COMP 348: Principles of Programming Languages
Assignment 2 on Lisp and C

Summer 2021, sections AA and AB

May 26, 2021

Contents

1 General Information **2**

2 Introduction **2**

3 Ground rules **2**

4 Your Assignment **2**

4.1 Functional Programming with Lisp 3

4.2 Procedural Programming with C 9

5 What to Submit **13**

6 Grading Scheme **14**

1 General Information

Date posted: Wednesday May 26th, 2020.

Date due: Tuesday June 15th, 2020, by 23:59[1].

Weight: 9% of the overall grade.

2 Introduction

This assignment targets two programming paradigms: 1) Functional Programming with Lisp, 2) Procedural Programming with C.

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) Functional Programming with Lisp, 2) Procedural Programming with C.

[1] see Submission Notes

4.1 Functional Programming with Lisp

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).

Examples:

```
> (sub-list '(1 6 12) 2 3)
(6 12)
> (sub-list '(1 6 12) 1 4)
NIL
> (sub-list '(1 6 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 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 elements without any duplication. Sample function output is shown below:

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

Q 3. Write a list function that receives a single argument and determines its depth. The depth of an atom is considered 0; the depth of a list with no inner list is considered 1; the depth of a list with inner lists, would the maximum dept of its elements plus 1.
Examples:

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

Q 4. In previous assignment, you wrote a Prolog query to generate the first n numbers of the Lucas sequence. In this assignment, rewrite the code in lisp so that your function generates the sequence and returns the result as a list.

Structures

Q 5. 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))
1
> (cog '(1 2 3 4 5))
```

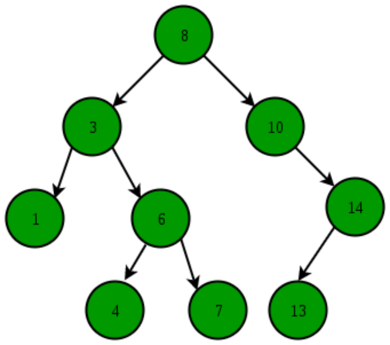
```
0
> (cog '(1 2 3 4 5 6))
0
```

Q 6. Write a lisp function to check whether a binary tree is a Binary Search Tree. 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.

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 7. Write a lisp function `inorder` that receives a tree (using the same structure as in the previous question), and returns its *in-order* traversal of the tree in the form of a list.
An example of the in-order traversal of the tree in the previous question is given in the following:

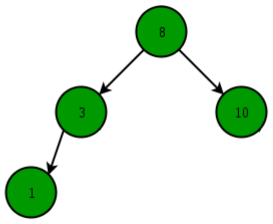
```
(inorder '(8 (3 (1 () ()) (6 (4 () ())(7 () ()))) (10 () (14 (13 () ()) ()))))
(1 3 4 6 7 8 10 13 14)
```

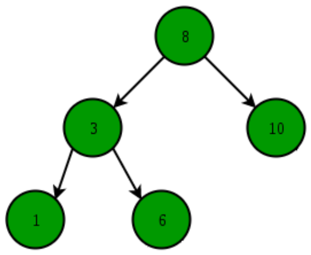
Bonus Question

Q 8. Write a lisp function `make-cbtree` that receives a list, and returns a *complete* binary tree whose nodes are populated from the elements of the list in the same order.
Examples:

```
(make-cbtree '(8 3 10 1))
(8 (3 (1 () ()) ()) (10 () ()))

(make-cbtree '(8 3 10 1 6))
(8 (3 (1 () ()) (6 () ())) (10 () ()))
```





Note: In a complete binary tree every level, except possibly the last, is completely filled, and all nodes in the last level are as far left as possible. See 3.

Hint: While both iterative and recursive solutions are acceptable, implementing an iterative solution is recommended. You may use axillary functions to process the list. A sample suggested algorithm is provided in the next page.

The following is a sample iterative algorithm to implement a construction of a complete binary tree.

```
;; Observe that a tree note is represented by:
;;     (h () ())
;; There are two placeholders for left and right subtrees.
;; The following algorithm uses a flag called left which
;; indicates that the first empty child is the left one.
;;
;; The algorithm is outlined in the following:
;;
;; 1. initialize root to null, placeholders list to empty, and
;;    left flag to true.
;; 2. if input list is empty return root
;; 3. set root to a newly created node from the head of the
;;    input list i.e. (h () ()) where h represents the
;;    first element in the input list.
;; 4. advance the list (set list to the tail of the list)
;; 5. append the root into the placeholders list
;; 6. loop while there are elements to process
;; 6.1. let n be a newly created node from the head of the input
;;      list (h () ())
;; 6.2. let ph be the first node in the list of place holders
;; 6.3. if left flag is true
;;         set the left child in ph to n
;;      otherwise
;;         set the right child in ph to n
;; 6.4. negate the left flag
;; 6.5. if left flag is true (we have consumed both left and right)
;;         remove the head node from the placeholders list
;; 6.6. append n to the list of place holders (to the end)
;; 6.7. repeat step 6
;; 7. return root (that is the very first node that was created)
```

Other solutions may exist.

4.2 Procedural Programming with C

IMPORTANT: Make sure the C++ compiler option is turned OFF.

