2009 November 30

Macro
Example Test Questions

1.

Write **one** macro definition `cfunc` that translates the following macro calls.
`(cfunc fname (parm))` translates into `(function fname (parm))`
`(cfunc fname (parm) int)` translates into `(int function fname (parm))`

2.

Write a Lisp macro nand that takes any number of arguments and returns true if at least one of the arguments is false.

```
(nand (> 3 2) (atom 'a) (numberp 5)) → nil
(nand (> 3 2) (atom 'a) (numberp 'b)) → T
```

3.

Write a Lisp macro **`mycase`** that translates the following macro call as shown. Assume the input will be error free. The input lists can be any length. Use standard Lisp functionals. If you need support functions, your answer should have only non-recursive support functions.

```
(mycase (C1 C2 ... Cn) (P1 P2 ... Pn))
```

Translates to the following.

```
(mycond (C1 (P1 P2 … Pn)) (C2 (P2 … Pn)) ... (Cn (Pn)) )
```

4.

B Complete the macro definition, without using backquote, of `our-if` that translates the following macro call

```
(our-if a then b)   translates into   (cond (a b))
```

C Complete the macro definition of `our-if` using backquote.

5.

Explain why recursive macro calls do not work in a Lisp macro definition. State two ways one can introduce recursion into a macro definition.

6.

Write a Lisp macro `mycase` that translates the following macro call. Assume the input will be error free. The input lists can be any length greater than 0

```
(mycase (C1 C2 ... Cn) (P1 P2 ... Pn))
```

translates to the following

```
(cond (C1 P1) (C2 P2) ... (Cn Pn))
```

7.

Write a macro function OUR-IF that translates the following macro calls.

`(our-if a then b)` translates into `(cond (a b))`
`(our-if a then b else c)` translates into `(cond (a b) (t c))`

8.

B Given the following function definition, where the intent is to execute either the thenPhrase or the elsePhrase depending upon the condition.

```
(defun new-if (condition thenPhrase elsePhrase)
    (cond (condition thenPhrase)
          (t elsePhrase)))
```

Explain the following, and clearly state why `new-if` does not work as expected.

```
>(setq a 4)
4
>(new-if (< 4 5) (setq a (1+ a)) (setq a (1- a)))
5
>a
4
>(new-if (> 4 5) (setq a (1+ a)) (setq a (1- a)))
4
>a
4
```

C How would you define `new-if` so that the following would occur? It behaves as expected.

```
>(setq a 4)
4
>(new-if (< 4 5) (setq a (1+ a)) (setq a (1- a)))
5
>a
5
>(new-if (> 4 5) (setq a (1+ a)) (setq a (1- a)))
4
>a
4
```

9.

A Given the following function definition, where the intent is to execute either `choice-1` or `choice-2` depending upon `reason`.

```
(defun choice (reason choice-1 choice-2)
    (cond (reason choice-1)
          (t choice-2)))
```

Explain the following, and clearly state why `chose` does not work as expected.

```
>(setq a 11)
11
>(choice (> 5 7) (setq a (- a 3)) (setq a (+ a 5)))
13   ;; expected 16
>(setq a 11)
11
>(choice (< 5 7) (setq a (- a 3)) (setq a (+ a 5)))
8
> a
13  ;; expected 8
```

B How would you define `choose` so that the following expected behaviour occurs?

```
>(setq a 11)
11
>(choice (> 5 7) (setq a (- a 3)) (setq a (+ a 5)))
16
>(setq a 11)
11
```

```
>(choice (< 5 7) (setq a (- a 3)) (setq a (+ a 5)))
8
> a
```

10.

Write a macro `select` that returns a form, which if executed, returns the sublist of items from a list such that, if you apply a given function to a selected item, the result bears the correct relationship to a given value.

The following are examples showing the result of executing the form returned by the macro.

(select item from '(10 20 25 15 30 12 23 5) if 1+ (item) > 20) ➔ (20 25 30 23)
(select item from '(10 20 25 15 30 12 23 5) if 1+ (item) > 21) ➔ (25 30 23)
(select item from '(10 20 25 15 30 12 23 5) if 1- (item) < 20) ➔ (10 20 15 12 5)
(select item from '(10 20 25 15 30 12 23 5) if 1- (item) < 19) ➔ (10 15 12 5)

Use the back quote style to write your macro. Do not explicitly use recursion. Do not use support functions, use lambda instead. Note that there are no nils in the result.

11.

Define the following Lisp macro. Use lambda instead of support functions. If possible avoid explicit recursion.

```
(select item from (v-1 e-1) … (v-p e-p)) →
    (cond ((equal item v-1) e-1) ... ((equal item v-p) e-p))
```

12.

Write a Lisp macro `rearrange` that translates the following macro call as shown. Assume the input will be error free. The input lists can be any length. There is a functional solution. Do not use explicit recursion. If you need support functions, your answer should have only non-recursive support functions. You are required to have comments clearly explaining, **in detail**, how your macro produces the translation.

```
(rearrange (A1 A2 … An) (B1 B2 … Bn) … (Last1 Last2 … Lastn) )
```

translates to the following

```
(A1 B1 … Last1 A2 B2 … Last2 … An Bn … Lastn)
```

13.

Complete the macro definition of `my-if` that translates the following macro calls.

```
(my-if a then (p1 p2 ...) (s1 s2 ...))
```

translates into `(cond (a (cond (p1 s1) (p2 s2) ... )))`

```
(my-if a then (p1 p2 ...) (s1 s2 ...) else c)
```

translates into `(cond (a (cond (p1 s1) (p2 s2) ... )) (t c))`