

# Principi programskih jezikov

1. izpit, 10. junij 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 spravljene 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 (21 točk)**

**a) (7 točk)** V funkcijskem programskem jeziku z zapisi ter podtipi po širini in globini predpostavimo  $\text{bool} \leq \text{int}$ . Dani so naslednji tipi zapisov:

$$\rho = \{a : \text{bool}\} \rightarrow \{b : \text{bool}; c : \text{int}\}$$

$$\sigma = \{a : \text{int}\} \rightarrow \{b : \text{bool}; c : \text{int}\}$$

$$\tau = \{a : \text{bool}\} \rightarrow \{b : \text{int}\}$$

$$\psi = \{\} \rightarrow \{b : \text{int}; c : \text{int}\}$$

V spodnji tabeli za vsak par z DA/NE označite, če je tip v vrstici podtip tipa v stolpcu:

$\leq$	$\rho$	$\sigma$	$\tau$	$\psi$
$\rho$				
$\sigma$				
$\tau$				
$\psi$				

**b) (7 točk)** Standardna knjižnica za OCaml vsebuje **desno**-asociativna binarna operatorja `@@` in `::` z glavnima tipoma:

$$(\texttt{@@}) : (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta$$
$$(\texttt{::}) : \alpha \rightarrow \alpha \texttt{ list} \rightarrow \alpha \texttt{ list}$$

Operator `@@` ima nižjo prioriteto kot operator `::`. Izračunajte (na dolgo) *glavni* tip naslednjega izraza:

```
fun (a, b) -> a @@ a @@ 1 :: b
```

**c) (7 točk)** V  $\lambda$ -računu implementirajte funkcijo `scott`, ki Churchovo število (`0'`, `1'`, `2'`, ...) pretvori v Scott-Churchevo število (`0`, `1`, `2`, ...)

```
0' := ^ f x . x ;
1' := ^ f x . f x ;
2' := ^ f x . f (f x) ;
3' := ^ f x . f (f (f x)) ;
4' := ^ f x . f (f (f (f x))) ;
5' := ^ f x . f (f (f (f (f x)))) ;
succ' := ^ n . ^ f x . f (n f x) ;
```

pretvori v Scott-Churchevo število (`0`, `1`, `2`, ...)

```
0 := ^ f x . x ;
1 := ^ f x . f 0 x ;
2 := ^ f x . f 1 (f 0 x) ;
3 := ^ f x . f 2 (f 1 (f 0 x)) ;
4 := ^ f x . f 3 (f 2 (f 1 (f 0 x))) ;
5 := ^ f x . f 4 (f 3 (f 2 (f 1 (f 0 x)))) ;
succ := ^ n . ^ f x . f n (n f x) ;
```

## 2. naloga (21 točk)

V OCamlu želimo implementirati simulator za Elbonijski skladovni računalnik. Simulator bo izvajal ukaze (instruction), ki se na nahajajo v ROM-u (rom : int -> instruction), in bral števila z vhoda (input : int list) z ukazom READ. Ko se izvajanje programa zaključi z ukazom EXIT, program vrne vrh sklada. Sklad je predstavljen s seznamom (stack : int list), kjer je vrh sklada prvi element seznama. Stanje računalnika je predstavljeno z zapisom machine\_state, ki hrani še neprebran vhod, ROM, sklad in programski števec IP (instruction\_pointer : int).

Nabor ukazov z opisi:

```
type instruction =
| NOOP (* no operation, IP := IP + 1 *)
| PUSH of int (* push integer constant onto stack, IP := IP + 1 *)
| ADD (* push sum of top two elements of stack, IP := IP + 1 *)
| SUB (* push difference of top two elements of stack, IP := IP + 1 *)
| EQ (* push 1 if top two elements of stack are equal otherwise push 0, IP := IP + 1 *)
| LT (* push 1 if top top element of stack is less than the second element
      otherwise push 0, IP := IP + 1 *)
| JMP of int (* relative jump IP := IP + rel *)
| JMPZ of int (* jump if zero on the stack IP := IP + rel, otherwise IP := IP + 1 *)
| EXIT (* stop execution, returning the top element of the stack *)
| READ (* push read integer, IP := IP + 1 *)
| DUP (* duplicate top element on the stack, IP := IP + 1 *)

type machine_state = {
  input : int list;
  rom : int -> instruction;
  stack : int list;
  instruction_pointer : int;
}
```

