第3章 数据类型和表达式

主要知识点：

- 基本字符和标识符
- 常量与变量
- 数据类型
- 运算符与表达式
- 数据类型转换
- 常用的输入输出函数
- 语句类型

C语言源程序是由一系列的字符组成的。编译器根据C语言语法规则来检查程序中的语言元素是否合法，编译器无法翻译有任何语法错误的程序，因此必须学会正确地编写代码。本章将为此打下基础。

3.1 基本字符和标识符

基本字符是C语言源程序最基本的元素，基本字符可用于构成标识符和表达式等，最终形成源程序。

3.1.1 基本字符

C语言的基本字符包括：字母、数字、空白符、标点和特殊字符。

(1) 字母：小写字母a～z共26个，大写字母A～Z共26个。

(2) 数字：0～9共10个。

(3) 空白符：空格符、水平制表符和换行符等统称为空白符。空白符只在字符常量和字符串常量中起作用，在其他地方出现时只起间隔作用，编译程序对它们忽略不计。因此，在源程序中使用空白符与否，对程序的编译不产生影响，但在程序中适当的地方使用空白符可以增加程序的清晰性和可读性，形成良好的编程风格。

(4) 标点和特殊字符：主要有，.；：？’”！|/\～_$%&^*－＋＜＞{}()[]＃等。

3.1.2 关键字

C语言的关键字就是明确保留的标记，它们具有严格的特定含义。虽然有些编译器还

会使用其他一些关键字，但 ANSI C 只规定了以下 32 个关键字，所有关键字都必须小写。这 32 个关键字分别如下。

1. 数据类型关键字 20 个

(1) 基本数据类型：void、char、int、float 和 double。

(2) 类型修饰关键字：short、long、signed 和 unsigned。

(3) 复杂类型关键字：struct、union、enum、typedef 和 sizeof。

(4) 存储级别关键字：auto、static、register、extern、const 和 volatile。

2. 流程控制关键字 12 个

(1) 跳转结构：return、continue、break 和 goto。

(2) 分支结构：if、else、switch、case 和 default。

(3) 循环结构：for、do 和 while。

3.1.3 标识符

C 语言的标识符有预定义标识符和用户定义标识符两类。

预定义标识符是 C 语言中预先定义并具有特殊含义的标识符。预定义标识符包括系统标准库函数名、编译预处理命令、头文件中定义的标识符等。例如：printf、scanf、main、include 等都属于预定义的标识符。预定义标识符允许用户对它们重新定义，但重新定义后会用新定义的含义替换它们原来的含义。

用户定义标识符常简称为标识符，指的是用户根据需要而自定义的变量名、函数名、数组名、数据类型名及宏名等。命名用户定义标识符时须遵守下面的命名规则：

(1) 标识符只能由英文字母、下划线(_)以及数字组成。不能包含空白字符(即换行符、空格和制表符等)。

(2) 标识符的第一个字符必须是英文字母或者下划线，而不能是数字。操作系统和 C 语言标准库里的标识符一般约定俗成以下划线开头，所以应避免用下划线作为自己定义的标识符的开头。

(3) 标识符中的英文字母区分大小写。例如：Score 和 score 是两个不同的标识符。

(4) 不能用关键字来给自定义的标识符命名。如果不是需要重新定义，就不要用预定义标识符来给自定义的标识符命名。

(5) 不同的 C 编译系统所能识别的标识符长度不同。ANSI C 可以识别标识符的前 31 个字符。

小提示：为标识符取直观且有意义的名称是提高程序可读性的重要方法。如果一个标识符由几个单词组成，可以用下划线来分隔这些单词，也可以使用大写单词首字母等方式。例如：tax_rate，TaxRate。如果标识符过长，可以根据一定的缩写规则来简化。例如：较短的单词可通过去掉“元音”形成缩写；较长的单词可取单词的前几个字母形成缩写。无论程序选用什么规则，对程序中所使用的缩写规则或约定，特别是特殊的缩写，都应该在源程序的开始处，进行必要的注释说明。

3.2 常量与变量

数据有两种基本表示形式，在程序中以常量和变量出现。

3.2.1 常 量

C 语言的常量是指在程序运行过程中，其值不会发生变化的量。C 语言支持多种类型的常量，其类型根据常量的书写形式识别。

1. 整型常量

整型常量由数值和常量后缀(L 或 l 表示长整型数、U 或 u 表示无符号整型数)构成，C 语言的整型常量有八进制、十六进制和十进制三种表现形式。

十进制整型常量：数值部分由数字 0～9 组成。例如：110、－139L、769U、12345ul 等。

八进制整型常量：数值部分以 0 开头，由数字 0～7 组成。例如：037、010L 等，八进制整型常量一般不使用负数。

十六进制整型常量：数值部分以 0x 或 0X 开头，由数字 0～9、a～z 或 A～Z 组成。例如：0x1a5、0XA 等，十六进制整型常量一般也不使用负数。

【实例 3-1】 整型常量的三种表现形式。

```
#include <stdio.h>
void main()
{
    printf("%d,%d,%d\n",20,020,0x20);
    printf("%d,%o,%x\n",20,020,0x20);
}
```

程序运行结果如图 3-1 所示。

图 3-1 整型常量的三种表现形式

程序分析：

printf("%d,%d,%d\n",20,020,0x20);的含义是用十进制整数的形式在屏幕上输出 20、020 和 0x20 三个常量，三个常量间用逗号分开。

printf("%d,%o,%x\n",20,020,0x20);的含义是分别用十进制、八进制和十六进制形式在屏幕上输出 20、020 和 0x20 三个常量，三个常量间用逗号分开。

2. 实型常量

实型常量也称为实数或浮点数。实型常量由数字、小数点和常量后缀构成。在 C 语言中规定实数只采用十进制，且都按双精度 double 型处理。但是，常量后缀 f 或 F 用于强制转换为单精度实数，而 l 或 L 则明确指定为双精度实数。(具体请参考 3.3.2 节)

实型常量有两种表示形式：十进制小数形式和指数形式。

(1) 十进制小数形式：由数字 0～9 和小数点组成，如 0.0、25.0、5.789L、－0.13、23.4F、6.7f 等均为合法的实数。特别地，31.0 可以表示为 31.，0.31 可以表示为.31。而 31 则不是合法的实数，因为它既没有小数点也没有实数后缀，所以它只是一个整型常量。

(2) 指数形式：数学表达式 $a\times10^n$ 的 C 语言表达式为 aEn 或 aen。其中，a 为尾数，它是一个十进制数；e 或 E 为指数标志；n 为指数，且只能为十进制整数，可以带符号。例如：2.1E5 即 2.1×10^5，3.7E-2 即 3.7×10^{-2}。指数形式的实型常量具体书写规则如下。

① 有且仅有一个字母 E 或 e，其左侧为尾数部分，右侧为指数部分。

② 指数与尾数部分都不可为空。例如：E-6 不是合法的指数形式实数，因为 E 的左边没有尾数部分。

③ 尾数部分可以是整数，也可以是小数。

④ 指数部分只能是整数。

⑤ 不能包含空格。例如：3.25□E□3 不是合法的指数形式，因为 E 的左右两边加了空格(本章约定在必要的位置用“□”来表示所在位置为空格字符)。

小提示：

(1) 数学意义上的常量在程序中不一定是常量，如 1/2、π、e(自然数)、23%等。在程序中 1/2 是一个值为 0 的整型表达式，π 是非法符号，e 是普通变量，23% 也是一个非法形式，表示 23%可以用 0.23、.23 或 2.3e－1 等。

(2) 实型常量默认为双精度型，有效位为 15 或 16 位，超出部分进行四舍五入。

【实例 3-2】 实型常量。

```
#include <stdio.h>
void main()
{
    printf("%f\n",1000);
    printf("%f\n",1000.0f);
    printf("%f\n",1000.0L);
    printf("%f\n",1000.0);
    printf("%e\n",1000.0);
}
```

程序运行结果如图 3-2 所示。

图 3-2 实型常量

程序分析：

(1) 语句 printf("%f\n",1000);的含义是用实型中的十进制小数形式在屏幕上输出整型常量 1000 的值。因为整型和实型在内存中的存储格式不同，所以不能正确地输出 1000。"%f"形式默认输出 6 位小数。

(2) 语句 printf("%f\n",1000.0f);与 printf("%f\n",1000.0L);的含义分别是用实型中的十进制小数形式在屏幕上输出单、双精度实型常量 1000.0 的值。

(3) 因为默认为双精度,所以 1000.0 与 1000.0L 的含义是相同的。

(4) 语句 printf("%e\n",1000.0);的含义是用实型中的指数形式在屏幕上输出实型常量 1000.0 的值。

3. 字符常量

1) 一般字符常量

一般字符常量就是单个可显示的字符,程序中字符常量写在一对单引号内(单引号'为字符常量的定界符)。

例如:'*'、'a'、'7'、'□'(两个单引号之间为空格字符)等。

思考:'7'和 7 是相同的常量吗?

