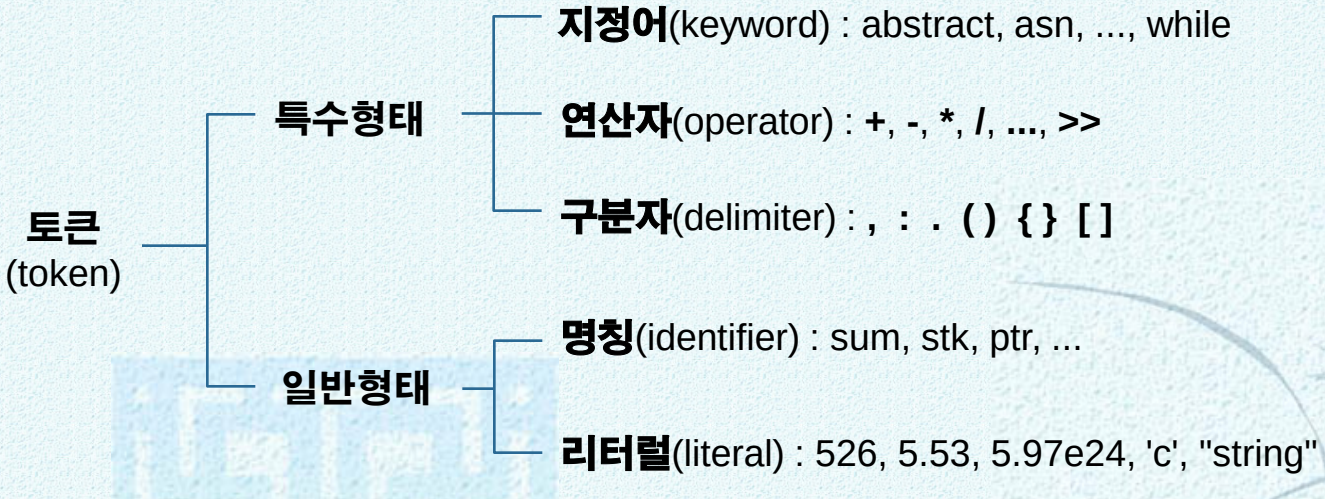
제 2 장 언어 구조

- 어휘 구조
- 자료형
- 연산자
- 형 변환

어휘 구조

- 어휘
  - 프로그램을 구성하고 있는 기본 소자
  - 토큰(token)이라 부름
  - 문법적으로 의미있는 최소 단위

- 토큰의 종류



지정어

- 프로그래밍 언어 설계 시에 그 기능과 용도가 이미 정의되어 있는 단어
- C# **지정어** (77**개**) – C# language specification (ECMA TC39/TG2)

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	goto
if	implicit	in	int	interface
internal	is	lock	long	namespace
new	null	object	operator	out
override	params	private	protected	public
readonly	ref	return	sbyte	sealed
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	void
volatile	while			

명칭 [1/2]

- 명칭의 의미
  - 자료의 항목(변수, 상수, 배열, 클래스, 메소드, 레이블)을 식별하기 위하여 붙이는 이름
- 명칭의 형태
  - 문자로 시작
  - 대소문자 구분
  - '@'기호 붙이면 지정어를 명칭으로 사용 가능

바른 명칭들 : sum, sum1, money_sum, moneySum, @int, 변수
틀린 명칭들 : 1sum, sum!, $sum, #sum, Money Sum, virtual

명칭 [2/2]

- **자바 문자 집합** (character set)
  - **유니코드**(Unicode)
  - **문자 표현** : 16 Bit
  - **세계 모든 언어 표현**

```
static  readonly double  π = 3.1415926535897;
```

