

# Principi programskih jezikov

3. izpit, 9. september 2024

Ime in priimek

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

Vpisna številka

1	
2	
3	
4	
$\Sigma$	

## 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 spravljn v torbi.
  - Prijavite se na spletno učilnico, kamor boste oddajali nekatere odgovore.
- Dovoljeni pripomočki: pisalo, brisalo, gradivo predčasno naloženo na e-učilnici, 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 pomagate 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 (30 točk)

a) (10 točk) Izračunajte (na dolgo) *glavni* tip naslednjega izraza:

```
let rec f s a b =  
  if a > 0  
  then f ((a <= b) :: s) (a - 1) b  
  else s  
in f []
```

b) (10 točk) Definirajte predikat `combine(Lists, List)`, ki združi seznam seznamov `List`s v en sam seznam `List` **brez uporabe** zunanjih predikatov. Primeri:

```
?- combine([], List).  
List = [].  
?- combine([[1],[2, 3],[4, 5, 6],[]], List).  
List = [1, 2, 3, 4, 5, 6].  
?- combine([[]], List).  
List = [].
```

c) (10 točk) Definirajte predikat `dedup(L, D)`, ki iz *urejenega* seznama `L` odstrani vsa števila, ki se ponovijo sodo mnogokrat. Če se število pojavi liho-krat, odstranimo le njegove ponovitve.

```
?- dedup([1, 1, 2, 3, 3, 3, 4, 5, 5, 5, 5, 5, 6, 6, 7], D).  
D = [2, 3, 4, 5, 7] ;  
false.  
?- dedup([1, 1, 1, 2, 6, 6, 7], D).  
D = [1, 2, 7] ;  
false.  
?- dedup([1, 1, 1], D).  
D = [1] ;  
false.  
?- dedup([1, 1], D).  
D = [] ;  
false.
```

## 2. naloga (35 točk)

a) (20 točk) Dokazite *delno* pravilnost spodnjega programa. Zapišite, katere invariante ste uporabili.

```
{ }  
i = 0 ;  
m = 1 ;  
while i < 32 do  
  i := i + 1 ;  
  m := m * 2  
done  
  
while i != 100 do  
  i := i + 3 ;  
  if i >= m then  
    i := i - m  
  else  
    pass  
  end  
done  
{ i = 100, m = 232 }
```

b) (15 točk) Dokazite ali ovrzite popolno pravilnost programa. Dovolj je le dobra utemeljitev.

### 3. naloga (40 točk)

V Prologu smo na vajah implementirali Turingov stroj. Ideja te naloge je na podoben način implementirati Turingov stroj v OCamlu.

