第 3 章 C 编程基础知识

本章内容提要

(1) 常量和变量。

(2) 基本数据类型。

(3) 基本运算符和表达式。

(4) 数据的类型转换。

本章介绍的是编程用的基础知识，为后面的编程套路(如选择结构、循环结构等)做准备，就如同学习武术，必须把基本功练好了，才能学套路。

本章主要内容有常量和变量，C 语言的基本数据类型，算术运算符、赋值运算符、自增自减运算符、逗号运算符以及由这些运算符所构成的表达式，数据的类型转换等。


常量和变量

3.1 常量和变量

C 程序中，可以使用的数据分为两类：常量和变量。

3.1.1 常 量

有些数据是“死数”，不可能变化，例如 2，它在任何时候都是 2；再如 3.14，永远都是 3.14。类似这样的不可能发生变化的数据称为常量。C 语言中的常量并非仅限于数值型的“常数”，还包括字符、字符串、符号常量和常变量等。本书在 3.2 节和 3.3 节中会分别讲述这些常量。

3.1.2 变 量

变量的概念特别重要，因为几乎每个程序都要用到变量。能否正确地理解变量将直接决定能否学好 C 语言。

1. 什么是变量

程序之所以需要变量，是因为在程序运行的过程中有些数据需要记住。变量就是用来记住这些数据的。

生活中,人们在求解一个问题时,通常都要记住一些数据,例如,要计算表达式 1+2+…+100 的值,在求解过程中,有两个数据必须记住:一是已经求得的和是多少;二是下一次该加的数是几。

用计算机解题也是如此,也需要记住这些数据。人类用大脑或纸来记录数据,而计算机则是用内存:在内存中开辟一部分空间,把数据化成二进制存放进去,这部分存储数据的空间便是变量,即**变量是内存中的一段存储区域**。至于为什么叫变量,是因为这一段存储空间所存的数据可以改变。例如下面的代码:

```
short a;                //定义变量 a,a 的值不确定
a=2;                    //执行后,a 的值是 2
a=3;                    //执行后,a 的值是 3
```

变量 a 刚分配空间时,其内容是不确定的,执行"a=2;"后,a 中的值变成 2;执行"a=3;"后,a 中的值变为 3。其变化过程如图 3-1 所示。

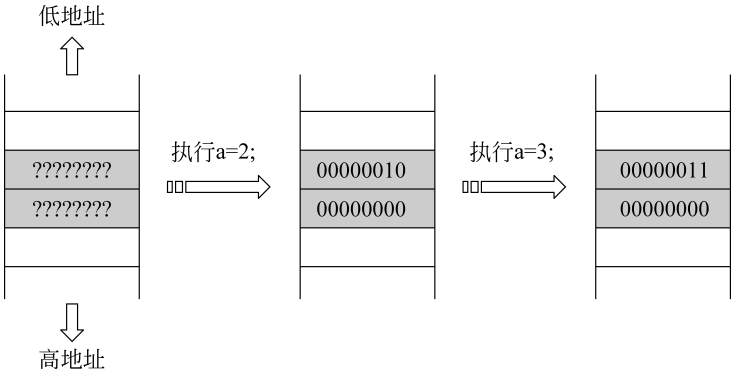

图 3-1 变量的变化过程

图 3-1 中灰色的 2 字节是系统分配给 a 用来存整数的,因为其中所存的数可以改变,所以把这 2 字节称为变量 a。

说明:本书表示内存的图形,一律以图形上方为内存的低地址,下方为高地址,以后不再说明。

说明:几乎所有微机的 CPU 在存数据时都采用小端模式,即先存低字节,再存高字节。

2. 变量的类型

C 语言中的数据类型有很多种,如整型、实型和字符型等。通常,每种类型的数据都要用与它同类型的变量来存储,故变量的类型也分很多种。

表 3-1 是常用的变量类型以及它们在内存空间中所占用的字节数。

表 3-1　常用的变量类型及相关数据

类　型	类型表示	Visual C++ 6.0 中		Turbo C 2.0 中		存储方式
		字节数	表示范围	字节数	表示范围	
字符型	char	1	$0\sim(2^8-1)$	1	0～255	ASCII 码
短整型	short	2	$-2^{15}\sim(2^{15}-1)$	2	−32 768～32 767	补码
整型	int	4	$-2^{31}\sim(2^{31}-1)$	2	−32 768～32 767	补码
长整型	long	4	$-2^{31}\sim(2^{31}-1)$	4	$-2^{31}\sim(2^{31}-1)$	补码
无符号短整型	unsigned short	2	$0\sim(2^{16}-1)$	2	0～65 535	补码
无符号整型	unsigned int	4	$0\sim(2^{32}-1)$	2	0～65 535	补码
无符号长整型	unsigned long	4	$0\sim(2^{32}-1)$	4	$0\sim(2^{32}-1)$	补码
浮点型	float	4	$-3.4\times10^{38}\sim3.4\times10^{38}$	4	$-3.4\times10^{38}\sim3.4\times10^{38}$	IEEE 754
双精度型	double	8	$-1.7\times10^{308}\sim1.7\times10^{308}$	8	$-1.7\times10^{308}\sim1.7\times10^{308}$	IEEE 754

说明：

（1）ASCII 码是为了在计算机中用二进制存储字符而制定的一种编码。本书附录 D 中列有常用字符及其 ASCII 码值。

（2）unsigned 是无符号的意思。C 语言中有些数据不可能是负数，没有必要用最高位表示正负，故 unsigned 类型的数据，其最高位不再表示负号，而是跟后面的位一样代表大小。

