CHAPTER 3

C++语言的控制结构　　第3章

程序是由一条条语句组成的，执行时从前到后（或说从上到下）顺序执行。但是，一个程序在执行过程中并不是每条语句都需要被执行到的。例如，编写一个判断整数奇偶性的程序，要求输入一个整数，如果是奇数，就输出“It's odd.”；如果是偶数，就输出“It's even.”。显然，该程序里会有两条输出语句：一条输出“It's odd.”；一条输出“It's even.”。在程序的一次运行中，这两条语句不可能都被执行，而是要根据输入的整数，选择一条语句来执行。这就要求程序设计语言提供根据某个条件来选择执行和不执行某些语句的机制，这个机制称为选择结构。

有时，还常常需要在程序中成千上万次地执行同样的操作。例如，要求编写一个程序，依次输出1到10 000的所有整数。不可能用编写10 000条cout语句的笨办法来实现，合理的一种做法是让一个变量的初始值等于0，然后反复执行以下操作10 000次：

```
该变量值增加1
输出该变量的值
```

程序设计语言中都会有将某些语句重复执行若干次的机制，这个机制称为循环结构。

一种程序设计语言中，只要有顺序执行结构、选择结构和循环结构，就能够用来编程解决各种问题了。

3.1　用if语句实现选择结构

在C++语言中，可以用if语句实现选择结构。if语句用法如下：

```
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 2
}
…
else if(表达式 n-1) {
    语句组 n-1
}
else {
    语句组 n
}
```

上述的“语句组”由多条语句构成。中间的 else if 可以编写任意多组。每个表达式代表一个条件,每个语句组都是符合某个条件情况下的一个执行分支。if 语句的执行逻辑是:从上至下先后计算各个表达式是否为真,若不为真,则不执行该表达式后面{}中的语句组,继续计算下一个表达式;若碰到一个表达式为真,则执行该表达式后面{}中的语句组,后面的表达式都不再计算,后面的语句组也都不再执行,接着执行整个 if 语句后面的其他语句。如果所有的表达式都为假,则执行最后的 else 对应的语句组 n,然后执行整个 if 语句后面的其他语句。表达式的值若为 0,则相当于是假;若为非 0,则相当于是真。

如果某个语句组只有一条语句,那么“{}”可以省略。

根据实际需要,如果没有那么多条件与选择分支,可以一个 else if 都不写,也可以不写最后的 else,而且也不是写了 else if 就一定要有 else。例如:

```
if(x > 0)
     cout << "x is positive" ;
```

上面这条 if 语句只在 x>0 时输出“x is positive”,如果 x 不大于 0,则什么都不做。

现在可以写出上面提到的判断整数奇偶性的程序了。

例 3.1 编写一个判断整数奇偶性的程序,要求输入一个整数,如果是奇数,就输出“It's odd.”,如果是偶数,就输出“It's even.”。

```
    //program 3.1.1.cpp 判断整数奇偶性
1.  #include <iostream>
2.  using namespace std;
3.  int main()
4.  {
5.      int n;
6.      cin >> n;
7.      if( n % 2 == 1)
8.          cout << "It's odd." ;              //当 n % 2 == 1 成立时会执行此语句
9.      else
10.         cout << "It's even." ;             //当 n % 2 == 1 不成立时会执行此语句
11.     return 0;
12. }
```

第 7 行的表达式“n % 2 == 1”就代表了一个条件,如果该表达式值为 true,就说明 n 除以 2 的余数是 1,那么 n 就是奇数,则执行第 8 行语句;如果该表达式值为 false,则不执行第 8 行语句,而执行 else 后面的第 10 行语句。

