第 3 章　三种基本结构的程序设计

3.1　程序的基本结构及 C 程序中的语句分类

3.1.1　程序的基本结构

计算机程序的一个重要方面就是描述问题求解的计算过程，即计算步骤。在程序设计语言中，一个计算步骤可以用一个基本语句实现也可以用一个控制结构实现。控制结构主要由控制条件和被控制的语句组成，不同的控制结构用于描述不同的控制方式，实现对程序中各种成分语句的顺序、选择和循环等方式的控制。

1966 年，Bohm 和 Jacopini 的研究表明，只需要采用顺序结构、选择结构和循环结构这三种控制结构就能够编写所有程序。

对于一些规模较大而又比较复杂的问题，解决的方法往往是把它们分解成若干个较为简单或基本的问题进行求解。这在程序设计中表现为：将一个大程序分解为若干个相对独立且较为简单的子程序，这些子程序就是过程与函数。大程序通过调用这些子程序来完成预定的任务。过程与函数的引入不仅可以较容易地解决一些复杂问题，而且更重要的是使程序有了一个层次分明的结构，这就是结构化程序设计“自顶向下、逐步求精、模块化”的基本思想。

因此，一个结构化程序是由顺序、选择和循环三种基本结构和过程（函数）结构组成的。结构化程序的开创者 N.Wirth 曾这样说过：“在程序设计技巧中，过程是很少几种基本工具中的一种，掌握了这种工具，就能对程序员工作的质量和风格产生决定性的影响”。N.Wirth 所说的过程就是 C 语言中的函数，我们将在第 5 章介绍，下面只对三种基本结构进行介绍。

（1）顺序结构。顺序结构是按照语句的书写顺序依次执行各语句序列。图 3-1(a)给出了顺序结构。图 3-1(a)中 A 框和 B 框表示基本的操作处理，可以是一个语句也可以是多个语句，它表示程序在执行完 A 框操作后，再去顺序执行 B 框的操作，即严格按照语句的书写顺序进行。因此，顺序结构是一种最基本的程序结构。

（2）选择结构。选择结构是按照条件判断选择执行某段语句序列。图 3-1(b)给出了选择结构。需要指出的是，在选择结构程序中，A 框和 B 框的操作只能二选一，即执行了 A 框操作，就不能再执行 B 框操作；而执行了 B 框操作，就不能再执行 A 框操作。无论是执行了 A 框操作还是执行了 B 框操作，接下来都会继续向下顺序执行后继的操作。

（3）循环结构。循环结构能够通过条件判断控制循环执行某段语句序列。按照条件和循环执行的语句段之间的关系，可以细分为当型循环结构和直到型循环结构。图 3-1(c)和

图 3-1(d)分别给出了当型循环结构和直到型循环结构。在当型循环结构中，需要先判断条件 P，然后执行 A 框操作，若一开始 P 就不成立，则 A 框操作一次也不执行。直到型循环与当型循环的区别是这种循环要先执行 A 框操作，然后再判断条件 P，即在直到型循环中，无论 P 条件是否成立，A 框操作至少会被执行一次。

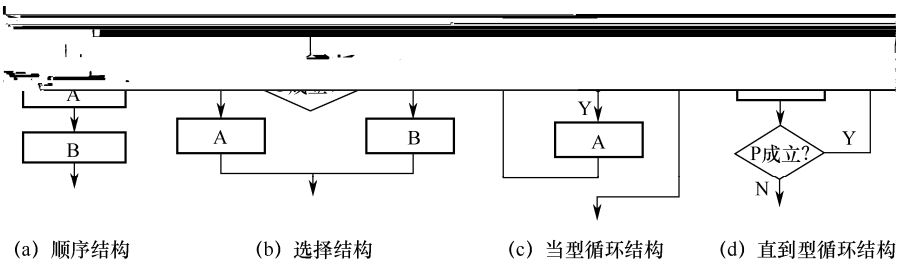

图 3-1　结构化程序设计的三种基本结构

关于三种基本结构有以下几点说明：

（1）无论是顺序结构、选择结构还是循环结构，它们的共同特点是只有一个入口和一个出口，整个程序是由若干个这样的基本结构组合而成的。

（2）三种基本结构中的 A 框和 B 框是广义的，它们可以是一个操作，也可以是另一种基本结构或者几种基本结构的组合。

（3）在选择结构和循环结构中都会出现判断框，但是选择结构会根据条件 P 是否成立来决定执行 A 框和 B 框中的哪一个操作，且执行后就会脱离该选择结构而顺序执行下面的其他结构；即选择结构中的 A 框和 B 框只能选择一个执行且只能执行一次。循环结构是在条件 P 成立时反复执行 A 框操作，直到条件 P 不成立时才跳出该循环结构而顺序执行下面的其他结构。

3.1.2　C 程序中的语句分类

C 语言中的语句分为简单语句和结构语句两类。简单语句是指那些不包含其他语句成分的基本语句；结构语句是指那些“句中有句”的语句，它是由简单语句或结构语句根据某种规则构成的。C 语言的语句分类情况如图 3-2 所示。

