National University of Singapore
School of Computing
CS1101S: Programming Methodology
Semester I, 2016/2017

Recitation 6
Lists and List Processing II

List Library

Recall the following library:

```
function map(f, xs) {
    return (is_empty_list(xs))
        ? []
        : pair(f(head(xs)), map(f, tail(xs)));
}

function filter(pred, xs){
    if (is_empty_list(xs)) {
        return xs;
    } else {
        if (pred(head(xs))) {
            return pair(head(xs), filter(pred, tail(xs)));
        } else {
            return filter(pred, tail(xs));
        }
    }
}

function accumulate(op, initial, sequence) {
    if(is_empty_list(sequence)) {
        return initial;
    } else {
        return op(head(sequence), accumulate(op, initial, tail(sequence)));
    }
}

var fold_right = accumulate;
```

Problems:

1. Suppose `xs` is bound to `list(1, 2, 3, 4, 5, 6, 7)`. Using `map`, `filter`, and/or `accumulate`, write an expression involving `xs` that returns:

   (a) `list(1, 4, 9, 16, 25, 36, 49)`

(b) `list(1, 3, 5, 7)`

(c)
```
list(list(1, 1), list(2, 2), list(3, 3), list(4, 4), list(5, 5),
       list(6, 6), list(7, 7))
```

(d) `list(list(2), list(list(4), list(list(6), false)))`

(e) The maximum element of `xs`: 7

(f) The last pair of `xs`: `pair(7, list())`

2. A *set* is represented by an *unordered list* that contains no duplicate elements. You are to implement the set operations *difference*, *union*, and *intersection*. In your functions, you should use the higher-order functions `filter`, `map` and `accumulate` whenever possible. You can also make use of the `remove_duplicates` function that you wrote for Discussion Group Exercises Week 6.

   (a) Write a function `set_difference` that takes in two sets as its two arguments and returns a set containing elements of the first set that are not in the second set. The order of the elements in the returned set does not matter. What is the order of growth in time for your function?

   ```
   function set_difference(xs, ys) {
       ...
   }
   ```

   Example calls:

   ```
   set_difference(list(1, 4, 3, 2), list(3, 2, 4));
   // Result: list(1)

   set_difference(list(1, 2, 3, 4), list(4, 5, 6));
   // Result: list(1, 2, 3)
   ```

   (b) Write a function `set_union` that takes in two sets as its two arguments and returns a set that contains elements that belong to either the first set or to the second set or to both. The order of the elements in the returned set does not matter. What is the order of growth in time for your function?

```
function set_union(xs, ys) {
    ...
}
```

Example calls:

```
set_union(list(1, 2, 3, 4), list(2, 3, 4));
// Result: list(1, 2, 3, 4)

set_union(list(1, 2, 3, 4), list(4, 5, 6));
// Result: list(1, 2, 3, 4, 5, 6)
```

(c) Write a function `set_intersection` that takes in two sets as its two arguments and returns a set containing only elements that belong to both the first and second sets. The order of the elements in the returned set does not matter. What is the order of growth in time for your function?

```
function set_intersection(xs, ys) {
    ...
}
```

Example calls:

```
set_intersection(list(1, 2, 3, 4), list(2, 3, 4));
// Result: list(2, 3, 4)

set_intersection(list(1, 2, 3, 4), list(4, 5, 6));
// Result: list(4)
```

3. **Homework:** Suppose a *set* is now represented by an *ordered list* that contains no duplicate elements. Define the set operation functions `set_difference`, `set_union`, and `set_intersection`. What is the order of growth in time for each function?