2) 特殊字符常量

特殊字符常量包括不可显示字符和在 C 语言中具有特殊意义的字符。为了表示这些特殊字符,C 语言提供了转义字符来表示它们。

转义字符的定界符也是单引号,单引号内包括反斜线和被转义的字符。例如,'\n'将 n 转义为不可显示的换行字符。'\''将定界符'转义为普通的单引号字符。它们都被当作单个字符,所以也是字符常量。常用的转义字符如表 3-1 所示。

表 3-1 常用的转义字符表

转义字符	名称	转义字符	名称
\n	换行符	\'	单引号
\t	水平制表符	\''	双引号
\b	退格	\\	反斜线
\r	回车符,不换行	\ddd	八进制 ASCII 码值(0～377)
\f	换页符	\xdd	十六进制 ASCII 码值(0～F9)

3) 以 ASCII 码值表示字符常量

如表 3-1 中的后两项所示,用相应的八进制或十六进制 ASCII 码值表示字符常量,是字符常量的另外两种表示形式。例如:'\t'也可以表示为'\11'或'\x09'(Tab 的 ASCII 码八进制为 11,十六进制为 9)。

4) 字符常量的存储

虽然程序中字符常量写在一对定界符单引号内,但在内存中不存储定界符,只存储定界符内的字符对应的 ASCII 码值(参见附录 C),每个字符占一个字节。例如:字符'a'的 ASCII 码是 97,在内存中存储'a'实际上只需一个字节,这一个字节用来存储十进制的 97,即二进制形式的 01100001。

5) 字符常量的操作

在 C 语言中允许字符以 ASCII 码值参与算术运算。例如:

printf ("%d", 'A'+3); 的输出结果为 68。本语句的含义是先求出'A'的 ASCII 码加上 3 之后的值,即先求出 65+3 的值 68,再按十进制整型格式在屏幕上输出 68。

printf ("%c", 'A'+3); 的输出结果为 D。先求出'A'的 ASCII 码加上 3 之后的值,即

先求出 65＋3 的值 68，再将 68 按字符型格式在屏幕上输出，即输出 ASCII 码为 68 的字符'D'。

4. 字符串常量

字符串常量是用定界符双引号括起来的 0 个或多个字符序列(可以包括任意转义字符)。

例如，"hello"和"Hello! \n What can I do for you? "都是合法的字符串常量。

当双引号中只包含 0 个字符时，""称为空串。

字符常量占一个字节的内存空间。字符串常量占的内存字节数(也称为字符串的长度，或简称串长)等于字符串中的字符个数加 1。增加的一个字节用来存放字符'\0'(ASCII 码为 0)，它是编译系统自动为字符串添加的结束标志。

思考：

(1) 请分别从常量类型和存储方式两个方面比较'a'和"a"的不同之处。

(2) 请分析只包含 1 个空格字符的字符串"□"与空串""的区别。

【实例 3-3】 字符常量。

```
#include <stdio.h>
void main()
{
    printf("%c\n",'a');
    printf("%s\n","a");
    printf("%s\t%d\n","\x63\t\n\143",sizeof("\x63\t\n\143"));
}
```

程序运行结果如图 3-3 所示。

图 3-3 实例 3-3 运行结果

程序分析：

(1) printf("%c\n",'a');语句的含义是在屏幕上输出字符'a'。

(2) printf("%s\n","a");语句的含义是在屏幕上输出字符串"a"。

(3) 运算符 sizeof 可以求一个对象在内存中所需的字节数。sizeof("\x63\t\n\141")的含义是求字符串"\x63\t\n\141"在内存中所需的字节数，即串长。

(4) printf("%s\t%d\n","\x63\t\n\143",sizeof("\x63\t\n\143"));语句的含义是在屏幕上依次输出字符串"\x63\t\n\143"、字符 Tab、字符串"\x63\t\n\143"的串长值。

5. 符号常量

C 语言中，可以用一个标识符来代表一个常量，称为符号常量。符号常量在使用之前要先定义，定义格式如下：

```
#define 符号常量名 常量
```

其中，符号常量名为用户自定义标识符，习惯上用大写字母；常量可以是数字常量，也可以是字符常量。这实际上是一条宏定义命令，它将常量定义为一个符号常量，编译器先将符号常量名用所定义的常量替换后再对程序进行编译。

采用符号常量具有以下几个好处：

(1) 使用符号常量可以将复杂的常量定义为简明的符号常量，使得书写简单，而且不易出错。例如在第4章中的实例4-1中有：

```
#define  PI  3.1415926
```

符号常量PI被定义为3.1415926，在程序中书写PI，显然比书写3.1415926要简明。

(2) 采用符号常量会给修改程序带来方便。例如，在一个程序中使用了某个符号常量共10次，若需要对这一常量值进行修改，只需在宏定义命令中对定义的常量值进行一次修改，而无须在程序中出现这一常量的10处都进行修改。

(3) 增加可读性和移植性。符号常量通常具有明确的含义。例如，在前面的宏定义命令中，很明显PI表示圆周率。使用符号常量可将程序中影响环境系统的参数，如字长等，定义在一个可被包含的文件中，在不同的环境系统下，通过修改包含文件中符号常量的定义值来达到兼容的目的，便于提高程序的移植性。

小提示：C语言中，符号常量习惯用大写字母表示，而一般的变量名则用小写字母表示，以示区别。

3.2.2 变量

1. 变量

通俗地讲，变量就是在程序运行过程中其值可能发生变化的量。实质上，变量就是数据的存储空间，之所以变量的值可以改变，是因为该变量对应的存储空间中的数据是可以更改的。

根据变量的类型不同，系统为每一个变量在内存中分配相应的存储空间，每个变量都有一个名字，通过变量名可以对其存储空间中的内容进行改变或引用，而其存储空间的具体物理地址表示方式为“&变量名”，其中，&为取地址符。

变量名属于用户自定义标识符，变量名的命名规则须遵守标识符的命名规则。

2. 变量定义

变量需先定义后使用。所谓变量定义，也叫变量声明或变量说明，就是说明变量类型。其功能是为说明的每一个变量按类型开辟存储空间(编译系统在对程序进行编译时，根据变量定义的类型为其分配逻辑空间，运行时分配物理的内存空间)。变量的类型决定了其存储数据的范围、精度和参与运算的种类等。

变量定义的一般格式如下：

```
类型标识符  变量名表;
```

其中，类型标识符是指C语言允许使用的有效类型(具体请参考表3-2和表3-3)，变量名表由一个或多个变量名组成，若定义多个变量，则变量之间用逗号分隔。

【实例 3-4】 变量定义。

```
#include <stdio.h>
void main()
{
    int a,b;
    char c;
    float f;
    printf("%d,%d\n",&a,&b);
    printf("%d\n",&c);
    printf("%d\n",&f);
    printf("%d,%d,%d\n",sizeof(int),sizeof(c),sizeof(f));
}
```

程序在 Visual C++6.0 中运行结果如图 3-4 所示。

程序分析：

(1) int a,b;语句定义了两个名字分别为 a 和 b 的变量，这两个变量的类型均为 int 类型。

(2) char c;语句定义了一个名字为 c 的变量，这个变量的类型为 char 类型。

(3) float f;语句定义了一个名字为 f 的变量，这个变量的类型为 float 类型。

(4) 本程序没有给所定义的变量赋初值，编译器可能会给出一个警告，先暂时忽略它，后面章节将会介绍给变量赋初值的方法。

(5) printf("%d,%d\n",&a,&b);语句的含义是在屏幕上依次输出变量 a 和 b 分得的物理内存空间的首地址(简称地址)的十进制整数形式。

(6) printf("%d,%d,%d\n",sizeof(int),sizeof(c),sizeof(f));语句的含义是在屏幕上用十进制整数的形式依次输出 int 类型在内存中所需字节数(也就是变量 a 或 b 所需的字节数)、变量 c 和 f 在内存中所需的字节数。

(7) 在 Visual C++6.0 中，本次程序运行时 4 个变量在内存中所占的内存如图 3-5 所示。根据运行结果和语句含义可以得出，这 4 个变量是按定义的顺序在内存中依次存放，地址由大到小，即伸展方向是由高地址向低地址扩展的。

