CMPSC 461: Programming Language Concepts

Prof. G. Tan, Spring 2022

Programming assignment 2: Racket Programming
Total: 24 points. Due on Mar 30th at 6pm in Canvas.

1. (3 points) Write a Racket `remove-when` function. It takes two parameters. The first is a boolean function f, and the second is a list l. The `remove-when` function returns a list with all elements x in l such that $f(x) = \#t$ being removed.

   For example,

   ```
   (remove-when (lambda (x) (> x 3)) '(10 1 7 2))
   ```

   should return the list (1 2), since both 10 and 7 are greater than 3.

   Define the `remove-when` function by using case analysis and recursion.

2. (4 points) Write a Racket `minInt` function, which takes a list of numbers, and returns its minimum element. For example, (minInt '(7 3 6 2)) should return 2. If the input list is empty, then it should return 0. Write the `minInt` function in Scheme in two steps:

   (a) (2 points) Write a `minInt_helper` function; it takes a number k and a list of numbers, x, as arguments, and returns the number which is the smallest among k and numbers in x. For example, (minInt_helper 5 '(4 5 6)) should return 4.

   (b) (2 points) Write the `minInt` function based on `minInt_helper`.

3. (3 points) In Racket, write a higher-order function `manycall` that takes three parameters: n, f, x. It calls f on x for n number of times, when n is even, but calls f on x for $n-1$ number of times, when n is odd. That is, `manycall` should return x when $n = 0$ or $n = 1$; it should return $f(f(x))$ when $n = 2$ or $n = 3$; it should return $f(f(f(f(x))))$ when $n = 4$ or $n = 5$; etc. As an example, `(manycall 7 plusOne 10)` should return 16. Hint: you can use built-in predicates `even?` and `odd?` to test whether a number is even or odd, respectively.

4. (5 points) Write a Racket function `to-words` that takes an integer between 0 and 199, inclusive, as a parameter and returns a list of words

corresponding to the integer's reading in English. Make your function have a small number of cases; i.e., don't write a COND expression with 200 cases. Example calls to your function are given below:

```
(to-words 13) ; should return (thirteen)
(to-words 42) ; should return (forty two)
(to-words 155) ; should return (one hundred and fifty five)
```

Note, in order to simplify things, we are not expecting dashes in the output (i.e., 42 is output as "forty two" not "forty-two"). Also, if the input number is outside of the range of [0,199], your program should output 'error. Hint: You may want to use the built-in integer division functions `quotient` and `remainder`.

5. (9 points) In this question, we are going to write a Racket function `wordMaxIndex` that calculates the maximum index of each word that appears in an input word list. For example, if the input is

```
'(time is long but life is short)
```

Then one output can be

```
((time 0) (long 2) (but 3) (life 4) (is 5) (short 6))
```

The index of the first word in a word list is 0; the index for the second word is 1; etc. Therefore, in the previous example, the maximum index for word "time" is 0; the maximum index for word "is" is 5, since "is" appears twice, at index 1 and 5. Note that you are not asked to sort the words in the output. Therefore, the output is correct as long as the indexes are correct.

Do the following steps to implement the word-index program. In our discussion, we call a word with an index a *word-index pair*, for example, `(short 6)` and `(is 5)` are word-index pairs. We call a list of word-index pairs a *word-index list*.

(a) (3 points) Write a function `initialWIList` that takes a list of words and creates a word-index list. The resulting word-index list should have the right index for every word in the list. For instance,

```
(initialWIList '(time is long but life is short))
```

should generate the following output:

```
((time 0) (is 1) (long 2) (but 3) (life 4) (is 5) (short 6))
```

Hint: develop `initialWIList` on top of a helper function `initialWIListHelper`, which takes an index k and a word list l as input. It assumes that the first word in l has index k and generates the right word-index list. For instance,

```
(initialWIListHelper 4 '(life is short))
```

should generate the following output:

```
((life 4) (is 5) (short 6))
```

(b) (2 points) Write a function `mergeWI`. It takes two inputs. The first is a word-index pair and the second is a word-index list. This function generates a new word-index list. If the input word-index pair has a corresponding pair in the word-index list with the same word, then only the pair with the bigger index should be kept; otherwise, the output word-index list should have the input word-index list with the word-index pair at the end of the list. For instance,

```
(mergeWI '(is 1) '((time 0) (is 5)))
```

should generate

```
((time 0) (is 5))
```

As another example

```
(mergeWI '(life 4) '((time 0) (is 5)))
```

should generate

```
((time 0) (is 5) (life 4))
```

(c) (2 points) Write a function `mergeByWord`, which takes a word-index list and produces a new word-index list; the output word-index list should have one word-index pair for each word that appears in the input list and the index should be the maximum index of all indexes for that word in the input list. For instance, if the input list is

```
((time 0) (is 1) (long 2) (but 3) (life 4) (is 5) (short 6))
```

then the output should be

```
((time 0) (long 2) (but 3) (life 4) (is 5) (short 6))
```

Write `mergeByWord` based on the `reduce` function we discussed in class and `mergeWI`. The `reduce` function is not built-in in Racket; you can type it in yourself:

```
(define (reduce f l v)
  (if (null? l) v
      (f (car l) (reduce f (cdr l) v))))
```

(d) (2 points) Finally, write a `wordMaxIndex` function that takes in a list of words and outputs the right word-index list; that is, for each word in the input, the output word-index list should contain a pair for the word with the maximum index. Write this function based on `initialWIList` and `mergeByWord`.

Notes For programming assignments, examples are given only for the purpose of clarification. By no means that our tests will solely be based on those examples. It's your responsibility to thoroughly test your code by designing your own test cases.

The Racket reference at `https://docs.racket-lang.org/reference/` contains a list of standard procedures provided by Racket (for example, the remainder and the square root functions). You may find them helpful during your programming.

Submission format: Put all your code into one DrRacket file and submit your file through Canvas. Please clearly mark your answers using comments so that we can tell the correspondence between your code and questions.

Please ensure that your file starts with a comment that includes your name and your Penn State network user id.