Name:____

CSE341 Autumn 2017, Final Examination
December 12, 2017

Please do not turn the page until 2:30.

Rules:

- The exam is closed-book, closed-note, etc. except for *both* sides of one 8.5x11in piece of paper.
- **Please stop promptly at 4:20.**
- There are **125 points**, distributed **unevenly** among **9** questions (most with multiple parts):
- **The exam is printed double-sided.**

Advice:

- Read questions carefully. Understand a question before you start writing.
- Write down thoughts and intermediate steps so you can get partial credit. But clearly indicate what is your final answer.
- The questions are not necessarily in order of difficulty. Skip around. Make sure you get to all the questions.
- If you have questions, ask.
- Relax. You are here to learn.

Name:____

1. (**18** points) (Racket Programming)

   (a) Write a Racket function `partition` that works as follows:

   - It takes two arguments that it assumes are a function (taking one argument) and a list.
   - It returns a pair (i.e., cons cell) of lists where each element of the input list is in exactly one of the output lists.
   - If the function passed to `partition` returns `#f` for an element in the input list, then the element should be in the list in the cdr of the pair, else it should be in the list of the car of the pair.

   (b) Use `partition` to write a Racket function `number-or-not` that takes a list and returns a pair of lists where the car holds all the numbers in the input list and the cdr holds all the other elements of the input list.

   (c) Suppose `xs` is bound to some Racket list and `number-or-not` is as defined in part (b).
   Is `(car (number-or-not xs))` equivalent to `(car (number-or-not (car (number-or-not xs))))`? Answer "Yes", "No", or "Depends on `xs`" and explain your answer in approximately 1–2 English sentences.

   (d) Suppose `f` is bound to some Racket procedure and `xs` is bound to some Racket list and `partition` is as defined in part (a).
   Is `(car (partition f xs))` equivalent to `(car (partition f (car (partition f xs))))`? Answer "Yes", "No", or "Depends on `f` and/or `xs`" and explain your answer in approximately 1–2 English sentences.

Name:____

2. (**10** points) (Scope and Mutation) Consider running the following Racket code (all together as one very-silly program):

```
(define x 7)

(define f1 (lambda () (begin (set! x (+ x 1)) (+ x 1))))

(define f2 (lambda (x) (begin (set! x (+ x 1)) (+ x 1))))

(define f3 (let ([x 5]) (lambda () (begin (set! x (+ x 1)) (+ x 1)))))

(define f4 (lambda () (let ([x 7]) (begin (set! x (+ x 1)) (+ x 1)))))

(define a1 (f1))
(define a2 (f2 x))
(define a3 (f3))
(define a4 (f4))

(define a5 (f1))
(define a6 (f2 x))
(define a7 (f3))
(define a8 (f4))
```

After the program above is evaluated:

(a) What is `a1` bound to?

(b) What is `a2` bound to?

(c) What is `a3` bound to?

(d) What is `a4` bound to?

(e) What is `a5` bound to?

(f) What is `a6` bound to?

(g) What is `a7` bound to?

(h) What is `a8` bound to?

Name:____

3. (**16** points) (Delayed Evaluation)

The streams we studied in class are *infinite lists* — they encode the idea of being able to ask for more list elements forever. This problem considers *infinite binary trees*, where, similarly, you can ask for more left/right children for forever. We will use this `struct` definition:

```
(struct inftree (left-th root right-th))
```

An infinite tree is a thunk that when called returns an `inftree` where the `left-th` and `right-th` fields hold infinite trees.

(a) Write a Racket function `doubling-tree` that takes a number `x` and returns an infinite tree such that:

- The root of the tree is the value `x`. (Since an infinite tree is a thunk, we mean calling the thunk returns a struct whose `root` field is `x`.)
- The left child of every tree node is an infinite tree whose root value is two times the value of the parent node.
- The right child of every tree node is an infinite tree whose root value is one plus two times the value of the parent node.

(b) Write a Racket function `sum-to-depth` that takes a number `i` and an infinite tree `t` and sums all the numbers in the tree that are at depth `i` or less of the tree. We "count from 1" meaning if `i` is 1 then the correct answer is the value of the root of the tree. If `i` is 2, the result is the sum of 3 numbers, and so on. You can assume all "values in the tree" are numbers.

