Programming Assignment Three: Ada Calculator

Design and Implementation

For the calculator there were three major tasks. 1) evaluate primary arithmetic operators with integers 2) evaluate arithmetic operators with floating point values and 3) evaluate floating point values with parenthesis as well. There is a file **`test-expression-answers.txt`** that shows what the sample output should be. I was able to follow the algorithm that was outlined for task #1 and before I got floats working I also added the parenthesis. For #2 I needed to use a different approach for reading in floating point values. I ended up finding the end of the value by searching for a substring, then slicing the buffer to get the full string representation of the next value. I converted the string to an integer, and if that got me an exception I tried a float instead. If the float conversion failed, then the expression was invalid.

The results of my program don't quite match up with the expression answers file because my calculator returns an integer value when two integers have operations performed on them. In my program I made a Generic_Number package that that allowed me to represent both integer and floating point types in the same record! Below in the same run you can also see that division also may or may not return a floating point depending on whether or not the numerator is wholly divisible by the denominator.

Sample run:

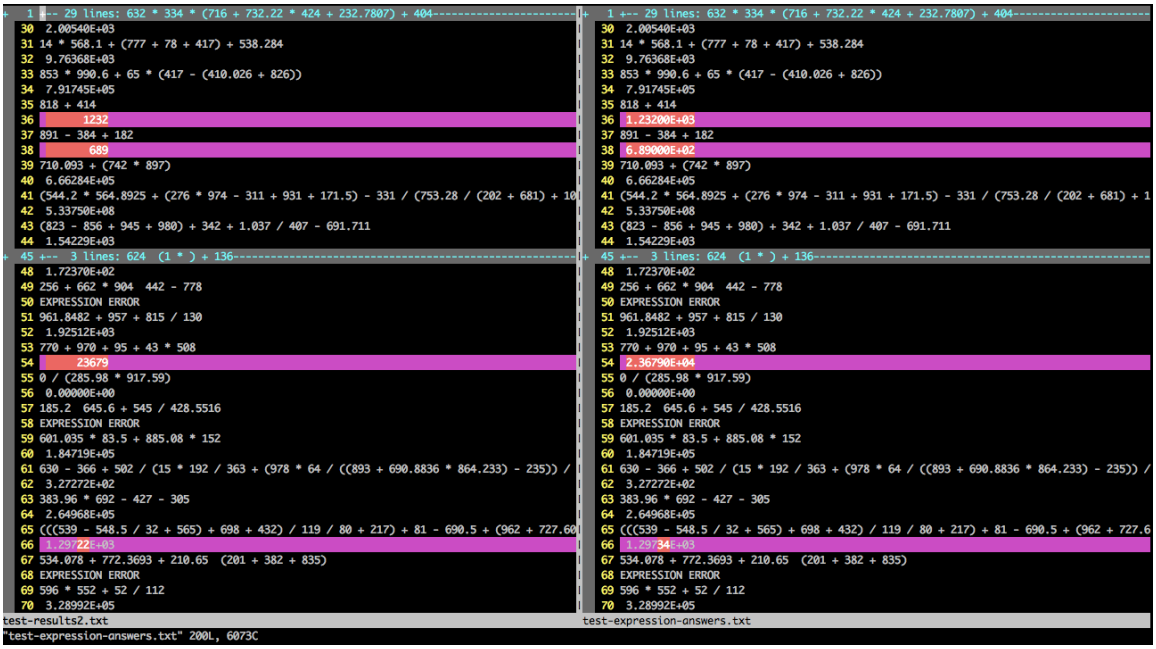
```
$ ./calculator
2 + 123
        125
123.34 + 134
 2.57340E+02

(12 / 3)
          4
12 / 5
 2.40000E+00
```

This is the primary idea how I was able to implement the "generic" type. It isn't truly generic, but rather is a discriminated record.

```
type Number_Type is (Float_Type, Integer_Type);

type Number(Option : Number_Type := Float_Type) is
  record
    Num_Type : Number_Type := Option;

    case Option is
      when Float_Type =>
        Float_Value : Float := 0.0;
      when Integer_Type =>
        Integer_Value : Integer := 0;
     end case;
  end record;

procedure Set(this : out Number; val : Float);
procedure Set(this : out Number; val : Integer);
```

Comparing the results:

```
cat data/test-expressions.txt | ./calculator > data/test-results.txt
vim -d data/test-results2.txt data/test-expression-answers.txt
```



Conclusion:

Ada has a lot of interesting features. I think some parts of it are a little difficult, but it has much more explicit language semantics, which is interesting to see after learning about all of these different features (like in mode, out mode, etc.).

Sources:

I had to do a lot of googling for this one. It was actually pretty difficult to find information compared to other languages. Here are a few of the links I managed to keep. These links go over OOP encapsulation structure (I didn't really use), the record trick I used, and operator overloading. I referenced wikibooks and rosetta code the most.

```
https://en.wikibooks.org/wiki/Ada_Programming/Object_Orientation
https://en.wikibooks.org/wiki/Ada_Programming/Types/record#Discriminated_record
https://rosettacode.org/wiki/Arithmetic/Rational#Ada
```