Microsoft Small Basic

프로그래밍 소개

제 1 장

소개

Small Basic 과 프로그래밍

컴퓨터 프로그래밍은 프로그래밍 언어를 사용해서 컴퓨터 소프트웨어를 만드는 과정이라고 정의합니다. 마치 우리가 영어나 스페인어, 또는 불어를 말하고 이해하는 것과 같이 컴퓨터는 어떤 언어로 쓰여진 프로그램을 이해할 수 있습니다. 이런 언어를 프로그래밍 언어라고 부릅니다. 초기에는 단지 몇 개의 프로그래밍 언어들이 있었고 그 언어들은 배우고 이해하기가 아주 쉬웠습니다. 그러나, 컴퓨터와 소프트웨어가 점점 복잡해 지면서 프로그래밍 언어는 빠르게 진화 하였고 더욱 복잡한 개념들을 포함하게 되었습니다. 그 결과 대부분의 현대 프로그래밍 언어와 그 개념은 초보자가 이해하기에는 쉽지 않게 되었고 사람들은 컴퓨터 프로그래밍을 배우고 시도하기를 꺼려 하기 시작하였습니다.

Small Basic 은 초보자들이 프로그래밍을 아주 쉽고 간편하게 그리고 재미있게 할 수 있도록 디자인한 프로그래밍 언어입니다. Small Basic 은 놀라운 프로그래밍의 세계로 쉽게 다가 갈 수 있는 징검다리가 되고자 합니다.

Small Basic 환경

Small Basic 환경에 대한 간단한 소개부터 시작해 보겠습니다. 처음 SmallBasic.exe 를 실행시키면 다음과 같은 창을 보게 될 것입니다.

그림 1 - Small Basic 환경

이것이 우리가 Small Basic 프로그램을 작성하고 실행하는 Small Basic 환경입니다. 이 환경은 몇 개의 고유한 구성요소로 나누어져 있고 그 구성요소는 숫자로 표시되어 있습니다.

[1]번으로 표시된 **편집기**는 Small Basic 프로그램을 작성하는 곳입니다. 샘플 프로그램이나 이전에 저장한 프로그램을 열면 그 프로그램들은 이 편집기에 나타나게 됩니다. 그 프로그램을 수정하고 나중에 사용하기 위해 저장할 수 있습니다.

또한 한 개 이상의 프로그램을 동시에 열어 작업할 수도 있습니다. 현재 작업하고 있는 각 프로그램은 각기 다른 편집기에 표시됩니다. 현재 작업하고 있는 프로그램을 포함하고 있는 편집기를 *활성화된 편집기*라고 부릅니다.

[2]번으로 표시된 **도구 모음**은 활성화된 편집기나 환경으로 명령을 내리기 위해 사용합니다. 앞으로 도구 모음 안에 있는 다양한 명령들을 배우게 될 것입니다.

[3]번으로 표시된 **화면**은 모든 편집기 창이 들어가는 장소입니다.

첫 번째 프로그램

이제 Small Basic 환경에 익숙해 졌으니 Small Basic 을 사용해서 프로그램을 시작해 보겠습니다. 위에서 언급했듯이 편집기는 프로그램을 작성하는 곳입니다. 편집기 안에서 다음과 같이 입력해 보겠습니다.

```
TextWindow.WriteLine("Hello World")
```

이것이 우리의 첫 번째 Small Basic 프로그램입니다. 만약 위와 같이 정확히 입력했다면 아래 그림과 같은 모습을 보게 될 것입니다.

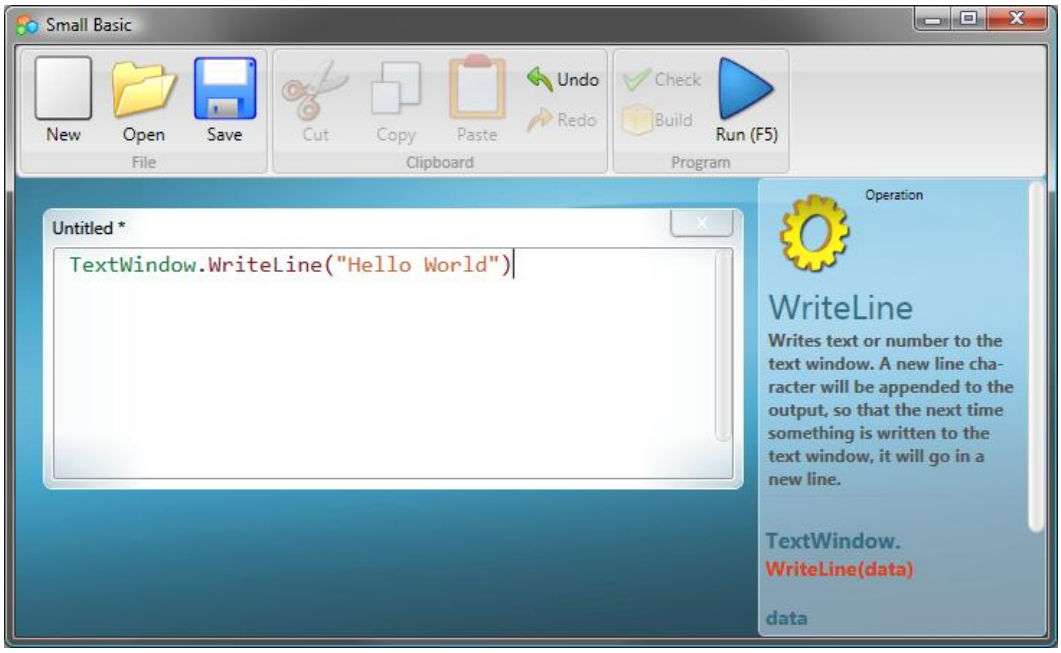

그림 2 – 첫 번째 프로그램

이제 우리는 우리의 첫 번째 프로그램을 입력했고 이제 그 프로그램을 실행시켜 보겠습니다. 우리는 도구 모음에 있는 실행 단추를 누르거나 혹은 키보드에 있는 F5 단축키를 사용해서 우리의 프로그램을 실행할 수 있습니다. 만약 모든 것이 잘 진행되었다면 우리 프로그램은 아래와 같은 결과를 나타낼 것입니다.

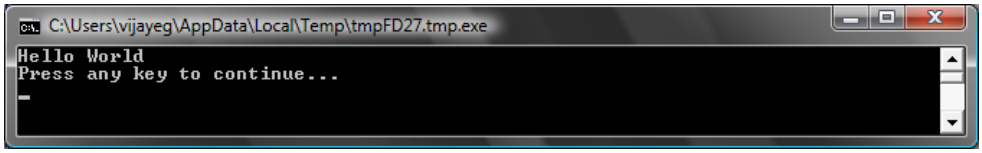

그림 3 – 첫 번째 프로그램 결과

축하합니다! 당신은 방금 당신의 첫 번째 Small Basic 프로그램을 작성하고 실행했습니다. 아주 작고 단순한 프로그램이지만, 그럼에도 불구하고 앞으로 진짜 컴퓨터 프로그래머가 되는 아주 큰 첫 발입니다. 이제 우리가 더 큰 프로그램을 만들기 전에 한가지 더 알아야 할 것이 있습니다. 우리는 방금 어떤 일이 일어났는지 이해해야 합니다. 우리가 정확히 무엇을 컴퓨터에게 명령했고 컴퓨터가 무엇을 해야 할 지를

어떻게 알았을까요? 다음 장에서는 우리가 방금 작성한 프로그램을 분석함으로써 그 답을 알아 보겠습니다.

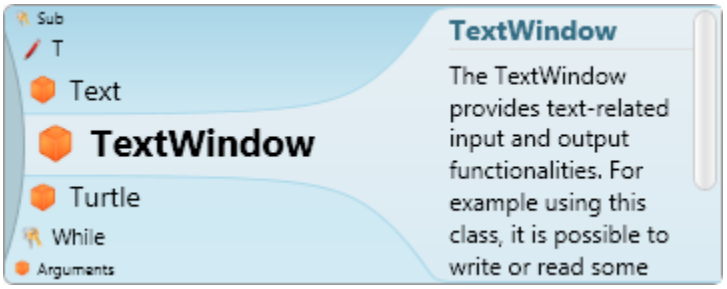

그림 4 - Intellisense

프로그램 저장하기

만약 Small Basic 을 종료한 후 방금 작성한 프로그램을 나중에 다시 작업하고 싶다면 프로그램을 저장할 수 있습니다. 사실 예기치 못한 사고나 전원의 단절로 인한 정보의 손실을 방지하기 위해 정기적으로 프로그램을 저장하는 것은 좋은 습관입니다. 도구 모음에 있는 "저장" 아이콘을 누르거나 "CTRL+S" 단축키 (CTRL 키를 누르고 있는 상태에서 S 키를 누르세요)를 사용해서 현재 프로그램을 저장할 수 있습니다.

첫 번째 프로그램을 입력하면서 일련의 항목 리스트와 함께 나타나는 팝업을 보았을 것입니다 (그림 4 4). 이것을 "intellisense" 라고 부르고 프로그램을 빠르게 입력하는데 도움을 줍니다. Up/Down 화살표 키를 사용하여 리스트를 살펴 볼 수 있고, 원하는 것을 찾았을 때 프로그램 안에서 선택한 항목을 추가하기 위해 <Enter> 키를 누를 수 있습니다.

제 2 장

첫 번째 프로그램 이해하기

컴퓨터 프로그램이란 도대체 무엇인가?

프로그램은 컴퓨터를 위한 일련의 명령들입니다. 이러한 명령들은 컴퓨터에게 정확히 무엇을 해야 하는지 말하고, 컴퓨터는 항상 그 명령들은 따릅니다. 인간과 마찬가지로 컴퓨터도 컴퓨터가 이해할 수 있는 언어로 된 명령들만을 따를 수 있습니다. 이러한 명령들을 프로그래밍 언어라고 부릅니다. 컴퓨터가 이해할 수 있는 아주 많은 언어들이 있고 **Small Basic** 도 그 중의 하나입니다.

여러분과 여러분 친구 사이에서 일어나는 대화를 상상해 보세요. 여러분과 여러분 친구들은 서로 정보를 주고 받기 위해 문장으로 만들어 지는 단어들을 사용할 것 입니다. 마찬가지로 프로그래밍 언어도 컴퓨터 에게 정보를 전달하는 문장으로 만들어 질 수 있는 단어들의 조합입니다. 그리고 프로그램은 기본적으로 개발자와 컴퓨터 모두에게 똑같이 이해되는 (어떤 경우는 단지 몇 개이고 어떤 경우 수천 개) 문장들의 집합입니다.

Small Basic 프로그램

일반적인 Small Basic 프로그램은 여러 개의 *문*으로 구성되어 있습니다. 프로그램의 모든 행은 문을 나타내고 모든 문은 컴퓨터를 위한 명령입니다. 우리가 컴퓨터에게 Small Basic 프로그램을 실행하라고 요청하면 컴퓨터는 프로그램을 받아서 그 프로그램의 첫 번째 문을 읽습니다. 컴퓨터는 우리가 무엇을 말하려고 하는지 이해하고 난 후 명령을 실행합니다. 컴퓨터가 첫 번째 문을 실행하면 다시 프로그램으로 돌아와서 두 번째 줄을 읽고 실행합니다. 컴퓨터는 프로그램의 마지막에 도달할 때까지 이 작업을 계속합니다. 그리고 프로그램은 종료됩니다.

첫 번째 프로그램으로 돌아가기

우리가 작성한 첫 번째 프로그램은 다음과 같습니다:

```
TextWindow.WriteLine("Hello World")
```

이것은 하나의 문으로 구성되는 아주 간단한 프로그램입니다. 이 문은 컴퓨터에게 Text Window 안에 **Hello World** 라는 한 문장을 출력하라고 말합니다.

컴퓨터의 관점에서 직역해서 말하면 다음과 같습니다:

> **Hello World** 라고 쓰세요

여러분은 이미 마치 문장이 여러 개의 단어로 쪼개 질 수 있는 것처럼 문도 더 작은 단위로 쪼개 질 수 있다는 것을 알아 챘을 수 도 있습니다. 첫 번째 문은 3 개의 구성 요소를 가지고 있습니다

a) TextWindow
b) WriteLine
c) "Hello World"

> 컴퓨터가 이해할 수 있는 많은 언어들이 있습니다. Java, C++, Python, VB 등은 단순한 소프트웨어 프로그램부터 복잡한 소프트웨어 프로그램까지 모두 개발하는 데 사용할 수 있는 강력한 현대 컴퓨터 언어입니다.

점, 괄호, 그리고 인용 어구는 컴퓨터에게 우리가 원하는 것을 이해시키기 위해 문에서 적절한 위치에 배치되어야 합니다.

여러분은 우리가 첫 번째 프로그램을 실행했을 때 나타난 검은 창을 기억 할 지도 모릅니다. 그 검은 창은 TextWindow 또는 콘솔이라고 부릅니다. 이것이 바로 프로그램의 결과가 나타나는 곳입니다. 우리 프로그램에서 **TextWindows** 는 *개체* 라고 부릅니다. 프로그램 안에는 우리가 사용할 수 있는 여러 개의 개체가 있습니다. 우리는 이 개체에 대해 여러 가지 다양한 연산을 수행 할 수 있습니다. 우리는 이미 우리 프로그램에서 WriteLine 연산을 사용했습니다. 또, WriteLine 뒤에 괄호 속에 **Hello World** 가 따라 옵니다. 이 텍스트는 WriteLine 연산의 입력으로 전해 지고 사용자에게 출력됩니다. 이것은 그 연산에 대한 입력이라고 부릅니다. 어떤 연산은 하나 이상의 입력을 갖고 어떤 연산을 아무 입력도 가지지 않습니다.

두 번째 프로그램

이제 우리의 첫 번째 프로그램을 이해하셨을 것입니다. 다음에는 조금 더 나아가 몇 가지 색깔을 추가해서 프로그램을 좀 더 멋지게 만들어 봅시다.

```
TextWindow.ForegroundColor = "Yellow"
TextWindow.WriteLine("Hello World")
```

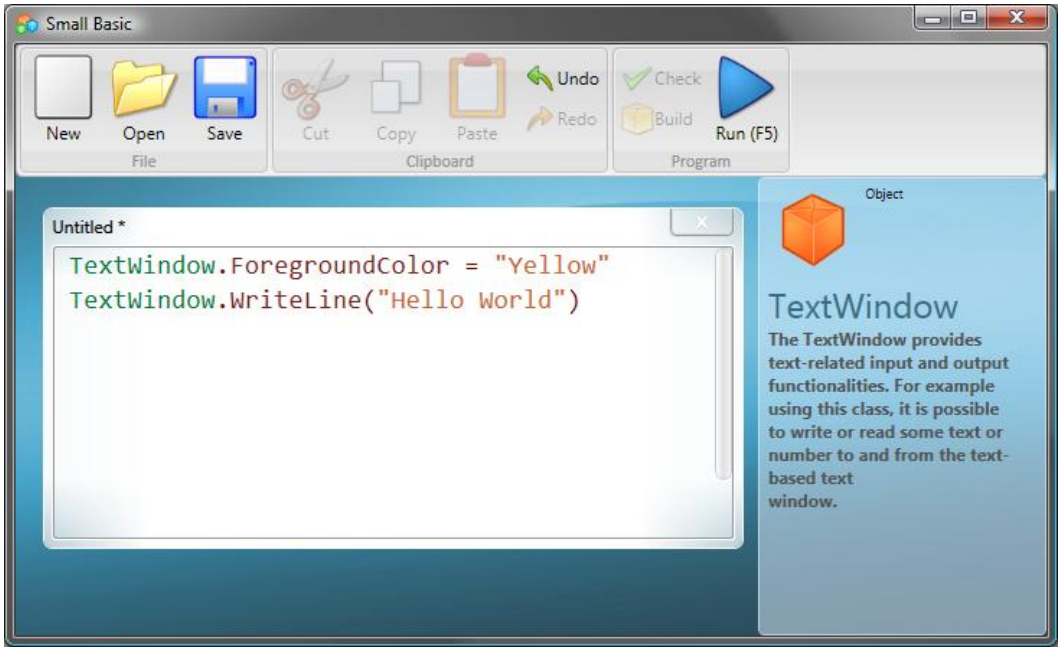

그림 5 – 색 추가

위의 프로그램을 실행하면 TextWindow 안에 이전과 같이 동일한 "Hello World" 구문을 출력하지만 이번에는 회색 대신 노란색으로 출력됩니다.