第 7 行也可以写成:

```
if( n % 2 )
```

因为如果 n 是奇数，则 n%2 的值就是非零值，同样能代表 true。

有时，在 if 语句的某个分支里面，还会做进一步的选择，因此，C++中的 if 语句是可以嵌套的，即在一条 if 语句的某个分支中，还可以再编写 if 语句。在分支只有一条语句而省略了前后的“{}”的情况下，有时容易造成困惑，如下面的程序段：

```
1.  #include <iostream>
2.  using namespace std;
3.  int main()
4.  {
5.      int a;
6.      cin >> a;
7.      if( a > 0)
8.          if ( a % 2 )
9.            cout << "good";
10.     else
11.          cout << "bad";
12.     return 0;
13. }
```

当输入－1 时，该程序应该输出“bad”，还是什么都不输出呢？

从程序的缩进格式看，似乎第 10 行的那个 else 应该是和第 7 行的那个 if 配对的，如果 a<=0，则应该执行第 11 行，输出“bad”。但实际上，C++语言规定，在没有通过“{}”明确表明 if 和 else 的配对关系时，else 是和离它最近的 if 相配对，所以本程序中的 else 实际上是和第 8 行的 if 配对的，即整个 if 语句实际上等效于：

```
if(a > 0) {
    if ( a % 2 )
        cout << "good";
    else
        cout << "bad";
}
```

上面这个写法显然是不会引发困惑的。在 a<=0 的情况下，程序没有输出。

如果想让 else 和前面那个 if 配对，则应该这样编写：

```
if(a > 0) {
    if ( a % 2 )
        cout << "good";
}
else
    cout << "bad";
```

此时如果 a<=0，则程序输出“bad”。

在编写嵌套 if 语句时，应当积极使用“{}”，以避免读程序者不小心看错了 if 和 else 的配对关系。

例 3.2　请编写一个程序，该程序输入一个年份，根据该年份是否是建国整十周年、建党整十周年以及是否是闰年给出不同的输出。

```
    //program 3.1.2.cpp 判断年份是否是建国、建党十周年或闰年
1.  #include <iostream>
```

```
2.  using namespace std;
3.  int main()
4.  {
5.      int year;
6.      cin >> year;
7.      if(year <= 0)
8.          cout << "Illegal year." ;
9.      else {
10.         cout << "Legal year." << endl;
11.         if( year > 1949 && (year - 1949) % 10 == 0 )                //建国整十周年
12.             cout << "Luky year.";
13.         else if(year > 1921 && !((year - 1921) % 10))               //建党整十周年
14.             cout << "Good year.";
15.         else if( year % 4 == 0 && year % 100 || year % 400 == 0 ) //闰年
16.             cout << "Leap year";
17.         else
18.             cout << "Common year.";
19.     }
20.     return 0;
21. }
```

下面列举出该程序对应于一些输入的输出结果：

```
-2 ↙
Illegal year.

1959 ↙
Legal year.
Luky year.

1931 ↙
Legal year.
Good year.

2008 ↙
Legal year.
Leap year.

2011 ↙
Legal year.
Common year.
```

第 13 行的表达式!((year－1921)％10)等效于(year－1921)％10＝＝0，因为(year－1921)％10 的值如果是 0 的话，那么!((year－1921)％10)的值就是非 0，等于是 true。

从上面的程序可以看出，在程序结构复杂的时候，**使用合适的缩进是非常重要的**。没有缩进的程序，阅读起来会非常困难，看不清哪些语句满足什么条件才会被执行。

初学者在使用 if 语句时很容易不小心写出类似下面的代码：

```
if( a = 5 )
    cout << "good";
```

编程者的本意是如果 a 等于 5 就输出“good”，但是本该写成“a ＝＝ 5”的，却误写成了

“a＝5”。编译器是不会发现这个错误的，它认为程序员就是要用表达式 a＝ 5 的值来作为判断的条件。结果就是即便原来 a 的值是不等于 5 的，计算表达式 a＝5（也就是执行表达式 a＝5）后，a 的值就变成了 5，而且表达式 a＝5 的值也是 5，即相当于 true。所以上面这条 if 语句，无论执行前 a 的值是多少，一定会输出“good”，而且把 a 的值变成了 5。

