A. 문자열 다루기

코딩 할 때 숫자만큼 많이 다루게 되는 데이터 형식이 바로 문자열입니다. string 형식은 그저 문자열을 담는 역할을 할 뿐 아니라, 문자열을 가공하기 위한 다양한 기능도 함께 제공합니다. 머리도 식힐 겸 저와 같이 이 기능들을 살펴 보시죠.

문자열 안에서 찾기

요리를 시작할 때 가장 먼저 하는 일이 재료를 씻고 다듬는 일이지요? 문자열을 다룰 때도 비슷한 준비가 필요합니다. 대부분의 문자열 가공 메소드는 문자열 내의 "어느 부분"을 가공할 것인지를 입력 받는데요, 친절하게도 string 형식은 바로 이 "어느 부분"을 찾아주는 기능을 다양하게 제공합니다. 다음 표는 string 형식이 제공하는 탐색 메소드의 종류와 역할을 설명합니다.

메소드	설명
IndexOf()	현재 문자열 내에서 찾고자 하는 지정된 문자 또는 문자열의 위치를 찾습니다.
LastIndexOf()	현재 문자열 내에서 찾고자 하는 지정된 문자 또는 문자열의 위치를 뒤에서부터 찾습니다.
StartsWith()	현재 문자열이 지정된 문자열로 시작하는지를 평가합니다.
EndsWith()	현재 문자열이 지정된 문자열로 끝나는지를 평가합니다.
Contains()	현재 문자열이 지정된 문자열을 포함하는지를 평가합니다.
Replace()	현재 문자열에서 지정된 문자열이 다른 지정된 문자열로 모두 바뀐 새 문자열을 반환합니다.

문자열 탐색 예제 프로그램을 만들어보겠습니다. 다음 코드는 위 표에서 언급한 메소드 모두를 이용하는 예제를 담고 있습니다.

Appendix A/StringSearch/MainApp.cs

```
01  using static System.Console;
02
03  namespace StringSearch
04  {
05      class MainApp
06      {
07          static void Main(string[] args)
08          {
09              string greeting = "Good Morning";
10
11              WriteLine(greeting);
```

```
12              WriteLine();
13
14              // IndexOf()
15              WriteLine("IndexOf 'Good' : {0}", greeting.IndexOf("Good"));
16              WriteLine("IndexOf 'o' : {0}", greeting.IndexOf('o'));
17
18              // LastIndexOf()
19              WriteLine("LastIndexOf 'Good' : {0}", greeting.LastIndexOf("Good"));
20              WriteLine("LastIndexOf 'o' : {0}", greeting.LastIndexOf("o"));
21
22              // StartsWith()
23              WriteLine("StartsWith 'Good' : {0}", greeting.StartsWith("Good"));
24              WriteLine("StartsWith 'Morning' : {0}", greeting.StartsWith("Morning"));
25
26              // EndsWith()
27              WriteLine("EndsWith 'Good' : {0}", greeting.EndsWith("Good"));
28              WriteLine("EndsWith 'Morning' : {0}", greeting.EndsWith("Morning"));
29
30              // Contains()
31              WriteLine("Contains 'Evening' : {0}", greeting.Contains("Evening"));
32              WriteLine("Contains 'Morning' : {0}", greeting.Contains("Morning"));
33
34              // Replace()
35              WriteLine("Replaced 'Morning' with 'Evening': {0}",
36                  greeting.Replace("Morning", "Evening"));
37          }
38      }
39  }
```

실행 결과:

```
>StringSearch.exe
Good Morning

IndexOf 'Good' : 0
IndexOf 'o' : 1
LastIndexOf 'Good' : 0
LastIndexOf 'o' : 6
StartsWith 'Good' : True
StartsWith 'Morning' : False
EndsWith 'Good' : False
EndsWith 'Morning' : True
Contains 'Evening' : False
Contains 'Morning' : True
Replaced 'Morning' with 'Evening': Good Evening
```

문자열 변형하기

string 형식은 문자열 중간에 또 다른 문자열을 삽입하거나 특정 부분을 삭제하는 등의 작업을 수행하는 메소드도 제공합니다. 이와 더불어 대문자/소문자로의 변환 메소드와 문자열 앞/뒤에 있는 공백을 제거하는 메소드도 제공하는데요, 공백 제거 메소드는 생각보다 자주 사용되니 관련 메소

드의 이름을 눈여겨봐주세요. 다음 표는 string 형식의 문자열 변형 기능 관련 메소드를 설명하고 있습니다.

