(define **toDec** ;deve convertire la stringa espressa con i simboli dell’insieme in numero decimale (lambda (insieme stringa) ;stringa = stringa di NUMERO INTERO insieme: valori in ordine crescente

(let ( (k (- (string-length stringa) 1)) ; K = ULTIMO INDICE DELLA STRINGA

(base (string-length base)) ) ;base = numero decimale da usare nei calcoli

(if (= k 0) (nuovaCifra (string-ref stringa k) base 0) ;la stringa contiene un solo elemento

(+ ;NUOVACIFRA + BASE \* PREC (\* base (toDec insieme (substring stringa 0 k)))

;funzione ricorsiva a cui viene passata la stringa senza l'ultimo carattere sommato progressivamente

(nuovaCifra (string-ref stringa k) base 0) ) ))))

;viene calcolato il valore numerico del carattere all'ultima posizione della stringa

;restituisce le cifre prima del “.”

(define **parte\_intera**

(lambda (stringa)

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

(if (char=? (string-ref stringa k) #\.) (substring stringa 0 k)

(parte\_intera (substring stringa 0 k)) )))) ;richiama se stessa togliendo l'ultimo carattere

(define **parte\_fraz** ;restituisce i caratteri a DX della virgola

(lambda (stringa)

(if (char=? (string-ref stringa 0) #\.) (substring stringa 1)

(parte\_fraz (substring stringa 1)) ))) ;richiama se stessa passando la stringa senza il carattere iniziale

;restuisce il **valore numerico del carattere** della stringa **in base ai simboli dell'insieme**. INDICE\_INSIEME = VALORE NUMERICO

(define **nuovaCifra**

(lambda (carattere insieme k) ;**k è l'indice in cui cercare il carattere**, vale 0 se è la prima volta che si chiama la funzione nuovaCifra

(if (< k (string-length insieme)) ;k deve essere minore della lunghezza della stringa, per non incorrere in una lettura fuori indice

(if (char=? (string-ref insieme k) carattere ) k ;CARATTERE TROVATO.

(nuovaCifra carattere insieme (+ k 1)) ) ;incrementa il contatore per cercare ancora nell'insieme

(string-append "ERRORE:" (string carattere) " non appartiene all'insieme") ))) ;stampa a video dell'errore

(define **checkSign** ;restituisce il segno della stringa

(lambda (stringa)

(cond ((char=? (string-ref stringa 0) #\+) #\+)

((char=? (string-ref stringa 0)#\-) #\- )

(else #\a )))) ; carattere indicante della mancanza del segno

(define **hasPoint? ;TRUE o FALSE se trova o meno il “.”**

(lambda (string)

(let ((k (- (string-length string) 1)))

(cond ((char=? (string-ref string k) #\.) #true ) ;controlla se l'ultimo carattere è una virgola

((= k 0) #false ) ;siccome ha gia controllato se è presente il . allora ritorna falso

(else (hasPoint? (substring string 0 k)) ))))) ;richiama se stessa togliendo l'ultimo carattere

(define **bin-rep->number** ;funzione chiamata dall'utente

(lambda (string) (rep->number "01"string) ))

(**define rep->number** ;funzione chiamata dall'utente per conversione generica in BASE 10

(lambda (insieme stringa) ;insieme contiene i simboli dell'alfabeto da cui partire con la conversione

(if (hasPoint? stringa) ;controllo se il numero inserito ha la VIRGOLA

(let ( (numdec ;valore decimale della stringa totale senza la virgola

(toDec insieme (string-append

(parte\_intera (substring stringa 1))

(parte\_fraz (substring stringa 1)) )))

(divisore ;numero da dividere per ottenere la conversione giusta

(expt (string-length insieme) (string-length (parte\_fraz (substring stringa 1))) ) ))

(cond ( (char=? (checkSign stringa) #\- ) (\* (/ numdec divisore) -1) )

(else (/ numdec divisore)))) ;se è positivo o se non ha segno si fa la divisione normale

;;NON TROVA LA VIRGOLA

(cond ( (char=? (checkSign stringa) #\+) (toDec insieme (substring stringa 1) ))

( (char=? (checkSign stringa) #\-) (\* -1 (toDec insieme (substring stringa 1) ) ))

(else (toDec insieme stringa ) )))))

;ESERCIZIO 3 22-01-2018 ;(powers-of-two 26) → (16 8 2)

(define powers-of-two ;restituisce la lista contenente le potenze di 2

;;in ordine decrescente che sommate danno origine a n

(lambda (n) ;n: INTERO >= 0

(cond ( (= n 0) Null ) ;CASO BASE

( (= n 1) (list 1) )

(else (let ( (lg (inexact->exact ;Log2(n) : approssimato difetto INTERO

(floor (/ (log n) (log 2)) )) ) )

(cons (expt 2 lg) ;(potenza di 2) < n più vicina a n stesso

(powers-of-two (- n (expt 2 lg))) ; n - (potenza di 2) ))))))

;ES 2 02-12-2019

(define offset (char->integer #\0) )

(define last-digit ;ritorna il simbolo maggiore codificabile

(lambda (base)

(integer->char ;carattere indicante la cifra codificata

(+ (- base 1) offset )))) ;posizione del carattere 0 ;base - 1 = simbolo maggiore

(define next-digit ;ritorna il simbolo successivo a quello inserito

(lambda (dgt) ;dgt = char

(string ;inserimiento del carattere calcolato in una stringa da ritornare

(integer->char ;riconversione in char della posizione calcolata

(+ (char->integer dgt) 1) )))) ;posizione successiva a quella di dgt

(define increment ;calcola il valore numerico dell'incremento di 1 unità del numero inserito

(lambda (num base) ;num stringa di simboli appartenente all'alfabeto base ; 2 <= base <= 10

(let ((digits (string-length num))) ;lunghezza della stringa numerica

(if (= digits 0) "1"

(let ((dgt (string-ref num (- digits 1)))) ;carattere in ultima posizione della stringa num

(if (char=? dgt (last-digit base)) (string-append (increment (substring num 0 (- digits 1)) base)

"0" ) (string-append (substring num 0 (- digits 1)) (next-digit dgt) )))))))