（3）float 和 double 型变量的存储方式遵循 IEEE 754 标准，详见 3.2.2 节。

（4）有些编译器还支持 long long、long double 等类型，需要时请自行学习。

除了上面给出的类型，C 语言中还有一种指针类型的变量，这种变量用来存储某种实体的地址，这里所说的实体包括变量、数组和函数等。

存储地址的变量称为指针变量。

指针变量在内存中所分配的字节数一般与 int 型变量相同。

C 语言中，可以用运算符 sizeof()求得某种类型或某项数据在内存中占用多少字节，使用时要在括号中写上类型名或变量名或表达式。例如：

```
char c;
float x;
printf("%d,%d\n",sizeof(char),sizeof(c));
printf("%d,%d\n",sizeof(float), sizeof(x));
```

运行结果：

```
1,1
4,4
```

试一试：自己编写程序验证一下表 3-1 中每种类型的变量需要多少字节的内存空间。

3. 变量的定义

1) 变量定义的格式

普通变量的定义格式:用类型名开头,加一个空格之后再写变量名(由程序员命名)。若定义多个变量,变量之间用逗号隔开。例如:

```
int a;                  //定义一个变量
float x,y,z;            //定义多个变量,变量之间用逗号隔开
```

指针变量的定义格式与普通变量定义格式类似,区别是:指针变量名字前面要多写一个 * ,例如:

```
char *p;                //定义了一个指针变量 p,用来存 char 型变量的地址
int *p1,*p2;            //定义两个指针变量 p1 和 p2,都用来存 int 型变量的地址
float x,*p3,y,*p4;      //定义了两个普通变量 x、y 和两个指针变量 p3、p4
```

变量必须先定义,然后才能使用。

定义变量的目的:一是给变量起一个名字,以便在程序中分辨它;二是把变量的类型告诉计算机,以便让计算机给变量分配空间。因为有了类型,计算机才能知道该给变量分配多少字节,才能知道变量的值用什么方式存储。例如,若是字符变量则分配 1 字节,变量的值用 ASCII 码存储;若是短整型变量则分配 2 字节,用补码存储……

变量定义的位置应该在同级别的执行语句之前(C99 取消了这条规定)。

例如:

```
int main()
{
    int a,b;            //变量定义
    float x;            //变量定义
    a=1;                //第一条执行语句
    ⋮
    if(a>b)
    {
        int s;          //花括号内变量的定义
        s=a+b;          //花括号内第一条执行语句
        …
    }
    ⋮
}
```

2) 变量的命名

不管是变量,还是今后要学到的数组、函数和结构体等,每样东西都应该有一个名字作为标识,其名字即为**标识符**。

C 语言对标识符有如下要求。

(1) 标识符只能由英文字母、数字和下画线组成,但不能以数字开头。

(2) C 语言是区分大小写的,即大小写被认为是两个不同的字符。例如,name 和 Name 是两个不同的标识符。

（3）不允许用关键字作为标识符。关键字是指已经赋予一定含义的字符序列，如 int、float、for、if、return 等。C 语言有 32 个关键字，见附录 C。

（4）标识符有长度限制，超过限制时，后面的字符不起作用。C89 限制的标识符长度是不超过 31 个字符。

注意：C 语言中，变量名不能与函数名相同。例如，已经有函数 max()，则变量名便不能用 max，反之亦然。

4. 变量的属性

每个变量都有值和地址两个属性。

变量的值指的是变量在内存中所存的内容。变量的地址指的是变量在内存中所处的位置，其起始地址称为**变量的地址**。

设有代码“short a=5;”，则程序运行时需要在内存中分配 2 字节作为变量 a 的存储区域，并且将 5 存放进去。设系统给 a 分配的空间是内存中 1027 和 1028 两个单元，如图 3-2 所示，则变量的值是 5，变量的地址是 1027。

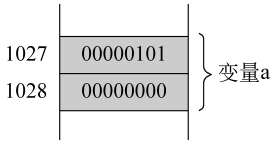

图 3-2　变量的两个属性

把内存的哪 2 字节分配给变量是不可预知的，但是变量分配在什么地方，系统是知道的。每当在内存中给一个变量分配了空间，系统都会把变量名和它的地址、类型等信息记录下来，以便将来找到它、存取它。

因此，在变量获取空间之后，其地址是可以被知道的，用取地址运算符 & 便可以获取变量的地址。

下面的程序可以输出整型变量 a 的两个属性。

```
#include <stdio.h>
int main()
{
    int a=5;
    printf("%d,%p\n",a,&a);          //%p 表示用十六进制数输出地址
    return 0;
}
```

运行结果：

```
5,00,12FF44
```

试一试：把代码中的“=5”去掉，运行一遍程序，看 a 还有没有值。

5. 变量的赋值和赋初值

如前所述，定义变量的目的是为了存储数据，而赋值或赋初值都可以实现这一目的。

1）赋值

在给变量分配空间的任务完成之后，再给变量存放数据，称为赋值。例如，给普通变量赋值：

```
int a;                                //在内存中给 a 分配空间
```

```
a=5*2;                                    //向 a 中存放数据,即赋值
```

