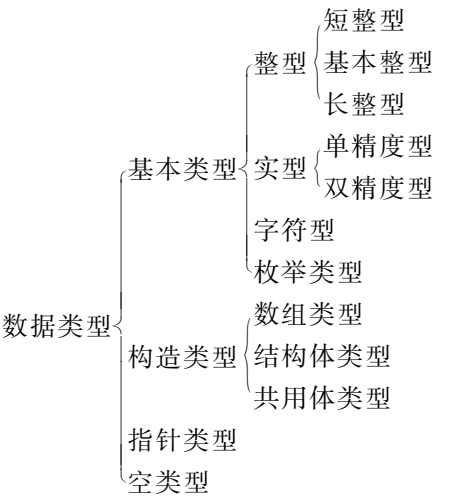
第 3 章　数据类型和表达式

主要知识点

- ◆ 标识符
- ◆ 常量和变量
- ◆ 基本数据类型
- ◆ 常用的输入输出函数
- ◆ 运算符和表达式
- ◆ 数据类型转换

在程序中使用的数据都要指定其数据类型。数据类型决定数据所占内存空间的大小、数据的合法取值范围和可以进行的运算。

C 语言提供了以下几种数据类型。



本章主要介绍 C 语言程序中常用的整型、实型和字符型等数据类型。不同的 C 编译系统对各种类型数据分配的存储空间大小不同。在 Visual C++ 6.0 编译系统中，整型、实型和字符型数据所占内存空间(字节数)和取值范围如表 3-1 所示。

表 3-1　常用基本数据类型的字节数

数据类型		类型标识符	字节数	取值范围
整型	短整型	short int(或 short)	2	$-32768(-2^{15})\sim 32767(2^{15}-1)$
	基本整型	int	4	$-2^{31}\sim 2^{31}-1$
	长整型	long int(或 long)	4	$-2^{31}\sim 2^{31}-1$

续表

数据类型		类型标识符	字节数	取值范围
实型	单精度	float	4	$-3.4\times10^{-38}\sim 3.4\times10^{38}$
	双精度	double	8	$-1.7\times10^{-308}\sim 1.7\times10^{308}$
字符型		char	1	$-128(-2^7)\sim 127(2^7-1)$

3.1 标识符

如同人们的名字一样,C语言中用标识符区分变量、符号常量、数组、函数等对象。标识符是由数字、字母和下画线3种字符组成的,且第一个字符必须为字母或下画线的字符序列。

例如,num、i、_io3等均为合法的标识符。M.D.Y、5b、wang-1等均为不合法的标识符。

在使用标识符时需要注意以下问题。

(1) 标识符应尽量做到见名知义,以增加程序的可读性。如用year表示"年",用month表示"月"。

(2) 标识符中的字母严格区分大小写。如Sum和sum是两个不同的标识符。

(3) 用户定义的标识符不要以下画线开头,因为库程序中经常用到以下画线开头的标识符。

(4) ANSI C标准没有规定标识符的长度,但不同的C编译系统对标识符的长度限制都有各自规定。因此,编写程序时需要了解所用系统对标识符长度的规定,以避免标识符出现混淆。

C语言预定义了32个标识符,如auto、double、if、for等(详见附录B),它们被赋予了特定的含义,不能另作他用,这些标识符称为关键字或保留字。

3.2 常量与变量

C语言程序中的数据一般以常量或变量的形式出现。

3.2.1 常量

常量是指在程序执行过程中,其值始终保持不变的量。C语言中的常量有两种:直接常量和符号常量。

1. 直接常量

直接常量一般从其字面形式即可判别,有不同类型的常量。例如,5、0、-12为整型常量,3.14、-2.45为实型常量,'1'、'a'、'#'为字符常量,"Beijing"、"a"、"123"为字符串常量。

2. 符号常量

C语言中可以用一个标识符代表一个常量,称为符号常量。使用符号常量的优点是能够见名知义,提高程序的可读性,便于程序的修改和维护。

在C语言中,通过预编译命令#define定义符号常量,其一般格式如下:

```
#define 符号常量名 直接常量
```

在 C 程序中，一般符号常量名用大写字母，变量名用小写字母，以示区别。

【实例 3-1】 已知长方形的长和宽，计算并输出其面积。

```
/* 实例 3-1 */
#include <stdio.h>
#define LENGTH 12
#define WIDTH 6
int main()
{
    int s;
    s = LENGTH * WIDTH;                    /* 计算面积 */
    printf("s = %d\n",s);
    return 0;
}
```

说明：

(1) 程序中用 LENGTH 表示长方形的长度为 12，用 WIDTH 表示其宽度为 6。比起 12 和 6，LENGTH 和 WIDTH 可读性明显要好。

(2) "/* … */"括起来的内容为注释，起到解释说明的作用，目的是便于阅读程序。将其删除不会影响程序运行结果。另一种注释语句的方法是用双斜杠"//"，一般用于一行的注释，例如，"/* 实例 3-1 */"可以写成"//实例 3-1"。

程序运行结果如图 3-1 所示。

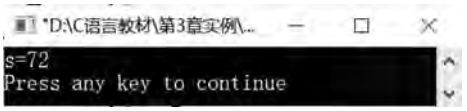


图 3-1 求长方形的面积

思考： 如何定义符号常量 PI 为 3.14159?

3.2.2 变量

在程序执行过程中，其值可以变化的量称为变量。变量的实质是内存中的一块存储单元，其大小由变量的类型决定。为便于引用，每个变量都有一个名字，通过变量名对其存储空间中的数据进行操作。变量名为合法的标识符，其命名规则依照 3.1 节标识符中的规定。

这里要注意区分两个概念：变量名和变量的值。变量名实际上是存储单元的符号地址，而变量的值是指存储单元中的数据，两者之间的关系如图 3-2 所示。

在 C 程序中，变量的使用遵循"先定义，后使用"的原则。定义变量的实质是程序申请内存空间，用于存储数据。

图 3-2 变量名和变量的值

1. 变量的定义

定义变量的一般格式如下：

```
类型标识符  变量名表列;
```

其中,类型标识符为 int、float、char 等数据类型名。变量名表列可以是一个变量名,也可以是多个变量名,多个变量名之间用逗号“,”分隔。

例如:

```
int step;     /*定义了整型变量 step,分配 4 字节的存储单元,用于存储整型数据*/
float s,v;    /*定义了两个单精度类型变量 s 和 v,各分配 4 字节的存储单元,用于存储单精度实
                型数据*/
```

可以使用一个特殊的运算符 sizeof 测试不同数据类型的变量所占的字节数。

【实例 3-2】 定义变量并测试变量所占字节数。

```
/*实例 3-2*/
#include <stdio.h>
int main()
{
    short int a;
    int b;
    char c;
    float f;
    printf("%d,%d,%d,%d\n",sizeof(a),sizeof(b),sizeof(c),sizeof(f));
    return 0;
}
```

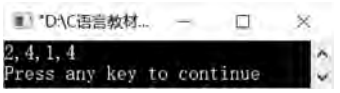


图 3-3 不同类型的变量所占字节数不同

说明:程序定义了 4 个不同类型的变量,并用 printf()函数输出这 4 个变量在内存中所占的字节数。

程序执行结果如图 3-3 所示。

2. 变量赋值

可以用赋值运算符“=”为变量赋值。

(1) 定义变量的同时为变量赋初值,也称为变量的初始化。

一般格式为:

```
类型标识符  变量名 1 = 值 1,变量名 2 = 值 2,…,变量名 n = 值 n;
```

例如:

```
int step = 1;          /*定义整型变量 step,其初值为 1*/
int sum = 0,cnt = 0;   /*定义整型变量 sum 和 cnt,它们的初值均为 0*/
```

(2) 先定义变量,然后为其赋值。

例如:

```
int step,sum = 0;      /*定义变量 step 和 sum,并且只为变量 sum 赋初值 0*/
step = 1;              /*为变量 step 赋值 1*/
```

点拨:一般定义自动变量(涉及变量的存储属性,详见 6.5.2 节变量的生存期)后,如果不对其赋值,其值不确定。如没有特殊说明,一般函数内定义的变量都是自动变量。

【实例 3-3】 变量赋初值和引用。

```
/*实例 3-3*/
#include <stdio.h>
```

```
int main()
{
    int step = 1,i;
    i = i + step;
    printf(" % d\n % d\n ",step,i);
    return 0;
}
```

图 3-4　变量赋初值

程序执行结果如图 3-4 所示。

说明：因为在程序中没有给变量 i 赋初值，导致输出结果异常。如果将定义语句改成

```
int step = 1,i = 1;
```

或在定义语句后加一个赋值语句“i=1;”，再运行程序，即可得到正常的结果。

3. 声明只读变量

定义变量时，如果在类型标识符前加关键字 const，则说明该变量为只读变量，其值不允许被修改。

【实例 3-4】 声明只读变量。

```
/ * 实例 3-4 * /
# include < stdio.h >
int main()
{
    const int step = 1;
    printf(" % d\n",step);
    step = 2;
    printf(" % d\n",step);
    return 0;
}
```

