CHAPTER 02

기본 문법

■ 표현식과 문장

- 표현식: 다음과 같이 값을 만들어내는 간단한 코드

```
273
10 + 20 + 30 * 2
"C# Programming"
```

- 문장: 표현식이 하나 이상 모이고, 마지막에 세미콜론이 찍힌 것. 코드 문장 마지막엔 종결의 의미로 세미콜론을 찍음

```
273;
10 + 20 + 30 + 2;
var name = "윤" + "인" + "성"
Console.Write("Hello C# Programming");
```

■ 키워드

- 키워드: 특별한 의미가 부여된 단어. C# 키워드는 다음과 같음

표 2-1 일반 키워드

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	contnue	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	ooperator	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			

■ 키워드

- 다음 키워드는 특정 위치에서만 키워드로 작동함

표 2-2 컨텍스트 키워드(또는 문맥 키워드)

add	and	alias	ascending	args
async	await	by	descending	dynamic
equals	file	from	get	global
group	init	into	join	let
managed	nameof	nint	not	notnull
nuint	on	or	orderby	partial
record	remove	required	scoped	select
set	unmanaged	value	var	when
where	with	yield		

■ 식별자

- 식별자: 이름을 붙일 때에 사용하는 단어. 다음과 같은 규칙을 지켜야 함
  - 키워드를 사용하면 안 됨
  - 특수 문자는 _만 허용함
  - 숫자로 시작하면 안 됨
  - 공백은 입력하면 안 됨
- 예를 들어 왼쪽의 단어는 식별자로 사용할 수 있지만 오른쪽은 사용할 수 없음

```
alpha
alpha10
_alpha
AlPha
ALPHA
```

O

```
break
273alpha
has space
```

X

■ 식별자

- 식별자를 만들 때는 알파벳을 사용하는 것이 관례
- 알파벳 외의 관례를 정리하면 다음과 같음

① 클래스, 속성, 메서드, 네임스페이스의 이름은 항상 대문자로 시작한다.

② 지역 변수와 전역 변수의 이름은 항상 소문자로 시작한다.

③ 여러 단어로 이루어진 식별자는 각 단어의 첫 글자를 대문자로 한다.

- ③번은 식별자의 의미를 파악하려고 만든 규칙

```
i love you → iLoveYou
i am a boy → iAmABoy
create server → createServer
```

■ 식별자

- 식별자를 구분하는 방법: 변수와 메서드
- 식별자 뒤에 괄호가 있으면 이 식별자를 메서드라고 부르고 그 이외의 것을 변수라고 부름

```
Console.WriteLine("Hello C# Programming");//메서드
Math.PI;//변수
Math.Floor(10.1);//메서드
Console.BackgroundColor//변수
```

- 첫 번째, 세 번째 식별자(WriteLine과 Floor)는 괄호가 있으므로 메서드
- 두 번째와 네 번째 식별자 (PI와 BackgroundColor)는 변수
- 메서드의 괄호 안에 넣는 것을 매개변수(Parameter) 라고 부름

■ 주석

- 주석은 프로그램의 진행에 전혀 영향을 주지 않는 코드로, 프로그램을 설명하는 데 사용하는 코드
- C#가 지원하는 두 가지 주석 처리 방법
  - //을 사용해 한 줄만 주석으로 처리
  - 둘째는 /*와 */를 사용해 여러 줄을 주석으로 처리하는 방법

표 2-3 주석 처리 방법

방법	표현
한 줄 주석 처리	// 주석
여러 줄 주석 처리	/* 주석 주석 */

■ 주석

- 다음 예시에서 Console.WriteLine( ) 메서드를 실행했지만 주석으로 처리했으므로 모두 무시됨

```
// 주석은 코드의 실행에 영향을 주지 않습니다.
/*
Console.WriteLine("C# Programming");
Console.WriteLine("C# Programming");
Console.WriteLine("C# Programming");
*/
```

■ 출력 방법

- 방법1 : Console 클래스의 WriteLine ( ) 메서드 사용

Console.WriteLine(출력하고_싶은_대상);

그림 2-1 Console.WriteLine() 메서드의 형태

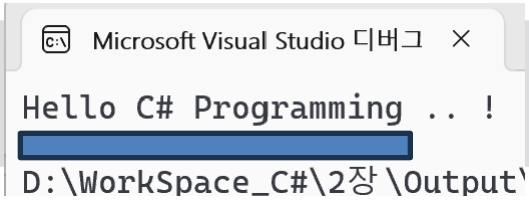
- **기본예제 2-1** C# 기본 출력 익히기 /2장/Output

코드 2-1 Hello 예제

```
01  static void Main(string[] args)
02  {
03      Console.WriteLine("Hello C# Programming .. !");
04  }
```

실행 결과

```
Hello C# Programming .. !
```



■ 정수

- 정수: 273, 52, -103, 0처럼 소수점이 없는 숫자
- C#은 정수를 다음과 같은 방법으로 생성

코드 2-2 정수 생성 /2장/DefaultData

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine(52);
04 }
```

52

■ 정수

▪만든 정수는 다음과 같은 사칙 연산자로 연산할 수 있음

표 2-4 기본적인 사칙 연산자

연산자	설명
+	덧셈 연산
–	뺄셈 연산
*	곱셈 연산
/	나눗셈 연산

코드 2-3 정수 덧셈 연산자 /2장/DefaultData

```
01 static void Main(string[] args)
02 {
03     // 325를 출력합니다.
04     Console.WriteLine(52 + 273);
05 }
```

```
325
```

■ 정수

- 연산을 할 때는 연산자 우선순위를 고려

코드 2-4 연산자 우선순위 /2장/DefaultData

```
01  static void Main(string[] args)
02  {
03      // 결과를 예측해봅시다.
04      Console.WriteLine(5 + 3 * 2);
05  }
```

11

■ 정수

- 정수를 사용하여 나머지 연산을 할 때는 좌변을 우변으로 나눈 나머지를 표시함

표 2-5 나머지 연산자

연산자	설명
%	나머지 연산자

코드 2-5 나머지 연산자 /2장/DefaultData

```
01  static void Main(string[] args)
02  {
03      Console.WriteLine(10 % 5);
04      Console.WriteLine(7 % 3);
05  }
```

```
0
1
```

■ 정수

- C#은 정수끼리 연산하면 결과도 정수로 나옴
- 나누었을 때에 소수점이 생기는 부분은 모두 사라짐
- 예를 들어 10/4는 2.5가 아니라 2

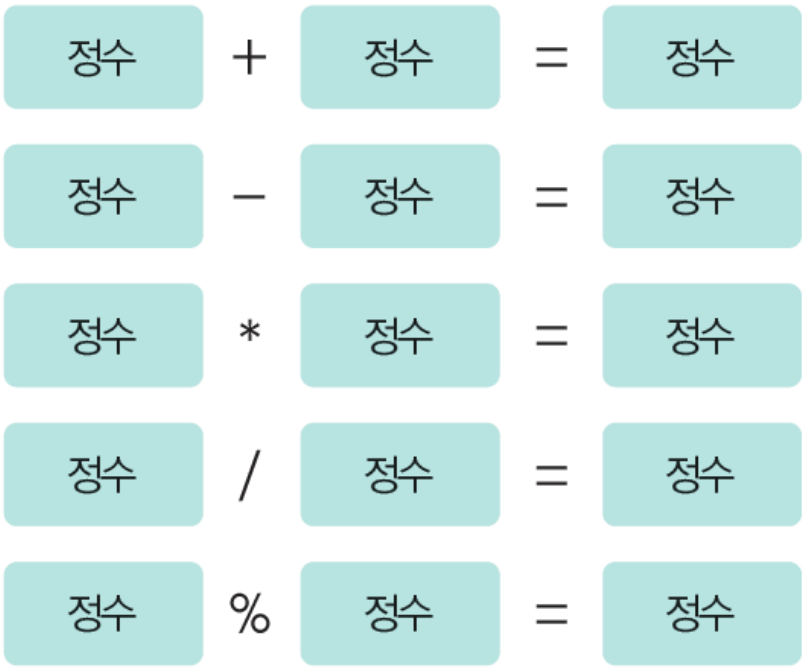


그림 2-2 정수 연산 결과

■ 정수

▪기본예제 2-2 정수와 연산자 /2장/IntegerBasic

코드 2-6 정수와 연산자

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine(1 + 2);
04     Console.WriteLine(1 - 2);
05     Console.WriteLine(1 * 2);
06     Console.WriteLine(1 / 2);
07     Console.WriteLine(1 % 2);
08 }
```

