

Ime in priimek

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

Vpisna številka

1	
2	
3	
Σ	

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.
- 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 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. ≥ 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 predložite 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: $A =$ _____

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) -> _____
```

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

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

Odgovor: _____

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`.

```
{ 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`.

```
[true]
```

```
x := a ;
```

```
i := 0 ;
```

```
while i < 100 do
```

```
  x := x * x * a ;
```

```
  i := i + 1
```

```
end
```

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

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
```

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, _____) .
```