第 3 章　C 语言流程控制

在本章中将学习到如下内容：

- 格式化输入/输出函数。
- C 语言中的顺序、分支和循环程序的设计方法。

通过前面的学习，我们已经能够编写一些比较简单的 C 程序了，但是在解决实际问题的时候，可能会遇到一些诸如满足某种条件进行怎样的处理，不满足又进行怎样的处理，或者某一个处理需要反复进行很多次处理这样的问题。那么在 C 语言中，怎样来解决这类问题呢？通过对本章的学习，将获得控制程序流程所需的全部基础知识。

3.1　格式化输入/输出函数

C 语言本身不提供输入/输出语句，输入/输出操作是由 C 函数库中的函数来实现的。在 C 标准库函数中提供了一些输入/输出函数。如在前面章节中使用过的 printf()函数和 scanf()函数。在使用的时候需要注意，不要误以为它们是 C 语言提供的“输入/输出语句”。printf 和 scanf 不是 C 语言的关键字，而只是函数的名字，实际上可以完全不用 printf 和 scanf 这两个名字，而另外编写两个输入/输出函数，用其他的函数名字来完成数据的输入和输出。printf()函数和 scanf()函数在系统文件 stdio.h 中声明，所以在程序的开始部分要使用编译预处理命令“#include <stdio.h>”

3.1.1　printf()函数

在日常生活中，经常要将华氏温度转换成摄氏温度，其转换公式如下：

$$c = \frac{5 \times (f - 32)}{9}$$

式中：c 表示摄氏温度，f 表示华氏温度。

程序清单 3.1　3_1.c 程序

```
#include <stdio.h>
void main()
{
    /*定义两个浮点型变量，celsius 表示摄氏温度，fahr 表示华氏温度*/
    float celsius, fahr;
    fahr = 100;                                          /*对变量 fahr 赋初值*/
    celsius = 5 * (fahr – 32) / 9;                       /*温度转换*/
    printf("fahr = %f, celsius = %f\n", fahr, celsius);  /*输出结果*/
}
```

程序的运行结果如图 3.1 所示。

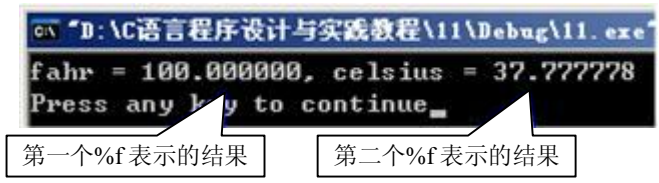


图 3.1　程序 3_1.c 运行结果

程序中调用 printf()函数输出结果时，将双引号内除了“%f”以外的内容原样进行输出，并在第一个“%f”的位置上输出变量 fahr 的值，在第二个“%f”位置输出变量 celsius 的值。

可以看出，printf()函数一般的调用格式为：

printf("格式控制字符串", 输出参数 1, 输出参数 2, …, 输出参数 n);

“格式控制字符串”是用双引号括起来的字符串（在 C 语言中用双引号括起来的都是字符串），也称作“转换控制字符串”，它包括两种信息。

（1）格式说明。格式说明由“%”和格式字符组成，如%d、%f 等，它的作用是将输出的数据转换成指定的格式输出。表 3.1 给出了格式说明符和用这些格式说明符输出的数据类型。

表 3.1　printf 函数的格式说明符及打印输出结果

格式说明	输出
%c	一个字符
%d	有符号十进制整数
%e	浮点数，以指数的形式输出（如 1.2e + 02）
%E	浮点数，以指数的形式输出（如 1.2E + 02）
%f	以小数的形式输出浮点数、十进制记数法
%g	根据数值的不同自动选择%f 或%e。%e 格式在指数小于–4 或者大于等于精度时使用
%G	根据数值的不同自动选择%f 或%E。%e 格式在指数小于–4 或者大于等于精度时使用
%o	无符号八进制整数
%s	字符串
%u	无符号十进制整数
%x	使用十六进制数字 0f 的无符号十六进制整数
%X	使用十六进制数字 0F 的无符号十六进制整数
%%	输出一个%

表 3.1 种列出了常用的格式说明符及其打印输出的结果。在读 C 程序的时候可能会发现 printf 函数还有如下几种形式：

printf("%ld", sum)， printf("%4.2f", sum)， printf("%–5.3s", s) …

细心的读者已经发现这几个 printf 函数与刚才所介绍的有一些区别，那么它们又具有什么意义呢？实际上，在 C 程序中，在表 3.1 的格式说明中的“%”和格式字符之间可以插入几种附加符号，如表 3.2 所示。

表 3.2　printf 函数的附加格式说明符

字符	说明
l	用于长整型整数，可加在格式符 d、o、x、u 前面
m（正整数）	数据最小宽度
n（正整数）	对于实数，表示输出 n 位小数；对于字符串，表示截取的字符个数
–	输出的数字或者字符在域内向左靠

在使用表 3.2 中的几种附加格式说明符的时候需要注意以下几点：

- 对于%md 和%ms，m 为指定输出数据的宽度，如果数据的位数小于 m，则左端补空格，若大于 m，则按实际位数输出。例如：

  printf("%4d, %4d", a, b)

  若 a = 12，b = 12345，则输出结果为：␣␣12，12345
- 对于%–ms，如果字符串的长度小于 m，则输出 m 列，不足的位数在右侧补空格。
- 对于%m.s，输出占 m 列，但只是取字符串中左端 n 个字符，这 n 个字符输出在 m 列的右侧，左端补空格。%–m.s 则在右侧补空格，如果 n > m，则 m 自动取 n 值，即保证 n 个字符正常输出。
- 对于%m.nf，指定输出的数据共占 m 列，其中 n 位小数。如果数值长度小于 m，则左端补空格。%–m.nf 与%m.nf 一样，只是使输出的数值向左端靠，右端补空格。

（2）普通字符。普通字符即需要原样输出的字符，如 printf("%d + %d = %d\n", a, b, sum)里面的+、=、空格和换行符（“\n”）。

输出参数是需要输出的一些数据。在程序 3_1.c 中，可以看到里面的空格和“=”都原封不动的输出，“%f”则转换成了相应变量的值输出。“\n”则起到一个换行的作用，将当前位置移到下一行的开头。

3.1.2　scanf()函数

在程序 3_1.c 中，将华氏温度转换成了摄氏温度，但程序的结果只是将华氏温度为 100 度的数据转换成了摄氏温度，很多时候，我们不希望数据是固定不变的，而是根据具体的需要，由用户来输入相应的数据。将程序稍加改动，输入任意的数据，程序都会给完成相应的转换。

程序清单 3.2　3_2.c 程序

```
#include <stdio.h>
void main()
{
    /*定义两个浮点型变量，celsius 表示摄氏温度，fahr 表示华氏温度*/
    float celsius, fahr;
    printf("Please input fahr:");
    scanf("%f", &fahr);                              /*调用 scanf 函数为变量 fahr 赋值*/
    celsius = 5 * (fahr – 32) / 9;                   /*温度转换*/
    printf("fahr = %f, celsius = %f\n", fahr, celsius);   /*输出结果*/
}
```

程序的运行结果如图 3.2 所示。

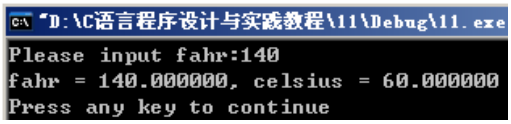


图 3.2　程序 3_2.c 运行结果

可以看出，程序使用了 scanf 函数，用户就可以任意地输入数据，然后程序将用户输入的华氏温度数据转换成对应的摄氏温度。

scanf 函数的作用是输入数据，其基本格式为：

scanf("格式控制"，地址列表);

“格式控制”的含义同 printf 函数的格式控制一样。其格式说明也是以%开始，以一个格式字符结束，中间可以插入附加的字符。表 3.3 和表 3.4 分别列出了 scanf 函数可以用到的格式字符和附加字符。

表 3.3　scanf 函数的格式说明符

格式说明符	意义
%c	输入单个字符
%d	输入有符号的十进制整数
%f	输入浮点数，可以用小数或者整数形式输入
%o	输入无符号的八进制数
%s	输入字符串，将字符串存放到一个字符数组中，输入时以非空白字符开始，以第一个空白字符结束，字符串以串结束标志'\0'作为其最后一个字符
%u	输入无符号的十进制整数
%x，%X	输入无符号的十六进制整数
%e，%E，%g，%G	与%f 的作用相同，e、f、g 可以互相替换而且大小写的作用相同

表 3.4　scanf 函数的附加格式说明字符

字符	意义
l	用于输入长整型数据，可用于%ld、%lo、%lx、%lu 以及 double 型数据%lf 或%le
h	用于输入短整型数据，可用于%hd、%ho、%hx
域宽	指定输入数据所占的列宽，应为一个正整数
*	表示本输入项在读入后不赋给相应的变量

为了更清楚地了解 scanf 函数，现在来研究一下 scanf 函数怎样读取输入。如果在程序中使用了一个%d 格式说明来读取一个十进制整数。scanf 函数每次读取一个输入字符，因为它试图读取一个整数，所以 scanf 函数希望发现一个数字字符，如果它发现了一个数字就保存之并读取下一个字符；如果接下来这个字符仍然是数字，它保存这个数字，并继续读取下一个字符。就这样，scanf 函数持续读取和保存字符直到它遇到一个非数字的字符。遇到非数字的字符就得出结论：已经读到了整数的尾部，读取的工作已经结束了。scanf 函数就会把这个非数

字的字符放回输入。这就意味着当程序下一次开始读取的时候，它将从放回输入的这个字符开始。最后，scanf 函数就会将自己读到的数据放到指定的变量中去（在地址表列中所指定的变量）。

如果在 scanf 函数中使用了域宽，那么 scanf 函数在读取字符个数的时候按照域宽所指定的数目来读取或者是第一个空白字符处（两者最先达到的一个）终止。scanf 函数在读取一个十进制整数的时候，如果读取的第一个非空白字符不是数字，将会发生什么现象呢？例如 scanf（"%d%d"，&m, &n），而我们输入的是“a56”，这时 scanf 函数读取 a 之后，发现并不是自己所期望的数字，按照上面的介绍，它将这个非数字的字符会放回输入，并没有把任何的值赋值给指定的变量 m。程序下一次读取时将在 a 处重新开始。如果 scanf 函数中只有%d 这种格式说明符的话，scanf 函数永远也不会越过 a 去读取后面的字符。

