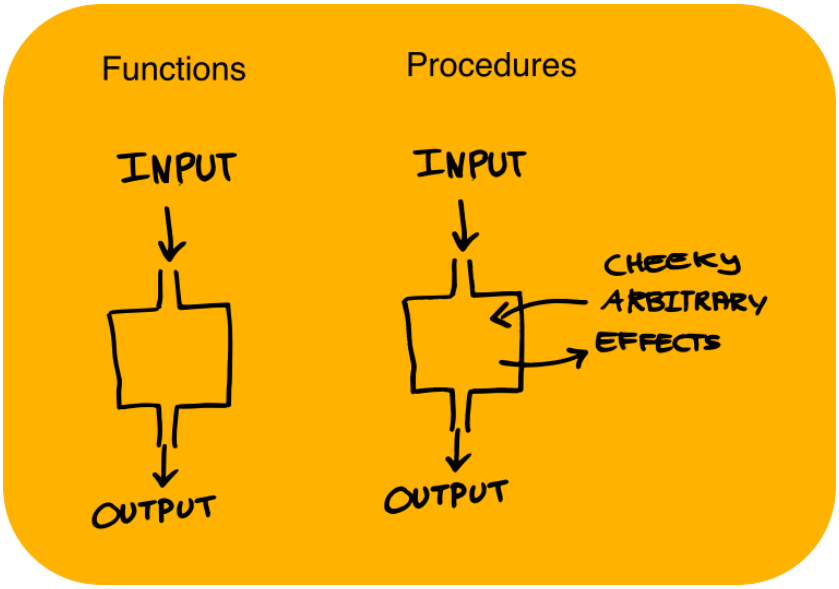
KONCEPTI PROGRAMSKIH JEZIKOV

IZTOK SAVNIK, MATJAŽ KRNC



Vaje za predmet Programiranje II

December 2018.

CIP – Kataložni zapis o publikaciji
Narodna in univerzitetna knjižnica, Ljubljana
ISBN 978-961-288-966-1 (pdf)
COBISS.SI-ID=297674496

Koncepti programskih jezikov
Vaje za predmet Programiranje II

Avtorja: Iztok Savnik, Matjaž Krnc

Samozaložba in oblikovanje: Matjaž Krnc

Fotografija na naslovnici: FPComplete Blog

Način dostopa (URL): `bit.ly/KPJ-vaje`

Koper, December 2018

Zbirka nalog pred vami predstavlja zbrano gradivo večletnega izvajanja predmeta *Programiranje II - Koncepti programskih jezikov* na Fakulteti za matematiko, naravoslovje in informacijske tehnologije, Univerza na Primorskem. Namen zbornika je bralcu ponuditi nekatere zanimive naloge, večinoma iz funkcijskega programiranja v slovenskem jeziku. Rešitve izbranih nalog se nahajajo v dodatku, na koncu zbirke. Morebitne napake prosim sporočite na `matjaz.krnc@upr.si`.

Za pomoč pri pisanju resitev se zahvaljujemo študentom: Andraž Cuderman, Davor Fon, Mateja Grižon, Nejc Razpotnik, Neja Skočir.

KAZALO

Del I

VAJE

1

LAMBDA RAČUN

Vaja 1.1. Napiši izraz v lambda računu, ki za dana parametra x in y izračuna povprečno vrednost x in y.

Apliciraj prej definiran lambda izraz na konkretnih parametrih in zapiši redukcijo izraza do vrednosti.

Vaja 1.2. Uporabi alfa konverzijo in beta redukcijo za ovrednotenje naslednjega stavka zapisanega z λ-računom

$$(\lambda a.\lambda b.a(ab))(\lambda a.a+1)\,1.$$

a) Izračunaj vrednost izraza.
b) Kaj naredi funkcija?

Vaja 1.3. Evaluiraj naslednje lambda izraze do vrednosti.
1) $(\lambda f.f(\lambda x.x))(\lambda x.x)$
2) $(\lambda x.\lambda y.x)zw$

Vaja 1.4. Dane imamo naslednje standardne kombinatorje lambda računa.

$$\begin{aligned} I &\equiv \lambda x.x; \\ K &\equiv \lambda x.\lambda y.x; \\ K^* &\equiv \lambda x.\lambda y.y; \\ S &\equiv \lambda x.\lambda y.\lambda z.xz(yz) \end{aligned}$$

Poenostavi naslednje izraze:

$$\begin{aligned} M &\equiv (\lambda x.\lambda y.\lambda z.zyx)aa(\lambda p.\lambda q.q); \\ M &\equiv (\lambda y.\lambda z.zy)((\lambda x.xxx)(\lambda x.xxx))(\lambda w.I); \\ M &\equiv SKSKSK \end{aligned}$$

Vaja 1.5. V naslednjih λ-izrazih prikaži vse oklepaje.

- $(\lambda x.xa)ax$
- $(\lambda z.zxz)(\lambda y.yx)z$

Vaja 1.6. Poišči vse proste (nevezane) spremenljivke v naslednjih λ-izrazih.

- $(\lambda b.xba)xb$
- $\lambda x.zy\lambda y.yx$

Vaja 1.7. Napiši naslednji izraz z čim manj oklepajev.

- $((xy)(\lambda y.(\lambda z.(z(\lambda y.(xy)))x)y))$

2

FUNKCIJSKI PROGRAMSKI JEZIKI

2.1 MATEMATIČNE FUNKCIJE

Vaja 2.1. Napiši funkcijo v ML, ki izračuna vsoto vrste:

$$1 + 4 + 9 + 16 + ... + n^2.$$

Vaja 2.2. Napiši funkcijo v ML, ki izračuna vsoto vrste

$$\sum_{x=0}^{n} 1/(2^x) = 1/1 + 1/2 + 1/4 + 1/8 + 1/16 + ... + 1/(2^n).$$

Preveri delovanje funkcije z implementacijo v Ocaml.

Vaja 2.3. V Ocaml napiši funkcijo, ki izračuna n-to Fibonaccijevo število definirano na sledeč način:

$$F_n = 1, \text{ če } n \in \{0, 1\}, \text{ ter } F_n = F_{n-1} + F_{n-2},$$

pri čemer zapis F_i predstavlja i-to Fibonaccijevo število.

Fibonaccijevo zaporedje: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ...

Vaja 2.4. Dana je OCaml funkcija naslednik (`let naslednik n = n+1`). Z uporabo funkcije naslednik in brez uporabe artimetičnih operacij naredi funkcijo `jeVsota(a:int, b:int, c:int)` , ki preveri ali je c vsota a in b (pri čemer velja $a, b \geq 0$).

Vaja 2.5. Funkcija `pfib: int*int -> int*int` je definirana na sledeč način:

$$\texttt{pfib}(i,j) = \begin{cases} (1,1); & i,j \leq 0; \\ \texttt{pfib}(i-1,0); & j = 0; \\ \texttt{pfib}(0,j-1); & i = 0; \\ \texttt{pfib}(i-1,j-1) + \texttt{pfib}(i-2,j-2); & else. \end{cases}$$

Operacija '+' je definirana nad pari na običajen način. Definiraj funkcijo `pfib` v Ocaml.

Vaja 2.6. *Ackermannova funkcija* je definirana z rekurzivnimi enačbami.

$$\begin{aligned} A(0,n) &= n+1 \\ A(m+1,0) &= A(m,1) \\ A(m+1,n+1) &= A(m,A(m+1,n)) \end{aligned}$$

Napiši funkcijo `A : int -> int -> int` , ki izračuna za dana parametra m in n vrednost zgoraj definirane rekurzivne funkcije.

Vaja 2.7. Dana je funkcija seštej : a' list -> int, ki sešteje elemente danega seznama.

```
# let rec sestej l = match l with
 | [] -> 0
 | a::r -> a+sestej r;;
val sestej : int list -> int = <fun>
# sestej [1;2;3];;
- : int = 6
```

Predstavi vsa stanja aktivacijskih zapisov ob klicu funkcije sestej [1;2;3].

2.2 SEZNAMI

Vaja 2.8. Napiši funkcijo sestej: `int list -> int`, ki sešteje elemente celoštevilskega seznama.

Primer:

```
# sestej [1;2;3;4;5];;
- : int = 15
```

Vaja 2.9. Napiši funkcijo `unija : int list -> int list -> int list`, ki za dana seznama celih števil vrne unijo. Pazi na duplikate!

Primer:

```
# unija [1;2;4;7] [2;4;7;9];;
- : int list = [1;2;4;7;9]
```

Vaja 2.10. Napiši funkcijo `zdruzi : int list -> int list -> int list`, ki sprejeme urejena seznama kot parametra in vrne urejen seznam, ki vsebuje elemente obeh vhodnih seznamov.

Primer:

```
# zdruzi [2;3;4] [5;7;9;10;13];;
- : int list = [2;3;4;5;7;9;10;13]
```

Vaja 2.11. Napiši funkcijo `zdruzi : int list -> int list -> int list`, ki združi dva seznama v tretji seznam tako, da vzame najprej en element iz prvega seznama potem dva elementa iz drugega seznama in tako naprej dokler ne pride do konca enega izmed vhodnih seznamov. Preostanek nepraznega seznama se da na konec novega seznama.

Primer:

```
# zdruzi [1;2;3] [5;6;7;8];;
- : int list = [1;5;6;2;7;8;3]
```

Vaja 2.12. Napiši funkcijo `vecjeod : int list -> int -> int list`, ki dobi seznam in število, vrne pa seznam, ki vsebuje samo elemente večje od podanega števila.

Primer:

```
# vecjeod [2;5;26;87;2;6]  5;;
- : int list = [26;87;6]
```

Vaja 2.13. Napišite funkcijo `seznamnm : int -> int list`, ki izpiše seznam števil od števila n do m. Velja $n \leq m$.

Primer:

```
# seznamnm 5 11;;
- : int list = [5;6;7;8;9;10;11]
```

Vaja 2.14. Napiši funkcijo `palindrom: int list -> bool`, ki preveri, če je podan seznam celih števil palindrom.

Primer:

```
# palindrom [1;2;3;2;1];;
- : bool = true
# palindrom [1;2;3];;
- : bool = false
```

Vaja 2.15. Napiši funkcijo `vsotaSodeLihe: int list -> int*int`, ki za dani seznam posebej sešteje soda in liha števila, ter vrne par, ki ima na prvem položaju vsoto lihih števil, na drugem pa vsoto sodih števil.

Primer:

```
# vsotaSodeLihe [1;1;1;2;4];;
- : int*int = (3,6)
```

Vaja 2.16. Napiši funkcijo `podseznam : int list -> int list -> bool`, ki preveri ali je seznam podan kot prvi parameter podseznam seznama podanega kot drugi parameter funkcije.

Primer:

```
# podseznam [1;2] [3;4;1;2];;
- : bool = true
# podseznam [1;2] [1;2;3];;
- : bool = true
# podseznam [1;2] [4;2];;
- : bool = false
```

Vaja 2.17. Dan imamo seznam znakov tipa `char list` v katerem se lahko pojavijo samo znaka `'a'` in `'b'`. Napiši funkcijo `cnta : char list -> int list`, ki pretvori sekvence znakov v seznam celih števil po naslednjih pravilih:

- `aaa -> 3`,
- `aa -> 2`,
- `a -> 1` in
- `x -> 0`, kjer je `x` poljuben znak.

Primer:

```
# cnta ['a','a','a','a','a','b','a','a'];;
- : int list = [3,2,0,2]
```

Vaja 2.18. Funkcijo za sortiranje seznama implementiraj na sledeč način:

1. Napiši pomožno funkcijo zamenjaj : `int list -> int list`, ki poišče v seznamu prvi zaporeden par `...::x::y::rep`, ki ni pravilno urejen: $x > y$. Funkcija naj vrne par sestavljen iz
   a) istega seznama, kjer sta x in y zamenjana ter
   b) true v primeru, da je bila zamenjava narejena in false sicer.
2. Napisi funkcijo `sortiraj : int list -> int list`, ki ponavlja izvajanje funkcije zamenjaj tako dolgo dokler ni seznam urejen.

