(define btd-rep ; valore: stringa "-", ".", "+"

(lambda (v) ; v: [-1, 0, 1]

(cond ((= v -1) "-")

((= v 0) ".")

((= v +1) "+") )))

(define btd-val ;converte il simbolo nel suo valore numerico

(lambda (s)

(cond

((string=? s "-") -1 )

((string=? s "+") 1 )

(else 0 ) )))

;conversione numero decimale in stringa BTR

(define btr-rep

(lambda (n) ; n: intero

(let ((r (remainder n 3)) (q (quotient n 3)) ) ; n = 3q + r dove -2 <= r <= +2

(cond ((= r -2)

(string-append (btr-rep (- q 1)) (btd-rep +1))) ; n = 3q - 2 ==> n = 3(q-1) + 1, q-1 < 0

((= r +2)

(string-append (btr-rep (+ q 1)) (btd-rep -1))) ; n = 3q + 2 ==> n = 3(q+1) - 1, q+1 > 0

((= q 0)

(btd-rep r)) ; n = r

(else (string-append (btr-rep q) (btd-rep r))) )))) ; n = 3q + r, q <> 0

;conversione stringa BTR in numero decimale

(define converti

(lambda (string)

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

(if (= k 0)

(btd-val string )

;k != 0

(+ ;nuovaCifra + BASE\* Risulatato PREC

(\* 3 (converti (substring string 0 k) ))

(btd-val (substring string k) ) ))))) ;ultima cifra della stringa

;restituisce la cifra delle unità della somma tra i 3 caratteri inseriti

(define **btr-digit-sum** ; val: carattere +/./-

(lambda (u v c) ; u, v, c: caratteri +/./-

(cond ((char=? u #\-) ; u v c

(cond ((char=? v #\-)

(cond ((char=? c #\-) #\.) ; - - -

((char=? c #\.) #\+) ; - - .

((char=? c #\+) #\-))) ; - - +

((char=? v #\.)

(cond ((char=? c #\-) #\+) ; - . -

((char=? c #\.) #\-) ; - . .

((char=? c #\+) #\.))) ; - . +

((char=? v #\+) c))) ; - + c

((char=? u #\.)

(cond ((char=? v #\-)

(cond ((char=? c #\-) #\+) ; . - -

((char=? c #\.) #\-) ; . - .

((char=? c #\+) #\.))) ; . - +

((char=? v #\.) c) ; . . c

((char=? v #\+)

(cond ((char=? c #\-) #\.) ; . + -

((char=? c #\.) #\+) ; . + .

((char=? c #\+) #\-))))) ; . + +

((char=? u #\+)

(cond ((char=? v #\-) c) ; + - c

((char=? v #\.)

(cond ((char=? c #\-) #\.) ; + . -

((char=? c #\.) #\+) ; + . .

((char=? c #\+) #\-))) ; + . +

((char=? v #\+)

(cond ((char=? c #\-) #\+) ; + + -

((char=? c #\.) #\-) ; + + .

((char=? c #\+) #\.))))) ; + + + ))) ;FINE

(define **btr-carry-sum** ; val: carattere +/./-

(lambda (u v c) ; u, v, c: caratteri +/./-

(cond ((char=? u #\-) ; u v c

(cond ((char=? v #\-)

(cond ((char=? c #\-) #\- ) ; - - -

((char=? c #\.) #\- ) ; - - .

((char=? c #\+) #\. ))) ; - - +

((char=? v #\.)

(cond ((char=? c #\-) #\- ) ; - . -

((char=? c #\.) #\. ) ; - . .

((char=? c #\+) #\. ))) ; - . +

((char=? v #\+) #\. ))) ; - + c

((char=? u #\.)

(cond ((char=? v #\-)

(cond ((char=? c #\-) #\- ) ; . - -

((char=? c #\.) #\. ) ; . - .

((char=? c #\+) #\. ))) ; . - +

((char=? v #\.) c) ; . . c

((char=? v #\+)

(cond ((char=? c #\.) #\.) ; . + -

((char=? c #\.) #\. ) ; . + .