可能读者会感到迷惑，要是 scanf 函数读取的第一个字符就是空白字符，那它会做怎样的处理？事实上，如果 scanf 函数读取的第一个字符或者最前面几个连续的字符是空白字符（空格、制表符和换行符）的话，它将跳过这些空白符。

在使用 scanf 函数的时候，还需要注意的是如果使用%s 这个格式说明符，那么除了空白字符以外的所有字符都是可以接受的。也就意味着 scanf 函数跳过最前面的空白字符，然后保存再次遇到空白字符之前的所有非空白字符。这就意味着%s 使 scanf 函数每次只能读取一个单词或者说一个不包含空白字符的字符串。那么可不可以使用域宽来使 scanf 函数在使用%s 读取输入的时候超过一个单词的长度？实际上，使用域宽的话，scanf 函数将在域宽长度结束处或者第一个空白字符处停止，所以不能通过域宽来使 scanf 函数用一个%s 格式说明符的时候读取多于一个单词的输入。在使用%s 将读取的内容存放在字符数组中的时候，系统自动在最后一位添加字符串结束标志“\0”。如果使用%c 格式说明符，那么所有的输入字符都是一样的，如果输入空格或者换行符的话，会将空格或者换行符赋值给相应的变量。

在 scanf 函数，允许把除表 3.3 和表 3.4 之外的其他的普通字符放到格式字符串中。除了空白字符之外的普通字符一定要与输入的字符串准确匹配。例如：

```
scanf("%d, %d", &m, &n);
```

在两个格式说明符之间有一个普通字符逗号，这个逗号在输入的时候必须输入，否则变量将得不到正确的赋值。必须像下面这样输入两个整数：

```
24, 32
```

因为在格式字符串中，逗号紧跟在第一个%d 的后面，所以在输入的时候这个逗号也必须紧跟在 24 的后面。由于 scanf 函数在读取数据的时候将会忽略要读取的数据前面的一些空白字符，所以可以在逗号后面键入一个或者任意多个空白字符。也就意味着下面的输入方式也可以接受：

```
24,        32
```

或者

```
24,
32
```

在 scanf 函数中，在格式控制字符串后面就是地址列表，需要注意，这里是地址列表而不是变量列表，也就意味着像下面这样来书写程序是不对的：

```
scanf("%d %d", m, n);
```

应该将“m, n”改为“&m, &n”。其中“&”就是 C 语言的取地址运算符。这一点初学者

一定要注意，不要在这里出现错误。

3.2　程序流程图

流程图是用一些图框来表示各种操作。用流程图来表示算法，直观形象，易于理解。美国国家标准化协会（ANSI）规定了一些常用的流程图符号，如图 3.3 所示，为世界各国程序员普遍采用。

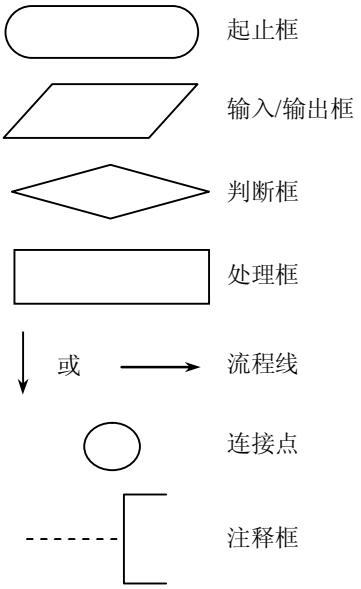


图 3.3　流程图的基本元素

图 3.3 中的菱形框的作用是对一个给定的条件进行判断，根据给定的条件是否成立来决定如何执行其后的操作。它有一个入口，两个出口，如图 3.4 所示。

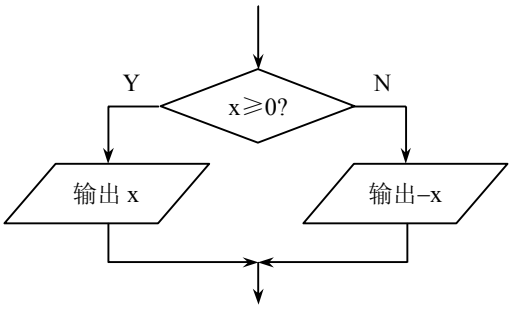


图 3.4　条件判断流程图

连接点（图 3.3 里面的小圆圈）是用于将画在不同地方的流程线连接起来。在流程图中，编号相同的连接点表示这些点是互相连接在一起的。注释框不是流程图中的必要部分，不反应流程和操作，只是为了对流程图中某些框的操作作必要的补充说明，以便于人们更好地理

解流程图。

1966 年，Bohra 和 Jacopini 提出了用顺序结构、选择结构和循环结构这三种基本结构可以表示任意复杂的程序。如果能使用流程图来表示这三种结构，也就意味着可以使用流程图来表示任意复杂的程序了。下面来研究一下怎样使用流程图来表示这三种基本结构。

（1）顺序结构。如图 3.5 所示，虚线框内是一个顺序结构。其中 A 和 B 两个框是顺序执行的。执行完 A 框指定的操作后接着执行 B 框所指定的操作。顺序结构是最简单的一种基本结构。

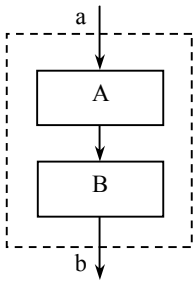


图 3.5　顺序结构

（2）选择结构。如图 3.6 所示，虚线框内是一个选择结构。此结构中必须包含一个判断框，根据给定的条件 p 是否成立来选择执行 A 框还是 B 框中所指定的操作。需要注意的是，无论 p 条件是否成立，只能执行 A 框或者 B 框之一，两者不可能同时执行。A 框或者 B 框有一个可以是空的，不执行任何操作。

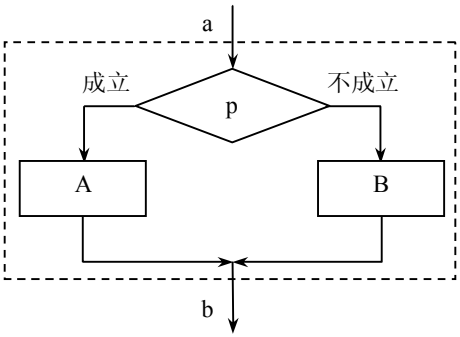
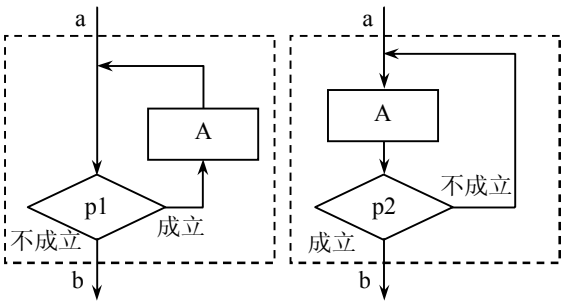


图 3.6　选择结构

（3）循环结构。又称重复结构，即反复执行某一部分的操作。可以分为以下两类：

- 当（while）型循环结构。如图 3.7（a）所示。其功能是：当给定的条件 p1 成立时，执行 A 操作，执行完 A 后，再判断条件 p1 是否成立，如果仍然成立，再执行 A 框，如此反复执行 A，直到某一次条件 p1 不成立为止，此时不执行 A，从 b 点脱离循环结构。
- 直到（until）型循环结构。如图 3.7（b）所示。其功能是：先执行 A，然后判断给定的条件 p2 是否成立，如果 p2 条件不成立，则再执行 A，然后再对 p2 条件作判断，如果 p2 条件仍然不成立，又执行 A……如此反复执行 A，直到给定的条件 p2 成立为止，此时不再执行 A，从 b 点脱离循环结构。



（a）当型循环结构　　（b）直到型循环结构

图 3.7　循环结构

已经证明，由以上 3 种基本结构组成的程序算法结构，可以解决任何复杂的问题。由基本结构所构成的程序属于“结构化”的程序，它不存在无规律的转向。只在基本结构内才允许存在分支和向前或向后的流程跳转。

3.3　顺序结构程序设计

顺序结构是 C 语言中比较简单的程序结构。顺序结构的程序是从 main()函数的第一句开始执行，直到 main()函数的最后一句。中间如果有函数调用，则会转到被调函数处执行，执行完被调函数后返回主调函数继续开始往下执行。在这个过程中，不存在分支结构和循环结构。

为了进一步了解顺序结构程序，下面来看几个顺序程序的例子。

求解一元二次方程 $ax^2+bx+c=0$ 的根。在这里系数 a、b 和 c 由键盘输入，且 a≠0。同时为了简单起见，假设输入的 a、b 和 c 满足 $b^2-4ac>0$。

根据我们现有的数学知识，很容易就可以求出满足上面条件的一元二次方程的两个根，分别为：

$$x_1=\frac{-b+\sqrt{b^2-4ac}}{2a}，\quad x_2=\frac{-b-\sqrt{b^2-4ac}}{2a}$$

为了编写程序方便，令 $p=\frac{-b}{2a}$，$q=\frac{\sqrt{b^2-4ac}}{2a}$，则方程的两个根为：

$$x_1=p+q，\quad x_2=p-q$$

根据上面的分析，求解一元二次方程的程序如程序清单 3.3 所示。

程序清单 3.3　3_3.c 程序

```
#include <stdio.h>
#include <math.h>
void main()
{
    float a, b, c, disc, x1, x2, p, q;          /*float 定义浮点型数据，即实型数据*/
    scanf("%f%f%f", &a, &b, &c);
    disc = b * b – 4 * a * c;
    p = –b / (2 * a);
    q = sqrt(disc) / (2 * a);                   /*sqrt()是库函数，功能是求平方根*/
```

```
    x1 = p + q;
    x2 = p – q;
    printf("x1 = %5.2f\nx2 = %5.2f\n", x1, x2);
}
```