Functions and Pointers

Q 9. Write a function in C that receives an array of integers and returns a pointer to its smallest element. The signature of the function as well as a short code on its usage are given in the following:

```
int* findmin(int* arr, int size);


int arr[] = {1, 4, 5, 6, -1};
int * m = findmin(arr, 5);
printf("%d", *m); // -1
```

Q 10. You want to write a C library that implements the selection sort to sort an array of integers. The selection sort (6) algorithm is outlined below:

> The algorithm divides the input list into two parts: a sorted sublist of items which is built up from left to right at the front (left) of the list and a sublist of the remaining unsorted items that occupy the rest of the list. Initially, the sorted sublist is empty and the unsorted sublist is the entire input list. The algorithm proceeds by finding the smallest element in the unsorted sublist, exchanging (swapping) it with the leftmost unsorted element (putting it in sorted order), and moving the sublist boundaries one element to the right.

In your implementation, make sure the following requirements are addressed:

1. The selection sort is implemented in a separate C file with appropriate header file.
2. The selection sort must use the *findmin* function, as implemented in the previous question.
3. Only the `selectionsort` function must be exported by the C library. All other definitions (including *findmin* or any other axillary functions) must be kept "internal".

4. Use the following code to test your implementation.

```
#include "selectionsort.h"
#include "selectionsort.h" // included twice

int arr[] = {1, 4, 5, 6, -1 };

int main() {
    int i;
    selectionsort(arr, 5);
    for(i = 0; i < 5; i++) printf("%d ", arr[i]);
    return 0;
}
```

Q 11. Modify the selection sort function such that it receives a third argument, a pointer to a "min" function, that is to be called by the sort implementation. In case the pointer is `NULL`, the default function (the original `findmin`) is to be called. This will be used in the next question.

Q 12. Write a short program to read an array of n integers, sorts them in both ascending and descending order, and prints the sorted arrays, along with the minimum, maximum, average, and the standard deviation.

Notes:

1. The array must be dynamically allocated and released upon the termination of the program.
2. In order to implement the descending order, a pointer to a `findmax` function must be passed to the sort function.
3. The `findmax` function is to be implemented "locally" in the main program. The term "locally" means the function must not be visible outside the code file.

LISP lists and the Linked Lists Implementation

Q 13. In this exercise we want to simulate the behaviour of LISP's list construction in C, where the elements are limited to lists and atoms of "char" type only.

```
typedef enum { ATOM, LIST } eltype;
typedef char atom;
struct _listnode;
typedef struct {
  eltype type;
  union {
    atom a;
    struct _listnode* l;
  };
} element;
typedef struct _listnode {
  element el;
  struct _listnode* next;
} * list;
const element NIL = { .type=LIST, .l=NULL };
```

A) Using the above, implement the following functions in C:

1. `element aasel(atom a);` AKA atom as element, returns an element whose content is set to atom a.

2. `element lasel(list l);` AKA list as element, returns an element whose content is set to the list, pointed by l.

3. `list cons(element e, list l);` that creates a new list whose *car* and *cdr* are the element e and the list l. While the memory for the newly created list is to be allocated dynamically.

4. `list append(list l1, list l2);` that creates a new list whose elements are shallow copies of elements in $l1$ and $l2$, appended.

5. `element car(element e);` that returns head of the list, represented by e; returns `NIL`, if e is not a list.

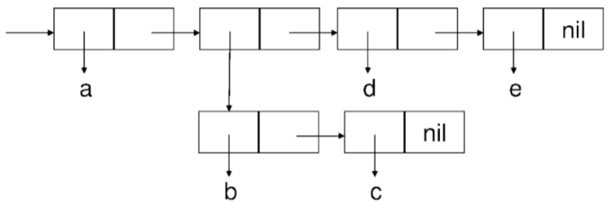
6. `list cdr(element e);` that returns tail of the list, represented by e.

7. `list cddr(element e);` that similarly returns the `cddr` of the list, represented by e.

8. `void free(list l);` that frees all the memory previously allocated by the whole list (including all its elements and its inner lists)

9. `void print(element e);` that prints the content of the element e. If e is an atom, it prints the char symbol enclosed in spaces, and if e it is a list, it (recursively) prints the elements of the list enclosed in parentheses ('(' and ')'). If e is `NIL`, the word "NIL" is printed (see the following example).

B) Write a short code to create and print the following list:



C) Print the car and the cdr of the above list, as well as the car of the car of the original list.

The output must look like the following:

```
( a ( b c ) d e )
 a
(( b c ) d e )
NIL
```

Make sure the list is freed before your program terminates.

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.

<u>IMPORTANT:</u> 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.

<u>Naming convention for uploaded file:</u> 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	5 marks
Q3	5 marks
Q4	10 marks
Q5	10 marks
Q6	10 marks
Q7	10 marks
Q8	10 marks
Q9	5 marks
Q10	10 marks
Q11	5 marks
Q12	5 marks
Q13	15 marks

Total: $100 + 10$ marks.

References

1. Common-Lisp: `https://common-lisp.net/downloads`
2. Lucas Sequence: `https://brilliant.org/wiki/lucas-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`
6. Selection Sort: `https://en.wikipedia.org/wiki/Selection_sort`