Spring 2013
Programming Languages
Homework 2

- This homework contains general questions and a Scheme programming assignment.
- Due on Tuesday, February 26, 2013 at 4:59 PM New York time.
- Late submissions to be penalized 15 points per day, until the deadline of 5:59 pm on Thursday, February 28.
- No submissions will be accepted after this deadline. Failure to submit a solution by the deadline will result in a zero homework score. In the event of problems **only**, submit by email to both `qh237@nyu.edu` and `hm1021@nyu.edu`, copying `plock@cs.nyu.edu`.
- Do not collaborate with any person for the purposes of answering homework questions.
- You must use the Racket Scheme interpreter for the programming portion of the assignment. You may download and install Racket for free from the course page. Racket is available for most popular platforms. *Important*: Be sure to select "R5RS" from the language menu before beginning the assignment. You can save your Scheme code to an `.rkt` file by selecting *Save Definitions* from the File menu. Be sure to comment your code appropriately and submit the `.rkt` file.

Static Linkages and Displays [15] Consider the following pseudo-code:

```
procedure outer ( a : integer ) is
  b : integer = 5;

    procedure inner ( c : integer ) is
       d : boolean = True;

       procedure innermost ( e : integer ) is
          d: real = 3.14;
          f: integer = -50;
       begin
          if e == 0 then
              print b, d, f;
          else
              innermost(e-1);
          end if;
       end;

    begin
       innermost(c);
    end;

begin
    inner(a);
end;
```

Please answer the following:

1. Assuming `outer(3)` is called, how many activation records will exist on the runtime stack when the line `print b, d, f;` above executes? Don't count the activation record of the procedure calling `outer(3)`.

2. Draw the runtime stack as it will exist when the base case (`e == 0`) is reached. Assume static linkages are being used to resolve non-local variable references. For each stack frame, indicate which local variables belong to that frame. Also draw the static linkages for each non-local variable.

3. For the same runtime stack described above, now assume a display is used to resolve non-local references. Draw the display data structure with arrows appropriately pointing into the stack.

Call-by-Name [10] Consider the following code, in which the presence of operator `=>` signifies call-by-name semantics for parameter `t` in the function `delayed`. In the event that functions contain a sequence of expressions, functions evaluate to the last expression.

```
object Time {
   def dotime(args: Array[String]) {
        delayed(time());
   }

   def time() = {
      println("Getting time")
      System.nanoTime
   }
   def delayed( t: => Long ) = {
      println("In delayed")
      println("Param: " + t)
      t
   }
}
```

Assume time() will evaluate to 3520. What will function `dotime` print?

Parameter Passing [10] Consider the following:

```
function foobar ( BigObject x )
{
   while ( x.DoMore() )
   {  x.DoOperation();  }
}
```

Assume the following:

- BigObject is 80 bytes large.
- 1 unit of time is required for every 8 bytes copied.
- A pointer (memory address) is 8 bytes.
- 1 unit of time is required to access an object on a stack frame.
- 3 units of time are required to call a subprogram (i.e., to set up and destroy the stack frame).
- 1 unit of time is required to evaluate the `while` loop's condition.
- The loop will execute 30 times.

What's the total cost of calling `foobar` under pass-by-value? Pass-by-reference? Explain your computation.

Lambda Calculus [30]

1. Let S and K stand for the following lambda expressions:

$$\mathrm{S} = \lambda x.\lambda y.\lambda z.xz(yz)$$

and

$$\mathrm{K} = \lambda x.\lambda y.x$$

The composition SKK is a lambda expression again:

$$\mathrm{SKK} = (\lambda x.\lambda y.\lambda z.xz(yz))\mathrm{K}\ \mathrm{K}$$

Apply a sequence of beta reductions to show that SKK is equivalent to the identity function:

$$\mathrm{SKK} = \lambda z.z$$

2. Reduce the following expression in two different ways. Give a brief explanation of your reduction steps.

$$(\lambda x.*\ x\ x)(+\ 2\ 3)$$

3. Consider the following lambda expressions. For each of the above, explain whether the expression can be legally β-reduced without any α-conversion at any step. For any expression below requiring an α-conversion, write the β-reduction twice: once after performing the α-conversion (the correct way) and once after not performing it (the incorrect way). Do the two methods reduce to the same expression?