程序 3_3.c 的运行结果如图 3.8 所示。

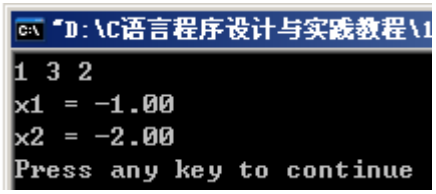


图 3.8　程序 3_3.c 的运行结果

从图 3.8 可以看出，方程 $x^2+3x+2=0$ 的根为 $x_1=-1$，$x_2=-2$。

3.3.1　算术运算符

在程序 3.3 中进行了很多的数学运算，如加（+）、减（–）、乘（*）、除（/）以及求平方根（sqrt 函数）等。需要注意的是，求平方根的 sqrt 函数不是 C 语言的算术运算符，它是 C 系统提供的一个库函数，在使用的时候，必须在程序的开头使用文件包含预处理指令#include 将头文件 math.h 包含到程序中。在编写程序中凡是要用到数学函数库中的函数，都应该将 math.h 包含到程序中。

C 语言中基本的算术运算符有如下 5 种：

（1）+（加法或者正值运算符，如 5+8、+7 等）；

（2）–（减法或者负值运算符，如 9–4、–7 等）；

（3）*（乘法运算符，如 6*8。在 C 语言中乘法运算使用“*”，而不是“×”）；

（4）/（除法运算符，如 6/4。在 C 语言中除法运算使用“/”，而不是“÷”）；

（5）%（求余运算，“%”的两侧要求都是整型数据，如 9 % 4 = 1）。

需要说明的是，在使用除法运算符“/”的时候，如果除数和被除数都是整数，则商也一定是整数。如 5/2 = 2 而不是 2.5，系统将舍去小数部分。但是如果参加运算的又一个是负数，则小数的舍入方向不固定。例如–5/2，在有的系统值为–2，有的系统为–3。大多数系统，如 Turbo C 采用的是“向零取整”。即 5/2 = 2、–5/2 = –2，取整后向零靠拢。

小提示：C 语言中平方的表示方法

在 C 语言中没法直接来表示两个数的平方 x^2，而是采用的两个 x 连乘的方法，例如 x * x。同理 x^3 的表示方法为：x * x * x。其他以此类推，当然幂次比较大的这种办法就不太实用了，需要采用本章后面即将学到的循环结构程序来处理，或者利用库函数 pow 来处理。

3.3.2　算术表达式

用算术运算符和括号将运算对象连接起来，符合 C 语法规则的式子就称为算术表达式。运算对象包含常量、变量、函数等。程序 3_3.c 中的“b*b – 4*a*c”、“–b/(2*a)”、“sqrt(disc)/(2*a)”

等都是算术表达式。

每一个算术表达式都会有一个确定的值，在表达式求值时，先按照算术运算符的优先级别来进行运算。例如先乘除后加减。如果一个运算对象两侧的运算符的优先级别相同，则按照规定的“结合方向”来处理。

C 语言中算术运算符的结合方向为“自左至右”。例如对于 a–b+c 这样的表达式，加法和减法运算的优先级别是相同的，根据其结合性，b 先与“–”结合，执行 a–b 的运算，然后再执行加 c 的运算。更多的运算符的优先级别和结合性参考附录三。

3.3.3　赋值表达式

由赋值运算符将一个变量和一个表达式连接起来的式子称为赋值表达式。其一般形式为：
＜变量＞＜赋值运算符＞＜表达式＞

如在程序 3_3.c 中的 p = –b/(2*a)、q = sqrt(disc)/(2*a)等就是赋值表达式。其计算过程是：先计算赋值运算符右侧的表达式的值，然后将这个值赋值给左边的变量。赋值表达式与其他的表达式一样也有一个具体的值，其值就是被赋值的变量的值。

在赋值表达式中，右侧的表达式也可以仍然是一个赋值表达式，如图 3.9 所示。

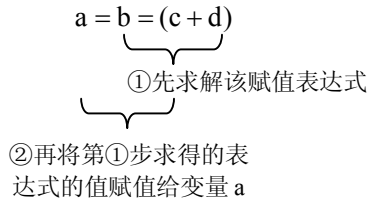


图 3.9　赋值表达式

从附录三可以知道赋值运算符的结合性是“自右而左”的结合顺序，因此系统会先求解 b = (c + d)，求解之后将这个赋值表达式的值，即变量 b 的值在赋值给变量 a，即求解表达式 a = b 。

知识点补充：复合赋值运算符

在赋值运算符“=”之前加上其他的运算符，就构成了复合赋值运算符。C 语言中提供的常用的复合赋值运算符和运算规则如表 3.5 所示。

表 3.5　常用的复合赋值运算符

a += y	等价于	a = a + y
b –= y	等价于	b = b – y
c *= y	等价于	c = c * y
d /= y	等价于	d = d / y
e %= y	等价于	e = e % y

在表 3.5 中的这些复合赋值运算符与“=”的优先级别相同。使用这些复合赋值运算符可以使表达式更加简洁，与更长的赋值表达式相比，可能会产生效率更高的机器代码。专业的

人士比较喜欢使用复合的赋值运算符，对于初学者来说，更重要的是保持程序的清晰易懂。

表中的 y 可以是常量、变量以及更复杂的表达式。

随堂练习

编写程序：从键盘上输入三角形的三边的边长，根据下面的公式求出三角形的面积 area。

$$area = \sqrt{s(s-a)(s-b)(s-c)}$$

假设输入的 a、b 和 c 三边的边长能够组成一个三角形。

3.4 选择结构程序设计

选择结构是 3 种基本结构之一。其作用是根据不同的条件来执行不同的操作。在 C 语言中，是否满足某个条件是靠关系表达式和逻辑表达式来实现的。在解决实际问题中经常会用到这种根据不同的条件来执行不同的操作。程序 3_4.c 就是将程序 3_3.c 进行了改进的一个程序，它根据 b^2-4ac 的值来判定方程 $ax^2+bx+c=0$ 根的情况。

程序清单 3.4　3_4.c 程序

```
#include <stdio.h>
#include <math.h>
void main()
{
    float a, b, c, disc, x1, x2, p, q;
    scanf("%f%f%f", &a, &b, &c);
    disc = b * b – 4 * a * c;
    p = –b / (2 * a);
    q = (float)sqrt(fabs(disc)) / (2 * a);        /*fabs()函数，求实数的绝对值*/
    if(disc > 1e–6)                               /*disc > 0，方程有两个不同的实根*/
    {
        x1 = p + q;
        x2 = p – q;
        printf("x1 = %5.2f\nx2 = %5.2f\n", x1, x2);
    }
    else if(fabs(disc) <= 1e–6)                   /*disc = 0，方程有相等的两个实根*/
    {
        printf("x1 = x2 = %5.2f\n", p);
    }
    else                                          /*否则，即 disc < 0 的情况，方程有两个共轭虚根*/
    {
        printf("x1 = %5.2f + %5.2fi\nx2 = %5.2f – %5.2fi\n", p, q, p, q);
    }
}
```

其运行结果如图 3.10 所示。

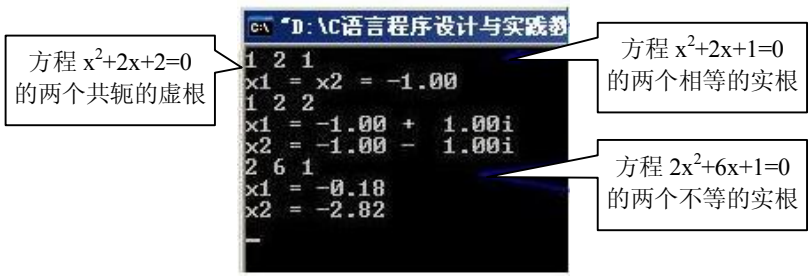


图 3.10　程序 3_4.c 运行结果

3.4.1　关系运算符和关系表达式

在数学上，经常出现一些比较大小的运算。例如，求解一元二次方程 $ax^2+bx+c=0$ 的时候，需要根据 b^2-4ac 的值是否小于 0 来决定该方程的根是虚根还是实根。这种比较大小的运算在 C 语言中可以用关系运算符来实现。

1. 关系运算符及其优先级别

在 C 语言中提供了 6 种关系运算符，如表 3.6 所示。在表中，前 4 种关系运算符（<、<=、>、>=）的优先级别相同，后 2 种（==、!=）的优先级别也相同，且前 4 种高于后 2 种。关系运算符的优先级别低于算术运算符，高于赋值运算符。

表 3.6　关系运算符

运算符	含义
<	小于
<=	小于或等于
>	大于
>=	大于或等于
==	等于
!=	不等于

2. 关系表达式

用关系运算符将两个表达式连接起来的式子就称为关系表达式。如：

$$a>b,\quad b*b-4*a*c>0,\quad (a=4)<(b=1),\quad (a>b)!=(c<d)$$

都是合法的关系表达式。

关系表达式也有一个值，与算术表达式不一样，关系表达式所描述的条件只有两种情况，“成立”或者“不成立”。所以在 C 语言中用“真”或者“假”来描述这两种情况。例如，“5<6”的值是“真”而“4>=10”的值是“假”。

在 C 语言中没有逻辑型数据来描述“真”或“假”，而是采用“1”或“0”来反映关系表达式的结果。如果某个关系表达式的值为“真”，则这个关系表达式的值就是“1”，如果为“假”，则值为“0”。

小提示

在使用关系运算符来判断某一个值为实数的表达式是否等于 0 时，由于实数在计算和存储时，会出现一些误差，因此不能直接使用“<表达式> == 0”来判断，因为这样可能会出现本来是 0 的值，但是由于存储误差而被判断成不等于 0。所以采取的办法是判断这个表达式的值的绝对值是否小于一个很小的数，如 10^{-6}，如果小于此数，就认为表达式的值等于 0。如程序 3_4.c 中的“fabs(disc) <= 1e–6”。

3.4.2 逻辑运算符和逻辑表达式

实际上，我们在决定做什么样的操作时有时候需要几个条件同时成立，有时候只需要在很多条件中满足其中的一个就可以。像这种需要几个条件同时成立或者多个条件中只要有一个成立的情况，单单靠关系运算符还不够，需要靠逻辑运算符来完成。