图 3-4 变量定义

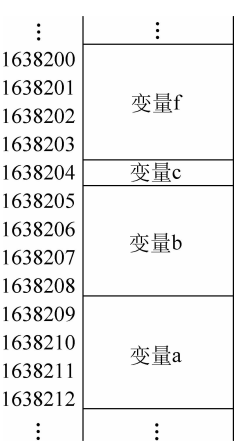

图 3-5 内存示意

3. 变量初值

在程序中经常需要对变量赋初值，一般可以用赋值运算符＝来为变量赋初值。

(1) 先定义变量，再用赋值语句来为变量赋初值。一般格式为：

变量 = 算术表达式;(注: 其中的 = 称为赋值号,而不是数学意义上的等号)

赋值语句的操作过程为先计算赋值号＝右侧的表达式的值，然后将计算结果存储在左侧的变量中。

例如：在实例 3-4 中定义变量 a,b,c,f 之后，可以加上以下赋值语句为变量 a,b,c,f 赋初值。

```
a = 8;                          /* 为变量 a 赋值为整数 8 */
b = a + 2;                      /* 为变量 b 赋值为整型表达式 a + 2 的值,即 10 */
c = 'a';                        /* 为变量 c 赋值为字符'a' */
f = 0.618;                      /* 为变量 f 赋值为实数 0.618 */
```

(2) 在定义变量的同时，还可以对其中全部或部分的变量赋初始值。例如：

```
int a,b = 1;                    /* 定义变量 a 和 b,并且只为变量 b 赋初值 1 */
int x = 1,y = 1;                /* 定义变量 x 和 y,并且 x 和 y 均赋初值 1 */
```

小提示：变量必须“先定义再引用”。

【实例 3-5】 变量赋初值和引用。

```
#include <stdio.h>
void main()
{
    int a = 2,b;
    printf("%d\n",a);
    printf("%d\n",b);
    a = a * 2;
    b = a + 3;
    printf("%d\n",a);
    printf("%d\n",b);
}
```

程序运行结果如图 3-6 所示。

图 3-6　变量赋初值和引用

程序分析：

int a＝2,b;语句的作用是定义两个 int 类型的变量 a 和 b,并且为 a 赋初值 2,但 b 没有赋初值。所以其后的语句 printf("%d\n",b);输出的 b 的值是－858993460,它是一个随机数。而执行 b＝a＋3;后再执行 printf("%d\n",b);输出的 b 是正确值 7。

3.3 数 据 类 型

3.3.1 概述

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数据类型、构造数据类型和自定义类型。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，它包括整型、实型、字符型和void类型。构造数据类型是根据已定义的一个或多个数据类型用构造的方法来定义的。也就是说，一个构造类型的值可以分解成若干个成员或元素，每个成员都是一个基本数据类型或又是一个构造类型。构造数据类型包括数组、结构体、共用体、枚举和指针等。

数据类型决定了数据的存储空间的大小和存储方式，进而决定了该类数据的取值范围和精度；另外，数据类型还决定了数据运算(操作)的规则。

3.3.2 基本数据类型

基本数据类型包括整型、实型、字符型和void类型。

1. 整型

为了控制取值范围和存储空间，C语言有三种类型的整型：short int、int和long int，且都具有有符号和无符号两种形式。int型为基本整型，short int表示相对较小的整型，long int表示相对较大的整型。无符号整型是用所有位来表示数字的，且总为正数。将整型定义为long或unsigned可以增大所表示的取值范围。修饰符signed可以省略，整型默认为有符号形式，且以补码表示。

C语言没有具体规定各类整型数据所占内存的字节数，只要求short型所占的字节数不能大于int型，而int型所占的字节数不能大于long型。具体某个整型数据所占的字节数及取值范围取决于特定的计算机和编译器。通常，基本整型(即int型)占用一个字的存储空间。例如：如果是16位的字长，基本整型用16位来表示，用其中的最高位表示符号位，另外的15位表示数值。

整型数据在取值范围内都是精确存储，常见的整型所占字节数和取值范围如表3-2所示。下面以unsigned short和short为例说明整型数据在内存中的存储方式。

表3-2 常见的整型及其所占字节数和取值范围表

名称		数据类型	所占字节数	取值范围
有符号整型	基本	[signed] int	2(16位编译器) 4(32位编译器)	−32 768～32 767(-2^{15}～$2^{15}-1$) −2 147 483 648～2 147 483 647(-2^{31}～$2^{31}-1$)
	短	short [int]	2	−32 768～32 767(-2^{15}～$2^{15}-1$)
	长	[signed] long [int]	4	−2 147 483 648～2 147 483 647(-2^{31}～$2^{31}-1$)
无符号整型	基本	unsigned [int]	2(16位编译器) 4(32位编译器)	0～65 535(0～$2^{16}-1$) 0～4 294 967 295(0～$2^{32}-1$)
	短	unsigned short [int]	2	0～65 535(0～$2^{16}-1$)
	长	unsigned long [int]	4	0～4 294 967 295(0～$2^{32}-1$)

(1) unsigned short 类型数据占 2 字节,16 位全部存储数值,例如:

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

表示 $2^{15}+2^1+2^0=32\,768+2+1=32\,771$。

(2) short 类型数据以补码形式存放,占 2 字节,最高位存储符号,0 表示正数,1 表示负数。例如:

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

表示正数 $2^1+2^0=3$。

而改变符号位后:

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

表示的是−32 765。

小提示:补码的求法如下:

正数的补码和原码(最高位为符号位,0 表示正数,1 表示负数)相同。

负数的补码:符号位保持为 1,原码的其他位取反后得反码,反码加 1 得补码。

例如,求−32 765 的补码:

−32 765 的原码($32\,765=2^{15}-3$):

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

−32 765 的反码:

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

−32 765 的补码:

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

2. 实型

实型也称浮点类型。一般情况下,实型采用单精度类型(float)和双精度类型(double)两种形式来存储。

无论是 float 型还是 double 型在内存中存储时都分为以下三个部分。

(1) 符号位:0 代表正,1 代表负。

(2) 指数部分:用于存储指数形式中的指数数据。

(3) 尾数部分:用于存储指数形式中的尾数部分。

尾数部分和指数部分所占位数的多少由各 C 语言编译系统自定。尾数部分占的位数越多,有效数字的位数也就越多,精度也就越高;指数部分占的位数越多,则能表示的取值范围就越大。表 3-3 给出了常见的实型所占字节数及取值范围。

表 3-3 常见的实型所占字节数及取值范围表

名 称	数 据 类 型	所占字节数	取 值 范 围
单精度实型	float	4	约为−3.4E38～3.4E38
双精度实型	double	8	约为−1.7E308～1.7E308

小提示：

(1) 当变量的值大于或小于其数据类型所能存储的值时，就会发生数据溢出问题。

(2) 当发生数据溢出时，一些高级程序设计语言输出的结果以 * 代表，提示程序设计人员改写程序中的数据类型。而 C 语言不检查数据是否溢出。例如：在 32 位编译器下执行语句“printf("%d",2147483648);”，输出为－2 147 483 648，没有任何提示。显然输出的结果与真实结果完全背离。因此在编程时一定要认真分析实际问题中数据可能的范围，并依此来选择所需的数据类型。

3. 字符型

ANSI C 提供了三种字符类型：char、unsigned char 和 signed char，普通的 char 类型相当于 unsigned char 还是 signed char 则取决于编译器。但三种字符类型都是按照一个字节存储的，在内存中存储其 ASCII 值。

4. void 类型

void 类型没有数值，它通常用于指定函数的返回类型。有一类函数，调用后不需要向调用者返回函数值，这种函数可以定义为“空类型”，其类型说明符为 void。

【实例 3-6】 void 类型。

```
#include <stdio.h>
void printastar()
{
    printf("*\n");
}
void main()
{
    printastar();
}
```

程序运行结果如图 3-7 所示。

图 3-7　void 类型

3.3.3　指针类型

指针类型是一种特殊的，同时又具有重要作用的数据类型。指针类型的值用来表示某个内存地址。由于计算机内存是保存程序指令和数据的地方，所以可用指针来访问和操作存储在内存中的数据，C 语言的指针是一个功能强大的工具。

(1) 在处理数据和数据表时，指针更高效。

(2) 通过作为函数参数，指针可用来从函数中返回多个值。

(3) 指针允许引用函数，因而便于把函数作为参数传递给其他函数。

(4) 使用指向字符串的指针数组，可节省内存的数据存储空间。

(5) 指针使得 C 语言支持动态内存管理。

(6) 指针为操作动态数据结构(如结构、链表、队列、栈和树等)提供了有效的工具。

(7) 指针可减少程序的长度和复杂度。

(8) 指针可提供程序的运行速度,从而减少程序的运行时间。

关于指针的应用会在后续章节陆续介绍,本节只介绍指针变量最简单的用法。

1. 指针变量的概念

地址就是内存单元的编号,每个内存单元都有这样一个地址(或编号),而且该地址(或编号)在C程序的运行过程中不会改变。因为地址总是对应某个内存单元,所以通常也将地址形象化地称为指针(Pointer),即指针实际上是某个内存单元的地址。

