第3章 程序基本结构

如果没有条件判断语句，世界上最先进的计算机也只是一个计算器。

人们之所以感到计算机无所不能，是因为计算机能不厌其烦地重复大量枯燥的工作。

3.1 程序语句

在C语言中，程序是以从上到下、从左到右的方式书写，但是程序的执行并不完全按照从上到下、从左到右的次序执行。可以通过特定控制结构，控制程序语句的执行次序以达到特定功能。

3.1.1 语句概述

C语言的语句(statement)以“;”作为分隔符，编译后产生机器指令。C语言语句可分为两大类：简单语句和复合语句。

1. 简单语句

从形式上看，简单语句是单独的一条语句。一般处于两个分号之间的语句可以看做简单语句。简单语句可分为说明语句，表达式语句，空语句。

1) 说明语句

对变量的定义，对函数原型的说明，对数据结构的说明和描述等都是说明语句。下面几条语句都是说明语句。

```
int i,j;                          //定义了两个整型变量
double x,y;                       //定义两个双精度浮点型变量
char a[20];                       //定义一个字符型数组变量
```

变量定义语句不产生可执行代码，只是告诉编译器两件事情：

(1) 应将变量放在内存的什么地方？

(2) 如何给变量分配内存？

2) 表达式语句

表达式语句由表达式加分号构成。例如：

```
total=total+limit;                //赋值语句,赋值表达式后跟分号构成
a=3;                              //赋值语句,赋值表达式后跟分号构成
```

```
func();                              //函数调用语句,函数调用表达式后面跟一分号
printf("Hello,world!\n");            //函数调用语句,函数调用表达式后面紧跟一个分号
```

3) 空语句

如果在两个分号之间没有任何内容,则两个分号之间的语句称为空语句。独立的空语句没有任何实际意义。空语句一般都是与循环语句配合使用。

4) 流程控制语句

流程控制语句主要有两大类,一类是选择结构语句,一类是循环结构语句。流程控制语句主要通过下面一些关键字与表达式适当组合构成。

if()…else…	选择结构
switch…case…	分支结构(多选结构)
for()…	for 循环结构
while()…	while 循环结构
do…while()	do…while 循环结构
continue	循环辅助控制语句
break	分支结构与循环结构辅助控制语句
return	函数返回语句
goto	无条件转向语句

2. 复合语句

复合语句也称为块语句,是用一对花括号{…}括起来的一组简单语句。

一般形式:

```
{
    [数据说明部分;]
    执行语句部分;
}
```

说明:

(1)“}”后不加分号。

(2) 复合语句的语法功能和单一语句相同。

(3) 复合语句可以嵌套。

(4) 理论上讲,复合语句可出现在任何简单语句可出现的地方,但是实际上复合语句一般总是与选择结构、循环结构一起使用。

3.1.2 结构化程序设计

基本思想:任何程序都可以用三种基本结构表示,这三种基本结构是:**顺序结构、选择结构、循环结构**。

结构化程序:由三种基本结构反复嵌套构成的程序叫做结构化程序。

3.2　顺序结构

顺序结构是程序设计中最基本也是最简单的结构，其基本要点是按照解决问题的思路，从上到下、从左到右的顺序写出相应的语句，它的执行顺序是自上而下，依次执行。

【例 3-1】 输入三角形的三边长，求三角形面积。

分析：为简单起见，设输入的三边长 a、b、c 能构成三角形。从数学上知道，求三角形面积的公式为：

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

其中，s=(a+b+c)/2。

根据上述公式可以直接编写程序如下：

```
1. #include <stdio.h>
2. #include <math.h>
3. void main()
4. {
5.     float a,b,c,s,area;
6.     scanf("%f%f%f",&a,&b,&c);
7.     s=1.0/2*(a+b+c);
8.     area=sqrt(s*(s-a)*(s-b)*(s-c));  //函数 sqrt()在头文件 math.h 中说明
9.     printf("a=%-7.2f, b=%-7.2f,c=%-7.2f,s=%-7.2f\n",a,b,c,s);
10.    printf("area=%-7.2f\n",area);
11. }
```

运行结果：

```
3 4 5
a=3.00, b=4.00,c=5.00,s=6.00
area=6.00
```

关于顺序结构的说明：

(1) 顺序结构的基本特点是语句之间的前后顺序非常重要。例如，a=3，b=5，现交换 a，b 的值，这个问题就好像交换两个杯子中的水，这当然要用到第三个杯子，假如第三个杯子是 c，那么正确的程序为：

```
c=a; a=b;b=c;
```