메소드	설명
ToLower()	현재 문자열의 모든 대문자를 소문자로 바꾼 새 문자열을 반환합니다.
ToUpper()	현재 문자열의 모든 소문자를 대문자로 바꾼 새 문자열을 반환합니다.
Insert()	현재 문자열의 지정된 위치에 지정된 문자열이 삽입된 새 문자열을 반환합니다.
Remove()	현재 문자열의 지정된 위치로부터 지정된 수만큼의 문자가 삭제된 새 문자열을 반환합니다.
Trim()	현재 문자열의 앞/뒤에 있는 공백을 삭제한 새 문자열을 반환합니다.
TrimStart()	현재 문자열의 앞에 있는 공백을 삭제한 새 문자열을 반환합니다.
TrimEnd()	현재 문자열의 뒤에 있는 공백을 삭제한 새 문자열을 반환합니다.

바로 예제 프로그램을 하나 만들어보겠습니다. 다음 코드는 위 표에서 언급한 메소드를 활용하는 예제를 담고 있습니다.

Appendix A/StringModify/MainApp.cs

```
01  using static System.Console;
02
03  namespace StringModify
04  {
05      class MainApp
06      {
07          static void Main(string[] args)
08          {
09              WriteLine("Lower() : '{0}'", "ABC".ToLower());
10              WriteLine("ToUpper() : '{0}'", "abc".ToUpper());
11
12              WriteLine("Insert() : '{0}'", "Happy Friday!".Insert(5, " Sunny"));
13              WriteLine("Remove() : '{0}'", "I Don't Love You.".Remove(2, 6));
14
15              WriteLine("Trim() : '{0}'", " No Spaces ".Trim());
16              WriteLine("TrimStart() : '{0}'", " No Spaces ".TrimStart());
17              WriteLine("TrimEnd() : '{0}'", " No Spaces ".TrimEnd());
18          }
19      }
```

```
20  }
```

실행 결과:

```
    Lower() : 'abc'
    ToUpper() : 'ABC'
    Insert() : 'Happy Sunny Friday!'
    Remove() : 'I Love You.'
    Trim() : 'No Spaces'
    TrimStart() : 'No Spaces '
    TrimEnd() : ' No Spaces'
```

문자열 분할하기

"MSFT, GOOG, AMZN, AAPL, RHT"

string 형식은 위와 같이 콤마(,)로 구분되어 있는 문자열에서 단번에 콤마를 제외한 내용을 단번에 배열로 만들어줄 수 있습니다. 이것을 가능하게 하는 것은 Split() 메소드입니다. 그런데 문자열 데이터가 항상 저렇게 일정한 간격 또는 문자로 구분된 예쁜 모습을 하고 있지는 않지요. 그럴 때는 이 장에서 가장 먼저 다뤘던 문자열 탐색을 이용해서 잘라낼 부분을 찾은 뒤, SubString() 메소드를 이용해서 하나씩 잘라내면 됩니다. 다음 표는 문자열 분할 기능을 수행하는 Split() 메소드와 SubString() 메소드를 설명합니다.

메소드	설명
Split()	현재 문자열을 지정된 문자를 기준으로 분리한 문자열의 배열을 반환합니다.
SubString()	현재 문자열의 지정된 위치로부터 지정된 수만큼의 문자로 이루어진 새 문자열을 반환합니다.

예제를 만들어봐야겠죠? 다음 코드는 Split() 메소드와 SubString() 메소드를 활용하는 예제입니다.