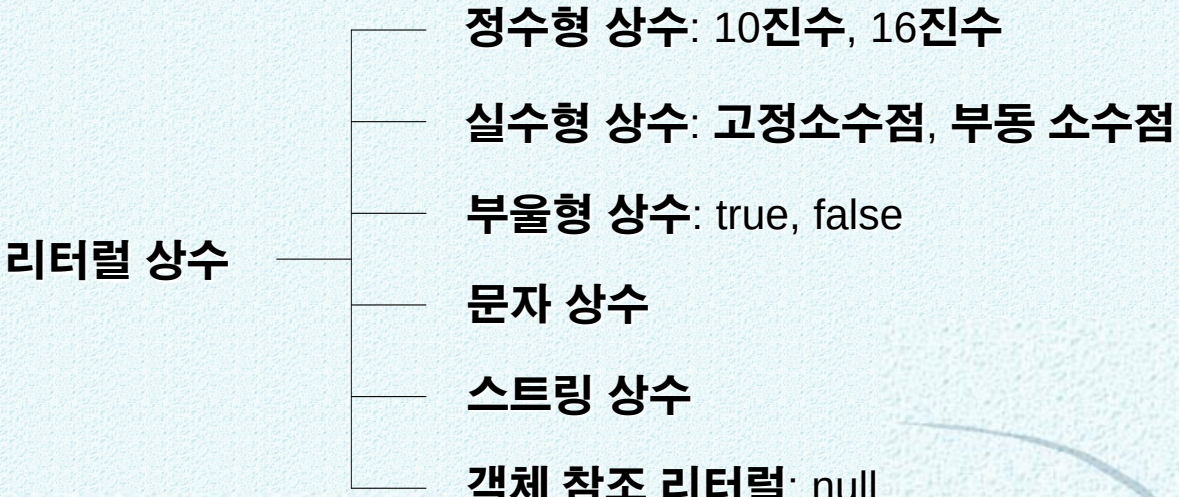
- **@ 기호** (at sign)
  - **지정어와 함께 사용할 때 지정어와 구분**
  - **일반 명칭과 함께 사용할 때 동일한 명칭으로 인식**

```
int @int = 10;      // right!
int i, @i;          // error!
```

리터럴

- 리터럴의 의미
  - 자신의 표기법이 곧 자신의 값이 되는 상수

- 리터럴의 종류



정수형 상수

- **정수형 상수의 종류**
  - 10**진수**(decimal)
  - 16**진수**(hexadecimal)
  - C#**에서는** 8**진수**(octal)**을 지원하지 않음**.

```
10진수 :    15,    255,    65535
16진수 :   0xF,   0xFF,    0xFFFF
```

- **정수형 상수의 비트 수**
  - default: 32-bit
  - long**형**: 64-bit (**접미어**: -L, -l)

실수형 상수

- **실수형 상수의 분류**
  - **지수**(exponent) **부분의 유무에 따라**
    - **고정소수점**(fixed-point) **수** : 1.414, 3.1415924, 0.00001
    - **부동소수점**(floating-point) **수**: 0.1414e01, 0.1414E1, 5E-5f
  - **용도에 따라**
    - float, double : **과학 연산**
    - decimal : **회계 연산**
  - **정밀도**(precision)**에 따라**
    - float **형** : **접미어** -f, -F
    - double **형** : default
    - decimal **형** : **접미어** -m, -M

부울형 상수와 문자 상수

- **부울형 상수**
  - binary value
  - false, true (**정수값 0과 1로 상호 변환되지 않음**.)

- **문자 상수**
  - **단일 인용부호**(single quote) **사이에 표현**
    - **예**) ‘a’ , ‘A’
  - escape sequence : **특수한 문자를 표현**

\'	single quote(\u0027)
\"	double quote(\u0022)
\0	null(\u0000)
\b	backspace(\u0008)
\f	form feed(\u000C)
\n	new line(\u000A)
\r	carriage return(\u000D)
\t	horizontal tab(\u0009)

스트링 상수

- **스트링 상수의 의미와 특징**
  - **이중 인용부호**(double quote) **사이에 표현된 스트링**
    - **예**) “hello world”, “I am a string.”
  - System.String **클래스의 객체로 취급**

