第3章 程序控制

3.1 概　　述

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即称为程序控制。程序控制语句有三类：选择语句、循环语句和跳转语句。这三类语句构成了由简单语句搭建程序大厦的基石。

3.2 选　　择

选择语句有 if 和 switch，其中尤其以 if 语句最为常见。

3.2.1 if

if 语句是最基本最常见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！”。则可能的参考代码如下。

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

上面的程序即采用形式 2,如果能按照预期输入,程序运行自然良好。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。例如如图 3-2 所示的输出。

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,你是人妖吧?!");
```

此代码即形式 3。

另外,为了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对,此处仅为演示之用
//其中 DateTime.Now.Hour 是用来获取当前时间的小时部分
Console.Write("请输入您的性别: ");
string sSex = Console.ReadLine();
if (sSex == "男")
    {
        if(DateTime.Now.Hour > 12)
            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!");
```

观察上面的代码,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,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)。

3.2.2 switch

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。然而两者并不是完全一样,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 3.2.1 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("泰国人妖!");
        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。)

3.3 循　　环

循环语句分为三类,分别为 for 循环、while 循环和 do…while 循环。

3.3.1 for

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 循环的上述三个部分并非必不可少的，可以有选择性地去除某几个部分，甚至可以把三个部分全部去除。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。

上面的一般形式，一般也可以改写为 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;
}
```

下面看一个示例：

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

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。如下示例：

```
//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，下面仅给出两个简单示例，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。如果将三个部分都取消，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，即应该这么写：

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

而不应该这么写：

```
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，则应注意优化的写法。例如：

```
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。所以可以做如下改写：

```
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。在 WinForm 或者 ASP.NET 中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。典型的应用如在 ListBox 中添加不重复项的情况，此时比较通用的做法是定义一个 bool 类型的标记变量 flag，然后在循环结束后通过 flag 的值来决定将要做何后续操作。该种方法的使用可以参看 7.3.7 节。

3.3.2 while

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，继而执行循环体之外的语句。

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 3.3.1 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;
}
```

3.3.3 do…while

do…while 循环是另外一种常见的循环形式，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 3.3.1 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 3.3.2 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。

3.4 跳　　转

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。

3.4.1 break

break 语句除了用于 switch 语句中，它更广的用途是用于退出循环，使用频率很高。即将程序的执行流程从循环内转到循环外的第一条语句。如以下示例：

```
for (int i = 1;i < 10;i++)
{
    if (i % 3 == 0)
        break;
    Console.WriteLine(i);
}
```

执行结果如图 3-3 所示：

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 仅从它所在的循环跳出，而不是跳转到所有循环的最外面。见如下示例：

```
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");
```


程序执行结果如图 3-4 所示。

图 3-3　break 语句

图 3-4　多层循环下的 break 语句

从结果明显地看到，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，仍然执行了三次。

3.4.2　continue

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，只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。见如下示例：

```
for (int i = 1;i < 10;i++)
{
    if (i % 3 == 0)
        continue;
    Console.WriteLine(i);
}
```

执行结果如图 3-5 所示：

从执行结果可以看到，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。

图 3-5　continue 语句

3.4.3　goto

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:
if (i<10)
{
    Console.WriteLine(i);
    i += 2;
    goto begin;
}
```

执行结果如图 3-6 所示。

图 3-6　使用 goto 实现循环

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。

3.4.4　return

这是一个使用频率极其高的关键词，用于从函数退出或者返回值，详见后文。

3.4.5　throw

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，详见后文。

3.5　问　与　答

◈ 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。

- 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。
- 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的意义。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。

◈ 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，所以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）：