Appendix A/StringSlice/MainApp.cs

```
01  using System;
02  using static System.Console;
03
04  namespace StringSlice
05  {
06      class MainApp
07      {
08          static void Main(string[] args)
09          {
10              string greeting = "Good morning.";
11
```

```
12              WriteLine(greeting.Substring(0, 5)); // "morning"
13              WriteLine(greeting.Substring(5)); // "Good"
14              WriteLine();
15
16              string[] arr = greeting.Split(
17                  new string[] {" "}, StringSplitOptions.None);
18              WriteLine("Word Count : {0}", arr.Length);
19
20              foreach (string element in arr)
21                  WriteLine("{0}", element);
22          }
23      }
24  }
```

실행 결과:

```
Lower() : 'abc'
ToUpper() : 'ABC'
Insert() : 'Happy Sunny Friday!'
Remove() : 'I Love You.'
Trim() : 'No Spaces'
TrimStart() : 'No Spaces '
TrimEnd() : ' No Spaces'
```

문자열 서식 맞추기

많은 독자 분들이 "서식"이라는 낱말을 두고 글꼴, 색상 같은 모양새를 떠올렸을 겁니다. 여기에서 말하는 서식은 조금 다른 의미인데요, 문자열이 일정한 틀과 모양을 갖추는 것을 의미합니다. 에를 들면 다음과 같이 말입니다.

```
제품명 : 망고주스
가격 : 1,500원
용량 : 250ml
유통기한 : 2034-10-20 12:11:11
```

```
제품명 : 자몽주스
가격 : 1,700원
용량 : 250ml
유통기한 : 2031-08-03 17:32:47
```

C#은 문자열 서식화에 사용할 수 있는 간편한 방법 두 가지를 제공합니다. 첫 번째는 string 형식의 Format() 메소드이고, 또 다른 하나는 문자열 보간(Interpolation)입니다. 두 방법 모두 문자열 틀을 두고, 그 틀을 이용해서 서식화된 새로운 문자열을 만들어낸다는 공통점이 있습니다. 우리는 먼저 Format() 메소드부터 살펴보겠습니다.

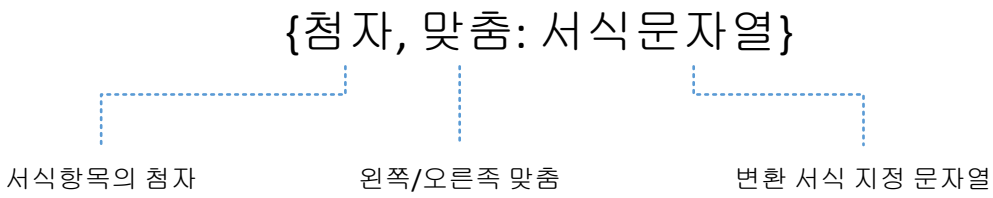
Format() 메소드

Format() 메소드를 사용하는 방법의 절반 정도는 이미 여러분도 알고 있습니다. 어째서냐고요? 우리가 그동안 열심히 사용해온 Console.WriteLine() 메소드가 서식화된 문자열을 출력할 때 내부에서 string.Format() 메소드를 사용하고 있는데, string.Format() 메소드의 사용 방법이 Console.WriteLine() 메소드의 사용 방법과 똑같기 때문입니다.

```
Console.WriteLine("제목 : {0}", "뇌를 자극하는 C# 프로그래밍");
```

Console.WriteLine() 메소드를 사용하는 코드는 이제 제법 친숙하지요? Console.WriteLine()을 이용해서 서식화된 문자열을 출력할 때는 첫 번째 매개변수로 "문자열 틀" 을 입력하고, 두 번째 매개변수부터는 문자열 틀 안에 집어넣을 데이터를 차례대로 입력합니다. 당연히 string.Format() 메소드의 사용 방법도 이와 동일합니다. 저는 지금부터 조금 더 세련된 서식화된 문자열을 만드는 방법을 설명하려 합니다. 물론 이 방법은 Console.WriteLine() 메소드를 호출할 때도 똑같이 적용할 수 있습니다.

문자열 틀에 입력하는 {0}, {1} .. 를 일컬어 "서식 항목(Format Item)"이라고 하는데요, 이 서식 항목에 추가 옵션을 입력함으로써 더 다양한 서식화를 수행할 수 있습니다. 서식 항목의 옵션은 다음과 같이 구성되어 있습니다.



