PROGRAMIRANJE
2014/15

podatkovni tipi – terke in seznami

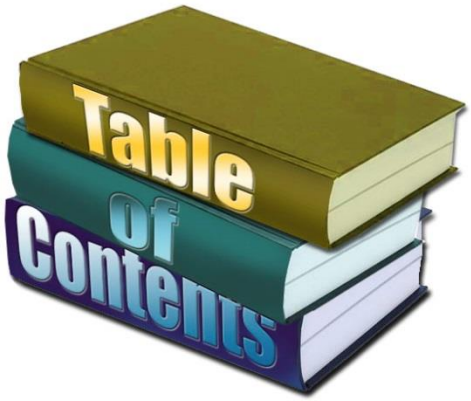
lokalna okolja

opcije

Ponovimo

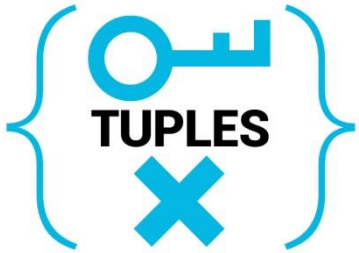
- prednosti funkcijskega programiranja
- vezave spremenljivk in funkcij
- statično in dinamično okolje
- enostavni izrazi (seštevanje, if-then-else)
- sintaktična in semantična evalvacija konstruktov programskega jezika
- podatkovni tipi:
  - int
  - bool
  - real
  - int * int -> int
- uporaba rekurzije

Pregled



- sestavljeni podatkovni tipi
  - terke (angl. tuples)
  - seznami (angl. lists)
- vezave v lokalnem okolju
  - namen
  - odstranitev odvečnih parametrov
  - optimizacija rekurzivnih klicev
- podatkovni tip „opcija“ (angl. *option*)

Terka (angl. *tuple*)



- podatkovni tip **nespremenljive** dolžine,
  sestavljen iz komponent **različnih** podatkovnih tipov

- **zapis terke**:

  ```
  (e1, e2, …, en)
  ```

  če je podatkovni tip `e1: t1, …, en: tn`,
  je terka tipa `t1 * t2 * … * tn`

- **dostop do elementov** terke `e`

  ```
  #n e
  ```

  kjer je `n` številka zaporedne komponente,
  `e` pa izraz-terka

Primeri uporabe terk

$$\texttt{int} \times \texttt{string} \xrightarrow{\texttt{f}} \texttt{int}$$

Napiši funkcije, s katerimi:

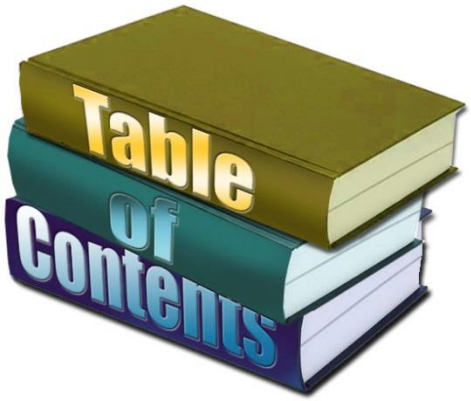
1. seštej števili, predstavljeni s terko (parom)
   ```
   fn : int * int -> int
   ```

2. obrni komponenti terke
   ```
   fn : int * int -> int * int
   ```

3. prepleti dve trimestni terki
   ```
   fn : (int * int * int) * (int * int * int)
   -> int * int * int * int * int * int
   ```

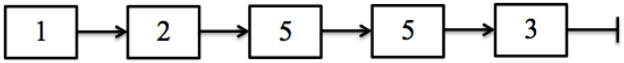
4. sortiraj komponenti terk po velikosti
   ```
   fn : int * int -> int * int
   ```

Pregled



- sestavljeni podatkovni tipi
  - terke (angl. tuples)
  - seznami (angl. lists)
- vezave v lokalnem okolju
  - namen
  - odstranitev odvečnih parametrov
  - optimizacija rekurzivnih klicev
- podatkovni tip „opcija“ (angl. *option*)

Seznam (angl. *list*)



- podatkovni tip **poljubne** dolžine, sestavljen iz komponent **enakih** podatkovnih tipov

- **zapis seznama s komponentami**:

  `[v1, v2, …, vn]`

  vsi elementi so istega podatkovnega tipa `t`

- **zapis seznama s sintakso** `glava::rep`

  če je glava vrednost `v0` in rep vrednost `[v1, v2, …, vn]`,
  ima zapis `glava::rep` vrednost `[v0, v1, …, vn]`
  pozor: glava je element, rep je seznam!

