Microsoft Small Basic

编程入门

第一章

入门

Small Basic 与编程

计算机编程被定义为使用编程语言创建计算机软件的过程。就像我们可以说并且理解英语，西班牙语或者法语，计算机能理解用特定语言写成的程序。这些特定语言被称为编程语言。最初，只有很少几种编程语言，并且它们都非常容易学习和理解。但是随着计算机和软件变得越来越精深，编程语言发展很快，并且随之汇集了更复杂的概念。从而造成现代大部分编程语言以及相关的概念对于初学者掌握起来而言颇具挑战性。这一事实已经开始阻碍人们学习或尝试计算机编程。

Small Basic 是一门针对初学者设计的使编程变得非常容易，亲切，有趣的编程语言。Small Basic 的目的 在于消除障碍，充当通往令人惊奇的计算机编程世界的踏脚石。

Small Basic 环境

让我们从对 Small Basic 环境的一个快速介绍开始。当你第一次运行 SmallBasic.exe，你会看到一个如下图所示的窗口。

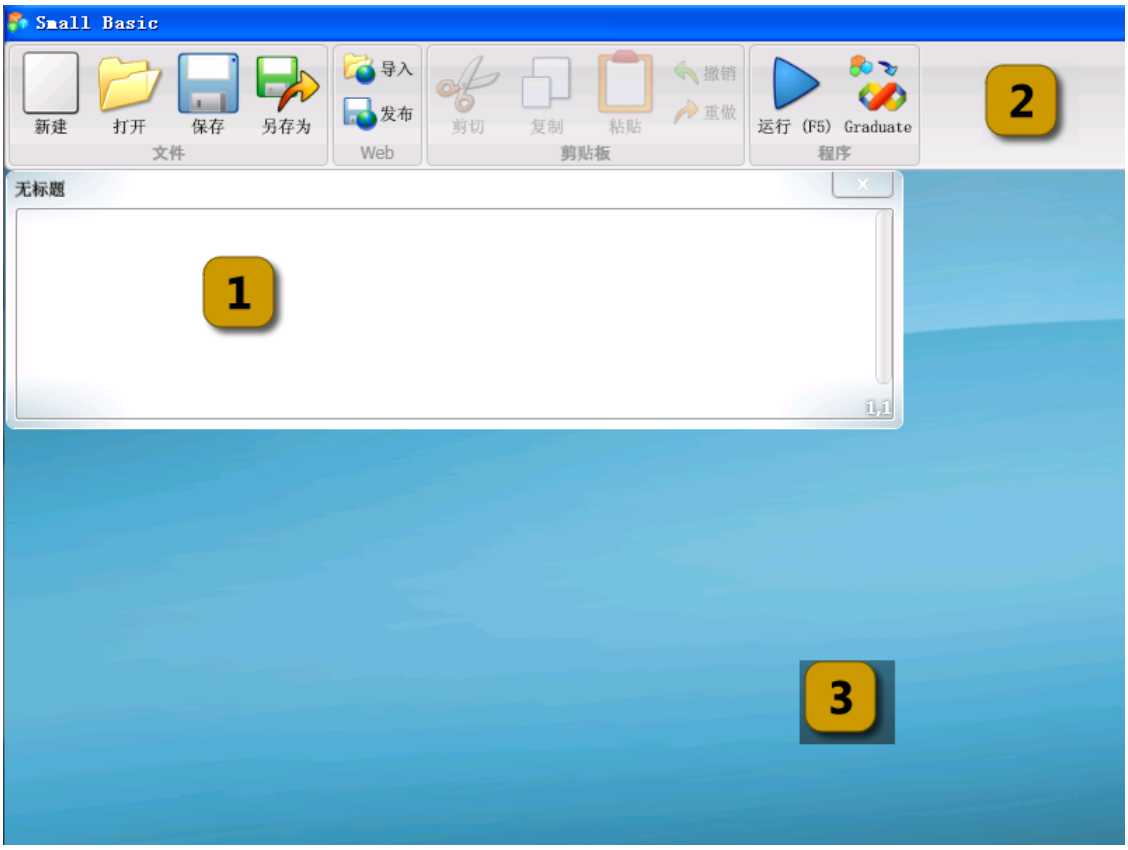

图 1 - Small Basic 环境

这就是 Small Basic 环境，我们将在这里编写和执行我们的 Small Basic 程序。这个环境有几个截然不同的部分。图中已经用数字标出。

编辑器，即标记为[1]的部分，我们将用来写我们的 Small Basic 程序。当你打开一个示例程序或者一个先前保存过的程序，它将显示在这个编辑器里。这样你就可以对其进行更改并且保存以备后用。

你也可以同时打开和工作在多个程序上。每个打开的程序将被显示在单独的编辑器里。包含你当前正工作在上面的程序的编辑器被称为*活动编辑器*。

工具栏，即标记为[2]的部分，是被用来向*活动编辑器*或环境发布命令的。随着我们的进展，我们将学到工具栏中各种各样的命令。

表面，即标记为[3]的部分，用来放置所有编辑器窗口。

我们的第一个程序

既然你已经熟悉了 Small Basic 环境，我们将进而在里面开始编程。正如我们刚刚在上面提到的，编辑器是我们写程序的地方。所以，让我们先在编辑器里输入下面这行。

```
TextWindow.WriteLine("世界你好")
```

这是我们的第一个 Small Basic 程序。如果你输入正确，你应该看到与下图相似的结果。

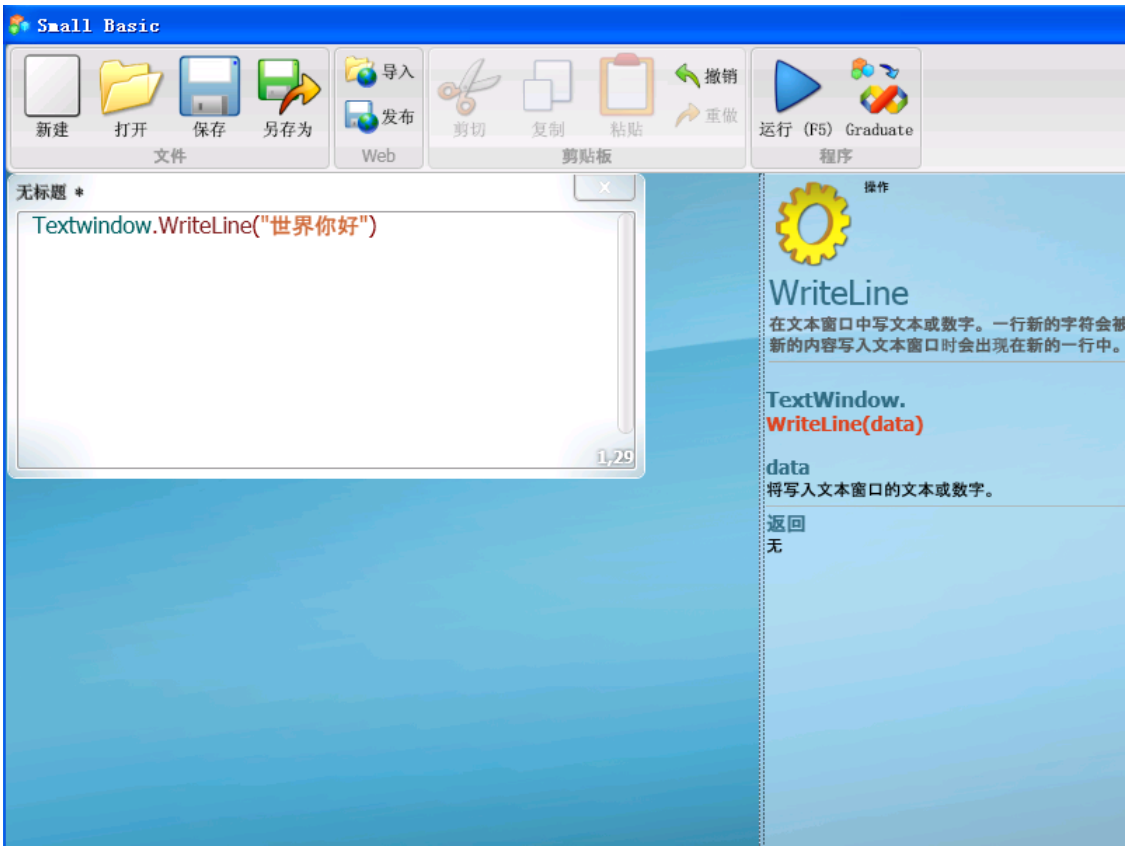

图 2 – 第一个程序

既然我们已经输入了我们的新程序，让我们来运行它看看会发生什么。我们可以通过点击工具栏上的*运行*按钮或者使用键盘上的 F5 快捷键来运行我们的程序。如果一切顺利，我们的程序将运行并得到如下图所示的结果。

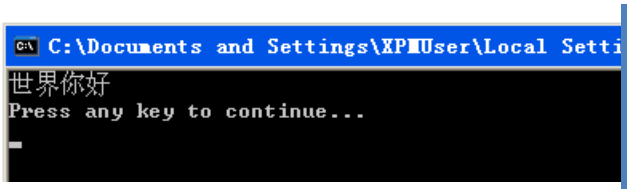

图 3 – 第一个程序的输出

在你输入你的第一个程序时，你可能已经注意到一个弹出窗口及一列条目（图 4）。这被称作“intellisense”（智能感知）。它帮助你能更快的输入你的程序。你可以通过按上/下箭头键来遍历这个列表。当你找到你想要的，你可以按回车键来将选中的条目插入到你的程序中。

恭喜！你已经编写并运行了第一个 Small Basic 程序。一个非常简单的小程序，然而却是通向成为一个真正计算机程序员的一大步！现在，在继续创建更大的程序之前，还有一个细节需要了解。我们必须要了解刚刚发生了什么——

我们到底告诉了计算机什么并且计算机是如何知道要做什么的？在下一章，我们将分析我们刚刚写的程序，从而对其进行理解。

图 4 – Intellisense （智能感知）

保存我们的程序

如果你想关闭 Small Basic 并且打算以后回来在你 刚输入的程序上继续工作，你可以保存程序。随时保存程序实际上是个很好的习惯，以至于你在意外关闭或断电时不会丢失信息。你可以通过点击工具栏上的“保存”图标或者使用快捷键“Ctrl+S”（在按下 Ctrl 键时按 S 键）来保存当前程序。

第二章

解读我们的第一个程序

什么是真正的计算机程序?

一个程序是一组计算机指令。这些指令准确地告诉计算机做什么，并且计算机总是遵循这些指令。就像人们一样，计算机只能遵循用它们能明白的语言指定的指令。这些语言被称为编程语言。有非常多的语言计算机能明白，**Small Basic** 是其中之一。

假象你和你的朋友之间有个会话。你和你的朋友用词语组成句子来彼此传递信息。相似地，编程语言包含词语的集合用来组成句子来将信息传递给计算机。程序基本上是多组语句（有时很少，有时多达数千）一起用同样的方式 让程序员和计算机彼此明白。

有很多计算机能明白的语言。Java, C++, Python, VB, 等都是强大的现代计算机语言。它们被用来开发简单到复杂的软件程序。

Small Basic 程序

一个典型的 Small Basic 程序由一组*语句*组成。 程序的每一行代表一条语句。每条语句是给计算机的一条指令。当我们让计算机执行一个 Small Basic 程序时，它取过程序并从第一个语句读起。它明白我们说的是什么，并且执行我们的指令。一旦执行完第一条语句，它回到程序继续读取并执行第二行。如此继续下去直到程序的结尾。至此，我们的程序执行完毕。

回到我们的第一个程序

这是我们写的第一个程序：

```
TextWindow.WriteLine("世界你好")
```

这是一个非常简单的只包含一条*语句*的程序。这条语句告诉计算机写一行内容为**世界你好**的文字到 Text Window。

这条语句在计算机里可逐字翻译成：

```
写世界你好
```

你可能已经注意到这条语句可以依次分解成更小的片断，就像句子可以分解成词语一样。在第一条语句中，我们有三个清楚的片段：

a) TextWindow
b) WriteLine
c) “世界你好”

点号，圆括号和引号都是必须被放到语句中恰当位置的标点符号，从而使计算机能够明白我们的意图。

你可能还记得在我们运行我们的第一个程序时出现的黑色窗口。这个黑色窗口叫做 TextWindow，有时也被称为控制台。那就是程序的结果输出的地方。**TextWindow**，在我们的程序中被叫做*对象*。在我们的程序中有许多这样的对象供我们使用。我们可以对这些对象执行一些不同的*操作*。在我们的程序里，我们已经使用了 *WriteLine* 操作。你可能还注意到 WriteLine 操作后面跟着放在引号中的**世界你好**。这段文字被作为输入传递给 WriteLine 操作，然后打印出来给用户。这被称为对该操作的 一个*输入*。一些操作接受一个或者多个输入，而其他的则不需要任何输入。

我们的第二个程序

既然你已经明白了我们的第一个程序，让我们继续加一些颜色来让它更新奇。

标点符号如引号，空格以及圆括号在计算机程序中非常重要。基于它们的位置和数量，它们可以改变所要表达的意思。

```
TextWindow.ForegroundColor =
"Yellow"
TextWindow.WriteLine("世界你好")
```

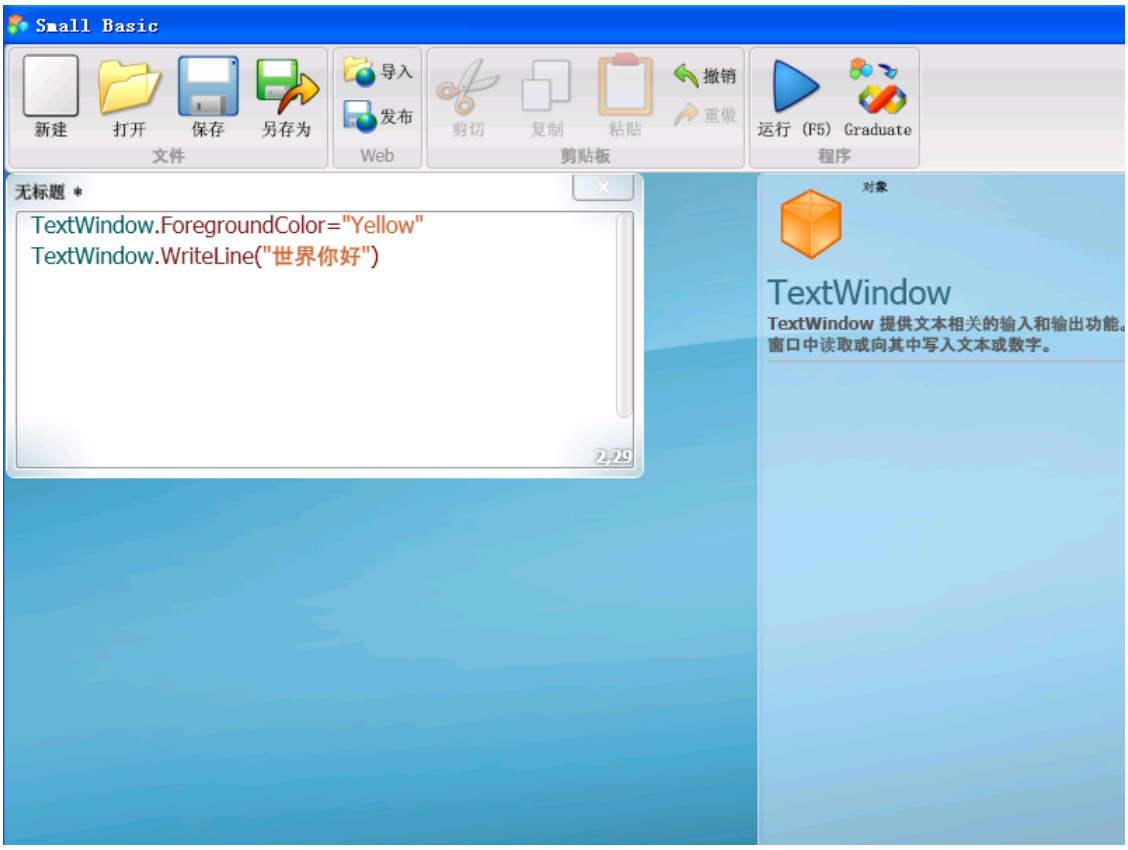

图 5 – 增加颜色

当你运行上面的程序，你将注意到在 TextWindow 中输出的还是“世界你好”，但是这次字体是黄色的而不是之前的灰色。

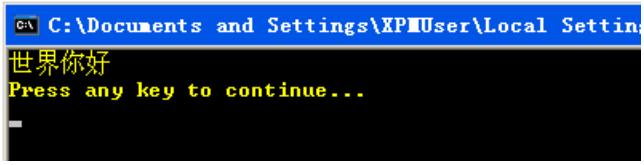

图 6 – 黄色的世界你好

注意加到我们最初的程序中的这条新语句。它使用了一个新单词， *ForegroundColor*（前景色），我们让它等于 “*Yellow*”。这意味着我们把“Yellow”指定给 *ForegroundColor*。现在，ForegroundColor 和 WriteLine 操作的区别是 ForegroundColor 没有接受任何输入而且也没有圆括号。取而代之的是跟了一个*等于*符号和一个词。 我们定义 ForegroundColor 为 TextWindow 的*属性*。下面是一些对 ForegroundColor 属性的有效值。试着用它们中的一个替换“Yellow”来看看结果 —— 不要忘记引号，这是必要的。

```
Black
Blue
Cyan
```

```
Gray
Green
Magenta
Red
White
Yellow
DarkBlue
DarkCyan
DarkGray
DarkGreen
DarkMagenta
DarkRed
DarkYellow
```