우리는 지금까지 서식항목의 첨자만 이용해 온 셈입니다. 첨자와 더불어 맞춤과 서식문자열을 사용하는 예는 다음과 같습니다.

```
Console.WriteLine("Total : {0, -7: D}", 123); // 첨자:0, 맞춤:-7, 서식문자열:D
```

서식 항목의 옵션들을 조합하면 매우 다양한 서식화를 구현할 수 있습니다. 이 서식항목에 대해 조금 더 자세히 설명 드리겠습니다. 이어지는 섹션에서는 다음과 같이 string.Format()을 이용한 세 가지 서식화에 대해 설명합니다.

- 왼쪽/오른쪽 맞춤
- 숫자 서식화
- 날짜 및 시간 서식화

왼쪽/오른쪽 맞춤

서식항목의 맞춤 옵션을 지정하면 해당 서식항목이 차지할 공간의 크기를 선택할 수 있고, 그 공간 안에서 왼쪽 또는 오른쪽에 데이터를 할당할지를 결정할 수 있습니다. 이해를 돕기 위해 예를 들어보겠습니다.

다음의 코드에서 result는 "ABCDEF"가 됩니다. 우리가 익히 알고 있는 부분입니다.

```
string result = string.Format("{0}DEF", "ABC");
// result : "ABCDEF"
```

이 코드를 조금 수정해볼게요. 다음과 같이 서식 문자열의 {0}에 -10이라는 맞춤 옵션을 입력해주면 result는 이제 "ABC DEF"가 됩니다. "DEF" 앞에 문자 10개가 들어갈 공간을 만들어 두고 앞(왼쪽)에서부터 "ABC"를 채워 넣는 겁니다.

```
string result = string.Format("{0,-10}DEF", "ABC");
// result : "ABC       DEF"
```

다음 그림은 string.Format("{0,-10}DEF", "ABC")의 결과가 어떻게 완성되었는지를 보여줍니다.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
A	B	C								D	E	F

이번에는 뒤(오른쪽)부터 채워서 서식화를 해보겠습니다. 10칸의 공간을 만들고 왼쪽부터 채울 때는 -10을 입력하지만, 오른쪽부터 채울 때는 –기호 없이 10만 입력하면 됩니다.

```
string result = string.Format("{0, 10}DEF", "ABC");
// result : "       ABCDEF"
```

다음 그림은 string.Format("{0, 10}DEF", "ABC")의 결과가 어떻게 완성되었는지를 보여줍니다.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
							A	B	C	D	E	F

문자열 서식을 왼쪽/오른쪽으로 맞추는 기능은 프로그램이 여러 개의 항목을 가지런하게 출력할 필요가 있을 때 특히 유용합니다. 예제 프로그램을 하나 만들어 보겠습니다.

Appendix A/StringFormatBasic/MainApp.cs

```
01  using System;
02  using static System.Console;
03
04  namespace StringFormatBasic
05  {
06      class MainApp
07      {
08          static void Main(string[] args)
09          {
10              string fmt = "{0,-20}{1,-15}{2, 30}";
11
12              WriteLine(fmt, "Publisher", "Author", "Title");
13              WriteLine(fmt, "Marvel", "Stan Lee", "Iron Man");
14              WriteLine(fmt, "Hanbit", "Sanghyun Park", "C# 7.0 Programming");
15              WriteLine(fmt, "Prentice Hall", "K&R", "The C Programming Language");
16          }
17      }
18  }
```

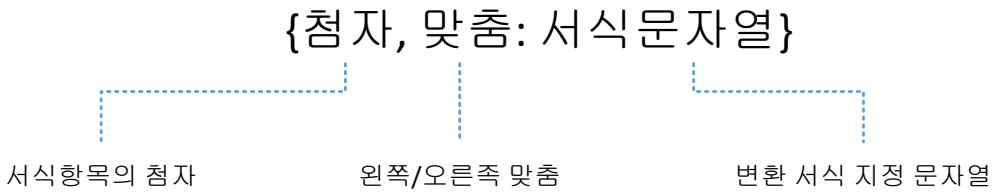
실행 결과:

```
Publisher           Author                              Title
Marvel              Stan Lee                          Iron Man
Hanbit              Sanghyun Park            C# 7.0 Programming
Prentice Hall       K&R                The C Programming Language
```

숫자 서식화

수 하나도 표현하는 방식은 매우 다양할 수 있습니다. 123456789라는 수가 있다고 해보죠. 이 수는 123,456,789 로 나타낼 수도 있고 0x75BCD15(16진수)로도 나타낼 수 있습니다. 1.234568E+002(지수 표현)로도 나타낼 수 있고요. string.Format() 메소드는 이렇듯 다양한 형태로 수를 서식화하는 기능을 지원합니다.