其中,=是赋值号,不是数学里的等于号,赋值就是存储:把赋值号右边表达式的值计算出来然后存储到左边变量的内存空间中。赋值后 a 的值是 10。

说明:C 语言中,=是赋值号,==才是等于号。

又如,给指针变量赋值:

```
int a, *p;                                //在内存中给 a 和 p 分配空间
p=&a;                                     //把 a 的地址存到 p 中,即给 p 赋值
```

2) 赋初值

在给变量分配空间的时候就向其中存放数据,称为赋初值。例如:

```
int a=10;
```

赋值和赋初值的区别:开辟空间和存放数据这两件事情若是分两次完成的,是赋值;开辟空间和存放数据这两件事情是一次就完成的,是赋初值。

定义若干变量时,可以只对一个或一部分变量赋初值,例如:

```
int a,b=1,c,d=3,e,f;
```

也可以给全部变量都赋初值,例如:

```
int a=5,b=5,c=5;
```

注意:变量初值相同时,不可以写成:

```
int a=b=c=5;
```


基本数据类型

3.2 基本数据类型

C 语言的数据类型分为基本类型和构造类型,基本类型指的是系统固有的类型,也是常用的类型;构造类型指的是用户自定义出来的类型。

C 语言的基本数据类型如表 3-1 所示。本节介绍这些基本数据类型的表示方法、数据存储方式以及输出方法。

3.2.1 整型数据

本节所说的整型数据包括 short、int、long、unsigned short、unsigned、unsigned long 等所有整数。

1. 整型常量的表示

程序中用到整型常量时,可以用 3 种进制表示:十进制、八进制和十六进制。例如:

```
int a,b,c,d,e;
```

```
a=100;                  //用十进制表示整数
b=0144;                 //用八进制表示整数,必须用 0 开头
c=-0144;
d=0x64;                 //用十六进制表示整数,必须用 0x 或 0X 开头
e=-0x64;
```

上面 5 个赋值语句执行后,5 个变量的值分别是 100、100、－100、100、－100。

注意：程序中不允许使用二进制。

2. 整型数据的存储

所有整数在计算机中都是以补码形式存放的。下面的代码定义了 5 个变量，5 个变量在内存中的存储状态如图 3-3 所示。

```
short a=5,b=-1,c=-32768;
unsigned short d=32768;
long e=65536;
```

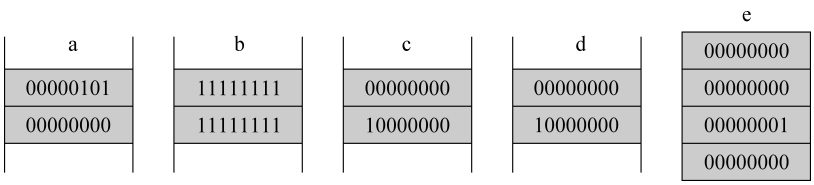

图 3-3　整数在内存中的存储

5 的补码是 00000000 00000101，－1 的补码是 11111111 11111111，－32768 的补码是 10000000 00000000，32768 的补码也是 10000000 00000000，65536 的补码是 00000000 00000001 00000000 00000000。

3. 整型数据的输出

说明：对于数据的输出格式，本书第 4 章会详细介绍，这里先简单介绍一些常用的输出格式。

(1) 带符号整数输出时可以用十进制、八进制或十六进制。例如：

```
int a=76;
long b=65536;
short c=26;
printf("%d\n",a);   //%d 表示用十进制输出整数
printf("%o\n",a);   //%o 表示用八进制输出整数
printf("%x,%X\n",a,a);              //%x 或%X 表示用十六进制输出整数
printf("%ld,%Lo,%lx\n",b,b,b);      //加上 L 或 l 表示输出长整数
printf("%hd,%ho,%hx\n",c,c,c);      //加上 h 表示输出短整数
```

输出结果：

```
76
114
4c,4C
65536,200000,10000
26,32,1a
```

试一试：在 TC 中输出长整数时，漏掉 L 或 l，会怎样？把 65536、65537、65538 用%d 格式输出看看，并解释原因。在 TC 中输出整数时，多加了 L 或 l，又会怎样？解释一下原因。

(2) 无符号整数一般用%u 格式输出，表示用十进制把一个数据当无符号整数来输出。例如：

```
unsigned int a=32768;
long b=50000;
printf("%u,%lu\n",a,b);          //%u 表示输出无符号整数
```

(3) 带符号整数可以当成无符号整数输出，反之亦然。例如：

```
short a=-1;
unsigned short b=65535;
printf("%hd,%hd\n",a,b);         //%hd,输出带符号短整数
printf("%hu,%hu\n",a,b);         //%hu,输出无符号短整数
```

这段代码的输出结果如下：

```
-1,-1
65535,65535
```

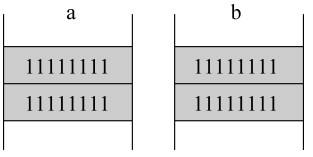

图 3-4 −1 和 65535 的存储状态

想一想：为什么会出现这样的结果？请根据图 3-4所示的存储状态解释其原因。

3.2.2 实型数据

1. 实型常量

1) 实型常量的表示

带小数点的常量称为实型常量。程序中的实型常量可以用两种形式表示。

(1) 小数形式。例如，3.14、−12.5、0.38、.2、−.3 等。

说明：当一个数是纯小数时，小数点前面的 0 可以省略。

(2) 指数形式。例如，1.25E−2、12.5E−3、0.0125E0 等。

指数形式相当于数学中的科学记数法。C 语言用 1.25E−2 这种形式代表数学中的 1.25×10^{-2}。可以看出，上面所列举的 3 个数，其大小是相同的。由此可见，同一个数可以有无限种表示方式。