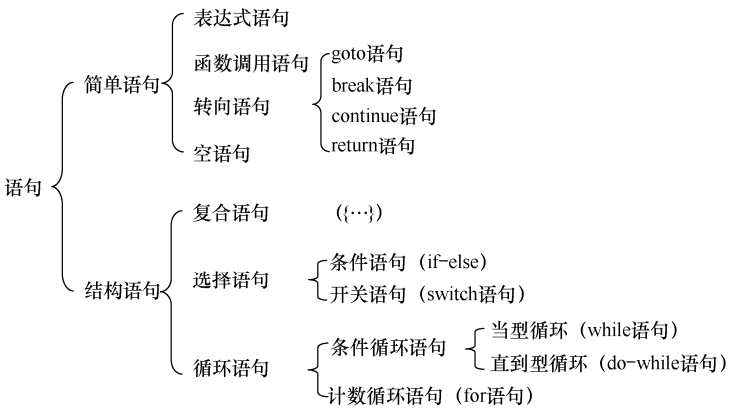

图 3-2　C 语言的语句分类情况

（1）表达式语句。在 C 语言中，由一个表达式加上一个分号“;”就构成了一个表达式语句。最典型的是由赋值表达式加上分号“;”就构成了赋值语句。表达式语句的一般形式为：

```
表达式;
```

例如：

```
i++;
k=k+2;
m=n=j=3;
a=1;
```

按照 C 语言的语法，任何表达式后面加上分号“;”均可构成表达式语句，例如，“x+y;”也是一个 C 语言的语句，但这种语句没有实际意义。一般来说，表达式的执行应能赋予或改变某些变量的值，或者说表达式能产生某种效果才能成为有意义的表达式语句。

（2）函数调用语句。由一个函数调用加一个分号“;”构成函数调用语句，其作用主要是完成该函数指定的操作。函数调用语句的一般形式为：

```
函数名(实际参数表);
```

例如：

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

该语句是由一个 printf 格式输出函数加上一个分号“;”构成的一个函数调用语句。

（3）空语句。仅由一个分号“;”构成的语句就是空语句，其一般形式为：

```
;
```

空语句是不执行任何操作的语句。C 语言引入空语句出于以下考虑：

① 为了构造特殊控制结构的需要。例如，循环控制结构的语法上需要一个语句作为该循环语句的循环体（这种结构语句必须“句中有句”）；当要循环执行的动作已经由循环控制部分完成时，就不再需要循环体语句了，但是为了满足结构语句这种“句中有句”的要求，此时就必须用一个空语句作为循环体。

② 在复合语句的末尾设置一个空语句作为转向的目标位置，以便 goto 语句能够将控制转移到复合语句的末尾。

（4）复合语句。用一对花括号“{ }”括起来的若干个语句称为复合语句，复合语句在语法上相当于一个语句（即从外部看一个复合语句就相当于一个语句）。复合语句的一般形式为：

```
{
    语句 1;
      ⋮
    语句 n;
}
```

需要注意的是，复合语句内的各个语句都必须以分号结束，并且在复合语句的标识“}”外不能加分号“;”。

（5）控制语句。控制语句用来规定语句的执行顺序。C 语言有如下 9 种控制语句：

① if…else，条件语句。

② switch，多分支选择语句，又称开关语句。

③ while，循环语句。
④ do…while，循环语句。
⑤ for，循环语句。
⑥ continue，结束本次循环。
⑦ break，退出循环或 switch 语句。
⑧ goto，转移语句。
⑨ return，返回语句。
这些语句的使用方法将在以后的章节中介绍。

3.2　顺序结构程序设计

3.2.1　赋值语句

顺序结构的程序在第 2 章中已多次出现，其中出现的函数调用语句（如 printf 和 scanf）也已在第 2 章中介绍过。下面，我们介绍顺序结构中出现的赋值语句。

赋值语句是由赋值表达式与分号“;”构成的，赋值语句的功能和特点都与赋值表达式相同，它是程序中使用最多的语句之一。赋值语句的一般形式为：

```
变量=表达式;
```

例如：

```
a=b+3;
```

与赋值表达式相同，赋值运算符“=”的左侧是变量而不能是常量或表达式。并且，赋值语句可以写成下面的形式：

```
变量=变量= …=变量=表达式;
```

它表示将最右侧的表达式逐一赋给赋值运算符左侧的每一个变量。例如：

```
a=b=c=d=10;
```

C 语言中有赋值表达式和赋值语句的概念，两者只差一个分号“;”，而其他大多数高级语言没有“赋值表达式”这个概念。因此，赋值表达式可以包含在其他表达式之中。例如：

```
if((a=b)>0)
    x=a;
```

按大多数语言的语法规定，if 后面的括号内是一个条件，如“if(x>0) …”，而在 C 语言中，这个 x 的位置可以是一个赋值表达式，如“a=b”；其作用是：先进行赋值运算（即先将 b 的值赋给 a）然后再判断 a 是否大于 0，若大于 0，则执行 x=a（if 语句的使用参考第 3.3 节的内容），但是在 C 语言中，像 if、while 和 do 语句的圆括号“( )”中一定是表达式，而不能是一个语句，如下面形式的语句是错误的。

```
if((a=b;)>0)
    x=a;
```

C 语言把赋值语句和赋值表达式区分开来，增加了表达式的种类，因此能够实现其他高级语言难以实现的功能。例如：

```
if((ch=getchar())=='\n');
```

这个语句的作用是：先从键盘输入一个字符赋给变量 ch，然后判断 ch 是否等于换行符'\n'，若等于换行符'\n'，则什么也不做。

此外要注意的是，必须清楚在变量定义中给变量赋初值和赋值语句的区别。给变量赋初值是变量说明的一部分，即必须一个变量接一个变量地定义，各变量之间（包括赋初值的变量和不赋初值的变量）必须用逗号“,”隔开。

