Lab 1

Due: At the end of lab

1 Lab 1 Specification

1.1 Introduction

The skeleton code at `https://classroom.github.com/a/tsYH9pia` is the start of today's lab. Follow the setup and import directions.

In Visual Studio, open the `comp3350_lab1.fs` file. It contains several function definitions. Try testing out the `hello` function by selecting its text and hitting ctrl-return (Mac) or alt-enter (Windows). A window should appear beneath that shows the result of the `hello` function. This is the interactive prompt that you are familiar with from class.

Enter `hello "world";;` on the command prompt, and you should get an appropriate response. Try modifying the function to say "Goodbye" instead, and reload it into the F# Interactive window. Test that it still behaves like you expect.

If you close Visual Studio, you can reopen any project by opening its '.sln' file.

1.2 Implementation

Refer to the **lec2** slides for language and syntax reminders.

Write and test the following two functions. Make sure that you test them with a range of inputs to make sure they work the way you expect. When you write code in a file, do not include the `;;` line endings. Only include it in interactive mode.

1.2.1 GCD

GCD returns the greatest common divisor of two integers. Write the recursive function that implements Euclid's algorithm:

```
int GCD(int x, int y) {
  if (y==0) return x;
  else return GCD(y, x % y);
}
```

```
> GCD 9 18;;
val it : int = 9
```

1.2.2 factorial

The factorial function is defined as:

```
0! = 1
n! = n * (n - 1)!
```

```
> factorial 5;;
  val it: int = 120
```

Notice that factorial can produce very large results – larger than the maximum value of a standard integer value.

```
> factorial 20;;
  val it : int = -2102132736
```

We can fix this overflow issue by using a type that holds arbitrarily large integers – **bigint**. We must create a factorial implementation that has the following signature:

```
val factorial : n:int -> bigint
```

Write a naive version of factorial (found in the lecture notes) and then convert the type of the return value and anything multiplying the return value to a `bigint`. This is similar to the `float` type conversion function we saw in class. Here is a sample of converting a value to a bigint:

```
> bigint 3;;
  val it : System.Numerics.BigInteger = 3 {IsEven = false;
                                          IsOne = false;
                                          IsPowerOfTwo = false;
                                          IsZero = false;
                                          Sign = 1;}
```

When working correctly, the bigint version of factorial should be able to handle very large return values:

```
> factorial 55;;
  val it : bigint =
    12696403353658275925965100847566516959580321051449436762275840000000000000
      {IsEven = true;
       IsOne = false;
       IsPowerOfTwo = false;
       IsZero = false;
       Sign = 1;}
```

2 Testing

With the two functions defined in interactive mode (using the same alt-enter or ctrl-return shortcut that you used for Hello World), test the results with different parameters. We will look at more unit testing in the future, but for now you can test the small functions manually. Assume that all inputs are ≥ 0.

Your submitted code should not include the `;;` line endings – that's just for interactive mode. Confirm that GCD returns an `int` and factorial returns a `bigint` (which is an alias for the type `System.Numerics.BigInteger`).

3 Grading

To submit, follow the directions in the project setup document. You can confirm that your submission is complete by checking that your push is visible in your repo on the Github website.

Be professional. Make results easy to understand and grade. Include only those parts to be graded. Leave comments where necessary, especially if it aids in grading.

Each of the 2 functions is worth 50% of the lab grade.