

--	--	--	--	--	--	--	--

Vpisna številka

1	
2	
3	
$\Sigma$	

## 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 mobilni telefon izklopljen in spravljen v torbi.
  - Prijavite se na spletno učilnico, kamor boste oddajali nekatere odgovore.
- Dovoljeni pripomočki: pisalo, brisalo, gradivo predčasno naloženo na spletni učilnici, in poljubno pisno gradivo.
- Rešitve vpisujte v polo ali jih oddajte preko spletne učilnice. Pri odgovorih, ki ste jih oddali na spletni učilnici, na poli zapišite “*odgovor je v datoteki <ime\_datoteke>*”.
- Če kaj potrebujete, prosite asistenta in ne sosedov.
- **Med izpitom ne zapuščajte svojega mesta** brez dovoljenja.
- Testna pola vam bo odvzeta **brez nadaljnjih opozoril**, če:
  - komunicirate s komerkoli (ali LLM-i), 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 pomagате komu prepisovati,
  - imate na vidnem mestu mobilni telefon 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.  $\geq 90$  točk, ocena 10
  2.  $\geq 80$  točk, ocena 9
  3.  $\geq 70$  točk, ocena 8
  4.  $\geq 60$  točk, ocena 7
  5.  $\geq 50$  točk, ocena 6

Veliko uspeha!

**1. naloga (35 točk)**

**a) (7 točk)** V Vzhodni Elboniji aritmetično operacijo zapišejo v krogu *pred* argumente, v Zahodni Elboniji pa *za* argumente. Na primer,  $(3 + 5) \times (5 - 2)$  na vzhodu pišejo  $\otimes \oplus 3\ 5 \ominus 5\ 2$  in na zahodu  $3\ 5 \oplus 5\ 2 \ominus \otimes$ .

Ko sta se vzhodna in zahodna Elbonija združili v Veliko Elbonijo, so v imenu elbonske sloge oba zapisa kar združili in dovolili oba hkrati:

$$\begin{aligned} \langle \text{izraz} \rangle &::= \langle \text{število} \rangle \\ &\quad | \oplus \langle \text{izraz} \rangle \langle \text{izraz} \rangle \quad | \ominus \langle \text{izraz} \rangle \langle \text{izraz} \rangle \quad | \otimes \langle \text{izraz} \rangle \langle \text{izraz} \rangle \\ &\quad | \langle \text{izraz} \rangle \langle \text{izraz} \rangle \oplus \quad | \langle \text{izraz} \rangle \langle \text{izraz} \rangle \ominus \quad | \langle \text{izraz} \rangle \langle \text{izraz} \rangle \otimes \\ \langle \text{število} \rangle &::= [0-9]^+ \end{aligned}$$

Ker pa je bila tako nastala slovnica dvoumna, so se v parlamentu skregali in kmalu je Velika Elbonija razpadla na Severno in Južno Elbonijo.

Zapišite niz znakov, ki ga lahko z zgornjo slovnico razčlenimo na dva različna načina, in za vsakega od njiju narišite pripadajoče sintaktično drevo.

**b) (7 točk)** V  $\lambda$ -računu definiramo izraze

$$A := \lambda x\ y\ z.\ x, \quad B := \lambda x\ y\ z.\ y \quad \text{in} \quad C := \lambda x\ y\ z.\ z.$$

Poiščite izraz  $F$ , za katerega velja

$$F\ A = B, \quad F\ B = C \quad \text{in} \quad F\ C = A.$$

c) (7 točk) V programskem jeziku s parametričnim polimorfizmom izračunajte *glavni* tip izraza

```
fun f -> fun (x,y) -> (f x, f x y)
```

Razviden naj bo tudi postopek reševanja.

d) (7 točk) Andrej je v OCamlu definiriral podatkovni tip dreves in funkcije:

```
type 'a drevo = Prazno | List of 'a | Veja of 'a drevo * 'a drevo

let rec listi = function
  | Prazno -> []
  | List x -> [List x]
  | Veja (l, r) -> listi l @ listi r

let rec zdruzi = function
  | [] -> []
  | [t] -> [t]
  | t1 :: t2 :: ts -> Veja (t1, t2) :: zdruzi ts

let rec drevo = function
  | [] -> Prazno
  | [t] -> t
  | ts -> drevo (zdruzi ts)

let krneki t = drevo (listi t)
```

Kaj počne funkcija `krneki`? Označite pravilni odgovor:

- (i) `krneki` je identiteta,
- (ii) `krneki` vrne seznam listov danega drevesa,
- (iii) `krneki` drevo z  $n$  listi preuredi v drevo z istimi listi in globino  $\Theta(\log n)$ ,
- (iv) `krneki` drevo z  $n$  listi preuredi v drevo z istimi listi in globino  $\Theta(\sqrt{n})$ .

