

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

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, USB ključ, in poljubno pisno gradivo.
- Rešitve vpisujte v polo ali jih oddajte preko spletne učilnice. Pri odgovorih, ki ste jih oddali preko spletne učilnice, na izpitno nalogo napišite "glej spletno učilnico – datoteka <ime\_datoteke>".
- 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.  $\geq 90$  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 (42 točk)

Programerji elbonijskega ministrstva za digitalizacijo so v OCamlu sestavili leksični analizator `elb_lexer` in razčlenjevalnik `elb_parser`:

```
type lexeme = PLUS | MINUS | TIMES | CONST of int

type expr =
  | Num of int
  | Add of expr * expr
  | Sub of expr * expr
  | Mul of expr * expr

let elb_lexer (string_expression : string) : lexeme list =
  let f = function
    | "-" -> MINUS
    | "+" -> PLUS
    | "*" -> TIMES
    | n -> CONST (int_of_string n)
  in
  List.map f
    (List.filter ((<>) "") (String.split_on_char ' ' string_expression))

let elb_parser (lexemes : lexeme list) : expr =
  let rec loop stack lexemes =
    match stack, lexemes with
    | s,          CONST n :: rest -> loop (Num n :: s) rest
    | e1 :: e2 :: s, PLUS      :: rest -> loop (Add (e2, e1) :: s) rest
    | e1 :: e2 :: s, MINUS     :: rest -> loop (Sub (e2, e1) :: s) rest
    | e1 :: e2 :: s, TIMES     :: rest -> loop (Mul (e2, e1) :: s) rest
    | [e], [] -> e
    | _ -> failwith "cannot parse"
  in
  loop [] lexemes
```

Na žalost so izgubili celotno dokumentacijo in sedaj ne vedo, kaj so pravzaprav implementirali.

**a) (7 točk)** Narišite drevo abstraktne sintakse, ki ga dobimo, ko razčlenimo niz

```
"100 2 3 + + 4 - 5 666 - *"
```

c) (7 točk) V Haskellu ali OCamlu implementirajte katerokoli funkcijo, ki ima glavni tip

$$('a \rightarrow 'b \rightarrow 'c) \rightarrow 'b \rightarrow 'a \rightarrow 'c$$

```
module type COMBINATORS =
sig
  val succ : int -> int
  val const : 'a -> 'b -> 'a
  val flip : ('a -> 'b -> 'c) -> 'b -> 'a -> 'c
  val negate : ('a -> bool) -> 'a -> bool
end
```

3

e) (7 točk) V  $\lambda$ -računu definiramo Churchova števila takole:

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

Zapišite  $\lambda$ -izraz **Even**, da za vsak  $n \in \mathbb{N}$  velja

$$\text{Even } \bar{n} = \begin{cases} \text{true} & \text{če je } n \text{ sodo število,} \\ \text{false} & \text{če je } n \text{ liho število.} \end{cases}$$

Pri tem je  $\text{true} := \lambda x y . x$  in  $\text{false} := \lambda x y . y$ . Na primer  $\text{Even } \bar{42} = \text{true}$ .

f) (7 točk) V  $\lambda$ -računu definiramo Scott-Churchova števila takole:

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

Zapišite  $\lambda$ -izraz **Odd**, da za vsak  $n \in \mathbb{N}$  velja

$$\text{Odd } \hat{n} = \begin{cases} \text{true} & \text{če je } n \text{ liho število,} \\ \text{false} & \text{če je } n \text{ sodo število.} \end{cases}$$

Pri tem je  $\text{true} := \lambda x y . x$  in  $\text{false} := \lambda x y . y$ . Na primer  $\text{Odd } \hat{42} = \text{false}$ .

## 2. naloga (30 točk)

a) (20 točk) Andrej je zapisal program, kjer sta  $p, q \in \mathbb{Z}$ :

```
x := p ;
y := q ;
while not (x = y) do
  x := x - 1 ;
  y := y + 1
done
```

Dokažite *delno* pravilnost programa  $P$  glede na spodnjo specifikacijo. Iz rešitve naj bo jasno razvidna invarianta zanke `while`. Operator  $\div$  predstavlja celoštevilsko deljenje in velja  $p, q \in \mathbb{Z}$ .

$\{ \text{true} \}$

$x := p ;$

$y := q ;$

`while not (x = y) do`

$x := x - 1 ;$

$y := y + 1$

`done`

$\{x = (p + q) \div 2\}$

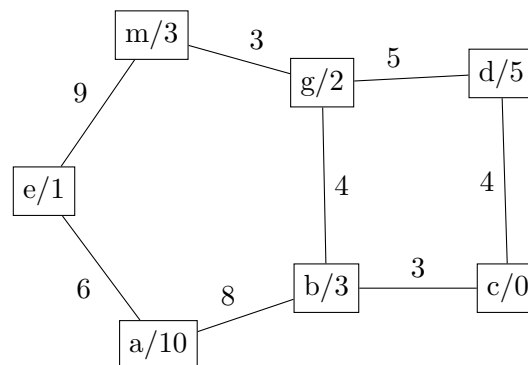
**b) (10 točk)** Ali program  $P$  zagotovo ustreza tudi naslednji specifikaciji za *popolno* pravilnost?

```
[  $p \geq q$  ]  
x := p ;  
y := q ;  
while not (x = y) do  
  x := x - 1 ;  
  y := y + 1  
done  
[  $x = (p + q) \div 2$  ]
```

Odgovor utemeljite z dokazom ali podajte protiprimer.

