Principi programskih jezikov, 1. izpit
18. junij 2019

								Σ	

Ime in priimek Vpisna številka

NAVODILA

- **Ne odpirajte te pole**, dokler ne dobite dovoljenja.
- **Preden začnete reševati test:**
  - Vpišite svoje podatke na testno polo z velikimi tiskanimi črkami.
  - Na vidno mesto položite osebni dokument s sliko in študentsko izkaznico.
  - Preverite, da imate mobitel izklopljen in spravljen v torbi.
  - Prjavite se na spletno učilnico, kamor boste oddajali odgovore.
- Dovoljeni pripomočki: pisalo, brisalo, USB ključ in poljubno pisno gradivo.
- Vse rešitve vpisujte v kviz na spletni učilnici.
- Če kaj potrebujete, prosite asistenta, ne sosedov.
- **Med izpitom ne zapuščajte svojega mesta** brez dovoljenja.
- Testna pola vam bo odvzeta **brez nadaljnjih opozoril**, če:
  - komunicirate s komerkoli, razen z asistentom,
  - komu podate kak predmet ali list papirja,
  - odrinete svoje gradivo, da ga lahko vidi kdo drug,
  - na kak drug način prepisujete ali pomagate komu prepisovati,
  - imate na vidnem mestu mobitel ali druge elektronske naprave.
- **Ob koncu izpita:**
  - Ko asistent razglasi konec izpita, **takoj** nehajte in zaprite testno polo.
  - **Ne vstajajte**, ampak počakajte, da asistent pobere **vse** testne pole.
  - **Testno polo morate nujno oddati.**
- Čas pisanja je 120 minut. Na tabli je zapisano, do kdaj imate čas.
- Predvideni ocenjevalni kriterij:

  ≥ 90 točk, ocena 10
  ≥ 80 točk, ocena 9
  ≥ 70 točk, ocena 8
  ≥ 60 točk, ocena 7
  ≥ 50 točk, ocena 6

Veliko uspeha!

1. naloga (30 točk)

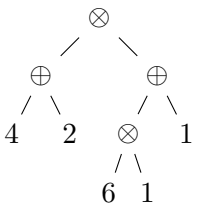
a) (6 točk) Na elbonijski vesoljski postaji uporabljajo nenavadno sintakso za zapis aritmetičnih izrazov:

$$\langle \text{izraz} \rangle ::= \langle \text{številka} \rangle \mid \ominus \langle \text{izraz} \rangle \mid \oplus \langle \text{izraz} \rangle \langle \text{izraz} \rangle \mid \otimes \langle \text{izraz} \rangle \langle \text{izraz} \rangle$$
$$\langle \text{številka} \rangle ::= [0\text{–}9]+$$

Simboli $\ominus$, $\oplus$ in $\otimes$ označujejo nasprotno vrednost, seštevanje in množenje. Na primer, izraz

$$\otimes \oplus 1\ 2 \oplus \ominus 6\ 20$$

ima vrednost 42. Zapišite izraz, ki predstavlja sintaktično drevo



(Namesto znakov $\oplus$, $\otimes$, $\ominus$ lahko v odgovoru uporabite $+$, $*$, $-$.)

b) (6 točk) Definiramo λ-izraze

$$a := \lambda f\, x\, .\, f(f(f(f\, x))), \qquad b := \lambda g\, y\, .\, g(g\, x)).$$

Kateremu izrazu je enak izraz $a\, b\, c\, d$?

(i) $c(c(c(c(c(c(c\, d)))))))$

(ii) $c(c(c(c(c(c(c(c\, d))))))))$

(iii) $c(c(c(c(c(c(c\, d)))))))$

(iv) $\lambda x\, .\, c\, d(c\, d(c\, d(c\, d(c\, d(c\, d(c\, d(c\, d\, x)))))))$

```
a := ^ f x . f ( f ( f ( f x ))) ;
:constant c
:constant d
b := ^ g y . g ( g y )

a b c d

c (c (b c (b (b c) (b (b (b c)) d))))
```

c) (6 točk) Timotej je v OCamlu sestavil funkciji `fold` in `g`:

```
let rec fold f acc = function
  | [] -> acc
  | x :: xs -> fold f (f acc x) xs

let g = fold (fun x ys -> x * (fold ( + ) 0 ys)) 0
```

Kaj izračuna funkcija `g`?

(i) vsoto zmnožkov števil v podseznamih danega seznama

(ii) zmnožek vsot števil v podseznamih danega seznama

(iii) vedno vrne 0

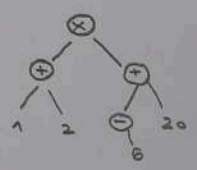
(iv) `g` ni funkcija

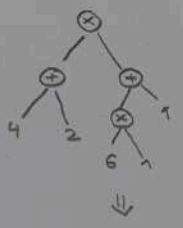
```
utop # g [[1; 2; 3]] ;;
- : int = 0
```

< izraz > ::= < število > | ⊖ <izraz> | ⊕ <izraz> <izraz> | ⊗ <izraz> <izraz>

< število > ::= [0-9]+

⊗ ⊕ 1 2 ⊕ ⊖ 6 20





⇓

⊗ ⊕ 4 2 ⊕ ⊗ 6 1 1

d) (6 točk) Peter je v prologu definiral predikat `a/0`:

```
a :- b, c ; d.
a :- e.
```

Katera logična formula je ekvivalentna temu zapisu?

(i) $a \Rightarrow (b \wedge c \vee d) \wedge e$

(ii) $a \Rightarrow (b \wedge c \vee d) \vee (a \Rightarrow e)$

(iii) $((b \wedge c \vee d) \Rightarrow a) \wedge (e \Rightarrow a)$

(iv) $((b \wedge c \vee d) \Rightarrow a) \vee (e \Rightarrow a)$

e) (6 točk) Dan je programski jezik z zapisi in podtipi, pri čemer za tipe zapisov uporabljamo podtipe v širino in globino, velja pa še $\texttt{int} \leq \texttt{float}$. Andrej je definiral tipa zapisov:

$$\texttt{type}\ a = \{x : \texttt{float}\}$$
$$\texttt{type}\ b = \{x : \texttt{int}; f : \texttt{int} \to \texttt{float}\}$$

Označite pravilne izjave:

(i) $a \leq b$

(ii) $b \leq a$

(iii) $(a \to \texttt{float} \to \texttt{float}) \leq (a \to \texttt{float})$

(iv) $(a \to \texttt{int}) \leq (b \to \texttt{float})$

2. naloga (35 točk)

Dokažite *popolno* pravilnost programa:

```
[ a < b ]
x := a ;
y := b ;
while x < y do
  x := x + 1 ;
  y := y - 1
done
[ a + b ≤ 2x ≤ a + b + 1 ]
```

DELNA PRAVILNOST

Invarianta: a + b = x + y, x ≤ y + 1

{ a < b }

x := a ;

{ a < b, x = a } <==> { x < b }

y := b ;

{ x < b, y = b } <==> { x < y }

=?=> { a + b = x + y, x ≤ y + 1 } => Q
(x + y = x + y, x < y => x ≤ y + 1)

while x < y do

{ a + b = x + y, x ≤ y + 1 } => Q

{ a + b = x + y, x < y }

{ a + b = x + 1 - 1 + y, x + 1 - 1 < y}

x := x + 1 ;

{ a + b = x - 1 + y, x < y + 1 } <==> { a + b = x + y - 1, x < y - 1 + 1 + 1 }

y := y - 1

{ a + b = x + y, x < y + 2 } <==> { a + b = x + y, x ≤ y + 1 } => Q

done

{ a + b = x + y, x ≤ y + 1 } <==>

{ a + b = x + y, x ≤ y + 1 , y ≤ x } <==>

{ a + b = x + y, y ≤ x ≤ y + 1 } <==>

{ a + b = x + y, x + y ≤ x + x ≤ x + y + 1 } <==>

{ a + b = x + y, x + y ≤ 2x ≤ x + y + 1 } Q <==>

[a + b ≤ 2x ≤ a + b + 1]

POPOLNA PRAVILNOST

e = z ... e < z

P: e = y + 1 - x, y + 1 - x < 0

[e = y + 1 - x, y + 1 - x < 0]

while x < y do

  [e = y + 1 - x, - x < - y - 1, x < y] # vzamemo močnejšo

  [e = y + 1 - (x + 1 - 1), x + 1 - 1 < y]

  x := x + 1 ;

  [e = y + 1 - x + 1, x - 1 < y]

  [e = y - 1 + 1 + 1 - x + 1, x < y + 1 - 1]

  y := y - 1

  [e = y + 1 + 1 - x + 1, x < y - 1]

  [e - 2 = y + 1 - x, x < y - 1]

done

[e = y + 1 - x, y + 1 - x < 0] => Q
while x < y do

  [e = y + 1 - x, - x < - y - 1, x < y] # vzamemo močnejšo

  x := x + 1;

  [e = y + 2 - x, x < y + 1]
  [e = y + 1 - 1 + 2 - x, x < y + 1 - 1 + 1]

  y := y - 1

  [e = y + 1 + 2 - x, x < y + 1 + 1]
  [e = y + 3 - x, x < y + 2]
  [y + 1 - x = e - 2, y + 2 - x > 0] # zmanjšuje in omejeno navzdol
done

3. naloga (35 točk)

Predikat `cesta/2` pove, kateri izmed krajev `a`, `b`, ..., `h` so neposredno povezani s cesto:

```
cesta(a, b).
cesta(a, e).
cesta(a, c).
cesta(b, d).
cesta(d, e).
cesta(e, f).
cesta(g, h).
```

Ceste so *dvosmerne*, torej iz `a` v `b` vodi cesta, po kateri lahko potujemo v obe smeri.

a) Sestavite predikat `povezava(X,Y)`, ki velja natanko tedaj, ko obstaja *neposredna* cesta od `X` do `Y` ali od `Y` do `X`.

b) Sestavite predikat `pot(X,Y)`, ki velja natanko tedaj, ko obstaja pot med `X` in `Y`. Primer uporabe:

```
?- pot(a,a).
true .
?- pot(f, a).
true.
```

Če ni rešitve, lahko poizvedba išče rešitev v nedogled.

c) Zapišite predikat `pot(X,Y,P)`, ki velja natanko tedaj, ko je `P` pot od `X` do `Y`. Primer uporabe:

```
?- pot(a, f, P).
P = [a, b, d, e, f] ;
...
```

d) Sestavite predikat `pot(X,Y,P,N)`, ki velja natanko tedaj, ko je `P` pot dolžine `N` od `X` do `Y`. Primer uporabe:

```
?- pot(a,f,P,5).
P = [a, b, d, e, f] ;
P = [a, b, a, e, f] ;
P = [a, e, f, e, f] ;
P = [a, e, a, e, f] ;
P = [a, e, d, e, f] ;
P = [a, c, a, e, f] ;
false.
```

e) Poleg cest med kraji so podane tudi njihove nadmorske višine:

```
visina(a, 10).
visina(b, 7).
visina(c, 12).
visina(d, 4).
visina(e, 6).
visina(f, 7).
visina(g, 0).
visina(h, 9).
```

Sestavite predikat `spust(X,Y)`, ki velja natanko tedaj, ko obstaja pot od `X` do `Y` brez vzpenjanja. Primer uporabe:

```
?- spust(a,f).
false.
?- spust(a,d).
true .
```

a)

```
povezava(X, Y) :-
    cesta(X, Y);
    cesta(Y, X).
```

b)

```
pot(X, X).

pot(X, Y) :-
    povezava(X, Y).

pot(X, Y) :-
    povezava(X, Y),
    cesta(Z, Y).
```

c)

```
# ne deluje
pot(X, X, [X]).
pot(X, Y, P) :-
    (
        cesta(X, PV);
        cesta(PV, X)
    ),
    pot(PV, Y, P2),
    P = [X|P2].
```

d)

```
pot(X, Y, P, N) :-
    pot(X, Y, P),
    length(P, N).
```

e)

```
spust(X, Y) :-
    povezava(X, Y),
    visina(X, H1),
    visina(Y, H2),
    H1 >= H2.
```

Principi programskih jezikov, 3. izpit
27. avgust 2018

								Σ	

Ime in priimek

Vpisna številka

NAVODILA

- **Ne odpirajte te pole**, dokler ne dobite dovoljenja.
- **Preden začnete reševati test:**
  - Vpišite svoje podatke na testno polo z velikimi tiskanimi črkami.
  - Na vidno mesto položite osebni dokument s sliko in študentsko izkaznico.
  - Preverite, da imate mobitel izklopljen in spravljen v torbi.
  - Prjavite se na spletno učilnico, kamor boste oddajali odgovore.
- Dovoljeni pripomočki: pisalo, brisalo, USB ključ in poljubno pisno gradivo.
- Rešitve vpisujte v kviz na spletni učilnici, 2. nalogo pa rešujete v to polo.
- Če kaj potrebujete, prosite asistenta, ne sosedov.
- **Med izpitom ne zapuščajte svojega mesta** brez dovoljenja.
- Testna pola vam bo odvzeta **brez nadaljnjih opozoril**, če:
  - komunicirate s komerkoli, razen z asistentom,
  - komu podate kak predmet ali list papirja,
  - odrinete svoje gradivo, da ga lahko vidi kdo drug,
  - na kak drug način prepisujete ali pomagate komu prepisovati,
  - imate na vidnem mestu mobitel ali druge elektronske naprave.
- **Ob koncu izpita:**
  - Ko asistent razglasi konec izpita, **takoj** nehajte in zaprite testno polo.
  - **Ne vstajajte**, ampak počakajte, da asistent pobere **vse** testne pole.
  - **Testno polo morate nujno oddati.**
- Čas pisanja je 120 minut. Na tabli je zapisano, do kdaj imate čas.
- Doseženih 100 točk šteje za maksimalno oceno. Veliko uspeha!

1. naloga (30 točk)

a) (6 točk) V antični Elboniji so uporabljali nenavadno sintakso za zapis aritmetičnih izrazov:

$$\langle \text{izraz} \rangle ::= \langle \text{številka} \rangle \mid \langle \text{izraz} \rangle \ominus \mid \langle \text{izraz} \rangle \langle \text{izraz} \rangle \oplus \mid \langle \text{izraz} \rangle \langle \text{izraz} \rangle \otimes$$
$$\langle \text{številka} \rangle ::= [0\text{–}9]+$$

Simboli $\ominus$, $\oplus$ in $\otimes$ označujejo nasprotno vrednost, seštevanje in množenje. Na primer, izraz

$$20\ 6\ \ominus\ \oplus\ 2\ 1\ \oplus\ \otimes$$

ima vrednost 42. Narišite sintaktično drevo, ki predstavlja zgornji izraz.

b) (6 točk) V λ-računu definiramo izraza

$$K := \lambda\, x\, y\, .\, x, \qquad\qquad S := \lambda\, x\, y\, z.\, (x\, z)\, (y\, z).$$

(i) Izračunajte vrednost izraza $S\, K\, K$. λ z . z

(ii) Izračunajte vrednost izraza $S\, K\, S$. λ z . z

```
:deep
K := ^ x y . x ;

S := ^ x y z . (x z)(y z) ;

A := S K K ;
B := S K S ;
```

c) (6 točk) Timotej je sestavil funkcijo v Haskellu:

```
h :: [[a]] -> [a]
h [] = []
h ([] : ys) = h ys
h ([x] : ys) = x : h ys
h ((_:xs) : ys) = h (xs : ys)
```

Kaj izračuna funkcija `h`?

(i) seznam zadnjih elementov vseh nepraznih seznamov danega seznama seznamov

(ii) seznam praznih seznamov danega seznama seznamov

(iii) seznam prvih elementov vseh nepraznih seznamov danega seznama seznamov

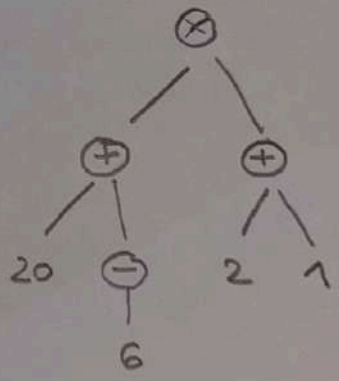
(iv) Haskell zavrne definicijo, ker ta vsebuje vsebuje napako

<izraz> ::= <število> | <izraz> ⊖ | <izraz> <izraz> ⊕ | <izraz> <izraz> ⊗

20 6 ⊖ ⊕ 21 ⊕ ⊗

⋮

42



d) (6 točk) Peter je sestavil predikat v prologu:

```
h([], []).
h([[]|Y], Z) :- h(Y,Z).
h([[X|_]|Y], [X|Z]) :- h(Y,Z).
```

Kaj pomeni `h(X,Y)`?

(i) `Y` je seznam zadnjih elementov vseh nepraznih seznamov seznama `X`

(ii) `Y` je seznam praznih seznamov seznama `X`

(iii) `Y` je seznam prvih elementov vseh nepraznih seznamov seznama `X`

(iv) prolog zavrne definicijo, ker ta vsebuje vsebuje napako

e) (6 točk) SML izraz

```
[(fn (x, y) => (y, x)), (fn (a, b) => (42, b))]
```

ima tip:

(a) $\texttt{int} \times \texttt{int} \to \texttt{int} \times \texttt{int}$

(b) $(\texttt{int} \times \texttt{int} \to \texttt{int} \times \texttt{int})\ \texttt{list}$

(c) $(\alpha \times \texttt{int} \to \texttt{int} \times \alpha)\ \texttt{list}$

(d) $(\alpha \times \beta \to \beta \times \alpha)\ \texttt{list}$

```
- : (int * int -> int * int) list = [<fun>; <fun>]
```

2. naloga (40 točk)

To nalogo lahko rešujte neposredno na izpitno polo, ki jo boste ob koncu izpita oddali, ali rešitev vtipkate v izpit na spletni učilnici.

