

C#프로그래밍의 이해

데이터 구조와 표현법

데이터를 가공하는 연산자

코드의 흐름 제어하기

04. 기본 데이터 형식 (11/15)

▶ 데이터 형식 바꾸기 (1/5)

- ▶ 형식 변환(Type Conversion) : 변수를 다른 데이터 형식의 변수에 옮겨 담는 것
 - ▶ 박싱과 언박싱도 값 형식과 참조 형식간의 형식 변환

1. 크기가 서로 다른 정수 형식 사이의 변환

- ▶ 작은 정수 형식의 변수에 있는 데이터를 큰 정수 형식의 변수로 옮길 때는 문제가 없지만, 그 반대의 경우, 원본 변수의 데이터가 형식 변환하려는 대상 변수의 용량보다 큰 경우에는 오버플로우(Overflow)가 발생

```
int x = 128; // sbyte의 최대값 127보다 1 큰 수  
Console.WriteLine(x); // 128 출력
```

```
sbyte y = (sbyte)x;  
Console.WriteLine(y); // -128 출력
```


04. 기본 데이터 형식 (13/15)

▶ 데이터 형식 바꾸기 (3/5)

3. 부호 있는 정수 형식과 부호 없는 정수 형식 사이의 변환

- ▶ 교재의 "부호 있는 정수와 부호 없는 정수" 항의 내용 참조

```
09      int a = 500;  
10      Console.WriteLine(a);  
11  
12      uint b = (uint)a;  
13      Console.WriteLine(b);  
14  
15      int x = -30;  
16      Console.WriteLine(x);  
17  
18      uint y = (uint)x;  
19      Console.WriteLine(y);
```



```
500  
500  
-30  
4294967266
```

04. 기본 데이터 형식 (14/15)

▶ 데이터 형식 바꾸기 (4/5)

4. 부동 소수점 형식과 정수 형식 사이의 변환

- ▶ 부동 소수점 형식의 변수를 정수 형식으로 변환하면 데이터에서 소수점 아래는 버리고 소수점 위의 값만 남김
- ▶ 0.1을 정수 형식으로 변환하면 0이 되지만, 0.9도 정수 형식으로 변환하면 0

```
09 float a = 0.9f;  
10 int b = (int)a;  
11 Console.WriteLine(b);  
12  
13 float c = 1.1f;  
14 int d = (int)c;  
15 Console.WriteLine(d);
```



0
1

04. 기본 데이터 형식 (15/15)

▶ 데이터 형식 바꾸기 (5/5)

5. 문자열을 숫자로, 숫자를 문자열로

```
string a = "12345";  
int b = (int)a; // b는 이제 12345?
```

```
int c = 12345;  
string d = (string)c;
```

컴파일 에러!!

- ▶ 숫자 → 문자열 : ToString() 메소드 이용
예) int a = 3;
string b = a.ToString();
- ▶ 문자열 → 숫자 : Parse() 메소드 이용
예) int a = int.Parse("12345");

05. 상수와 열거 형식 (1/2)

▶ 상수(Constant)

- ▶ 변수와는 달리 그 안에 담긴 데이터를 절대 바꿀 수 없는 메모리 공간
- ▶ 변수를 선언하고 프로그래머가 바꾸지 않으면 되지, 왜 상수가 필요한가?
 - ▶ 프로그래머는 실수를 하는 사람이기 때문에
 - ▶ 아무리 똑똑하고 꼼꼼해도 코드가 수천, 수만 라인에 이르면 어떤 변수를 바꿔도 되는지 또 어떤 변수는 바꾸면 안되는지 잊기 시작
- ▶ 상수 선언 방법 : 아래와 같이 `const` 키워드를 이용하여 선언

```
const int a = 3;  
const double b = 3.14;  
const string c = "bcdef";
```

이렇게 선언한 상수를 변경하려는 코드가 있을 때, 컴파일러가 에러를 일으킴

05. 상수와 열거 형식 (2/2)

- ▶ 열거형(Enumeration)
 - ▶ 열거된 형태의 상수
 - ▶ enum 키워드를 이용하여 선언

상수

```
const int RESULT_NO = 2;  
const int RESULT_CONFIRM = 3;  
const int RESULT_CANCEL = 4;  
const int RESULT_OK = 5;
```