执行后 a 的值是 5，b 的值是 3。如果改变其顺序，写成：

```
a=b; c=a; b=c;
```

则执行结果就变成 a 和 b 的值都是 5，不能达到预期的目的，初学者最容易犯这种错误。

(2) 大多数情况下顺序结构都是作为程序的一部分，与其他结构一起构成一个复杂的程序，例如，分支结构中的复合语句、循环结构中的循环体等。

3.3 选择结构

根据不同的条件采取不同的行动,这是人类的一种基本智能。程序语言有了选择语句,就具有了模仿人类智能的基本能力。

选择结构,指的是根据某个条件是否成立,选择执行程序的不同部分。选择结构也称为选择语句或者条件语句。

选择结构有三种常见的语法格式。

3.3.1 选择结构的基本语法格式

1. 语法格式一

```
if(expr1)
    S1;
```

执行过程:这是一种最基本的选择结构,如果条件表达式 expr1 的值为真,执行语句 S1;如果 expr1 的值为假,则跳过 S1,接着执行后续语句。S1 既可以是简单语句,也可以是多条语句组成的复合语句。

示例 1:

```
if(x<0)
    printf("%d",-x);  //简单语句,可以不用花括号
```

示例 2:

```
if(x<0)
{
    x=-x;
    printf("%d",x);
}                     //x<0 为真时,两条语句需要同时执行,是块语句,需要用花括号括起来
```

2. 语法格式二

```
if(expr1)
    S1;
else
    S2;
```

执行过程:如果条件表达式 expr1 的值为真,执行语句 S1,否则执行 S2,S2 执行完成后,接着执行后续语句。

示例:

```
if(x>y)
    printf("%d",x);    //简单语句,可以不用花括号
else
```

```
    printf("%d",y);
```

3. 语法格式三

```
if(expr1)
    S1;
else if(expr2)
    S2;
else if(expr3)
    S3;
else
    S4;
```

执行过程：如果 expr1 的值为真，执行语句 S1，否则判断 expr2 是否为真；如果 expr2 的值为真，执行语句 S2，否则判断 expr3 是否为真；如果 expr3 的值为真，执行 S3；如果 expr1，expr2，expr3 的值都不为真，则执行 S4。

应该注意的是，如果 expr1 的值为真，执行完 S1 后，不会判断 expr2、expr3 的真假，而是直接执行后续语句。

示例：

```
double score;
char grade;
if(score>=90)
    grade='A';
else if(score>=80 && score<90)
    grade='B';
else if(score>=70 && score<80)
    grade='C';
else if(score>=60 && score<70)
    grade='D';
else
    grade="E';
```

选择结构的说明：

(1) if 和 else 是系统关键字，if 后面的一对圆括号“()”不能缺少。圆括号中的 expr1、expr2 等是选择结构的**条件表达式**。

(2) C 语言规定，条件表达式 expr1、expr2 等可以是任何能计算出值的表达式，**条件表达式的值为 0 时表示假，为非 0 时表示真**。在实际使用过程中，建议条件表达式使用关系表达式或者逻辑表达式。

(3) 语句 S1，S2，S3 既可以是简单语句，也可以是复合语句。当语句是复合语句时，注意一定要用一对花括号“{}”将语句括起来。

(4) "if(exp1) S1;"是一个整体，构成一条语句。单独的 if(exp1)不构成一条语句。如果不小心写成下面的形式：

```
if(exp1);          //注意这行结尾多了一个分号
S1;
```

这种情况下,S1 语句总会执行,不受 exp1 值的影响。而 if(exp1)与其后的分号构成一条“空的条件语句”。

同样道理,

```
if(exp1) S1;
else S2;
```

也是一个整体,构成一条语句。

3.3.2 选择结构举例

【例 3-2】 从键盘输入一个实数 a,然后输出其绝对值。

分析:对于任意给定的一个数 a,如果 a≥0,绝对值等于它自身,不需要做任何处理;如果 a<0,则-a 就是 a 的绝对值,显然本例使用编程模式 1。

```
1. #include<stdio.h>
2. main()
3. {
4.     float a;
5.     scanf("%f",&a);
6.     if(a<0)                  //使用模式 1
7.         a=-a;                //一条简单语句,省略了一对花括号"{}"
8.     printf("a 的绝对值=",a);
9. }
```

