National University of Singapore
School of Computing
CS1101S: Programming Methodology
Semester I, 2013/2014

Recitation 5
Lists and List Processing I

JediScript

1. *v1*`===`*v2* - returns `true` if *v1* is identical to *v2*. This means they are exactly the same in case of numbers, boolean values or string, or come from the same (function, pair, empty list) creation, in case of function values, pairs and empty lists.
2. `equal(`*v1*`, `*v2*`)` - returns `true` if *v1* and *v2* enjoy structural equality. This is the case, if they are both the empty list. If they are both pairs, their head and tail need to enjoy structural equality. In all other cases, they must be identical (`===`).
3. `member(`*object*`, `*list*`)` - returns the first `tail` of *list* whose `head` is equal to *obj*, or `false`.

Problems:

1. Give printed value.

```
equal(1, 1) => true

equal(0, 1) => false

equal("foo", "foo") => true

equal("foo", "bar") => false

equal(0, "0") => false

equal(false, false) => true

equal(false, "false") => false

equal(pair(1, 2) , pair(1, "2")) => false

equal(pair(1, 2) , pair(1, 2)) => true

equal(list(1, 2, 3, 4, 5) , list(1, 2, 3, 4, 5)) => true

equal(list(list(1,2), list(2,3), list(4,5)), list(list(1,2), list(2,3), list(4,5))) => true

equal(list(list(1,2), list(2,3), list(4,5)), list(list(1,2), list(2,3))) => false

equal(list(list(1,2), list(2,3)), list(list(1,2), list(3,2))) => false
```

```
equal(list(), list()) => true

equal(list(), list(1)) => false

equal(list(1), pair(1,[])) => true
```

2. **[SICP Ex 2.38]**. The `accumulate` procedure is also known as `fold_right`, because it combines the first element of the sequence with the result of combining all the elements to the right. There is also a `fold_left`, which is similar to `fold_right`, except that it combines elements working in the opposite direction:

```
function fold_left(op, initial, sequence) {
    function iter(result, rest) {
        if(is_empty_list(rest)) {
            return result;
        } else {
            return iter(op(result, head(rest)), tail(rest));
        }
    }
    return iter(initial, sequence);
}

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;
```

What are the values of

```
function div(x, y) {
    return x / y;
}

fold_right(div, 1, list(1,2,3)) => 1.5
fold_left(div, 1, list(1,2,3)) => 1/6
fold_right(pair, list(), list(1,2,3)) => [1, [2, [3, []]]]
fold_left(pair, list(), list(1,2,3)) => [[[[], 1], 2], 3]
```

Give a property that op should satisfy to guarantee that `fold_right` and `fold_left` will produce the same values for any sequence.

Assume that the sequence at hand is X = (x_1 x_2 x_3 ... x_n).
`fold_right(op, initial, X)` => (x_1 op (x_2 op (x_3 op (... (x_n op initial) ....))))

`fold_left(op, initial, X)` => ((...(((initial op x_1) op x_2) op x_3) ....) op x_n)

`op` **should** then be a commutative as well as associative operator (e.g. addition, multiplication, union-set, and intersection-set) to ensure that `fold_left` and `fold_right` return the same values for any sequence.

Lesson learnt: be careful in converting a recursive `fold_like` function into its iterative version.