第3章 程序控制

一个合理实用的程序，不可能是从头至尾一行一行地依次顺序执行的。程序的执行顺序在更多的场合应该适当地进行改变。这种改变程序从第一行依次顺序执行到最后一行的机制称为程序控制。程序控制语句有3类：选择语句、循环语句和跳转语句。这3类语句构成了由简单语句搭建程序大厦的基石。

3.1 选择语句

选择语句有if和switch，其中尤其以if语句最为常见。

3.1.1 if语句

if语句是最基本的程序流程控制语句。if可以配合else或者else if来无限扩展选择执行的分支，当然在实际编码过程中，不会写很多的if和else if。if语句可能会有如下几种使用形式。无论采用下面的哪种方式，即使分支再多，最多也只会有一个分支获得执行。

(1) 一个分支：if (条件) { 语句序列; }

(2) 两个分支：if (条件) { 语句序列; } else {语句序列; }

(3) 多分支：if (条件) { 语句序列; } else if {语句序列; }……else {语句序列; }

(4) 嵌套：if (条件) { if语句序列;} else { if 语句序列;}

其执行机制是：判断各个条件，哪个条件成立则执行哪个分支相应的语句序列。若所有的条件都不成立，则直接执行整个if块后的语句。

例如，若有一语音播报程序，当获知客户的性别为男时，可以输出“先生，你好!”，反之如果是女士时，则输出“女士，你好!”。

```
Console.Write(" 请输入您的性别:");
string sSex = Console.ReadLine();
if (sSex == "男")
    Console.WriteLine("先生,你好!");
else
    Console.WriteLine("女士,你好!");
```

上述程序即采用形式(2)，程序执行结果如图3-1所示。

但是若用户随便输入，只要不输入“男”，则上面的程序都将把客户视为女性，自然不合理，如图 3-2 所示。

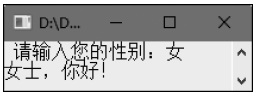

图 3-1　两分支的 if 语句——正常执行

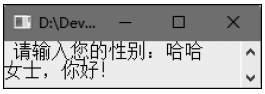

图 3-2　两分支的 if 语句——不正常执行

所以可以稍加修改，改善后的代码如下：

```
Console.Write(" 请输入您的性别:");
string sSex = Console.ReadLine();
if (sSex == "男")
    Console.WriteLine("先生,你好!");
else if(sSex == "女")
    Console.WriteLine("女士,你好!");
else
    Console.WriteLine("对不起,非法输入!");
```

此代码即形式(3)。

另外，为了使得给客户的提醒更具体点，比如根据当前时间，显示早上好、中午好、下午好、晚上好等比较具体的问候语，则可以编写如下代码：

```
// 如下仅考虑上午好和下午好两种问候;并且如下关于时间的判断并不严格,此处仅为演示之用,
你可以将其修改至合理.
Console.Write("请输入您的性别:");
string sSex = Console.ReadLine();
if (sSex == "男")
{
    if(DateTime.Now.Hour > 12) // DateTime.Now.Hour 用来获取当前时间的小时部分
        Console.WriteLine("先生,下午好!");
    else
        Console.WriteLine("先生,上午好!");
}
else if(sSex == "女")
{
    if(DateTime.Now.Hour > 12)
        Console.WriteLine("女士,下午好!");
    else
        Console.WriteLine("女士,上午好!");
}
else
    Console.WriteLine("对不起,输入有误!");
```

观察上述代码，不难发现，此即形式(4)，即 if 的嵌套使用。

课堂练习：请编写一个程序，根据用户输入的分数，输出其分数是优秀、良好、中等、及格或者是不及格。分级可以根据平时百分制的常规分级认定。

3.1.2 switch 语句

switch 语句与 if 语句一样，也是在众多分支中选择一个匹配的分支来执行。然而两者并不完全一样，并且在更多的情况下，对程序编码人员来说，用 if 会更习惯些。

