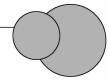
第3章

程序控制流程



语句是程序的最小单位。程序由一条一条语句组成，程序运行的过程就是语句逐条执行的过程，而语句执行的次序则称为流程。有了求解问题的算法，还要用程序实现出来，多数情况下，这种实现表现为一定数量的语句和执行流程。

3.1 语句

语句即一个在语法上自成体系的单位，一条语句能够表达一个完整的语义，是程序的基本功能单位。在 Visual Basic.NET 中有两种语句：声明语句和可执行语句。

3.1.1 声明语句

声明语句用于定义变量、常量、过程、属性和数组，还可以定义数据类型。下面的示例中包含三个声明语句：

```
Public Sub round()
    Const pi As Double=3.14159
    Dim radius As Double
End Sub
```

第一个声明是 Sub 语句，与其匹配的 End Sub 语句一起，它声明了一个名为 round 的过程。

第二个声明是 Const 语句，该语句声明常数 pi，并且指定 Double 数据类型和值 3.14159。

第三个声明是 Dim 语句，它声明变量 radius。

3.1.2 可执行语句

可执行语句用于执行一项操作。它可以调用过程、分支到代码中的另一个位置；可以循环执行多个语句；还可以计算表达式的值。赋值语句是可执行语句的一种特殊情况。下面的示例使用 If-Then-Else 控制结构根据变量的值运行不同的代码。

```
Dim age As Integer
age=10
```

```
If age<0 Then
    Console.WriteLine("该年龄不合法。")
Else
    Console.WriteLine("年龄为：{0}岁。", age)
End If
```

3.1.3 注释

可以在程序中编写注释，其目的是为了方便自己和别人阅读。在程序中添加适当的注释，是编程的一个良好习惯。在 Visual Basic. NET 中，有两种注释形式：单引号字符或者 REM 关键字。

```
'This is a single line comment
REM This is a single line comment
Console.WriteLine("'This will compile.")
```

需要注意的是，如果字符串中出现注释字符，则会按照一般的字符来处理，不再作为注释语句。

3.1.4 语句的写法

在 Visual Basic. NET 中，对于语句的写法有以下规定或惯例。

(1) 多数情况下，在一个程序行里只写一条语句，这样的程序写法清晰，便于阅读、理解和调试。

(2) 注意使用空格或 Tab 来做合理的间隔、缩进或对齐，使程序形成逻辑相关的块状结构，养成优美的程序编写风格。

(3) Visual Basic. NET 允许在一行里写多条语句，每两行语句之间用冒号分隔开，例如：

```
a=100 : b=100/10 : c=a+2
```

由于行是多数编译器在编译或调试时的基本单位，在一行里写多条语句的这种风格，即使编译器指明了某一行有错误也不能明确判别是哪条语句出的错，所以在一行里写多条语句的风格并不好。

(4) Visual Basic. NET 允许利用“续行符”将一条语句拆成多行来写，续行符依次包含一个“空格”、一个下划线字符“_”和一个“回车符”，例如：

```
MsgBox("Hello " & nameVar _
       & ". How are you?")
```

由于计算机屏幕宽度有限，过长的语句拆成多行来写是可能的。但需要注意两点：一是不能在关键字、标识符等中间拆分；二是字符串不能从中间拆分，例如：

```
MsgBox("Hello " & nameVar & ". How _
are you?")
```

(5) 在大多数情况下，可以在下一连续行上继续一条语句，而无须使用续行符。

在逗号之后省略续行符，例如：

```
Public Function GetUsername(ByVal username As String,
                ByVal delimiter As Char,
                ByVal position As Integer) As String
    Return username.Split(delimiter)(position)
End Function
```

在左括号“(”之后或右括号“)”之前省略续行符，例如：

```
Dim username=GetUsername(
    Security.Principal.WindowsIdentity.GetCurrent().Name,
    CChar("\"),1
)
```

还有许多情况下都可以省略续行符，这里就不一一列举了。

3.2　输入与输出

所谓输入是指从外部输入设备(如键盘、鼠标等)向计算机输入数据，输出是指从计算机向外部输出设备(如显示器、打印机等)输出数据。

在 Visual Basic.NET 中，我们可以采用控件来进行输入输出，如在一个文本框内输入数据，或者将数据输出到窗体上。还有一种最简单的输入输出方式就是控制台输入输出。控制台是一种简单的命令提示窗口，允许程序显示文本并从键盘接收输入。控制台中的输入输出由 System.Console 类来管理，Console 类包含了输入和输出数据到控制台窗口的方法。

控制台输入输出方式属于基本的输入输出方式，我们需要在程序中导入 System 命名空间，因此在代码的开头要加上如下一条语句：

```
Imports System
```

