1. ;; CMPSC 461 Project 2 Taylan Unal (tuu2)
2. ;;------------------------------------------------------------------------
4. ;; #1  higher order **function** dncall
5. ;; Takes 3 parameters (n f x), returns x when n=0, when n=1 returns f(f(x)), when n=2 returns f(f(f(f(x)))), etc.
6. ;; The **function** returns the result **of** calling f(x) 2n times.
7. ;; INPUT: f **is** a **boolean** **function**, l **is** a list **of** ints.
8. ;; OUTPUT: List **of** ints that pass condition **in** **boolean** **function**
10. (define (addOne x) (+ x 1)) ;;just a sample **function**, can be lambda.
12. (define (dncall n f x)
13. (**if** (= n 0)
14. x
15. (dncall (- n 1) f (f (f x))) ;; **else** **case**. starts stacking f(f(x)) **for** each recursive call determined by n
16. )
17. )
19. ;;------------------------------------------------------------------------
21. ;; #2 Scheme keep-**if**
22. ;; define **function** using **case** analysis **and** recursion
23. ;; INPUT: f **is** a **boolean** **function**, l **is** a list **of** ints.
24. ;; OUTPUT: List **of** ints that pass condition **in** **boolean** **function**
25. (define (keep-**if** f l)
26. (define output '())
27. (cond ((null? l) '()) ;; **if** list **is** null, return empty list.
28. ((f (car l)) ;; apply **function** **on** first element **of** list.
29. (cons (car l) (keep-**if** f (cdr l))));;**if** car l **true**, constructs new list **of** it **and** recursive call **of** the rest **of** the list.
30. (**else** (keep-**if** f (cdr l))) ;; **else** **case**, when car l **is** **not** **true**, simply continue **with** remainder **of** list.
31. )
32. )
34. ;;------------------------------------------------------------------------
36. ;; #3a least\_helper **function**
37. ;;Input: a number 'k', a list **of** numbers 'x'
38. ;;Output: the minimum element from a list **of** numbers
39. (define (least\_helper k x) ;; k **is** a number, x **is** a list **of** numbers
40. (**if** (null? x) k ;; **if** list **is** empty, return k, since k **is** least **in** **set** **of** k
41. (**if** (< (car x) k)
42. (least\_helper (car x) (cdr x))
43. (least\_helper k (cdr x))
44. )
45. )
46. )
47. ;;------------------------------------------------------------------------
49. ;; #3b least **function**
50. ;; Uses least\_helper **to** break down a list **and** find minimum element within list.
51. ;; Input: Inputs a list **of** numbers
52. ;; Output: the minimum element from a list **of** numbers
53. (define (least l)
54. (**if** (null? l)'()
55. (least\_helper (car l) (cdr l)) ;;starts recursive callstack **on** list, calls least\_helper **on** rest **of** list.
56. )
57. )
59. ;;------------------------------------------------------------------------
60. ;;------------------------------------------------------------------------
61. ;; #4 **to**-words **function**
62. ;; Input: A number from -99 **to** 99 inclusive
63. ;; Output: text representation **of** number **in** list format.
64. (define (**to**-words n)
65. (define ones '(one two three four five six seven eight nine ten eleven twelve thirteen fourteen fifteen sixteen seventeen eighteen nineteen))
66. (define tens '(twenty thirty fourty fifty sixty seventy eighty ninety))
68. (cond ;; START TOP COND
69. ((< n -99) '(ERROR: OUT OF BOUNDS))
70. ((> n 99) '(ERROR: OUT OF BOUNDS))
71. ((= n 0) '(zero))
73. ;;START NEGATIVE READOUT
74. ((< -100 n 0)
75. (cons 'negative ;;constructs a list based **on** output **of** below.