Vaja 2.19. Dan je seznam, ki vsebuje znake tipa char. Napiši funkcijo `ace`, ki sprejme seznam znakov in vrne true v primeru, da seznam znakov vsebuje znake seznama ['a';'c';'e'] v danem vrstnem redu in false sicer.

```
# ace ['a';'b';'r';'a';'k';'a';'d';'a';'b';'r';'a
   '];;
- : bool = false
# ace ['a';'b';'e';'c';'e';'d';'a'];;
- : bool = true
# ace ['c';'e';'d';'r';'a'];;
- : bool = false
```

Vaja 2.20. Dan je seznam, ki vsebuje vrednosti 0 in 1. Napiši funkcijo, ki naredi naslednjo transformacijo na seznamu. Vse pojavitve vzorca 111 zamenja z vrednostjo 3, vse pojavitve vzorca 11 z vrednostjo 2 ter ohrani samostojne enice 1 in ničle 0.

```
1 1 1 -> 3
1 1 -> 2
1 -> 1
0 -> 0
```

Funkcija vedno poskuša najprej zamenjati daljši niz enic.

Na primer, seznam `[1;1;0;1;1;1;1;1;0;1;0]` se pretvori v seznam `[2;0;3;2;0;1;0]`.

Vaja 2.21. Napiši funkcijo cikli : int -> int -> int, ki za klic cikli m n generira seznam n ciklov števil od 0 do m-1.
Primer:

```
# cikli 3 4;;
- : int list = [0; 1; 2; 0; 1; 2; 0; 1; 2; 0; 1;
   2]
```

Vaja 2.22. Dano imamo sekvenco DNK v obliki seznama znakov tipa `dnk list`, kjer je tip `dnk` definiran kot

```
type dnk = A | C | T | G

```

Na primer, niz znakov "ACAAGT"je predstavljen s seznamom `[A;\-C;A;A;G;T]`.

Napiši funkcijo `najpodniz : dnk list -> int*int`, ki poišče najdaljši podniz istih znakov tipa `dnk` ter izpiše pozicijo prvega znaka in dolžino niza.

Vaja 2.23. Dano imamo sekvenco DNK v obliki seznama znakov tipa `dnk list`, kjer je tip `dnk` definiran kot v Nalogi 2.22.

Napiši funkcijo `prestej : dnk list -> dnk*int list`, ki sešteje enake zaporedne znake v sekvenci in za vsako skupino kreira par, ki vsebuje `dnk` znak in število pojavitev znaka.

Primer:

```
# prestej [A;C;A;A;G;C;C];;
val - : dnk*int list = [(A,1);(C,1);(A,2);(G,1)
 ;(C,2)]

```

Vaja 2.24. Napiši funkcijo `zamenjaj : int list -> int list`, ki v enem prehodu poišče v seznamu celih števil vse zaporedne pare `x::y`, ki niso urejeni po naraščajočem vrstnem redu (`x<=y`) in jih obrne.

Z uporabo funkcije `zamenjaj` implementiraj sortiranje seznama.

Vaja 2.25. Relacijo $\mathcal{R}$ predstavimo s seznamom parov celih števil. Napiši funkcijo

```
razsiri : int*int list -> int list -> int list,

```

kjer je prvi parameter seznam parov `r` in drugi parameter seznam celih števil `s`. Rezultat funkcije `razsiri` naj bo seznam, ki vsebuje vsa števila `x` za katera velja `(e,x)` $\in \mathcal{R}$ za nek element e seznama `s`.

Vaja 2.26. Napiši funkcijo

```
zdruzi : int list -> int list -> int list,
```

ki združi urejena seznama celih števil tako, da je rezultat urejen in hkrati izloči večkratne ponovite elementov.

Vaja 2.27. Dana je funkcija `fib3`, ki je definirana na sledeč način:

```
fib3(n) = 1, za n=1,2,3
fib3(n) = fib3(n-1)+fib3(n-2)+fib3(n-3), za n
 >3.
```

Napiši funkcijo `fib3list : int -> int list`, ki generira seznam vrednosti funkcije fib3 od 1 do n, kjer je $n > 0$ parameter funkcije. Primer:

```
# fib3 6;;
_ : int list = [1; 1; 1; 3; 5; 9]
```

Vaja 2.28. Napiši funkcijo v Ocaml, ki za dani seznam celih števil sešteje soda in liha števila, ter vrne par, ki ima na prvem mestu vsoto lihih števil, na drugem pa vsoto sodih števil.

Vaja 2.29. Seznam tipa (string*string) list predstavlja lete neke letalske družbe. Vsak par opiše direkten let med dvemi mesti. Napiši funkcijo

```
povezava : (string*string) -> (string*string)
   list -> bool,
```

ki preveri ali obstaja povezava med dvemi mesti podanimi s prvim parametrom. Povezava je lahko direktna ali z enim vmesnim postankom. Seznam letov letalske družbe je podan z drugim parametrom. Funkcija vrne true v primeru, da povezava obstaja in false sicer.

Dodatne točke: Naj povezava pomeni poljubno povezavo, ki je sestavljena iz poljubnega števila vmesnih postankov.

Vaja 2.30. Naloga je sestavljena iz dveh delov.

a) Definiraj tip plist, ki predstavi seznam parov. Prva komponenta para vsebuje ključe tipa int in druga komponenta vrednosti tipa string.

b) Napiši funkcijo

```
vapply : plist -> (string -> string) -> plist,
```

ki aplicira funkcijo tipa `string -> string` (drugi parameter) na vseh drugih komponentah parov, ki so podani s prvim parametrom funkcije.

Rezultat naj bo seznam parov kjer je druga komponenta vhodnih parov zamenjana z vrednostjo funkcije `string -> string`.

2.3 POLIMORFIZEM

Vaja 2.31. Napiši polimorfično funkcijo

```
zdruzi : 'a list -> 'a list -> ('a*'a->'a) -> 'a list,
```

ki zdruzi dva enako dolga seznama poljubnih objektov, tako da združi istoležne objekte seznamov. Par istoležnih objektov seznamov združimo z uporabo tretjega parametra funkcije zdruzi, funkcijo tipa `'a*'a->'a`.

Napiši primer uporabe funkcije zdruzi nad seznamoma celih števil. Združitev dveh števil implementiraj z običajno vsoto.

Vaja 2.32. Napiši polimorfično funkcijo

```
ostevilci : 'a list -> (int * 'a) list,

```

ki oštevilči elemente od 1 do n, če je n dolžina vhodnega seznama. Funkcija ostevilci naj vrne seznam parov, kjer je prva komponenta indeks in druga komponenta orginalna vrednost iz seznama.

Vaja 2.33. Tip ('a*'b) list opisuje seznam parov kjer je prva komponenta ključ tipa 'a in druga komponenta vrednost tipa 'b.

Napiši funkcijo

meet : ('a*'b) list -> ('a*'b) list -> ('a*('b*'b)) list,

ki združi dva seznama sortirana po ključu tipa 'a. Pari iz vhodnih seznamov se združijo samo v primeru, da se ujemajo v ključu. Rezultat funkcije so pari sestavljeni iz ključa in vrednosti, ki vsebuje obe vrednosti iz vhodnih parov.

Primer:

```
meet [(1,2);(2,3);(4,5);(4,9)] [(2,4);(4,6)] ->
     [(2,(3,4));(4,(5,6));(4,(9,6))]
```

Vaja 2.34. Tip ('a*'b) list opisuje seznam parov, kjer je prva komponenta para ključ tipa 'a in druga komponenta vrednost tipa 'b.

Predpostavimo, da so vhodni seznami sortirani po ključu tipa 'a. Napiši funkcijo

```
diff : ('a*'b) list -> ('a*'b) list -> ('a*'b)
   list,
```

ki izračuna razliko dveh seznamov. Rezultat vsebuje vse pare iz prvega seznama s ključi, ki se ne pojavijo v drugem seznamu.
Primer:

```
diff [(1,2);(2,3);(4,5)];(5,6)] [(2,4);(4,6)] ->
   [(1,2);(5,6)]
```

Vaja 2.35. Napiši parametrično funkcijo

```
podseznam : 'a list -> int*int -> 'a list,
```

ki iz danega seznama izlušči podseznam določen z drugim parametrom funkcije (z,k), kjer z predstavlja indeks začetka podseznama in k indeks konca podseznama.

Predpostavimo, da je $z < k$ in $k < l$, kjer je l dolžina vhodnega seznama.

Vaja 2.36. Dan je seznam parov, ki vsebujejo ključ tipa 'k in vrednost tipa 'v. Napiši funkcijo

```
filter ('v->bool) -> 'k*'v list -> 'k list,
```

ki iz seznama parov določenim z 2. parametrom izbere tiste kluče za katere vrne funkcija `('v->bool)` določena s 1. parameterom vrednost true. Rezultat funcije filter je seznam takšnih ključev.
Primer:

```
# filter (function x -> x=0) [(1,0);(2,1);(3,0)
   ;(4,1)];;
- : int list = [1; 3]
```

2.4 FUNKCIJE VIŠJEGA REDA

Vaja 2.37. Napiši funkcijo višjega reda

`urediPar : 'a*'a -> ('a*'a->bool) -> 'a*'a`,

ki za dan par vrednosti tipa 'a*'a vrne isti par vrednosti urejen po velikosti. Za določitev vrstnega reda komponent para napiši funkcijo `vecji : 'a*'a -> bool`, ki vrne `true`, če je prva komponenta večja od druge. Uporabi funkcijo večji kot parameter funkcije `urediPar`.

Vaja 2.38. Napiši funkcijo višjega reda

`izberi l f: 'a list -> ('a -> bool) -> 'a list`,

ki iz seznama l izbere samo tiste elemente a za katere funkcija f vrne true. Primer:

```
# let f a = if a>2 then true else false;;
val f : int -> bool = <fun>
# izberi [1;2;3;5] f;;
- : int list = [3; 5]
```

Vaja 2.39. Napiši funkcijo višjega reda

```
  skrci : (int * int -> int) -> list int -> int,

```

katere prvi parameter `f : int*int -> int` je funkcija, ki združi dve vrednost iz seznama v en samo vrednost tipa `int`. Drugi parameter `skrci` je seznam celih števil.

Naj bo funkcija `f` prvi parameter in seznam `l = [i1; i2; ...; in]` drugi parameter. Pomen funkcije `skrci` lahko zapišemo na naslednji način.

```
  skrci f l = f i1 (f i2 f(...(f in-1 in)))

```

Predpostavi, da ima seznam `l` vsaj dva elementa!

Vaja 2.40. Napiši funkcijo višjega reda

```
foldx : 'a list -> 'b -> ('a -> 'b -> 'b) list ->
    'b,
```

ki nad seznamom, ki je podan s prvim argumentom, in začetno vrednostjo podano z drugim argumentom, aplicira funkcije iz seznama podanega kot tretji argument na sledeč način.

Naj bo prvi argument enak `[a1,a2,...,an]`, drugi argument `b`, in tretji argument enak `[f1,f2,...,fn]`. Rezultat dobimo na sledeč način:

```
    f1 a1  (f2 a2 ... (fn  an b) ...)
```

2.5 REKURZIVNI TIPI

Vaja 2.41. Dan imamo seznam, definiran z rekurzivnim podatkovnim tipom izraz.

```
type izraz =
      Nil
    | Stevilo of int * izraz
    | Oper of char * izraz ;;
```

Izraz vsebuje aritmetične izraze, ki so lahko sestavljeni iz števil (Stevilo) in operacij (Oper). Dovoljene operacije so plus '+' in minus '-'. Predpostavljamo, da izrazi opisujejo pravilne aritmetične izraze.

Napiši funkcijo ovrednoti : izraz -> int, ki izračuna vrednost izraza. Primer:

```
    e = 10 + 5 - 3
```

Vaja 2.42. Binarno drevo je definirano s tipom

```
type bdrevo = List of int |
            Drevo of bdrevo * int * bdrevo,
```

ki ima vrednosti v listih definirane kot `List v`, kjer je v celo število. Notranja vozlišča so definirana kot trojica `Drevo (l, v, d)`, kjer sta l in d levo in desno poddrevo, v pa je vrednost.

(i) Napiši funkcijo `prestej : bdrevo -> int`, ki vrne vsoto vrednosti podanih v listih vhodnega drevesa.

(ii) Napiši funkcijo `oznaci : bdrevo -> bdrevo,` ki drugo komponento vseh notranjih vozlišč izhodnega drevesa zamenja z vsoto vseh listov poddrevesa.

