```
(*************************************************************)
(************** 1. REKURZIVNO UJEMANJE VZORCEV  **************)
(*************************************************************)


(* uporaba anonimne spremenljivke, kjer ne potrebujemo vrednosti *)
fun dolzina (sez:int list) =
    case sez of
       [] => 0
     | (* NAMESTO: glava::rep *) _::rep => 1 + dolzina rep


(*************************************************************)
(* 1. PRIMER *)
(* se¹tevanje dveh seznamov po elementih; seznama morata biti enako dolga *)

(* SLAB NAÈIN: *)
exception LengthProblem

fun sestej_seznama (sez1, sez2) =
    case sez1 of
        [] => (case sez2 of
                    [] => []
                  | glava::rep => raise LengthProblem)
      | glava1::rep1 => (case sez2 of
                           [] => raise LengthProblem
                         | glava2::rep2 => (glava1+glava2)::sestej_seznama(rep1,rep2))

(* BOLJ©I NAÈIN z gnezdenjem vzorcev*)
fun sestej_seznama2 seznama =
    case seznama of
        ([], []) => []
      | (glava1::rep1, glava2::rep2) => (glava1+glava2)::sestej_seznama(rep1,rep2)
      | _ => raise LengthProblem


(*************************************************************)
(* 2. PRIMER *)
fun check_fibonacci sez =
    case sez of
      (glava::(drugi::(tretji::rep))) => (tretji = (glava+drugi)) andalso check_fibonacci (drugi::
(tretji::rep))
      | _ => true


(*************************************************************)
(* 3. PRIMER *)
datatype sodost = S | L | N

fun sodost_sestevanje (a,b) =
  let
      fun sodost x = if x=0 then N
                     else if x mod 2 = 0 then S
                     else L
  in
      case (sodost a, sodost b) of
          (S,L) => L
        | (S,_) => S
        | (L,L) => S
        | (L,_) => L
        | (N,x) => x
  end




(*************************************************************)
(************** 2. SKLEPANJE NA PODATKOVNI TIP  **************)
(*************************************************************)

(* ne deluje, unknown flex record *)
```

```
(*
fun sestej1 stevili =
  #1 stevili + #2 stevili
*)

(* moramo opredeliti podatkovni tip *)
fun sestej2 (stevili:int*int) =
    #1 stevili + #2 stevili

(* sklepanje na tip deluje pri uporabi vzorcev *)
fun sestej3 (s1, s2) =
    s1 + s2


(*****************************************************************)
(* polimorfizem pri sklepanju na tip *)

(* ni polimorfna *)
fun vsota_el sez =
    case sez of
        [] => 0
      | glava::rep => glava + vsota_el rep

(* je polimorfna *)
fun zdruzi (sez1, sez2) =
    case sez1 of
        [] => sez2
      | glava::rep => glava::zdruzi(rep, sez2)

(* je polimorfna *)
fun sestej_zapis {prvi=a, drugi=b, tretji=c, cetrti=d, peti=e} =
    a+d



(*****************************************************************)
(************************ 3. IZJEME  ***************************)
(*****************************************************************)


(* prej - brez uporabe izjem *)
(*
fun glava sez =
    case sez of
        [] => 0 (* !!! kasneje: exception *)
     | prvi::ostali => prvi
*)

exception PrazenSeznam

fun glava sez =
    case sez of
        [] => raise PrazenSeznam
     | prvi::ostali => prvi


(*****************************************************************)
(* primer izjeme pri deljenju z 0 *)
exception DeljenjeZNic

fun deli1 (a1, a2) =
    if a2 = 0
    then raise DeljenjeZNic
    else a1 div a2

fun tabeliraj1 zacetna =
    deli1(zacetna,zacetna-5)::tabeliraj1(zacetna-1)
    handle DeljenjeZNic => [999]


(*****************************************************************)