当然，如果不小心写了：

```
if( a = 0 )
    cout << "good";
```

那就无论如何不会输出“good”了。原因请读者自己分析。

上述错误即便在编程老手身上也常发生，而且一旦发生，很难查出来。避免上述错误的一个办法是养成把变量写在“＝＝”右边的习惯，如习惯于 “5＝＝a”这样的写法，那么即便不小心写成了“5＝a”，编译器也会报错，因为不能对一个常量进行赋值。

3.2　用 switch 语句实现选择结构

程序经常要根据不同的情况进行不同的处理，如果不同的情况很多，那么编写出来的 if 语句就会包含大量的 else if，例如：

```
if(n % 5 == 0) {
 …
}
else if(n % 5 == 1) {
 …
}
else if(n % 5 == 2) {
 …
}
else if(n % 5 == 3) {
 …
}
else {
 …
}
…
```

上面的代码编写起来有点麻烦，而且表达式 n％5 可能会被计算不止一次，不利于提高程序运行效率。为此，C++设计了 switch 语句，用于应对类似上面那样的情况。switch 语句的基本格式如下：

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

```
    …
    case 常量表达式 n:
        语句组 n
        break;
    default:
        语句组 n + 1
}
```

“switch”关键字右边的括号中的表达式，其值是整数类型的(char、unsigned char 类型也可以)。“case”右边的那些表达式，必须是常量表达式，即不包含变量，且值是编译时就能确定，不会在运行时发生改变的表达式。switch 语句的执行过程是计算出括号中表达式的值，假设等于 a，然后找到第一个值和 a 相等的“常量表达式 i”，执行其下面的语句组，一直执行到“break;”语句，整个 switch 语句就执行完毕，即便后面还有其他 case 分支的常量表达式值也等于 a，那些分支也不会被执行。如果所有“case”右边的常量表达式都不等于 a，则执行“default:”下面的语句组。

switch 语句中的语句组即使包含不止一条语句，也可以不用“{}”括起来。switch 语句也可以没有“default”部分。

例 3.3 请编写一个程序，接受一个整数作为输入，如果输入 1，则输出“Monday”；输入 2，则输出“Tuesday”；…；输入 7，则输出“Sunday”；输入其他数，则输出“Illegal”。

```
    //program 3.2.1.cpp 输出星期几的程序
1.  #include <iostream>
2.  using namespace std;
3.  int main()
4.  {
5.      int n;
6.      cin >> n;
7.      switch(n) {
8.        case 1:
9.            cout << "Monday";
10.           break;
11.       case 2:
12.           cout << "Tuesday";
13.           break;
14.       case 3:
15.           cout << "Wednesday";
16.           break;
17.       case 4:
18.           cout << "Thursday";
19.           break;
20.       case 5:
21.           cout << "Friday";
22.           break;
23.       case 6:
24.           cout << "Saturday";
25.           break;
26.       case 7:
27.           cout << "Sunday";
```

```
28.            break;
29.       default:
30.            cout << "Illegal";
31.    }
32.    return 0;
33. }
```

switch 语句在进入某个 case 分支后，会一直执行到第一个碰到的“break;”，哪怕这个“break;”是在后面的 case 分支里面。如果没有碰到“break;”，则会向下一直执行到 switch 语句末尾的“}”，包括“default:”部分的语句组也会被执行。初学者在使用 switch 语句时常常会忘了写“break;”，这会导致程序运行不正确。但有的时候，会需要多个 case 都执行相同的代码，那么就可以略去“break;”。例如下面的程序段：

```
    //program 3.2.2.cpp 缺少 break 的 case
1.  #include <iostream>
2.  using namespace std;
3.  int main()
4.  {
5.      int n;
6.      cin >> n;
7.      switch(n % 6) {
8.          case 0:
9.               cout << "case 0" << endl;
10.              break;
11.         case 1:
12.              cout << "case 1" << endl;
13.         case 2:
14.         case 3:
15.              cout << "case 2 or 3" << endl;
16.              break;
17.         case 4:
18.              cout << "case 4" << endl;
19.              break;
20.         default:
21.              cout << "default";
22.     }
23.     return 0;
24. }
```