- **축어적 스트링 상수**(verbatim string literal)
  - **스트링 상수 내에 에스케이프 문자열 표현**
  - **‘@’ 기호와 함께 스트링 상수 기술**

```
string a = “hello\t world”;      // hello      world
string b = “hello \\t world”;    // hello \t world
string c = @”hello \t world”;    // hello \t world
```

객체 참조 리터럴

- **객체 참조 리터럴** (object reference)
  - **널** (null)
  - **아무 객체도 가리키지 않는 상태**
  - **부적당하거나 객체를 생성할 수 없는 경우 사용**
  - **초기화에 사용**

주석 [1/2]

- 프로그램을 설명하기 위한 문장
  - 프로그램의 실행에는 무관
  - 프로그램 유지보수에 중요

- 주석의 종류.
  - // comment
    - //부터 새로운 줄 전까지 주석으로 간주
    - 예) int size = 100; //size는 100으로 최기화

  - /* comment */
    - /*와 다음 */ 사이의 모든 문자들은 주석으로 간주
    - 주석문 안에서 또 다른 주석이 포함될 수 없음
    - 예) /* C# 언어에서는 여러 줄의 주석을 위해 지금 사용하고 있는 주석의 형태를 지원하고 있다. */

주석 [2/2]

- /// comment
  - /// **다음의 문자들은 주석으로 간주**
  - C# **프로그램에 대한 웹 보고서를 작성하는데 사용하는 방법**
  - XML **태그를 이용하여 기술**
  - **컴파일 시에** /doc **옵션을 사용하여** XML **문서 생성**

    csc CommentApp.cs /doc:CommentApp.xml

  - XML **문서**
    - T:CommentApp - T is Type
    - M:CommentApp.Main - M is Method

자료형 [1/2]

- 자료형의 의미
  - 자료 객체가 갖는 형으로 구조 및 개념, 값의 범위, 연산 등을 정의

- 자료형의 종류

자료형 [2/2]

- C# 의 자료형은 공통자료형 시스템(CTS)에서 정의한 형식으로 표현할 수 있다.

```
// 다음 두 선언의 의미는 동일하다.
System.Int32 x;    // CTS 형으로 정수형 변수 x의 선언
int x;             // C# 형으로 정수형 변수 x의 선언
```

- CTS 형과 C# 자료형과의 관계

CTS 자료형	의 미	C#자료형	CTS 자료형	의 미	C#자료형
System.Object	객체형	object	System.Int64	64비트 정수형	long
System.String	스트링형	string	System.Uint64	64비트 부호없는 정수형	ulong
System.Sbyte	부호있는 바이트형	sbyte	System.Char	문자형	char
System.Byte	바이트형	byte	System.Single	단일 정밀도 실수형	float
System.Int16	16비트 정수형	short	System.Double	이중 정밀도 실수형	double
System.Uint16	16비트 부호없는 정수형	ushort	System.Boolean	불린형	bool
System.Int32	32비트 정수형	int	System.Decimal	10진수형	decimal
System.Uint32	32비트 부호없는 정수형	uint	System.Uint32	32비트 부호없는 정수형	uint

값형 - 정수형

- 정수형의 종류
  - 부호있는(signed) 정수형
    - sbyte(8비트), short(16비트), int(32비트), long(64비트)
  - 부호없는(unsigned) 정수형
    - byte(8비트), ushort(16비트), uint(32비트), ulong(64비트)
- 정수형의 크기

C# 자료형	CTS 형	크 기	최소값	최대값
sbyte	System.SByte	8 bit	-128	127
short	System.Int16	16 bit	-32768	32767
int	System.Int32	32 bit	-2147483648	2147483647
long	System.Int64	64 bit	-9223372036854775808	9223372036854775807
byte	System.Byte	8 bit	0	255
ushort	System.UInt16	16 bit	0	65535
uint	System.UInt32	32 bit	0	4294967295
ulong	System.UInt64	64 bit	0	18446744073709551615

