Име:....., ф.н.:....., гр.:.....

ТЕСТ ПО ФУНКЦИОНАЛНО ПРОГРАМИРАНЕ

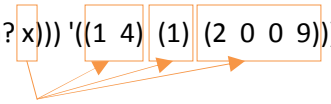
14.01.2009 г.

Спец. Компютърни науки, гр. 3 и спец. Информатика, гр. 4

ОТГОВОРИ И ПРИМЕРНИ РЕШЕНИЯ

Задача 1. Каква е оценката на:

```
> (map (lambda (x) (accumulate * 1 (filter even? x))) '((1 4) (1) (2 0 0 9)))
(4 1 0)
```



Задача 2. Да се дефинират функциите от по-висок ред за работа със списъци map и filter чрез използване на accumulate за списъци:

```
(define (accumulate combinator null-value l)
  (if (null? l) null-value
          (combinator (car l) (accumulate combinator null-value (cdr l)))))

(define (map f l)
   (accumulate (lambda (x y)   (cons  (f x)  y)          )    '()    l    ))

(define (filter pred? l)
   (accumulate (lambda (x y)   (if  (pred? x)   (cons x y)  y)          )    '()    l    ))
```

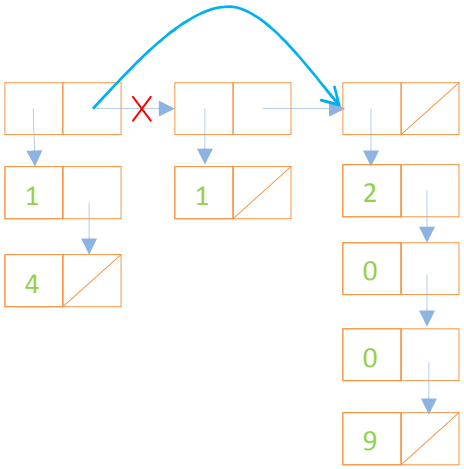
Задача 3. Да се дефинира процедура, която намира височината на двоичното дърво tree:

```
(define (height  tree)
          (if (empty-tree? tree) 0
                    (+ 1 (max (height (left-tree tree))
                              (height (right-tree tree)))))                    )
```

Задача 4. Какво прави следната процедура?

```
(define (f! x)
    (cond ((null? x) '())
          ((null? (cdr x)) '())
          (else (set-cdr! x (cddr x))
                (f! (cdr x)))))
```

Изтрива деструктивно от списъка x елементите на четни позиции.

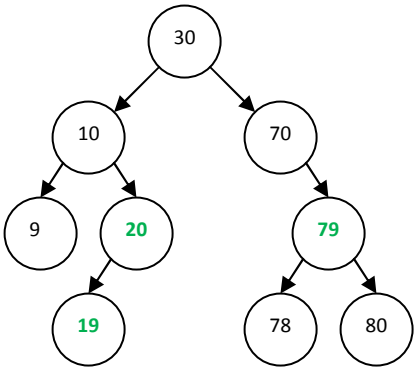



```
> (define l '((1 4) (1) (2 0 0 9)))
> (f! l)
> l
```

((1 4) (2 0 0 9))

Задача 5.

а) Попълнете дървото така, че да бъде двоично наредено:

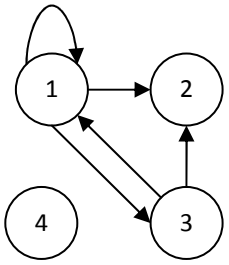


б) Довършете процедурата, която добавя елемент в двоично наредено дърво с цели числа по върховете:

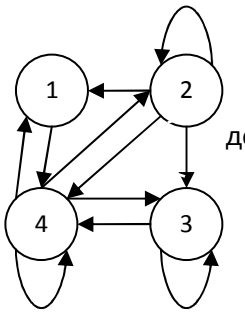
```
(define (add x tree)
   (cond ((null? tree)      (make-tree      x              the-empty-tree          the-empty-tree          ) )
         ((< x (root tree)) (make-tree      (root tree)    (add x (left-tree tree)) (right-tree tree)       ) )
         (else              (make-tree      (root tree)    (left-tree tree)        (add x (right-tree tree)) ) )))
```

Задача 6.

а) Даден е графът G. Напишете представянето му чрез асоциативен списък:



G



допълнението на G

```
(define  G '(    (1  1  2  3)  (2)  (3  1 2)  (4)                                    ))
```

⇔ (1 . (1 2 3)) (2 . ()) (3 . (1 2)) (4 . ())

б) Напишете процедура, която намира допълнението на графа g. Наготово могат да се използват процедури за работа със списъци като union.

```
(define (complement  g)
   (let ((vertices (map car g)))
      (map (lambda (x) (cons (car x) (difference vertices (cdr x)))) g))                )
```

Например при
x = (3 1 2) → (cons 3 (difference (1 2 3 4) (1 2))), т.е. (3 . (3 4)) ⇔ (3 3 4)

в) Довършете процедурата, която изтрива реброто (a,b) от графа g:

```
(define (delete-rib! a b g)
      (let ((x (assq a g)))
         (if (not (null? (cdr x)))  (set-cdr!    x    (filter     (lambda (y) (not (= y b)))          (cdr x))))))
```