第 3 章　数据类型及其运算

本章包含 C 语言的一些基本知识和基本概念，包括 C 语言的数据类型、C 语言的数据运算符及表达式等，它们是学习、理解与编写 C 语言程序的基础。

学习重点和难点：

- C 语言的数据类型。
- 常量和变量。
- 运算符和表达式。

学习本章后，读者将对 C 语言处理的数据及其运算有一个全面了解与初步把握。

3.1　本 章 引 例

程序＝数据结构＋算法，数据结构即不同类型数据表示，简单地说就是数据，是算法程序要加工处理的对象，每个程序首先要对处理的数据明确定义并做好数据准备。程序的数据是有不同类型（数据结构）的，对不同类型数据的加工处理（运算等）语言是有合理规定的。**编写程序的目标实际上是对不同类型数据，利用算法加工处理后输出正确结果**。程序运行时，不同类型的数据是分配在内存中的，对内存数据加工处理基本体现在 C 语言会提供哪些运算符，会形成哪些操作表达式。下面是本章的引例。

【例 3-1】 定义不同数据类型并运算输出结果（以计算圆与正方形的面积及周长为例），输出变量与类型内存字节数来认识不同类型数据及其内存分配与使用状况。

```
#include <stdio.h>
#define PI 3.141593                    //预处理命令
int main(void)
{ //①程序变量定义段
  char c1='a',c2,c3;                   //字符变量 c1,c2,c3
  int radius=2,side=4;                 //圆的半径 radius,正方形的边长 side
  double circum1,area1;                //分别定义圆的周长与面积变量
  float circum2,area2;                 //分别定义正方形的周长与面积变量
  int xor;                             //位操作异或运算结果变量
  //②程序执行操作(数据运算等处理)代码段
  c2=c1+3; c3=c1-32;                   //a 后面第 3 个字母赋值给 c2,a 转成大写字母赋给 c3
  circum1=2.0*PI* radius; area1=PI* radius* radius;  //计算圆的周长与面积值
  circum2=4 * (float)side; area2=(float)(side * side); //计算正方形的周长与面积值
  xor=side>>2 ^ radius;    //位操作:side>>2 结果为 1,"1^2"结果为 3 即 0001^0010=0011
  //③程序结果输出代码段
  printf("char 类型占 %d bytes, 变量 c1 占 %d bytes, 变量 c2 占 %d bytes\n",sizeof
(char),sizeof(c1),sizeof(c2));
  printf("int 类型占 %d bytes, 变量 radius 占 %d bytes, 变量 side 占 %d bytes\n",
```

```
sizeof(int),sizeof(radius),sizeof(side));
  printf("double 类型占 %d bytes, 变量 circum1 占 %d bytes, 变量 area1 占 %d bytes\n",
sizeof(double),sizeof(circum1),sizeof(area1));
  printf("float 类型占 %d bytes, 变量 circum2 占 %d bytes, 变量 area2 占 %d bytes\n",
sizeof(float),sizeof(circum2),sizeof(area2));
  printf("字母 %c 后面第 3 个字母是 %c,字母 %c 转成大写字母是 %c\n",c1,c2,c1,c3);
  printf("半径为:%d 的圆的周长与面积分别为:%lf, %lf\n",radius,circum1,area1);
  printf("边长为:%d 的正方形的周长与面积分别为:%f, %f\n",side,circum2,area2);
  printf("边长为:%d 的正方形的周长比半径为:%d 的圆的周长长:%lf\n", side,radius,
circum2-circum1);                    //这里直接输出"circum2-circum1"表达式运算结果
  printf("边长为:%d 的正方形的面积比半径为:%d 的圆的面积大:%lf\n", side,radius,
area2-area1);
  printf("边长值右移 2 位后与半径值进行异或操作的结果为:%d\n",xor);     //位操作
}
```

运行结果：

```
char类型占 1 bytes, 变量c1占 1 bytes, 变量c2 占 1 bytes
int类型占 4 bytes, 变量radius占 4 bytes, 变量side占 4 bytes
double类型占 8 bytes, 变量circum1占 8 bytes, 变量area1占 8 bytes
float类型占 4 bytes, 变量circum2占 4 bytes, 变量area2占 4 bytes
字母 a 后面第3个字母是 d, 字母 a 转成大写字母是 A
半径为: 2 的圆的周长与面积分别为: 12.566372, 12.566372
边长为: 4 的正方形的周长与面积分别为: 16.000000, 16.000000
边长为: 4 的正方形的周长比半径为: 2 的圆的周长长: 3.433628
边长为: 4 的正方形的面积比半径为: 2 的圆的面积大: 3.433628
边长值右移2位后与半径值进行异或操作的结果为: 3
请按任意键继续. . .
```

程序数据与执行代码内存示意图如图 3-1 所示。

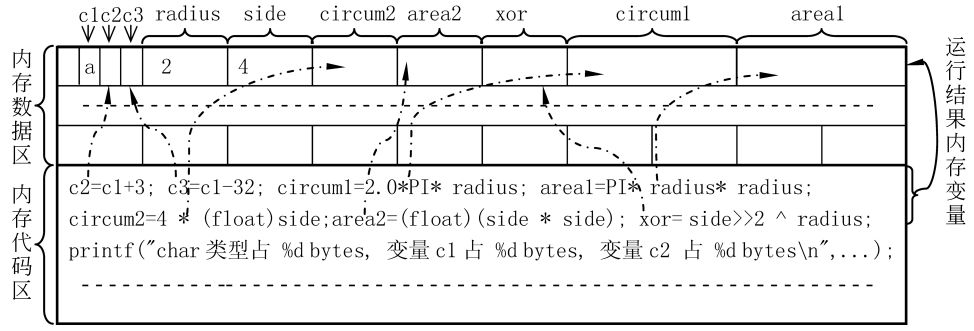


图 3-1 引例程序内存及其操作示意图

3.2 数 据 类 型

程序中使用的各种变量都应预先加以定义，即**先定义，后使用**。对变量的定义可以包括三个方面：数据类型、存储类型、作用域。

本章只介绍数据类型的说明，变量的其他方面的说明将在以后各章中陆续介绍。所谓数据类型是按被定义变量的性质、表示形式、占据存储空间的多少、构造特点等来划分的。在 C 语言中，数据类型可分为基本类型、构造类型、指针类型、空类型 4 大类（有 * 的是 C99

