COMP 348: Principles of Programming Languages

Assignment 2 on LISP

Fall 2021, sections U and DD

October 12, 2021

Contents

1 General Information

Date posted: Wednesday October 13th, 2021.

Date due: Tuesday November 2nd, 2021, by 23:59[1].

Weight: 5% of the overall grade.

2 Introduction

This assignment targets the functional programming paradigm using LISP.

3 Ground Rules

You are allowed to work on a team of 3 students at most (including yourself). Each team should designate a leader who will submit the assignment electronically. See Submission Notes for the details.

ONLY one copy of the assignment is to be submitted by the team leader. Upon submission, you must book an appointment with the marker team and demo the assignment. All members of the team must be present during the demo to receive the credit. Failure to do so may result in zero credit.

This is an assessment exercise. You may not seek any assistance from others while expecting to receive credit. **You must work strictly within your team).** Failure to do so will result in penalties or no credit.

4 Your Assignment

Your assignment is given in two parts, as follows. 1) List Processing, and 2) Structures.

[1] see Submission Notes

4.1 List Processing

For the following questions, implement the function in lisp. Some examples are provided to illustrate the behaviour of each function. Your implementation, however must consider all possible inputs.

Q 1. Write a lisp function called "sub-list" that takes a *list* and two indexes *from* and *to*, and returns a list whose elements are the elements of the input list within from and to indexes. The argument *to* is optional, and in case it is omitted (default to `NIL`), the list *length* is "logically" considered as its value. In case the indexes do not have proper values (i.e. are out of bound), the function simply returns `NIL` (see examples). Note that the unlike lisp, the function uses 1-based indexing.

Examples:

```
> (sub-list '(1 4 10) 2 3)
(4 10)
> (sub-list '(1 4 10) 2)
(4 10)
> (sub-list '(1 7 12) 1 4)
NIL
> (sub-list '(1 7 12) 0 1)
NIL
> (sub-list '(1 6 12) 4 2)
NIL
> (sub-list '(1 6 12))
ERROR *** - EVAL/APPLY: Too few arguments
```

NOTE: You may **NOT** use any built-in functions other than `car`, `cdr`, the list construction functions: `cons`, `list`, `append`, or `list-length`.

Hint: Use `NIL` as the default value for the optional *to* parameter.

Q 2. Write a lisp function called "sub-list2" that takes a *list* and two indexes *from* and *to*, and returns a list whose elements are the elements of the input list within from and to indexes.

The argument *to* is optional, and in case it is omitted (default to `NIL`), the list *length* is "logically" considered as its value. Unlike the above function in 1, if the indexes out of bound, the function used the default from (1) and default to (the length of the string). This function, too, uses 1-based indexing. If the value of the from parameter is greater than to, the function returns `NIL`.

Examples:

```
> (sub-list2 '(1 4 10) 2 3)
(4 10)
> (sub-list2 '(1 4 10) 2)
(4 10)
> (sub-list2 '(1 7 12) 1 4)
(1 7 12)
> (sub-list2 '(1 7 12) 0 1)
(1)
> (sub-list2 '(1 6 12) 4 2)
NIL
> (sub-list2 '(1 6 12))
ERROR *** - EVAL/APPLY: Too few arguments
```

NOTE: You may **NOT** use any built-in functions other than `car`, `cdr`, the list construction functions: `cons`, `list`, `append`, or `list-length`.

Q 3. Modify the function you wrote in the above (call ut "sub-list3") that takes the same type and number of parameters, but in case the from is greater than the to value, it returns the elements from the list in a reverse order, starting form the "to" and ending with the from". See the below examples:

Examples:

```
> (sub-list3 '(1 4 10) 3 2)
(10 4)
> (sub-list3 '(1 4 10) 3)
(10)
> (sub-list3 '(1 7 12) 4 0)
(12 7 1)
> (sub-list3 '(1 6 12))
ERROR *** - EVAL/APPLY: Too few arguments
```

NOTE: You may **NOT** use any built-in functions other than `car`, `cdr`, the list construction functions: `cons`, `list`, `append`, or `list-length`.