실행 결과

```
3
-1
2
0
1
```

■ 나머지 연산자와 부호

- 나머지 연산자의 부호는 왼쪽 피연산자의 부호를 따름

코드 2-7 음수와 나머지 연산자 /2장/DefaultData

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine(4 % 3);
04     Console.WriteLine(-4 % 3);
05     Console.WriteLine(4 % -3);
06     Console.WriteLine(-4 % -3);
07 }
```

실행 결과

```
1
-1
1
-1
```

■ 실수

- 실수를 만들려면 다음과 같이 소수점(.)을 사용

코드 2-8 실수 /2장/DefaultData

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine(52.273);
04 }
```

```
52.273
```

- 소수점이 들어가는 것과 안 들어가는 것은 차이가 굉장히 큼

코드 2-9 정수와 실수 /2장/DefaultData

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine(0);      // 정수입니다.
04     Console.WriteLine(0.0);    // 실수입니다.
05 }
```

```
0
0
```

■ 실수

▪ 기본예제 2-3 실수와 사칙 연산자 /2장/RealNumberBasic

코드 2-10 실수와 사칙 연산자

```
01  static void Main(string[] args)
02  {
03      Console.WriteLine(1.0 + 2.0);
04      Console.WriteLine(1.0 - 2.0);
05      Console.WriteLine(1.0 * 2.0);
06      Console.WriteLine(1.0 / 2.0);
07  }
```

실행 결과

```
3
-1
2
0.5
```

■ 문자

- 문자(Character): 글자 하나를 나타내는 자료형. 작은따옴표(')로 글자를 감싸서 만듦

'a'

그림 2-3 문자 표현

- C#은 알파벳뿐만 아니라 한국어, 중국어, 일본어, 태국어, 아랍어 등의 문자를 모두 표현할 수 있음

■ 문자

▪기본예제 2-4 문자

/2장/CharacterBasic

코드 2-12 문자

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine('A');
04     Console.WriteLine('가');
05     Console.WriteLine('나');
06 }
```

실행 결과

```
A
가
나
```

■ 문자열

- 문자열: 문자의 집합, 큰따옴표(")를 사용해 문자열을 생성
- 예: "abcdefg", "Hello World", "안녕하세요"가 문자열에 해당

코드 2-13 문자열 /2장/DefaultData

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine("안녕하세요");
04 }
```

■ 문자열

- 문자열 내에서 큰따옴표를 사용하고 싶을 때는 문자열 내부에 ₩"를 사용
- 이를 이스케이프 문자라고 부르며 특수 기능을 수행

표 2-6 자주 사용되는 이스케이프 문자

이스케이프 문자	설명	이스케이프 문자	설명
\t	수평 탭	\\	역 슬래시
\n	행 바꿈	\"	큰따옴표

■ 문자열

▪기본예제 2-5 이스케이프 문자 /2장/EscapeCharacter

- 이스케이프 문자 중에 ₩t, ₩n, ₩"를 사용

코드 2-14 이스케이프 문자

```
01  static void Main(string[] args)
02  {
03      Console.WriteLine("한빛\t아카데미");
04      Console.WriteLine("한빛\n아카데미");
05      Console.WriteLine("\"\"\"");
06  }
```

실행 결과

```
한빛    아카데미
한빛
아카데미
"""
```

■ 문자열

- 문자열은 문자열 연결 연산자를 사용해 합칠 수도 있음

표 2-7 문자열 연결 연산자

연산자	설명
+	문자열 연결 연산자

- **기본예제 2-6** 문자열 연결 연산자 /2장/StringConnection

코드 2-15 문자열 연결 연산자

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine("가나다" + "라마" + "바사아" + "자차카타" + "파하");
04 }
```

실행 결과

```
가나다라마바사아자차카타파하
```

■ 문자열

▪기본예제 2-7 문자 선택

/2장/StringSelector

표 2-8 문자 선택 괄호

연산자	설명
문자열[숫자]	문자 선택 괄호

▪"안녕하세요"라는 문자열의 0번째, 1번째, 3번째에 있는 문자를 선택

코드 2-16 문자 선택

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine("안녕하세요"[0]);
04     Console.WriteLine("안녕하세요"[1]);
05     Console.WriteLine("안녕하세요"[3]);
06 }
```

실행 결과

```
안
녕
세
```

여기서 잠깐!

■ 예외

- 문자열은 5자인데 100번째 글자를 선택하면?

코드 2-17 예외 /2장/DefaultData

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine("안녕하세요"[100]);
04 }
```

- 디버그 모드에서 코드를 실행하면 다음과 같이 오류가 발생

그림 2-4 예외 발생(디버그 모드)

여기서 잠깐!

■ 예외

- 릴리즈 모드에서 코드를 실행하면 다음과 같이 "처리되지 않은 예외"라는 오류가 나오면서 프로그램이 멈춤

실행 결과

```
Unhandled exception. System.IndexOutOfRangeException: Index was outside the
bounds of the array.
   at System.String.get_Chars(Int32 index)
   at DefaultData.Program.Main(String[] args) in C:\Users\arin\source\repos\
DefaultData\DefaultData\Program.cs:line 7

C:\Users\arin\source\repos\DefaultData\DefaultData\bin\Debug\net8.0\DefaultData.
exe(프로세스 23924개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면
자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

- 이렇게 코드를 실행하는 중에 발생하는 오류를 예외Exception 또는 런타임 오류Runtime Error오류라고 부름

여기서 잠깐!

■ 문자 덧셈 연산

- 문자도 연결되지 않을까 생각하고 다음과 같은 코드를 작성하는 사람도 있을 것

코드 2-18 문자 덧셈 /2장/DefaultData

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine('가' + '힣');
04 }
```

- 문자는 + 연산자를 사용해도 연결되지 않음

실행 결과

```
99235
```

여기서 잠깐!

```
Console.WriteLine("가" + "힣");
```

실행 결과

99235

	0	1	2	3	4	5
AC0	가	각	갂	갃	간	갅

	0	1	2	3
D70	휀	휁	휂	휃
D71	휐	휑	휒	휓
D72	휔	휕	휖	휗
D73	휘	휙	휚	휛
D74	휜	휝	휞	휟
D75	흐	흑	흒	흓
D76	흠	흡	흢	흣
D77	흰	흱	흲	흳
D78	흴	흵	흶	흷
D79	흸	흹	흺	흻
D7A	흼	흽	흾	흿

44032 + 55203 = 99235

AC00 = (A * 16^3) + (C * 16^2) + (0 * 16^1) + (0 * 16^0)
= (10 * 16^3) + (12 * 16^2) + (0 * 16^1) + (0 * 16^0)
= 44032

D7A3 = (D * 16^3) + (7 * 16^2) + (A * 16^1) + (3 * 16^0)
= (13 * 16^3) + (7 * 16^2) + (10 * 16^1) + (3 * 16^0)
= 55203

■ 불

- 불은 참과 거짓을 표현할 때에 사용함
- True와 false라는 두 가지 값밖에 없음

코드 2-19 불 /2장/DefaultData

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine(true);
04     Console.WriteLine(false);
05 }
```

```
True
False
```

■ 불

- C# 은 크기를 비교하는 연산자로 참과 거짓을 만듭니다. 이러한 연산자를 비교 연산자라고 부름

표 2-9 비교 연산자

연산자	설명
==	같다
!=	다르다
〉	왼쪽 피연산자가 크다
〈	오른쪽 피연산자가 크다
〉=	왼쪽 피연산자가 크거나 같다
〈=	오른쪽 피연산자가 크거나 같다

■ 불

▪기본예제 2-8 불과 비교 연산자 /2장/BoolBasic

코드 2-20 불과 비교 연산자

```
01  static void Main(string[] args)
02  {
03      Console.WriteLine(52 < 273);
04      Console.WriteLine(52 > 273);
05  }
```

실행 결과

```
True
False
```

■ 불

- 비교 연산자를 사용하면 불을 만들 수 있고, 불끼리는 논리 연산자를 사용할 수 있음
- C#는 다음의 세 가지 논리 연산자를 제공함

표 2-10 논리 연산자

연산자	설명
!	논리 부정 연산자
||	논리합 연산자
&&	논리곱 연산자

■ 불

- 논리 부정 연산자는 숫자의 부호를 반대로 만드는 - 연산자와 같은 형태로 사용
- **기본예제 2-9** 논리 부정 연산자 /2장/LogicalNot

코드 2-21 논리 부정 연산자

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine(!true);
04     Console.WriteLine(!false);
05     Console.WriteLine(!(52 < 273));
06     Console.WriteLine(!(52 > 273));
07 }
```

실행 결과

```
False
True
False
True
```

■ 불

논리합(||) 연산자(Logical OR):

두 개의 피연산자 중에 하나만 true이면 전체가 true

A	B	A || B
true	true	true
true	false	true
false	true	true
false	false	false

■ 불

- 논리곱(&&) 연산자 (Logical AND):

  두 개의 피연산자 모두 true이어야만 전체가 true

A	B	A && B
true	true	true
true	false	false
false	true	false
false	false	false

■ 불

- 논리 연산자가 가장 많이 사용되는 경우는 범위 판단과 관련된 부분
- 예를 들어 3시와 8시 사이라는 조건을 표현할 때, 3 < 현재 시각 < 8이라고 쓰면 오류 발생.

```
int number = 10;
Console.WriteLine(3 < number < 8);
```

(지역 변수) int number

CS0019: '<' 연산자는 'bool' 및 'int' 형식의 피연산자에 적용할 수 없습니다.