该程序在编译时出现错误信息，如图 3-5 所示。

说明：变量 step 为只读变量，只能被引用，程序中不能修改它的值。

图 3-5　只读变量

4. 指针变量

由于变量常常会占用多个内存单元，对应多个存储单元的地址，通常把起始地址作为该

变量的地址,在程序中变量的地址表示为“& 变量名”,其中 & 为取地址运算符。

一般变量的访问方式有两种,第一种是通过变量名(变量的符号地址)存取变量的值,称为“直接访问”方式;第二种是先把变量的地址存放在另一个变量中,通过该变量找到要访问的变量的地址,然后通过地址存取变量的值,称为“间接访问”方式。

一个变量的地址称为该变量的“指针”。存放变量的地址的变量,称为“指针变量”。

定义指针变量的一般形式为:

```
类型标识符   *变量名;
```

例如:

```
int *ip;        /*定义指针变量 ip,它可以保存一个整型变量的地址*/
float *fp       /*定义指针变量 fp,它可以保存一个单精度类型变量的地址*/
```

【实例 3-5】 定义整型变量 i 和指针变量 ip,ip 中保存变量 i 的地址。用直接访问的方式对变量 i 赋值为 3,用间接访问的方式对变量 i 赋值为 5,分别输出赋值后变量 i 的值。

```
/*实例 3-5*/
#include <stdio.h>
int main()
{
    int i,*ip;
    ip=&i;                          /*将变量 i 的地址保存在指针变量 ip 中*/
    printf("%x,%x\n",&i,&ip);       /*以十六进制整数格式输出 i 和 ip 的地址*/
    i=3;                            /*直接访问变量 i*/
    printf("i=%d\n",i);
    *ip=5;                          /*间接访问变量 i*/
    printf("i=%d\n",i);
    return 0;
}
```

程序执行结果如图 3-6 所示。

说明:

(1) 程序中有两个变量:变量 i 和指针变量 ip,从地址的输出结果可以看出它们各自占据不同的存储单元。

(2) 将变量 i 的地址保存在指针变量 ip 中,则可以形象地称为指针变量 ip“指向”变量 i。变量 i 与指针变量的关系如图 3-7 所示。

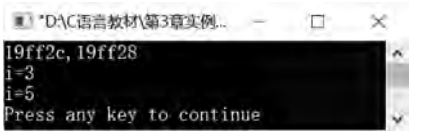


图 3-6 变量的两种访问方式

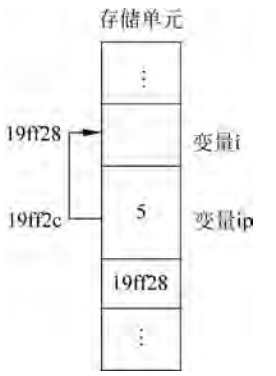


图 3-7 指针变量 ip 指向变量 i

(3) 变量的定义语句“int i，* ip;”中，变量名 ip 前的星号“ * ”仅仅是一个标识符号，表示变量 ip 是一个指针变量，与普通的整型变量 i 不同，以示区别。

(4) 与定义行中的不同，语句“ * ip=5;”中的星号“ * ”为指针运算符，其运算对象是指针变量，* ip 表示指针变量 ip 所指的对象，即变量 i。“ * ip=5;”与“i=5;”等价，从程序的执行结果也可以看出，执行该赋值语句后，变量 i 的值发生了变化。

点拨：一个指针变量只能指向一个同类型的变量，因为不同类型的变量在内存中所占的字节数不同，指针移动时的“跨度”就不同。指针的移动和指针的运算详见 5.5 节数组与指针。

3.3 基本数据类型

数据类型决定数据在内存中所占的字节数和存储方式，进而决定数据的取值范围和精度，数据类型还决定数据运算(操作)的规则。本节将介绍整型、实型和字符型等几种常用的基本数据类型。

3.3.1 整型

C 语言提供了短整型(short int)、基本整型(int)和长整型(long int)三种类型的整型，并且规定 short int 型所占的字节数不能大于 int 型，int 型所占的字节数不能大于 long int 型。具体某个整型数据所占的字节数取决于机器 CPU 的类型和编译器。例如，一般基本整型(int)占用一个字的存储空间。如果字长为 16 位，则基本整型用 16 位来表示；如果字长为 32 位，则基本整型用 32 位表示。

整型又分为有符号(signed)整型和无符号(unsigned)整型，其区别是取值范围不同，无符号整型所占字节数和取值范围见表 3-2。定义整型变量时，只要不指定为无符号型(类型标识符前面加 unsigned)，其隐含的类型就是有符号型(即 signed 可以省略)。

表 3-2 无符号整型所占字节数和取值范围

数据类型	类型标识符	字节数	取值范围
无符号短整型	unsigned short	2	0～65535($2^{16}-1$)
无符号基本整型	unsigned int	4	0～$2^{32}-1$
无符号长整型	unsigned long	4	0～$2^{32}-1$

1. 整型常量

在 C 语言中，整型常量有三种形式。

(1) 十进制整数，如 125、－23 等。

(2) 八进制整数，以数字 0 开头，由 0～7 组成的数字序列，如 0125 表示$(125)_8$，相当于十进制数 85($1\times8^2+2\times8^1+5\times8^0$)。

(3) 十六进制整数，以数字 0 和字母 x(或 X)开头，由 0～9，a～f 或 A～F 组成的序列，如 0x125 表示$(125)_{16}$，相当于十进制数 293($1\times16^2+2\times16^1+5\times16^0$)。－0x3c 相当于十进制数－60($3\times16^1+12\times16^0$)。

【实例 3-6】 整型常量的三种形式。

```
/*实例 3-6*/
#include <stdio.h>
int main()
{
    /*分别以十进制、八进制和十六进制整型格式输出十进制整数 10*/
    printf("%d, %o, %x\n", 10, 10, 10);
    /*以十进制整型格式输出(10)10、(10)8 和(10)16*/
    printf("%d, %d, %d\n", 10, 010, 0x10);
    return 0;
}
```

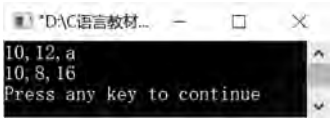


图 3-8 整型常量的输出

程序执行结果如图 3-8 所示。

点拨:

(1) 若整型常量后面加字母 l 或 L,则认为是长整型常量,如-45l、125L 等。

(2) 若整型常量后面加字母 u 或 U,则认为是无符号整型常量,如 23u、125U 等。

2. 整型变量

(1) 整型数据在内存中的表示。

数据在内存中是以二进制形式存放的。整数分为有符号整数与无符号整数。无符号整数的表示是基于二进制表示数据的方法。给定 n 位二进制序列,它所能表示的数值范围是$[0,2^n-1]$,如 16 位二进制数表示的无符号整数的范围是$[0,2^{16}-1]$,即[0, 65535]。

在计算机中,使用补码表示有符号整数。补码就是将位序列的最高位解释为负权。例如,给定 16 位二进制序列 1000 0000 0000 0001,如果它表示无符号整数,则它的值是$1\times2^{15}+1\times2^0=32769$,如果用补码理解它,则最高位是负权,也就是$-1\times2^{15}+1=-32767$。补码会将最高位理解为一个负数,直观看,就是当最高位是 1 的时候,是一个非常大的负数加上后面的正整数,后面的正整数越大,此数越接近 0,当最高位是 0 时,负权整个都是 0,剩下的几位如同无符号整数一样表示正整数。因此,16 位二进制数表示的有符号整数的范围是$[-2^{15}, 2^{15}-1]$,即[-32768, 32767]。n 位二进制数能表示的有符号整数范围是$[-2^{n-1},2^{n-1}-1]$。

下面以 unsigned short 和 short 为例,说明整型数据在内存中的存储方式。

unsigned short 类型数据占 2 字节,若 16 位二进制数如下:

1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

则表示$1\times2^{15}+1\times2^1+1\times2^0=32768+2+1=32771$。

short 类型数据占 2 字节,但数据以补码形式存放,最高位(最左侧)是符号位,符号位为 0 表示正数,符号位为 1 表示负数。若 16 位二进制数如下:

1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

则表示$-1\times2^{15}+1\times2^1+1\times2^0=-32768+2+1=-32765$。

若 16 位二进制数如下:

0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1

则表示 $1\times2^{14}+1\times2^{1}+1\times2^{0}=16384+2+1=16387$。

点拨：正数的补码和原码相同。

求负数的补码方法：先求该数的绝对值的原码，原码各位数取反后再加 1 得补码。

例如，求－32765 的补码步骤如下。

① 求 32765 的原码($32765=2^{15}-3$)。

0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1

② 各位数取反。

1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

③ 再加 1 得补码。

1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

(2) 整型变量的定义和使用。

遵循变量定义的格式，整型变量可以定义如下：

```
int sum;              /*定义整型变量 sum*/
short cnt;            /*定义短整型变量 cnt*/
long f;               /*定义长整型变量 f*/
unsigned int age;     /*定义无符号整型变量 age*/
```