- podatkovni tipi seznama: `int list`, `real list`, `(int * bool) list`, `int list list`, …

Dostop do elementov seznama

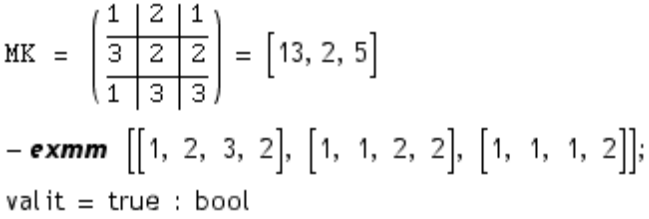
- **`null e`**
  vrne true, če je seznam prazen – `[ ]`
  `fn` `: 'a` **`list`** `->` **`bool`**

- **`hd e`**
  vrne glavo seznama (element)
  `fn` `: 'a` **`list`** `-> 'a`

- **`tl e`**
  vrne rep seznama (ki je seznam)
  `fn` `: 'a` **`list`** `->` **`'a list`**

  - `hd` in `tl` prožita izjemo (exception), če je seznam prazen

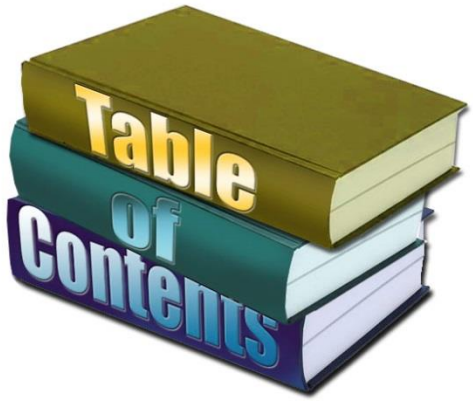
Primeri seznamov



Naloge:

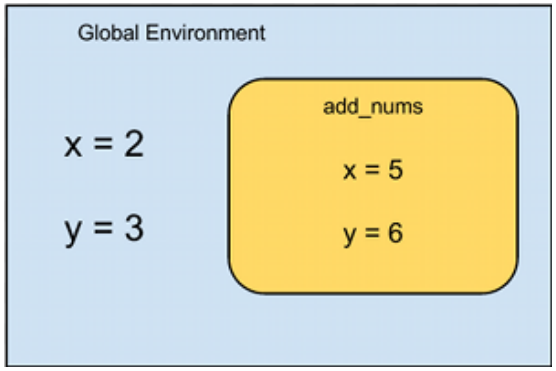
1. preštej število elementov v seznamu
   ```
   fn : int list -> int
   ```
2. izračunaj vsoto elementov v seznamu
   ```
   fn : int list -> int
   ```
3. vrni n-ti zaporedni element seznama
   ```
   fn : int list * int -> int
   ```
4. združi dva seznama
   ```
   fn : int list * int list -> int list
   ```
5. prepleti elemente obeh seznamov (do dolžine krajšega seznama)
   ```
   fn : int list * int list -> (int * int) list
   ```
6. izračunaj vsote elementov v terkah (parih števil) v seznamu
   ```
   fn : (int * int) list -> int list
   ```
7. filtriraj seznam predmetov glede na pozitivno oceno izpita
   ```
   fn : (string * int) list -> string list
   ```

Pregled



- sestavljeni podatkovni tipi
  - terke (angl. tuples)
  - seznami (angl. lists)
- vezave v lokalnem okolju
  - namen
  - odstranitev odvečnih parametrov
  - optimizacija rekurzivnih klicev
- podatkovni tip „opcija“ (angl. *option*)

Lokalno okolje



- funkcije uporabljajo globalno statično/dinamično okolje ➔ potrebujemo konstrukt za izvedbo **lokalnih vezav** v funkciji
  - lepše programiranje
  - potrebne so samo lokalno
  - zaščita pred spremembami izven lokalnega okolja
  - v določenih primerih: nujno za performanse (sledi…)!

- **izraz „let“:**
  - je samo izraz, torej je lahko vsebina funkcije
  - **sintaksa**: `let d1 d2 … dn in e end`
  - **preverjanje tipov**: preveri tip vezav `d1, …, dn` in telesa `e` v zunanjem statičnem okolju. Tip celega izraza let je tip izraza `e`.
  - **evalvacija**: evalviraj zaporedoma vse vezave in telo `e` v zunanjem okolju. Rezultat izraza let je rezultat evalvacije telesa `e`.