(* ¹e bolj splo¹no: prenos izjeme v parametru *)
```

```
(* fn : int * int * exn -> int *)
fun deli2 (a1, a2, napaka) =
    if a2 = 0
    then raise (* SPREMEMBA *) napaka
    else a1 div a2

fun tabeliraj2 (zacetna, moja_napaka) =
    deli2(zacetna, zacetna-5, moja_napaka)::tabeliraj2(zacetna-1, moja_napaka)
    handle moja_napaka => [999]


(*****************************************************************)
(* izjema s parametrom *)
exception MatematicnaTezava of int*string

fun deli3 (a1, a2) =
    if a2 = 0
    then raise MatematicnaTezava(a1, "deljenje z 0")
    else a1 div a2

fun tabeliraj3 zacetna =
    Int.toString(deli3(zacetna,zacetna-5)) ^ "  " ^ tabeliraj3(zacetna-1)
    handle MatematicnaTezava(a1, a2) => a2 ^ " stevila " ^ Int.toString(a1)




(*****************************************************************)
(************ 4. REPNA REKURZIJA Z AKUMULATORJEM  **************)
(*****************************************************************)

fun potenca (x,y) =
    if y=0
    then 1
    else x * potenca(x, y-1)

(* prevedba v repno rekurzijo *)
fun potenca_repna (x,y) =
    let
        fun pomozna (x,y,acc) =
            if y=0
            then acc
            else pomozna(x, y-1, acc*x)
    in
        pomozna(x,y,1)
    end


(*****************************************************************)
(* PRIMERI *)
(* obrni elemente seznama *)
fun obrni sez =
   case sez of
       [] => []
     | x::rep => (obrni rep) @ [x]

fun obrni_repna sez =
    let
        fun pomozna(sez,acc) =
            case sez of
                [] => acc
              | x::rep => pomozna(rep, x::acc)
    in
        pomozna(sez,[])
    end

(*****************************************************************)
(* prestej pozitivne elemente *)
fun prestejpoz sez =
    case sez of
        [] => 0
      | g::rep => if g>=0
```

```
                then 1+ prestejpoz rep
                else prestejpoz rep

fun prestejpoz_repna sez =
    let
        fun pomozna (sez, acc) =
            case sez of
                [] => acc
              | g::rep => if g>=0
                          then pomozna(rep, acc+1)
                          else pomozna(rep, acc)
    in
        pomozna(sez,0)
    end
```

```
(*************************************************************)
(**************** 0. FUNKCIJE VIŠJEGA REDA  ******************)
(*************************************************************)


(* 1. ILUSTRATIVEN PRIMER *)

fun operacija1 x = x*x*x
fun operacija2 x = x + 1
fun operacija3 x = ~x

val zbirka_operacij = (operacija1, "lala", operacija3, 144)

fun izvedi1 podatek =
           (#1 zbirka_operacij) ((#3 zbirka_operacij) podatek)

fun izvedi2 (pod, funkcija) =
    funkcija (pod+100)



(*************************************************************)
(* 2. FUNKCIJE KOT ARGUMENTI FUNKCIJ *)
(* ponavljajoèa se programska koda: *)
fun zmnozi_nkrat (x,n) =
    if n=0
    then x
    else x * zmnozi_nkrat(x, n-1)

fun sestej_nkrat (x,n) =
    if n=0
    then x
    else x + sestej_nkrat(x, n-1)

fun rep_nti (sez,n) =
    if n=0
    then sez
    else tl (rep_nti(sez, n-1))


(* faktorizacija ponavljajoèih delov kode v splo¹no funkcijo *)
fun nkrat (f, x, n) =
    if n=0
    then x
    else f(x, nkrat(f, x, n-1))

fun pomnozi(x,y) = x*y
fun sestej(x,y) = x+y
fun rep(x,y) = tl y

fun zmnozi_nkrat_kratka (x,n) = nkrat(pomnozi, x, n)
fun sestej_nkrat_kratka (x,n) = nkrat(sestej, x, n)
fun rep_nti_kratka (x,n) = nkrat(rep, x, n)


(*************************************************************)
(* 3. FUNKCIJE, KI VRAÈAJO FUNKCIJE  *)

fun odloci x =