由于变量常常会占用多个内存单元,对应着多个单元的地址,通常把这些地址中的起始地址作为该变量的地址。在使用该变量时,除了根据变量名来访问该变量外,还可根据该变量的地址及变量的类型相结合来访问该变量的值。如图3-5中变量a的地址值&a为1 638 212,a的类型为int型(占4个字节),就可通过以1 638 212为起始单元的相邻4个单元来访问变量a的内容(当然,实例3-4中还没有对变量赋初值)。通常将通过变量名来访问变量内容的方式称为直接访问方式,而通过变量的地址来访问变量内容的方式称为间接访问方式。

要使用间接访问方式首先需要找到变量的地址,用来保存其他变量的地址的变量称为指针变量,也常简称为指针。

2. 指针变量的定义

和其他类型的变量类似,要保存一个变量的地址,首先需要定义一个表示地址类型的变量,而根据前面的描述,变量的地址需要与变量的类型相结合。

定义指针变量的一般形式为:

```
数据类型 *变量名;
```

例如:语句int *p;的作用是把变量p定义为一个指针类型的变量,它指向一个int数据类型。具体包括以下三个含义。

(1) 符号*说明变量p是一个指针变量;

(2) 变量p需要一个内存地址作为它的值;

(3) 变量p是指向int类型的一个指针变量,也就是说,它可以接收一个int类型变量的地址。

语句int *p;使编译器为指针变量p分配存储空间。由于此时没有赋给p任何值,因而指针变量p中包含的是未知的值,此时它指向的也是未知的地址。

小提示:

(1) 在定义指针变量时,数据类型、*、变量名三个元素均不能省略,如将语句int *p;写为int p;时,变量p就成了一个整型变量,而不是指针变量了。

(2) 在定义指针变量时,*的位置并不重要,如int* p;和int *p;是同一条语句,但至少要在*前或*后添加一个空格,即不能写成int*p;。

(3) int* pA; int* pB;也可以合并为一条语句:int *pA, *pB;。注意指针变量pB前的*不能省略。语句int *pA, pB;相当于int* pA; int pB;即pA为一个指针变量,而pB只是一个整型变量。

3. 指针变量的初始化

把变量的地址赋给指针变量的过程称为指针变量的初始化。如前所述，所有未初始化的指针变量的值为未知的，这些值同样会被解释为内存地址。它们可能是有效地址，也可能是错误的值。由于编译器不会检测这类错误，因而含有未初始化指针的程序会产生错误的结果。因此，在使用指针之前，对它们进行初始化非常重要。

指针变量一旦定义，就可以使用赋值运算符来初始化该变量。

【实例 3-7】 指针变量的初始化。

```
#include <stdio.h>
void main()
{
    int a;
    int *p;
    printf("p的值为: %d\n", p);
    p = &a;
    printf("a的地址为: %d\n", &a);
    printf("p的值为: %d\n", p);
    printf("p的地址为: %d\n", &p);
}
```

程序运行结果如图 3-8 所示。

图 3-8 指针变量的初始化

程序分析：

(1) 语句 int *p;定义了一个指针变量 p，它指向 int 类型，也就是说它可以保存一个 int 类型变量的地址。

(2) 语句 printf("p 的值为：%d\n", p);输出未初始化时 p 的值。因 p 未初始化就使用，在有些编译器中可能会在编译时产生一个警告，但程序仍可以运行。此次运行结果中输出的值为：－858 993 460，显然，这不是正确的变量地址值。

(3) 执行语句 p = &a;时先求出 int 类型的变量 a 的地址值，把这个地址值赋值给指针变量 p，也就是对指针变量 p 进行初始化。初始化之后，p 才是指向变量 a 的指针。

(4) 语句 printf("a 的地址为：%d\n", &a);和 printf("p 的值为：%d\n", p);的输出都是 1 638 212，证明初始化之后，指针变量 p 的值确实是变量 a 的地址。

也可以把初始化和定义组合在一起。例如以下三条语句：

```
int a;
int *p;
p = &a;
```

与以下两条语句的作用是相同的：

```
int a;
int *p = &a;
```

还可以直接用一条语句表示：int a，*p = &a;

小提示 1：

在用变量的地址对指针进行初始化时，必须已对变量定义过。

例如：int *p = &a, a;和 int *p = &a; int a;都不正确。因为变量 a 被“先使用后定义”，这在 C 语言中是非法的。

小提示 2：

必须确保指针变量指向相应的数据类型。

例如：

```
float a;
int *p;
p = &a;
```

是错误的。

小提示 3：可以在同一条语句中同时进行数据变量的定义、指针变量的定义以及指针变量的赋值。

例如：float a，b，*pa = &a，*pb=&b;

小提示 4：可以在定义指针变量时赋初始值 0，但不可以用其他常量来为指针变量赋值。例如：int *p=0;是合法的，它表示指针变量 p 暂时不指向任何变量，而 int *p=2000;是非法的。

思考：可以在不同语句中使用同一指针先后指向不同的数据变量吗？可以使用不同的指针指向同一数据变量吗？

4. 通过指针访问变量

把变量地址赋值给指针变量之后，就可以使用一元运算符 * 来用指针间接访问变量的值了。* 的作用是获取某个指针所指向的变量，在使用时其后需要紧跟这个指针变量的名称。

【实例 3-8】 通过指针访问变量。

```
#include <stdio.h>
void main()
{
int a, b, *p = &a, *q = &b;
a = 1;
b = 2;
printf("a 的值为: %d\n", a);
printf("指针变量 p 所指的变量值为: %d\n", *p);  //*p 表示指针变量 p 所指的变量
printf("指针变量 q 所指的变量值为: %d\n", *q);  //*q 表示指针变量 q 所指的变量
p = &b;                    //把另一个变量 b 的地址赋给指针变量 p,此时 p 和 q 同时指向变量 b
*q = *p* 3;                //给指针变量 p 所指的变量的值乘上 3 然后再赋值给 q 所指的变量
printf("b 的值为: %d\n", b);
printf("指针变量 p 所指的变量值为: %d\n", *p);
printf("指针变量 q 所指的变量值为: %d\n", *q);
}
```

程序运行结果如图 3-9 所示。

图 3-9　通过指针访问变量

需要注意的是，取内容运算符“*”和表示指针变量“*”的作用是不同的，在程序中可根据“*”前是否有数据类型来判断它们。例如：float *p = &ave; *p = 3.5;。

(1) float *p = &ave;语句中的“*”前有 float 表示单精度实型，此时的“*”表示 p 是一个指针变量。

(2) *p = 3.5;语句中的“*”前没有表示数据类型的关键字，此时的“*”表示取指针 p 所指向的内容，即 p 所指向的变量。

(3) 不能出现如 float *p; *p = 3.5; 类型的程序段，因为程序段中并未写出指针 p 指向哪个变量，而 *p 的作用却正是要取这个变量，所以此时执行 *p 是非法的。

(4) 已知程序段 double sum = 3.1415; double *p = ∑则“&*p”的值为 p，“*&sum”的值为 sum，因为“&”和“*”的运算优先级相同，且均为右结合性，故在计算“&*p”时先执行“*p”得到 sum，再求“&sum”，得到 p; 同理，“*&sum”的值为 sum。

3.4　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。

3.4.1　算术运算符与算术表达式

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需要两个运算对象的运算符，C 语言中有以下 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 类型的运算对象应用求余运算。

(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的基本算术运算符，其结合性为自右向左。

自增自减运算符有以下 4 个基本形式。

(1) ＋＋i；变量自增 1 后再参与运算。

(2) i＋＋；变量参与运算后，再自增 1。

(3) －－i；变量自减 1 后再参与运算。

(4) i－－；变量参与运算后，再自减 1。

【实例 3-9】 自增自减运算符。

```
#include <stdio.h>
void 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);
}
```

程序运行结果如图 3-10 所示。

小提示：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。

图 3-10　自增自减运算符

3. 算术表达式

用基本算术运算符、自加自减运算符和圆括号将运算对象(常量、变量、函数等)连接起来的式子称为算术表达式。单个的常量、变量、函数可以看作是表达式的特例。

每个表达式有一个值及其类型,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)。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条。

(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。所以,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,运算结果取高一级的数据类型。具体参见 3.5 节。

思考:表达式 1/2*2.22 的值是什么? 表达式 1.0/2*2.22 或 2.22/2 的值是什么?