Q 4. Write a lisp function that receives a list as the input argument (the list is mixed up integers, decimals, characters and nested lists) and returns a flattened list containing all the atomic number without any duplication. Sample function output is shown below:

```
(flatten-nums-nodup '(1 2 (3 1) (a 2.5) (2 4.5) ((1 2))))
(1 2 3 2.5 4.5)
```

NOTE: The order of the elements must be preserved.

Q 5. Write a function in Lisp that receives an integer n and returns a list of first n numbers of a *tribonacci* sequence as a list.

The tribonacci series is a generalization of the Fibonacci sequence where each term is the sum of the three preceding terms.
The first few elements of the tribonacci sequence are:

$$0, 0, 1, 1, 2, 4, 7, 13, 24, 44, 81, 149, 274, ...$$

Examples:

```
> (tribonacci-seq 7)
(0 0 1 1 2 4 7)
> (tribonacci-seq 0)
NIL
> (tribonacci-seq 1)
0
```

A) Provide iterative solution

B) Provide recursive solution

NOTE: The order of the sequence must be respected. Using auxiliary functions is allowed however not necessary.

4.2 Structures

Q 6. Write a lisp function that receives a single element and determines its depth. The depth of an atom is defined as 0; the depth of a list with no inner list is considered 1; the depth of a list with inner lists, would the maximum deptamong all its inner elements plus 1. Examples:

```
> (depth NIL)
0
> (depth 1)
0
> (depth '(1))
1
> (depth '((2)))
2
> (depth '((2)(3 (6))(4)))
3
```

Q 7. Write a lisp function `cog` that receives a list and calculates its center of gravity. The center of gravity is defined as follows.

Suppose a list represents a bar with weights attached to the positions at indexes. Suppose the mid-point is represented by 0. The center of gravity is the distance from the midpoint where a positive value represents leaning towards right.

The center of gravity may be calculated by applying a simple weighted averaging over indexes, as demonstrated in the following example:

```
'(a (b c) d (e (f g)))

pos:     1         2         3         4
      -- + -- - -- + -- * -- + -- - -- + --
         a         |         d         |
                +--+--+              +--+--+
                |     |              |     |
                b     c              e  +--+--+
                                        |     |
* = midpoint                           f     g
```

The center of gravity may be obtain using the following formula:

$$\text{cog} = \sum_{i=1}^{N} n_i(i - \frac{N+1}{2}) / \sum_{i=1}^{N} n_i,$$

where N is the *length* of the list, and n_i is the total number of weights (elements) attached at position i.

$$\text{cog} = (1 \times (1 - 2.5) + 2 \times (2 - 2.5) + 1 \times (3 - 2.5) + 3 \times (4 - 2.5))/(7) = 0.357.$$

Examples:

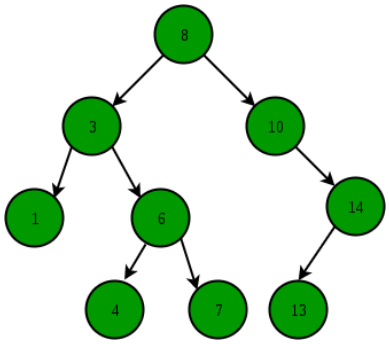
```
> (cog '(a (b c) d (e (f g))))
0.3571428571428571
> (cog '(a (b c) d (e f)))
0.1666666666666667
> (cog '(a (b c) (e f) d))
0
> (cog '(1 2 3 4 5))
0
> (cog '(1 2 3 4 5 6))
0
```

Q 8. A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties:

- The left sub-tree of a node has a key less than or equal to its parent node's key.
- The right sub-tree of a node has a key greater than to its parent node's key.