1. 逻辑运算符及其优先级别

C 语言中提供了 3 种逻辑运算符，如表 3.7 所示。表中“&&”和“||”是双目运算符，要求有两个运算量，如(a < b) && (c > d)、(a < b)||(c > d)。“!”是单目运算符，只需要一个运算量，如!(a < b)。

表 3.7 逻辑运算符

运算符	举例	运算规则
&&（逻辑与）	a && b	若 a、b 同时为真，a && b 为真，除此之外都为假
||（逻辑或）	a || b	若 a、b 之一为真，则 a || b 为真
!（逻辑非）	!a	若 a 为真。则!a 为假，反之，若 a 为假，则!a 为真

逻辑运算符的优先级别由高到低是：“!”、“&&”、“||”。其中“&&”和“||”低于关系运算符，而“!”高于算术运算符。

2. 逻辑表达式

用逻辑运算符将关系表达式或逻辑量连接起来的式子就是逻辑表达式。跟关系表达式一样，逻辑表达式的值也是一个逻辑量“真”或“假”。在表示逻辑表达式的计算结果的时候，仍然使用数值“1”代表逻辑真，以数值“0”代表逻辑假。但在判断某一个量是否是真的时候，则是以“0”值代表假，以“非 0”值代表真。如：

若 a = 5，b = 0，c = 2，则!a 的值为 0，因为 a 的值为 5，是一个非 0 值，被认作“真”。同理!b 的值为 1，a && c 的值为 1。

小提示：逻辑表达式的求解

在求解逻辑表达式的过程中，并非所有的逻辑运算符都被执行，只是在必须执行下一个逻辑运算符才能求出表达式的解时，才执行该运算符。如：

（1）a && b && c 只有在 a 为“真”的时候，才需要判断 b 的值，同时，也只有 a 和 b 都为“真”的情况下才需判别 c 的值。因为，只要 a 为“假”，不必判别 b 和 c 就可以确定整个表达式的值为“假”。同理，若 a 为“真”，b 为“假”，也不必判别 c。

（2）a || b || c 只要 a 为“真”，就不必判别 b 和 c。只有 a 为“假”才判别 b，a 和 b 都为

“假”，才判别 c。

例如当 a = 1，b = 2，c = 3，d = 4，m = 1，n = 1 在执行表达式(m = a > b)&&(n = c > d)后，由于“a > b”的值为“0”，因此 m = 0，而“n = c > d”不被执行，此时 n 仍保持原值“1”。

C 语言中的语句（一）

（1）表达式语句：表达式语句由一个表达式加上一个分号“;”构成。如 a = 3 是一个赋值表达式，而 a = 3; 是一个赋值语句，分号“;”是语句中不可缺少的组成部分，任何表达式加上“;”都成为相应的语句。

（2）函数调用语句：由一个函数调用加上一个“;”构成，如 printf("abc\n");。

（3）空语句：仅有一个分号的语句，它什么都不做。

（4）复合语句：可以使用{}把一些语句括起来就成为复合语句，如：

```
{
    z = x + y; t = z / 100; printf("%f", t);
}
```

3.4.3　if 语句与 switch 语句

从程序 3_4.c 中的

```
if(disc > 1e–6)
{
    …
}
else if(fabs(disc) <= 1e–6)
{
    …
}
else
{
    …
}
```

这组结构可以看出，if 语句是用来判定所给定的条件是否满足，然后根据判断的结果来选择做出什么样的操作。在 C 语言中，if 语句的使用方式有如下几种。

1. if 语句

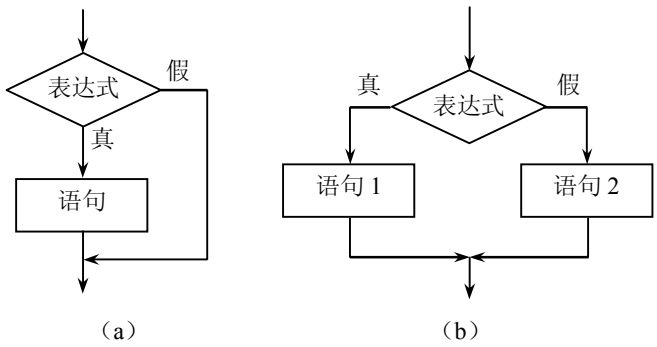
if 语句的一般格式如下所示：

```
if(表达式)
{
    语句组;
}
```

注意：这里没有“;”

如果表达式的值为“真”（或者非零），就执行语句组，否则程序就会跳过该语句组，直接执行 if 语句之后的其他语句。其程序流程如图 3.11（a）所示。通常，表达式是一个关系表达式或者逻辑表达式，根据表达式值的真假来判断是否执行语句组。更一般的情况，表达式可以是任意的表达式，表达式的值为 0 就被视为假，非 0 就被视为真。

当语句组只有一个语句的时候，if 语句的一对大括号可以省略。



（a）　　（b）

图 3.11　if 语句结构流程图

2. 在 if 语句中添加 else 子句

简单形式的 if 语句可以让根据条件的成立与否来判断是执行还是忽略某种操作。更多的时候，需要根据条件的成立与否在两个操作中选择一个来执行，这时候如果只是使用 if 语句，程序就显得笨拙，可以用 if…else 的形式来解决这个问题。

if…else 语句的一般形式如下：

```
if(表达式)
{
    语句组 1;
}
else
{
    语句组 2;
}
```

注意：这里没有“;”

注意：这里也没有“;”

if…else 语句的执行过程是：如图 3.11（b）所示，表达式的值为“真”，则执行语句组 1，否则，执行语句组 2。同样，当语句组 1 只有一条语句时，if 后面的一对大括号可以省略，语句组 2 只有一条语句时，else 后面的一对大括号也可以省略。

3. 多重选择 else if

在解决实际问题的时候，经常也会遇到多重选择的情况，这个时候就可使用 else if 来扩展 if…else 结构以适应这种情况。其一般格式如下：

```
if(表达式 1)
{
    语句组 1;
}
else if(表达式 2)
{
    语句组 2;
}
else if(表达式 3)
{
    语句组 3;
}
…
else if(表达式 m)
```

```
{
    语句组 m;
}
else
{
    语句组 n;
}
```

多重选择 else if 的程序流程如图 3.12 所示，当表达式 1 为真的时候则执行语句组 1，否则的话，将判断表达式 2 的真假，表达式 2 为真，则执行语句组 2，否则将判断表达式 3，……，表达式 m 为真，则执行语句组 m，否则执行语句组 n。同样，如果某一个语句组只有一条语句，相应的一对大括号可以省略。

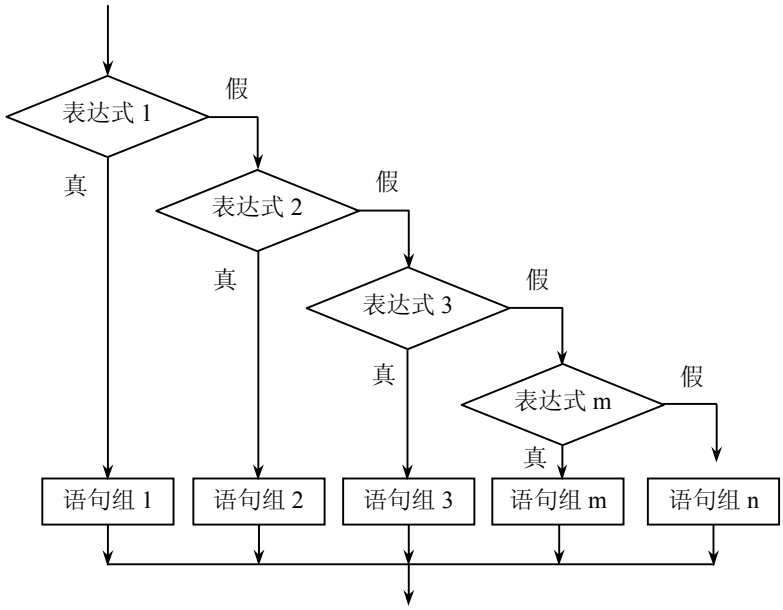


图 3.12　多重选择 else if 结构流程图

4. if 语句的嵌套

在 if 语句的语句组中又包含一个或多个 if 语句，称为 if 语句的嵌套。其一般形式如下：

```
if(表达式)
{
    if(表达式)
    {
        语句组 1;
    }
    else
    {
        语句组 2;
    }
}
```

内嵌 if 语句

同理，在每一个语句组里还可以继续嵌套 if 语句

```
else
{
    if(表达式)
    {
        语句组 3;
    }
    else
    {
        语句组 4;
    }
}
```

内嵌 if 语句

当有众多的 if 和 else 的时候，由于某些语句组只有一条语句的时候省略了大括号，计算机是怎样判断哪个 if 对应哪个 else 的？例如：

```
if（score >= 60）
    if（score <= 100）
        printf("PASS!\n");
else
    printf("FAILURE!\n");
```

什么时候输出“FAILURE!”？是在 score 小于 60 的时候，还是在 score 大于 100 的时候？即 else 是对应第一个 if 语句还是第二个？在 C 语言中，如果没有大括号指明，else 与和它最近接的一个 if 相匹配，即在 score 大于 100 的时候输出“FAILURE!”。程序的缩进使得 else 好像是与第一个 if 匹配的，但是，编译器是忽略缩排的。事实上，我们本来的意愿也就是需要与第一个 if 语句匹配，这个时候，需要加上大括号。如下所示：

```
if（score >= 60）
{
    if（score <= 100）
        printf（"PASS!\n"）;
}
else
    printf（"FAILURE!\n"）;
```

加上一对大括号

小提示：if 语句嵌套中的大括号

从技术的角度讲，if 和 if else 语句作为一个整体可以看成单个语句，所以在嵌套的时候可以不用加上大括号，然而，当语句很长的时候，大括号使人更容易读懂程序，更不容易出错。所以建议初学者将大括号都添加上去，不管语句组是一条语句还是多条。