Vaja 2.43. Binarno drevo je definirano s tipom itree, ki predstavlja binarno drevo katerega vozlišča vsebujejo vrednost tipa int.

```
 type itree = Nil | Node of itree*int*itree;;
```

Napiši funkcijo sumsub : itree -> itree, ki iz vhodnega drevesa konstruira novo drevo, ki namesto originalnih vrednosti v voliščih vsebuje vsoto vrednosti vozlišč levega in desnega pod-drevesa ter vrednosti danega vozlišča.
Primer:

```
# sumsub Node(Node(Nil,3,Nil),5,Node(Nil,2,Node(
    Nil,1,Nil)));;
_ : itree = Node(Node(Nil,3,Nil),11,Node(Nil,3,
    Node(Nil,1,Nil)))
```

Vaja 2.44. Dan je tip bdrevo, ki je definiran na naslednji način

```
type bdrevo = Leaf of int | Node of bdrevo * int
    * bdrevo;;
```

Napiši funkcijo bapply : bdrevo -> (int -> int) -> bdrevo, ki aplicira funkcijo (int -> int) na vseh vozliščih drevesa.

Vaja 2.45. Drevo je definirano z naslednjim tipom.

```
type bindrevo = List of int | Drevo of bindrevo *
    bindrevo ;;
```

Napiši funkcijo izpis : bindrevo -> int -> unit, ki izpiše vse liste katerih vrednost je večja od drugega parametra.

Vaja 2.46. Drevo int_tree je definirano na naslednji način.

```
type int_tree = { mutable key:int; mutable trees:
    int_tree list}
```

Napiši funkcijo višjega reda

```
tree_filter : int_tree -> int -> int list.
```

Funkcija vrne seznam vseh ključev iz drevesa podanega s prvim parametrim, ki so večji od vrednosti podane z drugim parametrom funkcije.

2.6 PARAMETRIZIRANI TIPI

Vaja 2.47. Dan imamo seznam, definiran z naslednjim tipom.

```
type 'a seznam =
      Nil
    | Vrednost of 'a * 'a seznam;;
```

Seznam je urejen v naraščajočem vrstnem redu glede na vrednost primerjalne funkcije primerjaj : 'a -> 'a -> int , ki vrne -1 če je prvi parameter večji od drugega, 0 če sta enaka in 1 v primeru, da je drugi parameter večji od prvega.

Napiši parametrično funkcijo:

```
dodaj : 'a seznam -> ('a -> 'a -> int) -> 'a
```

kjer je prvi parameter seznam v katerega dodajamo, drugi parameter je funkcija primerjaj in tretji parameter vrednost tipa 'a, ki jo dodajamo v seznam.

Vaja 2.48. Seznam definiramo z zapisom, ki ima dve komponenti: vrednost tipa `'a` in kazalec na naslednji element seznama tipa `'a seznam`, ki je bodisi prazen ali pa ima še vsaj en element.

```
type 'a element = {
   mutable vrednost:'a;
   mutable naslednji:'a seznam
}
and 'a seznam = Prazen | Element of 'a element
 ;;

```

Napiši funkcijo `podvoji : 'a seznam -> 'a seznam`, ki podvoji vse vrednosti v seznamu.

Vaja 2.49. Podatkovna struktura `seznam` je definirana na naslednji način.

```
type 'a element = {
   mutable vrednost: 'a;
   mutable naslednji: 'a seznam
}
and 'a seznam = Prazen | Element of 'a element
 ;;

```

Napiši funkcijo `zdruzi : 'a seznam -> 'a seznam -> 'a seznam`, ki ima za argumenta dva seznama tipa `'a seznam`.

(i) Funkcija naj združi seznama tako, da poveže konec prvega seznama z začetkom drugega ter kot rezultat vrne začetek prvega seznama.

(ii) Funkcija naj združi seznama tako, da naredi kopijo obeh seznamov v novem seznamu, ki vsebuje na novo konstruirane elemente.

Vaja 2.50. Dan je tip `grm`, ki je definiran na sledeč način:

```
type 'a grm =
    Nic
  | Ena of 'a * 'a grm
  | Dva of 'a grm * 'a * 'a grm;;
```

Napiši funkcijo `dolzinevej : 'a grm -> unit`, ki izpiše dolžine vej grma po principu levo-v-globino.

Vaja 2.51. Podan je tip `'a drevo` s katerim je predstavljeno binarno drevo.

```
type 'a drevo = {
    mutable levo:'a bin_drevo;
```

```
    mutable vozlisce:'a;
    mutable desno:'a bin_drevo
}
and 'a bin_drevo = Prazen | Vozlisce of 'a
 drevo ;;
```

Napiši funkcijo, ki izpiše vsa vozlišča tretjega nivoja drevesa, ki obstajajo v drevesu.

Vaja 2.52. Dano je drevo definirano z naslednjo podatkovno strukturo:

```
# type 'a tree = Empty | Node of 'a * 'a tree
 list ;;
```

Napiši funkcijo "prestej : 'a tree -> int", ki prešteje vozlišča drevesa.

Vaja 2.53. Dana je definicija parametričnega binarnega drevesa. Parameter tipa bindrevo je spremenljivka tipa 'a, ki predstavlja tip elementov v listih drevesa.

```
type 'a bindrevo = List of 'a | Drevo of 'a
 bindrevo * 'a bindrevo ;;
```

a) Napiši funkcijo izpis: 'a bindrevo -> ('a -> bool) -> unit, ki zpiše vse liste drevesa za katere vrne drugi parameter funkcije izpis – funkcija tipa 'a -> bool – vrednost true.

b) Napiši funkcijo obrni: 'a bindrevo -> 'a bindrevo, ki obrne vhodno drevo tako, da vsa leva poddrevesa zamenja z desnimi poddrevesi.

Vaja 2.54. Dano imamo splošno planarno drevo, ki je predstavljeno z naslednjim tipom:

```
type 'a tree = Empty
             | Node of 'a * 'a tree list ;;
```

Napiši funkcijo `filter :'a tree -> ('a -> bool) -> 'a tree`, ki pomeče ven iz drevesa vsa poddrevesa s korenom za katerega vrne funkcija tipa `'a -> bool`, podana kot drugi parameter, vrednost `false`. Rezultat je torej novo drevo brez izbranih poddreves.

Vaja 2.55. Dani so tip 'a izraz, ki predstavja aritmetične izraze nad vrednostmi tipa `'a`

```
type operacija = PLUS|MINUS;;
type 'a izraz = 'a | 'a izraz * operacija * 'a
 izraz;;
```

ter funkciji `plus :'a->'a->'a` in `minus :'a->'a->'a`, ki izračunata vsoto in razliko vrednosti tipa 'a.

Napiši funkcijo izracun : 'a izraz -> 'a, ki izračuna vrednost izraza tipa 'a izraz.
Primer:

```
# izracun (2,plus,(3,minus,1));;
- : int = 4
```

Vaja 2.56. Napiši polimorfično rekurzivno funkcijo obrni: 'a array -> 'a array, ki obrne vsebino polja.

Primer:

```
# obrni [|1;2;3|];;
- : int array = [| 3; 2; 1 |]
```

Vaja 2.57. S poljem želimo implementirati parametrično podatkovno strukturo minvrsta, ki hrani zadnjih n najmanjših vrednosti vstavljenih v podatkovno strukturo. Podatkovno strukturo minvrsta definiramo na sledeč način:

```
# type 'a minvrsta = 'a array;;
type minvrsta = 'a array
```

Z naslednjo funkcijo kreiramo primerek minvrsta.

```
# let kreiraj n v = Array.create n v;;
val kreiraj : int -> int array = <fun>
# let a : int minvrsta = kreiraj 10 0;;
val a : int minvrsta = [|0; 0; 0; 0; 0; 0; 0; 0;
   0; 0|]
```

Napiši parametrično funkcijo dodaj : 'a -> 'a minvrsta -> unit, ki doda novo vrednost v polje. Paziti je potrebno, da polje vedno vsebuje n najmanjših vrednosti.

Vaja 2.58. Dano je drevo, ki vsebuje dve vrsti elementov. Definirano je z naslednjo podatkovno strukturo:

```
  type ('a,'b) drevo =
     Prazno
   | Vozliscea of 'a * ('a,'b) drevo list;;
   | Vozlisceb of 'b * ('a,'b) drevo list;;
```

Napiši funkcijo razcepi : ('a,'b) drevo -> 'a list * 'b list, ki prepiše vse elemente Vozliscea v prvi seznam in vse elemente Vozlisceb v drugi seznam.

Vaja 2.59. Naloga je sestavljena iz dveh delov:

1. Definiraj parametrično podatkovno strukturo 'a splosnoDrevo, kjer spremenljivka tipa 'a predstavlja tip vrednosti vozlišča. Splošno drevo ima poljubno število poddreves.

2. Napiši funkcijo prestejOtroke : 'a splosnoDrevo -> int, ki izračuna povprečno število otrok za vsa vozlišča, ki niso listi drevesa.

Vaja 2.60. Definiraj parametriziran tip `'a seznam` z uporabo zapisov! Zapis naj ima dve komponenti: vrednost tipa `'a` in kazalec na naslednji zapis v seznamu oz. prazen seznam.

Napiši funkcijo `dolzina : 'a seznam -> int`, ki prešteje število vrednosti v seznamu.

Vaja 2.61. Definiraj parametrični tip `slika`, ki predstavlja dvo dimenzionalno računalniško sliko z neznanim tipom elementov (pik) polja, ki ga označimo z `'a`.

Napiši funkcijo višjega reda

```
zdruzi : 'a slika -> 'a slika -> ('a->'a->'a)
 -> 'a slika,

```

ki ima prva dva parametra sliki `s1` in `s2` ter tretji parameter funkcijo `f : 'a->'a->'a`, ki združi dve piki v eno piko. Funkcija `zdruzi` združi istoležne pike iz slik `s1` in `s2` z uporabo funkcije `f`.

Vaja 2.62. Dan je tip 'a struc definiran na naslednji način.

```
type 'a struc =
    Elm of 'a
  | Pair of 'a struc * 'a struc
  | Triple of 'a struc * 'a struc * 'a struc
```

Napiši polimorfično funkcijo

```
smap : ('a->'b) -> ('a struc) -> ('b struc),
```

ki aplicira funkcijo f določeno s 1. parametrom na vseh komponentah Elm x, ki so del 2. parametra. Rezultat naj bo struktura tipa 'b struc. Primer:

```
# smap (function x -> x+1)
      (Triple (Pair (Elm 1, Elm 2),Elm 3, Elm 4)
  );;
- : int struc = Triple (Pair (Elm 2, Elm 3), Elm
  4, Elm 5)
```

Vaja 2.63. Dano imamo tabelo funkcij definirano z naslednjim tipom

```
type 'a ftab = ('a->'a) array
```

Napiši funkcijo

```
tapply : int*'a list -> 'a ftab -> 'a list.
```

Naj ima funkcija `tapply l t` parameter l, ki je seznam parov oblike (i,a) in parameter t, ki je tabela funkcij. Funkcija `tapply` pretvori l v seznam vrednosti, ki jih dobimo z aplikacijo funkcij t.(i) na a.

Vaja 2.64. Definiraj parametrični tip slovar, ki vsebuje seznam parov. Prvi element parov naj vsebuje ključ tipa int in drugi element vsebuje vrednost poljubnega tipa.

Napiši funkcijo

```
preberi : slovar -> int -> 'a,
```

kjer 'a predstavlja tip druge komponente para v slovarju. Funkcija preberi vrne za dan ključ (2. parameter) iz slovarja (1. parameter) pripadajočo vrednost tipa 'a.

Vaja 2.65. Dan je tip drevo, ki je definiran na sledeč način:

```
# type 'a drevo =
  List
| Veja of  'a * 'a drevo
| Rogovila of 'a drevo * 'a * 'a drevo;;
```

Napiši funkcijo `dolzinevej : 'a drevo -> int list` , ki izpiše seznam globin listov drevesa po principu levo-v-globino. Pri tem velja, da je top vozlišče na nivoju 0.

Primer uporabe:

```
# dolzinevej (Rogovila (Veja (4,List),6,
              Rogovila (List,7,Veja (8,List))));;
int list = [2;2;3]
```

Vaja 2.66. Definiraj parametrični tip `key\_value` , ki predstavlja zapis z dvema komponentama, prva komponenta je ključ tipa 'a in druga komponenta je vrednost tipa 'b.

Na osnovi tipa `key\_value` definiraj parametrični tip slovar, ki je implementiran s poljem!

Dano imamo polimorfično funkcijo equal : 'a -> 'a -> bool, ki vrne true v primeru, da je prvi parameter enak drugemu in false sicer.

Napiši funkcijo duplikati : slovar -> slovar, ki iz slovarja odstrani vse duplikate.