【例 3-3】 从键盘任意输入三个数,输出三个数中最大的数和最小的数。

1. 分析

(1) 变量设置:设置两个变量 max3,min3 分别表示三个数中的最大值、最小值。

(2) 首先求出 a,b 两个数中的最大值和最小值,并分别赋给 max3 和 min3。

(3) 将上一步的 max3 和 min3 与 c 进行比较,求出三个数中的最大值和最小值。

2. 程序实现

根据上述分析,可以写出 C 语言程序如下。

```
1. #include<stdio.h>
2. void main()
3. {
4.     float a,b,c,max3,min3;
5.     printf("输入三个数 a,b,c\n");
6.     scanf("%f%f%f",&a,&b,&c);
7.     if(a>b)
8.     {
9.         max3=a;
```

```
10.         min3=b;
11.     }
12.     else
13.     {
14.         max3=b;
15.         min3=a;
16.     }
17.     if(max3<c)
18.         max3=c;          //一条简单语句,可以省略花括号
19.     if(min3>c)
20.         min3=c;          //一条简单语句,可以省略花括号
21.     printf("max3=%f min3=%f\n",max3,min3);
22. }
```

说明：

(1) 程序中第 9、10 行是由两条简单语句组成的语句块，表示满足条件 a＞b 的情况下，这两条语句必须同时执行，因此需要用第 8 行和第 11 行的一对花括号“{}”括起来，即第 7～11 行是一个整体，这是一条复合语句。

(2) 程序中第 14、15 行也是一个语句块，表示不满足 a＞b 这个条件的情况下，必须同时执行。因此也需要用一对花括号括起来，即第 12～16 行是一个整体。

(3) 最后，第 7～16 行又可以看成整体，表示一个 if…else 结构。

3.3.3　选择结构的嵌套

在选择结构的三种格式中，S1，S2，S3 本身还可以是选择结构语句，这称为选择结构的嵌套。

例如，对于语法格式一

```
if(exp1)
    S1;
```

如果语句 S1 本身是另一个“语法格式”一的选择结构，即 S1 是下面形式的另一个选择结构：

```
if(expr1_1)
    S1_1;
```

则整个语句可以表示成：

```
if(expr1)
    if(expr1_1)
        S1_1;
```

同样，如果 S1 用“语法格式二”替换，则整个选择结构变成：

```
if(exp1)
    if(expr1_1)
```

```
        S1_1;
    else
        S1_2;
```

【例 3-4】 求方程 ax=b 的解(a,b 是整数)。

分析：看到这个问题,读者马上会想到,方程的解是 x=b/a;只要知道 a,b 的值,直接输出 b/a 即可。然而这样编程是不完整的,正确的方法是根据 a,b 的值分为不同的情况分别处理：

$$x=\begin{cases} b/a & a\neq 0 \\ 任意数值 & a=0,b=0 \\ 无解 & a=0,b\neq 0 \end{cases}$$

根据 a 的值进行不同的处理。a 等于 0 是一种处理方式,a 不等于 0 是另一种处理方式。在 a 等于 0 的情况下,还要根据 b 的值进行不同处理。因此在编程时,要用到选择结构的嵌套。

根据上述分析,可写出如下两个程序。

程序 1：

```
#include<stdio.h>
main()
{
    int a,b;
    scanf("%d%d",&a,&b);
    if(a==0)
        if(b==0)
            printf("x=任意数值\n");
        else
            printf("无解");
    else
        printf("x=%f\n",(float)b/a);
}
```

程序 2：

```
#include<stdio.h>
main()
{
    int a,b;
    scanf("%d%d",&a,&b);
    if(a==0 && b==0)
        printf("x=任意数值\n");
    else if(a==0 && b!=0)
        printf("无解");
    else
        printf("x=%f\n",(float)b/a);
}
```

思考：选择结构的嵌套非常灵活，读者可以思考一下，本题还有其他的嵌套方式吗？

在嵌套的选择语句中，往往一个 else 语句前面可能有多个 if 语句。这个时候，else 到底与前面的哪一个 if 匹配呢？

为了解决嵌套条件语句的二义性问题，C 语言规定了 else 匹配的就近原则：**默认{ } 时，else 总是和它前面离它最近的未配对的 if 配对**。例如：

```
1. if (a==b)
2.     if(b==c)
3.         printf("a==b==c");
4.     else
5.         printf("a!=b");
```

