מושגים בשפות תכנות

תרגיל 5

להגשה עד 25/01/2017

1. Functions `map` and `reduce` are standard functions from traditional functional programming that achieved broader recognition as a result of Google's MapReduce method for processing and generating large data sets. While `map`, `reduce`, and a number of related functions are provided in many JavaScript implementations, `map` and `reduce` can also be defined relatively simply in JavaScript as follows:

```
function map (f, inarray) {
    var out = [];
    for(var i = 0; i < inarray.length; i++) {
        out.push( f(inarray[i]) )
    }
    return out;
}
function reduce (f, inarray) {
    if (inarray.length <= 1) return;
    if (inarray.length == 2) return f(inarray[0], inarray[1]);
    var r = inarray[0];
    for(var li = 1 ; li < inarray.length ; li++) {
        r = f(r, inarray[li]);
    }
    return r;
}
```

Function `map(f, inarray)` returns an array constructed by applying `f` to every element in `inarray`. Function `reduce(f, inarray)` applies the function `f` of two arguments to elements in the list, from left to right, until it reduces the list to a single element. For example:

```
js> map( function(x){return x+1}, [1,2,3,4,5])
2,3,4,5,6
js> reduce( function(x,y){return x+y}, [1,2,3,4,5])
15
js> reduce( function(x,y){return x*y}, [1,2,3,4,5])
120
```

These functions can be combined in various ways. For each of the following questions, you may use a JavaScript implementation to test your answer yourself, but turn your solution in as part of a written description for manual grading.

a. Explain how to use `map` and `reduce` to compute the sum of the first five squares, in one line. (The sum of the first three squares is $1^2 + 2^2 + 3^2$)

b. Explain how to use `map` and `reduce` to count the number of positive numbers in an array of numbers.

c. Explain how to use `map` and/or `reduce` to "flatten" an array of arrays of numbers, such as [[1,2],[3,4],[5,6],[7,8,9]], to an array of numbers. (Hint: Look for built-in JavaScript concatenation functions.)

2. This problem asks you to compare two sections of code. The first one has three declarations and a fourth statement that consists of an assignment and a function call inside curly braces:

```
var x = 5;
function f(y) { return (x + y) - 2 };
function g(h) { var x = 7; return h(x) };
{ var x = 10; z = g(f) };
```

The second section of code is derived from the first by placing each line in a separate function, and then calling all the functions with empty argument lists. In effect, each "`(function () {`" begins a new block because the body of each JavaScript function is in a separate block. Each "`})();`" closes the function body and calls the function immediately so that the function body is executed.

```
(function () {
    var x = 5;
    (function () {
        function f(y) {return (x + y) - 2};
        (function () {
            function g(h) {var x = 7; return h(x)};
            (function () {
                var x = 10; z = g(f);
            })();
        })();
    })();
})();
```

a. What is the value of `g(f)` in the first code example?

b. The call `g(f)` in the first code example causes the expression `(x+y)-2` to be evaluated. What are the values of `x` and `y` that are used to produce the value you gave in (a)?

c. Explain how the value of `y` is set in the sequence of calls that occur before `(x+y)-2` is evaluated.

d. Explain why x has the value you gave in (b) when `(x+y)-2` is evaluated.

e. What is the value of `g(f)` in the second code example?

f. The call `g(f)` in the second code example causes the expression `(x+y)-2` to be evaluated. What are the values of `x` and `y` that are used to produce the value you gave in (e)?

g. Explain how the value of `y` is set in the sequence of calls that occur before `(x+y)-2` is evaluated.

h. Explain why x has the value you gave in (f) when `(x+y)-2` is evaluated.