Vaja 2.67. Seznam imamo definiran na naslednji način.

```
# type 'a rnode = { mutable cont:'a; mutable next
   :'a rlist }
   and 'a rlist = Nil | Elm of 'a rnode;;
```

Napiši funkcijo

```
p filter : 'a rlist -> ('a -> 'a -> bool) -> 'a
   rlist,
```

ki vrne elemente seznama (1. parameter) za katere funkcija podana z 2. parametrom vrne vrednost true. Funkcijo filter napiši tako, da ohraniš kopije elementov, oz. tako, da se ne kreira nov seznam!

Vaja 2.68. Dano je drevo, ki vsebuje dve vrsti elementov in je definirano z naslednjo podatkovno strukturo:

```
type ('a, 'b) tree =
  Nil
| Nodea of 'a * ('a, 'b) tree list
| Nodeb of 'b * ('a, 'b) tree list;;
```

Napiši funkcijo

```
razcepi: ('a,'b) drevo -> 'a list * 'b list,
```

ki prepiše vse elemente Nodea v prvi seznam, ki postane prvi element vrnjenega para, in vse elemente Nodeb v drugi seznam, ki postane drugi element vrnjenega para.

Vaja 2.69. Splošno drevo 'a tree je definirano na naslednji način.

```
type 'a tree = { mutable key:'a; mutable trees: '
   a tree }
```

Napiši funkcijo tree_apply : 'a tree -> ('a -> 'b) -> 'b tree. Ključi vozlišč vhodnega drevesa se zamenjajo z vrednostmi funkcije (definirane z drugim parametrom) aplicirane na ključu vozlišča.

Vaja 2.70. Naloga je sestavljena iz dveh delov:

- Definiraj parametrični tip ('a,'b) key_val, ki predstavlja zapis z dvema imenovanima komponentama:
  - komponente key tipa 'a in
  - komponente value tipa 'b.
- Napiši polimorfično funkcijo višjega reda

  ```
  array_filter : ('a,'b) key_val array -> ('a
     -> bool) ->  ('a,'b) key_val array
  ```

  ki iz polja podanega s prvim parametrom prepiše v končno polje samo tiste zapise s ključem tipa 'a, za katere funkcija podana z drugim parametrom vrne true.

2.7 UNIJE

Vaja 2.71. Dan imamo tip

```
type geo_objekt = Tocka | Premica | Krog |
 Trikotnik
```

in seznam geometrijskih objektov, npr.

```
let gl: geo_objekt list =  [ Tocka; Tocka;
 Premica; Krog; Krog; ... ]
```

Napiši funkcijo, ki prešteje število pojavitev posameznega geometrijskega objekta v seznamu:

```
val prestej: geo_objekt list ->
                geo_objekt -> int = <fun>
```

Vaja 2.72. Definiraj podatkovne strukture v Ocaml, ki predstavijo karte Briškole ali neke druge igre s kartami, ki jo dobro poznaš.

Vaja 2.73. Seznam vrednosti in operacij je definiran s tipom formula:

```
# type oper = PLUS | MINUS;;
type oper = PLUS | MINUS
# type elm = LP | RP | Vr of int | Op of oper;;
type elm = LP | RP | Vr of int | Op of oper
# type formula = Nil | Form of elm * formula;;
type formula = Nil | Form of elm * formula
```

Napiši funkcijo izpis : formula -> unit, ki izpiše formulo. Predpostavljamo, da je formula podana v pravilni obliki.

Vaja 2.74. Seznam vrednosti in operacij je definiran s tipom formula:

```
# type oper = PLUS | MINUS;;
type oper = PLUS | MINUS
# type elm = Vr of int | Op of oper;;
type elm = Vr of int | Op of oper
# type formula = Nil | Form of elm * formula;;
type formula = Nil | Form of elm * formula
```

Primer:

```
# let s = Form(Vr(1),Form(Op(PLUS),Form(Vr(5),
   Form(Op(MINUS),Form(Vr(3),Nil)))));;
val s : formula = Form (Vr 1, Form (Op PLUS, Form
    (Vr 5, Form (Op MINUS, Form (Vr 3, Nil)))))
```

Vrednost formule s predstavlja izraz $1 + 5 - 3$.

Napiši funkcijo izracunaj : formula -> int, ki izračuna vrednost seznama. Predpostavljamo, da je formula podana v pravilni obliki.

Vaja 2.75. Definiraj podatkovni tip `simplKarta` s katerim predstavimo enostavne igralne karte.

1. Imamo štiri vrste kart: kraljica, kralj, fant in punca.
2. Karte imajo štiri barve: srce, kara, pik in križ.
3. Pri definiciji tipa `simplKarta` uporabi unijo!

Definiraj seznam kart tipa simplKarta, ki vsebuje naslednje karte: srčevo punco, križevega kralja in pikovega fanta.

Vaja 2.76. Naloga je sestavljena iz dveh delov:
a) Z uporabo unije definiraj tip 'a element, ki predstavi elemente naslednjih oblik:

```
1. elemente tipa 'a ali
2. sezname elementov tipa 'a element.
```

Primer elementa:

```
val a : int element = L [E 1; E 2; L [E 3; E 4]]
```

b) Napiši funkcijo print : 'a element -> unit, ki izpiše elemente po pravilu najprej-levo-v-globino.

Vaja 2.77. Izrazi zelo enostavnega jezika `TP` so sestavljeni iz vrednosti med katerimi so postavljene operaciji `TIMES` ali `PLUS`. Obe operaciji sta levo asociativne vendar ima operacija `TIMES` višjo prioriteto od `PLUS`. Primer izraza je:

```
1 PLUS 2 TIMES 3 TIMES 4,
```

kar ustreza aritmetičnem izrazu $1 + ((2 * 3) * 4)$. Izraze jezika TP lahko definiramo z naslednjimi tipi:

```
# type operation = PLUS | TIMES;;
type operation = PLUS | TIMES
# type element = Val of int | Op of operation;;
type element = Val of int | Op of operation
# type expr = list element;;
type expr = list element
```

(i) Napiši funkcijo `check : expr -> bool`, ki preveri ali je izraz pravilno napisan.

(ii) Napiši funkcijo `calc : expr -> int`, ki izračuna vrednost izraza.

Vaja 2.78. Izrazi enostavnega jezika z imenom TP vsebujejo cela števila ter operaciji PLUS in TIMES. Predpostavimo, da imata operaciji isto prioriteto. Naslednji izraz

```
1 PLUS 2 TIMES 3 TIMES 4,
```

ustreza aritmetičnem izrazu ((1 + 2) * 3) * 4). Izrazi jezika TP so definirani z naslednjimi tipi.

```
# type operation = PLUS | TIMES;;
type operation = PLUS | TIMES
# type element = Val of int | Op of operation;;
type element = Val of int | Op of operation
# type expr = list element;;
type expr = list element
```

Napiši funkcijo calc : expr -> int, ki izračuna vrednost danega izraza.

Vaja 2.79. Boolovi izrazi so predstavljeni z naslednjim rekurzivnim tipom.

```
type bool_exp =
   | Val of bool
   | Not of bool_exp
   | And of bool_exp * bool_exp
   | Or of bool_exp * bool_exp;
```

Napiši funkcijo

```
eval : bool_exp -> bool,
```

ki evaluira boolov izraz v vrednost.

Dodatna naloga: Napiši funkcijo, ki izpiše pravilnostno tabelo za dan boolov izraz.

Vaja 2.80. Tip text definira predstavitev teksta v urejevalniku besedil. Primerek tipa text je sestavljen iz vrstic, ki so sestavljene iz besed.

```
type text = Eot | Line of line * text
and line = Eol | Word of string * line
```

Napiši funkcijo search : text -> line -> bool, ki vrne true, če je sekvenca besed definirana z drugim parametrom pod-sekvenca v tekstu, ki je podan kot prvi parameter. Predostavimo, da je iskana sekvenca vedno v eni vrstici.

3 IMPERATIVNI PROGRAMSKI JEZIKI

3.1 NIZI

Vaja 3.1. V programskem jeziku Ocaml napišite funkcijo

```
obrniBesede: string -> string
```

ki izpiše vse besede vhodnega niza v obratnem vrstnem redu! Primer:

```
# obrniBesede "banana je lepa";;
- : string = "ananab ej apel"
```

Vaja 3.2. Napiši funkcijo v Ocaml, ki preveri ali je nek niz podniz nekega drugega niza. Pri tem ni nujno, da znaki drugega niza v prvem stoje zaporedoma, ujemati se mora le vrstni red.

Dva primera:

```
MATI  MATEMATIKA     |    SENO  SOSEDNOST
      MAT     I      |          S   E NO
      MA     TI      |            SE NO
      M     ATI      |
           MATI      |
```

Dodatna naloga: napiši funkcijo, ki vrne število vseh takšnih podnizov v danem nizu.

Vaja 3.3. Dan je niz znakov. Vaša naloga je izdelati metodo, ki za vhodni niz izpiše vse podnize danega niza.

Podpis funkcije:

```
podnizi: string -> unit
```

Oglejmo si primer niza "miza":

```
"miza"; "miz"; "mi"; "m"; "iza"; "iz"; "i"; "za";
    "z"; "a"
```

Iz primera lahko razberete enega od možnih algoritmov.

Vaja 3.4. Napiši funkcijo v ML, ki za dan vhodni niz znakov vrne true samo v primeru, da niz vsebuje vzorec "a+b+"t.j. enemu ali več znakov `a` sledi eden ali več znakov `b`.

Primer pravilnega niza: `"uzgaaabbbbdvcg"`

3.2 POLJA IN MATRIKE

Vaja 3.5. Napiši funkcijo v programskem jeziku ML, ki vrne število enic v polju poljubne dolžine sestavljeno iz ničel in enic. Funkcija naj ima naslednjo signaturo: prestejEnice: int array -> int.

Vaja 3.6. Kreiraj dve celoštevilski polji a in b velikosti 5 in definiraj svojo vsebino polj. Napiši funkcijo produkt, ki izračna novo polje velikosti 5, katerega vsebina so produkti istoležnih komponent a in b.

Primer:

```
# let a = [|1;2;3;2;1|];;
- : int array = [| 1; 2; 3; 2; 1 |]
# let b = [|2;3;1;2;3|];;
- : int array = [| 2; 3; 1; 2; 3 |]
# produkt a b;;
- : [| 2; 6; 3; 4; 3 |]
```

Vaja 3.7. Zaslon mobilnega telefona ima dimenzijo 500x700 barvnih točk. Barve ene točke predstavimo z zapisom, ki ima tri komponente tipa int. Komponente točke predstavljajo RGB zapis: intenzivnost rdeče, zelene in modre barve.

Napisati moramo funkcijo, ki na zaslonu prikaže delujoče stanje mobilnega telefona med zagonom, izbiro operaterja, itd. Funkcija naj premika piko velikosti 3x3 po sredini zaslona od leve proti desni.

1. Definiraj podatkovne strukture v Ocaml s katerimi predstavimo zaslon mobilnega telefona.

2. Napiši funkcijo "delam", ki iterativno premika piko velikosti 3x3 po sredini zaslona od leve proti desni. Ko pika pride na rob desne strani se spet pojavi na robu leve strani.

Vaja 3.8. Dana je matrika edit: char array, ki vsebuje tekst urejevalnika besedil. Predpostavljamo, da je tekst formatiran: besede so ločene z enim samim presledkom in ni drugih kontrolnih znakov (npr. LF, CR,...).

Definiraj funkcijo preštej, ki izračuna za vsako posamezno besedo iz polja edit število znakov v predponi besede, ki se ujemajo z nizom `zanka`.

Število besed, ki se ujemajo v 0, 1, 2, ... znakih shranimo v polje rezultat: int array in število ujemanj uporabimo za indeks polja.

Predpostavi, da sta obe polji že definirani.

Primer:

```
# prestej [|'z';'a';'n';'k';'a';' ';'j';'e';' ';
          'z';'a';'d';'n';'j';'i';'\v c';' ';'b
  ';'i';
```

```
          'l';'a';' ';'z';'a';'n';'i';'\v c'|];;
- : unit = ()
# rezultat;;
- : int array = [|2; 0; 1; 1; 0; 1|]
```

Vaja 3.9. Dano je dvodimenzionalno polje tipa int array array, ki predstavlja črno-belo sliko in posamezen element predstavlja intenziteto ene pike. Slika je predstavljena s tipom slika.

```
type slika = { x: int; y: int; p: int array array
   };;
```

Vzorci so manjše slike (tipa slika), ki jih lahko iščemo v kompletnih slikah. Vzorec se ujema z delom slike na lokaciji (i,j), če se intenzitete vseh pik vzorca ujemajo s delom slike pokritim z vzorcem.

a) Napiši funkcijo ujemanje : slika -> slika -> int*int, ki za dano sliko (prvi argument) poišče pojavitev vzorca (drugi argument) v sliki.

b) Kako bi poiskali vsa ujemanja vzorca s sliko? Opiši v kontekstu rešitve naloge a).

c) Dodatna naloga: Podobnost med dvema vzorcema je definirana na osnovi podobnosti posameznih pik. Če se dve pike razlikujeta manj kot je vrednost konstante Toleranca, potem so pike enake. Kako bi razširili metodo tako, da bi iskala podobne vzorce?