서식 항목의 그림을 다시 보시죠. 숫자 서식화는 이 중 서식문자열에 의해 결정됩니다.



아래의 표는 서식문자열에 사용할 수 있는 서식지정자(Format Specifier)를 보여줍니다. 더 많은 서식지정자에 대해서는 MSDN(https://msdn.microsoft.com/ko-kr/dwhawy9k)을 참고하세요.

서식 지정자	대상 서식	설명

D	10진수	입력된 수를 10진수로 서식화합니다.
		
		예)
		`WriteLine("{0:D}", 255);  // 255`
		`WriteLine("{0:D}", 0xFF); // 255`

X	16진수	입력된 수를 16진수로 서식화합니다.
		
		예)
		`WriteLine("{0:X}", 255);  // 0xFF`
		`WriteLine("{0:X}", 0xFF); // 0xFF`

N	콤마(,)로 묶어 표현한 수	입력된 수를 콤마로 구분하여 출력합니다. 예) `WriteLine("{0:N}", 123456789);  // 123,456,789.00`
F	고정소수점	입력된 수를 고정소수점 형식으로 서식화합니다. 예) `WriteLine("고정소수점: {0:F}", 123.45); // 123.45`
E	지수	입력된 수를 지수 표기로 서식화합니다. 예) `WriteLine("공학: {0:E}", 123.456789); // 1.234568E+002`

서식문자열은 위 표에서 언급한 서식지정자와 함께 "자릿수 지정자(Precision Specifier)"를 사용할 수 있습니다. 서식문자열이 Axx라면 A는 서식지정자, xx는 자릿수 지정자의 꼴로 사용하면 됩니다. 이 때 자릿수 지정자는 0에서 99까지의 값을 입력할 수 있습니다. 예를 들어 서식지정자 D와 자릿수 지정자 5로 서식문자열 D5를 입력했을 때, string.Format() 메서드는 123을 00123으로 서식화합니다.

이렇게 글로만 이야기할 게 아닌 것 같습니다. 예제 프로그램을 만들어보죠.

Appendix A/StringFormatNumber/MainApp.cs

```
01  using System;
02  using static System.Console;
03
04  namespace StringFormatNumber
05  {
06      class MainApp
07      {
08          static void Main(string[] args)
09          {
```

`WriteLine(string.Format("10진수: {0:D}", 123));`
과 동일

```
10              // D : 10진수
11              WriteLine("10진수: {0:D}", 123);
12              WriteLine("10진수: {0:D5}", 123);
13
14              // X : 16진수
15              WriteLine("16진수: 0x{0:X}", 0xFF1234);
16              WriteLine("16진수: 0x{0:X8}", 0xFF1234);
17
18              // N : 숫자
19              WriteLine("숫자: {0:N}", 123456789);
20              WriteLine("숫자: {0:N0}", 123456789); // 자릿수 0은 소숫점 이하를 제거함.
21
22              // F : 고정소수점
23              WriteLine("고정소수점: {0:F}", 123.45);
24              WriteLine("고정소수점: {0:F5}", 123.456);
25
26              // E : 공학용
27              WriteLine("공학: {0:E}", 123.456789);
28          }
29      }
30  }
```

실행 결과:

```
10진수: 123
10진수: 00123
16진수: 0xFF1234
16진수: 0x00FF1234
숫자: 123,456,789.00
숫자: 123,456,789
고정소수점: 123.45
고정소수점: 123.45600
공학: 1.234568E+002
```

