This assignment will get you familiar with Common Lisp macros. Lisp macros cause rewriting of code (as with C/C++ macros), but because arbitrary Lisp code can be run to determine the replacement code, this process can produce very significant rewrites. The power of macros also depends on the simplicity of Lisp syntax, which makes it relatively easy to generate legal Lisp code.

Remember that evaluation of a macro definition takes place in two phases: (1) macro expansion, which produces a new form, and then (2) evaluation of the new form to produce the macros value. Phase (1) takes place at compilation time, while phase (2) takes place at runtime. To debug a macro, it is often critical to see the result from the expansion phase of macro evaluation. This can be done by using the functions MACROEXPAND or MACROEXPAND-1, e.g., (MACROEXPAND '(macro-name arg1 arg2 ...)).

For this assignment, you are to program three different macros (submit them in a single file):

1. Your first macro is straightforward, in that it does not require any computation at expansion time—it can simply be done with an appropriate backquote form. This macro is to be named nil!. It is to take a single argument, a place (i.e., an accessor form as with setf), and it sets the arguments value to nil. E.g., (nil! (aref a 2)).

2. Anaphoric macros are macros that allow you to easily refer to an earlier form or computation. alambda is actually an anaphoric version of lambda. One of the limitations of a lambda is that it cannot be recursive because it does not have a name to refer to. We can fix this, however, by defining a macro named alambda, which allows use to use the symbol self inside of the alambda to implement a recursive call:

   (alambda (x)
     (if (= x 0)
       1
       (* x (self (1- x))))))

   Your job is to implement alambda. To do this you need to know about labels, a way to define and name local functions in Common Lisp.

3. The Lisp handout shows a version of my macro =-or. While this macro is fine for expressions like: (= -or x 10 23 15), it suffers from the problem of multiple evaluation of the fixed argument. This could be a problem with an expression like: (= -or (f1) 10 23 15). You are to implement a function, eq-or, which works like =-or (but uses eq instead of =) but does not suffer from the problem of multiple evaluation. You will need to use gensym to avoid variable capture.

(Make sure your code adheres to good Lisp style—e.g., proper indenting and spacing, standard Lisp approaches, etc.)