CHAPTER 3

第3章 控制结构

C 语言的控制结构只有 3 类：顺序结构、选择结构和循环结构。顺序结构是最简单的程序结构。在顺序结构中，程序从前到后按顺序依次执行各条语句或语句块，不跳过也不重复任何语句或语句块。在 C 语言当中，不含语句块的每条语句通常以分号“;”作为结束标志。最简单的语句可以只包含一个分号，该语句称为空语句。空语句不执行任何操作。有时采用空语句来延长程序的运行时间。被大括号“{ }”括起来的一条或多条语句通常称做语句块。下面给出顺序结构代码示例。

```
double a = 5.1;
double b = 7.2;
double c = (a+b)/2;
printf("%g 和%g 的平均值是%g。\n", a, b, c);
//输出: 5.1 和 7.2 的平均值是 6.15。↙
```

本章将依次介绍选择结构和循环结构。在选择结构中，程序依据条件选择相应的分支执行语句或语句块。选择结构包括 if 语句、if-else 语句和 switch 语句，这些语句也可以统称为选择语句。在循环结构中，程序不断重复执行指定的语句或语句块，直到循环结束。循环结构包括 for 语句、while 语句和 do-while 语句。组成循环结构的 for 语句、while 语句和 do-while 语句也可以统称为循环语句。选择语句和循环语句实际上都是复合语句，即在这些语句的组成部分当中还会包含语句、语句块或语句组，其中语句组也是由一条或多条语句组成。语句块与语句组的区别在于是否被一对大括号“{ }”括起来。在选择结构和循环结构中，还可能包含 break 语句和 continue 语句，其中 break 语句用来中断执行 switch 语句或循环结构，continue 语句只能用于循环结构并使得程序直接进入下一轮的循环。

3.1 选择结构

选择结构的特点是根据不同的条件选择执行不同的程序代码。选择结构包

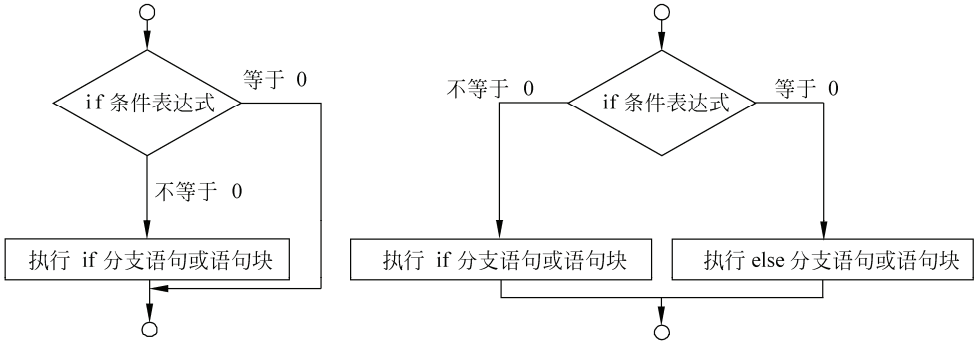
括 if 语句、if-else 语句和 switch 语句。其中 if 语句只有一个分支，当且仅当在满足 if 条件时，才执行在 if 语句中的语句或语句块。if-else 语句包含两个分支。如果满足 if 条件，则执行 if 分支的语句或语句块；否则，执行 else 分支的语句或语句块。switch 语句可以包含多个分支。switch 语句根据一个整数系列类型表达式的值，选择执行相应的分支。

3.1.1　if 语句和 if-else 语句

if 语句和 if-else 语句统称为**条件语句**。**if 语句的格式**是:

```
if (表达式)
    语句或语句块
```

其中，表达式必须是数值类型的表达式，可以是定点数，也可以是浮点数，称为**if 条件表达式**。在 if 语句当中的语句或语句块只能是单条语句或一个语句块，称为**if 分支语句或语句块**。如图 3-1（a）所示，只有当 if 条件表达式不等于 0 时，才会执行 if 分支语句或语句块。



（a）if 语句流程图　　（b）if-else 语句流程图

图 3-1　if 语句和 if-else 语句流程图

在流程图中，小圆圈表示流程图片断的连接点，即上面流程图连接入其他流程图组成更大流程图的连接点；菱形表示条件判断，而且将依据判断结果执行不同的分支；矩形表示正常的代码执行；箭头表示程序运行的路径。

下面给出 if 语句代码示例。

```
int studentScore = 95;                                          // 1
if (studentScore>90)                                            // 2
   printf("成绩优秀!\n"); // 结果输出: 成绩优秀!↙                  // 3
```

在上面第 2 行代码中，因为 studentScore = 95，所以 if 条件表达式“studentScore>90”成立。这样，上面第 3 行代码 if 分支语句“printf("成绩优秀!\n");”就会被执行，结果输出“成绩优秀!↙”。如果将上面第 1 行代码换为“int studentScore = 85;”，那么 if 条件表达式“studentScore>90”不成立，这样上面第 3 行代码就不会被执行，结果什么也没有被输出。

下面给出另外一个 if 语句代码示例。

```
int a = 10;                                                    // 1
int b = 5;                                                     // 2
if (a>b)                                                       // 3
{                                                              // 4
   printf("a=%d\n", a);       // 结果输出: a=10↙              // 5
   printf("b=%d\n", b);       // 结果输出: b=5↙               // 6
   printf("a 比 b 大。\n");// 结果输出: a 比 b 大。↙          // 7
} // if 结束                                                   // 8
```

在这个示例当中，if 分支部分是一个语句块。通过语句块使得 if 分支部分可以包含多条语句。

注意事项：在 if 分支语句块当中，作为语句块标志的一对大括号“{}”是不能去掉的。如果去掉，则 if 分支语句块就变为 if 分支语句，而且这条 if 分支语句就是原语句块的第一条语句。

例如，在上面的代码示例中，如果去掉其中第 4 行和第 8 行代码，则在第 3 行之后的代码变为

```
if (a>b)
   printf("a=%d\n", a);       // 只有当(a>b)，才会输出 a 的值
   printf("b=%d\n", b);       // 无论 a 是否大于 b，均会输出 b 的值
   printf("a 比 b 大。\n");   // 无论 a 是否大于 b，均会输出: a 比 b 大。↙
```

这时，最后两行代码“printf("b=%d\n", b);”和“printf("a 比 b 大。\n");”并不隶属于 if 语句。因此，无论 a 是否大于 b，均会输出 b 的值以及“a 比 b 大。↙”。这显然是有问题的。

if-else 语句包含两个分支。**if-else 语句的格式**是:

```
if (表达式)
   语句 1 或语句块 1
else
   语句 2 或语句块 2
```

