Example Test Questions for Lisp

Test questions can be based on the following sources: (1) the textbook(s), (2) readings, (3) lectures, (4) reports, (5) exercises, and (6) on-line notes and slides. They are based on topics from the beginning of the year up to the class before the test.

Consider all concepts and terminology used in the text book, reports, slides and classes and ask the typical questions - how, why, when, where and what - individually and in combination. In particular, variations are based on "describe", "explain", "define", "what is meant by", etc. The shorter programming exercises in the example programming exercises have been and may be asked as test questions.

Many of the following questions have appeared in previous year's tests and examinations. The list is by no means exhaustive.

1. Basic notions

1. Suppose we evaluate the following s-expressions.

```
-> (setq x '(c d))
(C D)
-> (setq y (append '(a b) x))
(A B C D)
->(eq (cddr y) x)
???
```

What value does the last s-expression evaluate to? Why?

2. What is the result of evaluating the following Lisp expression. Explain your answer.

```
(setq a `(cons a a))
```

3. Given the following function definition.

```
(defun oh (condition thenPhrase elsePhrase)
    (cond (condition thenPhrase)
          (t elsePhrase)))
```

Explain the following

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

4. Explain the following four results.

```
(caadaar '(('((x) (y)) ) z)          returns (x).
(caadr (caar '(('((x) (y)) ) z)))    returns (x).
(caar '(('((x) (y)) ) z))            returns '((x) (y)).
(caadr '((x) (y)))                   returns y.
```

5. Explain what is a Lisp symbol.

6. Explain what is a property list.

7. Explain how we get items into a property list and how we retrieve values from a property list.

8. Describe the data structures in Lisp: the atom, the list.

9. What are an association list and a property list? How do they differ? What is their function in Lisp.

10. Describe the type of programming problems for which Lisp is well suited.

11. How can a general tree (no limit to the number of descendants for any node) be represented as Lisp lists?

12. What is the funarg problem in Lisp? How does this relate to dynamic or static binding of variables?

13. How can dynamic binding be implemented in Lisp? How can static binding be implemented in Lisp? Is it possible to have both static and dynamic binding in Lisp? If so, how can the Lisp interpreter tell them apart?

14. Explain what is meant by dynamic and static scoping of variables.

15. Describe dynamic and static scoping of variables.

16. How can dynamic binding be implemented in Lisp?

17. How is static binding implemented in Lisp?

18. Is it possible to have both static and dynamic binding in Lisp?

19. If so, how can the Lisp interpreter tell them apart?

20. Explain the difference between static and dynamic binding.

21. Give an example of a function A that constructs a function B, then A prints B so you can see what A constructed (nothing fancy), and finally A executes the function B.

22. Explain what is meant by a lambda closure.

23. Describe how multi-dimensional matrices would be implemented in Lisp.

24. Prolog lists and Lisp lists are very similar in structure. Explain how they are similar? How they are different? How is dot-notation related to both list structures?

25. When is lambda closure used?

26. In the context of a Lisp program, what is an environment?

27. In a Lisp program, describe how lambda closures are related to the environment.

28. Explain how in Lisp one can write a function that can construct a new function and then execute it. Do not describe the use of macros, this is a more general question.

29. The following Lisp function is defined where the function reverse reverses the list `x`.

```
(defun test (x) (reverse x))
```

Will the following expressions execute correctly. If yes, explain why. If not, explain why not and modify the expression so it works.

```
(apply 'test '(a b c))
(funcall 'test (list 'a 'b 'c))
(eval 'test '(a b c))
(eval '(test (a b c)))
```

2. Macros

1. What are macros? When are they used?

2. Explain why recursive macro calls do not work in a Lisp macro definition.

3. State two ways one can introduce recursion into a macro definition.

4. How are macros defined in Lisp?

5. Recursion is the method of looping in Lisp programs. Suppose you want to write a macro that requires looping, explain how you would program the loop and why you are proposing that technique.

3. Functionals

1. What is functional programming? What are the prime attributes of functional programs?

2. What is a functional in Lisp? Give some examples of functionals.

3. What is the purpose of defining functionals? How do they affect programming?

4. Explain what the `mapcar' function is. Why is this function (and its cousins) so important in Lisp.

5. Why is Lisp called a functional language?

6. The following Lisp function is defined where the function `reverse` reverses the list `x`.

   ```
   (defun test (x) (reverse x))
   ```

   Will the following expressions execute correctly. If yes, explain why. If not, explain why not and modify the expression so it works.

   ```
   (apply 'test '(a b c))
   (funcall 'test (list 'a 'b 'c))
   (funcall 'test 'a 'b 'c))
   (eval 'test '(a b c))
   (eval '(test (a b c)))
   ```

9. The following is a definition of matrix product. Explain how the functions compute the matrix product. Use diagrams and symbolic notation and an explicit example in your explanation.

   ```
   (defun matProd (a b) (mapcar (bu 'prodRow (trans b)) a))
   (defun prodRow (bt r) (mapcar (bu 'ip r) bt))
   ```

   For ip, use the definition.

   ```
   (defun ip (a b) (reduce '+ (mapcar '* a b)))
   ```

4. Lambda Calculus

1. What are lambda expressions? What is their purpose in Lisp?

2. What is lambda-calculus? What is its relationship to Lisp?

4. Trace and annotate the evaluation of the following lambda expression.

```
{λ A,B . A * B[A]}[{λ B . {λ A . B + A}[1 + B]} [ 1 ] ,
                   {λ B . {λ A . B * A}} [ 3 ] ]

{λ C,D . D[C] + C}[{λ D . {λ C . C * D}[1 + D]} [ 2 ] ,
                           {λ C . {λ D . D * C}} [ 1 ] ]

{ λ A,B . F(A) * F(B)} [{λ X .{ λ C . C + X} [1 + Z] } [1]
                             ,{ λ C .{ λ X . X - C} [1 - Z] } [2]]

{ λ A,B . G(A) + G(B)}[ { λ C . { λ D . D + C}[1 + E]}[1]
                             , { λ F . { λ H . H - F}[1 - E]}[2] ]

{ λ A,B . A * B[A]}[ { λ B . { λ A . B + A}[1 + B]} [ 1 ]
                            , { λ B . { λ A . B * A}} [ 3 ] ]

{ λ C,D . D[C] + C}[ { λ D . { λ C . C * D}[1 + D]} [ 2 ]
                            , { λ C . { λ D . D * C}} [ 1 ] ]

{ λ A,B . A * B[A - 1]}[ { λ B . { λ A . B * A}[1 + B]} [ 2 ]

   , { λ B . { λ A . B + A}} [ 7 ] ]
```

5. In the Functionals exercise 9 your customer does not consider the function prodRow to be useful and doesn't want it to be defined at the global level. Use a lambda expression to remove the definition of prodRow.

5. Pattern Matching

1. Explain what unification means when matching patterns.

2. Describe the basics of how the unification algorithm for pattern matching in Wilensky Chapter 21 works. Include a description of the data structures and various cases that can occur and how they are treated. Do not write Lisp programs.

6. Database

1. How does the database program in Wilensky Chapter 22 organize and index the facts and rules. How does this reduce the time and effort in searching the data base.