在这段程序中，有两个 if 关键字，一个 else 关键字，根据配对原则，第 4 行的 else 应该与离它最近的第二个 if 配对。根据这个配对原则，无论是第 2、3 行的 if 分支还是第 4、5 行的 else 分支，都是在第 1 行的 if(a＝＝b)这个条件控制之下。然而根据程序中的输出语句看，程序设计者的本意并不是如此，程序设计者的本意是想让第 4、5 行的 else 子句与第一个 if 配对。那么如何解决这样的问题呢？

为了防止嵌套条件语句中 if…else 配对的二义性问题，可以采取加“{ }”的方法。对上例，根据程序的含义，应该修改成下面的形式：

```
1. if (a==b)
2. {
3.     if(b==c)
4.         printf("a==b==c");
5. }
6. else
7.     printf("a!=b");
```

【例 3-5】 某同学在练习编程时，写了下面一段程序，请分析该程序的输出结果。

```
 1. #include<stdio.h>
 2. main()
 3. {    int x=100,a=10,b=20;
 4.      int v1=5,v2=0;
 5.      if(a<b)
 6.          if(b!=15)
 7.              if(!v1)
 8.                  x=1;
 9.      else
10.          if(!v2)
11.              x=10;
12.              x=-1;
13.      printf("x=%d\n",x);
14. }
```

分析：该程序是个多层嵌套选择语句结构。分析的要点是要注意 if、else 的控制关系及

配对关系。

(1) 因为程序最后输出的是变量 x 的值,最后一条给 x 赋值的语句是第 12 行的赋值语句,由于第 10 行的 if 后面没有花括号,不是复合语句,因此,第 12 行的赋值语句不受第 10 行的 if 控制。同样,第 9 行的 else 后面也不是复合语句,因此第 12 行的赋值语句也不受 else 控制。

(2) 现在要分析第 9 行的 else 到底与哪一个 if 配对。根据配对原则,else 应该与离它最近的还没有配对的 if 配对,因此 else 应该与第 7 行的 if 配对。第 7 行的 if 受第 6 行的 if 控制,同样第 6 行的 if 受第 5 行的 if 控制。

(3) 由于第 6 行、第 5 行的 if 后面都没有花括号"{}",不是复合语句,也就是说第 6～8 行的语句都受第 5 行的 if 控制,第 10～11 行的语句受第 9 行的 else 控制。但是第 12 行的赋值语句不受任何条件语句控制。

(4) 根据上述分析,第 12 行的赋值语句不受任何条件语句控制,并且是最后一条赋值语句,因此,程序的输出结果应该是 x=－1。

输出结果:

```
x=-1
```

(5) 从本例可以体会到程序书写格式的重要性。事实上,上述程序可以调整一下格式,写成下面的形式,语句之间的逻辑关系一目了然,这样无论本人以后维护或者是其他人阅读,都更容易理解。例如,下面的程序可以一眼看出最后一条赋值语句与第 1 个 if 是平行的,不受任何 if、else 控制,可以马上断定输出结果是 x=－1。

```
        …
 9.     else
10.         if(!v2)
11.             x=10;
12.     x=-1;
13.     printf("x=%d\n",x);
14. }
```

3.3.4 条件运算符

条件运算符是一个三元运算符,其格式如下:

expr1? expr2: expr3

这个表达式由两个运算符"?"和":"组成。因为需要三个操作数(expr1,expr2,expr3)参与运算,所以称为三元运算符。

其中,expr1 是一个可判断真假的表达式。如果 expr1 的值是真(非零),则取 expr2 作为条件运算符的结果;如果 expr1 的值是假(零),则取 expr3 作为条件运算符的结果。因此条件运算符等价于如下的条件语句:

```
if(expr1)
    expr2;
```

```
else
    expr3;
```

【例 3-6】 下面两条语句执行后，变量 r 的值是多少？

```
int a=5,b=10,r;
r=a>b?a:b;
```

分析：因为 a＞b 的值是 0(假)，所以整个条件运算符的结果是 b，即 r＝b，b 的值是 10，所以 r 的值也是 10。

关于条件运算的使用，应注意下面几点：

(1) 条件运算符的优先级别比关系运算符和算术运算符都低。因此，如果有表达式：

a＞b?a：b＋1

相当于 a＞b?a：(b＋1)，而不相当于(a＞b?a：b)＋1。

(2) 条件运算符的结合方向为“自右至左”。如果有以下条件表达式：

a＞b?a：c＞d?c：d

相当于

a＞b?a：(c＞d?c：d)