所增加的新类型),如图 3-2 所示。

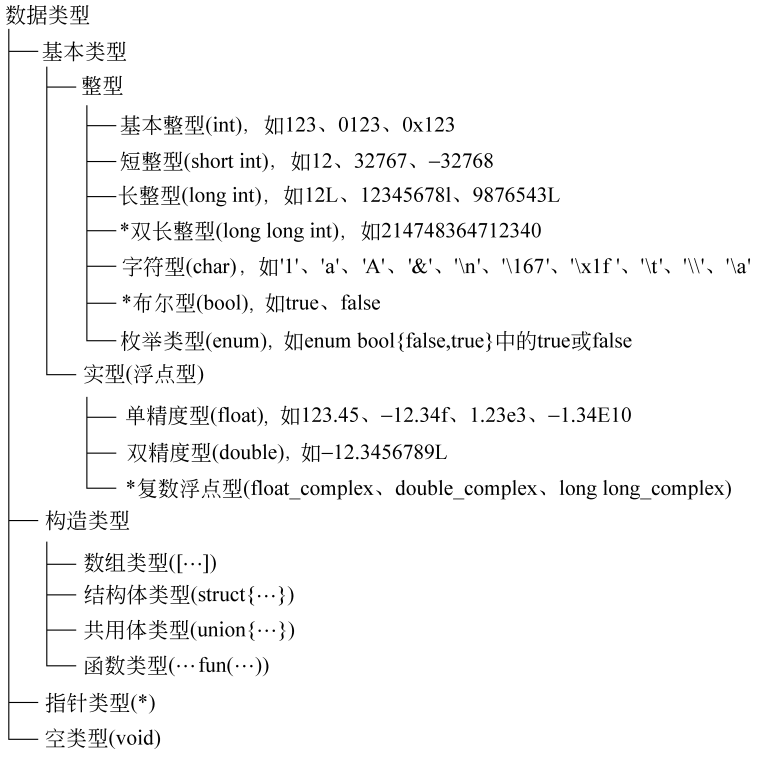


图 3-2　数据类型具体分类

1. 基本类型

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。基本数据类型包括整型与实型(浮点型),其中,枚举类型是用户自定义的整型集合类型。

2. 构造类型

构造类型是由一个或多个数据类型构造而成。也就是说,构造类型可以分解为一个或多个基本类型或小的构造类型。在 C 语言中,构造类型有以下几种:数组类型、结构体类型、共用体(联合)类型、函数类型。函数类型就是用户自定义的函数,函数类型描述了一个函数的接口,包括函数返回值的数据类型和参数的类型等。构造数据类型将在后续章节中详细叙述。

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,其值用来表示某个变量在内存储器中的地址。虽然指针变量的取值类似于整型量(如变量 i 分配的内存地址为 1245052),但这是两个类型完全不同的量,因此不能混为一谈,即不能把地址看成整型量或把整型量用作地址。

4. 空类型

空类型是不知道、不明确或不需要利用的一种标识类型。空类型的说明符为“void”。例如在调用函数值时,通常应向调用者返回一个函数值。这个返回的函数值是具有一定的

数据类型的，为此，需要说明该函数为某数据类型。但也有一类函数，调用后并不需要向调用者返回函数值，这种函数可以定义为“空类型”。

可能读者会问，数据为什么要区分类型呢？这个问题从不同类型变量具有不同性质、表示形式、存储空间大小、构造特点、运算种类等来说明，正是由于数据的多样性、不同数据运算处理的不同、所需内存空间的不同等，而不得不区分数据类型以提高程序处理的时间与空间效率、便利性、灵活性等。为此，不同计算机语言都是区分数据类型的，数据类型种类的多少是因语言而不同的。

了解不同的数据类型，关键要把握其数据表示形式、数据取值范围、数据占用内存空间大小、数据允许的运算种类等。实际上，面对具体问题，确定合适的数据类型及个数是编写程序的关键之一。

3.3 常量与变量

对于基本数据类型量，按其取值是否可改变又分为**常量**和**变量**两种（常变量以常量看待）。在程序执行过程中，其值不发生改变的量称为**常量**，其值可变的量称为**变量**。它们可与数据类型结合起来分类。例如，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、枚举常量、枚举变量。在程序中，**常量是可以不经说明而直接引用的，而变量则必须先定义后使用**。整型量包括整型常量、整型变量。

3.3.1 常量

在程序执行过程中，其值不发生改变的量称为常量。常量可分为直接常量、符号常量、常变量（从量不变的角度可归属为常量）等。

1. 直接常量（或字面常量）

整型常量如 12、0、－3；实型常量如 4.6、－1.23；字符常量如'a'、'b'。

2. 符号常量

用来标识变量名、符号常量名、函数名、数组名、类型名、文件名的有效字符序列称为标识符。在 C 语言中，用标识符来表示的常量，称为**符号常量**。

符号常量在使用之前必须先定义，其一般形式为：

```
#define  标识符  常量
```

其中，＃define 也是一条预处理命令（预处理命令都以＃开头），＃define 命令称为宏定义命令（在后面预处理程序中将进一步介绍），其功能是把该标识符定义为其后的常量值。一经定义，以后在程序中所有出现该标识符的地方均代之以该常量值。

习惯上，符号常量的标识符用大写字母，变量标识符用小写字母，以示区别。

3. 常变量

常变量又名只读变量，由 const 声明，格式为：

```
const 类型名 常变量名=常量
```

常变量有符号常量或一般常量的功效，其值一经定义初始化后就一直不变了。它实质上是个变量，但又具有值不变的常量特性。常变量编译时编译器会进行类型检查，这也是与

符号常量的区别之一。

【例 3-2】 符号常量的使用。

```
#define PRICE 30                    //定义符号常量 PRICE
#include<stdio.h>
int main(void)
{
    int total;
    const int num=10;               //定义常变量 num,以后 num 的值不能再改变
    total=num* PRICE;               //两个常量乘运算
    printf("total=%d",total);       //%d 对应于整型量的输出,详见第 4 章
    //return 0;这里注掉 return 0;,表示后面基本都要省略该语句
}
```

用标识符代表一个常量,称为符号常量。符号常量与变量不同,实际上符号常量在编译时,就被全部替换成它代表的常量而不存在了。为此,其值在作用域内不能改变,也更不能被赋值。常变量如普通变量一样分配内存空间而一直存在,只是其值不允许改变而已。

