CS351: Exam #1

March 4, 1999

NAME:________________________________

1. (22 points) Define **both** recursive and iterative versions of the function `sublist-before` which returns all elements in a list before the first occurrence of a given element. If the given element does not occur in the list, it should return a copy of the complete list. The function should take a keyword argument `:test`, which specifies what equality testing predicate to use (defaulting to `eql`). The only *list manipulation* functions you can use are `first, rest, cons, append, member, dolist, mapcar, mappend, list`, and `null`.

```
> (sublist-before '(1 2 3 a 4 a 5) 'a)
(1 2 3)
> (sublist-before '(1 2 3 4 5) 'a)
(1 2 3 4 5)
> (sublist-before '((1)(2)(a)(3)) '(a) :test #'equal)
((1) (2))
```

Recursive definition:

Iterative definition:

2. (22 points) For the following function definitions, write down the values returned for the sample calls **and** briefly describe in English what general operation the function performs.

```
(defun foo (x y tree)
  "Assumes x and y are atoms and tree is initially an arbitrarily nested list"
  (cond ((eq y tree) x)
        ((atom tree) tree)
        (t (cons (foo x y (first tree))
                 (foo x y (rest tree))))))

(foo 'this 'that '(this is that and that is this))
=>
```

English description of foo:

```
(defun bar (a b)
  "Assumes a and b are lists"
  (mappend #'(lambda (x) (mapcar #'(lambda (y) (list x y))
                                 b))
           a))

(bar '(a b c) '(1 2 3))
=>
```

English description of bar:

3. (A) (16 points) Show the trace produced by GPS when solving the following problem. Assume the final version of GPS from the text but *without* operator or goal reordering, i.e. assume operators, goals, and preconditions are pursued in exactly (and only) in the order shown. A sample answer is shown.

```
(setf *sample-ops*
      (list (op 'X :preconds '(B)   :add-list '(A) :del-list '(B))
            (op 'Y :preconds '(C E) :add-list '(B))
            (op 'W :preconds '(C)   :add-list '(E) :del-list '(C))
            (op 'Z :preconds '(D)   :add-list '(B))))

>(gps '(c e) '(a) *sample-ops*)
Goal: A
Consider: X
  Goal: B
  Consider: Y
    Goal: C
    Goal: E
  Action: Y
Action: X
((START) (EXECUTING Y) (EXECUTING X))

> (gps '(c d) '(a) *sample-ops*)
```

(B) (10 points) Now assume that potential operators are heuristically ordered by number of unsatisfied preconditions, as discussed in class and in the text. Show the new trace produced by GPS. Operators are repeated below for convenience.

```
(setf *sample-ops*
      (list (op 'X :preconds '(B)   :add-list '(A) :del-list '(B))
            (op 'Y :preconds '(C E) :add-list '(B))
            (op 'W :preconds '(C)   :add-list '(E) :del-list '(C))
            (op 'Z :preconds '(D)   :add-list '(B))))


> (gps '(c d) '(a) *sample-ops*)
```

4. (20 points) Below is a buggy version of the simple variable pattern matcher from the text. Assume all auxiliary functions are defined correctly. Show the **current** (possibly incorrect) output for the given examples and show how to correct the definition of `pat-match`.

```
(defun pat-match (pattern input &optional (bindings no-bindings))
  "Match pattern against input in the context of the bindings"
  (cond ((variable-p pattern)
         (match-variable pattern input bindings))
        ((eql pattern input) bindings)
        ((and (consp pattern) (consp input))
         (and (pat-match (first pattern) (first input) bindings)
              (pat-match (rest pattern) (rest input) bindings)))
        (t fail)))
```

```
(pat-match '(?x wants ?y to like ?x) '(Tom wants Mary to like Tom))
=>
```

```
(pat-match '(?x wants ?y to like ?x) '(Tom wants Mary to like Bob))
=>
```

```
Corrections needed:
```

5. (10 points) Below is the generic function for solving search problems.

```
(defun tree-search (states goal-p successors combiner)
  "Find a state that satisfies goal-p.  Start with states,
  and search according to successors and combiner."
  (dbg :search "~&;; Search: ~a" states)
  (cond ((null states) fail)
        ((funcall goal-p (first states)) (first states))
        (t (tree-search
             (funcall combiner
                      (funcall successors (first states))
                      (rest states))
             goal-p successors combiner))))
```

Briefly but clearly describe in English what the combiner must be in order to obtain each of the following types of search:

Depth-first:

Breadth-first

Best-first

Beam search