在控制台中，基本的输入是用 Console.Read 和 Console.ReadLine，基本的输出是用 Console.Write 和 Console.WriteLine。

3.2.1　输入输出方法

1. Console.Write

Write 是 Console 类的成员，它将一个文本字符串发送到程序的控制台窗口，字符串必须使用双引号括起来。例如：

```
Console.Write("This is a Visual Basic.NET program.")
```

需要注意的是，Write 在输出完字符串后，不会在字符串后添加换行符，例如：

```
Console.Write("This is sentence1.")
```

```
Console.Write("This is sentence2.")
Console.Write("This is sentence2.")
```

输出效果如下：

```
This is sentence1. This is sentence2. This is sentence3.
```

2. Console. WriteLine

WriteLine 是 Console 类的另一个成员，它和 Write 实现相同的功能，但会在每个输出字符串的结尾添加一个换行符。例如：

```
Console.WriteLine("This is sentence1.")
Console.WriteLine ("This is sentence2.")
Console.WriteLine ("This is sentence2.")
```

输出效果如下：

```
This is sentence1.
This is sentence2.
This is sentence3.
```

3. Console. Read

Read 是 Console 类的一个成员，它用于读取一个字符，返回该字符的 ASCII 码。例如：

```
Dim a As Integer
a=Console.Read
Console.WriteLine(a)
```

输出效果如下：

```
a↙
97
```

4. Console. ReadLine

ReadLine 是 Console 类的另一个成员，它用于读取一个字符串，返回值是 String 类型。例如：

```
Dim s As String
s=Console.ReadLine
Console.WriteLine(s)
```

输出效果如下：

```
Hello world!↙
Hello world!
```

可以利用 ReadLine 向控制台输入其他类型的数据，但程序中需要使用类型转换函数转换成相应类型，常用的类型转换函数有 CChar，CDbl 和 CInt 等。更多类型转换函数参考本书 2.5 节内容。

例如：

```
Dim i As Integer, d As Double
i=CInt(Console.ReadLine)
d=CDbl(Console.ReadLine)
Console.WriteLine(i)
Console.WriteLine(d)
```

输出效果如下：

```
100↙
3.1415926↙
100
3.1415926
```

3.2.2　格式化输出

有时我们希望数据能按指定的格式进行输出，如要求输出的小数只保留两位小数、以十六进制形式输出等。

Write 和 WriteLine 的常规形式中可以有一个以上的参数。

(1) 如果有多个参数，每两个参数间用逗号隔开。

(2) 第一个参数必须是字符串，称为“格式字符串”。

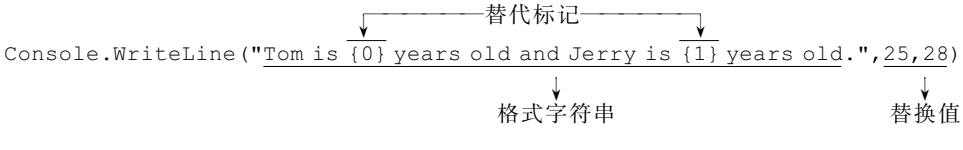
(3) 字符串可以包含替代标记，替代标记由一个整数及括住它的一对大括号组成。替代标记在格式字符串中标出位置，该位置将用一个替换值来替代。

(4) 紧跟着格式字符串的参数称为替换值，这些替换值从 0 开始编号。

WriteLine 语句形式如下：

```
Console.WriteLine(格式字符串(含替代标记),替换值 0,替换值 1,替换值 2,……)
```

例如：有两个替代标记，编号 0 和 1；以及两个替换值，它们的值分别为 25 和 28。



输出效果如下：

```
Tom is 25 years old and Jerry is 28 years old.
```

(5) 可以通过标准格式数字说明符来指定输出格式，标准格式数字说明符含义如表 3-1 所示。

表 3-1 标准格式数字说明符

名字和字符	意　义
货币 C、c	使用货币符号把值格式化为货币 精度说明符：小数位数
十进制数 D、d	十进制数字字符串，需要的情况下有负数符号。只能和整数类型匹配使用 精度说明符：输出字符串中的最少位数。如果实际数字的位数更少，则在左边用 0 填充
定点 F、f	带有小数点的十进制数字字符串，如果需要也可以有负数符号 精度说明符：小数的位数
常规 G、g	在没有指定说明符的情况下，会根据值转换为定点或科学计数法表示的紧凑形式 精度说明符：根据值
十六进制数 X、x(区分大小写)	十六进制数字字符串。十六进制数字 A～F 会匹配说明符的大小写形式 精度说明符：输出字符串中的最少位数。如果实际数的位数更少，则在左边用 0 填充
数字 N、n	和定点表示法相似，但是在每三个数字的一组中间有分隔符，从小数点开始往左数 精度说明符：小数的位数
百分比 P、p	表示百分比的字符串，数字会乘以 100 精度说明符：小数的位数
往返过程 R、r	保证输出字符串后如果使用 Parse 方法将字符串转化成数字，那么该值和原始值一样 精度说明符：忽略
科学记数法 E、e(区分大小写)	具有尾数和指数的科学记数法，指数前面加字母 E，E 的大小写和说明符一致 精度说明符：小数的位数