C 语言规定：用指数形式表示实数时，**E 前面必须有数字，E 后面必须是整数。**

说明：实数只能用十进制表示，上面两种表示方法都是十进制的。

2) 实型常量的类型

实型常量有单精度(float)和双精度(double)两种类型，有效数字分别是 7 位和 15

位，最后一位是近似值。

程序中表示实型常量时，可在实型常量后面加 F 或 f，表示它是单精度型，或者加 L 或 l，表示它是长双精度型。若什么都不加，如 1.2，则系统默认是 double 型。

编程经验：表示长双精度型时，最好用大写 L 而不是小写 l，后者容易被看成是 1。

2. 实型数据的存储

单精度和双精度型的数据，都是以浮点数的方式存储的，遵循 IEEE(Institute of Electrical and Electronics Engineers)754 标准。本书只介绍 float 型数据的存储方式，double 型数据的存储与 float 型类似。

float 型的任何数据，在存储前都必须先表示为下面的格式：

$$(\text{符号})\times M\times 2^{n}$$

其中，n 是指数，M 须满足条件：$1.0\leqslant M<2.0$。例如：

30.0，要先表示为 $+1.875\times 2^{4}$

-0.3925，要先表示为 -1.57×2^{-2}

这之后，计算机用 4 字节，分成三部分分别存储符号、指数部分和小数部分。三部分的位置及所占空间大小如表 3-2 所示。

表 3-2　float 型数据存储空间的分配

符号位(0 或 1)	指数部分(n+127)	小数部分(M−1)
占 1 位[第 31 位]	占 8 位[第 30 位～第 23 位]	占 23 位[第 22 位～第 00 位]

注：表中最右边是第 0 位，最左边是第 31 位。

注意：指数部分存储的是 $n+127$ 而不是 n，小数部分存储的是 $M-1$ 而不是 M。

下面分别说明这三部分怎样存储。

(1) 符号位。占 1 位，用 0 表示正，用 1 表示负。

(2) 指数部分。指数用 8 位存储，本来也有正负的，但是考虑到前面已经有一个正负号了，再设一个符号位不合适，所以 IEEE 754 标准规定：将指数部分加上 127 后再存储，例如，若实际指数 n 为 −2，则存储为 125；若实际指数 n 为 4，则存储为 131。这样规定的目的是，指数加上 127 后不会是负数，故不必设指数的符号位。

(3) 小数部分。用 23 位存储 $M-1$。按照规定，M 满足 $1.0\leqslant M<2.0$，这样，M 的小数点前面将肯定有一个 1，因此存储时就可以不存 1(将 1 默认了)，而只存小数点后面的纯小数 $M-1$，例如，对于 1.875，只存 0.875。这样做可以多存几位小数以提高数据精度。

综上所述，对于 $30.0=+1.875\times 2^{4}$，三部分的数据分别如下。

(1) 符号位：0(表示正)。

(2) 指数部分：10000011(其值为 131，表示实际指数是 4)。

(3) 小数部分：11100000000000000000000(0.875，表示实际小数是 1.875)。

故浮点数 30.0 的实际存储状态如图 3-5 所示。

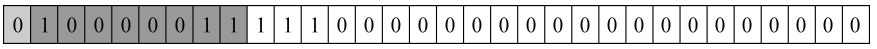

图 3-5　浮点数 30.0 的存储状态

特别注意：实数的存储不都是精确的。

例如，对实数 1.2，其纯小数部分 0.2 化为二进制是一个无限循环小数：00110011 00110011…要想存储这个值，需要无限多的内存单元，而实际上存储纯小数部分的空间只有 23 位，后面部分只能截掉或进位(0 舍 1 入)，所以实际存储的数据是近似值。

1.2 的实际存储状态是 0 01111111 00110011001100110011010(有进位)，它表示的小数是 1.200000047683715820312 5，比 1.2 稍大。

由于有些实数存储不精确，所以在程序中应尽量避免比较两个实型数据是否相等。下面的程序段的输出结果是“不相等”。

```
float x=0.2;
if(x==0.2)
    printf("相等\n");
else
    printf("不相等\n");
```

原因：x 是 float 型变量，存储的是 0.2 的近似值，有 7 位有效数字，而常数 0.2 默认是一个 double 型数据，有 15 位有效数字，它们的大小必然不同。

试一试：运行下面的程序，看运行结果是什么。请解释为什么会出现这样的结果。

```
#include <stdio.h>
int main()
{
    float x=-789.124;
    printf("%f\n",x);
    return 0;
}
```

说明：上面介绍的是 float 型数据的存储方式，double 型数据的存储方式与此类似，只不过指数部分和小数部分的位数更多，分别是 11 位和 52 位。

3. 实型数据的输出

单精度和双精度型数据都可以用%f 或%e(%E)格式输出，%f 格式是用小数形式输出，%e(%E)格式是用指数形式输出。例如：

```
float x=12345678,y=0.00314;
double z=123.456789123456789;
printf("%f, %f, %f\n",x,y,z);          //%f: 小数形式输出
printf("%e, %e, %e\n",x,y,z);          //%e: 指数形式输出
printf("%E, %E, %E\n",x,y,z);          //%E: 指数形式输出,结果中 E 大写
```