날짜 및 시간 서식화

이번엔 날짜와 시간을 서식화하는 방법을 알아보겠습니다. 날짜와 시간을 표현하기 위해서는 DateTime 클래스가 필요합니다. 다음 예제 코드를 보고 이야기를 계속 하겠습니다.

```
DateTime dt = new DateTime(2018, 11, 3, 23, 18, 22); // 2018년 11월 3일 23시 18분 22초
WriteLine("{0}", dt); // 국가 및 지역 설정에 따라 다른 결과 출력
```

위 코드는 한글 윈도우를 이용하고 있는 독자의 컴퓨터에서는 "2018-11-03 오후 11:18:22"를 출력합니다. 제어판에서 [국가 및 지역] 설정을 변경하지 않았다면 말입니다. [국가 및 지역] 설정에서 영어(미국) 형식으로 변경하면 위 코드는 "11/3/2018 11:18:22 PM"를 출력합니다. 제어판을 이용하지 않고도 System.Globalization.CultureInfo 클래스의 도움을 받으면 C# 코드만으로 날짜 및 시간 서식을 통제할 수 있습니다.

다음 표에는 날짜 및 시간을 서식화하는데 사용할 수 있는 서식지정자가 정리되어 있습니다. 더 많은 날짜 및 시간 서식 지정자는 MSDN(https://msdn.microsoft.com/ko-kr/8kb3ddd4)을 참고하세

요.

서식 지정자	대상 서식	설명
y	연도	● yy : 두 자릿수 연도(2018-11-03 23:18:22 ➔ 18) ● yyyy : 네 자릿수 연도(2018-11-03 23:18:22 ➔ 2018)
M	월	● M : 한 자릿수 월(2011-09-08 21:03:07 ➔ 1) ● MM : 두 자릿수 월 (2011-09-08 21:03:07 ➔ 01)
d	일	● d : 한 자릿수 일(2011-09-08 21:03:07 ➔ 8) ● MM : 두 자릿수 일 (2011-09-08 21:03:07 ➔ 08)
h	시(1~12)	● h : 한 자릿수 시(2011-09-08 21:03:07 ➔ 9) ● hh : 두 자릿수 시 (2011-09-08 21:03:07 ➔ 09)
H	시(1~23)	● H : 한 자릿수 시(2011-09-08 21:03:07 ➔ 21) ● HH : 두 자릿수 시(2011-09-08 21:03:07 ➔ 21)
m	분	● m : 한 자릿수 분(2011-09-08 21:03:07 ➔ 3) ● mm : 두 자릿수 분(2011-09-08 21:03:07 ➔ 03)
s	초	● s : 한 자릿수 초(2011-09-08 21:03:07 ➔ 7) ● ss : 두 자릿수 초(2011-09-08 21:03:07 ➔ 07)
tt	오전/오후	● tt : 오전/오후(2011-09-08 11:03:07 ➔ 오전, 2011-09-08 21:03:07 ➔ 오후)
ddd	요일	● ddd : 약식 요일(2018-11-03 23:18:22 ➔ 토) ● dddd : 전체 요일(2018-11-03 23:18:22 ➔ 토요일)

날짜 및 시간을 위 표에서 설명한 서식지정자를 이용하여 서식화하는 예는 다음과 같습니다.

```
DateTime dt = new DateTime(2018, 11, 3, 23, 18, 22);

// 12시간 형식: 2018-11-03 오후 11:18:22 (토)
WriteLine("12시간 형식: {0:yyyy-MM-dd tt hh:mm:ss (ddd)}", dt);

// 24시간 형식: 2018-11-03 23:18:22 (토요일)
WriteLine("24시간 형식: {0:yyyy-MM-dd HH:mm:ss (dddd)}", dt);
```

예제 프로그램을 만들어보기 전에 하나 더 이야기할 것이 있습니다. 우리는 한글을 사용하니까 "토" 또는 "토요일"이 자연스럽지만 다른 언어를 사용하는 사람들은 입장이 다릅니다. 영어권 사람에게는 "Sat"이나 "Saturday"라고 보여줘야지요. 한글 사용자에게는 "오전", "오후"를 보여주고 영어권 사용자에게는 "AM", "PM"을 보여줘야 합니다. 이 기능은 DateTime 객체가 제공하는 각 필드의 값을 이용해서 프로그래머가 직접 해당 문자열을 반환하는 코드를 구현할 수도 있지만, 앞서 언급했던 CultureInfo 클래스의 도움을 받으면 더욱 쉽고 완벽하게 해결할 수 있습니다. CultureInfo 클래스의 역할은 문화권 정보를 나타내는 건데요, DateTime.ToString() 메소드에 서식 문자열과 함께 이 클래스의 인스턴스를 입력하면 해당 문화권에 맞는 요일 이름을 얻을 수 있습

