ECS 140A – Programming Languages
Project 3: Lisp

Due: November 17, Wednesday, 2021, 11:59pm PT

This project specification is subject to change at any time for clarification.

Getting Started

There are three major methods of installing Common Lisp:

1. Using Portacle `https://portacle.github.io`
   Portacle is a complete IDE for Common Lisp. It has a similar interface as Emacs. It has a REPL environment in so that you may test your code piecewise more easily.
   (Note: For some reason, I couldn't install it on MacOS 11. It worked on older versions of MacOS. I'm not sure about MacOS 12.)

2. Using terminal:
   To install: enter `brew install clisp` in the command line.
   To execute a file (extension is `.lisp`): enter `clisp filename` in the command line.
   This is the easiest way to install and run, but it doesn't have REPL.

3. Using Emacs and SLIME
   To install: `https://lisp-lang.org/learn/getting-started/`
   This interface is not very easy to work unless you are familiar with Emacs.

There are two (unrelated) parts of the project.

Part 1: Fibonacci Sequence

The Fibonacci sequence is defined as follows:

$$F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$$

The sequence is

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...$$

In this part, you will write three functions related to the Fibonacci sequence. All three functions are written in the file *fibonacci.lisp*.

1. `nth-fib`
   Write a function called `nth-fib` that given a number n, returns the nth number in the Fibonacci sequence.
   Example: `(nth-fib 9)` should return the integer 34.

2. `fib`
   Write a function called `fib` that given a number *n*, returns the frist *n* numbers in the Fibonacci sequence.
   Example: `(fib 9)` should return the list `(0 1 1 2 3 5 8 13 21)`.
   Hint: You may define a helper function with extra parameters to store the previous two numbers and the current result list you have computed so far.

3. `fib-lt`
   Write a function called `fib-lt` that given a number *n*, returns all the numbers in the Fibonacci sequence that are less than *n*.
   Example: `(fib-lt 100)` should return the list `(0 1 1 2 3 5 8 13 21 34 55 89)`.

Note: You do not need to worry about integer flow.

Part 2: Pattern Matching Program

We say an **assertion** is a list of strings such as `(apple banana)` (**note: no quotations**). A **pattern** is a list of atoms that may contain the special symbols ! and *.

- The special symbol ! means matching zero or more atoms.
  Example: The pattern `(apple ! blueberry)` and the assertion `(apple banana orange blueberry)` match.
- The special symbol ! means matching zero or more characters inside an atom.
  Example: The pattern `(apple bl*rry)` and the assertion `(apple blueberry)` match.

Note: An assertion cannot contain the special symbols ! and *. You may assume patterns and assertions do not contain numbers as atoms. For example, `(apple 2 banana)` is not allowed but
`(apple2 banana)` is.

Write a function called `match` in the `match.lisp` file. Given a pattern and an assertion, the function returns true (`t` in Lisp) when they match and returns false (`nil` in Lisp) otherwise.

Examples:

1. `(match '(color apple red) '(color apple red))`
   ⇒ `t`
2. `(match '(color apple red) '(color apple green))`
   ⇒ `nil`
3. `(match '(! table !) '(this table supports a block))`
   ⇒ `t`
4. `(match '(this table !) '(this table supports a block))`
   ⇒ `t`
5. `(match '(! brown) '(green red brown yellow))`
   ⇒ `nil`
6. `(match '(! brown) '(green red brown brown))`
   ⇒ `t`
7. `(match '(red green ! blue) '(red green blue))`
   ⇒ `t`
8. `(match '(red gr*n blue) '(red green blue))`
   ⇒ `t`

9. `(match '(t* table is *n) '(this table is blue))`
   ⇒ `nil`

10. `(match '(color apple *) '(color apple red))`
   ⇒ `t`

11. `(match '(color * red) '(color apple red))`
   ⇒ `t`

12. `(match '(color * red) '(color apple green))`
   ⇒ `nil`

13. `(match '(color*red) '(color apple red)`
   ⇒ `nil`

14. `(match '(color ! * red) '(color apple red))`
   ⇒ `t`

Submission

Please submit both files in a zipped folder.

You should avoid using existing source code as a primer that is currently available on the Internet. You are also not allowed to use the parser tools found on the Internet.

You must specify in your readme file any sources of code that you have viewed to help you complete this project. All class projects will be submitted to MOSS to determine if students have excessively collaborated. Excessive collaboration, or failure to list external code sources will result in the matter being referred to Student Judicial Affairs.