【例 3-7】 (字符大写转小写)输入一个字符，判别它是否是大写字母，如果是，将它转换成小写字母；如果不是，不转换。

分析：

(1) 大写字母 A 的 ASCII 是 65，小写字母 a 的 ASCII 是 97，其差值是 32。

(2) 如果 ch 是大写字母(ch＞＝'A' && ch＜＝'Z')，ch＋32 结果就是小写字母，这里 32 是小写字母和大写字母 ASCII 码的差值。例如，B＋32 得到 b。

根据上述分析，写出如下程序。

```
1. main()
2. {  char ch;
3.    scanf("%c",&ch);
4.    printf("%c",(ch>='A' && ch<='Z')?ch+32:ch);
5. }
```

3.3.5　多分支选择结构

switch 语句是多分支选择语句，其语句格式为：

```
switch(expr)
{
  case M1:        /* 如果 expr 与 M1 相等，执行语句段 S_1，一个语句段可以有多条语句 */
      S_1;
      [break;]    /* break 语句是可选的，如果省略 break 语句，则接着执行后面的所有语句，
                     否则跳出 switch 语句，执行后面的 S_other 语句 */
  case M2:        /* 如果 expr 与 M2 相等，执行语句段 S_2 */
      S_2;
      [break;]    /* 如果省略 break 语句，接着执行后面的所有语句 */
```

```
    …
    case Mn:
        S_n;
        [break;]
    default:        /* 如果 expr 与 M1,M2,…,Mn 都不相等,执行语句段 S_d */
        S_d;
}
S_other;
    …
```

switch 语句的执行过程如下：

(1) 首先计算 switch 后面的条件表达式 expr 的值。expr 表达式必须是整型或者字符型(本质上,字符型也是一种整型)。

(2) 依次计算出每个 case 后面的常量表达式 M1,M2,…,Mn 的值,若它们的值不是整型,则会出现语法错误。

(3) 让 expr 依次同 M1,M2,…,Mn(进行比较,一旦遇到 expr 与某个值相等,则就从对应标号的语句开始向下执行,若没有碰到跳转类语句,将一直执行到右花括号为止才结束整个 switch 语句的执行,若 expr 与所有值都不同,则当带有 default 部分时,就从该标号位置起向下执行,否则不执行任何操作。

说明：

(1) case 子句可以有多个,但是 default 子句最多只能有一个。default 子句是可以省略的。如果省略 default 子句,则 expr 与所有 case 子句都不匹配时,什么也不执行。

(2) 表达式 expr,M1,M2,…,Mn 都是广义的整型表达式(整型和字符型),如果它们不是整型表达式,则会出现语法错误。

(3) M1,M2,…,Mn 都是常量表达式,如果 M1,M2,…,Mn 中出现变量,会出现语法错误。

(4) 在实际使用 switch 语句时,通常要求当执行完某个 case 分支后就结束整个语句的执行,而不让它继续执行下一个 case 分支,为此,可通过使用 break 语句来实现。在 switch 语句中遇到 break 时,将跳出整个 switch 语句块。

【例 3-8】 假如已知学生成绩的等级(A,B,C 等),要求输出百分制分数段,程序如下。

```
1. include<stdio.h>
2. void main()
3. {
4.     char grade;
5.     printf("Input grade:");
6.     scanf("%c",&grade);
7.     switch(grade)
8.     {
9.         case 'A':printf("90-100\n");
10.        case 'B':printf("80-89\n");
11.        case 'C':printf("70-79\n");
12.        case 'D':printf("60-69\n");
```

```
13.         case 'E':printf("0-59\n");
14.         default:printf("错误!\n");
15.     }
16. }
```