값형 - 실수형

- 실수의 표현 방법과 실수 연산은 IEEE 754 표준을 따름
- 실수형의 종류
  - 부동 소수점(floating-point)
    - float(32비트), double(64비트)
  - 10진 자료형(decimal)
    - 고도의 정밀도를 요하는 계산에 이용(회계나 금융관련 계산)
    - 28 유효 자릿수
    - 효율성이 떨어짐 (구조체로 처리하기 때문)

값형 - 문자형과 부울형

- 문자형
  - 16비트 유니코드(Unicode)를 사용
- 부울형(boolean type)
  - true와 false 중 하나의 값만을 가지는 자료형
  - 숫자 값을 가질 수 없음
  - 다른 자료형으로 변환 불가

값형 - 자료형에 대한 초기값

- **선언된 변수가 컴파일러에 의해 묵시적으로 갖게 되는 초기값**(initial value)

자료형	기본 표준값	초기값
byte	zero	(byte) 0
short	zero	(short) 0
int	zero	0
long	zero	0L
sbyte	zero	(byte) 0
ushort	zero	(short) 0
uint	zero	0
ulong	zero	0L
float	positive zero	0.0f
double	positive zero	0.0d
char	널(Null) 문자	'\u0000'
boolean	false	

값형 - 포인터형

- 포인터형의 종류
  - 매니지드 코드(managed code)
    - .NET 프레임워크가 관리하는 코드 부분
  - 언매니지드 코드(unmanaged code)
    - C#에서 작성할 수 없는 플랫폼 의존적인 작업의 코드 부분
- 포인터 관련 연산자
  - 주소 연산자(&): 변수의 주소를 반환
  - 값 연산자(*): 메모리 주소에 저장되어 있는 값을 반환
  - 포인터 참조 연산자(->): 구조체 멤버에 접근하여 멤버의 값을 반환
- 포인터형의 컴파일 방법과 주의사항
  - csc /unsafe PointerApp.cs
  - 포인터 연산은 반드시 unsafe 코드 블록 내에서만 가능

[예제 2.9 PointerApp.cs]

```
using System;
   class PointerApp {
      unsafe public static void Swap(int* px, int* py) {
         int tmp = *px;
         *px = *py;
         *py = tmp;
      }
      public static void Main() {
         int x = 1, y = 2;
         Console.WriteLine("Before : x = " + x + ", y = " + y);
         unsafe {
            Swap(&x, &y);
         }
         Console.WriteLine(" After : x = " + x + ", y = " + y);
      }
   }
```

컴파일 방법:

```
csc /unsafe PointerApp.cs
```

실행 결과 :

```
Before : x = 1, y = 2
After : x = 2, y = 1
```

값형 - 열거형

- 열거형의 의미
  - 서로 관련 있는 상수들의 모음을 심볼릭한 명칭의 집합으로 정의한 것
- 기호상수
  - 집합의 원소로 기술된 명칭
- 순서값
  - 집합에 명시된 순서에 따라 0부터 부여된 값
  - 정수형으로 교환하여 사용할 수 있다.

참조형 – 배열형 [1/3]