3. Examine following JavaScript code, that contains two implementations of a function calculating the Fibonacci sequence, also available at:
http://www.cs.tau.ac.il/~msagiv/courses/pl17/fibonacci.js

```
var naive_fibonacci = function f (n) {
    return (n===0 || n === 1) ? n : f(n-1) + f(n-2);
}

var fibonacci = (function () {
    var memo = [0, 1];
    var fib = function f (n) {
        var result = memo[n];
        if (typeof(result) === "undefined") {
            result = f(n-1) + f(n-2);
            memo[n] = result;
        }
        return result;
    };
    return fib;
})();
```

a. Try to compute the 100th Fibonacci number with both versions and see what happens. What is the time complexity to calculate the nth Fibonacci number in each implementation? Explain the differences.

b. Explain the purpose of the variable `memo`.

c. Where is the variable `memo` in scope?

d. When during the run-time is the variable `memo` live in memory?

e. Why was the variable `memo` defined like that, and what's the purpose of the anonymous function without parameters?

f. Use a similar technique to create a function `memoize`, that takes a function as an argument, and returns a new function which implements the given function with memoization (assume the given function takes a single numeric argument). Your implementation should allow the following code:

```
var cool_fibonacci = memoize(function(n) {
    return (n===0 ||n === 1) ? n : cool_fibonacci(n-1) +
cool_fibonacci(n-2);
});
console.log(cool_fibonacci(100) + " wow, this was fast!");
```

Note: For this question, sections (a)-(e) should be submitted in the PDF, and section (f) should be solved in the `fibonacci.js` file, which should be submitted.

4. The following files contain a JavaScript implementation of the famous Minesweeper game:
http://www.cs.tau.ac.il/~msagiv/courses/pl17/mineswex.html
http://www.cs.tau.ac.il/~msagiv/courses/pl17/mineswex.js
When loaded in a web browser, the files present a 2-dimensional board of cells. Each cell may or may not contain a mine (mines are placed randomly). When the user clicks a cell that contains a mine, the cell is painted red (and the user lost the game). When the user clicks a cell that does not contain a mine, the number of mines *around* that cell is revealed (i.e., the number of mines in the 8 adjacent cells).

   Your task is to modify the code such that when the user reveals a 0 cell (a cell with no mines around it), the cells surrounding it are also revealed automatically. If any of them is also a 0 cell, automatic revealing continues. Use high-order functions to intercept the event of a cell being fully revealed. Notice the fade-in effect, which must also be preserved by automatic revealing.

   Hint: refer to the functions `get_cell` and `is_cell_hidden` which are already implemented in the code, and use them. Their functionality and synopsis are described in the documentation block above them. These functions are implemented using the `jQuery` library.

5.

a. Give a step-by-step explanation of the type inference of the following OCaml function by the Hindley-Milner type inference algorithm:

```
let rec append x y =
    match x with
    | [] -> [y]
    | hd::tl -> append tl y
```

b. Is there something in the inferred type that indicates an error in the code?

c. Fix the error in the code (submit the fixed code in the PDF).

d. Explain the type of the fixed function, and the difference in the run of the Hindley-Milner algorithm on it compared to the run you described in (a).

6. **Global Bonus (25 points, global to all exercises)**
Implement the Hindley-Milner algorithm in JavaScript, and create a working web-page that allows the user to write an expression in "Baby ML", and interactively see the inferred type of the expression. Your implementation should include a lexer, a parser, and an implementation of Hindley-Milner type inference.

"Baby ML" is defined by the following grammar:

`e ::= integer_literal`	integer literals such as 0,1,2,...
`| true | false`	Boolean literals
`| id`	identifier
`| (e)`	parenthesized expression
`| (fun id -> e)`	lambda expression
`| (e e)`	function application
`| let id=e in e`	let expression
`| letrec id=e in e`	recursive let expression

The built-in primitive types of "Baby ML" are integer and boolean, and it also contains function types of the form `'a -> 'b` and pair types of the form `'a * 'b`. The user's code can also use the following built in functions:

```
plus : int -> int -> int
ite : bool -> 'a -> 'a -> 'a
pair : 'a -> 'b -> ('a * 'b)
```

Submit your solution in `hindley_milner.html` and `hindley_milner.js` (you may split the code to multiple `.js` files as long as you submit all of them).

בהצלחה!