Programming Languages
Homework 5
Due Wednesday, February 12th at 2 AM

Download this homework assignment, and fill in your answers, then convert this document into pdf as a report to GradeScope. We are scanning this report so please **do not change the first-page layout.** For question 1, put the answers for the sizes in the indicated areas below; For question 2, paste your code for each of the lisp functions.

1. Consider the following C++ declarations and assume int is 4 bytes, double is 8 bytes, and char is 1 byte:

```
struct Bar {
     double f;
     int i;
     char c,d,e,g;
};
union Foo {
     double f;
     char c;
     Bar b;
};

Foo a[10];
Bar b;
Foo f;
```

   a. **[10]** Give the size (in bytes and in decimal) of each of the following variables:

      - **Bar::c = 1 byte // This is an example**
      - b =
      - f =
      - a =

   b. **[10]** Give the address (in decimal) of each of the following assuming the address of a is 1000:

      - a[5].b.g =
      - a[4].b.i =
      - a[6].f =

2. **[80]** Write a LISP interpreter in LISP. There is a heavily commented start online in eval.l in my src/hw5 directory here. On Openlab, you can copy the file like this:

```
$ cp ~klefstad/public_html/public/141/hw5/eval.l <dest>
```

The functions I wrote completely have no point values associated with them.

NOTE: READ ALL OF THE COMMENTS AND BE SURE YOU UNDERSTAND THEM BEFORE YOU DO ANY IMPLEMENTATION. You can only use setq to modify the global-alist for evaluating defun and setq.

Paste your code for each of the lisp functions (including helper functions if any) below the problem statement below. Make sure you use some decent format (indentation) for graders to read clearly. You can use as much space as you need, as long as you don't change the layout of the first page above.

a. Write my-assoc that takes a variable and an alist (association list) and returns the association of the variable in that alist. You wrote this last week on homework.

```
(my-assoc 'b '((a . 1)(b . 2)(c . 3)(b . z)))   --> (b . 2)
(my-assoc 'z '((a . 1)(b . 2)(c . 3)(b . z)))   --> nil
```

b. Write my-eval which takes an expression and an alist. It returns the evaluation of that expression within that variable binding context. (See the start linked above.) You'll need to write my-eval-atom and my-apply.

c. Write my-apply that takes a function, a list of unevaluated actual parameters, and an alist and applies that function to the evaluated arguments. It should bind the formals to the evaluated actuals, then evaluate the lambda body in that scoping environment. Note it should return the value of the function body. Note the function may be a lambda or a symbol. If it is a symbol, it is a function name, and the function lambda will be in alist.

d. **[10]** Write my-eval-atom that will handle the evaluation of all types of atoms.

e. **[10]** Write my-apply-atom that will cons, car, cdr, eq, quote, cond, eval, defun, and setq as special cases. As a default, it should find the function on the alist and then recursively my-apply it to the arguments in the context of the alist.

f. **[10]** Write my-apply-lambda that will handle evaluation of user-defined function bodies.

g. **[10]** Write my-bind-formals that takes a list of formal parameters and a list of actual parameters and returns a new alist with each formal bound to its corresponding evaluated actual pushed onto the front of the supplied alist. This function is called by my-apply-lambda.

h. **[10]** Write my-eval-defun that puts a function definition on the global alist used by my-top. Note you must add them onto the global-alist by using setq.

i. **[20]** Write my-eval-cond that evaluates conditionals properly.

j. **[5]** Write my-eval-list that takes a list and an alist and evaluates each expression in the list in the context of the alist and returns the value of the last expression in the list. You must use this function for evaluating function bodies and true cond clauses as they allow a list of expressions (not just one).

k. **[5]** Write my-eval-setq that takes a var and a val and changes the value of var to be the evaluated val. This is called by my-apply-atom when the function is setq. Note it must change the value association on the global alist using setq.

l. Write my-top loops over a read eval print loop calling my-eval with the global alist instead of eval. This is the function you will call at the top level to test your eval.

3. Here are some sample expressions you may use to test your interpreter. You can use the real Lisp interpreter to see what the correct answer is for each if you are not sure. Run the following tests on your interpreter; we will use similar tests on the autograder.

```
(my-top) ; to get your my-eval evaluating top-level expressions
t
nil
"Hello"
10
'(a b c)
(car '(a b c))
(cdr '(a b c))
(cons 'd '(a b c))
(eq 'a 'a)
(eq '(a b) '(a b)) ; should be nil
(eq t t)
(eq nil nil)
(eq t nil)
(setq a '(a b c))
a
(defun rev (L A) (cond ((null L) A) (t (rev (cdr L) (cons (car L) A)))))
(rev  '(A B C D E) nil)
(rev a nil)
```

Submit two things:

- **a report with the <u>exact</u> template of page 1 and your answers typed in**
  - **NOTE: do not change the first-page layout or we will give it a 0**
- **submit the file "eval.l" that contains all your function definitions inside on Gradescope so we may run them through MOSS and the autograder**