例如：

```
int a=10,b;
```

但不允许在变量定义时连续使用赋值符号给多个变量赋初值。如下面的变量定义方式是错误的：

```
int a=b=c=10;
```

应该写成

```
int a=10,b=10,c=10;
```

而赋值语句则允许连续赋值。例如：

```
int a, b, c;
a=b=c=10;
```

3.2.2 顺序结构程序

顺序结构的程序基本上是由函数调用语句和表达式语句构成的，这种结构的程序在执行中的特点是：一个操作执行完成后就接着执行紧随其后的下一个操作。由于顺序结构非常简单，因此其求解的问题种类是有限的。

【例 3.1】输入三角形的三条边长，求三角形的面积。

解：已知三角形的三条边长分别为 a、b 和 c，则三角形的面积可用下面的公式求出：

$$p=\frac{1}{2}(a+b+c) \qquad s=\sqrt{p(p-a)(p-b)(p-c)}$$

程序如下：

```
#include<stdio.h>
#include<math.h>
void main()
{
    float a,b,c,p,s;
    printf("Input a,b,c=");
    scanf("%f,%f,%f",&a,&b,&c);
    p=1.0/2*(a+b+c);
    s=sqrt(p*(p-a)*(p-b)*(p-c));
    printf("s=%6.2f\n",s);
}
```

运行结果：

```
Input a,b,c=3,4,5↙
s=  6.00
```

该程序要注意以下两点：

（1）由于程序中使用了 C 语言的库函数 sqrt 来求平方根，因此必须在程序开始处用

include 命令给出所使用库函数的说明。include 命令必须以“#”开头，所说明的库函数文件名以“.h”作为其后缀，且该文件名用一对尖括号“<>”或一对双引号“" "”括起来。由于以#include 开头的命令行不是语句，因此其末尾不加分号“;”，在此使用的库函数为数学函数 math.h。

（2）求 p 值时的“1/2”在程序中必须写成“1.0/2”，若写为“1/2”则因两个整数相除后将舍去结果的小数部分而仅保留结果的整数部分，这样“1/2”的结果为 0，而“1.0/2”则是一个单精度数和一个整数相除，其结果为单精度数，故不受影响。这一点在编写程序中要尤为注意，否则会产生很大的误差。

【例 3.2】从键盘上输入 *a* 和 *b* 的值，然后交换它们的值并输出交换后的 *a* 和 *b* 的值。

解：在计算机中进行数据交换，如交换变量 a 和 b 的值，则不能简单地通过下面两条赋值语句实现：

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

因为当执行第 1 条赋值语句“a=b;”后，将变量 b 的值送入变量 a 的内存单元而覆盖了变量 a 原有的值，即 a 的原值已经丢失，此时已具有变量 b 的值（a 和 b 中都保存着 b 值）。接下来再执行第 2 条赋值语句“b=a;”，则将 a 中所保存的 b 值又送回到变量 b 的内存单元中，这样就无法实现将两个变量值相互交换的目的。因此，必须借助一个中间变量（如下面的 t）来实现 a、b 值交换的目的，其实现过程是利用连续三个赋值语句：

```
t=a;
a=b;
b=t;
```

即执行“t=a;”后将 a 值保存在 t 中，再执行“a=b;”将 b 值赋给 a（此时 a 中已为 b 值），最后执行“b=t;”将 t 中所保存的原 a 值赋给 b，即实现了 a 与 b 值的交换。图 3-3 给出变量 a 与 b 之间数据交换的过程。

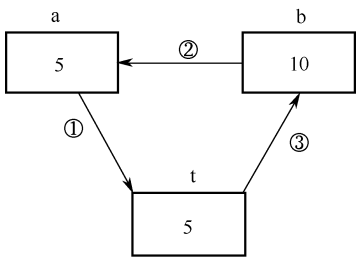

图 3-3　变量 a 与 b 之间数据交换的过程

程序如下：

```
#include<stdio.h>
void main()
{
    int a,b,t;
    printf("Input a,b=");
    scanf("%d,%d",&a,&b);
    printf("old data: a=%d,b=%d\n",a,b);        //输出变量 a 与 b 的原值
    t=a;
    a=b;
    b=t;                                        //实现变量 a 与 b 值的交换
    printf("new data: a=%d,b=%d\n",a,b);        //输出交换后 a 与 b 的新值
}
```

运行结果：

```
Input a,b=5,10↙
```

```
old data: a=5,b=10
new data: a=10,b=5
```

3.3 选择结构程序设计

选择结构通过选择语句实现。选择语句是根据是否满足条件来选择所应执行的语句，进而控制程序的执行顺序。选择语句共有两个：一个是 if 语句，另一个是 switch 语句。这两个语句用来实现程序的选择结构。

3.3.1 if 语句

if 语句是 C 语言中用来实现选择结构的重要语句，它根据给定的条件进行判断来决定执行某个分支语句（可以是复合语句）。C 语言的 if 语句有三种基本形式。

1. 单分支 if 语句

单分支 if 语句的一般形式为：

```
if(表达式)
    语句;
```

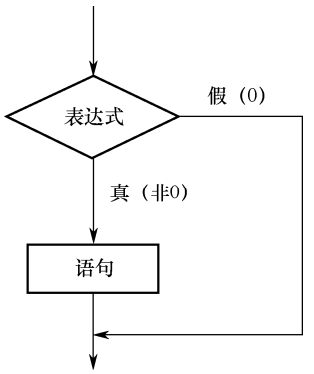

图 3-4　单分支 if 语句的执行流程

单分支 if 语句的功能首先是计算表达式的值，若表达式的值为非 0（即为真）则执行语句；若表达式的值为 0（即为假）则该 if 语句不起作用（相当于一个空语句），继续执行其后的其他语句。单分支 if 语句的执行流程如图 3-4 所示。