Dokončajte implementacijo funkcije run : machine\_state -> int. Pomagajte si priloženimi pomožnimi funkcijami (glejte datoteko asm\_unfinished.ml na učilnici):

```
let pop state =
  match state.stack with
  | x :: stack -> (x, { state with stack })
  | [] -> failwith "empty stack"

let read state =
  match state.input with
  | x :: input -> (x, { state with input })
  | [] -> failwith "empty input"

let double_pop state =
  let x1, state = pop state in
  let x2, state = pop state in
  (x1, x2, state)

let push x state = { state with stack = x :: state.stack }

let increment_instruction_pointer i state =
  { state with instruction_pointer = state.instruction_pointer + i }

let string_of_instruction = function
| NOOP -> "NOOP" | ADD -> "ADD" | SUB -> "SUB" | EQ -> "EQ"
| LT -> "LT" | EXIT -> "EXIT" | READ -> "READ" | DUP -> "DUP"
| PUSH i -> "PUSH " ^ string_of_int i
| JMP i -> "JMP " ^ string_of_int i
| JMPZ i -> "JMPZ " ^ string_of_int i

let print_debug_info { input; rom; stack; instruction_pointer } =
  Printf.printf "INFO: IP = %i (%s), stack = [%s], input = [%s]\n"
    instruction_pointer
    (string_of_instruction @@ rom instruction_pointer)
    (String.concat " ", " @@ List.map string_of_int stack)
    (String.concat " ", " @@ List.map string_of_int input)
```

```

let rec run (state : machine_state) =
  print_debug_info state;
  let incr = increment_instruction_pointer 1 in
  match state.rom state.instruction_pointer with
  | NOOP -> run (incr state)
  | _ -> failwith "not implemented"

let romA i = [| PUSH 0; READ; DUP; JMPZ 3; ADD; JMP (-4); JMPZ 1; EXIT |].(i)

let computerA input =
  run { rom = romA; stack = []; input; instruction_pointer = 0 }

```

Primeri izvajanja:

```

# run { rom = (fun i -> [| PUSH 42; EXIT |].(i)); stack = []; input = [1; 2];
  instruction_pointer = 0 } ;;
INFO: IP = 0 (PUSH 42), stack = [], input = [1; 2]
INFO: IP = 1 (EXIT), stack = [42], input = [1; 2]
- : int = 42

# run { rom = (fun i -> [| READ; EXIT |].(i)); stack = []; input = [1; 2];
  instruction_pointer = 0 } ;;
INFO: IP = 0 (READ), stack = [], input = [1; 2]
INFO: IP = 1 (EXIT), stack = [1], input = [2]
- : int = 1

# run { rom = (fun i -> [| ADD; EXIT |].(i)); stack = [-1; 1; 2]; input = [];
  instruction_pointer = 0 } ;;
INFO: IP = 0 (ADD), stack = [-1; 1; 2], input = []
INFO: IP = 1 (EXIT), stack = [0; 2], input = []
- : int = 0

# run { rom = (fun i -> [| SUB; EXIT |].(i)); stack = [-1; 1; 2]; input = [];
  instruction_pointer = 0 } ;;
INFO: IP = 0 (SUB), stack = [-1; 1; 2], input = []
INFO: IP = 1 (EXIT), stack = [-2; 2], input = []
- : int = -2

# run { rom = (fun i -> [| EQ; EXIT |].(i)); stack = [-1; 1; 2]; input = [];
  instruction_pointer = 0 } ;;
INFO: IP = 0 (EQ), stack = [-1; 1; 2], input = []
INFO: IP = 1 (EXIT), stack = [0; 2], input = []
- : int = 0

# run { rom = (fun i -> [| EQ; EXIT |].(i)); stack = [1; 1; 2]; input = [];
  instruction_pointer = 0 } ;;
INFO: IP = 0 (EQ), stack = [1; 1; 2], input = []
INFO: IP = 1 (EXIT), stack = [1; 2], input = []
- : int = 1

# run { rom = (fun i -> [| EQ; EXIT |].(i)); stack = [1; 1; 2]; input = [];
  instruction_pointer = 0 } ;;
INFO: IP = 0 (EQ), stack = [1; 1; 2], input = []
INFO: IP = 1 (EXIT), stack = [1; 2], input = []
- : int = 1

# run { rom = (fun i -> [| LT; EXIT |].(i)); stack = [-1; 1; 2]; input = [];
  instruction_pointer = 0 } ;;
INFO: IP = 0 (LT), stack = [-1; 1; 2], input = []
INFO: IP = 1 (EXIT), stack = [1; 2], input = []
- : int = 1

# run { rom = (fun i -> [| JMP 2; READ; EXIT |].(i)); stack = []; input = [];
  instruction_pointer = 0 } ;;
INFO: IP = 0 (JMP 2), stack = [], input = []
INFO: IP = 2 (EXIT), stack = [], input = []
Exception: Failure "empty stack".

# run { rom = (fun i -> [| JMP 2; READ; EXIT |].(i)); stack = [100]; input = [];
  instruction_pointer = 0 } ;;