第三章

引入变量

在我们的程序中使用变量

如果我们的程序能够对用户的姓名说“你好”而不只是单纯的“世界你好”，那样不是更好吗？为了能那样做，我们必须首先让用户告知他/她的姓名并将其保存到某个地方，然后输出用户的姓名和“你好”。让我们一起来看看这是如何做到的：

```
TextWindow.Write("输入你的姓名：")
name = TextWindow.Read()
TextWindow.WriteLine(name + "你好")
```

当你输入并执行这个程序，你将看到如下输出：

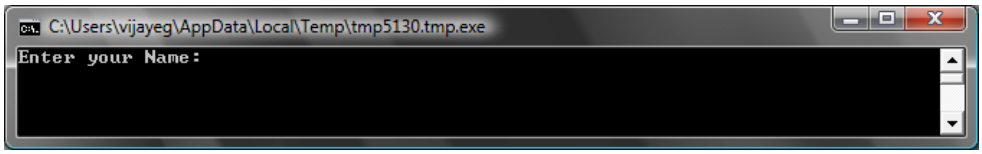

图 7 – 询问用户姓名

当你输入你的姓名并按 ENTER 键，你将看到如下输出：

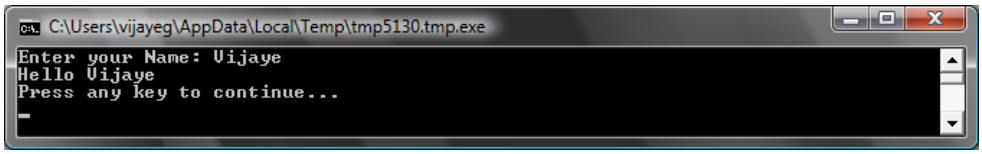

图 8 – 一个温馨的问候

现在，如果你再运行这个程序，你将被再次问同样的问题。你可以输入一个不同的姓名，计算机将对这个姓名说你好。

程序解析

在你刚刚运行的程序中，可能引起你注意的那行程序是：

```
name = TextWindow.Read()
```

Read() 看上去就像 *WriteLine()*，但是没有输入。它是一个操作并且基本上是告诉计算机等待用户输入些什么并按下 ENTER 键。一旦用户按下 ENTER 键，它将获取用户的输入并将其返回给程序。有趣的一点是无论用户输入的什么，现在都被存放在一个叫 **name** 的*变量*中。一个变量被定义为用来临时存储数值以备以后使用的地方。在上面的程序行中，**name** 被用来存储用户的姓名。

下面的一行也很有趣：

```
TextWindow.WriteLine(name + "你好")
```

这是我们使用存放在我们的变量，**name** 中的值的地方。我们取出 name 中的值并将它与“你好”一起写到 TextWindow。

一旦一个变量被设定，你可以多次使用它。例如，你可以如下这么做：

> Write，*与*WriteLine *相似，是 ConsoleWindow 上又一个操作。Write 允许你向 ConsoleWindow 写东西，但是后续文本将与当前文本在同一行。*

```
TextWindow.Write("输入你的姓名：
")
name = TextWindow.Read()
TextWindow.Write("你好，" + name + "。   ")
TextWindow.WriteLine("你最近怎么样，" + name + "？")
```

你将会看到如下的输入：

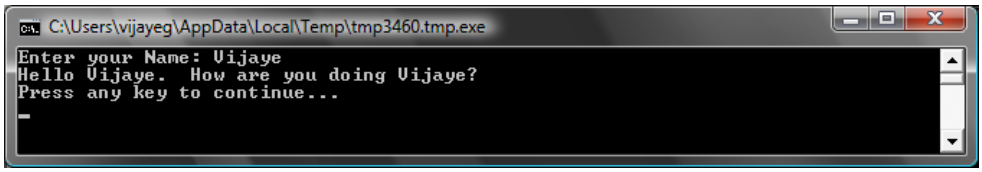

图 9 – 变量重用

变量命名规则

[TODO] 未完成

使用数字

我们刚才看到如何用变量存储用户的姓名。在接下来的程序中，我们将看到如何用变量存储和操作数字。让我们从一个非常简单的程序开始：

```
number1 = 10
number2 = 20
number3 = number1 + number2
TextWindow.WriteLine(number3)
```

当你运行这个程序，你将得到下面的结果：

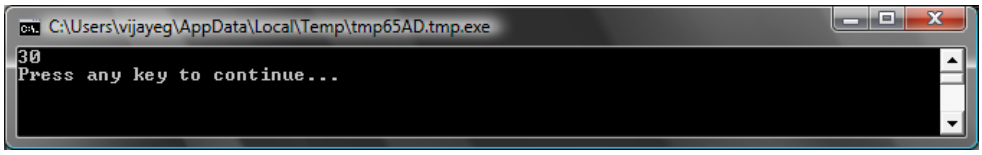

图 10 – 两数相加

在程序的第一行，你将变量 **number1** 赋值为 10。在第二行，你将变量 **number2** 赋值为 20。在第三行，你将 **number1** 和 **number2** 相加并将结果赋值给 **number3**。 因此，在这种情况下，**number3** 的值将是 30。并且这就是我们输出到 TextWindow 的结果。

现在，让我们对程序做轻微的修改并看看结果：

```
number1 = 10
number2 = 20
number3 = number1 * number2
TextWindow.WriteLine(number3)
```

注意数字没有放在引号中。对于数字，不需要引号。只有在使用文本时，才需要引号。

上面的程序将 **number1** 与 **number2** 相乘并将结果存放在 **number3** 中。你可以看到如下的程序运行结果：

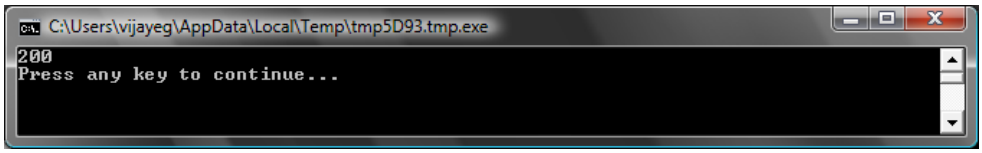

图 11 – 两数相乘

相似的，你可以对数字作减法或除法。这是减法：

```
number3 = number1 - number2
```

除法的符号是‘/’。程序看上去就像这样：

```
number3 = number1 / number2
```

这个除法的结果是：

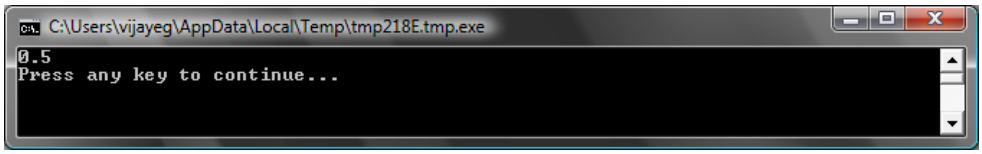

图 12 – 两数相除

一个简单的温度转换器

下一个程序我们将用公式 $℃ = \frac{5(°F-32)}{9}$ 将华氏温度转换成摄氏温度。

首先，我们从用户那里得到华氏温度并存放到一个变量中。这里有一个特殊的操作能让我们从用户那里读取数字，它就是 **TextWindow.ReadNumber**。

```
TextWindow.Write("输入华氏温度：  ")
fahr = TextWindow.ReadNumber()
```

一旦我们有存放在变量中的华氏温度，我们可以像这样将它转换为摄氏温度：

```
celsius = 5 * (fahr - 32) / 9
```

圆括号告诉计算机先计算 **fahr – 32** 然后再处理其它的。现在我们需要做的就是将结果输出给用户。将所有这些放到一起，就是我们的程序：

```
TextWindow.Write("输入华氏温度：  ")
fahr = TextWindow.ReadNumber()
celsius = 5 * (fahr - 32) / 9
TextWindow.WriteLine("摄氏温度是 " + celsius)
```

程序的运行结果是：

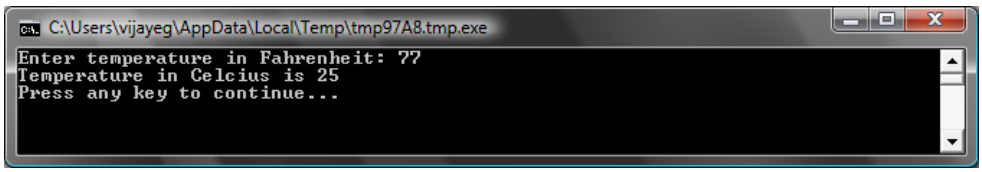

图 13 – 温度转换

第 4 章

条件与分支

回到第一个程序。如果我们的程序不仅仅会说“*Hello World* （你好，世界）”； 而是可以根据一天中的不同时间段说“*Good Morning World （早上好，世界）”*, 或者“ *Good Evening World （晚上好，世界）*”，那是不是一件很酷的事情？在下一个程序中， 我们将让计算机能够在中午 12 点以前说“*Good Morning World （早上好，世界）*”，在中午 12 点以后说“*Good Evening World （晚上好，世界）*”。

```
If (Clock.Hour < 12) Then
  TextWindow.WriteLine("Good Morning World")
EndIf
If (Clock.Hour >= 12) Then
  TextWindow.WriteLine("Good Evening World")
EndIf
```

根据你运行程序的时间，你将会看到下面的某一个输出。

图 14 - Good Morning World （早上好，世界）

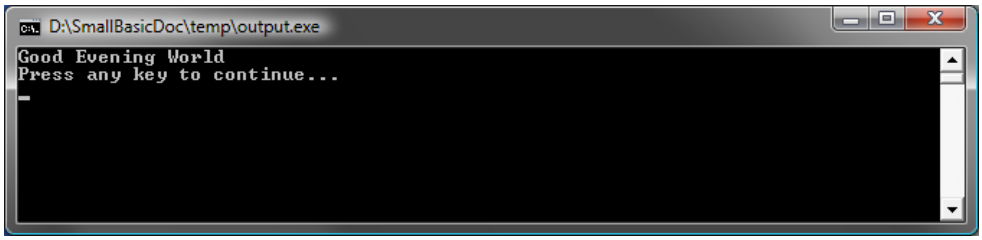

图 15 - Good Evening World （晚上好，世界）

让我们来分析一下程序的前三行。你肯定已经发现这几行程序告诉计算机如果 Clock.Hour 小于 12，就输出“Good Morning World. （早上好，世界。）”。**If**， **Then** 和 **EndIf** 是在程序运行过程中计算机能够理解的特殊语言。**If** 后面总是会跟一个条件判断。在这个例子中是（**Clock.Hour < 12**）。记住，为了让计算机理解你的意图，小括号在这里是必须的。条件判断后面跟的是 **then** 和要执行的实际操作。实际操作后面跟的是 **EndIf**， 这告诉计算机这个有条件的执行过程结束了。

在 Small Basic 中，你可以用 Clock 对象来访问当前的日期和时间。它还提供了一组属性，使你可以分别得到当前的日、月、年、小时、分钟、秒。

在 **then** 和 **EndIf** 之间可能会有不只一个操作。如果条件符合，计算机将会把它们全部执行。例如，你可以写如下程序：

```
If (Clock.Hour < 12) Then
  TextWindow.Write("Good Morning. ")
  TextWindow.WriteLine("How was breakfast?")
EndIf
```

Else

在这一章最开始的程序中，你可能已经注意到第二个条件有点多余。**Clock.Hour** 的值只可能小雨或者大于 12。 我们不需要真正检查第二个条件。对于这种情况，我们可以通过 **else**， 把两个 **if..then..endif** 语句缩短为一句。

如果我们用 **else** 重写，你将看到的下面的程序：

```
If (Clock.Hour < 12) Then
  TextWindow.WriteLine("Good Morning World")
Else
  TextWindow.WriteLine("Good Evening World")
EndIf
```

这个程序将和最开始的程序做完全同样的事情。这里我们学到了计算机编程中非常重要的一课：

> *在编程过程中，我们经常可以通过很多方法做同样的事情。很多时候，一种方法比其它的方法更有效。选择哪一种方法是由程序员决定的。当你编写了更多的程序获得了更多的经验后，你就可以了解不同的技术以及它们的优缺点。*

Indentation（缩进）

在所有的程序中，你都可以看到 *If, Else* 和 *EndIf* 是如何缩进的。缩进不是必须的，但它将帮助你更容易的理解程序的结构。因此，在程序块间缩进语句通常是一个很好的习惯。

Even（奇数）或者 Odd （偶数）

现在我们已经学会了 **If..Then..Else..EndIf** 语句。让我们写一个程序。 当你给出一个数字时，它可以判断是奇数还是偶数。

```
TextWindow.Write("Enter a number: ")
num = TextWindow.ReadNumber()
remainder = Math.Remainder(num, 2)
If (remainder = 0) Then
  TextWindow.WriteLine("The number is Even")
Else
  TextWindow.WriteLine("The number is Odd")
EndIf
```

运行这个程序，你将能看到它的输出如下：

图 16 – 奇数或偶数

在这个程序中，我们引入了一个新的很有用的操作，**Math.Remainder** 。是的，正如你已经看到的，**Math.Remainder** 将返回第一个数除以第二个数的余数。

Branching（分支）

还记得吗？在第二章中，你知道了计算机按从上到下的顺序一次处理程序中的一个语句。然而，有一种语句可以使计算机不按照顺序执行，而跳到另一条语句。我们看一下下面的程序。

```
i = 1
start:
TextWindow.WriteLine(i)
i = i + 1
If (i < 25) Then
  Goto start
EndIf
```

