Homework 1:
Introduction to CS 3

Homework due in class on the date indicated above. Late homeworks will be penalized by
5 \cdot 2^{n-1}\%$, where $n$ is the number of days the assignment is late. Please indicate which
problems, if any, took extra time.

1. Getting a UGCS account (if you do not already have one)

You may cooperate with another student on this problem.

If you do not already have a UGCS account, contact the UGCS sysadmins, your friendly TA, or the
instructor.

You should also make sure you have proper access to the Annenberg building. If you need help with
that, please talk to your TA.

2. Setting up your UGCS account

In order to use your UGCS account effectively with the CS 3 software, you will have to do some minor
setup tasks. First, if you are an emacs user, ensure you have the Modula-3 macro package installed: To
enable that macro package, simply log in to your UGCS account, cd to your home directory, and type
the following command:

% cat /home/mika/lib/dotemacs_for_m3 >> .emacs

Next ensure that the directories /home/mika/cm3/bin/ and /home/mika/bin.i686/ appear in your
path or PATH variable.

    export PATH=$PATH:/home/mika/cm3/bin:/home/mika/bin.i686

or to have a more permanent solution, add those lines to the bottom of your .bashrc file.

If you are using the C-shell, you make sure of that by adding the commands

    set path=($path /home/mika/cm3/bin /home/mika/bin.i686)

to the end of your .cshrc file; for other shells, please consult the manual or ask a friend (or a TA).

To check that you were successful, try running the mscheme Scheme interpreter. You should see the
following:

    urania-148: mscheme
    M-Scheme Experimental
    LITHP ITH LITHENING.
    >

Quit the Scheme interpreter by typing ctrl-D, or try evaluating some Scheme expressions in it. Maybe
start doing the rest of the assignment!

3. Time management

2\%
For 2% assignment credit, keep track of how much time you spend on each problem and turn that in with your other answers. We need the feedback to improve the class.

4. Lists

In lecture, we discussed how you can make a “memory” in pure LISP. To remind you, an M-expression that accomplishes this is the higher-order function

\[ \text{mem} = \lambda((m), \lambda(((), m))) \]

If we consider \texttt{two = mem(2)}, for instance, we see the value of that expression is the niladic function \( \lambda(((), 2) \). We can retrieve the value by applying the function.

\[ \text{two()} = 2 \]

We also discussed the LISP primitives \texttt{cons, car, and cdr}, defined as follows (for all \( x, y \)):

\[ \text{car(cons}(x, y)) = x \]
\[ \text{cdr(cons}(x, y)) = y \]

Give an implementation (as an M-expression) for \texttt{cons, car, and cdr} using only \( \lambda \) expressions.

Next convert your M-expressions into S-expressions and test them using the Mscheme interpreter. (Ignore the fact that there are predefined routines of the same names as yours; your definitions will override the built-in routines.)

There are several ways of solving this problem using only what we’ve discussed in lecture so far. If you need to use McCarthy’s conditional \( (G_0 \rightarrow S_0, G_1 \rightarrow S_1, \ldots) \), it is written in Scheme as follows:

\[
\text{(cond (}(G_0 \text{ } \text{S}_0)
\text{ (}(G_1 \text{ } \text{S}_1)
\ldots
\text{ )})
\]

Equality testing of integers is done with the \texttt{=} operator. For instance,

\[ > (\text{=} \; 0 \; 0) \]
\[ \text{#t} \]
\[ > (\text{=} \; 1 \; 0) \]
\[ \text{#f} \]

To save your sanity, you need not type the expressions directly into the Scheme interpreter. You can put your expressions in a file of your choosing, say, \texttt{x.scm}, and load them into Scheme using \texttt{(load "x.scm")}

Turn in your M-expressions and S-expressions.

5. Strong mathematical induction

Weak mathematical induction (or just “mathematical induction”) is something you are undoubtedly familiar with. Assume that some property \( P(n), n \in \mathbb{Z} \) holds for \( n = 0 \), i.e., that we may assert (can prove)

\[ P(0) \]

\[ ; \]

\[ 2 \]
assume further that we can prove that if \( P(n) \) holds for \( n = k \), then it also holds for \( n = k + 1 \); or symbolically, that we can prove

\[
P(k) \Rightarrow P(k + 1).
\]

Then the principle of weak mathematical induction says that we may conclude

\[
P(n), \forall n \geq 0.
\]

“Strong” mathematical induction says that if we can prove that \( P(i) \) holding for all \( i \) such that \( 0 \leq i < n \) implies that \( P(n) \) holds, then \( P(n) \) holds for all \( n \in \mathbb{Z}^+ \), or symbolically,

\[
\forall n : n \geq 0 : (\forall i : 0 \leq i < n : P(i)) \Rightarrow P(n)
\]

implies

\[
\forall n : n \geq 0 : P(n).
\]

i. As an exercise, prove that weak and strong mathematical induction are equivalent as stated (over the non-negative integers).