例如：

```
Console.WriteLine("{0:C},{1:D4},{2:F4}", 12.3, 12, 3.1415926)
Console.WriteLine("{0:G4},{1:X}", 123.456789, 1000)
Console.WriteLine("{0:N2}", 123456789.123456)
Console.WriteLine("{0:P2},{1:R},{2:E4}", 0.123456789, 123.456789,
123.456789)
```

输出效果如下：

```
¥12.30,0012,3.1416
123.5,3E8,
123,456,789.12
12.35%,123.456789,1.2346E+002
```

(6) 可以使用任意数量的替代标记和任意数量的替换值。并且替换值可以以任何顺序使用，还可以在格式字符串中替换任意值。

例如，代码中使用了三个标记但只有两个替换值。而且值 1 被用在了值 0 之前，值 1 还被用了两次。

```
Console.WriteLine("The top three countries is: {1},{0} and {1}.", "Korea",
"China")
```

输出效果如下：

```
The top three countries is: China, Korea and China.
```

3.3　程序顺序结构

3.3.1　顺序执行

通常情况下，语句以其出现的顺序执行，一个语句执行完会自动转到下一个语句开始执行，这样的执行称为顺序执行。顺序执行反映了程序“按部就班”的执行规律，多数情况下，程序的执行就是这样的。

顺序执行的次序非常重要，例如求三个同学的平均成绩，其执行顺序就应该如图 3.1 所示。

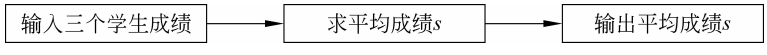


图 3.1　求三个学生平均成绩的执行顺序

显然，三个步骤中颠倒任意一个次序，结果都会不正确。程序代码如下：

```
Module Module1
    Sub Main()
        Dim x, y, z, s As Double
        Console.WriteLine("请输入第一个学生的成绩：")
        x=CDbl(Console.ReadLine)
        Console.WriteLine("请输入第一个学生的成绩：")
        y=CDbl(Console.ReadLine)
        Console.WriteLine("请输入第一个学生的成绩：")
        z=CDbl(Console.ReadLine)
        s=x+y+z
        s=s/3
        Console.WriteLine("平均成绩为{0:F1}。", s)
    End Sub
End Module
```

顺序执行有一种特殊的情形就是过程执行。程序执行到过程调用时，会暂停当前的执行流程，进入过程中开始一段新的执行流程，从过程返回后再继续当前的执行流程。

过程中的执行流程可以是程序流程的任意形式，它似乎是顺序执行中的一段“小插曲”。然而正是这种执行机制使得程序可以由简单的顺序执行激活更多层次、更多嵌套、更复杂的执行流程，从而满足算法求解的执行需要。

3.3.2 跳转执行

除最简单的程序外，顺序执行对于解决问题来说是不够的。从问题求解的一般过程来看，我们还需要跳转执行。

跳转语句可以无条件地转移控制。Visual Basic. NET 提供了许多转移控制语句，包括 GoTo，Continue，Return，Exit，End，Stop 和 Throw 等。这里先介绍 GoTo 语句。

GoTo 语句的作用是使程序无条件地跳转到由标签标记的语句，语法形式为

```
GoTo 标签
```

对 GoTo 的使用有以下几点说明：

(1) 标签既可以是有效的 Visual Basic. NET 标识符，也可以是整数。

(2) 标签必须出现在代码行的行首，之后必须有冒号，即使其后没有语句，也不能省略冒号。

(3) GoTo 语句只能跳转到其所在过程内的行，且该行必须有 GoTo 可以引用的标签。

(4) 不能使用 GoTo 语句跳转到 For-Next，For Each-Next，Try-Catch-Finally，With-End With，Using-End Using 结构内。

(5) 结构化程序设计方法主张限制使用 GoTo 语句，因为 GoTo 语句使代码的阅读和维护变得更加困难，应该尽可能使用控制结构代替 GoTo 语句。

【例 3.1】 求若干个学生的总成绩，如果输入负数则结束程序。程序代码如下：

```
 1  Module Module1
 2      Sub Main()
 3          Dim score, sum As Double
 4          sum=0
 5  again:
 6          score=CDbl(Console.ReadLine)
 7          If score<0 Then
 8              Console.WriteLine("总成绩为: {0:F1}", sum)
 9          End If
10          sum=sum +score
11          GoTo again
12      End Sub
13  End Module
```