使用符号常量与常变量的好处有:**含义清楚;能做到"一改全改";只读不写**等。

3.3.2 变量

其值可以改变的量称为变量。一个变量应该有一个名字,在内存中占据一定的存储单元(或若干连续的存储单元),存放相应的变量值。**变量定义必须放在变量使用之前**,一般放在函数体的定义与声明部分,即在第一条可执行语句前定义,否则会出现语法错。

对变量要区分变量名、变量值、存储单元(有唯一变量地址)3 个不同的概念。例如变量 sum,它的变量名为 sum,变量值假设为 100,存放 sum 的存储单元的地址(即变量地址,用 &sum 获取与表示)可能为 1 250 000。变量 sum 的 3 个概念如图 3-3 所示。

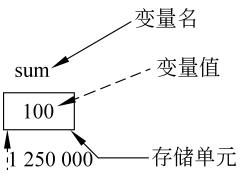


图 3-3 变量 3 个不同概念示意图

编译系统根据变量的数据类型自动为其分配内存空间,并将内存地址与变量名进行关联。编程者根据变量名使用变量,运行时程序根据变量对应的内存地址对数据进行读或写。

3.4 整型数据

整型数据分为整型常量数据与整型变量数据。

3.4.1 整型常量

整型常量就是整常数。在 C 语言中使用的整常数有十进制、八进制和十六进制三种。

1. 十进制整常数

十进制整常数没有前缀,其数码为 0～9。

以下各数是合法的十进制整常数:237、－568、65 535、1627。

以下各数不是合法的十进制整常数:023(不能有前导 0)、23D(含有非十进制数码)。

在程序中是根据前缀来区分各种进制数的，因此在书写常数时不要把前缀弄错造成结果不正确。

2. 八进制整常数

八进制整常数必须以 0 开头，即以 0 作为八进制数的前缀，数码取值为 0～7。八进制数通常是无符号数。

以下各数是合法的八进制数：015(十进制为 13)、0101(十进制为 65)、0177777(十进制为 65 535)。

以下各数不是合法的八进制数：256(无前缀 0)、03A2(包含非八进制数码)、－0127(出现了负号)。

3. 十六进制整常数

十六进制整常数的前缀为 0X 或 0x，其数码取值为 0～9、A～F 或 a～f(代表 10～15)。

以下各数是合法的十六进制整常数：0X2A(十进制为 42)、0XA0(十进制为 160)、0XFFFF(十进制为 65 535)。

以下各数不是合法的十六进制整常数：5A(无前缀 0X)、0X3H(含有非十六进制数码)。

整型常数的后缀：整型常数可以有后缀。在 16 位字长的机器上，基本整型的长度也为 16 位，因此表示的数的范围也是有限定的。如 16 位的基本整型常数，其十进制无符号整常数的范围为 0～65 535，有符号数为－32 768～＋32 767；八进制无符号数的表示范围为 0～0177777；十六进制无符号数的表示范围为 0X0～0XFFFF 或 0x0～0xFFFF。如果使用的数超过了上述范围，就必须用长整型数来表示。长整型数是用后缀“L”或“l”来表示的。例如：

(1) 十进制长整型常数：158L(十进制为 158)、358000L(十进制为 358 000)。

(2) 八进制长整型常数：012l(十进制为 10)、077l(十进制为 63)、0200000L(十进制为 65 536)。

(3) 十六进制长整型常数：0X15L(十进制为 21)、0XA5L(十进制为 165)、0X10000L(十进制为65 536)。

长整型数 158L 和基本整型常数 158 在数值上并无区别。对于 158L，因为是长整型量，C 编译系统将为它分配 4B 存储空间；而对于 158，因为是基本整型，有的系统可能只分配 2B 存储空间。因此在运算和输出格式上要予以注意，避免出错。

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”或“u”。例如，358U、0x38Au、235Lu 均为无符号数。

前缀和后缀可同时使用以表示各种类型的数。例如，0XA5Lu 表示十六进制无符号长整型数 A5，其十进制数为 165。

4. 枚举常量

enum weekday{sun，mon，tue，wed，thu，fri，sat}定义了枚举类型 weekday，枚举类型 weekday 中的 sun，mon，tue，wed，thu，fri，sat 等为枚举常量，实际上其默认值分别为 0、1、2、3、4、5、6。有关枚举类型的详细内容请参阅 10.7 节。

3.4.2 整型变量

1. 整型数据在内存中的存放形式

如果定义了一个整型变量 i：

```
int i; i=10;
```

表示i的值为10。

整型数据是以补码表示的：正数的补码和原码相同；负数的补码，将该数的绝对值的二进制形式按位取反再加1。

例如，10的补码为00000000 00001010；－10的补码为11111111 11110110。若为4字节整数，则10的补码为00000000 00000000 00000000 00001010；－10的补码为11111111 11111111 11111111 11110110。

由此可见，左面的第一位是表示符号的，正数为0，负数为1。

说明：原码表示法在数值前面增加了一位符号位(即最高位为符号位)，该位为0表示正数，为1表示负数，其余二进制位表示数值的大小。例如，假设用8位二进制表示一个整数，＋10的原码为00001010，－10的原码就是10001010；假设用4字节即32位二进制表示一个整数，＋10的原码为00000000 00000000 00000000 00001010，－10的原码就是10000000 00000000 00000000 00001010。

【二维码：整数的机内表示和存储】：★03-01——整数的机内表示和存储.mp4

2. 整型变量的分类

1) 基本型

基本型的类型说明符为int，在内存中占2B或4B。

2) 短整量

短整量的类型说明符为short int或short。所占字节和取值范围均与基本型相同或在**VC++ 2010/6.0**中字节数为基本型的一半(即基本型为4B，短整数为2B)。

3) 长整型

长整型的类型说明符为long int或long，在内存中占4B。

无符号型：类型说明符为unsigned。

无符号型又可与上述三种类型匹配而构成。

(1) 无符号基本型：类型说明符为unsigned int或unsigned。

(2) 无符号短整型：类型说明符为unsigned short。

(3) 无符号长整型：类型说明符为unsigned long。

各种无符号类型量所占的内存空间字节数与相应的有符号类型量相同，但由于省去了符号位，故不能表示负数，而正整数范围扩大1倍。

以VC++ 2010/6.0中4字节整数基本型来说，整数的范围及其变化如下。

(1) 从00000000 00000000 00000000 00000000开始逐个加1变化到01111111 11111111 11111111 11111111，整数范围对应为0～2 147 483 647[0～(2^{31}－1)]。