运行结果：

```
Input grade:A
90-100
80-89
70-79
60-69
0-59
错误!
```

程序说明；

(1) 从程序运行结果看，从键盘输入等级'A'后，希望输出的值应该是 90－100，现在程序的实际输出结果显然不合理。

(2) 从程序结果可以看出 switch 语句的执行过程，首先计算 grade 的值是'A'，然后计算每一个 case 标号的值，发现第一个 case 标号的值是'A'，与 switch 后面的表达式 grade 的值匹配，所以首先执行该 case 语句下的语句。由于程序中没有 break 语句，所以第一个 case 语句执行完成后，会继续按顺序执行后续所有语句。因此正确的做法是在每一个 case 分支的最后添加一条 break 语句。

修改后的程序如下：

```
 1. #include<stdio.h>
 2. void main()
 3. {
 4.     char grade;
 5.     printf("Input grade:");
 6.     scanf("%c",&grade);
 7.     switch(grade)
 8.     {
 9.         case 'A':printf("90-100\n");
10.                  break;
11.         case 'B':printf("80-89\n");
12.                  break;
13.         case 'C':printf("70-79\n");
14.                  break;
15.         case 'D':printf("60-69\n");
16.                  break;
17.         case 'E':printf("0-59\n");
18.                  break;
19.         default:printf("错误!\n");
20.     }
21. }
```

【例 3-9】 分析下面 switch 语句的执行过程。

```
switch(a) {
            case 1:
                c1++;
                printf("%d",c1);
                break;
            case 2:
                c2++;
            case 3:
                c3++;
            case 4:
                c4++;
            default:
                c++;
}
```

分析：对这个 switch 语句，根据 a 的值分为下面三种情况讨论。

(1) 如果 a 的值是 1，则 a 的值与“case 1”匹配，这时执行语句“case 1”后面的所有语句“c1++;printf("%d",c1);”，然后跳出 switch 语句。

(2) 如果 a 的值是 2，a 的值与“case 2”匹配，执行“case 2”子句下的语句“c2++;”，由于“case 2”子句的语句段中没有 break 语句，因此接着执行“c3++;c4++;c++”。

(3) 如果 a 的值是 0 或者其他值，由于没有任何 case 子句与 0 匹配，所以执行 default 子句下面的语句段，即执行“c++”。

下面用一个例子说明嵌套选择结构与 switch 多分支语句的比较。

【例 3-10】 某运输公司的运费按运输距离进行分段计费，设 s 表示距离，运费打折情况如下：

s＜250	没有打折
250≤s＜500	2%折扣
500≤s＜1000	5%折扣
1000≤s＜2000	8%折扣
2000≤s＜3000	10%折扣
s≥3000	15%折扣

设每千米每吨的基本运费为 p，货物重量为 w，折扣为 d%，总运费 f 的计算公式为：

$$f=p\times w\times s\times(1-d/100)$$

请设计一个程序，从键盘输入 p，w，s 的值，计算运费并输出。

分析：本题可以用两种方法实现：其一，使用嵌套选择结构；其二，使用 switch 语句。

1. 嵌套条件语句实现

```
#include<stdio.h>
main()
```

```
{
    int s;
    float p,w,d,f;
    printf("请输入运费单价 p,货物重量 w,距离 s");
    scanf("%f%f%d",&p,&w,&s);
    if(s<250) d=0;
    else if(s<500) d=2;
        else if(s<1000) d=5;
            else if(s<2000) d=8;
                else if(s<3000) d=10;
                    else d=15;
    f=p*w*s*(1-d/100);
    printf("运费 f=%f8.2\n",f);    /*%8.2 表示按浮点格式输出,其中
                                       宽度是 8,小数点后保留两位 */
}
```

思考：在程序中,如果写成 if(250＜＝s＜500)可以吗？直接写成 if(s＜500)是否能完成判断功能？

2. switch 语句实现

这里不能直接对 s 使用 switch 语句,由于 s 的一个区间对应一个折扣值,因此应该设法将 s 的一个区间变换成一个整数。仔细观察运费折扣表,可以发现距离的区间变化是以 250 为单位变化的。用 250 去除 s,得到一个整数 c,即 c＝s/250,每一个 c 对应一个运费折扣值。据此分析,可以写出程序如下：

```
#include<stdio.h>
main()
{
    int s,c;
    float p,w,d,f;
    printf("请输入运费单价 p,货物重量 w,距离 s");
    scanf("%f%f%d",&p,&w,&s);
    c=s/250;
    switch(c){
        case 0: d=0; break;
        case 1: d=2; break;
        case 2:
        case 3: d=5; break;
        case 4:
        case 5:
        case 6:
        case 7: d=8; break;
        case 8:
        case 9:
```

```
20.         case 10:
21.         case 11: d=10; break;
22.         default: d=15;
23.     }
24.     f=p*w*s*(1-d/100);
25.     printf("运费 f=%f8.2\n",f);
26. }
```