例如：

```
int a=5, b=3;
if(a==b) printf("a=b");
if(3) printf("OK! ");
if('a') printf("%d",'a');
```

以上语句都是合法的。第 1 个 if 语句因表达式“a==b”的值为“假”而相当于一个空语句；第 2 个 if 语句因表达式的值为 3（即非 0）按“真”处理，即输出“OK!”；第 3 个 if 语句的表达式为字符'a'（非 0）也按“真”处理，即输出'a'的 ASCII 码值 97。

【例 3.3】输入任意两个整数，并按由大到小的顺序输出。

解：程序如下：

```
#include<stdio.h>
void main()
{
    int a,b,t;
    printf("Input a,b=");
    scanf("%d,%d",&a,&b);
    if(a<b)                    //若 a<b 则交换 a 和 b 的值
    {
        t=a;
```

```
            a=b;
            b=t;
        }
        printf("%d,%d\n",a,b);
    }
```

运行结果：

```
    Input a,b=5,10↙
    10,5
```

2．双分支 if 语句

双分支 if 语句的一般形式为：

```
    if(表达式)
        语句 1;
    else
        语句 2;
```

双分支 if 语句的功能是：若表达式的值为非 0（即为真）则执行语句 1；否则执行语句 2。双分支 if 语句的执行流程如图 3-5 所示。

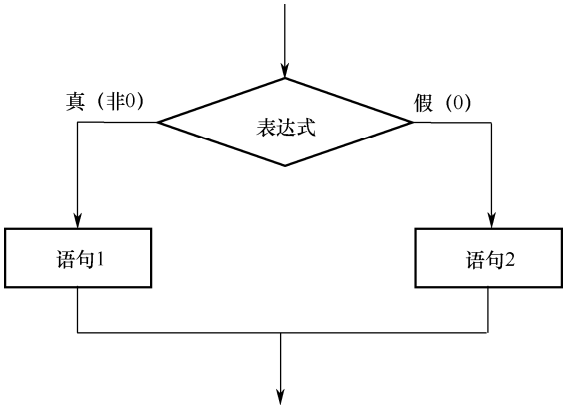

图 3-5　双分支 if 语句的执行流程

【例 3.4】输入任意两个整数，并按由大到小的顺序输出。

解：程序如下：

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

运行结果：

```
    Input a,b=5,10↙
    10,5
```

【例 3.5】判断下面的（1）和（2）是否等效。

```
    (1) if(a>0&&b>0)  a=a+b;
        if(a<=0&&b<=0)  b=a-b;
    (2) if(a>0&&b>0)  a=a+b;
          else  b=a-b;
```

解：当 a>0 并且 b>0 时，（1）和（2）等效，因为两者执行的都是“a=a+b;”语句，而其他语句均不执行。但是，当 a>0 或 b>0 这两个条件中有一个不满足时，（1）和（2）就

不等效了，这是因为（1）中的第 2 个 if 语句是在 a≤0 和 b≤0 同时满足的条件下才执行“b=a−b;”语句，而（2）的 else 是当 a≤0 和 b≤0 中有一个条件满足时就执行“b=a−b;”语句。即（2）中 else 后面语句执行的条件范围比（1）中第 2 个 if 语句执行的条件范围宽，所以（1）和（2）不等效。

3．多分支 if 语句

多分支 if 语句是通过多个双分支 if 语句的复合来实现多分支功能的。多分支 if 语句的一般形式为：

```
if(表达式 1)
    语句 1;
else if(表达式 2)
        语句 2;
    else if(表达式 3)
            语句 3;
            …
        else if(表达式 n)
                语句 n;
            else
                语句 n+1;
```

多分支 if 语句的功能是：依次判断每个表达式的值，若某个表达式 *n* 的值为真（非 0），则执行语句 *n*，然后结束整个多分支 if 语句的执行，接下来执行其后的其他语句；若所有表达式的值都为假（即为 0），则执行语句 *n*+1。多分支 if 语句的执行流程如图 3-6 所示。