(2) 从10000000 00000000 00000000 00000000开始逐个加1变化到11111111 11111111 11111111 11111111，整数范围对应为－2 147 483 648～－1(－2^{31}～－2^{0})(注意：是从“最小负数”到－1)。

为此，Visual C++ 2010/6.0中4字节基本型整数的范围为－2 147 483 648～2 147 483 647，其他字节数整数的范围情况类似。

注意：整数按补码方式存储，最大正整数再加1，突变为最小负整数。C语言里整数的

表示范围是有限的，变化范围由存储整数的字节数决定，对于4字节的整数来说，其变化范围如图3-4所示，超出变化范围时就产生整数溢出，超出部分被舍去。

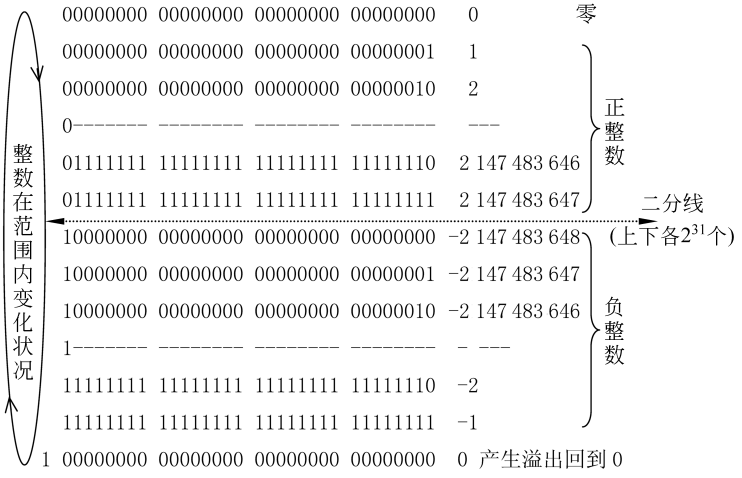


图3-4 C语言整数的变化范围示意图

注意：C语言整数按补码方式存储，最大正整数再加1后得到的数突变为最小负整数。

表3-1列出了Turbo C(或Win-TC)、Visual C++ 2010中各类整型量所分配的内存字节数及数的表示范围。

表3-1 整型类型的数据及其数值范围

整型类型	Win-TC或Turbo C		Visual C++ 2010	
	字节数	数值范围	字节数	数值范围
int	2	−32 768～32 767 [即-2^{15}～$(2^{15}-1)$]	4	−2 147 483 648～2 147 483 647 [即-2^{31}～$(2^{31}-1)$]
unsigned int	2	0～65 535[即0～$(2^{16}-1)$]	4	0～4 294 967 295[即0～$(2^{32}-1)$]
short int	2	−32 768～32 767	2	−32 768～32 767
unsigned short int	2	0～65 535	2	0～65 535
long int	4	−2 147 483 648～2 147 483 647	4	−2 147 483 648～2 147 483 647
unsigned long	4	0～4 294 967 295	4	0～4 294 967 295
long long int	无	无	8	−92 233 720 368 854 775 808～ 922 337 203 685 477 807
unsigned long long	无	无	8	0～18 446 744 073 709 551 615

说明：Turbo C与Win-TC主要是界面不同，其实质基本相同。为此，本书在Win-TC环境中的实践，基本适用于Turbo C环境。

如下程序可以输出各种类型被分配的字节数，不同的C编译环境下程序运行结果会有差异。

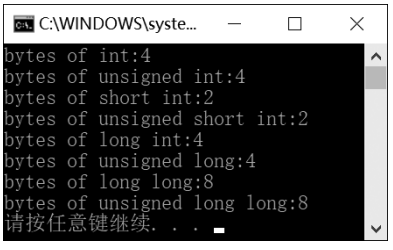
【例3-3】 显示多种类型被分配的字节数。

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

```
int main(void)
{  int i=1;
   unsigned int ui=1;
   short int si=1;
   unsigned short int usi=1;
   long int li=1;
   unsigned long ul=1;
   long long ll=1;                    //C99新增类型,Visual C++2010已支持实现
   unsigned long long ull=1;          //C99新增类型,Visual C++2010已支持实现
   printf("bytes of int:%d\n",sizeof(i));      //输出各变量占用的字节数
   printf("bytes of unsigned int:%d\n",sizeof(ui));
   printf("bytes of short int:%d\n",sizeof(si));
   printf("bytes of unsigned short int:%d\n",sizeof(usi));
   printf("bytes of long int:%d\n",sizeof(li));
   printf("bytes of unsigned long:%d\n",sizeof(ul));
   printf("bytes of long long:%d\n",sizeof(ll));
   printf("bytes of unsigned long long:%d\n",sizeof(ull));
   }
```

说明：这里sizeof(i)或sizeof(int)能显示int类型占用字节数,其他类型也类似。在不同的C语言编译环境中,用sizeof()能把数据类型、运算表达式等占用字节的情况完全搞清楚。

VC++ 2010中运行结果如下：



说明：没有特别说明时,运行结果是指在Visual C++ 2010或6.0中的运行结果,下同。

【例3-4】 领会整数编码方式。

```
#include<stdio.h>
int main(void)
{  //01111111111111111111111111111111->4字节最大正整数,十六进制表示为0x7fffffff
   int i=0x7fffffff;
   //11111111111111111111111111111111->4字节最大无符号正整数,十六进制表示为0xffffffff
   unsigned int ui=0xffffffff;
   //0111111111111111->2字节最大正整数,十六进制表示为0x7fff
   short int si=0x7fff;
   //1111111111111111->2字节最大无符号正整数,十六进制表示为0xffff
   unsigned short int usi=0xffff;
   long int li=0x7fffffff;
   unsigned long ul=0xffffffff;
```

```
    long long int lli=0x7fffffffffffffff;
    unsigned long long ull=0xffffffffffffffff;
    printf("%d \t%d\n",i,i+1);                          //输出最大正整数及其加 1
    printf("%u \t%u\n",ui,ui+1);                        //输出最大无符号整数及其加 1
    printf("%d",si);  printf(" \t\t%d\n",++si);         //输出最大短整数及其加 1
    printf("%u",usi); printf(" \t\t%u\n",++usi);        //输出最大无符号短整数及其加 1
    printf("%ld \t%ld\n",li,li+1);                      //输出最大长整数及其加 1
    printf("%u \t%u\n",ul,ul+1);                        //输出最大无符号长整数及其加 1
    printf("%lld \t%lld\n",lli,lli+1);                  //输出最大长长整数及其加 1
    printf("%llu \t%llu\n",ull,ull+1);                  //输出最大无符号长长整数及其加 1
    }
```