说明：本题使用整除将一个区间变换成整数，在 switch 多分支选择编程中经常用到这种技巧。

3.3.6 浮点数的相等性比较

浮点数是对数学中实数的一种近似表示。在数学上两个实数 x＝y，由于受浮点数表示精度的影响，不能写成 x＝＝y。比如 x，y 的值从数学上讲都是 3.0，但是，在 C 语言中，当 x，y 是浮点型变量时，其值可能是 x＝3.000001，y＝3.000002，此时关系表达式“x＝＝y”的值是 0(假)。

错误的表示方法：

```
if(x==y)
    …
```

正确的表示方法：

```
if(fabs(a-b)<eps)     //函数 fabs 求浮点数的绝对值，函数原型在头文件 math.h 中
    …
```

这里，fabs 是求浮点数的绝对值的函数，eps 是一个比较小的值，比如可以设置 eps＝1.0e－6。

3.4 循环结构

如何计算 1＋2＋3＋…＋10 的和并输出结果？

读者应该已经想到，将这 10 个数直接加起来，输出即可。

```
void main()
{
    printf("10 个自然数的和=%d",1+2+3+4+5+6+7+8+9+10);
}
```

那么，如何计算 1＋2＋3＋…＋100 这样前 100 个或前 1000 个自然数的和呢？显然继续使用上述的办法已经行不通。

事实上，前 n 个自然数的求和问题可以用公式表示：$S_n＝S_{n-1}＋n$。即，前 n 个自然数的和等于前 $n－1$ 个自然数的和再加上 n。下面给出前几项的求和过程。

初值：　$S_0=0$

n	S_n
1	$S_1=S_0+1=1$
2	$S_2=S_1+2=3$
3	$S_3=S_2+3=6$
4	$S_4=S_3+4=10$
…	

可以看出，这个过程是相同规律的重复计算，每次计算只是加数不同。在程序设计中，涉及大量类似的重复计算，当然，这种重复不是简单重复，而是每次重复计算的数据或状态会发生某些变化。对于重复计算，通常使用循环语句来处理。

C 语言提供了 5 个与循环有关的关键字：while，do，for，break，continue。其中三个关键字 while，do，for 直接用于构造三种主要循环语句，即习惯上的 while 循环，do…while 循环，for 循环。break，continue 用于循环辅助控制。

3.4.1　for 循环结构

for 循环的一般格式：

```
for(表达式 1;表达式 2;表达式 3)
    S;
```

for 语句的执行过程：

(1) 首次进入循环时，执行＜表达式 1＞，＜表达式 1＞的主要作用是给循环变量赋初值。

(2) 判断＜表达式 2＞的值，若其值为非 0，则执行一遍循环体 S，否则结束整个 for 语句的执行。循环体 S 可以是单条语句，也可以是多条语句组成的复合语句，如果是复合语句，需要用一对花括号“{}”将复合语句括起来。

(3) 循环体执行结束后，返回去计算＜表达式 3＞，＜表达式 3＞一般用于对循环变量增加或减少某个数值。

(4) 自动转向第(2)步执行。

【例 3-11】 用 for 循环计算前 n 个自然数的和。

```
#include<stdio.h>
main()
{
    int i,n,s;
    printf("输入自然数 n=");
    scanf("%d",&n);
    s=0;
    for(i=1;i<=n;i=i+1)
    {  //循环体只有一条语句,所以包围该语句的"{}"可以省略
        s=s+i;
    }
```

```
    printf("前%d个自然数的和=%d\n",n,s);
}
```

运行结果：

```
输入自然数 n=99
前 99 个自然数的和=4950
```

程序说明：

(1) n 表示用于求和的自然数的个数。i 控制循环的执行，即 i 是循环控制变量，s 用于表示自然数的和。

(2) 本程序的核心是下面三条语句：

```
s=0;
for(i=1;i<=n;i=i+1)
    s=s+i;
```

(3) 循环语句中，for 后面的括号中有三个表达式，i=1 称为初值表达式，其作用是给循环变量 i 赋初值；i<=n 是条件表达式，如果这个表达式的值为真(非 0)，执行循环体 s=s+i，否则退出循环；i=i+1 是循环增量表达式，其作用是将循环控制变量 i 的值加 1。

