PARADIGMATA PROGRAMOVÁNÍ I
Zkouška – písemná část

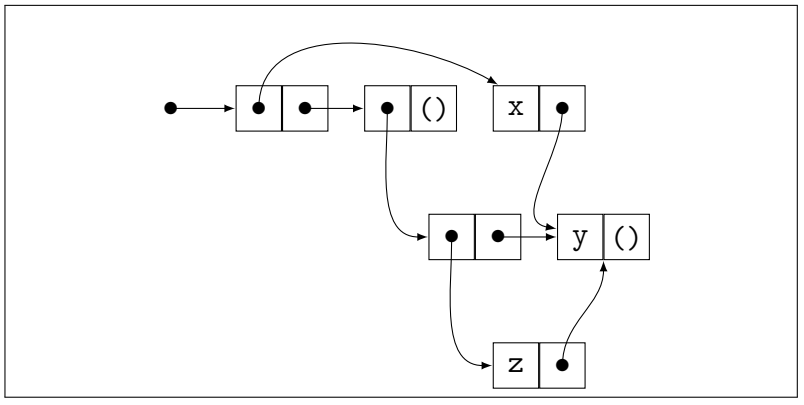
1. Bez použití `build-list` a `list-ref` napište *lineárně rekurzivní* proceduru `list-pref`, která pro daný seznam l a nezáporné celé číslo n vrací n-prvkový prefix seznamu l.

   ```
   (list-pref '(a b c d e) 3) ⟹ (a b c)
   (list-pref '(a b c d e) 2) ⟹ (a b)
   (list-pref '(a b c d e) 0) ⟹ ()
   ```

2. Napište *iterativní verzi* procedury `position-if`, která pro proceduru f a neprázdný seznam hodnot vrátí pozici nejlevější hodnoty ze seznamu, pro kterou je výsledek aplikace procedury f pravdivý (ve zobecněném smyslu). Pokud žádná taková hodnota v seznamu neexistuje, procedura vrací `#f`. Proceduru napište tak, aby během její činnosti nebyly vytvářeny žádné seznamy. Pokud budete potřebovat definovat pomocnou proceduru, vytvořte ji pomocí pojmenovaného `let`.

   ```
   (position-if even? '(1 3 5 7 9)) ⟹ #f
   (position-if even? '(1 3 6 8 9)) ⟹ 2
   ```

3. Napište jeden výraz, který po svém vyhodnocení vytvoří páry *s následující fyzickou strukturou*:



4. Napište, na co se vyhodnotí následující výraz a *zakreslete hierarchii prostředí,* která během vyhodnocení vzniknou. Lokální prostředí označte indexy vzestupně podle toho, v jakém pořadí vznikají.

```
((car (apply
       (let ((x (lambda () 10)))
 (lambda (y)
   (list x (x) y)))
       20 '())))
```

Otoč ▷▷

5. Následující kód vyjádřete ekvivalentně *bez kvazikvotování.*

```
((lambda (y) (y y `(10 ,+ 20)))
 (lambda (x y)
   (if (null? y)
       '()
       `(10 ,@(x x (cdr y)) ,y 20))))
```

6. Vysvětlete, co je *currying* a ukažte příklad jeho použití. Vysvětlete, proč by princip curryingu nešlo použít, pokud bychom uvažovali interpret jazyka Scheme používající dynamický rozsah platnosti symbolů.

7. Napište, *na co se vyhodnotí* následující výrazy.

```
;; a
((lambda (x) (x)) (lambda x x))

;; b
(list 10 + 20)

;; c
((lambda x x) 1 2 3)

;; d
(and (cond) (+))

;; e
(apply (lambda x 20) 'a 'b '(c))

;; f
(map map (list + -) '((1) (2)))

;; g
(foldl list '() '(a b c))

;; h
(let ((x -)) (eval `(,x 10)))

;; i
(map cons (list 'a 'b))

;; j
`(+ 2 '(+ ,(+ 3 4) 5))
```