VS

열거형

```
enum DialogResult {YES, NO, CANCEL, CONFIRM, OK }
```

중복된 값을 가진 상수를 선언
할 위험도 없고, 읽기에도 편함

06. NULLABLE 형식

- ▶ C# 컴파일러는 변수의 메모리 공간에 반드시 어떤 값이든 넣도록 강제함
- ▶ 하지만 프로그래밍을 하다 보면 어떤 값도 가지지 않는 변수가 필요할 때가 생김. 0이 아닌, 정말 비어있는 변수, 즉 null한 변수가 필요할 때가 생김
- ▶ Nullable 형식은 이런 경우를 위해 사용
 - ▶ 데이터 형식 뒤에 '?'만 붙여주면 끝
예) `int? a = null;`
`float? b = null;`
`double? c = null;`

07.VAR: 자동 형식 지정

- ▶ C#은 강력한 형식 검사를 하는 언어이지만, 약한 형식 검사를 하는 언어의 편리함도 지원
- ▶ int, string 같은 명시적 형식 대신 **var** 를 사용해서 변수를 선언하면 컴파일러가 자동으로 해당 변수의 형식을 지정

예) var a = 3; // a는 int 형식
 var b = "Hello"; // b는 string 형식

08. 공용 형식 시스템 (CTS) (1/2)

- ▶ C#의 모든 데이터 형식 체계는 사실 C# 고유의 것이 아님
- ▶ .NET 프레임워크의 형식 체계의 표준인 공용 형식 시스템(Common Type System)을 따르고 있을 뿐임
- ▶ 공용 형식 시스템의 이름은 “모두가 함께 사용하는 데이터 형식 체계”라고 뜻.
즉, 공용 형식 시스템은 .NET 언어들 이라면 반드시 따라야 하는 데이터 형식 표준
- ▶ 마이크로소프트가 “공용”형식 시스템을 도입한 이유는 .NET 언어들끼리 서로 호환성을 갖도록 하기 위해서임
- ▶ 우리가 앞에서 살펴봤던 모든 데이터 형식에 관련한 이야기가 곧 CTS의 이야기

08. 공용 형식 시스템 (CTS) (2/2)

▶ CTS 형식 표

Class name	C# 형식	C++ 형식	비주얼 베이직 형식
System.Byte	byte	unsigned char	Byte
System.SByte	sbyte	char	SByte
System.Int16	short	short	Short
System.Int32	int	int 또는 long	Integer
System.Int64	long	__int64	Long
System.UInt16	ushort	unsigned short	UShort
System.UInt32	uint	unsigned int 또는 unsigned long	UInteger
System.UInt64	ulong	unsigned __int64	ULong
System.Single	float	float	Single
System.Double	double	double	Double
System.Boolean	bool	bool	Boolean
System.Char	char	wchar_t	Char
System.Decimal	decimal	Decimal	Decimal
System.IntPtr	없음.	없음.	없음
System.UIntPtr	없음.	없음.	없음
System.Object	object	Object*	Object
System.String	string	String*	String

C#프로그래밍의 이해

데이터를 가공하는 연산자

01. C#에서 제공하는 연산자 둘러보기

▶ C#이 제공하는 주요 연산자의 목록

분류	연산자
산술 연산자	+, -, *, /, %
증가/감소 연산자	++, --
관계 연산자	<, >, ==, !=, <=, >=
조건 연산자	?:
논리 연산자	&&, , !
비트 연산자	<<, >>, &, , ^, ~
할당 연산자	=, +=, -=, *=, /=, %=, &=, =, ^=, <<=, >>=

▶ 연산자 사용 예

할당 연산자 덧셈 연산자

```
int result = 3 + 4;
```

02. 산술 연산자 (1/2)

▶ 산술연산자

- ▶ 수치 형식의 데이터를 다루는 연산자
(정수 형식과 부동 소수점 형식, Decimal 형식에 대해서만 사용 가능)
- ▶ 두 개의 피연산자가 필요한 이항 연산자(Binary Operator)

연산자	설명	지원 형식
+	양쪽 피연산자를 더합니다.	모든 수치 데이터 형식
-	왼쪽 피연산자에서 오른쪽 피연산자를 차감합니다.	모든 수치 데이터 형식
*	양쪽 피연산자를 곱합니다.	모든 수치 데이터 형식
/	왼쪽 연산자를 오른쪽 피연산자로 나눈 몫을 구합니다.	모든 수치 데이터 형식
%	왼쪽 연산자를 오른쪽 피연산자로 나눈 후의 나머지를 구합니다.	모든 수치 데이터 형식