其中，表达式称为 **if 条件表达式**，必须是数值类型的表达式，可以是定点数，也可以是浮点数。if 下方的语句 1 或语句块 1 称为 **if 分支语句或语句块**，else 下方的语句 2 或语句块 2 称为 **else 分支语句或语句块**。这里的语句 1 和语句 2 均只能是单条语句，语句块 1 和语句块 2 也只能是一个语句块。如图 3-1（b）所示，只有当 if 条件表达式不等于 0 时，才会执行 if 分支语句或语句块；否则，执行 else 分支语句或语句块。下面给出 if-else 语句代码示例。

```
int studentScore = 85;                                         // 1
if (studentScore>=60)                                          // 2
   printf("通过考试!\n");                                      // 3
else                                                           // 4
   printf("考试没通过，请继续努力!\n");                        // 5
```

上面代码示例将输出“通过考试!↙”。

在 if 语句和 if-else 语句当中的分支语句仍然可以是 if 语句或 if-else 语句。但这时，需要注意 if 和 else 在同一个语句块中的最近配对原则。

注意事项：**if 和 else 的最近配对原则**：在同一个语句块中，else 部分总是按照 if-else 语句格式与最近的未配对的 if 部分配对，构成 if-else 语句。如果 else 部分无法与 if 部分配对构成符合 if-else 语句格式的语句，那么将出现编译错误。根据这一原则，如果在 if-else 语句当中的 if 分支语句或语句块仍然是一条 if 语句，那么该 if 语句代码的编写应当采用语句块的形式；否则，该 if 语句将与 else 部分配对成为 if-else 语句。下面给出具体的示例代码进行说明。

这里给出示例代码，说明 if 和 else 最近配对原则可能出现的问题，以及如何避免可能出现的错误。具体的代码如下。

```
int month = 12;                                          // 1
int day = 30;                                            // 2
if (month==12)                                           // 3
{                                                        // 4
   if (day==31)                                          // 5
      printf("这是一年的最后一天!\n");                    // 6
}                                                        // 7
else                                                     // 8
   printf("这不是一年的最后一个月!\n");                   // 9
```

虽然第 5 行的 if 比第 3 行的 if 离第 8 行的 else 更近一些，但第 5 行的 if 与第 8 行的 else 不在同一个语句块中。因此，上面代码第 8 行的 else 只会与第 3 行的 if 相配对，而不会与第 5 行的 if 相配对。上面代码第 4~7 行是一个语句块，只包含一条 if 语句。在 if-else 语句格式中，允许其中的 if 分支语句或语句块是一条语句。那么，能否去掉第 4 行和第 7 行代码，即去掉作为语句块标志的一对大括号“{ }”？在去掉这两行代码之后，语法仍然是正确的。这时，上面的代码变为

```
int month = 12;                                          // 1
int day = 30;                                            // 2
if (month==12)                                           // 3
   if (day==31)                                          // 5
      printf("这是一年的最后一天!\n");                    // 6
else                                                     // 8
   printf("这不是一年的最后一个月!\n");                   // 9
```

运行上面的代码，将会输出“这不是一年的最后一个月!↙”。为什么会这样？为什么 12 月份会不是一年的最后一个月？我们分析一下修改之后的代码。根据 if 和 else 最近配对原则，在修改之后的代码中，上面代码第 8 行的 else 会与第 5 行的 if 相配对。这样，修改之后的代码实际上等价于

```
int month = 12;                                          // 1
```

```
int day = 30;                                                    // 2
if (month==12)                                                   // 3
{                                                                // 4
    if (day==31)                                                 // 5
        printf("这是一年的最后一天!\n");                          // 6
    else                                                         // 7
        printf("这不是一年的最后一个月!\n");                      // 8
}                                                                // 9
```

这样，对于 12 月份，只要日期不是 31，就会输出“这不是一年的最后一个月!↙”。对于其他月份，反而什么都不会输出。将“if (day==31) printf("这是一年的最后一天!\n");”部分按语句块的形式编写，就不会出现这种与预期不相符的逻辑。总之，在 if-else 语句当中的 if 分支语句或语句块仍然是一条 if 语句的正确语句格式是：

```
if (表达式 1)
{
    if (表达式 2)
        语句 1 或语句块 1
}
else
    语句 2 或语句块 2
```

如果 if-else 语句的 else 分支语句仍然是一条 if-else 语句，则通常写成

```
if (表达式 1)
    语句 1 或语句块 1
else if (表达式 2)
    语句 2 或语句块 2
else
    语句 3 或语句块 3
```

这个过程可以有限次重复下去，形成如下的语句格式：

```
if (表达式 1)
    语句 1 或语句块 1
else if (表达式 2)
    语句 2 或语句块 2
……
else if (表达式 n)
    语句 n 或语句块 n
else
    语句(n+1)或语句块(n+1)
```

下面给出代码示例。

```
int studentScore = 85;                                           // 1
if (studentScore>=90)                                            // 2
```

```
    printf("成绩优秀!\n");                                    // 3
else if (studentScore>=80)                                    // 4
    printf("成绩良好!\n");                                    // 5
else if (studentScore>=70)                                    // 6
    printf("成绩中等!\n");                                    // 7
else if (studentScore>=60)                                    // 8
    printf("成绩合格!\n");                                    // 9
else                                                          // 10
    printf("考试没通过，请继续努力!\n");                      // 11
```

上面的代码对成绩进行分类。如果分数大于或等于 90 分，则成绩优秀；如果分数介于 80 和 89 之间，则成绩良好；如果分数介于 70 和 79 之间，则成绩中等；如果分数介于 60 和 69 之间，则成绩合格；如果分数低于 60，则考试没通过，需要继续努力。

3.1.2　switch 语句

switch 语句也常称为**分支语句**。switch 语句的格式如下。

```
switch (表达式)
{
case 常数1:
    语句组1
case 常数2:
    语句组2
……
case 常数n:
    语句组n
default:
    语句组(n+1)
}
```

上面 switch 语句第一行的表达式称为 **switch 表达式**，它必须是整数系列类型的表达式。在 switch 语句的一对大括号“{ }”内是一系列的 case 分支和一个 default 分支。每个 **case 分支**在关键字 case 和空格之后紧接着一个常数，这个常数的数据类型必须与 switch 表达式相匹配，称为 **case 常数**。在同一条 switch 语句当中，各个 case 常数必须各不相等；否则，无法通过编译。在 case 常数之后是冒号，然后是 **case 分支语句组**。**default 分支**在同一条 switch 语句中最多出现一次，也可以不出现。在 default 分支中的语句组称为 **default 分支语句组**。case 分支语句组和 default 分支语句组均由一条或多条语句组成，而且每个 case 分支语句组和 default 分支语句的最后一条语句通常是 **break 语句**。如果在这些分支语句组的中间出现 break 语句，那么在 break 语句之后的语句实际上将不会起作用。当然，这些分支语句组也可以不含 break 语句。

