Type Casting

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

September 5, 2023 (05:12:38 PM)

Contents

1 Numerical Datatypes **1**
- 1.1 Literals and Variables . . . 1
- 1.2 Operations . . . 2

2 Casting **2**
- 2.1 Cast Operator . . . 2
- 2.2 Implicit and Explicit Casting . . . 3

This lab serves multiple goals:

- To help you develop good lab habits,
- To declare, assign, and display variables of different datatypes,
- To understand how to use arithmetic operators,
- To experience the differences in numerical datatypes,
- To learn how to "cast" between numerical datatypes,
- To understand what types of operations are "legal".

1 Numerical Datatypes

For this part, it is recommended to have the datatypes cheatsheet[1] readily available. Note that there are numerous references at the bottom of the cheatsheat. You are encouraged to open those links and have a look at the official documentation, if you have not already done so. Being able to reference the official documentation will help you in your studies.

1.1 Literals and Variables

This part should first be attempted using pen and paper before using an IDE.

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:

[1] labs/Casting/../../datatypes_in_csharp.html

- How many variables are declared?
- For each of those variables, give their name, datatype, and value.

Solution:

There are 5 variables.

Name	Datatype	Value
`a`	`int`	21
`b`	`int`	4
`f`	`float`	2.5000000
`d`	`double`	-1.3
`m`	`decimal`	2.5

1.2 Operations

Consider the following expressions, all of which use the variables we declared above. For each of them, decide if they are legal, and if so, determine 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`			

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 statement

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

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` has higher precision than `float` and `double`, it requires an explicit cast from either of those types, as you would be attempting to "force" imprecise data into a datatype that is supposedly extremely precise. Think about measuring wood with an inaccurate tape measure and then cutting it with laser precision; that is what storing a `float` as a `decimal` is!