(c) What does `(sum-to-depth 3 (doubling-tree 1))` evaluate to?

(d) Write a Racket function `stream-of-lefts` that takes an infinite tree and returns a stream such that the n^{th} value produced by the stream is the value in `t` found by taking the left-child branch $n-1$ times (so the first value returned is the value at the root).

(e) Describe in about one English sentence the stream produced by `(stream-of-lefts (doubling-tree 1))`.

The next page has additional space for your answers.

Name:____

More space for Problem 3.

Name:________________________________

4. (**22** points) (Interpreter implementation) Below is some of the code we provided you for Homework 5 (MUPL).

```
(struct var  (string) #:transparent)  ;; a variable, e.g., (var "foo")
(struct int  (num)    #:transparent)  ;; a constant number, e.g., (int 17)
(struct add  (e1 e2)  #:transparent)  ;; add two expressions
(struct isgreater (e1 e2)    #:transparent) ;; if e1 > e2 then 1 else 0
(struct ifnz (e1 e2 e3) #:transparent) ;; if not zero e1 then e2 else e3
(struct fun  (nameopt formal body) #:transparent) ;; a recursive(?) 1-argument function
(struct call (funexp actual)       #:transparent) ;; function call
(struct mlet (var e body) #:transparent) ;; a local binding (let var = e in body)
...
(struct closure (env fun) #:transparent)

(define (envlookup env str)
  (cond [(null? env) (error "unbound variable during evaluation" str)]
        [(equal? (car (car env)) str) (cdr (car env))]
        [#t (envlookup (cdr env) str)]))

(define (eval-under-env e env)
  (cond [(var? e)
         (envlookup env (var-string e))]
        ...
        ))
(define (eval-exp e)
  (eval-under-env e null))
```

(a) Write a Racket function `envremove` that takes an environment `env` and a string `s` and returns an environment that is like `env` except it has no bindings for `s`. Note it is fine if `env` has no bindings for the string, then the result will just be a copy of `env`.

(b) Consider adding a `letinstead` construct to MUPL with this struct:
`(struct letinstead (varnew varold body) #:transparent)`
The result is the result of evaluating the subexpression in `body` in an environment that is like the current environment except (1) the MUPL variable (Racket string) `varnew` is bound to what the MUPL variable `varold` is bound to in the current environment and (2) the MUPL variable `varold` is not bound to anything (so it cannot be used in `body`). If `varold` is not bound in the current environment, an unbound-variable error should occur. Give the case of `eval-under-env` for `letinstead` expressions.

(c) Consider adding a `letalso` construct to MUPL with this struct:
`(struct letalso (varnew varold body) #:transparent)`
The result is the result of evaluating the subexpression in `body` in an environment that is like the current environment except the MUPL variable (Racket string) `varnew` is bound to what the MUPL variable `varold` is bound to in the current environment. Unlike in part (b), `varold` remains in the environment. As in part (b), if `varold` is not bound in the current environment, an unbound-variable error should occur. Give the case of `eval-under-env` for `letalso` expressions.

(d) Recall we can use Racket functions like MUPL macros. One of the additions in part (b) and (c) can be implemented as such a macro rather than extending the interpreter. Write this macro.

(e) For the addition that cannot be implemented as a macro, explain in approximately 1 English sentence why it cannot be.

Put your solutions on the next page.

Name:____

Put your answers to Problem 4 here.

Name:____

5. (**10** points) (ML Type-checking and Soundness)

   We assume the purpose of the ML type system is to prevent treating values as though they had a different type (e.g., treating a number like a function, a tuple like a string, etc.).

   Assume the following function type-checks and has type `t1 * t2 -> t3`:

```
fun f (x,y) = e
```

   For each of the proposed function calls below, give one of these three answers, no explanation required:

   A. Type-checks in ML

   B. Does not type-check in ML, but would be sound to allow it [with no other change to how ML is implemented]

   C. Does not type-check in ML, and would not be sound to allow it [with no other change to how ML is implemented]

   (a) `f a` where `a` has type `t1`

   (b) `f (a,b,c)` where `a` has type `t1` and `b` has type `t2` and `c` has type `t4`

   (c) `f p` where `p` has type `t1 * t2`

   (d) `f (b,a)` where `a` has type `t1` and `b` has type `t2`

   (e) `f a b` where `a` has type `t1` and `b` has type `t2`.