그림 2-5 연속된 범위 연산 오류

- C#은 이러한 범위를 표현할 때 반드시 범위 연산자를 사용해야 함

■ 불

- 다음과 같은 범위는 x < 3 또는 x > 8이라고 표현
- 논리합 연산자는 이렇게 나뉜 두 가지 범위를 모두 선택할 때 사용

그림 2-6 x < 3 또는 8 < x의 범위

- 논리곱 연산자는 다음과 같이 두 가지 범위로 한꺼번에 선택되는 부분을 찾을 때에 사용

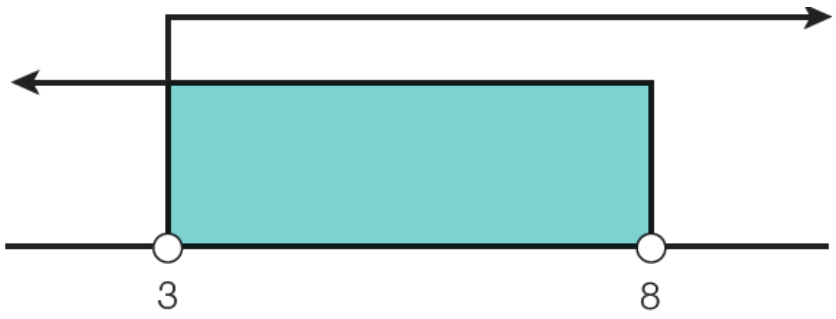


그림 2-7 x > 3 그리고 x < 8의 범위

■ 불

▪기본예제 2-10 불과 논리 연산자 /2장/LogicalOperator

코드 2-22 불과 논리 연산자

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine(DateTime.Now.Hour < 3 || 8 < DateTime.Now.Hour);
04     Console.WriteLine(3 < DateTime.Now.Hour && DateTime.Now.Hour < 8);
05 }
```

실행 결과

```
False
True
```

■ 변수

- 변수: 값을 저장할 때 사용하는 식별자.
- 변수는 다음 두 단계를 거쳐 사용할 수 있음
  - 변수를 선언한다.
  - 변수에 값을 할당한다.
- 변수를 만드는 것을 '변수를 선언한다'고 표현하고, 그러한 변수에 값을 넣는 것을 '변수에 값을 할당한다'라고 표현

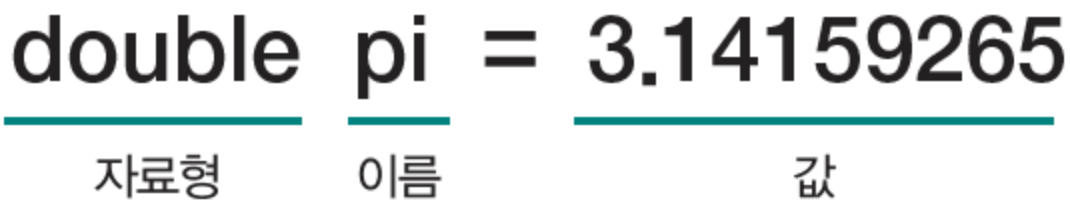
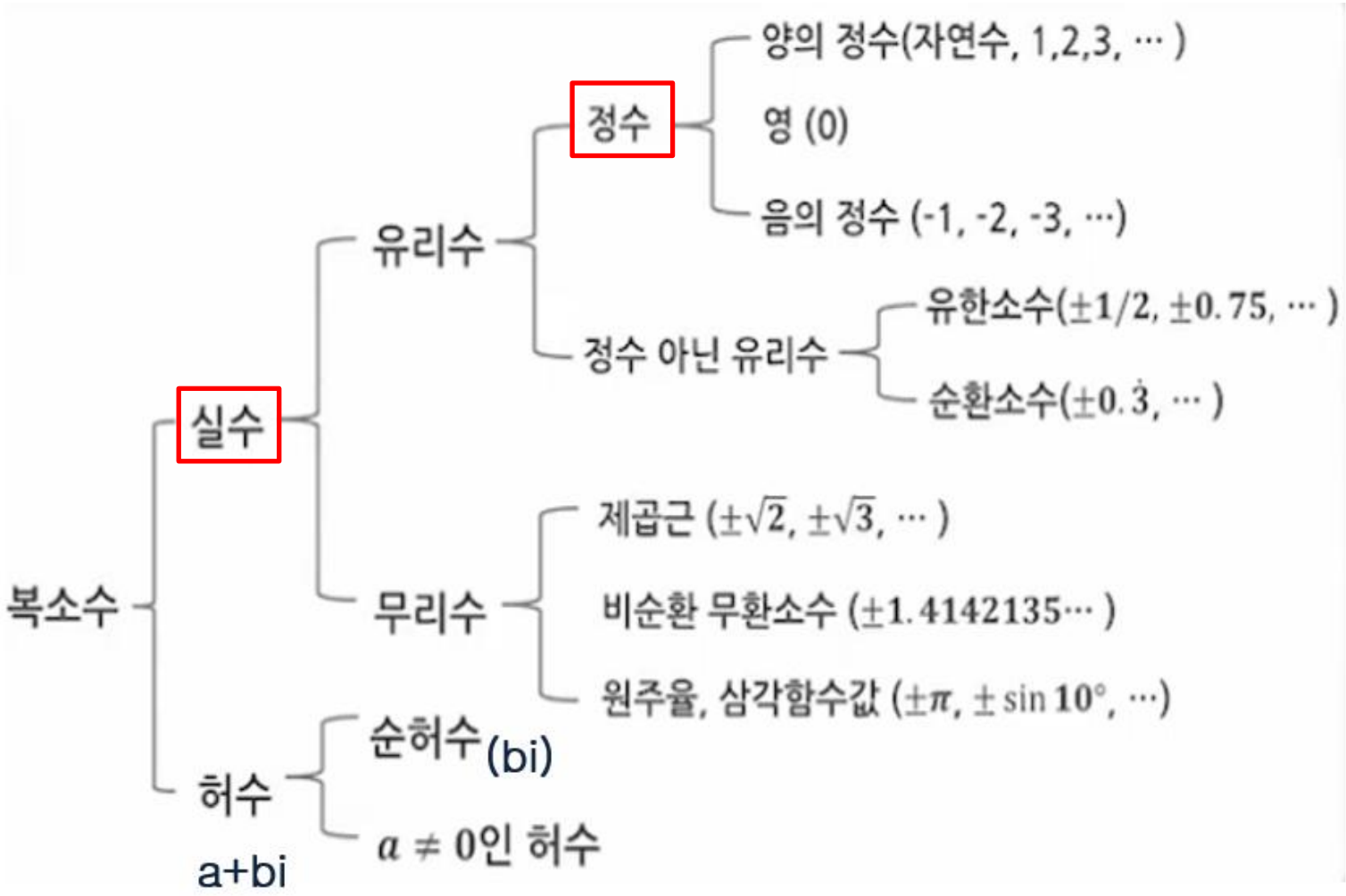


그림 2-8 변수 선언 예

■ 정수 & 실수 자료형


복소수
실수
유리수
정수
양의 정수(자연수, 1,2,3, …)
영 (0)
음의 정수 (-1, -2, -3, …)
정수 아닌 유리수
유한소수(±1/2, ±0.75, …)
순환소수(±0.3, …)
무리수
제곱근 (±√2, ±√3, …)
비순환 무환소수 (±1.4142135…)
원주율, 삼각함수값 (±π, ± sin 10°, …)
허수
a+bi
순허수(bi)
a ≠ 0인 허수

■ 정수 자료형

- C#은 관련 자료형이 정말 많이 있지만 일반적으로 [표 2-13]의 정수 자료형을 가장 많이 사용

표 2-13 정수 자료형

키워드	설명
int	4바이트의 정수
long	8바이트의 정수

1 Byte = 8 Bit
4 Byte = 32 Bit
8Byte = 64 Bit
1 Bit : 0, 1 / 8 Bit = 2^8 /
32 Bit = 2^{32} / 64 Bit = 2^{64}

- 이러한 자료형을 사용해서 다음과 같이 정수 변수를 만들 수 있음