02. 산술 연산자 (2/2)

▶ 산술 연산자의 사용 예

```
Console.WriteLine( 3 + 4 ); // 7 출력
```

```
int result = 15 / 3;
```

```
Console.WriteLine( result ); // 5 출력
```

03. 증가 연산자와 감소 연산자 (1/2)

- ▶ 증가 연산자는 피연산자의 값을 1 증가
- ▶ 감소 연산자는 피연산자의 값을 1 감소

연산자	이름	설명	지원 형식
++	증가 연산자	피연산자의 값을 1 증가시킵니다.	모든 수치 데이터 형식과 열거 형식
--	감소 연산자	피연산자의 값을 1 감소시킵니다.	모든 수치 데이터 형식과 열거 형식

▶ 사용 예

```
int a = 10;  
a++; // a는 11  
a--; // a는 10
```


03. 증가 연산자와 감소 연산자 (2/2)

- ▶ 전위/후위 연산자의 차이

- ▶ 증가 연산자

```
int a = 10;  
Console.WriteLine( a++ ); // 11이 아닌, 10을 출력. 출력 후에 a는 11로 증가.  
Console.WriteLine( ++a ); // 12를 출력.
```

- ▶ 감소 연산자

```
int a = 10;  
Console.WriteLine( a-- ); // 9가 아닌, 10을 출력. 출력 후에 a는 9로 감소.  
Console.WriteLine( --a ); // 8을 출력.
```

04. 문자열 결합 연산자

▶ `int result = 123 + 456;`

▶ Result는 579

수치 형식에 사용하는 + 연산자는 덧셈 연산자

▶ `string result = "123" + "456";`

▶ Result는 "123456"

String 형식에 사용하는 + 연산자는 덧셈 연산자가 아닌, 문자열 결합 연산자

05. 관계 연산자 (1/2)

- ▶ 관계 연산자(Relational Operator)는 두 피연산자 사이의 관계를 확인하는 연산자

연산자	설명	지원 형식
<	왼쪽 피연산자가 오른쪽 피연산자보다 작으면 참, 아니면 거짓	모든 수치 형식과 열거 형식
>	왼쪽 피연산자가 오른쪽 피연산자보다 크면 참, 아니면 거짓	모든 수치 형식과 열거 형식
<=	왼쪽 피연산자가 오른쪽 피연산자보다 작거나 같으면 참, 아니면 거짓	모든 수치 형식과 열거 형식
>=	왼쪽 피연산자가 오른쪽 피연산자보다 크거나 같으면 참, 아니면 거짓	모든 수치 형식과 열거 형식
==	왼쪽 피연산자가 오른쪽 피연산자와 같으면 참, 아니면 거짓	모든 데이터 형식에 대해 사용 가능
!=	왼쪽 피연산자가 오른쪽 피연산자와 다르면 참, 아니면 거짓	모든 데이터 형식에 대해 사용 가능. string 와 object 형식에 대해서도 사용이 가능합니다.

05. 관계 연산자 (2/2)

▶ 관계 연산자의 사용 예

```
bool result;
```

```
result = 3 > 4; // 거짓
```

```
result = 3 >= 4; // 거짓
```

```
result = 3 < 4; // 참
```

```
result = 3 <= 4; // 참
```

```
result = 3 == 4; // 거짓
```

```
result = 3 != 4; // 참
```

06. 논리 연산자 (1/2)

▶ 논리 연산

- ▶ 숫자가 피연산자인 연산을 말하듯, 부울 연산(Boolean Operation)이라고도 하는 논리 연산(Logical Operation)은 참과 거짓으로 이루어지는 진리값이 피연산자인 연산
- ▶ 진리표

A	B	A && B	A B	!A
참	참	참	참	거짓
참	거짓	거짓	참	거짓
거짓	거짓	거짓	거짓	참
거짓	참	거짓	참	참

06. 논리 연산자 (2/2)

▶ 논리 연산자의 사용 예

```
int a = 3;  
int b = 4;  
bool c = a < b && b < 5; // c는 true  
bool d = a > b && b < 5; // d는 false (a > b가 거짓이므로)  
bool e = a > b || b < 5; // e는 true (b < 5가 참이므로)  
bool f = !e;           // f는 false (참인 true를 부정했으므로)
```