((char=? c #\+) #\+ ))))) ; . + +

((char=? u #\+)

(cond ((char=? v #\-) c) ; + - c

((char=? v #\.)

(cond ((char=? c #\-) #\.) ; + . -

((char=? c #\.) #\.) ; + . .

((char=? c #\+) #\+))) ; + . +

((char=? v #\+)

(cond ((char=? c #\-) #\. ) ; + + -

((char=? c #\.) #\+ ) ; + + .

((char=? c #\+) #\+))))) ; + + + ))) ;FINE

(**define normalized-btr** ;rimuove gli 0 più significativi

(lambda (str)

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

(cond ((= k 0) ;se l'ultimo char = . si ritorna .

(if (char=? (string-ref str k) #\.) "." str )

)

(else (if (char=? (string-ref str 0) #\.) (normalized-btr (substring str 1 )) str )

) ))))

;restituisce la **cifra meno significativa di str**

(define **lsd**

(lambda (str)

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

(if (= k -1) #\. ;stringa str vuota (string-ref str k) ))))

;**restituisce tutta la stringa - lsd**

(define **head**

(lambda (str)

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

(if (= k -1) "" ;stringa str vuota (substring str 0 k) ))))

(define **btr-sum**

(lambda (add1 add2)

(btr-carry-sum add1 add2 #\.) )) ; #\. RIPORTO INIZIALE

(define **btr-carry-sum**

(lambda (add1 add2 c)

(let ((k1 (- (string-length add1) 1)) (k2 (- (string-length add2) 1)))

(cond ;se non sono uguali si aggiunge (kMag - Kmin) "." a SX della stringa più corta

((< k1 k2) ; add1 < add2 perciò si allunga add1

(btr-carry-sum (string-append (substring "......." 0 (- k2 k1)) add1) add2 c) )

((< k2 k1) ; add2 < add1 perciò si allunga add2

(btr-carry-sum add1 (string-append (substring "......." 0 (- k1 k2)) add2) c) )

((or (= k1 0)(= k2 0)) ;restituisce "carry"+"sum"

(string (btr-carry (string-ref add1 k1) (string-ref add2 k2) c)

(btr-digit-sum (string-ref add1 k1) (string-ref add2 k2) c)) )

(else ;toglie l'ultimo carattere a DX della stringa, e imposta come c il btr-carry della somma delle ultime 2 cifre

(normalized-btr (string-append

(btr-carry-sum (substring add1 0 k1) (substring add2 0 k2)

(btr-carry (string-ref add1 k1) (string-ref add2 k2) c) ;carry )

;btr-digit-sum dell'ultima cifra delle stringhe

(string (btr-digit-sum (string-ref add1 k1) (string-ref add2 k2) c)) ) ) )))))

;ESERCIZIO 2 03-09-2019

(define **btd-rep**

(lambda (v)

(cond ( (= v -1) "-" )

( (= v 0) "." )

( (= v +1) "+" ))))

(define **btd-val**

(lambda (char)

(cond ( (char=? char #\-) -1 )

( (char=? char #\+) 1 )

( (char=? char #\.) 0 ) )))

(define **btr-rep**

(lambda (n)

(let ( (r (remainder n 3))

(q (quotient n 3)) )

(cond ( (= r -2) (string-append (btr-rep (- q 1)) (btd-rep +1)) )

( (= r +2) (string-append (btr-rep (+ q 1)) (btd-rep -1)) )

( (= q 0) (btd-rep r) )

(else (string-append (btr-rep q) (btd-rep r)) )))))

(define **btr-list-rec ;ritorna una lista**

(lambda (s i) ;s: STRINGA, i: INDICE

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

(if (= k 0) ;stringa con 1 solo carattere

(list (\* (expt 3 i)

(btd-val (string-ref s k)) ))

(append (btr-list-rec (substring s 0 k) (+ i 1))

(list (\* (expt 3 i)

(btd-val (string-ref s k)) )) )))))

(define **btr-list** ;restituisce una lista con i valori numerici pesati in base all'indice

(lambda (n) ;n: INTERO positivo/negativo

(btr-list-rec (btr-rep n) 0) ))