如图 3-2 所示，在执行 switch 语句时，首先计算 switch 表达式的值，然后依次将该表达式的值与各个 case 常数进行匹配。如果该表达式的值刚好等于某个 case 常数，则进入该 case 分支，执行相应的 case 分支语句组。如果该 case 分支语句组不含 break 语句，则会继

续执行下一个 case 分支或 default 分支的语句组，直到执行到 break 语句或整个 switch 语句结束。如果 switch 表达式的值与任何一个 case 常数都不相等，并且 switch 语句含有 default 分支，则执行 default 分支语句组。

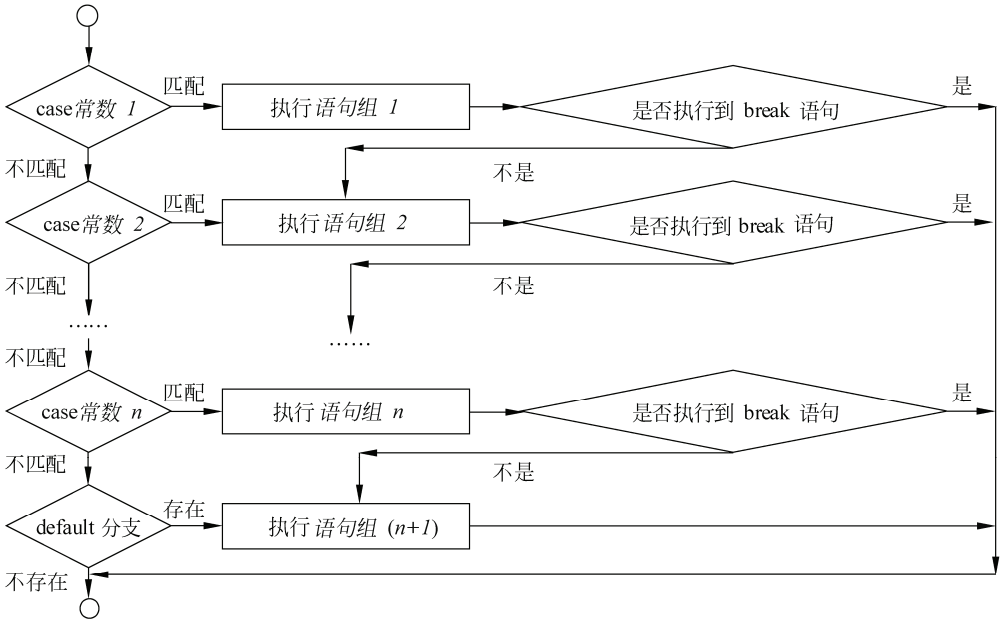


图 3-2　switch 语句流程图

下面给出一个 switch 语句示例。

```
char grade = 'A';                                          // 1
switch(grade)                                              // 2
{                                                          // 3
case 'A':                                                  // 4
   printf("百分制成绩: 90~100。\n");                        // 5
   break;                                                  // 6
case 'B':                                                  // 7
   printf("百分制成绩: 80~89。\n");                         // 8
   break;                                                  // 9
case 'C':                                                  // 10
   printf("百分制成绩: 60~79。\n");                         // 11
   break;                                                  // 12
case 'D':                                                  // 13
   printf("百分制成绩: 0~59。\n");                          // 14
   break;                                                  // 15
default:                                                   // 16
   printf("无效成绩。\n");                                  // 17
} // switch 结束                                           // 18
```

在这个示例中，因为变量 grade 的值是'A'，所以程序会进入 case 'A'分支，执行该 case 分支语句组，输出“百分制成绩: 90～100。↙”。因为该 case 分支语句组的最后一条语句是 break 语句，如上面第 6 行代码所示，所以 switch 语句运行到这里就自动结束了。我们还可以修改上面第 1 行代码，改变 grade 的值，从而执行 switch 语句的不同 case 分支或执行 default 分支的语句组。

下面给出一个不含 break 语句的 switch 语句应用示例。

```
int month = 5;                                          // 1
int dayRemain = 0;                                      // 2
switch(month)                                           // 3
{                                                       // 4
case 1:                                                 // 5
   dayRemain += 31;                                     // 6
case 2:                                                 // 7
   dayRemain += 28;                                     // 8
case 3:                                                 // 9
   dayRemain += 31;                                     // 10
case 4:                                                 // 11
   dayRemain += 30;                                     // 12
case 5:                                                 // 13
   dayRemain += 31;                                     // 14
case 6:                                                 // 15
   dayRemain += 30;                                     // 16
case 7:                                                 // 17
   dayRemain += 31;                                     // 18
case 8:                                                 // 19
   dayRemain += 31;                                     // 20
case 9:                                                 // 21
   dayRemain += 30;                                     // 22
case 10:                                                // 23
   dayRemain += 31;                                     // 24
case 11:                                                // 25
   dayRemain += 30;                                     // 26
case 12:                                                // 27
   dayRemain += 31;                                     // 28
} // switch 结束                                        // 29
printf("距离年终还剩余%d 天。\n", dayRemain);           // 30
```

这个示例假设某一年 2 月份总共是 28 天，希望统计从 month 这个月开始到年终还剩余多少天。在这个示例中，因为变量 month 的值是 5，所以程序会进入 case 5 分支，执行该 case 分支语句组，统计 5 月份的天数，使得 dayRemain= 31。因为该 case 分支语句组不含 break 语句，所以程序会进入该 switch 语句的下一个 case 分支，即 case 6 分支，继续统计 6 月份的天数，使得 dayRemain=31+30=61。这个过程不断继续下去，直到最后一个 case 分支，即 case 12 分支，程序统计了最后一个月的天数，该 switch 语句才执行结束。这时，程序

已经统计了从 5 月份到 12 月份的总天数，得到 dayRemain=245。因此，运行上面第 30 行代码将输出“距离年终还剩余 245 天。”。

注意事项：（1）在 switch 语句中，**switch 表达式**必须是定点数类型的表达式。

（2）在 switch 语句中，**case 常数**必须是定点数类型的常数，不能是浮点数类型的常数。

（3）在同一条 switch 语句当中，**所允许的 case 分支的总个数**总是有限的。C 语言标准规定，在同一条 switch 语句中，case 分支的个数不能超过 1023。不过，实际所允许的 case 分支个数依赖于 C 语言支撑平台。

3.2 循环结构

循环结构非常适合发挥计算机的强大运算能力。循环结构的特点是不断重复执行位于在循环体内的程序代码，直到不满足循环条件。**有限性是正常计算机程序的基本特点**。因此，对于正常的循环结构，应当设计合理的**循环条件**使得循环最终能够在有限的步骤之后结束。循环结构包括 for 语句、while 语句和 do-while 语句。下面分别介绍这些循环语句。