Note that “strong” mathematical induction can be used on partial orders as well as on total orders. Let \( \leq \) be a partial order and define \( < \) as usual, i.e., \( x < y \equiv x \leq y \land x \neq y \). Strong mathematical induction can be used on any well-founded order, which means any order for which it is guaranteed that a decreasing sequence of values is of finite length.

ii. Consider ordered pairs of the form \((a, b)\) where \( a \) and \( b \) are non-negative integers. Let \((a, b) < (c, d)\) mean \( a < c \lor (a = c \land b < d)\). Show that \( < \) is a well-founded order.

6. Recursion and induction

In lecture, you saw a simple Scheme expression which evaluates to the length of a list:

\[
\text{(define length}
\text{\quad (lambda (L)}
\text{\quad \quad (cond ((null? L) 0)
\text{\quad \quad \quad (#t (+ 1 (length (cdr L)))))))})
\]

i. Prove, using weak mathematical induction, that this expression’s value is indeed the length of a Scheme list \( L \).

After graduating, you are hired by a company that writes “mission-critical” software in Scheme. You are asked to write a function called \texttt{equal?}, which defines a type of equivalence of any combination of \texttt{cons} cells and integers. We say that two structures are \texttt{equal?} if they have the same topology and if their leaves have the same numerical values. The structures can, in fact, be regarded as trees (whether they actually are trees or not is a matter for another day).

For instance,

\[
\text{(define a (cons 3 4))}
\]

\[
\text{(equal? (cons 1 (cons (cons 1 2) (cons a a)))
\text{\quad (cons 1 (cons (cons 1 2) (cons (cons 3 4) (cons 3 4))))))}
\]

should evaluate to \#t.

ii. Give the Scheme expression for \texttt{equal?}. It need only work on any syntactically legal combination of \texttt{cons} and integers. Try it on the computer.
Because the application is mission-critical, your manager demands that you prove that your `equal?` function is correct. He tells you that he learned in school that the way to ensure that the expansion of the function terminates is to introduce something called a “variant function” \( V \), which is an integer function of the arguments of the `equal?` function, and to prove that the variant function is bounded below and each evaluation of `equal?` is defined in terms of `equal?`s for which \( V \) is decreasing.

iii. Give a function \( V \) that satisfies your manager’s request and use it to show by induction that `equal?`’s definition is correct.

Of course, you think that no sane programmer would do things this way. It is “obvious”, you say, that `equal?`’s expansion terminates because it is defined in terms of `equal?` on “smaller” trees and it is a waste of time to introduce an integer \( V \).

iv. State a partial order on trees of `cons` cells that allows you to use strong induction to show that your `equal?` is correct and that justifies your feeling that your code is “obviously” correct without needing to introduce an integer variant function \( V \).

7. More recursion

Consider \( g(x, y) \) as follows

\[
g(x, y) = \begin{cases} 
0 & \text{when } x = 0 \land y = 0 \\
g(x, y - 1) + 1 & \text{when } x = 0 \land y > 0 \\
g(x - 1, y + 1) & \text{otherwise}
\end{cases}
\]

i. Is \( g \) well-defined? What is its domain?

ii. Give a nonrecursive definition for \( g \).

Consider \( f(x, y) \) as follows

\[
f(x, y) = \begin{cases} 
y + 1 & \text{when } x = y \\
f(x, f(x - 1, y + 1)) & \text{otherwise}
\end{cases}
\]

iii. Is \( f \) well-defined?

iv. Does any \( f \) exist that satisfies the definition? Is it unique?

8. Tree representation

i. Describe a way to represent trees that can have an arbitrary number of children per node using lists as defined in class.

ii. Using this representation, create a Scheme function that evaluates to the maximum number of children of any node in the entire tree. Prove that it is correct.

9. Symmetry of \( \land \) and \( \lor \) (E.W. Dijkstra and C.S. Scholten)

Let \( \equiv \) stand for logical equivalence and \( \lor \) stand for the logical or. Let \( X \) and \( Y \) be arbitrary boolean scalars. Implement \( \land \), logical `and`, using only \( \equiv \) and \( \lor \). In other words, write down a formula in only \( \equiv, \lor, X, \) and \( Y \) that is true exactly when \( X \land Y \) is true and is false exactly when \( X \land Y \) is false.

You may use the following resources to help you do this assignment:

- To find out more than you ever wanted to know about the Scheme language, you can look in the textbook from MIT:
  

- Last but not least, your friendly neighborhood TA and instructor.