07. 조건 연산자

- ▶ 조건 연산자(Conditional Operator) ?:는 조건에 따라 두 값 중 하나의 값을 반환

- ▶ 사용 형식

조건식 ? 참일_때의_값 : 거짓일_때의_값

- ▶ 사용 예

```
int a = 30;  
string result = a == 30 ? "삼십" : "삼십아님" ; // result는 "삼십"
```

08. 비트 연산자 (1/6)

- ▶ 비트 연산자 : 비트 수준에서 데이터를 가공하는 연산자

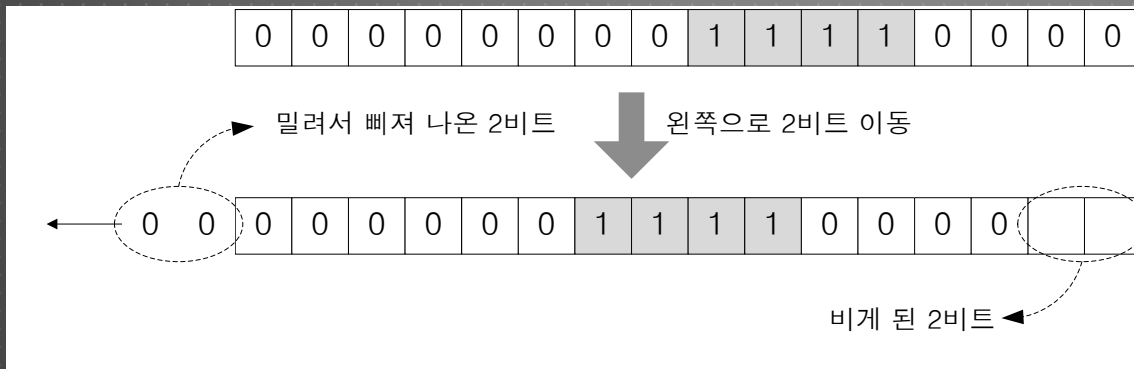
연산자	이름	설명	지원 형식
<<	왼쪽 시프트 연산자	첫 번째 피연산자의 비트를 두 번째 피연산자의 수만큼 왼쪽으로 이동시킵니다.	첫 번째 피연산자는 int, uint, long, ulong 이며 피연산자는 int 형식만 지원합니다.
>>	오른쪽 시프트 연산자	첫 번째 피연산자의 비트를 두 번째 피연산자의 수만큼 오른쪽으로 이동시킵니다.	<< 와 같습니다.
&	논리곱(AND) 연산자	두 피연산자의 비트 논리곱을 수행합니다.	정수 계열 형식과 bool 형식에 대해 사용할 수 있습니다.
	논리합(OR) 연산자	두 피연산자의 비트 논리합을 수행합니다.	&와 같습니다.
^	배타적 논리합(XOR) 연산자	두 피연산자의 비트 배타적 논리합을 수행합니다.	&와 같습니다.
~	보수(NOT) 연산자	피연산자의 비트를 0은 1로, 1은 0으로 반전시킵니다. 단항 연산자입니다.	int, uint, long, ulong에 대해 사용이 가능합니다.

08. 비트 연산자 (2/6)

- ▶ 시프트 연산자(Shift Operator) : 비트를 왼쪽이나 오른쪽으로 이동
 - ▶ `Int a = 240 << 2;` // 240을 왼쪽으로 2비트 옮겨 보자

0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

1) 240을 비트로 나타낸 모습 :



2) 전체 비트를 왼쪽으로 2비트 밀어내고 밀려 나간 왼쪽의 두 개 비트는 제거

0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

3) 오른쪽에 비어있는 두 비트 공간은 0으로 채움

08. 비트 연산자 (3/6)

▶ 시프트 연산자 사용 예

```
int a = 240; // 00000000 00000000 00001111 00000000
int result_1 = a << 2 ; // 00000000 00000000 00111100 00000000
int result_2 = a >> 2 ; // 00000000 00000000 00000011 11000000
```