运行结果如下。

```
2147483647      -2147483648
4294967295      0
32767           -32768
65535           0
2147483647      -2147483648
4294967295      0
9223372036854775807     -9223372036854775808
18446744073709551615    0
请按任意键继续. . .
```

说明：运行本程序，能感受到最大正整数加 1 后突变到最小整数的情况，能进一步理解到正、负整数在相应字节范围内的布局情况，也能更好地理解各种整数的表示范围。

为了简洁与节省篇幅，如前只截取运行结果，而不再显示 DOS 运行结果窗体的边框，下同。

3. 整型变量的定义

变量定义的一般形式为：

类型说明符 变量名标识符，变量名标识符，…；

例如：

```
int a,b,c; (a,b,c 为整型变量)
long x,y; (x,y 为长整型变量)
unsigned p,q; (p,q 为无符号整型变量)
```

在书写变量定义时，应注意以下几点。

(1) 允许在一个类型说明符后定义多个相同类型的变量，各变量名之间用逗号(,)间隔，类型说明符与变量名之间至少用一个空格间隔。

(2) 最后一个变量名之后必须以“;”号结尾。

(3) 变量定义必须放在变量使用之前，一般放在函数体的开头部分。

【例 3-5】 整型变量的定义与使用。

```
#include<stdio.h>
int main(void)
{
   int a,b,c,d;                          //定义
   unsigned u;
   a=12;b=-24;u=10;                      //赋值
```

```
    c=a+u;d=b+u;                          //运算
    printf("a+u=%d,b+u=%d\n",c,d);        //输出 a+u=22,b+u=-14
}
```

4. 整型数据的溢出

【例 3-6】 整型数据的溢出。

```
#include<stdio.h>
int main(void)
{
    int a,b;                              //可改为: unsigned int a,b;
    a=2147483647;                         //是 VC++ 2010/6.0 int 最大值
    b=a+1;
    printf("%d,%d\n",a,b);                //可改为: printf("%u,%u\n",a,b);
}
```

运行结果如下。

```
2147483647,-2147483648
```

2 147 483 647 的二进制表示为 01111111111111111111111111111111。

−2 147 483 648 的二进制表示为 10000000000000000000000000000000。

思考：上面例题中带注释语句,做相应替换修改后,运行结果如何？为什么？

【例 3-7】 不同整型类型间的混合运算。

```
#include<stdio.h>
int main(void){
    long x; int a,c;
    x=5; a=7;
    c=x+a;                                //int 与 long 类型变量间运算
    printf("c=x+a=%d\n",c);               //输出 c=x+a=12
}
```

说明：从程序可以看到,x 是长整型变量,a 是基本整型变量。它们之间允许进行运算,运算结果为长整型。但 c 被定义为基本整型,因此最后结果为基本整型。本例说明不同类型的量可以参与运算并相互赋值。其中的类型转换是由编译系统自动完成的。有关类型转换的规则详见 3.9 节。

3.5 实 型 数 据

实型数据分为实型常量数据与实型变量数据两种。

3.5.1 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,实数只采用十进制表示,它有两种形式：**十进制小数形式或指数形式**。

1. 十进制小数形式

由符号位(正号可省略)＋带小数点的数(0～9 和小数点组成)构成。

例如,0.0,25.0,5.789,0.13,5.0,+300.,-267.8230 等均为合法的实数。

注意:必须有小数点。

2. 指数形式

由带小数点的十进制数+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 为带小数点的十进制数或为整数,n 为十进制整数,均带符号)

其值为 $a\times10^n$。例如:

2.1E5(等于 2.1×10^5)　　3e-2(等于 3.0×10^{-2})

0.5e7(等于 0.5×10^7)　　-2.8E-2 (等于 -2.8×10^{-2})

以下不是合法的实数。

345(无小数点,是整型数)　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)

标准 C 允许浮点数使用后缀。后缀为"f"或"F"即表示该数为浮点数。例如,356.0f 和 356.是等价的。例 3-8 说明了这种情况。

【例 3-8】 浮点类型数的输出。

```
#include<stdio.h>
int main(void){
   printf("%f\n ",356.); //356.实数常量作为 double,%f 对应于浮点数的输出,详见第 4 章
   printf("%f\n ",356);        //输出 0.000000,说明整数与%f 不匹配
   printf("%f\n ",356.0f);     //356.0f 为 float
}
```

运行结果如下。

```
356.000000
 0.000000
 356.000000
```

3.5.2 实型变量

1. 实型变量的分类

实型变量分为单精度(**float 型**)、**双精度(double 型)和长双精度(long double 型)**三类。

2. 实型数据在内存中的存放形式

实型数据按数据的指数形式存储。实数在内存中的指数形式存放一般格式如下。

数符	指数部分	小数部分

实数无论是单精度还是双精度在存储中都分为以下三个部分。

(1) 数符或符号位(s):0 代表正,1 代表负。

(2) 指数部分或指数位(e):用于存储科学记数法中的指数数据,并且采用指数移位存储。n 位指数位能表示 0～(2^n-1),为了能表示负指数,往往采用减个偏移量,如减$(2^{n-1}-1)$来实现负指数的表示。而存储时则用实际指数值加偏移量如$(2^{n-1}-1)$来存入指数部分,这

就是所谓的指数移位存储。

(3) 小数部分或尾数部分(f):存放数据科学记数法中的小数数字部分。

指数部分与小数部分的位数因实数占用字节数及C编译系统的不同实现而异,但不论是float还是double,一般在存储方式上都是遵从IEEE(Institute of Electrical and Electronic Engineers,美国电气和电子工程师协会,IEEE被国际标准化组织授权为可以制定标准的组织)规范的。