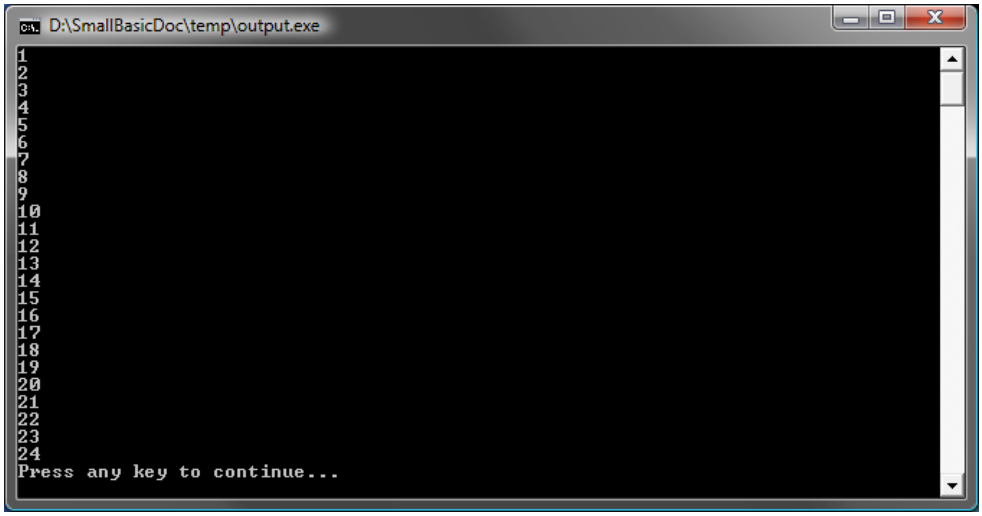

图 17 – 使用 Goto

在上面的程序中，我们把 1 付给变量 **i** 。接着，我们加了一个新的语句，它以冒号结束（：）

```
start:
```

这被称作一个标记。标记就像是一个计算机可以理解的书签。只要你能保证它们的命名是唯一的，你可以任意命名一个标记，而且可以在程序中加入任意多的标记。

另一个有趣的语句是：

```
i = i + 1
```

这句话告诉计算机把变量 **i** 加 1，然后再把结果付回给 **i** 。所以如果原来 **i** 的值是 1，在执行这句话之后，**i** 的值将变成 2。

最后，

```
If (i < 25) Then
  Goto start
EndIf
```

这部分程序告诉计算机，如果 **i** 的值小于 25，就开始从标记 **start** 执行语句。

Endless execution （无限执行）

使用 **Goto** 语句，你可以让计算机任意多次的执行一些操作。例如，你可以把奇数或偶数的程序改写成下面的样子。程序将永远执行。你可以通过点击命令框右上角的 Close（X）来终止程序。

```
begin:
TextWindow.Write("Enter a number: ")
num = TextWindow.ReadNumber()
remainder = Math.Remainder(num, 2)
If (remainder = 0) Then
  TextWindow.WriteLine("The number is Even")
Else
  TextWindow.WriteLine("The number is Odd")
EndIf
Goto begin
```

图 18 – 奇数或偶数程序无限执行

第 5 章

循环

For 循环

让我们再以前面章节用到的代码来举例说明。

```
i = 1
start:
TextWindow.WriteLine(i)
i = i + 1
If (i < 25) Then
  Goto start
EndIf
```

这段代码的运行结果是：按顺序打印数字 1 到 24。在提供了便捷的变量递增方法的编程语言中，这样的变量递增过程是很常见的。以上这段程序与下面的程序输出结果相同：

```
For i = 1 To 24
  TextWindow.WriteLine(i)
EndFor
```

输出结果是：

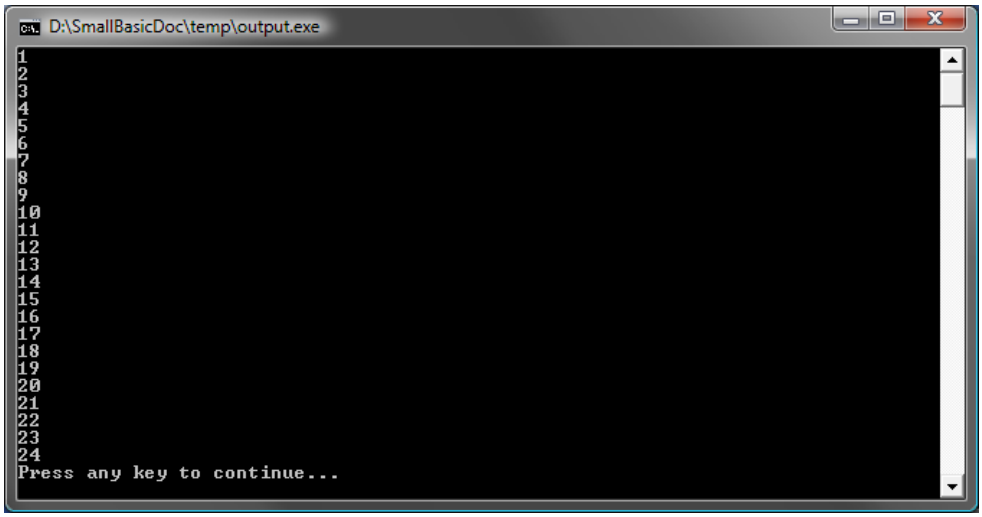

图 19 – 使用 For 循环

请注意，我们将 8 行的程序减少到 4 行，而实际运行结果与 8 行的程序完全相同！还记得前面我们提到的“完成一个任务通常会有多种方法”吗？这就是一个典型的例子。

For..EndFor 在编程术语中被称为一个*循环*。在循环中可以使用一个变量，给变量赋以初值和终值，并让计算机自动执行变量递增。变量每递增一次，计算机便执行一次 **For** 到 **EndFor** 之间的程序。

如果希望变量以 2（而不是以 1）为基准递增，例如，要输出 1 至 24 之间的所有奇数，即可以使用循环来实现。

```
For i = 1 To 24 Step 2
  TextWindow.WriteLine(i)
EndFor
```

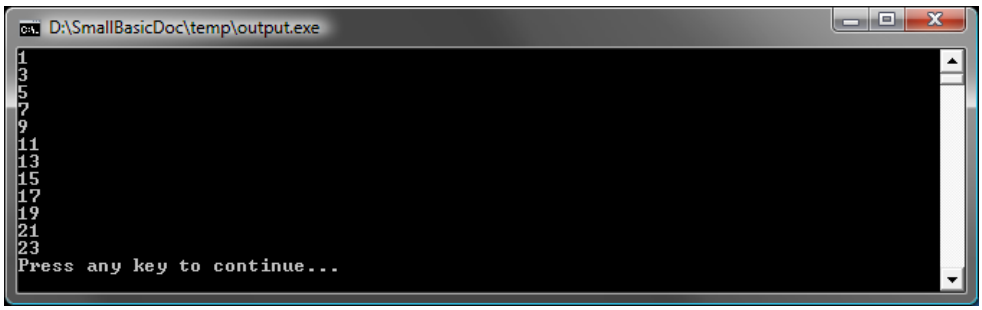

图 20 – 仅输出奇数

For 语句中的 **Step 2** 告诉计算机以 2 为基数对变量 **i** 递增（而不是以 1）。通过使用 **Step** 可以根据需要指定任意递增基数。甚至可以指定负增量来进行倒数，示例如下：

```
For i = 10 To 1 Step -1
  TextWindow.WriteLine(i)
EndFor
```

图 21 – 倒数

While 循环

While 循环是另一种循环方法，该方法在无法预测循环次数的情况下非常实用。前面介绍的 For 循环按照预先定义的次数执行程序，而 While 循环则是在指定条件为真的情况下重复运行程序。以下示例中，我们将在指定变量大于 1 的情况下，对变量进行等分。

```
number = 100
While (number > 1)
  TextWindow.WriteLine(number)
  number = number / 2
EndWhile
```

图 22 – 等分循环

在以上程序中，对变量 *number* 赋以初值 100，在 *number* 大于 1 的情况下重复运行 While 循环。在循环中，输出 *number* 的值，再除以 2，对其进行有效地等分。正如预期，程序的输出结果为逐步被等分的序列数字。

因为我们无法事先得知程序要循环运行的次数，如果用 For 循环实现以上程序就会相当困难。通过使用 While 循环，可以方便地检查条件，再确定是否要继续执行循环或退出。

有趣的是，每个 While 循环都可以分解成一组 If..Then 语句。例如，可以将以上程序重写如下，其输出结果相同：

```
number = 100
startLabel:
TextWindow.WriteLine(number)
number = number / 2

If (number > 1) Then
  Goto startLabel
EndIf
```

实际上，计算机内部会将每个 While 循环重写成一组带有一个或多个 Goto 语句的 If..Then 语句。

第 6 章

图形入门

目前为止，在所有例子中，我们都用 TextWindow 解释 Small Basic 语言的基础知识。然而，Small Basic 还有一组强大的图形功能。我们将在这章中学习。

GraphicsWindow 引言

就如 TextWindow 使我们可以处理 Text （文本）和 Numbers（数字）, Small Basic 提供了 **GraphicsWindow**，我们可以用它来画图。让我们从显示 GraphicsWindow 开始。

```
GraphicsWindow.Show()
```

当你运行这个程序的时候，你会发现你得到的是一个下图所示的白色 Window（窗体），而不是一个通常情况下的空的文本窗体。这个窗体本身没什么用，但这章你所有的工作都将基于它。你可以通过右上角的‘X’来关闭这个窗口。

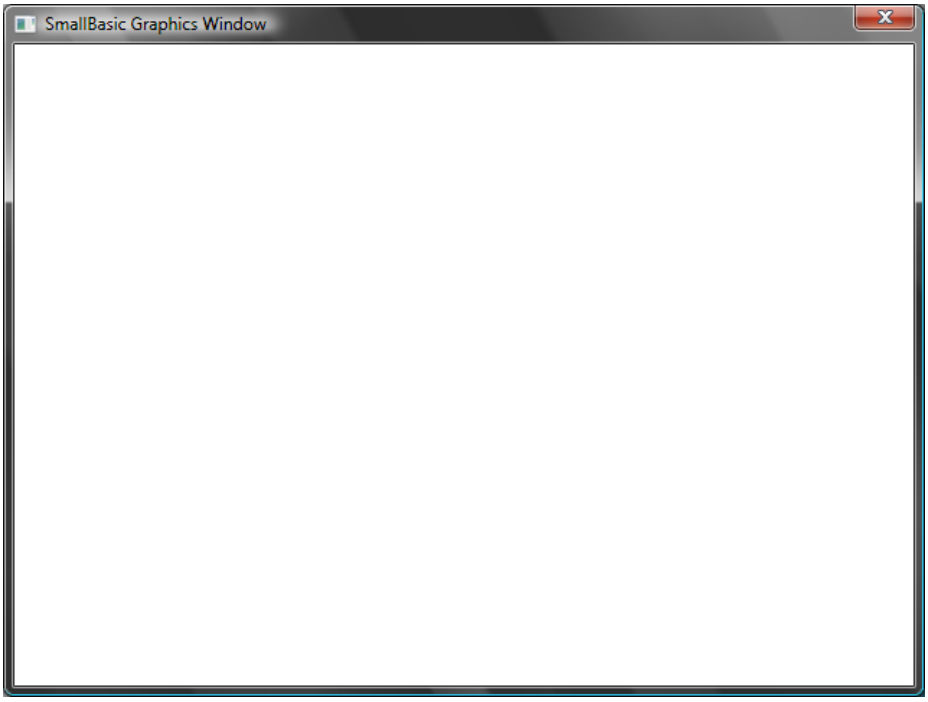

图 23 – 一个空的 Graphics Window

设置 Graphics Window

这个图形窗口允许你根据自己的需要调整它的外观。你可以改变它的标题、背景和大小。让我们继续对它进行一些修改。这将帮助我们熟悉窗体。

```
GraphicsWindow.BackgroundColor = "SteelBlue"
GraphicsWindow.Title = "My Graphics Window"
GraphicsWindow.Width = 320
GraphicsWindow.Height = 200
GraphicsWindow.Show()
```

下面是一个个性化的图形窗口的样子。你可以把它的背景颜色改变成附录 B 中列出的任意一种颜色。试试这些属性看看窗体的外观是如何改变的。

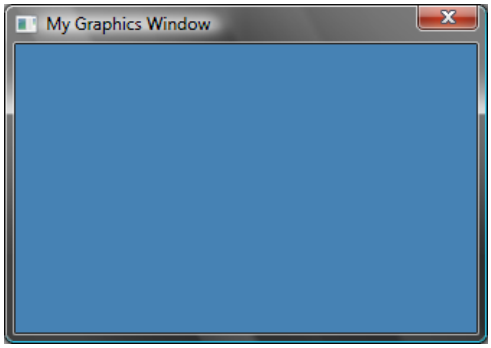

图 24 – 一个 Custom Graphics Window （个性化图形窗体）

画线

一旦你有了 GraphicsWindow，我们可以在它上面画图形、文本甚至是图片。让我们先画一些简单的图形。下面是在 Graphics Window 上画两条线的程序。

```
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200
GraphicsWindow.DrawLine(10, 10, 100, 100)
GraphicsWindow.DrawLine(10, 100, 100, 10)
```

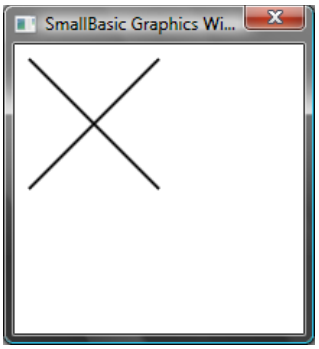

图 25 – 交叉线

这两行程序设置了窗体并且画了两条交叉线。*DrawLine* 后面紧跟的前两个数字指定了线开始端的 x，y 坐标；另外两个数字指定了线结束端的 x，y 坐标。计算机图形中有趣的一件事是（0，0）坐标代表了窗体的左上角。实际上，窗体被认为是在坐标系的第二象限。

我们使用 web 颜色符号（(#RRGGBB）代表颜色，而不是用名字。例如，#FF0000 代表红色，#FFFF00 代表黄色，依此类推。我们将会学更多的关于颜色的知识。[TODO Colors chapter]

[TODO: 插入 象限图]

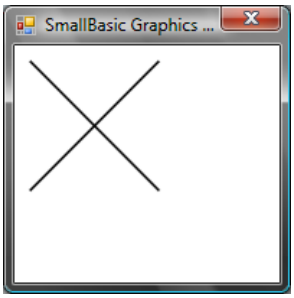

图 26 – 坐标图

如果我们回到画线的程序，有趣的是 Small Basic 允许你修改线的属性。比如颜色和它的宽度。首先，我们用下面的程序改变线的颜色。

```
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200
GraphicsWindow.PenColor = "Green"
GraphicsWindow.DrawLine(10, 10, 100, 100)
GraphicsWindow.PenColor = "Gold"
GraphicsWindow.DrawLine(10, 100, 100, 10)
```

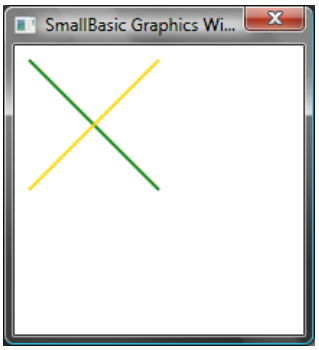

图 27 – 改变线的颜色

现在，我们改变大小。在下面的程序中，我们把线的宽度从默认的 1 改为 10。

```
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200
GraphicsWindow.PenWidth = 10
GraphicsWindow.PenColor = "Green"
GraphicsWindow.DrawLine(10, 10, 100, 100)
GraphicsWindow.PenColor = "Gold"
```

```
GraphicsWindow.DrawLine(10, 100, 100, 10)
```

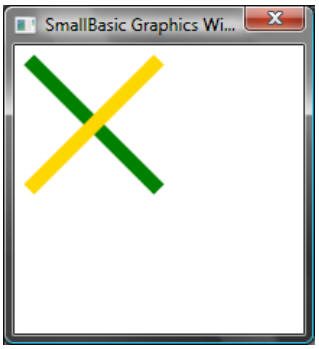

图 28 – 粗的彩色线

PenWidth 和 *PenColor* *改变画线的笔*。它们不仅影响线，还会影响属性改变后画的所有图形。

使用上一章我们学的循环语句，我们可以轻松的写一个程序。这个程序可以画很多条线。画线用的笔会越来越粗。

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.Width = 200
GraphicsWindow.Height = 160
GraphicsWindow.PenColor = "Blue"