使用 if 语句实现了多分支选择，可以看到，如果分支情况较多的时候，则需嵌套的 if 语句层数较多，程序冗长，可读性降低。在 C 语言中还可以使用 switch 语句来直接处理这种多分支选择的情况。它的一般形式如下：

```
switch(表达式)
{
    case  常量表达式 1:语句组 1;
    case  常量表达式 2:语句组 2;
    …
    case  常量表达式 n:语句组 n;
```

```
    default:        语句组 n+1;
}
```

例如：

```
switch(c)
{
    case 0: d = 0; break;
    case 1: d = 0.02; break;
    case 2:
    case 3: d = 0.05; break;
    case 4:
    case 5:
    case 6:
    case 7: d = 0.08; break;
    case 9:
    case 10:
    case 11: d = 0.1; break;
    case 12: d = 0.15;break;
}
```

对 switch 语句做以下几点说明：

（1）switch 后面括号内的表达式可以是任意类型的表达式，当表达式的值与某一个 case 后面的常量表达式的值相等时，就执行此 case 后面的语句，如果所有 case 后面的常量表达式的值都没有与表达式匹配的，则执行 default 后面的语句。default 可以省略。

（2）每一个 case 的常量表达式的值必须互不相同，否则就会出现互相矛盾的现象。而且其值必须是一个常量（在程序运行过程中，值不会发生改变的量）。各个 case 和 default 的出现次序不影响执行结果。

（3）执行完一个 case 后面的语句后，程序流程转移到下一个 case 继续执行，直到 switch 语句执行结束或者遇见 break 语句的时候才结束 switch 语句的执行。“case 常量表达式”只是起语句标号的作用，并不是在该处进行条件判断。在执行 switch 语句时，根据 switch 后面表达式的值找到匹配的入口标号，就从此标号开始执行，不再进行判断。因此应该在需要跳出 switch 结构的 case 分支处使用 break 语句来终止 switch 语句的执行。

（4）多个 case 可以共用一组执行语句。如上例的

```
case 4:
case 5:
case 6:
case 7: d = 0.08; break;
```

当 c 的值为 4、5、6 或 7 时，都执行“d=0.08;break;”这组语句。

随堂练习

有一个函数:

$$y=\begin{cases} x & x<1 \\ 2x-1 & 1\leqslant x<10 \\ 3x-11 & x\geqslant 10 \end{cases}$$

编写一程序，从键盘上输入 x 的值，根据上面的函数，求出 y 的值，并输出。

3.4.4 选择结构程序举例

程序 3_5.c 的功能是对于输入的两个实数，按照代数值由小到大的顺序输出。

程序清单 3.5　3_5.c 程序

```
#include <stdio.h>
void main()
{
    float a, b, t;
    scanf("%f%f", &a, &b);
    if(a > b)
    {
        t = a;
        a = b;
        b = t;
    }
    printf("%5.2f,%5.2f\n", a, b);
}
```

对变量 a 和变量 b 的值进行交换。注意变量 t 的作用

程序 3_5.c 的运行结果如图 3.13 所示。

图 3.13　程序 3_5.c 的运行结果

程序 3_6.c 的功能是对输入的三个实数，编程找出其中最大的一个并输出。

程序清单 3.6　3_6.c 程序

```
#include <stdio.h>
void main()
{
    float a, b, c, max;
    scanf("%f%f%f", &a, &b, &c);
    if(a > b)
    {
        max = a;
    }
    else
    {
        max = b;
    }
    if(c > max)
    {
        max = c;
```

⇔ max = (a > b) ? a : b;

```
    }
    printf("max = %5.2f\n", max);
}
```

程序 3_6.c 的思路很明确，先找出 a 和 b 中较大的一个，将其赋值给变量 max，然后比较 max 与 c 的大小，如果 c 大于 max，则将 c 的值赋值给 max。这样变量 max 的值就是 a、b 和 c 三个变量的值中最大的。程序运行的结果如图 3.14 所示。

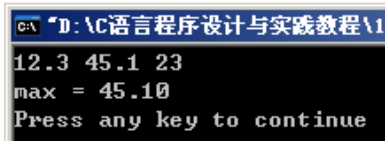


图 3.14　程序 3_6.c 的运行结果

程序清单 3.6 中的

```
if(a > b)
{
    max = a;
}
else
{
    max = b;
}
```

无论表达式“a > b”成立与否，都在执行一个赋值语句，并且向同一个变量 max 赋值。这几行程序可以用一个条件运算符来处理：

$max = (a > b)?a : b;$

其中“$max = (a > b)?a : b$”是一个“条件表达式”。它的执行过程是：如果（a>b）为真，则条件表达式取值 a，否则取值 b。

条件运算符要求有 3 个操作对象，称三目运算符，它是 C 语言中唯一的一个三目运算符。其一般形式为：

表达式 1？表达式 2:表达式 3

其执行顺序为：先求解表达式 1，若为真（非 0），则求解表达式 2，并把表达式 2 的值作为整个条件表达式的值。若表达式 1 为假（0），则求解表达式 3，并把表达式 3 的值作为整个条件表达式的值。

对于条件运算符的使用，需要注意以下几点。

（1）条件运算符的优先级别高于赋值运算符，但是比关系运算符和算术运算符要低。如 $max = (a > b)?a : b$ 这个表达式是先求解条件表达式，再将条件表达式的值赋值给变量 max。其中 $(a > b)$ 的括号也可以取消。

（2）条件运算符的结合方向为“自右向左”，如有：

$a > b?a : c > d?c : d$

等价于：$a > b?a : (c > d?c : d)$。为了程序的可读性好，建议加上括号。

（3）条件表达式中的表达式 2 和表达式 3 可以是任意的表达式。表达式 1 与表达式 2 和表达式 3 的类型也可以不同。

程序 3_7.c 为运输公司对客户计算运费。路程 s（单位：km）越远，每吨每千米运费越低。标准如下：

s < 250	没有折扣
250 ≤ s < 500	2%的折扣
500 ≤ s < 1000	5%的折扣
1000 ≤ s < 2000	8%的折扣
2000 ≤ s < 3000	10%的折扣
3000 ≤ s	15%的折扣

设每吨每千米货物的基本运费为 p，货物重为 w，距离为 s，折扣为 d，则总运费 f 的计算公式为

$$f = p \times w \times s \times (1 - d)$$

据此写出的程序如程序 3_7.c 所示。

程序清单 3.7　3_7.c 程序

```
#include <stdio.h>
void main()
{
    int s;
    double float p, w, d, f;
    scanf("%f%f%d", &p, &w, &s);
    if(s < 250)
    {
        d = 0;
    }
    else if(s >= 250 && s < 500)
    {
        d = 0.02;
    }
    else if(s >= 500 && s < 1000)
    {
        d = 0.05;
    }
    else if(s >= 100 && s < 2000)
    {
        d = 0.08;
    }
    else if(s >= 2000 && s < 3000)
    {
        d = 0.1;
    }
    else if(s >= 3000)
    {
        d = 0.15;
    }
```

```
    f = p * w * s * (1 – d);
    printf("freight = %15.4f\n", f);
}
```

程序 3_7.c 的运行情况如图 3.15 所示。

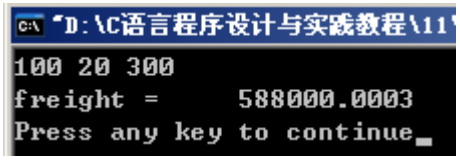


图 3.15　程序 3_7.c 的运行结果

小提示：几个条件同时成立的表示方法

在前面已经提到过，要表示几个条件同时成立，需要用到逻辑与（&&）运算符。但是很多初学者受数学上的表示方法的影响，往往直接使用“250 <= s < 500”这样的表达式来表示 s 大于或等于 250，同时 s 小于 500 这样的条件。恰好，这样书写，是一个合法的关系表达式，C 编译器不会报告错误信息。很多人误以为这个表达式能正确描述 s 在[250，500）这个区间内。实际上，根据关系运算符的结合性，表达式“250 <= s < 500”先计算“250 <= s”，其值是一个逻辑值，要么是值“1”，要么是值“0”，而无论是 1 还是 0，小于 500 肯定是成立的，所以表达式“250 <= s < 500”无论 s 取何值，该表达式的值都是“1”。无法正确表示 s 在区间[250，500）内。

下面再来分析例程序 3_7.c，折扣的变化是有规律的：折扣的变化点都是 250 的倍数。利用这一点，可以定义变量 c，c 的值为 s / 250 。c 代表 250 的倍数。当 c < 1 时，表示 s < 250，无折扣；1 ≤ c < 2 时，表示 250 ≤ s < 500，折扣 d = 2%；2 ≤ c < 4 时，d = 5%；4 ≤ c < 8 时，d = 8%；8 ≤ c < 12 时，d = 10%；c ≥ 12 时，d = 15%。据此，使用 switch 语句来实现程序清单 3_7.c。

程序清单 3.8　3_8.c 程序

```
#include <stdio.h>
void main()
{
    int c, s;
    float p, w, d, f;
    scanf("%f%f%d", &p, &w, &s);
    if(s >= 3000)
    {
        c = 12;
    }
    else
    {
        c = s / 250;
    }
    switch(c)
    {
        case 0: d = 0; break;
```

```
            case 1: d = 0.02; break;
            case 2:
            case 3: d = 0.05; break;
            case 4:
            case 5:
            case 6:
            case 7: d = 0.08; break;
            case 8:
            case 9:
            case 10:
            case 11: d = 0.1; break;
            case 12: d = 0.15; break;
        }
        f = p * w * s * (1 – d);
        printf("freight = %15.4f\n", f);
}
```

注意：c、s 是整型变量，因此 c = s / 250 为整数。当 s ≥ 3000 时，令 c = 12，而不使 c 随 s 增大，这是为了在 switch 语句中便于处理，用一个 case 就可以处理所有 s ≥ 3000 的情况。

随堂练习

编程：从键盘上输入一百分制成绩，要求输出成绩相应的等级 A、B、C、D 和 E。其中 90 分以上为 A，80 ~ 89 分为 B，70 ~ 79 分为 C，60 ~ 69 分为 D，60 分以下为 E。