Vaja 3.10. Slika računalniške naprave je definirana z naslednjim tipom

```
type slika = int array array,
```

kjer so pike, ki sestavljajo sliko predstaljene z 0 ali 1. Pika je osvetljena, če ima vrednost 1.

Napiši funkcijo

```
xor : slika -> slika -> slika,
```

ki dve slike kombinira tako, da naredi operacijo `xor` nad istoležnimi pikami.

Predpostavimo, da so vhodne slike istih dimenzij.

Vaja 3.11. Urejevalnik besedil ima tekst predstavljen z dvo dimenzionalno matriko znakov tipa `tekst`.

```
type tekst = char array array;;
```

Napiši funkcijo `poisci : tekst -> string -> int*int list`, ki vrne seznam koordinat kjer se začne v tekstu iskani niz podan kot drugi parameter.

Vaja 3.12. Urejevalnik besedil ima tekst predstavljen z dvo-dimenzionalno matriko znakov tipa `tekst`.

```
type tekst = char array array;;

```

Napiši funkcijo `poisci\_vertikalno : tekst -> string -> int*int list`, ki vrne seznam koordinat kjer se v tekstu začne vertikalni niz znakov podan kot drugi parameter.

Vaja 3.13. Slika neke računalniške naprave je definirana z naslednjim tipom

```
type slika = int array array,

```

kjer so pike, ki sestavljajo sliko predstavljene z 0 ali 1. Pika je osvetljena, če ima vrednost 1.

(i) Napiši funkcijo `zasukaj90 : slika -> slika`, ki zasuče sliko za 90 stopinj v smeri urinega kazalca.

(ii) Napiši funkcijo `zasukaj90xn : slika -> int -> slika`, ki zasuče sliko za kot `n*90` v smeri urinega kazalca, kjer je `n` drugi parameter funkcije.

Pazi na dimenzije slik!

Vaja 3.14. Definiraj tip `slika` s katerim predstavimo sliko sestavljeno iz 100x100 točk. Vsaka točka je predstavljana z intenziteto in barvo—obe vrednosti predstavimo s celim številom. Predpostavljamo, da imamo že napisano funkcijo `pika : int*int -> bool`, ki pove ali je na dani koordinati pika. Barva pike ni pomembna. Nekje na sliki je narisan krog z radijem 5.
Napiši funkcijo `poisci : slika -> (int*int)`, ki poišče središče kroga.

Vaja 3.15. Urejevalnik besedil ima tekst predstavljen z dvo dimenzionalno matriko znakov.
a) Definiraj tip tekst, ki predstavlja dvo-dimenzionalno matriko znakov velikosti 100x1000 (1000 vrstic po 100 znakov)..
b) Napiši funkcijo

```
    zamenjaj_navpicno : tekst -> string -> string
    -> tekst,
```

ki poišče vse pojavitve niza (2. parameter) vertikalno v tekstu urejevalnika (1. parameter) in jih zamenja z drugim nizom (3. parameter). Predpostavimo, da sta niza enako dolga.

Vaja 3.16. Definiraj parametrični `('k,'v) ppolje`, ki predstavlja polje parov 'k*'v. Prva komponenta para je ključ tipa 'k in druga vrednost tipa 'v.
Predpostavljaj:
a) Imamo dano funkcijo enako : 'k->'k->bool, ki definira enakost ključev, in funkcijo zdruzi : 'v->'v->'v, ki združi dve vrednosti tipa 'v v eno samo.
b) Polja tipa `('k,'v) ppolje` so vedno sortirana po vrednosti ključa `'k`! Napiši funkcijo:

```
  stik ('k,'v) ppolje -> ('k,'v) ppolje -> ('k,'v
   ) ppolje,
```

ki naredi stik polj tako, da s funkcijo zdruzi združi vrednosti vseh parov, ki se ujemajo v ključu. Rezultat naj bo torej seznam parov tipa ('k,'v).

Vaja 3.17. Definiraj funkcijo

```
zmesaj : 'a array -> (int*int) array -> 'a array,
```

ki kot prvi parameter sprejme polje vrednosti tipa 'a, kot drugi parameter pa polje parov s katerimi so definirane zamenjave elementov. Funkcija naj vrne zakodirano polje.

Polje parov vsebuje pare indeksov s katerimi je definirana zamenjava dveh elementov vhodnega polja.

Vaja 3.18. Predpostavi, da je definiran razred Array, ki vsebuje polje elementov tipa 'a.

```
class ['a] Array (ini: 'a) =
object
     method size : int
     method set : int -> 'a -> unit
     method get : int -> 'a
end
```

Definiraj podrazred ArrayM, ki za dan indeks tipa int lahko vsebuje več kot eno vrednost tipa 'a. Definiraj naslednje metode razreda.

```
get : int -> 'a list     (*vrni elemente z
   indeksom i*)
set : int -> 'a -> unit  (*dodaj element tipa 'a
   k elementom z indeksom i*)
del : int -> 'a -> unit  (*izbri\v si el. tipa 'a
    iz  mnozice elementov z indeksom i*)
```

čim bolje uporabi metode razreda Array.

Vaja 3.19. Matrika je predstavljena s tipom int array array. Napiši funkcijo

```
podniz : int array array -> int array -> bool,
```

ki preveri ali se niz celih števil (2. parameter) pojavi v matriki (1. parameter) na kateri izmed diagonalnih črt, ki poteka iz točk na levi in spodnji strani matrike navzgor proti desni in zgornji strani matrike.

Vaja 3.20. Dano imamo sortirano polje celih števil. Definiraj funkcijo encode, ki vrne polje parov kjer je prva komponenta element vhodnega polja in druga komponenta število pojavitev elementa v polju.

Primer:

```
encode [|1;1;3;4;4;5|] -> [|(1,2);(3,1);(4,2)
   ;(5,1)|]
```

Vaja 3.21. Naloga je sestavljena iz treh delov:

1. Definiraj razred matrix za predstavitev matrik, ki shranjujejo elemente poljubnega tipa 'a.
   a) Argumenti razreda naj definirajo dimenzije matrike in začetno vrednost elementov.
   b) Napiši metodo `set : int*int -> 'a -> unit`, kjer prvi parameter predstavlja indekse elementa in drugi parameter hrani novo vrednost indeksiranega elementa.
   c) Napiši metodo get : int*int -> 'a, ki vrne vrednost elementa indeksiranega s prvim parametrom metode.
2. Definiraj razred `int\_matrix` kot podrazred razreda `matrix`.
3. V razredih matrix in int_matrix definiraj metodo
   `equals : int*int -> int*int -> bool`, ki primerja dva elementa matrike in vrne boolovo vrednost true, če sta enaka in false sicer.

Vaja 3.22. Tekst urejevalnika je shranjen kot polje seznamov besed:

```
type text = string list array;;
```

Definiraj funkcijo `find\_replace : text -> string -> string -> text`, ki zamenja vse pojavitve niza podanega z drugim parametrom z nizom znakov podanim s tretjim parametrom v tekstu podanim s prvim parametrom. Rezutat je tekst z zamenjanim nizom.

Vaja 3.23. Definiraj parametrični tip `'v ppolje`, ki predstavlja polje (array!) parov `string*'v`. Prva komponenta para je ključ tipa string in druga vrednost tipa 'v.

Predpostavljaj, da je polje tipa 'v ppolje sortirano po vrednosti ključa tipa string!

Napiši funkcijo

```
stik 'v ppolje -> 'v ppolje -> 'v ppolje,
```

ki naredi stik dveh polj tako, da rezultat vsebuje pare obeh polj urejenih po ključu tipa string. Primer:

```
# stik [|("ab",10),("de",9) |] [| ("bc",8),("cd"
   ,12)|];;
- : int ppolje = [|("ab",10),("bc",8),("cd",12),(
   "de",9)|]
```

3.3 ZAPISI

Vaja 3.24. Dana je n-terica, ki predstavlja naslednje podatke študenta:

1. ime in priimek,

2. letnik (1 - prvi, 2 - drugi, 3 - tretji),
3. povprečna ocena (6 - zadostno; 7 -dobro; 8,9 - prav dobro; 10 - odlično) in
4. hobiji.

Tip n-terice je predstavljen z naslednjo definicijo.

```
# type student = string*int*int*(string list);;
type student = string * int * int * string list
```

Napiši funkcijo izpis: student -> unit, ki pretvori podatke o študentu v tekstovno obliko. Poskusite uporabiti vzorce, da bi dobili kratko in razumljivo kodo. Primer:

```
# izpis ("Tone Novak",20,1,8,("kolesarjenje"));;
```

Študent Tone Novak obiskuje prvi letnik. Njegova povrečna ocena je prav dobro. Hobiji studenta so: kolesarjenje.

Vaja 3.25. Drevo a_tree je definirano na naslednji način.

```
type 'a a_tree = { mutable key:'a;
                   mutable trees: a_tree list }
```

Napiši funkcijo višjega reda

```
tree_filter : 'a a_tree -> ('a -> bool) -> 'a
   list.
```

Funkcija vrne seznam vseh ključev iz drevesa podanega s prvim parametrim, za katere vrne funkcija podana z drugim parametrom vrednost true.

3.4 KLASIČNE PODATKOVNE STRUKTURE

Vaja 3.26. Dana je polimorfična izvedba sklada, ki je implementirana v ocaml modulu Stack. Osnovne operacije za delo s skladom so razvidne iz naslednjega primera uporabe modula Stack.

```
# let s = Stack.create ();;
val s : '_a Stack.t = <abstr>
# Stack.push 'a' s; Stack.push 'b' s; Stack.push
   'a' s;;
- : unit = ()
# Stack.iter print_char s;;
aba- : unit = ()
```

Napišite program, ki s pomočjo uporabe modula `Stack` ugotovi ali predstavlja vnešeni niz oklepajev pravilno gnezdeno zaporedje:

```
() OK
()() OK
```

```
(())() OK
())( NEOK
()()(())) NEOK
```

Vaja 3.27. Implementirati moramo enostaven *poljski* kalkulator (HP sintaksa), ki deluje na naslednji način.

- V primeru da vnesemo število, ga da na vrh delovnega sklada.
- Če vtipkamo operacijo (plus, minus, deli in množi) vzame! zadnja dva operanda iz sklada in izvede željeno operacijo in rezultat postavi na vrh sklada.

Primer izvajanja kalkulatorja:

```
> 1
1
> 2
2
> plus
3
> 2
2
> minus
1
>
```

3.5 REKURZIVNI TIPI

Vaja 3.28. Dan je tip `2seznam` s katerim je predstavljen dvojno povezan seznam.

```
type 2seznam = {
     value: int;
     mutable next: 2seznam;
     mutable prev: 2seznam
}
```

Cela števila hranimo po naraščajočem vrstnem redu. Napiši funkcijo `odaj : 2seznam -> int -> 2seznam`, ki doda število (2. parameter) na pravo mesto v seznam.

3.6 IMPLEMENTACIJA FUNCKCIJ

Vaja 3.29. Dana je naslednja funkcija.

```
let rec vsota a = match a with 0 -> 0
                | x -> x + vsota (x-1);;
```

Predstavi sklad aktivacijskih zapisov, ki se razvijejo ob klicu vsota 3;;

Vaja 3.30. Dana je funkcija fib, ki izračuna Fibonaccijevo število.

```
# let rec fib n =
    if n < 2 then 1 else fib(n-1) + fib(n-2);;
val fib : int -> int = <fun>
```

Predstavi zaporedje aktivacijskih zapisov, ki se aktivirajo pri izvajanju funkcije fib 4.

4 OBJEKTNO ORIENTIRANI PROGRAMSKI JEZIKI

4.1 DEFINICIJA RAZREDOV

Vaja 4.1. Definiraj naslednja dva razreda za delo s celimi števili.

1. Cela števila bi radi obravnavali kot objekte. Definiraj osnovno aritmetiko za delo s celimi števili: seštevanje, odštevanje, celoštevilsko deljenje in množenje.