(1) float遵从的是IEEE R32.24——指数部分占8位,小数部分占23位,如图3-5所示。

(2) double遵从的是IEEE R64.53——指数部分占11位,小数部分占52位,如图3-5所示。

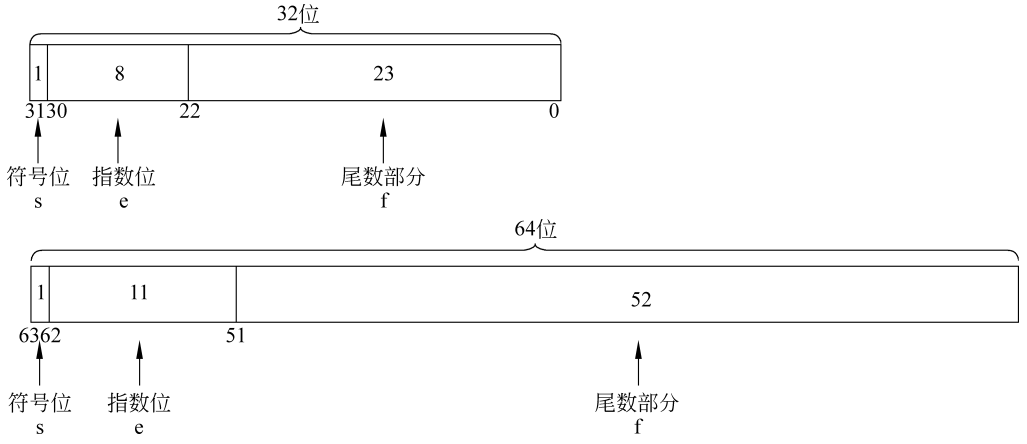


图3-5 32位float与64位双精度double的存储方式

注意:小数部分占的位(bit)数愈多,数的有效数字愈多,精度愈高。指数部分占的位数愈多,则能表示的数值范围愈大。

以单精度数120.5为例,其存储方式说明如图3-6所示。

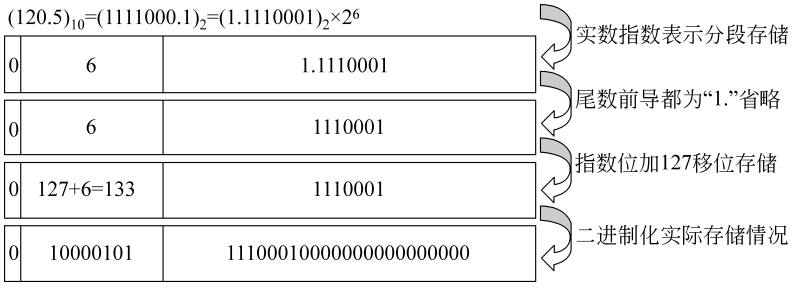


图3-6 单精度数120.5的二进制存储方式

【例3-9】 验证浮点数的存储方式。

```
#include<stdio.h>
int main(void){
   union              //定义共用体变量 u1,详见后续章节
   {  int i;          //int i 与 float f 共用 4 个字节
      float f;        //i 与 f 共用,意味着存入 f,可以以整型数据看待并通过 i 输出
   } u1;
```

```
    u1.f=(float)120.5;
    printf("%d  %o  %x\n",u1.i,u1.i,u1.i);
}
```

运行结果如下。

```
1123090432  10274200000  42f10000
```

上面的程序及其运行结果表明，120.5 的 float **二进制指数存储形式** = $(01000010111100010000000000000000)_2=(1123090432)_{10}=(10274200000)_8=(42F10000)_{16}$。

程序结果验证了实数的二进制指数存储形式。

在 Turbo C(或 Win-TC)中，单精度型占 4 字节(32 位)内存空间，其数值范围为 −3.4E+38～3.4E+38，只能提供 **7 位有效数字**。双精度型占 8 字节(64 位)内存空间，其数值范围为 −1.7E+308～1.7E+308，可提供 **16 位有效数字**，如表 3-2 所示。

表 3-2　实数的二进制指数存储形式

类型说明符	比特数(字节数)	有效数字	数值范围(**绝对值**)
float	32(4)	6～7	0 及 1.18E−38～3.40E+38
double	64(8)	15～16	0 及 2.3E−308～1.7E+308
long double	128(16)	18～19	0 及 3.4E−4932～1.1E+4932

【参考知识】

(1) 理解实数范围是如何计算出来的。

这是很有意义的事。这里不妨以 32 位单精度 float 类型实数数值范围的计算来说明。如上所述 32 位单精度数 x 由符号位 s、指数位 e、尾数 f 三部分组成，其中，s 占 1 位，e 占 8 位，f 占 23 位，那么 x 能表示的规格化的最大正数、最小正数、最大负数、最小负数分别是多少？

需要说明的是，单精度浮点数使用 23 位有效数字。但是，浮点格式假设有效数字的整数部分永远为 1，并且该整数 1 不在 23 位中存放。这样实际上有效数字的精度达到了 24 位。指数使用 8 位，它的范围为 0～255，称为移位指数，意思是必须从指数中减去一个数(称为偏移量或者是偏差值)，对单精度浮点数而言，这个值是 127。当指数是 0 和 255 时，指数有别的含义(即用于特殊用途)，因此实际指数的范围是 1～254，减去 127 移位后，范围是 −126～+127(二进制指数)。

这样，真值 x 可表示为(注意此例不是 IEEE 754 标准定义的格式)：

$$x=(-1)^s\times(1.f)\times 2^{e-127}$$

按照真值 x 与浮点数表示格式之间的上述公式关系，能计算出 x 的数值范围如下。

① 当 s=0，1.f 取最大值$[1+(1-2^{-23})]$(f 为全 1，即 23 个 1)，且 e−127 取最大正数 254−127=127 时，所表示的最大正数为$[1+(1-2^{-23})]\times 2^{127}\approx 3.40\times 10^{38}$。

② 当 s=0，1.f 取最小值 1.0(f 为全 0，即 23 个 0)，且 e−127 取最小负数 1−127=−126 时，所表示的最小正数为 $1.0\times 2^{-126}\approx 1.18\times 10^{-38}$。

③ 当 s=1(为负数)，1.f 取最小值 1.0，且 e−127 取最小负数 1−127=−126 时，所表

示的最大负数为$-1.0\times2^{-126}\approx-1.18\times10^{-38}$。

④ 当s=1(为负数),1.f取最大值[1+(1−2^{-23})],且e−127也取最大正数(2^8-2)−127=127时,所表示的最小负数为−[1+(1−2^{-23})]×$2^{127}\approx-3.40\times10^{38}$。

