(define unisciListe ;restituisce una lista ;ANALOGO a APPEND

(lambda (lista1 lista2)

(cond ( (null? lista1) lista2 )

( (null? lista2) lista1 )

(else (cons (car lista1)

(unisciListe (cdr lista1) lista2) )))))

(define list-pos

(lambda (lista index) ;lista in cui cercare, index intero

(let ( (k (- (length lista) 1)) ) ; k = lunghezza lista

(cond ((or (< index 0) (> index k)) null ) ;INDICE NON VALIDO

((= index 0) (car lista) ) ;restituisce il primo elemento della lista

(else ;toglie il primo elemento della lista e decrementa i

(list-pos (cdr lista) (- index 1)) )))))

;quando i = 0 restituisce il (car listaFINALE) = (list-ref LISTAINZIALE i)

;LISTA FINALE = (cdr listaINZIALE) ripetuto i VOLTE

(define invertiLista ;ANALOGA a REVERSE Non EFFICIENTE a causa delle grandi quantità di operazioni

(lambda (lista)

(if (null? lista) null

(unisciListe

(invertiLista (cdr lista))

(list (car lista)) )))) ; il primo carattere va messo per ultimo

(define rovescia\_RIC

(lambda (lista1 listaR)

(if (null? lista1)

listaR

(rovescia\_RIC (cdr lista) (cons (car lista1 listaR ) ) ))))

(define rovescia

(lambda (list1)

(rovescia\_RIC list1 null )))

(define before ;restituisce gli elementi a sx della chiave

(lambda (key lista)

(let ( (listLength (- (length lista) 1)) (lastElement (car (reverse lista) )) )

(if (= listLength 0) lista

(if (= key (car lista)) null

(if (= key lastElement)

(reverse (cdr (reverse lista)))

(before key (reverse (cdr (reverse lista)))) ))))))

(define after ;restituisce gli elementi a dx della chiave

(lambda (key lista)

(let ( (listLength (- (length lista) 1)) (lastElement (car (reverse lista) )) )

(if (= listLength 0) lista

(if (= key lastElement) null

(if (= key (car lista))

(cdr lista) ;toglie il primo elemento

(after key (cdr lista)) )))))) ;ricorsione togliendo il primo elemento

(define next (lambda (x) (+ x 1) ) ) ;restituisce il successivo di un numero

(define mappa ;(map function list)

(lambda (proc lista) ;proc = procedura usata ; lista = lista di numeri/caratteri

(let ((k (- (length lista) 1)) )

(if (= k 0) (list (proc (list-ref lista k)))

(append (list (proc (car lista)) )

(mappa proc (cdr lista) ) )))))

;; COMPOSIZIONE FUNZIONALE ;; f: D-> E g: E -> F, h: D -> F

;; per ogni x in E h(x) = g(f(x))

(define compFunc (lambda (g f) (lambda (x) (g (f x)) ) ) ) ;g e f sono due funzioni

(define belong? (lambda (x lista) ;ES 7 LABORATORIO

(if (null? lista) #false

(if (= x (car lista)) #true ;se contiene il carattere restituisce true

(belong? x (cdr lista)) )))) ;ricorsione togliendo il primo elemento

;restituisce la posizione dell'elemento x all'interno della stringa

(define position

(lambda (x lista)

(if (belong? x lista)

(if (= x (car lista)) 0 ;se il numero coincide con il primo elemento restituisce l'indice

(+ (position x (cdr lista)) 1) ) ;incrementa il contatore

#false ))) ;NON APPARTIENE

;restituisce una lista in ordine crescente senza ripetizioni aggiungendo x

(define sorted-ins (lambda (x lista)

(let ( (listLength (- (length lista) 1)) )

(if (< x (car lista)) (cons x lista) ;aggiunge x come primo elemento della lista essendo il minore

(if (= x (car lista)) lista ;se gli elementi sono uguali si restituisce la lista inziale

(if (= listLength 0) (append lista (list x)) ;CASO BASE la lista ha solo 1 elemento

(cons (car lista) (sorted-ins x (cdr lista)) )))))))

(define sorted-list ;restituisce una lista ordinata crescente senza ripetizioni

(lambda (lista)

(let ((listLength (- (length lista) 1) ) )

(if (= listLength 0) lista ; ;se la lista ha solo 1 elemento ritorna la lista stessa

(sorted-ins (car lista) (sorted-list (cdr lista)) )))))

;es 10 02-12-2019

;ritrona una lista di caratteri in ordine crescente aggiungendo char

(define sorted-ins lambda (lista char) ;inserisce il char nella lista in ordine crescente

(let ((lungh (- (length lista) 1)) )

(if (null? lista) (list char)

(if (char>? (car lista) char)

(cons char lista)

(if (char<? (car lista) char) cons (car lista) ;primo elemento della lista

(sorted-ins (cdr lista) char) ) ;lista ordinata togliendo il primo elemento

lista )))))) ;se sono uguali si ritorna la lista stessa

(define sorted-char-list (lambda (stringa)

(let ( (lungh (- (string-length stringa) 1)) )

(if (string=? stringa "") null ;CASO BASE STRINGA VUOTA

(if (= lungh 0) ;stringa di 1 carattere

(list (string-ref stringa lungh)) ;ritorna una lista con il carattere rimasto

(sorted-ins (sorted-char-list (substring stringa 1) ) ;toglie il primo carattere

(string-ref stringa 0) )))))) ;primo carattere della stringa