### 3. naloga (38 točk)

Klemen je pred spanjem pogledal vse epizode »Black mirror«, zdaj pa ga tlači nočna mora, v kateri se vozi z avtom po cestnem omrežju, prikazanem na sliki.



Oznake na povezavah so dolžine *dvosmernih* cest, ki povezujejo vozlišča. V vsakem vozlišču je zaloga goriva, na primer v vozlišču d je 5 enot goriva. Preden se Klemen odpelje iz vozlišča, vedno dotoči vse razpoložljivo gorivo. V sanjah ima avto dovolj velik rezervoar, da se nikoli ne napolni do konca. Avto porabi eno enoto goriva za eno enoto razdalje. Primer: če ima avto 2 enoti goriva v rezervoarju, se lahko pelje po poti a–b–c, ne more pa se peljati po poti a–e–m, ker mu za povezavo e–m zmanjka goriva.

Da bo nočna mora popolna, bomo Klemna usmerjali s prologom. Začetne zaloge goriva v vozliščih in cestne povezave predstavimo s predikatoma `zacetne_zaloge` in `cesta`:

```
zacetne_zaloge([a/10, b/3, c/0, d/5, e/1, m/3, g/2]).
```

```
cesta(a, b, 8).
cesta(b, c, 3).
cesta(c, d, 4).
cesta(a, e, 6).
cesta(e, m, 9).
cesta(m, g, 3).
cesta(g, d, 5).
cesta(g, b, 4).
```

Pozor, `cesta` navaja samo po eno od obeh smeri.

**a) (8 točk)** Sestavite predikat `sprazni(V, Z1, Z2)`, ki sprazni zaloge goriva v vozlišču V, pri čemer so Z1 trenutne zaloge in Z2 zaloge, ko spraznimo V. Primer:

```
?- sprazni(c, [a/5, b/3, c/4, d/8], Z2).
Z2 = [a/5, b/3, c/0, d/8].
```

**b) (8 točk)** Sestavite predikat `natoci(V, G1, Z1, G2, Z2)`, ki sprejme vozlišče `V`, trenutno gorivo v avtu `G1` in zaloge `Z1`. Gorivo iz vozlišča `V` pretoči v avto, da dobi novo stanje goriva `G2` in zaloge `Z2`. Primer:

```
?- natoci(c, 4, [a/5, b/3, c/4, d/8], G2, Z2).
G2 = 8, Z2 = [a/5, b/3, c/0, d/8].
```

**c) (8 točk)** Sestavite predikat `etapa(V1, G1, Z1, V2, G2, Z2)`, ki prevozi cesto med `V1` in `V2`, če ta obstaja in če ima avto dovolj goriva. Pred etapo dotoči zalogo iz `V1`. Pri tem sta `G1` in `Z1` začetno gorivo v avtu in vozliščih, ter `G2` in `Z2` končno gorivo v avtu in vozliščih. Primeri:

```
?- etapa(a, 5, [a/10, b/3, c/0, d/5, e/1, m/3, g/2], b, G2, Z2).
G2 = 7, Z2 = [a/0, b/3, c/0, d/5, e/1, m/3, g/2] .
?- etapa(a, 5, [a/10, b/3, c/0, d/5, e/1, m/3, g/2], c, G2, Z2).
false.
?- etapa(e, 5, [a/10, b/3, c/0, d/5, e/1, m/3, g/2], m, G2, Z2).
false.
?- etapa(e, 9, [a/10, b/3, c/0, d/5, e/1, m/3, g/2], m, G2, Z2).
G2 = 1, Z2 = [a/10, b/3, c/0, d/5, e/0, m/3, g/2] .
```

**d) (8 točk)** Sestavite predikat `pot(L, G1, Z1, G2, Z2)`, ki prevozi pot, navedeno v seznamu vozlišč `L`, če je to možno. Pri tem sta `G1` in `Z1` začetno gorivo v avtu in vozliščih, ter `G2` in `Z2` končno gorivo v avtu in vozliščih. Pozor, pot lahko vodi večkrat skozi isto vozlišče. Primeri:

```
?- pot([a], 0, [a/10, b/3, c/0, d/5, e/1, m/3, g/2], G2, Z2).
G2 = 0, Z2 = [a/10, b/3, c/0, d/5, e/1, m/3, g/2] .
?- pot([a, b, c, d], 0, [a/10, b/3, c/0, d/5, e/1, m/3, g/2], G2, Z2).
false.
?- pot([a, b, c, d], 5, [a/10, b/3, c/0, d/5, e/1, m/3, g/2], G2, Z2).
G2 = 3, Z2 = [a/0, b/0, c/0, d/5, e/1, m/3, g/2] .
```

**e) (6 točk)** Navedite poizvedbo, ki ugotovi, ali lahko avto prevozi pot `a-e-m-g-b-c-d-g-b-a`, če je na začetku v rezervoarju 22 enot goriva. Začetne zaloge v vozliščih so navedene s predikatom `zacetne_zaloge`. Kakšen je odgovor?