    if x>10
    then (let fun prva x = 2*x in prva end)
    else (let fun druga x = x div 2 in druga end)



(*************************************************************)
(* 4. ANONIMNE FUNKCIJE *)

fun zmnozi_nkrat_skoraj (x,n) =
    nkrat(let fun pomnozi (x,y) = x*y in pomnozi end,   x, n)
```

```
fun zmnozi_nkrat_mega (x,n) = nkrat(fn (x,y) => x*y, x, n)
fun sestej_nkrat_mega (x,n) = nkrat(fn(x,y) => x+y, x, n)
fun rep_nti_mega (x,n) = nkrat(fn(_,x)=>tl x, x, n)


(*************************************************************)
(* primer na seznamu - anon. fun. in izogib ovijanju funkcij v funkcije *)
fun prestej sez =
    case sez of
        [] => 0
      | glava::rep => 1 + prestej rep

fun sestej_sez sez =
    case sez of
        [] => 0
      | glava::rep => glava + sestej_sez rep

(* faktorizacija *)
fun predelaj_seznam (f, sez) =
    case sez of
        [] => 0
      | glava::rep => (f sez) + (predelaj_seznam (f,rep))

fun prestej_super sez = predelaj_seznam (fn x => 1, sez)
fun sestej_sez_super sez = predelaj_seznam(hd, sez)   (* hd namesto fn x => hd x !!! *)


(*************************************************************)
(* 5. MAP IN FILTER *)

fun map (f, sez) =
    case sez of
        [] => []
      | glava::rep => (f glava)::map(f, rep)

fun filter (f, sez) =
    case sez of
        [] => []
      | glava::rep => if (f glava)
                      then glava::filter(f, rep)
                      else filter(f, rep)


(* PRIMERI *)

(* preslikaj seznam seznamov v seznam glav vgnezdenih seznamov *)
fun nal1 sez = map(hd, sez)
(* preslikaj seznam seznamov v seznam dolŸin vgnezdenih seznamov *)
fun nal2 sez = map(prestej, sez)
(* preslikaj seznam seznamov v seznam samo tistih seznamov, katerih dolŸina je dalj¹a od 2 *)
fun nal3 sez = filter(fn x => (prestej x) >= 2, sez)
(* preslikaj seznam seznamov v seznam vsot samo lihih elementov vgnezdenih seznamov *)
fun nal4 sez =
    map(sestej_sez,
        map(
            fn el => filter(fn x => x mod 2 = 1, el),
            sez)
       )


(*************************************************************)
(******************** 1. MAP IN FILTER ***********************)
(*************************************************************)
fun map (f, sez) =
    case sez of
        [] => []
      | glava::rep => (f glava)::map(f, rep)

fun filter (f, sez) =
    case sez of
        [] => []
      | glava::rep => if (f glava)
```

```
                    then glava::filter(f, rep)
                    else filter(f, rep)

(* PRIMER: preslikaj seznam seznamov v seznam samo tistih seznamov, katerih dol▒ina je liha *)
fun lihi_podsez sez = filter(fn x => (List.length x) mod 2 = 1, sez)


(*************************************************************)
(************************ 2. FOLD ****************************)
(*************************************************************)

fun fold (f, acc, sez) =
    case sez of
        [] => acc
      | glava::rep => fold(f, f(acc, glava), rep)


fun f2 xs = fold ((fn (x,y) => x andalso y >= 0), true, xs)

(* PRIMER 1: vsota elementov *)
fun vsota_el sez = fold(fn (x,y) => x+y, 0, sez);

(* PRIMER 2: dol▒ina seznama *)
fun dolzina_sez sez = fold(fn (x,y) => x+1, 0, sez);

(* PRIMER 3: izberi zadnji element v seznamu *)
fun zadnji sez = fold (fn (x,y) => y, hd sez, sez)

(* PRIMER 4: skalarni produkt [a,b,c]*[d,e,f] = ab+be+cf *)
fun skalarni [v1, v2] =
    fold(fn (x,y) => x+y, 0, map(fn (e1,e2) => e1*e2, ListPair.zip(v1,v2)))
  | skalarni _ = raise Fail "napaèni argumenti";

(* PRIMER 5: izberi nti element v seznamu *)
fun nti (sez, n) =
    fold(fn(x,(y,z)) => z, ~1,