```
int a = 10
long b = 20
```

그림 2-9 정수 자료형 선언 예

■정수 자료형

▪기본예제 2-11 정수 변수 생성

/2장/IntegerVariable

코드 2-23 정수 변수 생성

```
01 static void Main(string[] args)
02 {
03     int a = 273;
04     int b = 52;
05
06     Console.WriteLine(a + b);
07     Console.WriteLine(a - b);
08     Console.WriteLine(a * b);
09     Console.WriteLine(a / b);  //5.25
10     Console.WriteLine(a % b);
11 }
```

실행 결과

```
325
221
14196
5
13
```

■ 정수 자료형

- 컴퓨터는 [그림 2-10]과 [그림 2-11]처럼 정수를 2진수로 나타냄

0000 0000 0000 0000 0000 0000 0000 0001

그림 2-10 1의 2진수 표현

0000 0000 0000 0000 0000 0000 0000 0100

그림 2-11 4의 2진수 표현

- 4바이트(32비트)인 int 자료형은 2^{32}개의 숫자

(-2,147,483,648~2,147,483,647)만 나타낼 수 있음

■ 정수 자료형

코드 2-24 오버플로 /2장/Variables

```
01 static void Main(string[] args)
02 {
03     int a = 2147483640;
04     int b = 52273;
05     Console.WriteLine(a + b);
06 }
```

- 현실 세계의 수학이라면 21475359l3이 나오지만 C#은 음수를 출력

실행 결과

```
-2147431383
```

■ 정수 자료형

- 2,147,483,647에 1을 더하면 int 자료형의 범위를 넘어버리면서 -2,147,483,648이 됨
- 이러한 현상을 오버플로(Overflow)라고 부름

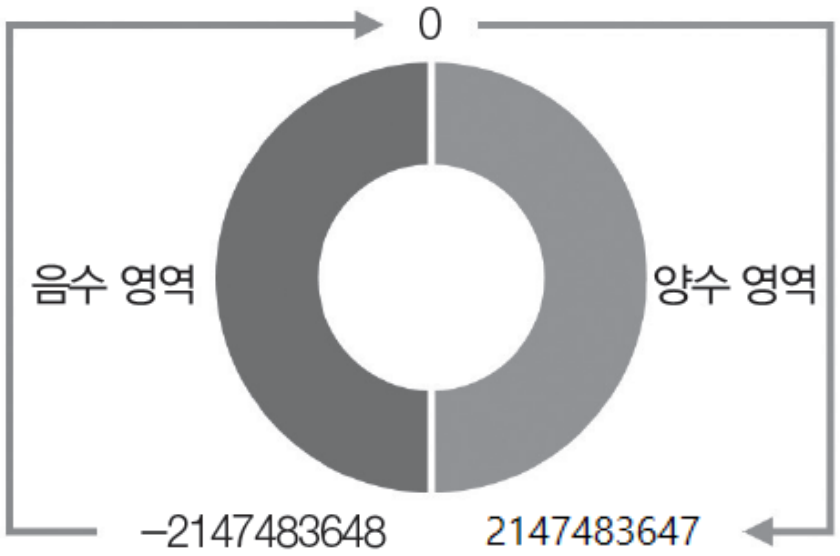


그림 2-12 int 자료형의 범위

■정수 자료형

▪기본예제 2-12 오버플로

/2장/Overflow

코드 2-25 오버플로

```
01 static void Main(string[] args)
02 {
03     int a = 2000000000;
04     int b = 1000000000;
05     Console.WriteLine(a + b);
06 }
```

실행 결과

```
-1294967296
```

■정수 자료형

▪기본예제 2-12 오버플로

/2장/Overflow

코드 2-26 자료형 변환을 사용한 해결 방법

```
01 static void Main(string[] args)
02 {
03     uint unsignedA = 2000000000;
04     uint unsignedB = 1000000000;
05     Console.WriteLine(unsignedA + unsignedB);
06 }
```

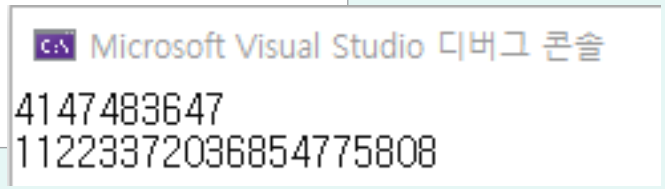
여기서 잠깐!

■ unsigned 자료형

- int 자료형은 -2,147,483,648부터 2,147,483,647까지 나타낼 수 있음
- 음수를 사용하지 않는다면 -2,147,483,648부터 -1까지의 숫자가 낭비됨
- 이런 낭비를 막고자 C#은 unsigned 자료형(부호가 없는 자료형)을 제공
- int와 long 키워드 대신 uint와 ulong 키워드를 사용

코드 2-27 uint와 ulong 자료형 /2장/Variables

```
01 static void Main(string[] args)
02 {
03     uint unsignedInt = 4147483647;
04     ulong unsignedLong = 11223372036854775808;
05
06     Console.WriteLine(unsignedInt);
07     Console.WriteLine(unsignedLong);
08 }
```



MaxValue와 MinValue

- C#은 int.MaxValue와 int.MinValue라는 코드로 손쉽게 최댓값과 최솟값을 알아낼 수 있음

코드 2-28 int 자료형의 최댓값과 최솟값 /2장/Variables

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine(int.MaxValue);
04     Console.WriteLine(int.MinValue);
05 }
```

여기서 잠깐!

■ MaxValue와 MinValue

- long 자료형의 최댓값과 최솟값은 다음과 같은 방법으로 알아낼 수 있음

코드 2-29 long 자료형의 최댓값과 최솟값 /2장/Variables

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine(long.MaxValue);
04     Console.WriteLine(long.MinValue);
05 }
```

```
Microsoft Visual Studio 디버그
9223372036854775807
-9223372036854775808
```

■실수 자료형

- 실수 변수를 만들 때는 [표 2-14]의 자료형을 사용

표 2-14 실수 자료형

키워드	설명
float	4바이트의 실수
double	8바이트의 실수

■실수 자료형

▪기본예제 2-13 실수 변수 생성 /2장/RealNumberVariable

코드 2-30 실수 변수 생성

```
01 static void Main(string[] args)
02 {
03     double a = 52.273;
04     double b = 103.32;
05
06     Console.WriteLine(a + b);
07     Console.WriteLine(a - b);
08     Console.WriteLine(a * b);
09     Console.WriteLine(a / b);
10 }
```

실행 결과

```
155.593
-51.047
5400.84636
0.50593302361595
```

■문자 자료형

- 문자 변수를 만들 때는 [표 2-15]의 자료형을 사용

표 2-15 문자 자료형

키워드	설명
char	문자

■문자 자료형

▪기본예제 2-14 문자 변수 생성

/2장/CharacterVariable

코드 2-31 문자 변수 생성

```
01 static void Main(string[] args)
02 {
03     char a = 'a';
04     Console.WriteLine(a);
05 }
```

실행 결과

```
a
```

여기서 잠깐!

■ sizeof 연산자와 char 자료형의 크기

- C#에서 char 자료형은 2바이트(sizeof 연산자로 확인)

sizeof(자료형)

그림 2-13 sizeof 연산자 형태

코드 2-32 sizeof 연산자 /2장/Variables

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine("int: " + sizeof(int));
04     Console.WriteLine("long: " + sizeof(long));
05     Console.WriteLine("float: " + sizeof(float));
06     Console.WriteLine("double: " + sizeof(double));
07     Console.WriteLine("char: " + sizeof(char));
08 }
```

실행 결과

```
int: 4
long: 8
float: 4
double: 8
char: 2
```

여기서 잠깐!