76. (cond
77. ((< 0 (**abs** n) 20) ;;ALL VALUES FROM 1-19 are **in** ONES LIST
78. (cons (list-ref ones (- (**abs** n) 1)) '() ))
79. ((< 20 (**abs** n) 30)
80. (cons 'twenty (list (list-ref ones (- (**abs** n) 21)))))
81. ((< 30 (**abs** n) 40)
82. (cons 'thirty (list (list-ref ones (- (**abs** n) 31)))))
83. ((< 40 (**abs** n) 50)
84. (cons 'fourty (list (list-ref ones (- (**abs** n) 41)))))
85. ((< 50 (**abs** n) 60)
86. (cons 'fifty (list (list-ref ones (- (**abs** n) 51)))))
87. ((< 60 (**abs** n) 70)
88. (cons 'sixty (list (list-ref ones (- (**abs** n) 61)))))
89. ((< 70 (**abs** n) 80)
90. (cons 'seventy (list (list-ref ones (- (**abs** n) 71)))))
91. ((< 80 (**abs** n) 90)
92. (cons 'eighty (list (list-ref ones (- (**abs** n) 81)))))
93. ((< 90 (**abs** n) 100)
94. (cons 'ninety (list (list-ref ones (- (**abs** n) 91)))))
95. ((**integer**? (quotient (**abs** n) 10)) ;;used **for** clean tens values without ones
96. (list (list-ref tens (- (quotient (**abs** n) 10) 2))))
97. (**else** '(ERROR: CANT RETURN RESULT (NEG)))))) ;; **if** no **case** found **for** n. UNLIKELY
98. ;;END NEGATIVE READOUT
100. ;; START POSITIVE READOUT
101. ((< 0 n 20) ;;ALL VALUES FROM 1-19 are **in** ONES LIST
102. (cons (list-ref ones (- n 1)) '() ))
103. ((< 20 n 30)
104. (cons 'twenty (list (list-ref ones (- n 21)))))
105. ((< 40 n 50)
106. (cons 'fourty (list (list-ref ones (- n 41)))))
107. ((< 50 n 60)
108. (cons 'fifty (list (list-ref ones (- n 51)))))
109. ((< 60 n 70)
110. (cons 'sixty (list (list-ref ones (- n 61)))))
111. ((< 70 n 80)
112. (cons 'seventy (list (list-ref ones (- n 71)))))
113. ((< 80 n 90)
114. (cons 'eighty (list (list-ref ones (- n 81)))))
115. ((< 90 n 100)
116. (cons 'ninety (list (list-ref ones (- n 91)))))
117. ((**integer**? (quotient n 10))
118. (list (list-ref tens (- (quotient n 10) 2)))) ;;used **for** clean tens values without ones
119. (**else** '(ERROR: CANT RETURN RESULT (POS))) ;; **if** no **case** found **for** n. UNLIKELY
120. ;; END POSITIVE READOUT
121. );; END TOP COND
122. )
123. ;;------------------------------------------------------------------------
124. ;;------------------------------------------------------------------------
125. ;; #5 Develop scheme **program** that calculates relevant **word** counts
126. ;; Member **function** **is** a **boolean** **function** that checks **if** an element **is** member **of** given list.
127. ;; INPUT: Input a element, list **of** elements.
128. ;; OUTPUT: Boolean #t **or** #f **if** element **is** a member **of** a list **or** **not**
129. (define (member? x l)
130. (**if** (null? l) #f
131. (**if** (equal? x (car l)) #t
132. (member? x (cdr l)))
133. )
134. )
135. ;;------------------------------------------------------------------------
136. ;; #5a **write** a **function** filterWords that takes a list **of** words **and** irrelevant words, returns an output list **with** irrelevant words removed.
137. ;; INPUT: List **of** words, list **of** words **to** filter out
138. ;; OUTPUT: returns list **with** irrelevant words filtered out
140. (define (filterWords words irrWords)
141. (**if** (null? words) '()
142. (**if** (member? (car words) irrWords)
143. (filterWords (cdr words) irrWords)
144. (append (list (car words)) (filterWords (cdr words) irrWords))
145. )
146. )
147. )
148. ;;------------------------------------------------------------------------
150. ;; #5b **write** a **function** iniWordCountList that takes a list **of** words, creates a **word**-count list
151. ;; INPUT: A list **of** words
152. ;; OUTPUT: each **word** has count **of** 1. So create a **set** **of** sublists within a larger list
154. (define (iniWordCountList words)
155. (map (lambda (x) (list x 1)) words) ;; **for** every **word** **in** words, creates a map **of** the **word** **and** 1.
156. )
158. ;;------------------------------------------------------------------------
160. ;; #5c **write** a **function** mergeWordCounts, takes **in** two inputs
161. ;; First input **is** **word**-count pair **and** second **is** **word**-count list
162. ;; **function** will generate a new **word**-count list.
164. (define (mergeWordCounts a x) ;;  a x. list1 list2
165. (**if** (null? x) (list a)
166. (**if** (equal? (car a) (car (car x)))
167. (cons (list (car a) (+ (car (cdr a)) (car (cdr (car x))))) (cdr x))
168. (cons (car x) (mergeWordCounts a (cdr x)))
169. )
170. )
171. )
172. ;;------------------------------------------------------------------------
173. ;;------------------------------------------------------------------------
175. ;; #5d **write** a **function** mergeByWord that takes **word**-count list, outputs updated **word**-count list that condenses
176. ;; Input: A given **word**-count list [ie '((time 1) (is 1))']
177. ;; Output: Reduced list **with** no duplicates, increases wordcount **of** each element based **on** how many duplicates
179. (define (mergeByWord l) ;; f **is** part c **function**, l **is** input list
180. (**if** (null? l) '()
181. (mergeWordCounts (car l) (mergeByWord (cdr l))) ;;(mergeWordCounts (car l) (mergeWordCounts (cdr l)))
182. )
183. )
185. ;;mergeByWord based **on** Reduce:
186. (define (reduce f l v)
187. (**if** (null? l) v
188. (f (car l) (reduce f (cdr l) v))))
190. ;;------------------------------------------------------------------------
192. ;; #5e Write a **function** relevantWord
193. ;; Input: A list **of** words, A list **of** relevant words
194. ;; Output: Correct **word** count list
196. ;; x **is** input words, y **is** relevant words
197. (define (relevantWordCount x y)
198. (mergeByWord (iniWordCountList (filterWords x y)))
199. )
201. ;;------------------------------------------------------------------------

–