Dokažite *delno* pravilnost programa:

```
{ x ≤ y }
c := y ;
if z <= x then
  a := z ;
  b := x ;
else
  a := x ;
  if z <= y then
    b := z
  else
    b := y ;
    c := z
  end
end
{ a ≤ b ∧ b ≤ c }
```

```
{ x ≤ y }
c := y ;
{ x ≤ y, c = y }
if z <= x then
  { x ≤ y, c = y, z ≤ x }
  a := z ;
  { x ≤ y, c = y, a ≤ x }
  b := x ;
  { b ≤ y, c = y, a ≤ b }
  { b ≤ c, a ≤ b }
else
  { x ≤ y, c = y, z > x }
  a := x ;
  { a ≤ y, c = y, z > a }
  if z <= y then
    { a ≤ y, c = y, z > a, z ≤ y }
    b := z
    { a ≤ y, c = y, b > a, b ≤ y }
    { a ≤ c, a < b, b ≤ c }
    { a ≤ b, b ≤ c }
  else
    { a ≤ y, c = y, z > a, z > y }
    b := y ;
    { a ≤ b, c = b, z > a, z > b }
    { a ≤ b, z > a, z > b }
    c := z
    { a ≤ b, c > a, c > b }
    { a ≤ b, b ≤ c}
  end
  { a ≤ b, b ≤ c}
end
{ a ≤ b ∧ b ≤ c }
```

3. naloga (40 točk)

Andrej je sestavil preprost program v Haskellu za predstavitev naravnih števil v eniškem sistemu:

```
-- eniska predstavitev stevil
data Stevilo =
    Z             -- nic
  | S Stevilo     -- naslednik
  deriving (Eq, Show)

-- primer: stevilo 5 je petkratni naslednik stevila 0
pet :: Stevilo
pet = S (S (S (S (S Z))))

vsota :: Stevilo -> Stevilo -> Stevilo
vsota Z y = y
vsota (S x) y = S (vsota x y)

produkt :: Stevilo -> Stevilo -> Stevilo
produkt Z _ = Z
produkt (S x) y = vsota (produkt x y) y

stevilo :: Integer -> Stevilo
stevilo 0 = Z
stevilo n = S $ stevilo $ (n - 1)
```

Program predelajte v Prolog:

1. Atoma `z` in `s` naj predstavljata nič in operacijo naslednik. Na primer `s(s(s(s(s(z)))))` predstavlja število pet.
2. Definirajte predikat `vsota/3`, kjer `vsota(X,Y,Z)` pomeni, da je `Z` vsota `X` in `Y`.
3. Definirajte predikat `produkt/3`, kjer `produkt(X,Y,Z)` pomeni, da je `Z` zmnožek `X` in `Y`.
4. Definirajte predikat `stevilo/2`, kjer `stevilo(N,X)` pomeni, da je `N` običajno celo število v prologu in `X` isto število predstavljeno v eniškem sistemu.

1.

```
z.
n(X) :-
    X = z;
    X = n(_).
```

2.

```
vsota(z, X, X).
vsota(n(X), Y, n(Z)) :-
    vsota(X, Y, Z).
```

3.

```
produkt(z, _, z).
produkt(n(X), Y, Z) :-
    vsota(P1, Y, Z),
    produkt(X, Y, P1).
```

4.

```
stevilo(0, z).
stevilo(N, n(X)) :-
    N #= NX + 1,
    stevilo(NX, X).
```

5. Poišči skupne delitelje števil 252 in 294.

```
stevilo(252, A), stevilo(294, B), produkt(D, _, A), produkt(D, _, B),  stevilo(N, D).
```

Principi programskih jezikov, 2. izpit
5. julij 2018

								Σ	

Ime in priimek

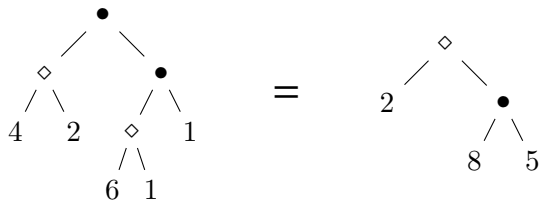
Vpisna številka

NAVODILA

- **Ne odpirajte te pole**, dokler ne dobite dovoljenja.
- **Preden začnete reševati test:**
  - Vpišite svoje podatke na testno polo z velikimi tiskanimi črkami.
  - Na vidno mesto položite osebni dokument s sliko in študentsko izkaznico.
  - Preverite, da imate mobitel izklopljen in spravljen v torbi.
  - Prjavite se na spletno učilnico, kamor boste oddajali odgovore.
- Dovoljeni pripomočki: pisalo, brisalo, USB ključ in poljubno pisno gradivo.
- Rešitve vpisujte v kviz na spletni učilnici, 2. nalogo pa rešujete v to polo.
- Če kaj potrebujete, prosite asistenta, ne sosedov.
- **Med izpitom ne zapuščajte svojega mesta** brez dovoljenja.
- Testna pola vam bo odvzeta **brez nadaljnjih opozoril**, če:
  - komunicirate s komerkoli, razen z asistentom,
  - komu podate kak predmet ali list papirja,
  - odrinete svoje gradivo, da ga lahko vidi kdo drug,
  - na kak drug način prepisujete ali pomagate komu prepisovati,
  - imate na vidnem mestu mobitel ali druge elektronske naprave.
- **Ob koncu izpita:**
  - Ko asistent razglasi konec izpita, **takoj** nehajte in zaprite testno polo.
  - **Ne vstajajte**, ampak počakajte, da asistent pobere **vse** testne pole.
  - **Testno polo morate nujno oddati.**
- Čas pisanja je 120 minut. Na tabli je zapisano, do kdaj imate čas.
- Doseženih 100 točk šteje za maksimalno oceno. Veliko uspeha!

1. naloga (30 točk)

a) (6 točk) Elbonijci so zelo napredna družba, zato aritmetične izraze predstavijo kar z drevesi, a uporabljajo drugačne simbole kot mi. Na obisku Elbonije je slovenski predsednik obiskal vrtec, kjer je bila na tabli napisana enakost:



Vzgojiteljica je pojasnila, da vadijo seštevanje in množenje. K predsedniku je pristopila deklica, ga pocukala za rokav, in nekaj vprašala. Prevajalka je prevedla: "Gospod v lepi obleki, kakšna je vrednost izrazov na tabli?" Vse oči so bile uprte v predsednika, ki je prebledel, a šef varnostne službe, ki je pred leti opravil predmet Principi programskih jezikov, mu je priskočil na pomoč. Katero število je prišepnil šef varnostne službe predsedniku?

b) (6 točk) V λ-računu definiramo izraza

$$O := \lambda\, x\, y\, .\, x, \qquad\qquad I := \lambda\, x\, y\, .\, y.$$

Predstavljamo si, da je O bit nič in I bit ena. Definirajte λ-izraz X, ki izračuna bitni XOR, se pravi, da zadošča enačbam

$$X\, O\, O = O, \qquad X\, O\, I = I,$$
$$X\, I\, O = I, \qquad X\, I\, I = O.$$

c) (6 točk) Kaj počne naslednja funkcija v SML?

```
fun f xs =
    let fun e [] [] = true
          | e (u :: us) (v :: vs) = (u = v) andalso e us vs

        fun s ys [] = e ys xs
          | s ys (z :: zs) = s (z :: ys) zs
    in
        s [] xs
    end
```

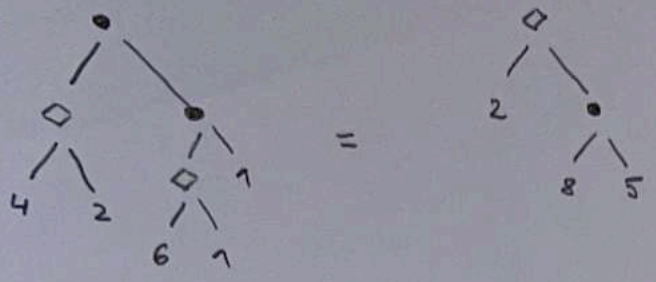
(a) ugotovi, ali sta prvi in zadnji element seznama `xs` enaka,

(b) preveri, ali so vsi elementi seznama `xs` med seboj enaki,

(c) vedno vrne `true`,

(d) ugotovi, ali je `xs` palindrom (se ne spremeni, če ga obrnemo).

a)



◇ : množenje $(4 \cdot 2) + (6 \cdot 1) + 1 \Rightarrow 8 + 6 + 1 = 15$

• : seštevanje $2 \cdot (8+5) = 26$

• : množenje $(4+2) \cdot (6+1) \cdot 1 \Rightarrow 6 \cdot 7 = \underline{42}$ ✓

◇ : seštevanje $2 + 8 \cdot 5 = \underline{42}$

b) XOR

A	B	
0	0	→ 0
0	1	→ 1
1	0	→ 1
1	1	→ 0

$0 := \lambda xy.\, x$ ("true")

$I := \lambda xy.\, y$ ("false")

NOT $:= \lambda x.\, x\ I\ 0$

XOR $:= \lambda xy.\, x\ y\ (\text{NOT}\ y)$

~~d)~~ (6 točk) Andrej je definiral signaturo v SML:

```
signature S =
sig
    type t
    val pi : real
    val f : t -> t -> t
    val g : 'a -> 'a list
end
```

Timotej je implementiral štiri strukture:

```
structure Foo =
struct
  type t = int -> int
  val pi = 3.141592653589793
  fun g x = [x]
  fun r x = [x]
  fun f h k x = k (h x)
end

structure Bar =
struct
  type t = bool
  type s = int * int
  fun f b c = b
  fun g k = k :: g (k + 1)
  val pi = if 17 * 18 < 20 * 15 then 42.0 else 23.0
end

structure Baz =
struct
  type t = bool
  fun f (h, k) = (fn x => h (k x))
  fun g _ = []
  val pi = 42
end

structure Qux =
struct
  type t = bool
  fun f (h, k) = (fn x => h (k x))
  fun r x = [x]
  val pi = 3.141592653589793
end
```

Označite tiste strukture, ki zadoščajo signaturi `S`.

e) (6 točk) V prologu je dan predikat `appears(F,S)`, ki pomeni, da se superjunak `S` pojavi v filmu `F`. Dana je baza dejstev:

```
appears(iron_man, iron_man).
appears(the_incredible_hulk, hulk).
appears(iron_man_2, iron_man).
appears(iron_man_2, black_widow).
appears(avengers, iron_man).
appears(avengers, captain_america).
appears(avengers, hulk).
appears(avengers, thor).
appears(avengers, black_widow).
appears(avengers, hawkeye).
appears(captain_america_civil_war, captain_america).
appears(captain_america_civil_war, iron_man).
appears(captain_america_civil_war, black_widow).
appears(captain_america_civil_war, spider_man).
appears(captain_america_civil_war, black_widow).
appears(captain_america_civil_war, hawkeye).
appears(captain_america_civil_war, ant_man).
appears(captain_america_civil_war, vision).
appears(spiderman_homecoming, iron_man).
appears(spiderman_homecoming, spider_man).
```

Zapišite poizvedbo prologu, ki v spremenljivko `S` prireja superjunake, ki se pojavijo v *vsaj dveh* filmih. Poizvedba sme istega superjunaka našteti večkrat.

```
vsajDva(S) :-
  appears(F1, S),
  appears(F2, S),
  dif(F1, F2).
```

2. naloga (40 točk)

To nalogo rešujte neposredno na izpitno polo, ki jo boste ob koncu izpita oddali.

a) (30 točk) Dokažite *delno* pravilnost programa:

```
{ n > 0 }
s := 0 ;
k := 0 ;
a := 1 ;
while k <= n do
  s := s + a ;
  a := a * n ;
  k := k + 1
done
{ (n − 1) · s = a − 1 }
```

b) (10 točk) Dokažite še popolno pravilnost, se pravi, utemeljite, da se zanka `while` pri danih predpostavkah vedno zaključi.

DELNA PRAVILNOST

{n > 0}

s := 0 ;
k := 0 ;
a := 1 ;

{ n > 0, s = 0, k = 0, a = 1} ==> { (n - 1) * s = a - 1 } <=> I

while k <= n do

  { (n - 1) * s = a - 1 } <=> I
  { n * s + a * n - s - a = a * n - 1 }
  { (n - 1) * (s + a) = a * n - 1 }

  s := s + a ;

  { (n - 1) * s = a * n - 1 }

  a := a * n ;

  { (n - 1) * s = a - 1 }

  k := k + 1 # ne potrebujemo, ker ni v končnem pogoju

  { (n - 1) * s = a - 1 } <=> I
done

{ (n - 1) · s = a - 1 } <=> I

I: (n - 1) · s = a - 1

POPOLNA PRAVILNOST

[n - k = e, n - k ≥ 0]
while k <= n do

  [n - k = e, n - k ≥ 0]

  s := s + a ;

  [n - k = e, n - k ≥ 0]

  a := a * n ;

  [n - k = e, n - k ≥ 0]
  [n - (k + 1 - 1) = e, n - (k + 1 - 1) ≥ 0]

  k := k + 1

  [n - k + 1 = e, n - (k - 1) ≥ 0]
  [n - k = e - 1, n - k + 1 ≥ 0]
  [n - k = e - 1, n - k ≥ - 1]

  Količina n - k se je zmanjšala in je omejena navzdol, torej vel

3. naloga (40 točk)

To nalogo lahko rešujete v SML ali v Haskellu. Če jo rešujete v SML, za tok podatkov uporabite podatkovni tip

```
datatype 'a stream = Cons of 'a * (unit -> 'a stream)
```

in če jo rešujete v Haskellu, podatkovni tip

```
data Stream a = Cons (a, Stream a)
```

Neskončno zaporedje podatkov včasih vsebuje ponavljajoče se znake, na primer:

$$a, a, a, a, a, a, a, b, c, d, e, e, e, e, e, b, b, b, b, \ldots$$

Peter se je domislil kodiranja, pri katerem n-kratno ponovitev znaka x predstavi s parom (x, n). Na primer, zgornji tok bi predstavil s kodiranim tokom

$$(a, 7), (b, 1), (c, 1), (d, 1), (e, 5), (b, 4), \ldots,$$

(Petru se še ni posvetilo, da se ne splača kodirati blokov brez ponavljajočih se znakov.)

a) (20 točk) Definirajte funkcijo v SML

```
val decode : ('a * int) stream -> 'a stream
```

oziroma funkcijo v Haskellu

```
decode :: Stream (a, Int) -> Stream a
```

ki kodiran tok podatkov spremeni nazaj v prvotni tok.

b) (20 točk) Definirajte funkcijo v SML

```
val encode : ''a stream -> (''a * int) stream
```

oziroma funkcijo v Haskellu

```
encode :: Eq a => Stream a -> Stream (a, Int)
```

ki tok podatkov pretvori v kodiran tok podatkov.

Timotej je opazil, da lahko nastopi težava pri kodiranju toka, v katerem se ena vrednost ponavlja v nedogled. Kako vaša rešitev deluje na takem toku? Odgovor zapišite v komentar.

```
datatype 'a stream = Cons of 'a * (unit -> 'a stream)

fun force s = s ()

fun chop 0 _ = []
  | chop n (Cons (x, s)) = x :: chop (n-1) (force s)

fun repeat 0 x s = force s
  | repeat n x s = Cons (x, fn () => repeat (n-1) x s)

fun decode (Cons ((x, n), s)) =
    repeat n x (fn () => decode (force s))

fun encode (Cons (x, s)) =
    let fun accumulate x n (Cons (y, s)) =
          if x = y then
              accumulate x (n+1) (force s)
          else
              Cons ((x, n), fn () => accumulate y 1 (force s))
    in
        accumulate x 1 (force s)
    end
```

Principi programskih jezikov, 1. izpit
20. junij 2018

1	
2	
3	
Σ	

Ime in priimek

Vpisna številka

NAVODILA

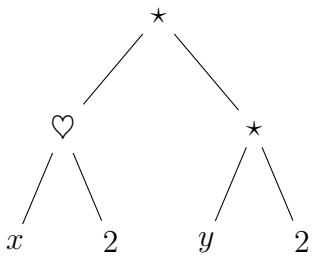
- **Ne odpirajte te pole**, dokler ne dobite dovoljenja.
- **Preden začnete reševati test:**
  - Vpišite svoje podatke na testno polo z velikimi tiskanimi črkami.
  - Na vidno mesto položite osebni dokument s sliko in študentsko izkaznico.
  - Preverite, da imate mobitel izklopljen in spravljen v torbi.
- Dovoljeni pripomočki: pisalo, brisalo, USB ključ in poljubno pisno gradivo.
- Rešitve vpisujte v kviz na spletni učilnici.
- Če kaj potrebujete, prosite asistenta, ne sosedov.
- **Med izpitom ne zapuščajte svojega mesta** brez dovoljenja.
- Testna pola vam bo odvzeta **brez nadaljnjih opozoril**, če:
  - komunicirate s komerkoli, razen z asistentom,
  - komu podate kak predmet ali list papirja,
  - odrinete svoje gradivo, da ga lahko vidi kdo drug,
  - na kak drug način prepisujete ali pomagate komu prepisovati,
  - imate na vidnem mestu mobitel ali druge elektronske naprave.
- **Ob koncu izpita:**
  - Ko asistent razglasi konec izpita, **takoj** nehajte in zaprite testno polo.
  - **Ne vstajajte**, ampak počakajte, da asistent pobere **vse** testne pole.
  - **Testno polo morate nujno oddati.**
- Čas pisanja je 120 minut. Na tabli je zapisano, do kdaj imate čas.
- Doseženih 100 točk šteje za maksimalno oceno. Veliko uspeha!

