Problem 1a

```
fun all_except_option (s,xs) =
  case xs of
      [] => NONE
    | x::xs' => if same_string(s,x)
                then SOME xs'
                else case all_except_option(s,xs') of
                         NONE => NONE
                       | SOME y => SOME(x::y)

```

- It is fine for a solution to be more complicated if the reason is that it allows `s` to be in `xs` multiple times although the problem does not require handling this situation.
- It is fine to use the = operator directly instead of `same_string` even though the specification asked for `same_string`.
- A helper function is really not needed here, but many people will probably use one. Generally consider a good such solution to be worth a 4, but a 5 is okay if it is really nice.

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 1b

Here is a sample solution:

```
fun get_substitutions1 (substitutions,str) =
    case substitutions of
        [] => []
      | x::xs => case all_except_option(str,x) of
                     NONE => get_substitutions1(xs,str)
                   | SOME y => y @ get_substitutions1(xs,str)

```

- Give at most a 3 if they use helper functions other than `all_except_option` and ML's append operator `@`. This includes functions defined in get_substitutions1 as they are not helpful here.
- If a solution uses a local val binding for no useful purpose, still give a 5 if everything else is great, but this really is inferior style. For example:

```
fun get_substitutions1 (substitutions,str) =
    case substitutions of
        [] => []
      | x::xs => let val foo = all_except_option(str,x)
                 in case foo of
                        NONE => get_substitutions1(xs,str)
                      | SOME y => y @ get_substitutions1(xs,str)
                 end
```

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 1c

Here is a sample solution:

```
fun get_substitutions2 (substitutions,str) =
    let fun loop (acc,substs_left) =
        case substs_left of
            [] => acc
          | x::xs => loop ((case all_except_option(str,x) of
                                NONE => acc
                              | SOME y => acc @ y),
                           xs)
    in
        loop ([],substitutions)
    end
```

In addition to general style, the most important thing we are checking is that there is a tail recursive local helper function.

- Give at most a 2 if they do not define a helper function.
- Give at most a 3 if their helper function is not defined locally (inside `get_substitutions2`).
- Give at most a 3 if the helper function is not tail-recursive: where it calls itself (only one place is needed), there should be no more work afterward. So the result of the recursive call should not be an argument to anything like another function, `@`, etc.
- Give at most a 3 if you cannot easily find the initial call to the helper function with `[]` for an accumulator argument.

The sample solution uses a case-expression directly for the accumulator argument to the recursive call loop. This is not necessary for good style. A fine alternative is something like:

```
case all_except_option(str,x) of
    NONE => loop(acc,xs)
  | SOME y => loop(acc @ y,xs)
```

It is also okay if a solution uses local variables in good style for computing the values that are then passed to the recursive call.

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 1d

Here is a sample solution:

```
fun similar_names (substitutions,name) =
    let
        val {first=f, middle=m, last=l} = name
        fun make_names xs =
           case xs of
               [] => []
             | x::xs' => {first=x, middle=m, last=l}::(make_names(xs'))
    in
        name::make_names(get_substitutions2(substitutions,f))
    end
```

- It is fine, in fact even slightly better, to have the record pattern in the function argument, something like fun `similar_names (substitutions,{first=f, middle=m, last=l}) =`
- Give at most a 4 if they do not use one of the `get_substitutions` functions defined earlier.
- Give at most a 3 for a solution longer than 25 lines, not including comments. (Exact line count is never important, but this is over 2.5 times as long as 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 2a

Here is a sample solution:

```
fun card_color card =
    case card of
        (Clubs,_)    => Black
      | (Diamonds,_) => Red
      | (Hearts,_)   => Red
      | (Spades,_)   => Black
```

- Do not penalize solutions that do not use the wildcard pattern (something like `(Clubs,x)` for each pattern).
- Do not penalize solutions that take an argument that is a pair pattern (something like `fun card_color (suit,value) =`).
- Do not penalize solutions that use function-pattern syntax, something like:

```
 fun card_color (Clubs,_) = Black
   | card_color (Diamonds,_) = Red ...
```

- Give a 4 for this solution, which does not use nested patterns:

```
fun card_color card =
    let val (s,v) = card
    in case s of
           Clubs => Black
         | Diamonds => Red
         | Hearts => Red
         | Spades => Black
    end
```

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 2b

Here is a sample solution:

```
fun card_value card =
    case card of
        (_,Jack) => 10
      | (_,Queen) => 10
      | (_,King) => 10
      | (_,Ace) => 11
      | (_,Num n) => n
```

Follow similar guidelines as for the previous problem.

