常使一些粗心的人忽略了内存的变换，以下针对图 2.8 中的主要类型转换，分别举例说明。

当字符型数据与整型数据运算时，如表达式是'a'+1024，具体计算过程是：首先占 1 个字节的字符型数据'a'被转换为 4 个字节的整数 97，然后两个整数运算，得到结果是 1121。计算过程如图 2.9 所示。

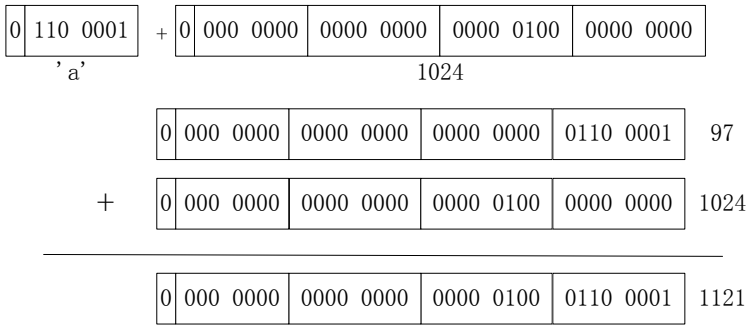


图 2.9　字符型数据与整型数据的加法运算

当短整型数据与长整型数据运算时，如设 short int x=1; int y=65536; 表达式是 x+y，具体计算过程是：首先占 2 个字节的短整型数据 1 被转换为占用 4 个字节的长整型数据 1，然后两个长整数相加，得到结果是 65537。计算过程如图 2.10 所示。

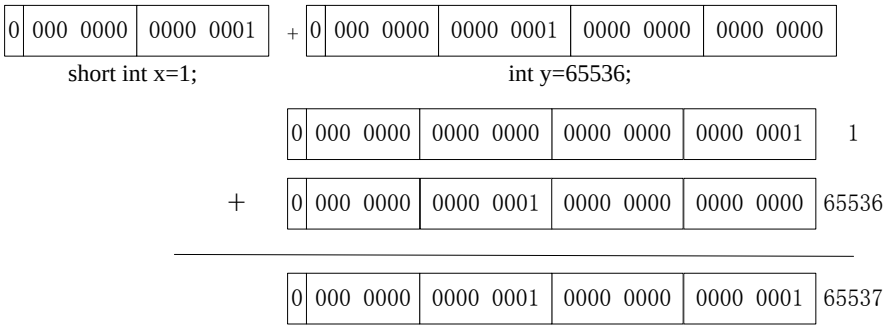


图 2.10　短整型数据与整型数据的加法运算

当整型数据与实型数据运算时，如表达式 100+1.25，具体计算过程是：首先占 4 个字节的整型数据 100 被转换为了占用 8 个字节的双精度实型数据 100.0，然后两个实型数据相加，得到结果是 101.25。

当单精度实型数据与双精度实型数据运算时，如设 float x; double y; 表达式是 x+y，具体计算过程是：首先占 4 个字节的单精度实型数据 x 被转换为了占用 8 个字节的双精度实型数据，然后两个同类型数据的运算。

提醒初学者注意，尽管 C 语言提供了自动类型转换的功能，但是仍应尽量避免对不同类型的变量（数据）进行赋值等操作。

2.5.2　强制类型转换

在一些应用情形中，自动类型转换不能达到编程的目的，这时程序员可以利用强制类型转换符将运算对象转换为指定的类型。强制类型转换的一般形式是：

（类型名）(表达式)

若表达式为单个数据，则表达式的括号可以省略。示例如下：

(double)a　　// 将 a 的值转换为 double 型

(float)(a+b)　　// 将 a+b 的值转换为 float 型

(int)(x/y) %3　　// 将(x/y)的值转换为 int 型，再进行求余运算

由于类型转换运算符只对紧随其后的表达式进行类型转换，因此，以下相似的表达式有不同的计算结果，应引起注意。设有 int a=5,b=3;

(float)(a/b)的计算过程是：先计算 a/b，因为是整数除法，结果是 1；最后(float)将 1 转换为 1.0。

(float)a/b 的计算过程是：首先(float)将 a 的值转换为临时性的中间数据 5.0；其次 5.0/b 引发自动类型转换，将 b 的值转换为另一个临时性的中间数据 3.0；最后计算 5.0/3.0，值是 1.666667。

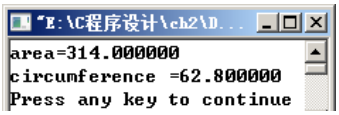
需要注意的是，在对数据进行强制类型转换时，系统会自动生成一些所需类型的中间数据，运算后，系统自动销毁这些数据。注意这个过程不会改变表达式中原始数据的类型和值。

2.6　完成任务 2.1 的参考程序

```
#include<stdio.h>
#define PI 3.14
```

```
int main()
{
    int radius=10;
    double area, circumference;

    area = PI* radius * radius;              //计算圆的面积
    circumference = 2*PI *radius;            //计算圆的周长
    printf("area=%f\n", area);               //输出圆的面积值
    printf("circumference =%f\n", circumference);   //输出圆的周长值

    return 0;
}
```