该代码段的输出如下：

```
12345678.000000, 0.003140, 123.456789
1.234568e+007, 3.140000e-003, 1.234568e+002
1.234568E+007, 3.140000E-003, 1.234568E+002
```

用%f 时，默认输出 6 位小数(不管有多少位有效数字，总是输出 6 位小数)。

说明：用%f 格式输出时程序员可以用诸如%.3f 这样的方式指定小数位数，参见第 4 章 printf()函数的介绍。

用%e(%E)时，按标准格式输出，即小数点前有且仅有 1 位非 0 的有效数字。而小数点后面有几位小数以及 e(E)后面的指数部分占几位，取决于编译器。

3.2.3　字符型数据

数据不仅仅指数值，还包括字符，字符也是 C 程序中经常要处理的数据。C 语言可以处理的字符有英文字母(大小写)、数字、标点、空格及其他一些符号，见附录 D。

1. 字符常量

单个的字符是字符常量。程序中要表示一个字符常量，不能直接写字符名，因为会引起二义性。例如：

```
int a=1;
char c=a;                    //此处的 a 是变量 a 还是字符 a?
```

为了区分变量和字符常量，C 语言规定：字符必须放在一对单引号之中。例如：

```
char c1='a',c2='A',c3=' ';    //正确
```

这样，C 语言中，单引号就被赋予了一个含义，即它是定界符，用来表示一个字符的前界和后界。也就是说，在 C 语言中，单引号(')已经不是单引号了，而是定界符。

说明：程序中的单引号是不分左右的。

那么，C 语言中若用到单引号，怎么写呢？显然不能写成'''，因为这样写编译器会把它们都认作定界符。为了表示中间的单引号不是定界符，而是单引号，C 语言又做了规定：在中间的单引号前面加上一条反斜线(\)，表示它不是定界符，而是单引号，所以，程序中单引号应该表示成'\''，例如：

```
char ch='\'';                //给变量 ch 存入一个单引号
printf("%c",ch);             //%c 表示要输出一个字符
```

输出结果如下：

```
'
```

反斜线(\)的作用是把后面字符的本来含义转为另外的含义。例如，'\n'，若不加反斜线代表字符 n 本身，加上反斜线后就变成了换行符。

这种用\开头的字符，称为**转义字符**。C 语言中的转义字符很多，表 3-3 列出的是常

用的一些转义字符及作用。

表 3-3 常用的转义字符及作用

转义字符	代表的含义	输出该字符的结果
'\''	一个单引号(')	输出:'
'\"'	一个双引号(")	输出:"
'\\'	一条反斜线(\)	输出:\
'\b'	退格键(Backspace)	光标向前移动一格(退回一格)
'\n'	换行符	光标移动到下行开头
'\r'	回车键(CR)	光标移动到本行开头
'\t'	水平制表符(Tab)	光标移动到下一个 Tab 位置
'\ooo' 如'\101'	一个字符,该字符的 ASCII 码值用八进制表示是 ooo	输出该字符。注:ooo 代表八进制数,最多 3 位
'\xhh' 如'\x41'	一个字符,该字符的 ASCII 码值用十六进制表示是 hh	输出该字符。注:hh 代表十六进制数,最多 2 位

考考你:到目前为止,要表示字符'A',共有 3 种方法,你知道是哪 3 种吗?

2. 字符数据的存储

计算机中只能存储 0 和 1,任何数据都必须先化成 0 和 1 才能存储,字符也不例外。多数计算机都是用“存字符的 ASCII 码值”的方法来存储字符。

基本 ASCII 码表(见附录 D)中只有 128 个字符,加上后来扩充的 128 个,不过才 256 个,所以 C 语言规定:字符用 1 字节存储。

设有如下代码:

```
char c1='A',c2='1';
```

则 c1 和 c2 两个变量在内存中的存储状态如图 3-6 所示。

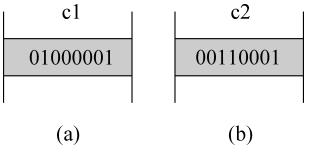

图 3-6 字符数据的存储

想一想:整数 1 在内存中怎样存储,与字符'1'是一样的吗?它们的十进制数分别是多少?

3. 字符数据的大小

由于字符在计算机中实际存储的是其 ASCII 码值,是个整数,所以,C 语言认为字符也有大小,其 ASCII 码值就是它的大小。

因此,字符数据既可以用作字符,也可以用作整数。给字符变量赋值时,既可以赋字符,也可以赋整数。例如:

```
char c1='A',c2=65,c3=' ',c4='1';
printf("%c,%c,%c,%c\n",c1,c2,c3,c4);          //%c 表示要输出一个字符
printf("%d,%d,%d,%d\n",c1,c2,c3,c4);          //字符可当作整数输出
printf("%d,%d,%d\n",c1+c2,c1+1,'A'+1);        //字符可参与运算
```

运行结果如下：

```
A,A, ,1
65,65,32,49
130,66,66
```

把整数当成字符输出也可以，只要不超过 255。例如，printf("%c", 65)，结果是 A。

4. 字符数据的输出

如前面代码所示，输出字符型数据，可以用 printf()函数(%c 或%d 格式)。除了 printf()函数之外，还可以使用 putchar()函数，putchar()函数的使用方法将在第 4 章中介绍。