在计算机中，当要表示的数据超出计算机所使用数据的表示范围时，就会产生数据的溢出。整型变量在使用时要注意“数据溢出”问题。

【实例 3-7】 整型数据的溢出。

```
/*实例 3-7*/
#include <stdio.h>
int main()
{
    short a,b;
    a = 32767;
    b = a + 1;
    printf("a = %d\n",a);
    printf("b = %d\n",b);
    return 0;
}
```

程序执行结果如图 3-9 所示。

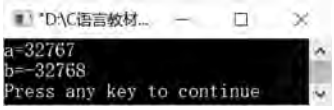


图 3-9 整型数据的溢出

说明：从程序的运行结果看，变量 b 的值出现“异常”状况。这是因为一个有符号的短整型变量占据 2 字节的内存空间，能表示的数据范围是[-2^{15}, $2^{15}-1$]，即[－32768,32767]。十进制整数 32767 的二进制数表示为 0111 1111 1111 1111，它加上 1 后得到二进制数 1000 0000 0000 0000，此二进制数刚好是十进制整数－32768 的补码。

3.3.2 实型

实型也称浮点类型。C语言中实型分为单精度类型(float)、双精度类型(double)和长双精度类型(long double)三种形式,本章主要介绍常用的单精度和双精度两种类型。

1. 实型常量

实型常量又称浮点数(floating point number),即实数。浮点数有两种表示形式。

(1) 小数形式。由数字和小数点组成,如3.14、-0.268、326.4等。实数3.0可以写成3.,而0.3可以写成.3。

(2) 指数形式。由尾数部分、字母e(或E)、指数部分组成,如48.6235e12,其中的48.6235为尾数,12为指数,e为基数10,表示48.6235×10^{12}。48.6235e12还可以表示为4.86235e13、4862.35e10、0.4862355e14等,其中的4.86235e13称为"规范化的指数形式",即尾数部分小数点左边有且只有一位非零的数字。例如,2.312e6、3.4256e-5、1.5673E-2等都属于规范化的指数形式。

指数形式的实数书写必须遵循如下规则。

① 有且仅有一个字母e或E,且字母e(或E)的左右两侧不能包含空格。例如,3.25□E3、3.25E□3都不是合法的实数(□表示空格)。

② 指数与尾数部分都不可缺省,且指数部分只能是整数。例如,E-3、1.2E、2.3E1.2都不是合法的实数。

点拨:C语言编译系统一般将实数作为双精度数来处理,这样虽然使得计算结果更精确,但却降低了运算速度。若要明确某个实数为单精度数,则在该数最后加字母f或F,例如,-0.26f、326.4F等。长双精度实数的表示方法是在数值最后加字母l或L,例如,256.33l、-6.4L等。

2. 实型变量

(1) 实型数据在内存中的表示与整型数据的存储形式不同,实型数据在内存中是按指数形式存储的,分为以下三部分。

① 数符:符号位,0代表"正",1代表"负"。

② 尾数:一般采用纯小数形式存储。

③ 阶码:存储指数,包括符号位。

具体尾数部分和指数部分所占位数的多少则由各C语言编译系统自定。对于占4字节的单精度数来说,不少编译系统用24位存储尾数(包括符号位),用8位存储阶码(包括符号位)。尾数部分所占的位数决定实数的精度,位数越多,精度也就越高。阶码部分所占的位数决定实数的取值范围,位数越多,能够表示的数值范围越大。表3-3为常用实型数据的取值范围及精度。

表3-3 常用实型数据的取值范围及精度

数据类型	字节数	有效数字	取值范围
单精度类型(float)	4	6～7	-3.4×10^{-38}～3.4×10^{38}
双精度类型(double)	8	15～16	-1.7×10^{-308}～1.7×10^{308}

(2) 实型变量的定义和使用。

实型变量可以定义如下：

```
float ave;          /* 定义单精度型变量 ave */
double pi;          /* 定义双精度型变量 pi */
```

【实例 3-8】 在两个不同的实型变量中保存同一个实数，并输出这两个变量的值。

```
/* 实例 3-8 */
#include <stdio.h>
int main()
{
    float fx;
    double fy;
    fx = 123456.789e6;
    fy = 123456.789e6;
    printf("fx = %f\n",fx);
    printf("fy = %f\n",fy);
    return 0;
}
```

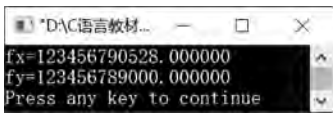


图 3-10 实型数据的精度

程序执行结果如图 3-10 所示。

说明：

(1) 程序编译时出现如下警告信息：

```
warning C4305: '=' : truncation from 'const double' to 'float'
```

这是因为在执行赋值语句“fx＝123456.789e6;”时，赋值号“＝”右侧的 123456.789e6 是 double 型常量，左侧的 fx 是 float 型变量，类型不一致，系统预警此次操作可能出现截断误差。

(2) printf 函数中的“%f”表示以小数形式输出实型数据，默认保留 6 位小数。

(3) 从程序运行结果可以看出，不同类型的实型变量，保存实数的有效数字不同。超过有效数字范围的数字是不精确的，所以，实型变量在使用时要注意“舍入误差”问题。

【实例 3-9】 实型数据的舍入误差。

```
/* 实例 3-9 */
#include <stdio.h>
int main()
{
    float a,b;
    a = 123456.789e4;
    b = a + 30;
    printf("a = %f\n",a);
    printf("b = %f\n",b);
    return 0;
}
```

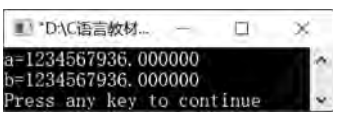


图 3-11 实型数据的舍入误差

程序执行结果如图 3-11 所示。

说明：从理论上来说，变量 a 的值应该是 1234567890.0，变量 b 的值应该是 1234567920.0，但程序输出两个变量的值却是另外的数。原因是变量 a 和 b 都是 float 类

型,只有7个有效数字,不能把实数123456.789e4的每个数字都精准地保存起来,后面的数字是不精确的。因此,实数计算中应避免将一个很大的数和一个很小的数相加或相减,以免造成误差。

3.3.3 字符型

字符型数据占1字节,且内存中存储的是字符的ASCII码。

1. 字符常量

字符常量一般是用单引号“'”括起来的一个字符,例如,'3'、'a'、'A'、'#'等。除了这些普通字符常量外,C语言还提供一种特殊形式的字符常量,是用单引号“' ”括起来并且以斜杠“\”开头的字符序列,例如,'\n'、'\t'、'\b'等,常用于表示一些转义字符。常用的转义字符如表3-4所示。

表3-4 常用的转义字符

转义字符	意　义	ASCII码	转义字符	意　义	ASCII码
\n	换行,移到下行行首	10	\\	反斜杠	92
\t	水平制表	9	\?	问号	63
\v	垂直制表	11	\'	单引号	39
\b	退格	8	\"	双引号	34
\r	回车,移到本行行首	13	\ooo	1～3位八进制数	
\f	换页	12	\xhh	1～2位十六进制数	
\a	振铃	7	\0	空操作字符	0

说明:

(1) 注意'\n'和'\r'的区别,前者是换行,当前位置移到下一行的行首;后者是回车不换行,当前位置移到本行的行首。

(2) '\ooo'表示1～3位八进制数表示的字符,例如,'\102'表示八进制数102,即十进制数66,表示大写字母B(见附录C常用字符的ASCII码)。'\0'是一种特殊情况,表示空操作,常用于字符串中,是字符串的结束标记。

(3) '\xhh'表示1～2位十六进制数表示的字符,例如,'\x41'表示十六进制数41,即十进制数65,表示大写字母A。

2. 字符串常量

字符串常量是用双引号括起来的字符序列,例如,"Zhejiang"、"$123.5"、"789#"等。若字符串中没有字符,即"",表示空字符串(简称“空串”)。

从直观上区分,字符常量与字符串常量有以下两个不同。

(1) 字符常量用单引号,字符串常量用双引号。

(2) 字符常量的单引号中只有一个字符(转义字符除外),字符串常量的双引号中可以有多个字符。

字符常量和字符串常量本质的区别是它们的存储形式不同。每个字符串常量结尾会存储一个字符串结束标记'\0'(即空操作字符)。例如,字符'a'和字符串"a"在内存中的存储如图3-12所示,保存'a'只要1字节,而保存"a"需要2字节的内存空间,除了保存字母a之外,还需要保存字符串结束标记'\0'。

