CS355: Paradigms for Programming Lab

Instructions are provided on a separate sheet. Read and follow them very carefully. No grievances will be considered if you do not follow the instructions.

THE SCHEME OF THINGS

Q1 [2]. Write a Scheme procedure `f` that computes the following function:

$$f(n) = \begin{cases} n, & \text{if } n < 3 \\ f(n-1) + 2f(n-2) + 3f(n-3), & \text{if } n \geq 3 \end{cases}$$

Example interaction:

```
> (f 3)
4
```

Q2 [3]. Write a higher order Scheme procedure `filtered-fold` that folds only those elements that satisfy a given predicate. For example, assuming we have an `even?` procedure that checks whether a number is even, `filtered-fold` should work as follows:

```
> (define l (list 2 3 4 5 6 7 8))
> (filtered-fold even? + 0 l)
20
```

Q3 [7]. You are to write a simple pattern-matching program in Scheme. Your initial solution should consist of a `sublist` function that takes two lists as arguments. It should return `#t` if the first list appears as a contiguous sublist somewhere within the second list, and `#f` otherwise:

```
> (sublist '(c d e) '(a b c d e f g))
#t
> (sublist '(a c e) '(a b c d e f g))
#f
> (sublist '(f) '(a b c d e f g))
#t
```

Once you have this working, use it to build an `lgrep` function that returns the lists within a list of lists that contain a given sublist:

```
> (lgrep '(c d e) '((a b c d e f g)
                    (c d c d e)
                    (a b c d)
                    (h i c d e k)
                    (x y z)))
((a b c d e f g) (c d c d e) (h i c d e k))
```

HASKELLING A CURRY

Q4 [3]. Define a function `divide :: Int -> [a] -> ([a], [a])` that forms a pair out of a list divided at the index `n-1`. An example is given below. (Hint: You can use the standard functions `take` and `drop`.)

```
> l = [1,2,3,4,5,6,7]
> divide 3 l
([1,2,3], [4,5,6,7])
```

Q5 [4]. When the great Indian mathematician Srinivasan Ramanujan was ill in a London hospital, he was visited by the English mathematician G.H. Hardy. Trying to find a subject of conversation, Hardy remarked that he had arrived in a taxi with the number 1729, a rather boring number it seemed to him. Not at all, Ramanujan instantly replied, it is the first number that can be expressed as two cubes in essentially different ways: 13 + 123 = 93 + 103 = 1729.

Define a function `quads :: Int -> [(Int, Int, Int, Int)]` that takes an integer and returns a list of all different quadruples (a, b, c, d) in the range $0 < a, b, c, d \leq n$ such that $a^3 + b^3 = c^3 + d^3$. Example:

```
> quads 20
[(1,12,9,10),(2,16,9,15)]
```

Q6 [6]. The Great Province will conduct an election soon, and APP students have been tasked with writing a program to determine the winner. Of course, people may ask for more details about other candidates as well. Our job is to ensure a simple, modular and easily verifiable system so that it lives up to the expectations of this important democratic exercise. Result: APPers choose Haskell!

Say the votes obtained are recorded in a list of Strings. Thus:

```
votes :: [String]
votes = ["P1", "P2", "P3", "P2", "P2", "P1"]
```

denotes six votes received in favour of three candidates P1, P2 and P3.

Start by defining a function `count :: Eq a => a -> [a] -> Int` that takes the name of a candidate and a list of votes, and returns the number of votes received by that candidate:

```
> count "P1" votes
2
```

Next, to supply the complete results to the election commission, define a function `result :: Ord a => [a] -> [(Int, a)]` that collates the election results sorted by the names of candidates:

```
> result votes
[("P1", 2), ("P2", 3), ("P3", 1)]
```

Finally, for people who just care abut the breaking news, write a function `winner :: Ord a => [a] -> a` that returns the name of the winning candidate:

```
> winner votes
"P2"
```

One PC to someone who writes a single-line definition of `winner` that uses both the `result` and the `count` functions. (Feel free to use any module imports in your solution.)

LOGS SMALLER THAN PROGRAMS

Q7 [10]. In this question, you would perform basic type inference using Prolog for an aspiring programming language Kyun. This language just has three kinds of assignment "facts", denoting assignments of variables, numbers and booleans, respectively.

```
assign(v, y) // denotes "x = y"
assign(x, 10) // denotes "x = 10"
assign(x, true) // denotes "x = true"
```

Assume that numbers can only be positive integers, booleans can either be `true` or `false`, and variables can only be single-lettered alphabets.

Your task is to write a Prolog program for assigning one of the types `{int, bool, any}` to each of the variables in a given Kyun program, ignoring the order of the statements. Thus, if a variable gets two conflicting types from different parts of the program, you would infer its type as `any`. The answer should be obtainable by passing a variable name to the structure `type`, and should be printed in a way that it is the single answer without the Prolog engine waiting for any more input.

Consider the following examples, where the expected output is mentioned as a comment against each `type` query:

```
assign(x, 10).
assign(z, true).

// Your solution here

? type(x, W). // W = int
? type(z, W). // W = bool
```

```
assign(y, 20).
assign(w, y).

// Your solution here

? type(y, W). // W = int
? type(w, W). // W = int
```

```
assign(x, 10).
assign(z, true).
assign(z, x).

// Your solution here

? type(x, W).  // W = int
? type(z, W).  // W = any
```

Note that your solution should **not** have any testcases (`assign`) or queries (`type`), as we would prepend and append them from our side, for each testcase. Also, you should not use any imports. (Hint: The one-argument procedure `number` tells whether the passed argument is a number.)