程序运行结果如下：

```
60.5↵
80↵
50↵
-1↵
总成绩为: 190.5
```

3.4　程序选择结构

3.4.1　If 语句

1. If-Then 语句和 If-Then-Else 语句

(1) If-Then 形式

```
If 表达式 [Then]
    语句
End If
```

或者：

```
If 表达式  Then  语句
```

(2) If-Then-Else 形式

```
If 表达式 [Then]
    语句 1
Else
    语句 2
End If
```

或者：

```
If 表达式  Then  语句 1  Else  语句 2
```

其中语句 1 或语句 2 称为子语句，两个转向分支称为 If 分支和 Else 分支，表达式称为选择条件。

第(1)种形式的 If-Then 语句的执行过程是：先计算表达式的值，无论表达式为何种类型，均将这个值按逻辑值处理。如果其逻辑值为真，则执行子语句，然后执行 If 语句的后续语句；如果表达式的值为假，则什么也不做，直接执行 If 语句的后续语句。其执行流程图如图 3.2(a)所示。

第(2)种形式的 If-Then-Else 语句执行过程是：先计算表达式的值，无论表达式为何种类型，均将这个值按逻辑值处理。如果其逻辑值为真，则执行子语句 1，然后执行 If-Then-Else 语句的后续语句；如果表达式的值为假，则执行子语句 2，然后执行 If-Then-Else 语句的后续语句。其执行流程图如图 3.2(b)所示。

下面对 If 语句的用法做详细说明。

(1) If 语句中的子语句既可以是单独的一条语句，也可以是若干条语句的集合，例如：

```
1  If a>b Then
2      t=a
3      a=b
```

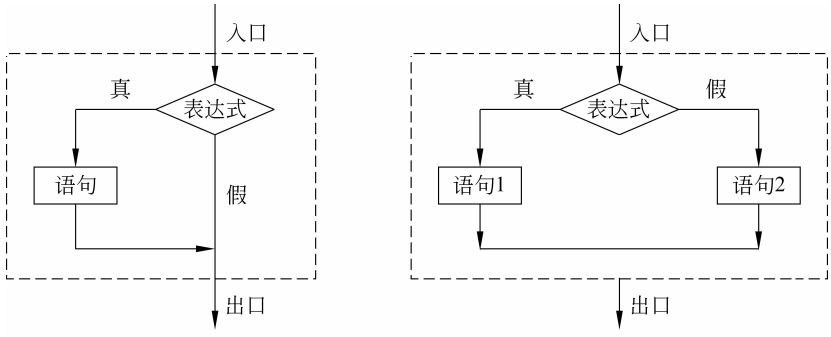


图 3.2　两种形式的 If 语句执行流程

```
4      b=t
5  End If
```

(2) 从两种形式的 If 语句的执行流程可知,If 语句有一个共同的入口和出口,执行流程的不同依赖子表达式值的不同,这种形式就是选择结构。从流程图的虚框来看,If 语句所形成的选择结构可以抽象为顺序结构的一步。这反过来提示我们在编程时可以先抽象设计顺序结构的一步,再使用选择结构细化。

(3) If 语句的表达式可以是 Visual Basic. NET 的任意表达式,但由于其结果是按逻辑值来处理的,通常情况下,选择条件是关系表达式或逻辑表达式,应该谨慎出现别的表达式。例如,选择条件是数值或算术表达式:

```
Dim a As Integer=100
Dim b As Integer=-100
If a Then Console.WriteLine(a)
If a +b Then Console.WriteLine(a +b)
```

在上面代码中,直接将数值按逻辑值来处理,遵循“非 0 数值的逻辑值为真,0 的逻辑值为假”原则,因此第一个表达式 a 的逻辑值为真,第二个表达式 a+b 的逻辑值为假。

(4) If-Then-Else 语句和 If 运算符似乎很像,例如:

```
If a >=b Then
    max=a
Else
    max=b
End If
```

和

```
max=If(a >=b, a, b)
```

结果是完全一样的,条件运算符的写法似乎更简洁。实际上 If-Then-Else 语句是语句,它可以包含任意多的表达式,或任意多的语句组合。而 If 运算符则能力有限,它仅仅局限于表达式。因此 If-Then-Else 语句可以替代 If 运算符,反之则不成立。