           filter(fn (x,y) => x=n,
                  ListPair.zip (List.tabulate (List.length sez, fn x => x+1),
                                sez)
                 )
          )



(*************************************************************)
(******************* 3. LEKSIKALNI DOSEG *********************)
(*************************************************************)

(*
(* 1. primer *)

val a = 1            (* a=1 *)
fun f1 x = x + a     (* fn: x => x+1 *)
val rez1 = f1 3      (* rez1 = 4 *)
val a = 5            (* a=5 *)
val rez2 = f1 3      (* rez2 = (fn: x => x+1) 3 = 4 *)

(* 2. primer *)
val c = 1            (* c=1 *)
fun f2 b = c + b     (* fn: b => b+1 *)
val c = 5            (* c=5 *)
val b = 2            (* b=2 *)
val rez = f2 (c+b)   (* rez = (fn: b => b+1) (5+2) = 7+1 = 8 *)

(* 3. primer *)
val u = 1                   (* u = 1 *)
fun f v =
    let
        val u = v + 1       (* u = v+1 *)
    in
        fn w => u + v + w   (* f = fn w => v+1 + v + w *)
    end
```

```
val u = 3                   (* u = 3 *)
val g = f 4                 (* g = (fn w => v+1 + v + w) 4  --> 4+1+4+w = 9+w *)
val v = 5                   (* v = 5 *)
val w = g 6                 (* w = (fn w => 9+w) 6 --> 15*)

*)

(*************************************************************)
(************** 4. PREDNOSTI LEKSIKALNEGA DOSEGA **************)
(*************************************************************)

(*
(* PREDNOST 1:
     - neodvisnost lokalnih spremenljivk od zunanjega okolja
     - neodvisnost funkcije od argumentov *)
(* spodnji funkciji sta enakovredni *)
fun fun1 y =
    let
        val x = 3
    in
        fn z => x + y + z
    end

fun fun2 y =
    let
        val q = 3
    in
        fn z => q + y + z
    end

val x = 42 (* ne igra nobene vloge *)
val a1 = (fun1 7) 4
val a2 = (fun2 7) 4



(* PREDNOST 2:
     - podatkovni tip lahko doloèimo pri deklaraciji funkcije *)
val x = 1
fun fun3 y =
    let val x = 3
    in fn z => x+y+z end   (* int -> int -> int *)
val x = false              (* NE VPLIVA NA PODATKOVNI TIP KASNEJ▒EGA KLICA! *)
val g = fun3 10            (* vrne fn, ki pri▒teje 13 *)
val z = g 11               (* 13 + 11 = 24 *)



(* PREDNOST 3:
     - ovojnica shrani ("zapeèe") interne podatke za klic funkcije *)
fun filter (f, sez) =
    case sez of
        [] => []
      | x::rep => if (f x)
                  then x::filter(f, rep)
                  else filter(f, rep)

fun vecjiOdX x = fn y => y > x
fun brezNegativnih sez = filter(vecjiOdX ~1, sez)
(* POZOR:
- x je neodvisen od x-a v funkciji filter; èe ne bi bil,
  bi primerjali elemente same s sabo (x, ki je argument predikata
  in x, ki nastopa kot glava v funkciji filter
- prvi argument v klicu filter() --- vecjiOdX ~1 --- je ovojnica,
  ki hrani shranjen interni x, ki je neodvisen od x v filter() *)

*)
```

```
(*************************************************************)
(********************** 1. CURRYING **************************)
(*************************************************************)

fun vmejah_terka (min,max,sez) =
    filter(fn x => x>=min andalso x<=max, sez)


fun vmejah_curry min =    (* razlièica, ki uporablja currying *)
    fn max =>
       fn sez =>
          filter(fn x => x>=min andalso x<=max, sez)

(* sintaktiène olep□ave *)
fun vmejah_lepse min max sez =
    filter(fn x => x>=min andalso x<=max, sez)
(*
vmejah_lepse 5 15 [1,5,3,43,12,3,4];
vmejah_curry 5 15 [1,5,3,43,12,3,4];
*)


(*************************************************************)
(******************** 2. DELNA APLIKACIJA ********************)