小提示 2:

(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。要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) 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。

(4) 程序中将 e 视为变量,而不是数学领域认为的"自然数";在程序中可以用函数调用 exp(1)表示自然数 e。

(5) 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 A/(B*C)更容易理解。

(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。

3.4.2　赋值运算符与赋值表达式

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

赋值表达式的功能是先计算赋值号右边的表达式的值再赋给左边的变量。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 中，凡是表达式可以出现的地方均可出现赋值表达式。例如：x=(a=5)+(b=8) 是合法的。它的意义是把 5 赋给 a，8 赋给 b，再把 a，b 相加，和赋给 x，故 x 应等于 13。当然，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上分号就构成为语句。例如：x=8；a=b=c=5；都是赋值语句。

小提示：

(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：a+10=a*2 不是合法的表达式。

(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，结合性是右结合。

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

它的求值过程为：

(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)。

小提示：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。

3.4.3 关系运算符与关系表达式

1. 关系运算符

在程序中经常需要比较两个量的大小关系，以决定程序下一步的工作。比较两个量的运算符称为关系运算符。在 C 语言中有以下 6 个关系运算符。

<：小于

<=：小于或等于

>：大于

>=：大于或等于

==：等于

!＝：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 6 个关系运算符中，＜、＜＝、＞和＞＝的优先级相同，它们的优先级高于＝＝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 1　关系运算符　表达式 2

例如：a ＋ b＞c－d 和'a' ＋ 1 ＜ c 都是合法的关系表达式，由于表达式 1 和表达式 2 也可以又是关系表达式，因此也允许出现嵌套的情况，例如：a＞(b＞c)和 a!＝(c＝＝d)等。关系表达式的值是 0 和 1 中的一个。1 表示“真”，0 表示“假”。

例如，若有语句 int a ＝ 1，b ＝ 2，c ＝ 3；则：

表达式 a ＋ b ＜＝ c ＋ 8 相当于(a ＋ b) ＜＝ (c ＋ 8)，其值为 1。

表达式 a ＝ 8 ＞＝ a ＋ b ＋ (c ＝ 6)相当于 a ＝ (8 ＞＝ a ＋ b ＋ (c ＝ 6))，先求出表达式 8 ＞＝ a ＋ b ＋ (c ＝ 6)的值为 1，再把结果 1 赋值给变量 a。

表达式 10 ＜ a ＜ 20 相当于(10 ＜ a) ＜ 20，10 ＜ a 的值为 0，0＜ 20 的值为 1，所以表达式 10 ＜ a ＜ 20 的值为 1。显然，C 语言中 10 ＜ a ＜ 20 与数学表达式 10 ＜ a ＜ 20 的含义截然不同。

小提示：一般而言，在 C 程序中可以对实型数据进行大小的比较，但通常不对实型数据进行是否相等的判断。因为实型数据在内存中不是精确存放，有一定的误差，如果一定要判断两个实型数据是否相等，可以用它们的差的绝对值与一个很小的数相比，如果小于此数，就认为它们是相等的。例如：可以用类似 fabs(x－y) ＜ 1e－5 的表达式来判断实型数据 x 与 y 是否相等。

3.4.4　逻辑运算符与逻辑表达式

1. 逻辑运算符

C 语言中提供了三种逻辑运算符 &&(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。逻辑运算的结果只有 1 和 0。逻辑运算符的运算规则如表 3-4 所示。

表 3-4　逻辑运算符的运算规则表

a	b	a&&b	a||b	！a
0	0	0	0	1
0	非 0	0	1	1
非 0	0	0	1	0
非 0	非 0	1	1	0

(1) 与运算 && 参与运算的两个运算对象都为非 0 时，结果才为 1，否则为 0。例如，5＞0 && 4＞2，由于 5＞0 为 1，4＞2 也为 1，相与的结果也为 1。

(2) 或运算||参与运算的两个运算对象只要有一个为非 0，结果就为 1。只有当两个运算对象都为 0 时，结果才为 0。例如：5＞0||5＞8，由于 5＞0 为 1，相或的结果为 1。

(3) 参与非运算！的运算对象为非 0 时，结果为 0；运算对象为 0 时，结果为 1。例如：!(5>0)的结果为 0。

小提示：在判断一个表达式的逻辑值时，非 0 的数值都作为 1。例如：由于 5 和 3 均为非 0，所以 5&&3 即为 1&&1，结果为 1。

2. 逻辑表达式

逻辑表达式的一般形式为：

[表达式 1]　逻辑运算符　表达式 2

说明：

(1) 当逻辑运算符为！时，须省略表达式 1。！具有右结合性，要写在运算对象的左边，并且仅对紧跟其后的运算对象进行逻辑非运算。例如：！a+b>c 相当于(！a)+b>c，而不是!(a+b)>c。

(2) 其中的表达式 1 和表达式 2 可以又是逻辑表达式，从而组成嵌套的情形。例如：(a&&b)&&c。逻辑表达式的值是式中所有逻辑运算的最后值。

(3) && 和||具有左结合性，要严格按照从左到右的顺序依次运算。但在逻辑表达式的求解过程中，如果计算到某个逻辑运算符时就已经可以确定整个表达式的最终结果，系统不再对其后的表达式求值。例如：假定 a=1，b=1，c=1，d=1，m=10，n=10，计算表达式(m=a>b)&&(n=c>d))的值时需先求出(m=a>b)的值，即求出 a>b 为 0，把 0 赋值给 m，(m=a>b)作为逻辑表达式其值为 0。此时已可确定整个表达式(m=a>b)&&(n=c>d))的值为 0，所以不再执行给 n 赋值的操作。

【实例 3-10】 逻辑表达式。

```
#include <stdio.h>
void main()
{
    char c = 'k';
    int i = 1, j = 2, k = 3;
    double x = 3e+5, y = 0.85;
    printf("%d, %d\n", !x*!y, !!!x); printf("%d, %d\n", x||i&&j-3, i<j&&x<y);
    printf("%d, %d\n", i==5&&c&&(j=8), x+y||i+j+k);
}
```

程序运行结果如图 3-11 所示。

图 3-11　逻辑表达式

程序分析：本例中！x 和！y 都为 0，！x*！y 也为 0，故其输出值为 0。由于 x 为非 0，故!!! x 的逻辑值为 0。对于 x||i&&j-3，先计算 j-3 的值为非 0，再求 i&&j-3 的逻辑值为 1，故 x||i&&j-3 的逻辑值为 1。对于 i<j&&x<y，由于 i<j 的值为 1，而 x<y 为

0,故表达式的值为 1 和 0 相与,最后为 0。对于 i== 5&&c&&(j=8),由于 i==5 为假,即值为 0,该表达式由两个与运算组成,所以整个表达式的值为 0,而不需继续计算。对于 x+y||i+j+k,由于 x+y 的值为非 0,故整个表达式的值为 1。

3.4.5 条件运算符与条件表达式

如果在条件语句中,只执行单个的赋值语句时,常可使用条件表达式来实现。不但使程序简洁,也可提高运行效率。

1. 条件运算符

条件运算符由"?"和":"构成,它是唯一的三目运算符,即需三个运算对象。

2. 条件表达式

由条件运算符和运算对象连接而成的表达式称为条件表达式。

条件表达式的一般形式为:

```
表达式 1 ?表达式 2 : 表达式 3
```

条件表达式的求值规则为:如果表达式 1 为真(非 0),则以表达式 2 的值作为整个条件表达式的值,否则以表达式 3 的值作为整个条件表达式的值。条件表达式通常用于赋值语句之中。

例如,语句 max=(a>b)? a:b;的含义是:如果 a>b 成立,则把 a 的值赋值给 max,否则把 b 的值赋值给 max。

说明:

(1) 条件运算符的运算优先级低于关系运算符和算术运算符,但高于赋值运算符和逗号运算符。因此 max=(a>b)? a:b 可以去掉括号而写为 max=a>b? a:b。

(2) 条件运算符? 和: 是一对运算符,不能分开单独使用。

(3) 条件运算符具有右结合性。当一个表达式中出现多个条件运算符时,应该将位于最右边的问号与距它最近的冒号匹配,并按这一原则正确区分各条件运算符的运算对象。例如:a>b? a:c>d? c:d 应理解为 a>b? a:(c>d? c:d)这也就是条件表达式嵌套的情形,这里的表达式 3 又是一个条件表达式。

3.4.6 逗号运算符与逗号表达式

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起来组成一个表达式,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,以此类推,最后求表达式 n 的值。把表达式 n 的值作为整个逗号表达式的值。

【实例 3-11】 逗号运算符举例。

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

```
void main()
{
    int a = 1,b = 2;
    a = (a = a + b,a = a * b,++a);
    printf("a = %d\n",a);
}
```