3.2.1 for 语句

for 语句是 C 语言的 3 种循环语句之一。**for 语句的格式**是

```
for（初始化表达式；条件表达式；更新表达式）
    循环体
```

其中，**循环体**一般是一条语句或一个语句块。

如图 3-3 所示，在执行 for 语句时，**初始化表达式**只会被计算一次。初始化表达式通常用来初始化循环所需要的变量，因此通常由一个或多个赋值运算表达式组成。如果是多个赋值运算表达式，则采用逗号分隔开。

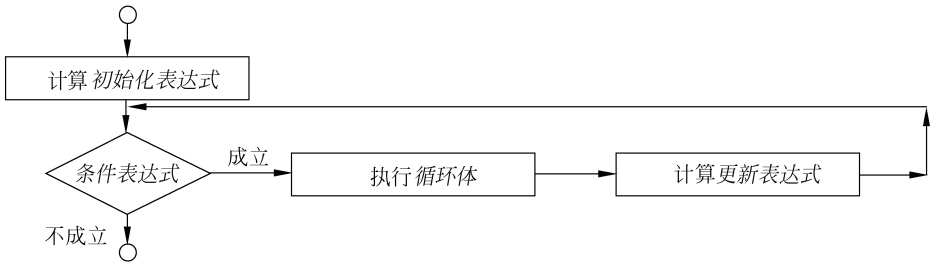


图 3-3　for 语句流程图

下面给出**采用 for 语句实现计算从 1 到 100 之和的示例**。

```
int i, n, sum;                                                    // 1
for (i=1, n=100, sum=0; i<=n; i++)                                // 2
    sum+=i;                                                       // 3
printf("sum=%d。\n", sum); // 结果输出: sum=5050。↙               // 4
```

在上面的示例中，初始化表达式是“i=1, n=100, sum=0”，由 3 个赋值运算表达式组成，相邻的赋值运算表达式采用逗号分隔。因此，这个初始化表达式实际上就是一个**逗号运算表达式**。

小甜点：**初始化表达式**还可以为空。下面给出相应的代码示例。

```
int n = 100;                                                    // 1
int sum = 0;                                                    // 2
int i = 1;                                                      // 3
for ( ; i<=n; i++)                                              // 4
    sum+=i;                                                     // 5
printf("sum=%d。\n", sum); // 结果输出：sum=5050。↙              // 6
```

在上面的示例中，初始化 for 循环所需要的各个变量的工作已经由第 1~3 行代码完成。因此，for 语句的初始化表达式为空。

注意事项：在 C 语言的 for 语句中，通常不允许在初始化表达式中定义变量。下面给出相应的代码示例。

```
int n = 100;                                                    // 1
int sum = 0;                                                    // 2
for (int i=1; i<=n; i++) // 其中“int i=1”无法通过编译          // 3
    sum+=i;                                                     // 4
printf("sum=%d。\n", sum);                                      // 5
```

上面第 3 行代码通常无法通过编译，不能在初始化表达式中定义变量 i。

说明：不过，有些 C 语言支撑平台遵循 C++语法规则，可以编译通过上面的代码。但是，为了保证 C 语言代码的通用性或者说可移植性，仍然不建议在 for 语句的初始化表达式中定义变量。

如图 3-3 所示，在计算初始化表达式之后，开始**计算并判断 for 语句的条件表达式**。如果条件表达式不等于 0，则**表明条件表达式成立**，这时就会执行 for 语句的循环体。如果条件表达式等于 0，则**表明条件表达式不成立**，这时就会结束 for 语句的执行。

在执行完一遍 for 语句的循环体之后，就会计算**更新表达式**。更新表达式通常用来更新循环涉及的变量。因此，更新表达式通常是自增或自减或赋值类运算表达式。如果需要在更新表达式中改变多个变量的值，则通常采用**逗号运算表达式**，即用逗号分隔多个自增或自减或赋值类运算表达式。

小甜点：允许**更新表达式**为空。如果需要更新循环涉及的变量的值，还可以将更新表达式改写为语句放入循环体内部。下面给出相应的代码示例。

```
int i, n, sum;                                                    // 1
for (i=1, n=100, sum=0; i<=n; ) // 更新表达式为空                  // 2
{ // for 循环体开始                                               // 3
    sum+=i;                                                       // 4
    i++; // 更新表达式变成为在循环体中的更新语句                  // 5
} // for 循环体结束                                               // 6
printf("sum=%d。\n", sum); // 结果输出: sum=5050。↙               // 7
```

在上面代码中，第2行 for 语句的更新表达式为空。变量 i 的更新是循环体的最后一条语句，位于上面代码的第5行。这种写法是允许的，只是没有原来的简洁。不过，当更新表达式比较复杂时，可以考虑采用这种方式编写代码。

如图 3-3 所示，在计算更新表达式之后，又会重新开始**计算并判断 for 语句的条件表达式**。如果条件表达式成立，则会继续执行 for 语句的循环体；否则，就会结束 for 语句的执行。这个过程会不断重复下去，直到条件表达式不成立或者在执行循环体时遇到了 break 语句。break 语句和 continue 语句将在后面的章节进行讲解。

注意事项：**编写循环语句的两个常见问题**如下。

（1）整个循环是否得到正确的初始化。因为 for 语句具有显式的初始化表达式，所以采用 for 语句出现这种问题的情况比较少。

（2）对于 for 语句，应当注意更新表达式与条件表达式，既要保证循环体的正常执行，又要保证最终会终止循环。

采用 for 语句的常见场景是要求**重复执行循环体 n 遍**。下面给出实现这一目标的两种常见写法。**第一种写法**如下。

```
int n = 100;                                                      // 1
int i;                                                            // 2
for (i=1; i<=n; i++)                                              // 3
    循环体 // 这里的循环体需要换成实际可行的代码才可以通过编译  // 4
```

如上面第 3 行代码所示，这种写法变量 i 从 1 开始计数，因此条件表达式采用“<=”运算。上面计算从 1 到 100 之和的示例非常适合于这种写法。**另外一种写法**如下。

```
int n = 100;                                                      // 1
int i;                                                            // 2
for (i=0; i<n; i++)                                               // 3
    循环体 // 这里的循环体需要换成实际可行的代码才可以通过编译  // 4
```

如上面第 3 行代码所示，这种写法变量 i 从 0 开始计数，因此条件表达式采用“<”运算。因为 C 语言代码更习惯于从 0 开始计数，例如以后会学到的数组的下标就是从 0 开始计数，所以这种写法更为常见，只是一定要注意这时的条件表达式采用“<”运算。