   (a) $(\lambda xy\,.\,yx)(\lambda x\,.\,x\,y)$

   (b) $(\lambda x\,.\,xz)(\lambda xz\,.\,x\,y)$

   (c) $(\lambda x\,.\,x\,y)(\lambda x\,.\,x)$

4. In the lecture slides, we discussed lambda expressions representing the function `PLUS` and the numerals $\ulcorner 0 \urcorner$, $\ulcorner 1 \urcorner$, ...

   Reduce the lambda expression `PLUS` $\ulcorner 1 \urcorner$ $\ulcorner 2 \urcorner$ and show that it reduces to $\ulcorner 3 \urcorner$.

If the tools you are using to submit your solution supports the λ character, please use it in your solution. If not, you may write \lam as a substitute for λ.

Scheme[35] In all parts of this assignment, implement iteration using recursion. Do NOT use the iterative `do` construction in Scheme. Do not use any function ending in "!" (e.g. `set!`).

Some helpful tips:

1. Scheme library function `append` appends two lists.
2. Scheme library function `list` turns an atom into a list.
3. You might find it helpful to define separate "helper functions" for some of the solutions below.
4. the conditions in "if" and in "cond" are considered to be satisfied if they are not `#f`. Thus `(if '(A B C) 4 5)` evaluates to 4. `(cond (1 4) (#t 5))` evaluates to 4. Even `(if '() 4 5)` evaluates to 4, as in Scheme the empty list `()` is not the same as the Boolean `#f`. (Other versions of LISP conflate these two.)

Please complete the following. You may not look at or use solutions from any source when completing these exercises. Plagiarism detection will be utilized for this portion of the assignment:

1. The function `foldl` ("fold left") accepts the following parameters: a binary function f, a *seed value* s and a list $L = l_1, \ldots, l_n$. The algorithm works by first evaluating $x_1 = f(s, l_1)$, then $x_2 = f(x_1, l_2), \ldots$. For example, `foldl` will evaluate a list of size 3 as follows: $f(f(f(s, l_1), l_2), l_3))$. If the list is empty, `foldl` evaluates to s. Write the function for `foldl` in Scheme.
2. *Insertion sort* is a simple list sorting algorithm. It begins with an unsorted source list and an empty sorted list. The algorithm takes a number x from the unsorted source list and *inserts* it into the proper place in the sorted list $y_1, \ldots, y_n$. This is accomplished by comparing x to each y_i until $x > y_i$, at which point x is inserted between y_{i-1} and y_i. This process is repeated until all numbers in the unsorted source list have been inserted.

   Write an insertion sort function that accepts two parameters k and a list $y_1, \ldots, y_n$ and sorts the first k numbers.
   Example: `(insertion-sort 3 '(5 2 3 9 1))` should evaluate to `(2 3 5 9 1)`. *Hint: break the problem into subproblems and write a separate function for each subproblem.*
3. Write a function `(mapfun FL L)` that takes a list of functions `FL` and a list `L` and applies each element of `FL` to the corresponding element of `L`. For instance,

   ```
   (mapfun (list cadr car cdr) '((A B) (C D) (E F)))
   ```

should return `(B C (F))` since `(cadr '(A B)) => B, (car '(C D)) => C`, and `(cdr '(E F)) => (F)`.

If `FL` and `L` are of different lengths, it should stop whenever it reaches the end of either one; e.g.

```
(mapfun (list cadr car) '((A B) (C D) (E F)))
```

and

```
(mapfun (list cadr car cdr) '((A B) (C D)))
```

should both return (B C).

4. Write a function `(comparatorFuns N)` which takes as argument an integer `N` and evaluates to a list $(f_1, f_2, \ldots, f_n)$ where $f_k(X)$ is `#t` if $X > k$.

   Thus `(mapfun (comparatorFuns 4) '(20 10 0 -10))` evaluates to `(#t #t #f #f)`

   since $20 > 1$, $10 > 2$, $0 > 3$, $-10 > 4$ evaluate to `#t`, `#t`, `#f`, `#f`, respectively.

   `(mapfun (filter (comparatorFuns 4) 3) '(10 0))` evaluates to `(#t #f)`.

5. Write a function `(filter pred L)`, where `pred` is a predicate and `L` is a list of atoms. Function `filter` outputs a list such that an item i in L will appear in the output list if `pred(i)` is true. For instance, `(filter even? '(1 2 3 4))` should return `(2 4)`. Write the function `filter` in Scheme.