3.2.4　字 符 串

程序中有时候需要用到一串字符，即字符串，而不是一个字符。

1. 字符串的表示

C 语言规定，字符串必须用一对双引号括起来，如"ab c"、"12"、" "、"A"。双引号中可以没有字符，如""，表示一个空串。

说明：程序中的双引号也是不分左右的。

C 语言中只有字符串常量，没有字符串类型的变量。

2. 字符串的存储

虽然没有字符串变量，但字符串在处理时，仍要在内存中找空间存储。

存储字符串时，总是先把双引号中的每个字符按顺序存储到内存中(连续存放)，然后再在后面多存一个空字符('\0')。例如，"AB"在内存中占 3 字节，存储状态如图 3-7 所示。

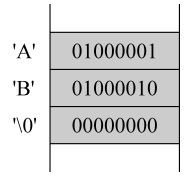

图 3-7　字符串的存储

空字符的 ASCII 码是 0，表示为'\0'。

之所以最后要多存一个空字符，是为了给字符串加一个结束标志，不然，将来使用字符串时，不知道字符串是到哪结束的。

注意：空字符和空格并不是同一个字符，空字符的 ASCII 码是 0，空格的 ASCII 码是 32，它们的存储状态不同，数值大小不同，作用也不同。

考考你："A"和'A'是否相同？若不同，区别是什么？

3. 字符串的输出

输出字符串时，printf()函数中要使用%s 格式。例如：

```
printf("%s%s%s%s%s\n","ABCD"," ","123","","xyz");
```

输出结果如下：

```
ABCD 123xyz
```

除了 printf()函数外,还可以使用 puts()函数输出字符串,puts()函数的使用方法将在 10.3 节中介绍。

3.3 符号常量和常变量

除了前面已经讲述过的几种常量外,C 语言中还有两种常量:符号常量和常变量。

3.3.1 符号常量

为方便编程,增加程序的可读性,程序中经常需要定义符号常量。例如:

```
#define PI 3.141593
```

其中的 PI 称为符号常量,它代表后面的 3.141593。

说明:其实这是一条编译预处理命令,称为宏定义,参见第 9 章。

注意:符号常量的定义是一条命令,不是语句,故后面不需要有分号。

定义符号常量之后,程序中用到圆周率时,既可以写 3.141593,也可以写 PI,如"s=PI*r*r;"。显然,使用后者更方便,这便是定义符号常量的第一个好处。

定义符号常量的第二个好处是便于修改程序。例如:

```
#define NUM 60
```

用 NUM 代表人数 60。假设程序中很多地方都用到这个 NUM,当人数发生变化时,例如少了一个人,只需要把上面代码中的 60 改为 59 即可。若不用符号常量,程序中都写成了 60,当人数减少时,需要修改多处源代码。

定义符号常量还有一个好处:增加程序的可读性。若写成 60,阅读程序的人看到 60 并不一定把它当成人数,还可能把它当做年龄、体重、分数等,写成 NUM 则不易引起误解。

符号常量只是个符号,它不是变量,内存中没有它的空间,所以不能赋值,也不能这样定义:

```
#define PI=3.141593          //错误的符号常量定义
```

符号常量名通常使用大写字母。

3.3.2 常变量

有些 C 编译器中允许定义常变量(有些书上称为常量),常变量的定义方法如下:

```
const int n=60,m=50;          //定义两个常变量并初始化
const float x=3.14;           //定义一个常变量并初始化
```

常变量定义要用 const 开头,后面部分与变量的定义类似,只不过要初始化。

注意:常变量在定义时必须初始化。

说明：常变量的定义是一条语句，后面有分号。

常变量其实也是变量，也在内存中分配空间(同时要初始化)，但初始化后就不允许再变了。因为不能变，具有常量的特点，故称为常变量。

常变量只可以赋初值，不能赋值，因为赋值就等于是改写。

下面的代码有两处语法错误：

```
const float x;                    //错误,常变量定义未赋初值
const int a=1;
a=1;                              //错误,常变量不允许赋值
```

定义常变量之后，程序中可以随时使用它，但不能改变它。

运算符和表达式以及类型转换

3.4　运算符和表达式

本节介绍 C 语言中最基本的运算符和表达式。

3.4.1　算术运算符

算术运算是最常用的运算，C 语言中的算术运算与数学中的算术运算不尽相同。

1. 算术运算符

算术运算符有 7 个：+(正号)、-(负号)、*(乘)、/(除)、%(求余)、+(加)、-(减)。

说明：C 语言中，乘号用 * 表示，且不可省略。例如，a * b 不能写成 ab。

%是求余运算符，它用来求出两个整数相除之后的余数。例如，8%3 的值是 2，20%7 的值是 6。

说明：求余运算的结果，其符号应与%前面那个数的符号相同。例如，-5%3 的结果是-2，而 5%-3 的结果是+2。

求余运算符要求参与运算的两个量必须都是整数。其余运算符对此没有要求，参与运算的数可以是整数，也可以是其他类型的数据。

2. 算术运算符的目数

7 个算术运算符中，+(正号)、-(负号)都是单目运算符，剩下 5 个都是双目运算符。单目运算符是指它只需要一个运算量(即操作数)，例如-5，只需要在负号后写一个数，前边不需要，故它是单目运算符。双目运算符则需要两个运算量，例如，a+b，加号前后各需要一个运算量。

3. 算术运算符的优先级

算术运算符中，+(正号)、-(负号)的优先级别最高，*、/、%的优先级别次之，+(加号)、-(减号)的优先级别最低(参看附录 E)。

根据优先级可知,表达式－5 * －2＋2%3 与 ((－5) * (－2))＋(2%3)运算次序相同。

算术运算时,当两个运算符的优先级别不同时,先运算哪个取决于优先级;当两个运算符优先级别相同时,先运算哪个取决于它们的结合性。

4. 算术运算符的结合性

算术运算符中,两个单目运算符的结合性都是自右至左,简称右结合性。其余运算符的结合性都是自左至右,称为左结合性。

左结合性是指:当两个运算符优先级别相同时,要先算左边的。例如,20 * 3%7,应先算左边的乘法,相当于是(20 * 3)%7,而不是 20 * (3%7),两者结果不同。右结合性是指:当两个运算符优先级别相同时,要先算右边的。例如,－＋2,相当于是－(＋2)。

考考你:既然＋和 * 都是左结合性,那为什么 a＋b * c 要先算 b * c?