For i = 1 To 10
  GraphicsWindow.PenWidth = i
  GraphicsWindow.DrawLine(20, i * 15, 180, i * 15)
endfor
```

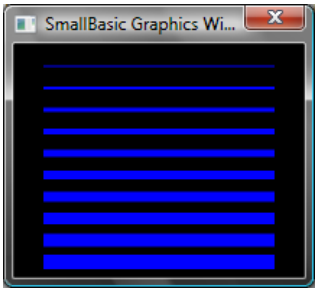

图 29 – 不同的笔的宽度

这个程序有趣的部分是循环语句。每循环一次，我们加大笔的宽度，然后在原来的线下面画一条新线。

画图和填充图形

说到画图形，对于每一个图形通常都有两类操作。它们分别是画操作和填充操作。画操作用笔画出图形的轮廓；填充操作用刷子为图形上色。例如在下面的程序中，有两个矩形。一个是用红笔画出来的，另一个是用绿色的刷子填充的。

```
GraphicsWindow.Width = 400
GraphicsWindow.Height = 300

GraphicsWindow.PenColor = "Red"
GraphicsWindow.DrawRectangle(20, 20, 300, 60)

GraphicsWindow.BrushColor = "Green"
GraphicsWindow.FillRectangle(60, 100, 300, 60)
```

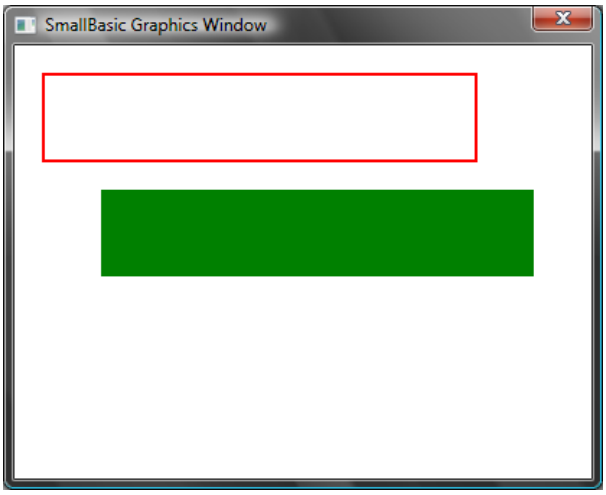

图 30 画图和填充

要画和填充一个矩形，你需要四个数字。前两个代表矩形左上角的 X 和 Y 坐标。第三个数字代表矩形的宽度，第四个代表它的高度。 实际上，画和填充椭圆也一样。看下面的程序。

```
GraphicsWindow.Width = 400
GraphicsWindow.Height = 300

GraphicsWindow.PenColor = "Red"
GraphicsWindow.DrawEllipse(20, 20, 300, 60)

GraphicsWindow.BrushColor = "Green"
GraphicsWindow.FillEllipse(60, 100, 300, 60)
```

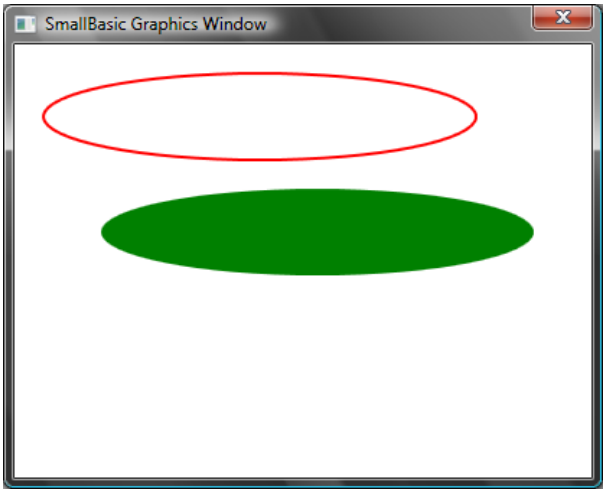

图 31 – 画和填充椭圆

椭圆形是广义上的圆形。如果你像画圆形，你要定义相同的宽度和高度

```
GraphicsWindow.Width = 400
GraphicsWindow.Height = 300

GraphicsWindow.PenColor = "Red"
GraphicsWindow.DrawEllipse(20, 20, 100, 100)

GraphicsWindow.BrushColor = "Green"
GraphicsWindow.FillEllipse(100, 100, 100, 100)
```

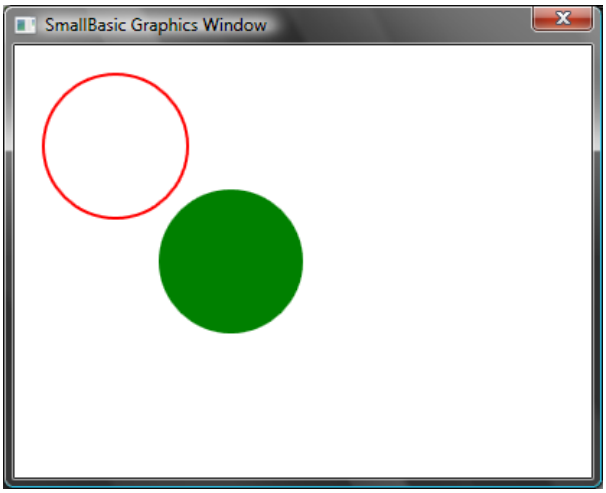

图 32 – 圆形

第 7 章

绘制图形

在这一章节中，我们将运用前面学到的知识来做一些有趣的事。在本章的示例中，我们将向你展示几个有趣的方法，让你可以结合目前所学到的全部知识来创建看上去非常酷的应用程序。

矩形

以下的示例利用一个循环语句绘制一组大小逐渐递增的矩形。

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightBlue"
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200

For i = 1 To 100 Step 5
  GraphicsWindow.DrawRectangle(100 - i, 100 - i, i * 2, i * 2)
EndFor
```

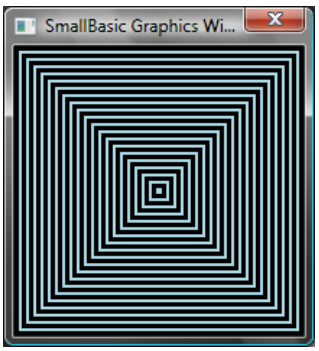

图 33 – 矩形组

圆形

在前面程序的基础上加以改变，绘制一组圆形（而不是矩形）。

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightGreen"
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200

For i = 1 To 100 Step 5
  GraphicsWindow.DrawEllipse(100 - i, 100 - i, i * 2, i * 2)
EndFor
```

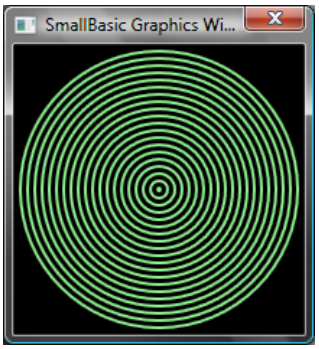

图 34 – 圆形组

随机的图案

以下程序利用 GraphicsWindow.*GetRandomColor* 为画笔设置随机颜色，在使用 Math.*GetRandomNumber* 设置圆形的 x 和 y 座标。这个有趣的程序将两种运算巧妙地结合，每次运行时都可以绘制出不同的图案。

```
GraphicsWindow.BackgroundColor = "Black"
For i = 1 To 1000
  GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
  x = Math.GetRandomNumber(640)
  y = Math.GetRandomNumber(480)
  GraphicsWindow.FillEllipse(x, y, 10, 10)
EndFor
```

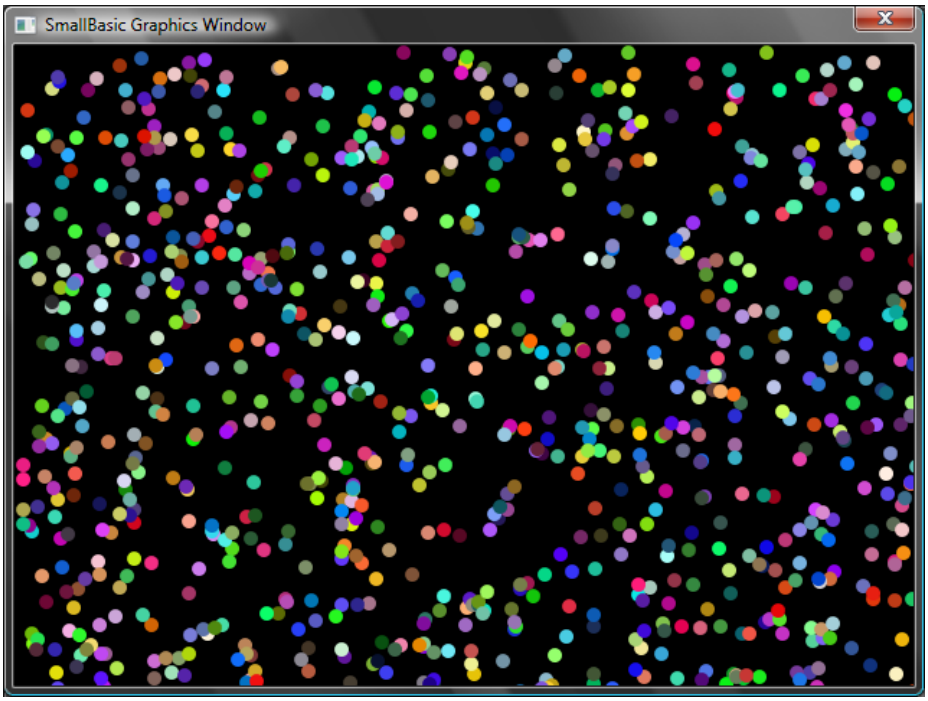

图 35 – 随机的图案

分形

以下程序使用随机数绘制一个简单的三角分形。分形是一种几何图形，它可以被分割成多个子图形，其中每个子图形都与该分形形状完全相同。在这个示例中，程序将绘制一个由几百个小三角形组成的大三角形，其中每个小三角形都与大三角形完全相同。由于这个程序需要运行几秒钟，我们可以在程序运行时看到三角形从小点开始，逐渐形成大三角形的过程。程序本身的逻辑很难描述，大家可以尝试研究一下。

```
GraphicsWindow.BackgroundColor = "Black"
x = 100
y = 100

For i = 1 To 100000
  r = Math.GetRandomNumber(3)
```

```
  ux = 150
  uy = 30
  If (r = 1) then
    ux = 30
    uy = 1000
  EndIf

  If (r = 2) Then
    ux = 1000
    uy = 1000
  EndIf

  x = (x + ux) / 2
  y = (y + uy) / 2

  GraphicsWindow.SetPixel(x, y, "LightGreen")
EndFor
```

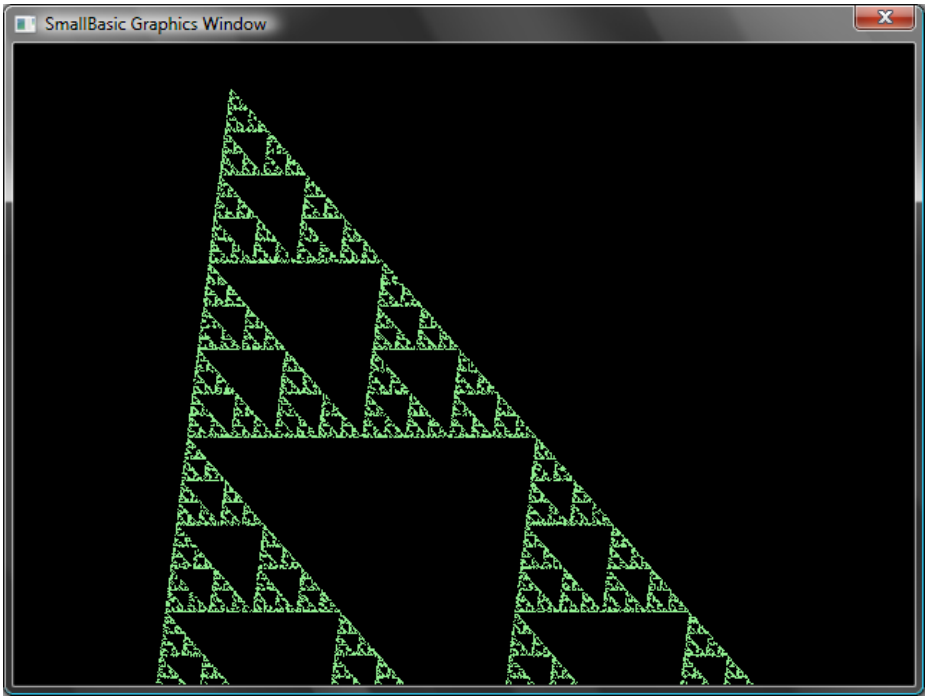

图 36 – 三角分形

如果你想认真观察小三角形逐渐形成三角分形的过程，可以在循环中引入 **Proram.***Delay* 延迟。这个运算通过读取指定的数值（以毫秒为单位），延迟程序的执行时间。以下为修改后的程序，修改的代码以粗体显示。

```
GraphicsWindow.BackgroundColor = "Black"
x = 100
y = 100

For i = 1 To 100000
  r = Math.GetRandomNumber(3)
  ux = 150
  uy = 30
  If (r = 1) then
    ux = 30
    uy = 1000
  EndIf

  If (r = 2) Then
    ux = 1000
    uy = 1000
  EndIf

  x = (x + ux) / 2
  y = (y + uy) / 2

  GraphicsWindow.SetPixel(x, y, "LightGreen")
  Program.Delay(2)
EndFor
```

通过增加延迟时间，可以让程序执行得更慢。你可以将延迟时间调整到自己喜欢的数值。

我们还可以对程序进行其它修改，将以下代码：

```
GraphicsWindow.SetPixel(x, y, "LightGreen")
```

替换为：

```
color = GraphicsWindow.GetRandomColor()
GraphicsWindow.SetPixel(x, y, color)
```

修改后的程序将以随机颜色绘制三角形。

第8章

Turtle Graphics（海龟绘图法）

Logo 语言

二十世纪 70 年代，有一种叫做 Logo 的编程语言，它非常简单但功能强大，当时被一些研究人员所使用。后来开发人员在这种编程语言中加入了现在被称为“Turtle Graphics（海龟绘图法）”的技术，并在屏幕上显示一个“海龟”图案，通过诸如 *Move Forward（向前移动）*，*Turn Right（向右转）*，*Turn Left（向左转）*等命令移动龟标。通过控制龟标的移动，编程人员可以绘制有趣的图形。这一新技术的引入，使得 Logo 变得更加简单易用，并对各个年龄的编程爱好者产生了吸引力，因此在 80 年代广为流行。

Small Basic 语言中也引入了 **Turtle**（龟标）对象，编程人员可以通过 Small Basic 自带的很多命令调用这个对象。在本章中，我们将使用龟标绘制图形。

Turtle

程序初始化时，我们需要在屏幕上显示龟标。代码如下：