1. naloga (30 točk)

a) (6 točk) V Elboniji uporabljajo za aritmetične izraze drugačne simbole kot v Sloveniji. Sintaksa je podana s pravili:

$$
\begin{aligned}
\langle\text{aritmetični-izraz}\rangle &::= \langle\text{srčni-izraz}\rangle \\
\langle\text{srčni-izraz}\rangle &::= \langle\text{zvezdni-izraz}\rangle \mid \langle\text{srčni-izraz}\rangle \heartsuit \langle\text{zvezdni-izraz}\rangle \\
\langle\text{zvezdni-izraz}\rangle &::= \langle\text{zaboden-izraz}\rangle \mid \langle\text{zvezdni-izraz}\rangle \star \langle\text{zaboden-izraz}\rangle \\
\langle\text{zaboden-izraz}\rangle &::= \langle\text{spremenljivka}\rangle \mid \langle\text{številka}\rangle \mid \dagger\langle\text{zaboden-izraz}\rangle \mid (\langle\text{srčni-izraz}\rangle) \\
\langle\text{spremenljivka}\rangle &::= [a - zA - z]+ \\
\langle\text{številka}\rangle &::= [0 - 9]+
\end{aligned}
$$

Simbol $\dagger$ ima prednost pred $\star$, ki ima prednost pred $\heartsuit$. Simbol $\heartsuit$ je levo asociativen in $\star$ desno asociativen. Drevo



predstavlja elbonijski aritmetični izraz. Zapišite ga v konkretni sintaksi s čim manjšim številom oklepajev.

b) (6 točk) Timotej je pognal program

```
while k > 0 do
  if k mod 2 = 0 then
    d := d + 1
  else
    skip
  end ;
  k := k div 2
done
```

v okolju $[a \mapsto 0, d \mapsto 3, k \mapsto 42]$. Kakšno je končno okolje, ko se program konča:

1. $[a \mapsto 1, d \mapsto 6, k \mapsto 1]$
2. $[d \mapsto 6, k \mapsto 0]$
3. $[a \mapsto 0, d \mapsto 6, k \mapsto 0]$
4. $[a \mapsto 0, d \mapsto 0, k \mapsto 0]$

< aritmetični - izraz > ::= < srčni - izraz >

< srčni - izraz > ::= < zvezdni - izraz > | < srčni - izraz > ♡ < zvezdni - izraz >

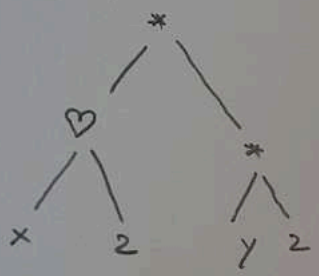
< zvezdni - izraz > ::= < zaboden - izraz > | < zvezdni - izraz > * < zaboden - izraz >

< zaboden - izraz > ::= < spremenljivka > | < številka > | + < zaboden - izraz > | (< srčni - izraz >)

< spremenljivka > ::= [a - zA - Z] +

< številka > ::= [0 - 9] +

+ ... * ... ♡



(x ♡ 2) * y * 2

c) (6 točk) Andrej je sestavil program `P`:

```
while n > 1 do
  if n mod 2 = 0 then
    n := n / 2
  else
    n := 3 * n + 1
done
```

Označite vse specifikacije, ki jim zadošča Andrejev program:

(a) `[n = 3] P [true]`

(b) `{true} P {n = 1}`

(c) `{n = 0} P {n = 1}`

(d) `[n = 0] P [n = 1]`

d) (6 točk) V λ-računu evaluiramo izraz

$$(\lambda f x\,.\,f(fx))(\lambda f\,.\,ff)(\lambda x\,.\,x)$$

Kateri izraz dobimo?

(a) $\lambda z\,.\,z$

(b) $\lambda f\,.\,ff$

(c) $\lambda x\,.\,x(xx)$

(d) izraza ne moremo evaluirati

```
A := (^ f x . f ( f x ))(^ f . f f )(^ x . x)

# A
λ x . x
```

e) (6 točk) Dan je parametrični tip

$$((\alpha \to \beta \to \gamma) \to \alpha \times \beta \to \gamma)\ \texttt{list}$$

Označite vse SML izraze, ki imajo ta tip:

(a) `[]` - : 'a list = []

(b) `fn f => fn (x, y) => f y x` - : ('a -> 'b -> 'c) -> 'b * 'a -> 'c = <fun>

(c) `(fn f => fn (x, y) => f y x) :: []` - : (('a -> 'b -> 'c) -> 'b * 'a -> 'c) list = [<fun>]

(d) `[(fn f => fn (x, y) => f x y)]` - : (('a -> 'b -> 'c) -> 'a * 'b -> 'c) list = [<fun>]

a) [n = 3] P [true]

Pomeni, da se bo pri danem predpogoju program P končal.

To drži: 3 - 10 - 5 - 16 - 8 - 4 - 2 - 0.

b) {true} P {n = 1}

Pomeni, da če se bo ukaz P končal, potem bo veljalo { n = 1 }.
Za vse predpogoje pri katerih se ukaz P konča, bo veljal končni pogoj {n = 1}.
To ni res, protiprimer je {n < 1}, npr. [n -> -7].
Protiprimer mora biti podan kot začetno stanje, iz katerega je razvidno, da specifikacija velja.

c) { n = 0} P {n = 1}

Pomeni, da če velja {n = 0} in se bo ukaz P končal, potem bo veljalo {n = 1}.
To ne drži, ker se bo program P končal (zanka se ne bo izvedla) in vrednost n bo 0.

d) [n = 0] P [n = 1]

Tudi ne velja.

2. naloga (40 točk)

Plačilno-kreditna kartica je predstavljena z naslednjimi podatki:

1. ime in priimek (neprazno zaporedje znakov dolžine največ 21)
2. vrsta kartice (debit ali credit)
3. izdajatelj (neprazno zaporedje znakov dolžine največ 16)
4. datum veljavnosti (mesec in leto)
5. številka kartice (16 števk)

Na primer, kreditna kartica asistenta Petra je predstavljena s podatki:

```
Peter Gabrovsek
Debit
Visa
4643 0400 0042 3451
05/21
```

a) (15 točk) V SML sestavite podatkovni tip `kartica`, s katerim predstavimo kartico. Tip načrtujte tako, da bo čim manj vrednosti predstavljalo neveljavne kartice, se pravi tako, da bo imela funkcija `validiraj`, ki jo boste definirali spodaj, čim manj dela.

b) (5 točk) Definirajte vrednost `asistent` tipa `kartica`, ki predstavlja kreditno kartico vašega asistenta Petra.

c) (5 točk) Definirajte vrednost `profesor` tipa `kartica`, ki predstavlja *neveljavno* kartico, se pravi tako, ki *ne* zadošča zgoraj naštetim pogojem.

d) (15 točk) V SML sestavite funkcijo

```
validiraj : kartica -> bool
```

ki preveri, ali so podatki o dani kartici veljavni, pri čemer preveri vse zgoraj naštete pogoje.

```
type vrsta =
  | Debit
  | Credit

type mesec = Jan | Feb | Mar | Apr | Maj | Jun | Jul | Avg | Sep | Okt | Nov | Dec


type stevilka = int * int * int * int * int * int * int * int *
                int * int * int * int * int * int * int * int

type kartica = {
  imePriimek : string;
  kartica : vrsta;
  izdajatelj : string;
  datumVeljavnosti : mesec * int;
  stevilkaKartice : stevilka
}

(* b *)
let asistent : kartica = {
  imePriimek = "Peter Gabrovsek";
  kartica = Debit;
  izdajatelj = "Visa";
  datumVeljavnosti = (Maj, 21);
  stevilkaKartice = (4,6,4,3,0,4,0,0,0,0,4,2,3,4,5,1)
}

(* c *)

let profesor : kartica = {
  imePriimek = "";
  kartica = Debit;
  izdajatelj = "Neveljavno ime";
  datumVeljavnosti = (Mar, 400000);
  stevilkaKartice = (4,6,4,3,0,4,0,0,0,0,4,2,3,4,5,1)
}

(* d *)

let preveriImePriimek s =
  String.length s > 0 && String.length s <= 21


let preveriIzdajatelj s =
  String.length s > 0 && String.length s <= 16

let preveriDatum (m, l) =
  l >= 0 && l < 100

let rec preveriStevilko = function
  | [] -> true
  | (x :: xs) -> (x >= 0 && x <= 9 && (preveriStevilko xs))

let to_list = function
  | (a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p) -> [a;b;c;d;e;f;g;h;i;j;k;l;m;n;o;p]

let validiraj k =
  (preveriImePriimek k.imePriimek &&
   preveriIzdajatelj k.izdajatelj &&
   preveriDatum k.datumVeljavnosti &&
   preveriStevilko (to_list k.stevilkaKartice))
```

Podatkovni tipi

Naloga: podatkovni tip e-mail

V SML definirajte podatkovni tip `email`, ki vsebuje naslednje podatke:

- pošiljatelj
- seznam naslovnikov
- datum in čas
- zadeva (angl. subject)
- vsebina sporočila

Za pošiljatelja in naslovnika uporabite niz znakov (`string`). Datum in čas predstavimo z naslednjimi podatki, ki so cela števila: leto, mesec, dan, ura. minuta, sekunda.

Zapišite izraz, ki predstavlja naslednje sporočilo:

```
From: Andrej Bauer <Andrej.Bauer@andrej.com>
To: Timotej Lazar <Timotej.Lazar@fri.uni-lj.si>, Peter Gabrovšek <Peter
Date: 2018-05-29 09:55:42
Subject: Izpit iz PPJ

Prosim, da rešitve izpita popravljata zelo strogo.

Lep pozdrav, Andrej
```

```
type date = {
  leto : int;
  mesec : int;
  dan : int;
  ura : int;
  minuta : int;
  sekunda : int
}

type email = {
  posiljatelj : string;
  seznam_naslovnikov: string list;
  datum_in_cas : date;
  zadeva : string;
  vsebina : string
}


let sporocilo = {
  posiljatelj = "Andrej Bauer <Andrej.Bauer@andrej.com>";
  seznam_naslovnikov = ["Timotej Lazar <Timotej.Lazar@fri.uni-lj.si>,
                        Peter Gabrovšek <Peter.Gabrovsek@fri.uni-lj.si>"];
  datum_in_cas = {
    leto = 2018;
    mesec = 05;
    dan = 29;
    ura = 9;
    minuta = 55;
    sekunda = 42
  };
  zadeva = "Izpit iz PPJ";
  vsebina = "Prosim, da rešitve izpita popravljata zelo strogo.

             Lep pozdrav, Andrej"
}

let rec list_to_string = function
  | [] -> ""
  | x :: xs -> x ^ (list_to_string xs)

let date_to_string d =
  (string_of_int d.leto) ^ "-" ^ (string_of_int d.mesec) ^ "-" ^ (string_of_int d.dan) ^ " " ^ (string_of_int d.ura) ^ ":" ^
(string_of_int d.minuta) ^ ":" ^ (string_of_int d.sekunda)


let izpis msg =
  String.concat "\n" [
    ("From: " ^ msg.posiljatelj);
    ("To:" ^ (list_to_string msg.seznam_naslovnikov));
    ("Date: " ^ (date_to_string msg.datum_in_cas));
    ("Subject: " ^ msg.zadeva);
    msg.vsebina
  ];;

print_string (izpis sporocilo);;
```

3. naloga (50 točk)

To nalogo rešujte v prologu. Cezarjeva šifra je starodavni sistem šifriranja sporočil, pri katerem vsako črko čistopisa zamaknemo za k mest v abecedi (črke na koncu abecede se krožno zamaknejo na začetek abecede). Na primer, če je ključ $k = 3$, se v angleški abecedi beseda "zebra" šifrira kot "cheud".

Dogovorimo se, da delamo z angleško abecedo in v ta namen definiramo predikat `abeceda/1`, ki določa vrstni red črk v angleški abecedi:

```
abeceda([a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z]).
```

Besedo v prologu predstavimo s seznamom atomov, na primer `[z,e,b,r,a]`.

Navodilo: če vam kake podnaloge ne uspe rešiti, lahko v ostalih podnalogah predpostavite, da imate njeno rešitev in delo nadaljujete po najboljših močeh.

a) (15 točk) Sestavite predikat `rotiraj/3`, kjer `rotiraj(K,A,B)` pomeni, da dobimo seznam `B` tako, da seznam `A` krožno zamaknemo za `K` mest. Primer:

```
?- rotiraj(2, [l,j,u,b,l,j,a,n,a], B).
B = [u,b,l,j,a,n,a,l,j]

?- abeceda(A), rotiraj(3, A, B).
A = [a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z],
B = [d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,a,b,c].
```

b) (15 točk) Sestavite predikat `preslikaj/4`, kjer `preslikaj(A,B,X,Y)` pomeni, da v seznamu `A` poiščemo element `X` in vrnemo istoležni element `Y` v seznamu `B`. Predpostavite lahko, da sta seznama `A` in `B` podana, enako dolga in sestavljena iz različnih atomov. Primer:

```
?- preslikaj([a,b,c], [c,a,b], c, Y).
Y = b.

?- preslikaj([a,b,c,d,e,f], [u,v,w,x,y,z], d, Y).
Y = x.
```

c) (10 točk) Sestavite predikat `cezar/3`, kjer `cezar(K,In,Out)` pomeni, da dobimo `Out`, ko `In` šifriramo s Cezarjevo šifro z zamikom `K`. Primer:

```
?- cezar(3, [z,e,b,r,a], Out).
Out = [c,h,e,u,d]

?- cezar(12, In, [x,v,g,n,x,v,m,z,m]).
In = [l,j,u,b,l,j,a,n,a]
```

d) (10 točk) Peter je Timoteju poslal šifrirano sporočilo "ypfyjzufhubqxua". Dešifrirajte ga! Iz vaše rešitve naj bo razvidno, kako ste uporabili prolog pri postopku reševanja.

a)

```
abeceda([a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z]).

rotiraj(K, A, A) :-
    K #=< 0.

rotiraj(K, [H | T], B) :-
    K #> 0,
    K1 #= K - 1,
    append(T, [H], S),
    rotiraj(K1, S, B).
```

b)

```
preslikaj([X |_], [Y | _], X, Y).

preslikaj([_ | T1], [_ | T2], X, Y) :-
    preslikaj(T1, T2, X, Y).
```

c)

```
cezar(K, In, Out) :-
    abeceda(A),
    rotiraj(K, A, B),
    maplist(preslikaj(A, B), In, Out).
```

d)

```
K in 0..25, label([K]), Out = [y,p,f,y,j,z,u,f,h,u,b,q,x,u,a], cezar(K, In, Out).
```

Principi programskih jezikov
2. izpit, 24. junij 2021

1	
2	
3	
Σ	

Ime in priimek

Vpisna številka

NAVODILA

- **Ne odpirajte te pole**, dokler ne dobite dovoljenja.
- **Preden začnete reševati test:**
  - Vpišite svoje podatke na testno polo z velikimi tiskanimi črkami.
  - Na vidno mesto položite osebni dokument s sliko in študentsko izkaznico.
  - Preverite, da imate mobitel izklopljen in spravljen v torbi.
- Dovoljeni pripomočki: pisalo, brisalo, in poljubno pisno gradivo.
- Vse rešitve vpisujte v polo.
- Če kaj potrebujete, prosite asistenta, ne sosedov.
- **Med izpitom ne zapuščajte svojega mesta** brez dovoljenja.
- Testna pola vam bo odvzeta **brez nadaljnjih opozoril**, če:
  - komunicirate s komerkoli, razen z asistentom,
  - komu podate kak predmet ali list papirja,
  - odrinete svoje gradivo, da ga lahko vidi kdo drug,
  - na kak drug način prepisujete ali pomagate komu prepisovati,
  - imate na vidnem mestu mobitel ali druge elektronske naprave.
- **Ob koncu izpita:**
  - Ko asistent razglasi konec izpita, **takoj** nehajte in zaprite testno polo.
  - **Ne vstajajte**, ampak počakajte, da asistent pobere **vse** testne pole.
  - **Testno polo morate nujno oddati.**
- Čas pisanja je 120 minut. Na vidnem mestu je zapisano, do kdaj imate čas.
- Predvideni ocenjevalni kriterij:
  1. ≥ 90 točk, ocena 10
  2. ≥ 80 točk, ocena 9
  3. ≥ 70 točk, ocena 8
  4. ≥ 60 točk, ocena 7
  5. ≥ 50 točk, ocena 6

  Veliko uspeha!

1. naloga (35 točk)

a) (7 točk) Stara elbonijska vraža pravi, da ima tri leta nesreče, kdor sešteje tri števila na en mah. Elbonijska aritmetika zato dopušča samo dve zaporedni seštevanji (oklepajev ne poznajo):

$$\begin{aligned}
\langle \text{vsota} \rangle &::= \langle \text{vsotica} \rangle \mid \langle \text{vsotica} \rangle + \langle \text{zmnožek} \rangle \\
\langle \text{vsotica} \rangle &::= \langle \text{zmnožek} \rangle \mid \langle \text{zmnožek} \rangle + \langle \text{zmnožek} \rangle \\
\langle \text{zmnožek} \rangle &::= \langle \text{število} \rangle \mid \langle \text{zmnožek} \rangle \times \langle \text{število} \rangle \\
\langle \text{število} \rangle &::= [0\text{-}9]^+
\end{aligned}$$

V sosednji Severni Elboniji pa velja, da ima štiri leta nesreče, kdor sešteje štiri števila na en mah. Zapišite slovnična pravila za severno-elbonijsko aritmetiko, ki je podobna elbonijski, le da dopušča samo ***dve*** in ***tri*** zaporedna seštevanja:

```
<vsotica> ::= <zmnožek> | <zmnožek> + <zmnožek> | <zmnožek> + <zmnožek> + <zmnožek>
```

b) (7 točk) Napišite *poizvedbo* v prologu, ki preveri, ali obstaja seznam $[x_1, x_2, \ldots, x_{42}]$ dolžine 42, ki je enak seznamu $[x_3, \ldots, x_{42}, x_1, x_2]$, ki ga dobimo, ko prestavimo prva dva elementa na konec. Priporočamo uporabo predikatov `length` in `append`.

```
rotiraj(K, A, A) :-
    K #=< 0.

rotiraj(K, [H | T], B) :-
    K #> 0,
    K1 #= K - 1,
    append(T, [H], S),
    rotiraj(K1, S, B).
```

```
?- length(A, 42), rotiraj(2, A, B).
```

c) (7 točk) Izpeljite *glavni tip* funkcije `f`, ki je v OCamlu definirana kot

```
type order = Less | Greater | Equal
let f (x, y, z) = function Less -> x | Equal -> y | Greater -> z
```

Odgovor:

```
type order = Less | Greater | Equal
val f : 'a * 'a * 'a -> order -> 'a = <fun>
```

d) (7 točk) Ko je bil Klemen v vrtcu, je že znal programirati v ukaznem programskem jeziku. Za računanje kvadratnih korenov je spisal program `P`:

```
k := 0 ;
while k * k ≠ n do
  k := k + 1;
done ;
```

Katere od naslednjih specifikacij veljajo?