- Stroj začne izvajanje programa v stanju `Q 0`.
- Ko pride v stanje `Final`, se izvajanje konča.
- Neskončen trak (`tape`) stroja je prestavljen z dvema seznamoma.
- V seznamu `left` so celice, ki so levo od celice na katero glava stroja trenutno kaže. Pri tem iz seznama seveda izpustimo neskončno neuporabljenih praznih celic na levi. Zaradi učinkovitejše implementacije so elementi seznama v obratnem vrstnem redu (prvi element seznama je najbližji glavi).
- Prva celica seznama `right` je celica, na katero trenutno kaže glava stroja. Preostali elementi seznama so celice, ki so desno od glave stroja (zopet z izjemo neskončno praznih celic na desni).
- Celica je lahko prazna (`Blank`), hrani vrednost 0 (`Zero`) ali vrednost 1 (`One`). Neobiskane celice ne hranimo.
- Program je opisan s tabelo parov `(state * cell) * (state * cell * direction)`. Zadnji par seznama za program `parity` pravi, da če je Turingov stroj v stanju `Q 1` in če njegova glava gleda na prazno celico (`Blank`) potem bo naslednje stanje `Final`, glava pa bo na trak zapisala `One` in ostala na mestu.

```
type state = Q of int | Final
type direction = Left | Right | Stay
type cell = Blank | Zero | One
type tape = { left : cell list; right : cell list }
type program = ((state * cell) * (state * cell * direction)) list

let copy : program =
[
  ((Q 0, Blank), (Final, Blank, Stay));
  ((Q 0, One), (Q 2, Blank, Right));
  ((Q 2, Blank), (Q 3, Blank, Right));
  ((Q 2, One), (Q 2, One, Right));
  ((Q 3, Blank), (Q 4, One, Left));
  ((Q 3, One), (Q 3, One, Right));
  ((Q 4, Blank), (Q 5, Blank, Left));
  ((Q 4, One), (Q 4, One, Left));
  ((Q 5, Blank), (Q 0, One, Right));
  ((Q 5, One), (Q 5, One, Left));
]
let parity : program =
[
  ((Q 0, Zero), (Q 0, Zero, Right));
  ((Q 0, One), (Q 1, One, Right));
  ((Q 0, Blank), (Final, Zero, Stay));
  ((Q 1, Zero), (Q 1, Zero, Right));
  ((Q 1, One), (Q 0, One, Right));
  ((Q 1, Blank), (Final, One, Stay));
]
let plus1 : program =
[
  ((Q 0, One), (Q 0, One, Right));
  ((Q 0, Blank), (Final, One, Stay));
]

let action direction tape : tape =
  let move_left = function
    | { left = []; right } -> { left = []; right = Blank :: right }
    | { left = h :: t; right } -> { left = t; right = h :: right }
  in
  let move_right = function
    | { left; right = [] } -> { left = Blank :: left; right = [] }
    | { left; right = h :: t } -> { left = h :: left; right = t }
```

```

in
match direction with
| Stay -> tape
| Left -> move_left tape
| Right -> move_right tape

```

a) (6 točk) Implementirajte funkciji za branje celice pod glavo in pisanje v njo: `read : tape -> cell` in `write : cell -> tape -> tape`. Primeri:

```

# read { left = []; right = [Zero; One] };;
- : cell = Zero
# read { left = [Zero]; right = [] };;
- : cell = Blank
# write One { left = [Zero]; right = [] };;
- : tape = {left = [Zero]; right = [One]}
# write One { left = [Zero]; right = [Zero] };;
- : tape = {left = [Zero]; right = [One]}
# write One { left = [Zero]; right = [Blank; Zero] };;
- : tape = {left = [Zero]; right = [One; Zero]}

```

b) (12 točk) Funkcija `action : direction -> tape -> tape` za izvajanje premikov po traku je že implementirana. Implementirajte funkcijo `step : program -> state -> tape -> state * tape`, ki sprejme program, trenutno stanje stroja in trak, vrne pa novo stanje in nov trak po enem izvedenem koraku. Primeri:

```

# step plus1 (Q 0) {left=[]; right=[One; Blank; Blank]};;
- : state * tape = (Q 0, {left = [One]; right = [Blank; Blank]})
# step plus1 (Q 0) {left = [One]; right = [Blank; Blank]};;
- : state * tape = (Final, {left = [One]; right = [One; Blank]})

```

c) (12 točk) Implementirajte funkcijo `run : program -> state -> tape -> tape`, ki požene program iz začetnega stanja in vhodnega traku do konca:

- če je v stanju `Final`, se ustavi;
- sicer izvede en korak in znova pokliče `run`.

Primer:

```
# run plus1 (Q 0) {left=[]; right=[One; One; One]};;
- : tape = {left = [One; One; One]; right = [One]}
```

d) (10 točk) Implementirajte funkcijo `turing : program -> cell list -> cell list`, ki izvede program na vhodnem traku `{left=[]; right=cells}`. Argument tipa `cell list` je torej seznam celic, ki so desno od glave, levo od glave pa je trak prazen. Tudi izhodni seznam celic naj bo navaden seznam vseh simbolov na traku (z izjemo praznih na levi in desni) v pravem vrstnem redu.

Primeri:

```
# turing plus1 [One; One; One];;
- : cell list = [One; One; One; One]

# turing copy [One; One; One];;
- : cell list = [One; One; One; Blank; One; One; One]

# turing parity [One; Zero; One];;
- : cell list = [One; Zero; One; Zero]

# [Zero; Zero; One; One] |> turing parity |> turing parity;;
- : cell list = [Zero; Zero; One; One; Zero; Zero]
```