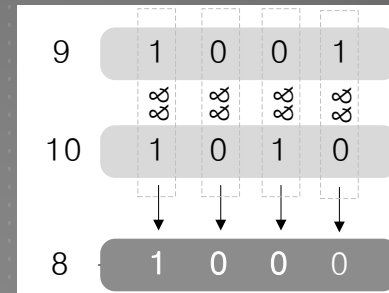
08. 비트 연산자 (4/6)

- ▶ 논리 연산 : 참 또는 거짓의 진리 값을 피연산자로 하는 연산

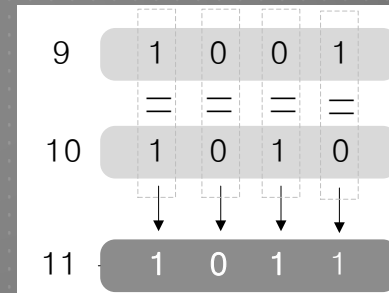
연산자	이름	설명	지원 형식
&	논리곱(AND) 연산자	두 피연산자의 비트에 대해 논리곱을 수행합니다.	정수 계열 형식과 bool 형식에 대해 사용할 수 있습니다.
	논리합(OR) 연산자	두 피연산자의 비트에 대해 논리합을 수행합니다.	&와 같습니다.
^	배타적 논리합(XOR) 연산자	두 피연산자의 비트에 대해 배타적 논리합을 수행합니다.	&와 같습니다.
~	보수(NOT) 연산자	피연산자의 비트에 대해 0은 1로, 1은 0으로 반전시킵니다. 단항 연산자입니다.	int, uint, long, ulong에 대해 사용이 가능합니다.

08. 비트 연산자 (5/6)

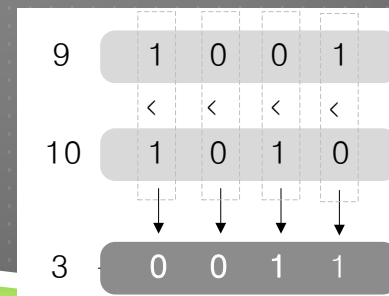
- ▶ 9(1001)와 10(1010)의 논리곱 예



- ▶ 9(1001)와 10(1010)의 논리합 예

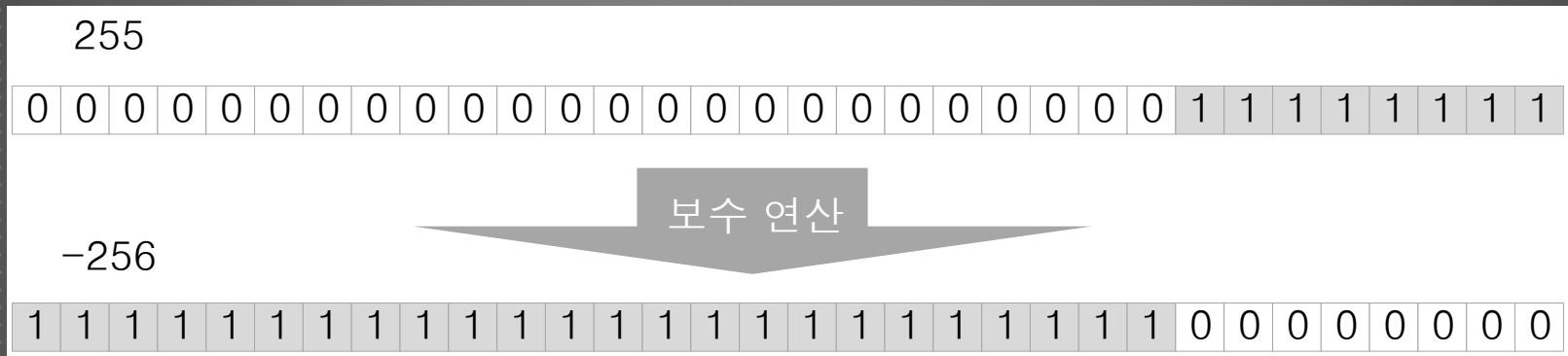


- ▶ 9(1001)와 10(1010)의 배타적 논리합 예



08. 비트 연산자 (6/6)