3.2.2 while 语句

while 语句是 C 语言的 3 种循环语句之一。while 语句的格式是：

```
while (条件表达式)
    循环体
```

其中，循环体一般是一条语句或一个语句块。

注意事项：在 while 语句当中，“while (*条件表达式*)”的后面没有分号，除非循环体是空语句。如果“while (*条件表达式*)”的后面紧跟着分号，则 while 语句的循环体是空语句。这通常不是一种正常的情况，因为一个正常的循环至少应当具有可以引起循环结束的语句。

如图 3-4 所示，在执行 while 语句时，先计算并判断条件表达式。如果条件表达式不等于 0，则表明条件表达式成立，这时就会执行 while 语句的循环体。如果条件表达式等于 0，则表明条件表达式不成立，这时就会结束 while 语句的执行。在执行 while 语句的循环体之后，又会重新开始计算并判断 while 语句的条件表达式。如果条件表达式成立，则会继续执行 while 语句的循环体；否则，就会结束 while 语句的执行。这个过程会不断重复下去，直到条件表达式不成立或者在执行循环体时遇到了 break 语句。break 语句和 continue 语句将在后面的章节进行讲解。

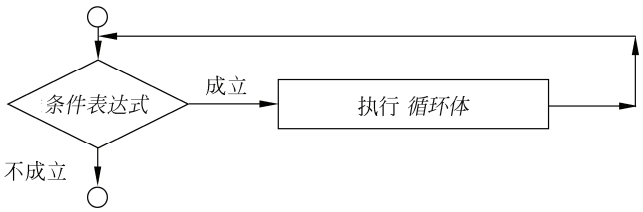


图 3-4　while 语句流程图

从 while 语句的执行过程可以看出，通常应当在 while 语句之前完成 while 语句的循环初始化。而对循环变量的更新则应当在 while 语句的循环体中完成，从而保证最终会终止循环。可以根据这个思想，将 for 语句改写成为 while 语句。

下面给出采用 **while** 语句实现计算从 **1** 到 **100** 之和的示例。

```
int sum = 0;                                                    // 1
int n = 100;                                                    // 2
int i = 1;                                                      // 3
while (i<=n)                                                    // 4
{                                                               // 5
    sum+=i;                                                     // 6
    i++;                                                        // 7
} // while 结束                                                 // 8
printf("sum=%d。\n", sum); // 结果输出: sum=5050。↙             // 9
```

在上面的示例中，对 while 语句循环的初始化在 while 语句之前的第 1～3 行就已经完成。在 while 语句的循环体中，也就是第 7 行代码处，实现对 while 语句的循环变量的更新。读者可以自行比较上面的代码与 3.2.1 小节采用 for 语句实现的示例代码。

注意事项：在上面的示例代码中，第 5 行和第 8 行作为语句块标志的一对大括号“{ }”是不能去掉的。如果去掉第 5 行和第 8 行的代码，则 while 语句的循环体从一个语句块变为一条语句“sum+=i;”。这样，在执行 while 语句时，变量 i 的值一直都不会发生变化，从而造成 while 条件表达式“i<=n”永远成立，程序进入死循环，即无法正常终止 while 语句的运行。因为 while 语句的运行无法正常结束，所以无法正常运行到第 7 行代码“i++;”。

3.2.3 do-while 语句

do-while 语句是 C 语言的 3 种循环语句之一。do-while 语句的格式是：

```
do
    循环体
while (条件表达式);
```

其中，循环体一般是一条语句或一个语句块。

注意事项：在 do-while 语句当中，“while (*条件表达式*)”的后面紧接着分号，表明 do-while 语句结束。在关键字 do 的后面没有分号，除非循环体是空语句。在常规情况下，do-while 语句的循环体不会是空语句，因为一个正常的循环通常至少应当具有可以引起循环结束的语句。

如图 3-5 所示，在执行 do-while 语句时，先直接执行循环体，再计算并判断条件表达式。如果条件表达式不等于 0，则表明条件表达式成立，这时就继续执行 do-while 语句的循环体。如果条件表达式等于 0，则表明条件表达式不成立，这时就会结束 do-while 语句的执行。在执行 do-while 语句的循环体之后，又会重新开始计算并判断 do-while 语句的条件表达式。如果条件表达式成立，则会继续执行 do-while 语句的循环体；否则，就会结束 do-while 语句的执行。这个过程会不断重复下去，直到条件表达式不成立或者在执行循环体时遇到了 break 语句。break 语句和 continue 语句将在后面的章节进行讲解。

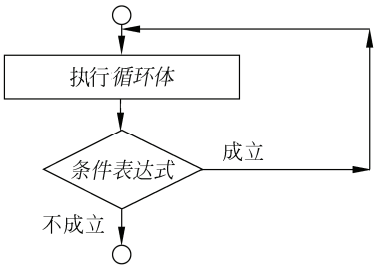


图 3-5　do-while 语句流程图

小甜点：如图 3-5 所示，在执行 do-while 语句时，循环体至少会被执行一遍。

从 do-while 语句的执行过程可以看出，通常应当在 do-while 语句之前完成 do-while 语句的循环初始化。而对循环变量的更新则通常应当在 do-while 语句的循环体中完成，从而保证最终会终止循环。可以根据这个思想，将 for 语句改写成为 do-while 语句。

下面给出采用 **do-while** 语句实现计算从 **1** 到 **100** 之和的示例。

```
int sum = 0;                                                    // 1
int n = 100;                                                    // 2
int i = 1;                                                      // 3
do                                                              // 4
{                                                               // 5
    sum+=i;                                                     // 6
    i++;                                                        // 7
}                                                               // 8
while (i<=n);                                                   // 9
printf("sum=%d。\n", sum); // 结果输出: sum=5050。↙             // 10
```

在上面的示例中，对 do-while 语句循环的初始化在 do-while 语句之前的第 1～3 行就已经完成。在 do-while 语句的循环体中，也就是第 7 行代码处，实现对 do-while 语句的循环变量的更新。读者可以自行比较上面的代码与 3.2.2 小节采用 while 语句和第 3.2.1 小节采用 for 语句实现的示例代码。

3.2.4　continue 语句

C 语言标准规定 continue 语句只能用在循环语句中。continue 语句的写法如下。

```
continue;
```

如图 3-6 所示，当执行循环语句遇到 continue 语句时，程序会自动结束循环体剩余代码的运行。然后，对于 for 语句，则会立即计算更新表达式，并依据条件表达式，决定是重新继续执行一遍循环体还是结束循环语句；对于 while 语句和 do-while 语句，则会立即计算并判断条件表达式，决定是重新继续执行一遍循环体还是结束循环语句。这个过程可以不断地重复下去，直到循环语句运行结束。