■ 문자 자료형과 연산자

- 문자 자료형은 문자열 자료형보다 정수 자료형에 가까움

코드 2-33 문자 자료형과 연산자 /2장/Variables

```
01 static void Main(string[] args)
02 {
03     char a = 'a'; //아스키 코드: 97
04     char b = 'b'; //아스키 코드: 98
05
06     Console.WriteLine(a + b);
07     Console.WriteLine(a - b);
08     Console.WriteLine(a * b);
09     Console.WriteLine(a / b);
10     Console.WriteLine(a % b);
11 }
```

아스키코드

097	a
098	b
099	c
100	d

실행 결과

```
195
-1
9506
0
97
```

■문자열 자료형

▪문자열 변수를 선언할 때는 [표 2-16]의 자료형을 사용

표 2-16 문자열 자료형

키워드	설명
string	문자열 자료형

■문자 자료형

▪기본예제 2-15 문자열 변수 생성 /2장/StringVariable

코드 2-34 문자열 변수 생성

```
01 static void Main(string[] args)
02 {
03     string message = "안녕하세요";
04
05     Console.WriteLine(message + "!");
06     Console.WriteLine(message[0]);
07     Console.WriteLine(message[1]);
08     Console.WriteLine(message[2]);
09 }
```

실행 결과

```
안녕하세요!
안
녕
하
```

■ sizeof 연산자와 string 자료형

- string 자료형은 sizeof 연산자로 자료형의 크기를 구할 수 없음
- 다음 코드를 실행하면 컴파일 오류가 발생

코드 2-35 sizeof 연산자와 string 자료형 /2장/Variables

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine("string: " + sizeof(string));
04 }
```

- 문자열은 다른 기본 자료형과 다르기 때문
- 비주얼 스튜디오에서 자료형 위에 마우스를 올려놓으면 자료형의 진짜 원형을 찾을 수 있음

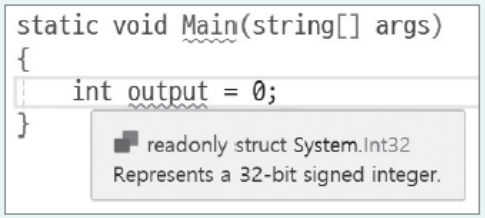


그림 2-14 int 자료형의 원형은 struct System.Int32 구조체

■ sizeof 연산자와 string 자료형

- 기본 자료형들의 원형은 [표 2-17]과 같음

표 2-17 기본 자료형의 원형

기본 자료형	원형	기본 자료형	원형
int	struct System.Int32	float	struct System.Single
long	struct System.Int64	double	struct System.Double
char	struct System.Char	bool	struct System.Boolean
string	class System.String		

■불 자료형

- 불 변수를 선언할 때는 [표 2-18]의 자료형을 사용

표 2-18 불 자료형

키워드	설명
bool	불 자료형

■불 자료형

▪기본예제 2-16 불 변수 생성 /2장/BoolVariable

코드 2-36 불 변수 생성

```
01  static void Main(string[] args)
02  {
03      bool one = 10 < 0;
04      bool other = 20 > 100;
05
06      Console.WriteLine(one);
07      Console.WriteLine(other);
08  }
```

실행 결과

```
False
False
```

05 복합 대입 연산자

■복합 연산자

- 복합 연산자: 자료형에 적용하는 기본 연산자와 = 연산자를 함께 사용해 구성하는 연산자
- a += 10이라고 사용하면 a = a + 10이라고 하는 것과 같은 결과를 냄
- 숫자에 적용할 수 있는 다른 연산자들도 같은 방법으로 사용

표 2-19 숫자에 적용하는 복합 대입 연산자

연산자	설명
+=	숫자 덧셈 후 대입 연산자
-=	숫자 뺄셈 후 대입 연산자
*=	숫자 곱셈 후 대입 연산자
/=	숫자 나눗셈 후 대입 연산자
%=	숫자 나머지 연산 후 대입 연산자

■복합 연산자

- 문자열도 마찬가지로 +(문자열 연결 연산자)와 = 연산자를 합쳐 [표 2-20]의 복합 대입 연산자를 사용할 수 있음

표 2-20 문자열에 적용하는 복합 대입 연산자

연산자	설명
+=	문자열 연결 후 대입 연산자

■복합 연산자

▪기본예제 2-17 숫자와 관련된 복합 대입 연산자/2장/AssignmentOperator

- 정수 자료형의 변수 output을 만들고 복합 대입 연산자를 적용

코드 2-37 숫자와 관련된 복합 대입 연산자

```
01 static void Main(string[] args)
02 {
03     int output = 0;
04     output += 52;
05     output += 273;
06     output += 103;
07
08     Console.WriteLine(output);
09 }
```

실행 결과

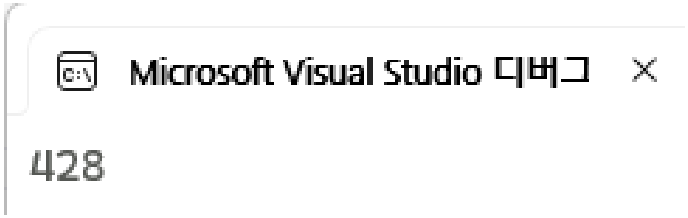
```
428
```

■복합 연산자

▪기본예제 2-17 숫자와 관련된 복합 대입 연산자/2장/AssignmentOperator

코드 2-38 숫자와 관련된 복합 대입 연산자의 다른 방식

```
01 static void Main(string[] args)
02 {
03     int output = 0;
04     output = output + 52;
05     output = output + 273;
06     output = output + 103;
07
08     Console.WriteLine(output);
09 }
```



■복합 연산자

▪기본예제 2-18 문자열과 관련된 복합 대입 연산자 /2장/StringAssignmentOperator

- 문자열 자료형의 변수 output을 만들고 복합 대입 연산자를 사용

코드 2-39 문자열과 관련된 복합 대입 연산자

```
01 static void Main(string[] args)
02 {
03     string output = "hello ";
04     output += "world ";
05     output += "!";
06
07     Console.WriteLine(output);
08 }
```

실행 결과

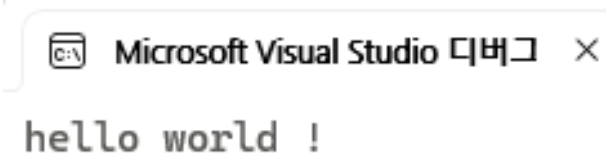
```
hello world !
```

■복합 연산자

▪ **기본예제 2-18** 문자열과 관련된 복합 대입 연산자 /2장/StringAssignmentOperator

코드 2-40 문자열과 관련된 복합 대입 연산자 예제 풀어쓰기

```
01 static void Main(string[] args)
02 {
03     string output = "hello ";
04     output = output + "world ";
05     output = output + "!";
06
07     Console.WriteLine(output);
08 }
```



■증감 연산자

- 증감 연산자는 단항 연산자로 변수앞과 뒤에 ++ 기호와 -- 기호를 붙여 만듦

표 2-21 증감 연산자

연산자	설명
[변수]++	기존 변수의 값에 1을 더합니다(후위).
++[변수]	기존 변수의 값에 1을 더합니다(전위).
[변수]--	기존 변수의 값에서 1을 뺍니다(후위).
--[변수]	기존 변수의 값에서 1을 뺍니다(전위).

■증감 연산자

▪기본예제 2-19 증감 연산자

/2장/IncrementOperator

- 변수 number를 초기화하고 ++ 연산자와 -- 연산자를 사용

코드 2-41 증감 연산자

```
01 static void Main(string[] args)
02 {
03     int number = 10;
04     number++;
05     Console.WriteLine(number);
06     number--;
07     Console.WriteLine(number);
08 }
```

실행 결과

```
11
10
```

- 한 줄에 독립적으로 증감 연산자를 사용할 때는 전위와 후위의 차이를 알 수 없음

■증감 연산자

▪기본예제 2-20 증감 연산자의 전위와 후위 /2장/IncrementOperatorPosition

- 증감 연산자의 전위 형태와 후위 형태를 사용

코드 2-42 증감 연산자의 후위 형태

```
01 static void Main(string[] args)
02 {
03     int number = 10;
04     Console.WriteLine(number);
05     Console.WriteLine(number++);
06     Console.WriteLine(number--);
07     Console.WriteLine(number);
08 }
```

실행 결과

```
10
10
11
10
```

■증감 연산자

▪기본예제 2-20 증감 연산자의 전위와 후위 /2장/IncrementOperatorPosition

코드 2-43 증감 연산자의 전위 형태

```
01 static void Main(string[] args)
02 {
03     int number = 10;
04     Console.WriteLine(number);