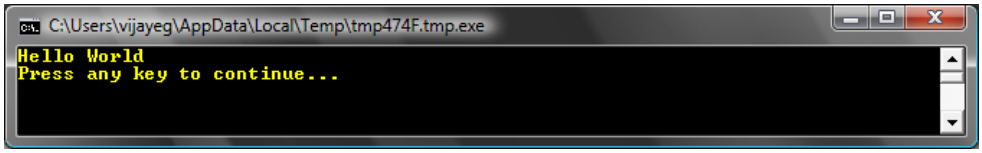

그림 6 – 노란색 Hello World

이전 프로그램에 추가된 새로운 문을 살펴 봅시다. 그 문은 *"Yellow"* 라는 값을 사용한 *ForegroundColor* 라는 새로운 단어를 사용합니다. 이것은 "Yellow"를 *ForegroundColor* 에 할당했다는 것을 의미 합니다. ForegroundColor 와 WriteLine 연산의 차이점은 ForegroundColor 는 어떤 입력도 받아 들이지 않고 어떤 괄호도 필요하지 않습니다. 대신 동일 부호와 단어가 뒤에 따라 옵니다. 우리는 ForegroundColor 를 TextWindow 의 속성이라고 합니다. 다음은 ForegroundColor 속성에 대한 유효한 값들입니다. "Yellow" 를 이 중 하나의 값으로 바꾸어 보고 결과를 봅시다. 따옴표를 잊지 마세요. 따옴표는 꼭 필요합니다.

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

제 3 장

변수 소개

프로그램에서 변수 사용하기

만약 우리 프로그램이 "Hello World?" 대신 "Hello" 와 함께 사용자 이름을 말한다면 더 좋지 않을까요? 그렇게 하기 위해서는 우선 사용자에게 사용자 이름을 물어 보고 그 이름을 어디엔가 저장한 후 "Hello"와 함께 그 사용자 이름을 출력해야 합니다. 그럼 어떻게 하는지 살펴 봅시다:

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

위와 같이 입력하고 프로그램을 실행하면 다음과 같은 결과를 보게 될 것 입니다:

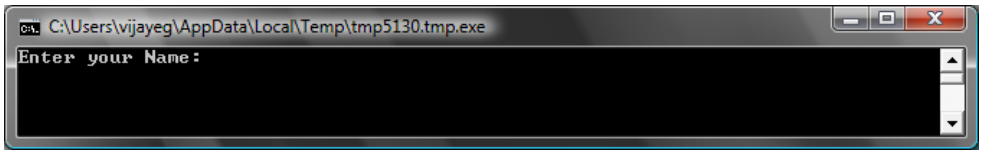

그림 7 – 사용자 이름 물어 보기

그리고 사용자 이름을 입력하고 ENTER 를 누르면 결과는 다음과 같습니다:

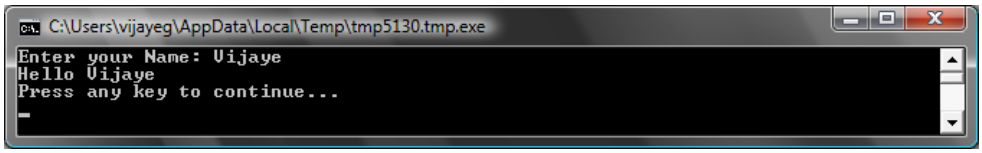

그림 8 - 다정한 Hello

만약 프로그램을 다시 실행하면 똑 같은 질문을 받게 될 것입니다. 다른 사용자 이름을 입력하면 컴퓨터는 그 이름과 함께 "Hello" 를 출력합니다.

프로그램 분석

방금 실행한 프로그램에서 아마 관심을 끄는 부분은 다음과 같을 것입니다:

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

Read() 는 *WriteLine()* 와 비슷하게 보이지만 입력이 없습니다. 이것은 컴퓨터에게 사용자가 무엇인가를 입력하고 ENTER 키를 누를 때까지 기다리라고 말하는 연산입니다. 사용자가 ENTER 키를 누르면 사용자가 입력한 것을 받아서 프로그램에게 보냅니다. 재미있는 점은 사용자가 무엇을 입력한 것이 무엇이던지 간에 그것은 **name** 이라는 변수에 저장됩니다. 변수는 어떤 값을 임시적으로 저장하고 나중에 사용할 수 있는 어떤 장소라고 할 수 있습니다. 윗줄에서 **name** 은 사용자 이름을 저장하기 위해 사용되었습니다.

다음 줄도 재미있습니다:

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

이 부분은 **name** 변수에 저장된 값을 사용하는 곳입니다. Name 안에 있는 값을 꺼내서 "Hello" 뒤에 붙인 후 TextWindow 에 출력합니다.

일단 변수가 정해지면 그 변수는 몇 번이고 사용할 수 있습니다. 예를 들면 다음과 같은 것도 가능합니다:

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

```
TextWindow.WriteLine("How are you doing " + name + "?")
```

그러면 다음과 같이 출력됩니다:

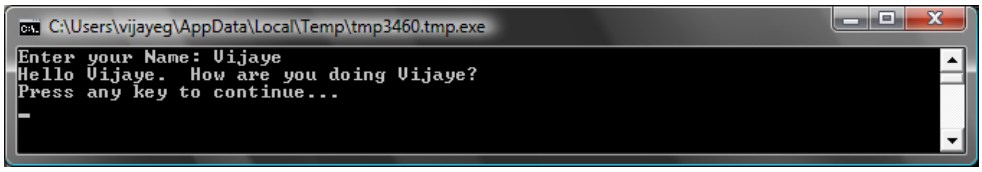

그림 9 – 변수 다시 사용하기

변수 이름 규칙

[TODO]

숫자와 친해지기

방금 사용자 이름을 저장하기 위해 변수를 어떻게 사용하는지 살펴 보았습니다. 다음 프로그램에서는 변수에 숫자를 어떻게 저장하고 다루는지 알아 보겠습니다. 아주 쉬운 프로그램부터 시작해 보겠습니다:

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

이 프로그램을 실행하면 결과는 다음과 같습니다:

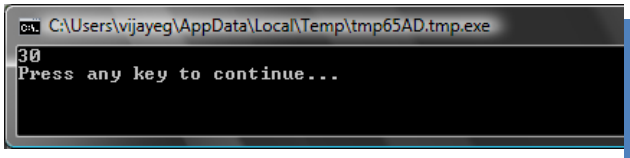

그림 10 – 두 숫자 더하기

숫자는 숫자 앞뒤에 따옴표가 없는 것에 주의하세요. 숫자에는 따옴표가 필요 없습니다. 따옴표는 텍스트를 사용할 때에만 필요합니다.

프로그램의 첫 번째 줄에서 **number1** 변수에 10 이라는 값을 할당합니다. 그리고 두 번째 줄에서 **number2** 변수에 20 이라는 값을 할당했습니다. 세 번째 줄에서 **number1** 과 **number2** 를 더하고 그

결과를 **number3** 에 할당합니다. 그리고, 이 경우 **number3** 은 30 이라는 값을 가지게 되고 그 값은 TextWindows 에 출력됩니다.

그러면 이 프로그램을 약간 변경한 후 결과를 봅시다:

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

위의 프로그램은 **number1** 과 **number2** 를 곱하고 그 결과를 **number3** 에 저장합니다. 결과는 다음과 같습니다:

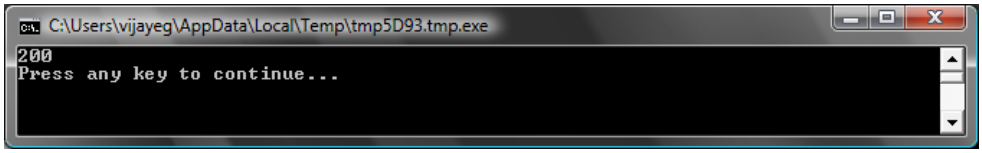

그림 11 – 두 숫자 곱하기

비슷한 방법으로 숫자를 빼고 나눌 수 있습니다. 빼기는 다음과 같습니다:

```
number3 = number1 - number2
```

나누기 기호는 '/'이고 다음과 같습니다:

```
number3 = number1 / number2
```

나누기의 결과는 다음과 같습니다:

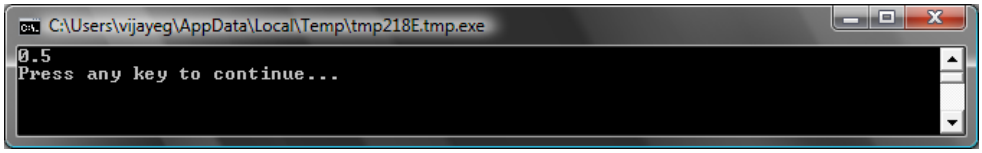

그림 12 – 두 숫자 나누기

간단한 온도 변환기

다음 프로그램에서 화씨 온도를 섭씨 온도를 바꾸기 위해 $℃ = \frac{5(°F-32)}{9}$ 이라는 공식을 사용하겠습니다.

우선 사용자로부터 화씨로 온도를 받아서 그 온도를 변수에 저장합니다. **TextWindow.ReadNumber** 는 사용자가 입력한 숫자를 읽을 수 있는 특별한 연산입니다.

```
TextWindow.Write("Enter temperature in Fahrenheit: ")
fahr = TextWindow.ReadNumber()
```

일단 화씨 온도를 변수에 저장하면 다음과 같이 섭씨로 변환할 수 있습니다:

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

괄호는 컴퓨터에게 **fahr – 32** 부분을 먼저 계산하고 나머지를 계산하라고 말합니다. 이제 남은 일은 사용자에게 결과를 출력하는 일입니다. 이 모든 것을 더한 프로그램은 다음과 같습니다:

```
TextWindow.Write("Enter temperature in Fahrenheit: ")
fahr = TextWindow.ReadNumber()
celsius = 5 * (fahr - 32) / 9
TextWindow.WriteLine("Temperature in Celsius is " + celsius)
```

프로그램 결과는 다음과 같습니다:

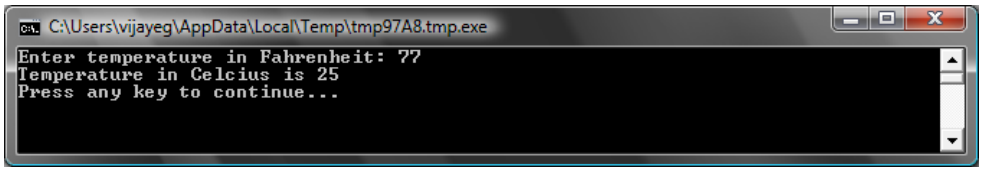

그림 13 – 온도 변환

제 4 장

조건과 분기

만약 우리의 첫 번째 프로그램이 항상 Hello World 라고 하지 않고 하루의 시간에 따라 *Good Morning World,* 혹은 *Good Evening World* 라고 말할 수 있다면 멋지지 않을까요? 다음 프로그램에서는 컴퓨터가 오후 12 시 전에는 *Good Morning World*, 그리고 오후 12 시 이후에는 *Good Evening World* 라고 출력하도록 만들어 보겠습니다.

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

언제 프로그램을 실행하느냐에 따라 다음과 같은 결과를 볼 수 있습니다:

그림 14 - Good Morning World

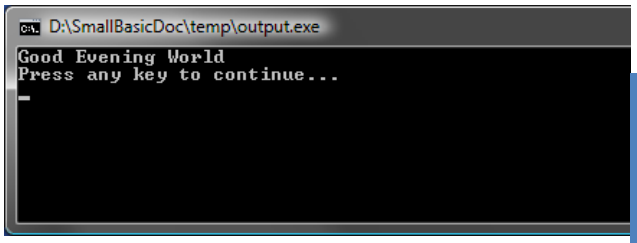

그림 15 - Good Evening World

Small Basic 에서는 Clock 개체를 사용하여 현재 날짜와 시간을 알 수 있습니다. 또한 여러 속성을 통해 현재 년, 월, 일, 시간, 분, 초 등을 제공합니다.

프로그램의 첫 번째 세 줄을 분석해 보도록 하겠습니다. 여러분은 이미 이 줄이 컴퓨터에게 Clock.Hour 가 12 보다 작으면 "Good morning World"를 출력하라고 명령한다는 것을 알고 있습니다. **If**, **Then** 와 **EndIf** 는 프로그램이 실행할 때 컴퓨터가 이해할 수 있는 특별한 단어들 입니다. **If** 다음에는 항상 조건이 따라 오고, 이 경우 그 조건은 (**Clock.Hour < 12)** 입니다. 컴퓨터가 여러분이 의도하는 바를 이해하기 위해서는 항상 괄호가 필요하다는 것을 기억해야 합니다. 그 조건 다음에는 **then** 과 실제로 수행될 행동들이 따라 옵니다. 그리고 그 행동 뒤에 **EndIf** 가 오고 이 것은 컴퓨터에게 조건적인 수행이 종료되었음을 말해 줍니다.

Then 와 **EndIf** 사이에 하나 이상의 행동들이 올 수 있고 조건이 유효하면 컴퓨터는 모든 행동들을 수행합니다. 예를 들면 아래와 같은 문장도 가능합니다:

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

Else

이번 장 앞부분의 프로그램에서 여러분은 어쩌면 두 번째 조건이 불필요하다고 생각하셨을지도 모르겠습니다. **Clock.Hour** 값은 12 보다 작거나 그렇지 않을 수 있습니다. 우리는 실제로 두 번째

검토가 필요하지 않습니다. 이와 같은 경우 우리는 두 개의 **if..then..endif** 문을 새로운 **else** 단어를 사용하여 하나로 줄일 수 있습니다.