为此,单精度实数的数值范围如图3-7所示。据此,占8字节的双精度型数的数值范围也容易计算出来了。

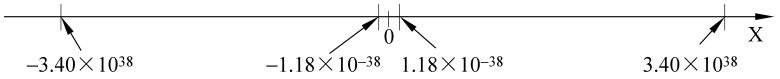


图3-7 单精度实数的数值范围

(2) 理解实数有效数字位数是如何确定的。

上面以单精度8位阶码,1位符号,剩下23位尾数,算出2^{-23}=0.000 000 119 209 289 550 781 25,这是23位尾数能表达的最小小数,意味着不能表达出比其更小的小数了,有效数字位数为6或7位(小数点右第7位为非0数)。双精度11位阶码,1位符号,剩下52位尾数,算出2^{-52}= 0.000 000 000 000 000 222 044 604 925 031 308 084 726 333 618 16,为此,有效数字位数为15或16位(小数点右第16位为非0数)。

3. 实型变量定义

实型变量定义的格式和书写规则与整型相同。例如:

```
float x,y; (x,y为单精度实型量)
double a,b,c; (a,b,c为双精度实型量)
```

4. 实型数据的舍入误差

由于实型变量是由有限的存储单元组成的,因此能提供的有效数字总是有限的。

【例3-10】 实型数据的舍入误差。

```
#include<stdio.h>
int main(void)
{  float a,b;
   a=123456.789e5;
   b=a+20;
   printf("%f\n",a);          //输出: 12345678848.000000
   printf("%f\n",b);          //输出: 12345678848.000000(竟然与输出a的结果相同?)
}
```

说明:从运行结果看,误差明显。注意:C语言中1.0/3*3的结果并不精确等于1。

【例3-11】 浮点数的有效数位。

```
#include<stdio.h>
int main(void)
{
   float a;
   double b;
   a=33333.33333;
   b=33333.33333333333333;
```

```
    printf("%f\n%f\n",a,b);
    printf("%20.10lf\n%20.10lf\n",a,b);
}
```

运行结果如下。

```
33333.332031
33333.333333
    33333.3320312500
    33333.3333333333
```

说明：从本例可以看出，由于 a 是单精度浮点型，有效位数只有 7 位。而整数部分已占 5 位，故小数点右 2 位后之后均为无效数字。b 是双精度型，有效位为 16 位。

但 VC++ 2010/6.0、Turbo C(或 Win-TC)都规定默认情况下小数点后最多保留 6 位，其余部分四舍五入。

5. 实型常数的类型

实型常数(如 3.14)不再区分单、双精度，都统一按双精度 double 型处理。

3.6 字符型数据

字符型数据包括字符常量和字符变量。字符型数据在内存中用一个字节来存放，可分为有符号字符型和无符号字符型两类。字符型数据的存储空间和值的范围如表 3-3 所示。

表 3-3 字符型数据的存储空间和值的范围

类　　型	字节数	取值范围	存储示意图
signed char(有符号字符型)	1	0～127，－128～－1 (即－128～127，即$-2^7 \sim 2^7-1$)	00000000～01111111， 10000000～11111111
unsigned char(无符号字符型)	1	0～255，即 $0 \sim 2^8-1$	00000000～11111111

一个字节的字符型数据存储的值对应某一个字符的编码值，即该字符编码。常见的字符编码有 ASCII、Extended ASCII(EASCII)、ISO-8859-1、ANSI、unicode、GB2312、GBK、UTF-8 等。不支持汉字的 C 语言编译器主要是用 ASCII 或 EASCII，ASCII 与 EASCII 编码表见附录 A。

说明：ASCII(American Standard Code for Information Interchange)用 7 位二进制编码，从 0 到 127，共 128 个常用字符。EASCII 为扩展 ASCII 码，用 8 位二进制编码，从 0 到 255，共 256 个常用字符。

3.6.1 字符常量

字符常量是用单引号(' ')括起来的一个字符。例如：'a'、'b'、'='、'+'、'?'，这些都是合法字符常量。在 C 语言中，字符常量具有以下特点。

(1) 字符常量只能用单引号括起来，不能用双引号或其他括号。

(2) 字符常量只能是单个字符(转义字符表达形式例外)，不能是字符串。

字符可以是字符集中的任意字符。但数字被定义为字符型之后就不能参与数值运算了。如'5'和 5 是不同的。'5'是字符常量，一般不能参与算术运算。若非常规字符与整数运

算(如x=10+'5';),是把字符转换成其ASCII码后再参与运算的。

(3)''不含字符的空字符是不允许使用的,它与'_'、'\0'是完全不同的。

3.6.2 转义字符

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"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。

常用的转义字符及其含义请参见表1-1。

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。表中\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表示的ASCII代码。例如,'\101'表示字母'A','\102'表示字母'B','\134'表示反斜线,'\x0A'表示换行,'\0'表示空字符或字符串结束标志等。

注意:转义字符也要用单引号括起来,而在字符串中的转义字符等都是省略单引号的。

【例3-12】 转义字符的使用。

```
int main(void)        //从篇幅考虑 #include <stdio.h>有时在程序前省略,后同
{
   printf("  ab   c\tde\rf\n");          //注意\t,\r,\n的含义
   printf("hijk\tL\bM\n");               //\b退格后把L删除了
   printf("012x4y8s3\0614w2\n");         //\061为字符'1'。注意:\ddd,d为八进制数字
   printf("012x4y8s3\0674w2\n");         //\067为字符'7'
   printf("012x4y8s3\0684w\1342\n");     //\06为控制字符红桃符,\134为'\'字符
   printf("012x4y8s3\084w2\n");          //\0字符串结束标志,即空字符
   printf("\n");
}  //请注意字符串中转义符的存在情况,请仔细核查运行结果
```

运行结果如下。

```
f ab   c de
hijk    M
012x4y8s314w2
012x4y8s374w2
012x4y8s3♦84w\2
012x4y8s3
```

3.6.3 字符变量

字符变量用来存储字符常量,即单个字符。字符变量的类型说明符是char。字符变量类型定义的格式和书写规则都与整型变量相同。例如:

```
char a,b;
```