小甜点：continue 语句通常作为条件语句 if 语句或 if-else 语句的一部分出现在循环语句的循环体中。如果直接将 continue 语句作为一条独立的语句放入循环语句的循环体中，则在 continue 语句之后的循环体语句都将不起作用。

例程 3-1　接受输入 5 个整数并计算其中正整数的平均值例程。

例程功能描述：该例程依次接受 5 个整数的输入，统计并输出其中正整数的平均值。

例程解题思路：设计整数类型的变量 i 和 n，用来控制输入整数的个数以及循环运行的总次数。设计整数类型的变量 number，用来保存输入的整数。设计整数类型的变量 sum，用来保存正整数之和；并用整数类型的变量 k 统计正整数的个数。利用 continue 语句，跳过对负整数和 0 的统计。最后输出正整数的平均值。在计算平均值时，将整数运算转化成为双精度浮点数运算，提高计算精度。例程由一个源程序文件 C_PositiveNumberAverage.c 组成，具体的程序代码如下。

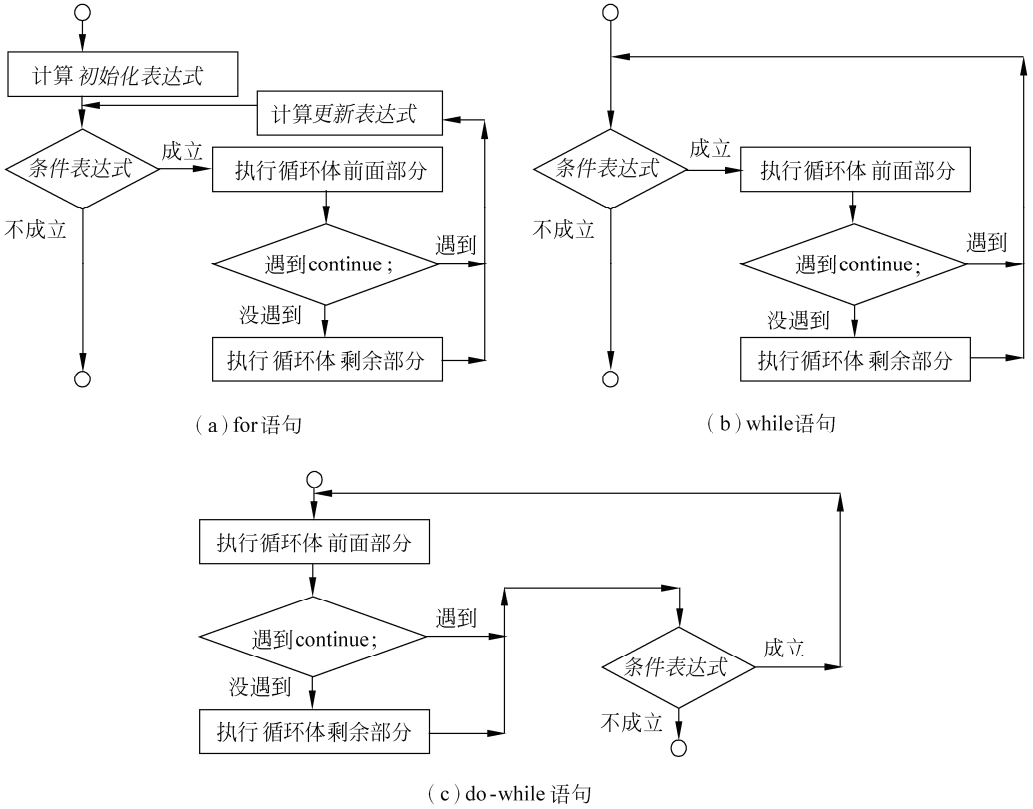


图 3-6　包含 continue 语句的循环语句流程图

// 文件名：C_PositiveNumberAverage.c；开发者：雍俊海	行号
#include <stdio.h>	// 1
#include <stdlib.h>	// 2
	// 3
int main(int argc, char* args[ ])	// 4
{	// 5
int i, k, number, sum, n;	// 6
for (i=1, k=0, sum=0, n=5; i<=n; i++)	// 7
{	// 8
printf("请输入第%d 个整数: ", i);	// 9
scanf("%d", &number);　　// 在 VC 平台中，应将 scanf 改为 scanf_s	// 10
if (number<=0) // 跳过 0 和负整数	// 11
continue;	// 12
sum += number; // 对正整数求和	// 13
k++;　　// 统计正整数个数	// 14
} // for 循环结束	// 15
if (k>0)	// 16
printf("正整数的平均值是%g。\n", sum/(double)k);	// 17
else printf("没有输入正整数。\n");	// 18

```
    system("pause"); // 暂停住控制台窗口                                    // 19
    return 0; // 返回 0 表明程序运行成功                                    // 20
} // main 函数结束                                                        // 21
```

可以对上面的代码进行编译、链接和运行。下面给出一个运行的结果示例。

```
请输入第 1 个整数：12↙
请输入第 2 个整数：-1↙
请输入第 3 个整数：13↙
请输入第 4 个整数：-2↙
请输入第 5 个整数：14↙
正整数的平均值是 13。
请按任意键继续...
```

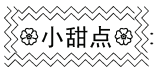
例程进一步说明：如果去掉第 11 行的代码，则上面第 13 行和第 14 行的代码将都不会起作用。上面第 11 行和第 12 行代码的共同作用，使得 0 和负整数不会进入统计，即当输入的是 0 或负整数时，上面第 13 行和第 14 行的代码都不会被执行。上面第 16 行代码通过“if (k>0)”使得第 17 行的运算“sum/(double)k”不会出现除数为 0 的情况。

3.2.5　break 语句

C 语言标准规定 break 语句只能用在 switch 语句和循环语句中。break 语句的写法如下。

```
    break;
```

第 3.1.2 小节已经介绍了在 switch 语句中的 break 语句。因此，这里只介绍在循环语句中的 break 语句。如图 3-7 所示，当执行循环语句遇到 break 语句时，程序会立即自动结束整个循环语句的运行。



: break 语句通常作为条件语句 if 语句或 if-else 语句的一部分出现在循环语句的循环体中。如果直接将 break 语句作为一条独立的语句放入循环语句的循环体中，则在 break 语句之后的循环体语句将都不起作用。

例程 3-2　接受以 0 或负整数为结束标志的多个正整数输入，并计算其中正整数的平均值例程。

例程功能描述：该例程依次接受整数的输入。若输入的是正整数，则继续输入；若输入的是 0 或负整数，则表示输入结束。对于输入的所有正整数，统计并输出这些正整数的平均值。