程序运行结果截图如下：



程序说明：

由于周长和面积的计算均需要用到常量 π，因此可将其处理成符号常量。以上示例的程序代码量较少，符号常量 PI 被引用的次数也不多，符号常量的优点尚不明显。随着程序规模的增大，符号常量的必要性将凸显出来。

C 语言源程序的书写格式是非常自由的，但好的书写格式对于阅读、调试和修改程序来说是非常重要的。本书给出几点书写格式的建议：

（1）程序采用缩进格式；

（2）不同含义的源代码之间加空行，这样阅读代码时就比较简单和轻松了。

（3）“{”后不写代码，“}”前不写代码，这样便于在 VC6 编程环境中用组合键 Alt+F8 操作对所选源代码进行格式化。

（4）关键语句后添加必要的注释，以提升源代码的可读性。

2.7 程序设计实例

【例 2.8】 将华氏温度 100 转换为对应的摄氏温度。

分析：

设 C 表示摄氏温度，F 表示华氏温度，华氏温度转换为摄氏温度的计算公式是：

$$C=5\times(F-32)/9$$

本程序中需要存储华氏温度 100 这个数值，以及存储转换后的摄氏温度数值，因此需要定义两个温度变量。为了便于直观理解变量的逻辑含义，这里用变量 temC 表示摄氏温度，temF 表示华氏温度。接下来还需为这两个变量确定数据类型。确定变量类型时，主要考虑该变量的取值、取值范围及取值精度，以占用较少内存的类型为佳。对于 temC 和 temF 变量而言，因其保存的是实数，故其数据类型选 float 或 double 均可，但考虑到本题所存实数的精度不用太高，故数据类型选 float。

参考程序：

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

```
int main()
{
    float temC, temF=100;

    temC=5*(temF-32)/9;
    printf("华氏温度%5.1f 对应的摄氏温度是%5.1f \n",temF, temC);

    return 0;
}
```

程序运行结果截图如下：

本章小结

本章的主要内容总结如下：

（1）各种字面常量的表示方法，以及符号常量的定义、使用方法。

（2）变量定义、初始化等语法知识以及内存中的变量结构。

（3）标识符的命名规则，以及良好的命名规范。

（4）整型、实型、字符型数据的分类及内存表示方法。

数据类型	数据类型符号	占用字节数	数值范围
整型	int	4	-2^{31}~$2^{31}-1$
单精度实型	float	4	-3.4×10^{38}~3.4×10^{38}（保留 6 位有效数字）
双精度实型	double	8	-1.79×10^{308}~1.79×10^{308}（保留 17 位有效数字）
字符型	char	1	-128~127

（5）字符型数据的内存表示方法，举例说明了转义字符的使用方法。

（6）当表达式中包含多种类型数据时，自动类型转换的方向及相应的转换规则；并说明了强制类型转换符的应用方法。

（7）算术表达式、关系表达式、逻辑表达式、运算符的优先级与结合性等概念，讨论了自增、自减运算符的使用方法及注意事项。

习题 2

一、读程序，写结果。

1.

```
#include <stdio.h>
#define N 2
#define M N+3
int main()
{
```

```
	int a=0256, b=256, c=0x256;
	printf("%d, %d, %d\n", a,b,c);
	printf("%d\n", M*4);
	return 0;
}
```

2.

```
#include<stdio.h>
int main()
{
	char c1='x',c2='z';
	int x1=1, x2=2;
	printf("%d,%d\n",c1,c2);
	printf("%c,%c\n",c1,c2);
	printf("%c,%c\n",'0'+x1, '0'+x2);
	printf("%d,%d\n",'0'+x1, '0'+x2);
	return 0;
}
```

3.

```
#include <stdio.h>
int main()
{
	char a='B',b=33;
	char c='0',d='9';
	a=a-'A'+'0';
	b=b*2;
	printf("%c,%c\n",a,b);
	c++,d--;
	printf("%c,%c\n",c,d);
	return 0;
}
```

4.

```
#include<stdio.h>
int main()
{
	int i=8,j=10;
	int m=++i;
	int n=j++;
	int x=-i++;
	int y=j+12/++j;
	printf("%d,%d,%d,%d\n",i,j,m,n);
	printf("%d,%d,%d,%d\n",i,j,x,y);
	return 0;
}
```

二、简答题。

1. 请将下面各数用二进制、八进制、十六进制数来表示。

   10，32，65，2050，32767 ，-130，-510，-32760

2. 字符常量与字符串常量有什么区别？

3. 请写出执行以下赋值语句后，数据在变量中的存储形式。

   int a=3, b=-3;

   short int c=3, d=-3;

   unsigned short int e=32768;

   char c1=’a’, c2=’A’, c3=’0’;

   unsigned char c4=128, c5=255;

4.写出下列数学表达式对应 C 语言表达式。

   x^3+2x^2+3x+1

   $$\sqrt{|x|} \neq \frac{4a}{bc}$$

5. 写出下列条件对应的 C 语言表达式。

   1~100 内的奇数

   闰年的判定条件

6. 设 double a=2;int b=7; double c=4.7, d=2.5; 请写出以下表达式的值。

   a+b%3*(int)(a+c)%2/4;

   (float)(a+b)/2+(int)c%(int)d

7. 设 a=1,b=5,c=7，计算下列各逻辑表达式的值

   c-a>b&&a!=c||!b

   !(a+b)-c&&b/7

   c-b||!(a+c)&&b