CSCE476/876 **Fall 2023**

Homework 2: Programming Assignment in Lisp
Structures and Missionaries & Cannibals

Assigned on: Friday, September 8^{th}, 2023

Due: Monday, September 18^{th}, 2023

The goal of this assignment is to familiarize yourselves with structures in Common Lisp, learn about interval relations for temporal reasoning in AI, and increase you understanding of the Farmer's Dilemma problem by extending it to solve to the Missionaries and Cannibals puzzle.

- Section 1 introduces structures in Common Lisp (20 points)
- Section 2 introduces Allen's time relations. (20 points)
- Section 3 is the Missionaries and Cannibals puzzle itself. (60 points)

Unless the GTA gives you other instructions, you are advised to submit each problem in a separate file in the following format:

`<lastname>-family.lisp`,
`<lastname>-AllenRelations.lisp`,
and `<lastname>-MCpuzzle.lisp`.

Include any tests, comments, readme instructions, etc. as comments in each of the files.
Graduate students will be held to a stricter standard in terms of code clarity and documentation.

1 Structures in LISP (20 points)

In this section, we will see how to create structures to represent people and how to declare and store how people are siblings.

1. (1 point) Declare a global variable `*people*` as an empty list. We are going to store all the population in this list

2. (2 points) Using `defstruct` create the data type `person`, with three attributes: name (initial value 'noname), age (initial value 0), and siblings (initial value `nil`) to store, respectively, a person's name, age, and list of siblings. The third attribute will be a list of pointers to the structures of the siblings of the person.

3. (2 points) Write a function `add-member` that takes a person name and age as input, creates a structure of the type `person` with that name and age, and pushes the structure in the global variable `*people*`.

4. (2 points) Write a function `find-person` that takes as input a name and returns the structure with that name in the list `*people*`. If no such structure exists, it returns `nil`.

5. (2 points) Write a function `make-siblings` that takes as input two names, finds the structures with those names in the list `*people*`, then adds the first person in the list of siblings of the second person (i.e., pushes the structure of the first person in the list of siblings of the second person) and vice versa.

6. (4 points) Define a function `initialize-people` that takes no input but,

   (a) Using `add-member`, adds the following four people: Bob age 21, Susan age 18, Frank age 16, and Mary age 14.

   (b) Then, using `make-siblings` on each combination of two people, declares that Bob, Susan, and Frank are siblings. Note that Mary is not their sibling.

7. (2 points) Write a function `sibling-names` that takes a name as input and returns a list of the *names* of the siblings. Note, it should not return the list of the structures of the siblings, but the list of their names. Hint: use mapcar on the attribute `siblings` of the structure found by `find-person`.

8. (1 point) Use the `print` function when calling `find-person` on each of the four people. Watch how the structure is printed.

   Does your Lisp listener loop while printing the structures of siblings? If it does not, it means that your Lisp listener is controlling the printing 'depth' to prevent looping over siblings while printing the structures. If it does, the problem is the `print` function of the structure `person`.

9. (4 points) Each symbol in Lisp has a print function, which displays some information when the symbol is evaluated. Read about the print function of a `defstruct` and how to override the default `print` function by modifying the declaration of the structure `person`. Change the defstruct `person` to override the default print function to make it print the name, age, and the list of siblings using the function `sibling-names` defined above.

   Example Output:

   ```
   Mary 14 nil
   Bob 21 (Susan Frank)
   ```

2 Allen's Time Relations (20 points)

This section of the homework deals with time intervals, which are the building blocks for temporal reasoning. For more background on the subject, goto pages 448 and 449 of AIMA. Figure 1 introduces all 13 possible qualitative relationships that may exist between two intervals. These relations are called Allen relations for qualitative temporal reasoning after James Allen who identified them in his seminal paper [1].

