CSC324 Spring 2007 March 8, 2007
St. George Campus University of Toronto

Homework Assignment #4
Due Date: Monday March 19, 2007, by 1pm

Silent Policy

A silent policy will take effect 24 hours before this assignment is due. This means that no question about this assignment will be answered, whether it is asked on the newsgroup, by email, or in person.

Total Marks

This assignment is out of 40 marks and constitutes 5% of your final course grade.

Handing in this Assignment

What to hand in on paper:

Please use an *unsealed* letter-sized envelope, attaching the cover page provided to the front. Note that without a properly completed and signed cover page your assignment will not be marked. Please place your solutions to all questions inside the envelope. You must hand in this part of the assignment in the CSC324 drop box in the Bahen Computer Lab, BA2220.

What to hand in electronically:

In addition to your paper submission, you must submit your code electronically. Use the file names specified in the questions. To submit the files electronically, use the CDF secure Web site:
`https://www.cdf.utoronto.ca/students`

Warning: marks will be deducted for incorrect submission. Note that if the code submitted electronically differs from the code submitted on paper, we will only mark the electronically submitted version (if, in such a case, you put comments, etc. only on paper, we will mark the question as if no comments, etc. were provided).

Since we will test your code electronically, you must:

- *make certain that your code runs on CDF*,
- use the exact file names specified,
- use the exact function names and argument specified,
- not load any files,
- not display anything but the function output (no text messages to the user, fancy formatting, etc.)

Clarification Page and Newsgroup

Important corrections (hopefully few or none) and clarifications to the assignment will be posted on the Assignment 4 Clarification page, linked from the CSC324 home page. You are also responsible for monitoring the CSC324 newsgroup.

How you will be marked

Code will be marked with respect to correctness, style, and documentation.
Warning: Code that doesn't type check will receive a mark of zero for correctness.

Assignment #4

Due Date: This assignment is due on **Monday March 19, 2007 at 1pm**.

Please include this cover sheet, when handing in your paper assignment.

Last Name:

First Name:

Email:

CDF Login:

Student Number:

Grace days used to date:

Grace days used for this assignment:

Date and Time you are handing this in:

All answers are my own, written in isolation without help from others. This submission is in accordance with the University of Toronto Code of Behaviour on Academic Matters (http://www.artsandscience.utoronto.ca/ofr/calendar/rules.htm#behaviour).

Signature ______________________________

Question 1. (20 marks)

Submit all your code in a file named *a4-1.sml*.

For this question, you will write a mathematical expression evaluator for a simple calculator language. Your evaluator will take a program in the form of parsed abstract syntax, and execute it, producing an integer resulting from performing all of the mathematical operations (including applications of user-defined functions) designated by the program.

You can assume that the user's program has been parsed into an abstract syntax represented by the datatype **Expr**, as follows:

```
type Name = string;

datatype Expr = Const of int
                        | Plus of Expr * Expr
                        | Mult of Expr * Expr
                        | Neg of Expr
                        | App of Fun * Expr
                        | Var of Name
and      Fun = Abs of Name * Expr;
```

Each element of type **Expr** represents a parsed expression of our simple calculator language. An expression can be a constant integer, addition or multiplication of two expressions, the negation of an expression, or the application of a function to an expression. Functions are defined by a variable bound by the function and an expression representing the function's body. For example, the Scheme function `(lambda (x) (* 2 (+ x 1)))` would be represented as:

```
Abs("x", Mult(Const 2, Plus(Var "x", Const 1)))
```

In the subquestions that follow, you will write fucntions necessary for evaluating expressions in our simple calculator language.

a. (5 marks) Define a function **substitute : (Name, Expr, Expr) → Expr**. Let us refer to the elements in **substitute**'s argument tuple as $(n, e1, e2)$. n is a **Name** representing the name of a variable, e.g., `"x"`. $e1$ and $e2$ are expressions, **Expr**. The function **substitute** replaces every occurrence of Var n in $e2$, with $e1$. For example:

```
- substitute ("x", Var "y", Var "x");
val it = Var "y" : Expr
- (* substitute every "x" in expression Var "x" by expression Var "y". *)

- substitute ("z", Var "y", Var "x");
val it = Var "x" : Expr
- (* substitute every "z" in expression Var "x" by expression Var "y".  There is none. *)

- substitute ("z", Var "y", Const 1);
val it = Const 1 : Expr
- (* substitute every "z" in expression Const 1 by expression Var "y" *)

- substitute ("z", Var "y", Neg(Plus(Var "x", Var "z")));
val it = Neg (Plus (Var "x",Var "y")) : Expr
- (* substitute every "z" in expression Neg(Plus(Var "x", Var "z")) by expression
Var "y". *)
```

```
- substitute ("z", Var "y", App(Abs("x", Plus(Var "z", Var "x")), Const 3));
val it = App (Abs ("x",Plus (Var "y",Var"x")),Const 3) : Expr
- (* substitute every "x" in expression App(Abs("x", Plus(Var "z", Var "x")), Const3)
by expression Var "y". *)
```

b. (10 marks) Write an evaluator for closed expressions of type `Expr`. You should write a single function **eval : Expr** → **int** that reduces a given closed expression to an integer by evaluating the expression. Constants cannot be further simplified, and the mathematical operations `Plus`, `Mult`, and `Neg` are evaluated in the obvious fashion, so that:

```
- eval (Const 2);
val it = 2 : int

- eval (Plus ((Const 2), (Const 3)));
val it = 5 : int

- eval (Neg (Mult ((Const 2), (Const 3))));
val it = ~6 : int
```