程序运行结果如图 3-12 所示。

图 3-12　逗号运算符

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。

3.4.7　其他运算符与表达式

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
```

取地址运算符 & 的运算对象只能是变量，运算结果是变量在内存中第一个字节的地址，即变量的首地址，简称变量的地址。使用时要注意 & 与变量名之间不要有空格。

【实例 3-12】 用指针来求两个整数中的较大数。

```
#include <stdio.h>
void main()
{
    int a, b, * pmax = 0;
    a = 3;
    b = 4;
    pmax = a > b?&a:&b;
    printf(" %d\n ", * pmax);
}
```

程序运行结果如图 3-13 所示。

程序分析：

(1) pmax＝a＞b? &a：&b；语句的含义是如果 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。

图 3-13　求两个整数中的较大数

(2) printf("%d\n", * pmax);语句中的 * 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. 长度运算符 sizeof

长度运算符 sizeof 也是单目运算符,它用于确定一个对象所需的存储空间的大小,即存储该对象所需的内存字节数。用它构成的表达式的一般形式为:

```
sizeof(数据类型名)或 sizeof(变量名)
```

【实例 3-13】 以下程序可以显示在一台特定的机器和编译器上各种基本类型所需的存储空间。

```
#include <stdio.h>
void main()
{
printf("sizeof(char) = %d\n",sizeof(char));
printf("sizeof(short) = %d\nsizeof(int) = %d\nsizeof(long) = %d\n",sizeof(short),sizeof
(int),sizeof(long));
printf("sizeof(signed) = %d\nsizeof(unsigned) = %d\n",sizeof(signed),sizeof(unsigned));
printf("sizeof(float) = %d\nsizeof(double) = %d \n",sizeof(float),sizeof(double));
printf("sizeof(void) = %d\n",sizeof(void));
}
```

在 Visual C++6.0 中程序运行结果图 3-14 所示。

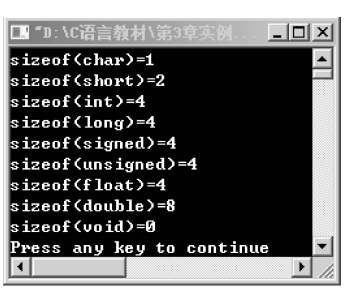

图 3-14　实例 3-13 程序运行结果

由于 C 语言在基本类型的存储方面非常灵活,可能因不同的机器和编译器而不同。但是可以保证:

```
sizeof(char) = 1
sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long)
sizeof(signed) = sizeof(unsigned) = sizeof(int)
```

```
sizeof(float) <= sizeof(double)
```

小提示：虽然使用 sizeof()这种函数式的写法，但 sizeof 不是函数，而是操作符。

3.5 数据类型转换

在 C 语言的表达式中，允许对不同类型的数值型数据进行运算。当对不同类型的数据进行运算时，应当首先将其转换成相同的数据类型，然后进行运算。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。

3.5.1 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时，整型数据的数值大小不变，只是把形式改为实型形式，即小数点后加若干个 0，然后赋值。

2. 单、双精度实型

float 型数据赋值给 double 型变量时只是在尾部加 0 延长为 double 型数据，然后赋值。double 型数据赋值给 float 型变量时，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。

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，就作为负数处理。

4. int 型与 1ong 型

假定 int 型占两个字节，long 型数据赋给 int 型变量时，将低 16 位值赋值给 int 型变量，而将高 16 位截断舍弃。将 int 型数据赋值给 long 型变量时，其数值大小不变，只是把形式改为 long 型再赋值。

5. 无符号整数

将一个 unsigned 型数据赋给一个长度相同的整型变量时，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。

小提示：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。

3.5.2 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时，编译器就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 3-15 所示。

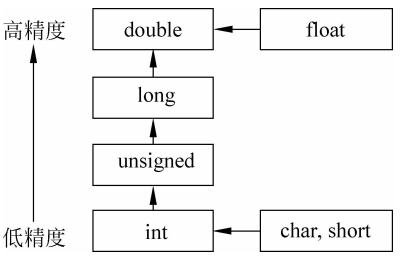

图 3-15　常见的类型转换规则

说明：

(1) 图中横向箭头表示必须进行的转换，如两个 float 型数参加运算，虽然它们类型相同，但仍要先转成 double 型再进行运算，结果也为 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，然后两者再进行运算，结果为 long 型。

(3) 如果一个运算符两边的运算对象类型不同，较低类型需转换为较高类型，然后再参加运算，这样不会降低运算的精度。

(4) 当较低类型的数据转换为较高类型时，一般只是形式上有所改变，而不影响数据的大小。

【实例 3-14】 自动类型转换与赋值类型转换。

```
#include <stdio.h>
void main()
{
  int a = 1;
  float b = 2.9;
  double c = 3.8;
  a = a + b + c;
  printf("a = %d\n",a);
}
```

程序运行结果如图 3-16 所示。

图 3-16　自动类型转换与赋值类型转换

程序分析：

(1) 在 Visual C++6.0 中编译时语句 float b＝2.9；会产生一个警告：“'initializing'：truncation from 'const double ' to 'float '”。这是因为常量 2.9 是一个实型常量，对于没有加后缀 f 或 F 的实型常量，C 语言是按 double 的形式来处理的。把 double 型的常量赋值给 float 型的变量时编译器需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 中编译语句 a=a+b+c;时也会产生一个警告:“ '=' : 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,而把一个 double 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:

① 首先取 int 型的变量 a 的值 1,把 int 型的 1 转换为 double 类型的 1.0。

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

③ 然后求出转换后的两个 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 再和 double 型的变量 c 的值 3.8 相加,得出最后 double 型的结果 7.7。

⑤ 最后丢弃 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。

小提示:参与运算的某个变量如果需要自动类型转换,这个转换只是在该运算的过程中发生,并不会改变该变量自身的类型和值。所以执行语句 a=a+b+c;时,变量 a 和 b 的类型仍然分别是 int 和 float 型。

3.5.3 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,其格式是:

```
(类型标识符) 运算对象
```

例如:表达式(int) 0.681 的值为 1。

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)和(int)(0.681)+0.432 的值分别为 int 型的 1 和 double 型的 1.432。

3.6 常用的输入输出函数

C 语言输入输出函数有很多,本节介绍 4 个常用的函数:scanf()、printf()、getchar()和 putchar()。特别强调,如果程序需要使用它们,需要包含 stdio.h 头文件。

3.6.1 标准输入输出函数

C 标准库提供了两个控制台格式化输入输出函数 printf()和 scanf(),这两个函数可以在标准输出输入设备上以各种不同的格式写/读数据。printf()函数用来向标准输出设备(屏幕)写数据;scanf()函数用来从标准输入设备(键盘)上读数据。下面详细介绍这两个函数的用法。

1. 标准输出函数 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("格式化字符串",输出表);
```

输出表是需要输出的一系列表达式,其个数必须与格式化字符串所说明的输出参数个

数一样多，各表达式之间用“,”分开，且顺序一一对应，否则将会出现意想不到的错误。

格式化字符串可以包括两部分内容：一部分是普通字符，包括可打印字符和转义字符。可打印字符按原样输出（如果有汉字系统支持也可以输出汉字），在输出结果中起提示作用；转义字符则控制产生特殊的输出效果。另一部分是格式字符，以“%”开始，后跟一个或几个格式字符，用来确定输出内容格式。C 中格式字符串的一般形式为：%[标志][输出最小宽度][.精度][长度]类型，其中方括号[]中的项为可选项。各项的意义介绍如下。

1）类型

类型字符用来表示输出数据的类型，其格式符和意义如表 3-5 所示，其中的输出结果为 Visual C++6.0 中的输出结果。

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
c	输出单个字符	char ch='a'; printf("%c",ch);	a
d	以十进制形式输出带符号整数（正数不输出符号）	int a=10; printf("%d",a);	10
f	以小数形式输出单、双精度实数	float a=123456.78f; printf("%f",a);	123456.781250
e	以指数形式输出单、双精度实数	double a=123456.78; printf("%e",a);	1.234568e+005
E	以指数形式输出单、双精度实数	double a=123456.78; printf("%e",a);	1.234568E+005
o	以八进制形式输出无符号整数	int a=10;　printf("%o",a);	12
x	以十六进制形式输出无符号整数	int a=10;　printf("%x",a);	a
X	以十六进制形式输出无符号整数	int a=10;　printf("%x",a);	A
u	以十进制形式输出无符号整数	printf("%u",-123);	4294967173
s	输出字符串	printf("%s","hello");	hello