$[n > 0]$	P	[true]	DA	NE
$[n > 0]$	P	[false]	DA	NE
$\{n > 0\}$	P	$\{k^2 = n\}$	DA	NE
$[n > 0]$	P	$[k^2 = n]$	DA	NE
$[n > 0]$	P	$[k^2 \geq n]$	DA	NE

e) (7 točk) Zapišite *kakršenkoli* modul `A`, ki ustreza signaturi

```
module type CHANNEL =
sig
  type t
  val init : unit -> t
  val write : t -> string -> unit
  val read : t -> string
end
```

Odgovor:

```
module A : CHANNEL =
struct

  type t = PPJ
  let init _ = PPJ
  let write _ _ = ()
  let read _ = "PPJ"

end
```

2. naloga (25 točk)

Dokažite *polno* pravilnost programa.

```
[ y ≤ z ]

if x ≤ y then

  x := y

else

  skip

end ;

if x ≥ z then

  x := z

else

  skip

end

[ y ≤ x ≤ z ]
```

[y ≤ z]

if x ≤ y then

  [y ≤ z, x ≤ y] <==> [x ≤ y ≤ z]

  x := y

  [x = y ≤ z]

else

  [y ≤ z, x > y] <==> [x > y ≤ z]

  skip

  [x > y ≤ z]

end ;
  [y ≤ z, x > y, x = y]

if x ≥ z then

  [y ≤ z, x ≥ z]

  x := z

  [y ≤ z = x]

else

  [y ≤ z, x < z]

skip

  [y ≤ z, x < z]

end

  [y ≤ z, x < z, x = z]

[y ≤ x ≤ z]

3. naloga (25 točk)

V davnih časih so imeli mobiteli tipkovnice, na katerih so bile številke in črke:

Ker je bilo na vsaki tipki več črk, iz zaporedja pritiskov ni bilo vedno možno razbrati, katero besedo je natipkal uporabnik. Na primer, 6 7 3 5 lahko pomeni katerokoli od 81 štiričrkovnh besed:

mpdj, mpdk, mpdl, . . . , osfj, osfk, osfl.

V pomoč uporabniku so telefoni vsebovali spisek veljavnih besed. Ko je uporabnik natipkal zaporedje številk, so se prikazale samo veljavne besede iz slovarja. Na primer, v zgornjem primeru bi se namesto vseh 81 možnosti prikazale besede "orel", "osel" in "opel".

a) (10 točk) V prologu sestavite predikat `preslikaj(Beseda,Stevke)`, ki velja, kadar `Beseda` natipkamo s zaporedjem stevk `Stevke`. V pomoč vam je predikat `tipke`:

```
tipke(2, [a,b,c]).
tipke(3, [d,e,f]).
tipke(4, [g,h,i]).
tipke(5, [j,k,l]).
tipke(6, [m,n,o]).
tipke(7, [p,q,r,s]).
tipke(8, [t,u,v]).
tipke(9, [w,x,y,z]).
```

Primer uporabe:

```
?- preslikaj([o,r,e,l], Stevke).
Stevke = [6, 7, 3, 5] ;
false.
```

```
preslikaj([], []).

preslikaj([B | Besede], [S | Stevke]) :-
   tipke(S, BS),
   member(B, BS),
   preslikaj(Besede, Stevke).
```

b) (15 točk) Sestavite predikat `moznosti(Veljavne,Stevke,Beseda)`, ki velja, kadar je `Stevke` zaporedje števk, in je `Beseda` beseda s seznama `Veljavne`, ki bi jo lahko dobili s `Stevke`.

```
?- moznosti([[o,r,e,l],[o,p,i,c,a],[o,s,e,l],[r,i,b,a]], [6,7,3,5], Beseda).
Beseda = [o, r, e, l] ;
Beseda = [o, s, e, l] ;
false.

?- moznosti([[i,z,p,i,t],[j,e],[l,a,h,e,k]], Stevke, Beseda).
Stevke = [4, 9, 7, 4, 8],
Beseda = [i, z, p, i, t] ;
Stevke = [5, 3],
Beseda = [j, e] ;
Stevke = [5, 2, 4, 3, 5],
Beseda = [l, a, h, e, k] ;
false.
```

```
moznosti([], _, _) :- false.
moznosti([V | O], S, M) :-
    preslikaj(V, N), M = V, S = N;
    moznosti(O, S, M).
```

4. naloga (25 točk)

V OCamlu sestavite še funkcijo

```
moznosti : char list list -> int list -> char list list
```

ki deluje podobno kot predikat `moznosti` iz prejšnje naloge. Funkcija sprejme seznam veljavnih besed `besede` in zaporedje števk `stevke` ter vrne seznam tistih besed iz `besede`, ki bi jih lahko dobili s števkami `stevke`. V pomoč naj vam bo asociativni seznam

```
let tipke : (char * int) list = [
  ('a',2); ('b',2); ('c',2);
  ('d',3); ('e',3); ('f',3);
  ('g',4); ('h',4); ('i',4);
  ('j',5); ('k',5); ('l',5);
  ('m',6); ('n',6); ('o',6);
  ('p',7); ('q',7); ('r',7); ('s',7);
  ('t',8); ('u',8); ('v',8);
  ('w',9); ('x',9); ('y',9); ('z',9)]
```

Primer uporabe:

```
# moznosti
   [['o';'r';'e';'l'];['o';'p';'i';'c';'a'];['o';'s';'e';'l'];['r';'i';'b';'a']]
   [6;7;3;5] ;;
- : char list list = [['o'; 'r'; 'e'; 'l']; ['o'; 's'; 'e'; 'l']]
```

Programirate lahko tudi v Haskellu. *(Naslednja stran je prazna.)*

```
let tipke : (char * int) list = [
  ('a',2); ('b',2); ('c',2);
  ('d',3); ('e',3); ('f',3);
  ('g',4); ('h',4); ('i',4);
  ('j',5); ('k',5); ('l',5);
  ('m',6); ('n',6); ('o',6);
  ('p',7); ('q',7); ('r',7); ('s',7);
  ('t',8); ('u',8); ('v',8);
  ('w',9); ('x',9); ('y',9); ('z',9)] ;;

let rec find crka tipka = function
  | [] -> false
  | (c, st) :: tail -> if crka = c && st = tipka then true else find crka tipka tail

let rec can_spell beseda vhodne_tipke =
  match beseda with
  | [] -> true
  | crka :: p_besede -> match vhodne_tipke with
    | [] -> false
    | tipka :: p_tipk -> if find crka tipka tipke then can_spell p_besede p_tipk else false

let moznosti besede vhodne_tipke =
  let rec moz_acc acc = function
    | [] -> acc
    | beseda :: tail -> if can_spell beseda vhodne_tipke
        then moz_acc (beseda :: acc) tail
        else moz_acc acc tail
  in List.rev(moz_acc [] besede)
```

Principi programskih jezikov, 2. izpit
28. junij 2019

								Σ	

Ime in priimek

Vpisna številka

NAVODILA

- **Ne odpirajte te pole**, dokler ne dobite dovoljenja.
- **Preden začnete reševati test:**
  - Vpišite svoje podatke na testno polo z velikimi tiskanimi črkami.
  - Na vidno mesto položite osebni dokument s sliko in študentsko izkaznico.
  - Preverite, da imate mobitel izklopljen in spravljen v torbi.
  - Prjavite se na spletno učilnico, kamor boste oddajali odgovore.
- Dovoljeni pripomočki: pisalo, brisalo, USB ključ in poljubno pisno gradivo.
- Vse rešitve vpisujte v kviz na spletni učilnici.
- Če kaj potrebujete, prosite asistenta, ne sosedov.
- **Med izpitom ne zapuščajte svojega mesta** brez dovoljenja.
- Testna pola vam bo odvzeta **brez nadaljnjih opozoril**, če:
  - komunicirate s komerkoli, razen z asistentom,
  - komu podate kak predmet ali list papirja,
  - odrinete svoje gradivo, da ga lahko vidi kdo drug,
  - na kak drug način prepisujete ali pomagate komu prepisovati,
  - imate na vidnem mestu mobitel ali druge elektronske naprave.
- **Ob koncu izpita:**
  - Ko asistent razglasi konec izpita, **takoj** nehajte in zaprite testno polo.
  - **Ne vstajajte**, ampak počakajte, da asistent pobere **vse** testne pole.
  - **Testno polo morate nujno oddati.**
- Čas pisanja je 120 minut. Na tabli je zapisano, do kdaj imate čas.
- Predvideni ocenjevalni kriterij:

  ≥ 90 točk, ocena 10
  ≥ 80 točk, ocena 9
  ≥ 70 točk, ocena 8
  ≥ 60 točk, ocena 7
  ≥ 50 točk, ocena 6

Veliko uspeha!

1. naloga (30 točk)

a) (6 točk) Elbonija uporablja naslednjo sintakso za zapis aritmetičnih izrazov:

$$\langle \text{izraz} \rangle ::= \langle \text{številka} \rangle \mid \ominus \langle \text{izraz} \rangle \mid \langle \text{izraz} \rangle \oplus \langle \text{izraz} \rangle \mid \langle \text{izraz} \rangle \otimes \langle \text{izraz} \rangle$$
$$\langle \text{številka} \rangle ::= [0\text{–}9]+$$

Simboli $\ominus$, $\oplus$ in $\otimes$ označujejo nasprotno vrednost, seštevanje in množenje. Elbonijsko razumevanje aritmetike je pomanjkljivo, zato nimajo nikakršnih dogovorov o prioriteti in asociativnosti operacij, prav tako pa ne poznajo oklepajev. Na primer, elbonijski izraz $2 \otimes 3 \oplus 4$ bi po naše lahko razumeli kot $(2 \cdot 3) + 4$ ali kot $2 \cdot (3 + 4)$.

Katere so možne vrednosti izraza $\ominus 2 \oplus 3 \otimes 4$?

(i) 4 in 10

(ii) -20, -14, 4 in 10

(iii) -20, -14 in 10

(iv) 10

b) (6 točk) V okolju $[n \mapsto 10, i \mapsto 0, v \mapsto 0]$ evaluiramo program

```
v := 0 ;
while i < n do
  v := v + i ;
  i := i + 1
done
```

Kakšno bo okolje, ko bo program končal?

(i) $[n \mapsto 10, i \mapsto 9, v \mapsto 45]$

(ii) $[n \mapsto 10, i \mapsto 10, v \mapsto 45]$

(iii) $[n \mapsto 10, i \mapsto 10, v \mapsto 55]$

(iv) $[n \mapsto 10, i \mapsto 11, v \mapsto 55]$

```
int main()
{

   int v = 0;
   int n = 10;
   int i = 0;
   while (i < n) {
      v = v + i;
      i = i + 1;
   }
   printf("%d %d %d", n, i, v);

   return 0;
}

10 10 45
```

c) (6 točk) Definiramo λ-izraze

$$I = \lambda x \,.\, x, \qquad K = \lambda x\, y \,.\, x, \qquad L = \lambda x\, y \,.\, y.$$

Katere od naslednjih enakosti veljajo? (Napačno izbrani odgovori štejejo -2 točki.)

(i) $K\,I\,L = L\,L\,L\,L$

(ii) $K\,I\,L\,L = L\,L\,L$

(iii) $K\,I\,L\,L\,L = L\,L$

(iv) $K\,I\,L\,L\,L\,L = L$

```
I := ^ x . x ;
K := ^ x y . x ;
L := ^ x y . y
```

```
# K I L
λ x . x

# L L L L
λ x . x
```

```
# K I L L
λ _ y . y

# L L L
λ _ y . y
```

```
# K I L L L
λ y . y

# L L
λ y . y
```

```
# K I L L L L
λ _ y . y

# L
λ _ y . y
```

d) (6 točk) Kateri od naslednjih prolog programov je ekvivalenten formuli $(a \vee b) \Rightarrow (c \wedge d)$?

(i)
```
c :- a, b.
d :- a, b.
```

(ii)
```
c :- a; b.
d :- a; b.
```

(iii)
```
a :- c; d.
b :- c; d.
```

(iv)
```
a :- c, d.
b :- c, d.
```

e) (6 točk) V turistični agenciji *Bratko Travels* uporabljajo prolog. Letalske povezave med letališči predstavijo z relacijo `polet(X,Y,T)`, ki pomeni, da obstaja polet met letališčema `X` in `Y`, ki traja `T` ur. Primer podatkov, ki jih hranijo:

```
polet(ljubljana, frankfurt, 1).
polet(ljubljana, amsterdam, 2).
polet(ljubljana, london, 2).
polet(frankfurt, ljubljana, 1).
polet(frankfurt, london, 2).
polet(frankfurt, amsterdam, 1).
polet(frankfurt, newyork, 9).
polet(amsterdam, ljubljana, 2).
polet(amsterdam, frankfurt, 1).
polet(amsterdam, newyork, 6).
polet(london, ljubljana, 2).
polet(london, frankfurt, 2).
polet(london, newyork, 6).
polet(newyork, frankfurt, 9).
polet(newyork, amsterdam, 6).
polet(newyork, london, 6).
```

Zapišite poizvedbo v prologu, s katero ugotovimo, ali je možno leteti od Ljubljane do New Yorka z natanko enim prestopanjem tako, da je skupni čas letenja manjši od 9 ur.

```
polet(ljubljana, X, T1), polet(X, newyork, T2), T1 + T2 < 9.
X = amsterdam,
T1 = 2,
T2 = 6 ;
X = london,
T1 = 2,
T2 = 6.
```

2. naloga (35 točk)

Dokažite *popolno* pravilnost programa, kjer je n pozitivno celo število:

```
[ 0 < n ]
k := 1 ;
while k * k < n do
  k := k + 1
done
[ (k − 1)^2 < n ≤ k^2 ]
```

Opomba: delna pravilnost je vredna 20 točk, dokaz zaustavitve programa pa 15 točk.

DELNA PRAVILNOST

I: (k - 1)^2 < n

[0 < n]
k := 1 ;

{ n > 0, k = 1} => (k - 1)^2 < n

while k * k < n do

{ k^2 < n, (k - 1)^ 2 < n }

{ (k + 1 - 1)^2 < n, (k + 1 - 1 - 1)^ 2 < n }

k := k + 1

{ (k - 1)^2 < n, (k - 2)^ 2 < n } =>

{ (k - 1)^2 < n }

done

{ (k - 1)^2 < n, k^2 < n} =>

[(k - 1)^2 < n ≤ k^2]

(k - 1)^2 < k^2 < n

POPOLNA PRAVILNOST

z = n - (k - 1)^2
n - (k - 1)^2 ≥ 0

[z = n - (k - 1)^2, n - (k - 1)^2 ≥ 0]

while k * k < n do

[z = n - (k - 1)^2, n - (k - 1)^2 ≥ 0, k^2 < n]

[z = n - (k + 1 - 1 - 1)^2, n - (k + 1 - 1 - 1)^2 ≥ 0, (k + 1 - 1)^2 < n]

k := k + 1

[z = n - (k - 2)^2, n - (k - 2)^2 ≥ 0, (k - 1)^2 < n]

[z > n - (k - 1)^2 ≥ 0, (k - 1)^2 < n]

done

3. naloga (35 točk)

Obravnavajmo aritmetične izraze s konstantami, spremenljivkami in seštevanjem:

$$\begin{aligned}\langle\text{expression}\rangle &::= \langle\text{number}\rangle \mid \langle\text{variable}\rangle \mid \langle\text{expression}\rangle + \langle\text{expression}\rangle\\ \langle\text{number}\rangle &::= [0\text{–}9]+\\ \langle\text{variables}\rangle &::= [a\text{–}z]+\end{aligned}$$

V OCamlu definiramo podatkovni tip, ki predstavlja abstraktno sintakso izrazov, in funkcijo

```
eval : (string * int) list -> expression -> int
```

ki v danem okolju evaluira dani izraz:

```
type expression =
  | Number of int
  | Var of string
  | Plus of expression * expression

let rec eval env = function
  | Number k -> k
  | Var x -> List.assoc x env
  | Plus (e1, e2) -> eval env e1 + eval env e2
```

a) (5 točk) V OCamlu definirajte vrednost `izraz`, ki predstavlja izraz $3 + (x + 5)$, nato pa z uporabo funkcije `eval` izračunajte njegovo vrednost v okolju $[(x, 7), (y, 2)]$:

```
let izraz = ···
let vrednost = eval ···
```

b) (30 točk) Če upoštevamo, da je seštevanje asociativno in komutativno, lahko izraze optimiziramo tako, da vse celoštevilske konstante zberemo skupaj. Na primer:

$$\begin{aligned}(2 + 3) + 1 &= 6\\ (x + 5) + (y + (x + 2)) &= 7 + x + y + x\\ 0 + x &= x\end{aligned}$$

Vrstni red spremenljivk smo ohranili, prav tako nismo združili dveh x v $2 \cdot x$, ker nimamo množenja. Kako asociramo rezultat, ni pomembno.

Sestavite funkcijo

```
optimize : expression -> expression
```

ki sprejme izraz in ga optimizira, kot je prikazano zgoraj. Primeri uporabe:

```
# optimize (Plus (Number 2, Plus (Number 3, Number 1))) ;;
- : expression = Number 6
# optimize (Plus (Plus (Var "x", Number 5),
                  Plus (Var "y", Plus (Var "x", Number 2)))) ;;
- : expression = Plus (Plus (Plus (Number 7, Var "x"), Var "y"), Var "x")
# optimize (Plus (Number 0, Var "x")) ;;
- : expression = Var "x"
```

Namig: profesor Bauer je nalogo rešil tako, da je definiral dve pomožni funkciji: prva iz danega izraza izračuna celoštevilsko konstanto in seznam spremenljivk, ki se pojavijo v njem; druga iz celoštevilske konstante in seznama spremenljivk sestavi izraz.

```
type expression =
  | Number of int
  | Var of string
  | Plus of expression * expression

let rec eval env = function
  | Number k -> k
  | Var x -> List.assoc x env
  | Plus (e1, e2) -> (eval env e1) + (eval env e2)

let izraz = Plus(Number(3), Plus(Var("x"), Number(5)))
let izraz2 = Plus(Plus(Var("x"), Number(5)), Plus(Var("y"), Plus(Var("x"), Number(2))))
let izraz3 = Plus(Var("x"), Number(0))
let izraz4 = Plus(Number(3), Plus(Number(4), Number(2)))

let vrednost = eval [("x",7); ("y",2)] izraz

let rec sum num exp = match exp with
  | Number n -> n
  | Var v -> 0
  | Plus (l, r) -> (num + ((sum num l) + (sum num r)))

let rec flatten vars exp = match exp with
  | Number n -> []
  | Var v -> [v]
  | Plus (l, r) -> (vars @ (flatten vars l) @ (flatten vars r))

let optimize exp =
  let vars = flatten [] exp in
  let cons = sum 0 exp in
  let rec temp l = match l with
    | [] -> Number(cons)
    | [y] -> if cons > 0 then Plus(Var(y), Number(cons)) else Var(y)
    | x::xs -> Plus(Var(x), (temp xs)) in
  temp vars
```

Principi programskih jezikov
1. izpit, 5. junij 2020

1	
2	
3	
Σ	

Ime in priimek

Vpisna številka

NAVODILA

- **Ne odpirajte te pole**, dokler ne dobite dovoljenja.
- **Preden začnete reševati test:**
  - Vpišite svoje podatke na testno polo z velikimi tiskanimi črkami.
  - Na vidno mesto položite osebni dokument s sliko in študentsko izkaznico.
  - Preverite, da imate mobitel izklopljen in spravljen v torbi.
- Dovoljeni pripomočki: pisalo, brisalo, in poljubno pisno gradivo.
- Vse rešitve vpisujte v polo.
- Če kaj potrebujete, prosite asistenta, ne sosedov.
- **Med izpitom ne zapuščajte svojega mesta** brez dovoljenja.
- Testna pola vam bo odvzeta **brez nadaljnjih opozoril**, če:
  - komunicirate s komerkoli, razen z asistentom,
  - komu podate kak predmet ali list papirja,
  - odrinete svoje gradivo, da ga lahko vidi kdo drug,
  - na kak drug način prepisujete ali pomagate komu prepisovati,
  - imate na vidnem mestu mobitel ali druge elektronske naprave.
- **Ob koncu izpita:**
  - Ko asistent razglasi konec izpita, **takoj** nehajte in zaprite testno polo.
  - **Ne vstajajte**, ampak počakajte, da asistent pobere **vse** testne pole.
  - **Testno polo morate nujno oddati.**
- Čas pisanja je 120 minut. Na vidnem mestu je zapisano, do kdaj imate čas.
- Predvideni ocenjevalni kriterij:
  1. ≥ 90 točk, ocena 10
  2. ≥ 80 točk, ocena 9
  3. ≥ 70 točk, ocena 8
  4. ≥ 60 točk, ocena 7
  5. ≥ 50 točk, ocena 6

  Veliko uspeha!

