General Guidelines

- For problems 1-6, all the required functions except one can be written more elegantly and concisely with a val binding rather than a fun binding, but we do *not* want to penalize repeatedly students who used fun bindings. So please follow this compromise: For problems 1-3, give a score of 5 to elegant solutions that use val bindings or fun bindings, but on problems 4-6, give at most a 4 to solutions that use a fun binding. (In problems 5 and 6, a fun binding more clearly leads to unnecessary function wrapping, and Problems 4 and 5 explicitly required a val binding.)
- Several problems required using particular other functions. In many cases, the auto-grader already checked for this, so you should not re-penalize for the same thing. In other cases, it was not feasible to check this automatically. The per-problem instructions will make clear what to look for and what has already been auto-graded.
- While this assignment did not explicitly forbid functions like `hd` and `isSome` and constructs with the `#` character, it is usually better style to use pattern-matching. So you do not need to penalize forbidden functions explicitly (it is *possible* to score a 5 with them), but we *anticipate* most excellent-style functions will use pattern matching.
- As in prior homeworks, give at most a 3 if you see code like `e1 := e2` or `!e`.

Problem 1

Here is a sample solution:

```
val only_capitals = List.filter (fn s => Char.isUpper (String.sub(s,0)))
```

- Recall we are allowing fun bindings for problems 1, 2, and 3, so this solution deserves a 5:

```
fun only_capitals xs = List.filter (fn s => Char.isUpper (String.sub(s,0))) xs
```

- While the problem explicitly allowed assuming all strings had at least one character, do not penalize a solution that is elegant but is more complicated because it deals with empty strings
- Give at most a 4 to a solution that does not use an anonymous function.
- The auto-grader already checked for using `List.filter`, `Char.isUpper`, and `String.sub`, so do not penalize elegant solutions that do not use them (though it makes the problem much more difficult not to use them).

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 2

Here is a sample solution:

```
val longest_string1 =
    List.foldl (fn (s,sofar) => if String.size s > String.size sofar
                                then s
                                else sofar)
                                ""
```

- Recall we are allowing fun bindings for problems 1, 2, and 3, so a solution like this deserves a 5:

```
fun longest_string1 xs = List.foldl (fn ...) "" xs
```

- Give at most a 4 for a solution that does not use an anonymous function that returns either a string or a string along with that string's size. (Returning the string's size is a little more complicated but arguably a little more efficient.)
- The auto-grader already checked for using `foldl` (which is also `List.foldl`) and `String.size`, so do not penalize elegant solutions that do not use them.

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 3:

Here is a sample solution:

```
val longest_string2 =
    List.foldl (fn (s,sofar) => if String.size s >= String.size sofar
                                then s
                                else sofar)
                                 ""
```

- Follow similar guidelines as for Problem 2.
- Even though the problem said to use `foldl`, do not penalize here a solution that uses `List.foldr` instead. Such a solution just makes it harder to do Problem 4.

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 4

Here is a sample solution:

```
fun longest_string_helper f =
    List.foldl (fn (s,sofar) => if f(String.size s,String.size sofar)
                                then s
                                else sofar)
                                ""
val longest_string3 = longest_string_helper (fn (x,y) => x > y)
val longest_string4 = longest_string_helper (fn (x,y) => x >= y)
```

- For `longest_string_helper`, a solution similar to the one above makes `longest_string3` and `longest_string4` easiest to write, but solutions that use the function argument `f` differently can still be okay, likely a 4 or a 3.
- For `longest_string3` and `longest_string4`, we required val bindings and partial application of `longest_string_helper`. The auto-grader did not check for this, so give at most a 3 for not following these guidelines.
- Even though these alternate solutions to `longest_string3` and `longest_string4` technically use a feature we did not show you, they are slightly more elegant, so solutions with them can receive a 5:

```
val longest_string3 = longest_string_helper op>
val longest_string4 = longest_string_helper op>=
```

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 5

Here is a sample solution:

```
val longest_capitalized = longest_string1 o only_capitals
```

- The auto-grader did already check for using the o operator, but it did not check for a val binding instead of a fun binding. So give a 4 for:

```
fun longest_capitalized xs = (longest_string1 o only_capitals) xs
```

- Give at most a 4 for solutions that are elegant but do not reuse `longest_string1` and `only_capitals`. (Of course, using `longest_string3` instead is totally fine.)
- Give at most a 3 for solutions that find a way to use `o` but not for the idea of composing finding-strings-with-capitals and finding-the-longest-string.

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 6

Here is a sample solution:

```
val rev_string = String.implode o rev o String.explode
```

- Even though the assignment did not explicitly require a val binding here, it is more elegant than a fun binding and we are grading style, so give a 4 for:

```
fun rev_string s = (String.implode o rev o String.explode) s
```

- Give a 3 or a 4 for solutions that do not use the same library functions as above since solutions are likely to be significantly more convoluted (but if there is another short way we did not think of, then that may be okay).