上面的程序，如果输入 1，则会输出：

```
case 1
case 2 or 3
```

因为进入 case 1 分支，执行完第 12 行后，没有碰到“break;”，于是程序继续向下执行到第 16 行，第 16 行是“break;”，到此 switch 语句执行完毕。

如果输入 2 或 3，则都会执行第 15 行，然后执行第 16 行的“break;”时结束整个 switch 语句的执行。此种情况下程序输出：

```
case 2 or 3
```

3.3 用 for 语句实现循环结构

C++语言提供了 for 语句，用来重复执行相同的语句若干次，执行的次数一般是某个常量或某个变量的值。for 语句的基本格式如下：

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

如果语句组只由一条语句构成，那么也可以去掉“{}”。

通常，在需要将一些语句循环执行若干次的情况下，使用 for 语句比较方便。

for 语句流程图如图 3.1 所示，for 语句的执行过程如下。

(1) 计算“表达式 1”。

(2) 计算“表达式 2”的值。如果其值为 true，则执行“{}”中的语句组，然后转到步骤(3)；如果为 false，则不再执行“{}”中的语句组，for 语句结束，转到步骤(5)。

(3) 计算“表达式 3”。

(4) 转到步骤(2)。

(5) 从 for 语句后面继续往下执行程序。

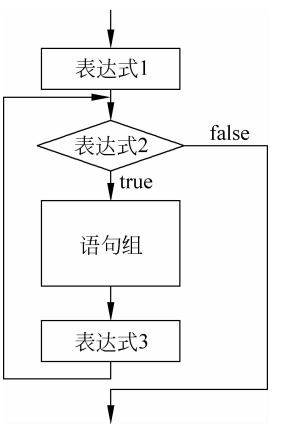

图 3.1 for 语句流程图

下面来看一个 for 语句的例子：

```
int i;
for( i = 0;i < 26; ++i ) {
     cout << char('a' + i ) << endl;
}
```

上面这个 for 语句，由于循环体中只有一条语句，“{}”其实也可以去掉。这条 for 语句的作用是依次输出 26 个小写字母。由于字符'a' ～ 'z'的 ASCII 码是连续的，所以'a'+i 就是第 i 个字母的 ASCII 码('a'是第 0 个字母，'z'是第 25 个字母)，将'a'+i 转换成 char 类型输出，就会输出第 i 个字母。上述 for 语句，每次循环都输出第 i 个字母，然后 i 的值增加 1。只要 i 的值还小于 26，就依然会执行循环。i 的值在循环开始执行前被置成了 0，因此循环一共会执行 26 次。

上面语句中的 i 用来控制循环执行的次数，称为循环控制变量。一段程序中经常会用到多个 for 循环。大家一般习惯将循环控制变量定义为 i 或 j，编写一个 for 循环之前，要想想是不是要定义循环控制变量 i 或 j，如果前面已经定义过了，就不能再定义，否则编译的时候会引起变量重复定义的错误。老要想这件事，比较麻烦，因此 C++允许在编写 for 语句时，在“表达式 1”处定义一个或多个变量，而且这些变量的作用域仅限于 for 语句之内。这样，就可以在“表达式 1”处定义循环控制变量，不用管前面是不是定义过了。于是，先输出 26 个小写字母，再间隔着输出 13 个大写字母的程序，可以如下编写：

```
for( int i = 0;i < 26; ++i )
    cout << char('a' + i ) << endl;
for( int i = 0;i < 26; i += 2 )                    //循环控制变量并非每次只能加 1
    cout << char('A' + i ) << endl;
```