```
Turtle.Show()
```

程序运行后，将显示一个白色窗口，这个窗口与我们在前面章节中看到的窗口唯一的不同就是窗口的中央多了一个龟标。这个龟标将会按照我们的指令绘制指定的图案。

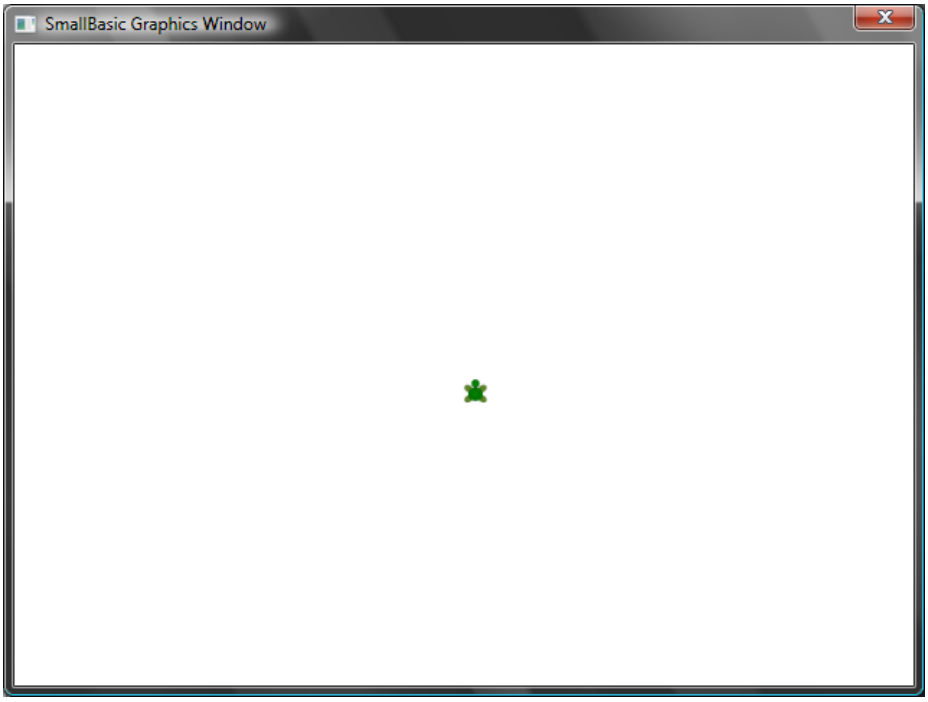

图 37 – 窗口中央显示有龟标

移动和绘制

Move（移动）是这只“海龟”可以听懂的命令之一。这个运算会读取用户输入的一个数值。该数值即是龟标要移动的距离。例如，在下面的代码中，我们指定龟标移动 100 象素。

```
Turtle.Move(100)
```

程序运行后，可以看到龟标缓慢地向上移动 100 象素，并在它经过的地方画下一条线。当龟标移动到指定位置后，将画出以下图案。

使用龟标绘制图形时，可以不调用 Show()。计算机在执行操作时会自动显示龟标。

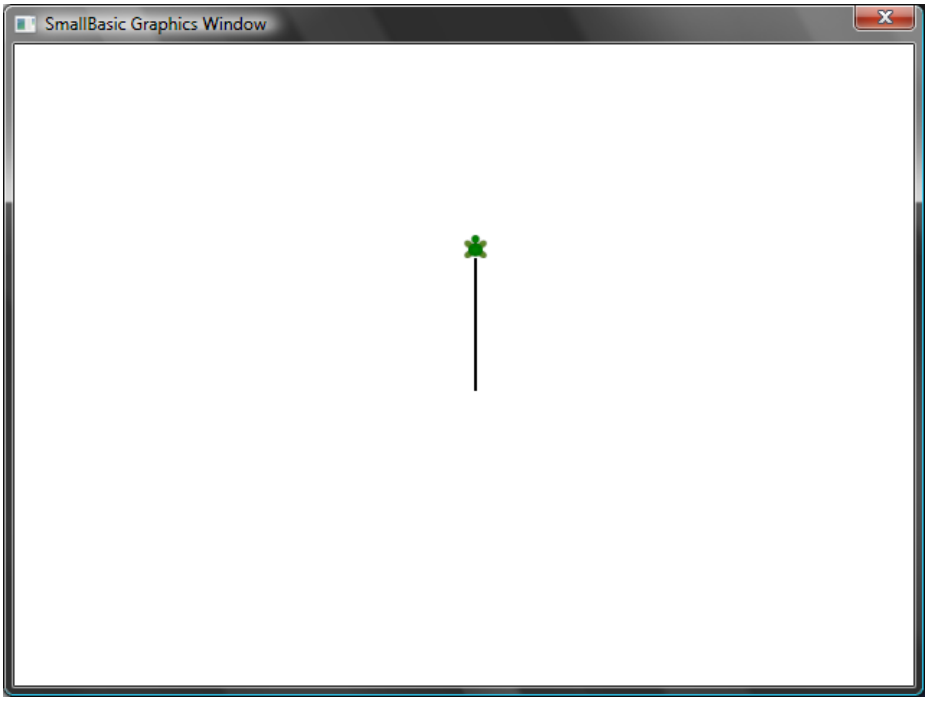

图 38 – 移动 100 象素

绘制正方形

正方形由四条边组成，两条为水平方向，两条为垂直方向。因此，要绘制一个正方形，我们需要用龟标画一条线，右转，再画一条线，然后重复同样的动作直至画完四条边。我们可以将这些动作转换成以下代码。

```
Turtle.Move(100)
Turtle.TurnRight()
Turtle.Move(100)
Turtle.TurnRight()
Turtle.Move(100)
Turtle.TurnRight()
Turtle.Move(100)
Turtle.TurnRight()
```

程序运行后，龟标将会逐条边绘制出一个正方形，其结果如下图。

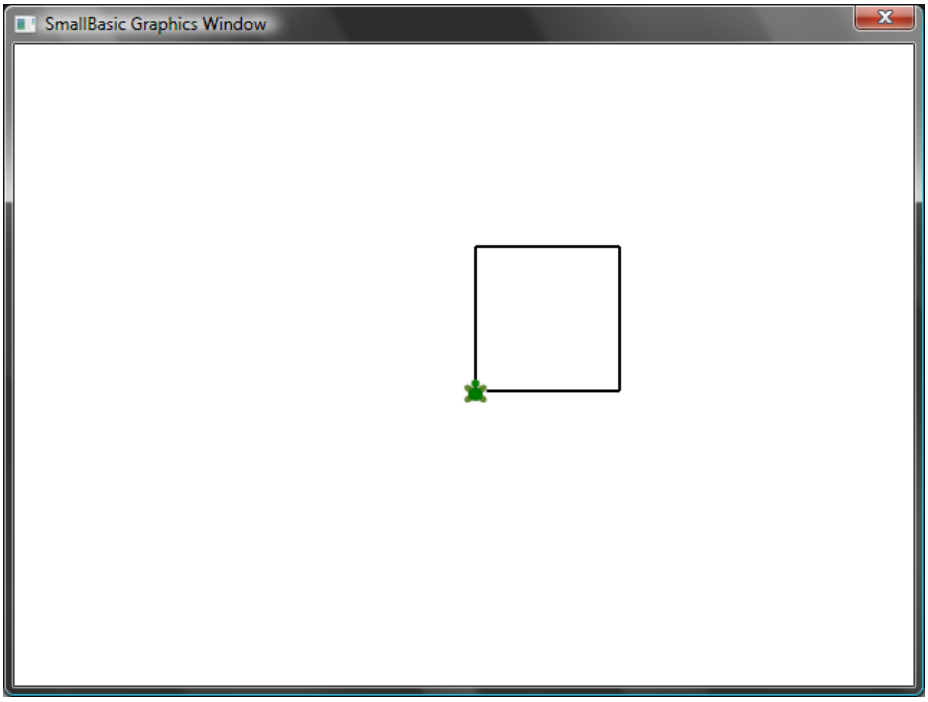

图 39 – 龟标绘制出一个正方形

如果用心观察，就会发现我们在一遍又一遍地使用同样的指令，确切地说是重复使用了四次。我们在前面介绍过，可以使用循环来实现像这样的重复命令。拿上面的这个程序来举例，如果用 **For..EndFor** 循环来实现，程序就会变得简单很多。

```
 For i = 1 To 4
  Turtle.Move(100)
  Turtle.TurnRight()
EndFor
```

修改颜色

这个示例中龟标所在的 GraphicsWindow （图画窗口）和我们在前一章中所看到的相同。也就是说，前面所学的运算都可以在这里使用。例如，以下这段代码可以绘制一个每条边颜色不同的方形。

```
For i = 1 To 4
  GraphicsWindow.PenColor = GraphicsWindow.GetRandomColor()
  Turtle.Move(100)
  Turtle.TurnRight()
EndFor
```

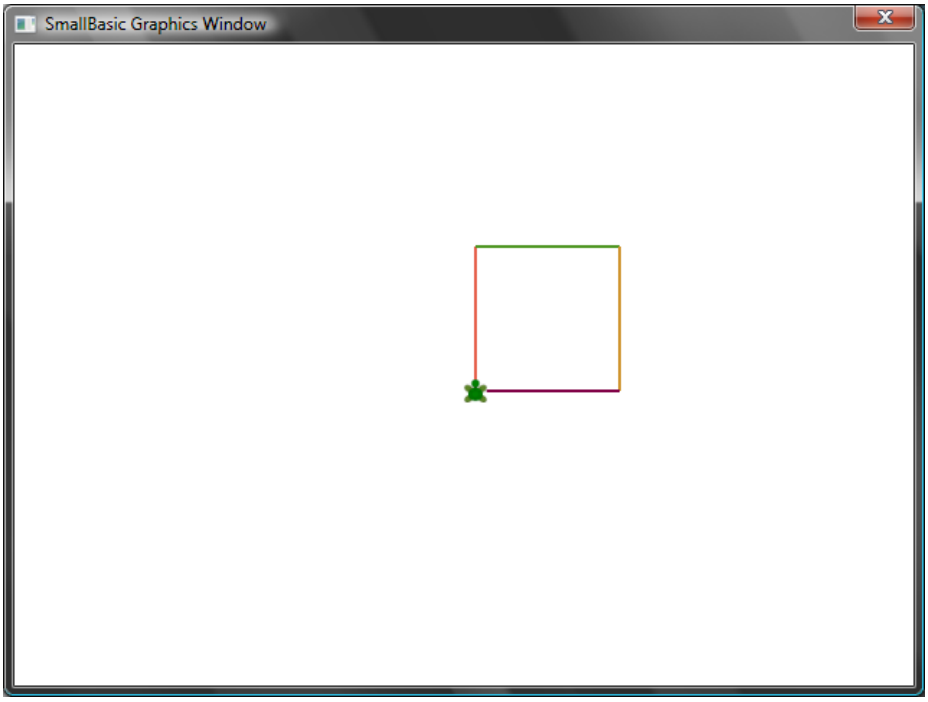

图 40 – 修改颜色

绘制复杂图形

龟标除了可以执行 **TurnRight** （向左转）和 **TurnLeft** （向右转）运算以外，还可以执行 **Turn** （转向）运算。这个运算会按照用户指定的数值将龟标旋转相应的角度。使用这个运算，我们就可以绘制任意形状的多边形。以下程序可以绘制一个六边形。

```
For i = 1 To 6
  Turtle.Move(100)
  Turtle.Turn(60)
EndFor
```

运行一下这个程序，看看它的输出是否真的是一个六边形。因为边与边的夹角是 60 度，我们就使用 **Turn(60)**。对于这样每个边都等长的多边形，边与边之间的夹角即是 360 除以多边形的边数。我们可以利用这个原理并通过使用变量，实现一个可以绘制任意边数的多边形的通用程序。

```
sides = 12

length = 400 / sides
angle = 360 / sides

For i = 1 To sides
```

```
  Turtle.Move(length)
  Turtle.Turn(angle)
EndFor
```

使用这个程序，我们可以通过改变 **sides** （边数）变量绘制任意多边形。如果将变量设置为 4，将会绘制出和我们开始编写的那个程序一样的四边形。如果将变量设置为一个很大的数值，比如 50，将会绘制出的一个接近圆形的图案。

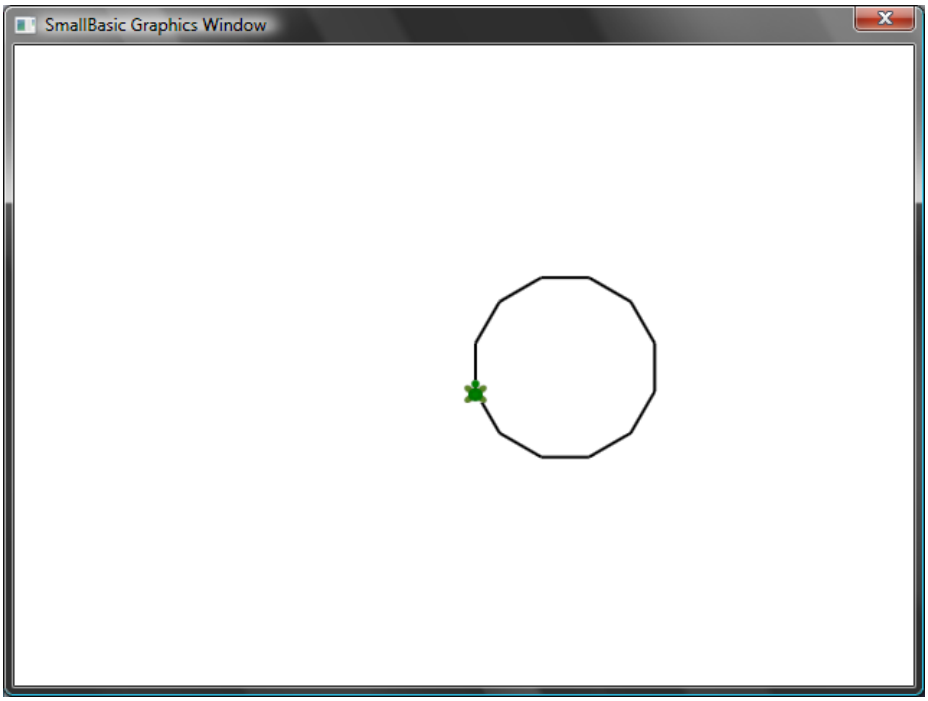

图 41 – 绘制一个十二边形

使用以上技术，利用龟标绘制多个圆形，并在绘制每个圆形之后移动一段距离，我们将会得到一个有趣的输出结果。

```
sides = 50
length = 400 / sides
angle = 360 / sides

Turtle.Speed = 9

For j = 1 To 20
  For i = 1 To sides
    Turtle.Move(length)
    Turtle.Turn(angle)
  EndFor
```

```
  Turtle.Turn(18)
EndFor
```

以上程序中有两个嵌套的 **For..EndFor** 循环，内层的循环 (*i = 1 to sides*) 和多边形程序相似，负责输出圆形。外层的循环 (*j = 1 to 20*) 负责在每次绘制圆形后移动龟标。在程序中，我们指定使用龟标绘制 20 个圆形。程序的运行结果是一个如下所示的有趣图案。

在以上程序中，通过将 Speed （速度）设置为 9，可以让“海龟”走得快一点儿。你也可以通过修改这个属性设置让“海龟”按照你喜欢的速度移动。

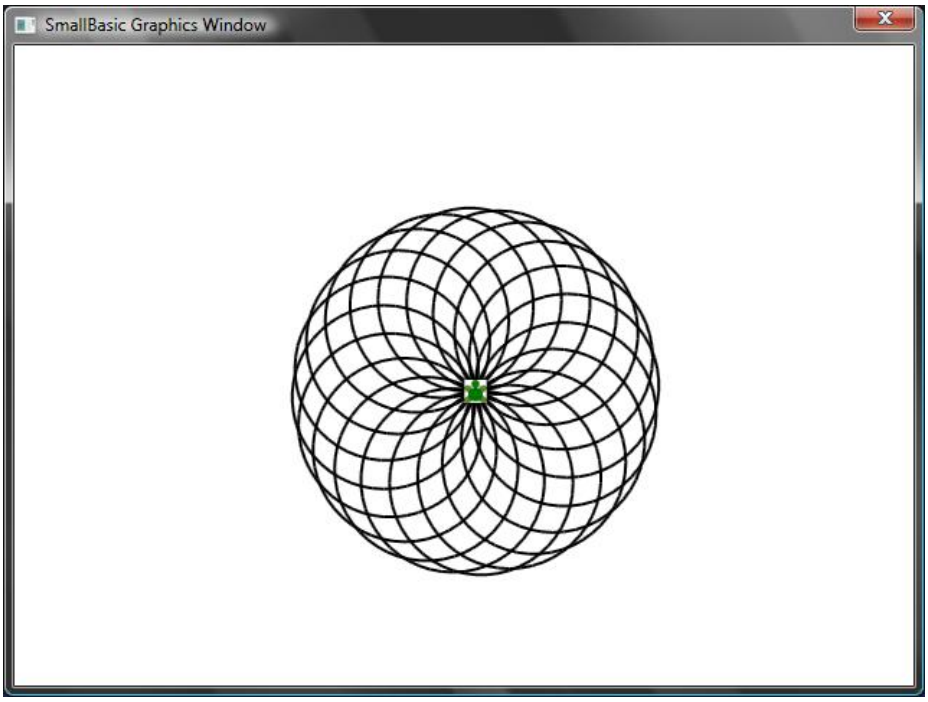

图 42 – 在圆形中移动

自由移动

可以使用 **PenUp**（停笔）运算让龟标停止移动。这样，就可以将龟标移动到屏幕的任意位置，而不留下痕迹。如果想重新使用龟标绘制图案，可以调用 **PenDown**（落笔）。通过使用这两个命令，可以绘制出像虚线这样的有趣效果。以下程序就使用这个方法绘制出一个以虚线组成的多边形。