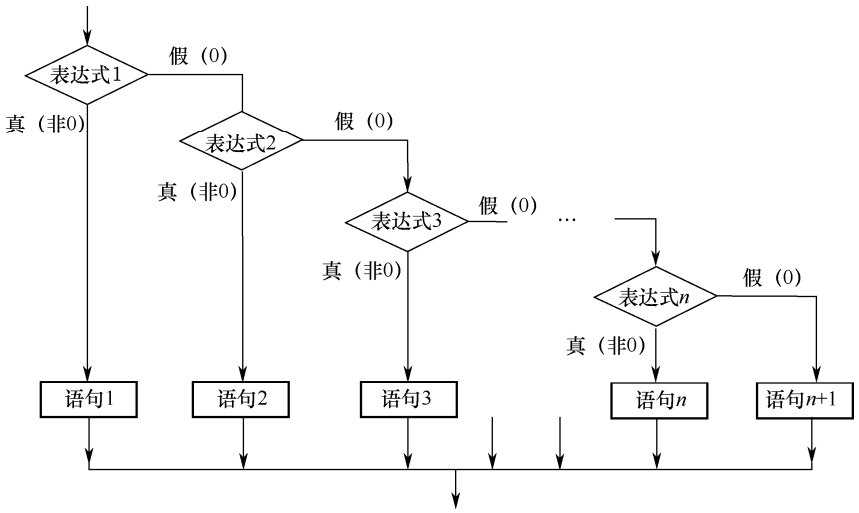

图 3-6　多分支 if 语句的执行流程

【例 3.6】判断由键盘输入的字符类型。

解：判断键盘输入的字符类型，可以根据附录 1 的 ASCII 码表来判断，即 ASCII 码值

小于 32 时为控制字符，在'0'～'9'之间为数字字符，在'A'～'Z'之间为大写英文字母，在'a'～'z'之间为小写英文字母，其余则为其他字符。这是一个多分支的选择问题，因此用多分支 if 语句实现。程序如下：

```
#include<stdio.h>
void main()
{
    char c;
    printf("Input a character:");
    c=getchar();
    if(c<32)
        printf("This is a control character.\n");
    else
        if(c>='0'&&c<='9')
            printf("This is a digit.\n");
        else
            if(c>='A'&&c<='Z')
                printf("This is a capital letter.\n");
            else
                if(c>='a'&&c<='z')
                    printf("This is a small letter.\n");
                else
                    printf("This is another character.\n");
}
```

在 if 语句的使用过程中应注意以下几点：

（1）if 语句中的表达式一般为表达式或逻辑表达式，C 语言在判断表达式时，只要表达式的值不为 0 就认为是真，只有表达式的值为 0 时才认为是假。因此表达式可以是任意类型的表达式（如整型、实型、字符型和指针类型的表达式等），这是 C 语言与其他高级语言的不同之处。例如：

```
if(c=getchar( ))
    printf("%c",c);
```

即输入一个字符并赋给变量 c，只要 c 值不等于 0（为真）就输出所输入的字符。

（2）分号“;”是语句的标志，因此 else 之前的语句必须有分号“;”。例如，下面的 if 语句是错误的：

```
if(a>b)
    printf("a>b")
else
    printf("a<b");
```

（3）分支语句可以是一个语句，也可以是多个语句复合而成的一个语句。若条件成立或不成立所需执行的语句不止一个，则必须使用复合语句。例如，当 a>b 时需交换 a 和 b 值，对应的 if 语句写成如下形式：

```
if(a>b)
    t=a;
    a=b;
    b=t;
```

这种形式是错误的。因为属于 if 语句范围的仅是一个“t=a;”语句，而“a=b;b=t;”是 if 语句的后继语句。交换 a 值和 b 值正确的操作是：当条件“a>b”为真时，“t=a;a=b;b=t;”三个语句都执行；当条件“a>b”为假时，“t=a;a=b;b=t;”三个语句都不执行。而上面 if 语句的实际执行则是：当条件“a>b”为真时，“t=a;a=b;b=t;”三个语句都将执行，虽然结果正确，但后两个语句“a=b;b=t;”并不是 if 语句中的部分，而是作为 if 语句的后继语句来执行的；当条件“a>b”为假时，if 语句相当于一个空语句，即语句“t=a;”并不执行，这时仍执行 if 语句的后继语句“a=b;b=t;”，执行这两条本不该执行的语句将得到错误的结果。因此，正确的写法应该是：

```
if(a<b)
{
    t=a;
    a=b;
    b=t;
}
```

（4）在 if 语句表达式的括号“( )”之后不能加分号“;”。例如，当 a>b 时，输出“a>b”，对应的 if 语句写式如下形式：

```
if(a>b);
    printf("a>b");
```

这种形式是错误的。因为当表达式 a>b 为真时执行的语句为空语句“;”，而“printf("a>b");”语句则是 if 语句的后继语句，即无论 a>b 是否为真都执行 printf 语句输出同一个结果：a>b，所以该 if 语句失去了判断的意义。

3.3.2 if 语句的嵌套

当 if 语句中的内嵌语句是一个或多个 if 语句时，就形成了 if 语句的嵌套。下面就给出了三种不同的 if 语句嵌套形式：

```
(1) if (表达式 1)
       if(表达式 2) 语句 1;
       else  语句 2;
    else
       if(表达式 3) 语句 3;
       else  语句 4;
(2) if (表达式 1)
       if(表达式 2) 语句 1;
       else  语句 2;
    else  语句 3;
(3) if (表达式 1) 语句 1;
    else
       if(表达式 2) 语句 2;
       else  语句 3;
```

注意：else 总是与它之前最近的尚未与 else 匹配的那个 if 配对，这就是 else 的“就近匹配”原则。若想要 else 不遵循该原则，则可用花括号“{ }”来改变匹配关系。

例如：

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

此时，else 与第 1 个 if 匹配，若没有花括号“{ }”，则这个 else 与第 2 个 if 匹配。

【例 3.7】以下程序运行的结果是_____。

```
#include<stdio.h>
void main()
{
    int x=1,y=2,z=3;
    if(x>y)
        if(y<z)
            printf("%d",++z);
        else
            printf("%d",++y);
    printf("%d\n",x++);
}
```

A．331　　B．41　　C．2　　D．1

解：由于 else 总是与在它之前且离它最近的那个 if 匹配，因此程序中的 else 与第 2 个 if 匹配。程序执行时首先判断“x>y”，由于 x 等于 1 而 y 等于 2 即“x>y”的结果为假，所以跳过内嵌的第 2 个 if-else 语句而直接执行最后一个输出语句“printf("%d\n",x++)”，即先输出 x 的值，然后 x 再增 1，故输出的结果为 1，故选 D。