- 배열형의 의미
  - 같은 형의 여러 개의 값을 저장하는데 사용하는 자료형
  - 순서가 있는 원소들의 모임

- 배열을 사용하기 위한 과정
  - 배열 선언
    - 배열이름, 차원, 그리고 원소의 형 등을 명시

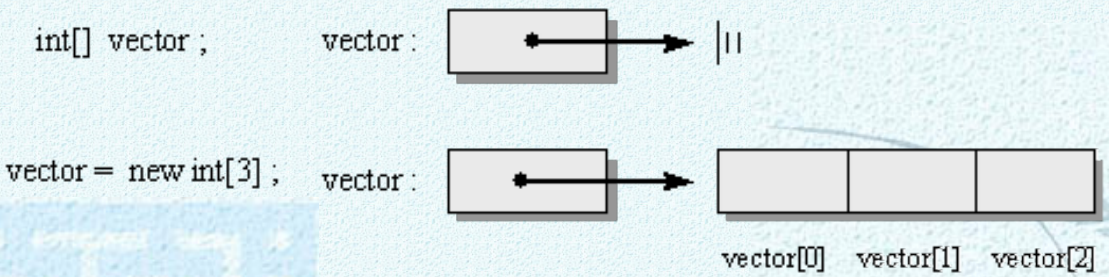
```
int[]    vector;                          //1차원 배열
short[,] matrix;                          //2차원 배열
object[] myArray;
int[]    initArray = {0, 1, 2, 3, 4, 5};  //선언과 함께 초기값 부여
```

참조형 – 배열형 [2/3]

- 배열 객체 생성
  - new 연산자를 통해서 동적으로 생성
  - 배열의 객체를 생성함으로써 배열 이름은 특정 배열 객체를 가리킴

```
vector = new int[100];
matrix  = new short[10,100];
myArray = new Point[3];
```

- 배열 객체 선언과 생성



참조형 – 배열형 [3/3]

- 배열에 값 저장
  - 배열의 인덱스는 0부터 시작
  - Length 프로퍼티 통한 배열의 길이 접근
  - 인덱스 범위 초과 : IndexOutOfRangeException 발생

```
int[] vector = new int[100];
// ...
for (int i=0; i < vector.Length; i++) vector[i] = i;
```

- 배열의 배열
  - 배열의 원소가 다시 배열이 되는 배열
  - 다차원 배열과 구분
  - 각 원소에 해당하는 배열이 서로 다른 크기를 가질 수 있다.

```
int[][] arrayOfArray;
```

[예제 2.12 ArrayOfArray.cs]

```
using System;
  class ArrayOfArrayApp {
    public static void Main() {
      int[][] arrayOfArray = new int[3][];                    // declaration
      int i, j;
      for (i = 0; i < arrayOfArray.Length; i++)               // creation
        arrayOfArray[i] = new int[i+3];
      for (i = 0; i < arrayOfArray.Length; i++)               // using
        for (j = 0; j < arrayOfArray[i].Length; j++)
          arrayOfArray[i][j] = i*arrayOfArray[i].Length + j;
      for (i = 0; i < arrayOfArray.Length; i++) {             // printing
        for (j = 0; j < arrayOfArray[i].Length; j++)
          Console.Write(" " + arrayOfArray[i][j]);
        Console.WriteLine();
      }
    }
  }
```

실행 결과 :

```
 0 1 2
 4 5 6 7
 10 11 12 13 14
```

스트링형

- 스트링형의 의미
  - 문자열을 표현하기 위해 사용하는 자료형
  - System.String 클래스형과 동일한 자료형
- StringBuilder 클래스
  - 효율적으로 스트링을 다루기 위한 클래스
  - 객체에 저장된 내용을 임의로 변경가능
  - 스트링 중간에 삽입, 추가시키는 다양한 메소드 제공

연산자 [1/2]

- 식(expression)
  - 문장에서 값을 계산하는데 사용
  - 식은 연산자(operator)와 피연자(operand)로 구성
  - 식의 값에 따라
    - 산술식, 관계식, 논리식으로 구분

- 연산자(operator)
  - 식의 의미를 결정
  - 피연산자가 어떻게 계산될지를 나타내는 기호
  - C#언어 스펙 48개의 연산자 정의

연산자 [2/2]

연산자 종류

C# 언어의 연산자

- **산술 연산자** : + - * / % **단항+ 단항-**
- **관계 연산자** : > >= < <= == !=
- **논리 연산자** : && || !
- **증감 연산자** : **전위++ 전위– 후위++ 후위--**
- **비트 연산자** : & | ^ ~ << >>
- **조건 연산자** : ? :
- **배정 연산자** : = += -= *= /= %= &= |= ^= <<= >>=
- **캐스트 연산자** : (**자료형**)
- **형 검사 연산자** : is as
- **배열 연산자** : []
- **메소드 연산자** : ()
- **멤버 접근 연산자** : .
- **지정어 연산자** : new typeof checked unchecked