3.5 循环结构程序设计

如果要计算“1+2+3+…+100”的值，根据现有的知识，只能一项一项地求和，这是一件非常烦琐的事情，如果数据更大点，就更加烦琐，甚至根本无法完成。要解决这类问题，就需要用到循环。循环结构是结构化程序设计的基本结构之一，它和顺序结构、选择结构共同作为各种复杂程序的基本构造单元。C 语言中提供的循环语句有 while、do…while 以及 for 语句。

3.5.1 从 while 语句学自增和自减运算符

程序 3_9.c 使用 while 循环来求“1+2+3+…+100”的值。

程序清单 3.9　3_9.c 程序

```
#include <stdio.h>
void main()
{
    int i = 1, sum = 0;
    while(i <= 100)
    {
        sum += i;
        i++;
    }
}
```

sum 的初始值一定要赋 0 值

while 循环语句，循环体为“sum+=i;i++;”。其中“i++”等价于“i= i+1”

```
    printf("1+ 2 + 3 + … + 100 = %d\n", sum);
    }
```

程序 3_9.c 的流程图如图 3.16 所示，运行结果如图 3.17 所示。

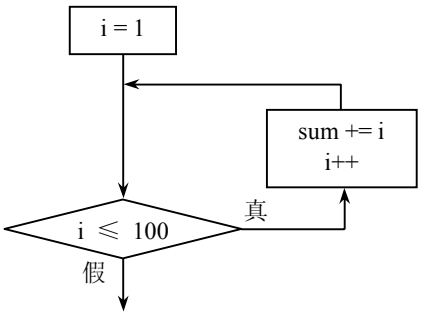


图 3.16　程序 3_9.c 的流程图

图 3.17　程序 3_9.c 的运行结果

while 语句用来实现“当型”循环结构，其一般形式如下：

```
while(表达式)
{
    语句组;
}
```

当语句组只有一条语句的时候可以省略大括号。在语句组中必须包含能改变表达式的值的语句来使表达式的值最终变为假，否则循环永远不会中止。例如：

```
index = 1;
while(index < 5)
{
    printf("Hello,C Program!\n");
}
```

上面的代码段就会陷入“死循环”，程序会无休止地输出“Hello，C Program!”。因为变量 index 的值为 1，如果 index 的值不发生变化，index < 5 这个条件将永远成立，while 语句也就会一直循环下去。所以在循环体中应该添加能够使循环正常结束的语句。如程序 3_10.c 所示。

程序清单 3.10　3_10.c 程序

```
#include <stdio.h>
void main()
{
    int index = 1;
    while(index < 5)
    {
        printf("index = %d\n", index);
        index++;
        printf("after index++ index = %d\n", index);
    }
    printf("The loop has finished.\n");
}
```

每次循环的时候，执行 index++，index 增加 1，当 index<5 不成立的时候就会退出循环

程序 3_10.c 的运行结果如图 3.18 所示。从图 3.18 可以看出在第 4 次循环中，执行完 index++ 后，index 的值已经是 5，然而循环此时并不结束，继续执行下面的输出语句 printf("after index++ index = %d\n", index)。执行完输出语句后，再去判断表达式 index < 5 是否成立，此时发现该表达式为假，才结束 while 循环，执行 while 后面的语句。

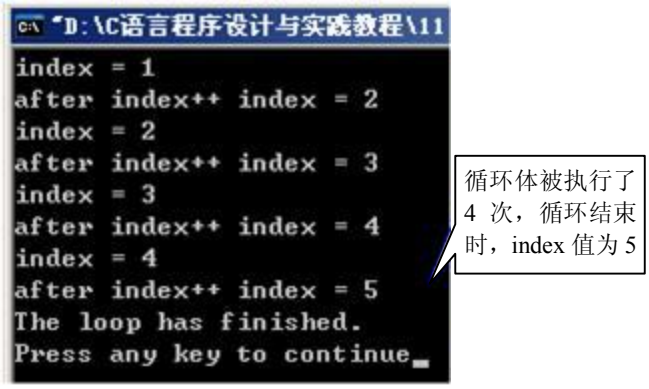


图 3.18　程序 3_10.c 的运行结果

如果 while 语句的表达式刚开始就不成立，如将程序清单 3.10 中的 index 的初始值改成 10，这时，程序永远不会执行循环体，因为条件 index < 5 一开始就为假。

1. 自增、自减运算符

在程序 3_10.c 中使用了自增运算符“++”，其作用是使变量的值增 1。与之对应的还有自减运算符“--”，其作用是使变量的值减 1。

自增、自减运算符的使用有两种方式：

前缀模式：++变量，--变量（在使用变量之前，先使变量的值加（减）1）。

后缀模式：变量++，变量--（在使用变量之后，使变量的值加（减）1）。

从表上看，“++变量”和“变量++”的作用都是使变量的值增 1，但是它们还是有区别的。“++变量”是先让变量增 1 后，再使用变量的值；而“变量++”则是先使用变量的值，变量再增 1。自减运算符“--”的原理一样。

程序清单 3.11　3_11.c 程序

```
#include <stdio.h>
void main()
{
    int a = 1, b = 1;
    int aplus, plusb;
    aplus = a++;
    plusb = ++b;
    printf("a = %d\naplus = %d\nb = %d\nplusb = %d\n", a, aplus, b, plusb);
}
```

程序 3_11.c 的运行结果如图 3.19 所示。从图 3.19 可以看出，执行“aplus = a++”是先将 a 的值 1 复制给 aplus，aplus 的值为 1，然后 a 的值再增 1 变成 2。而“plusb = ++b”则是先使 b 的值增 1 变成 2，然后再将 b 的值赋值给 plusb，plusb 的值为 2。

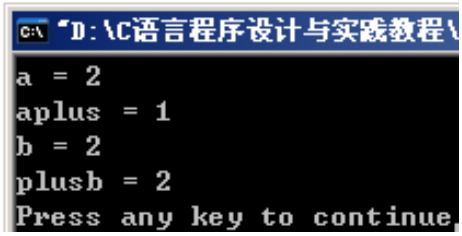


图 3.19　程序 3_11.c 的运行结果

自增和自减运算符有很高的优先级，常用的运算符只有圆括号比它们的优先级高。其结合性是“自右向左”。所以“x * y++”代表“x * (y++)”而不是“(x * y)++”。而后者也不是一个合法的表达式，自增、自减运算符只能用于变量，而不能用于常量或表达式。“5++”或“(x * y)++”都是不合法的。

2. 含有自增、自减运算符的表达式的不确定性

C 语言提供的运算符和表达式使用非常灵活，这种灵活性往往也会带来麻烦。由于 ANSI C 并未规定表达式中的子表达式的求解顺序，由各个编译系统自己决定，则表达式：

```
a = fun1() + fun2()
```

并非所有的编译系统都是先求解 fun1()，然后再求解 fun2()。在一般的情况下，可能先求解 fun1()或者先求解 fun2()的结果是一样的，但是在很多时候也是不一样的，如：

```
n = 2;
a = fun1(n) + fun2(n++);
```

如果先求解函数 fun1，则 a = fun1(2) + fun2(2)，如果先求解 fun2，则 a = fun1(3)+fun2(2)。

又如：

```
n = 2;
y = n++ + n++;
```

当该语句被执行后，n 的值变成 4，但是 y 的值是不确定的。有的编译器可能将 n = 2 作为表达式中所有 n 的值，从而求出 y 为 4。但也有的编译器按照自左向右的顺序来求解表达式，先求解第一个 n++，这个时候 n++先使用 n 的值 2，然后 n 再增 1，变成 3。在求解第二个 n++的时候，同理，先使用 n 的值 3，然后 n 再变成 4。此时 y 值为 5。

对于这类问题，在 C 语言中还有一些，例如：

```
n = 2;
printf("%d, %d\n", n, n++);
```

在调用函数时，C 语言并未规定对实参的求解顺序，对于上例，有的系统从左到右求解，输出“2, 2”，而多数系统则是从右向左进行求解，输出“3, 2”。

对于上述的一些不确定性，我们在写程序的时候一定要避免，不要写出别人看不懂，也不知道系统会怎样执行的代码。在使用自增或自减运算符的时候，要遵循以下两个原则：

（1）如果一个变量出现在同一个函数的多个参数中时，不要用自增或自减运算符。

（2）如果一个变量多次出现在一个表达式里时，不要用自增或自减运算符。

3.5.2　do…while 语句

下面使用 do…while 语句来改写程序 3_9.c（求 1+2+3+…+100 的值）。程序清单如下：

程序清单 3.12　3_12.c 程序

```
#include <stdio.h>
void main()
{
    int i = 1, sum = 0;
    do
    {
        sum += i;
        i++;
    }while(i <= 100);
    printf("1 + 2 + 3 + ... + 100 = %d\n", sum);
}
```

分号“;”不能少

程序的运行结果与图 3.17 相同。其流程图如图 3.20 所示。从图 3.20 可以看出，程序先执行循环体语句，然后再判断 while 后面的条件是否成立。当表达式为真，则返回重新执行循环体语句，如此反复，直到 while 后面的表达式为假才结束循环。

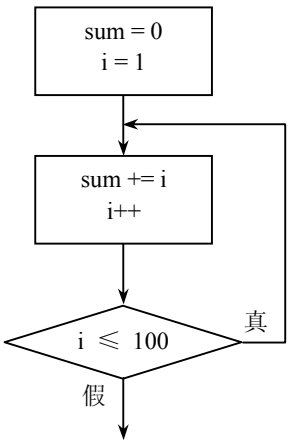


图 3.20　程序 3_12.c 的流程图

可以看到 do…while 语句的一般形式如下：

```
do
{
    循环体语句组;
}while(表达式);
```

注意分号“;”不能少

可以看到，对于同一个问题，可以使用 while 语句，也可以使用 do…while 语句，二者可以互相替换。但是它们还是有区别的，while 语句先判断条件是否成立，然后根据条件的结果来决定是否执行循环体，而 do…while 语句先执行循环体一次，然后再判断表达式成立与否。即 do…while 语句至少要执行循环体一次。例如：