图 3-12　字符和字符串的存储形式不同

3. 字符变量

(1) 字符数据在内存中的表示。

在计算机中,非数值的数据,如字母、文字或其他符号是用二进制编码表示的。西文字符最常用的是 ASCII 编码。对于汉字,我国也制定了相应的编码方案。本节主要介绍西文字符在计算机中的表示。

标准 ASCII 码使用 7 位二进制数来表示所有的大、小写字母,数字字符 0～9,标点符号以及一些特殊转义字符。为了便于处理,在 ASCII 码的最高位增加一位 0,用 1 字节存储一个 ASCII 码,也即用 1 字节存储一个西文字符。例如,字母 a 的 ASCII 码为十进制数 97,在内存中的存储如下:

0	1	1	0	0	0	0	1

(2) 字符变量的定义和使用。

字符变量可以定义如下:

```
char c1,c2;          /* 定义字符变量 c1 和 c2 */
```

一个字符变量只能存放一个字符,可以先定义变量再赋值,也可以在定义时为变量赋初值。

例如:

```
char c1 = 'a',c2;    /* 定义字符变量 c1 和 c2,并为 c1 赋初值 */
c2 = 'b';            /* 为变量 c2 赋值为字符常量'b' */
```

既然字符在计算机内存中存储的是它的 ASCII 码,其存储形式与整型数据的存储类似,如此,字符数据与整型数据就可以通用。

【实例 3-10】 字符数据与整型数据通用。

```
/* 实例 3-10 */
#include <stdio.h>
int main()
{
    int i;           /* 定义整型变量 i */
    char c;          /* 定义字符变量 c */
    c = 97;          /* 为变量 c 赋值为整数 97 */
    i = 'a';         /* 为变量 i 赋值为字符常量'a' */
    printf("%c, %d\n", c, c);      /* 分别以字符和十进制整数格式输出变量 c 的值 */
    printf("%c, %d\n", i, i);      /* 分别以字符和十进制整数格式输出变量 i 的值 */
    return 0;
}
```

说明:程序中将一个整数赋值给变量 c,将一个字符常量赋值给变量 i,分别用%c(字符格式)和%d(十进制整数格式)输出变量的值。因为字符数据与整型数据通用,所以输出结

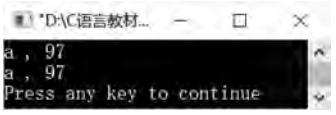


图 3-13　字符数据与整型数据通用

果完全相同。

程序执行结果如图 3-13 所示。

利用字符数据与整型数据通用的特点,在 C 语言程序中,要实现英文字母的大小写转换就很容易。

【实例 3-11】 小写字母转换成大写字母。

```
/* 实例 3-11 */
#include <stdio.h>
int main()
{
    char ch;                    /* 定义字符变量 ch */
    ch = 'a';                   /* 为变量 ch 赋值为字符常量'a' */
    printf("%c\n", ch);         /* 输出变量 ch 中保存的字符 */
    ch = ch - 32;               /* 将小写字母转换成大写字母 */
    printf("%c\n",ch);          /* 输出转换后的结果 */
    return 0;
}
```

说明:因为小写字母的 ASCII 码值比大写字母大 32,所以小写字母转换成大写字母用语句"ch=ch-32;"实现。反之,若要将大写字母转换成小写字母,则用语句"ch=ch+32;"实现。

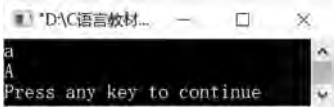


图 3-14　小写字母转换成大写字母

程序执行结果如图 3-14 所示。

点拨:字符数据和整型数据通用有一个前提,那就是在 1 字节存储空间能够表示的数值范围内,否则可能发生溢出问题,例如:

```
char ch = 180;
printf("%d",ch);
```

输出结果为-76。因为 char 型即 signed char 型,占 1 字节的内存空间,能够表示的有符号数据范围是[$-128(-2^7)\sim127(2^7-1)$],显然 180 超过了正数的范围。如果将语句"char ch=180;"改为"unsigned char ch=180;"后,输出结果为 180,因为 unsigned char 的取值范围是[0～255]。

3.4　常用的输入输出函数

C 语言的输入输出函数有很多种,本节主要介绍四个最常用的输入输出函数:printf()、scanf()、putchar()和 getchar()。特别强调,如果程序中需要使用这几个函数,必须在程序的开始位置加上预处理命令:

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

3.4.1　格式化输入输出函数

C 语言标准库提供了格式化输出函数 printf()和格式化输入函数 scanf()。printf()函

数用来向标准输出设备(屏幕)输出数据,scanf()函数用来从标准输入设备(键盘)读取输入数据。下面详细介绍这两个函数的用法。

1. 输出函数 printf()

printf()函数是格式化输出函数,一般用于向标准输出设备(屏幕)按规定格式输出信息。

printf()函数的调用格式为:

```
printf(格式控制, 输出表列);
```

其中"格式控制"是用双引号括起来的一个字符串,也称"格式控制字符串"。格式控制字符串分为格式字符串和非格式字符串两种。格式字符串是以%开头的字符串,在%后面跟各种格式字符,用来说明输出数据的类型、长度、小数位等。非格式字符串是指普通字符,在输出时原样输出。

例如:

```
printf("%d,%d",a,b);
```

其中"%d"是格式字符串,","是非格式字符串,也即普通字符,a 和 b 是输出表列。

2. 格式控制

对不同类型的数据,使用不同的格式字符。常用的格式字符有以下几种。

(1) d 格式符,输出一个有符号的十进制数。

%d:按整型数据的实际长度输出。

%md:m 是指定的输出字段的宽度。如果数据的位数小于 m,则左端补以空格;若大于 m,则按实际位数输出。

%ld:输出长整型数据,其中的 l 是 long 的意思。

例如:

```
printf("%5d\n%5d\n",23,-567);
```

输出结果为:

```
□□□23    (□表示 1 个空格)
□ -567
```

(2) c 格式符,输出一个字符。

例如:

```
char ch='y'
printf("%c",ch);
```

运行时输出:y

一个大小在 0~127 的整数,也可以用"%c"使之按字符形式输出,在输出前,系统会将该整数作为 ASCII 码转换成相应的字符。例如:

```
int a=65;
printf("%c",a);
```

运行时输出:A

输出时也可以指定域宽,例如:

```
printf("%6c",ch);
运行时输出:□□□□□A    (□表示一个空格)
```

如果整数比较大,则把它的最后1字节的信息以字符形式输出。同理,一个字符数据也可以按整型数据输出。

【实例3-12】 输出字符数据。

```
/*实例3-12*/
#include<stdio.h>
int main()
{
int i=99;
char ch='b';
printf("%d,%5c\n",ch,i);          /*以十进制和字符形式输出*/
return 0;
}
```

程序运行结果如图3-15所示。

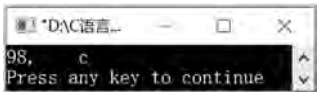


图3-15 输出字符数据

(3) f格式符,以小数形式输出实型数据。

%f:不指定输出数据的宽度,整数部分全部输出,小数部分输出6位。

%m.nf:输出的数据占m列(小数点“.”也占1列),其中有n位小数,如果数值长度小于m,则左端补空格。

%-m.nf:输出的数据占m列(小数点“.”也占1列),其中有n位小数,如果数值长度小于m,则右端补空格。

注意:在用%f输出时要注意数据有效数字的位数。

【实例3-13】 输出实型数据。

```
/*实例3-13*/
#include<stdio.h>
int main()
{
float i=1234.567856789;
double j=1234.567856789;
printf("%f\n%15.2f\n%-10.3f\n%f\n",i,i,i,j);      /*以指定格式输出*/
return 0;
}
```

程序运行结果如图3-16所示。

说明:图3-16第1行的输出结果,理论上应该输出1234.567856,而实际输出1234.567871,这是因为float型数据只能保证6位或7位有效数字,左边第7位数字(也就是第3位小数)以后的数字并不能保证是绝对精确的。

图3-16 输出实型数据

其他常用的格式字符还有 s、e(E)、g(G)、o、x(X)、u 等，其格式字符、格式字符意义、举例和输出结果如表 3-5 所示。

表 3-5　常用的格式字符及意义

格式字符	格式字符意义	举　　例	输出结果
s	输出字符串	char ch[]="helloworld!"; printf("%s\n%10.6s\n%-10.3s\n%1.2s\n",ch,ch,ch,ch);	helloworld □□□□hellow hel he

e(E)	以指数形式输出单、双精度实数	float i=1234.567856789;
		double j=1234.567856789;
		printf("%e\n%15.2e\n%-10.3e\n%e\n",i,i,i,j);
			1.234568e+003
			□□□□□□1.23e+003
			1.235e+003
			1.234568e+003

g(G)	以%f、%e 中较短的输出宽度输出单、双精度实数	double i=12345678954321; printf("%f\n%e\n%g\n",i,i,i);	12345678954321.000000 1.234568e+013 1.23457e+013
o	以八进制形式输出整数(不输出前缀 0)	int a=10; printf("%o",a);	12
x(X)	以十六进制形式输出整数(不输出前缀 0x)	int a=10; printf("%x\n%X\n",a,a);	a A

u	以十进制形式输出无符号整数(正整数)	int m,n;
		m=-1;
		n=32767;
		printf("%d,%d\n",m,n);
		printf("%u,%u\n",m,n);
			-1, 32767
			4294967295, 32767