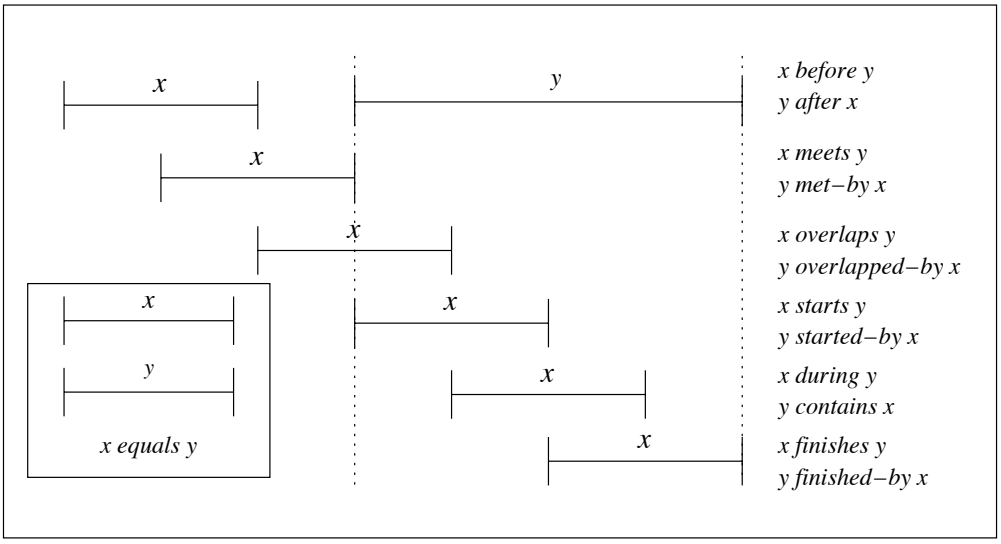

Figure 1: Predicates on time intervals

You are asked to declare a structure to store intervals then functions to determine which of the relationships hold between two given intervals.

1. Declare, using `defstruct`, a structure of type interval, that has three attributes, namely, `task`, `begin` and `end`, which are, respectively, the task-name, and the two endpoints (positive integers) of the interval.

2. Implement, as a function, each of the relationships listed below and illustrated in Figure 1. The function takes as input two interval structures and determines whether or note the relation below between them hold.

   - *Meet(i,j)* $\Leftrightarrow$ *End(i) = Start(j)*
   - *Before(i,j)* $\Leftrightarrow$ *End(i)* $<$ *Start(j)*
   - *After(i,j)* $\Leftrightarrow$ *Before(j,i)*
   - *During(i, j)* $\Leftrightarrow$ *Start(j)* $\leq$ *Start(i)* $\bigwedge$ *End(i)* $\leq$ *End(j)*
   - *Overlap(i,j)* $\Leftrightarrow$ $\exists$ *k During(k,i)* $\bigwedge$ *During(k, j)*

- *Equals(i,j)* $\Leftrightarrow$ *Start(i) = Start(j)* $\bigwedge$ *End(i) = End(j)*
- *Finishes(i,j)* $\Leftrightarrow$ *End(i) = End(j)*
- *Contains(i,j)* $\Leftrightarrow$ *Start(i)* $<$ *Start(j)* $\bigwedge$ *End(i)* $>$ *End(j)*

3. Write a function that tests each of the above relations on the intervals: $i \leftarrow [5, 10]$ and $j \leftarrow [7, 13]$. This function should display a relation, its inputs, and the result using `format` to standard output.

   Example Output:

   ```
   Before [1, 2] [3, 4] t
   Before [1, 2] [1, 2] nil
   (format t "~A [~A, ~A] [~A, ~A] ~A~%" fn <fill in the rest>)
   ```

3 Missionaries & Cannibals puzzle (60 points)

The puzzle is defined as follows: Three missionaries and three cannibals must cross a river using a boat that can carry at most two people at a time. All six people start on one bank of the river, and the goal is to find a series of boat rides that results in everyone on the second bank, with the following restrictions:

- The boat cannot cross the river with no one on board
- The cannibals on either bank cannot outnumber the missionaries on that bank (lest the cannibals eat the missionaries)

Implement a solution to the Missionaries & Cannibals puzzle. In the file with you code, briefly describe how your program works and the functions you used to get the solution, as well as the solution to the puzzle. Your program should solve this puzzle using recursion, and display each of the boat rides used to solve the puzzle when run. Consider the following hints:

- Study the Lisp code for the farmer's dilemma, which has been made available on the website of the course under the section 'Recitation.' Run it in ACL and trace the main functions until you understand how they work. Then, using the code for the Farmer's dilemma as a model, write the Lisp code for solving the Missionaries and Cannibals puzzle.
- About the state description:
  - In this problem, there is no need to distinguish among the missionaries or among the cannibals. You do not have to list them individually in each state, but you can only list their number.
  - Also, you do not need to represent both banks of the river in a state; but you could store, in a state, the side of the boat and the number of missionaries and cannibals with the boat.

- About the actions:
  - One or two missionaries can take the boat. One or two cannibals can take the boat. One missionary and one cannibal can take the boat.
  - Do not allow actions that leave more cannibals than missionaries on the bank once the boat leaves.
  - Do not allow actions that make the number of cannibals or missionaries become less than 0 or more than 3 in any state.

Example Output:

Assuming that there are 3 missionaries and 3 cannibals with the boat on the first bank and the second bank is empty:

```
Leaving Bank 1 with 1 M and 1 C
(format t "Leaving Bank ~A with ~A M and ~A C~%" bank missionaries cannibals)
```

References

[1] James F. Allen. An interval based representation of temporal knowledge. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 221–226, Vancouver, Canada, 1981.