

28/06/2001

MARASPIN STEFANO
Matr. N° 57054

LABORATORIO DI PROGRAMMAZIONE

ESERCIZI SCHEME

; ESERCIZIO 1:

*; Questa procedura riceve in ingresso [da input utente] due liste,
; di cui ogni elemento rappresenta una posizione decimale, con la più
; significativa ovviamente posizionata as sinistra e ne esegue la somma.
; Le liste vengono dapprima rovesciate in maniera tale da poter essere
; calcolati eventuali riporti (carry); successivamente viene eseguito un
; controllo per verificare la lunghezza delle liste. Infine viene fatta
; una chiamata alla funzione somma che ne esegue appunto la somma degli
; elementi.*

```
(define sum-lists
  (lambda ()
    (display "Inserire una lista di numeri interi compresi tra 0 e 9:")
    (newline)
    (let ((list1 (reverse (read))))
      (newline)
      (display "Inserire una seconda lista di interi compresi tra 0 e 9:")
      (newline)
      (let ((list2 (reverse (read))))
        (reverse (somma (parifica list1 list2) (parifica list2 list1) 0))))))
```

; Procedure di Supporto:

```
(define parifica
  (lambda (source confronto)
    (if (< (length source) (length confronto))
        (parifica (append source '(0)) confronto) source)))
```

```
(define somma
  (lambda (list1 list2 carry)
```

*; Se esaminata l'ultima posizione delle liste, il valore di carry
; é 0, vengono semplicemente ritornata la lista somma.
; Altrimenti, alla stessa viene aggiunto il carry*

```
  (if (and (null? list1) (null? list2))
      (if (= 0 carry)
          null
          (cons 1 null))
```

*; A seconda che vi sia o meno riporto, si procede a formare la lista, unendo alla somma del
; car delle liste il risultato di una chiamata ricorsiva a somma, il cui parametro carry
; cambia appunto a seconda del risultato del confronto tramite if*

```
    (if (> (+(+ (car list1) (car list2)) carry) 9)
        (cons (remainder (+(+ (car list1) (car list2)) carry) 10)
              (somma (cdr list1) (cdr list2) 1))
        (cons (+(+ (car list1) (car list2)) carry)
              (somma (cdr list1) (cdr list2) 0))))
  )
)
```

; ESERCIZIO 2:

*; Questa procedura Scheme accetta in ingresso due liste ordinate [da input utente]
; e ritorna una lista, risultante dal merge delle stesse. Anch'essa è ordinata.
; Per costruirla si esegue una procedura che analizza il car di entrambe le liste
; e in base ad esso, provvede ad ordinarne gli elementi.*

(define merge-lists

```
(lambda ()  
  (display "Inserire una lista ordinata.")  
  (newline)  
  (let ((list1 (read)))  
    (display "Inserire la seconda lista ordinata.")  
    (newline)  
    (let ((list2 (read)))  
      (order list1 list2))))))
```

(define order

```
(lambda (list1 list2)  
  (cond ((null? list1) list2)  
        ((null? list2) list1)  
        (> (car list1) (car list2))  
          (cons (car list2) (order list1 (cdr list2))))  
        (< (car list1) (car list2))  
          (cons (car list1) (order (cdr list1) list2))  
        (= (car list1) (car list2))  
          (cons (car list1) (order (cdr list1) (cdr list2))))  
        (else (display 'Error: Unexpected Result))))))
```

; ESERCIZIO 3:

*; Questa procedura costruisce un albero di ricerca binario, dopo aver ricevuto in
; ingresso due numeri n ed m, rappresentanti gli estremi.
; La radice è $p = (n+m) / 2$, mentre i sottoalberi sx e dx, si basano su una
; ricorsione, applicata agli estremi n p-1, p+1 m*

(define build-tree

```
(lambda ()  
  (display "Inserire l'estremo inferiore dell' intervallo.")  
  (newline)  
  (let ((n (read)))  
    (display "Inserire l'estremo superiore dell' intervallo.")  
    (newline)  
    (let ((m (read)))  
      (make-tree n m))))))
```

(define make-tree

```
(lambda (n m)  
  (let ((p (quotient (+ n m) 2)))  
    (if (> n m)  
        null  
        (list p (make-tree n (- p 1)) (make-tree (+ p 1) m))))))
```