【例 3.8】闰年的判断方法：若某年（以阳历表示）是 4 的倍数而不是 100 的倍数，或者是 400 的倍数，则这一年是闰年。试编写判别闰年的程序。

解：由题意可知，闰年首先能够被 4 整除（即用 4 取余为 0）；在被 4 整除的年份中显然含有被 100 整除的年份，但也不能将这些被 100 整除的年份统统排除在闰年以外，因为其中能够被 400 整除的仍然是闰年，所以在能被 4 整除同时又能被 100 整除的年份中，找出能被 400 整除的那些闰年来。程序如下：

```
#include<stdio.h>
void main()
{
    int y,b;
    printf("Input year:");
    scanf("%d",&y);
    if(y%4==0)
        if(y%100==0)
            if(y%400==0)
                b=1;        //能被 4 整除、能被 100 整除又能被 400 整除
            else
                b=0;        //能被 4 整除、能被 100 整除但不能被 400 整除
        else
            b=1;            //能被 4 整除但不能被 100 整除
    else
        b=0;                //不能被 4 整除
```

```
        if(b)
            printf("%d is a leapyear.\n",y);
        else
            printf("%d is not a leapyear.\n",y);
    }
```

此题也可将题设条件用一个布尔表达式来描述，能够被 4 整除且不能被 100 整除或者能够被 400 整除的年份是闰年。由此得到程序如下：

```
    #include<stdio.h>
    void main()
    {
        int y;
        printf("Input year:");
        scanf("%d",&y);
        if((y%4==0&&y%100!=0)||y%400==0)
            printf("%d is a leapyear.\n",y);
        else
            printf("%d is not a leapyear.\n",y);
    }
```

运行结果：

```
    Input year:2010↙
    2010 is not a leapyear.
```

3.3.3 条件运算符与条件表达式

若条件语句 if 中只执行单个赋值语句，则可以用条件表达式取代 if 语句。条件表达式是包含条件运算符的表达式；条件运算符是 C 语言中唯一的三目运算符，即有三个参与运算的量。条件表达式的一般形式为：

```
    表达式 1 ? 表达式 2 : 表达式 3
```

其求值规则为：先求解表达式 1 的值，若表达式 1 的值为真（非 0），则表达式 2 的值即为整个条件表达式的值；否则表达式 3 的值即为整个条件表达式的值。例如，10>8?5:15 的值是 5 而 10<8?5:15 的值是 15。

对于条件语句：

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

可用条件表达式语句（条件表达式后加分号“;”即构成条件表达式语句）写为：

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

使用条件表达式应注意以下几点：

（1）条件运算符的优先级高于赋值运算符而低于关系运算符和算术运算符。例如：

```
    max=(a>b)?a:b+1;
```

可以去掉括号而写为：

```
    max=a>b?a:b+1;
```

（2）条件运算符的结合方向是自右至左的。例如：

```
    a>b?a:c>d?c:d;
```

相当于

```
a>b?a:(c>d?c:d);
```

这也是条件表达式嵌套的情况，即其中的表达式 3 又是一个条件表达式（当然表达式 2 也可以是一个条件表达式）。

（3）条件表达式不能取代一般的 if 语句，只有 if 语句内嵌的语句为赋值语句且两个分支都给同一个变量赋值时才能代替 if 语句。

（4）在条件表达式中，表达式 1 的类型可以与表达式 2 和表达式 3 的类型不一致。例如：

```
a?'x':'y'
```

（5）条件运算符“?”和“:”是一对运算符，不能拆开单独使用。

【例 3.9】输入一个字符，若是小写英文字母，则转换成对应的大写英文字母；若是大写英文字母则保持不变。

```
#include<stdio.h>
void main()
{
    char ch;
    printf("Input one char:");
    ch=getchar();
    ch=ch>='a'&&ch<='z'?ch-32:ch;
    putchar(ch);
    putchar('\n');
}
```

运行结果：

```
Input one char:h↙
H
```

3.3.4　switch 语句

if 语句本质上是两路分支的选择结构，若要用于多路分支，则 if 语句就必须采用嵌套形式，这使程序的可读性降低。对于多路分支问题，C 语言提供了更加简练的语句，即可以直接使用多分支选择语句 switch 来实现多种情况的选择。switch 语句的一般形式如下：

```
switch(表达式)
{
    case 常量表达式 1: 语句 1;
    case 常量表达式 2: 语句 2;
                ⋮
    case 常量表达式 n: 语句 n;
    default: 语句 n+1;
}
```

switch 语句的执行过程是：首先计算表达式的值，并逐个与 case 后面的常量表达式的值相比较，若表达式的值与某个常量表达式的值相等，则执行该常量表达式后面的语句，此后遇到下面其他 case 后的语句就不再判断（包括 default 后面的语句），即顺序向下执行直到遇到 break 语句则跳出 switch 语句，或执行到 switch 语句的结束标志“}”处；然后继续执行 switch 语句的后继语句。若表达式的值与所有 case 后面的常量表达式值均不相等，

则执行 default 后面的语句，若没有 default 部分，则此时 switch 语句相当于一个空语句。例如：

```
switch(class)
{
   case 'A': printf("GREAT!\n");
   case 'B': printf("GOOD!\n");
   case 'C': printf("OK!\n");
   case 'D': printf("NO!\n");
   default: printf("ERROR!\n");
}
```

若 class 的值为'B'，则输出结果是：

```
GOOD!
OK!
NO!
ERROR!
```

若 class 的值为'D'，则输出结果是：

```
NO!
ERROR!
```

因此，switch 语句的功能是：根据 switch 后面表达式的值找到匹配的入口，然后从这个入口开始执行下去且不再进行判断。因此，为了保证只执行一条分支上的语句，就要在每个分支语句结束处都增加一个 break 语句来强制终止 switch 语句的执行（最后一个分支语句处可不加 break 语句），即跳出 switch 语句。例如：

```
switch(class)
{
   case 'A': printf("GREAT!\n"); break;
   case 'B': printf("GOOD!\n"); break;
   case 'C': printf("OK!\n"); break;
   case 'D': printf("NO!\n"); break;
   default: printf("ERROR!\n");
}
```