程序清单 3.13　3_13.c 程序

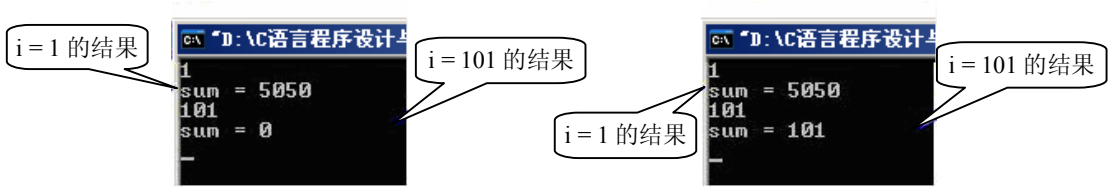
```
/*程序 A*/
#include <stdio.h>
void main()
```

```
/*程序 B*/
#include <stdio.h>
 void main()
```

```
{
    int sum = 0, i;
    scanf("%d", &i);
    while(i <= 100)
    {
        sum += i;
        i++;
    }
    printf("sum = %d\n", sum);
}
```

```
{
    int sum = 0, i;
    scanf("%d", &i);
    do
    {
        sum += i;
        i++;
    }while(i <= 100);
    printf("sum = %d\n", sum);
}
```

程序的运行结果如图 3.21 所示。可以看出，当输入的 i 值都满足 i≤100 的时候，程序 A 和程序 B 的运行结果相同。而当输入的 i 值不满足 i≤100 的时候，二者的结果就不同了，因为此时对于 while 语句来说，循环体一次也不会被执行，而 do…while 语句则会执行循环体一次。



（a）程序 A 的运行结构　　（b）程序 B 的运行结构

图 3.21　程序 3_13.c 的运行结果

3.5.3　灵活强大的循环语句——for 语句

C 语言中的 for 语句是使用最为灵活的语句。同样，先使用 for 语句来改写程序 3_9.c 求 1+2+3+…+100 的值。程序清单如下：

程序清单 3.14　3_14.c 程序

```
#include <stdio.h>
void main()
{
    int i, sum = 0;
    for(i = 0; i <= 100; i++)
    {
        sum += i;
    }
    printf("1 + 2 + 3 + ... + 100 = %d\n", sum);
}
```

for 语句，注意括号中的三个表达式是用“;”分隔的

循环体语句

程序的运行结果与图 3.17 相同。可以看出，for 语句的一般形式如下：

```
for(表达式 1;表达式 2;表达式 3)
{
    循环体语句组;
}
```

当循环体语句组只有一条语句的时候，大括号可以省略。建议初学者不要省略，以避免出错。for 语句的流程如图 3.22 所示。从图中可以看出，其执行过程如下：

（1）求解表达式 1；

（2）求解表达式 2，如果为真，则执行循环体语句组，执行结束后转步骤（3）；如果表达式的结果为假，则结束循环，转步骤（5）；

（3）求解表达式 3；

（4）转步骤（2）；

（5）循环结束，执行循环语句之后的其他语句。

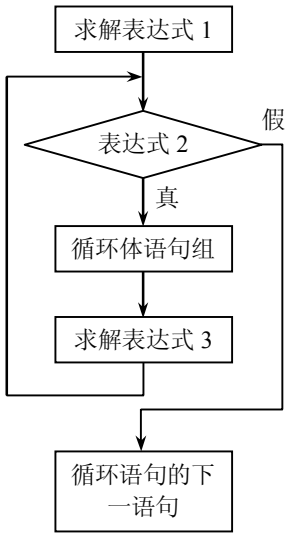


图 3.22　for 语句执行流程

for 语句的 3 个表达式均可省略，但是里面的分号“;”不能省略。需要注意的是，省略相应的表达式，就需要在其他地方弥补相应表达式的功能，以保证程序的正常运行。

3.5.4　逗号运算符和逗号表达式

逗号运算符扩展了 for 循环语句的灵活性，它可以使 for 循环语句使用多个初始化或循环变量更新表达式。例如：

程序清单 3.15　3_15.c 程序

```
#include <stdio.h>
void main()
{
    int i, sum;
    for(i = 1, sum = 0; i <= 100; sum += i,i++);
    printf("sum = %d\n", sum);
}
```

逗号表达式

有“;”，表示 for 语句没有循环体语句。

程序 3_15.c 也是求 1+2+3+…+100 的值，其运行结果如图 3.23 所示。可以看到，在 for 语句中使用了逗号表达式，使得 for 语句非常简练。

图 3.23　程序 3_15.c 的运行结果

从程序 3_15.c 可以看出，用逗号运算符可以将两个或两个以上的表达式连接起来，称为逗号表达式，其一般形式如下：

表达式 1,表达式 2,…,表达式 n

逗号表达式的求解过程为：按照先后顺序，依次求解表达式 1，表达式 2，……，最后求解表达式 n。并将最后求解的表达式 n 的值作为整个逗号表达式的值。如程序 3_15.c 中的：

i = 1, sum = 0;

先求解赋值表达式 i = 1，然后再求解表达式 sum = 0，并将表达式 sum = 0 的值作为整个逗号表达式的值，所以该逗号表达式的值为 0。

从附录三可知，逗号运算符是所有运算符中级别最低的，所以“a = 3 * 5, a * 2”应理解成“(a = 3 * 5), (a * 2)”而不是“a = (3 * 5, a * 2)”。在逗号表达式的子表达式可以是任意的表达式，包括逗号表达式。如(a = 3 * 5, a * 2), a–4。

逗号运算符并不只限于在 for 语句中使用。同时，需要引起注意的是，在定义时变量之间的分隔符，以及在 printf()函数和 scanf()函数中一些分隔符不属于逗号运算符。如下面语句中的“,”都不是逗号运算符。

```
int i, j, k;
printf("%d,%d,%d\n", i, j, k);
```

小提示：几种循环语句的比较

（1）while 语句、do…while 语句和 for 语句都可以用来处理同一问题，很多时候，它们之间可以互换。

（2）在 while 和 do…while 语句中，只是在 while 后的括号里指定了循环条件，应该在循环体内包含使循环能够正常结束的语句，避免出现死循环。而 for 语句在表达式 3 中含有使循环趋于结束的操作，甚至可以将循环体中的操作全部放到表达式 3 中，因此 for 语句使用更为灵活，功能更为强大。

（3）用 while 和 do…while 语句循环时，循环变量的赋初值应该在 while 和 do…while 语句之前完成，而 for 语句则可以在表达式 1 中完成。

3.5.5　循环结构程序举例

编写程序：输入一正整数 m，判断 m 是否是素数（只能被 1 和它本身整除的数），并输出结果。

具体的实现方法：将 2～$\sqrt{m}$ 范围内所有的数去除 m，如果 m 能够被其中任意一个数整除，则此时 m 必然不是素数，结束循环。如果在 2～$\sqrt{m}$ 范围内没有一个数能整除 m，则 m 一定是素数。程序如下：

程序清单 3.16　3_16.c 程序

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

```
#include <math.h>
void main()
{
    int i, m, k;
    printf("Please input a number: ");          /*在屏幕上提示用户输入数据*/
    scanf("%d", &m);
    k = (int)sqrt(m);                           /*将函数 sqrt 的返回值强制转化成 int 型*/
    for(i = 2; i <= k; i++)
    {
        if(m % i == 0)
        {
            break;
        }
    }
    if(i > k)
    {
        printf("%d is a prime number.\n", m);
    }
    else
    {
        printf("%d is not a prime number.\n", m);
    }
}
```

如果“m % i == 0”成立，则表明有一个 i 能整除 m。m 不是素数，后面的没有必要再判断，直接结束循环

“i > k”表明在 for 语句中没有执行“break”语句，即没有一个 i 能整除 m，m 是素数

程序 3_16.c 的运行结果如图 3.24 所示。

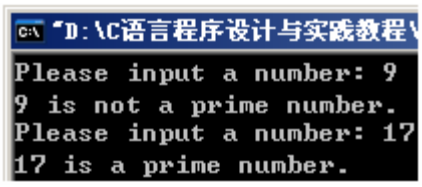


图 3.24　程序 3_16.c 运行结果

在 switch 语句中，已经接触到 break 语句，其作用是使程序流程跳出 switch 语句，继续执行 switch 语句后的其他语句。在程序 3_16.c 中，break 语句的作用是程序流程从循环中跳出，即提前结束循环，接着执行循环语句后面的语句。

break 语句不能使用在循环语句和 switch 语句之外的其他语句中。

例：编写程序把 100 以内不能被 3 整除的正整数输出。

程序清单 3.17　3_17.c 程序

```
#include <stdio.h>
void main()
{
    int i, count = 0;
    for(i = 1; i <= 100; i++)
    {
```

```
            if(i % 3 == 0)
            {
                continue;
            }
            count++;                    /*对输出的数据个数进行计数*/
            printf("%5d", i);
            if(count % 10 == 0)         /*使输出结果以每行 10 个数据输出*/
            {
                printf("\n");
            }
        }
    }
```

continue 语句：结束本次循环，接着进行下一次是否循环的条件判定

程序 3_17.c 的运行结果如图 3.25 所示。程序中的 continue 语句的作用是：结束本次循环，即跳过循环体中其他尚未执行的语句，接着判定是否进行下一次循环。在程序中如果条件“i % 3 == 0”为真，则 continue 语句将会使程序流程转向 for 语句的表达式 3，而忽略后面尚未执行完的循环体。

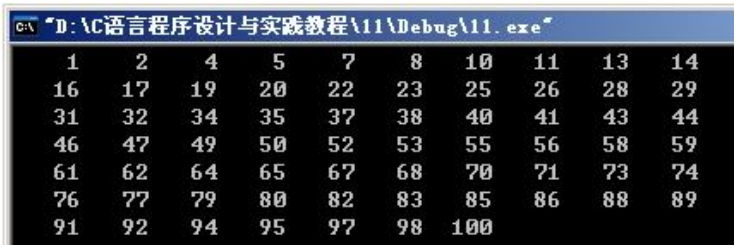


图 3.25　程序 3_17.c 运行结果

当然在程序 3_17.c 中也可以直接用一个 if 语句来处理不能被 3 整除的数：

```
if(i % 3 != 0)
{
    printf("%5d", i);
}
```