```

```

INFO: IP = 0 (JMP 2), stack = [100], input = []
INFO: IP = 2 (EXIT), stack = [100], input = []
- : int = 100

# run { rom = (fun i -> [| JMPZ 2; READ; EXIT |].(i)); stack = [0; 100]; input = [-1];
  instruction_pointer = 0 } ;;
INFO: IP = 0 (JMPZ 2), stack = [0; 100], input = [-1]
INFO: IP = 2 (EXIT), stack = [100], input = [-1]
- : int = 100

# run { rom = (fun i -> [| JMPZ 2; READ; EXIT |].(i)); stack = [3; 100]; input = [-1];
  instruction_pointer = 0 } ;;
INFO: IP = 0 (JMPZ 2), stack = [3; 100], input = [-1]
INFO: IP = 1 (READ), stack = [100], input = [-1]
INFO: IP = 2 (EXIT), stack = [-1; 100], input = []
- : int = -1

# run { rom = (fun i -> [| DUP; EXIT |].(i)); stack = [3; 100]; input = [];
  instruction_pointer = 0 } ;;
INFO: IP = 0 (DUP), stack = [3; 100], input = []
INFO: IP = 1 (EXIT), stack = [3; 3; 100], input = []
- : int = 3

# computerA [42; -2; 60; 0; 1337];;
INFO: IP = 0 (PUSH 0), stack = [], input = [42; -2; 60; 0; 1337]
INFO: IP = 1 (READ), stack = [0], input = [42; -2; 60; 0; 1337]
INFO: IP = 2 (DUP), stack = [42; 0], input = [-2; 60; 0; 1337]
INFO: IP = 3 (JMPZ 3), stack = [42; 42; 0], input = [-2; 60; 0; 1337]
INFO: IP = 4 (ADD), stack = [42; 0], input = [-2; 60; 0; 1337]
INFO: IP = 5 (JMP -4), stack = [42], input = [-2; 60; 0; 1337]
INFO: IP = 1 (READ), stack = [42], input = [-2; 60; 0; 1337]
INFO: IP = 2 (DUP), stack = [-2; 42], input = [60; 0; 1337]
INFO: IP = 3 (JMPZ 3), stack = [-2; -2; 42], input = [60; 0; 1337]
INFO: IP = 4 (ADD), stack = [-2; 42], input = [60; 0; 1337]
INFO: IP = 5 (JMP -4), stack = [40], input = [60; 0; 1337]
INFO: IP = 1 (READ), stack = [40], input = [60; 0; 1337]
INFO: IP = 2 (DUP), stack = [60; 40], input = [0; 1337]
INFO: IP = 3 (JMPZ 3), stack = [60; 60; 40], input = [0; 1337]
INFO: IP = 4 (ADD), stack = [60; 40], input = [0; 1337]
INFO: IP = 5 (JMP -4), stack = [100], input = [0; 1337]
INFO: IP = 1 (READ), stack = [100], input = [0; 1337]
INFO: IP = 2 (DUP), stack = [0; 100], input = [1337]
INFO: IP = 3 (JMPZ 3), stack = [0; 0; 100], input = [1337]
INFO: IP = 6 (JMPZ 1), stack = [0; 100], input = [1337]
INFO: IP = 7 (EXIT), stack = [100], input = [1337]
- : int = 100

```

### 3. naloga (28 točk)

a) (21 točk) Razširjeno Fibonaccijevo zaporedje  $\dots, F(-3), F(-2), F(-1), F(0), F(1), \dots$  je podano z rekurzivno definicijo

$$F(0) = 0, \quad F(1) = 1, \quad F(i+2) = F(i+1) + F(i).$$

$i$	$\dots$	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	$\dots$
$F(i)$	$\dots$	13	-8	5	-3	2	-1	1	0	1	1	2	3	5	8	13	$\dots$

Dokažite *delno* pravilnost programa, kjer je  $N \in \mathbb{Z}$ :

```
{ true }
x := 1 ;
y := 0 ;
z := 0 ;
if N > 0 then
  while not (y = N) do
    z := z + 1 ;
    w := x ;
    x := y ;
    y := y + w ;
  done
else
  while not (y = N) do
    z := z - 1 ;
    w := y ;
    y := x ;
    x := w - x ;
  done
end
{ N = F(z) }
```

b) (7 točk) Dokažite ali ovrzite popolno pravilnost programa.