1. naloga (30 točk)

a) (6 točk) V Elbonji varčujejo črnilo, zato znaka za množenje ne pišejo, ampak namesto njega pustijo presledek. Operacijo seštevanja označijo s piko $\bullet$. V ta namen uporabljajo naslednjo sintakso za zapis aritmetičnih izrazov, kjer ␣ označuje presledek:

$$\begin{aligned}
\langle\text{izraz}\rangle &::= \langle\text{multiplikativni}\rangle \mid \langle\text{izraz}\rangle \bullet \langle\text{multiplikativni}\rangle \\
\langle\text{multiplikativni}\rangle &::= \langle\text{osnovni}\rangle \mid \langle\text{osnovni}\rangle\text{␣}\langle\text{multiplikativni}\rangle \\
\langle\text{osnovni}\rangle &::= (\langle\text{izraz}\rangle) \mid \langle\text{število}\rangle \\
\langle\text{število}\rangle &::= [0\text{–}9]+
\end{aligned}$$

Narišite sintaktično drevo za izraz $20\ (4 \bullet 2)\ 1 \bullet 3 \bullet (19\ 20)$.

b) (6 točk) V λ-računu predstavimo števila s Churchovimi numerali, na primer,

$$\begin{aligned}
0 &:= \lambda f\, x\, .\, x, \\
1 &:= \lambda f\, x\, .\, fx, \\
2 &:= \lambda f\, x\, .\, f(fx), \\
3 &:= \lambda f\, x\, .\, f(f(fx)).
\end{aligned}$$

Katero število je $\lambda g\, y\, .\, 3\ 2\ g\ y$?

```
:deep ;
:eager ;
0 := ^ f x . x ;
1 := ^ f x . f x ;
2 := ^ f x . f ( f x ) ;
3 := ^ f x . f ( f ( f x )) ;
^ g y . 3 2 g y

λ g y . g (g (g (g (g (g (g y)))))))

Število 8.
```

Odgovor: število _____

Znak za množenje: —

Znak za seštevanje : •

<izraz> ::= <multiplikativni> | <izraz> • <multiplikativni> // LEVO

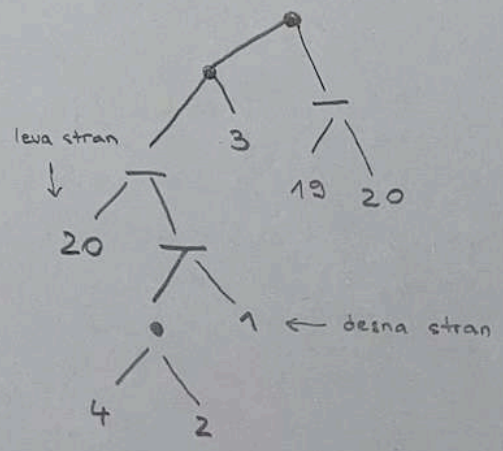
<multiplikativni> ::= <osnovni> | <osnovni> _ <multiplikativni> // DESNO

<osnovni> ::= (<izraz>) | <število>

<število> ::= [0-9]+

PRIORITETA : —, •

$$(((20\_(4 \bullet 2)\_1)) \bullet 3) \bullet (19\_20)$$



c) (6 točk) Klemen je v OCamlu definiral vrednost `prod`:

```
let prod =
  let rec loop acc = function
    | [] -> acc
    | x :: xs -> loop (x * acc) xs
  in
  loop 1
;;
```

Kakšen tip ima `prod`? _____

val prod : int list -> int = <fun>

d) (6 točk) Andrej je v OCamlu definiral podatkovni tip

```
type 'a trie = Node of 'a | Trie of ('a trie) list
```

1. zapišite vrednost tipa `int trie`: Node 3
2. zapišite vrednost tipa `'a trie`: Trie [ ]
3. zapišite vrednost, ki je različna od prejšnjih dveh: Node 4

e) (6 točk) V programskem jeziku z zapisi in podtipi je Marcel definiral tipe

$$\begin{aligned} \tau &= \{a : \mathsf{bool}\} \to \{u : \mathsf{bool}\} \\ \sigma &= \{a : \mathsf{bool}, u : \mathsf{bool}\} \\ \rho &= \{a : \mathsf{bool}, b : \mathsf{bool}\} \to \{u : \mathsf{bool}, v : \mathsf{bool}\} \end{aligned}$$

Označite pravilne trditve, kjer $\leq$ pomeni "podtip (po vrstnem redu, globini in širini)":

(a) $\tau \leq \sigma$

(b) $\sigma \leq \tau$

(c) $\tau \leq \rho$

(d) $\rho \leq \tau$

o je zapis, druge pa funkcije !

2. naloga (40 točk)

a) (30 točk) Dokažite *delno* pravilnost programa:

$\{a > 0 \wedge b > 0\}$

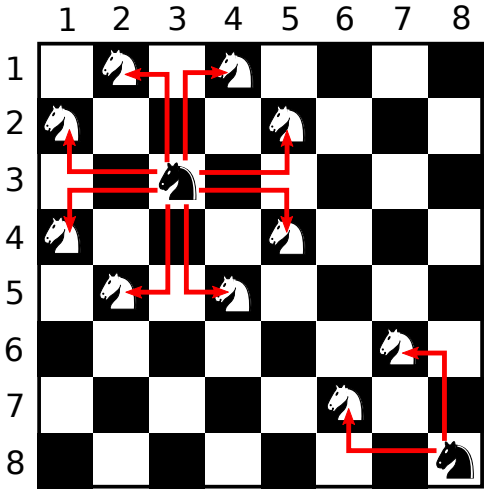
```
if b = 1 then
  k := a
else
  k := 0 ;
  while b * k < a do
    k := k + 1
  done
end
```

$\{a - b \cdot k < b\}$

b) (10 točk) Dokažite še *popolno* pravilnost zgornjega programa.

3. naloga (40 točk)

Polja na šahovnici velikosti 8×8 označimo s koordinatami (x, y), kjer velja $1 \leq x \leq 8$ in $1 \leq y \leq 8$, glej sliko, ki prikazuje tudi možne poteze šahovskega skakača.



Polje $(\texttt{X}, \texttt{Y})$ v prologu zapišemo z izrazom `X/Y`, saj prolog nima urejenih parov.

a) (5 točk) Sestavite predikat `polje(X/Y)`, ki velja natanko tedaj, ko sta `X` in `Y` veljavni koordinati:

```
?- polje(9/2).
false.
?- polje(X/Y).
X = Y, Y = 1 ;
...
```

Poskrbite, da poizvedba `polje(X/Y)` vrne vseh 64 odgovorov. Uporabiti smete programiranje z omejitvami ali kak drug pristop.

```
:-use_module(library(clpfd)).
polje(X/Y) :-
    [X, Y] ins 1..8,
    label([X, Y]).
```

b) (10 točk) Sestavite predikat `premik(P,Q)`, ki velja natanko tedaj, ko sta `P` in `Q` veljavni polji in se lahko skakač premakne s polja `P` na polje `Q`.

```
?- premik(5/8, Q).
Q = 3/7 ;
Q = 4/6 ;
Q = 6/6 ;
Q = 7/7 ;
false.
?- premik(1/1, 2/2).
false.
?- premik(1/0, Q).
false.
```

```
premik(P, Q) :-
    polje(P),
    polje(Q),
    P = X1/Y1, Q = X2/Y2,
    (1 is abs(X1 - X2), 2 is abs(Y1 - Y2);
    (2 is abs(Y1 - Y2), 1 is abs(X1 - X2)
).
```

c) (15 točk) Sestavite predikat `sprehod(L)`, ki velja natanko tedaj, ko je `L` seznam veljavnih polj in za vsaki *zaporedni* polji `P` in `Q` v seznamu `L` velja `premik(P,Q)`. Polja v sprehodu se smejo ponavljati.

```
?- sprehod([4/4, P, 5/5]).
P = 3/6 ;
P = 6/3 ;
false.
?- sprehod([4/4, 4/5]).
false.
?- sprehod(L).
L = [] ;
L = [1/1] ;
L = [1/2] ;
...
```

```
sprehod([]).
sprehod([H1, H2 | L]) :-
    premik(H1, H2),
    sprehod([H2 | L]).
```

d) (10 točk) Zapišite poizvedbo, ki ugotovi, ali obstaja sprehod dolžine 64 od polja `1/1` do polja `8/8`.

```
length(64, L), L = [1/1 | _], append(_, [8/8], L), sprehod(L).
```

Principi programskih jezikov
2. izpit, 24. junij 2020

1	
2	
3	
Σ	

Ime in priimek

Vpisna številka

NAVODILA

- **Ne odpirajte te pole**, dokler ne dobite dovoljenja.
- **Preden začnete reševati test:**
  - Vpišite svoje podatke na testno polo z velikimi tiskanimi črkami.
  - Na vidno mesto položite osebni dokument s sliko in študentsko izkaznico.
  - Preverite, da imate mobitel izklopljen in spravljen v torbi.
- Dovoljeni pripomočki: pisalo, brisalo, in poljubno pisno gradivo.
- Vse rešitve vpisujte v polo.
- Če kaj potrebujete, prosite asistenta, ne sosedov.
- **Med izpitom ne zapuščajte svojega mesta** brez dovoljenja.
- Testna pola vam bo odvzeta **brez nadaljnjih opozoril**, če:
  - komunicirate s komerkoli, razen z asistentom,
  - komu podate kak predmet ali list papirja,
  - odrinete svoje gradivo, da ga lahko vidi kdo drug,
  - na kak drug način prepisujete ali pomagate komu prepisovati,
  - imate na vidnem mestu mobitel ali druge elektronske naprave.
- **Ob koncu izpita:**
  - Ko asistent razglasi konec izpita, **takoj** nehajte in zaprite testno polo.
  - **Ne vstajajte**, ampak počakajte, da asistent pobere **vse** testne pole.
  - **Testno polo morate nujno oddati.**
- Čas pisanja je 120 minut. Na vidnem mestu je zapisano, do kdaj imate čas.
- Predvideni ocenjevalni kriterij:
  1. ≥ 90 točk, ocena 10
  2. ≥ 80 točk, ocena 9
  3. ≥ 70 točk, ocena 8
  4. ≥ 60 točk, ocena 7
  5. ≥ 50 točk, ocena 6

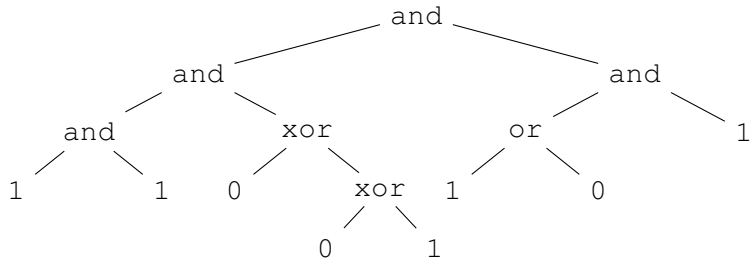
  Veliko uspeha!

1. naloga (25 točk)

a) (6 točk) V Elbonji za zapis logičnih izrazov uporabljajo samo operatorje *in* (`and`), *ali* (`or`) in *ekskluzivni ali* (`xor`). V ta namen uporabljajo naslednjo sintakso:

$$\begin{aligned}
\langle\text{izraz}\rangle &::= \langle\text{ekskluzivni}\rangle \mid \langle\text{ekskluzivni}\rangle \texttt{ or } \langle\text{izraz}\rangle \\
\langle\text{ekskluzivni}\rangle &::= \langle\text{konjuktivni}\rangle \mid \langle\text{konjuktivni}\rangle \texttt{ xor } \langle\text{eksluzivni}\rangle \\
\langle\text{konjuktivni}\rangle &::= \langle\text{osnovni}\rangle \mid \langle\text{osnovni}\rangle \texttt{ and } \langle\text{konjuktivni}\rangle \\
\langle\text{osnovni}\rangle &::= \texttt{(}\ \langle\text{izraz}\rangle\ \texttt{)} \mid \texttt{0} \mid \texttt{1}
\end{aligned}$$

Zapišite izraz **brez nepotrebnih oklepajev**, ki predstavlja sintaktično drevo



Odgovor: _____

((1 and 1) and (0 xor (0 xor 1))) and ((1 or 0) and 1)

b) (6 točk) V λ-računu smo definirali izraz $A := (\lambda x\,.\,\lambda y\,.\,x\,y)\,y$. Izračunajte izraz $A\,A$ do konca in označite pravilni odgovor:

(a) $y\,(\lambda z\,.\,y\,z)$

(b) $(\lambda y\,.\,y\,y)(\lambda y\,.\,y\,y)$

(c) izraz se računa v nedogled

(d) nič od zgoraj naštetega

Pazite na pravilno uporabo vezanih in prostih spremenljivk!

A = (^ x y . x y) y = (^ x z . x z) y = ^ z . y z

AA = (^ z . y z) (^ z . y z) = y (^ z . y z)

c) (7 točk) Implementirajte *kakeršenkoli* modul z imenom `Cow`, ki ustreza podpisu

```
module type BOVINE =
sig
  type t
  val cow : t
  val equal : t -> t -> bool
  val to_string : t -> string
end
```

Odgovor:

```
module Cow : BOVINE =
struct
  (* Tu vpisite vsebino modula *)
```

```
type t = PPJ
let cow = PPJ
let equal _ _ = false
let to_string _ = "PPJ"
```

```
end
```

d) (6 točk) Izpeljite *glavni tip* funkcije `f`, ki je v OCamlu definirana kot

```
let f a b = b a
```

```
val f : 'a -> ('a -> 'b) -> 'b = <fun>
```

2. naloga (35 točk)

a) (15 točk) Dokažite *delno* pravilnost programa:

```
{b > 1}
i := 2 ;
j := 0 ;
while j < b do
  i := i + i + i - 2;
  j := j + 1 ;
end
{i = 3^b + 1}
```

DELNA PRAVILNOST

I: {i = 3^j + 1, j ≤ b}

{b > 1}

i := 2 ;

{ b > 1, i = 2 }

j := 0 ;

{ b > 1, i = 2, j = 0} => {i = 3^j + 1, j ≤ b} I

while j < b do

{i = 3^j + 1, j < b, j ≤ b} =>

{ i = 3^j + 1, j + 1 ≤ b } I

{ 3i = 3 * 3^j + 3, j + 1 ≤ b }

{ i + i + i - 2 = 3^(j + 1) + 1, j + 1 ≤ b }

i := i + i + i - 2;

{ i = 3^(j + 1) + 1, j + 1 ≤ b }

j := j + 1 ;

{ i = 3^j + 1, j ≤ b } I

end
{ i = 3^j + 1, j ≤ b, j ≥ b} => {i = 3^j + 1, j = b } I =>

{i = 3^b + 1}

* (j + 1 ≤ b) => j ≤ b

POPOLNA PRAVILNOST

z = b - j ≥ 0

b) (5 točk) Ali se zgornji program vedno ustavi? Če menite da se ustavi, navedite nenegativno celoštevilsko količino, ki se v zanki `while` zmanjšuje. Odgovora ni treba utemeljiti.

(a) Ni nujno, da se pri danih pogojih program vedno ustavi.

(b) Program se vedno ustavi, ker se zmanjšuje količina b - j.

c) (15 točk) Implementirajte program iz vprašanja (a) v OCamlu ali v Haskellu kot funkcijo

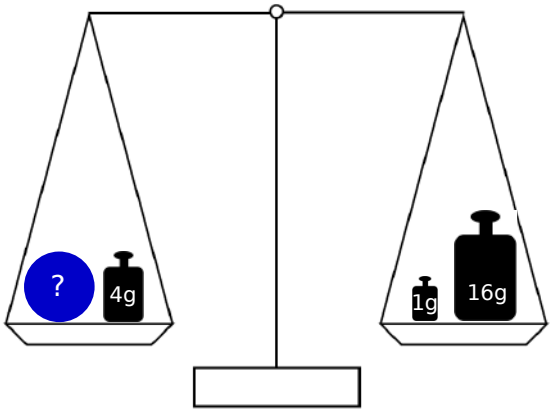
```
power3plus1 : int -> int
```

ki sprejme nenegativno celo število b in vrne enako vrednost, ki jo določa speficikacija. Funkcija naj ne uporablja zanke `while` ali `for`. Za vse točke naj bodo vsi rekurzivni klici *repni*.

```
let power3plus1 b =
  let rec pomozna i j =
    if j < b
    then pomozna (i + i + i - 2) (j + 1)
    else
      i
  in
  pomozna 2 0
```

3. naloga (50 točk)

Imamo tehtnico in uteži, kot je prikazano na spodnji sliki. Če želimo stehtati modro kroglo, jo postavimo skupaj z utežmi na tehtnico, tako da je doseženo ravnovesje. Iz prikazane razporeditvene uteži lahko sklepamo, da ima modra krogla maso $1g + 16g - 4g = 13g$.



a) (5 točk) V prologu sestavite predikat `balance(L, R, B)`, ki velja natanko tedaj, ko je `B` *bilanca* na tehtnici, ker je `L` seznam uteži na levi strani tehtnice in `R` seznam uteži na desni. Se pravi, `B` je razlika skupne mase uteži na desni in skupne mase uteži na levi. Primeri uporabe:

```
?- balance([], [], B).
B = 0.
?- balance([4], [1, 16], B).
B = 13.
?- balance([1, 42], [1, 2, 3], B).
B = -37.
```

Namig: prav vam bosta prišla predikat `sum/2` iz vaj in predikat `sum/3` iz knjižnice `clpfd`.

```
balance(L, R, B) :-
    sum(L, LS),
    sum(R, BR),
    B is BR - LS.
```

```
balance([], [], 0).

balance([H | L], R, B) :-
    balance(L, R, BL),
    B is BL - H.

balance(L, [H | R], B) :-
    balance(L, R, BR),
    B is BR + H.
```

b) (15 točk) Sestavite predikat `split(Ws, L, R)`, ki velja natanko tedaj, ko seznama uteži `L` in `R` predstavljata razporeditev uteži na levi in desni strani tehtnice, pri čemer uporabljamo samo uteži s seznama `Ws`. Na tehtnico lahko postavimo vsako utež iz `Ws` *največ enkrat*.

