# CAB 402 Assignment One Report – James Flannery

## Efficiency and Effectiveness

For my tests, all three approaches produced almost equally effective code. All managed to produce matching FASTA output for tests 0 through 10, and all produced matching Homolog output for tests 0 through, 9. Interestingly, both the pure and impure approaches produced the exact same, incorrect, Homolog output for test 10. These are only a few lines of incorrect output, which suggests that my math is off somewhere. The C# approach was wildly different, though, with the majority of lines incorrect.

With regards to efficiency, there are several notable differences, and observable patterns. Most generally, the Impure F# approach runs the fastest, followed by the pure F# approch, with C# much slower. Do note that the time axis is logarithmic, so the differences in F# execution times is small, but the C# times are significantly larger.

These differences match expectations, due to the nature of each implementation. Between the two F# approaches, the pure version needs to create a new map for every change made, whereas the impure approach adds the changed element to the already existing map. This is especially noticible with loss, which involves removing a single element of the map in impure F# versus creating an entire new map in the pure version.

Why, then, is the C# code the slowest? It’s possible, of course, that it’s just because of my particular implementation. Certainly that has some impact. But I also believe it to be a result of the general Object-Oriented style. C# has small gains in speed because it does not need to create a new collection in full, but these tests are entirely data processing. It lacks the quick processing functions so often used in F#, such as iter, map and especially fold.

The extremely slow execution time, though, is almost certainly due to an error in implementation.

Ease of programming  
This is hard to gauge, and quite subjective. Firstly, I was already familiar with C#, but I had to learn most of the F# as I went along. On top of that I started F# first, so naturally I had a better understanding of the problem by the time I started the stateful implementations. With all that in mind, I’ll do my best to give an overview of how difficult I believe each style to be.

Surpisingly, I actually think that the Pure F# approach is the easiest. Being able to think entirely in terms of data made it very clear what needed to be done for each evolutionary function. Type inference, once I (finally) understood it properly, meant many errors were clear immediately. Being able to pass functions as arguments allows for very elegent code, once you get your head around it.  
  
Higher order functions, especially fold, are much much easier to write than using counters and holding variables and all that. That being said, the syntax for these can be quite finicky. Also, whilst there are less errors, they are often much harder to understand than in C#. These are problems that would disappear once you became more familiar with the language, though.

Impure F# is mostly the same, but the differences are all negatives. Dealing with objects is just a hassle, especially because they can’t be picked up using type inference, so you have to specify everything manually. I understand the need for this stuff, but I’d avoid it where possible.

C# is certainly easier to visualize. That is one of the main draws of OO, after all. But there’s a lot of code to write. So while I might not say that it’s harder to program, it definitely takes more effort.   
  
I do think, though, that C# is much more general. It may just be my unfamiliarity with the language, but F# seems good for processing data, and not much else. I’ve no idea how you would go about creating something with a UI, for instance, or really anything with state. Likely you’d want to use C# for that, and pass data to F# for processing instead.   
  
Ease of comprehension, maintenance  
This, too, is subjective. It also depends on what you mean by easier. C# is more clear in how it works, to be sure. Everything is well defined and laid out. To me, though, it takes much longer to understand how everything fits together. Well chosen class and method names help, of course, but there is just much more code to go through.   
   
I find F# code great because you can follow the path of data through the program, and it will *always* follow the same path. Making changes can be easier in C#, since you know which variable or method should be in which class. But it’s also easier to break things. Changes you make can effect some other method in some other class, and you may not realise until you track down the bug weeks later. F# error messages are ambiguous, but they are immediate.

So yes, it’s subective. I personally think F# is much easier to read for small projects, whereas C# is much better for larger ones. Seems to be a running theme here.

Conciseness of Code  
F# is more concise. Pure F# more so than impure.  
The F# implementations have about 200 lines of code, C# has two or three times that, spread over 8 pages.  
C# has a lot of extraneous brackets and syntax.   
Higher-order functions, especially with lambdas, can express large operations in one line.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Pure F#** | **Impure F#** | **C#** |
| Test 0 (Create) | 4 | 4 | 26 |
| Test 1 (Snip) | 7 | 5 | 13 |
| Test 2 (Insert) | 3 | 6 | 4 |
| Test 3 (Delete) | 6 | 4 | 4 |
| Test 4 (Duplicate) | 3 | 7 | 20 |
| Test 5 (Loss) | 7 | 2 | 4 |
| Test 6 (Fission) | 6 | 4 | 16 |
| Test 7 (Fusion) | 6 | 6 | 3 |
| Test 8 (Speciaton) | 6 | 4 | 68 |
| Test 9 (Combined) | 6 | 4 | 90 |
| Test 10 (Combined) | 89 | 7 | 4081 |

This data is just copied from the test program provided. Graph was generated in Excel.