

Homework 6

Due Sunday, April 19 by 11:59pm

Handout 19
CSCI 334: Spring 2020

Turn-In Instructions

For this assignment, your completed submission should consist of two parts.

1. A \LaTeX source file called `hw6.tex` and pre-built PDF called `hw6.pdf` that contains your solutions to the “thought questions” that accompany the coding questions. Be sure to `git add` all necessary files (e.g., images) if your \LaTeX depends on it. For full credit, please ensure that your \LaTeX file builds without error.
2. For each coding question in this assignment, create a project directory. For example, the source directory for question 1 should be in a folder called “q1”. You should be able to `cd` into this directory and then run the program by typing the command “`dotnet run`”. Each program should be split into two pieces: a “`Program.fs`” file that contains the `main` method and associated program-startup routines (like argument parsers), and another “`Library.fs`” file that contains the function(s) of interest in the question. All library code should be in a module named “CS334”. Be sure to provide usage output (defined in `main`) for all programs that require arguments. For full credit, your program should both build and run correctly.

Turn in your work using the Github repository assigned to you. The name of the Github repository will have the form `cs334hw6_<your user name>`. For example, if working solo, my repository would be called `cs334hw6_dbarowy`. You should have received an invite to commit in the repository in your email. If you did not receive this email, please contact me right away!

Pair Programming and Honor Code: You may optionally collaborate with one other person on the submission for this assignment. If you work with a partner, choose one of the two partner repositories for your submission. In the other repository, be sure to leave a `collaborators.txt` file that states who you worked with and the name of the repository where the code may be found. If you would like me to pair you with a partner, please let me know.

Sanity Check: A good way to check that you submitted your assignment correctly is to `git clone` your repository to a new folder and then try building/running everything.

This assignment is due on Sunday, April 19 by 11:59pm.

Reading

1. (Required) “A Brief Introduction to F#”
2. (Required) “A Slightly Longer Introduction to F#”
3. (Required) “Why Functional Programming Matters” by John Hughes.
4. (As Needed) “Appendix A: Introduction to \LaTeX ”

Problems

Q1. (10 points) F# Types

Explain the F# type for each of the following function declarations:

- (a) `let a x y = x + y / 2.0`
- (b) `let b(x,y) = x + y / 2.0`
- (c) `let c f = fun y -> f y`
- (d) `let d f x = f (f x)`
- (e) `let e x y b = if b y then x else y`

Since you can (and should) simply type these expressions into the `fsharp` interpreter to determine the type, be sure to write a brief explanation to demonstrate that you understand *why* the function has the type you give. A good explanation says something like “a is a function that takes as input ... and it returns as output ...”, being careful to explain what each of those inputs and outputs are. In particular, be sure to explain why each input and output has the type inferred by `fsharp`.

Be sure to answer this question in your `LATEX` file.

Q2. (10 points) Sum of Squares

Define a function `sumSquares(n: int) : int` that, given a nonnegative integer n , returns the sum of the squares of the numbers from 1 to n :

```
> sumSquares 4;;  
val it : int = 30
```

```
> sumSquares 5;;  
val it : int = 55
```

You may define this function recursively (`let rec sumSquares ...`) or by using a `fold` (if you’re looking for a challenge).

The project directory for this question should be called “q2”. You should be able to run your program on the command line by typing, for example, “`dotnet run 4`”.

Q3. (10 points) List duplication

Define a function `listDup(e: 'a)(n: int) : 'a list` that takes an element, e , of any type, and a non-negative number, n , and returns a list with n copies of e :

```
> listDup "moo" 4;;  
val it : string list = ["moo"; "moo"; "moo"; "moo"]
```

```
> listDup 1 2;;  
val it : int list = [1; 1]
```

```
> listDup (listDup "cow" 2) 2;;  
val it : string list list = [["cow"; "cow"]; ["cow"; "cow"]]
```

The project directory for this question should be called “q3”. You should be able to run your program on the command line by typing, for example, “`dotnet run moo 4`”.