说明：

(1) 除了格式字符 e、g、x 在使用时可以大写外，其他格式字符必须小写。

(2) 可以在 printf()函数的格式控制字符串中使用转义字符，如\n、\t、\b、\r 等。

(3) 在输出字符“%”时要注意，必须在格式控制中连写两个“%”才能得到预期的结果。例如：

```
printf("%d%%",100);
```

输出结果为：

```
100%
```

下面再举一个综合使用 printf()函数的例子。

【实例 3-14】 printf()函数的综合应用。

```
/*实例 3-14*/
#include<stdio.h>
int main()
{
    int a=123;
    float b=123.456789;
```

```
    double c = 123.456789;
    char d = 'A';
    printf("%d,%5d\n",a,a+1);
    printf("%f,%lf,%6.4f\n",b,b,b);
    printf("%lf,%f,%6.4lf\n",c,c,c);
    printf("%c,%8c\n",d,d);
    printf("%.4s\n","this is my test!");
    printf("%8.4s\n","this is my test!");
    return 0;
}
```

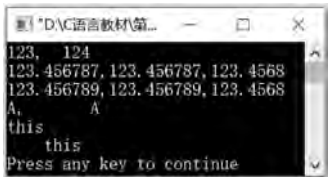


图 3-17　printf()函数的综合应用

程序运行结果如图 3-17 所示。

读者可以自行分析图 3-17 的输出结果。

点拨：使用 printf()函数时还要注意输出表项中的求值顺序。不同的编译系统不一定相同，可能从左到右，也可能右到左。VC++ 6.0 是按从右到左的顺序进行求值的。

3. 输入函数 scanf()

(1) scanf()函数的一般形式。

scanf()函数是格式化输入函数，它从标准输入设备(键盘)读取输入的信息。一般格式为：

```
scanf(格式控制,地址表列);
```

其中，“格式控制”的含义与 printf()函数相同，“地址表列”是由若干地址组成的表列，可以是变量的地址，也可以是字符串的首地址，中间用逗号隔开。地址由地址运算符“&”后面跟变量名组成，例如，&a 和 &b 分别表示变量 a 和变量 b 的地址。下面先看一个数据输入的例子。

【实例 3-15】 scanf()函数的使用。

```
/*实例 3-15*/
#include<stdio.h>
int main()
{
    int a,b,m,n;
    scanf("%d,%d",&a,&b);
    scanf("%d%d",&m,&n);
    printf("a is %d,b is %d\n",a,b);
    printf("m is %d,n is %d\n",m,n);
    return 0;
}
```

程序执行结果如图 3-18 所示。

(a) 正确输入数据　　(b) 错误输入数据

图 3-18　scanf()函数的使用

为什么会出现这样的结果呢？当格式控制中出现“%d,%d”，即两个%d之间用逗号隔开时，输入的数据之间也必须用逗号隔开。当格式控制中出现“%d%d”，即两个%d之间没有符号时，输入的两个数据之间可以用一个或多个空格隔开，也可以用回车键、Tab键隔开。

(2) 格式控制。

格式控制的一般形式为：

```
%[*][域宽][长度]类型
```

其中有方括号[]的项为可选项。

① “类型”是scanf()函数的格式字符，与printf()函数中的格式符相同。

② “*”用来表示该输入项读入后不赋予对应的变量，即跳过该输入值。例如：

```
scanf("%d%*d%d",&a,&b);
```

当输入12 34 56时，把12赋给变量a，34被跳过，56赋给变量b。

③ “域宽”是输入数据的宽度，应该是正整数。例如：

```
scanf("%6d",&x);
```

当输入123456789时，将123456赋给变量x，其余部分截去。又如：

```
scanf("%3d%4d",&x,&y);
```

当输入123456789时，将123赋给变量x，将4567赋给变量y，其余部分截去。

④ “长度”格式符为l和h，l表示输入长整型数据(可以用%ld，%lo，%lx，%lu)和double型数据(可以用%lf，%le)，h表示输入短整型数据(可以用%hd，%ho，%hx)。

(3) 使用scanf()函数要注意的几个问题。

① scanf()函数中要求给出变量地址，所以“&”符号必不可少。例如，下面的语句是错误的：

```
scanf("%d,%d,%d",x,y,z);
```

正确的语句是：

```
scanf("%d,%d,%d",&x,&y,&z);
```

② 如果在格式控制字符串中除了格式声明外还有其他字符，那么，在输入数据时在对应的位置上必须输入与这些字符一样的字符。例如：

```
scanf("x=%d,y=%d,z=%d",&x,&y,&z);
```

那么在输入时必须输入：

```
x=1,y=2,z=3
```

如果输入：

```
1,2,3
```

就会产生错误，系统不会把1,2,3分别赋给变量x,y,z。

③ scanf()函数中没有精度控制。例如，下面的语句是错误的：

```
scanf("%10.2f",&m);
```

④ 在 scanf()函数中用"%c"为字符类型的变量输入值时,空格和转义字符均作为有效字符。例如,执行语句"scanf("%c%c%c",&c1,&c2,&c3);",输入:a□b□c↙,则 c1 得到字符'a',c2 得到字符'□',c3 得到字符'b',其余的输入被丢弃(□表示 1 个空格)。

表 3-6 是 scanf()函数的各种输入形式汇总。

表 3-6 scanf()函数的各种输入形式汇总

输 入 语 句	输 入 形 式
scanf("%d,%d",&a,&b);	5,6
scanf("%d%d",&a,&b);	5□6 或 5□□□6 或 5(Tab 键)6 或 5(回车键)6
scanf("%d:%d",&a,&b);	5:6
scanf("a=%d,b=%d",&a,&b);	a=5,b=6

前两种是比较常用的输入形式。

3.4.2 字符输入输出函数

虽然使用 printf()和 scanf()函数可以实现字符的输出和输入,但 C 语言还提供了专门的单个字符输出函数 putchar()和输入函数 getchar()。

1. 字符输出函数 putchar()

putchar()函数的功能是输出单个字符到屏幕上。一般形式为:

```
putchar(c)
```

其中,参数 c 可以是字符常量、整型常量、字符变量或整型变量(其值在字符的 ASCII 码范围内),但不能是字符串。

【实例 3-16】 putchar()函数的格式和使用方法。

```
/* 实例 3-16 */
#include <stdio.h>
int main()
{
    char ch = 'a';
    putchar(ch);
    putchar(98);
    putchar('\n');
    putchar(ch + 2);
    putchar('d');
    putchar('\n');
    return 0;
}
```

图 3-19 putchar()函数的使用

程序运行结果如图 3-19 所示。

说明:

(1) putchar()函数不仅能输出普通字符,还可以输出转义字符。

(2) 一个 putchar()函数只能输出一个字符。

2. 字符输入函数 getchar()

函数 getchar()的作用是从键盘读取一个字符,没有参数,函数的值就是从输入设备中

得到的字符。

执行语句 getchar()时，系统等待用户的输入，直到按回车键结束。如果用户输入了多个字符，则 getchar()只取第一个字符，多余的字符(包括回车符)存放到键盘缓冲区中，如果再一次执行 getchar()，则程序直接从键盘缓冲区读入。

【实例 3-17】 getchar()函数的格式和使用方法。

```
/* 实例 3-17 */
#include <stdio.h>
int main()
{
    char ch1,ch2;
    ch1 = getchar();
    ch2 = getchar();
    putchar(ch1);
    putchar(ch2);
    putchar('\n');
    return 0;
}
```

程序运行结果如图 3-20 所示。

图 3-20 getchar()函数的使用

在图 3-20 中，输入了“abcdefg”总共 7 个字符，按回车键后，系统才开始接收数据，只接收了前面的 2 个字符'a'和'b'。

【实例 3-18】 用 getchar()函数从键盘输入一个小写字母，把它转换为大写字母，再用 putchar()函数输出该大写字母。

```
/* 实例 3-18 */
#include <stdio.h>
int main()
{
    char ch1,ch2;
    ch1 = getchar();
    ch2 = ch1 - 32;
    putchar(ch2);
    putchar('\n');
    return 0;
}
```

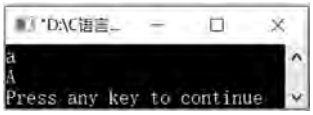


图 3-21 小写字母转换为大写字母

程序运行结果如图 3-21 所示。

在图 3-21 中，输入小写字母 a，按回车键后直接输出大写字母 A。

3.5 运算符与表达式

C 语言的表达式是由运算符将各种类型的变量、常量、函数等运算对象按一定的语法规则连接而成的式子。编译器能够按照运算符的运算规则完成相应的运算处理，求出运算结果，这个结果就是表达式的值。