```
sides = 6

length = 400 / sides
```

```
angle = 360 / sides

For i = 1 To sides
  For j = 1 To 6
    Turtle.Move(length / 12)
    Turtle.PenUp()
    Turtle.Move(length / 12)
    Turtle.PenDown()
  EndFor
  Turtle.Turn(angle)
EndFor
```

和上面的程序一样，这个程序也有两个嵌套的循环，内部的循环绘制虚线，外部的循环根据指定数值绘制多边形的边。在这个的示例中，我们指定 **sides**（边数）变量为 6，得到一个由虚线组成的六边形，如下所示。

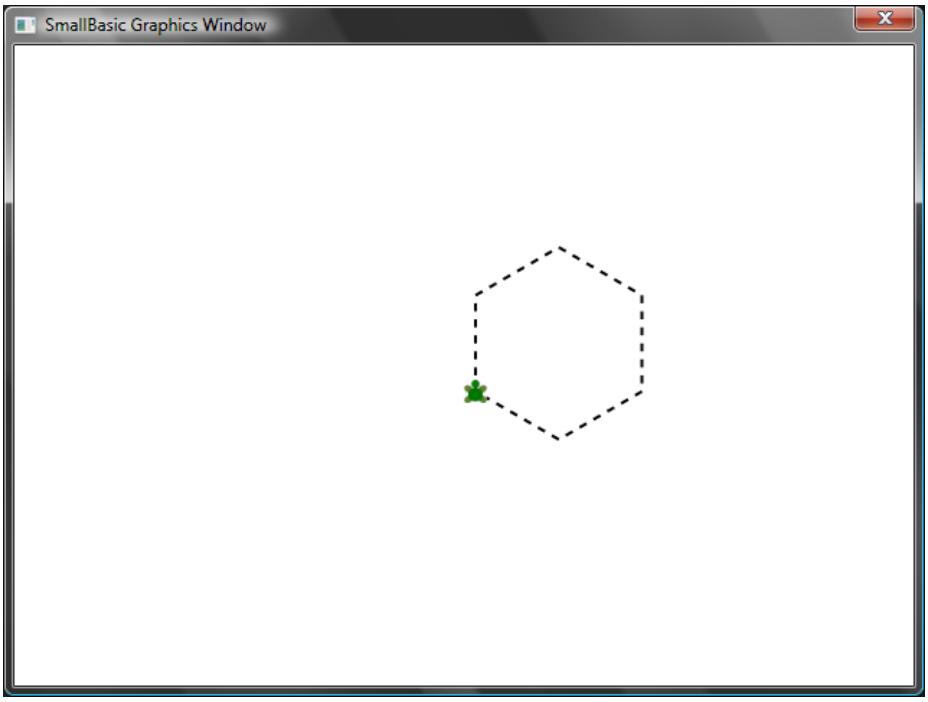

图 43 – 使用 PenUp 和 PenDown

第九章

子例程（Subroutines）

在编程时我们经常遇到需要重复执行相同一组步骤的情况。这种情况下，多次重写相同的语句是不合理的。这时我们需要借助子例程。

一个子例程是一个较大的程序中实现特定功能的一小段代码，子例程可以在程序的任意位置被调用。子例程由一个以 **Sub** 关键字开头的名字标识并以 **EndSub** 关键字结束。例如，下面的代码段表示了一个叫做 *PrintTime*（打印时间）的子例程，它可以把当前时间显示到 TextWindow（文本窗口）中。

```
Sub PrintTime
  TextWindow.WriteLine(Clock.Time)
EndSub
```

下面的程序包含了该子例程并从多个地方调用它。

```
PrintTime()
TextWindow.Write("Enter your name: ")
name = TextWindow.Read()
TextWindow.Write(name + ", the time now is: ")
PrintTime()

Sub PrintTime
  TextWindow.WriteLine(Clock.Time)
EndSub
```

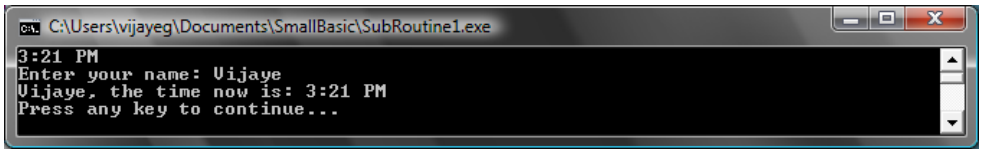

图 44 – 调用一个简单子例程

您可以通过调用 *SubroutineName()* 来执行子例程。通常地，符号“()” 对于告诉计算机您打算执行一个子例程是必需的。

使用子例程的好处

如我们在上边看到的，子例程帮助我们减少了大量编程时输入的代码。一旦子例程 *PrintTime* 写好了，您就可以从程序的任意地方调用它来显示当前时间。

此外，子例程帮助我们将复杂问题简化为较简单的片段。比方说您有一个复合公式要求解，您可以编写几个子例程先求解该复合公式的较小的部分。然后您可以把结果合并起来求解原始的复合公式。

请记住，您只能在同一个程序中调用 SmallBasic 子例程。您不能从其他程序中调用该子例程。

子例程还能帮助提高程序的可读性。换句话说，如果您的程序中公用部分都是名称清楚的子例程，您的程序就会变得容易读和理解。如果您想理解其他人写的程序或者您希望您的程序能被其他人理解，子例程尤为重要。有时，子例程对于您理解您自己的程序也会有帮助，比方您一周前写的程序。

使用变量（variables）

您可以在子例程内访问和使用程序中的任何变量。作为例子，下面的程序接受两个数并打印出较大的那个数。请注意变量 *max*（最大）同时在子例程内部和外部被使用。

```
TextWindow.Write("Enter first number: ")
num1 = TextWindow.ReadNumber()
TextWindow.Write("Enter second number: ")
num2 = TextWindow.ReadNumber()

FindMax()
TextWindow.WriteLine("Maximum number is: " + max)

Sub FindMax
  If (num1 > num2) Then
    max = num1
  Else
```

```
    max = num2
  EndIf
EndSub
```

程序输出如下所示。

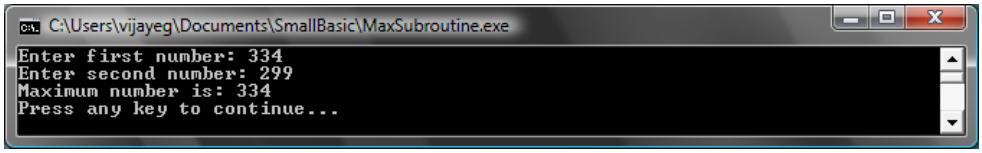

图 45 – 使用子例程得到较大数

让我们看另一个例子以说明子例程的用法。这次我们使用一个图形程序计算以变量 *x* 和 *y* 存储的点。然后调用一个子例程 DrawCircleUsingCenter 来绘制以 *x* 和 *y* 为圆心的圆。

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightBlue"
GraphicsWindow.Width = 480
For i = 0 To 6.4 Step 0.17
  x = Math.Sin(i) * 100 + 200
  y = Math.Cos(i) * 100 + 200

  DrawCircleUsingCenter()
EndFor

Sub DrawCircleUsingCenter
  startX = x - 40
  startY = y - 40

  GraphicsWindow.DrawEllipse(startX, startY, 120, 120)
EndSub
```

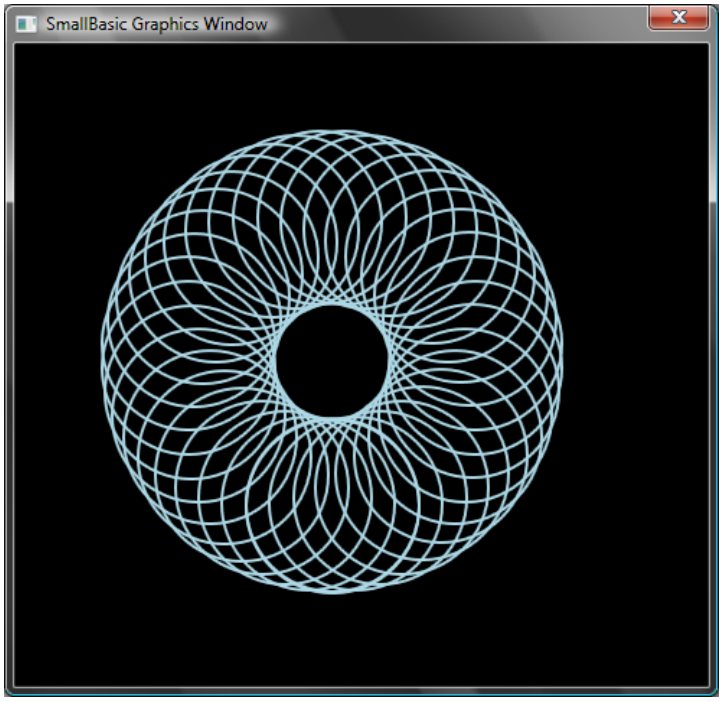

图 46 – 图形子例程示例

在循环（Loops）内调用子例程

有时子例程可在循环内被调用，那段时间内子例程执行同一组语句但处理一个或多个变量的不同值。例如，您有个判断一个数是否为质数的子例程 *PrimeCheck*（质数检测）。您可以写一个程序让用户输入一个数然后您的程序通过子例程判断这个数是否为质数。参见下面的程序。

```
TextWindow.Write("Enter a number: ")
i = TextWindow.ReadNumber()
isPrime = "True"
PrimeCheck()
If (isPrime = "True") Then
  TextWindow.WriteLine(i + " is a prime number")
Else
  TextWindow.WriteLine(i + " is not a prime number")
EndIf

Sub PrimeCheck
  For j = 2 To Math.SquareRoot(i)
    If (Math.Remainder(i, j) = 0) Then
      isPrime = "False"
      Goto EndLoop
    EndIf
```

```
  Endfor
EndLoop:
EndSub
```

PrimeCheck 子例程得到 *i* 的值并试图用较小的数来除它。如果有一个数能整除 *i*，说明 *i* 不是质数。这时子例程将 *isPrime* 的值设为“False”（假）并退出。如果这个数不能被更小的数整除那么它的 *isPrime* 值仍保持为“True”（真）。

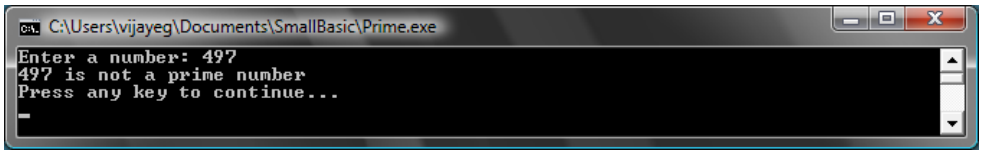

图 47 – 质数检测

现在您有一个可以作质数检测的子例程了，您也许想用它列出比如 100 之内的所有质数。我们可以很容易地修改程序从一个循环中调用 *PrimeCheck* 来实现。每次循环运行时都会给子例程一个不同的数。让我们通过下面的例子来解释。

```
For i = 3 To 100
  isPrime = "True"
  PrimeCheck()
  If (isPrime = "True") Then
    TextWindow.WriteLine(i)
  EndIf
EndFor

Sub PrimeCheck
  For j = 2 To Math.SquareRoot(i)
    If (Math.Remainder(i, j) = 0) Then
      isPrime = "False"
      Goto EndLoop
    EndIf
  Endfor
EndLoop:
EndSub
```

上面的程序中，*i* 的值在每次循环时都被更新。在循环内部，有一个到子例程 *PrimeCheck* 的调用。子例程 *PrimeCheck* 取得 *i* 的值并计算 *i* 是否为质数。结果保存在变量 *isPrime* 中并可在子例程外部被循环所访问。如果 *i* 是质数则它的值会被打印出来。由于循环从 3 开始直到 100，我们就得到了 3 到 100 间的所有质数的列表。下面是程序运行结果。

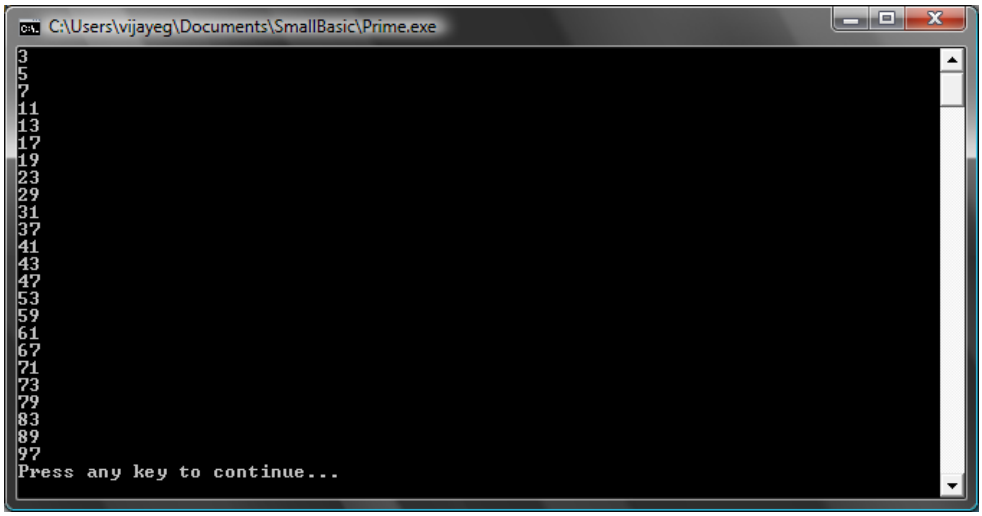
C:\Users\vijayeg\Documents\SmallBasic\Prime.exe
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
Press any key to continue...

