CMPSC 461: Programming Language Concepts
Assignment 2. Due: Sep. 22, 11:59PM

For this assignment, you need to submit your solution as one single file named as "code.rkt" to Gradescope. You may NOT use any of Schemes imperative features (assignment/loops) or anything else not covered in class. You should use Racket (https://racket-lang.org) for your implementation. For all problems, you can assume all inputs obey the types as specified in a problem.

Testing and Auto Grading We will use the Gradescope Autograder feature to grade the assignment. So it is important to start from the code template "code.rkt" on Canvas, which only has dummy implementations, but is nevertheless useful since it obeys a few important requirements for auto grading, such as file name, function names etc. For you submission, just replace the dummy definitions with your implementations.

Before the submission deadline, you can see your partial score based on a public set of tests on Gradescope. You can debug your code and resubmit any time before the deadline. After the deadline, a final grade will be assigned based on the entire testsuite, including the public and a few extra private tests.

Debug Hint: if the autograder hangs for > 5 minutes, or you see an error message like "The results uploaded to Gradescope were not formatted correctly.", that means your submitted code fails to terminate on at least one of the public or hidden private tests. Carefully check your code to avoid non-terminating cases.

Problem 1 [20pt] In lecture, we define Church number $\underline{n}$ as $\lambda f\ n.\ (f^n\ z)$, where $(f^n\ z)$ represents the n-fold composition of function f applied to z (i.e., $f(f(\ldots(f\ z)))$ where f is repeated for n times).

a) (5pt) Implement a function `funPower`, which takes a function f, an integer n and returns the function f^n. For example, (`(funPower sqrt 2) 16`) should return 2.

b) (5pt) Implement a function `encode`, which takes a natural number n and returns the church encoding of n. For example, (`encode 2`) should return $\underline{2}$ (i.e., the function $\lambda f.\ \lambda z.\ f\ (f\ z)$).

c) (5pt) Implement a function `decode`, which takes $\underline{n}$ (the church encoding of some natural number n) and returns n. For example, (`decode (encode 2)`) should return 2.

d) (5pt) Implement a function `MULT`, which takes two church numbers $\underline{n_1}$, $\underline{n_2}$ and returns the church number of $n_1 \times n_2$ (i.e., $\underline{n_1 \times n_2}$). For example, (`decode (MULT (encode 2) (encode 3))`) should be 6.

Problem 2 [5pt] Define a **recursive function** `merge`, which takes two **sorted** lists of numbers, and returns one merged list where all numbers are sorted. Assume that all elements are sorted in the increasing order. For example,

```
(merge '(2 7) '()) ; returns (2 7)
(merge '(1 1 2) '(1 3 5)) ; returns (1 1 1 2 3 5)
(merge '(1 1 6 8) '(2 7)) ; returns (1 1 2 6 7 8)
```

Problem 3 [5pt] Define a **recursive function** `findMax` to return the largest element from a list of non-negative numbers (note that the list may contain nested lists). If the list is empty, your implementation should return 0. For example,

```
(findMax  '(4 5 1))       ; returns 5
(findMax  '(5 1 (3 (4 8)))) ; returns 8
(findMax  '(1 3 (3 3) () 6 (1))) ; returns 6
```

Problem 4 [5pt] Define a **recursive function** `lstEvenSum`, which when given a list of numbers, it returns the sum of numbers at *even* positions in the list. Your implementation should return 0 if the list is empty or has length 1. You can assume that the input is a list of numbers **without nested lists**. For example,

```
(lstEvenSum '(1)) ; returns 0
(lstEvenSum '(1 2 3)) ; returns 2
(lstEvenSum '(1 2 3 4)) ; returns 6
```

Problem 5 [15pt] Implement the following functions in Scheme using `map` and `foldl`. **DO NOT** use recursive definition for this problem.

a) (5pt) Define a function `trunc`, which takes a lower bound a, an upper bound b and a list of numbers, and replaces all numbers that are $< a$ with a and all numbers that are $> b$ with b. For example,

```
(trunc 0 1 '(-2 -1 0 1 2)) ; returns '(0 0 0 1 1)
(trunc 0.5 1.5 '(-2 -1 0 1 2)) ; returns '(0.5 0.5 0.5 1 1.5)
(trunc 1 1 '(-2 -1 0 1 2)) ; returns '(1 1 1 1 1)
```

b) (5pt) Define a function `lstOR`, which takes a list of Booleans and returns `#f` if and only if all of the Booleans are false. For your convenience, (`lstOR` $'()$) is defined as `#f`. For example,

```
(lstOR '(#t #f)) ; returns #t
(lstOR '(#f #f)) ; returns #f
(lstOR '()) ; returns #f
```

c) (5pt) Define a function `geq`, which takes two lists of numbers $(x_1\ x_2\ \ldots\ x_n)$, $(y_1\ y_2\ \ldots\ y_n)$ and returns `#t` if and if and only if $x_i \geq y_i$ for all $1 \leq i \leq n$. You can assume that both lists have the same length. For example,

```
(geq  '(5 4 3) '(2 3 3)) ; returns #t
(geq  '(1 0) '(0 1)) ; returns #f
(geq  '() '()) ; returns #t
```