【例 3.2】 求三个数中的最小数。程序代码如下：

```
Module Module1
    Sub Main()
        Dim a, b, c, min As Integer
        a=CInt(Console.ReadLine)
        b=CInt(Console.ReadLine)
        c=CInt(Console.ReadLine)
        If a >b Then
            min=b
        Else
            min=a
        End If
        If min >c Then
            min=c
        End If
        Console.WriteLine("min={0}", min)
    End Sub
End Module
```

程序运行结果如下：

```
2↵
1↵
3↵
min=1
```

2. If-Then-Elseif 语句

If-Then-Elseif 语句的作用是根据不同的选择条件来确定执行哪个语句块。语句形式为

```
If  表达式 1  Then
    语句 1
Elseif  表达式 2  Then
    语句 2
⋮
Elseif  表达式 n  Then
    语句 n
Else
    语句 m
End If
```

If-Then-Elseif 语句的执行过程是：先判断表达式 1 的逻辑值，如果为真则执行语句 1，然后结束整个结构；当表达式 1 为假的时候，再判断表达式 2 的逻辑值，如果表达式 2 为真，则执行语句 2，为假就再判断表达式 3 的逻辑值，依此类推。因此，子语句 n 若要执

行，则前面的表达式均为假且表达式 n 为真；子语句 m 若要执行，则所有的表达式全部为假。

其执行流程图如图 3.3 所示。

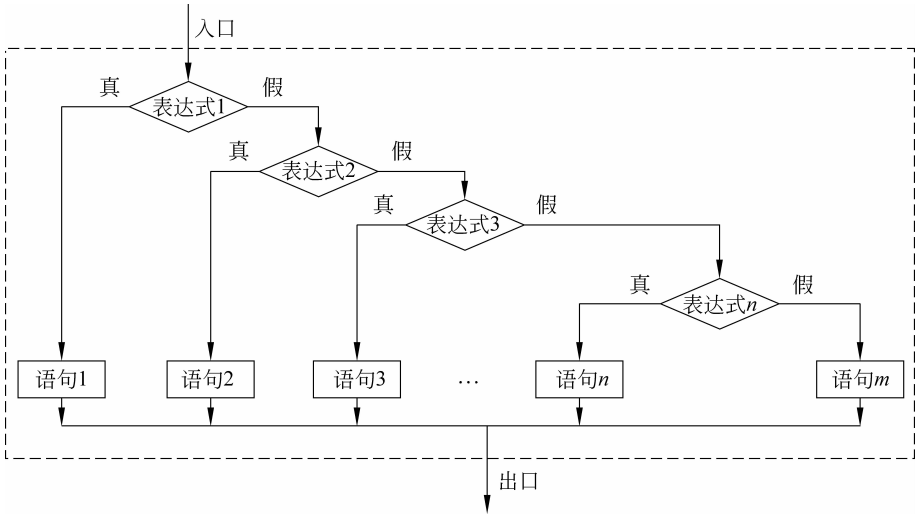


图 3.3　If-Then-Elseif 语句的执行流程

【例 3.3】 编程输出成绩分类，90 分以上为 A，80～89 为 B，…，60 以下为 E。程序代码如下：

```
Module Module1
    Sub Main()
        Dim score As Integer
        Console.WriteLine("请输入学生成绩：")
        score=CInt(Console.ReadLine)
        If score >=90 Then
            Console.WriteLine("A")
        ElseIf score >=80 Then
            Console.WriteLine("B")
        ElseIf score >=70 Then
            Console.WriteLine("C")
        ElseIf score >=60 Then
            Console.WriteLine("D")
        Else
            Console.WriteLine("E")
        End If
    End Sub
End Module
```

3.4.2　Select Case 语句

Select Case 语句的作用是计算给定的表达式的值，根据结果来选择执行哪一路分

支，语句形式为：

```
Select Case 表达式
    Case 表达式列表 1
        语句 1
    Case 表达式列表 2
        语句 2
        ⋮
    Case 表达式列表 n
        语句 n
    Case Else
        语句 n+1
End Select
```

Select Case 语句的执行流程为：先计算表达式的值，再判断表达式的值和哪个表达式列表匹配，如果和表达式列表 n 匹配上了，则执行语句 n，然后退出整个结构。当表达式和所有表达式列表都匹配不上时，则执行语句 $n+1$，然后退出整个结构。