05     Console.WriteLine(++number);
06     Console.WriteLine(--number);
07     Console.WriteLine(number);
08 }
```

실행 결과

```
10
11
10
10
```

■증감 연산자

▪기본예제 2-20 증감 연산자의 전위와 후위 /2장/IncrementOperatorPosition

- 후위는 문장을 실행한 이후에 값을 변경하라는 의미
- Console.WriteLine(number++) 코드는 Console.WriteLine(number)를 실행하고 변수 number에 1을 더함

코드 2-44 후위 증감 연산자 /2장/Operators

```
01  static void Main(string[] args)
02  {
03      int number = 10;
04      Console.WriteLine(number);
05      Console.WriteLine(number); number += 1;————Console.WriteLine(number++)
06      Console.WriteLine(number); number -= 1;————Console.WriteLine(number--)
07      Console.WriteLine(number);
08  }
```

■증감 연산자

▪기본예제 2-20 증감 연산자의 전위와 후위 /2장/IncrementOperatorPosition

- 해당 문장을 실행하기 전에 값을 더하는 것이 전위

코드 2-45 증감 연산자 이해도 확인 /2장/Operators

```
01 static void Main(string[] args)
02 {
03     int number = 10;
04     Console.WriteLine(number++);
05     Console.WriteLine(++number);
06     Console.WriteLine(number--);
07     Console.WriteLine(--number);
08 }
```

- 코드를 실행하면 차례대로 10, 12, 12, 10을 출력

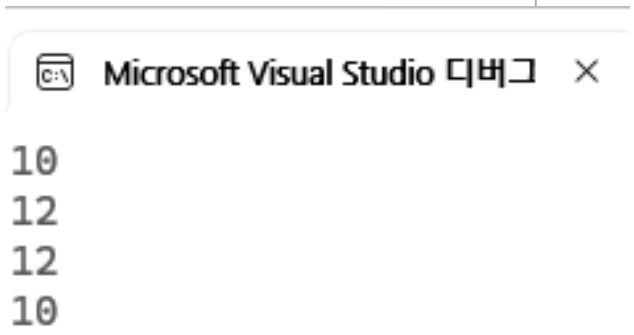
■증감 연산자

▪기본예제 2-20 증감 연산자의 전위와 후위 /2장/IncrementOperatorPosition

- 특별한 이유가 아니라면 한 줄에 하나의 증감 연산자만 독립적으로 사용하기 바람

코드 2-46 여러 줄로 나누어 적은 증감 연산자 /2장/Operators

```
01 static void Main(string[] args)
02 {
03     int number = 10;
04     Console.WriteLine(number);
05     number++;
06     number++;
07     Console.WriteLine(number);
08     Console.WriteLine(number);
09     number--;
10     number--;
11     Console.WriteLine(number);
12 }
```



■자료형 출력

- 자료형을 확인하는 가장 쉬운 방법은 해당 변수에 마우스 커서를 가져다 대는 것

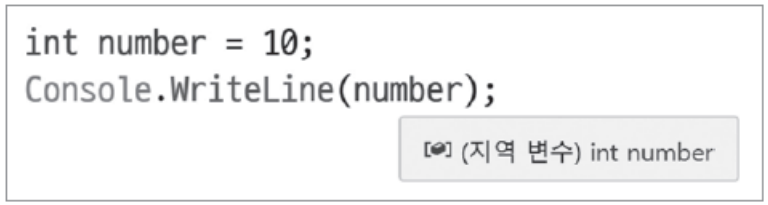


그림 2-15 마우스를 사용한 자료형 확인

- 프로그램 내부에서 자료형을 출력하고 싶은 경우에는 [표 2-22]의 GetType( ) 메서드를 사용

표 2-22 자료형 확인 메서드

메서드	설명
GetType()	해당 변수의 자료형을 추출합니다.

■자료형 출력

▪기본예제 2-21 GetType( ) 메서드 활용

/2장/TypeCheck

- GetType( ) 메서드를 활용해서 자료형을 출력

코드 2-47 GetType() 메서드 활용

```
01 static void Main(string[] args)
02 {
03     // 변수를 선언합니다.
04     int _int = 273;
05     long _long = 522731033265;
06     float _float = 52.273F;
07     double _double = 52.273;

08     char _char = '글';
09     string _string = "문자열";
10
11     // 출력합니다.
12     Console.WriteLine(_int.GetType());
13     Console.WriteLine(_long.GetType());
14     Console.WriteLine(_float.GetType());
15     Console.WriteLine(_double.GetType());
16     Console.WriteLine(_char.GetType());
17     Console.WriteLine(_string.GetType());
18 }
```

실행 결과

```
System.Int32
System.Int64
System.Single
System.Double
System.Char
System.String
```

■자료형 출력

▪기본예제 2-22 직접적인 GetType( ) 메서드 활용 /2장/DirectTypeCheck

- 숫자 또는 문자열에 직접 적용할 수도 있음

코드 2-48 직접적인 GetType() 메서드 활용

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine((273).GetType());
04     Console.WriteLine((522731033265L).GetType());
05     Console.WriteLine((52.273F).GetType());
06     Console.WriteLine((52.273).GetType());
07     Console.WriteLine('자').GetType());
08     Console.WriteLine("문자열").GetType());
09 }
```

실행 결과

```
System.Int32
System.Int64
System.Single
System.Double
System.Char
System.String
```

08 var 키워드

■자료형 지정

- C#에서는 [표 2-23]의 var 키워드로 변수의 자료형을 자동으로 지정

표 2-23 var 키워드

키워드	설명
var	자료형을 자동으로 지정합니다.

- 변수를 선언하면 초기화할 때에 자료형이 자동 지정

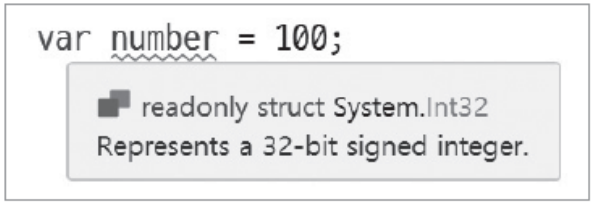


그림 2-16 var 키워드로 자료형 자동 결정

■자료형 지정

- int 자료형으로 선언된 변수를 string 자료형으로 바꾸거나 하는 것은 불가능

코드 2-50 var 키워드의 제약 /2장/VarKeyword

```
01  var number = 100;
02  number = "변경";
```

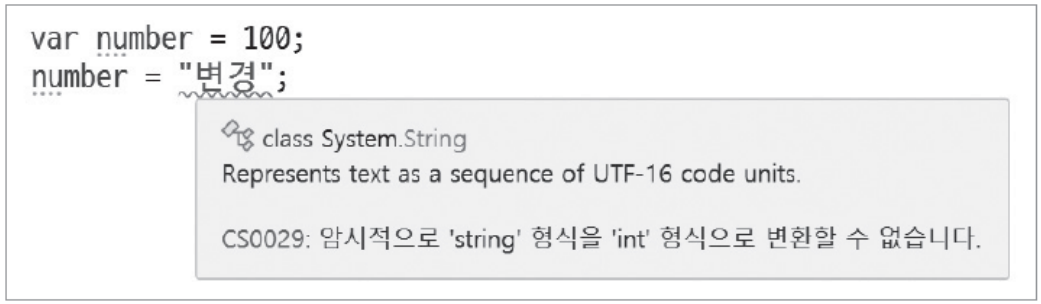


그림 2-17 var 키워드 제약

08 var 키워드

■자료형 지정

- var 키워드를 사용하려면 다음과 같은 두 가지 조건을 만족해야 함
  - 지역 변수로 선언하는 경우
  - 변수를 선언과 동시에 초기화하는 경우

■자료형 지정

- 지역변수: C#에서는 메서드 내부에 선언한 변수

코드 2-51 지역 변수 /2장/VarKeyword

```
01 class Program
02 {
03     var global = 52;──────────인스턴스 변수(var 키워드 사용 불가)입니다.
04
05     static void Main(string[] args)
06     {
07         var local = 273;──────지역 변수(var 키워드 사용 가능)입니다.
08     }
09 }
```

- 메서드 내부에 선언할 때에만 var 키워드를 적용할 수 있음

■자료형 지정

- 메서드 외부에 var 키워드를 사용하면 다음과 같이 오류가 발생

```
class Program
{
    var global = 52;
    CS0825: 상황별 키워드 'var'는 지역 변수 선언이나 스크립트 코드에만 표시할 수 있습니다.
    static void Main(string[] args)
    {

    }
}
```