其语法形式如下：

```
switch (表达式)
{
    case 值 1:
        语句序列;
        break;
    case 值 2:
        语句序列;
        break;
    …
    case 值 n:
        语句序列;
        break;
    default:
        语句序列;
        break;
}
```

其执行机制是：根据表达式的值，在各个 case 中寻找匹配的，如果找到匹配的 case，则执行相应的语句序列直到遇到 break，否则执行 default 分支，当然前提是 default 分支存在的情况下。

但是，需要注意如下事项。

(1) switch 的表达式的值只能是整型(byte、short、int、char 等)、字符串或枚举(其实枚举可以视为整型的特例)。

(2) 各个 case 下的 break 不可或缺；但是若某几个 case 共用一段语句序列时，break 可以不要。

(3) switch 语句同 if 语句一样，也可以嵌套。

仍以第 3.1.1 节的示例为例进行讲解说明。

```
Console.Write(" 请输入您的性别:");
string sSex = Console.ReadLine();
switch (sSex)
{
    case "男":
        Console.WriteLine("先生,你好!");
        break;
    case "女":
        Console.WriteLine("女士,你好!");
        break;
```

```
        default:
            Console.WriteLine("非法输入!");
            break;
    }
```

若某些 case 对应的语句块相同，则 break 可以省略。例如，上例根据用户输入的性别，若用户输入"男"或者"女"，程序输出"性别正常"，否则输出"性别不正常"。

```
Console.Write(" 请输入您的性别:");
string sSex = Console.ReadLine();
switch (sSex)
{
    case "男":
    case "女":
        Console.WriteLine("性别正常");
        break;
    default:
        Console.WriteLine("性别不正常");
        break;
}
```

两种合法的性别对应的 case 块，共用一个输出，此时可以采用上述这种写法。

课堂练习：请编写一个程序，要求使用 switch 语句完成。根据用户输入的分数，来输出其分数是优秀、良好、中等、及格或者是不及格。分级可以根据平时百分制的常规分级认定。

3.2 循环语句

循环分 3 类，分别为 for 循环、while 循环和 do…while 循环。

3.2.1 for 语句

for 语句是一种使用极其灵活的循环语句。其一般形式如下：

```
for (初始化语句; 条件测试语句; 迭代语句)
{
    循环语句序列     // 循环体,该处的语句序列会被反复执行直至循环结束
}
```

其中，初始化语句通常用于给循环变量赋初值，此处的循环变量往往就是计数器；而条件测试语句往往用来判断循环是否需要继续执行，当此处为 true 时循环继续，否则不再继续；而迭代语句往往用来实现对循环变量值的更改，正是该更改使得循环变量的值向使循环结束的趋势变化。

另外，需要指出的是，for 循环的上述 3 个部分并非必不可少的，可以有选择性地去除某

几个部分，甚至可以把3个部分全部去除。这样就可以得到for循环的多种变体形式。

其执行机制是：首先执行初始化语句，其次执行条件测试语句，当条件测试语句返回true时，接着执行循环语句序列，最后执行迭代语句，这样第一次循环即结束。除第一次循环需要执行初始化语句，其他时刻不会再执行。从第二次循环开始，每次首先执行条件测试语句，如果成立则执行循环语句序列，再执行迭代语句；然后又进入下一轮循环的条件测试语句判断，直至该语句不成立时循环结束。示例如下：

```
// 100以内等差数列的输出  1 2 3 4…
for(int i = 1;i < 100;i++)
    Console.WriteLine(i);
```

上面的一般形式也可以改写为while循环，其对应的while循环代码如下：

```
初始化语句;
while (条件测试语句)
{
    // do sth.
    迭代语句;
}
```

另外，for循环中迭代语句部分虽然用自增表达式最常见，然而却并不是必须这么做。

```
// 100以内奇数等差数列的输出  1 3 5 7…
for(int i = 1;i < 100;i += 2)
    Console.WriteLine(i);
// 100以内等比数列的输出  1 2 4 8…
for(int i = 1;i < 100;i * = 2)
    Console.WriteLine(i);
```

for循环的变体很多，此处不一一说明，仅给出2个简单示例，有兴趣的读者可以参看其他书籍，或者自行测试。

```
// 100以内奇数等差数列的输出  1 3 5 7…
for(int i = 1;i < 100;)
{
    Console.WriteLine(i);
    i += 2;
}
// 100以内等比数列的输出  1 2 4 8…
int i = 1;
for(;i < 100;)
{
    Console.WriteLine(i);
    i * = 2;
}
```