C语言支持非常丰富的运算符：按照操作对象的个数，运算符可以分为单目运算符、双目运算符和三目运算符；按照功能可以分为算术运算符、关系运算符、逻辑运算符、条件运算符和逗号运算符(关系运算符、逻辑运算符、条件运算符等内容在第4章中详细介绍)。在表达式中，各运算对象参与运算的先后顺序不仅要遵守运算符优先级别的规定，还要受运算符结合性的制约，以便确定是自左向右进行运算还是自右向左进行运算。

3.5.1 算术运算符

1. 基本算术运算符

(1) 单目算术运算符。

单目算术运算符有两个：＋和－，分别是求正和求负运算符。它们只有一个运算对象，并且运算符写在运算对象的左面，结合性是自右向左。

(2) 双目算术运算符。

双目算术运算符是指有两个运算对象的运算符，C语言有5个双目算术运算符。

＋：加法运算符。

－：减法运算符。

＊：乘法运算符。

/：除法运算符。

％：求余运算符、模运算符。

说明：

(1) C语言限定求余运算只对整型运算对象进行，求余运算的结果等于两数相除后的余数，整除时结果为0。例如：7％5的运算结果为2。不能对float或double类型的运算对象进行求余运算。

(2) 除法运算的运算对象均为整型时，结果也为整型，舍去小数；如果运算量中有一个是实型，则结果为双精度实型。例如：4/5的运算结果为0，4.0/5的运算结果为0.8。

(3) 双目算术运算符的结合性都是自左向右。

(4) ＊、/和％三个运算符的优先级相同，＋和－两个运算符的优先级相同，＊、/和％优先级高于＋和－。

2. 自增、自减运算符

自增运算符＋＋的功能是使变量的值自增1，自减运算符－－的功能是使变量的值自减1。自增、自减运算符的运算对象只能是变量，它们的运算优先级高于双目算术运算符，其结合性为自右向左。

假设整型变量i的值为2，自增、自减运算符有以下4个基本形式。

(1) ＋＋i：变量i自增1；表达式“＋＋i”的值为3，即变量i自增后的值。

(2) i＋＋：表达式“i＋＋”的值为2，即变量i原来的值；变量i自增1。

(3) －－i：变量i自减1；表达式“－－i”的值为1，即变量i自减后的值。

(4) i－－：表达式“i－－”的值为2，即变量i原来的值；变量i自减1。

【实例3-19】 自增、自减运算符的应用。

```
/* 实例 3-19 */
#include <stdio.h>
```

```
int main()
{
    int a, i = 1;
    a = ++i;
    printf("a = %d,i = %d\n",a,i);
    a = i--;
    printf("a = %d,i = %d\n",a,i);
    printf("%d\n",i++);
    printf("%d\n",i);
    printf("%d\n",--i);
    printf("%d\n",i);
    return 0;
}
```

程序运行结果如图 3-22 所示。

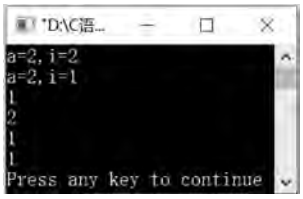


图 3-22　自增、自减运算符的应用

点拨：自增、自减运算的对象是变量。一般不要在一个表达式中对同一个变量进行多次自增或自减运算，以免造成错误的理解或得到错误的运算结果。

3. 算术表达式

用基本算术运算符、自增或自减运算符和圆括号将运算对象(常量、变量、函数等)连接的式子称为算术表达式。单个常量、变量和函数可以看作是表达式的特例。

每个表达式都有一个值及其类型，也即计算表达式所得结果的值和类型。

表达式求值按运算符的优先级和规定的结合方向进行(参见附录 A　运算符的优先级与结合性)。基本算术运算符按先乘除后加减的规则。例如：3＋6＊5 结果为 33，而(3＋6)＊5 结果为 45。

表达式类型规则有以下两点。

(1) 同类型数据运算结果类型不变，如整数与整数运算的结果一定是整数，舍去小数。例如：4/5 的运算结果为 0。

(2) C 语言支持不同类型数据的混合运算，运算结果类型由参与运算的数据决定。不同类型的数据进行运算时，运算结果一般取高一级的数据类型。

思考：表达式 1/2＊2.22 与 1.0/2＊2.22 的值分别是什么？

关于运算符和表达式，需要注意以下问题。

(1) 算术表达式类似数学上的公式，但不能把算术表达式误写成对应的数学公式。例如：3x＋2y 因为少了乘号＊而不是合法的 C 语言算术表达式。

(2) 表达式 a＋b/c＊d 与(a＋b)/(c＊d)的意义完全不同。注意圆括号的使用方法，即使需要多层括号也一律使用圆括号，而不能用中括号或大括号代替。

(3) 利用圆括号可改变运算的优先级，但有时即使无须改变运算的优先级也可利用括号使算术表达式的运算次序清晰、直观。例如：表达式 A/B/C，就不如用 A/(B＊C)更容易理解。

(4) C 语言基本字符集之外的字符不能出现在表达式中，如 π、β、ξ 等。

(5) 程序中将 e 视为变量，而不是数学领域认为的“自然数”；在程序中若需要表示自然数 e，可以调用函数 exp()。

(6) C语言支持不同类型数据的混合运算,但计算机在计算时要将低精度的类型转换为高精度后才进行运算。考虑到运行效率,应尽可能地减少这种转换。例如:"float a,b=3; a=b/2;"就不如"float a,b=3.0; a=b/2.0;"效率高。

3.5.2 赋值运算符

1. 基本赋值运算符和赋值表达式

基本赋值运算符为"=",由"="连接的表达式称为赋值表达式。其一般形式为:

```
变量 = 表达式
```

赋值表达式的功能:先计算赋值号右边的表达式的值,再把该值赋给左边的变量。例如:表达式"c=a+b"的功能是先计算 a+b 的值,再把该值赋给变量 c。

赋值运算符具有右结合性。例如:"a=b=c=5"可理解为"a=(b=(c=5))"。

在C语言中,凡是表达式可以出现的地方均可出现赋值表达式。例如:表达式"x=(a=5)+(b=8)"是合法的。它的意义是把5赋给变量 a,8赋给变量 b,变量 a 和变量 b 相加之和赋给变量 x,故变量 x 等于13。但是,这样的表达式降低了程序的可读性。

按照C语言规定,任何表达式在其末尾加分号就构成为语句。例如:"x=8;"和"a=b=c=5;"都是赋值语句。

关于赋值运算,需要注意以下问题。

(1) 赋值运算符不是数学中的等号,而是进行赋值操作。例如:表达式"a=a+10"的含义是求出 a+10 的值,再把该值赋给变量 a。

(2) 赋值运算符的左侧只能是变量,不能是常量或表达式。例如:假设变量 a 的值是10,"a+10=a*2"虽然在数学上成立,但它不是合法的C语言表达式。

(3) 在程序中可以多次给一个变量赋值,每赋一次值,相应的存储单元中的数据就被更新一次。

2. 复合赋值运算符及复合赋值表达式

在赋值运算符之前加上其他运算符可以构成复合赋值运算符。在C语言中共有10种复合赋值运算符,其中算术类的复合赋值运算符有+=、-=、*=、/=。赋值运算符的两个符号之间不能有空格。复合赋值运算符与基本赋值运算符的优先级相同,结合性为右结合。

由复合赋值运算符连接的式子称为复合赋值表达式。其一般形式为:

```
变量  复合赋值运算符  表达式
```

其求值过程如下。

(1) 求出右边表达式的值。

(2) 右边表达式的值与左边的变量值进行运算。

(3) 将(2)中的运算结果赋给左边的变量。

例如:x*=y+1 相当于 x=x*(y+1)而不是 x=x*y+1。x+=x-=x 相当于 x=x+(x=x-x)。

【实例 3-20】 复合赋值运算符的应用。

```
/* 实例 3-20 */
#include <stdio.h>
```

```
int main()
{
    int a;
    printf("please input:");
    scanf("%d",&a);
    a+=a*=a/=a-6;
    printf("the result is:%d\n",a);
    return 0;
}
```