例程解题思路：设计整数类型的变量 n，用来统计输入的正整数的总个数。设计整数类型的变量 number，用来保存输入的整数。设计整数类型的变量 sum，用来保存正整数之和。因为需要接受多个整数的输入，所以需要采用循环语句。这里采用 for 循环语句。因为无法提前知道输入的正整数的总个数，所以 for 语句的条件表达式为 1，即这个 for 语句并不通过 for 语句条件表达式来结束循环，而是利用 break 语句结束循环。在 for 语句的循环体内，接受整数的输入。如果输入的是正整数，则统计已经输入的正整数的总个数，并

计算已经输入的正整数的和。一旦发现输入的是 0 或负整数，则调用 break 语句，立即结束 for 循环语句。在结束 for 语句之后，变量 n 的值已经是输入的正整数的总个数，变量 sum 已经是输入的所有正整数之和。因此，这时可以计算平均值。在计算平均值时，可以考虑将整数运算转化成为双精度浮点数运算，以提高计算精度。例程由一个源程序文件 C_MultiplePositiveNumberAverage.c 组成，具体的程序代码如下。

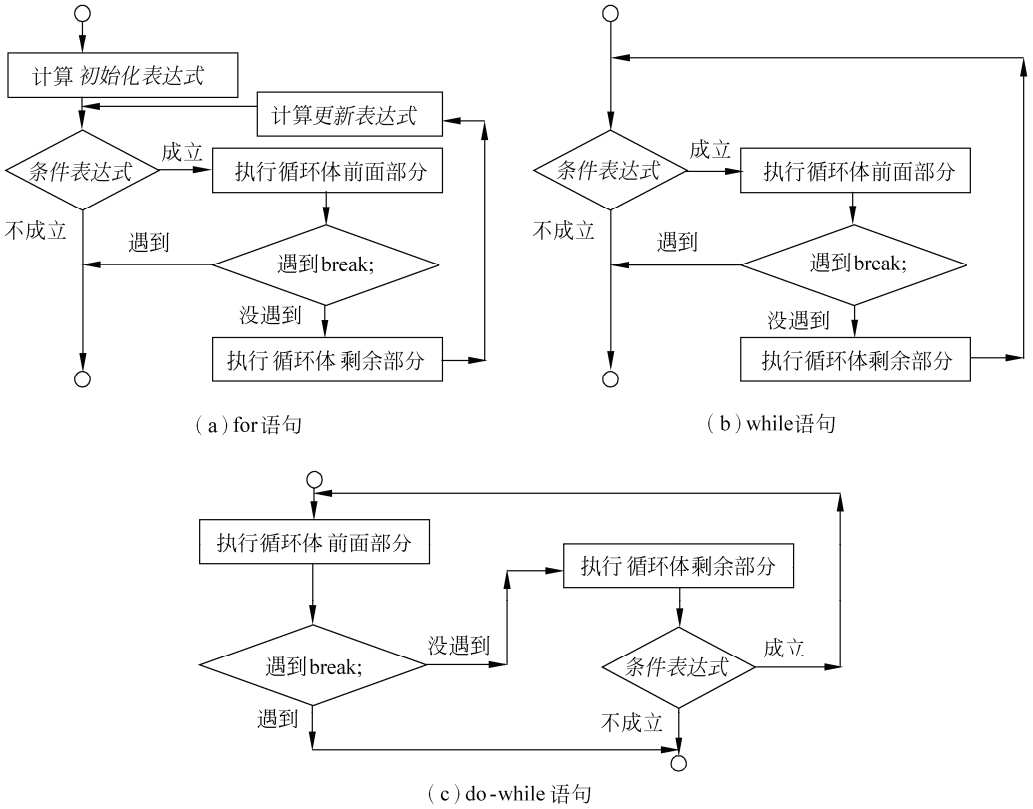


图 3-7　包含 break 语句的循环语句流程图

// 文件名：C_MultiplePositiveNumberAverage.c；开发者：雍俊海	行号
#include <stdio.h>	// 1
#include <stdlib.h>	// 2
	// 3
int main(int argc, char* args[ ])	// 4
{	// 5
int n, sum, number;	// 6
for (n=0, sum=0; 1;)	// 7
{	// 8
printf("请输入第%d 个整数: ", (n+1));	// 9
scanf_s("%d", &number);	// 10

```
        if (number<=0) // 0 或负整数表示输入结束                    // 11
            break;                                                  // 12
        sum += number; // 对正整数求和                              // 13
        n++; // 统计正整数个数                                      // 14
    } // for 循环结束                                              // 15
    if (n>0)                                                        // 16
        printf("正整数的平均值是%g。\n", sum/(double)n);            // 17
    else printf("没有输入正整数。\n");                             // 18
    system("pause"); // 暂停住控制台窗口                           // 19
    return 0; // 返回 0 表明程序运行成功                           // 20
} // main 函数结束                                                 // 21
```

可以对上面的代码进行编译、链接和运行。下面给出一个运行的结果示例。

```
请输入第 1 个整数：5↙
请输入第 2 个整数：6↙
请输入第 3 个整数：7↙
请输入第 4 个整数：8↙
请输入第 5 个整数：-1↙
正整数的平均值是 6.5。
请按任意键继续...
```

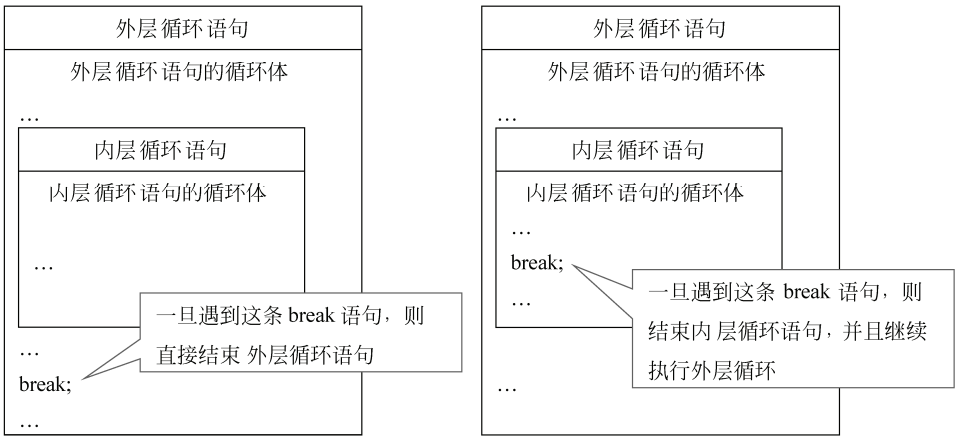
例程进一步说明：如果去掉第 11 行的代码，则上面第 13 行和第 14 行的代码将都不会起作用。上面第 11 行和第 12 行代码的共同作用，使得一旦发现输入的是 0 或负整数，则立即结束 for 循环语句。上面第 16 行代码通过“if (n>0)”使得第 17 行的运算“sum/(double)n”不会出现除数为 0 的情况。