图 48 – 质数

第十章

数组

现在您应当很清楚地了解如何使用变量了，并且迄今为止您仍对编程感到有趣，对吗？

让我们花一点儿时间来重温一下我们使用变量编写的第一个程序。

```
TextWindow.Write("Enter your Name: ")
name = TextWindow.Read()
TextWindow.WriteLine("Hello " + name)
```

在这个程序中，我们把接收并存储用户名的变量起名为 **name** 。然后我对这个用户说“你好”。现在，让我们假定有多于一位的用户，比方说，有五位用户。我们该如何存储所有的名字呢？有一个办法是这样的：

```
TextWindow.Write("User1, enter name: ")
name1 = TextWindow.Read()
TextWindow.Write("User2, enter name: ")
name2 = TextWindow.Read()
TextWindow.Write("User3, enter name: ")
name3 = TextWindow.Read()
TextWindow.Write("User4, enter name: ")
name4 = TextWindow.Read()
TextWindow.Write("User5, enter name: ")
name5 = TextWindow.Read()

TextWindow.Write("Hello ")
```

```
TextWindow.Write(name1 + ", ")
TextWindow.Write(name2 + ", ")
TextWindow.Write(name3 + ", ")
TextWindow.Write(name4 + ", ")
TextWindow.WriteLine(name5)
```

运行上面的程序时您将看到以下结果：

图 49 – 未使用数组

很明显我们肯定会有一种更好的办法来写这样一个简单的程序，对吗？尤其是计算机非常擅长做重复性的任务，我们为什么很麻烦地要为每一位新用户重新写一段相同的代码呢？这里的技巧就是把多个用户的名字保存在同一个变量中。如果我们可以做到这一点，我们就可以使用前边章节中学过的 **For** 循环了。这里我们就要引入数组了。

什么是数组？

一个数组是一种特殊类型的变量，它可以同时保存多于一个的值。也就是说，如果要保存五个用户的名字，我们不必创建 **name1**，**name2**，**name3**，**name4** 和 **name5** 的五个变量，我们可以仅仅使用 **name** 来存储全部五个用户名字。这种同时存储多个值的方法叫做“索引”（index）。例如，**name[1]**，**name[2]**，**name[3]**，**name[4]** 和 **name[5]** 可以分别保存一个值。数字 1，2，3，4 和 5 被称为数组的*索引*（*indices*）。

尽管 **name[1]**，**name[2]**，**name[3]**，**name[4]** 和 **name[5]** 看上去像不同的变量，但它们实际上是一个变量。您也许会问它的好处是什么。使用数组存储值的最大好处就是您可以使用另一个变量来指定索引，这样您可以很容易地在循环中访问数组。

现在，让我们来看看如何使用我们新学的数组知识来重写刚才的程序。

```
For i = 1 To 5
  TextWindow.Write("User" + i + ", enter name: ")
  name[i] = TextWindow.Read()
EndFor

TextWindow.Write("Hello ")
```

```
For i = 1 To 5
  TextWindow.Write(name[i] + ", ")
EndFor
TextWindow.WriteLine("")
```

程序的易读性大大提高了，不是吗？请注意两行加粗的行。第一行把值存入数组而第二行把值由数组读出来。**name[1]** 中存储的值不会被 **name[2]** 中存储的值影响。因此很大程度上您可以把 **name[1]** 和 **name[2]** 当做具有相同身份的两个不同变量。

图 50 – 使用数组

上面的程序与没有使用数组的程序运行结果几乎相同，除了在 *Mantis* 后的逗号。我们可以通过重写输出循环来修正它：

```
TextWindow.Write("Hello ")
For i = 1 To 5
  TextWindow.Write(name[i])
  If i < 5 Then
    TextWindow.Write(", ")
  EndIf
EndFor
TextWindow.WriteLine("")
```

数组的索引

在前边的程序中您看到了我们使用数字作为索引从数组中存取数值。实际上索引不仅仅限于数字，我们也可以使用文字索引。例如，下面的程序中，我们询问并保存一个用户的信息然后将信息输出。

```
TextWindow.Write("Enter name: ")
user["name"] = TextWindow.Read()
TextWindow.Write("Enter age: ")
user["age"] = TextWindow.Read()
TextWindow.Write("Enter city: ")
user["city"] = TextWindow.Read()
TextWindow.Write("Enter zip: ")
user["zip"] = TextWindow.Read()

TextWindow.Write("What info do you want? ")
index = TextWindow.Read()
TextWindow.WriteLine(index + " = " + user[index])
```

图 51 – 使用非数字索引

多于一个维度

比方说您想把您所有朋友的名字和电话号码保存起来并能查询他们的电话号码，就像电话簿一样。您应该如何编写这样一个程序呢？

这个例子涉及两套索引（也被认为是数组的维度，即二维数组）。假设我们用朋友的“小名”来辨识每一位朋友。这就是我们的数组的第一套索引。一旦我们得到了朋友的变量，第二套索引，**name** 和 **phone number** 就可以帮助我们得到朋友的全名和电话号码。

我们可以像这样来存储数据：

```
friends["Rob"]["Name"] =
"Robert"
friends["Rob"]["Phone"] = "555-
6789"

friends["VJ"]["Name"] =
"Vijaye"
friends["VJ"]["Phone"] = "555-4567"
```

数组索引是不区分大小写的。如同正常的变量，数组索引的匹配会忽略大小写。

```
friends["Ash"]["Name"] = "Ashley"
friends["Ash"]["Phone"] = "555-2345"
```

由于我们的数组 **friends** 有两套索引，该数组被称为二维数组。

这时我们就可以使用朋友的小名作为输入值然后输出朋友的全名及电话号码信息了。下面是完整的程序代码：

```
friends["Rob"]["Name"] = "Robert"
friends["Rob"]["Phone"] = "555-6789"

friends["VJ"]["Name"] = "Vijaye"
friends["VJ"]["Phone"] = "555-4567"

friends["Ash"]["Name"] = "Ashley"
friends["Ash"]["Phone"] = "555-2345"

TextWindow.Write("Enter the nickname: ")
nickname = TextWindow.Read()

TextWindow.WriteLine("Name: " + friends[nickname]["Name"])
TextWindow.WriteLine("Phone: " + friends[nickname]["Phone"])
```

图 52 – 一个简单的电话簿

使用数组表示网格

多维数组一个通常的用法是表示网格和表格。网格有行和列，他们恰好可以放入一个二维数组。下面是一个简单的将箱子放入网格的示例：

```
rows = 8
columns = 8
size = 40
```

```
For r = 1 To rows
  For c = 1 To columns
    GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
    boxes[r][c] = Shapes.AddRectangle(size, size)
    Shapes.Move(boxes[r][c], c * size, r * size)
  EndFor
EndFor
```

这个程序把小矩形加入并调整位置以组成一个 8x8 的网格。除了放置这些小矩形，该程序也把它们存入一个数组。这样做使得我们可以容易地追踪这些小矩形并在需要的时候再次使用它们。

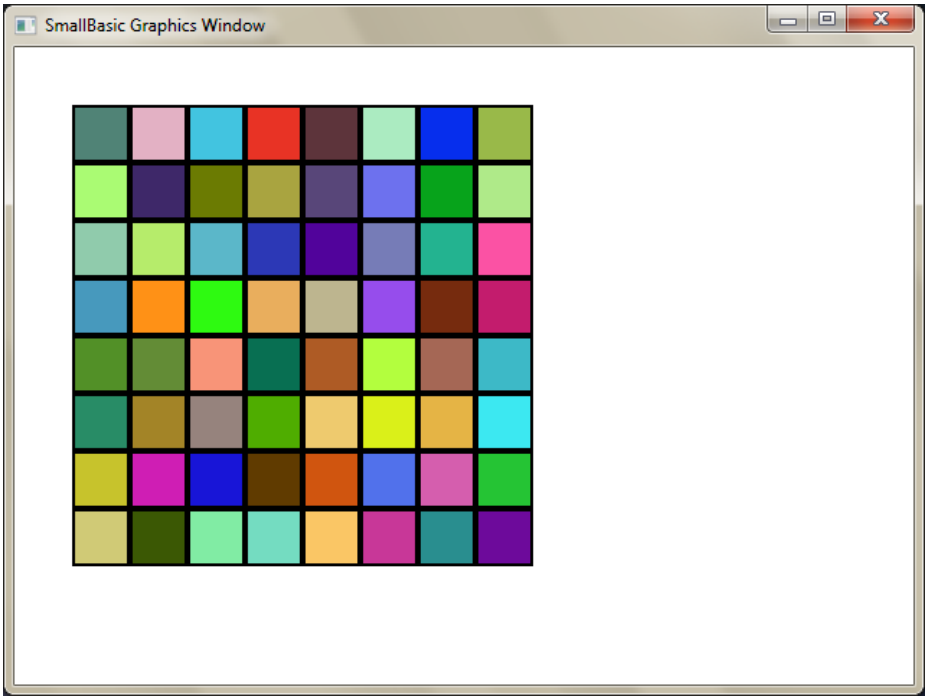

图 53 – 在网格中放置小矩形

例如，把下面的代码附加到前面的程序最后就可以把小矩形以动画的形式移动到窗口的左上角。

```
For r = 1 To rows
  For c = 1 To columns
    Shapes.Animate(boxes[r][c], 0, 0, 1000)
    Program.Delay(300)
  EndFor
EndFor
```

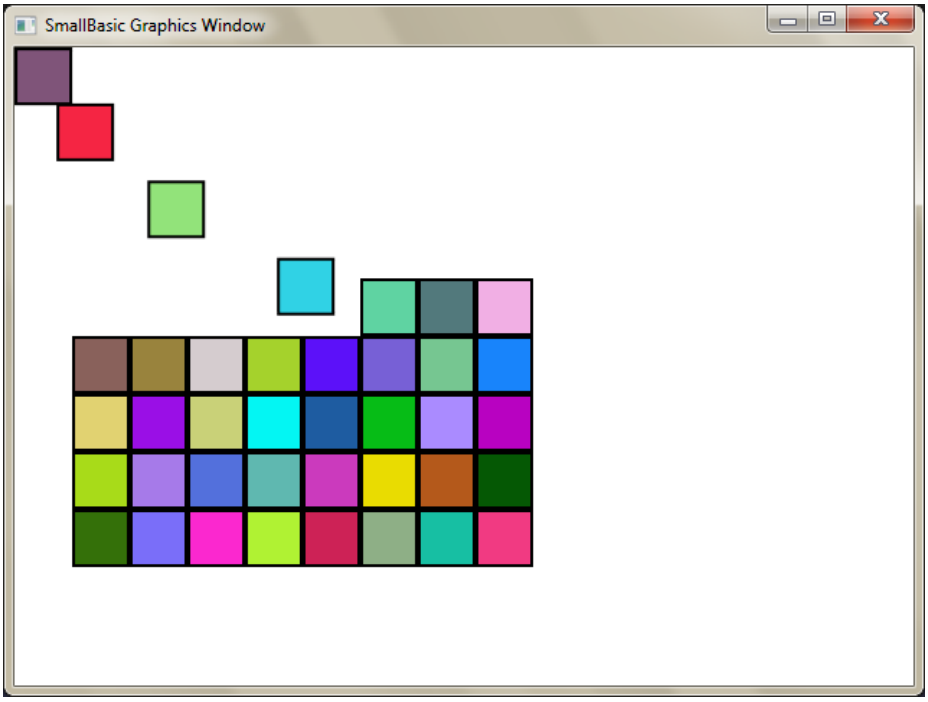

图 54 – 在网格中追踪小矩形

第十一章

事件（Events）和交互（Interactivity）

在最前面的两章，我们介绍了具有属性和操作的对象。除了属性和操作，一些对象还带有事件（Event）。事件就像被触发的信号，例如，为了应对类似移动或点击鼠标的用户操作。某种程度上说，事件同操作是相对的。对于操作，您作为一个程序员调用操作让计算机做一些事情；然后对于事件，计算机通知您一些有意思的事情发生了。

事件有什么用？

事件是介绍程序交互性的核心。如果您想让用户同您的程序交互，您必须使用事件。比方说，您正在编写一个 Tic-Tac-Toe 游戏（也有叫井字棋的，是一种类似于五子棋的连线游戏）。您希望能让用户选择他的角色，对吗？这里就涉及到事件 – 您在您的程序中使用事件来收到用户输入。如果这样的解释难以明白，不要担心，我们来看一个非常简单的示例来帮助您理解什么是事件以及如何使用事件。

下面是一个非常简单的程序，只有一行语句和一个子例程。子例程使用 *GraphicsWindow*（图形窗口）对象的 *ShowMessage*（显示消息）操作显示一个对话框给用户。

```
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
  GraphicsWindow.ShowMessage("You Clicked.", "Hello")
EndSub
```

上面程序中一个有意思的部分是我们将一个子例程指派到 GraphicsWindow 对象的 MouseDown（按下鼠标）事件的那一行。您可能注意到了 MouseDown 看起来非常像一个属性 – 除了指定一些值（我们将子例程 *OnMouseDown* 指派给它）。这就是事件的特殊性 – 当事件发生时，子例程会自动被调用。在这个例子中，每次用户在 GraphicsWindow 上点击鼠标时子例程 *OnMouseDown* 都会被调用。让我们继续，运行并使用这个程序。任何时间当您在 GraphicsWindow 上点击鼠标时您都将看到如下面所示的对话框。

图 55 – 对事件的反应

这种事件处理是非常强大的，它支持非常有创造性和有意思的程序。使用这种方式编写的程序通常被成为事件驱动（event-driven）的程序。

您可以修改子例程 *OnMouseDown* 实现比弹出一个对话框更多的事情。例如像下面的程序，您可以在用户点击鼠标的地方绘制大蓝点。

```
GraphicsWindow.BrushColor = "Blue"
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
  x = GraphicsWindow.MouseX - 10
  y = GraphicsWindow.MouseY - 10
  GraphicsWindow.FillEllipse(x, y, 20, 20)
EndSub
```

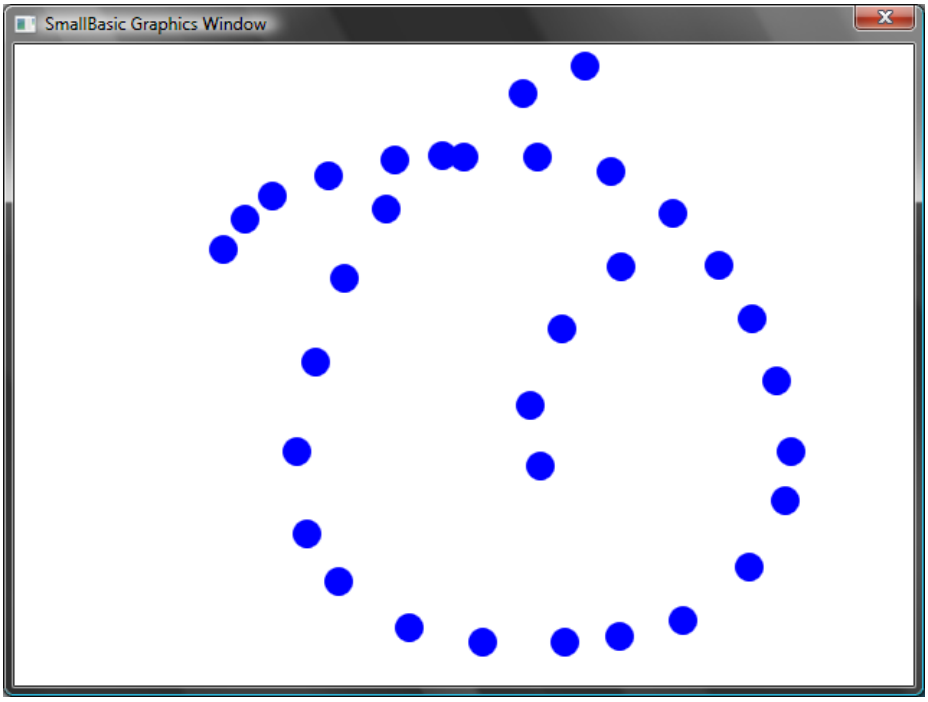

图 56 – 处理鼠标点击事件

请注意在上面程序中，我们使用 *MouseX* 和 *MouseY* 来取得鼠标的坐标位置。然后我们用鼠标坐标点作为圆心绘制一个圆。

处理多个事件

您想处理多少事件是没有限制的。您甚至可以使用一个子例程来处理多个事件。但是，您每次只能处理一个事件。如果您试图指派两个子例程给同一个事件，只有后面的子例程会起作用。

为了说明这一点，让我们使用前边的示例并添加一个处理键盘按键的子例程。同时，我们让新的子例程改变笔刷的颜色，这样当您点击鼠标时您将看到不同颜色的点。

```
GraphicsWindow.BrushColor = "Blue"
GraphicsWindow.MouseDown = OnMouseDown
GraphicsWindow.KeyDown = OnKeyDown

Sub OnKeyDown
  GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
EndSub

Sub OnMouseDown
  x = GraphicsWindow.MouseX - 10
  y = GraphicsWindow.MouseY - 10
  GraphicsWindow.FillEllipse(x, y, 20, 20)
```

```
EndSub
```