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 7

Here is a sample solution:

```
fun first_answer f xs =
    case xs of
        [] => raise NoAnswer
      | x::xs' => case f x of NONE => first_answer f xs'
                            | SOME y => y

```

- You can give a 5 for minor variations like using a val binding for holding the result of the call `f x` even though it just makes the solution longer, but a good solution should be a simple recursive function.
- If everything else is short and elegant, you can give a 5 for a solution that uses `isSome` and `valOf`, but most such solutions will probably be a 4 or 3.

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 8

Here is a sample solution:

```
fun all_answers f xs =
    let fun loop (acc,xs) =
        case xs of
            [] => SOME acc
          | x::xs' => case f x of
                          NONE => NONE
                        | SOME y => loop((y @ acc), xs')
    in loop ([],xs) end
```

- Follow similar guidelines as for the previous problem, understanding that this problem requires a more sophisticated solution, so solutions are naturally a little longer and more complicated.
- Do not require tail recursion as in the sample solution.

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 9

Here is a sample solution:

```
val count_wildcards = g (fn () => 1) (fn _ => 0)
val count_wild_and_variable_lengths = g (fn () => 1) String.size
fun count_some_var (x,p) = g (fn () => 0) (fn s => if s = x then 1 else 0) p
```

- Solutions that use variables for the anonymous-function arguments are fine, so for example, `val count_wildcards = g (fn x => 1) (fn x => 0)` can be part of a solution that gets a 5.
- Even though val bindings are slightly more elegant for (a) and (b), you can give a 5 for fun bindings.
- Give a 4 for unnecessary function wrapping like `(fn x => String.size x)`.

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 10

Here is a sample solution:

```
fun check_pat pat =
    let fun get_vars pat =
          case pat of
              Variable s => [s]
            | TupleP ps => List.foldl (fn (p,vs) => get_vars p @ vs) [] ps
            | ConstructorP(_,p) => get_vars p
            | _ => []
        fun unique xs =
          case xs of
              [] => true
            | x::xs' => (not (List.exists (fn y => y=x) xs'))
                        andalso unique xs'
    in
        unique (get_vars pat)
    end
```

There are multiple reasonable ways to approach this problem, so a good solution does not have to follow the algorithm above. Some other fine possibilities:

- Instead of putting all variables in a list and then looking for uniqueness, we can have an accumulator of variables-seen-so-far and then on each variable compare it against all those in the accumulator.
- A more efficient way to check for uniqueness is to sort the list of variables and then make sure no two adjacent variables are the same.
- A solution could use `count_some_var` as a helper function to check that each variable found in the pattern has a count of 1.

So just look for generally good style.

The treatment of **TupleP** is a particular place to focus. While the sample's approach of using **List.foldl** and recursion is particularly concise, it is not necessary to use fold here to get a 5 -- an explicit recursive helper function is okay.

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 11

Here is a sample solution:

```
fun match (valu,pat) =
    case (valu,pat) of
        (_,Wildcard)    => SOME []
      | (_,Variable(s)) => SOME [(s,valu)]
      | (Unit,UnitP)    => SOME []
      | (Const i, ConstP j)    => if i=j then SOME [] else NONE
      | (Tuple(vs),TupleP(ps)) => if length vs = length ps
                                  then all_answers match (ListPair.zip(vs,ps))
                                  else NONE
      | (Constructor(s1,v), ConstructorP(s2,p)) => if s1=s2
                                                   then match(v,p)
                                                   else NONE
      | _ => NONE
```

- An excellent solution would use pattern-matching on the pair of valu and pat in some way leading to the 7 branches above.
- For most of the branches, there is really only one approach, but small syntactic differences are no problem, for example **SOME ((s,valu)::[])** is fine for the variable case.
- The two most interesting and important branches are the ones that need to call match recursively. As the problem states, we are not requiring a solution to use all_answers and **ListPair.zip**, but a score of 5 should have a solution as easy to read (if not quite as short) as the above. Otherwise, give at most a 4. (Note: If not using **ListPair.zip**, you do not need to check for equal lengths first, but you do need to check for the lists having different lengths in some way.)

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Problem 12

Here is a sample solution:

```
fun first_match valu patlst =
    SOME (first_answer (fn pat => match (valu,pat)) patlst)
    handle NoAnswer => NONE
```

- The auto-grader did not check for using **first_answer**, so you should check for this. Give at most a 2 for a solution that does not call first_answer as this was required.
- Otherwise look for generally good style. While the sample solution does not use any local variables, it is fine to use them.

Remember that you are grading on general style, not how close to the sample solution a student solution is. It is perfectly fine for a solution to be significantly different from the sample, as long as it has good style.

Challenge Problem

You do not need to assess the challenge problem, but you are welcome to provide feedback.

Mark as completed