КОНТРОЛНА РАБОТА № 1 ПО ФУНКЦИОНАЛНО ПРОГРАМИРАНЕ
Специалност „Компютърни науки“, 2-ри курс, 1-ви поток
19.11.2016 г.

Задача 1. Да се напише функция `(sum-numbers a b)`, приемаща два аргумента, която намира сбора на числата в интервала `[a,b]`, чиито цифри са в низходящ (>=) ред.
Примери:

```
(sum-numbers 1 9) → 45
(sum-numbers 199 203) → 200
(sum-numbers 219 225) → 663
```

Задача 2. Да се напише функция (`num-bigger-elements lst`), която за даден списък от числа `lst` връща като резултат списък с елементи от вида (lst_i n_i), където lst_i е i-тият елемент на `lst`, а n_i е броят на елементите на `lst`, които са по-големи от lst_i.
Примери:

```
(num-bigger-elements '(5 6 3 4)) → '((5 1) (6 0)
                                     (3 3) (4 2))
(num-bigger-elements '(1 1 1)) → '((1 0) (1 0) (1 0))
```

Задача 3. Ако `f` и `g` са числови функции и `n` е естествено число, да се напише функция от по-висок ред (`switchsum f g n`), която връща като резултат функция, чиято стойност в дадена точка `x` е равна на `f(x)+g(f(x))+f(g(f(x)))+` ... (сумата включва `n` събираеми).
Примери:

```
((switchsum (lambda (x) (+ x 1))
            (lambda (x) (* x 2)) 1) 2) → 3
((switchsum (lambda (x) (+ x 1))
            (lambda (x) (* x 2)) 2) 2) → 9
((switchsum (lambda (x) (+ x 1))
            (lambda (x) (* x 2)) 3) 2) → 16
((switchsum (lambda (x) (+ x 1))
            (lambda (x) (* x 2)) 4) 2) → 30
```

Задача 4. Нека дефинираме операцията `A ~~ B` между квадратни матрици, където където `A` е матрица от произволни стойности, а `B` е матрица от двуаргументни функции, както следва:
`A ~~ B` ни дава нова матрица `C`, **която се състои от резултата на частичното прилагане на функциите от** `B` **с първи аргумент от** `A`. Частичното прилагане се извърша между съответните по позиция елементи на двете матрици, т.е. функцията `f` с позиция `(i,j)` в матрицата `B` се прилага частично към аргумента `a` с позиция `(i,j)` в матрицата `A`.

1) Напишете функция `(~~ A B)` която реализира гореописаната операция.
2) Напишете функция `(<<~ A x)`, която приема матрица от едноаргументни функции `A` и произволна стойност `x`. Оценката на обръщението към функцията е нова матрица, всеки от елементите на която е равен на резултата от прилагането на съответната по позиция функция от `A` върху аргумент `x`.

Пример:

```
(define A (list (list 1 2)
                (list 3 4)))
(define F (list (list + +)
                (list + +)))
(<<~ (~~ A F) 1) → '((2 3) (4 5))
```

Задача 5. Да се напише функция `(game-of-life board)`, която симулира една итерация от "Играта на живота", наподобяваща зараждането, развитието и упадъка на съвкупност от живи организми.

Правила на „Играта на живота“

Вселената е представена като двумерна матрица. Елементите на матрицата се разглеждат като *клетки*, всяка от които може да се намира в едно от две възможни състояния: *жива* или *мъртва* (*празна)*. Времето тече на дискретни стъпки (итерации). Всяка клетка взаимодейства със своите не повече от осем съседи (съседните клетки по ред, стълб и диагонал) и пресмята новото си състояние (състоянието си на следващата итерация) на базата на състоянията на своите съседи на текущата итерация, спазвайки следните правила:

- (***живот***) всяка жива клетка с две или три живи съседни клетки остава жива на следващата итерация;
- (***умиране***) всяка жива клетка с по-малко от две живи съседни клетки умира от самота, а всяка жива клетка с повече от три живи съседни клетки умира от пренаселване;
- (***раждане***) всяка празна клетка с точно три живи съседни клетки се превръща в жива клетка.

Всички раждания и умирания на дадена итерация се извършват едновременно.

Аргументът `board` представлява двумерна матрица (списък от списъци), където живите клетки са отбелязани с `1`, а празните – с `0`.

Функцията трябва да върне нова двумерна матрица, която представлява следващата итерация на играта.

Примери:

```
(define block (list (list 0 0 0 0)
                    (list 0 1 1 0)
                    (list 0 1 1 0)
                    (list 0 0 0 0)))
(define blinker (list (list 0 1 0)
                      (list 0 1 0)
                      (list 0 1 0)))
(define single-cell (list (list 0 0 0)
                          (list 0 1 0)
                          (list 0 0 0)))
```

```
(define last-man-standing (list (list 1 0 0)
                                (list 0 1 0)
                                (list 0 0 1)))
(game-of-life block) → '((0 0 0 0) (0 1 1 0)
                         (0 1 1 0) (0 0 0 0))
(game-of-life blinker) → '((0 0 0) (1 1 1) (0 0 0))
(game-of-life single-cell) → '((0 0 0) (0 0 0) (0 0 0))
(game-of-life last-man-standing) → '((0 0 0)
                                     (0 1 0)
                                     (0 0 0))
```