산술 연산자

- 의미
  - 수치 연산을 나타내는 연산자

- 연산자 종류
  - **단항 산술 연산자**: +, -
  - **이항 산술 연산자**: +, -, *, /, %

```
x  =  -5 ;          // 음수 5
x  =  -(-5) ;       // 양수 5
x  =  -(3-5) ;      // 양수 2
```

- % : **나머지 연산자**(remainder operator)
  - x % y = x - (x / y) * y

관계 연산자

- 의미
  - 두 개의 값을 비교하는 이항 연산자
  - 연산결과 : true or false
  - 관계 연산자가 포함된 식 : 관계식
    - for, while, do-while의 조건식

- 연산자 우선순위
  - 관계 연산자는 산술 연산자보다 우선순위가 낮다.

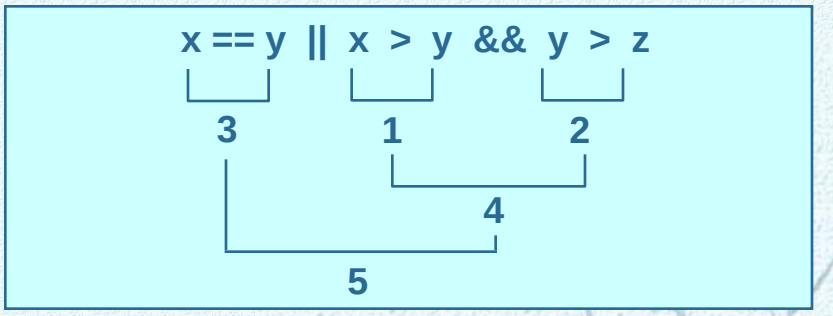
관계 연산자 우선순위		
연산자		우선순위
비교 연산자 항등 연산자	> >= < <= == !=	↕ (높음) (낮음)

```
b == x < y   ===>   b == (x < y)
a > b + c    ===>   a > (b + c)
```

논리 연산자

- 의미
  - 두 피연산자의 논리 관계를 나타내는 연산자
- 연산자 종류
  - 논리곱(&&), 논리합(||), 논리부정(!)
- 연산자 우선순위
  - 논리연산자는 산술 연산자나 관계연산자보다 연산 순위가 낮다.

논리 연산자			
연산 의미	연산자	우선순위	형태
논리부정 논리곱 논리합	! && ||	↕ (높음) (낮음)	!x x && y x || y



[예제 2.18 LogicalOperatorApp.cs]

```
using System;
   class LogicalOperatorApp {
      public static void Main() {
         int x=3, y=5, z=7;
         bool b;
         b = x < y && y < z;
         Console.WriteLine("Result = " + b);
         b = x == y || x < y && y > z;
         Console.WriteLine("Result = " + b);
      }
   }
```

실행 결과 :

```
Result = True
Result = False
```

증가 및 감소 연산자

- 의미
  - 정수형 변수의 값을 하나 증가시키거나 감소시키는 연산자
- 연산자 기호
  - ++, --
  - 변수가 아닌 식에는 사용 못함: (a+b)++ // error
  - 실수형 적용 안됨 : f++ //error: f is float
- 연산자 종류
  - 전위 연산자(prefix operator)

```
n = 1;
x = ++n;      // x=2, n=2
```

  - 후위 연산자(postfix operator)

```
n = 1;
x = n++;      // x=1, n=2
```

비트 연산자

- 의미
  - 비트 단위로 연산을 수행하는 연산자
  - 피연산자는 반드시 정수형
- 연산자 종류
  - 비트 논리곱(&), 비트 논리합(|), 비트 배타적 논리합(^), 왼쪽 이동(<<), 오른쪽 이동(>>), 1의 보수(~)
- 연산자 우선순위