니다. 사용 예는 다음과 같습니다.

```
CultureInfo ciKo = new CultureInfo("ko-KR");

// 2018-11-03 오후 11:18:22 (토)
WriteLine(dt.ToString("yyyy-MM-dd tt hh:mm:ss (ddd)", ciKo));

CultureInfo ciEn = new CultureInfo("en-US");

// 2018-11-03 PM 11:18:22 (Sat)
WriteLine(dt.ToString("yyyy-MM-dd tt hh:mm:ss (ddd)", ciEn));
```

CultureInfo의 생성자에 사용되는 문화권 이름은 MSDN(https://msdn.microsoft.com/ko-kr//bb896001.aspx)을 참조하시기 바랍니다.

이제 날짜와 시간을 서식화하는 예제 프로그램을 만들어 보겠습니다.

Appendix A/StringFormatDatetime/MainApp.cs

```
01  using System;
02  using System.Globalization;
03  using static System.Console;
04
05  namespace StringFormatDatetime
06  {
07      class MainApp
08      {
09          static void Main(string[] args)
10          {
11              DateTime dt = new DateTime(2018, 11, 3, 23, 18, 22);
12
13              WriteLine("12시간 형식: {0:yyyy-MM-dd tt hh:mm:ss (ddd)}", dt);
14              WriteLine("24시간 형식: {0:yyyy-MM-dd HH:mm:ss (dddd)}", dt);
15
16               CultureInfo ciKo = new CultureInfo("ko-KR");
17              WriteLine();
18              WriteLine(dt.ToString("yyyy-MM-dd tt hh:mm:ss (ddd)", ciKo));
19              WriteLine(dt.ToString("yyyy-MM-dd HH:mm:ss (dddd)", ciKo));
20              WriteLine(dt.ToString(ciKo));
21
22              CultureInfo ciEn = new CultureInfo("en-US");
23              WriteLine();
24              WriteLine(dt.ToString("yyyy-MM-dd tt hh:mm:ss (ddd)", ciEn));
25              WriteLine(dt.ToString("yyyy-MM-dd HH:mm:ss (dddd)", ciEn));
26              WriteLine(dt.ToString(ciEn));
27          }
28      }
29  }
```

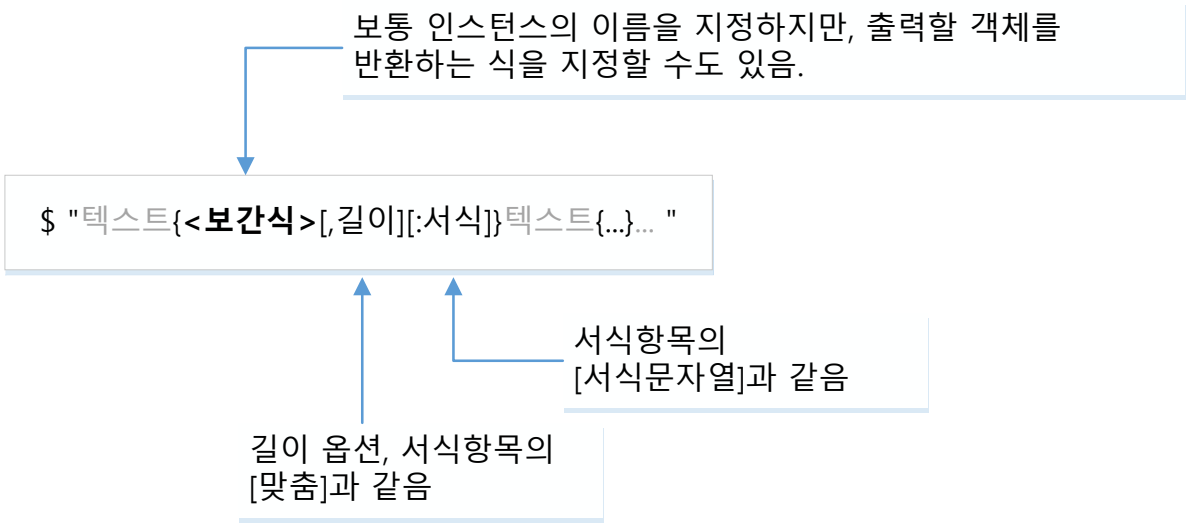
실행 결과:

```
12시간 형식: 2018-11-03 오후 11:18:22 (토)
24시간 형식: 2018-11-03 23:18:22 (토요일)

2018-11-03 오후 11:18:22 (토)
2018-11-03 23:18:22 (토요일)
2018-11-03 오후 11:18:22

2018-11-03 PM 11:18:22 (Sat)
2018-11-03 23:18:22 (Saturday)
11/3/2018 11:18:22 PM
```

