KMI/YPP1 Paradigmata programování 1
úkol č. 4 (zápočtový úkol)

1. Naprogramujte proceduru `euclid`, která vypočte největší společný dělitel dvou zadaných čísel pomocí Euklidova algoritmu. Proceduru implementujte
   (a) jako iterativní proceduru, kde je pro modifikaci lokálního prostředí využita speciální forma `define`,
   (b) jako iterativní proceduru pomocí pojmenovaného `let`.

   Příklady použití:

   ```
   > (euclid 9 24)
   3

   > (euclid 17 25)
   1
   ```

2. Pomocí `foldl` nebo `foldr` naprogramujte proceduru `count` zjišťující počet výskytů prvků v daném seznamu.

   Příklady použití:

   ```
   > (count '(3 1 3 2 1 2 3 3 3))
   ((3 . 5) (1 . 2) (2 . 2))

   > (count '(d b a c b b a))
   ((d . 1) (b . 3) (a . 2) (c . 1))
   ```

3. Vytvořte proceduru `histogram`, která počítá histogram daného obrázku. Uvažujte pouze osmibitovou barevnou hloubku, barvy jednotlivých pixelů jsou tedy vyjádřeny čísly 0 až 255. Obrázek je reprezentován pomocí seznamu, jehož jednotlivé prvky jsou opět seznamy určující hodnoty pixelů na jednotlivých řádcích. Např., obrázek daný hodnotami pixelů

   ```
     0  100    80
   255    0   255
     0  100   255
     0    0     0
   ```

   je reprezentován seznamem

   ```
   ((0 100 80) (255 0 255) (0 100 255) (0 0 0))
   ```

   Příklady použití:

   ```
   > (histogram '((0 100 80) (255 0 255) (0 100 255) (0 0 0)))
   ((0 . 6) (100 . 2) (80 . 1) (255 . 3))
   ```

4. Pomocí rekurze naprogramujte proceduru `map-index-pred`. Tuto proceduru znáte z předchozího úkolu. Přijímá tři parametry: predikát ◁*pred*?▷, procedura ◁f▷ a seznam ◁l▷ = (a_1 a_2 ... a_n). Procedura `map-index-pred` vrací seznam (b_1 b_2 ... b_n) stejné délky jako ◁l▷, jeho prvky b_i jsou $f(a_i)$ pro indexy splňující predikát ◁*pred*?▷ a a_i pro indexy nesplňující predikát ◁*pred*?▷. Např:

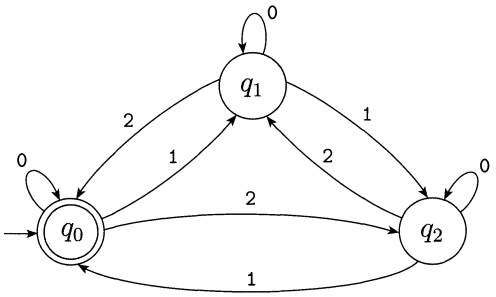
```
> (map-index-pred odd? sqr '(2 3 4 5))
(2 9 4 25)
```

Prvky seznamu na lichých indexech – tedy prvky 3 a 5 – jsou ve výsledném seznamu umocněny. Protože indexy prvků 2 a 4 jsou 0 a 2, tedy nejsou liché, jsou tyto prvky ve výsledném seznamu stejné jako v původním. Další příklad:

```
> (map-index-pred (lambda(i) (< i 2)) - '(1 2 3 4 5))
(-1 -2 3 4 5)
```

Prvky na indexech menší než 2 jsou ve výsledném seznamu nahrazeny jejich opačnou hodnotou, ostatní zůstávají stejné.

5. Napište iterativní proceduru `divided-by-three?`, která zjistí, jestli je součet čísel v daném seznamu (obsahující pouze čísla 0, 1 a 2) dělitelný třemi. K implementaci použijte tzv. *konečný automat*, jehož činnost je popsána orientovaným grafem znázorněným na následujícím obrázku:



Vrcholy označené jako q_0, q_1, q_2 představují stavy, ve kterých se automat může nacházet, hrany znázorňují do jakého stavu automat přejde zpracováním čísla určeného ohodnocením dané hrany. Činnost automatu je podrobněji popsána na následujícím příkladu:

```
> (divided-by-three? '(2 0 2 1 1))
#t
```

Stav automatu na začátku zpracování seznamu je vždy q_0 (vrchol označený šipkou, která vede „odnikud“). Automat může zpracovávat seznam čísel např. zleva doprava, tzn. začne číslem 2. Přejde tedy ze stavu q_0 do stavu q_2 (díky hraně, která směřuje z q_0 do q_2 a je ohodnocena číslem 2). Poté zpracovává další číslo v seznamu, tzn. číslo 0. Hrana ohodnocená nulou vycházející z vrcholu q_2 vede opět do vrcholu q_2, automat tedy zpracováním čísla 0 svůj stav nezmění. Dál je přečteno číslo 2, automat přejde do stavu q_1. Zpracováním čísla 1 přejde z q_1 zpět do q_2 a na závěr přečtením čísla 1 přejde do stavu q_0. Pokud bude stav automatu po přečtení všech čísel q_0[1], pak je součet všech čísel dělitelný třemi. Pokud bude jeho stav na konci zpracování jiný než q_0, pak součet není dělitelný třemi. Další příklad:

[1] Jedná se o tzv. koncový stav; typicky bývá označen dvojitým kroužkem. V tomto případě je počáteční a koncový stav automatu jeden a týž.

```
(divided-by-three? '(0 1 2 2 2 2 1))
#f
```

Automat po přečtení všech čísel skončí ve stavu q_1, tzn., že součet čísel 0+1+2+2+2+2+1 není dělitelný třemi.