Else 를 사용하여 다시 프로그램을 쓴다면 다음과 같습니다:

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

이 프로그램은 이전 프로그램과 완전히 동일하게 동작하고, 이 것은 우리에게 컴퓨터 프로그래밍에 있어서 아주 중요한 교훈을 줍니다

> *일반적으로 프로그래밍에서는 같은 일을 여러 가지 방법으로 할 수 있습니다. 종종 여러 방법들 중 더 나은 방법이 있습니다. 어느 것을 선택하느냐는 프로그래머에게 달려 있습니다. 여러분이 더 많은 프로그램을 작성하고 더 많은 경험을 쌓을수록 여러분은 서로 다른 테크닉들과 그들의 장점과 단점에 대해 알게 될 것입니다.*

들여쓰기

모든 예제에서 *If, Else* 와 *EndIf* 사이의 문장들이 어떻게 들여 쓰여 졌는지 볼 수 있습니다. 이 들여쓰기는 필요하지는 않습니다. 컴퓨터는 들여쓰기 없이도 아무 문제 없이 프로그램을 이해합니다. 그러나, 들여쓰기는 우리가 프로그램의 구조를 보고 쉽게 이해할 수 있도록 도와 줍니다. 그러므로, 블록 사이의 문장을 들여 쓰는 것은 일반적으로 좋은 습관으로 간주됩니다.

짝수 혹은 홀수

이제 **If..Then..Else..EndIf** 문을 가지고 좀 더 재미있게 어떤 주어진 숫자에 대해 그 숫자가 짝수인지 홀수 인지 알아 보는 프로그램을 작성해 봅시다.

```
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
```

이 프로그램을 실행하면 결과는 다음과 같습니다:

그림 16 – 짝수 혹은 홀수

이 프로그램에서 새롭고 유용한 **Math.Remainder** 라는 연산을 소개하였습니다. 맞습니다, 여러분이 이미 알아챈 것과 같이 **Math.Remainder** 는 첫 번째 숫자로 두 번째 숫자를 나누고 그 나머지를 제공합니다.

분기

두 번째 장에서 여러분은 컴퓨터가 프로그램의 문장을 한번에 하나씩, 그리고 위에서 아래로 처리한다는 것을 배웠습니다. 그러나, 이 순서와는 달리 컴퓨터가 다른 문장으로 점프하여 넘어 가게 만드는 특별한 문장이 있습니다. 다음 프로그램을 살펴봅시다.

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

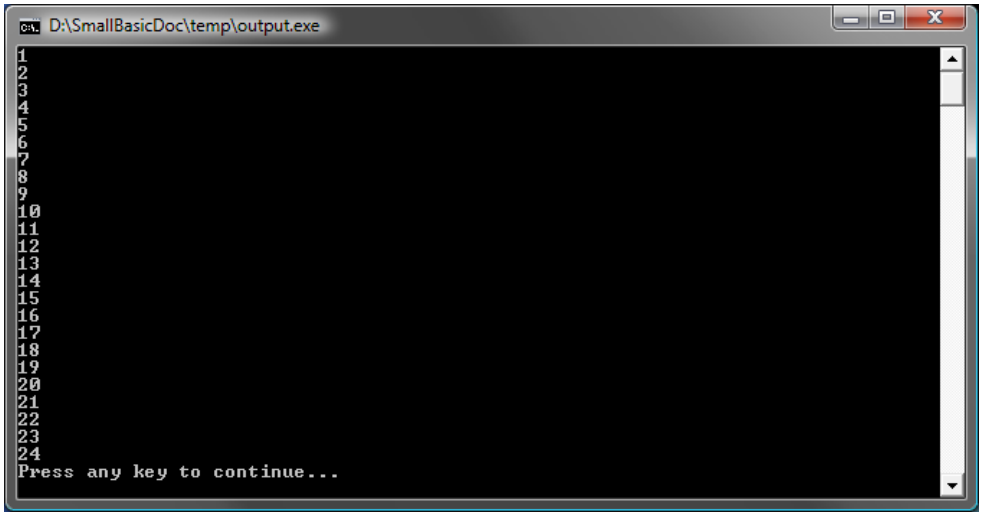

그림 17 - Goto 사용하기

위의 프로그램에서 우리는 변수 **i** 에 1 이라는 값을 할당했습니다. 그리고 콜론(:)으로 끝나는 새로운 문장을 추가했습니다.

```
start:
```

이것은 *라벨* 이라고 부릅니다. 라벨은 컴퓨터가 이해하는 북마크와 같습니다. 여러분은 북마크의 이름을 마음대로 지을 수 있고 프로그램에 이름이 다르기만 하면 원하는 대로 많은 라벨을 추가할 수 있습니다.

또 다른 재미있는 문장은 다음과 같습니다:

```
i = i + 1
```

위 문장은 컴퓨터에게 변수 **i** 에 1 을 더하고 다시 그 값을 **i** 에 할당합니다. 그러므로, 이 문장 전에 i 의 값이 1 이었다면 이 문장을 실행시킨 후에는 2 가 됩니다

마지막으로,

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

i 의 값이 25 보다 작으면 북마크 **start** 부터 문장을 실행하는 부분입니다.

끝이 없는 실행

Goto 문을 사용하여 컴퓨터가 어떤 일을 여러 번 반복하게 하도록 만들 수 있습니다. 예를 들면, 짝수 혹은 홀수 프로그램을 아래와 같이 변형하면 이 프로그램은 영원히 실행할 것입니다. 창의 오른쪽 상단 구석에 있는 종료 (X) 버튼을 눌러서 프로그램을 중지할 수 있습니다.

```
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
```

그림 18 – 끝없이 실행하는 짝수 혹은 홀수

제 5 장

반복문

For 문

이전 장에서 작성한 프로그램을 살펴 봅시다.

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

이 프로그램은 1 에서 24 까지 숫자를 순서대로 출력합니다. 변수를 증가시키는 과정은 프로그래밍에서 아주 일상적이므로 프로그래밍 언어는 일반적으로 이를 위한 쉬운 방법을 제공합니다. 위의 프로그램은 아래 프로그램과 동일합니다:

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

```
EndFor
```

그리고 결과는 다음과 같습니다:

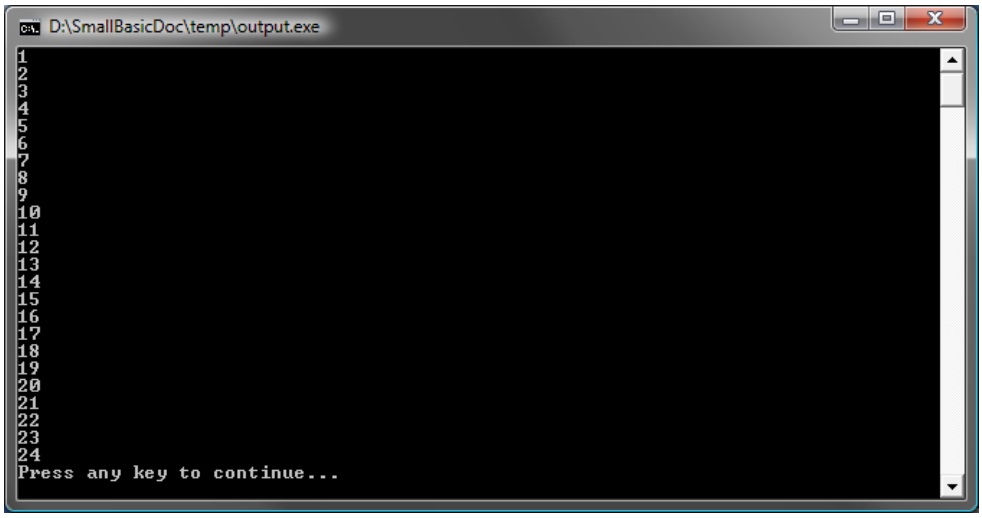

그림 19 – For 문 사용

우리는 8 줄의 프로그램을 4 줄 프로그램으로 줄였지만 여전히 8 줄 프로그램과 완전히 같은 일을 한다는 점에 주목하십시오! 같은 일을 하는데 여러 가지 다른 방법이 있다고 한 말 기억하세요? 방금 아주 좋은 예제를 보았습니다.

For..EndFor 은 프로그래밍 용어에서 *반복문* 이라고 부릅니다. 어떤 변수에 대해 어떤 초기값과 종료값을 주면 컴퓨터가 변수를 증가시켜 줍니다. 컴퓨터가 변수를 증가시킬 때는 언제나 **For** 과 **EndFor** 사이에 있는 문장을 실행합니다.

하지만 만약 변수가 1 대신 2 씩 증가하기를 원하는 경우에도 (예를 들면 1 과 24 사이에 있는 모든 홀수를 출력하기를 원하는 경우) 반복문을 사용할 수 있습니다.

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

그림 20 – 홀수만

For 문의 **Step2** 부분은 컴퓨터에게 i 의 값을 일반적인 1 대신에 2 씩 증가하라고 말합니다. **Step** 을 사용하여 원하는 증가값을 정할 수 있습니다. 아래 예제와 같이 **Step** 에 음수값을 명시해서 컴퓨터가 거꾸로 수를 셀 수 있도록 할 수도 있습니다.

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

그림 21 – 거꾸로 세기

While 문

While 문은 몇 번을 반복해야 할지 미리 알 수 없을 때 특히 유용한 또 다른 반복문의 한 방법입니다. For 문이 미리 정해진 숫자만큼 동작한다면 While 문은 미리 주어진 조건이 참이 될 때까지 동작합니다. 아래 예제는 결과가 1 보다 클 때까지 주어진 숫자를 이등분합니다.

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

```
  number = number / 2
EndWhile
```

그림 22 – 이등분하는 반복문

위의 프로그램은 *number* 에 100 을 할당하고 *number* 가 1 보다 큰 동안 계속 실행합니다. 반복문 안에서 *number* 를 출력하고 그 숫자를 2 로 나눕니다. 예상한 대로 프로그램의 결과는 계속해서 반으로 줄어드는 숫자입니다.

반복문을 몇 번 돌려야 하는지 알 수 없기 때문에 For 문을 사용하여 프로그램을 작성하기는 아주 어렵습니다. While 문을 사용하면 조건을 검토하고 컴퓨터에게 반복문을 계속할지 멈출지 요청하기가 아주 간단합니다.

흥미롭게도 모든 while 문은 If..Then 문으로 나타낼 수 있습니다. 예를 들어, 위의 프로그램은 최종 결과에 영향을 주지 않고 다음과 같이 다시 작성될 수 있습니다.

실제로 컴퓨터는 내부적으로 모든 while 문을 몇 개의 Goto 문과 함께 If..Then 을 사용하는 문장으로 다시 작성합니다.

```
number = 100
startLabel:
TextWindow.WriteLine(number)
number = number / 2
If (number > 1) Then
  Goto startLabel
EndIf
```

제 6 장

그래픽 시작하기

지금까지의 예제에서는 Small Basic 언어의 기본을 설명하기 위해 TextWindow 를 사용하였습니다. 하지만, Small Basic 은 강력한 그래픽 기능을 가지고 있고 이 장에서 그 기능들을 살펴 보도록 하겠습니다.

GraphicsWindow 소개

Small Basic 에서 TextWindow 을 사용하여 텍스트나 숫자에 대한 작업을 할 수 있듯이 **GraphicsWindow** 을 사용하여 그리는 작업을 할 수 있습니다. 우선 GraphicsWindow 를 표시하는 것부터 시작해 봅시다.

```
GraphicsWindow.Show()
```

이 프로그램을 실행했을 때 지금까지 보았던 검은 색 텍스트 창 대신에 아래 그림과 같은 흰 색 창이 생기는 것을 알 수 있을 것입니다. 아직은 이 창을 가지고 할 일이 별로 없지만, 이 창은 이 장에서 할 작업을 위한 기본 윈도우 입니다. 오른쪽 상단에 있는 'X' 단추를 눌러서 이 창을 닫을 수 있습니다.

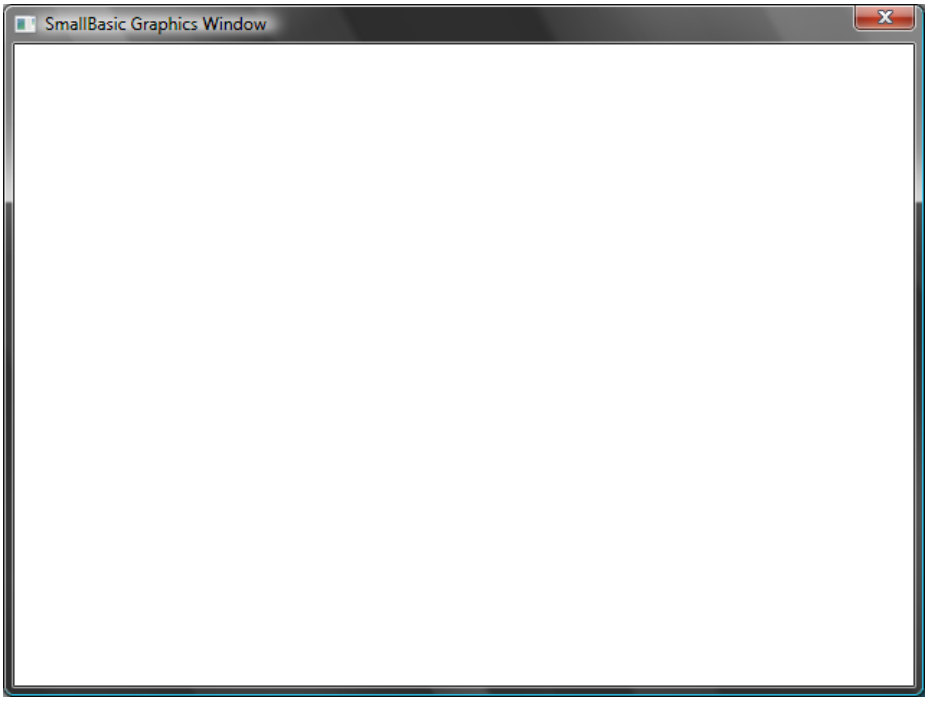

그림 23 - 빈 그래픽 창

그래픽 창 설정하기

제목, 배경색, 크기 등의 그래픽 창의 모양을 원하는 대로 바꿀 수 있습니다. 이 창에 대해 좀 더 친숙해지기 위해 모양을 변경해 봅시다.

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

여기 변경된 그래픽 창의 모습이 있습니다. 배경색을 부록 B 에 있는 값 중의 하나로 바꿀 수 있습니다. 창의 모양을 어떻게 바꿀 수 있는지 이 속성들을 가지고 시도해 봅시다.

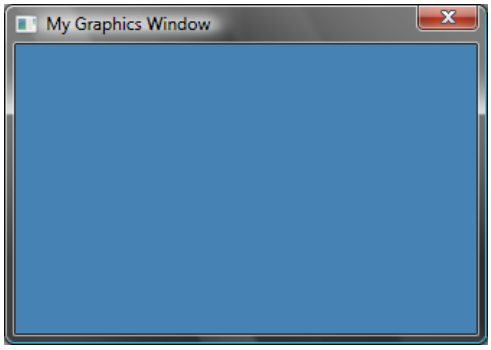

그림 24 – 사용자 지정 그래픽 창

선 그리기

모양, 텍스트, 심지어 그림까지도 GraphicsWindow 위에 그릴 수 있습니다. 우선 간단한 모양 몇 개를 그려 봅시다. 여기 그래픽 창에 선 몇 개를 그린 프로그램이 있습니다.