비트 연산자			
연산 의미	연산자	우선순위	사용 예
비트 논리곱	&	(높음)	x & y
비트 논리합	|	↕	x | y
비트 배타적 논리합(exclusive OR)	^	↕	x ^ y
왼쪽 이동(left shift)	<<	↕	x << y
오른쪽 이동(right shift)	>>	↕	x >> y
1의 보수(one's complement)	~	(낮음)	~x

[예제 2.20 BitOperatorApp.cs]

```
using System;
   class BitOperatorApp {
      public static void Main() {
         int x=9, y=3;
         Console.WriteLine(x + " & " + y + " = " + (x&y));
         Console.WriteLine(x + " | " + y + " = " + (x|y));
         Console.WriteLine(x + " ^ " + y + " = " + (x^y));
         Console.WriteLine("~10 = " + (~10));
      }
   }
```

실행 결과 :

```
9 & 3 = 1
9 | 3 = 11
9 ^ 3 = 10
~10 = -11
```

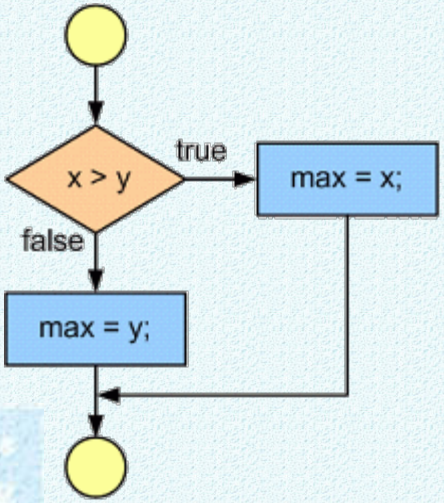
조건 연산자

- 의미
  - 의미가 if문장과 같은 삼항 연산자
- 형태
  - 식1 ? 식2 : 식3

```
max = x > y ? x : y ;
```

⟺

```
if ( x > y ) max = x;
else max =y;
```



[예제 2.22 ConditionalOperatorApp.cs]

```
using System;
   class ConditionalOperatorApp {
      public static void Main() {
         int a, b, c;
         int m;
         Console.Write("Enter three numbers : ");
         a = Console.Read() - '0';
         b = Console.Read() - '0';
         c = Console.Read() - '0';
         m = (a > b) ? a : b;
         m = (m > c) ? m : c;
         Console.WriteLine("The largest number = " + m);
      }
   }
```

입력 데이터 :

```
Enter three numbers : 526
```

실행 결과 :

```
The largest number = 6
```

복합 배정 연산자

- 의미
  - 이항 연산자와 배정연산자가 결합하여 이루어진 연산자

식1 = 식1 op 식2	⟺	식1 op= 식2

op:	- 산술 연산자: + - * / % - 비트 연산자: & | ^ << >>

- 복합 배정 연산자 사용 예

sum = sum + 1;	⟺	sum += 1;
x = x * y +1	✗	x *= y + 1

캐스트 연산자

- 의미
  - 자료형 변환 연산자
- 형태

```
(자료형) 식
```

- 캐스트 연산자 사용 예

```
(int) 3.75          ===>  3
(float) 3           ===>  3.0
(float) (1 / 2)     ===>  0.0
(float) 1/2         ===>  0.5
```

형 검사 연산자

- 연산자 종류
  - 데이터 타입이 지정한 타입과 호환 가능한지 검사: is

```
obj is <type>
```

  - 주어진 값을 지정한 타입으로 변환: as

```
obj as <type>
```

지정어 연산자

- 의미
  - 연산의 의미를 C# 지정어로 나타낸 연산자
- 연산자 종류
  - 객체 생성 연산자: new
  - 객체 형 반환 연산자: typeof
  - 오버플로 검사 연산: checked
  - 오버플로 무시 연산: unchecked