**e) (7 točk)** V programskem jeziku s funkcijskimi tipi, tipi zapisov ter podtipi po širini in globini podajte tipe  $\tau$ ,  $\sigma$  in  $\rho$ , da velja

$$\tau \leq \rho, \quad \sigma \leq \rho, \quad \tau \not\leq \sigma \quad \text{in} \quad \sigma \not\leq \tau.$$

Pozor, nikjer nismo zapisali, da programski jezik vsebuje `int` in `float`! Uporabiti smete *samo* funkcijske tipe in tipe zapisov.

## 2. naloga (35 točk)

a) (20 točk) Dokažite *delno* pravilnost programa

```
{ n > 0 }  
i := 1 ;  
k := 0 ;  
while k = 0 do  
  if i = n then  
    k := 1  
  else  
    i := i + 1  
  end  
done  
{ i = n }
```

Vse spremenljivke zavzemajo celoštevilске vrednosti. Poskrbite, da bo v vaši rešitvi razvidna invarianta zanke.

Opomba: pazite, kako pišete logične izraze – konjunkcija  $\wedge$  ima prednost pred disjunkcijo  $\vee$ , se pravi  $p \wedge q \vee r = (p \wedge q) \vee r$ .

b) (15 točk) Ali velja tudi popolna pravilnost? Odgovor utemeljite.

### 3. naloga (40 točk)

Stanje celičnega avtomata je podano s seznamom ničel in enic, katerega elemente imenujemo *celice*. Vsaka celica ima levo in desno sosedo, pri čemer štejemo, da je zadnja celica soseda prve in prva celica desna soseda zadnje (seznam sklenemo v cikel).

Celice v celičnem avtomatu se spreminjajo skladno s pravili oblike  $[a, b, c] \mapsto d$ , ki pomenijo: če nastopijo v seznamu zaporedne celice  $a$ ,  $b$ ,  $c$ , se  $b$  spremeni v  $d$ . V prologu zapišemo tako pravilo s predikatom `rule([a, b, c], d)`. Primer avtomata:

```
rule([0,0,0], 0).
rule([0,0,1], 1).
rule([0,1,0], 1).
rule([0,1,1], 1).
rule([1,0,0], 1).
rule([1,0,1], 0).
rule([1,1,0], 0).
rule([1,1,1], 0).
```

Na primer, stanje `[0,0,1,1,0,1,0]` se z zgornjimi pravili pretvori v `[0,1,1,0,0,1,1]`. Ko pravila znova in znova uporabljamo, dobimo zaporedna stanja:

```
[0,0,1,1,0,1,0]
[0,1,1,0,0,1,1]
[0,1,0,1,1,1,0]
[1,1,0,1,0,0,1]
[0,0,0,1,1,1,1]
[1,0,1,1,0,0,0]
```

**a) (10 točk)** Sestavite predikat `pad(In,Out)`, ki velja takrat, ko dobimo seznam `Out` iz seznama `In` tako, da mu dodamo na začetek zadnji in na konec prvi element. Tako nastalemu seznamu `Out` pravimo *dopolnjeno stanje*. Primeri:

```
?- pad([0,0,0,1], Out).
Out = [1, 0, 0, 0, 1, 0].
?- pad([1], Out).
false.
?- pad([], Out).
false.
```

**b) (10 točk)** Sestavite predikat `change(PaddedState,NextState)`, ki iz *dopolnjenega stanja* `PaddedState` izračuna naslednje stanje `NextState`. Primeri:

```
?- gen([0,0,0,1,0,0,0], NextState).
NextState = [0, 1, 1, 1, 0].
?- gen([0,1,1,1,0], NextState).
NextState = [1, 0, 0].
?- gen([1,0,0], NextState).
NextState = [1].
?- gen([1,0], NextState).
NextState = [].
?- gen([1], NextState).
false.
```

**c) (10 točk)** Sestavite predikat `next(State,NextState)`, ki iz (nedopolnjenega) stanja `State` izračuna naslednje stanje `NextState`. Primeri:

```
?- next([0,0,1,1,0,1,0], S).
S = [0, 1, 1, 0, 0, 1, 1].
?- length(S,7), next(S, [0,0,0,1,0,0,0]).
S = [1, 1, 0, 1, 1, 1, 1].
```

**d) (10 točk)** Sestavite predikat `history(N,L)`, ki velja, ko `L` predstavlja `N` zaporednih stanj.  
Primeri:

```
?- history(5, [[0,0,1,0,0]|Cont]).  
Cont = [[0, 1, 1, 1, 0], [1, 1, 0, 0, 1], [0, 0, 1, 1, 1], [1, 1, 1, 0, 0]].
```

Namig: nastavitev

```
?- set_prolog_flag(answer_write_options,[max_depth(0)]).
```

prologu pove, da mora prikazati sezname v celoti.