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

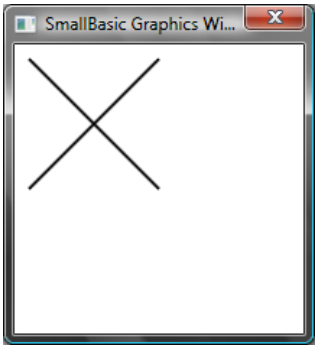

그림 25 – 교차

프로그램의 첫 두 줄은 창을 설정하고 그 다음 두 줄은 교차되는 선을 그립니다. *DrawLine* 뒤에 오는 첫 두 숫자는 시작하는 x 와 y 좌표를, 나머지 두

색의 이름을 사용하는 대신 웹의 색 표기법 (#RRGGBB)을 사용할 수 있습니다. 예를 들면, #FF0000 은 빨강색을 나타내고, #FFFF00 은 노란색을 나타냅니다. 앞으로 색에 대해 더 자세히 알아 보겠습니다.

숫자는 끝나는 x 와 y 좌표를 지정합니다. 컴퓨터 그래픽에서 흥미로운 점은 (0, 0) 좌표는 창의 왼쪽 상단에서 시작한다는 것입니다. 실제로, 좌표 공간에서 창은 두 번째 사분면으로 간주 됩니다.

[할 일: 사분면 그림 삽입]

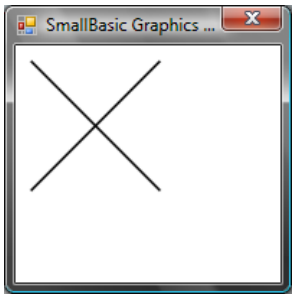

그림 26 – 좌표 지도

프로그램으로 다시 돌아가서, Small Basic 은 선의 색이나 두께와 같은 속성을 변경할 수 있습니다. 우선 아래와 같이 선의 색을 변경해 봅시다.

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

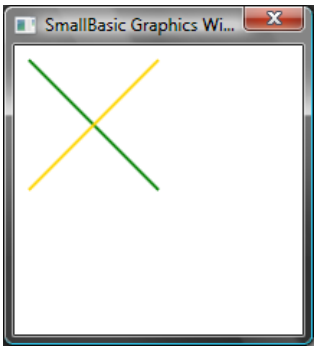

그림 27 – 선의 색 변경하기

이제 크기도 변경해 봅시다. 아래 프로그램에서는 선의 두께를 기본값이 1 대신에 10 으로 변경합니다.

```
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)
```

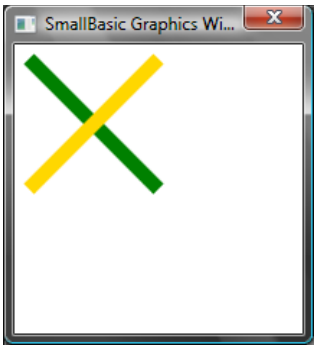

그림 28 – 두껍게 색칠한 선

PenWidth and *PenColor* 은 선을 그리는 펜을 변경합니다. 속성이 변경되면 선 뿐만 아니고 선이 그리는 형태에도 영향을 미칩니다.

지난 장에서 배운 반복문을 사용하면 펜의 두께를 증가시키면서 선들을 그리는 프로그램을 쉽게 작성할 수 있습니다.

```
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
```

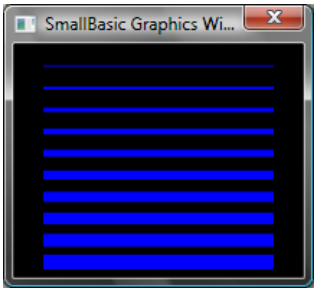

그림 29 – 다양한 펜 두께

프로그램에서 매번 반복문이 돌 때 때마다 *PenWidth* 가 증가하고 이 전의 줄 밑에 새로운 줄을 그리는 점이 흥미롭습니다.

모양 그리고 채우기

모양을 그릴 때 모든 일반적으로 두 가지 타입의 연산이 있습니다. *Draw* 연산과 *Fill* 연산입니다. Draw 연산은 펜을 사용하여 모양의 아웃라인을 그리고 Fill 연산은 브러시를 사용하여 모양을 칠합니다. 예를 들면, 아래 프로그램에는 두 개의 직사각형이 있는데 하나는 빨강색 펜을 사용하여 그렸고, 다른 하나는 초록색 브러시를 사용하여 채웠습니다.

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

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

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

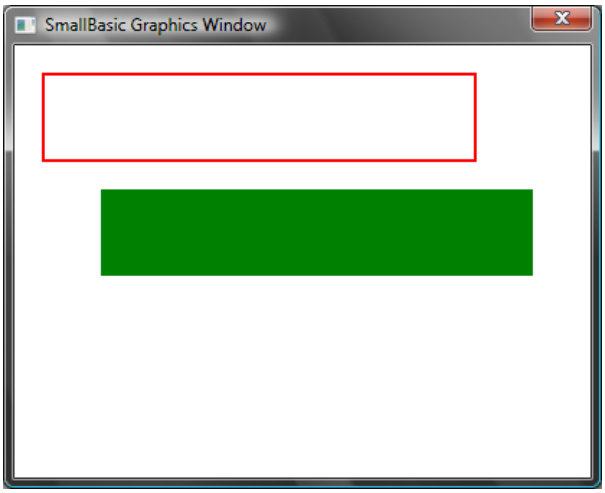

그림 30 그리기와 채우기

직사각형을 그리거나 채울 때 4 개의 숫자가 필요합니다. 첫 두 숫자는 직사각형의 왼쪽 상단의 X 와 Y 좌표를 나타냅니다. 세 번째 숫자는 직사각형의 넓이, 네 번째 숫자는 높이를 나타냅니다. 아래 프로그램에서 보듯이 타원을 그리거나 채울 때에도 같이 방법을 적용합니다.

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

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

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

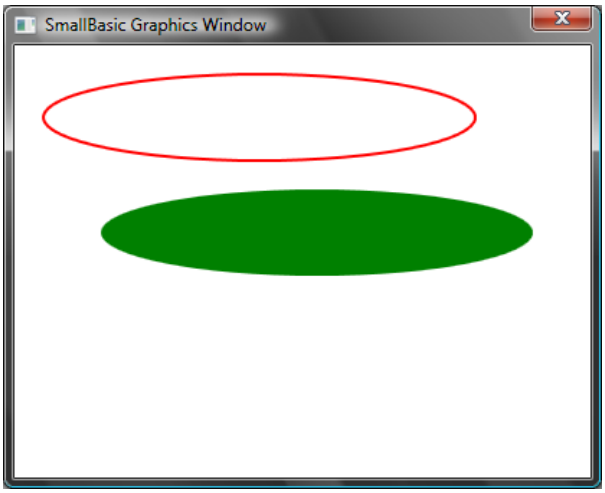

그림 31 - 타원 그리기와 채우기

타원은 원의 일반적인 경우입니다. 원을 그리려면 높이와 넓이를 동일하게 명시합니다.

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

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

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

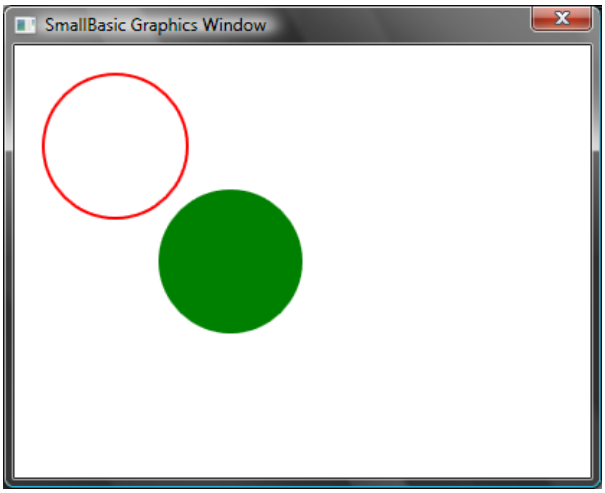

그림 32 – 원

제 7 장

모양 즐기기

지금까지 배운 것으로 좀 재미있는 것을 해보겠습니다. 이 장에서는 지금까지 배웠던 것들을 다양하게 조합하여 멋있게 보이는 프로그램을 만들어 봅니다.

연속 사각 모양

점점 크기를 늘려 가면서 여러 개의 직사각형을 반복하여 그립니다.

```
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
```

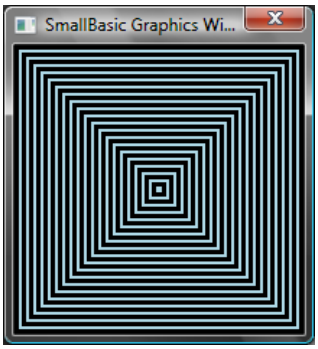

그림 33 – 각뿔 모양

연속 원형 모양

이전 프로그램을 변형하여 직사각형 대신 원을 그립니다.

```
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
```

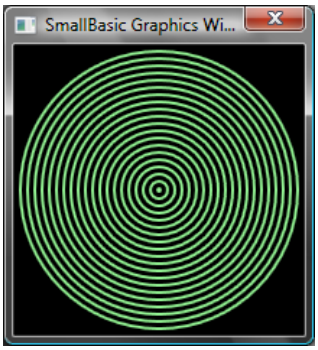

그림 34 – 원뿔 모양

무작위 수

이 프로그램은 *GraphicsWindow.GetRandomColor* 연산을 이용하여 브러시의 색상을 무작위로 설정하고 *Math.GetRandomNumber* 를 이용하여 원의 x 와 y 좌표를 설정합니다. 이 두 연산을 조합하여 실행할 때마다 다른 결과를 나타내는 재미있는 프로그램을 만들 수 있습니다.

```
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
```

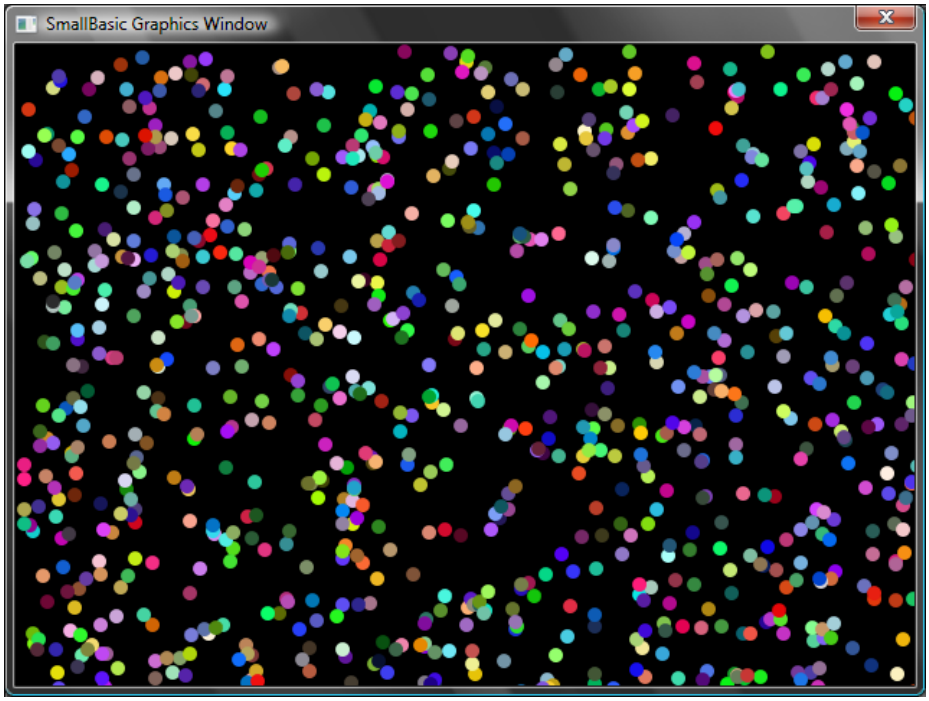

그림 35 – 무작위

프랙탈

다음 프로그램은 무작위 수를 이용하여 간단한 세모 프랙탈을 그립니다. 프랙탈은 부분으로 쪼갤 수 있는 기하학적 모양으로 각 부분은 부모의 모양과 정확히 일치합니다. 이 경우 프로그램은 부모 삼각형과 동일한 수 백 개의 삼각형을 그립니다. 이 프로그램은 몇 초 동안 실행되므로 단순한 점들로부터 삼각형이 천천히 만들어지는 것을 실제로 볼 수 있습니다. 이 논리는 설명하기 다소 어려우므로 이 부분은 스스로 한번 공부해 보세요.

```
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
```

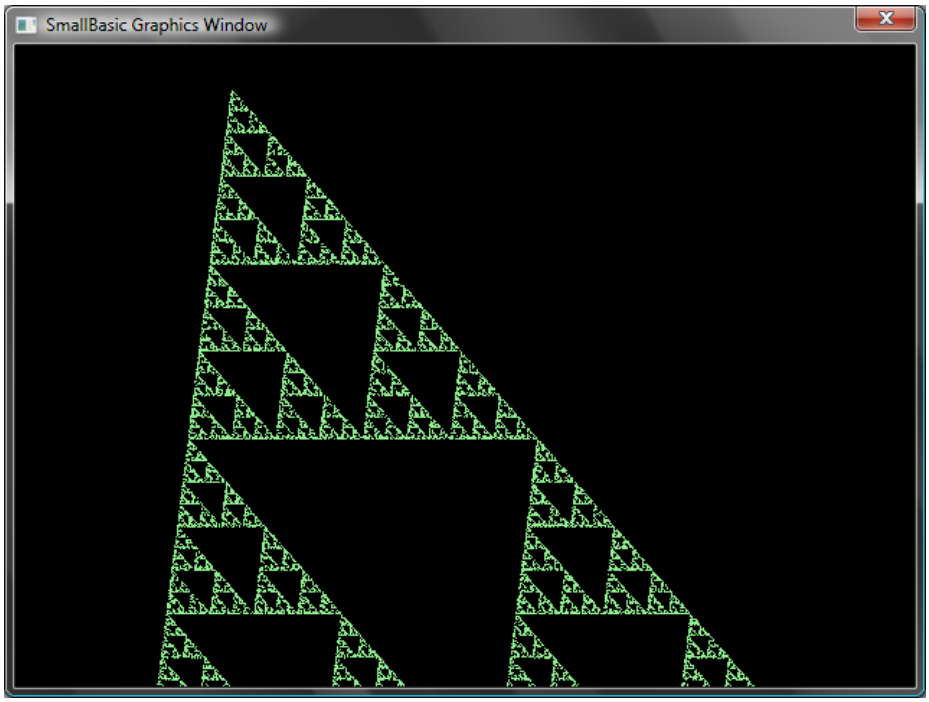

그림 36 - 세모 프랙탈

점들이 프랙탈을 천천히 형성하는 것을 정말로 보고 싶으면 **Program.***Delay* 연산을 이용하여 반복문을 지연시킬 수 있습니다. 이 연산은 얼마나 지연시킬지 밀리초 단위로 지정되는 숫자를 받습니다. 아래는 수정된 프로그램이고, 수정된 줄을 굵게 표시했습니다.

```
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
```

지연을 크게 하면 프로그램이 더 느려집니다. 숫자를 변경해 보면서 자기가 좋아하는 대로 바꿔보십시오.

프로그램을 더 변경하려면 아래 줄을

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

다음 줄로 수정해 보십시오.

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

이렇게 하면 무작위 색깔로 세모의 픽셀을 그리게 할 수 있습니다.

