

# PCP — Lecture 04

Fall 2020 September 3, 2020

## Last Time - Datatypes and Variables

- Presentation of the different datatypes (value types and reference types).
- Decreasing precision from `float` to `double`, `double` to `decimal`.
- Corresponding literals.
- `int` and `string` variables: declaration, assignment, initialization, and manipulation.
- Miscellaneous information on variables and introduction to numeric operations.

## 1 Operations on `int` (reminder)

Suppose we are given an `int` variable, `myVar`, with which we can perform the following operations:

Operation	Arithmetic Operator	Algebraic Expression	Expression
Addition	+	$x + 7$	<code>myVar + 7</code>
Subtraction	−	$x - 7$	<code>myVar - 7</code>
Multiplication	*	$x \times 7$	<code>myVar * 7</code>
Division	/	$x/7, x \div 7$	<code>myVar / 7</code>
Remainder (a.k.a. modulo)	%	$x \bmod 7$	<code>myVar % 7</code>

For the remainder, courtesy of [wikipedia](#):

For example, the expression “5 mod 2” would evaluate to 1, because 5 divided by 2 has a quotient of 2 and a remainder of 1, while “9 mod 3” would evaluate to 0, because the division of 9 by 3 has a quotient of 3 and a remainder of 0; there is nothing to subtract from 9 after multiplying 3 times 3.

We use `=` to assign a value to a variable, but we can also use `=` followed by one of the previous operator to obtain an “augmented assignment operators” or “compound assignment operators”:

```
int a = 3, b, c; // Multiple declarations with an assignment.
b = 34 + a;
a = a - 1; // Self-assignment
a -= 1; // Shorthand, this is the same as a = a - 1;
```

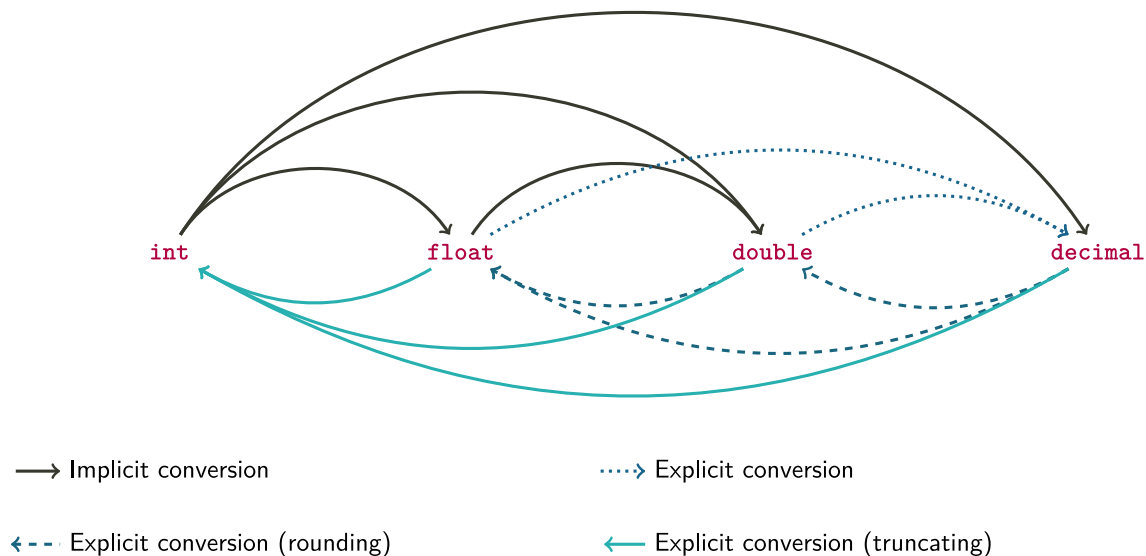
## 2 Implicit and Explicit Conversions Between Numeric Datatypes

As we saw in the [cheatsheet](#), we can store e.g. a `float` literal inside a `double` variable (that’s an automatic, or *implicit* conversion), but we can not store a `double` literal inside a `float` variable. C# refuses to do this “automatically”, because we risk to lose information, but we can “force” it to perform this operation *explicitly* using *cast operators*.

```
float b = 4.7F;
int a = (int) b; // A cast operator is simply the name of the datatype between
                ↪ parenthesis. Here, we convert a float into an int.
```

Using casting allows us to go “against” those safe-guards, and can lead to the following complications:

- Storing an imprecise number using a precise datatype (e.g. from `double` to `decimal`).
- Truncating a floating-point number as a “truncated” number (e.g. from `double` to `int`).
- Rounding a precise number to fit a less precise datatype (e.g. from `decimal` to `double`).



Note that *you can*, actually, store a `float` literal inside a `double`, but that you *can not* store a `double` or a `float` literal inside a `decimal`.

### 3 Operations on other numerical datatypes

Actually, there is a `+` operator for `float`, `double`, and `decimal`, and the same goes for `-`, `*`, and `%`

Casting can be useful when one wants to divide integers:

```
int pie = 21;
int person = 5;
```

Operation	Result	Type of the multiplication used
<code>pie / person</code>	4	<code>int</code>
<code>(float)pie / person</code>	4.2	<code>float</code>
<code>pie / (float)person</code>	4.2	<code>float</code>
<code>(float)(pie / person)</code>	4	<code>int</code>

Note that the integer division simply “truncates”, and does not round up (that is,  $19 / 10$  would return 1).

Also note that in “4.2”, “4” is the integer, “.” (period) is the decimal separator, not to be mixed with “,” (comma), the thousands separator.

When performing an operation involving different datatypes, the result type of the operation is the “most precise” datatype if it is allowed (i.e., an implicit conversion can take place), otherwise an error is returned. Refer to the “Result Type of Operations” chart from the [cheatsheet](#) for more detail.

## 4 Operations on Strings

We saw that we could perform interpolation on strings to “fetch” the value of a variable and “insert” it into a string; however, we can use other operations as well. For instance, we can use the + sign for strings as well, to perform an operation called “concatenation”:

```
string name = "Bob";
string greetings = $"Welcome, {name}!\nI'll be your guide."; // interpolation

string text = "Hi there" + ", my name is Marie."; // Concatenation. This "join" the
    ↪ strings together.

Console.WriteLine(text); // We can write directly the name of the variable, instead of
    ↪ using interpolation, if we wish.
```

Note that the + sign denotes different operations depending on the type of the operands (string or int, for instance).

## 5 Reading a String From the User

We can ask *the user* to give their name to the program while it is running as RunTime input. We can use a statement like:

```
string yourFirstName = Console.ReadLine();
```

to store the value entered by the user from the output window in the string variable labeled `yourFirstName`.