ECE 421
Assignment 1

Arun Woosaree
XXXXXXX

January 15, 2020

1

Functional programming is a declarative paradigm. Computation is treated as the evaluation of mathematical functions. Expressions and declarations are used as opposed to statements. Given a set of inputs, the output will always be the same. Data is immutable, and as a result, functions have no 'side-effects'.

2

Explain what it does

Right at the beginning, we see that the program produces a side-effect, which is using IO to output some data. This contradicts the idea of a 'purely' functional language. However, if a program never has side effects, it would be pretty boring. . . .

say something more about side effect

We see that the program is written in a declarative style, even though Haskell is generally considered a purely functional language. . .

3

Immutability is nice, because it makes things predictable. . .

4

```
void add(&int x, int y, int z){
  *x = y + z;
}
```

idk

5

a)

- sqrtx takes an input, and returns the squared value of the input
- imparativefun returns the sum of squares of its input, which is a list of numbers
- functionalfun does the same thing, but does so in a functional manner instead of an imparative way

b)

I would argue that te functionalfun is more maintainable as it's easier to understand what the function is doing. Also, even though the functions do the same thing, functionalfun there is less to read??

c)

```
let fourthpower x = sqrt sqrt x
```

6

- changing the file system is a side effect, so it is not a pure function
- inserting a record – if we're talking about a function that makes a database request, then it can be functional, however, if we're talking about the function that the database internally uses, the database would be updated, which means that a value is being mutated, so therefore not a pure function
- making an http call can be a pure function
- printing to the screen requires a side-effect, namely I/O, so it is not a pure function
- querying the DOM can be a pure function
- accessing the system state can be a pure function, as long as the state is not being mutated
- pure functions return the same output every time for a given set of inputs. Any randomness violates this, so Math.random() is not a pure function

7

```
fn functionalfun (x : isize) -> isize {
    return (1..=x).map(|i| i*i + 2).sum();
}
```

8

9

my guess is that haskell is fine with it and rustc goes REEEEEEEEEEEEEEEEEEEEE