제 8 장

거북 그래픽

로고

1970 년대에는 소수의 연구자들에 의해서 사용되던 '로고'라고 불리던 아주 단순하지만 강력했던 프로그래밍 언어가 있었습니다. 그 후 누군가 이 언어에 화면에 나타나서 **전진**, **우회전**, **좌회전** 등의 명령에 반응하는 "거북 (Turtle)"을 사용하는 "거북 그래픽 (Turtle Graphics)"을 기능을 추가했습니다. 사람들은 이 거북이를 사용하여 재미있는 도형을 화면에 그릴 수 있게 되었습니다. 이러한 기능덕분에 이 언어는 다양한 연령대의 사람들이 손쉽게 사용할 수 있게 되었고 1980 년대에는 절정의 인기를 누렸습니다.

Small Basic 은 Small Basic 프로그램에서 호출되는 거북이 개체와 많은 명령어들을 제공하고 있습니다. 이 장에서는 거북을 사용하여 화면에서 그래픽을 그려보도록 하겠습니다.

거북 (Turtle)

거북이를 화면에 보이게 하는 것부터 시작합시다. 아주 간단한 한 줄짜리 명령으로 할 수 있습니다.

```
Turtle.Show()
```

이 프로그램을 실행시키면 이전 장에서 우리가 보았던 것과 똑 같은 흰 창이 나타나는데, 기존의 창과 다른 점은 화면 한가운데 거북이가 있다는 것입니다. 이것이 우리가 내리는 지시에 따라 움직이며 우리가 원하는 대로 그려주는 거북이입니다.

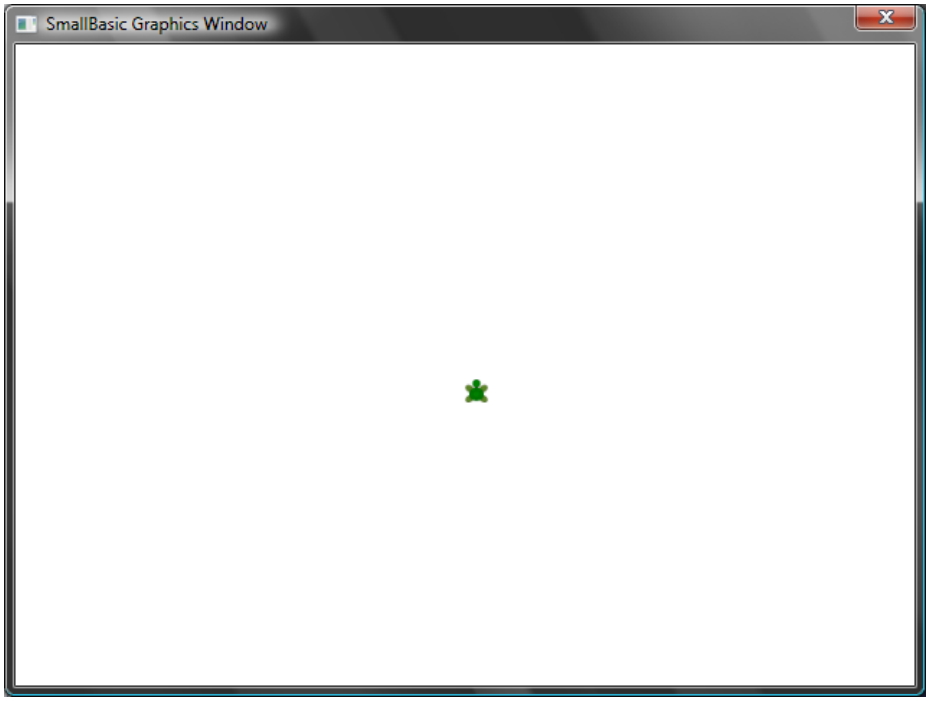

그림 37 - 거북이가 보인다

움직이고 그리기

거북이가 이해하는 명령중의 하나는 **Move** 입니다. 이 연산은 숫자를 입력으로 필요합니다. 해당 숫자는 거북이가 얼마나 멀리 움직일지를 지정합니다. 다음의 예제에서는 거북이에게 100 픽셀만큼 움직이라고 지시합니다.

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

이 프로그램을 실행시키면 거북이가 천천히 위쪽으로 100 픽셀만큼 움직이는 것을 볼 수 있습니다. 거북이가 움직이는 만큼 뒤로 선이 그려집니다. 거북이가 움직이는 것을 마쳤을 때 결과는 다음과 같습니다.

거북이에게 연산을 시킬 때 Show()를 호출할 필요는 없습니다. 어떠한 거북 연산이 수행될 때에도 거북은 자동으로 보여지도록 되어있습니다.

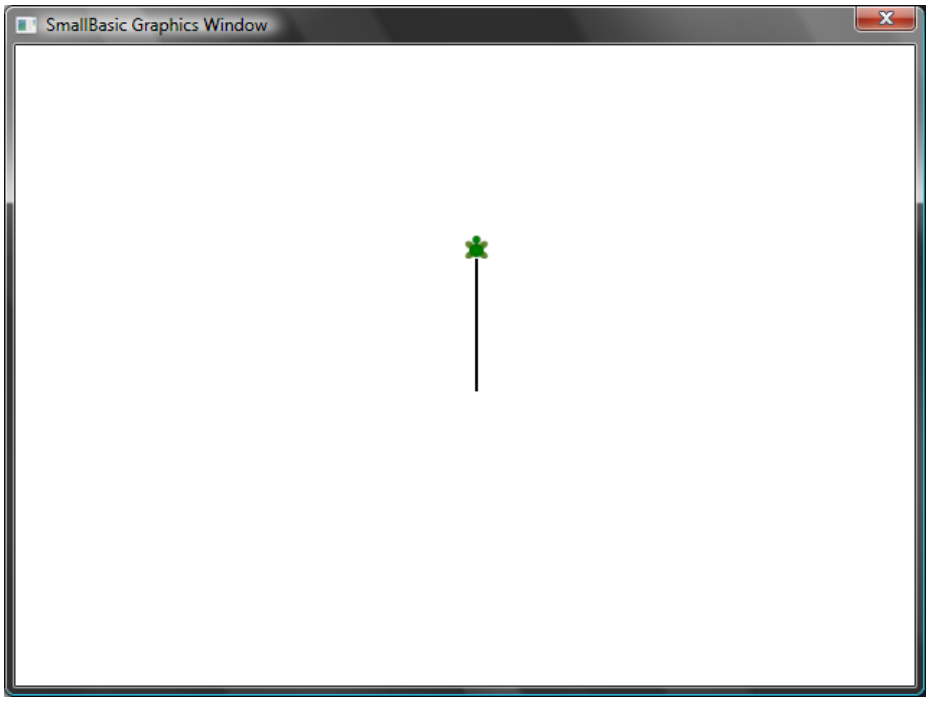

그림 38 – 100 픽셀만큼 움직이기

정사각형 그리기

정사각형은 네 개의 면 즉, 두 개의 세로 면과 두 개의 가로 면으로 이루어져 있습니다. 사각형을 그리기 위해서는 거북이가 선을 그리고, 우회전 하고, 다른 선을 그리는 동작을 네 면을 모두 그릴 때까지 계속 시켜야 합니다. 이 과정을 프로그램으로 옮기면 다음과 같습니다.

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

이 프로그램을 실행시키면 거북이가 한 번에 한 개의 선을 그리면서 사각형을 그리는 것을 볼 수 있는데, 결과적으로 아래 그림과 같게 됩니다.

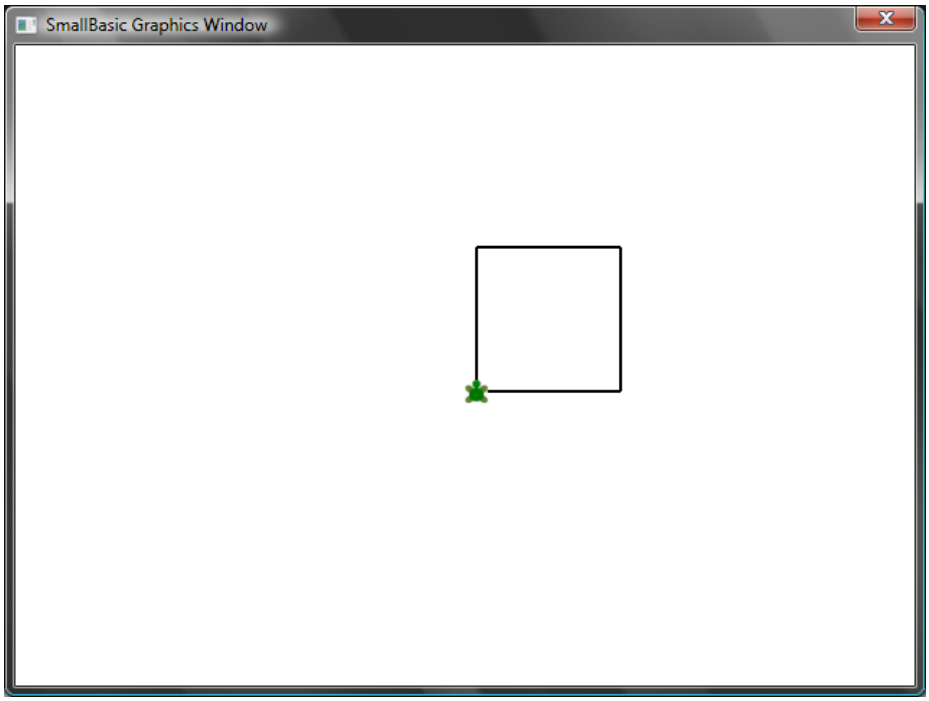

그림 39 - 사각형을 그리는 거북이

재미 있는 점은 똑 같은 두 개의 명령이 (정확히 네 번) 계속 반복된다는 것입니다. 반복되는 명령은 반복문을 이용하여 실행시킬 수 있다는 것을 이미 배웠습니다. **For..EndFor** 문을 사용하면 프로그램이 아주 단순해 집니다.

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

색 바꾸기

거북이는 우리가 이전 장에서 본 것과 정확하게 똑같은 GraphicsWindow 위에서 선을 그립니다. 그러므로 이전 장에서 배운 모든 연산을 여기에도 사용할 수 있습니다. 예를 들면, 다음 프로그램은 정사각형의 각 면을 각기 다른 색으로 그립니다.

```
For i = 1 To 4
  GraphicsWindow.PenColor = GraphicsWindow.GetRandomColor()
```

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

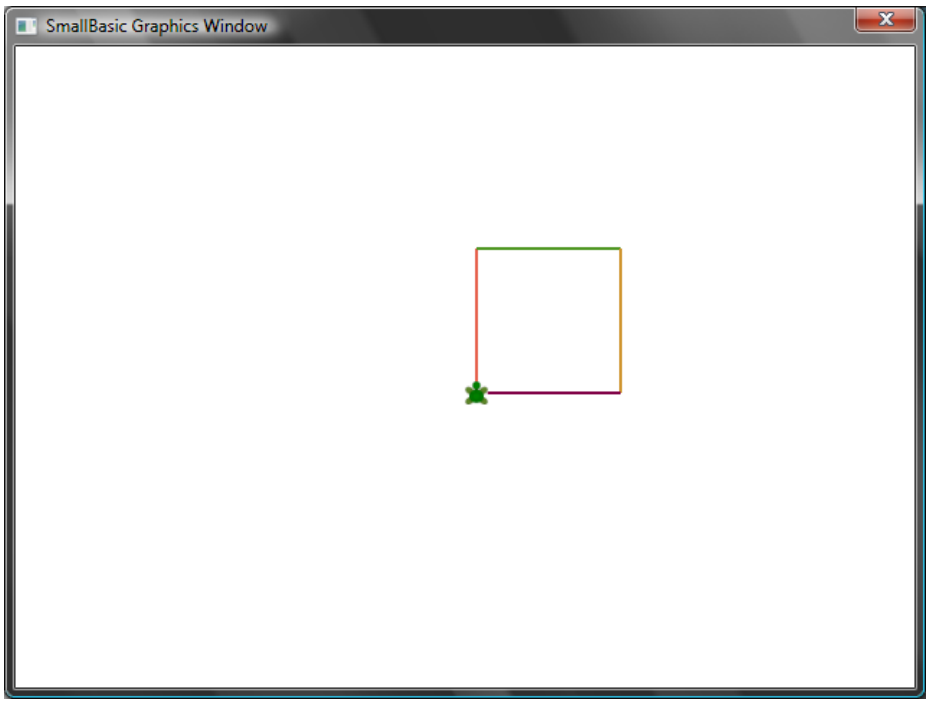

그림 40 - 색 바꾸기

복잡한 도형 그리기

거북이에게는 **TurnRight** 와 **TurnLeft** 연산 이외에도 **Turn** 연산이 있습니다. 이 연산은 회전 각도를 지정하는 입력이 필요합니다. 이 연산을 사용하면 어떤 다각형이라도 그릴 수 있습니다. 다음 프로그램은 육각형 (여섯 면을 가진 다각형)을 그립니다.

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

이 프로그램이 실제로 육각형을 그리는지 살펴 봅시다. 면간의 각도가 60 도이므로 **Turn(60)**을 사용하였습니다. 모든 면이 동일한 다각형의 경우 면간의 각도는 360 을 면의 수로 나눔으로써 쉽게

얻을 수 있습니다. 이런 정보를 가지고 변수를 사용한다면 몇 개의 면을 가진 다각형이라도 쉽게 일반적인 프로그램으로 나타낼 수 있습니다.

```
sides = 12
length = 400 / sides
angle = 360 / sides
For i = 1 To sides
  Turtle.Move(length)
  Turtle.Turn(angle)
EndFor
```

이 프로그램을 사용하여 **sides** 변수 값만 바꾸면 어떠한 다각형이라도 그릴 수 있습니다. 4 를 넣으면 우리가 시작했던 정사각형을 얻을 수 있습니다. 50 정도의 아주 큰 숫자를 넣으면 원과 구분하기 힘들게 됩니다.

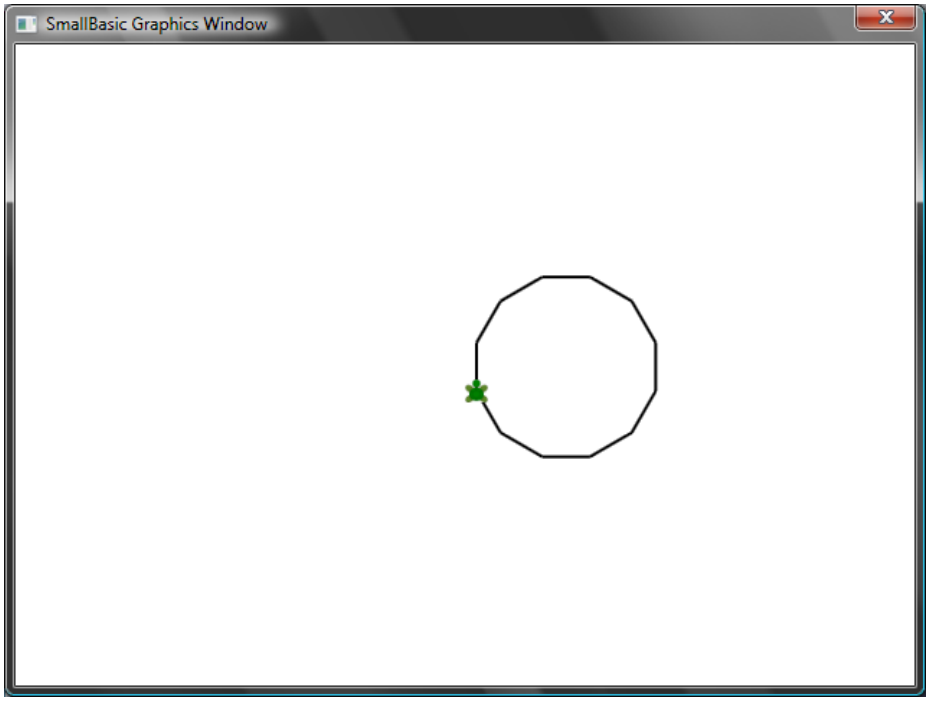

그림 41 - 12 각형 그리기

조금 전에 배운 기술을 이용하면 거북이로 하여금 여러 개의 원을 매번 조금씩 비키게 그리게 하여 재미있는 결과를 만들어낼 수 있습니다.

```
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
```

위의 프로그램에서, Speed 를 9 로 지정하여 거북이의 속도를 빠르게 하였습니다. 거북이의 속도는 1 부터 10 까지 원하는 만큼 빠르게 지정 할 수 있습니다.