Lokalno okolje

- novost: uvedemo pojem dosega spremenljivke (angl. *scope*)
- v lokalnem okolju imamo lahko tudi vezave lokalnih funkcij

```
fun sestej(c: int) =
    let
        val a = 5
        val b = a+c+1
    in
        a+b+c
    end
```

```
fun povprecje(sez: int list) =
    let
        fun stevilo_el(sez: int list) =
            if null sez
            then 0
            else 1 + stevilo_el(tl sez)
        fun vsota_el(sez: int list) =
            if null sez
            then 0
            else hd sez + vsota_el(tl sez)
        val vsota = Real.fromInt(vsota_el(sez))
        val n = Real.fromInt(stevilo_el(sez))
    in
        vsota/n
    end
```

Lokalno okolje

- notranje funkcije lahko uporabljajo zunanje vezave, odvečne (podvojene) reference lahko torej odstranimo

```
fun sestej1N (n: int) =
    let
       fun sestejAB (a: int, b: int) =
           if a=b then a else a + sestejAB(a+1, b)
    in
       sestejAB(1, n)
    end
```

b je vedno enak n in se med rekurzijo ne spreminja!

```
fun sestej1N (n: int) =
    let
       fun sestejAB (a: int) =
           if a=n then a else a + sestejAB(a+1)
    in
       sestejAB(1)
    end
```

(Ne)učinkovitost rekurzije

- težave lahko nastopijo pri večkratnih rekurzivnih klicih

```
fun najvecji_el (sez : int list) =
    if null sez
    then 0  (* maksimum praznega seznama je 0? *)
    else if null (tl sez)
    then hd sez
    else if hd sez > najvecji_el(tl sez)
    then hd sez
    else najvecji_el(tl sez)
```

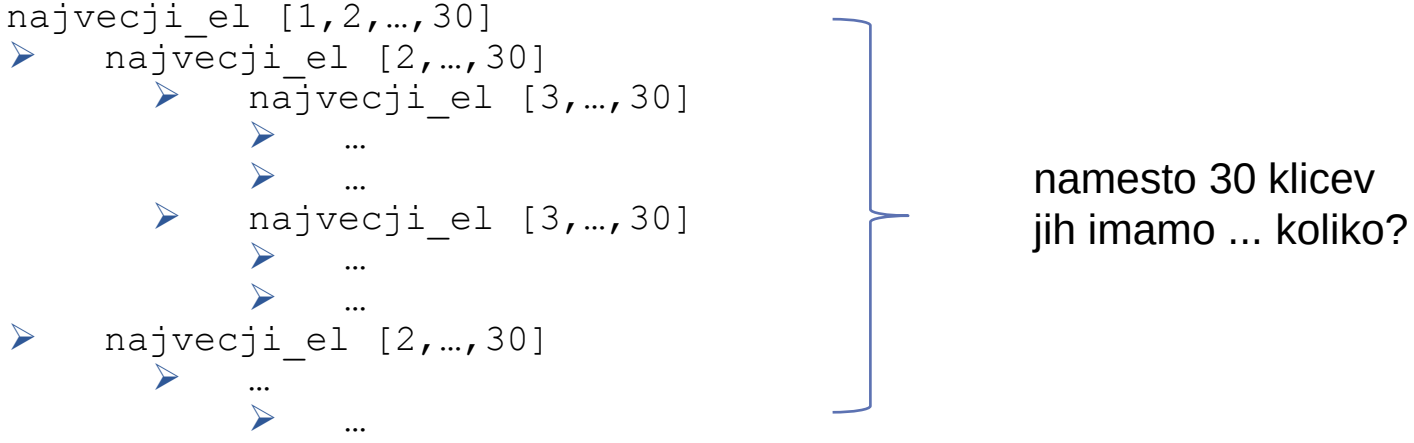
- (brez težav) izvedba v primeru klica:
  `najvecji_el [30,29,28,27,26,…,7,6,5,4,3,2,1]`
  Vedno se kliče samo prvi rekurzivni klic (glava je večja od maksimuma v repu), torej:

  `najvecji_el[30,…,1]` ➔ `najvecji_el[29,…,1]`
  ➔ `najvecji_el[28,…,1]` ➔ … ➔ `najvecji_el[1]` ➔ konec

Učinkovitost rekurzije

```
fun najvecji_el (sez : int list) =
    if null sez
    then 0  (* maksimum praznega seznama je 0? *)
    else if null (tl sez)
    then hd sez
    else if hd sez > najvecji_el(tl sez)
    then hd sez
    else najvecji_el(tl sez)
```

- Kaj pa izvedba v primeru klica:
  `najvecji_el [1,2,3,4,5,6,7,8,9,10,…,26,27,28,29,30]`
- Vedno se kličeta oba rekurzivna klica, torej:



Učinkovitost rekurzije

- rešitev: uporaba lokalne spremenljivke, ki hrani rezultat rekurzivnega klica

```
fun najvecji_el (sez : int list) =
    if null sez
    then 0
    else if null (tl sez)
    then hd sez
    else let val max_rep = najvecji_el (tl sez)
         in
              if hd sez > max_rep
              then hd sez
              else max_rep
         end
```


Ujemanje vzorcev