Function applications are evaluated by evaluating the body of the applied function after substituting every instance of the function's bound variable with the value of the second part of the function application. For example:

```
- eval (App (Abs ("x", Plus (Var "x", Const 2)), (Const 3)));
val it = 5 : int
- (* The initial expression is equal to eval (Plus (Const 3, Const 2)) *)

- eval (App (Abs ("x", Plus (Var "x", Var "x")), Plus (Const 12, Const 9)));
val it = 42 : int
- (* The initial expression is equal to
         eval (Plus (Plus (Const 12, Const 9), Plus (Const 12, Const 9)) *)

- eval (App(Abs("x", App(Abs("y", Plus(Var "x", Var "y")), Var "x")), (Const 3)));
val it = 6 : int
- (* The initial expression is equal to
         eval (App(Abs("y", Plus((Const 3), Var "y")), (Const 3))) *)
- (* which is equal to eval (Plus((Const 3), (Const 3))) *)
```

If an expression is not closed (i.e. if it contains variables that are outside of function definitions, or not mentioned in function definitions) then you should raise an exception, **EvalException**. For example, evaluating the following should raise an exception:

```
- eval (Var "x");
uncaught exception EvalException
  raised at: A4-solns.sml:22.29-22.42

- eval (Plus (Var "x", Const 3));
uncaught exception EvalException
  raised at: A4-solns.sml:22.29-22.42

- eval (App (Abs ("y", Plus(Var "y", Var "x")), Const 3));
uncaught exception EvalException
  raised at: A4-solns.sml:22.29-22.42
```

c. (5 marks) Notice that our tiny programming language is not quite completely functional, in the sense that functions cannot be passed as parameters to functions, or returned from functions. Answer each of the following questions (Just put the first two in comments in your submission.):

c-1 What about the datatype `Expr` prevents returning functions as values?

c-2 Briefly state how to fix this problem.

c-3 Write a new datatype declaration `BetterExpr` in which functions are first-class expressions that may be returned from functions and passed as arguments to functions. Don't worry about enforcing correctness of expressions in the datatype declaration (i.e. "What happens if I try to add two functions?" or "What happens if I try to apply something that isn't a function"). That's what a type-checker would do if we had one.
Hint: You should only have to add one line to Expr and change one line already in Expr.

Question 2. (12 marks)

Submit all your code in a file named *a4-2.sml*.

We call x a fixed point of function f if $f(x) = x$. Write a function **(fix f) : (”a → ”a) → (”a → ”a)** that, given a unary function `f`, returns a function that applies `f` to its argument repeatedly until a fixed point is reached (i.e. until `(f x = x)`). Note that the functions returned by `fix` may not terminate for all inputs. For example, `(fix (fn x =>  x))` never reaches a fixed point if we apply it to anything other than 0. Don’t worry about these cases, we’ll just assume that they are *supposed* to run forever.

Here are some examples. Let’s define two functions, **mytail** and **halve** as:

```
fun mytail [] = []
|   mytail (x::xs) = xs ;

fun halve x = x div 2;
```

Then, `fix` should behave as follows:

```
- (fix half) 2;
val it = 0 : int
- (fix half) 100;
val it = 0 : int
- (fix half) ~100;
val it = ~1 : int
- (fix mytail) [];
val it = [] : ?.X1 list
- (fix mytail) [1, 2, 3];
val it = [] : int list
```

In the questions that follow, you will write three different implementations of `fix`, called `fixa`, `fixb`, and `fixc`

a. (4 marks) Write function **fixa** using a globally defined helper function. This function should not return a function and `fixa` should *not* be recursive.

b. (4 marks) Write function **fixb** without using any helper functions. In this case, `fixb` should be recursive, and each recursive call should return an unnamed function expression.

c. (4 marks) Write function **fixc** using no helper functions and no unnamed functions.

Hint: you can do this with a curried function definition of the following form:
    fun fixc f x = ...

We saw currying in class with the function sum3.
See also:
  http://en.wikipedia.org/wiki/Currying
  Ullman, *Elements of ML Programming*, Section 5.5

Question 3. (8 marks)

Submit all your code in a file named *a4-3.sml*.

For each of the following function descriptions, state what type ML would assign to a correct implementation of the function. Do not create any user-defined datatypes.

a. (2 marks) A function that takes a comparison predicate and a list as input and returns true if and only if the list is sorted according to the comparison predicate.

b. (2 marks) A function that takes a list of functions $f_1, \dots f_n$, (for any $n \geq 0$), and returns a function that computes $\lambda x. f_1(f2(\dots f_n(x)))$. (This is just the usual way of writing lambda terms.)

c. (2 marks) A function that, given a list of integers, returns a pair consisting of the sum of the even positions and the sum of the odd positions of the list.

d. (2 marks) A function that, for a given pair of functions f and g and a list of pairs (x,y) determines whether $f(g(x),g(y)) = g(f(x,y))$ in the given list.