위 프로그램은 두 개의 **For..EndFor** 반복문이 있는데, 하나가 다른 하나를 감싸고 있습니다. 안쪽의 반복문 (*i = 1 to sides*)은 다각형 프로그램과 비슷하며 원을 그리는데 사용됩니다. 바깥쪽의 반복문 (*j = 1 to 20*)는 한 개의 원을 그릴 때마다 거북을 조금씩 옆으로 비키게 합니다. 이렇게 하여 거북은 20 개의 원을 그립니다. 이 모든 것을 합쳐 이 프로그램은 아래 그림에서 보이는 재미있는 형태를 보여줍니다.

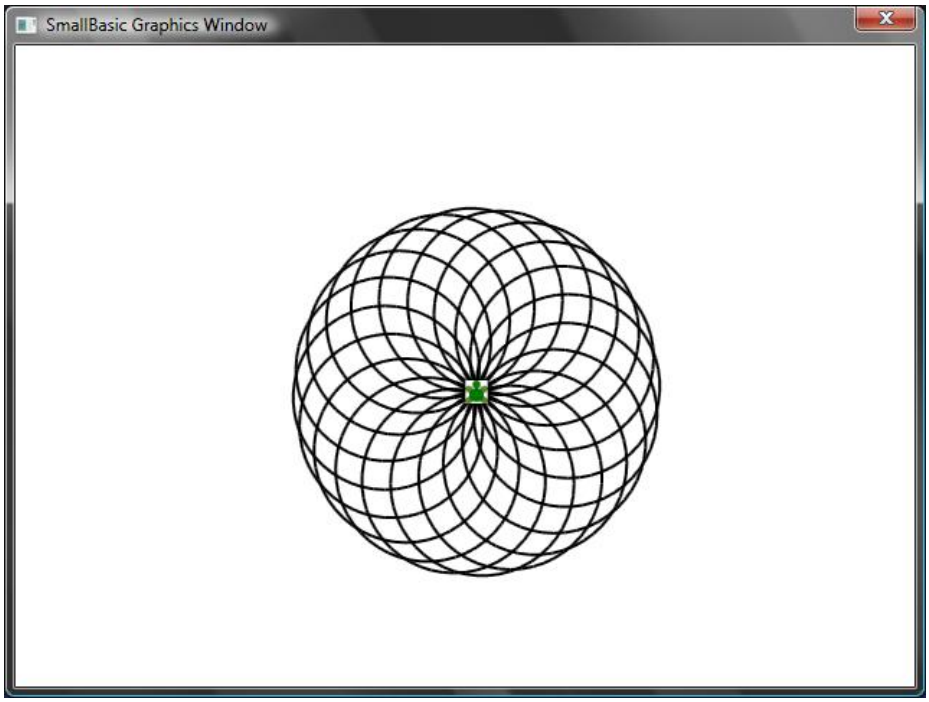

그림 42 – 원 안으로

이리저리 움직이기

PenUp 연산을 이용하면 거북이가 선을 그리지 않도록 할 수 있습니다. 이렇게 하면 거북을 화면 어디로든 선을 그리지 않고 옮길 수 있습니다. **PenDown** 연산을 호출하면 거북에게 다시 그리도록 합니다. 이 연산들은 점선과 같은 재미있는 효과를 내는데 사용될 수 있습니다. 다음 프로그램은 점선으로 이루어진 다각형을 그립니다.

```
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
```

이 프로그램은 두 개의 반복문으로 이루어져 있습니다. 안쪽의 반복문은 점선 한 개를 그리고 바깥쪽의 반복문은 얼마나 많은 선을 그릴지 규정합니다. 이 예제에서는 **side** 변수로 6 을 사용하였으므로 점선으로 이루어진 육각형이 그려집니다.

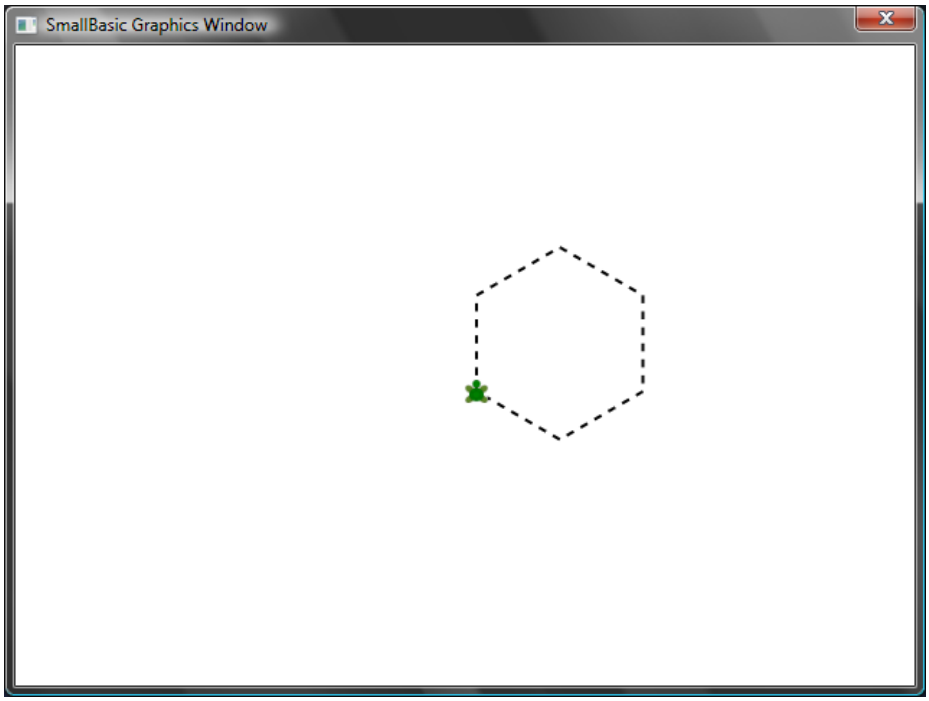

그림 43 - PenUp 과 PenDown 사용하기

제 9 장

서브루틴

프로그래밍을 하다 보면 동일한 단계를 계속 반복해야 하는 경우가 종종 발생합니다. 이런 경우 동일한 명령문을 계속 반복해서 쓴다는 것은 말이 되지 않겠죠? 이런 상황에서 서브루틴이 편리하게 사용됩니다.

서브루틴은 큰 프로그램 안의 한 코드 부분으로서, 매우 특정한 역할을 하며 프로그램의 어디서나 호출될 수 있습니다. 서브루틴은 **Sub** 키워드 뒤에 오는 이름으로 식별되며 **EndSub** 키워드로 끝납니다. 예를 들어, 다음 코드 조각은 이름이 *PrintTime* 이고 현재 시각을 TextWindow 로 인쇄하는 서브루틴을 나타냅니다.

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

다음은 서브루틴을 포함하며 이를 다양한 곳에서 호출하는 프로그램입니다.

```
PrintTime()
TextWindow.Write("Enter your name: ")
name = TextWindow.Read()
```

```
TextWindow.Write(name + ", the time now is: ")
PrintTime()
Sub PrintTime
  TextWindow.WriteLine(Clock.Time)
EndSub
```

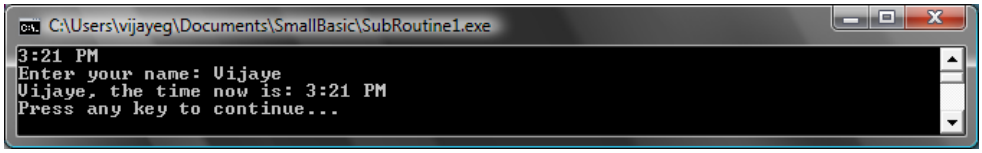

그림 44 – 간단한 서브루틴 호출

서브루틴이름() 을 호출하는 방식으로 서브루틴을 실행합니다. 컴퓨터에게 서브루틴이 필요함을 알리려면, 늘 그렇듯이 "()" 기호가 필요합니다.

같은 프로그램 안에 있는 Small Basic 서브루틴만을 호출할 수 있다는 점을 기억하세요. 다른 프로그램 안에 있는 서브루틴을 호출할 수는 없습니다.

서브루틴의 장점

위에서 보았듯이, 서브루틴은 입력해야 할 코드의 양을 줄여줍니다. *PrintTime* 서브루틴을 쓴 후, 프로그램 어디에서든 이것을 호출할 수 있으며 현재 시각을 인쇄할 수 있습니다.

추가로, 서브루틴은 복잡한 문제를 간단한 조각으로 분리하는 데 도움이 됩니다. 풀어야 할 복잡한 수식이 있다고 가정할 때, 복잡한 수식의 작은 조각을 해결하는 몇 개의 서브루틴을 쓸 수 있습니다. 그 후, 그 각각의 결과를 합쳐 원래의 복잡한 수식의 답을 구할 수 있습니다.

서브루틴은 프로그램의 신뢰성을 향상시키는데도 도움을 줍니다. 달리 말해, 프로그램에서 공통으로 실행되는 부분에 이름이 잘 정해진 서브루틴을 사용한다면, 프로그램이 읽기 쉽고 이해하기 쉬워집니다. 다른 사람의 프로그램을 이해하거나, 다른 사람이 내 프로그램을 이해하기를 원할 경우 이는 매우 중요합니다. 프로그램을 쓴 후 일주일 후에 다시 읽어야 하는 경우처럼, 내 프로그램을 읽을 때도 매우 유용합니다.

변수 사용하기

서브루틴에서 프로그램 상의 모든 변수를 접근하고 사용할 수 있습니다. 예를 들어, 다음 프로그램은 두 개의 숫자를 받아들여 둘 중 큰 숫자를 프린트합니다. *max* 변수가 서브루틴의 내부와 외부에서 사용된 것을 주목하십시오.

```
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
```

프로그램의 출력은 다음과 같습니다.

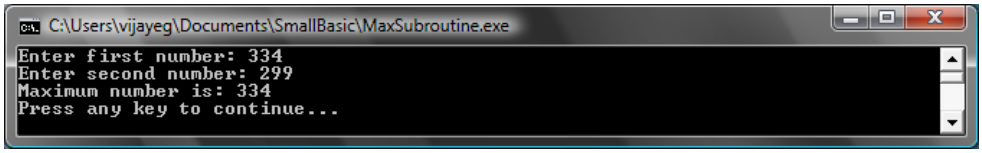

그림 45 – 서브루틴을 사용하여 두 수의 최대값 구하기

서브루틴의 사용을 보여주는 다른 예를 살펴봅시다. 이번에는 변수 *x* 및 *y*에 저장되는 다양한 점을 계산하는 그래픽 프로그램을 사용할 것입니다. 그 후 이 x 및 y 를 중심으로 원을 그리는 **DrawCircleUsingCenter** 서브루틴을 호출합니다.

```
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
```

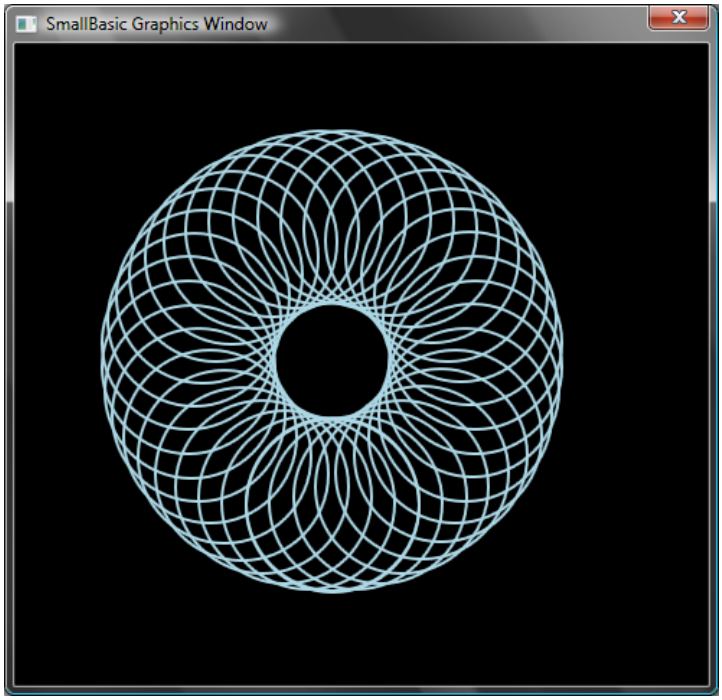

그림 46 – 서브루틴을 사용한 그래픽 예제

반복문 내에서 서브루틴 호출하기

한 개 이상 변수의 서로 다른 값으로 동일한 명령문을 실행하는 반복문에서 종종 서브루틴이 호출됩니다. 예를 들어, 특정 숫자가 소수 인지 아닌지를 결정하는 *PrimeCheck* 이라는 서브루틴이 있다고 가정합시다. 아래와 같은 서브루틴을 사용해서, 사용자가 특정 값을 입력하면 그 것이 소수인지 아닌지를 말해주는 프로그램을 작성할 수 있습니다. 다음 프로그램 이를 표현합니다.

```
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 서브루틴은 i 라는 숫자를 받아들인 후 i 보다 작은 숫자로 i 를 나누어 봅니다. 특정 숫자가 i 를 나누는 데 성공하고 나머지가 없다면 이 i 는 소수가 아닙니다. 이 시점에서 해당 서브루틴은 *isPrime* 의 값을 "False"로 설정하고 빠져 나옵니다. 해당 숫자가 자기 보다 작은 숫자로 나누어지지 않는다면 *isPrime* 의 값은 "True"로 유지 됩니다.

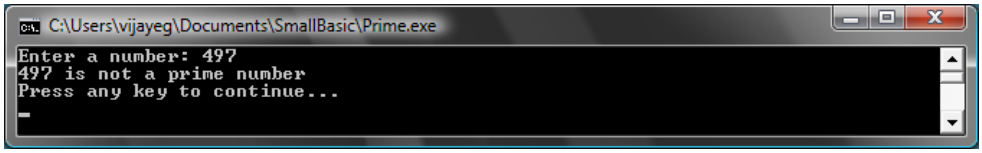

그림 47 – 소수 검사하기

자, 이제 소수 테스트를 해 줄 서브루틴이 있으니, 소수 목록을 만드는데 사용해 볼까요? 100 이하의 소수를 구해 봅시다. 위 프로그램을 수정하고 반복문 안에서 *PrimeCheck* 호출을 하도록 하는 것은 아주 간단합니다. 이 프로그램은 반복문이 실행되는 동안 서브루틴에게 계산시킬 서로 다른 값을 줍니다. 아래 예가 어떻게 수행되는지 살펴봅시다.