这样，若 class 的值为'B'，则输出结果是：

```
GOOD!
```

注意：在 switch 语句“}”之前的语句后面可不加 break 语句。

使用 switch 语句应该注意以下几点：

（1）switch 后面常量表达式的类型可以是整型、字符型或枚举型，但不能是其他类型。如单精度型和双精度型的值由于存在计算误差而难以进行相等比较，可强制转换为整型后再进行相等比较。

（2）常量表达式的类型应与 switch 后“()”中的表达式类型一致。

（3）case 后面常量表达式的值必须互不相同，否则将出现多个入口的错误。例如，下面的 switch 语句是错误的。

```
switch(x)
{
   case 2+3: 语句 i;
             ⋮
```

```
        case 8-3: 语句 j;
                ⋮
    }
```

（4）多个 case 可以共享一组执行语句。例如：

```
    switch(ch)
    {
        case 'A':
        case 'B':
        case 'C':
        case 'D': printf("Pass!\n");
            ⋮
    }
```

当 ch 的值为'A'、'B'、'C'或'D'时都会执行“printf("Pass!\n");”语句。

（5）switch 结构可以嵌套，即在一个 switch 语句中可以嵌套另一个 switch 语句，但要注意 break 语句只能跳出当前层的 switch 语句。例如：

```
    int x=1,y=0;
    switch(x)
    {
    case 1: switch(y)
        {
            case 0: printf("x=1,y=0\n");
                 break;
            case 1: printf("x=1,y=1\n");
         }
    case 2: printf("x=2\n");
    }
```

运行结果：

```
    x=1,y=0
    x=2
```

本来不应该再输出“x=2”。这是因为 break 语句仅结束了内层 switch 语句，由于外层的“case 1”语句后无 break 语句，因此继续执行“case 2”后的语句，正确的写法如下：

```
    int x=1,y=0;
    switch(x)
    {
    case 1: switch(y)
        {
            case 0: printf("x=1,y=0\n");
                 break;
            case 1: printf("x=1,y=1\n");
                      }
         break;
    case 2: printf("x=2\n");
    }
```

【例 3.10】用数字 1～7 代表从星期一到星期日，根据键盘上输入的数字，输出该数字所代表星期几的英文单词。

解：程序如下：

```
#include<stdio.h>
void main()
{
   int a;
   printf("Input data:");
   scanf("%d",&a);
   switch(a)
   {
      case 1: printf("Monday\n");
           break;
      case 2: printf("Tuesday\n");
           break;
      case 3: printf("Wednesday\n");
           break;
      case 4: printf("Thursday\n");
           break;
      case 5: printf("Friday\n");
           break;
      case 6: printf("Saturday\n");
           break;
      case 7: printf("Sunday\n");
           break;
      default: printf("Input error!\n");
   }
}
```

运行结果：

```
Input data:4↙
Thursday
```

【例 3.11】输入 2 个运算量及 1 个运算符，如 5+3，用程序实现四则运算并输出运算结果。

解：首先输入参加运算的 2 个数和 1 个运算符，然后根据运算符做相应的运算。但是在做除法运算时应先判别除数是否为 0，若为 0，则运算非法，给出错误提示。若运算符号不是“+”“–”“*”或“/”，则运算同样非法，也给出错误提示，对其他情况则输出运算结果。程序如下：

```
#include<stdio.h>
void main()
{
   float a,b,result;
   int flag=0;                                  //0 为合法，1 为非法
   char ch;
   printf("Input expression:a+(-、*、/)b:\n");
   scanf("%f%c%f",&a,&ch,&b);
   switch(ch)                                   //根据运算符进行相关运算
   {
```

```
            case '+': result=a+b;
                  break;
            case '-': result=a-b;
                  break;
            case '*': result=a*b;
                  break;
            case '/': if(!b)
                 {
                    printf("divisor is zero!\n");      //显示除数为 0
                    flag=1;                            //置非法标志
                 }
                 else
                    result=a/b;
                  break;
            default: printf("Input error!\n");         //显示输入错误
                  flag=1;                              //置非法标志
        }
        if(!flag)                                      //若合法则输出计算结果
            printf("%f %c %f=%f\n",a,ch,b,result);
    }
```

运行结果：

```
Input expression:a+(-、*、/)b:
8+3↙
8.000000 + 3.000000=11.000000
```

3.4　循环结构程序设计

程序的循环结构是用循环语句实现的。程序中有时需要反复执行某段语句序列，这个语句序列我们称之为循环体。每次循环前都要做出是继续执行循环体还是退出循环的判定，这个循环终止条件的判定是由表达式来完成的。所以，循环语句至少要包含循环体和判定循环终止条件的表达式这两个部分。

循环语句分为两种类型：一种是条件循环语句，包括当型（while）和直到型（do…while）两种形式的循环语句；另一种是计数（for）循环语句。若能预先确定循环的次数则使用 for 循环语句；否则应使用 while 或 do…while 循环语句。虽然 C 语言已经将 for 语句的功能扩展到足以取代 while 和 do…while 语句的地步，但是从程序的易读性考虑，最好是根据具体情况来选择使用这三种循环语句。