不用担心两个 i 会导致重复定义的错误，因为它们各自的作用域（就是能起作用的范围）都仅限于它所处的 for 语句内。

对循环控制变量 i 进行加 1 时，许多程序员以及教科书都会写“i＋＋”，而不是“＋＋i”。表面上看没有区别，实际上“＋＋i”的写法是一种更好的习惯，显得更加专业和规范，因为＋＋i 比 i＋＋执行速度快。尤其在学了“运算符重载”和“标准模板库 STL”后就更能体会这一点了。

for 循环结构中的“表达式 1”和“表达式 3”都可以是用逗号连接的若干个表达式。例如下面的程序段：

```
for(int i = 15, j = 0; i > j; i - = 2 ,j += 3)
    cout << i << "," << j << endl;
```

输出结果是：

```
15,0
13,3
11,6
```

例 3.4　请编写一个程序，输入一个正整数 n，输出它的所有因子。

此程序的基本思路就是枚举，对于不超过 n 的每个正整数，都判断一下是否是 n 的因子，如果是，则输出。

```
    //program 3.3.1.cpp 输出正整数的所有因子
1.  #include < iostream >
2.  using namespace std;
3.  int main()
4.  {
5.      int n;
6.      cin >> n;
7.      for(int i = 1;i < = n; ++i)
8.       if( n % i == 0 )
9.           cout << i << endl;
10.     return 0;
11. }
```

上面的 for 循环中只有一条 if 语句，if 语句中也只有一条输出语句，所以都用不着写“{}”了。

该程序枚举的顺序是从小到大。如果要求从大到小的顺序输出所有因子，那么枚举的顺序也应该从大到小，程序中的 for 循环可以如下改写：

```
for( int i = n;i >= 1; --i )
     if( n % i == 0 )
          cout << i << endl;
```

在一个 for 循环中还可以执行另一个 for 循环。例如下面的两重循环：

```
for(int i = 0;i < n; ++i) {
    …
    for(int j = 0; j < m; ++j ) {
        …
    }
    …
}
```

假设 n 和 m 的值在执行过程中不发生变化，那么外重循环每执行 1 次，内重循环就执行 m 次，内重循环的执行次数一共是 n×m 次。在上面的程序段中，内重循环每次开始的时候，都会先将 j 的值初始化为 0。

例 3.5 给定正整数 n 和 m，在 1 至 n 这 n 个数中，取出两个不同的数，使得其和是 m 的因子，问有多少种不同的取法。

解题的思路就是枚举所有两个数的不同取法，看其和是否为 m 的因子，如果是，就将取法总数目加 1。枚举取两个数的不同取法，可以用两重 for 循环实现。程序如下：

```
    //program 3.3.2.cpp 取出两数使得和是另一个数的因子,有多少种取法
1.  #include <iostream>
2.  using namespace std;
3.  int main()
4.  {
5.      int n,m;
6.      int total = 0;                          //取法总数
7.      cin >> n >> m;
8.      for(int i = 1; i < n; ++i )             //取第一个数,共 n-1 种取法
9.          for(int j = i + 1; j <= n; ++j )    //第二个数要比第一个数大,以免取法重复
10.             if( m % (i + j) == 0 )
11.                 ++total ;
12.     cout << total;
13.     return 0;
14. }
```

另外 for 语句括号里面的“表达式 1”、“表达式 2”、“表达式 3”任何一个都可以不写，甚至可以全都不写，但是“;”必须保留。如：

```
for( ; i < 100; ++i )
    cout << i ;
```

假设 i 在 for 语句之前已经有合理的值了，那么上面的 for 语句中就没必要写“表达式 1”了。

如果不写“表达式 2”，就意味着永远执行循环（也称死循环）。例如：

```
for( ; ; )
    cout << "hello" << endl;
```

上面语句就会不停输出”hello”。

在 for(; ;) 这样的循环中可以通过 break 语句跳出循环，后文会提到。