#### 4. naloga (35 točk)

Elbonijski gradbeni inženir je dobil načrte za izdelavo ječ. Ječa je pravokotni kletni prostor z “debelimi” zidovi (glej slike). Inženir je zmeden, saj mu je arhitekt podal le število potrebnih zidov v posamezni “vrstici/stolpcu”, ter kje bodo zveri/zakladi. Zaradi načina gradnje se dva zidova ne moreta dotikati le v ogliščih. Na spodnjih dveh slikah lahko vidimo možno postavitev zidov za dva različna “načrta”. S pomočjo Prologa mu olajšajte delo.



a) (14 točk) S pomočjo predikata `nice_corners/4`, ki velja, če

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \neq \begin{bmatrix} A & B \\ C & D \end{bmatrix} \neq \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

```
nice_corners(A, B, C, D) :-  
    (A = B; C = D, A = C; B = D), !.
```

definirajte predikat `nice_corners(M)`, ki velja takrat, ko matrika (seznam seznamov) `M` ne vsebuje podmatrik  $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$  in  $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ . Primeri:

```
?- nice_corners([[0,1], [0,1]]).  
true ;  
false.  
?- nice_corners([[0,1], [1,0]]).  
false.  
?- nice_corners([[1,0], [0,1]]).  
false.  
?- nice_corners([[1,1,0], [0,1,0], [0,1,1]]).  
true ;  
false.  
?- nice_corners([[1,1,0], [0,1,0], [0,1,1]]).  
true ;  
false.  
?- nice_corners([[1,0,1], [0,1,0], [0,1,1]]).  
false.  
?- nice_corners([[1,0,1], [0,0,1], [0,1,0]]).  
false.
```

Lahko si pomagata s pomožnimi predikati.



b) (21 točk) Implementirajte predikat `dungeon(M, H, V)`, ki velja takrat, ko:

- je matrika `M` (seznam seznamov) binarna,
- je seznam `H` vsota vseh vrstic matrike `M`,
- je seznam `V` vsota vseh stolpcev matrike `M`,
- so vse vrstice enako dolge,
- so vsi stolpci enako dolgi.

*Lahko si pomagata s pomožnimi predikati.*

Primeri.

```
?- dungeon(M, [1,2,3], [3, 2, 1]).
M = [[1, 0, 0], [1, 1, 0], [1, 1, 1]].

?- dungeon(M, [1,2,4,3], [4,3,2,1]).
M = [[1, 0, 0, 0], [1, 1, 0, 0], [1, 1, 1, 1], [1, 1, 1, 0]].

?- dungeon(M, [1,1,3,3], [3,2,2,1]), nice_corners(M), maplist(portray_clause, M).
[0, 0, 0, 1].
[1, 0, 0, 0].
[1, 1, 1, 0].
[1, 1, 1, 0].
M = [[0, 0, 0, 1], [1, 0, 0, 0], [1, 1, 1, 0], [1, 1, 1, 0]] ;
[1, 0, 0, 0].
[1, 0, 0, 0].
[1, 1, 1, 0].
[0, 1, 1, 1].
M = [[1, 0, 0, 0], [1, 0, 0, 0], [1, 1, 1, 0], [0, 1, 1, 1]] ;
false.

?- M =
[[0, _, 0, _, _, _, _, _],
[_ , _ , _ , _ , _ , _ , _ , _],
[_ , _ , _ , _ , _ , 0, _ , _],
[_ , _ , _ , _ , _ , _ , _ , _],
[_ , _ , _ , _ , _ , _ , _ , _],
[_ , _ , _ , _ , _ , _ , _ , 0],
[_ , _ , _ , _ , _ , _ , _ , _],
[_ , _ , _ , _ , _ , _ , _ , 0]],
dungeon(M, [6,3,1,3,6,1,6,0], [0,6,1,5,3,2,3,6]), maplist(portray_clause, M).
[0, 1, 0, 1, 1, 1, 1, 1].
[0, 1, 0, 1, 0, 0, 0, 1].
[0, 0, 0, 0, 0, 0, 0, 1].
[0, 1, 0, 1, 0, 0, 0, 1].
[0, 1, 0, 1, 1, 1, 1, 1].
[0, 1, 0, 0, 0, 0, 0, 0].
[0, 1, 1, 1, 1, 0, 1, 1].
[0, 0, 0, 0, 0, 0, 0, 0].
M = [[0, 1, 0, 1, 1, 1, 1, 1]|...] ;
[0, 1, 0, 1, 1, 1, 1, 1].
[0, 1, 0, 1, 0, 0, 0, 1].
[0, 0, 0, 0, 0, 0, 0, 1].
[0, 1, 0, 1, 0, 0, 0, 1].
[0, 1, 1, 1, 1, 0, 1, 1].
[0, 1, 0, 0, 0, 0, 0, 0].
[0, 1, 0, 1, 1, 1, 1, 1].
[0, 0, 0, 0, 0, 0, 0, 0].
M = [[0, 1, 0, 1, 1, 1, 1, 1]|...].
```