Primeri uporabe:

```
?- split([1], L, R).
L = [1], R = [] ;
L = [], R = [1] ;
L = R, R = [].

?- split([1,2,3], L, R).
L = [1, 2, 3], R = [] ;
L = [1, 2], R = [3] ;
...
% (skupno 27 odgovorov)

?- split([1,1,3], [3], R).
R = [1, 1] ;
R = [1] ;
R = [1] ;
R = [] ;
false.
```

Uteži v seznamih `L` in `R` vedno naštejemo v enakem vrstnem redu, kot so podane v seznamu `Ws`. Na primer poizvedba `?- split([1,2,3],L,R)` poda rešitev `L=[1,2]`, kot je prikazano v zgornjem primeru, in *ne* poda rešitve `L=[2,1]`, ker le-ta ne spoštuje vrstnega reda `[1,2,3]`.

```
split([], [], []).

split([W | Ws], [W | Ls], R) :-
    split(Ws, Ls, R).

split([W | Ws], L, [W | Rs]) :-
    split(Ws, L, Rs).

split([_ | Ws], L, R) :-
    split(Ws, L, R).
```

```
split([], [], []).

split([W | Ws], L, R) :-
    (L = L1, R = R1;
     L = [W | L1], R = R1;
     L = L1, R = [W | R1]
    ),
split(Ws, L1, R1).
```

c) (10 točk) Sestavite predikat `measure(Ws, W)`, ki velja natanko tedaj, ko lahko z utežmi s seznama `Ws` stehtamo predmet z maso `W`. Primera uporabe:

```
?- measure([1,3], W).
W = -4 ;
W = 2 ;
W = -1 ;
W = -2 ;
W = 4 ;
W = 1 ;
W = -3 ;
W = 3 ;
W = 0.
```

```
?- measure([1,1], W).
W = -2 ;
W = 0 ;
W = -1 ;
W = 0 ;
W = 2 ;
W = 1 ;
W = -1 ;
W = 1 ;
W = 0.
```

V rešitvi smete uporabiti `balance/3` in `split/3`, tudi če niste rešili podnalog (a) in (b).

```
measure(Ws, W) :-
    split(Ws, L, R),
    balance(L, R, W).
```

d) (10 točk) Sestavite predikat `measure_interval(Ws, A, B)`, ki velja natanko tedaj, ko lahko z utežmi v seznamu `Ws` stehtamo predmete z masami od `A` do vključno `B`. Primeri:

```
?- measure_interval([1,3], 0, 4]).
true.
?- measure_interval([W1,W2,W3], 5, 3).
true.
?- measure_interval([1,2,3], 0, 8).
false.
```

Za čast in slavo pospešite rešitev z uporabo predikata `once(Q)`, ki vrne le prvo rešitev cilja `Q`.

```
measure_interval(_, A, B) :-
    A #> B.

measure_interval(Ws, A, B) :-
    measure(Ws, A),
    A1 is A + 1,
    measure_interval(Ws, A1, B).
```

e) (10 točk) Zapišite poizvedbo, ki poišče nabor štirih uteži z masami 1 do 40, s katerimi lahko tehtamo predmete z masami na intervalu $[0, 40]$.

Poizvedba:

```
length(Ws, 4), Ws ins 1..40, label(Ws), measure_interval(Ws, 0, 40).
```

Principi programskih jezikov
3. izpit, 19. avgust 2020

1	
2	
3	
Σ	

Ime in priimek

Vpisna številka

NAVODILA

- **Ne odpirajte te pole**, dokler ne dobite dovoljenja.
- **Preden začnete reševati test:**
  - Vpišite svoje podatke na testno polo z velikimi tiskanimi črkami.
  - Na vidno mesto položite osebni dokument s sliko in študentsko izkaznico.
  - Preverite, da imate mobitel izklopljen in spravljen v torbi.
- Dovoljeni pripomočki: pisalo, brisalo, in poljubno pisno gradivo.
- Vse rešitve vpisujte v polo.
- Če kaj potrebujete, prosite asistenta, ne sosedov.
- **Med izpitom ne zapuščajte svojega mesta** brez dovoljenja.
- Testna pola vam bo odvzeta **brez nadaljnjih opozoril**, če:
  - komunicirate s komerkoli, razen z asistentom,
  - komu podate kak predmet ali list papirja,
  - odrinete svoje gradivo, da ga lahko vidi kdo drug,
  - na kak drug način prepisujete ali pomagate komu prepisovati,
  - imate na vidnem mestu mobitel ali druge elektronske naprave.
- **Ob koncu izpita:**
  - Ko asistent razglasi konec izpita, **takoj** nehajte in zaprite testno polo.
  - **Ne vstajajte**, ampak počakajte, da asistent pobere **vse** testne pole.
  - **Testno polo morate nujno oddati.**
- Čas pisanja je 120 minut. Na vidnem mestu je zapisano, do kdaj imate čas.
- Predvideni ocenjevalni kriterij:
  1. ≥ 90 točk, ocena 10
  2. ≥ 80 točk, ocena 9
  3. ≥ 70 točk, ocena 8
  4. ≥ 60 točk, ocena 7
  5. ≥ 50 točk, ocena 6

  Veliko uspeha!

1. naloga (35 točk)

a) (7 točk) Elbonijski direktorat za standarde je uvedel novo sintakso aritmetičnih izrazov. Vsa števila zapisujejo s poševnicami v eniškem sistemu, na primer ///// je število pet (nihče ni pomislil na število nič). Ljudstvo je bilo navadušeno, saj je v Elboniji poševnica znak za srečo. Direktorat je zato spremenil tudi zapis seštevanja in razglasil, da se namesto znaka $+$ odslej za seštevanje uporabi poševnica $/$. Množenje so pisali s $\times$. Njihova nova sintaksa je torej naslednja:

$$\begin{aligned} \langle \text{izraz} \rangle &::= \langle \text{multiplikativni} \rangle \mid \langle \text{izraz} \rangle / \langle \text{multiplikativni} \rangle \\ \langle \text{multiplikativni} \rangle &::= \langle \text{število} \rangle \mid \langle \text{multiplikativni} \rangle \times \langle \text{število} \rangle \\ \langle \text{število} \rangle &::= /^{+} \end{aligned}$$

V državi sedaj vlada zmeda, zato so vas poklicali na pomoč. Direktorju direktorata morate pojasniti, da je možno nekatere izraze razčleniti na več načinov. V ta namen mu predočite izraz

$$// \times ////$$

Narišite *dve* različni drevesni predstavitvi zgornjega izraza, s katerima boste direktorju prikazali dvoumnost nove sintakse.

Prva različica: Druga različica:

b) (7 točk) V λ-računu denifirajte izraz A tako, da bo veljalo

$$(\lambda x \,.\, A\, x)(\lambda x \,.\, A\, x) = y$$

Odgovor: $A =$ ^ x . y

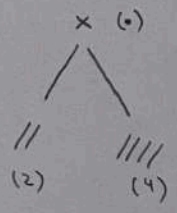
///// ... 5

/ ... število ali seštevanje

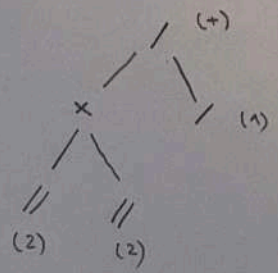
x ... množenje

// x ////

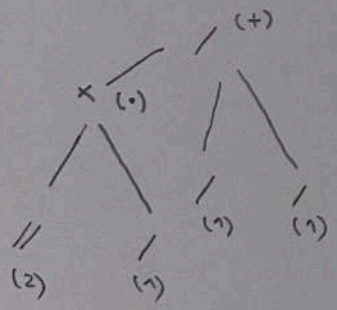
①



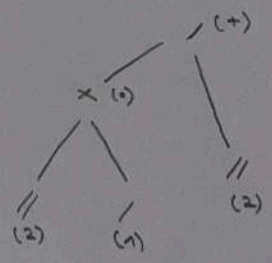
②



③



④



c) (7 točk) V OCamlu definiramo podatkovni tip dreves, v katerih so vozlišča označena s celimi števili:

```
type tree = Empty | Node of int * tree * tree
```

Sestaviti želimo funkcijo `sum : tree -> int`, ki sešteje cela števila v vozliščih drevesa:

```
# sum Empty ;;
- : int = 0
# sum (Node (20, Node (3, Empty, Empty), Node (19, Empty, Empty))) ;;
- : int = 42
```

Dopolnite implementacijo funkcije `sum`:

```
let rec sum = function
  | Empty -> _____________________
  | Node (k,l,r) -> _____________________
```

```
let rec sum = function
| Empty -> 0
| Node(k, l, r) -> k + sum l + sum r
```

d) (7 točk) Izpeljite *glavni tip* funkcije `f`, ki je v OCamlu definirana kot

```
let f g = g [0; 1; 2]
```

Odgovor: _____

```
utop # let f g = g [0; 1; 2] ;;
val f : (int list -> 'a) -> 'a = <fun>
```

e) (7 točk) V Ocamlu definiramo tip

```
type oseba = {ime : string ; priimek : string ; rojstvo : int }
```

Med spodnjimi izrazi označite tiste, ki imajo tip `oseba`:

(a) `{ime="Kekec"; priimek=None; rojstvo=1918}`

(b) `{ime="Kekec"; rojstvo=1918}`

(c) `{ime="Kekec"; priimek=""; rojstvo=(let s=1000 in s + 918)}`

(d) `{ime="Mojca"; priimek="Pokraculja"; rojstvo=1920}`

(e) `{priimek="Pokraculja"; ime="Mojca"; rojstvo=1/0}`

2. naloga (35 točk)

a) (20 točk) Dokažite *delno* pravilnost programa. Iz vaše rešitve naj bo jasno razvidna invarianta zanke `while`.

$\{\text{true}\}$

```
x := a ;

i := 0 ;

while i < 100 do

  x := x * x * a ;

  i := i + 1

end
```

$\{x = a^{2^{101}-1}\}$

b) (15 točk) Dokažite še *polno* pravilnost programa. Iz vaše rešitve naj bo jasno razvidno, katera količina zagotavlja zaustavitev zanke `while`.

$[\text{true}]$

```
x := a ;

i := 0 ;

while i < 100 do

  x := x * x * a ;

  i := i + 1

end
```

$[x = a^{2^{101}-1}]$

DELNA PRAVILNOST

I: x = a^2^(i + 1) - 1, i ≤ 100

{true}

x := a ;

{ x = a }

i := 0 ;

{ x = a, i = 0 } => { x = a^2^(i + 1) - 1, i ≤ 100 } I

while i < 100 do
  { i < 100 } => { x = a^2^(i + 1) - 1, i ≤ 100 } I

  { i < 100, x * x * a = a^2^(i + 2) - 1 }

  x := x * x * a ;

  { i < 100, x = a^2^(i + 2) - 1 }

  { i + 1 ≤ 100, x = a^2^(i + 2) - 1 } <=>

  { i + 1 ≤ 100, x = a^2^(i + 1 + 1) - 1 }

  i := i + 1

  { i ≤ 100, x = a^2^(i + 1) - 1 } I

end
{i ≥ 100, i ≤ 100, x = a^2^(i + 1) - 1 } I

{i = 100, x = a^2^(i + 1) - 1 } =>

{x = a^2^101 - 1}

POPOLNA PRAVILNOST

e = 101 - i
101 - i > 0

Potrebno seboj nesti tudi invarianto, zaradi preglednosti ni napisano.

[i = 0, x = a, x = a^2^(i + 1) - 1, i ≤ 100]

while i < 100 do
  [e = 101 - i, 101 - i > 0]

  x := x * x * a ;

  [e = 101 - (i + 1 - 1), 101 - (i + 1 - 1) > 0]

  i := i + 1

  [e = 101 - i + 1, 101 - i + 1 > 0]

  [e - 1 = 101 - i, 101 - i > - 1]

  [e > 101 - i, 101 - i > - 1]

end

⊗

$$x^2 a = \left(a^{2^{i+1}} - 1\right)^2 a =$$

$$= \left(a^{2^{i+2}} - 2\right) a =$$

$$= a^{2^{i+2}} - 1 \quad \checkmark$$

3. naloga (40 točk)

V OCamlu definiramo podatkovni tipi `number`, s kateri predstavimo cela števila:

```
type number = Zero | Succ of integer | Pred of integer
```

Vrednost `Zero` predstavlja število 0, `Succ n` naslednik `n` ter `Pred n` predhodnik `n`. Vsako število lahko predstavimo na več načinov. Na primer, število 0 je predstavljeno z vrednostmi

```
Zero
Pred (Succ Zero)
Succ (Pred Zero)
Pred (Pred (Succ (Succ Zero)))
Pred (Succ (Succ (Pred Zero)))
...
```

Med vsemi je najbolj "ekonomična" predstavitev seveda `Zero`, ker ne vsebuje nepotrebnih konstruktorjev.

a) (20 točk) Sestavite funkcijo `simp : number -> number`, ki dano predstavitev pretvori v najbolj ekonomično, se pravi tako, ki ima najmanjše možno število konstruktorjev. Primeri:

```
# simp (Pred (Succ (Succ (Pred (Pred (Succ (Pred Zero)))))));;
- : number = Pred Zero
# simp (Succ Zero);;
- : number = Succ Zero
```

```
type number = Zero | Succ of integer | Pred of integer

let rec simp = function
  | Zero -> Zero
  | Pred a -> (match simp a with
      | Succ a -> a
      | a -> Pred a)
  | Succ a -> (match simp a with
      | Pred a -> a
      | a -> Succ a)
```

```
let rec simp = function
  | Pred(Succ(x)) -> simp x
  | Succ(Pred(x)) -> simp x
  | Pred(x) -> Pred(x)
  | Succ(x) -> Succ(x)
  | Zero -> Zero
```

b) (20 točk) Enako predstavitev celih števil uporabimo tudi v Prologu, le da moramo konstruktorje pisati z malo začetnico. Na primer, število 3 predstavimo z izrazom

```
succ(pred(succ(succ(succ(pred(succ(zero)))))))
```

ki pa ni najbolj ekonomičen. Dopolnite spodnji predikat `simp(A,B)`, ki velja, ko je `B` najbolj ekonomična predstavitev `A`. Primer uporabe:

```
?- simp(succ(pred(succ(succ(succ(pred(succ(zero)))))))), B).
B = succ(succ(succ(zero))) ;
false.
```

```
simp(_____________________, _____________________).

simp(succ(A), _____________________)
  :- simp(A, pred(C)).

simp(succ(A), _____________________)
  :- simp(A, zero).

simp(succ(A), _____________________)
  :- simp(A, succ(C)).

simp(pred(A), _____________________)
  :- simp(A, _____________________).

simp(pred(A), _____________________)
  :- simp(A, _____________________).

simp(pred(A), _____________________)
  :- simp(A, _____________________).
```

b) (20 točk) Enako predstavitev celih števil uporabimo tudi v Prologu, le da moramo konstruktorje pisati z malo začetnico. Na primer, število 3 predstavimo z izrazom

```
succ(pred(succ(succ(succ(pred(succ(zero)))))))
```

ki pa ni najbolj ekonomičen. Dopolnite spodnji predikat `simp(A,B)`, ki velja, ko je `B` najbolj ekonomična predstavitev `A`. Primer uporabe:

```
?- simp(succ(pred(succ(succ(succ(pred(succ(zero)))))))), B).
B = succ(succ(succ(zero))) ;
false.
```

next → succ prev → pred

```
simp(zero, zero).

simp(succ(A), C)
  :- simp(A, pred(C)).

simp(succ(A), next(zero))
  :- simp(A, zero).

simp(succ(A), next(next(C)))
  :- simp(A, succ(C)).

simp(pred(A), C)
  :- simp(A, next(C)).

simp(pred(A), prev(zero))
  :- simp(A, zero).

simp(pred(A), prev(prev(C)))
  :- simp(A, prev(C)).
```

Principi programskih jezikov
1. izpit, 14. junij 2021

1	
2	
3	
Σ	

Ime in priimek

Vpisna številka

NAVODILA

- **Ne odpirajte te pole**, dokler ne dobite dovoljenja.
- **Preden začnete reševati test:**
  - Vpišite svoje podatke na testno polo z velikimi tiskanimi črkami.
  - Na vidno mesto položite osebni dokument s sliko in študentsko izkaznico.
  - Preverite, da imate mobitel izklopljen in spravljen v torbi.
- Dovoljeni pripomočki: pisalo, brisalo, in poljubno pisno gradivo.
- Vse rešitve vpisujte v polo.
- Če kaj potrebujete, prosite asistenta, ne sosedov.
- **Med izpitom ne zapuščajte svojega mesta** brez dovoljenja.
- Testna pola vam bo odvzeta **brez nadaljnjih opozoril**, če:
  - komunicirate s komerkoli, razen z asistentom,
  - komu podate kak predmet ali list papirja,
  - odrinete svoje gradivo, da ga lahko vidi kdo drug,
  - na kak drug način prepisujete ali pomagate komu prepisovati,
  - imate na vidnem mestu mobitel ali druge elektronske naprave.
- **Ob koncu izpita:**
  - Ko asistent razglasi konec izpita, **takoj** nehajte in zaprite testno polo.
  - **Ne vstajajte**, ampak počakajte, da asistent pobere **vse** testne pole.
  - **Testno polo morate nujno oddati.**
- Čas pisanja je 120 minut. Na vidnem mestu je zapisano, do kdaj imate čas.
- Predvideni ocenjevalni kriterij:
  1. ≥ 90 točk, ocena 10
  2. ≥ 80 točk, ocena 9
  3. ≥ 70 točk, ocena 8
  4. ≥ 60 točk, ocena 7
  5. ≥ 50 točk, ocena 6

  Veliko uspeha!