(*************************************************************)

(* PRIMER 1: vrne samo □tevila od 1 do 10 *)
val prva_desetica = vmejah_curry 1 10;

(* PRIMER 2: obrne vrstni red argumentov *)
fun vmejah2 sez min max = vmejah_lepse min max sez;
(* doloèi zgornjo mejo fiksnega seznama *)
val zgornja_meja = vmejah2 [1,5,2,6,3,7,4,8,5,9] 1;

(* PRIMER 3. primeri z uporabo map/filter/foldl *)
val povecaj = List.map (fn x => x + 1);
val samoPozitivni = List.filter (fn x => x > 0);
val vsiPozitivni = List.foldl (fn (x,y) => y andalso (x>0)) true;  (* pozor, vrstni red arg v fn! *)



(*************************************************************)
(********************** 3. MUTACIJA **************************)
(*************************************************************)

fun zdruzi_sez sez1 sez2 =
    case sez1 of
        [] => sez2
      | g::rep => g::(zdruzi_sez rep sez2)

val s1 = [1,2,3]
val s2 = [4,5]
val rezultat = zdruzi_sez s1 s2

(*
- val x = ref 15;
val x = ref 15 : int ref
- val y = ref 2;
val y = ref 2 : int ref
- (!x)+(!y);
(*************************************************************)
(****************** 4. DOLOÈANJE TIPOV  **********************)
(*************************************************************)

(* PRIMER 1 *)
fun fakt x =                (* 1.   fakt: 'a -> 'b *)   (* 3.  fakt : int -> __ *)  (* 6.  fakt: int -
> int *)
    if x = 0                (* 2.   x: 'a; 'a = int, zato da primerjava z 0 uspe *)
    then 1                  (* 4.   rezultat funkcije je 'b = int *)
    else x*(fakt (x-1))     (* 5.   mora biti skladno s 4; x: int, (fakt x): int, 'b = int *)
```

```
(* PRIMER 2 *)
fun f (q, w, e) =           (* 1.  f: 'a * 'b * 'c -> 'd *)
                            (* 3.  f: ('f * 'g) list * 'b * 'c -> 'd *)
                            (* 5.  f: ('f * 'g) list * bool list * 'c -> 'd *)
                            (* 8.  f: ('f * int) list * bool list * 'c -> int *)
    let val (x,y) = hd(q)   (* 2.  'a = 'e list;  'e = ('f * 'g);  'a = ('f * 'g) list *)
    in if hd w              (* 4.  'b = 'h list;  'h = bool;  'b = bool list *)
       then y mod 2         (* 6.  y: int; 'd = int *)
       else y*y             (* 7.  skladno s 6 velja y: int; 'd = int *)
    end


(* PRIMER 3 *)
(* fun compose (f,g) = fn x => f (g x)   *)
(* val koren_abs = compose (Math.sqrt, abs);
      je enakovredno kot
   val koren_abs2 = Math.sqrt o abs;  *)

fun compose1 (f,g) =   (* 1.  f: 'a -> 'b;  g: 'c -> 'd *;
                           compose: ('a -> 'b) * ('c -> 'd) -> 'e                        *)
                       (* 6.  compose: ('a -> 'b) * ('c -> 'a) -> ('c -> 'b)            *)
    fn x => f (g x)    (* 2.  x: 'c, 'e: 'c -> NEKAJ                                    *)
                       (* 3.                  g: 'c -> 'd;  g x: 'd                      *)
                       (* 4.                  f: 'a -> 'b;  f (g x) = 'b  --> velja 'd=='a! *)
                       (* 5.         'e: 'c -> 'b                                        *)



(*****************************************************************)
(****************** 5. OMEJITEV VREDNOSTI  ***********************)
(*****************************************************************)

(*
val sez = ref [];     (* sez je tipa 'a list ref *)
sez := !sez @ [5];    (* v seznam dodamo int *)
sez := !sez @ true;   (* pokvari pravilnost tipa seznama! *)
*)


(*
val sez = ref [];                  (* NE DELUJE: omejitev vrednosti *)
val xx = ref NONE;                 (* NE DELUJE: omejitev vrednosti *)

val xxx = ref []: int list ref;    (* RE©ITEV 1: opredelimo podatkovne tipe *)