```
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
```

위 프로그램에서, *i* 값은 반복문이 실행되는 동안 계속 바뀝니다. 반복문에서 서브루틴 *PrimeCheck* 로의 호출이 이루어 집니다. 서브루틴 *PrimeCheck* 은 *i* 값을 받아들여 *i* 가 소수인지 아닌지를 계산합니다. 이 결과는 변수 *isPrime* 에 저장되고 서브루틴 밖의 반복문에서 이 변수에 접근합니다. 소수로 판명되면 이 변수 *i* 는 출력됩니다. 반복문은 3 에서 시작하여 100 까지 수행되므로 모든 출력되는 숫자는 3 과 100 사이에 있습니다. 이 프로그램의 결과는 다음과 같습니다.

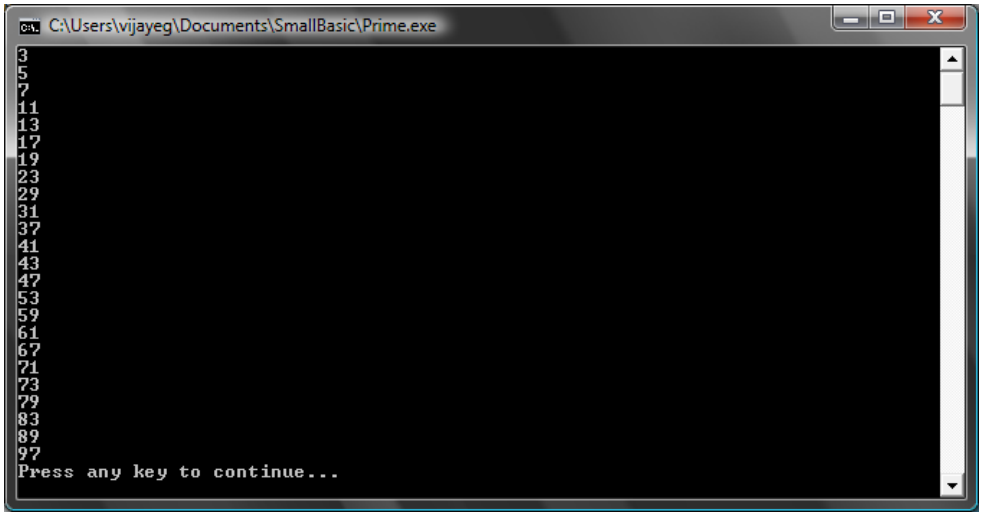

그림 48 – 소수

제 10 장

배열

이제는 변수를 어떻게 사용하는지 아시겠지요? 제 10 장으로 올때까지 재미있으셨나요?

우리가 변수를 사용해서 만든 프로그램을 다시한번 볼까요:

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

이 프로그램에서는 사용자 이름을 받아서 name 이라는 변수에 저장하였습니다. 자, 이제는 사용자가 한명 이상 있다고 가정해봅시다. 우리가 어떻게 모든 이름을 다 저장할수있을까요? 한가지 방법은 아래와 같습니다:

```
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)
```

위 프로그램을 실행하면 아래 결과를 얻을것입니다:

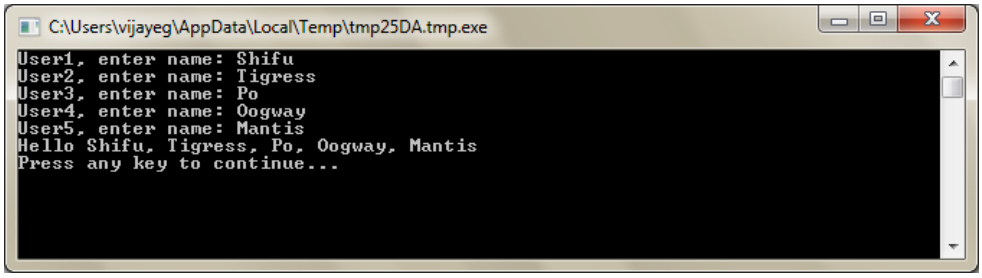

그림 49 – 배열을 사용하지 않은 경우

이런 간단한 프로그램을 만들기에 더 좋은 방법이 있겠지요? 특히 컴퓨터는 반복작업을 잘하기 때문에, 우리가 같은 코드를 각 새로운 사용자를 위해 반복해서 쓸 이유가 없겠지요. 간단한 방법은 같은 변수를 사용하여 한개 이상의 사용자 이름을 저장한후에 가지고 오는것입니다. 우리가 이런 방법으로 할수있다면, 이미 배운 for 반복문 을 사용할수 있습니다. 이때 배열이 우리에게 도움이 될때이지요.

배열은 무엇인가?

배열은 한번에 한개 이상의 값을 가질수있는 특별한 종류의 변수입니다. 이것이 의미하는것은 다섯사용자의 이름을 저장하기 위해서 **name1, name2, name3, name4, name5** 을 만드는 대신, name 을 사용하여 다섯 사용자의 이름을 저장할수있습니다. 우리가 여러값을 저장하는 방법은 "index(인덱스)"

라는것을 사용하는것입니다. 예를 들어, **name[1], name[2], name[3], name[4], name[5]** 이 각각 값을 저장 할수 있습니다. 이 1, 2, 3, 4, 5 는 배열의 인덱스입니다.

name[1], **name[2]**, **name[3]**, **name[4]**, **name[5]** 이 모두 다른값인것 처럼 보이지만, 실제로 이 다섯개는 하나의 변수입니다. 그럼 이것의 장점이 무엇이냐고 물으실수있겠지요. 배열에 값을 저장하는 가장 좋은 점은 반복문 안에 배열들을 쉽게 액세스 할수 변수를 사용하여서 인덱스를 지정할수 있다는 것입니다.

이제 배열을 사용하여 이전 프로그램을 다시 만들어서 우리가 배운 지식을 사용하여볼까요.

```
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("")
```

훨씬 읽기 쉽지요? bold 로된 두 줄을 보세요. 처음것은 배열에 값을 저장하고 두번째것은 배열에서 그값을 읽어옵니다. **name[1]** 에 저장하는 값은 **name[2]** 에 저장하는값에 의해 영향을 받지 않습니다. 그렇기에, 대부분 경우에 **name[1]** 과 **name[2]** 을 같은 ID 을 가진 두 변수로 취급할수 있습닌다.

그림 50 – 배열 사용한 경우

위에 프로그램은 배열이 없는것과 Mantis 끝에 따옴표외에는 똑같은 결과를 보여줍니다. 이것은 아래와 같이 반복문을 다시 프린트 함으로서 고칠수있습니다:

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

인덱스 배열

이전 프로그램에서 우리가 숫자를 인덱스로 사용하여서 배열에서 값을 저장하고 가지고 오는것을 보았습니다. 인덱스가 숫자에만 제한되는건 아니고 문자 인덱스를 사용해도 유용합니다. 예를 들어, 아래 프로그램에서, 우리는 사용자에 대한 여러 정보를 요청하고 저장한후에 사용자가 요청한 정보를 출력합니다.

```
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])
```

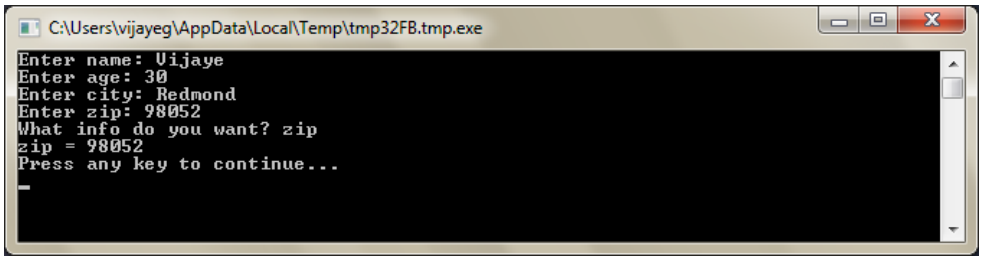

그림 51 – 숫자가 아닌 인덱스 사용하기

다차원

만약 모든 친구들의 이름과 전화 번호를 저장해서 필요한 경우 친구들의 전화 번호를 찾아 보고 싶다고 가정해 봅시다. 전화 번호부와 같은 이런 프로그램은 어떻게 만들 수 있을까요?

이 경우 두 세트의 인덱스가 필요하고 이것은 배열의 차원이라고도 합니다. 친구를 그 친구의 별명으로 부른다고 가정해 봅시다. 이 별명은 배열의 첫 번째 인덱스가 됩니다. 이 첫번 째 인덱스를 사용해서 친구 변수를 가져 오면, name 과 phone number 라는 두번 째 인덱스는 그 친구의 실제 이름과 전화 번호를 알아 낼 수 있습니다.

배열 인덱스는 대소 문자를 구별하지 않습니다. 일반적인 변수와 같이 배열 인덱스도 정확한 대소문자로 일치할 필요는 없습니다.

이 데이터를 저장하는 방법은 다음과 같습니다.

```
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"
```

Friend 라는 같은 배열에 두 개의 인덱스가 있으므로 이 배열은 이차원 배열이라고 합니다.

이렇게 프로그램을 만들었으면 친구의 별명을 입력으로 받아서 친구에 대해 저장한 정보를 출력할 수 있습니다. 이 모든 것을 포함하는 프로그램은 다음과 같습니다.

```
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"])
```

그림 52 – 간단한 전화번호부

배열을 사용하여 그리드 표시 하기

일반적으로 다중 차원 배열을 사용해서 그리드나 테이블을 나타낼 수 있습니다. 그리드는 행과 열이 있고, 이는 다중 차원 배열에 적합합니다. 그리드에 네모를 표시하는 간단한 프로그램은 다음과 같습니다.

```
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
```

이 프로그램은 사각형을 추가하고 8x8 그리드에 표시 합니다. 또한, 사각형을 배열에 저장합니다. 배열에 저장하면 사각형을 추적하고 나중에 필요한 경우 다시 사용하기 편리합니다.

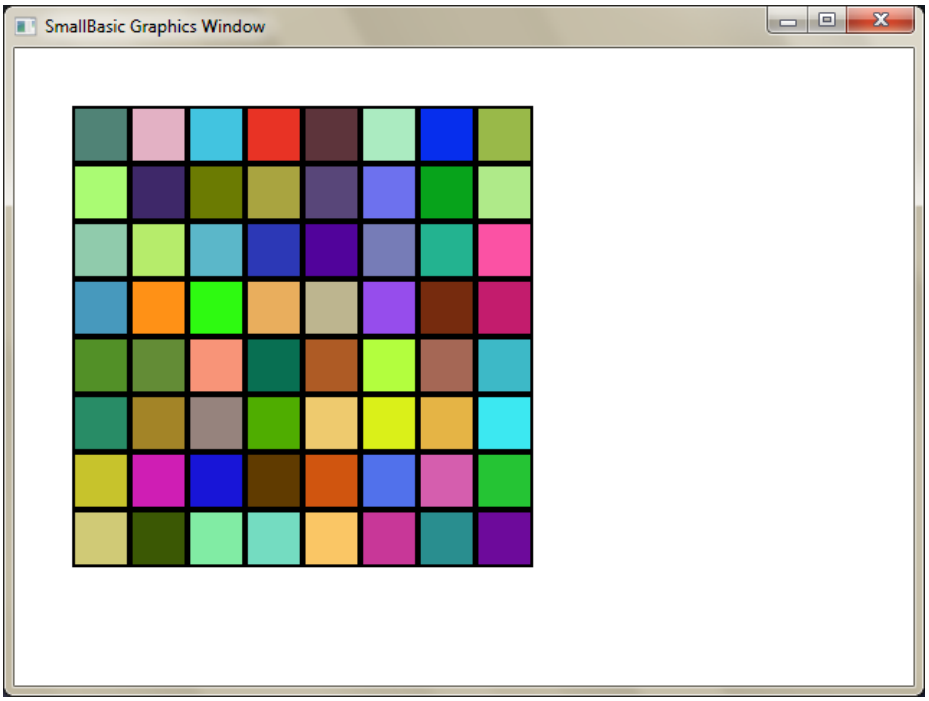

그림 53 - 그리드에 사각형 표시하기

예를 들면, 이전 프로그램 끝에 다음 코드를 추가하면 사각형들이 왼쪽 위 모서리로 움직입니다.

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

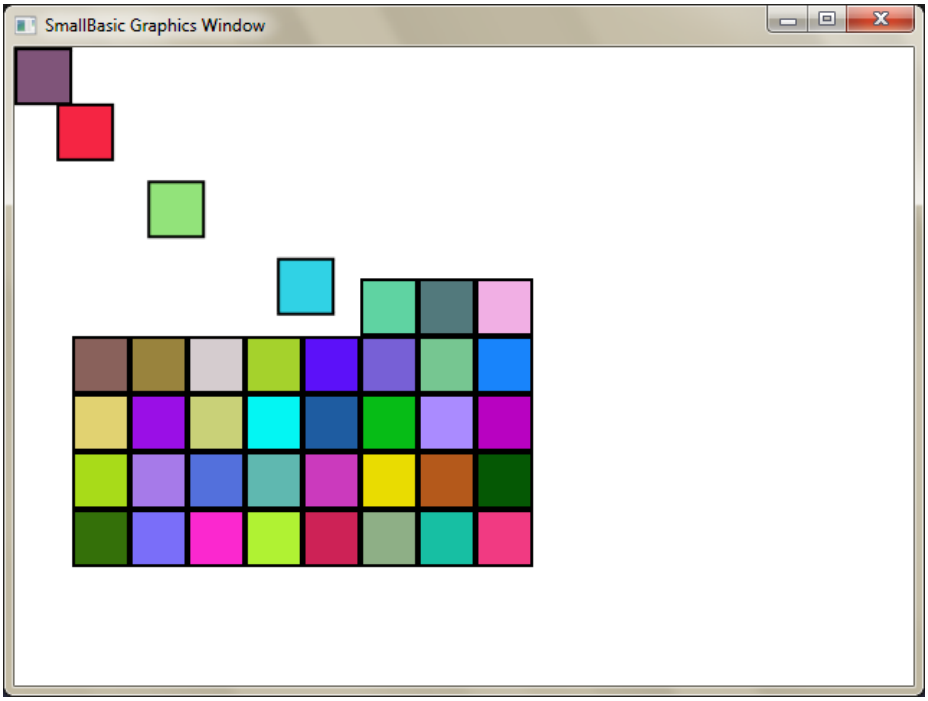

그림 54 -그리드의 사각형 추적하기

제 11 장

이벤트와 대화형 작업

첫 두 장에 걸쳐 속성과 연산을 갖는 개체에 대해 소개했습니다. 속성과 연산 이외에, 어떤 개체는 ***이벤트***라는 것을 갖습니다. 이벤트는 사용자 동작에 대한 응답으로 발생하는 일종의 신호와 같습니다. 어떤 면에서 이벤트는 연산과 상반되는 개념입니다. 연산이란 프로그래머가 컴퓨터에게 어떤 일을 하도록 만드는 것인 반면, 이벤트의 경우는 컴퓨터가 프로그래머에게 흥미로운 어떤 일이 발생했음을 알려주는 것이라 하겠습니다.

이벤트가 어떻게 유용하게 사용될까요?

프로그램에서 대화형 작업을 소개하는 데 있어 이벤트는 아주 중요합니다. 사용자가 프로그램과 대화형 작업을 할 수 있도록 하려면, 이 이벤트를 사용합니다. Tic-Tac-Toe 게임을 만든다고 생각해 봅시다. 사용자가 게임을 선택할 수 있도록 해야겠죠? 여기에 이벤트가 사용됩니다. 이벤트를 통해 프로그램 내에서 사용자의 입력을 받습니다. 이 것이 이해가 되지 않는다 해도 걱정하지 마십시오. 이벤트가 무엇이며 어떻게 사용되는지 이해를 돕는 매우 간단한 예를 살펴보겠습니다.

다음은 단 한 문장과 단 한 개의 서브루틴을 갖는 매우 간단한 프로그램입니다. 서브루틴은 GraphicsWindow 개체 상에 *ShowMessage* 연산을 사용하여 사용자에게 메시지 상자를 보여줍니다.