在上面的两个小示例中，第一个示例取消了迭代语句部分，而第二个示例则将初始化语句部分和迭代部分都取消了，然而程序仍能正确执行。如果将 3 个部分都取消，只留下循环语句序列部分，则构成一个死循环。

读者可以根据 for 循环的执行机制分析如上两段小程序。

当循环变量仅仅用于循环计数而无其他作用时，最好将循环变量 i 的作用域限制在 for 循环的结构内部，即：

```
for(int i = 0;i < n;i++)
```

而不应该按下述写法：

```
int i = 0;
for(i = 0;i < n;i++)
```

由于条件测试表达式会反复执行，所以如果该表达式来自于一个费时的函数，且该函数与循环变量无关，则应注意优化写法。

```
int GetMaxLength()
{
    System.Threading.Thread.Sleep(2000); // 模拟一个耗时的操作
    return 100;
}
for(int i = 0;i < GetMaxLength();i++)
{
    // do sth.
}
```

这样，虽然 GetMaxLength() 的返回值与循环变量 i 无关，但每次循环都会执行 GetMaxLength()，白白浪费了大量的时间。所以可以改写如下：

```
int max = GetMaxLength();
for(int i = 0;i < max;i++)
{
    // do sth.
}
```

修改后，这个耗时的操作将只会执行一遍。不少读者容易犯类似的错误，例如：

```
for(int i = 0;i < Convert.ToInt32(textBox1.Text);i++)
{
    // do sth.
}
```

此外还有另外一类问题，就是使用循环来做某种匹配，当匹配到时即退出循环。典型应用如在 ListBox 中添加不重复的项，比较通用的做法是定义一个 bool 标记变量 flag，然后在循环结束后通过 flag 值来决定做何种后续操作，参见第 7.4.6 节。

3.2.2 while 语句

while 循环是另外一种常见的循环形式，其一般形式如下：

```
while (条件表达式)
{
    循环语句序列;
}
```

其执行机制是：首先执行条件表达式，若为真则执行循环语句序列，接着再执行条件表达式，直到条件表达式不成立退出循环为止，继而执行循环之外的语句。

当条件表达式第一次就不成立时，此时循环语句序列不会获得任何执行机会。

仍以第 3.2.1 节中输出 100 以内的奇数等差数列为例进行说明。

```
// 100 以内奇数等差数列的输出 1 3 5 7…
int i = 1;
while (i < 100)
{
    Console.WriteLine(i);
    i += 2;
}
```

3.2.3 do…while 语句

do…while 循环是另外一种常见的循环，与 while 循环基本完全一样，其一般形式如下：

```
do
{
    循环语句序列;
} while (条件表达式);
```

其执行机制是：首先执行循环语句序列，然后执行条件表达式，若为真则接着执行循环语句序列，接着再执行条件表达式，直到条件表达式不成立退出循环而执行循环之外的语句。

从上面的叙述可以看到，do…while 循环中的循环语句序列至少将获得一次执行机会。这也是 do…while 与 while 的不同之处。

仍以第 3.2.1 节中的输出 100 以内的奇数等差数列为例进行说明。

```
// 100 以内奇数等差数列的输出 1 3 5 7…
int i = 1;
do{
```

```
        Console.WriteLine(i);
        i += 2;
    }
    while (i < 100);
```

对比第 3.2.2 节的 while 循环，也许看不到差别，但是若将 i 的初值赋为不小于 100（例如 1000）的整数，然后再执行上面两段程序，将会看到：while 循环对应的程序不会有任何输出，而 do…while 循环则会输出 1000。

