



1 converting from and between integral-types(integer-like types)

- Integer - which are arbitrary-precision integer
- Int - which fixed-width machine-specific integers, its range of Int is -2^{31} to $+2^{31} - 1$

```
-- convert Integral to Num
fromIntegral::(Integral a, Num b)=>a->b
```

2 converting from and between fractional-types

```
-- convert from Rational to Fractional
fromRational::(Fractional a)=>Rational->a
-- convert from Real to Rational
toRational::(Real a)=>a->Rational
```

3 converting from and between real-types

- Rational - which are arbitrary-precision fractions
- Double - which are double-precision float-point numbers

```
-- convert from Real to Fractional
realToFrac::(Real a, Fractional b)=>a->b
-- convert from Rational to Fractional
fromRational::(Fractional a)=>Rational->a
```

4 converting from and between floating-point precision types

```
-- converting between float and double can be done using GHC-specific functions in the C
-- Float <=> Double
float2Double::Float -> Double
double2Float::Double -> Float
```

5 unboxed and boxed types

- unboxed type[**value**]: Int, Float, Double(int, float, double in C)
- boxed type[**pointer**]: int*, void* (pointer in C)

It is easier to explain with low-level language like C. Unboxed types are like int, float, double, etc. Boxed types are like int*, float*, double*, etc. If you have got the int, you always know the value as it's represented in the bit-pattern, therefore, it is not lazy. It must be strict too, as all values of int are valid and not bottom.

However, given an int* you may choose to dereference the pointer late to get the actual value (thus lazy), and it is possible to have invalid pointer.(it contains BOTTOM, i.e. non-strict)