**1. 묵시적 형 변환(Implicit Conversion)**

* **기본 자료형 간의 변환을 컴파일러가 ‘알아서’ 함**
* 모든 기본 자료형 간의 변환이 가능한 것은 아님
* **특별한 문법이 필요하지 않음**

ex) int num1 = 100000;  
ex) long num2 = num1;

* **정수형을 부동소수점 실수형으로 변환할 때는 손실이 생길 수밖에 없다**
* 정밀도에서 차이가 발생하여 다른 값이 들어갈 수도 있음
* **정밀도 손실(Loss of Precision)**
* 아래의 경우 자료의 값이 달라지는 경우가 발생할 수 있음
* int/long에서 float으로 변환
* long에서 double로 변환
* **승격(Promotion)**
* 컴파일러가 자동으로 실수형이나 부동소수형 자료의 이진 표현을 확장
* 산술 연산자나 논리 연산자가 제대로 동작하게 하거나 ALI(산술논리장치)가 좀 더 효율적으로 돌 수 있게 하려고 사용  
  ex) double num1 = 3.7;  
  ex) int num2 = 10;  
  ex) Console.WriteLine(num1 + num2); // 연산의 결과는 double 형으로 승격됨
* **작은 형에서 큰 형으로 변환**
* ex) 32비트에서 64비트 형으로 변환  
  - 문제 없음(64비트가 32비트를 포함할 수 있음)
* **큰 형에서 작은 형으로 변환**
* 값이 충분히 작으면 문제가 없음  
  - 문제 없음(64비트에 32비트만 포함되어 있다면 32비트가 수용할 수 있음)  
  - 따라서 수용 가능한 비트 수를 넘어가면 문제가 발생함
* 프로그래밍 언어에 따라서 변환을 못하게 할 수 있음  
  - C, C++에서는 가능하지만 C#에서는 불가능
* 값이 크다면 문제가 발생함  
  - 런타임 중에 값이 어떻게 될지 모름  
  - 이런 상황을 ‘정보의 손실’이라고 함

**2. 명시적 형 변환**

* **두 double 형 수를 더하기**
* 이때 정수형으로 명시적 형변환을 한 경우 소수점은 버려지게 됨
* **명시적 형 변환**
* long형에서 int형으로  
  - long형의 비트 표현을 고려해 봤을 때 32비트까지 담긴 내용만을 int에 저장함  
  - 문제가 발생할 여지가 있음
* double에서 int형으로  
  - 컴파일러가 double형 값을 대상으로 int형으로 표현하고자 할 때 정수 부분만 표현해줌  
  - double형과 int형의 2진수 비트 패턴은 완전히 다른 방식으로 작동하기 때문에 연관성은 없음
* 모든 자료형이 변환되는 것은 아님  
  - 예를 들어, int형에서 bool형으로의 형변환은 불가능함
* **명시적 형 변환은 프로그래머의 의도를 나타냄**
* 강한 타입 언어인 C#에서는 묵시적 형 변환이 불가능한 상황에서 명시적으로 표현해주지 않는 경우에는 컴파일러가 오류를 표시함

**3. 산술 연산자**

**4. 부호 있는/없는 자료형(Signed/Unsigned Types)**

* **부호 있는 자료형**
* 음수와 양수를 모두 표현  
  - sbyte, short, int, long
* **부호 없는 자료형**
* 양수만 표현  
  - byte, ushort, uint, ulong
* **왜 byte만 부호 없는 쪽이 기본인가?**
* 이를 알기 위해서는 문자형(char)의 역사를 살펴봐야 함  
  - ASCII는 처음에는 총 128개-7비트로 표현이 가능했음 byte에서 1비트가 남음  
  - 다른 나라들이 자신의 언어를 남은 1비트에 추가하려는 노력이 발생  
  - 이때 최상위 비트(부호비트)가 1이 되어 음수로 표현하게 되어 버림  
  - 따라서 8비트 문자형들은 unsinged가 좀 더 합리적이라고 판단
* **언제 사용하는가?**
* 부호 있는 변수와 부호 없는 변수는 굉장히 중요함  
  - **프로그래머의 명백한 의도**를 보여주기 때문
* int age = 17;  
  - 예를 들어 ‘나이’와 같은 int형 코드는 음수가 되면 안 됨  
  - 이를 방지하기 위함
* **오버플로우(overflow) / 언더플로우(underflow)**
* 연산의 결과로 자료형보다 큰/작은 수가 나오는 경우를 말함
* 자료형의 크기는 변하지 않기 때문에 넘어가버린 비트는 버림
* 오버/언더플로우가 발생할 경우 원치 않는 값이 나올 수 있음  
  - 오버플로우는 최대값에서 최소값으로,  
  - 언더플로우는 최소값에서 최대값으로
* **변수의 자료형을 선택할 때 주의할 점?**
* 변수의 역할에 알맞은 자료형을 선택할 것  
  - 사용할 자료형이 충분히 데이터를 담을 수 있을지
* 오버/언더플로우를 고려해서 자료형을 선택할 것

**5. 상수의 접두사(prefix)와 접미사(postfix)**

* **접두사와 접미사**
* 접두사와 접미사는 상수의 앞이나 뒤에 상수(리터럴)를 꾸미는 기호  
  - 이를 통해 리터럴에 대한 추가적인 정보를 제공
* **상수 접두사**
* 상수 앞에 붙는 기호로, 진법을 나타냄  
  - 2진수  
   - 0B-, **0b-** ex) int num1 = **0b**10;  
  - 16진수  
   - 0X-, **0x-** ex) int num1 = **0x**10;
* 일반적으로 소문자를 더 많이 사용함
* 일반적으로 접두사는 진법을 나타냄
* **상수 접미사**
* 상수 뒤에 붙는 기호로, 상수의 자료형을 나타냄  
  - 부호 여부  
   - **-u**  
   ex) uint num1 = 10**u**;  
  - long형  
   - **-l**  
   ex) long num1 = 10**u**;  
   ex) ulong num2 = 10**lu**;  
   ex) ulong num2 = 10**ul**; // 조금 더 일반적인 형태  
  - float형  
   - **-f**/-F  
   ex) float num1 = 10.0**f**;  
   ex) float num2 = 10.0**F**;

- double형  
 - **없음** 혹은 -d/-D  
 ex) double num2 = 10.0; // double의 경우 안 쓰는 경우가 더 많음  
 ex) double num2 = 10.0**d**;  
 ex) double num3 = 10.0**D**;