2. Pozitivna cela števila skupaj z ničlo so poseben primer celih števil, ki jih imenujemo naravna števila. Vse operacije prilagodi tako, da v primeru, da je parameter operacije negativen, operacija najprej zamenja parameter z absolutno vrednostjo, ga pomnoži s 100 ter šele nato opravi željeno operacijo.

Napiši tudi konstruktorja celih in naravnih števil. Bodi pozoren na uporabo dedovanja pri konstrukciji rešitve.

Vaja 4.2. Definiraj razred, ki vsebuje polje celih števil sortirano po velikosti. Napiši metodo `zdruzi : int array -> int array`, ki pridruži polju celih števil našega razreda že sortirano polje a tako, da je rezultat sortiran.

Vaja 4.3. Definirajte razred matrika s katerim predstavimo dvodimenzionalno matriko realnih števil. Katere atribute potrebujemo?

1. Definiraj bralce in pisalce razreda matrika ter kodo s katero se inicializira na novo kreirana matrika.

2. Napišite metodo `invertirajx2 : unit`, ki invertira matriko po obeh diagonalah.

Vaja 4.4. Firma iz avtomobilske industrije bi želela predstaviti motorje v programskem jeziku Ocaml. Podatkovna struktura naj predstavi hierarhično kompozicijo motorja iz komponent.

Vsaka komponenta motorja ima identifikator, ime, stanje in seznam pod-komponent, ki jih predstavimo spet kot komponente. Imamo še naslednje podatke.

- Stanje komponente pove ali je komponenta delujoča z boolovo vrednostjo (Dela | Nedela).

- Celoten motor je predstavljen kot komponenta.

- Osnovne komponente (osnovni deli) nimajo pod-komponent.

1. Definiraj objektno predstavitev motorja oz. komponent motorja.
2. Predpostavimo, da imamo na voljo funkcijo `test : boolean`, ki za dano osnovno! komponento (identifikator) vrne vrednost boolovo vrednost (Dela | Nedela).

Napiši funkcijo `oznaci : unit`, ki označi vse komponente motorja z Dela oz. Nedela. Edino pravilo, ki ga moramo upoštevati: komponenta, ki ima vse delujoče pod-komponente je tudi sama delujoča (Dela) sicer pa ni delujoča (Nedela).

4.2 HIERARHIJE RAZREDOV

Vaja 4.5. Vrsto (FIFO) in sklad (LIFO) celih števil želimo implementirati z uporabo polja.

Najprej definiraj razred `zbirka`, ki realizira vrsto kjer lahko dodajamo in odvzemamo elemente na začetku in na koncu vrste. Razreda `vrsta` in `sklad` implementiraj kot specializaciji razreda zbirka.

- Vrsta naj ima operaciji `enqueue` in `dequeue`. Prva operacija doda nov element na začetek vrste in druga odvzame element iz konca vrste.
- Sklad naj ima operaciji `push` in `pop`. Operacija `push` doda nov element na vrh sklada in `pop` odvzame elemet iz vrha sklada.

Vaja 4.6. Definiraj hierarhijo razredov za predstavitev geometrijskih objektov: točka, premica in krog.

Vsak geometrijski objekt naj ima funkcijo

```
premakni_se : int -> int -> unit,
```

kjer predstavljata parametra premika po x in y osi.

Vaja 4.7. Roboti živijo v fiksnem vnaprej definiranem dvo-dimenzionalnem svetu točk. Svet ima koordinati x=1..100 in y=1..100. Meje sveta so vgrajene v robote. Na svetu imamo tri vrste robotov.

1. Osnovni robot, ki je definiran z začetno točko x,y in odmikom dx po osi x in odmikom dy po osi y, ki ga naredi ob premiku. Ko pride do konca sveta po koordinati x se dx zamenja z -dx in ko prispe do konca sveta po koordinati y se dy zamenja z -dy.
2. Ponikajoči robot, ki se premika na enak način kot osnovni robot le da vsakih k premikov ponikne in se ne nariše.
3. Cikcak robot, ki se prav tako premika enako kot osnovni robot le da še skače po x osi dve točki od premika osnovnega robota na levo in v naslednjem koraku dve točki na desno in tako naprej.

Definiraj svet robotov v programskem jeziku Ocaml.

- Vsak robot naj se predstavi ob kreaciji, tako da uporabnik vidi katerim razredom robot pripada.
- Napiši metodo `premik : unit`, ki premakne robota na naslednjo pozicijo v odvisnosti od vrste robota. Metoda `premik` najprej izračuna naslednjo pozicijo robota in jo nato izpiše (t.j. izpiše točko x,y) oz. je tudi ne izpiše v primeru ponikajočega robota.

Vaja 4.8. V dvo-dimenzionalnem svetu robotov imamo dve vrsti robotov: x-robota, ki se premika samo po x-osi in y-robota, ki se premika samo po y-osi. Svet je definiran na področjih [-10..10] po x-osi in [-10..10¸] po y-osi. x-robot je definiran na lokaciji, ki je na y-osi, medtem ko ima y-robot lokacijo na x-osi.

Premik robota po x-osi implementiramo tako, da prištejemo x-koordinati 1 oz. -1, odvisno od tega ali se robot premika desno ali levo. Ko robot pride do roba sveta, se obrne v nasprotno smer. Premikanje robota po y-osi je definirano enako kot v primeru premikanja po x-osi le da so osi zamenjane in se robot premika navzgor in navzdol.

1. Definiraj razrede s katerimi predstavimo x-robota in y-robota. Uporabi skupen koren hierarhije razredov `robot`, ki realizira skupne značilnosti robotov.
2. Vsi razredi naj imajo definirano metodo `premik : unit`.

Vaja 4.9. Definirati je potrebno hierarhijo razredov za predstavitev podatkov o študentih v informacijskem sistemu ŠIS. Splošne podatke o študentih predstavimo v razredu Oseba. Bolj specifične podatke predstavimo v razredih Student, PodiplomskiStudent in Asistent. Asistent je poseben primer podiplomskega študenta.

V razredih bi želeli hraniti naslednje podatke: ime in priimek, naslov, tel.številka, vpisan letnik, vpisan program, obstoječa izobrazba in povprečna ocena.

Naloga ima tri dele:

1. Definiraj razrede v programskem jeziku Ocaml.
2. Realiziraj razrede tako, da inicializatorji razreda inicializirajo objekt z začetnimi vrednostmi.
3. V hierarhiji razredov implementiraj metodo `predstavi :unit`, ki predstavi vse lastnosti poljubnega primerka razredov v hierarhiji. Uporabljaj dedovanje in prekrivanje metod!

Vaja 4.10. Hiša je sestavljena iz `N` nadstropij. Vsako nadstropje ima `M` sob. V vsaki sobi imamo termometer. Temperaturo hiše dolo˘cimo tako, da izračunamo povprečje meritev temperature v vseh sobah.

Predpostavimo, da imamo dano funkcijo `temperatura : int -> int -> int`, ki za dano nadstropje in številko sobe vrne vrednost temperature v stopinjah.

- Predstavi opisano okolje z razredi `soba`, `nadstropje` in `hisa`.
- Vsak razred mora imeti metodo `odcitaj_temp : int -> int -> int`, ki vrne temperaturo objekta in postavi trenutno vrednost temperature za dan objekt.
- Razred `hisa` naj vsebuje metodo `povprecna_temp : int`, ki izračuna povprečno teperaturo hiše po zgoraj opisanem postopku.

4.3 REKURZIVNE STRUKTURE OBJEKTOV

Vaja 4.11. Dano je drevo, ki je definirano z razredom `vozel`. Vsako vozlišče hrani vrednost ključa `kljuc`. Za predstavitev pod-dreves uporabimo tip `'a option`.

```
class vozel k =
object
  val kljuc = k
  val levo = None
  val desno = None
end
```

Napiši metodo `preveriUrejenost : boolen`, ki preveri ali je dano drevo urejeno. V danem korenu imajo vsa vozlišča v levem poddrevesu manjšo vrednost ključa in vsa vozlišča v desnem poddrevesu večjo vrednost ključa.

Dodatna naloga: Kako bi sproti pri preverjanju urejenosti ponovno uredil drevo, če imaš dano metodo za vstavljanje v urejeno drevo?

Vaja 4.12. Dano je drevo, ki je definirano z razredom `vozel`. Vsako vozlišče hrani vrednost ključa `kljuc`. Za predstavitev pod-dreves uporabimo tip `'a option`.

```
class vozel k =
object
  val kljuc = k
  val levo = None
  val desno = None
end
```

1. Napiši metodo `vsota : 'a`, ki za dano vozlišče izračuna vsoto vseh ključev vozlišč pod-drevesa.
2. Izpiši korene poddreves s vsoto < 10.

Vaja 4.13. Naprave so sestavljene iz manjših naprav, ki so spet lahko sestavljene iz manjših naprav, itd. Vsaka naprava ima svoje ime, tip in teˇzo.

(A) Definiraj razred `naprava`. Bodi pozor-na/en na definicijo parametrov razreda in na inicializacijo razreda.

(B) Definiraj metodo `prestej : int`, ki za dano napravo vrne število vseh komponent.

(C) Definiraj metodo `listi : naprava list` razreda `naprava`, ki v seznamu vrne liste drevesa vsebovanosti naprav za dano napravo (objekt).

4.4 ABSTRAKTNI RAZREDI

Vaja 4.14. Geometrijski objekt je definiran z virtualnim razredom `geo`, ki vsebuje definiciji naslednjih virtualnih metod.

```
method virtual predstavi : string
method virtual narisi : unit
```

Metoda `predstavi` vrne niz znakov s katerim je opisan geometrijski objekt. Metoda `narisi` nariše dani geometrijski objekt.

Definiraj abstraktni razred `geo` in razrede `tocka`, `krog` in `premica` kot implementacije abstraktnega razreda `geo`. Uporabi dedovanje kjer je mogoče.

Implementiraj metodi predstavi in narisi za vse tri konkretne razrede. Predpostavi, da imamo že napisane naslednje funkcije.

```
narisi_tocko : int*int -> unit
narisi_premico : int*int -> int*int -> unit
narisi_krog : int*int -> int -> unit
```

Uporabi prekrivanje metod in kodo razporedi tako, da bolj specifične metode uporabijo prekrite metode, kjer je mogoče.

Vaja 4.15. Definiraj hierarhijo razredov za geometrijske objekte točko, krog in kvadrat. Poskusi v čim večji meri uporabiti gradnike objektnega modela za učinkovito predstavitev hierarhije.

1. Definiraj razrede `tocka`, `krog` in `kvadrat`. Uporabi abstraktni razred za vrh hierarhije razredov.

2. Za vse geometrijske objekte definiraj metodo `premakni`, ki premakne dan geometrijski objekt. Uporabi dedovanje in prekrivanje metod!

3. Opiši potek inicializacije objekta ob kreiranju novega objekta.

4.5 PARAMETRIČNI RAZREDI

Vaja 4.16. Implementiraj razred class `'a seznam`, ki naredi ovojnico okoli vgrajenega tipa `list`. Implementiraj metode, ki realizirajo običajne operacije nad seznami vključno z operacijo `::` (`cons`), operacijo `@` in testom vsebovanosti `member : 'a -> 'a seznam -> boolean`.

Vaja 4.17. Definiraj parametriziran razred `'a polje`, ki realizira ovojnico okoli tipa `array`. Razred `'a polje` naj ima definirane naseldnje metode.

```
method popravi : int -> 'a -> unit
method preberi : 'a
```

Metoda `popravi` za dan indeks in vrednost tipa 'a postavi vsebino elementa polja z danim indeksom na novo vrednost. Metoda `preberi` vrne za dan indeks vsebino elementa polja.

Polje naj bo kreirano ob kreaciji objekta. Ob kreaciji podamo tudi velikost polja.

Z uporabo prej definiranega razreda `polje` definiraj razred `real_polje`, ki vsebuje realna števila. Re-definiraj metodo preberi tako, da vrne samo celo vrednost realnega števila.

Vaja 4.18. Definiraj razred `queue`, ki implementira običajno vrsto z uporabo polja. Vrsta naj vsebuje elemente tipa `'a`. Razred naj vsebuje naslednje metode.

```
enqueue : 'a -> unit
dequeue : 'a
```

Metoda `enqueue` dodaja elemente po vrsti v polje. V primeru, da pridemo do konca polja začnemo dodajati spet na začetku. Pri dodajanju preveri ali je vrsta že polna.