3.3 跳转语句

跳转语句有 break、continue、goto、return、throw，这几个语句都能够改变程序的执行流程。其中尤其以 break、return 最为常见，throw 则用于异常处理，而 goto 一般都不推荐使用，因为它可能导致程序难以阅读、维护，给人混乱的感觉。

3.3.1 break 语句

break 语句除了用于 switch 语句中，其更多是用于退出循环，即将程序的执行流程从循环内转到循环外的第一条语句。其使用频率很高。

```
for (int i = 1;i < 10;i++)
{
    if (i % 3 == 0)
        break;
    Console.Write(i + "\t");          // 输出：  1  2
}
```

可见，当 i=3 时，由于满足 if 的条件，故执行 break，导致程序跳出了循环，后续的数值无法输出。

当存在多层循环嵌套时，break 仅从其所在的循环跳出，而不是跳出所有循环。

```
for(int j = 1;j < 4;j++)
{
    for (int i = 1;i < 10;i++)
    {
        if (i % 3 == 0)
            break;
        Console.WriteLine(i);
    }
    Console.WriteLine("内层循环结束");
}
Console.WriteLine("外层循环结束");
```

执行结果如图 3-3 所示。

从结果可见，虽然在内层循环中使用 break 退出了循环，但外层循环不受影响，仍然执行了 3 次。

图 3-3　多层循环下的 break 语句

3.3.2　continue 语句

continue 容易与 break 混淆起来，它也是一个用于循环控制的语句，其作用不是退出整个循环，而是将程序的执行流程提前跳转到下一次循环，执行流程仍然在循环内。这一点与 break 不一样，break 使得程序的执行流程从循环内跳转到了循环外。

```
for (int i = 1;i < 10;i++)
{
    if (i % 3 == 0)
        continue;
    Console.Write(i + "\t");          // 最终输出为:  1 2 4 5 7 8
}
```

从执行结果可见，凡是满足被 3 整除的数值都没有被输出，其他数值都被正常输出，表明程序遇到 continue，并未跳到循环外，只是略过了某些满足条件的循环而已。读者可以仔细对照上面的示例程序，体会 continue 与 break 的不同。

3.3.3　goto 语句

goto 关键字一般不推荐使用。不过有时使用 goto 可以大大简化程序代码，例如从嵌套层次很深的代码块中直接跳转到最外层。goto 在使用时需要配合一个行标签，即表明其跳转的目的位置。

下面以使用 goto 语句实现循环为例来说明其用法。

```
int i = 1;
begin:       // 行标签
if (i < 10)
{
    Console.Write(i + "\t");  // 最终输出为:  1 3 5 7 9
    i += 2;
    goto begin;
}
```

分析程序，不难看到，每当程序执行到 goto begin；时，程序的执行流程跳转到了 if 语句处开始执行，从而实现了循环。

3.3.4　return 语句

这是一个使用频率极高的关键词，用于从函数（方法）退出或者返回值，详见第 4.5 节。

3.3.5 throw 语句

这是一个用于异常处理的语句。当发生异常时，可以借助该关键字改变程序的正常执行流程，详见第 15.1 节。

3.4 问 与 答

3.4.1 if 和 switch 分别应用于什么场合

✌ if 比 switch 更加灵活强大，可以这么认为，凡是能使用 switch 的场合，肯定可以使用 if 来完成，但反过来却不一定。但在如下场合可以优先考虑 switch。

(1) 测试表达式的值为离散值，而非连续值；且取值个数不太多的场合。例如整型数据、枚举、字符等，当取值个数不多时都符合该条件。

(2) 测试表达式的值本身为连续值，但经过某种处理可以转化为离散值的场合。例如经常对分数按照某几个段来划分等级，此时可以将分数与 10 作整除运算即转换为离散值。

❗ 此处所说的离散不是数学上严谨的离散意义。例如 1,2,…在数学上是离散的，但当判断成绩分数的等级时，显然可以认为它们是连续的，而认为 60,70,…才是离散的。