```
GraphicsWindow.MouseDown = OnMouseDown
Sub OnMouseDown
```

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

위 프로그램에서 흥미로운 부분은 GraphicsWindow 개체의 **MouseDown** 이벤트에 서브루틴 이름을 대입하는 줄입니다. 특정 값을 대입하는 대신 서브루틴 *OnMouseDown* 을 대입한다는 점에서 MouseDown 은 속성과 매우 비슷하다는 것을 눈치 채셨을 겁니다. 이것이 이벤트의 특별한 점입니다. 이벤트가 발생하면, 자동적으로 서브루틴이 호출됩니다. 이 경우는 사용자가 GraphicsWindow 상에서 마우스를 통해 클릭을 할 때마다, 서브루틴 *OnMouseDown* 이 호출됩니다. 계속해서 프로그램을 실행하고 이를 시도해 보십시오. GraphicsWindow 상에서 마우스로 클릭할 때마다 아래 그림과 같은 메시지 상자를 보게 될 것입니다.

그림 55 – 이벤트에 대한 응답

이러한 이벤트 처리는 매우 강력하며 개발자가 매우 창의적이고 재미있는 프로그램을 만들도록 도와 줍니다. 이런 방식으로 쓰여진 프로그램을 종종 이벤트-구동 프로그램이라고 부릅니다.

OnMouseDown 서브루틴을 변경하여 메시지 상자를 뜨게 하는 것 보다 다른 것들을 수행할 수 있습니다. 예를 들어, 다음 프로그램과 같이 사용자가 마우스를 클릭한 위치에 큰 파란 점을 그릴수도 있습니다.

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

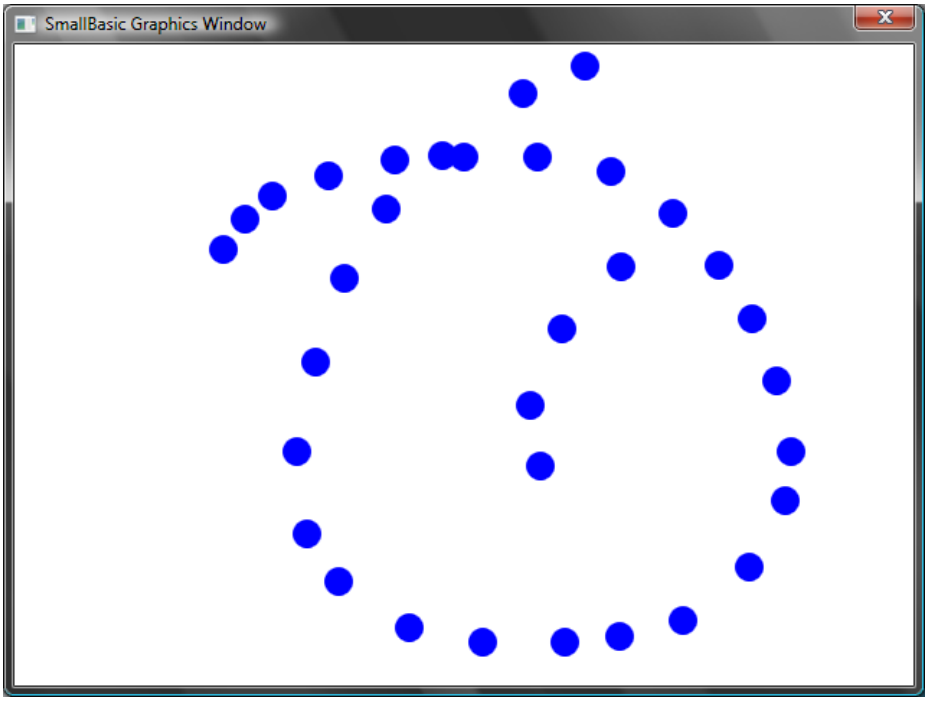

그림 56 – 마우스를 눌렀을 때 발생하는 이벤트 처리

위 프로그램에서 마우스 현재 좌표를 구하기 위해 *MouseX*와 *MouseY*를 사용했음을 주목하십시오. 그 다음 이를 사용하여 마우스 좌표를 원의 중심으로 하는 원을 그립니다.

여러 이벤트 처리

처리할 이벤트 개수에는 특별한 제한은 없습니다. 여러 이벤트 처리에 단 한 개의 서브루틴을 사용할 수도 있습니다. 그렇지만 한번에 하나의 이벤트를 처리해야 합니다. 같은 이벤트에 두 개의 서브루틴을 대입한다면, 나중 것이 이깁니다.

여러 이벤트 처리 설명을 위해, 이전 예제에 키 누름을 처리하는 서브루틴 하나를 추가하겠습니다. 이 추가되는 서브루틴은 브러시의 색깔을 바꾸도록 만들어, 마우스가 클릭되면 다른 색의 점을 찍도록 합니다.

```
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
```

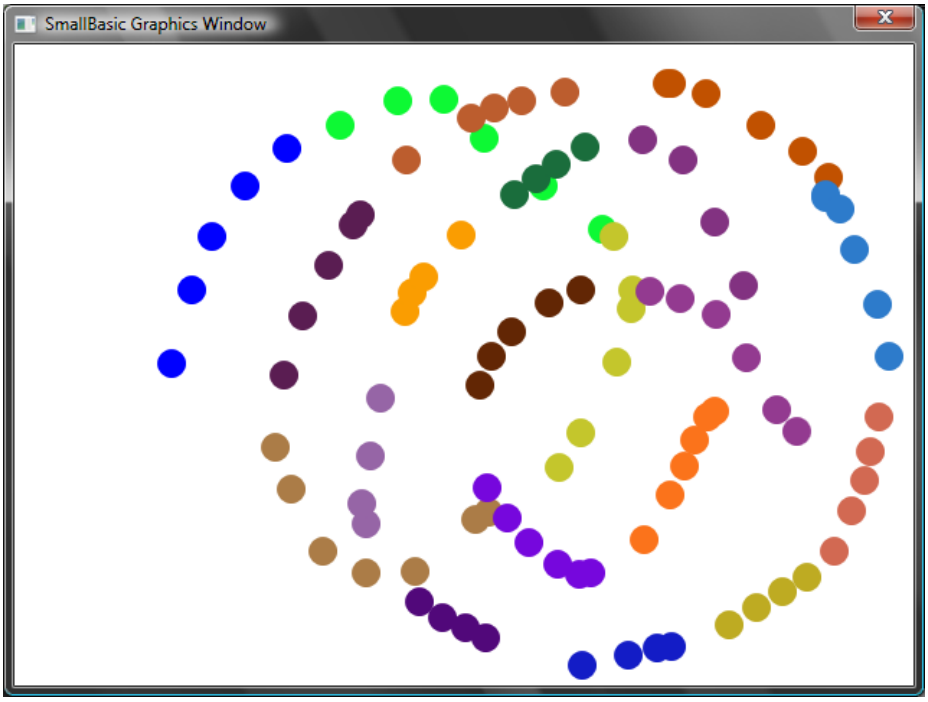

그림 57 – 여러 이벤트 처리

이 프로그램을 실행하여 창을 클릭하면, 파란 점이 나타납니다. 이제 특정 키를 입력하고 마우스를 다시 클릭하면, 다른 색깔의 점이 나타날 것입니다. 키를 눌렀을 때 브러시의 색을 무작위로 바꾸는 *OnKeyDown* 이 실행되는 것입니다. 그 후 마우스를 클릭하면, 이 새롭게 설정된 색(무작위 색의 점)으로 원이 그려지는 것입니다.

그리기 프로그램

이벤트 및 서브루틴으로 무장하여, 우리는 창에 그림을 그리는 프로그램을 만들 수 있습니다. 문제를 작은 부분으로 나누기만 한다면, 이러한 프로그램을 만드는 것은 놀랍게도 매우 쉽습니다. 첫 단계로, 그래픽 창 위의 어디로든 마우스를 이동하여 이 지나간 흔적을 남기는 프로그램을 만듭니다.

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

그러나 이 프로그램을 실행하면, 첫 줄은 항상 윈도우의 왼쪽 상단 가장자리(0, 0)에서 시작됩니다. *MouseDown* 이벤트를 처리하고 이벤트가 들어왔을 때 *prevX* 와 *prevY* 값을 캡처함으로써 이 문제를 해결할 수 있습니다.

또한, 사용자가 마우스를 아래로 눌렀을 때만 흔적을 남기고자 합니다. 다른 상황에서는 선을 그리고 싶지 않습니다. 이 동작을 구현하려면, **Mouse** 개체의 *IsLeftButtonDown* 속성을 이용할 것입니다. 이 속성은 왼쪽 마우스 단추가 눌려진 상태로 유지되고 있는가를 알려줍니다. 이 값이 true 면 선을 그릴 것이고, 아닌 경우에는 선을 그리지 않습니다.

```
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
```

부록 A

재미있는 샘플

거북 프랙탈

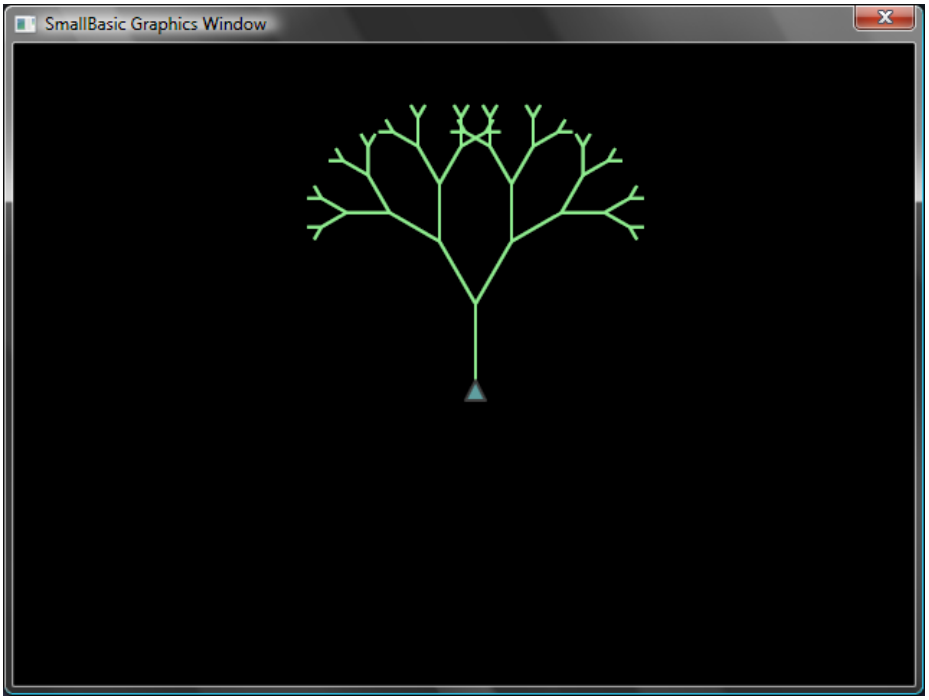

그림 58 – 나무 프랙탈을 그리는 거북이

```
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
```

Flickr 사진

그림 59 - Flickr 에서 가져온 사진

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

동적인 데스크톱 배경화면

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

막대 게임

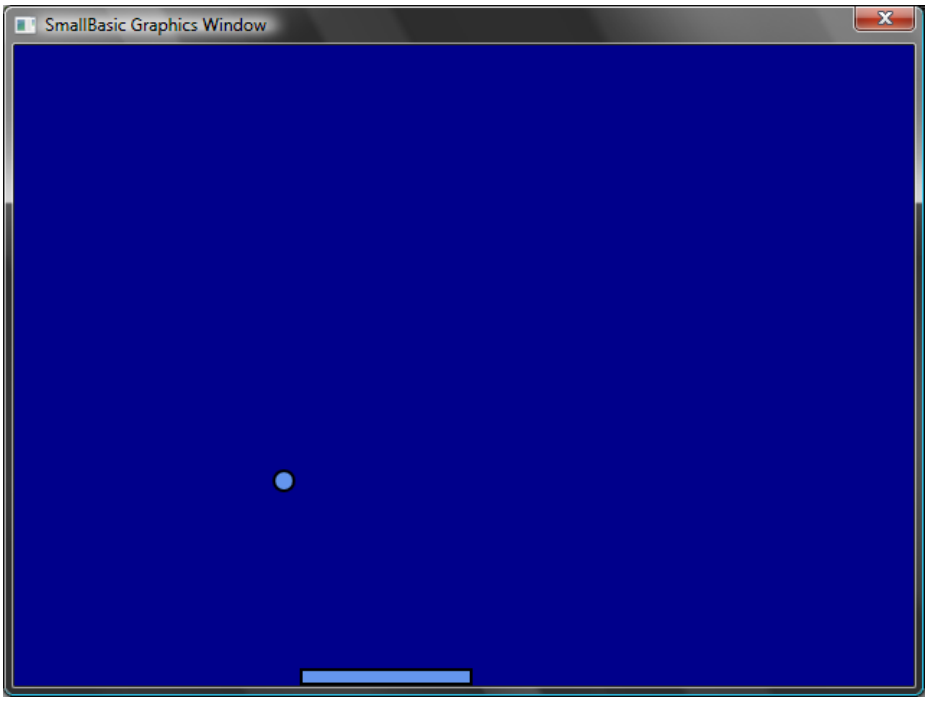

그림 60 - 막대 게임

```
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
```

부록 B

색

다음은 기본 색조에 의해 분류된 Small Basic 이 지원하는 색들의 이름 목록입니다.

빨간색

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

분홍색

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

주황색

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

노란색

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

LightGoldenrodYellow	#FAFAD2
PapayaWhip	#FFEFD5
Moccasin	#FFE4B5
PeachPuff	#FFDAB9
PaleGoldenrod	#EEE8AA
Khaki	#F0E68C
DarkKhaki	#BDB76B

보라색

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

초록색

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

파란색

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

갈색

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

흰색

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

회색

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

SlateGray	#708090
DarkSlateGray	#2F4F4F
Black	#000000