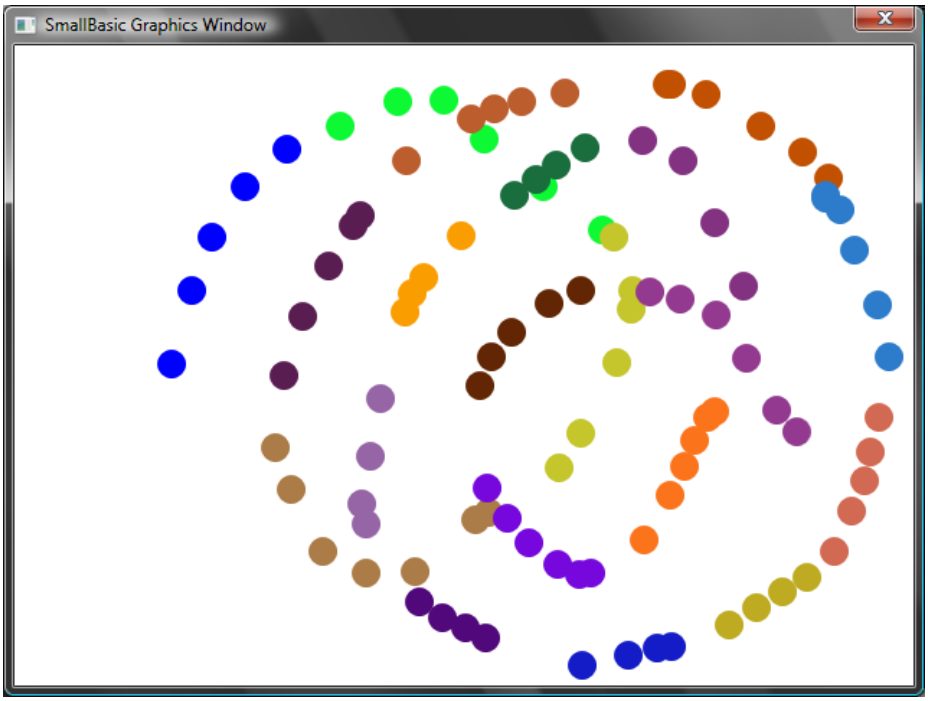

图 57 – 处理多个事件

原来当您运行程序并在窗口中点击时，您将看到蓝色的点。现在，如果您按键盘上的任意键然后再点击鼠标，您将看到不同颜色的点。当您按下键盘上的键时子例程 *OnKeyDown*（按下键盘按键）被执行，它将笔刷的颜色改为随机颜色。然后您再点击鼠标时，您就会看到一个使用新设置的颜色（随机产生的颜色）的圆点。

一个绘图程序

由于事件和子例程的帮助，我们现在可以编写让用户在窗口中绘图的程序了。只要我们把问题分解为较小的模块，编写这样一个程序就变得简单得令人吃惊了。第一步，我们写一个允许用户在图形窗口任意移动鼠标的程序，鼠标走过的地方会留下一道轨迹。

```
GraphicsWindow.MouseMove = OnMouseMove

Sub OnMouseMove
  x = GraphicsWindow.MouseX
  y = GraphicsWindow.MouseY
  GraphicsWindow.DrawLine(prevX, prevY, x, y)
  prevX = x
  prevY = y
```

```
EndSub
```

不过，当您运行这个程序时，第一道轨迹总是从窗口的左上角即(0, 0) 开始。我们可以通过处理 *MouseDown* 事件并捕捉当事件发生时的 *prevX* 和 *prevY* 值来修复这个问题。

同时，我们只需要把用户按下鼠标时的轨迹记录下来。在其他时间我们不必绘制轨迹线。为了实现这样的行为，我们将使用 **Mouse**（鼠标）对象的 *IsLeftButtonDown*（鼠标左键按下）属性。该属性告诉我们鼠标左键是否被按下。如果这个值为 true（真），我们就画轨迹线，否则就不画。

```
GraphicsWindow.MouseMove = OnMouseMove
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
  prevX = GraphicsWindow.MouseX
  prevY = GraphicsWindow.MouseY
EndSub

Sub OnMouseMove
  x = GraphicsWindow.MouseX
  y = GraphicsWindow.MouseY
  If (Mouse.IsLeftButtonDown) Then
    GraphicsWindow.DrawLine(prevX, prevY, x, y)
  EndIf
  prevX = x
  prevY = y
EndSub
```

（待续）

附录 A

趣味实例

龟标分形图

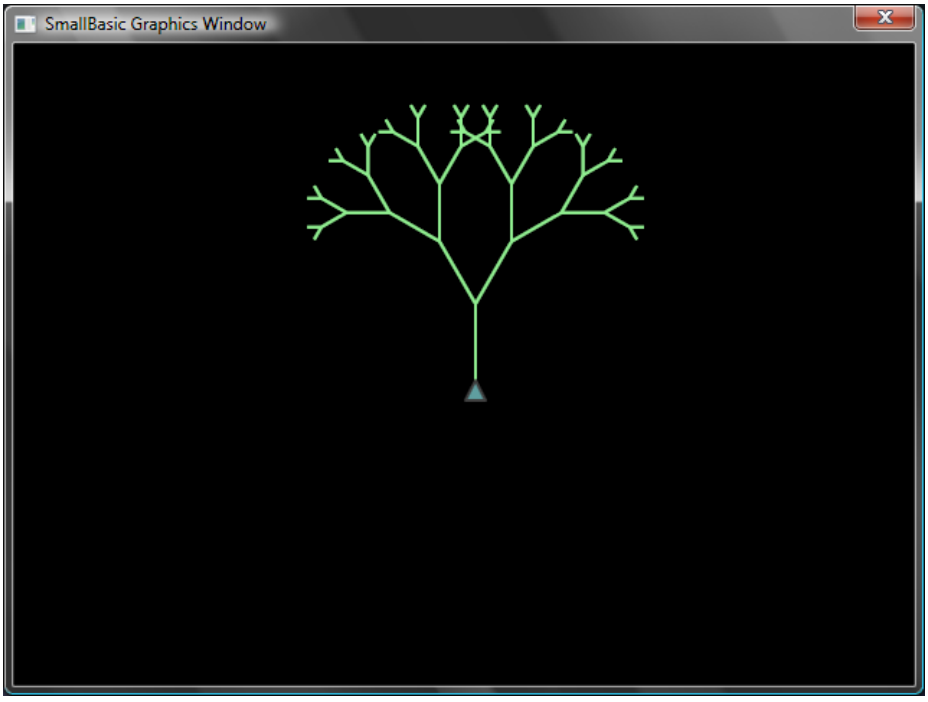

图 58 – 龟标树形图

```
angle = 30
delta = 10
distance = 60
```

```
Turtle.Speed = 9
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightGreen"
DrawTree()

Sub DrawTree
  If (distance > 0) Then
    Turtle.Move(distance)
    Turtle.Turn(angle)

    Stack.PushValue("distance", distance)
    distance = distance - delta
    DrawTree()
    Turtle.Turn(-angle * 2)
    DrawTree()
    Turtle.Turn(angle)
    distance = Stack.PopValue("distance")

    Turtle.Move(-distance)
  EndIf
EndSub
```

来自于 Flickr 的照片

图 59 – 抓取 Flickr 的照片

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
    pic = Flickr.GetRandomPicture("mountains, river")
    GraphicsWindow.DrawResizedImage(pic, 0, 0, 640, 480)
EndSub
```

动态桌面墙纸

```
For i = 1 To 10
  pic = Flickr.GetRandomPicture("mountains")
  Desktop.SetWallPaper(pic)
  Program.Delay(10000)
EndFor
```

船桨弹球游戏

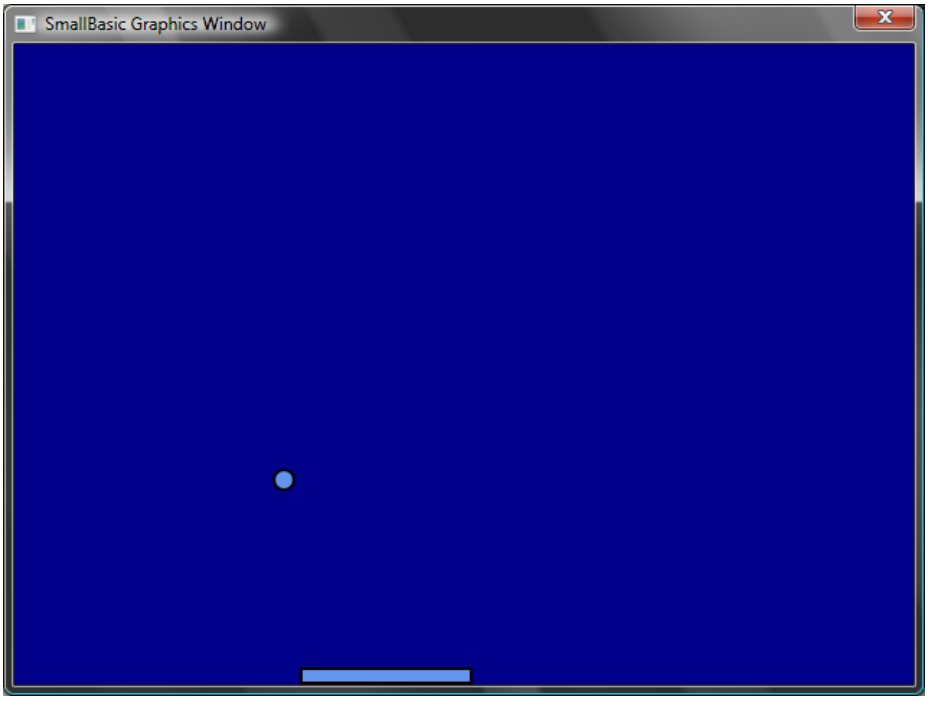

图 60 – 船桨弹球游戏

```
GraphicsWindow.BackgroundColor = "DarkBlue"
paddle = Shapes.AddRectangle(120, 12)
ball = Shapes.AddEllipse(16, 16)
```

```
GraphicsWindow.MouseMove = OnMouseMove

x = 0
y = 0
deltaX = 1
deltaY = 1

RunLoop:
  x = x + deltaX
  y = y + deltaY

  gw = GraphicsWindow.Width
  gh = GraphicsWindow.Height
  If (x >= gw - 16 or x <= 0) Then
    deltaX = -deltaX
  EndIf
  If (y <= 0) Then
    deltaY = -deltaY
  EndIf

  padX = Shapes.GetLeft(paddle)
  If (y = gh - 28 and x >= padX and x <= padX + 120) Then
    deltaY = -deltaY
  EndIf

  Shapes.Move(ball, x, y)
  Program.Delay(5)

  If (y < gh) Then
    Goto RunLoop
  EndIf

GraphicsWindow.ShowMessage("You Lose", "Paddle")

Sub OnMouseMove
  paddleX = GraphicsWindow.MouseX
  Shapes.Move(paddle, paddleX - 60, GraphicsWindow.Height - 12)
EndSub
```

附录 B

颜色

TODO: 描述颜色 + 十六进制数

这里是一个 Small Basic 支持的颜色列表, 按它们的基础色调分类的.

Red Colors （红色）

IndianRed	#CD5C5C
LightCoral	#F08080
Salmon	#FA8072
DarkSalmon	#E9967A
LightSalmon	#FFA07A
Crimson	#DC143C
Red	#FF0000
FireBrick	#B22222
DarkRed	#8B0000

Pink Colors （粉色）

Pink	#FFC0CB
LightPink	#FFB6C1
HotPink	#FF69B4
DeepPink	#FF1493
MediumVioletRed	#C71585
PaleVioletRed	#DB7093

Orange Colors （橘红色）

LightSalmon	#FFA07A
Coral	#FF7F50
Tomato	#FF6347
OrangeRed	#FF4500
DarkOrange	#FF8C00
Orange	#FFA500

Yellow Colors （黄色）

Gold	#FFD700
Yellow	#FFFF00
LightYellow	#FFFFE0
LemonChiffon	#FFFACD
LightGoldenrodYellow	#FAFAD2

PapayaWhip	#FFEFD5
Moccasin	#FFE4B5
PeachPuff	#FFDAB9
PaleGoldenrod	#EEE8AA
Khaki	#F0E68C
DarkKhaki	#BDB76B

Purple Colors （紫色）

Lavender	#E6E6FA
Thistle	#D8BFD8
Plum	#DDA0DD
Violet	#EE82EE
Orchid	#DA70D6
Fuchsia	#FF00FF
Magenta	#FF00FF
MediumOrchid	#BA55D3
MediumPurple	#9370DB
BlueViolet	#8A2BE2
DarkViolet	#9400D3
DarkOrchid	#9932CC
DarkMagenta	#8B008B
Purple	#800080
Indigo	#4B0082
SlateBlue	#6A5ACD
DarkSlateBlue	#483D8B
MediumSlateBlue	#7B68EE

Green Colors （绿色）

GreenYellow	#ADFF2F
Chartreuse	#7FFF00
LawnGreen	#7CFC00
Lime	#00FF00
LimeGreen	#32CD32
PaleGreen	#98FB98
LightGreen	#90EE90
MediumSpringGreen	#00FA9A
SpringGreen	#00FF7F
MediumSeaGreen	#3CB371
SeaGreen	#2E8B57
ForestGreen	#228B22
Green	#008000
DarkGreen	#006400
YellowGreen	#9ACD32
OliveDrab	#6B8E23
Olive	#808000
DarkOliveGreen	#556B2F
MediumAquamarine	#66CDAA
DarkSeaGreen	#8FBC8F
LightSeaGreen	#20B2AA
DarkCyan	#008B8B
Teal	#008080

Blue Colors （蓝色）

Aqua	#00FFFF
Cyan	#00FFFF
LightCyan	#E0FFFF
PaleTurquoise	#AFEEEE
Aquamarine	#7FFFD4
Turquoise	#40E0D0
MediumTurquoise	#48D1CC
DarkTurquoise	#00CED1
CadetBlue	#5F9EA0
SteelBlue	#4682B4

LightSteelBlue	#B0C4DE
PowderBlue	#B0E0E6
LightBlue	#ADD8E6
SkyBlue	#87CEEB
LightSkyBlue	#87CEFA
DeepSkyBlue	#00BFFF
DodgerBlue	#1E90FF
CornflowerBlue	#6495ED
MediumSlateBlue	#7B68EE
RoyalBlue	#4169E1
Blue	#0000FF
MediumBlue	#0000CD
DarkBlue	#00008B
Navy	#000080
MidnightBlue	#191970

Brown Colors （棕色）

Cornsilk	#FFF8DC
BlanchedAlmond	#FFEBCD
Bisque	#FFE4C4
NavajoWhite	#FFDEAD
Wheat	#F5DEB3
BurlyWood	#DEB887
Tan	#D2B48C
RosyBrown	#BC8F8F
SandyBrown	#F4A460
Goldenrod	#DAA520
DarkGoldenrod	#B8860B
Peru	#CD853F
Chocolate	#D2691E
SaddleBrown	#8B4513
Sienna	#A0522D
Brown	#A52A2A
Maroon	#800000

White Colors （白色）

White	#FFFFFF
Snow	#FFFAFA
Honeydew	#F0FFF0
MintCream	#F5FFFA
Azure	#F0FFFF
AliceBlue	#F0F8FF
GhostWhite	#F8F8FF
WhiteSmoke	#F5F5F5
Seashell	#FFF5EE
Beige	#F5F5DC
OldLace	#FDF5E6
FloralWhite	#FFFAF0
Ivory	#FFFFF0
AntiqueWhite	#FAEBD7
Linen	#FAF0E6
LavenderBlush	#FFF0F5
MistyRose	#FFE4E1

Gray Colors （灰色）

Gainsboro	#DCDCDC
LightGray	#D3D3D3
Silver	#C0C0C0
DarkGray	#A9A9A9
Gray	#808080
DimGray	#696969
LightSlateGray	#778899
SlateGray	#708090

DarkSlateGray	#2F4F4F
Black	#000000