그림 2-18 지역 변수가 아닌 곳에는 var 키워드 사용 불가

■자료형 지정

- var 키워드를 다음과 같이 덩그러니 써놓으면 오류가 발생

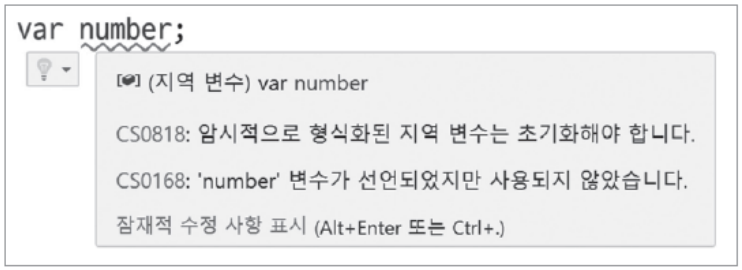


그림 2-19 var 키워드는 선언과 초기화를 함께 작성

- 다음과 같이 선언과 초기화를 함께 해야 함

코드 2-52 var 키워드의 선언과 초기화 동시 수행 /2장/VarKeyword

```
01 static void Main(string[] args)
02 {
03     var number = 20;
04 }
```

■ var 키워드로 long 자료형과 float 자료형인 변수 생성

- long 자료형, float 자료형으로 선언하고 싶다면 숫자 뒤에 L 또는 F 등의 기호를 붙여 명시적으로 해당 숫자가 어떤 자료형인지 알려줘야 함

코드 2-53 var 키워드를 사용한 다양한 자료형 선언 /2장/VarKeyword

```
01 static void Main(string[] args)
02 {
03     var numberA = 100L;       // long 자료형
04     var numberB = 100.0;      // double 자료형
05     var numberC = 100.0F;     // float 자료형
06 }
```

여기서 잠깐!

L

코드 2-54 long 자료형을 나타내는 기호: 소문자

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine(123456 + 65432l);
04 }
```

실행 결과

```
188888
```

- 오른쪽의 숫자가 654321이 아니라 65432L
- 코드를 작성할 때는 다음과 같이 long 자료형임을 나타내는 L은 대문자로 사용하기 바람

여기서 잠깐!

L

코드 2-55 long 자료형을 나타내는 기호: 대문자

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine(123456 + 65432L);
04 }
```

- 비주얼 스튜디오에서 l을 소문자로 사용하면 다음과 같은 경고도 뜸

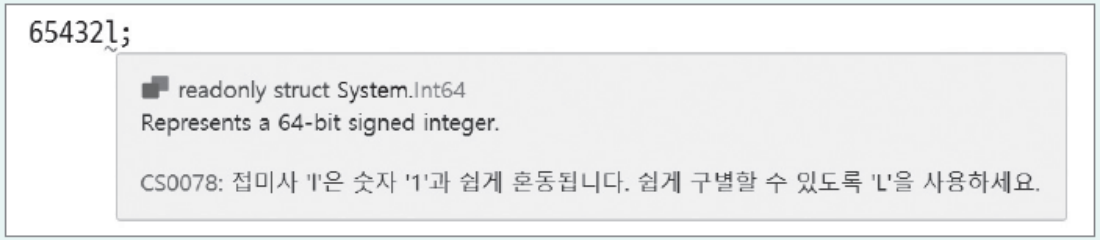


그림 2-20 소문자 경고

■입력

- C#에서 사용자의 입력을 받을 때는 [표 2-24]의 메서드를 사용

표 2-24 입력 메서드

메서드	설명
Console.ReadLine()	사용자로부터 한 줄의 문자열을 입력 받습니다.

- ReadLine( ) 메서드를 살펴보면 [그림 2-21]
- ReadLine( ) 메서드의 결과가 string? 자료형으로 나온다는 이야기

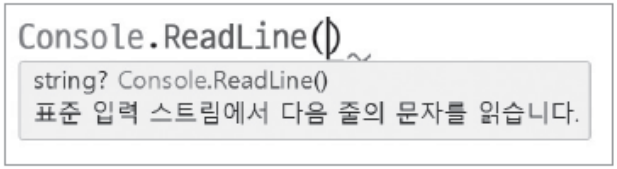


그림 2-21 입력 메서드의 반환형

■입력

▪기본예제 2-23 문자열 입력과 출력 /2장/Input

- 결과를 string 자료형의 변수 input에 넣어주고 입력 받은 문자열을 출력
- 코드를 실행한 다음 아무것이나 입력하고 엔터를 입력

코드 2-56 문자열 입력과 출력

```
01 static void Main(string[] args)
02 {
03     string input = Console.ReadLine();
04     Console.WriteLine("input: " + input);
05 }
```

실행 결과

```
아무것이나 입력
input: 아무것이나 입력
```

10 자료형 변환

- **자료형 변환(Type Casting): 한 자료형을 다른 자료형으로 바꾸는 것**
- **강제 자료형 변환**
  - 큰 자료형에서 작은 자료형으로 변환될 때는 데이터가 깨질 수 있음

코드 2-57 자료형 변환 /2장/Casts

```
01 static void Main(string[] args)
02 {
03     // long 자료형을 int 자료형으로 변환합니다.
04     long longNumber = 2147483647L + 2147483647L;
05     int intNumber = longNumber;
06     Console.WriteLine(intNumber);
07 }
```

■강제 자료형 변환

- long 자료형에 4294967294라는 숫자를 넣고 이를 int 자료형에 넣으려고 함
- int 자료형의 한계 수치는 2147483647이기 때문에 [그림 2-22]와 같은 오류를 냄

```
static void Main(string[] args)
{
    // long 자료형을 int 자료형으로 변환합니다.
    long longNumber = 2147483647L + 2147483647L;
    int intNumber = longNumber;
    Console.Writel(
}
```

(지역 변수) long longNumber

CS0266: 암시적으로 'long' 형식을 'int' 형식으로 변환할 수 없습니다. 명시적 변환이 있습니다. 캐스트가 있는지 확인하세요.

잠재적 수정 사항 표시 (Alt+Enter 또는 Ctrl+.)

그림 2-22 자료형 변환 실패

■강제 자료형 변환

- 자동으로 변환되지 않는다는 오류가 뜨지만 강제로 이를 변환할 수 있음
- 강제 자료형 변환: 강제로 자료형을 변환하는 것

코드 2-58 강제 자료형 변환 /2장/Casts

```
01 var a = (int)10.0;
02 var b = (float)10;
03 var c = (double)10;
```

■강제 자료형 변환

▪기본예제 2-24 강제 자료형 변환

/2장/ExplicitConversion

- long 자료형을 int 자료형으로 변환

코드 2-59 강제 자료형 변환

```
01  static void Main(string[] args)
02  {
03      // long 자료형을 int 자료형으로 변환합니다.
04      long longNumber = 2147483647L + 2147483647L;
05      int intNumber = (int)longNumber;
06      Console.WriteLine(intNumber);
07  }
```

실행 결과

```
-2
```

■강제 자료형 변환

▪기본예제 2-24 강제 자료형 변환 /2장/ExplicitConversion

- 데이터손실: 큰 자료형에서 작은 자료형으로 변환할 때 숫자가 넘쳐 없어질 수 있는 현상
- 다음과 같은 코드는 강제적으로 자료형을 변환해도 값이 충분히 수용되므로 데이터 손실이 일어나지 않음

코드 2-60 강제 자료형 변환의 데이터 손실 미발생 /2장/Casts

```
01 static void Main(string[] args)
02 {
03     // long 자료형을 int 자료형으로 변환합니다.
04     long longNumber = 52273;
05     int intNumber = (int)longNumber;
06     Console.WriteLine(intNumber);
07 }
```

■자동 자료형 변환

- C#은 데이터 손실이 일어나지 않는 범위에 한해서 자동으로 자료형을 변경

- **기본예제 2-25** 숫자 손상 /2장/NumberLost

  - 큰 자료

코드 2-61 숫자 손상

```
01 static void Main(string[] args)
02 {
03     // long 자료형을 int 자료형으로 변환합니다.
04     long longNumber = 2147483647L + 2147483647L;
05     int longToInt = (int)longNumber;
06     Console.WriteLine(longToInt);

07     // double 자료형을 int 자료형으로 변환합니다.
08     double doubleNumber = 52.27310332;
09     int doubleToInt = (int)doubleNumber;
10     Console.WriteLine(doubleToInt);
11 }
```

실행 결과

```
-2
52
```

■자동 자료형 변환

- int 자료형의 숫자는 long 자료형에 넣어도 절대 깨지지 않고 자동 자료형 변환이 일어남

표 2-25 자동 자료형 변환

기존 자료형	자동 변환되는 자료형
int	long, float, double
long	float, double
char	int, long, float, double
float	double

■자동 자료형 변환

▪기본예제 2-26 자동 자료형 변환 /2장/ImplicitConversion

- int 자료형을 long과 double 자료형으로 자동 자료형 변환

코드 2-62 자동 자료형 변환

```
01 static void Main(string[] args)
02 {
03     // int 자료형의 숫자를 생성합니다.
04     int intNumber = 2147483647;
05
06     // int 자료형을 long 자료형으로 자동 변환합니다.
07     long intToLong = intNumber;
08     Console.WriteLine(intToLong);
09
10     // int 자료형을 double 자료형으로 자동 변환합니다.
11     double intToDouble = intNumber;
12     Console.WriteLine(intToDouble);
13 }
```

실행 결과

```
2147483647
2147483647
```

■다른 자료형을 숫자로 변환

- 문자열과 불을 숫자 자료형으로 변환하려면 어떻게 해야 할까?