Metoda `dequeue` jemlje elemente po vrsti iz začetka polja. Ko pridemo do konca polja, začnemo od začetka. Pri jemanju elementov iz vrste preveri ali je vrsta prazna.

Velikost vrste naj bo parameter razreda, ki se takoj pri kreaciji objekta zaokroži navzgor na KB.

Inicializator razreda naj takoj po kreaciji objekta vstavi v vrsto 10 ničel.

Vaja 4.19.

1. Definiraj razred `zbirka`, ki hrani zbirko vrednosti tipa `'a`. Implementiraj naslednje metode.

   ```
   dodaj_zacetek : 'a -> unit
   beri_zacetek : 'a
   dodaj_konec : 'a -> unit
   beri_konec : 'a
   ```

2. Razreda `vrsta` (FIFO) in `sklad` (FILO) implementiraj kot specializaciji razreda `zbirka`. Definiraj ustrezne metode razredov `vrsta` (`enqueue` in `dequeue`) in `sklad` (`push` in `pop`).

Vaja 4.20. Slika naj bo predstavljena s trojicami, ki vsebujejo x in y koordinati ter barvo.

1. Definiraj parametriziran abstraktni razred `'a tocka`, s katero definiramo točke predstavljene z dvema koordinatama tipa int in barvo tipa 'a.

   Definiraj abstraktno metodo `enakost : 'a tocka -> bool`, ki primerja dano točko s točko, ki je podana kot parameter metode `enakost`.

   Definiraj konkreten razred `itocka`, kjer je tip barve celo število (`int`). Razred `itocka` podeduje vse lastnosti razreda `tocka` in implementira metodo `enakost`.

2. Z uporabo prej definiranega abstraktnega razreda `'a tocka` definiraj parametriziran razred `'a slika`, kjer tip `'a` spet predstavlja tip barve točk slike.

   Skiciraj konkretizacijo parametriziranega razreda `'a slika` v razred `intslika`, kjer je spremenljivka tipa `'a` enaka tipu `int`.

Vaja 4.21. V svetu eno-dimenzionalnih robotov se roboti pomikajo po premici levo in desno. Premica ima izhodišče, ki ima vrednost 0. Roboti se lahko pomikajo za korak levo ali korak desno.

Imamo dve vrsti robotov, ki se premikajo po področju [-100..100]:

1. Robot se premakne za eno enoto v izbrano smer. Ko pride do meje se obrne. To vrsto robota predstavimo z razredom `robot1`.

2. Robot se premika naključno za eno enoto desno ali levo. To vrsto robota predstavimo z razredom `robot2`.

(i) Definiraj robote z razredi povezanimi v hierarhijo dedovanja. Vrh hierarhije dedovanja naj bo razred `robot`.

(ii) Definiraj za vse robote metodo `premik`, ki naredi naslednji premik robota.

(iii) Definiraj razred `robot3` kot podrazred razreda `robot1`. Z uporabo prekrivanja definiraj premik tega robota kot dvakraten premik robota iz razreda `robot1`.

Vaja 4.22. (A) Definiraj razred `matrika`, ki predstavlja `mxn` matrike realnih števil. Razred naj se inicializira s parametroma `m` in `n`.

(B) Definiraj naslednje metode razreda `matrika`.

```
get : int -> int -> float
set : int -> int -> float -> unit
mul : matrika -> unit
```

Vse metode spreminjajo matriko predstavljeno z objektom, ki izvrši metodo. Na primer, metoda mul pomnoži dano matriko (objekt) z matriko, ki je podana kot parameter. Rezultat je shranjen v matriko (objekt), ki izvrši metodo.

(C) Definiraj podrazred `kmatrika`, ki predstavlja kvadratno matriko `nxn`.

Vaja 4.23. Kalkulatorji, ki uporabljajo obrnjeno poljsko notacijo shranjujejo operande na skladu.

Definiraj abstraktni razred `rp_calc`, ki implementira kalkulator osnovan na obrnjeni poljski notaciji. Uporabnik komunicira z kalkulatorjem preko vmesnika `rp_calc`. Vrednosti s katerimi dela kalkulator naj bodo poljubnega tipa 'a.

Operacije kalkulatorja so `plus`, `minus`, `mult` in `divide`. Vsaka operacija vzame iz sklada vrhnje dve vrednosti, izračuna operacijo in vrne rezultat na vrh sklada.

Razred `rp_calc` naj implementira tudi operaciji `push` in `pop`.

Definiraj konkreten razred `int_rpc` kot implementacijo razreda `rp_calc` for za konkreten tip `'a=int`.

Vaja 4.24. Dan je razred `sklad`, ki realizira običajen sklad celih števil.

```
class Sklad =
object
     method size : int
     method push : int -> unit
     method pop : int
end
```

Definiraj razred `vrsta`, ki implementira vrsto z dvema skladoma. Prvi sklad predstavlja začetek vrste in drugi sklad predstavlja konec vrste. Razred Vrsta naj vsebuje operaciji:

```
enqueue : int -> unit,
dequeue : int.
```

V primeru, da je drugi sklad prazen in uporabimo operacijo `dequeue`, potem najprej obrnemo prvi sklad in ga shranimo kot drugi sklad.

Vaja 4.25. Dan je razred `Sklad`, ki realizira parametriziran sklad celih števil

```
class ['a] Sklad =
object
     method size : 'a
     method push : 'a -> unit
     method pop : 'a
end
```

Definiraj razred Kalkulator, ki bo osnova za implementacijo enostavnega kalkulatorja, ki deluje z uporabo obrnjene poljske notacije. Kalkulator deluje na naslednji način.

število, ki ga vnesemo da na vrh delovnega sklada. Operacije plus, minus, mnozi in deli vzamejo zadnja dva operanda iz sklada, opravijo operacijo in vrnejo rezultat na sklad.

Kako bi naredili splošen kalkulator, ki lahko uporablja poljubno predstavitev števil?

Vaja 4.26. Definiraj binarno drevo z objekti, ki predstavljajo vozlišča drevesa. Definiraj parametriziran razred `vozlisce` katerega parameter tip `'a` naj predstavlja tip vrednosti vozlišča.

Pod-drevesi danega vozlišča definiraj kot opcijske vrednosti s čimer se izognemo definiciji praznega pod-drevesa. Opcijske vrednosti definiramo z uporabo tipa `'a option`.

```
type 'a option = Some of 'a | None

```

Napiši metodo, ki vstavi novo vozlišče v skrajno levo vejo drevesa.

Vaja 4.27. V programskem jeziku Ocaml imamo definirano podatkovno strukturo seznam na naslednji način:

type 'a element = mutable vrednost:'a; mutable naslednji:'a seznam and 'a seznam = Prazen | Element of 'a element ;;

Napiši funkcijo zdruzi : 'a seznam -> 'b seznam -> ('a * 'b) seznam, ki združi elemente dveh seznamov v en sam seznam tako, da prepiše istoležne elemente v pare. če ne obstaja istoležni par potem naj ima komponenta vrednost () : unit.

Vaja 4.28. Napiši polimorfično funkcijo rapp : ('a->'a)->int->('a->'a). Klic rapp f n applicira funkcijo f n-krat na danem parametru:

rapp f n = (function x -> f (...(f x)...))

Primer:

```
# let f x = x+3;;
val f : int -> int = <fun>
# (rapp f 3) 1;;
- : int = 10
```

Namig: Uporabiš lahko funkcijo compose:

```
# let compose f g x = f (g x) ;;
val compose : ('a -> 'b) -> ('c -> 'a) -> 'c -> '
   b = <fun>
```

Vaja 4.29. Definiraj uporabniški tip 'a btree, ki predstavlja binarno drevo katerega vozlišča vsebujejo vrednost tipa 'a.

Napiši funkcijo izomorfni : 'a btree -> 'a btree -> bool, ki preveri če imata podani drevesi enako strukturo ne glede na vrednosti v vozliščih.

5 MODULARNI PROGRAMSKI JEZIKI

Vaja 5.1. Definiraj modul za delo s pari celih števil. Tip par je definiran z naslednjim stavkom: type par = int*int. Modul naj vsebuje funkcije:

```
sestej a b: par -> par -> par
odstej a b: par -> par -> par
mnozi a b: par -> par -> par
```

Pomen operacij *sestej, odstej* in *mnozi* je običajen:

```
sestej (2,3) (1,2) -> (3,5)
odstej (2,3) (1,2) -> (1,1)
odstej (2,3) (1,2) -> (2,6)
```

Vaja 5.2. Imlementiraj modul za delo z vrsto. V vrsto dodajamo na začetek in odvzemamo objekte na koncu vrste. Bodi pozoren/a na naslednje vidike:

- Tip elementa vrste naj se definira ob kreaciji vrste. Tip elementa je torej poljuben tip `'a` in vrsta je parametrični tip `'a vrsta` .

- Dodaj kodo, ki bo preprečevala pisanje v polno vrsto ter branje iz prazne vrste.

- Za implementacijo vrste lahko uporabiš poljubno podatkovno strukturo.

Vaja 5.3. Turistična Agencija počitniških aranžmajev bo implementirala informacijski sistem. Definirajte modul `Aranzma`, ki vsebuje kodo za delo z aranžmaji. Aranžma je opisan z:

- destinacija (opisno),
- tip_namestitve (opisno),
- trajanje (število), in
- cena (število).

Modul mora poleg same kreacije in ogleda aranžmaja omogočati še popravljanje imena destinacije, popravljanje tipa namestitve, trajanja in cene.

Za aranžma definiraj modula `Agent` in `Stranka`, kjer `Agent` v aranžmaju lahko pogleda aranžma preimenuje destinacijo, določi nov tip namestitve ter popravi trajanje in ceno. `Stranka` lahko zgolj pogleda določeni aranžma.

Vaja 5.4. Novo leto je za nami, vendar se želimo letos že zgodaj pripraviti na naslednje novoletno praznovanje. V ta namen boste naredili modul, ki nam bo pomagal pri izbiri jelke. Jelka naj bo sestavljena iz krosnje in debla. Modul naj zna narediti jelko s poljubno visokim deblom in krošnjo.

a) Naredi funkcijo `ustvari()`, ki ustvari prazno jelko, funkcijo `povejVisino`, ki vrne vsoto višine debla in krošnje, ter funkciji za nastavljanje višine debla in krošnje.

b) Naredi funkcijo za izris jelke (glej primer). Krosnja naj bo sestavljena iz zvezdic `(*)`, deblo pa iz minusov `(-)`. Bodi pozoren/a na presledke.

```
   (*)
  (***)
 (*****)
(*******)
   (-)
   (-)
```

Vaja 5.5. Potrebujemo modul za delo s podatkovno strukturo, ki hrani urejeno sekvenco elementov danega parametričnega tipa 'a v naraščajočem vrstnem redu. Modul bomo imenovali `Usekvenca`. Tip podatkovne strukture, ki predstavlja urejeno sekvenco imenuj `Usekvenca.t`.

Ob kreiranju podatkovne strukture tipa Usekvenca.t podamo kot parameter funkcijo primerjaj : 'a -> 'a -> int, ki vrne -1 v primeru da je prvi parameter manjši od drugega, 0 v primeru, da sta parametra enaka in 1 v primeru, da je drugi parameter večji od prvega. Pozor, funkcijo primerjaj si mora modul zapomniti, ker jo potrebuje pri dodajanju, brisanju in iskanju elementov.

Definiraj naslednje funkcije modula:

```
kreiraj : ('a -> 'a -> int) -> Usekvenca.t
dodaj : Usekvenca.t -> 'a -> unit
izbrisi : Usekvenca.t -> 'a -> unit
min : Usekvenca.t -> 'a
max : Usekvenca.t -> 'a
```

Implementiraj funkcijo kreiraj, eno izmed funkcij dodaj in izbriši ter eno izmed funkcij min ali max.

Vaja 5.6. Implementirati moramo enostaven kalkulator za računanje s celimi števili, ki uporablja poljsko notacijo. Kalkulator deluje na naslednji način:

V primeru da vnesemo število, ga da na vrh delovnega sklada, in če vtipkamo operacijo (plus, minus, deli in množi) vzame! zadnja dva operanda iz sklada, izvede željeno operacijo in rezultat postavi na vrh sklada.

