KMI/YPP1 Paradigmata programování 1
úkol č. 2

1. Vytvořte proceduru `koreny`, která bere jako argumenty koeficienty kvadratické rovnice a počítá její kořeny. Jestliže existují reálné kořeny, pak jsou vráceny jako prvky tečkového páru. Pokud neexistují, pak procedura vrátí `#f`.

   Příklady použití:

   ```
   > (koreny 1 3 2)
   (-1 . -2)

   > (koreny 1 2 1)
   (-1 . -1)

   > (koreny 2 2 1)
   #f
   ```

2. Vytvořte proceduru `2max`, která bere jako argumenty tři čísla a vrací pár `(`a `.` b`)`, kde a je maximální z těchto tří čísel a b je maximální ze zbývajících dvou.

   Příklady použití:

   ```
   > (2max 1 2 3)
   (3 . 2)

   > (2max 2 1 2)
   (2 . 2)

   > (2max 2 1 1)
   (2 . 1)

   > (2max 2 3 2)
   (3 . 2)
   ```

3. Vytvořte proceduru vyššího řádu `my-cons` realizující konstrukci páru. Tato procedura bude přijímat dva argumenty odpovídající prvkům vytvářeného páru. Aplikací vznikne procedura jednoho argumentu. Je-li tímto argumentem `#f`, bude vrácen druhý prvek páru, jinak bude vrácen první prvek páru. Dále vytvořte procedury `my-car` a `my-cdr`. Aplikací těchto procedur na pár vytvořený pomocí `my-cons` bude vrácen první, resp. druhý prvek prvek daného páru.

   Příklady použití:

   ```
   > (define p1 (my-cons 3 8))
   > (p1 #t)
   3
   ```

```
> (p1 #f)
8
> (my-car p1)
3
> (my-cdr p1)
8
```

4. Implementujte proceduru `switch`, která vymění prvky páru vytvořeného pomocí procedury `my-cons`.

   Příklady použití:

```
> (define p1 (my-cons 3 8))
> (define p2 (switch p1))
> (my-car p2)
8
> (my-cdr p2)
3
```

5. Implementujte vlastní aritmetiku komplexních čísel. Komplexní číslo $a + ib$ bude reprezentováno tečkovým párem `(a . b)`. Konstruktor `make-c` vytvoří pár reprezentující komplexní číslo, selektory `real` a `imag` vratí reálnou a imaginární část, procedura `conj` vrátí komplexně sdružené číslo, a procedury `c+`, `c-`, `c*`, `c/` vypočítají součet, rozdíl, součin a podíl dvou komplexních čísel.

   Příklady použití:

```
> (define c1 (make-c 1 2))
> (define c2 (make-c 3 -1))
> c1
(1 . 2)

> (real c1)
1

> (imag c1)
2

> (conj c1)
(1 . -2)

> (c+ c1 c2)
(4 . 1)

> (c* c1 c2)
(5 . 5)
```