其执行流程图如图 3.4 所示。

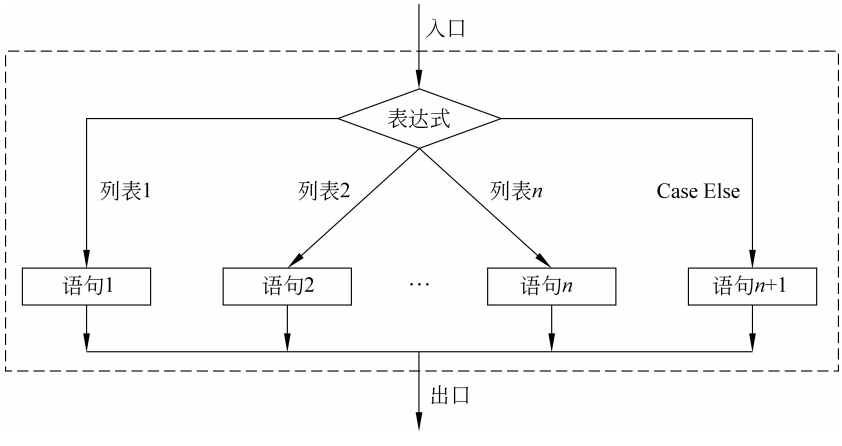


图 3.4　Select Case 语句的执行流程

下面对 Select Case 语句的用法做详细说明。

(1) 如果表达式与多个列表的值匹配，则只有跟在第一个匹配列表项之后的语句才会执行。例如下面的代码输出结果为 x=10。

```
Select Case x
    Case 5
        Console.WriteLine("x=5")
    Case 10
        Console.WriteLine("x=10")
    Case 20 -10
        Console.WriteLine("x=20-10")
    Case 30
```

```
        Console.WriteLine("x=30")
End Select
```

(2) 表达式所允许的数据类型有 Boolean, Byte, Char, Date, Double, Decimal, Integer, Long, Object, SByte, Short, Single, String, UInteger, ULong 和 UShort。

(3) 在 Case 子句里可以有多个表达式列表,每两个表达式列表中间用逗号隔开。例如下面的代码输出结果为 between 6-10。

```
Dim x As Integer=10
Select Case x
    Case 1, 2, 3, 4, 5
        Console.WriteLine("between 1-5")
    Case 6, 7, 8, 9, 10
        Console.WriteLine("between 6-10")
    Case 11, 12, 13, 14, 15
        Console.WriteLine("between 11-15")
    Case 16, 17, 18, 19, 20
        Console.WriteLine("between 16-20")
    Case Else
        Console.WriteLine("Other numbers.")
End Select
```

(4) 可以使用关键字 Is 从而在 Case 子句里使用条件表达式,Is 就相当是表达式的值。例如下面的代码输出结果为 The number is less than 10。

```
Dim x As Integer=10
Select Case x
    Case Is <=10
        Console.WriteLine("The number is less than 10")
    Case 11
        Console.WriteLine("You entered eleven.")
    Case Is >=100
        Console.WriteLine("The number is greater than or equal to 100.")
    Case Else
        Console.WriteLine("The number is between 12 and 99.")
End Select
```

(5) 可以通过关键字 To 在 Case 子句里确定一个数值范围,用于和表达式的值进行匹配。例如下面的代码输出结果为 y=0.04。

```
Dim x As Integer=365
Select Case x
    Case 1 To 100
        Console.WriteLine("y=0.01")
    Case 101 To 200
        Console.WriteLine("y=0.02")
```

```
    Case 201 To 300
        Console.WriteLine("y=0.03")
    Case 301 To 400
        Console.WriteLine("y=0.04")
End Select
```

(6) 每一个 Case 标签后的表达式列表的数据类型应该与表达式的类型相同,如果不相同应该可以进行隐式类型转换。

(7) 一个 Select Case 语句中最多只能有一个 Case Else 标签,而且必须将 Case Else 放在所有 Case 标签的后面。

(8) 可以通过 Exit Select 语句退出 Case 或者 Case Else 语句块,控制流程立即转到 Select Case 语句的后续语句。

(9) Select Case 语句可以相互嵌套。每个嵌套的 Select Case 语句必须完整地包含在外部 Select Case 语句的单个 Case 或 Case Else 语句块内。

【例 3.4】 判断学生成绩在哪一个分数段内。程序代码如下:

```
 1  Module Module1
 2      Sub Main()
 3          Dim grade As Char
 4          grade=Console.ReadLine()
 5          Select Case grade
 6              Case "A"
 7                  Console.WriteLine("90——100 分")
 8              Case "B"
 9                  Console.WriteLine("80——90 分")
10              Case "C"
11                  Console.WriteLine("70——80 分")
12              Case "D"
13                  Console.WriteLine("60——70 分")
14              Case "E"
15                  Console.WriteLine("不及格")
16              Case Else
17                  Console.WriteLine("error!")
18          End Select
19    End Sub
20  End Module
```

