Homework: DefineLang and FuncLang

Learning Objectives:

1. Write programs in DefineLang, FuncLang
2. Get familiar with the concepts of recursive functions, high-order functions and currying

Instructions:

1. Total points: 54 pt
2. Early deadline: March 2 (Wed) 11:59 pm, Regular deadline March 4 (Fri) 11:59 pm.
3. Download hw5code.zip from Canvas
4. Set up the programming project following the instructions in the tutorial from hw2 (similar steps)
5. How to submit:
   - For questions 1 & 2, you can write your solutions in latex or word and then convert it to pdf. Please provide the solutions in one pdf file.
   - For questions 3 & 4, please submit your solutions in one zip file with all the source code files (just zip the complete project's folder).
   - Submit the zip file and one pdf file to Canvas under Assignments, Homework 5

Questions:

1. (8 pt) Write FuncLang programs to process a list of strings. Here, a string is a list of characters and each character is represented using a number.

   (a) (4 pt) Given a *character* and *a list of strings*, find the number of occurrence of the character in each string. See the example below.
   $ (find 88 (list (list 77 73) (list 89)))
   ((0) (0))
   $ (find 88 (list (list 77 73) (list 88) (list 88 88 76) (list 88 88 88 88)))
   ((0) (1) (2) (4))

   (b) (4 pt) Given *a list of strings*, return a string that *concatenates* all the strings in the list and characters are sorted in *decending* form. See the example below.
   $ (sort (list (list 1 2) (list 3 4) (list 5)))
   (5 4 3 2 1)
   $ (sort (list (list 1 2) (list 5 4) (list 5)))
   (5 5 4 2 1)
   $ (sort (list (list 1 2) (list 4 3) (list 5)))
   (5 4 3 2 1)

2. (6 pt) Write a function, `triangle`, which takes a *number* and produces *a list*; each element of the *list* is a list of symbols. When triangle is called with a non-negative integer, n, it returns a list containing n number of lists. The first inner list has n elements, the second inner list has n-1 element, and so on until you reach the top with only one element list, which forms the shape of a triangle. Each of the inner lists contain only the numbers 0 and 1 and they alternate across lists. The result always has the 0 as the first element of the first inner list, if any. In the following examples, we have formatted the output to show the result more clearly, but your output will not look the same; it is sufficient to just get the outputs that are equal to those shown. Spaces in the lists are just for displaying purposes and you are not required to print them.

```
$ (triangle 0)
()
$ (triangle 1)
((0))
$ (triangle 2)
((0 1)
  (1))
$ (triangle 3)
((0 1 0)
  (1 0)
    (0))
$ (triangle 4)
((0 1 0 1)
  (1 0 1)
    (0 1)
      (1))
$ (triangle 5)
((0 1 0 1 0)
  (1 0 1 0)
    (0 1 0)
      (1 0)
        (0))
```

3. (20 pt) Extend the FuncLang interpreter by supporting “>” and “<” and “=” on strings and lists, supporting “=” on boolean values. For “=”, we return true if the two strings have the exact length and content. Two list values are considered equal if they have the same size and each element of the list is equal to corresponding element in the other list. For “<” and “>”, the string and list comparison is done using the length of the strings and lists. That is, “> first second” returns true if the first string/list is longer than the second string/list; and “< first second” returns true if the first string/list is shorter than the second string/list.

   For example,

```
$ (= "abc" "abc")
#t
$ (= "abc" "abcdef")
#f
```

```
$ (> "abc" "abcd")
#f
$ (< "abc" "abcdef"))
#t
$ (= #t #t)
#t
$ (= #t #f)
#f


$ (= (list) (list))
#t
$ (= (list 1 2 3 4) (list 1 2 3 4))
#t
$ (= (list 1 2 3 4) (list 1 2 3 4 5))
#f
$ (= (list 1 2 3 4 (list)) (list 1 2 3 4 (list)))
#t
$ (= (car (list 1 2 3)) 1)
#t
$ (= (car (list 1 2 3)) 2)
#f
$ (= (cdr (list 1 2 3)) 2)
#f
$ (= (cdr (list 1 2 3)) (list 2 3))
#t
$ (= (cdr (list 1 2 3)) (cdr (list 4 2 3)))
#t
$ (= (cons 0 (list 1 2)) (list 0 (list 1 2)))
#f
$ (= (cons 0 (list 1 2)) (list 0 1 2))
#t
$ (> (list 1 2) (list))
#t
$ (> (list) (list 1))
#f
$ (< (list 1 2) (cdr (list 2 3 4 5)))
#t
```

4. (20 pt) Extend the syntax and semantics of the Funclang language to add support for a switch expression. The signature of switch-case is following:

```
(switch (e0)
  (case e1 body)
  (case e2 body)*
  (default body)
```

)

The switch expression will check whether the value of e0 is equal to the following cases from one by one. If equal, value of the corresponding body expression is returned as the result. If no matching is found, the value of the body of default is returned. There must be at least one `case` clause and exactly one `default`. Some examples:

```
$ (define x 0)
$ (switch (x) (case 0 3) (case 1 4) (case 2 2))
error
$ (switch (x) (case 0 3) (case 1 4) (default 2))
3
$ (define foo (lambda (var) (switch (var) (case 1 (+ var 2)) (case 2 (- var 2)) (case 3 (* var 2)) (case
4 (/ var 2)) (default var))))
$(foo 1)
3
$ (foo 2)
0
$ (foo 3)
6
$ (foo 4)
2
$ (foo 5)
5
```