2）标志

标志字符有－、＋、＃、空格和 0，共 5 种，其意义如表 3-6 所示。

表 3-6　标志字符及意义

标志字符	标志字符的意义
－	结果左对齐，右边填空格
＋	输出符号（正号或负号）
空格	输出值为正时冠以空格，为负时冠以负号
＃	对 c、s、d、u 类无影响；对 o 类，在输出时加前缀 0；对 x 或 X 类，在输出时加前缀 0x 或者 0X；对于所有的实数形式，＃保证了即使不跟任何数字，也打印一个小数点字符
0	对于所有的数字格式，用前导 0 填充字段宽度，若出现－标志或者指定了精度（对于整数），忽略

3）输出最小宽度

输出最小宽度是用十进制整数来表示输出的最少位数。若实际位数多于定义的输出最小宽度，则按实际位数输出，若实际位数少于定义的宽度则补以空格或 0。特别注意：实型

数据的小数点占一位。

4）精度

精度格式符以“.”开头，后跟十进制整数。如果输出实数，精度表示小数的位数；如果输出的是字符，精度表示输出字符的个数；若实际位数大于所定义的精度数，则截去超过的部分。

5）长度

长度格式符常用的有 h 和 l/L 两种。

h 和整数转换说明符一起使用，表示按短整型输出，例如：%hu，%hx，%hd。

l/L 和整数转换说明符一起使用，表示按长整型输出 long int 类型的数值，例如：%ld，%8lu。

【实例 3-15】 printf()函数。

```
#include <stdio.h>
void main()
{
    int a = 123;
    float b = 123.456789f;
    double c = 123.456789;
    char d = 'A';
    printf("a = %d, %5d\n",a,a + 1);
    printf("b = %f, %lf, %6.4f\n",b,b,b);
    printf("c = %lf, %f, %6.4lf\n",c,c,c);
    printf("d = %c, %8c\n",d,d);
    printf("%.4s\n","this is my test!");
    printf("%8.4s\n","this is my test!");
}
```

程序运行结果如图 3-17 所示。

图 3-17　printf()函数输出格式

程序分析：

(1) 语句 printf("a＝%d,%5d\n",a,a＋1);中的“a＝”和“,”为可打印字符，所以按原样输出。输出时在“%d”位置输出变量 a 的值，“%5d”位置输出表达式 a＋1 的值。其中“\n”为转义字符，输出时产生换行的效果。

(2) 在 printf()函数中，符号 l 对 f 类型无影响，所以%f 和%lf 格式的输出相同。

(3) 输出 float 实数 b 时，%6.4f 指定输出最小宽度为 6，精度为 4，由于实际长度超过 6 所以整数部分应该按实际位数输出，小数位数超过 4 位部分被截去。

(4) 输出字符量 d 时，%8c 指定输出最小宽度为 8，所以在输出字符 p 之前补加 7 个

空格。

(5) 输出字符串"this is my test!"时,精度.4 表示只输出串中的前 4 个字符。格式串"%8.4s\n"指定输出最小宽度为 8,超过了精度 4,所以在输出时补加 8－4＝4 个空格。

小提示: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。Visual C++6.0 是按从右到左进行求值的。

【实例 3-16】 printf()函数输出表项中的求值顺序。

```
#include <stdio.h>
void main()
{
  int i=8;
  printf("%d\n%d\n",++i,--i);
}
```

程序运行结果如图 3-18 所示。

图 3-18　printf()函数求值顺序

2. 标准输入函数 scanf()

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("格式化字符串",地址表);
```

其中的地址表项给出各变量的地址,如有多个,中间用逗号隔开。

其中的格式化字符串由三类字符构成:格式化说明符、空白字符和非空白字符。

1) 格式化说明符

格式化说明符的一般形式为:

```
%[*][输入数据宽度][长度]类型
```

各项的意义如下:

(1) 类型。常见的类型格式化说明符如表 3-7 所示。

表 3-7　常见的类型格式化说明符

格式化说明符	格式字符说明
%c	读入一个字符
%d	读入一个十进制整数
%o	读入一个八进制整数
%x 或%X	读入一个十六进制整数
%s	读入一个字符串(把空格也作为输入结束的标志)
%f	读入一个实数

续表

格式化说明符	格式字符说明
%e 或%E	读入一个实数
%u	读入一个无符号十进制整数
%%	读%符号

(2) 格式化字符串中的其他项说明如表 3-8 所示。

表 3-8　其他项说明表

修　饰　符	说　　明
长度	L/l：输入“长”数据；h：输入“短”数据
输入数据宽度	整型常数，指定输入数据所占宽度
*	空读一个输入数据或忽略一个输入数据

2) 空白字符

空白字符会使 scanf()函数在读操作中略去输入中的一个或多个空白字符，输入的空白字符可以是 Tab、空格和回车符等，直到第一个非空白符出现为止。

3) 非空白字符

一个非空白字符会使 scanf()函数在读入时剔除掉与这个非空白字符相同的字符。

【实例 3-17】 scanf()函数。

```
#include <stdio.h>
void main()
{
    int a,b,c;
    scanf("%d%d%d",&a,&b,&c);
    printf("%d,%d,%d\n",a,b,c);
}
```

程序分析：

(1) &a、&b、&c 中的 & 是取地址运算符，分别获得这三个变量的内存地址。scanf()函数中的变量前的取地址运算符不可省略。

(2) "%d%d%d"的含义是输入三个数值都需是十进值整型的格式。输入时，在两个数据之间可以用一个或多个空格、Tab 键、回车键分隔，不可以用逗号和其他非空白字符分隔。例如：

① 如图 3-19(a)所示，程序运行时输入 3□4□5↙(本节约定用“□”来表示空格符，用“↙”表示回车符)时，三个变量可以得到正确的值。

(a)　(b)　(c)

图 3-19　scanf()函数的数据输入方式 1

② 如图 3-19(b)所示，程序运行时输入 3↙4↙5↙时，三个变量可以得到正确的值。

③ 如图 3-19(c)所示，程序运行时输入 3,4,5↙时，后两个变量没能得到正确的值。

scanf()的格式控制串可以使用其他非空白字符，但在输入时必须输入这些字符。

【实例 3-18】 把实例 3-17 的代码稍作改动，变成：

```
#include <stdio.h>
void main()
{
    int a,b,c;
    scanf("%d,%d,%d",&a,&b,&c);
    printf("%d,%d,%d\n",a,b,c);
}
```

如图 3-20(a)所示，程序运行时输入 3,4,5↙时，三个变量可以得到正确的值。

需要注意：输入时","一定要跟在数字后面。如图 3-20(b)所示，程序运行时输入 3,□4,□5↙时，三个变量可以得到正确的值。如图 3-20(c)所示，程序运行时输入 3□,4□,5↙时，后两个变量没能得到正确的值。

(a) (b) (c)

图 3-20 scanf()函数的数据输入方式 2

思考：如果把语句 scanf("%d,%d,%d",&a,&b,&c);改为语句 scanf("a=%d,b=%d,c=%d ",&a,&b,&c);，程序运行时要如何输入呢？

小提示：

用 scanf()函数为 double 类型的变量输入值时不能用"%f"，而需用"%lf"或"%Lf"；但在用 printf()函数输出一个 double 类型的表达式时，既可以用"%f"，也可以用"%lf"或"%Lf"。

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);时输入：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。

3.6.2 字符输入输出函数

虽然使用 printf()和 scanf()函数可以实现单个字符的输出和输入，但 C 还提供了专门的单个字符输出和输入的函数 putchar()和 getchar()函数(准确地说，它们是在 stdio.h 中定义的宏)。

1. 字符输出函数 putchar()

putchar()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 可以是字符型的常量、变量和表达式，但不能是字符串；c 也可以是整型常

量、变量和表达式，但其值被作为 ASCII 值，输出 ASCII 值所对应的字符。

【实例 3-19】 putchar 函数的格式和使用方法。

```
#include <stdio.h>
 void main()
 {
     char ch = 'a';
     putchar(ch);
     putchar(98);
     putchar('\n');
     putchar(ch + 2);
     putchar('d');
     putchar('\n');
}
```

程序运行结果如图 3-21 所示。

图 3-21　实例 3-19 运行结果

2. 字符输入函数 getchar()

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()，则程序就直接从键盘缓冲区读入。

【实例 3-20】 getchar()函数的格式和使用方法。