Write a lisp function to check whether a binary tree is a Binary Search Tree.
A list representing the structure of a sample binary tree is given in the following:

```
'(8 (3 (1 () ()) (6 (4 () ())(7 () ()))) (10 () (14 (13 () ()) ())))
```



Q 9. Write a two lisp functions: `in-order` and `pre-order` that receive a tree (using the same structure as in the previous question), and returns its *in-order* and *pre-order* traversal of the given tree as list, respectively.
An example of the in-order traversal of the tree in the previous question is given in the following:

```
> (in-order '(+ (- (1 () ()) (* (4 () ())(7 () ()))) (/ (7 () ()) (6 () ()))))
(1 - 4 * 7 + 7 / 6)

> (pre-order '(+ (- (1 () ()) (* (4 () ())(7 () ()))) (/ (7 () ()) (6 () ()))))
(+ - 1 * 4 7 / 7 6)
```

Bonus Question

Q 10. Write a function namely `exp-eval` that receives an expression in a pre-order notation (see previous question), evaluates it, and returns the result.

If the expression is not a well-formed pre-order expression it returns `NIL`.

```
> (exp-eval 1) ; not well-formed, not a list
NIL
> (exp-eval '(1))
1
> (exp-eval '(+ - 1 * 4 7 / 7 6))
-25.83333333
> (exp-eval '(- - 1 )) ; not well-formed, not enough operands
NIL
> (exp-eval '(+ 1 2 3)) ; not well-formed, too many expressions -- 3
NIL
```

5 What to Submit

The whole assignment is submitted by the due date under the corresponding assignment box. Your instructor will provide you with more details. It has to be completed by ALL members of the team in one submission file.

Submission Notes

Clearly include the names and student IDs of all members of the team in the submission. Indicate the team leader.

IMPORTANT: You are allowed to work on a team of 3 students at most (including yourself). Any teams of 4 or more students will result in 0 marks for all team members. If your work on a team, ONLY one copy of the assignment is to be submitted. You must make sure that you upload the assignment to the correct assignment box on Moodle. No email submissions are accepted. Assignments uploaded to the wrong system, wrong folder, or submitted via email will be discarded and no resubmission will be allowed. Make sure you can access Moodle prior to the submission deadline. The deadline will not be extended.

Naming convention for uploaded file: Create one zip file, containing all needed files for your assignment using the following naming convention. The zip file should be called a#_studids, where # is the number of the assignment, and studids is the list of student ids of all team members, separated by (_). For example, for the first assignment, student `12345678` would submit a zip file named `a1_12345678.zip`. If you work on a team of two and your IDs are `12345678` and `34567890`, you would submit a zip file named `a1_12345678_34567890.zip`.
Submit your assignment electronically on Moodle based on the instruction given by your instructor as indicated above: `https://moodle.concordia.ca`
Please see course outline for submission rules and format, as well as for the required demo of the assignment. A working copy of the code and a sample output should be submitted for the tasks that require them. A text file with answers to the different tasks should be provided. Put it all in a file layout as explained below, archive it with any archiving and compressing utility, such as WinZip, WinRAR, tar, gzip, bzip2, or others. You must keep

a record of your submission confirmation. This is your proof of submission, which you may need should a submission problem arises.

6 Grading Scheme

Q1	10 marks
Q2	10 marks
Q3	10 marks
Q4	10 marks
Q5	20 marks
Q6	10 marks
Q7	10 marks
Q8	10 marks
Q9	10 marks
Q10	10 marks

Total: $100 + 10$ pts.

References

1. Common-Lisp: `https://common-lisp.net/downloads`
2. Tribonacci Numbers:
   `https://en.wikipedia.org/wiki/Generalizations_of_Fibonacci_numbers`
3. Binary Tree: `https://en.wikipedia.org/wiki/Binary_tree`
4. Binary Search Tree (BST): `https://en.wikipedia.org/wiki/Binary_search_tree`
5. Tree Traversal: `https://en.wikipedia.org/wiki/Tree_traversal`