Primer izvajanja kalkulatorja:

```
> 1
1
> 2
2
> plus
```

```
3
> 2
2
> minus
1
>
```

a) Definiraj modul Poljski, ki implementira poljski kalkulator. Modul naj vsebuje naslednje funkcije:

```
(*inicializacija: *)
kreiraj: unit -> Poljski.t
(*vnese \v stevilo na sklad in ga vrne:*)
vnos: Poljski.t -> int -> int
(*se\v steje vrhnja dva elem iz sklada in vrne
   rezultat:*)
plus: Poljski.t -> int
(*od\v steje:  *)
minus: Poljski.t -> int
(*zmno\v zi:  *)
mnozi: Poljski.t -> int
(*deli:  *)
deli: Poljski.t -> int
```

b) Kako bi posplošili kalkulator, da bi znal delati s poljubnim tipom števil npr. tudi z realnimi števili? Skiciraj na kratko rešitev.

c) Dodatna naloga: implementacija b)

Vaja 5.7. Napisati želimo modul Slika za delo s slikami, ki so definirane z naslednjim parametričnim tipom:

```
type 'a slika = { mutable x: int; mutable y: int;
          mutable p: 'a array array };;
```

Slika je torej dvodimenzionalno polje tipa 'a array array. Posamezna točka slike je predstavljena kot vrednost tipa 'a—uporabljamo lahko različne tipe za predstavitev ene točke slike.

Napiši naslednje funkcije modula:

- kreiraj: kreira novo sliko z danimi parametri,
- zrcali_x: zrcali sliko po x osi in
- zrcali_y: zrcali sliko po y osi.

Definiraj vmesnik modula, ki omogoča dostop do predstavljenih funkcij.

Vaja 5.8. Definiraj modul za obdelovo dvodimenzionalnih slik Slika.

a) Slika je predstavljena s trojicami, ki vsebujejo x in y koordinati ter barvo. Vse tri vrednosti so predstavljene s celimi števili.

Definirati je potrebno tip Slika.tip, s katero predstavimo sliko.

Definiraj funkcijo Slika.kreiraj s katero kreiramo novo sliko. Sam določi parametre in rezultat funkcije Slika.kreiraj.

b) Vzorce lahko predstavimo z manjšimi slikami. Točka na sliki se ujema s točko vzorca, če se ujemata v barvah. Napiši funkcijo

```
Slika.ujemanje : 'a Slika.tip -> 'a Slika.tip ->
   (init*int) -> bool,
```

ki za dano sliko (1.parameter) in vzorec (2.parameter) ter koordinati (x,y) (3.parameter) vrne true v primeru, da se vzorec ujema s sliko na koordinatah (x,y) in false sicer.

c) (dodatna naloga) Recimo, da bi želeli definirati fleksibilen modul Slika, ki zna delati z različnimi predstavitvami barv.

Za delo z barvami si definiramo majhen modul `Barva`, ki vsebuje definicijo tipa `Barva.tip` in operacijo

```
Barva.enakost : Barva.tip -> Barva.tip -> bool,
```

ki pove ali sta dve barvi enaki. Skiciraj definicijo funktorja Slika in modula Barva.

5.1 FUNKCIONALI

Vaja 5.9. Implementiraj parametrični modul za delo z urejenimi seznami UrejenSeznam.

Modul UrejenTip, ki služi kot parameter modulu UrejenSeznam naj bo uporabljen za definicijo tipa elementov seznama ter definicjo funkcije UrejenTip.primerjaj a b, ki vrne 0 v primeru a=b, 1 v primeru a>b in -1 v primeru a<b. Funkcija UrejenTip.primerjaj definira urejenost elementov urejenega seznama.

Vaja 5.10. Podan je modul Stack, ki realizira sklad elementov tipa 'a..

```
module type Stack =
sig
   type 'a t
   exception Empty
   val create: unit -> 'a t
   val push: 'a -> 'a t -> unit
   val pop: 'a t -> 'a
end
```

Definiraj modul Queue, ki s pomočjo dveh skladov implementira vrsto. Prvi sklad predstavlja začetek vrste in drugi sklad predstavlja konec vrste. Implementiraj funkciji enqueue and dequeue !

Namig: če je kateri izmed skladov prazen potem lahko drugega "obrnemo".

Vaja 5.11. Definiraj modul, ki zna delati z enodimenzionalnimi polji poljubnega (!) tipa. Tip elementa polja je torej spremenljivka tipa.e

Funkcije: kreiranje, spajanje, uničevanje, odvzem.

Vaja 5.12. Naredi modul "garaza", ki:

a) odpira in zapira vrata garaže z enim samim ukazom (če so vrata odprta jih zapre, sicer naj jih odpre);

b) parkira (odpelje/pripelje) avtomobile ali motorje, pri čemer je število vozil omejeno, motor pa zasede polovico prostora avtomobila;

c) izpiše število in tip vozil v parkiranih v garaži.

Del II

DODATEK

Rešitev za vajo 2.1:

```
function n -> n * (n + 1) * (2*n + 1)/6
```

Rešitev za vajo 2.2:

```

let rec vsota_vrste n = match n with
| 1 -> 1.
| _ -> 1./.2.**(float_of_int n) +. (vsota_vrste (
   n-1))
```

Rešitev za vajo 2.3:

```
let rec fib n = match n with
| 0 -> 1
| 1 -> 1
| _ -> fib (n-2) + fib (n-1);;
```

Rešitev za vajo 2.4:

```
let rec jeVsota (a, b, c) = match b with
| 0 -> if (a = c) then true
else false
| b -> jeVsota ((naslednjik a), b-1, c)
```

Rešitev za vajo 2.5:

```
let vsota (a,b)(c,d) = (a+c, b+d)

let rec pfib (a,b) = match (a,b) with
| (i, j) when i <= 0 && j<=0 -> (1,1)
| (i, 0) -> pfib (i-1, 0)
| (0, j) -> pfib (0, j-1)
| (i, j) -> vsota (pfib (i-1, j-1)) (pfib (i-2, j
   -2))
```

Rešitev za vajo 2.7:

```
sestej [1;2;3] = 1+ sestej [2;3]
sestej [2;3] = 2+ sestej[1]
sestej[1] = 1+ sestej []
sestej[] = 0
sestej[1] = 1+ 0=1
sestej [2;3] = 2+ 1=3
sestej [1;2;3] = 1+ 3=4
```

Rešitev za vajo 2.8:

```
let rec sestej sez = match sez with
| []->0
| hd::tl -> hd+(sestej tl)
```

Rešitev za vajo 2.9:

```
let rec najdi e = function
| [] -> false
| h::t ->if( h == e) then true else najdi e t

let rec unija l1 l2 =
match l1 with
| [] -> l2
| h::t -> if najdi h l2 then unija t l2
else unija t (h::l2)
```

Rešitev za vajo 2.10:

```
let zdruzi sez1 sez2 = sez1 @sez2

(* ali *)

let rec zdruzi sez1 sez2 = match (sez1, sez2)
   with
| ([], s) -> s
| (t, []) -> t
| (a::b, c::d) -> if a<=c then [a]@ (zdruzi b (c
   ::d))
else [c]@(zdruzi (a::b) d)
```

Rešitev za vajo 2.11:

```
et rec zdruzi (sez1,sez2) = match (sez1,sez2)
   with
| ([],x) -> x
```

```
| (x,[]) -> x
| (g1::[],g2::r2) -> g1::g2::r2
| (g1::r1,g2::[]) -> g1::g2::r1
| (g1::r1,g2::g22::r2) -> g1::g2::g22:: zdruzi (
   r1,r2);;
```

Rešitev za vajo 2.12:

```
let rec vecjeod sez n = match sez with
| []->[]
| hd::tl -> if(hd>n) then hd::(vecjeod tl n) else
    (vecjeod tl n)
```

Rešitev za vajo 2.13:

```
let rec seznamnm n m =
if n > m then []
else n :: seznamnm (n+1) m;;
```

Rešitev za vajo 2.14:

```
let palindrom sez =
sez = List.rev sez
```

Rešitev za vajo 2.15:

```
let rec vsotaSodeLihe sez = match sez with
| [] -> (0, 0)
| a::b -> let (l,s) = vsotaSodeLihe b in
if (a mod 2 = 1) then
(l+a, s)
else
(l, s+a)
```

Rešitev za vajo 2.16:

```
let rec podseznam sez1 sez2 = match (sez1, sez2)
   with
| ([], _) -> true
| (a::b, c::d) when List.length sez1 <= List.
   length sez2 -> if (a=c) then podseznam b d
   else false
| _ -> false
```

Rešitev za vajo 2.17:

```
let rec cnta sez = match sez with
| [] -> []
| 'a'::'a'::'a'::r -> 3 :: cnta r
| 'a'::'a'::r -> 2 :: cnta r
| 'a'::r -> 1 :: cnta r
| _::r -> 0 :: cnta r;;
```

Rešitev za vajo 2.19:

```
let rec ace1 sez count = match sez with
| [] -> false
| a::b -> if (count = 0 && a = 'a') then ace1 b 1
else if(count = 1 && a = 'c') then ace1 b 2
else if (count = 2 && a = 'e') then true
else ace1 b count

let ace sez = ace1 sez 0
```

Rešitev za vajo 2.20:

```
let rec fja list = match list with
| [] -> []
| a::[] -> [a]
| a::b when a=0 -> a::(fja b)
| a::b::c when a=1 && b=0 ->[a; b]@(fja c)
| a::b::c when a=1 && b=1 && c=[] -> [2]@(fja c)
| a::b::c::d when a=1 && b=1 -> if c=1 then [3]@(
   fja d)
else [2; c]@(fja d)
```

Rešitev za vajo 2.21:

```
let rec stevila n m = let x=m-1 in
if n > x then []
else n :: stevila (n+1) m;;

let rec cikli m n = match n with
| 0 -> []
| _ -> (stevila 0 m)@cikli m (n-1)
```

Rešitev za vajo 2.26:

```
  let rec zdruzi list1 list2 = match (list1,
   list2) with
  | ([], []) -> []
  | (a, []) -> a
  | ([], b) -> b
  | (a::b, c::d) ->
      if (a < c ) then a::(zdruzi b (c::d))
      else if (c<a) then c::(zdruzi (a::b) d)
           else a::(zdruzi b d)

```

Rešitev za vajo 2.42: Naloga (i):

```
let rec prestej dr = match dr with
| List x -> x
| Drevo (a,b,c) -> prestej a + prestej c
```

Naloga (ii):

```
let rec oznaci dr = match dr with
| List a -> dr
| Drevo (d,f,g) -> Drevo (oznaci d, prestej dr ,
   oznaci g)
```

Rešitev za vajo 2.56:

```
let obrni polje = let len=Array.length polje in
for i=0 to (len/2) do
let temp = polje.(i) in
polje.(i) <- polje.(len-i-1);
polje.(len-i-1) <- temp
done;
polje;;
```

Rešitev za vajo 2.58:

```
let razcepi drevo =
let a = ref[] in
let b = ref[] in
let rec raz dr = match dr with
| Prazno ->(!a,!b)
| Vozliscea(x,y) -> a := !a @ [x]; raz y
| Vozlisceb(x,y) -> b := !b @ [x]; raz y
in
raz drevo;;
```

Rešitev za vajo 2.71:

```
let rec prestej sez geo_objekt= match sez with
| [] -> 0
| hd::tl -> if(hd = geo_objekt)then 1 + (prestej
   tl geo_objekt) else prestej tl geo_objekt;;
```

Rešitev za vajo 2.75:

```
type vrstaKarte = Kraljica | Kralj | Fant | Punca
type barvaKarte = Srce | Kara | Pik | Kriz
type simplKarta = Vrsta of vrstaKarte*barvaKarte
let seznamKart = [(Punca,Srce);(Kralj,Kriz);(Fant
   ,Pik)]
```

Rešitev za vajo 3.5:

```
let stEnic polje =
let count = Array.make 1 0 in
for i = 0 to Array.length polje - 1 do
if (polje.(i) = 1) then count.(0) <- count.(0) +
   1 done; count.(0);;
```

Rešitev za vajo 3.6:

```
let produkt a b =
let polje = Array.make 5 0 in
for i=0 to 4 do
polje.(i) <- a.(i)*b.(i)
done;
polje;;
```

Rešitev za vajo 3.29:

```
vsota(3) = 3 + vsota(2)
vsota(2) = 2+ vsota(1)
vsota(1) = 1+ vsota (0)
vsota (0) = 0
vsota(1) = 1+0=1
vsota(2) = 2+1=3
vsota(3) = 3+3=6
```

Rešitev za vajo 3.30:

```
fib(4) = fib(3) + fib(2)
fib(3) = fib(2) + fib(1)
fib(2) = fib(1) + fib(0)=1
fib(2) = 1+0 = 1
fib(3) = 1+1 = 2
fib(4) = 2+1 = 3
```