3.4.1　while 语句

while 语句用来实现当型循环，其一般形式为：

```
while (表达式)
    语句;
```

其中，表达式是循环条件，语句为循环体。在执行 while 语句时，先对表达式的循环条件

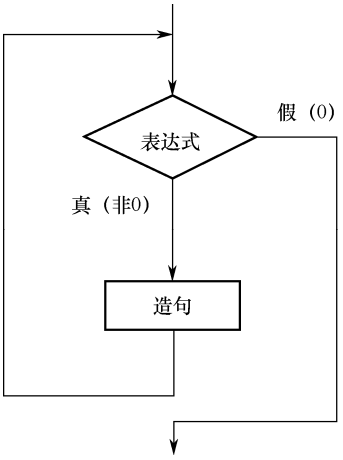

图 3-7　while 语句的执行流程

进行计算，若其值为真（非 0），则执行循环体中的语句；然后继续重复刚才表达式值的计算并判断，若为真则再次执行循环体语句，直到表达式的值为假（为 0），循环结束，程序转至 while 循环语句之后的下一个语句继续执行。while 语句的执行流程如图 3-7 所示。

使用 while 语句时应注意以下几点：

（1）循环体是一个语句。若循环体由多个语句组成，则必须用花括号“{}”括起来的复合语句表示。

（2）循环体内一定要有使表达式（循环条件）的值为假（即 0）的操作，否则循环将永远进行下去而形成死循环。

（3）while 语句中的表达式一般是关系表达式或逻辑表达式，但也可以是数值表达式或字符表达式，只要其值非 0 就执行循环体语句。

（4）while 语句的特点是“先判断，后执行”，若表达式的值一开始就为 0，则循环体语句一次也不执行（相当于一空语句）。但是要注意的是：由于要先判断，因此表达式至少要执行一次。

【例 3.12】分析下面程序的运行结果。

```
(1) #include<stdio.h>
    void main()
    {
        int x=2;
        while(x--)
            printf("%d\n",x);
    }
(2) #include<stdio.h>
    void main()
    {
        int x=2;
        while(x--);
        printf("%d\n",x);
    }
```

```
(3) #include<stdio.h>
    void main()
    {
        int x=2;
        while(x)
            printf("%d\n",x);
    }
(4) #include<stdio.h>
    void main()
    {
        int x=0;
        while(x--)
            x--;
        printf("%d\n",x);
    }
```

解：程序（1）中的“x--”是 x 先参与操作然后再减 1。因此，while 语句中的表达式“x--”就是先判断 x 值是否为 0，若非 0 则执行循环体语句 printf；若为 0 则结束 while 语句的循环。但是在判断后且在执行循环体语句 printf 或者结束 while 循环语句之前，还应执行 x 的自减 1 操作。即该 while 语句的执行步骤如下：

① 对 x 值进行判断；

② x 自减 1；

③ 根据①的判断结果：若非 0 则执行循环体语句 printf 然后转到①；否则结束 while 循环。

因此每次判断 x 不等于 0 时都要执行这个 printf 语句（当然在执行 printf 语句前 x 先减

1)。若 x 等于 0（判断后 x 仍要减 1）则不再执行 printf 语句而结束循环。因此，程序的执行结果如下：

```
1
0
```

程序（2）中的 while 语句的表达式与程序（1）的相同，而循环体语句是一个空语句（分号“;”），即什么也不执行。因此每判断 x（判断后再减 1）一次只要其值非 0 则执行一次循环体，即空语句；当 x 为 0 时，先判断再减 1，若判断 x 值为 0 则意味着要结束循环，并且判断后 x 减 1 则使 x 值为−1。所以，结束循环后的 printf 语句（它不属于循环体语句）输出值为−1。

程序（3）中的 while 语句的表达式为 x，其循环体语句与程序（1）的相同，即每次判断 x 不等于 0 时都要执行这个 printf 语句，由于表达式和循环体语句中都没有对 x 值进行修改，即 x 值始终为 2，因此每次判断 x 值都为非 0，所以该程序的 while 循环是一个死循环，即无休止的输出 2。

程序（4）中的 x 初值为 0，即 while 语句的表达式在判断 x 值时已为 0，即不执行循环体语句“x−−;”，并结束循环。但表达式“x−−”则是先判断后减 1，因此循环结束后由 printf 语句输出−1。

【例 3.13】利用下面公式求 π 值，要求 n=10000。

$$\frac{\pi}{4}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots+\frac{1}{4n-3}-\frac{1}{4n-1}+\cdots$$

解：本题是求累加和问题，可用 while 语句实现。但是在循环体语句中应分解为两步实现求累加和的任务：① 计算通项 $\frac{1}{4n-3}-\frac{1}{4n-1}$（$n=1,2,\cdots,10000$）的值存于 m；② 用语句“sum=sum+m;”求累加和。程序如下：

```
#include<stdio.h>
void main()
{
    int n=1;
    float m,sum=0;
    while(n<=10000)
    {
        m=1.0/(4*n-3)-1.0/(4*n-1);      //求通项的值
        sum=sum+m;                      //求累加和
        n++;
    }
    printf("PI=%f\n",4*sum);
}
```

运行结果：

```
PI=3.141384
```

该程序要注意以下两点：

（1）变量 sum 和 n 的初始值：sum 必须先置 0，否则 sum 是一个随机数，若不置 0 则结果必然出错；设置 n 初始值为 1，这是求通项的需要。

（2）在求通项的语句中，绝不能写成“m=1/(4*n−3)−1/(4*n−1);”，这样，两个整数相