CSE 341 Section Handout #2
Cheat Sheet

Patterns

```
fun name(pattern1) = expression1
|   name(pattern2) = expression2
...
|   name(patternN) = expressionN;
```

Examples:

```
(* Computes n!, or 1 * 2 * 3 * ... * n-1 * n.
   Precondition: n >= 0. *)
fun factorial(0) = 1
|   factorial(n) = n * factorial(n - 1);

(* Computes the sum of the elements of a list of integers. *)
fun sum([]) = 0
|   sum(first :: rest) = first + sum(rest);
```

`let` expressions (1)

```
(* for a 'local variable' *)
let
    val name = expression
in
    expression
end;
```

Example:

```
(* Computes x^100.  A somewhat silly function. *)
fun hundredthPower(x: real) =
    let
        val fourth = x * x * x * x
        val twentieth = fourth * fourth * fourth * fourth * fourth
    in
        twentieth * twentieth * twentieth * twentieth * twentieth
    end;
```

`let` expressions (2)

```
(* for a 'helper function' *)
let
    fun name = expression
in
    expression
end;
```

Example:

```
(* Computes the least common multiple (LCM) of x and y. *)
fun lcm(x, y) =
    let
        fun gcd(x, 0) = x
        |   gcd(x, y) = gcd(y, x mod y)
    in
        x * y div gcd(x, y)
    end;
```

CSE 341 Section Handout #2
Questions

(When solving these problems, use pattern matching rather than using `if-then-else` *statements or calling functions like* `length`*,* `hd`*, and* `tl`*. Also use* `let` *declarations as necessary to help you solve the problems.*

1. Write a function `twoSame` that produces `true` if a list of values has two consecutive values that are equal. For example, the call of `twoSame([5, ~3, 19, 19, 2, 24, 7])` would produce `true`.

2. Write a function called `stutter` that takes a list as an argument and that produces the list formed by replacing each value in the list with two of that value. For example, the call of `stutter([1, 2, 3])` should produce `[1,1,2,2,3,3]`.

3. Write a function called `stutterString` that produces the result of replacing each character of a string with two of that character. For example, the call of `stutterString("hello")` should produce `"hheelloo"`.

4. Write a function `isPrime` that takes an integer *n* and that produces `true` if *n* is a prime number and `false` if it is not. For example, the call of `isPrime(1031)` should produce `true`. By definition, a number is prime if it is divisible only by itself and 1. Your function should produce `false` for all numbers less than 2.

5. Consider the following inefficient attempt to compute the maximum value in a list:

```
fun bad_max([x]) = x
|   bad_max(x::xs) =
        (print(".");
        if x > bad_max(xs) then x
        else bad_max(xs));

fun max([x]) = x
|   max(x::xs) =
        let val m = max(xs)
        in (print("."); if x > m then x else m)
    end;
```

What is the complexity of this function? Rewrite it to be O(*n*).

6. Write a function called `cycle` that takes an integer *n* and a list and that produces the list obtained by moving the first *n* values to the end of the list. For example, `cycle(4, [1, 2, 3, 4, 5, 6])` should produce `[5,6,1,2,3,4]`. You may assume that *n* is less than or equal to the length of the list. (HINT: Cycle a single value to the end of the list *n* different times.)

7. Write a variation of the function described in problem 5 called `cycle2` that computes its result efficiently. In particular, it should only perform a single list append. You may call the built-in function `rev` which is an efficient implementation of the reverse operation.

CSE 341 Section Handout #2
Solutions

1.
```
fun twoSame([]) = false
|   twoSame([x]) = false
|   twoSame(x::y::rest) = (x = y) orelse twoSame(y::rest);
```

2.
```
fun stutter([]) = []
|   stutter(x::xs) = x::x::stutter(xs);
```

3.
```
fun stutterString(str) = implode(stutter(explode(str)));
```

4.
```
fun isPrime(2) = true
|   isPrime(n) =
        let
            fun explore(m) =
                if m >= n then true
                else n mod m <> 0 andalso explore(m + 2)
        in
            n > 2 andalso n mod 2 <> 0 andalso explore(3)
        end;
```

5. The `bad_max` function will be O(*n*) for a list that is in reverse sorted order, but it will be O(2^n) for a list in sorted order. The method becomes linear when you introduce a `let`:
```
fun max([x]) = x
|   max(x::xs) =
        let
            val m = max(xs)
        in
            if x > m then x else m
        end;
```

6.
```
fun cycle(0, lst) = lst
|   cycle(n, x::xs) = cycle(n - 1, xs @ [x]);
```

7.
```
fun cycle2(n, lst) =
    let
        fun loop(0, list, back) = list @ rev(back)
        |   loop(n, x::xs, back) = loop(n - 1, xs, x::back)
    in
        loop(n, lst, [])
    end;
```