; LCS = LONGEST COMMON SUBSEQUENCE

(define llcs ;determina la lunghezza (INTERO) della LCS

(lambda (a b) ;a b = STRINGHE

(cond ((or (string=? a "") (string=? b "") ) 0 ) ;se una delle 2 stringhe è vuota ;la LCS di "" "testoCASUALE" è "", perciò la sua lunghezza è 0

((char=? (string-ref a 0) (string-ref b 0)) (+ 1 (llcs (substring a 1) (substring b 1)) ) )

(else ;le due stringhe iniziano con 2 char diversi

(max (llcs (substring a 1) b)

(llcs a (substring b 1)) )))))

;-------------------------------------------------------------

(define longer ;restituisce la stringa più lunga tra le 2

(lambda (str1 str2) ;STRINGA

(let ( (k1 (string-length str1))

(k2 (string-length str2)) )

(if (< k1 k2) str2 str1 ))))

;LCS determina la sottosequenza comune ordinata da SX a DX delle due stringhe inserite

(define lcs ;restituisce la STRINGA LCS

(lambda (a b) ;a b = STRINGHE

(cond ((or (string=? a "") (string=? b "")) "" ) ;se una stringa è vuota si restituisce LCS = ""

((or (= (string-length a) 1) (= (string-length b) 1))

;in caso di stringa lunga 1 solo char controllare il char con tutti gli altri della stringa maggiore

(if (= (string-length a) 1)

(if (char=? (string-ref a 0) (string-ref b 0)) ;controlla il primo carattere delle due stringhe

(string-append a (lcs a (substring b 1)) ) ;toglie il primo carattere di b

(lcs a (substring b 1)) )

(if (char=? (string-ref a 0) (string-ref b 0)) ;controlla il primo carattere delle due stringhe

(string-append b (lcs (substring a 1) b) ) ;toglie il primo carattere di a

(lcs (substring a 1) b ) ) ) )

( (char=? (string-ref a 0) (string-ref b 0)) ;le due stringhe iniziano con lo stesso char

; carattere uguale + la LCS delle due stringhe senza il primo carattere

(string-append (substring a 0 1) ;PRIMO CARATTERE di A o B

(lcs (substring a 1) (substring b 1)) )) ;toglie il primo carattere da entrambe le stringhe

(else ;le 2 stringhe iniziano con char diversi

(longer ;restituisce la stringa più lunga tra due LCS

(lcs (substring a 1) b) ;toglie il primo char di a

(lcs a (substring b 1)) ))))) ;toglie il primo char di b

;-------------------------------------------------------------

;ESERCIZIO 3 esame 21-01-2019

;(xlcs "atrio" "arto") → "\*/\*/t\*" ;(xlcs "cincia" "piani") → "/p\*a\*/\*/"

(define stars ;conta il numero di "\*" in q

(lambda (q) ;q; STRINGA

(if (string=? q "") 0

(let ( (n (stars (substring q 1))) )

(if (char=? (string-ref q 0) #\\*) (+ n 1) n )))))

(define better ;restituisce la stringa con più "\*", cioè con più caratteri comuni

(lambda (u v) (if (< (stars u) (stars v)) v u )))

(define xlcs

(lambda (s t)

(cond ( (string=? s "") t ) ;caratteri restanti non comuni da esprimere in chiaro

( (string=? t "") ;se S ha un carattere diverso dal primo della LCS

(string-append "/" (xlcs (substring s 1) t) ) ) ;TOGLIE IL PRIMO CARATTERE DI S

( (char=? (string-ref s 0) (string-ref t 0))

(string-append "\*" (xlcs (substring s 1) (substring t 1)) ) )

(else (better (string-append "/" (xlcs (substring s 1) t) );caratteri da rimuovere da S

(string-append (substring t 0 1) (xlcs s (substring t 1)) ) ))))) ;caratteri da rimuovere da T

;-------------------------------------------------------------

;ES 3 02-12-2019

;(lcs "pino" "pino") → ((1 1 "p") (2 2 "i") (3 3 "n") (4 4 "o")) ;(lcs "pelo" "peso") → ((1 1 "p") (2 2 "e") (4 4 "o"))

(define lcs ; restituisce una lista di terne (posU posV carattere\_comune)

(lambda (u v) (lcs-rec 0 u 0 v ))) ;= i ;= j ; u, v: stringhe

(define better ;restituisce la lista più lunga

(lambda (x y) ;x y : LISTE

(if (< (length x) (length y)) y x )))

(define lcs-rec

(lambda (i u j v) ;u v stringhe; i j = posizioni

(cond ( (or (string=? u "") (string=? v "")) null )

( (char=? (string-ref u 0) (string-ref v 0))

(cons (list (+ i 1) (+ j 1) (string (string-ref v 0))) ;prima lista contenente l'elemento comune

(lcs-rec (+ i 1) (substring u 1) (+ j 1) (substring v 1)) )) ;richiama togliendo il carattere comune e incrementando le posizioni

(else (better ;vengono aumentate le posizioni per indicare

(lcs-rec (+ i 1) (substring u 1) j v) ;tolto il primo char di U

(lcs-rec i u (+ j 1) (substring v 1)) ))))) ;tolto il primo char di V

;-------------------------------------------------------------

;ESERCIZIO 1 25-07-2019 ;LISTA DI TUTTE LE LCS

;(lcs-list "arto" "atrio") → ("ato" "aro")

(define lcs-list ; val : lista di stringhe

(lambda (u v) (lcs-rec "" u v) )) ; u, v : stringhe

(define best ;restituisce la stringa più lunga

(lambda (a b) ;a, b: LISTE

(if (< (string-length (car a)) (string-length (car b))) b

(if (> (string-length (car a)) (string-length (car b))) a

(cons (car a) (list (car b))) ))))

(define lcs-rec

(lambda (p u v) ;p = LCS progressiva

(cond ( (or (string=? u "") (string=? v "")) (list p) )

( (char=? (string-ref u 0) (string-ref v 0))

(lcs-rec (string-append p (substring u 0 1)) ;aggiunto a p il carattere comune tra u v

(substring u 1)

(substring v 1) ) )

(else (best (lcs-rec p (substring u 1) v)

(lcs-rec p u (substring v 1)) )))))

;ES 7 02-12-2019

(define shared ;restituisce la lista ordinata degli elementi comuni

(lambda (u v) ;u v: LISTE ORDINATE di NUMERI INTERI POSITIVI

(if (or (null? u) (null? v)) null

(if (= (car u) (car v))

(cons (car u)

(shared (cdr u) (cdr v)) );toglie i primi elementi da entrambe

(better (shared (cdr u) v) (shared u (cdr v)) )))))