ESERCIZI JAVA

```
// Esercizio 4:
// La seguente classe permette di svolgere le verifiche richieste su un
// numero n ricevuto in ingresso.
// Una semplice interfaccia per l'utente permette di selezionare quale
// metodo della classe utilizzare sul parametro n, comune a tutte le
// funzioni.

class numProp {

    public void maxval (int n) {

// Inizializzo una variabile times che mi tornerà in uscita il
// valore di k, nell'espressione di tipo 2^k da verificare.

        int times = 0;

// Qui duplico la variabile n, in maniera tale da conservare
// Valore originale per l'output

        int temp = n;

// Ciclo for che verifica la divisibilità per due e controlla
// (temp mod 2). Ogni volta che questa operazione é possibile,
// la variabile times viene incrementata.

        for (times = 0; (temp%2)==0; times++)
            temp=temp/2;

// Facile ora il controllo. Se times > 0, vuol dire che la divisione
// per 2^k é stata possibile e times ritorna appunto il valore k

        if (times>0)
            System.out.println (n + " e' divisibile per 2^" +
                times);
        else
            System.out.println (n + " non e' divisibile per due");
    }

    public void divisori (int n) {

// Inizializzate le variabili del numero di divisori e il più
// grande di essi

        int divisori=0;
        int max=0;

// Verifico ora la divisibilità del numero per tutti i numeri
```



```
        System.out.println ("Il numero " +
            n + " non e' un
fattoriale.");
    }
```

```
public void existsqr (int n) {
```

```
// Semplice inizializzazione di una variabile k usata per il
// confronto con n
```

```
    int k=1;
```

```
// Confronto tra k^k ed n, finché ovviamente k^k < n
```

```
    while (Math.pow (k,k)<n)
        k=k+1;
```

```
// Se a questo punto k^k = n, é verificata la proprietà richiesta
```

```
    if (Math.pow(k,k)==n)
        System.out.println ("Il numero " + n +
            " puo' essere " +
            " scritto come " +
            k + "^" + k + ".");
```

```
    else
        System.out.println ("Il numero " + n +
            " non puo' essere " +
            " scritto come k^k.");
    }
```

```
public static void main (String [] args) {
```

```
// Semplice interfaccia per l'utente
// Viene dapprima inserito il numero da analizzare
```

```
    EasyIn e =new EasyIn ();
    System.out.println ("Inserire un numero " +
        "intero positivo:");
```

```
    int n = e.readInt();
```

```
    int selection;
```

```
    numProp dummy = new numProp();
```

```
// Ora viene proposto l'elenco delle opzioni consentite
// dal programma
```

```

        System.out.println ("\nQuale Verifica Intendi " +
            " Eseguire sul Numero " +
            n +"?\n");

System.out.println ("1 - Se é divisibile per 2^k");
System.out.println ("2 - Trovare il numero di Divisori e il Maggiore tra essi");
System.out.println ("3 - Se esiste un intero tale che "+ n +" é uguale a k!");
System.out.println ("4 - Se esiste un intero tale che "+ n +" é uguale a k^k\n");

        selection = e.readInt();

// In base alla selezione, si fa la chiamata al metodo corrispondente tra
// quelli precedentemente implementati

        switch (selection) {

                case 1: dummy.maxval(n); break;
                case 2: dummy.divisori(n); break;
                case 3: dummy.exist(n); break;
                case 4: dummy.existsqr(n); break;
                default:
System.out.println ("E' stata effettuata una scelta non valida.\n"+
                    "Ripetere l'esecuzione del programma.\n");
                }

        }
}

```

```
// Esercizio 5;  
// Acquisito in entrata il vettore array, viene analizzato, scorrendone gli indici  
// Se il valore corrispondente di ognuno é maggiore del numero k, vengono spostati  
// su un vettore parallelo, altrimenti vengono spostati alla prima posizione libera  
// disponibile Il vettore bigger[] contenente gli elementi > k viene poi riversato  
// in coda al vettore array[] che viene finalmente mostrato, a ripartizione  
// avvenuta
```

```
class Partition {  
  
    public static void main (String[] args) {  
  
        EasyIn e=new EasyIn();  
        System.out.print ("Inserire la dimensione dell'array : ");  
        int n=e.readInt();  
  
        // Dichiarazione dell'Array  
        int [] array = new int [n];  
  
        // Creazione di un Array della stessa dimensione del primo  
        // Sarà impiegato per la memorizzazione dei dati > n  
        // Gli altri saranno invece lasciati in array[], anche se  
        // traslati. Sarà poi un semplice merge a fornire l'array  
        // in uscita che soddisfa la proprietà richiesta  
  
        int [] bigger = new int [n];  
  
        // Variabili temporanee usate come indici dei vettori  
        // array[] e bigger[] nei quali saranno suddivisi i numeri  
        // a seconda del loro confronto con n  
  
        int indexb = 0;  
        int q = 0;  
  
        System.out.println ("Inserire gli elementi interi dell'array :");  
        for (int i=0; i<n ; i++) {  
            int t=e.readInt();  
            array[i]=t;  
        }  
  
        System.out.print ("Inserire un numero intero per la partizione : ");  
        int k=e.readInt();  
  
        for (int x=0; x<n; x++) {  
  
            if (array[x] > k) {  
                bigger[indexb] = array[x];  
                indexb++;  
            }  
  
            else {  
                array[q] = array[x];  
                q++;  
            }  
        }  
    }  
}
```

```
    }  
}  
  
// A questo punto non mi resta che copiare i valori momentaneamente  
// memorizzati su bigger su array, a partire dalla posizione q  
// raggiunta  
// e dalla nuova posizione di partenza di bigger, cioè 0.  
  
indexb = 0;  
  
for (int tmp = q; q<n; q++) {  
    array[q] = bigger[indexb];  
    indexb++;  
}  
  
System.out.print ("L'array ripartito e': {");  
for (int i=0; i<(n-1); i++) {  
    System.out.print (array[i] +",");  
}  
  
System.out.print (array[(n-1)] + "});"  
  
}  
}
```