关于算术运算,需要特别注意的是,**两个整数运算,最终结果还是整数**。例如,9/5 的结果是 1,1/2 的结果是 0,－5/3 的结果是－1。结果为负时,多数机器采用向零取整的方法将小数截掉,不四舍五入。

说明:向零取整是指,尽量使取整后的结果绝对值更小,离 0 更近。比如－9/5,可以是－1,也可以是－2,但－1 的绝对值更小,故取整后值为－1。

试一试:已知华氏温度到摄氏温度的转换公式是 c＝5/9 * (f－32)。下面程序段用来计算摄氏温度,其中华氏温度 f 是由键盘输入的,请写出完整的程序并运行之,看结果是否正确。若不正确,找出原因。

```
int f,c;
scanf("%d",&f);          //输入整数作为华氏温度
c=5/9 * (f-32);
printf("%d\n",c);
```

提示:上面程序中,若想让 5/9 的结果是实数,应该写成 5./9(或 5/9.),让其中一个数变成实数,其结果就是实数(带“.”的数,系统默认是 double 型)。

3.4.2 赋值运算符和赋值表达式

1. 赋值运算符

赋值运算符就是＝,表示“存储”,即把赋值号右边表达式的值存给左边的变量,3.1 节已做过一些介绍,这里再补充三点。

(1) 左值的概念。

可以出现在赋值号左边的式子,称为**左值**(lvalue,即 left value)。左值必须有内存空间且允许赋值。常用的左值是变量,但常变量不是左值。例如:

```
int a=1;
```

```
const int b=2;
a=2;                    //变量作为左值,正确
b=20;                   //语法错误:常变量不是左值
```

(2) C 语言中还有一些**复合的赋值运算符**,表 3-4 列出的是 5 个与算术运算有关的,还有 5 个与位运算有关的,将在第 15 章中介绍。

表 3-4　复合的赋值运算符及含义

运算符	举　例	相当于
+=	a+=2	a=a+2
-=	a-=b	a=a-b
*=	a*=b+c	a=a*(b+c)
/=	a/=b+c	a=a/(b+c)
%=	a%=5	a=a%5

(3) 赋值运算符的结合性是自右至左。若有两个赋值号,要先执行右边的。

例如,a=b=2 相当于是 a=(b=2)。

2. 赋值表达式

若一个表达式最后进行的是赋值运算,则该表达式是赋值表达式。

说明:C 语言中,一个表达式最后执行的是“什么”运算,就把该表达式称为“什么”表达式。例如,若最后执行的是算术运算,则它是算术表达式;若最后执行的是逻辑运算,则它是逻辑表达式……

C 语言中的所有表达式都是有值的。例如,算术表达式 5-2*1 的值是 3,关系表达式 5>3 的值是 1(即“真”)等。

赋值表达式也有值。C 语言规定:**一个赋值表达式的值,等于赋值后左边变量的值**。例如,若赋值表达式是 a=3*2,则表达式的值就是赋值后 a 的值,即 6。

赋值表达式的值可以参与运算,如 b=5+(a=3*2),执行后 b 的值是 11。

上面表达式的运算过程是:先执行 a=3*2,即把 6 赋给 a,再计算 5+(赋值表达式的值),即 5+6,得 11,最后再把 11 存入变量 b。

赋值表达式的值可以赋给另一个变量,如 a=(b=2)。因赋值号是右结合性,故可以省略括号,写成 a=b=2。

还可以再复杂一点:a=b=c=2,它同 a=(b=(c=2))等价。

考考你:上面 a=b=c=2 运算时,a、b、c 3 个变量哪个最先得到 2?哪个最后得到?若赋值表达式没有大小,还能不能像上面这样连等?

3.4.3　自增自减运算符

自增运算符是++,自减运算符是--。两个运算符都是单目运算符,都是右结合性,运算优先级与正负号相同,参见附录 E。

自增运算符用于给变量增加一个 1,自减运算符用于给变量减少一个 1。

自增和自减运算符都有两种用法,本书主要以++为例介绍两种用法的作用及区别。

1. 自增运算符

自增运算符++分为前++和后++,写在变量前面的是前++,写在变量后面的是后++,两者作用不同。例如下面的两段程序代码:

```
int i=1,m;
m=++i;     //++写在变量前,是前++
printf("%d,%d\n",m,i);
```

```
int i=1,m;
m=i++;     //++写在变量后,是后++
printf("%d,%d\n",m,i);
```