Q4. (20 points) Peanut Butter and Jelly

`pbj(n: int) : string` is a function that returns a *sentence*, which is a **string** composed of a sequence of *words*. n is a positive, nonzero integer supplied by the user. For each consecutive integer between 1 and n inclusive, `pbj` either appends the empty string to its output or a *word*. A word is a string that contains the substring “**peanutbutter**” if n is evenly divisible by 3 or “**jelly**” if n is evenly divisible by 5. Each outputted word in the final sentence should be separated by a single space character. The entire sentence must end with the word “**time**” and a “.” character. Finally, use recursion to solve this problem.

Here are the first ten outputs of the program.

```
$ dotnet run 1
time.
$ dotnet run 2
time.
$ dotnet run 3
peanutbutter time.
$ dotnet run 4
peanutbutter time.
$ dotnet run 5
peanutbutter jelly time.
$ dotnet run 6
peanutbutter jelly peanutbutter time.
$ dotnet run 7
peanutbutter jelly peanutbutter time.
$ dotnet run 8
peanutbutter jelly peanutbutter time.
$ dotnet run 9
peanutbutter jelly peanutbutter peanutbutter time.
$ dotnet run 10
peanutbutter jelly peanutbutter peanutbutter jelly time.
```

The project directory for this question should be called “q4”. You should be able to run your program on the command line by typing, for example, “`dotnet run 10`”.

Q5. (30 points) Zipping and Unzipping

- (a) Write a function `zip(xs: 'a list)(ys: 'b list) : ('a * 'b) list` that computes the product of two lists of arbitrary length. You should use pattern matching to define this function:

```
> zip [1;3;5;7] ["a";"b";"c";"d"];;  
val it : (int * string) list = [(1, "a"); (3, "b"); (5, "c"); (7, "d")]
```

When one list is longer than the other, repeatedly pair elements from the longer list with the *last* element of the shorter list.

```
> zip [1;3] ["a";"b";"c";"d"];;  
val it : (int * string) list = [(1, "a"); (3, "b"); (3, "c"); (3, "d")]
```

In the event that one or both lists are completely empty, return the empty list. Note that in `fsharp`, calling the function as below will produce an error because `F#` cannot determine the type of the element of an empty list.

```
> zip [1;3;5;7] [];;
```

```
    zip [1;3;5;7] [];;  
~~~~~
```

```
code/stdin(14,1): error FS0030: Value restriction. The value 'it'  
has been inferred to have generic type
```

```
    val it : ((int * int * int * int) * '_a) list
```

Either define 'it' as a simple data term, make it a function with explicit arguments or, if you do not intend for it to be generic, add a type annotation.

To make empty lists work, explicitly provide a type for the return value.

```
> let xs : (int * int) list = zip [1;3;5;7] [];;  
val xs : (int * int) list = []
```

- (b) Write the inverse function, `unzip(xs: ('a * 'b) list) : 'a list * 'b list`, which behaves as follows:

```
> unzip [(1,"a"); (3,"b"); (5,"c"); (7,"de")];;  
val it : int list * string list = ([1; 3; 5; 7], ["a"; "b"; "c"; "de"])
```

- (c) Write `zip3(xs: 'a list)(ys: 'b list)(zs: 'c list) : ('a * 'b * 'c) list`, that zips three lists.

```
> zip3 [1;3;5;7] ["a";"b";"c";"de"] [1;2;3;4];;  
val it : (int * string * int) list =  
    [(1, "a", 1); (3, "b", 2); (5, "c", 3); (7, "de", 4)]
```

You must use `zip` in your definition of `zip3`.

- (d) Provide a `main` function that exercises all of the above cases, plus a few more that you think of yourself.
- (e) Why can't you write a function `zip_any` that takes a list of any number of lists and zips them into tuples? From the first part of this question it should be clear that for any fixed n , one can write a function `zipn`. The difficulty here is to write a single function that works for all n . In other words, can we write a single function `zip_any` such that `zip_any [list1;list2;...;listk]` returns a list of k -tuples no matter what k is? Answer this question using \LaTeX .

The project directory for this question should be called "q5". You should be able to run this program using "dotnet run" without any additional arguments.