코드 2-63 문자열을 숫자로 변환 /2장/Casts

```
01 static void Main(string[] args)
02 {
03     // string 자료형을 int 자료형으로 변환합니다.
04     string numberString = "52273";
05     int intNumber = (int)numberString;
06     Console.WriteLine(intNumber);
07 }
```

- 이 코드는 오류가 발생해서 실행조차 되지 않음

■다른 자료형을 숫자로 변환

▪다른 자료형을 숫자로 변환할 때는 [표 2-26]의 메서드를 사용

표 2-26 문자열을 숫자로 변환하는 메서드

메서드	설명
int.Parse()	다른 자료형을 int 자료형으로 변경합니다.
long.Parse()	다른 자료형을 long 자료형으로 변경합니다.
float.Parse()	다른 자료형을 float 자료형으로 변경합니다.
double.Parse()	다른 자료형을 double 자료형으로 변경합니다.

■다른 자료형을 숫자로 변환

▪기본예제 2-27 문자열을 숫자로 변환 /2장/StringTo

- 다음 코드는 문자열을 int, long, float, double 자료형으로 변환

코드 2-64 문자열을 숫자로 변환

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine(int.Parse("52"));
04     Console.WriteLine(long.Parse("273"));
05     Console.WriteLine(float.Parse("52.273"));
06     Console.WriteLine(double.Parse("103.32"));
07
08     Console.WriteLine(int.Parse("52").GetType());
09     Console.WriteLine(long.Parse("273").GetType());
10     Console.WriteLine(float.Parse("52.273").GetType());
11     Console.WriteLine(double.Parse("103.32").GetType());
12 }
```

실행 결과

```
52
273
52.273
103.32
System.Int32
System.Int64
System.Single
System.Double
```

여기서 잠깐!

■ FormatException 예외

- Parse() 메서드를 사용할 때에 해당 자료형으로 변환 불가능한 문자를 넣으면 어떻게 될까?

코드 2-65 숫자로 변환할 수 없는 문자열을 변환하는 경우 /2장/Casts

```
01 Console.WriteLine(int.Parse("52.273"));
02 Console.WriteLine(int.Parse("abc"));
```

- 이러한 예외가 발생하지 않게 Parse () 메서드의 매개변수에는 변환 가능한 문자열만 넣기 바람

■다른 자료형을 문자열로 변환

- 다른 자료형을 문자열로 변환할 때는 ToString( ) 메서드를 사용

표 2-27 문자열로 변환하는 메서드

메서드	설명
ToString()	문자열로 변환합니다.

- int 자료형을 문자열로 변환하고 싶다면 다음과 같은 코드를 사용

(10).ToString()

- double 자료형을 문자열로 변환하고 싶다면 다음과 같은 코드를 사용

(52.273).ToString()

■다른 자료형을 문자열로 변환

▪기본예제 2-28 기본 자료형을 문자열로 변환 /2장

코드 2-66 기본 자료형을 문자열로 변환

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine((52).ToString());
04     Console.WriteLine((52.273).ToString());
05     Console.WriteLine('a').ToString());
06     Console.WriteLine((true).ToString());
07     Console.WriteLine((false).ToString());
08
09     Console.WriteLine((52).ToString().GetType());
10     Console.WriteLine((52.273).ToString().GetType());
11     Console.WriteLine('a').ToString().GetType());
12     Console.WriteLine((true).ToString().GetType());
13     Console.WriteLine((false).ToString().GetType());
14 }
```

실행 결과

```
52
52.273
a
True
False
System.String
System.String
System.String
System.String
System.String
```

■다른 자료형을 문자열로 변환

▪기본예제 2-29 소수점 제거 /2장/DoubleToString

- ToString( ) 메서드를 사용하면 실수의 소수점을 원하는 곳까지만 출력할 수 있음

코드 2-67 소수점 제거

```
01 static void Main(string[] args)
02 {
03     double number = 52.273103;
04     Console.WriteLine(number.ToString("0.0"));
05     Console.WriteLine(number.ToString("0.00"));
06     Console.WriteLine(number.ToString("0.000"));
07     Console.WriteLine(number.ToString("0.0000"));
08 }
```

실행 결과

```
52.3
52.27
52.273
52.2731
```

■다른 자료형을 문자열로 변환

▪기본예제 2-30 숫자와 문자열 덧셈 /2장/StringPlusNumber

- 숫자와 문자열을 더할 수 있는 모든 경우를 나열하고 출력
- 숫자와 문자열을 놓고 + 기호를 사용하면 문자열 연결 연산자로 적용됨

코드 2-68 숫자와 문자열 덧셈

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine(52 + 273);
04     Console.WriteLine("52" + 273);
05     Console.WriteLine(52 + "273");
06     Console.WriteLine("52" + "273");
07 }
```

실행 결과

```
325
52273
52273
52273
```

■ 간단한 문자열 변환

- 문자열과 다른 자료형을 더하면 문자열 연결을 수행하게 됨

코드 2-69 간단한 문자열 변환 /2장/Casts

```
01 int number = 52273;
02 string outputA = number + "";
03 Console.WriteLine(outputA);
04
05 char character = 'a';
06 string outputB = character + "";
07 Console.WriteLine(outputB);
```

- 문자는 문자열과 비슷하니까 바로 문자열로 변환할 수 있다고도 생각할 수 있는데, 불가능

```
char character = 'a';
string output = character;
```

(지역 변수) char character

CS0029: 암시적으로 'char' 형식을 'string' 형식으로 변환할 수 없습니다.

그림 2-23 문자 문자열 변환 오류

■다른 자료형을 불로 변환

- 문자열을 불 자료형으로 변환할 때는 [표 2-28]의 메서드를 사용

표 2-28 문자열을 불로 변환하는 메서드

메서드	설명
bool.Parse()	문자열을 불 자료형으로 변환합니다.

■다른 자료형을 불로 변환

▪기본예제 2-31 문자열을 불로 변환 /2장/StringToBool

코드 2-70 문자열을 불로 변환

```
01 static void Main(string[] args)
02 {
03     Console.WriteLine(bool.Parse("True"));
04     Console.WriteLine(bool.Parse("true"));
05     Console.WriteLine(bool.Parse("False"));
06     Console.WriteLine(bool.Parse("false"));
07 }
```

실행 결과

```
True
True
False
False
```

■ 음수밖에 없는 숫자

- int 자료형의 범위는 -2147483648부터 2147483647까지인데, 결국 -2147483648에 대응되는 양수가 없음
- -2147483648에는 음수 변환 연산자(-)를 붙여도 자기 자신이 됨

코드 2-71 int 자료형 최솟값의 음수

```
01 static void Main(string[] args)
02 {
03     int output = int.MinValue;
04     Console.WriteLine(-output);
05 }
```

실행 결과

```
-2147483648
```

■ 음수밖에 없는 숫자

- 다음과 같은 코드는 오류가 발생

```
static void Main(string[] args)
{
    Console.WriteLine(-(-2147483648));
}
```

readonly struct System.Int32
부호 있는 32비트 정수를 나타냅니다.

CS0220: checked 모드에서 컴파일하면 작업이 오버플로됩니다.

그림 2-24 숫자를 직접 입력하면 오류 발생

- int 자료형을 사용하는 프로그래밍 언어(C, C++, 자바) 등에서는 모두 이와 비슷한 현상이 발생