运行结果分别是:

```
2,2
```

```
1,2
```

之所以出现不同的结果,是因为++i 和 i++的求解过程不同。

首先需要强调的是,**++i 和 i++都是表达式,且两个表达式的值都是 i**。只不过,两个 i 的值并不相同。

对于左侧代码中的++i 来说,由于++写在前面,所以要先给 i 加 1(变成 2),再取 i 的值(即 2)作为表达式(++i)的值。

对于右侧代码中的 i++来说,由于 i 写在前面,++写在后面,所以先取 i 的值(即 1)作为表达式(i++)的值,然后再给 i 加 1,使 i 变成 2。

因此,左侧的"m=++i;"相当于是以下两行代码:

```
i=i+1;              //先给 i 加 1
m=i;                //表达式(++i)的值赋给 m
```

而右侧的"m=i++;"相当于如下两行代码:

```
m=i;                //表达式(i++)的值赋给 m
i=i+1               //给 i 加 1
```

一句话:**表达式++i 的值是加 1 之后的 i,而表达式 i++的值是加 1 之前的 i**。无论是 i++还是++i,求解过程中都给 i 加了 1。

若表达式 i++和++i 都不参与运算,则它们的作用相同,下面两条语句等价:

(1) i++;

(2) ++i;

它们都相当于是"i=i+1;"。

说明:设开始时 i=1,则上面两条语句分别相当于:

(1) 1; //这是表达式 i++的值与分号构成的语句,此后要给 i 加 1。

(2) 2; //这是表达式++i 的值与分号构成的语句,此前已给 i 加 1。

由于求解表达式过程中都给 i 加了 1,只不过一个在求值后,一个在求值前,因此,实际上前面两条语句分别等价于:

```
(1) 1;
    i= i+ 1;        //后加
(2) i= i+ 1;        //先加
    2;
```

其中，“1;”和“2;”两条语句没有任何实际意义，可以去掉，所以，最初的两种写法就都成了：

(1) i=i+1;

(2) i=i+1;

因此说，“i++;”和“++i;”的作用完全相同。

2. 自减运算符

自减运算符--的用法与++用法类似，不再赘述。

需要指出的是，自增和自减运算都相当于赋值运算，因此它们只能作用于变量，不能对表达式和常量进行自增或自减。下面写法都是错误的：

```
(a+b)++;          //相当于写成 a+b=a+b+1;     语法错误：a+b 不是左值
2--;              //相当于写成 2=2-1;          语法错误：2 不是左值
```

3.4.4　逗号运算符和逗号表达式

1. 逗号运算符

C 语言中，“,”也是一个运算符，称为逗号运算符。

逗号运算符是一个双目运算符，其结合性自左至右，其优先级在 C 语言的所有运算符中最低。

2. 逗号表达式

1) 逗号表达式的格式

表达式 1，表达式 2

即用逗号把两个式子连接起来。如“2,3”、“a=2，a * 3”都是逗号表达式。

2) 逗号表达式的求值方法

C 语言中的表达式都是有值的。逗号表达式的值等于逗号后面那个式子的值，即表达式 2 的值。例如，表达式“2,3”的值是 3，表达式“a=2,a * 3”的值是 a * 3 的值，即 6。但是，要想计算 a * 3，必须先执行 a=2。

所以，逗号表达式的求值方法是先求解表达式 1，再求解表达式 2，表达式 2 的值就是逗号表达式的值。

3) 多重逗号表达式

一个逗号表达式，可以作为另一个逗号表达式中的“表达式 1”，例如：

```
(a=1,b=2),a+b
```

上式也是一个逗号表达式,只不过其中内嵌了一个逗号表达式。由于逗号表达式的结合性是自左至右,上面的表达式可以去掉括号直接写成:

```
a=1,b=2,a+b
```

它的求值顺序是先执行 a=1,再执行 b=2,最后求解 a+b 的值作为整个表达式的值。

还可以继续不断嵌套,使逗号表达式成为如下模样:

```
式子 1,式子 2,式子 3,式子 4,…
```

其求值方法:自左至右按顺序求解每个式子,最后一个式子的值是整个表达式的值。


编程经验:其实,很多情况下使用逗号表达式的目的并不是求整个表达式的值,而是让计算机按顺序去求解每个表达式以完成一个个操作,例如,a=1,b=2,c=3。这种情况通常发生在需要用一条语句完成几条语句的功能时,但一般情况下尽量不要这样用。


考考你:m=1,2,3,4 是逗号表达式还是赋值表达式?整个表达式的值是多少?表达式求解之后,m 的值是多少?

3.4.5 类型转换运算符

C 语言中,有时候需要人为地把某种类型的数据转换为程序需要的类型,这时候就需要用类型转换运算符。例如,a、b 是任意整数,求(a-b)/(a+b)的值。程序代码如下:

```
int a,b;
float result;
scanf("%d%d",&a,&b);
result=(a-b)/(a+b);
printf("%f\n",result);
```

运行上面程序,从键盘输入 6 和 4,输出为 0,结果不正确。问题出在(a-b)/(a+b)这个表达式上,因为分子和分母都是整型数据,相除后结果必然还是整数(0)。

要想得到实数,至少应该把分子和分母中的一个变成实数。其方法是,使用类型转换运算符把表达式写成(float)(a-b) / (a+b) 或 (a-b) / (float) (a+b)。

类型转换运算符的格式如下:

```
(类型名) (表达式)
```

作用是把表达式的值强制转换为指定的类型。

需要指出的是,在类型转换前,系统要先求解表达式的值,然后将该值在运算器中进行处理、转换,被处理的是运算器中的结果,而不是表达式本身。表达式的类型和数值都不发生改变。例如:

```
float x=3.14;
int m;
```