程序运行结果如图 3-23 所示。

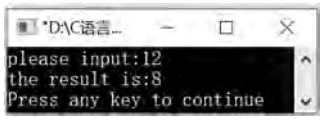


图 3-23　复合赋值运算符的应用

点拨：复合赋值运算符有利于提高编译效率并产生质量较高的目标代码，但在使用过程中也要注意程序的可读性。

3.5.3　逗号运算符

在 C 语言中，逗号“,”也是一种运算符，称为逗号运算符。逗号运算符的优先级是所有运算符中最低的，结合性为左结合。

用逗号表达式将两个或多个表达式连接组成一个表达式，称为逗号表达式。其一般形式为：

```
表达式 1,表达式 2,…,表达式 n
```

逗号表达式的求值过程是：先求表达式 1 的值，再求表达式 2 的值，依次类推，最后求表达式 n 的值。而表达式 n 的值即作为整个逗号表达式的值。

【实例 3-21】　逗号运算符的应用。

```
/*实例 3-21*/
#include<stdio.h>
int main()
{
    int a=1,b=2;
    a=(a=a+b,a=a*b,++a);
    printf("a=%d\n",a);
    return 0;
}
```

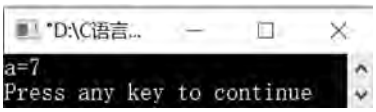


图 3-24　逗号运算符的应用

程序运行结果如图 3-24 所示。

说明：

(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，并不一定是要求整个逗号表达式的值。例如：假设变量 a、b 和 c 的值分别是 1、2 和 3，执行语句“a=a*2，b=b*2，c=c*2;”的结果是分别给变量 a、b 和 c 重新赋值为 2、4 和 6。其中的逗号表达式“a=a*2，b=b*2，c=c*2;”的值虽然求出是 6，但语句执行完之后，这个值也被丢弃。

(2) 并不是在所有出现逗号的地方都组成逗号表达式，如在变量说明语句(例如：int a，b,c;)中和函数参数表(例如：pow(20,3))中逗号只是用作各变量之间的间隔符。

3.5.4 指针运算符

与指针相关的运算符总共有两个：一个是取地址运算符(&)，另一个是间接寻址运算符(*)(也称“指针运算符”)。它们都是单目运算符。

1. 取地址运算符

取地址运算符(&)是单目运算符，用它构成表达式的一般形式是：

```
& 变量名
```

利用取地址运算符 & 可以取得变量在内存中的地址。例如：

```
int x, * px;
x = 8;
px = &x;                                /* 取变量 x 的地址给指针变量 px */
```

【实例 3-22】 取地址运算符的应用。

```
/* 实例 3-22 */
#include <stdio.h>
int main()
{
    int x, * px;
    x = 8;
    px = &x;
    printf("x = %d\n",x);
    printf("* px = %d\n", * px);
    printf("px = %p\n",px);          /* 输出变量 x 在内存中的地址 */
    return 0;
}
```

程序运行结果如图 3-25 所示。

图 3-25 取地址运算符的应用

说明：格式符%p 以十六进制形式输出变量的地址。

【实例 3-23】 用指针求两个整数中的较大数。

```
/* 实例 3-23 */
#include <stdio.h>
int main()
{
    int a, b, * pmax = 0;
    printf("please input:\n");
    scanf("%d,%d",&a,&b);
    pmax = a > b?&a:&b;
    printf("%d\n", * pmax);
    return 0;
}
```

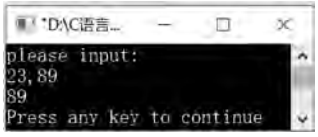


图 3-26 用指针求两个整数中的较大数

程序运行结果如图 3-26 所示。

说明：

(1) 语句“pmax=a>b?&a:&b;”的作用是如果 a>b，则把变量 a 的地址赋值给指针变量 pmax，也就是让指针变量 pmax 指向变量 a，否则让指针变量 pmax 指向变量 b。

(2) 语句“printf("%d\n", * pmax);”中的 * pmax 的含义是指针变量 pmax 指向的变量的内容，也就是所指向的变量的值。

2. 指针运算符

使用指针运算符(*)可以间接访问指针变量所指向的变量，访问过程是先取得指针变量所存放的变量地址，根据变量地址找到需要访问的变量，其使用格式为：

```
* 指针变量名
```

【实例 3-24】 指针运算符的应用。

```
/ * 实例 3-24 * /
# include < stdio.h >
int main()
{
    int a, * pa;
    float b, * pb;
    pa = &a;
    pb = &b;
    a = 3;                          / * 或 * pa = 3; * /
    b = 4;                          / * 或 * pb = 4; * /
    printf("a = %d, * pa = %d\n",a, * pa);
    printf("b = %3.1f, * pb = %3.1f\n",b, * pb);
}
```

程序运行结果如图 3-27 所示。

说明：

(1) 语句“pa＝&a;”是让指针变量 pa 指向变量 a。

(2) * pa 是先通过指针变量 pa 所存放的变量 a 的地址，找到变量 a 对应的内存单元，然后对其进行取值或赋值等操作。

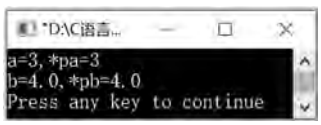


图 3-27 指针运算符的应用

3.6 数据类型转换

在 C 语言表达式中，允许对不同类型的数据进行某一操作或混合运算，当不同类型的数据进行操作时，应当首先将其转换成相同的数据类型，然后进行操作。数据类型转换有赋值类型转换、自动类型转换和强制类型转换三种形式。

3.6.1 赋值类型转换

当赋值运算符两边的运算对象类型不同时，编译器自动将赋值运算符右侧表达式的类型转换为左侧变量的类型，具体转换规则如下。

1. 实型与整型

将实型数据(单、双精度)赋值给整型变量时，舍弃实型的小数部分，只保留整数部分。例如，执行语句“int a＝5.9;”，变量 a 得到的是整数 5。

将整型数据赋值给实型变量，数值大小不变，只是把形式改为实型形式，即小数点后带若干 0，然后赋值。

2. 单、双精度实型

float 型数据只是在尾部加 0 延长为 double 型数据参加运算,然后直接赋值。double 型数据转换为 float 型时,通过截尾数来实现,截断前要进行四舍五入操作。

3. char 型与 int 型

int 型数据赋给 char 型变量时,只保留其最低 8 位,舍弃其他位。

例如,执行语句"char a=366; printf("%x, %x", 366, a);",输出结果为 16e,6e,只取低 8 位。

char 型数据赋给 int 型变量时,通常 int 型变量得到的是其 ASCII 码值,而有一些编译程序在转换时,若 char 型数据的 ASCII 码值大于 127,即作为负数处理。

例如,执行语句"int a='x'; printf("%d",a);",输出结果为 120,即小写字母 x 的 ASCII 码值。

4. short 型与 int 型

short 型占 2 字节,int 型数据赋给 short 型变量时,将低 16 位值赋给 short 型变量,而将高 16 位截断舍弃。将 short 型数据赋给 int 型变量时,其外部值保持不变,而内部形式有所改变。

5. 无符号整数

将一个 unsigned 型数据赋给一个占据同样长度存储单元的整型变量时,原值照赋,内部的存储方式不变,但外部值却可能改变。将一个非 unsigned 整型数据赋给长度相同的 unsigned 型变量时,内部存储形式不变,但外部表示时总是无符号的。

需要说明的是,赋值类型转换是编译器自动进行的,所以也可归入自动类型转换。

3.6.2 自动类型转换

自动类型转换在编译时由编译程序按照一定规则自动完成,不需要人为干预。C 语言要求同一运算的运算对象的数据类型必须相同。因此,在表达式中如果有不同类型的数据参与同一运算,则编译器在编译时自动按照规定的规则将其转换为相同的数据类型。转换规则如图 3-28 所示。

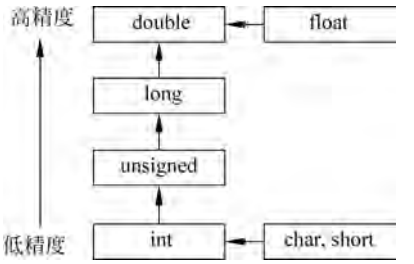


图 3-28 常见类型转换规则

说明:

(1) 图 3-28 中横向箭头表示必需的转换,如两个 float 型数据参加运算,虽然它们类型相同,但仍要先转成 double 型再进行运算,结果为 double 型。

(2) 纵向箭头表示当运算符两边的运算数为不同类型时的转换,如一个 long 型数据与一个 int 型数据一起运算,先自动将 int 型数据转换为 long 型,然后两者进行运算,结果为 long 型。

(3) 如果一个运算符两边的运算对象类型不同,一般是低精度类型数据转换为高精度类型,然后进行运算,这样不会降低运算的精度。

(4) 当低精度类型的数据转换为高精度类型时,一般只是形式上有所改变,不影响数据内容。

【实例 3-25】 自动类型转换与赋值类型转换。

```
/* 实例 3-25 */
#include <stdio.h>
int main()
{
    int a = 1;
    float b = 2.9;
    double c = 3.8;
    a = a + b + c;
    printf("a = %d\n",a);
    return 0;
}
```

程序运行结果如图 3-29 所示。

图 3-29 自动类型转换与赋值类型转换

说明:

(1) 在 Visual C++ 6.0 中编译时,语句"float b=2.9;"会产生警告"'initializing' : truncation from 'const double ' to 'float'"。这是因为对于没有专门加后缀 f 或 F 的实型常量 2.9,C 语言是按 double 型数据来处理的。把 double 型的常量赋值给 float 型的变量时,编译器自动进行数据类型转换,为了避免转换过程中出现精度降低的情况,给出这个警告。

(2) 在 Visual C++ 6.0 中编译时,语句"a=a+b+c;"会产生警告" '=' : conversion from 'double' to 'int', possible loss of data"。这是因为表达式 a+b+c 的值的类型为 double 型,而把一个 double 型的值赋给一个 int 型的变量 a 时,会丢弃小数部分。

(3) 语句"a=a+b+c;"的执行过程如下。

① 取 int 型的变量 a 的值 1,把 int 型的 1 转换为 double 型的 1.0。

② 取 float 型的变量 b 的值 2.9,把 float 型的 2.9 转换为 double 型的 2.9。

③ 求出转换后的两个 double 型的运算对象 1.0 与 2.9 的和,结果也为 double 型的 3.9。

④ double 型的 3.9 和 double 型的变量 c 的值 3.8 相加,得出最后 double 型的结果 7.7。

⑤ 丢弃 double 型的结果 7.7 的小数部分,只把其整数部分 7 赋值给 int 的变量 a。

点拨: 参与运算的某个变量如果要进行自动类型转换,这个转换只是在运算过程中发生,并不会改变该变量的类型。所以执行语句"a=a+b+c;"的过程中,变量 a 和 b 的类型仍然分别是 int 和 float 型。

3.6.3 强制类型转换

强制类型转换的方法是在被转换对象(或表达式)前加类型标识符,其格式为:

```
(类型标识符) 运算对象
(double) x
(int) (m+n)
(float) (5%2)
```

例如:表达式(int) 1.681 的值为 1。

要特别注意的是,强制类型转换的对象是表达式的值,而不是被转换变量本身,被转换变量的类型是不变的,例如:

```
int x=8;
double y=(double) x;
```

在执行上面两条语句后,变量 x 的类型仍然是整型。

【实例 3-26】 强制类型转换。

```
/*实例 3-26*/
#include<stdio.h>
int main()
{
    float f=8.56;
    printf("(int)f=%d,f=%f\n",(int)f,f);
    return 0;
}
```

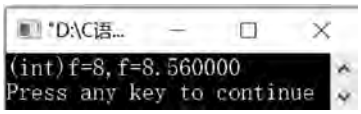


图 3-30 强制类型转换

程序运行结果如图 3-30 所示。

说明:从图 3-30 的运行结果可以看出,表达式(int)f 的类型是整型,而 f 的类型仍然是实型。

强制类型转换的运算对象是紧随其后的运算对象,如果要对整个表达式的值进行类型转换,必须给表达式加圆括号。例如:表达式(int)(0.681+0.432)的值为 int 型的 1,表达式(int)(0.681)+0.432 的值为 double 型的 0.432。

3.7 应用实例

理解常量和变量的概念,掌握 C 语言的常用基本数据类型、运算符和输入输出函数的使用,本节通过一个简单实例,学习如何解决实际问题。

【实例 3-27】 分别输入某个学生的 5 门课程成绩,求该学生的平均成绩并输出结果。

分析:

(1) 根据题目要求确定变量个数,需要 5 个变量保存课程成绩,1 个变量保存平均成绩。

(2) 明确数据类型,每门课程的成绩为整型,可以选择 int 型,平均值为实型,可以选择

float 型。

(3) 命名变量,变量的名字属于 C 语言的标识符,为变量命名要按照标识符的命名规则,不能出现非法的变量名。

程序代码:

```
/* 实例 3-27 */
#include <stdio.h>
int main()
{
    int score1,score2,score3,score4,score5;
    float ave;
    printf("请输入 5 门课的成绩,输入的时候用逗号分开:\n");
    scanf("%d,%d,%d,%d,%d",&score1,&score2,&score3,&score4,&score5);
    ave = (score1 + score2 + score3 + score4 + score5)/5.0;
    printf("The average score is %3.1f \n",ave);
    return 0;
}
```

程序运行结果如图 3-31 所示。

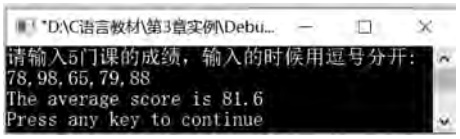


图 3-31　求 5 门课的平均分

思考:编程实现求一个三角形的周长时,需要定义什么类型的变量?共需要定义几个变量?通过键盘输入什么?屏幕上输出什么?

本章小结

本章主要介绍标识符、常量与变量、基本数据类型、常用的输入输出函数、运算符与表达式、数据类型转换等内容。这些都是 C 语言的编程基础,只有在学习过程中深刻理解这些概念,才能在以后的编程过程中正确地使用它们。

C 语言中用标识符区分变量、符号常量、数组、函数等对象。标识符是由数字、字母和下画线 3 种字符组成的,且第一个字符必须为字母或下画线的字符序列。

C 语言程序中的数据一般以常量或变量的形式出现。常量是指在程序执行过程中,其值始终保持不变的量。C 语言中的常量有两种:直接常量和符号常量。

在程序执行过程中,其值可以变化的量称为变量。变量的实质是内存中的一块存储单元,其大小由变量的类型决定。为便于引用,每个变量都有一个名字,通过变量名对其存储空间中的数据进行操作,变量名为合法的标识符。

数据类型决定数据在内存中所占的字节数和存储方式,进而决定数据的取值范围和精度,数据类型还决定了数据运算(操作)的规则,C 语言常用的数据类型有整型、实型和字符型等。

C语言的输入输出函数有很多种,主要有四个最常用的输入输出函数:printf()、scanf()、putchar()和getchar()。特别强调,如果程序中需要使用这几个函数,必须包含头文件stdio.h。

C语言的表达式是由运算符将各种类型的变量、常量、函数等运算对象按一定的语法规则连接而成的式子。编译器能够按照运算符的运算规则完成相应的运算处理,求出运算结果,这个结果就是表达式的值。

C语言支持非常丰富的运算符:按照操作对象的个数,运算符可以分为单目运算符、双目运算符和三目运算符;按照功能可以分为算术运算符、关系运算符、逻辑运算符、条件运算符和逗号运算符等。

在C语言表达式中,允许对不同类型的数据进行某一操作或混合运算,当不同类型的数据进行操作时,应当首先将其转换成相同的数据类型,然后进行操作。数据类型转换有赋值类型转换、自动类型转换和强制类型转换三种形式。

习　　题

1. 填空题。

(1) C语言的标识符只能由字母、_____和_____组成。

(2) 在Visual C++ 6.0中,int型数据分配的字节数是_____,double型数据分配的字节数是_____,char型数据分配的字节数是_____。

(3) 定义符号常量NUM为100的语句为_____。

(4) 设int a=1; float f=3.1; 则表达式10+'a'+3.14*f的值的类型是_____。

(5) 设m是三位的正整数,百位、十位、个位上的数字可分别表示为_____、_____和_____。

(6) 设今天是星期六,计算100天后是星期几的表达式是_____。

(7) 数学上的|x-y|(x和y是实数)的C语言表达式为_____(注:使用函数fabs(x)可以求出实数的绝对值)。

(8) 算术表达式$\frac{5ab}{4c}$的C语言表达式为_____。

(9) 算术表达式$\sqrt{s(s-a)(s-b)(s-c)}$的C语言表达式为_____。(注:使用sqrt(x)可以求出x的算术平方根。)

(10) 设int b=12; 则执行语句"b-=b+=b*b;"后,b的值是_____。

(11) 转义字符中,_____表示换行,_____表示反斜杠,_____表示水平制表。

(12) 设int a=8,b=9; 则表达式a--的值为_____,表达式++b的值为_____。

(13) 字符串常量"Nanhu"在内存中占用的字节数为_____。

(14) 设int a; float y=3.14; 则执行语句"a=(int)y;"后,变量a的数据类型是_____,变量y的数据类型是_____。

(15) 设int a; 逗号表达式a=10,a++,a*a的值为_____。

2. 编程题。

(1) 编写程序,键盘输入某班级男生人数m和女生人数f,计算并输出该班男、女生的

比例(百分比形式,保留两位小数)。

(2) 某个学生的学号为 202201,性别为 m,身高为 185cm,体重为 63.5kg。编写程序,将该学生的信息保存到各个变量中(变量名自拟),并输出该学生的信息。

(3) 编写程序,键盘输入一个十进制的三位正整数,输出其按权展开的形式。例如,输入 678,输出 678=600+70+8。

(4) 假设从 aA 开始,将英文字母按 1～26 编号。编写程序,输入一个序号,输出对应的小写和大写英文字母。例如,输入 5,输出 eE。

(5) 使用指针,编写程序,求两个整数的差。