3.4.3　选择结构的嵌套

在 If 语句和 Select 语句中,分支的子语句可以是任意的控制语句,当这些子语句又是一个 If 语句或者 Select 语句时,就构成了选择结构的嵌套。

1. If 语句的嵌套

第一种形式就是 If-Then-Elseif 语句,详细内容见 3.4.1 节。

第二种形式是在 If 和 Else 分支中嵌套 If 语句，语法形式为

```
If  表达式 1  Then
    If  表达式 2  Then
        语句 1
    Else
        语句 2
    End If
Else
    If  表达式 3  Then
        语句 3
    Else
        语句 4
    End If
End If
```

其中，表达式 2 和表达式 3 所在的 If 语句为内嵌 If 语句。

【例 3.5】 判断某一年是否是闰年。判断闰年的条件是：能被 4 整除但不能被 100 整除的年份是闰年，或者能够被 400 整除的年份也是闰年。判断流程图如图 3.5 所示。

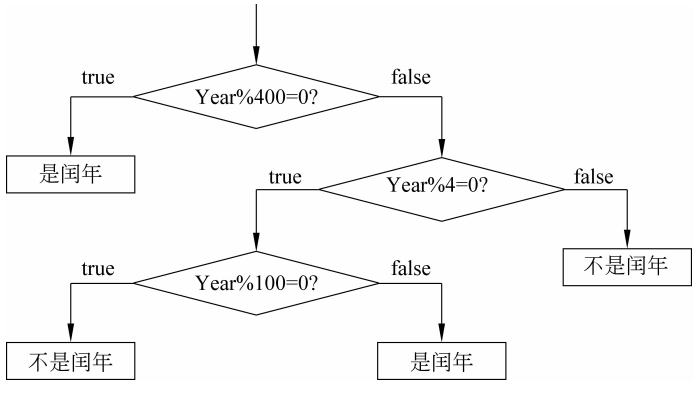


图 3.5　闰年的判断条件

程序代码如下：

```
 1  Module Module1
 2      Sub Main()
 3          Dim year As Integer
 4          year=CInt(Console.ReadLine)
 5          If year Mod 400=0 Then
 6              Console.WriteLine("{0} year is a leap year.", year)
 7          Else
 8              If year Mod 4=0 Then
 9                  If year Mod 100=0 Then
10                      Console.WriteLine("{0} year is not a leap year.", year)
11                  Else
```

```
12                    Console.WriteLine("{0} year is a leap year.", year)
13                End If
14            Else
15                Console.WriteLine("{0} year is not a leap year.", year)
16            End If
17        End If
18    End Sub
19 End Module
```

2. Select 语句的嵌套

Select Case 语句也是可以嵌套的，例如下面的程序：

```
 1 Module Module1
 2     Sub Main()
 3         Dim a As Integer=15, b As Integer=21, m As Integer=0
 4         Select Case a Mod 3
 5             Case 0
 6                 m=m +1
 7                 Select Case b Mod 2
 8                     Case 0
 9                         m=m +1
10                         Exit Select
11                     Case Else
12                         m=m +1
13                 End Select
14             Case 1
15                 m=m +1
16         End Select
17         Console.WriteLine(m)
18     End Sub
19 End Module
```

第 4 行 Case 分支执行，第 7 行 Select 语句执行，当第 10 行 Exit Select 执行时结束 Select Case b Mod 2 语句，转到第 14 行继续执行上一级 Select Case a Mod 3 语句的 Case 分支。

程序运行结果为：2。

从上面的例子可以看出，Select 语句中的 Exit Select 只能终止包含它的 Select 语句，而不是终止所有的 Select 语句。使用嵌套的 Select 语句后，程序的分支变得更加复杂了。

3.4.4　选择结构程序举例

【例 3.6】 输入某天的日期(年、月、日)，输出第二天的日期。

分析：设某天为(y,m,d)，那么第二天的日期为$(y,m,d+1)$。但如果 $d+1$ 大于本

月的最大天数(如31天)时,则第二天的日期为(y,m+1,1)。如果m+1大于12时,则第二天的日期为(y+1,1,1)。且当月份为1,3,5,7,8,10,12月时,每月的天数为31天;月份为4、6、9、11时,每月的天数为30天。平年2月的天数为28天,闰年2月的天数为29天。

程序代码如下:

```
Module Module1
  Sub Main()
        Dim y, m, d, days As Integer
        y=CInt(Console.ReadLine)
        m=CInt(Console.ReadLine)
        d=CInt(Console.ReadLine)
        Select Case m                    '计算每月的天数
            Case 2
                days=28
                If (y Mod 4=0 And y Mod 100 <>0) Or (y Mod 400=0) Then
                    days=days +1     '闰年2月为29天
                    Exit Select
                End If
            Case 4 : Case 6 : Case 9 : Case 11
                days=30
            Case Else
                days=31
        End Select
        d=d +1
        If d >days Then
            d=1 : m=m +1
        End If
        If m >12 Then
            m=1 : y=y +1
        End If
        Console.WriteLine("{0}-{1}-{2}", y, m, d)
    End Sub
End Module
```