Problem 2c

Here are two sample solutions:

```
fun remove_card (cs,c,e) =
    case cs of
        [] => raise e
      | x::cs' => if x = c then cs' else x :: remove_card(cs',c,e)

fun remove_card (cs,c,e) =
    let fun f cs =
            case cs of
                [] => raise e
              | x::cs' => if x = c then cs' else x :: f cs'
    in
        f cs
    end
```

- There are unlikely to be too many ways to make solutions much more complicated without introducing poor style.
- Do not penalize solutions that define their own function to see if two cards are equal rather than using the = operator.

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

Problem 2d

Here are two sample solutions:

```
fun all_same_color cs =
    case cs of
        [] => true
      | [_] => true
      | head::neck::tail => card_color head = card_color neck
          andalso all_same_color(neck::tail)

fun all_same_color cs =
    case cs of
        head::neck::tail => card_color head = card_color neck
          andalso all_same_color(neck::tail)
      | _ => true

```

- Give at most a 4 for not having a single case expression with a nested pattern like `head::neck::tail` (though they can use other variable names). But if you see a more complicated pattern like `head::(rest as neck::tail)` do not penalize it. (It technically uses a feature we did not see, but it is arguably better than the sample solution.)
- Do not penalize solutions that use longer patterns for the one-element list than `[_]`. So all of these are also fine:

`[x]` `_::[]` `x::[]` `_::nil` `x::nil`

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 2e

Here is a sample solution:

```
fun sum_cards cs =
    let fun loop (acc,cs) =
      case cs of
          [] => acc
        | c::cs' => loop (acc + card_value c, cs')
    in
      loop (0,cs)
    end

```

As with problem 1c, our focus is on making sure there is a tail-recursive locally defined helper function:

- Give at most a 2 if they do not define a helper function.
- Give at most a 3 if their helper function is not defined locally (inside `sum_cards`).
- Give at most a 3 if the helper function is not tail-recursive: where it calls itself (only one place is needed), there should be no more work afterward. So the result of the recursive call should not be an argument to anything like another function, `+`, etc.
- Give at most a 3 if you cannot easily find the initial call to the helper function with `0` for an accumulator argument.

Other things:

- Give at most a 4 if the solution does not use `card_value` as a helper function.

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 2f

Here is a sample solution:

```
fun score (cs,goal) =
    let
        val sum = sum_cards cs
    in
        (if sum >= goal then 3 * (sum - goal) else goal - sum)
        div (if all_same_color cs then 2 else 1)
    end
```

This problem has an especially large number of reasonable solutions that would be good style. So use your best judgment. Do not necessarily penalize solutions that use conditional expressions less cleverly than the sample solution. However, give at most a 4 if `sum_cards` can be called more than once when the body of score is evaluated.

Problem 2g

Here is a sample solution:

```
fun officiate (cards,plays,goal) =
    let
        fun loop (current_cards,cards_left,plays_left) =
            case plays_left of
                [] => score(current_cards,goal)
              | (Discard c)::tail =>
                loop (remove_card(current_cards,c,IllegalMove),cards_left,tail)
              | Draw::tail =>
                (* note: must score immediately if go over goal! *)
                case cards_left of
                    [] => score(current_cards,goal)
                  | c::rest => if sum_cards (c::current_cards) > goal
                               then score(c::current_cards,goal)
                               else loop (c::current_cards,rest,tail)
    in
        loop ([],cards,plays)
    end
```

For a longer function like this, we cannot expect solutions to look much like the sample solution, so let's list some things to look for:

- A local helper function keeps track of remaining cards, moves, and held cards in some way
- Use of previously defined functions: `remove_card`, `sum_cards`, `score`
- Good use of case-expressions and pattern-matching. Do not require the nested patterns in the outer case expression above where we have different patterns for the head of the list -- a nested case expression is okay instead.
- A moderately longer solution is okay, but a solution twice as long (in terms of amount of code -- do not worry about how much is put on each line) is probably doing something unnecessary.

Use the general scoring guidelines: 5 for a great solution, 4 for one with a small number of issues, etc.

Problems 3a and 3b

You do not need to provide feedback on problems 3a and 3b (the challenge problems), but you are welcome to give text feedback on these problems here if you wish.

Mark as completed