Symbolic Computation
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. Convert the following S-expressions in list notation to fully dotted notation.

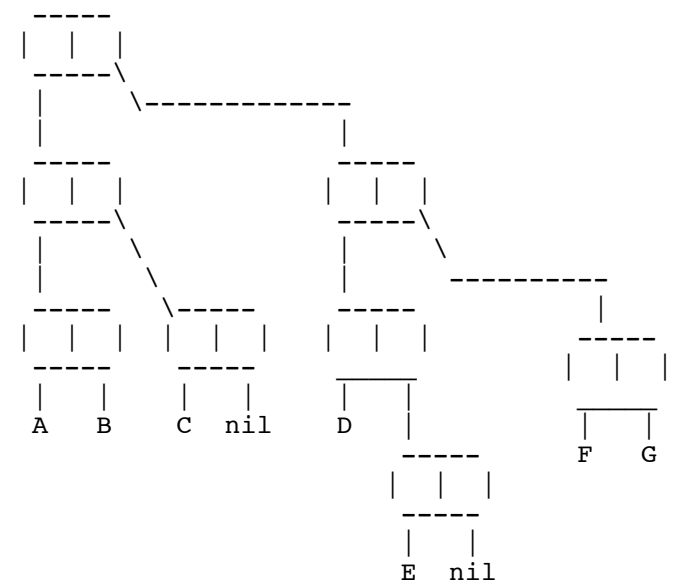
```
(((A) B C) (D E (F)) G)

(A ((B D) (E)) F)

((A B) ((D) E) (F))

((A (B C )) (D E) F G)

(A ((B D) (E)) F)

(((A B) C) (D (E F) G))

(((A) B C) (D E (F)) G)
```

2. Convert the following fully dotted S-expressions to list notation as much as possible (some dots may have to be kept).

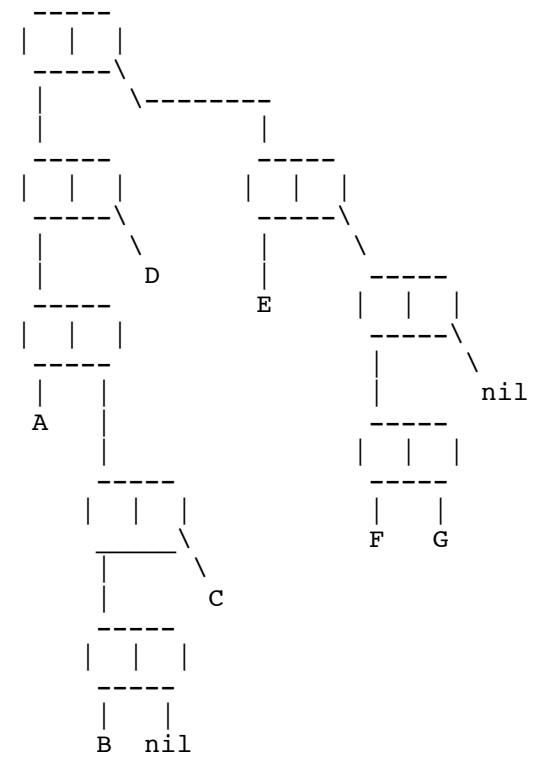
```
(((A . ((B . C) . nil)) . D) . (E . ((F . (G . nil)))))

(((A . B) . (C . nil)) . ((D . (E . nil)) . (F . G)))

((A .(B .(C . nil))).(((D .(E . nil)). F).(G . nil)))

((A . (B . (C . D))) . (E . (((F . G) . nil) . nil )))

(((A . B) . (C . nil)) . ((D . (E . nil)) . (F . G)))

(((A . ((B . nil) . C)) . D) . (E . ((F . G) . nil)))

(((A . ((B . C) . nil)) . D) . (E . ((F . (G . nil)))))
```

3. Write the following representations as Lisp would print them.

Cons cell diagram 1



Cons cell diagram 2



4. Represent the following s-expressions in terms of cons cells and pointers (i.e. binary trees). Draw by hand. Using a computer is a waste of time.

```
(C.((A.NIL).D))
```

```
(NIL.((A)))
((A) ((B)) C)
```

5. 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?

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

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

7. 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
```

7a. 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.
```

8. Explain what is a Lisp symbol.

9. Explain what is a property list.

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

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

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

13. Explain what is a recursive interpreter. Using pseudo-code outline how eval and apply interpret Lisp programs.

14. Define the structure of S-expressions and List expressions in Lisp. Use (a) a BNF-like grammar; (b) a syntax diagram.

15. Explain the basic five operations in Lisp: car, cdr, cons, eq and atom.

16. Define the structure of a function call in Lisp.

17. What are the appropriate problem types for using Lisp as the programming language?

18. Describe the structure of a function definition in Lisp.

19. Describe the syntax and semantics of conditional expressions in Lisp.

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

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

22. What is garbage collection? Outline how garbage collection works.

23. 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?

24. What are reference counts in the context of Lisp lists? How are they used? What are the problems with using reference counts?

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

26. In interpreted languages it is possible to write programs which can, at execution time, construct other programs and execute them. Explain how this is done in Lisp.

27. 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 notion.

28. 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.

29. Explain what is meant by a lambda closure.

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

4. How are macros defined in Lisp?

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. In Backus notation matrix multiplication is defined by the following expression.

   matProd ::=
   (□ □ innerProduct) **o** (□ distributeLeft) **o** distributeRight **o** [**1** , transpose **o 2**]

   Use the following general representation for matrices to illustrate and explain step by step how the above expression evaluates the matrix product `A x B`.

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1p} \\ a_{21} & a_{22} & \dots & a_{2p} \\ & \dots & & \\ a_{n1} & a_{n2} & \dots & a_{np} \end{pmatrix} \qquad B = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1q} \\ b_{21} & b_{22} & \dots & b_{2q} \\ & \dots & & \\ b_{p1} & b_{p2} & \dots & b_{pq} \end{pmatrix}$$

7. 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)))
```

10. The following is a solution to matrix multiplication using Backus notation for functionals. Explain how the expression computes matrix multiplication.

    (□ □ ip) o (□ distl) o distr o [1 , trans o 2]

    where -- □ means "apply to each", equivalent to Lisp mapcar.
    -- (□ □ x) is equivalent to (□ (□ x)) where the outer □ says to apply the inner function, (□ x),
    -- ip is inner product
    -- distl is distribute left
    -- distr is distribute right
    -- trans is transpose
    -- 1 is Lisp first

-- 2 is Lisp second
-- ∘ is function composition

11. Give and explain in Backus notation the definition of the following Lisp function. The input is a list of a pair of integers; for example (9 4). Do not give definitions of any functions used by `mystery`.

```
(defun mystery (pair)
(/ (reduce '* (range (1+ (max (- (first pair) (second pair))
                      (second pair))) (first pair)) 1)
    (factorial (min (- (first pair) (second pair))
                    (second pair)))))
```

12. Assume the following functional program in Backus notation is correct. The input is a list of integers. ∗ and – are the multiplication and subtraction operators.

    `f ::= =1 o length ∘ 1 ; – o [ f o (∗ (bu ∗ 2)) o` **tail** `, 1 ]`

    Explain step by step what the program does on the input < 1, 3, 5 >

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?

3. Give some examples of lambda calculus expressions? Describe how they are evaluated.

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 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.