[예제 2.26 IsAsOperatorApp.cs]

```
using System;
  public class IsAsOperatorApp {
    static void IsOperator(object obj) {
      Console.WriteLine(obj + " is int : " + (obj is int));
      Console.WriteLine(obj + " is string : "+(obj is string));
    }
    static void AsOperator(object obj) {
      Console.WriteLine(obj + " as string == null : " +
                  (obj as string == null));
    }
    public static void Main() {
       IsOperator(10);
       IsOperator("ABC");
       AsOperator(10);
       AsOperator("ABC");
    }
  }
```

실행 결과 :

```
10 is int : True
10 is string : False
ABC is int : False
ABC is string ... True
10 as string == null : True
ABC as string == null : False
```

연산자 우선순위

연 산 자	결합법칙	우선순위
() [] . 후위++ 후위-- new typeof checked unchecked	좌측결합	(높음)
단항+ 단항- ! ~ 전위++ 전위-- (자료형)	우측결합	
* / %	좌측결합	
+ -	좌측결합	
<< >>	좌측결합	
< > <= >= is as	좌측결합	
== !=	좌측결합	
&	좌측결합	
^	좌측결합	
|	좌측결합	
&&	좌측결합	
||	좌측결합	
? :	우측결합	
= += -= *= /= %= &= ^= |= <<= >>=	우측결합	(낮음)

형 변환

- 묵시적 형 변환(implicit type conversion)
  - 컴파일러에 의해 자동적으로 수행되는 형 변환
  - 작은 크기 자료형 → 큰 크기 자료형

- 명시적 형 변환(explicit type conversion)
  - 프로그래머가 캐스트 연산자를 사용하여 수행하는 형 변환
  - 형태
    - (자료형) 식
  - 큰 크기 자료형에서 작은 크기 자료형으로 변환 시 정밀도 상실

- 형 변환 금지
  - bool
    - 같은 자료형 이외에 다른 자료형으로의 변환 금지

[예제 2.28 LosePrecisionApp.cs]

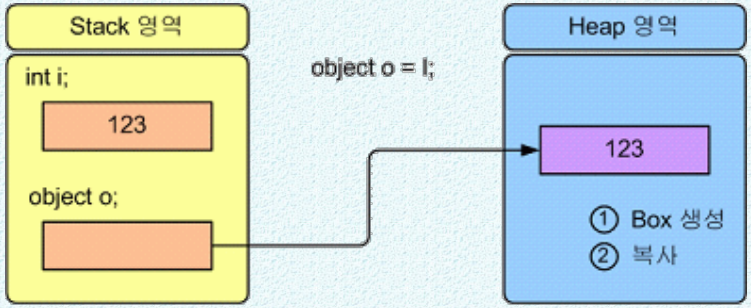
```
using System;
   class LosePrecisionApp {
      public static void Main() {
         int big = 1234567890;
         float approx;
         approx = (float)big;
         Console.WriteLine("difference = " + (big - (int)approx));
      }
   }
```

실행 결과 :

```
difference = -46
```

박싱과 언박싱

- 박싱(boxing)
  - 값형의 데이터를 참조형으로 변환하는 것
  - 컴파일러에 의해 묵시적으로 행해짐
  - 박싱 과정



- 언박싱(unboxing)
  - 참조형의 데이터를 값형으로 변환하는 것
  - 반드시 캐스팅을 통하여 명시적으로 행해짐
  - 반드시 박싱될 때 형으로 언박싱을 해주어야 함

[예제 2.30 BoxingUnboxingApp.cs]

```
using System;
   class BoxingUnboxingApp {
      public static void Main() {
         int foo = 526;
         object bar = foo;              // foo is boxed to bar.
         Console.WriteLine(bar);
         try {
            double d = (short)bar;
            Console.WriteLine(d);
         } catch (InvalidCastException e) {
            Console.WriteLine(e + "Error");
         }
      }
   }
```

실행 결과 :

```
526
System.InvalidCastException:
at BoxingUnboxingApp.Main() Error
```