Computer Science 320
Concepts of Programming Languages
Problem Set 7: Pattern-Matching in SML

Out: Friday, March 17, 2006
Due: ***4:59 PM*** on Friday, March 24, 2006

There are 10 problems in this set, each worth 10 points, except for the last which is worth 20 points. The harder problems are marked with a a single * (average difficulty) or two ** (higher-than-average difficulty). For the easy points, start with the unmarked problems.

A major difference between Scheme and SML has to do with types. But this is only the most obvious difference between the two languages. There are several others that are no less significant, at least in the way programmers write their code.

One such difference is *pattern-matching*, which Scheme does not support. Pattern-matching is particularly useful in the manipulation of lists. At first glance, this may not appear as a big advantage, as most operations on lists using pattern-matching are easily re-written using `cons`, `car`, `cdr` and `null?` in Scheme (or also using `::`, `hd`, `tl` and `null` in SML). For many problems, however, pattern-matching is more convenient to use and requires far less code to write. Experienced SML programmers know how to exploit this feature, so that the writing of code becomes less error-prone and easier to document.

In all solutions for this assignment, you are not allowed to use the SML primitive operators `tl` ***and*** `hd`***. And you have to use pattern-matching as much as possible. See handouts*** `HD21.list-ops.sml` ***and*** `HD22.list-ops-again.sml` ***for several examples.***

Problem 1 The standard recursive definition of Euclid's Algorithm for the *greatest common divisor* (gcd) in SML can be written as:

```
fun gcd (m,n) =
      if m = 0 then n else gcd(n mod m, m);
```

Write a new definition of `gcd` using pattern-matching and no conditional (i.e., `if`...`then`...`else`...). (The `gcd` function is discussed on page 48 of Paulson's book – henceforth referred to as just [Paulson].)

Problem 2 The function `duplicate` consumes a list `L` and returns a list where every entry in `L` is duplicated. For example, `(duplicate [1,2,3])` returns the list `[1,1,2,2,3,3]` and `(duplicate [true,false])` returns the list `[true,true,false,false]`. Write a definition of `duplicate` using pattern-matching, no `hd` and `tl`, and no conditional (i.e., `if`...`then`...`else`...).

** **Problem 3** Exercise 2.19, page 53, in [Paulson]. The definition of the `GCD` function should use pattern-matching as much as possible and the conditional (i.e., `if`...`then`...`else`...) as little as possible. *Hint 1*: Try to first write the definition of `GCD` using conditionals and no pattern-matching, and then eliminate conditionals (whenever possible) using pattern-matching. *Hint 2*: Note that one case is missing in the specification of `GCD` on page 53, namely, when the first argument of `GCD` is odd and the second argument of `GCD` is even. You will have to account for this missing case in your code.

Problem 4 The function `maxList` consumes a list `L` of strings and returns the largest string in `L` – "largest" according to lexicographic ordering. For example, `(maxList ["ab", "bac", "abde"])` returns `"bac"`.

```
fun maxList (L: string list) =
     if tl(L) = []                     (* L is a single element *)
     then hd(L)                        (* the single element is the maximum *)
     else                              (* assume there are at least 2 elements *)
        if hd(L) > hd(tl(L))           (* the first element exceeds the second *)
        then maxList(hd(L)::tl(tl(L))) (* eliminate second element *)
        else maxList(tl(L))            (* eliminate first element *);
```

Write a new definition of `maxList` using pattern-matching, and no `hd` and `tl`. You may have to use conditionals, but you will get credit for minimizing their use.

Problem 5 Consider the function `prod` on page 75 of [Paulson], in the last paragraph preceding Exercise 3.1. Write a new definition of `prod` which uses pattern-matching, and no `hd` and `tl`.

Problem 6 Exercise 4.1, page 126, in [Paulson]. You first need to read pages 124-126 preceding the exercise.

Problem 7 Exercise 4.2, page 126, in [Paulson]. You first need to read pages 124-126 preceding the exercise.

Problem 8 Consider the following definition in SML:

```
fun foo []      = [ [] ]
  | foo (x :: xs) =
      let  val s =  foo xs
      in
           (map ( fn e => (x :: e) ) s) @ s
      end;
```

Given a list `L` as input, what does `(foo L)` return as output? Justify in a couple of lines.

* **Problem 9** *(20 points)* Suppose we represent sets by lists, where an entry occurs at most once. Use pattern-matching, and no `hd` and `tl`, to define the following in SML:

1. The function `member` consumes a pair `(x,S)` and returns `true` if `x` is a member of the set represented by the list `S`, and `false` otherwise.

2. The function `delete` consumes a pair `(x,S)` and returns a list representing the set `S` $-$ {`x`}, i.e., the element `x` is deleted from `S`. Remember you may assume that `x` appears at most once in `S`.

3. The function `insert` consumes a pair `(x,S)` and returns a list representing the set `S` $\cup$ {`x`}. Remember there cannot be two occurrences of `x` in a list representing a set.

You will get credit for minimizing the use of conditionals. Assume that two members of the set represented by the list `S` can be compared using "=", i.e., they belong to one of the so-called *equality types*: integer, boolean, character, and string.