3.4.2 if 和 switch 的各个分支的书写顺序有影响吗

✌ 虽然 if 和 switch 的各个分支是平行关系，其书写顺序对程序的结果不会有影响，但是其书写顺序对程序的执行效率是有影响的。一般而言，应该将可能性最大，即最有可能匹配的分支放到最前面，而将最不可能的分支放到最后面，这样可以避免很多不必要的判断和计算。例如，需要针对当前大学的学生做某项测试，年龄分段标准为 13～18(少年班的大学生年龄都会落在该区间)、19～24(一般大学本科生即在该区间)、25～30(研究生则落在该区间)，则一种可能的代码如下(下面仅为表意，代码是不可执行的，也不符合 C# 语法)。

```
if (13 - 18 & DoIt())
{
    // ...
}
else if (19 - 24 & DoIt())
{
    // ...
}
else if (25 - 30 & DoIt())
{
    // ...
}
else
```

```
{
    // ...
}

DoIt()
{
    // 模拟一个比较耗时的操作
    System.Threading.Thread.Sleep(2000);
    return true;
}
```

假如有 10 000 个测试对象，其中 6000 人介于 19～24 岁，3000 人介于 25～30 岁，1000 人介于 13～18 岁。在这里不详细比较，仅大概计算下，若按上面的代码，10 000 次判断中只有 1000 次匹配第一个分支，但由于将 13～18 放在第一个分支，所以即使不匹配该分支的场合也要执行测试，也就是另外 9000 次都白白去执行了一趟 DoIt()，也就是无谓地浪费了 9000×2＝18 000s，而如果能做如下调整：

```
if (19 - 24 & DoIt())
{
    // ...
}
else if (25 - 30 & DoIt())
{
    // ...
}
else if (13 - 18 & DoIt())
{
    // ...
}
```

则耗时将大大减少，请读者自行比较两种方式下的耗时情况。

以上代码只是为了说明不同写法所耗费的时间，至于存在的诸多不合理，读者不必在此深究，理解了在合适的场合使用最合适的方式书写代码即可。

上文所说的：将匹配可能性最大的分支放到最前面，此规则也适用于 switch。

上述规则也并不是任何情况下都应该遵从的，当执行各个分支的计算耗时不多时，也没有必要这么做，因为这样的违背人们习惯的排序方式，会让代码变得不易维护。例如将 100 以内的数值以 10 间隔，分为 10 段，假如不按人们的认知顺序来书写，会让读代码的人感觉无所适从。

3.4.3 如何避免太深的嵌套

无论是 if，还是 for 等循环，都应该避免太深的嵌套。具体如何避免该问题，当然要根据具体情况来分析，下面仅以一种常见的 if 嵌套方式来说明该问题，希望读者活学活用。

看如下的嵌套代码：

```
if(A)
{
    if(B)
    {
        if(C)
        {
            // do sth.
        }
    }
}
```

这里仅写了 3 层嵌套，实际应用中很多代码编写人员会写出更深的嵌套层次。这应该是极力避免的。看下面一种可能的改造方式。

```
if(!A)
{
    return;
}
if(!B)
{
    return;
}
if(!C)
{
    return;
}
// do sth.
```

3.4.4 for、while、do…while 分别应用于什么场合

首先需要说明的是，3 种循环在很多情况下其实是通用的，只要愿意，可以使用任何一种。不过一般情况下，可以依据下述原则来选择，仅供参考。

(1) 若循环的次数是已知的，选用 for 循环。

(2) 若循环次数未知，但可以确保至少会执行一次，则可以使用 do…while 循环。

(3) 若循环次数完全未知，可以使用 while 循环。

除了此处给出的 3 种循环，后文将介绍另外一种循环——foreach。

3.4.5 如何知道程序执行耗费的时间

要想实现该功能，首先需要引用命名空间 System.Diagnostics，在该命名空间下有一个类 Stopwatch，可以利用该类完成计时的功能。该类常用的属性和方法分别如表 3-1 和表 3-2 所示。