```
#include <stdio.h>
 void main()
 {
    char ch1,ch2;
    printf("ch1 = ");
    ch1 = getchar();
    getchar();
    printf("ch2 = ");
    ch2 = getchar();
    printf("\nch1 = ");
    putchar(ch1);
    printf("\nch2 = ");
    putchar(ch2);
    putchar('\n');
 }
```

如图 3-22 所示，运行程序时输入 q↙ w↙。ch1 和 ch2 分别得到'q'和'w'。

图 3-22　实例 3-20 运行结果

思考：如果在本程序中去掉语句 getchar();，ch1 和 ch2 分别得到什么字符？

3.7　语 句 类 型

程序(或函数)的基本组成是语句。语句一般有输入、输出、赋值或控制等功能，通过对语句的组织，可以编写完成不同数据处理功能的程序(或函数)。因此语句是程序设计语言的基础。

3.7.1　说明性语句

说明性语句也叫注释，它是供编程人员阅读的，与程序的运行没有任何关系。注释的格式为“/* …… */”，注释语句可以出现在程序的任意位置，在很多编译环境下也可以用“//……”注释，但它只能出现在语句的行尾或独立为一行。例如：

【实例 3-21】 说明性语句示例。

```
/**************
作者:计算机学习第 1 小组
版本:1.0
程序功能:数组排序
所用算法:冒泡
**************/
#include<stdio.h>
void main()
{
    ⋮
}
```

3.7.2　表达式语句

表达式语句由表达式加上分号“;”组成。其一般形式为：

```
表达式;
```

执行表达式语句就是计算表达式的值。例如：

(1) x1=(-b+sqrt(b*b-4*a*c))/(2*a);　/*赋值语句*/

(2) a+b;　/*加法运算语句，但计算结果没有赋值给任何变量，无实际意义*/

(3) i++;　/*自增 1 语句，变量 i 的值增加 1 */

程序中对操作对象的运算处理大多通过赋值表达式语句(简称赋值语句)来实现。

3.7.3 控制语句

控制语句用于控制程序的流程,以实现程序的各种结构方式。它们由特定的语句关键字组成。C语言有9种控制语句,具体使用方法见第4章。控制语句可分成以下三类。

(1) 条件判断语句:if语句和switch语句。

(2) 循环执行语句:do while语句、while语句和for语句。

(3) 转向语句:break语句、goto语句、continue语句和return语句。

3.7.4 复合语句

把多个语句用括号{}括起来组成的一个语句称为复合语句。

在程序中应把复合语句看成是单条语句,而不是多条语句。例如:

```
{
  x = y + z;
  a = b + c;
  printf("x = %d,a = %d", x, a);
}
```

是一条复合语句,而不是三条语句。复合语句内的各条语句都必须以分号";"结尾,在括号"}"外不需再加分号。

3.7.5 空语句

只有分号";"而没有表达式的特殊语句称为空语句。空语句是什么也不执行的语句。对于空语句的应用举例如下。

(1) 有时需要纯粹消耗CPU时间,起到延时的作用,可以设计一个循环体为空语句的循环。例如:

```
for(i = 1;i < 1000;i++)
    {
        ;
    }
```

(2) 为了程序的结构清楚,可读性好,或为了方便自动测试,对于一些不完备的分支,可以用空语句补全。例如:

```
if(a!= 0)
    s += a;
else
    ;
```

(3) 为了以后扩充新功能方便而定义的函数的函数体部分可以暂时只用一个空语句代替。

3.7.6 函数调用语句

函数调用语句由函数名、实际参数加上分号";"组成。其一般形式为:

函数名(实际参数表);

函数调用语句的功能就是执行一次函数调用。例如：综合案例中的 showMenu();语句是对无参函数 showMenu()的一次调用，其作用是显示系统运行主界面。

printf("hello! ");语句的作用是调用 printf()函数，在屏幕上显示实参字符串"hello! "。

有参函数的调用原理会在函数一章中做出详细讲解。

3.8 应 用 实 例

在前面章节中，介绍了 C 语言的基本字符、标识符、运算符、数据类型转换、表达式、语句及常用的输入输出函数，本节详细分析一个简单实例，目的不仅是复习这些零碎的内容，也通过它来体味如何解决实际问题。

【实例 3-22】 编程求两个实数中的较大数。

思路分析：

(1) 选取数据类型——所要处理的数据的数据类型为实型，可以选择 float，也可以选择 double，它们表达的数据范围和精确度不同。假设要比较这两个实数不是太大，小数位数也不是太多，这两种类型都没问题。在此，选取 double。

(2) 确定变量个数——保存两个实数需要两个变量，为较大数也可以安排一个变量，这样共需三个实型变量。

(3) 命名变量——变量的名字属于 C 语言的标识符，为变量命名必须按照标识符的命名规则，不能出现非法的变量名。在此，为三个实型变量分别命名为 realnum1、realnum1 和 realmax。

(4) 定义变量——用变量定义语句。例如：double realnum1，realnum2，realmax;。

(5) 输入要处理的数据——调用 scanf()函数输入要比较的两个实数。比如：scanf("%lf，%lf"，&realnum1，&realnum2);。注意：接收从键盘上为 double 型变量输入的数据时需要的格式符是%lf，不是%f。

(6) 处理输入数据——根据输入的两个实数的大小求出较大的实数，把较大实数赋值给 realmax。在此，可以用一条带有条件表达式的赋值语句：realmax = realnum1 > realnum2? realnum1:realnum2;。

(7) 输出结果——调用 printf()函数输出结果。比如：printf("the max realnum of %lf and %lf is %lf \n"，realnum1，realnum2，realmax);。这里用%lf 与%f 的显示效果是相同的。

程序示例：

```
#include <stdio.h>
void main()
{
    double realnum1,realnum2,realmax;
    scanf("%lf,%lf",&realnum1,&realnum2);
    realmax = realnum1 > realnum2?realnum1:realnum2;
    printf("the max realnum of %lf and %lf is %lf \n",realnum1,realnum2,realmax);
}
```

当然，输入输出可以有多种形式，本程序只列出其中一种而已。

(1) 一般情况下，会在调用 scanf()函数前加上一条输入数据的提示。比如该程序中可以在语句 scanf("%lf,%lf", &realnum1, &realnum2);前加上一句 printf("please input realnum1 and realnum2);。

(2) 为方便输入，也可以把 scanf("%lf,%lf", &realnum1, &realnum2);改成以下 4 条语句：

```
printf("realnum1 = ");
scanf(" % lf ", &realnum1);
printf("realnum2 = ");
scanf(" % lf ", &realnum2);
```

思考：求一个三角形的周长时，需要定义什么类型的变量？共需几个变量？从键盘输入什么？从屏幕输出什么？试着编写程序。

习　　题

1. 填空题

(1) C 语言的标识符只能由字母、_____和_____组成。

(2) C 语言的关键字规定全部由_____字母组成。

(3) C 语言的转义字符是由_____符号开始的单个字符或者若干个字符组成的。

(4) C 语言的自定义标识符是由_____或_____开头的字母，数字，下划线组成的一串符号。ANSI C 规定标识符的长度是小于等于_____个字符。

(5) char w; int x;float y;则表达式 w * x+z-3.14 的结果类型是_____。

(6) 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_____。

(7) 今天是星期五，100 天后是星期_____。

(8) sin17°的 C 语言表达式为_____。(注：使用函数 sin(x)可以求出弧度值为 x 的正弦值。)

(9) C 语言的语句主要分为_____语句、_____语句、_____语句、_____语句、_____语句、_____语句、_____语句等。

(10) $\left|\frac{a+bc}{abc}\right|$ 的 C 语言表达式为_____。(注：使用 fabs(x)可以求出 x 的绝对值。)

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

2. 以下哪个是合法的八进制整数？

081　80　0x100　0Xff　0101　1010

3. 以下哪些是合法的十六进制整数？

0x77　0XABC　0Xacd　0x001　oX34　oxfff　0XeffA　0101

4. 以下哪些是合法常量？

1.234e04　1.234e0.4　1.234e+4　1.234e0　e-7　.4E0　1.8E　3.E4.　0.E1

.0E0　'\0'　'\010'　'\88'　'\101'　2L　1.2e-0　'\xxx'　'\x01'　'\xab'　'\Xabc'

5. 以下哪些是合法的变量名?

ab　count%　for　3a　num&　for_me

6. 阅读以下程序,分析输出结果。

```
#include <stdio.h>
void main()
{
    printf("%d\t%d\t%d\n",'a','a'+1,'b');
    printf("%c\t%c\t%c\n",'a','a'+1,'b');
}
```

7. 阅读以下程序,分析程序的功能。

```
#include <stdio.h>
void main()
{
    int a,b,c,*p=0;
    printf("a=");
    scanf("%d",&a);
    printf("b=");
    scanf("%d",&b);
    printf("c=");
    scanf("%d",&c);
    p=a>b?&a:&b;
    p=*p>c?p:&c;
    printf("*p=%d\n",*p);
}
```

8. 编写一个求两个整数中的最小数的程序。

9. 不用指针,编写一个求三个实数中的最小数的程序。

10. 编写一个程序,测试 printf()函数中格式化字符串中各项的作用。