CS 345 - Programming Languages
Fall 2010

Homework #6

Due: 2pm CST (in class), November 18, 2010

YOUR NAME: ______________________________

Collaboration policy

No collaboration is permitted on this assignment. Any cheating (*e.g.*, submitting another person's work as your own, or permitting your work to be copied) will automatically result in a failing grade. The Computer Science Department Code of Conduct can be found at `http://www.cs.utexas.edu/academics/conduct/`.

Late submission policy

This homework is due at the **beginning of class** on **November 18**. All late submissions will be subject to the following policy.

You start the semester with a credit of 3 late days. For the purpose of counting late days, a "day" is 24 hours starting at 2pm on the assignment's due date. Partial days are rounded up to the next full day. You are free to divide your late days among the take-home assignments any way you want: submit four assignments 1 day late, submit one assignment 3 days late, *etc.* After your 3 days are used up, no late submissions will be accepted and you will automatically receive 0 points for each late assignment.

You may submit late assignments to Vitaly Shmatikov (CSA 1.114—slide under the door if the office is locked). **If you are submitting late, please indicate how many late days you are using.**

Write the number of late days you are using: ______

Homework #6 (35 points)

Problem 1 (2 points)

In his Turing Award lecture, John Backus argued that programs written in functional languages are more suited to parallel execution than imperative programs because subexpressions without side effects can be evaluated in any order.

Give at least two concrete reasons why functional programming languages do not provide a complete solution to the problem of writing programs that can be executed efficiently in parallel.

Problem 2 (4 points)

Write a tail-recursive implementation of the Scheme `foldr` function.

Problem 3 (4 points)

Write a Scheme function `(mean lst)` which computes the mean of a list of integers but traverses the list only once. Your function must compute both the sum and the length of the list in one pass and return the mean.

Problem 4 (5 points)

Write a Prolog program `prodsum` which, given two lists, computes the product of pairwise sums of their elements. For example, the query `prodsum([2,3,4], [3,4,5], X)` should return `X = 315` because $(2+3) \cdot (3+4) \cdot (4+5) = 315$. You can assume that the lists have equal length.

Is your function tail-recursive? Explain.

Problem 5

Consider the following Prolog program:

```
split(_, [], [], []).
split(P, [X|Xs], [X|Ls], Bs) :- P>X, split(P, Xs, Ls, Bs).
split(P, [X|Xs], Ls, [X|Bs]) :- P=<X, split(P, Xs, Ls, Bs).
```

Problem 5a (2 points)

What does this program do?

Problem 5b (4 points)

Suppose the program is rewritten as:

```
split(_, [], [], []).
split(P, [X|Xs], [X|Ls], Bs) :- P>X, !, split(P, Xs, Ls, Bs).
split(P, [X|Xs], Ls, [X|Bs]) :- split(P, Xs, Ls, Bs).
```

What does this program do now? Explain the difference between the two programs on a concrete example.

Problem 6 (4 points)

Describe at least two common Web programming tasks for which higher-order functions are very useful.

Problem 7

This problem asks you to compare two sections of JavaScript 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 functions with empty argument lists. 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);
         })()
       })()
    })()
})()
```

Problem 7a (2 points)

What is the value of `g(f)` in the first chunk of code?

Problem 7b (3 points)

Which values of `x` and `y` are used when evaluating the expression `(x+y)-2` in the first chunk of code? Why?

Problem 7c (2 points)

What is the value of `g(f)` in the second chunk of code?

Problem 7d (3 points)

Which values of `x` and `y` are used when evaluating the expression `(x+y)-2` in the second chunk of code? Why?