程序运行结果如下:

```
2004↵
2↵
28↵
2004-2-29
```

【例3.7】 输入月份m和日期d,按下面的对应关系输出相应的星座。

3.21-14.20 白羊	4.21-5.20 金牛	5.21-6.20 双子	6.21-7.22 巨蟹
7.23-8.22 狮子	8.23-9.22 处女	9.23-10.22 天秤	10.23-11.22 天蝎
11.23-12.22 射手	12.23-1.20 摩羯	1.21-2.20 水瓶	2.21-3.20 双鱼

分析：显然，可以使用 If 语句多重嵌套来逐一比较输出。但从图 3.6 中可以看出，日期区间是有规律的，如果将大于 21 或 23 的日期的月份加 1，则星座对应关系就可以用月份来描述，就可以使用 Select Case 语句。12 月 23 日后月份加 1 会有“13 月”，这个“13 月”其实和 1 月是一样的，可以利用 Case Else 分支。

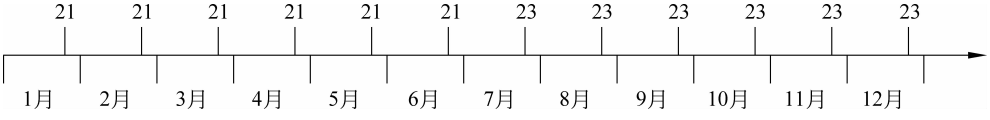


图 3.6　星座与日期规律

程序代码如下：

```
Module Module1
    Sub Main()
        Dim m, d, t As Integer
        m=CInt(Console.ReadLine)
        d=CInt(Console.ReadLine)
        t=IIf(m<7, 21, 23)
        If d >=t Then
            m=m +1
        End If
        Select Case m
            Case 2
                Console.WriteLine("水瓶座")
            Case 3
                Console.WriteLine("双鱼座")
            Case 4
                Console.WriteLine("白羊座")
            Case 5
                Console.WriteLine("金牛座")
            Case 6
                Console.WriteLine("双子座")
            Case 7
                Console.WriteLine("巨蟹座")
            Case 8
                Console.WriteLine("狮子座")
            Case 9
                Console.WriteLine("处女座")
            Case 10
                Console.WriteLine("天秤座")
            Case 11
                Console.WriteLine("天蝎座")
            Case 12
                Console.WriteLine("射手座")
            Case Else
```

```
34                Console.WriteLine("摩羯座")
35          End Select
36      End Sub
37  End Module
```

3.5 程序循环结构

利用循环，可以重复执行相同的语句块。循环结构、顺序结构和选择结构共同作为各种复杂程序的基本构造单元。

Visual Basic.NET 提供了 4 种不同的循环机制：For-Next 语句、While-End While 语句、Do-Loop 语句和 For Each-Next 语句。

3.5.1 For-Next 语句

For-Next 语句的作用是计算给定的表达式，根据结果判定是否执行循环语句，For-Next 语句可以通过循环控制变量控制循环的次数。语句形式为

```
For 循环控制变量 [As datatype]=初值 To 终值 [Step 步长]
    语句
Next [循环控制变量]
```

其中，语句又称为循环体，带有大括号的部分为可以根据实际情况省略的部分。

For-Next 语句的执行过程如下，执行流程图如图 3.7 所示。

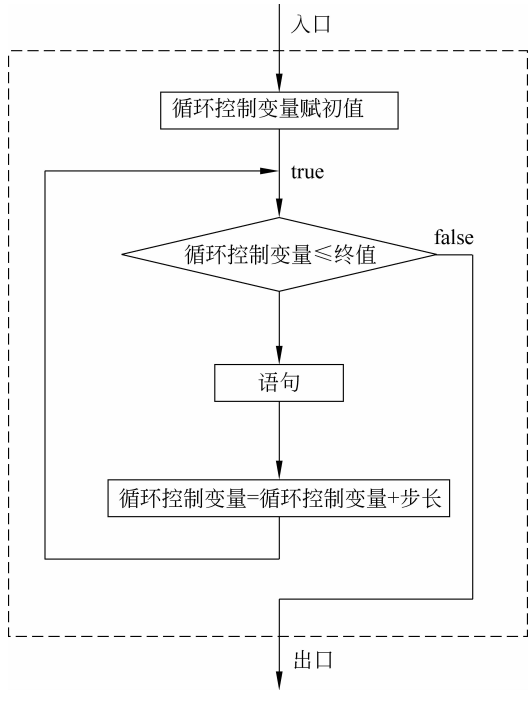


图 3.7 For-Next 语句流程图