CSCI 1301 - Lab 06

Clément Aubert

January 29, 2018

Deadlines: This lab needs to be completed before taking Lab 07.

Dependencies: Part I must be completed **before** Part II.

Part I - First Use of the Debugger

a. Open one of your project. It can be any project, as long as it uses variables.
b. Make sure you can compile and run it.
c. Hit F11 once. Observe what happen. Move the black screen where the printing happen (called the *console*, or *Command prompt*) to see your code. Hit F11 a couple of times, and see what happen. You are executing your code line by line, and can track many useful information, including the value and datatype of your variables (in the "Locals" panel / window).
d. You can find out at https://msdn.microsoft.com/en-us/library/y740d9d3.aspx (VS 2015) or https://docs.microsoft.com/en-us/visualstudio/debugger/navigating-through-code-with-the-debugger (VS 2017) alternative ways of starting the debugging functionnalities of VS.

Part II - Numeric Datatypes

For this part, I recommend opening the web page or the numeric version of the document we just studied in class. Note that it contains numerous references at its end.

Making Simple Calculations

This part should be carried out without using VS.

Assume we have the following statements:

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

a. Answer the following:

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

b. For each of the following operations, tell if they are legal, and if so, give the result and its corresponding datatype:

- `a / b`
- `b * f`
- `d + f`
- `d + b`
- `a + m`
- `f / m`
- `d * m`

Cast Operator

a. Start by checking your answers to the previous problem using VS. Create a new project, copy the code above in the `Main` method, and run the debugger to find out the answers to question a. For question b., write your own statements to perform the calculations. For instance, if you believe that the result of `d + f` is of type `int`, write something like

```
int tempVariable = d + f;
Console.WriteLine($"The value of d+f is {tempVariable}");
```

b. Remember that we can't assign a `float` litteral to an `int` variable. Check it for yourself by adding the following to your program:

```
int intVar = f; // 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?)
```

c. VS is suggesting that we use a "cast" to "force" C# to store the value of the variable `f` into the variable `a`. To do so, replace the previous statement with the following:

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

d. Using the debugger, observe the value stored in `intVar`. Can you tell if the value stored in `f` was rounded or truncated before being stored in the variable `intVar`? Conduct further experiments if needed to answer this question.

e. Add the following in your code:

```
decimal decVar = 12344321.4999999991M;
double douVar = (double)decVar;
float floVar = (float)douVar;
```

And observe, using the debugger or printing instructions, the gradual loss of precision.

Part III - Reading Other Numeric Datatypes

a. Create a new project
b. Declare three variables: one of type `int`, one of type `float`, and one of type `double`.
c. You may remember that we can use the following to read an `int` from the user (assuming your `int` variable is named `intVar`):

```
intVar = int.Parse(Console.ReadLine());
```

Similarly, we can read `float` using `float.Parse(Console.ReadLine())`, and double using `double.Parse(Console.ReadLine())`.

d. Write statements that ask the user to enter an `int`, a `float` and a `double`, store those values in the appropriate variables, and then print them.

Part IV (Optional) - Pushing Further

The following are two independent tasks, to widen your understanding of this class, and to prepare you for the next labs.

a. Compute in your head the result of the following operation: 1000000.0 + 1.2 - 1000000.0. Now, implement it using `float`, `double`, and `decimal`:

```
Console.Write("With floats:\n\t");
Console.WriteLine(1000000.0f + 1.2f - 1000000.0f);
Console.Write("With double:\n\t");
Console.WriteLine(1000000.0 + 1.2 - 1000000.0);
Console.Write("With decimal:\n\t");
Console.WriteLine(1000000.0m + 1.2m - 1000000.0m);
```

Can you explain this behaviour?

b. Use the `.ToString()` method introduced in the previous lab (Part III, b.) to convert the `int`, `float` and `double` variables of Part III into `string`, and store them in a `string` variable. Use concatenation to create a single string containing those three values, and print them with a single `Console.WriteLine` instruction.