val mojaf = map (fn x => x+1);     (* ni polimorfna, deluje *)
val mojaf1 = map (fn x => 1);      (* teŸava: polimorfizem + klic funkcije map *)
fun mojaf2 sez = map (fn x => 1) sez     (* RE©ITEV 2: ovijemo vrednost v funkcijo *)

- mojaf [1,2,3]
- mojaf1 [1,2,3]
*)
```

```
(*************************************************************)
(****************** 1. VZAJEMNA REKURZIJA  *******************)
(*************************************************************)

(* PRIMER 1: sodost in lihost stevil *)
fun sodo x =
    if x=0
    then true
    else liho (x-1)
and liho x =
    if x=0
    then false
    else sodo (x-1)


(* PRIMER 2: rekurzija v podatkovnih tipih *)
datatype zaporedje1 = A of zaporedje2 | Konec1
     and zaporedje2 = B of zaporedje1 | Konec2

(* A (B (A (B (A Konec2)))); *)

(* ideja za konèni avtomat, ki sprejema nize oblike [1,2,1,2,...] *)




(*************************************************************)
(********************** 2. MODULI  ***************************)
(*************************************************************)

(* PRIMER 1: Modul za delo z nizi *)

structure Nizi =
struct

val prazni_niz = ""

fun dolzina niz =
    String.size niz

fun prvacrka niz =
    hd (String.explode niz)

fun povprecnadolzina seznam_nizov =
    Real.fromInt (foldl (fn (x,y) => (String.size x)+y) 0 seznam_nizov)
    /
    Real.fromInt (foldl (fn (_,y) => y+1) 0 seznam_nizov)
end



(* PRIMER 2: Modul za delo s polinomi *)

(* podpisi *)
signature PolinomP1 =
sig
    datatype polinom = Nicla | Pol of (int * int) list
    val novipolinom : int list -> polinom
    val mnozi : polinom -> int -> polinom
    val izpisi : polinom -> string
end

signature PolinomP2 =
sig
    type polinom
    val novipolinom : int list -> polinom
    val izpisi : polinom -> string
end

signature PolinomP3 =
sig
    type polinom
```

```
    val Nicla : polinom
    val novipolinom : int list -> polinom
    val izpisi : polinom -> string
end


(* modul *)
structure Polinom :> PolinomP3 =
struct

datatype polinom = Pol of (int * int) list | Nicla;

fun novipolinom koef =
    let fun novi koef stopnja =
            case koef of
                [] => []
              | g::r => if g<>0
                        then (stopnja-1,g)::(novi r (stopnja-1))
                        else (novi r (stopnja-1))
    in
        Pol (novi koef (List.length koef))
    end

fun mnozi pol konst =
    case pol of
        Pol koef => if konst = 0
                    then Nicla
                    else Pol (map (fn (st,x) => (st,konst*x)) koef)
      | Nicla => Nicla

fun izpisi pol =
    case pol of
        Pol koef => let val v_nize = (map (fn (st,x) => (if st=0
                                                         then Int.toString(x)
                                                         else Int.toString(x) ^ "x^" ^ Int.toString
(st))) koef)
                    in foldl (fn (x,acc) => (acc ^ " + " ^ x))
                             (hd v_nize)
                             (tl v_nize)
                    end
      | Nicla =>  "0"

end


(*
- Polinom.mnozi (Polinom.novipolinom [7,6,0,0,0,4]) 2;
val it = Pol [(5,14),(4,12),(0,8)] : Polinom.polinom
-  Polinom.mnozi (Polinom.novipolinom [7,6,0,0,0,4]) 0;
val it = Nicla : Polinom.polinom
- Polinom.mnozi (Polinom.Nicla) 3;
val it = Nicla : Polinom.polinom
- Polinom.izpisi (Polinom.mnozi (Polinom.novipolinom [7,6,0,0,0,4]) 2);
val it = "14x^5 + 12x^4 + 8" : string
*)

(*
(* uporabnik kr¹i pravila uporabe *)
- Polinom.izpisi (Polinom.Pol [(3,1),(1,2),(16,0),(~5,3)]);
val it = "1x^3 + 2x^1 + 0x^16 + 3x^~5" : string
*)
```