▶ 보수 연산 예



▶ 보수연산자의 사용 예

```
int a = 255;  
int result = ~a; // result는 -256
```

09. 할당 연산자

- ▶ 할당 연산자(Assignment) : 변수 또는 상수에 피연산자를 할당

연산자	이름	설명
=	할당 연산자	오른쪽 피연산자를 왼쪽 피연산자에게 할당합니다.
+=	덧셈 할당 연산자	$a += b$; 는 $a = a + b$; 와 같습니다.
-=	뺄셈 할당 연산자	$a -= b$; 는 $a = a - b$; 와 같습니다.
*=	곱셈 할당 연산자	$a *= b$; 는 $a = a * b$; 와 같습니다.
/=	나눗셈 할당 연산자	$a /= b$; 는 $a = a / b$; 와 같습니다.
%=	나머지 할당 연산자	$a \% b$; 는 $a = a \% b$; 와 같습니다.
&=	논리곱 할당 연산자	$a \& b$; 는 $a = a \& b$; 와 같습니다.
=	논리합 할당 연산자	$a b$; 는 $a = a b$; 와 같습니다.
^=	배타적 논리합 할당 연산자	$a \wedge b$; 는 $a = a \wedge b$; 와 같습니다.
<<=	왼쪽 시프트 할당 연산자	$a \ll b$; 는 $a = a \ll b$; 와 같습니다.
>>=	오른쪽 시프트 할당 연산자	$a \gg b$; 는 $a = a \gg b$; 와 같습니다.

10. 연산자의 우선 순위

우선 순위	종류	연산자
1	증가/감소 연산자	후위 ++/-- 연산자
2	증가/감소 연산자	전위 ++/-- 연산자
3	산술 연산자	* / %
4	산술 연산자	+ -
5	시프트 연산자	<< >>
6	관계 연산자	< > <= >= is as (is와 as 연산자는 뒷부분에서 설명합니다.)
7	관계 연산자	== !=
8	비트 논리 연산자	&
9	비트 논리 연산자	^
10	비트 논리 연산자	
11	논리 연산자	&&
12	논리 연산자	
13	조건 연산자	?:
14	할당 연산자	= *= /= %= += -= <<= >>= &= ^= =

코드의 흐름 제어하기

01. 분기문 (1/7)

- ▶ 분기문 (Branching Statement)
 - ▶ 프로그램의 흐름을 조건에 따라 여러 갈래로 나누는 흐름 제어 구문
- ▶ C#은 다음 두 가지의 분기문을 제공
 - ▶ 한번에 하나의 조건만 평가할 수 있는 if 문
 - ▶ 한번에 여러 개의 조건을 평가할 수 있는 switch문

01. 분기문 (2/7)

- ▶ if, else, 그리고 else if (1/3)

- ▶ 형식

```
if ( 조건식 )  
    참인경우에_실행할_코드;
```

```
if ( 조건식 )  
{  
    // 참인 경우에  
    // 실행할  
    // 코드  
}
```

- ▶ 예제

```
int a = 10;  
  
if ( (a % 2) == 0 )  
    Console.WriteLine("짝수");
```

01. 분기문 (3/7)

- ▶ if, else, 그리고 else if (2/3)
 - ▶ a를 0으로 나눈 나머지가 0인 경우와 그렇지 않은 경우

```
if ( (a % 2) == 0 )  
    Console.WriteLine("짝수");  
if ( (a % 2) != 0 )  
    Console.WriteLine("홀수");
```

각 if 절 밑에 있는 코드가 언제 실행되는지는 읽을 때마다 해독해야 함

VS

```
if ( (a % 2) == 0 )  
    Console.WriteLine("짝수");  
else  
    Console.WriteLine("홀수");
```

if 절의 평가식이 성공한 경우와 실패한 경우에 실행되는 코드가 명확하게 보임

01. 분기문 (4/7)

▶ if, else, 그리고 else if (3/3)

- ▶ if의 분기가 세 갈래 이상이 되어야 하는 경우에 사용하는 else if

```
int a = - 10;  
  
if ( a < 0 )  
    Console.WriteLine("음수");  
else if ( a > 0 )  
    Console.WriteLine("양수");  
else  
    Console.WriteLine("0");
```

01. 분기문 (5/7)

- ▶ if 문 중첩해서 사용하기

```
if ( number > 0 )  
{  
    if ( number % 2 == 0 )  
        Console.WriteLine("0보다 큰 짝수.");  
    else  
        Console.WriteLine("0보다 큰 홀수.");  
}  
else  
{  
    Console.WriteLine("0보다 작거나 같은 수.");  
}
```

중첩된 if 문