表 3-1　Stopwatch 常用属性

属　性	说　明
Elapsed	已经历了多久，为 TimeSpan 类型
ElapsedMilliseconds	已经历的毫秒数，long 型
ElapsedTicks	已经历的 Tick 数，long 型
IsRunning	Stopwatch 是否仍然在工作

表 3-2　Stopwatch 常用方法

方　法	说　明
Reset()	重置计时，即将上表中的属性置 0
Restart()	重启计时
Start()	启动计时
Stop()	停止计时

若要统计某段程序的执行耗时情况，一种最简单的使用方式是，在该代码块的前面调用 Stopwatch 实例对象的 Start()方法，而在代码块的后面调用该实例对象的 Stop()方法，此时再读取其 ElapsedMilliseconds 等属性即可知道程序执行耗费了多少时间。

```
static void Main(string[] args)
{
    Stopwatch sw = new Stopwatch();
    Console.WriteLine("开始计时");
    sw.Start();
    int s = 0;
    for (int i = 0; i < 10000000; i++)
        s += i;
    sw.Stop();
    Console.WriteLine("执行完毕,停止计时.程序执行耗时{0}毫秒",
sw.ElapsedMilliseconds);
}
```

程序执行效果如图 3-4 所示。

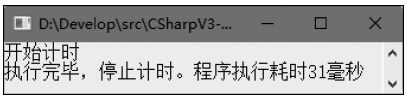

图 3-4　Stopwatch 演示

3.4.6　如何产生随机数

产生随机数是一个很常用的功能。下面讲解如何利用 System.Random 来产生随机数，其实例化对象主要有 3 种方法，如表 3-3 所示。

表 3-3　Random 常用方法

方　法	说　明
NextBytes()	批量生成随机数
NextDouble()	生成 0 至 1.0 之间的随机数，即[0.0,1.0)
Next()	Next()：产生非负随机整数 Next(int max)：生成介于 0 至 max 之间的随机整数，即[0,max) Next(int min,int max)：生成介于 min 至 max 之间的随机整数，即[min,max)。注意：min 可取负数

示例：产生随机数

```
Random r = new Random();
int i = r.Next(100);          // 随机产生一个介于 0 到 100 之间的随机整数,无法取到 100
int j = r.Next(60, 100);      // 随机产生一个介于 60 到 100 之间的随机整数,无法取到 100
int t = r.Next(-100,100);     // 产生介于 -100 到 100 之间的随机整数
for (int k = 0; k<20; k++)
    Console.Write(r.Next(100) + "\t");
```

3.4.7　什么是程序集

程序集根据不同的分类标准，可能有多种叫法，比如可以分为单文件程序集和多文件程序集，可以分为共享程序集和私有程序集。此处暂时不在该概念细节上深究。为了帮助读者理解，这里仅以一个狭隘而又直观的观点来认识程序集：在 Visual Studio 中写代码最终得到的 exe 文件和 dll 文件就是一个程序集。

3.5　思考与练习

(1) 从键盘上输入两个整数，由用户回答它们的和、差、积、商和取余运算结果，并统计出正确答案的个数。

(2) 求出 1～400 所有能被 9 整除的数，并输出每 5 个的和。

(3) 找出介于 2～10 000 的所有素数(请使用 for、while、do-while 这 3 种循环分别实现)。

(4) 父子年龄问题。设计一个程序，指定父子两人当前年龄，由程序完成两个任务的计算。①目前父亲年龄是儿子年龄的多少倍；②计算多少年后，父亲年龄变为儿子年龄的二倍。

(5) 从键盘输入 3 个整数，求出其中最大值和次最大值的和，以及 3 者的方差。

(6) 编程实现输入一个正整数 n，将其转换为二进制表达并输出。

(7) 编写一段程序，运行时向用户提问“你今年多少岁？(1～100)”，接受输入后判断其属于何种人生状态(婴儿、童年、少年、青年、中年、老年；各个年龄段如何分级请自行确定)，并要求在用户输入非法数据时给予适当提示，整个程序在用户输入 exit 时才退出，否则应反复循环上述问题让用户作答。