3.6.4 字符数据的存储与使用

每个字符变量被分配一个字节的内存空间,因此只能存放一个字符。字符值是以ASCII码的形式存放在变量的内存单元之中的。而显示或打印时才会转换成对应的字符或

控制功能(例如'\n'换行字符产生换行控制效果)。

例如,x 的十进制 ASCII 码是 120,y 的十进制 ASCII 码是 121。对字符变量 a、b 赋予'x'和'y'值:

```
a='x'; b='y';
```

实际上是在 a、b 两个单元内存放 120 和 121 的二进制代码:

```
a 为 0 1 1 1 1 0 0 0
b 为 0 1 1 1 1 0 0 1
```

所以,也可以把它们看成是整型量。C 语言允许对整型变量赋以字符值,也允许对字符变量赋以整型值。在输出时,允许把字符变量按整型量输出,也允许把整型量按字符量输出,即在输出时按要求转成整数或字符。

整型量为 2 或 4 字节量,字符量为单字节量,当整型量按字符型量处理时,只有低 8 位字节参与处理,如下例所示。

【例 3-13】 向字符变量赋予整数值。

```
int main(void)
{  char a,b;
   a=1089;//(1089)10=(00000100 01000001)2,低 8 位=(01000001)2=(65)10='A'的编码
   b=121;
   printf("%c,%c\n",a,b);                    //%c 对应于字符的输出
   printf("%d,%d\n",a,b);                    //%d 对应字符的 ASCII 编码值输出
}
```

运行结果如下。

```
A,y
65,121
```

本程序中定义 a、b 为字符型,但在赋值语句中赋以整型值。从结果看,a、b 值的输出形式取决于 printf 函数格式串中的格式符,当格式符为“%c”时,对应输出的变量值为字符,当格式符为“%d”时,对应输出的变量值为整数。

【例 3-14】 字符型值的输出。

```
int main(void)
{  char a,b;                                 //定义
   a='a';  b='b';                            //赋值
   a=a-32; b=b-32;                           //运算
   printf("%c,%c\n%d,%d\n",a,b,a,b);         //输出
}
```

运行结果如下。

```
A,B
65,66
```

本例中,a、b 被说明为字符变量并赋予字符值,C 语言也允许字符变量参与数值运算,即是用字符的 ASCII 码参与运算。由于大小写字母的 ASCII 码相差 32,因此运算后把小

写字母换成大写字母,然后分别以整型和字符型输出。

3.6.5 字符串常量

字符串常量是由一对双引号括起的字符序列。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。这是与BASIC等语言不同的,但是可以用一个字符数组来存放一个字符串常量,这将在数组一章内予以介绍。

(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符数加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:

① 字符串"C program"在内存中所占的字节(共10字节)为C program\0。

② 字符常量'a'和字符串常量"a"虽然都只有一个字符,但在内存中的情况是不同的。

③ 'a'在内存中占1字节,可表示为a。

④ "a"在内存中占2字节,可表示为a\0。

注意:<string.h>中定义了一系列专门的字符串处理函数。

3.7 变量赋初值

在程序中常常需要对变量赋初值,以便使用变量。C语言程序中可有多种方法为变量提供初值。本节先介绍在进行变量定义的同时给变量赋以初值的方法。这种方法称为初始化。在变量定义中赋初值的一般形式为:

类型说明符 变量1[=值1],变量2[=值2],……;

其中,[=值1]、[=值2],表示定义变量的同时,完成对变量的初始化,[]为可选部分。

例如:

```
int a=3;
int b,c=5;
float x=3.2,y=3f,z=0.75;
char ch1='K',ch2='P';
```

注意:在定义初始化时不允许连续赋值,如"int a=b=c=5;"是不符合语法的。

【例3-15】 整型变量定义时的初始化。

```
int main(void)
{  int a=3,b,c=5;                          //定义并初始化
   b=a+c;                                  //运算
   printf("a=%d,b=%d,c=%d\n",a,b,c);       //输出 a=3,b=8,c=5
}
```

注意:在变量的使用中选取合适的类型非常重要,这将直接影响到编写程序的复杂度

与程序的效率。更要注意数据处理中可能产生的数据溢出(往往是因数据类型选取不当)和数据舍入误差(往往是因数据混合运算),因为这将直接影响程序运行的正确性。

3.8 算术运算符和表达式

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。

C语言的运算符不仅具有不同的优先级,而且还有一个特点,就是它的结合性。在表达式中,各运算量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。这种结合性是其他高级语言的运算符所没有的,因此也增加了C语言的学习难度。

3.8.1 C语言运算符简介

C语言的运算符可分为以下几类。

(1) **算术运算符**——用于各类数值运算。包括加(+)、减(-)、乘(*)、除(/)、求余(或称模运算,%)、自增(++)、自减(--)共7种。

(2) **关系运算符**——用于比较运算。包括大于(>)、小于(<)、等于(==)、大于或等于(>=)、小于或等于(<=)和不等于(!=)6种。

(3) **逻辑运算符**——用于逻辑运算。包括与(&&)、或(||)、非(!)3种。

(4) **位操作运算符**——参与运算的量,按二进制位进行运算。包括位与(&)、位或(|)、位非(~)、位异或(^)、左移(<<)、右移(>>)6种。

(5) **赋值运算符**——用于赋值运算,分为简单赋值(=)、复合算术赋值(+=,-=,*=,/=,%=)和复合位运算赋值(&=,|=,^=,>>=,<<=)3类共11种。

(6) **条件运算符**——这是一个三目运算符,用于条件求值(?:)。

(7) **逗号运算符**——用于把若干表达式组合成一个表达式(,)。

(8) **指针运算符**——用于取内容(*)和取地址(&)两种运算。

(9) **求字节数运算符**——用于计算数据类型或表达式所占的字节数(sizeof)。

(10) **特殊运算符**——有括号()、下标[]、成员(->,.)4种。

这些运算符的基本操作功能、优先级与结合性详见附录B。

3.8.2 算术运算符和算术表达式

1. 基本的算术运算符

(1) 加法运算符"+":加法运算符为双目运算符,即应有两个量参与加法运算,如a+b、4+8等。具有**左结**合性。

(2) 减法运算符"-":减法运算符为双目运算符。但"-"也可作负值运算符,此时为单目运算,如-x,-5等具有右结合性,对应的"+"也可作单目正值运算符,如+x、+5,但可能应用较少,因此附录B运算表中也没列出。

(3) 乘法运算符"*"、除法运算符"/":双目运算符,具有左结合性。

参与运算量均为整型时,以上运算结果也为整型。