

# Type Casting

<https://csci-1301.github.io/about#authors>

June 19, 2021 (10:06:12 AM)

## Contents

|                                   |          |
|-----------------------------------|----------|
| <b>1 Numerical Datatypes</b>      | <b>1</b> |
| 1.1 Literals and Variables        | 1        |
| 1.2 Operations                    | 2        |
| <b>2 Casting</b>                  | <b>3</b> |
| 2.1 Cast Operator                 | 3        |
| 2.2 Implicit and Explicit Casting | 4        |

## 1 Numerical Datatypes

For this part, it is recommended to have the datatypes cheatsheet<sup>1</sup> readily available. Note that it contains numerous references at its end. You are encouraged to open those links, if you have not already, to have a look at the official documentation, which should not scare you.

### 1.1 Literals and Variables

This part should be first carried out without using an IDE, but with pen and paper.

Assume we have the following statements:

```
int a = 21, b = 4;
float f = 2.5000000f;
double d = -1.3;
decimal m = 2.5m;
```

Answer the following:

- How many variables are declared?
- What are their datatypes?
- What are their values?
- What are their names?

---

<sup>1</sup>[.././datatypes\\_in\\_csharp.html](https://csci-1301.github.io/datatypes_in_csharp.html)

## 1.2 Operations

- Consider the following expressions. For each of them, tell if they are legal and if so, give the result and its corresponding datatype. The first two are given as examples:

| Operation | Legal? | Result | Datatype |
|-----------|--------|--------|----------|
| a + d     | Yes    | 19.7   | double   |
| m + f     | No     | N/A    | N/A      |
| a / b     |        |        |          |
| b * f     |        |        |          |
| d + f     |        |        |          |
| d + b     |        |        |          |
| a + m     |        |        |          |
| f / m     |        |        |          |
| d * m     |        |        |          |

You can check your answers using an IDE: create a new project, copy the variable declarations and assignments, and write your own statements to perform the calculations in the `Main` method. For instance, if you want to check that the result of `a + d` is of type `double`, write something like:

```
double tempVariable1 = a + d;
Console.WriteLine($"The value of d+f is {tempVariable1}");
int tempVariable2 = a + d; // This line should give you an error.
```

## 2 Casting

### 2.1 Cast Operator

Create a new project, and then do the following.

1. Add in your program the following:

```
float floatVar = 4.3f;
int intVar = floatVar; // This statement will give you an error
```

You will get an error that reads

```
Cannot implicitly convert type 'float' to 'int'. An explicit conversion exists (are
↪ you missing a cast?)
```

Can you explain it?

2. Your IDE is suggesting that we use a “cast” to “force” C# to store the value of the variable `floatVar` into the variable `intVar`. To do so, replace the previous statement with the following:

```
int intVar = (int)floatVar; // This statement will compile
```

3. Using a `Console.WriteLine` statement, observe the value stored in `intVar`. Can you tell if the value stored in `floatVar` was rounded or truncated before being stored in the variable `intVar`? Conduct further experiments if needed to answer this question.

## 2.2 Implicit and Explicit Casting

1. Look back at the warning given by the IDE. It uses the term “implicitly convert” before introducing the cast operator.
2. While you needed a cast to convert a `float` to an `int`, do you need one to convert an `int` to a `float`? Try the following:

```
int intVar = 21;
float floatVar = intVar; // Does this need a cast?
```

Generally, you need an explicit cast if an implicit conversion would lead to data loss. Since all possible `int` values are also valid `float` values, no explicit cast is needed!

3. Do these cases need an explicit cast, or will an implicit conversion work? Try them in your IDE to check your answers!
- `double` to `int`
  - `int` to `double`
  - `float` to `double`
  - `double` to `float`
  - `int` to `decimal`
  - `decimal` to `float`
  - `float` to `decimal`

That last result may have been surprising. While `decimal` is higher precision than `float` and `double`, it requires an explicit cast from either of those types, as you want to “force” imprecise data into a datatype that is supposedly extremely precise. Think about measuring wood with an inaccurate tape measurer and then cutting it with laser precision: that is what storing a `float` into a `decimal` is!