Conversion of Numeric Types

Many conversions happen automatically (called implicit type casts)

Given `int i = 5, j = 4;` `double x = 1.5, y = 2.1;`:

Example	Explanation
`double z = 7;`	conversion from int to double (7.0)
`int n = 6.7;`	6.7 is truncated to `int`, setting n to 6
`double z = 1/3;`	Gotcha! 1/3 done as (compile time) integer division, giving zero, converted to double (0.0)
`double z = 1.0/3.0;`	Correct way, or 1/3.0 or 1.0/3; result is 0.3333..
`double z = i/j;`	Gotcha! i and j are ints, so i/j is done as integer division. Result (1) converted to `double` (1.0)
`int n = i+x;`	i is converted to `double`, $i + x$ done as double giving 6.5, which is truncated to `int`, result: 6
`int n = x+y;`	$x + y$ is evaluated as double, giving 3.6. Result then truncated to `int`, result: 3

To override implicit casting, C allows you to write an **explicit typecast**: place the destination type in brackets before the expression to be converted, bracketing that expression where necessary

Example	Explanation
`double z = (double) i/j;`	Gotcha! Placing the (double) cast here does nothing, as before the result is 1.0
`double z = ((double)i)/j;`	Ok, this converts i to double first, forcing j to be converted to double too, so that the division can happen in double. The result is now 1.25 as we expected.
`double z = i/((double)j);`	This does the right thing too.
`z = ((double)i)/((double)j);`	This does the right thing too.