Zkouška z PAPR1 ZS 2018/19
Ukázková písemka

1. Následující funkce je napsána v pseudokódu. Přepište ji jako proceduru Scheme:

   function Signum(a)
   if $a > 0$ **then**
   return 1
   else if $a < 0$ **then**
   return -1
   else
   return 0
   end if
   end function

2. Napište proceduru `sum-digits`, která sečte cifry zadaného kladného celého čísla. Příklad:

```
> (sum-digits 123)
6
```

   Můžete použít procedury `quotient` (celočíselné dělení) a `remainder` (zbytek po dělení):

```
> (quotient 123 10)
12
> (remainder 123 10)
3
```

3. Co bude výsledkem vyhodnocení následujícího výrazu ve výchozím prostředí? Popište postup vyhodnocení a jaká prostředí během něj vzniknou.

```
(let ((x 1)
      (y 2))
  ((lambda (x)
     (/ x y))
   y))
```

4. Napište proceduru `first-less`, která pro dvě dané posloupnosti vrátí nejmenší index, ve kterém je člen první posloupnosti menší než člen druhé. Posloupnosti reprezentujeme jako obvykle procedurou. Příklad:

```
> (first-less (lambda (n) (+ n 20)) sqr)
6
```

5. Napište proceduru `compose-sequences`, která pro dvě posloupnosti vrátí posloupnost prvků první posloupnosti o indexech daných druhou posloupností: má-li první posloupnost členy a_n a druhá b_n (vždy pro $n = 0, 1, 2, 3, \ldots$), pak výsledná posloupnost má členy a_{b_n}. Posloupnosti reprezentujeme jako obvykle procedurami. Příklad (s použitím procedury `display-sequence` z přednášky):

```
> (display-sequence sqr)
0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324,
361, ...
> (display-sequence (lambda (n) (* n 2)))
0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, ...
> (display-sequence (compose-sequences sqr (lambda (n) (* n 2))))
0, 4, 16, 36, 64, 100, 144, 196, 256, 324, 400, 484, 576, 676, 784, 900, 1024, 1156,
1296, 1444, ...
```

6. Navrhněte vhodnou datovou reprezentaci komplexních čísel a s její pomocí napište procedury pro součet a součin komplexních čísel.

7. Napište proceduru `each-nth`, která pro daný seznam a číslo $n > 0$ vrátí seznam obsahující každý n-tý prvek původního seznamu. Příklady:

```
> (each-nth '(1 2 3 4 5 6 7 8) 2)
(2 4 6 8)
> (each-nth '(1 2 3 4 5 6 7 8) 3)
(3 6)
```

8. Strom reprezentujeme jako pár, který má v *car* hodnotu kořenového uzlu a v *cdr* seznam větví. Každá větev je opět strom, reprezentovaný stejným způsobem. Napište proceduru `find-path`, která pro daný strom a hodnotu najde uzel s touto hodnotou a ve formě seznamu vrátí hodnoty uzlů mezi kořenem a nalezeným uzlem. V případě neúspěchu vrátí `#f`. Příklady:

```
> (define tree '(1 (2 (3) (4)) (5 (6 (7) (8)) (9))))
> (find-path tree 4)
(1 2 4)
> (find-path tree 8)
(1 5 6 8)
> (find-path tree 5)
(1 5)
> (find-path tree 10)
#f
```

9. Napište proceduru `sum-squares`, která vrátí součet druhých mocnin svých argumentů:

```
> (sum-squares 1 2 3)
14
> (sum-squares 4)
16
> (sum-squares)
0
```

10. Popište, jak byste do našeho interpretu Scheme doplnili speciální operátor `let`. Čím podrobnější a přesnější popis bude, tím lépe.