Functional Programming I (CS4620) Assignment 1*

Getting Started (Due: October 8. Marks: 6)

Introduction

This assignment is about getting used to `ghc` and `ghci` and about learning different techniques to implement functions. When you're finished with this assignment you should know: (1) how to use `ghci`, (2) how to define functions on top of existing functions, (3) how to define them using pattern-matching, (4) how to define them using partial applications, …

Please remember the following.

- Every function definition should include a proper type signature, unless the instructions clearly state there is no need for a signature. Not only is adding them a proper form of documentation but it is also a good exercise.

  Please note that you should infer the type of the function yourself, *based on what we have studied in class.* So, e.g. if `ghci` tells you the type of one of your functions is `Foldable t => t [a] -> [a]` because it is equivalent to `concat` then you shouldn't use that type but use `[[a]] -> [a]` instead because we never studied the class `Foldable`.
- Every function should include a brief discussion of the implementation technique.
- Unless specified otherwise, you should only use techniques that we've studied in the lectures up to and including the week during which this assignment was released. These techniques correspond to Chapters 1–5 from the book.

Assignment Details

The following are the assignment's tasks. Please present all solution in the same order as the tasks. In addition, please make sure you identify each solution by adding a single-line comment along the following lines.

```
-{}- Task~$x$.
```

All functions must have a signature unless the tasks explicitly says you don't have to. In addition, your functions should not use user-defined auxiliary functions, unless the tasks explicitly says you may use them.

*This assignment is for students taking CS4620. You are not allowed to share this assignment or solutions with others. This means, for example, that you are not allowed to post this assignment and/or post solutions on `GitHub`, on social media, or other public media.

Implementing `f`

The function `f` is a predicate, which takes an `Int` and returns a `Bool`. The return value is `True` if and only if the argument is 1, 2, 3, or 5.

Task 1 (10%). *Implement a function `f1`, which should be equivalent to `f`. The implementation should use the conditional statement but it should not use pattern matching or guards.*

Task 2 (10%). *Implement a function `f2`, which should be equivalent to `f`. The implementation should use guards but it should not use the conditional statement or pattern matching.*

Task 3 (10%). *Implement a function `f3`, which should be equivalent to `f`. The implementation should use pattern matching but it should not use the conditional statement or guards.*

Before you continue, please make sure all your functions have a type signature and a short explanation about the implementation technique.

Implementing g

The predicate g returns the opposite of `f`.

Task 4 (10%). *Implement the function `g1`, which should be equivalent to `g`. The implementation should not use pattern matching, guards, conditions, or anonymous functions. You may use your implementation of `f1`.*

Task 5 (10%). *Implement the function `g2`, which should be equivalent to `g`. The implementation, which should be different from that of `g1`, should use an anonymous functions. You may use your implementation of `f1`.*

Before you continue, please make sure all your functions have a type signature and a short explanation about the implementation technique.

Slave Labour: Tasks Without Marks

The list `predicates` consists of your functions `f1`, `f2`, `f3`, `g1`, and `g2`.

```
predicates = [f1,f2,f3,g1,g2]
```

Task 6. *For zero marks, define `predicates`. There is no need to add a type signature.*

List Comprehensions

Task 7 (10%). *Define a function `get_predicate`, which takes in an `Int` argument, i, and returns the ith member from the list `predicates`. **There is no need to add a signature** and there is no need to add error handling.*

Task 8 (10%). *Implement a function called `get_and_apply`, which takes an `Int` argument, i, and (then) an `Int` argument, v, and which returns the value that is returned if you call the ith member of the list `predicates` and pass it the argument v. There is no need to add error handling.*

Before you continue, please make sure all your functions have a type signature and a short explanation about the implementation technique.

Task 9 (10%). *Using a list comprehension, implement a function called `apply_to_all`, which takes in an `Int` argument, a, and returns the list consisting of the applications of the functions in `predicates` to a.*

Task 10 (10%). *Implement a function called* `count_false_applications`*, which takes in an* `Int` *argument,* `a`*, and returns the number of the functions in* `predicates` *which return* `False` *if you apply them to* `a`*.*

Before you continue, please make sure all your functions have a type signature and a short explanation about the implementation technique.

Final Task

Task 11 (10%). *Using a list comprehension, implement a function called* `has_opposites`*, which takes in an* `Int` *argument,* `a`*, and returns* `True` *if and only if some of the functions in* `predicates` *return a different value when you apply them to* `a`*. You are not allowed to return* `True` *(because that would work). Your implementation should also work if you change the definition of* `predicates`*, even if you define it as the empty list.*

Before you continue, please make sure all your functions have a type signature and a short explanation about the implementation technique.

Submission Details

- Your program should start with a comment like the following:

```
{-
 - Name: Fill in your name.
 - Number: Fill in your student ID.
 - Assignment: Assignment Number.
 -}
```

- Use the CS4620 *Canvas* area and upload your program as a single *.tgz* archive called *Lab-1.tgz* before 23.59pm, October 8, 2022. To create the *.tgz* archive, do the following:
  - Create a directory `Lab-1` in your working directory.
  - Copy `Main.hs` into the directory. Do not copy any other files into the directory.
  - Run the command '`tar cvfz Lab-1.tgz Lab-1`' from your working directory. The option '`v`' makes `tar` very chatty: it should tell you exactly what is going into the `.tgz` archive. Make sure you check the `tar` output before submitting your archive.
  - Notice that file names in `Unix` are case sensitive and should not contain spaces.
- Notice that the format is `.tgz`: do *not* submit `zip` files, do *not* submit `tar` files, do *not* submit `bzip` files, and do *not* submit `rar` files. If you do, it may not be possible to unzip your assignment.
- Marks are deducted for poor choice of variable names and/or poor layout.
- As explained in lecture 4, you should make sure your assignment submission should have a `Main` class with a `main` in it. The `main` should be the main thread of execution of the program.
- No marks shall be awarded for scripts that do not compile.