注意事项：如果出现多重嵌套的循环语句，即在循环语句的循环体内仍然含有循环语句，则当遇到 break 语句时，只是立即结束该 break 语句所在的那一层的循环语句，而不会结束其外层的循环语句（当然，这个前提是存在外层的循环语句）。图 3-8 给出了 break 语句在两重嵌套循环语句中的运行示例。

如图 3-8 所示，在两重嵌套循环语句中，如果 break 语句出现在外层循环语句中，则一旦运行到这条 break 语句，则立即会结束这两重嵌套循环语句的运行；如果 break 语句出现在内层循环语句中，则一旦运行到这条 break 语句，只是结束内层循环语句的运行，然后继续执行在外层循环体内并且在内层循环语句之后的语句，即整个外层的循环语句仍然会继续运行。下面给出程序片断示例。

```
int i, k;                                                           // 1
for (i=0; i<3; i++)                                                 // 2
{                                                                   // 3
    for (k=0; k<5; k++)                                             // 4
    {                                                               // 5
```

```
            printf("%d", k);                                    // 6
            if (k==1)                                           // 7
                break;                                          // 8
        } // 内层 for 循环结束                                  // 9
    } // 外层 for 循环结束                                      // 10
```



（a）break 语句在外层循环语句中　　（b）break 语句在内层循环语句中

图 3-8　包含 break 语句的两重嵌套循环语句

这个程序片断的运行结果是：

```
010101
```

在上面的程序片断中，因为 break 语句在内层 for 循环语句中，所以这条 break 语句不会影响外层的 for 循环语句。因此，对于以变量 i 为计数器的外层的 for 循环语句，它的循环体将会执行 3 次。这样，从第 4 行到第 9 行的内层 for 循环语句也就会执行 3 次。对于内层 for 循环语句，当 k 为 0 和 k 为 1 时均会输出 k 的值；而且当 k 为 1 时，在输出 k 的值之后会运行 break 语句，从而造成立即结束内层 for 循环语句的运行。因此，每次执行内层 for 循环语句实际上只是输出“01”。因为内层 for 循环语句执行 3 遍，所以最终的输出是“010101”。

3.3　小结

C 语言的 3 类控制结构各有特点。选择语句包括 if 语句、if-else 语句和 switch 语句，这些语句使得程序可以根据不同的条件执行不同的语句。循环语句包括 for 语句、while 语句和 do-while 语句，这三者之间可以互相转换。通常采用哪种语句编写代码简洁，就采用哪种。其中，最常用的是 for 语句，因为它在形式上最符合循环的特征。continue 语句和 break 语句在一定程度上起到辅助的作用。通常要慎重使用 continue 语句和 break 语句。

小甜点：C 语言语句的结束标志有可能是分号“;”，也有可能是语句块。

3.4　习题

习题 3.1　简述 C 语言有哪些控制结构。

习题 3.2　选择语句包括哪些类型的语句?

习题 3.3　循环语句包括哪些类型的语句? 它们的区别是什么?

习题 3.4　请判断下面各个结论的对错。

（1）在 for 语句中，初始化表达式和更新表达式均允许为空。

（2）在 C 语言的各种循环语句中，采用 for 语句的运行效率是最高的，因此 for 语句也是最常用的。

习题 3.5　请写出下面程序片断输出的内容，并指出下面程序排版的不合理之处。

```
int x=-1;                                                    // 1
int y=0;                                                     // 2
if  (x>=0)                                                   // 3
   if (x>0) y=1;                                             // 4
else  y=-1;                                                  // 5
printf("x=%d, y=%d", x, y);                                  // 6
```

习题 3.6　下面程序片断是否含有语法错误？如果没有，请写出其运行结果的输出内容。

```
char ch='B';                                                 // 1
switch (ch)                                                  // 2
{                                                            // 3
case 'A':                                                    // 4
case 'a':                                                    // 5
   printf("优秀");                                           // 6
   break;                                                    // 7
case 'B':                                                    // 8
case 'b':                                                    // 9
   printf("良");                                             // 10
default:                                                     // 11
   printf("再接再厉");                                       // 12
}                                                            // 13
```

习题 3.7　下面程序片断是否含有错误？如果没有，请写出其运行结果的输出内容。

```
double a=6;                                                  // 1
switch(a)                                                    // 2
{                                                            // 3
```

```
case 6:                                                    // 4
   printf("Saturday。\n");                                 // 5
   break;                                                  // 6
case 7:                                                    // 7
   printf("Sunday。\n");                                   // 8
   break;                                                  // 9
default:                                                   // 10
   printf("Unknown。\n");                                  // 11
   break;                                                  // 12
} // switch 语句结束                                       // 13
```

习题 3.8　下面程序片断是否含有错误？如果没有，请写出其运行结果的输出内容。

```
int a = 6;                                                 // 1
switch(a)                                                  // 2
{                                                          // 3
case 6.0:                                                  // 4
   printf("Saturday。\n");                                 // 5
   break;                                                  // 6
case 7.0:                                                  // 7
   printf("Sunday。\n");                                   // 8
   break;                                                  // 9
default:                                                   // 10
   printf("Unknown。\n");                                  // 11
   break;                                                  // 12
} // switch 语句结束                                       // 13
```

习题 3.9　请找出并更正下面程序片断的错误。

```
char ch=0;                                                 // 1
char sum=0;                                                // 2
                                                           // 3
while (ch<5);                                              // 4
{                                                          // 5
   ch++;                                                   // 6
   sum+=ch;                                                // 7
}                                                          // 8
printf("sum=%d", sum);                                     // 9
```

习题 3.10　请总结 break 语句的用法。

习题 3.11　请总结 continue 语句的用法。

思考题 3.12　能否写出不含分号的 C 语言语句?

思考题 3.13　能否写出不以分号结尾的 C 语言语句?

习题 3.14　请编写程序，接受输入 10 个整数，计算并输出其平均数。

习题 3.15　请编写程序，接受输入 1 个正整数，计算并输出不超过这个正整数的所有“水仙花数”。这里“水仙花数”是一个正整数，它的各个十进制位的立方和等于它本身。例如，1 是“水仙花数”，因为 $1=1^3$。再如，153 是“水仙花数”，因为 $153=1^3+5^3+3^3$。

习题 3.16　请编写程序，接受一系列整数的输入，其中输入的最后一个整数是 0。要求计算并输出除了整数 0 之外其他输入的整数的最大值和最小值。

习题 3.17　请编写程序，接受 3 个正整数 y、m 和 d 的输入。请判断 y 年 m 月 d 日是否是一个合法日期。如果是一个合法的日期，则输出这一天是星期几；否则，请输出字符串“这是一个无效的日期”。