문자열 보간

Format() 메소드와 더불어 서식화된 문자열을 편리하게 다룰 수 있는 또 다른 방법으로는 문자열 보간(Interpolation)이 있습니다. 이 기능은 C# 6.0에서 새로 도입된 기능으로, 한결 더 편리하게 문자열의 양식을 맞출 수 있도록 프로그래머를 도와줍니다. 보간(補間)이라는 낱말은 비거나 누락된 부분을 채운다는 뜻입니다.

문자열 보간이 string.Format() 메소드와 다른 점은 문자열 틀 앞에 $ 기호를 붙인다는 것과 서식 항목에 첨자 대신 식이 들어간다는 것 정도입니다. 이 식에는 변수나 객체의 이름을 그대로 넣어 사용할 수도 있고, 상수를 입력해 쓸 수도 있으며, 조건에 따라 다른 값을 출력하는 코드가 들어갈 수도 있습니다. 다음 그림은 문자열 보간에서 사용되는 문자열 틀의 구조를 보여줍니다.



다음 표에는 문자열 보간 예제 코드 몇 가지가 담겨있습니다. 이해를 돕기 위해 같은 결과를 출력하는 string.Format() 메소드 예제도 준비했으니 비교하며 살펴보세요.

string.Format()	문자열 보간
WriteLine("{0}, {1}", 123, "최강한화");	WriteLine($"{123}, {"최강한화"}");
WriteLine("{0,-10,D5", 123);	WriteLine($"{123,-10,D5}");

int n = 123; string s = "최강한화"; WriteLine("{0}, {1}", n, s);	int n = 123; string s = "최강한화"; WriteLine($"{n}, {s}");
int n = 123; WriteLine("{0}", n>100?"큼":"작음", s);	int n = 123; WriteLine($"{(n > 100 ? "큼":"작음")}");

여러분은 문자열을 서식화하는 두 방식 중 어떤 쪽이 마음에 드는지요? string.Format()과 문자열 보간을 비교해보면 문자열 보간이 보다 읽기도 좋고 코드의 양도 적은 것을 알 수 있습니다. C# 6.0 이상이 지원되는 개발환경이라면 문자열 보간을 사용하는 것이 합리적입니다. 물론 C# 5.0 이전 버전을 이용하고 있는 개발자는 선택의 여지가 없이 string.Format()을 이용해야겠지만 말입니다.

이제 문자열 보간 예제 프로그램을 만들어 보겠습니다.

Appendix A/StringInterpolation/MainApp.cs

```
01  using System;
02  using static System.Console;
03
04  namespace StringInterpolation
05  {
06      class MainApp
07      {
08          static void Main(string[] args)
09          {
10              string name = "김튼튼";
11              int age = 23;
12              WriteLine($"{name, -10}, {age:D3}");
13
14              name = "박날씬";
15              age = 30;
16              WriteLine($"{name}, {age,-10:D3}");
17
18              name = "이비실";
19              age = 17;
20              WriteLine($"{name}, {(age > 20 ? "성인" : "미성년자")}");
21          }
22      }
23  }
```

실행 결과:

```
김튼튼        , 023
박날씬, 030
이비실, 미성년자
```