(4) 循环体中，s=s+i 的含义是，取出上一次 s 的值，与当前变量 i 的值相加，然后将结果再赋给 s。

3.4.2 while 循环结构

while 语句的一般格式：

```
while(expr)
    S;
```

while 循环的执行过程：

(1) 当 expr 的值为真(非 0)时，执行循环体 S；否则退出循环。

(2) 循环体执行完后，再返回去重新计算 expr 的值。

【例 3-12】 用 while 循环计算前 n 个自然数的和。

```
 1. #include<stdio.h>
 2. main()
 3. {
 4.     int i,n,s;
 5.     printf("输入自然数 n=");
 6.     scanf("%d",&n);
 7.     i=1;
 8.     s=0;
 9.     while(i<=n)
10.     {  //循环体开始
11.        s=s+i;
12.        i=i+1;  //通过改变 i 的值,改变条件表达式 i<=n 的值
```

```
13.     }  //循环体结束,由于循环体中有两条语句,所以一对"{}"不能少
14.     printf("前%d个自然数的和=%d\n",n,s);
15. }
```

运行结果：(略)

3.4.3　do…while 循环结构

do…while 语句的一般格式：

```
do{
    程序块 S;
}while(expr)
```

do…while 循环的执行过程：

(1) 首先执行一次程序块 S。

(2) 然后判断 expr 的值是否为真(非 0)，如果条件表达式 expr 为真，继续返回去执行循环体 S；否则退出循环。

3.4.4　循环辅助语句

C 语言中提供了两个用于循环的辅助控制关键字 break 和 continue，用于特殊情况下的循环结束和退出。关键字 break、continue 后面直接跟一个分号，就构成 break 语句和 continue 语句。这两个语句可用于 while、do…while、for 循环语句中的任何一种。下面以 for 循环语句来说明其功能和用法，如表 3-1 所示。

表 3-1　循环辅助语句的用法

	break 语句	continue 语句

格式	for(expr1;expr2;expr3)
	{
	…;
	S1;
	if(expr)
	break;
	S2;
	…;
	}
	S3;
	…
		for(expr1;expr2;expr3)
		{
		…;
		S1;
		if(expr)
		continue;
		s2;
		…;
		}
		S3;
		…

功能	如果 expr 为真，直接退出当前循环，执行 for 循环的下一条语句 S3。也就是说，如果 expr 为真，执行语句 break，而 break 的作用是跳出循环，执行循环后面的 S3 语句	如果 expr 为真，结束本次循环中后续语句的执行，即忽略 s2 到“}”之间的所有语句，而开始新一轮循环，即重新计算 expr3 和 expr2，如果 exp2 为真，开始新的一轮循环体的执行

【例 3-13】 从键盘输入 n 个小于 100 的正整数，求 n 个正整数的和，如果其和大于等于 200 则结束，并输出整数个数 n 和 n 个整数的和。

分析：

(1) 设用 x 表示输入的整数,S 表示输入的整数的和,n 表示输入的有效整数的个数。

(2) 每次输入一个数据,对 n 加 1,并对 s 进行一次累加。

(3) 在循环体内判断 s 的值,如果 s≥200,退出循环,否则继续循环。由于输入的数据可能不是小于 100 的正整数,所以应该对输入的整数进行合法性判断。

(4) 根据本题的特点,使用 while 循环或者 do…while 循环较好。

```
#include<stdio.h>
main()
{  int n=0,x,s;
   s=0;
   while(s<200){
       printf("输入 100 以内的正整数:")
       scanf("%d",&x);
       if(x<0||x>=100){
           printf("输入数据不合法!\n");
           continue;
       }
       s=s+x;
       n++;
   }
   printf("n=%d , s=%d\n",n,s);
}
```

3.4.5 三种基本循环结构的比较

在下面的表 3-2 中,列出了三种基本循环语句的格式、执行过程及示例代码。通过对比分析,希望读者能够领悟到三种循环语句的各自特点及使用方法。

表 3-2 三种基本循环结构的比较

	while 循环	do…while 循环	for 循环

格式	`while(expr)`
	`  S;`
		`do{`
		`  S;`
		`}while(expr)`
			`for(expr1;expr2;expr3)`
			`  S;`

执行流程