1. naloga (35 točk)

a) (7 točk) Elbonijski direktorat za standarde je uvedel novo aritmetiko z lokalnimi definicijami, ki jih zapišemo z določilom `where`. Na primer, v izrazu

```
3 - Y where Y = 4 - X where X = (0 - 1 - 2)
```

najprej izračunamo $X = (0 - 1 - 2) = -3$, nato $Y = 4 - (-3) = 7$ in dobimo končno vrednost $3 - 7 = -4$. Nova slovnica za elbonijske aritmetične izraze se glasi:

$$
\begin{aligned}
\langle \text{vezava} \rangle &::= \langle \text{odštevalni} \rangle \mid \langle \text{odštevalni} \rangle \texttt{ where } \langle \text{spremenljivka} \rangle \texttt{ = } \langle \text{vezava} \rangle \\
\langle \text{odštevalni} \rangle &::= \langle \text{osnovni} \rangle \mid \langle \text{vezava} \rangle \texttt{ - } \langle \text{osnovni} \rangle \\
\langle \text{osnovni} \rangle &::= \langle \text{število} \rangle \mid \langle \text{spremenljivka} \rangle \mid (\langle \text{vezava} \rangle) \\
\langle \text{število} \rangle &::= [\texttt{0-9}]^+ \\
\langle \text{spremenljivka} \rangle &::= [\texttt{A-Z}]^+
\end{aligned}
$$

V državi sedaj vlada zmeda, zato so vas poklicali na pomoč. Direktorju direktorata morate pojasniti, da je možno nekatere izraze razčleniti na več načinov. V ta namen mu predočite izraz

```
Y where Y = 4 - Y where Y = 0 - 1 - 2
```

Narišite ***različni*** drevesni predstavitvi zgornjega izraza, s katerima boste direktorju prikazali dvoumnost nove sintakse.

Prva različica: Druga različica:

b) (7 točk) V λ-računu denifirajte dva ***različna*** izraza A in B tako, da velja

$$(\lambda x \,.\, x\, x\, x)\, A = A$$
$$(\lambda y \,.\, y\, y\, y)\, B = B$$

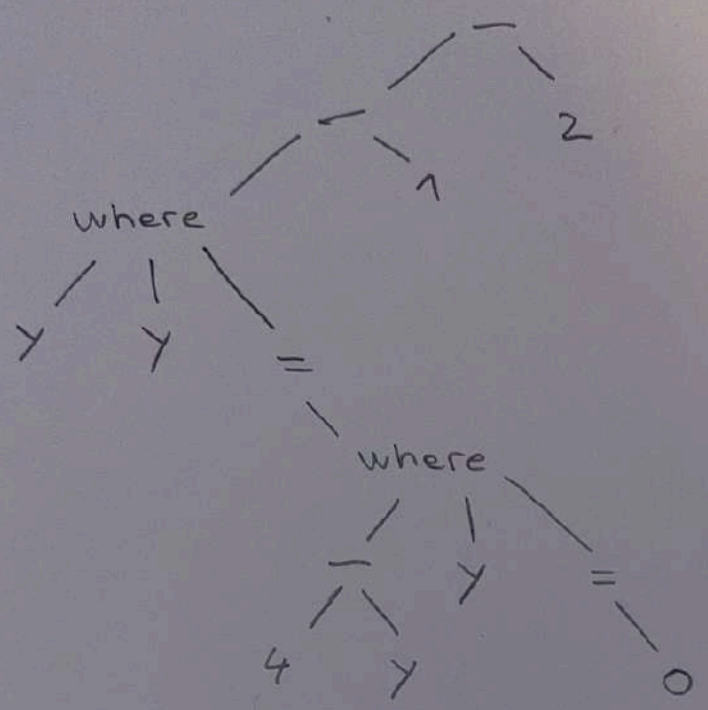
(Izraza, ki se razlikujeta le v poimenovanju vezanih spremenljivk sta *enaka*.) Odgovor:

$A :=$ _____

$B :=$ _____

A : = ^ x. x ;
B := ^ x. (^ y. y) x

y where y = 4 - y where y = 0 - 1 - 2



c) (7 točk) V OCamlu definiramo podatkovni tip dreves:

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

Sestaviti želimo funkcijo `listje : 'a drevo -> int`, ki prešteje liste v drevesu:

```
# listje List ;;
- : int = 1
# listje (Veja (Rogovila (Veja List, Rogovila
    (Rogovila (List, Veja List, List), List, Veja List), Plod "oreh")))
- : int = 6
```

Dopolnite implementacijo funkcije `listje`:

```
let rec listje = function
  | List -> _____________________________
  | Plod p -> _____________________________
  | Veja v -> _____________________________
  | Rogovila _________ ->
      ___________________________________
```

d) (7 točk) Izpeljite *glavni tip* funkcije `dodaj`, ki je v OCamlu definirana kot

```
let dodaj f x = f () :: "in" :: x
```

Odgovor: _____

val dodaj : (unit -> string) -> string list -> string list = <fun>

```
let rec listje = function
  | List -> 1
  | Plod(_) -> 0
  | Veja (l) -> listje l
  | Rogovila(a, b, c) -> listje a + listje b + listje c
```

e) (7 točk) Andrej rad sprašuje študente o zapisih in podtipih zapisov. V ta namen je definiral tipa zapisov:

$$\rho = \{\mathtt{a} : \{\mathtt{b} : \mathtt{int}\} \to \mathtt{bool}; \mathtt{c} : \mathtt{int}\}$$
$$\sigma = \{\mathtt{a} : \{\mathtt{b} : \mathtt{int}; \mathtt{c} : \mathtt{int}\} \to \mathtt{bool}\}$$

- Navedite kakšno vrednost tipa ρ: let p = {a = fun y -> true; c = 5}
- Navedite kakšno vrednost tipa σ: let o = {a = fun y -> true}
- Ali velja $\rho \leq \sigma$? DA
- Ali velja $\sigma \leq \rho$? NE

Pri relaciji $\leq$ upoštevajte podtipe po širini in globini.

```
type tip4 = {b:int}
type tip3 = {b : int; c : int}
type tip2 = {a : tip3  -> bool}
type tip5 = {a:tip4 -> bool;c :int}
```

2. naloga (35 točk)

a) (20 točk) Dokažite *delno* pravilnost programa. Iz vaše rešitve naj bo jasno razvidna invarianta zanke `while`.

$\{1 \leq j\}$

Invarianta: i ≤ j

```
i := 1 ;

while i + i <= j do

    i := i * 2

end

k := j - i
```

$\{j = i + k \wedge 0 \leq 2k < j\}$

b) (15 točk) Dokažite še *polno* pravilnost programa. Iz vaše rešitve naj bo jasno razvidno, katera količina zagotavlja zaustavitev zanke `while`.

$[1 \leq j]$

```
i := 1 ;
```

```
while i + i <= j do
```

```
    i := i * 2
```

```
end
```

```
k := j - i
```

$[j = i + k \wedge 0 \leq 2k < j]$

3. naloga (40 točk)

Klemen se na morju igra s prelivanjem vode med kanglicami. V vsakem koraku lahko naredi eno od naslednjih potez:

- Izprazni eno od kanglic.
- Napolni eno od kanglic do roba.
- Pretoči vodo iz ene kanglice v drugo, dokler ni prva prazna ali druga polna.

Na primer, če ima prazno kanglico prostornine $3\,\ell$ in kanglico prostornine $5\,\ell$, v kateri so že $4\,\ell$ vode, lahko napolni prvo ali drugo kanglico, izprazni drugo, ali pretoči $3\,\ell$ iz druge v prvo. Klemna je od nekdaj zanimalo, kako bi z danimi kanglicami v nekaj potezah izmeril želeno količino vode.

Pomagali mu bomo odgovoriti na vprašanje v Prologu. Trenutno stanje kanglic predstavimo s seznamom

$$[v_1/c_1, v_2/c_2, \ldots, v_n/c_n]$$

pri čemer v_i/c_i pomeni, da ima i-ta kanglica prostornino c_i in da je v njej v_i litrov vode. Vse prostornine so seveda pozitivne in vse količine vode nenegativne. (Pozor, v prologu zapis `V/C` *ne* označuje ulomka ali deljenja, ampak urejeni par `V` in `C`.)

a) (5 točk) Sestavite predikat `resitev(X,L)`, ki velja, kadar je v eni od kanglic s seznama `L` natanko `X` litrov vode. Primer uporabe:

```
?- resitev(3, [4/11, 3/5, 0/7]).
true.

?- resitev(4, []).
false.
```

Odgovor:

```
resitev(X, [X /_| _]).

resitev(X, [H / _ | T]) :-
    dif(X, H),
    resitev(X, T).
```

b) (8 točk) Sestavite predikat `napolni(L,M)`, ki velja, kadar lahko dobimo seznam kanglic `M` iz seznama `L` tako, da napolnimo eno od *še ne polnih* kanglic. Primer uporabe:

```
?- napolni([4/11, 0/3, 7/7], M).
M = [11/11, 0/3, 7/7] ;
M = [4/11, 3/3, 7/7] ;
false.

?- napolni([11/11, 7/7], M).
false.

?- napolni([], M).
false.
```

Odgovor:

```
napolni([C / V | T], M) :-
    ( V #> C, M = [V/V | T] );
    napolni(T, NT),
    append([C/V], NT, M).
```

c) (7 točk) Sestavite predikat `sprazni(L,M)`, ki velja, kadar lahko dobimo seznam kanglic `M` iz seznama `L` tako, da spraznimo eno od *nepraznih* kanglic. Primer uporabe:

```
?- sprazni([4/11, 0/3, 7/7], M).
M = [0/11, 0/3, 7/7] ;
M = [4/11, 0/3, 0/7] ;
false.

?- sprazni([0/3, 0/7], M).
false.

?- sprazni([], M).
false.
```

Odgovor:

```
sprazni([C / V | T], M) :-
    ( C #> 0, M = [0/V | T] );
    sprazni(T, NT),
    append([C/V], NT, M).
```

d) (10 točk) Klemen je sestavil predikat `pretoci1(V1/C1, V2/C2, W1/C1, W2/C2)`, ki velja, kadar s pretakanjem vode iz kanglice `V1/C1` v kanglico `V2/C2` dobimo kanglici `W1/C1` in `W2/C2`:

```
pretoci1(V1/C1, V2/C2, W1/C1, W2/C2) :-
  V1 > 0, V2 < C2, W2 is min(V1+V2,C2), W1 is V1+V2-W2.
```

Poleg tega je sestavil še predikat `izberi2(L, X, Y, M)`, ki iz seznama `L` izbere dva elementa `X` in `Y` in je `M` enak `L` brez izbranih dveh elementov:

```
izberi1([X|L], X, L).
izberi1([Y|M], X, [Y|L]) :- izberi1(M, X, L).

izberi2(M, X, Y, L) :- izberi1(M, X, K), izberi1(K, Y, L).
```

Sestavite predikat `pretoci(L, M)`, ki velja, kadar lahko seznam kanglic `M` dobimo iz seznama `L` tako, da izberemo dve kanglici in pretočimo vodo iz ene v drugo. (Vrstnega reda kanglic ni treba ohraniti.) Primer uporabe:

```
?- pretoci([4/10, 0/3, 7/7], M).
M = [1/10, 3/3, 7/7] ;
M = [1/7, 10/10, 0/3] ;
M = [4/7, 3/3, 4/10] ;
false.

?- pretoci([3/7, 2/3], M).
M = [2/7, 3/3] ;
M = [0/3, 5/7] ;
false.

?- pretoci([2/7, 0/3], M).
M = [0/7, 2/3] ;
false.

?- pretoci([2/7], M).
false.
```

Odgovor:

e) (10 točk) Na koncu sestavite še predikat `poteze(V, L, M)`, ki velja, kadar je `M` seznam seznamov kanglic, ki predstavlja zaporedje potez, ki vodijo od začetnega stanja kanglic `L` do kanglic, od katerih vsaj ena vsebuje `V` litrov vode. Primer uporabe:

```
?- length(M,3), poteze(4, [0/3, 0/7], M).
M = [[0/3, 0/7], [0/3, 7/7], [4/7, 3/3]] ;
false.

?- length(M,4), poteze(2, [1/1, 0/3], M).
M = [[1/1, 0/3], [1/1, 3/3], [0/1, 3/3], [2/3, 1/1]] ;
M = [[1/1, 0/3], [0/1, 0/3], [0/1, 3/3], [2/3, 1/1]] ;
M = [[1/1, 0/3], [0/1, 1/3], [1/1, 1/3], [0/1, 2/3]] ;
M = [[1/1, 0/3], [0/1, 1/3], [0/1, 3/3], [2/3, 1/1]] .

?- length(M,10), poteze(3, [0/2, 0/4], M).
false.
```

Odgovor:

Principi programskih jezikov
3. izpit, 7. september 2021

1	
2	
3	
Σ	

Ime in priimek

Vpisna številka

NAVODILA

- **Ne odpirajte te pole**, dokler ne dobite dovoljenja.
- **Preden začnete reševati test:**
  - Vpišite svoje podatke na testno polo z velikimi tiskanimi črkami.
  - Na vidno mesto položite osebni dokument s sliko in študentsko izkaznico.
  - Preverite, da imate mobitel izklopljen in spravljen v torbi.
- Dovoljeni pripomočki: pisalo, brisalo, in poljubno pisno gradivo.
- Vse rešitve vpisujte v polo.
- Če kaj potrebujete, prosite asistenta, ne sosedov.
- **Med izpitom ne zapuščajte svojega mesta** brez dovoljenja.
- Testna pola vam bo odvzeta **brez nadaljnjih opozoril**, če:
  - komunicirate s komerkoli, razen z asistentom,
  - komu podate kak predmet ali list papirja,
  - odrinete svoje gradivo, da ga lahko vidi kdo drug,
  - na kak drug način prepisujete ali pomagate komu prepisovati,
  - imate na vidnem mestu mobitel ali druge elektronske naprave.
- **Ob koncu izpita:**
  - Ko asistent razglasi konec izpita, **takoj** nehajte in zaprite testno polo.
  - **Ne vstajajte**, ampak počakajte, da asistent pobere **vse** testne pole.
  - **Testno polo morate nujno oddati.**
- Čas pisanja je 120 minut. Na vidnem mestu je zapisano, do kdaj imate čas.
- Predvideni ocenjevalni kriterij:
  1. ≥ 90 točk, ocena 10
  2. ≥ 80 točk, ocena 9
  3. ≥ 70 točk, ocena 8
  4. ≥ 60 točk, ocena 7
  5. ≥ 50 točk, ocena 6

  Veliko uspeha!

1. naloga (35 točk)

a) (7 točk) Ker Elbonija doživlja globoko ekonomsko krizo, na vsakem koraku varčujejo. Direktorat za aritmetiko je zato prepovedal nepotrebno uporabo podvojenih oklepajev, se pravi, da izraz ne sme vsebvati podizrazov oblike $((\cdots))$. Stara slovnica aritmetičnih izrazov dvojne oklepaje dopušča:

$$\begin{aligned} \langle \text{izraz} \rangle &::= \langle \text{multiplikativni} \rangle \mid \langle \text{izraz} \rangle + \langle \text{multiplikativni} \rangle \\ \langle \text{multiplikativni} \rangle &::= \langle \text{osnovni} \rangle \mid \langle \text{multiplikativni} \rangle \times \langle \text{osnovni} \rangle \\ \langle \text{osnovni} \rangle &::= \langle \text{število} \rangle \mid (\langle \text{izraz} \rangle) \\ \langle \text{število} \rangle &::= [0\text{–}9]^+ \end{aligned}$$

Popravite jo tako, da bodo podvojeni oklepaji neveljavni. Na primer, izraz $3 + (4 \times (5))$ je dovoljen, izraz $3 + (4 \times ((5)))$ ni dovoljen zaradi $((5))$ in izraz $((3 + 4)) \times 5$ ni dovoljen zaradi $((3 + 4))$.

```
<izraz> ::= <multiplikativni> | <izraz> + <multiplikativni>
<multiplikativni> ::= <osnovni> | <multiplikativni> × <osnovni>
<osnovni> ::= <število> | (<izrazi2>)
<število> ::= [0-9]+

<izraz2> ::= <multiplikativni2> | <izraz> + <multiplikativni>
<multiplikativni2> ::= <osnovni2> | <multiplikativni> × <osnovni>
<osnovni2> ::= <število>
```

b) (7 točk) V logiki včasih poleg *neresnice* `F` in *resnice* `T` uporabljamo še vrednost *mogoče* `M`. Temu primerno razširimo tudi logične veznike, tako da na primer velja $\texttt{F} \wedge p = \texttt{F}$ in $\texttt{T} \vee p = \texttt{T}$, ne glede na vrednost p. Natančneje, razširjena logična disjunkcija $\vee$ je definirana takole:

$$p \vee q = \begin{cases} \texttt{T} & \text{če } p = \texttt{T} \textit{ ali } q = \texttt{T}, \\ \texttt{F} & \text{če } p = \texttt{F} \textit{ in } q = \texttt{F}, \\ \texttt{M} & \text{sicer.} \end{cases}$$

Klemen je predstavil razširjene resničnostne vrednosti v λ-računu z izrazi

$$\texttt{T} := \lambda x\, y\, z\, .\, x, \qquad \texttt{M} := \lambda x\, y\, z\, .\, y, \qquad \texttt{F} := \lambda x\, y\, z\, .\, z.$$

Zapišite izraz `OR`, ki izračuna razširjeni logični veznik $\vee$. Na primer, veljati mora

$$\texttt{OR F}\ q = q, \qquad \texttt{OR M F} = \texttt{M}, \qquad \texttt{OR T F} = \texttt{T}.$$

```
T := ^ x y z . x ;
M := ^ x y z . y ;
F := ^ x y z . z ;

if := ^ p x y z . p x y z ;

OR := ^ x y . if x T (if y T M M) (if y T M F);

:eager
:deep
:constant q
```

c) (7 točk) Izpeljite *glavni tip* OCaml funkcije

```
let twist f = fun (x, y) -> f (y, x)
```