Name:____

6. (**10** points) (Soundness and Completeness) Recall ML, like most statically typed languages, has a type system that is (a) sound, (b) not complete, and (c) allows for type-checker implementations that always terminate. Suppose we change the ML type checker so that it still always terminates and still accepts all the programs it accepted before but it also accepts some more programs.

   For each of the following, answer "possible" if it is possible or "impossible" if it is not possible.

   (a) The type system is still sound and still not complete.

   (b) The type system is no longer sound and still not complete.

   (c) The type system is still sound and is now complete.

   (d) The type system is no longer sound and is now complete.

Name:____

7. (**10** points) (Ruby mixins) Recall the Ruby `Enumerable` mixin provides many useful methods to classes that include it under the assumption that the classes that do include it implement `each`, a method that takes a block taking one argument and calls (yields to) the block once for "each element" (whatever that means for the class).

   (a) Write Ruby code to change the `Enumerable` mixin so that in addition to all its previous functionality it also has a method `each_until_nil` that behaves like `each` *except* it appears to "do nothing" as soon as it encounters a `nil` object. For example,
   `[3,4,nil,5,nil,nil,7,8,9].each_until_nil {|x| puts x.to_s}`
   would print `3 4` (with a newline inbetween but that's a detail of `puts`).
   Hint: Call `each`, which will yield to its block for all elements, but maintain state so that the first yield passing `nil` and all later yields have no effect.

   (b) Further extend `Enumerable` to provide a method `first_nil_index` which returns the number of elements the receiver would enumerate before the first `nil` value is produced. For example,
   `[3,4,nil,5,nil,nil,7,8,9].first_nil_index` would evaluate to `2`
   and `[3,4,5].first_nil_index` would evaluate to `3`.

Name:____

8. (**16** points) (OOP) Port the ML code below to Ruby by transforming it into an OOP style with 10 class definitions. Note: Sample solution is over 50 lines, but all lines are short and *many* are `end`. Next page has additional room in case it's needed.

```
datatype plant = Tree | Flower | Vine
datatype animal = Dog | Giraffe | Centipede | Snake
datatype lifeForm = Plant of plant | Animal of animal
fun speak lf =
  case lf of
      Plant _ => ""
    | Animal a => case a of
                      Dog => "bark"
                    | Giraffe => ""
                    | Centipede => "munch"
                    | Snake => "hiss"
fun num_legs lf =
  case lf of
      Plant _ => 0
    | Animal a => case a of
                      Dog => 4
                    | Giraffe => 4
                    | Centipede => 100
                    | Snake => 0
fun is_name_of_a_data_structure lf =
   case lf of
      Plant Tree => true
    | _ => false
```

Name:____

More space for Problem 8.

Name:____

9. (**13** points) (Subtyping) This problem considers a language like in lecture containing (1) records with mutable fields, (2) functions, and (3) subtyping. Like in lecture, subtyping for records includes width subtyping and permutation subtyping but not depth subtyping, and subtyping for functions includes contravariant arguments and covariant results.

   We assume we have three functions bound to the variables `f1`, `f2`, and `f3` as follows, where we are showing the types of the functions but not their bodies:

```
val f1 : {x : int, y : {a : int, b : int}, z : int} -> int

val f2 : ({x : int, y : {a : int, b : int}, z : int} -> int) -> int

val f3 : (int -> {x : int,  y : {a : int, b : int}, z : int}) -> int
```

   (a) Precisely describe *all* the possible types of values that the type system allows for arguments to `f1`. For example, if the language had no subtyping, the right answer would be, "exactly one type, `{x : int, y : {a : int, b : int}, z : int}`" but with subtyping this would be incorrect because other types are also allowed.

   (b) Is the set of types you described in part (a) finite or infinite?

   (c) Precisely describe *all* the possible types of values that the type system allows for arguments to `f2`.

   (d) Is the set of types you described in part (c) finite or infinite?

   (e) Precisely describe *all* the possible types of values that the type system allows for arguments to `f3`.

   (f) Is the set of types you described in part (e) finite or infinite?

Name:____

Here is an extra page where you can put answers. If you use this page, please write "see also last page" or similar on the page with the question.