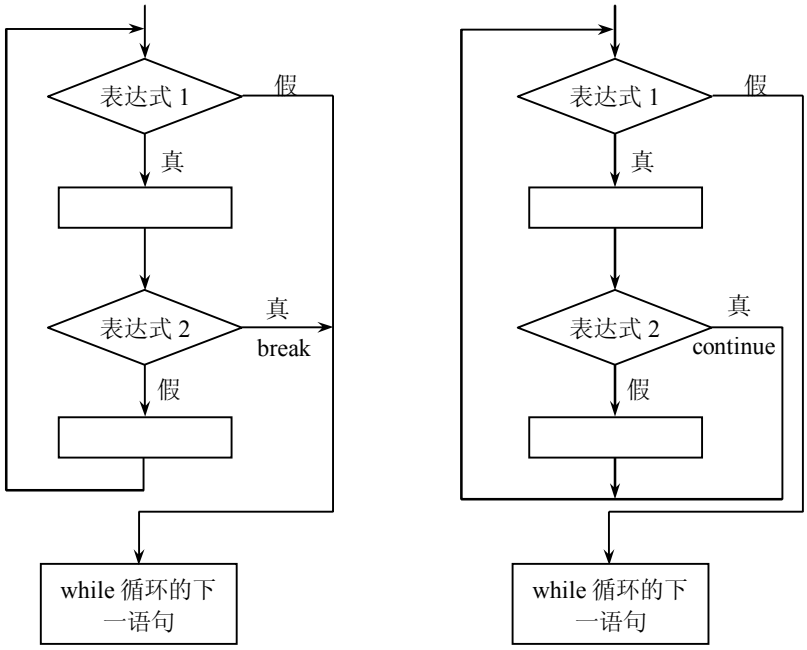
小提示：continue 语句和 break 语句的区别

continue 语句只是结束本次循环，而不是终止循环。而 break 语句则是结束整个循环，不再判断循环的条件是否成立。如有以下两个循环结构：

```
（1）while（表达式 1）
     {
       …
       if（表达式 2）break;
       …
     }
```

```
（2）while（表达式 1）
     {
       …
       if（表达式 2）continue;
       …
     }
```

程序结构（1）的流程图如图 3.26（a）所示，可以看到当表达式 2 为真的时候，结束循环，执行循环语句后的语句。程序结构（2）的流程图如图 3.26（b）所示，当表达式 2 为真的时候，忽略循环体其他尚未执行的语句，接着判断表达式 1 的真假。



（a）break 语句示例　　（b）continue 语句示便

图 3.26　break 语句和 continue 语句的区别

例：输出 100～200 间所有的素数，每行输出 10 个数据。

从程序 3_16.c 中，已经了解怎样来判断给定的数 m 是否是素数，要找出 100～200 间所有的素数，只需要将这个范围内所有的整数一一进行判断，如果是素数，则输出。据此，程序清单如下：

程序清单 3.18　3_18.c 程序

```
#include <stdio.h>
#include <math.h>
void main()
{
    int m, k, i, count = 0;
    for(m = 101; m <= 200; m += 2)
    {
        k = (int)sqrt(m);
        for(i = 2; i <= k; i++)
        {
            if(m % i == 0)
            {
                break;
            }
        }
        if(i > k)
        {
            printf("%5d", m);
            count++;
```

外循环，控制 m 的范围。这里 m 的增量为 2，对偶数不进行判断

内循环，判断 m 是否是素数

break 语句和 continue 语句只影响包含它的最里层循环

```
            }
            if(count % 10 == 0)          /*输出的数据满 10 个则进行换行处理*/
            {
                printf("\n");
            }
        }
        printf("\n");
}
```

程序的运行结果如图 3.27 所示。程序 3_18.c 中在一个循环体内又包含另一个完整的循环结构，称为循环嵌套。内嵌的循环中还可以嵌套循环。3 种循环语句可以相互嵌套，理论上，嵌套的层数没有限制，但是嵌套的层数太多，会降低程序的可读性。

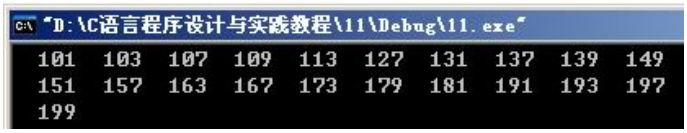


图 3.27　程序 3_18.c 的运行结果

在循环嵌套中的 break 语句和 continue 语句只能影响包含它的最里层循环，对外层循环没有影响，如程序 3_16.c 中，当 m % i == 0 条件成立时，将执行 break 语句结束循环，此时结束的是内循环，即结束判断 m 是否是素数的循环，而对外循环没有影响。

C 语言中的语句（二）

C 语言提供的控制语句，用于完成一定的控制功能。

（1）if（表达式）…else…：条件语句，当表达式为真，则执行 if 后面的语句组，否则执行 else 后面的语句组。

（2）for（表达式 1；表达式 2；表达式 3）…：for 循环语句。表达式 1 只在循环语句执行前执行一次，然后判断表达式 2 的真假，如果为真，循环体就被执行一次，然后执行表达式 3（一般为循环变量更新），接着再次判断表达式 2 的真假，如此反复，直到表达式 2 为假的时候，才结束循环。

（3）while（表达式）…：while 循环语句。如果表达式为真，则执行循环体，直到表达式为假的时候结束循环。

（4）do…while（表达式）：do…while 循环语句，先执行循环体，然后判断表达式是否成立，如果成立则继续执行循环体，直到表达式为假时结束循环。

（5）continue：结束本次循环，忽略循环体中尚未执行的语句，接着判断循环条件来决定是否进行下一次循环。

（6）break：终止执行 switch 语句或者循环语句。

（7）switch（表达式）…case…default…：多分支选择语句。程序控制按照表达式的值跳转到相应的 case 标签处，从此开始执行，直到 switch 语句结束或者遇到 break 语句结束 switch 语句的执行。如果没有与表达式相匹配的 case 标签，则执行 default 的语句。

（8）return：从函数返回的语句。

（9）goto：程序控制流程跳转语句，在结构化程序设计中不提倡使用 goto 语句。

随堂练习

输出所有的“水仙花数”。所谓“水仙花数”是指一个 3 位数，其各位数字的立方和等于该数本身，例如，153 是一个水仙花数，因为 $153=1^3+5^3+3^3$。

3.6 小结

本章主要介绍了 C 语言中的流程控制语句。在程序设计的过程中，任意复杂的程序结构都可以分解成顺序、选择和循环三种基本结构。

顺序结构的程序是最简单的 C 程序，按照编写代码的先后顺序依次执行。通过顺序结构的程序介绍了 C 语言中算术运算符和算术表达式，赋值运算符和赋值表达式。

选择结构的程序可以处理一些分支的情况。C 语言中提供 if（if…else…）语句、switch 语句用来实现选择结构的程序。在构造某一个条件的时候，通常是关系表达式，即用关系运算符构成的表达式。为了构造比较复杂的条件，可以使用逻辑运算符将多个表达式组合起来。同时，C 语言中提供的条件运算符也可以处理一些简单的分支情况。

循环结构的程序可以处理有时候需要反复执行某一个程序段的情况。C 语言中提供的循环语句有 while、do…while 和 for 三种语句。同时，通过循环语句还介绍了使用自增、自减运算符来使程序更加简练。

break、continue 和 goto 是跳转语句，使程序流程跳转到程序的其他位置。break 语句使程序流程跳转到紧跟在包含它的循环或 switch 末尾的下一条语句。continue 语句使程序流程跳过包含它的循环语句剩余部分，开始下一循环周期。goto 语句导致程序流程跳转到指定标签定位的语句。冒号用来将被标记语句同它的标签分隔。标签名遵循变量名命名规则，被标记语句可以出现在 goto 之前或之后。如：

```
goto label;
…
label：语句
```

3.7 习题

1．输入一个实数 x，计算并输出下式的值，直到最后一项的绝对值小于 10^{-5}（保留两位小数）。

$$s=x+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!}+\cdots$$

2．输入圆的半径 r，求该圆的周长、面积。结果保留 2 位小数。

3．写程序，输入一个 4 位的年份数据，判断是否是闰年。所谓闰年是指符合下面两个条件的任意一个的年份。

（1）能被 4 整除，但不能被 100 整除；

（2）能被 400 整除。

4．求 Fibonacci 数列的前 40 个数。该数列的通项式如下：

$$\begin{cases} F_1 = 1 & (n = 1) \\ F_2 = 2 & (n = 2) \\ F_n = F_{n-1} + F_{n-2} & (n \geqslant 3) \end{cases}$$

同时，这也是一个古老而有趣的古典数学问题：有一对兔子，从出生后第三个月起每个月都生一对兔子。小兔子长到第 3 个月又生一对兔子。假设所有的兔子都不死，问每个月兔子的总数是多少？

5．有一分数序列：

$$\frac{2}{1}, \frac{3}{2}, \frac{5}{3}, \frac{8}{5}, \frac{13}{8}, \frac{21}{13}, \cdots$$

求出这个数列前 20 项之和。

6．猴子吃桃问题。猴子第一天摘下若干桃子，当即吃了一半，还不过瘾，又多吃了一个。第二天早上将剩下的桃子吃掉一半，又多吃一个。以后每天早上都吃了前一天剩下的一半零一个。到第 10 天早上再想吃时，发现只剩一个桃子了。编程求第一天共摘了多少桃子。

7．百钱买百鸡问题：鸡翁一值钱五，鸡母一值钱三，鸡雏三值钱一。凡百钱买百鸡，编程求鸡翁、鸡母、鸡雏各几何。

8．输入两个正整数 m 和 n，编程求其最大公约数和最小公倍数。

提示：求最大公约数的算法如下：

（1）将两个数中较大的放在变量 m 中，较小的放在 n 中；

（2）求出 m 被 n 除后的余数；

（3）若余数为 0，转步骤（7），否则转步骤（4）；

（4）把除数作为新的被除数，余数作为新的除数；

（5）求出新的余数；

（6）重复步骤（3）～（5）；

（7）输出 n。n 即为最大公约数。

最小公倍数=(m×n)÷最大公约数。

9．一个数如果恰好等于它的因子之和，这个数就称为“完数”。例如 6 的因子为 1、2、3，而 6=1+2+3，因此 6 是“完数”。编程序找出 1000 之内的所有完数，并按下面的格式输出其因子：

6 its factors are 1，2，3