Odgovor: _____

('a * 'b -> 'c) -> 'b * 'a -> 'c

d) (7 točk) Sestavite program `P`, ki ustreza specifikaciji in dokažite njegovo pravilnost:

$\{ \quad x > 0 \wedge y > 0 \quad \}$
`P`
$\{ \quad x > y \vee y > x \quad \}$

{ x > 0 ∧ y > 0 }
if (x > y) then
{ x > 0, y > 0, x > y }
skip
{ x > 0, y > 0, x > y } ->
{ x > y }
else
{ x <= y }
{ x - 1 <= y - 1 }
x := x - 1
{ x <= y - 1 }
{ x + 1 <= y }
{ x < y }
end
{ x > y ∨ y > x }

e) (7 točk) Sestavite program `Q`, ki ustreza specifikaciji in dokažite njegovo pravilnost:

$\{ \quad x > 0 \wedge y > 0 \quad \}$
`Q`
$\{ \quad x > y \wedge y > x \quad \}$

{ x > 0 ∧ y > 0 }
{true}
while(!(x > y) && !(x < y)) do
{ true, x <= y, x >= y }
skip
{ true, x <= y, x >= y } =>
{ true }
done
{ true, x > y, x < y } <=>
{ x > y ∧ y > x }

2. naloga (35 točk)

Andrej je v OCamlu programiral različico klasične igrice Minesweeper. Igra poteka na minskem polju, v katerem položaje min in igralca predstavimo s celoštevilskimi koordinatami (x, y). Igralec mora z zaporedjem korakov "dol", "gor", "levo" in "desno" prispeti od danega začetnega položaja do končnega, ne da bi stopil na mino. Korake predstavimo s podatkovnim tipom

```
type korak = Dol | Gor | Levo | Desno
```

a) (15) Sestavite funkcijo `premik : int*int -> korak -> int*int`, ki sprejme trenutni položaj in korak ter vrne naslednji položaj. Primeri:

```
# premik (0,3) Dol ;;
- : int * int = (0, 2)
# premik (0,3) Levo ;;
- : int * int = (-1, 3)
```

```
type korak = Dol | Gor | Levo | Desno

let premik (x, y) = function
  | Dol -> (x, y - 1)
  | Gor -> (x, y + 1)
  | Levo -> (x - 1, y)
  | Desno -> (x + 1, y)
```

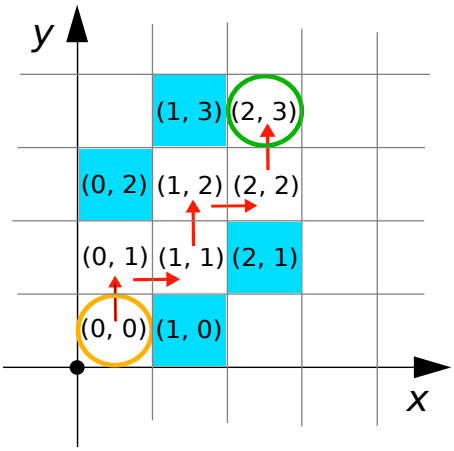
b) (20) Sestavite funkcijo

```
varna_pot : (int*int) list -> int*int -> int*int -> korak list -> bool
```

ki sprejme seznam položajev min, začetni položaj, končni položaj in seznam korakov. Funkcija vrne `true`, če dani seznam korakov vodi od začetnega do končnega položaja po poti, ki ne vsebuje mine. Primeri:

```
# varna_pot [] (0,0) (2,3) [Gor; Desno; Gor; Desno; Gor] ;;
- : bool = true
# varna_pot [(1,1)] (0,0) (2,3) [Gor; Desno; Gor; Desno; Gor] ;;
- : bool = false
# varna_pot [(0,2); (1,0); (1,3); (2,1)] (0,0) (2,3)
            [Gor; Desno; Gor; Desno; Gor] ;;
- : bool = true
```

Za vse točke naj bo funkcija *repno rekurzivna*.



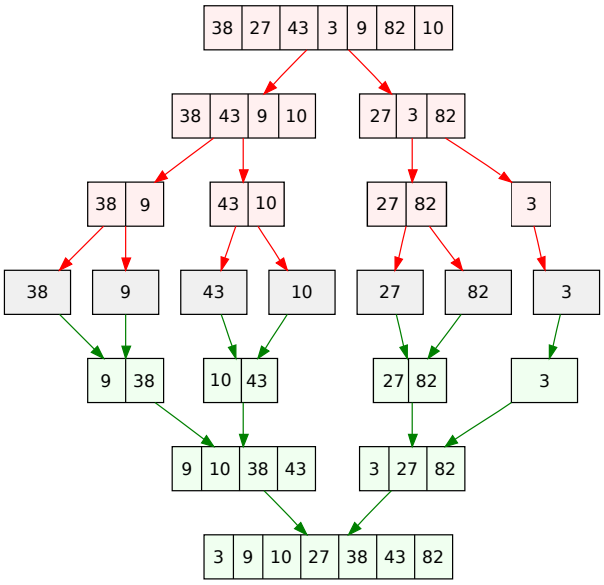

```
let rec mine mines p =
  match mines with
  l [] -> false
  l h :: t -> if p = h then true else mine t p;;

let rec varna_pot mines s e = function
  l [] -> s = e
  l h :: t -> if mine mines (premik s h)
     then false
     else varna_pot mines (premik s h) e t;;
```

```
let rec varna_pot mine x y = function
l [] -> if x = y then true else false
l l :: ls -> if List. mem (premik x l) mine then false
          else varna_pot mine (premik x l) y ls ;;
```

3. naloga (30 točk)

V Prologu bomo sestavili program za urejanje seznamov z zlivanjem (angl. *merge sort*). To je postopek tipa "deli & vladaj", ki neurejeni seznam razdeli na dva enako dolga seznama, ju rekurzivno uredi in nato zlije v urejen seznam, kot to prikazuje spodnji primer. Pozor: običajno seznam razdelimo na pol, tu pa ga razdelimo na elemente na lihih in sodih mestih.



Najprej zapišimo glavni predikat `uredi(L,S)`, ki velja, kadar je `S` po velikosti urejen seznam `L`:

```
uredi([], []).
uredi([X], [X]).
uredi([X1,X2|Xs], LS) :-
  razdeli([X1,X2|Xs], Lihi, Sodi),
  uredi(Lihi, LihiS),
  uredi(Sodi, SodiS),
  zlij(LihiS, SodiS, LS).
```

Sedaj je treba sestaviti še predikata `razdeli` in `zlij`.

a) (10 točk) Sestavite predikat `razdeli(Xs,Ys,Zs)`, ki velja, kadar seznam `Xs` radelimo na seznama `Ys` in `Zs` tako, da so v `Ys` elementi iz lihih in v `Zs` elementi iz sodih položajev seznama `Xs`. Primeri:

```
?- razdeli([42], Ys, Zs).
Ys = [42], Zs = [].

?- razdeli([1,3,5,7,9,42], Ys, Zs).
Ys = [1, 5, 9], Zs = [3, 7, 42].
```

```
razdeli(_____, _____, _____).
razdeli(_____, _____, _____).

razdeli(_____, _____, _____) :-
  razdeli(_____, _____, _____).
```

```
razdeli([], [], []).
razdeli([O], [O], []).
razdeli([O, E | T], [O | OL], [E | EL]) :- razdeli(T, OL, EL).
```

b) (20 točk) Sestavite predikat `zlij(Xs,Ys,Zs)`, ki velja kadar je izpolnjen naslednji pogoj: če sta `Xs` in `Ys` urejena seznama, je `Zs` urejen seznam elementov iz `Xs` in `Ys`. Primeri:

```
?- zlij([3,5,6,10,14], [1,2,5,6], Zs).
Zs = [1, 2, 3, 5, 5, 6, 6, 10, 14] ;
false.

?- zlij([], [1], Zs).
Zs = [1].
```

Rešitev:

```
zlij([],L,L ).
zlij([HIT],L,[HIM]):-
   zlij(T,L,M).
```

```
zlij([], [], []).
zlij(X, [], X).
zlij([], X, X).
zlij([X | O1], [Y | O2], M) :-
(X #< Y, zlij(O1, [Y | O2], N), append([X], N, M));
(X #>= Y, zlij([X | O1], O2, N), append([Y], N, M)).
```

Principi programskih jezikov, 3. izpit
23. avgust 2019

								Σ	

Ime in priimek Vpisna številka

NAVODILA

- **Ne odpirajte te pole**, dokler ne dobite dovoljenja.
- **Preden začnete reševati test:**
  - Vpišite svoje podatke na testno polo z velikimi tiskanimi črkami.
  - Na vidno mesto položite osebni dokument s sliko in študentsko izkaznico.
  - Preverite, da imate mobitel izklopljen in spravljen v torbi.
  - Prjavite se na spletno učilnico, kamor boste oddajali odgovore.
- Dovoljeni pripomočki: pisalo, brisalo, USB ključ in poljubno pisno gradivo.
- Vse rešitve vpisujte v kviz na spletni učilnici.
- Če kaj potrebujete, prosite asistenta, ne sosedov.
- **Med izpitom ne zapuščajte svojega mesta** brez dovoljenja.
- Testna pola vam bo odvzeta **brez nadaljnjih opozoril**, če:
  - komunicirate s komerkoli, razen z asistentom,
  - komu podate kak predmet ali list papirja,
  - odrinete svoje gradivo, da ga lahko vidi kdo drug,
  - na kak drug način prepisujete ali pomagate komu prepisovati,
  - imate na vidnem mestu mobitel ali druge elektronske naprave.
- **Ob koncu izpita:**
  - Ko asistent razglasi konec izpita, **takoj** nehajte in zaprite testno polo.
  - **Ne vstajajte**, ampak počakajte, da asistent pobere **vse** testne pole.
  - **Testno polo morate nujno oddati.**
- Čas pisanja je 120 minut. Na tabli je zapisano, do kdaj imate čas.
- Predvideni ocenjevalni kriterij:

  ≥ 90 točk, ocena 10
  ≥ 80 točk, ocena 9
  ≥ 70 točk, ocena 8
  ≥ 60 točk, ocena 7
  ≥ 50 točk, ocena 6

  Veliko uspeha!

1. naloga (40 točk)

a) (8 točk) V λ-računu definiramo funkciji

$$\mathsf{true} := \lambda xy\,.\,x \qquad \text{in} \qquad \mathsf{false} := \lambda xy\,.\,y.$$

Katera od naslednjih funkcij predstavlja operacijo xor, torej vrne true, če sta argumenta p in q različni boolovi vrednosti, in false, če sta enaki boolovi vrednosti:

1. $\lambda p\,q\,.\,p\,(q\,\mathsf{false}\,\mathsf{true})\,q$
2. $\lambda p\,q\,.\,p\,q\,p$
3. $\lambda p\,q\,.\,p\,q\,q$
4. $\lambda p\,q\,.\,(q\,\mathsf{false}\,\mathsf{true})\,p\,q$

b) (8 točk) Obravnavamo naslednje izjave o pravilnosti programa P:

(1) $\{n = 1\}\ P\ \{\mathsf{false}\}$

(2) $[n = 1]\ P\ [\mathsf{false}]$

(3) $\{n = 1\}\ P\ \{\mathsf{true}\}$

(4) $[n = 1]\ P\ [\mathsf{true}]$

1. če je n=1 in se P ustavi sledi false sepravi maš opcijo da se P samo ne ustavi

2. če je n=1 se P ustavi sledi false, seprav n != 1 ker če bi bil =1 bi se pač ustavil in bi blo narobe

ter naslednje razlage:

(A) Če je $n = 1$, se program P ustavi.

(B) Če je $n = 1$, se program P ne ustavi.

(C) $n \neq 1$.

(D) Izjava velja za vse P.

Vsaki od izjav priredite njeno razlago:

(1): B (2): C (3): D (4): A

c) (8 točk) V OCamlu zapišite kako funkcijo, ki ima tip *natanko* `(('a -> 'a) -> 'b) -> 'b`.

d) (8 točk) V jeziku z zapisi uporabljamo podtipe v širino in globino. Relacijo podtip označimo z $\leq$ in definiramo tipa zapisov:

$$\mathsf{type}\ s = \{x : \{a : \mathtt{int}\}, y : \{b : \mathtt{int} \to \{\ \}\}\}$$
$$\mathsf{type}\ u = \{x : \{a : \mathtt{int}\}\}.$$

Zapišite tak tip t, da bo veljalo $s \leq t \leq u$, hkrati pa $t \neq s$ in $t \neq u$.

t = {x: {a: int}, y: {}}

e) (8 točk) Usmerjen graf z vozlišči v prologu predstavimo s predikatom `povezava/2`, kjer `povezava(X,Y)` pomeni, da od vozlišča `X` do vozlišča `Y` poteka povezava.

Dan je naslednji graf z vozlišči a, b, c, d, e:

```
povezava(a, b).
povezava(b, c).
povezava(c, a).
povezava(c, d).
povezava(d, e).
povezava(e, c).
povezava(b, e).
```

Trikotnik je taka trojica vozlišč X, Y, Z, da je X povezan z Y, Y z Z in Z z X. V prologu definiramo trikotnik takole:

```
trikotnik(X,Y,Z) :-
  povezava(X, Y),
  povezava(Y, Z),
  povezava(Z, X).
```

Koliko trikotnikov najde v zgornjemm grafu poizvedba

```
?- trikotnik(X,Y,Z).
```

Odgovor: 6

2. naloga (30 točk)

Elbonijci so se naveličali sintakse artimetičnih izrazov. Te dni jih bolj zanimajo *neprazna* dvojiška zaporedja ničel in enic. Ker so vraževerni, verjamejo, da so uročena vsa zaporedja, ki vsebujejo tri zaporedne ničle. Na primer, zaporedja 000, 100011, 10010000 so uročena, medtem ko zaporedja 1, 10, 00100, 00111 niso uročena.

Slovnična pravila za *vsa* neprazna zaporedja lahko predstavimo s slovnico:

$$\langle \text{zaporedje} \rangle ::= \texttt{0} \mid \texttt{1} \mid \texttt{0}\langle \text{zaporedje} \rangle \mid \texttt{1}\langle \text{zaporedje} \rangle$$

V svojih rešitvah smete privzeti slovnično pravilo $\langle$zaporedje$\rangle$.

a) (15 točk) Zapišite slovnična pravila za *uročena* neprazna zaporedja, se pravi taka, ki vsebujejo tri zaporedne ničle.

b) (15 točk) Zapišite slovnična pravila za *neuročena* neprazna zaporedja, se pravi taka, ki *ne* vsebujejo treh zaporednih ničel.

3. naloga (30 točk)

Timotej se ukvarja z optimizacijo preprostega zbirnika, ki ima samo dva ukaza:

1. ukaz `MOV` i, j prebere vrednost pomnilniške lokacije i in jo zapiše v pomnilniško lokacijo j,
2. ukaz `ADD` i, j, k prebere vrednosti pomnilniških lokacij i in j ter njuno vsoto zapiše v lokacijo k.

Pomnilniške lokacije naslavljamo z nenegativni celimi števili. Na primer, če je stanje pomnilnika $[1, 2, 5]$ in izvedemo program

`MOV` $1, 0$
`ADD` $2, 2, 1$
`ADD` $0, 1, 2$

dobimo pomnilnik $[2, 10, 12]$.

V OCamlu predstavimo ukaz z vrednostjo tipa

```
type instruction =
  | Mov of int * int
  | Add of int * int * int
```

in pomnilnik s seznamom celih števil.

a) (10 točk) Sestavite funkcijo

```
val writes_to : int -> instruction -> bool
```

kjer `writes_to k c` ugotovi, ali ukaz `c` zapiše vrednost v lokacijo `k`, se pravi, da je oblike `MOV` i, k ali `ADD` i, j, k.

b) (10 točk) Sestavite funkcijo

```
val reads_from : int -> instruction -> bool
```

kjer `reads_from k c` ugotovi, ali ukaz `c` prebere vrednost lokacije `k`, se pravi, da je oblike `MOV` k, j ali `ADD` i, k, j ali `ADD` k, i, j.

c) (10 točk) Timotej je ugotovil, da lahko zaporedje ukazov optimizira tako, da nekatere ukaze zbriše, ne da bi to vplivalo na učinek programa. Postopek optimizacije zaporedja ukazov $[c_1; c_2; \ldots; c_n]$ je naslednji:

1. Optimizramo rep zaporedja $[c_2; \ldots; c_n]$ in dobimo zaporedje $[c_2'; \ldots; c_m']$.
2. Denimo, da ukaz c_1 zapiše vrednost v lokacijo k. Tedaj:
   (a) če c_2' bere z lokacije k, potem ukaza c_1 ne smemo odstraniti s seznama, sicer
   (b) če c_2' piše na lokacijo k, potem smemo c_1 odstraniti, sicer
   (c) je c_2' neodvisen od lokacije k, zato preverimo iste pogoje za $[c_3'; \ldots; c_m']$.

Na primer, zgornji postopek optimizira ukaze na levi v ukaze na desni:

`MOV` $1, 2$		
`MOV` $0, 1$		`MOV` $1, 2$
`MOV` $4, 4$		`MOV` $4, 4$
`MOV` $1, 3$	$\longrightarrow$	`ADD` $0, 0, 3$
`ADD` $0, 0, 3$		`MOV` $2, 1$
`MOV` $2, 1$		

Sestavite funkcijo

```
val optimize : instruction list -> instruction list
```

ki optimizira seznam ukazov glede na zgornja pravila.

```
type instruction =
  | Mov of int * int
  | Add of int * int * int

let write_to k = function
  | Add(_ ,_ ,z) -> if z = k then true else false
  | Mov(_, z) -> if z = k then true else false

let reads_from k = function
  | Add(z, i, j) -> if z = k || i = k then true else false
  | Mov(z, i) -> if z = k then true else false

let rec odstrani exp1 = function
  | Add(_, _, k) -> write_to k exp1
  | Mov(_, k) -> write_to k exp1

let rec neodstrani exp1 = function
  | Add(_, _, k) -> reads_from k exp1
  | Mov(_, k) -> reads_from k exp1

let rec optimize = function
  | [] -> []
  | h :: [] -> [h]
  | h :: t ->
      let rec preveri h = function
        | h1 :: t1 -> if neodstrani h1 h then h :: List.rev(h1 :: t1)
            else if odstrani h1 h then List.rev(h1 :: t1)
            else (preveri h t1) @ [h1]
        | [] -> [h]
      in preveri h (List.rev (optimize t))
```