COMP 348: Principles of Programming Languages

Assignment 2 on Lisp and C

Fall 2020, sections U and DD

October 17, 2020

Contents

1 General Information

Date posted: Monday October 12th, 2020.

Date due: Thursday November 12th, 2020, by 23:59[1].

Weight: 8% 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 4 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. Note that Your implementation must work for all form of possible inputs.

Q 1. Write a lisp function that takes a *list* and two indexes *from* and *to*, and returns "sub-list" whose elements are the elements within from and to indexes.
e.g.:

```
> (sub-list '(1 6 12) 2 3)
(6 12)

> (sub-list '(1 6 12) 4 2)
NIL
```

In case the indexes are out of bound, the function simply returns `NIL`.
NOTE: You may **NOT** use any built-in functions other than `car`, `cdr`, or list construction functions: `cons`, `list`, and `append`.

Q 2. Modify the above function as in the following:

1. : add default parameters for from and to: in case the *from* is missing, it would be 1; in case the *to* is missing, the *length* of the list may be considered as the default value. You may use the *list-length* built-in function for that.

2. : in case *from* is greater that *to*, the function must return the sublist in reverse order.

```
> (sub-list '(1 6 12) 2)
(6 12)

> (sub-list '(1 6 12) 3 1)
(12 6 1)

> (sub-list '(1 6 12) nil 1)
(1)
```

Structures

Q 3. 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 that are numbers, without any duplication. Sample function output is shown below:

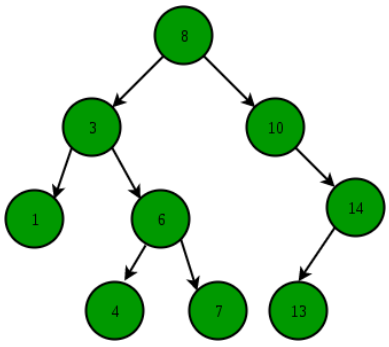
```
(flatten-numbers '(1 2 (3 1) (a 2.5) (2 4.5) ((1 2))))
(1 2 3 2.5 4.5)
```

Q 4. Write a lisp program 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.

The list representing the structure of a sample binary tree is given in the following:

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

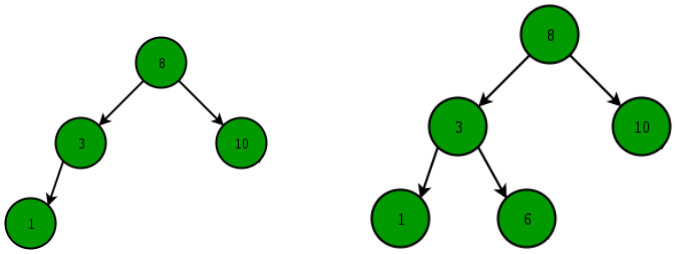


Q 5. Write a lisp function called `split` that receives a list and returns a new list whose elements are the first and the second halves of the original list. Sample function outputs are given below:

```
(split '(1 2 3 4))
((1 2) (3 4}}

(split '(1 2 3 4 5))
((1 2 3) (4 5))

(split '(1 (2 3) 4 5 6))
((1 (2 3) 4) (5 6))
```

Q 6. 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. An example is given in the following:

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

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



You may use auxiliary functions to processed the list, including the *split* function in above.

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 2.

Short Programs

Q 7. Write a lisp function triangle that takes an integer as the argument and prints a triangle of stars as shown in the following figure. If the input is 0, decimal or string, it should print an appropriate error message. positive integers print left-justified triangles, whereas for the negative numbers the printed triangles are right-justified.

Example:

```
(triangle 7)
*******
******
*****
****
***
**
*
```

```
(triangle 4)
****
***
**
*
```

```
(triangle -5
*****
 ****
  ***
   **
    *
```

```
(triangle -1)
*
```

```
> (triangle 2.5)
invalid number; please enter a positive or a negative integer

> (triangle 0)
invalid number; please enter a positive or a negative integer
```

Q 8. In previous assignment, you wrote a Prolog query to generate the first n numbers of the Lucas sequence. In this assignment, rewrite the code as a lisp function that generates the sequence and returns it as a list.

4.2 Procedural Programming with C

Functions and Pointers

In this section you implement some short programs in C.

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 (5) 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.

Linked Lists and Strings

Q 13. Write a C program that creates and prints out a linked list of strings.

1. Define your link structure so that every node can store a string of up to 255 characters.
2. Implement the function `insert_dictionary_order` that receives a word (of type `char*`) and inserts is into the right position.
3. Implement the `print_list` function that prints the list.
4. In the main function, prompt the user to enter strings (strings are separated by *white-spaces*, such as the space character, tab, or newline).
5. Read and insert every string into the link list until you reach a single dot "." or EOF.
6. Print the list.

Notes:

1. Use `scanf()` to read strings. Use `strcpy()` for copying strings.
2. The single dot string denotes the end of text and therefore must **NOT** be considered as an input string.

A sample text is given in the following:

```
This is a sample text.
The file will be terminated by a single dot: .
The program continues processing the lines because the dot (.)
did not appear at the beginning.
. even though this line starts with a dot, it is not a single dot.
The program stops processing lines right here.
.
You won't be able to feed any more lines to the program.
```

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 4 students at most (including yourself). Any teams of 5 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	5 marks
Q2	5 marks
Q3	5 marks
Q4	5 marks
Q5	5 marks
Q6	10 marks
Q7	10 marks
Q8	5 marks
Q9	5 marks
Q10	10 marks
Q11	5 marks
Q12	10 marks
Q13	20 marks

Total: 100 marks.

References

1. Common-Lisp: `https://common-lisp.net/downloads`
2. Binary Tree: `https://en.wikipedia.org/wiki/Binary_tree`
3. Binary Search Tree (BST): `https://en.wikipedia.org/wiki/Binary_search_tree`
4. Lucas Sequence: `https://brilliant.org/wiki/lucas-numbers/`
5. Selection Sort: `https://en.wikipedia.org/wiki/Selection_sort`
6. Merge Sort: `https://en.wikipedia.org/wiki/Merge_sort`
7. C string manipulation:

   `https://en.wikibooks.org/wiki/C_Programming/String_manipulation`