Programiranje 2:
Koncepti programskih jezikov

2. domača naloga

April 2021

Naloge se rešuje samostojno, v primeru prepisovanja se upošteva univerzitetni pravilnik. Oddajte eno TXT datoteko z OCaml kodo na `http://e.famnit.upr.si`. Pred oddajo nujno testirajte svojo kodo, ki se mora pravilno prevesti[1]. Če oddate kodo, ki se ne prevede, bo vaš izdelek točkovan z 0 točkami. Rešitev naj vsebuje vaše ime, priimek, smer in vpisno številko – ti podatki naj bodo vsebovani kot komentar. Poskusite rešiti vsako oštevilčeno nalogo kot eno samo funkcijo.

Srečno!

1 Ugasni luč/Lights Out (30%)

V tej nalogi bomo razvili funkcije, ki se lahko uporabijo za implementacijo igre *lights out*. Osnovna pravila te igre najdemo na

`https://en.wikipedia.org/wiki/Lights_Out_(game)`.

1. Napiši funkcijo flip

   ```
   val flip : bool array array -> int -> int -> bool array array = <fun>
   ```

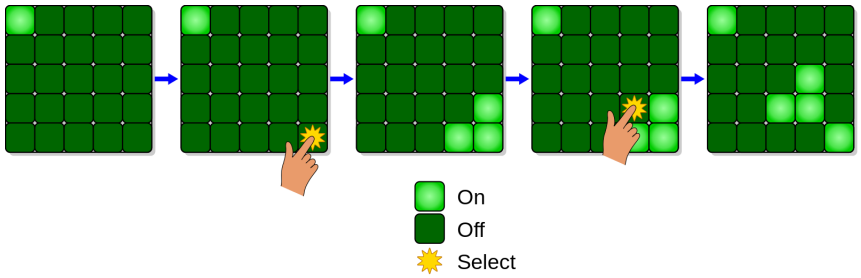
   katere vhod so bool matrika in dve celi števili i, j. Funkcija negira vrednost (true→false, false→true) na lokaciji i, j v matriki, poleg tega pa tudi negira vse (do) 4 horizontalne/vertikalne sosedne elemente (glej sliko).

2. Napiši funkcijo print_matrix

   ```
   val print_matrix : bool array array -> unit = <fun>
   ```

   ki vhodno bool matriko sprinta na ekran (true →"T", false→"F").

[1] Uspešno prevajanje lahko preverite s pomočjo orodja dosegljivega na `https://try.ocamlpro.com/`.




```
Sample input:
# let matrix=  [|[|false; true; true; false|]; [|true; false; false; true|];
[|false; false; true; true|]|];;
# print_matrix matrix;;
FTTF
TFFT
FFTT
# flip matrix 1 4;;
# print_matrix matrix;;
FTFT
TFFF
FFTT
```

2 Drsne ploščice (35%)

V tej nalogi bomo razvili funkcije, ki se lahko uporabijo za implementacijo igre *sliding puzzle*. Osnovna pravila te igre najdemo na

`https://en.wikipedia.org/wiki/Sliding_puzzle`.

1. Napiši tip slidingTile, ki vsebuje char matriko in dve celi števili x in y. Števili x in y sta koordinati manjkajoče ploščice.

2. Napiši funkcijo

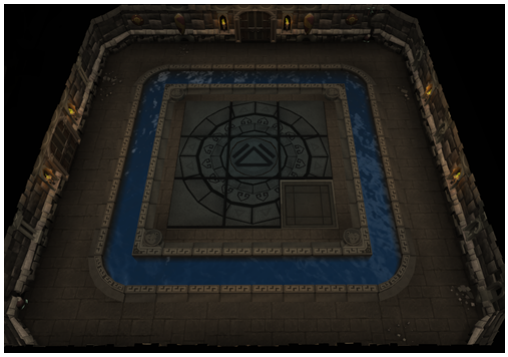
   ```
   val slide : slidingTile -> int -> int -> slidingTile = <fun>
   ```

   katere vhod je slidingTile in dve celi števili, ki predstavljajo lokacijo v matriki. Funkcija premakne element (ploščico) is dane lokacije in jo premakne tam kjer je manjkajoča ploščica in vrne spremenjen slidingTile. Če pa dana lokacija ni sosedna z manjkajočo ploščico, ali pa je izven območja matrike, potem pa vrne slidingTile nespremenjen.

3. Napiši funkcijo

   ```
   val print_tile : slidingTile -> unit = <fun>
   ```

ki sprinta matriko vhodnega slidingTile na ekran. Tam kjer je manjkajoča ploščica, naj se izpiše presledek namesto znaka iz matrike.




```
# let a=...
# print_tile a;;
123
456
78
# slide a 2 2;;
# print_tile a;;
123
456
78
# slide a 3 2;;
# print_tile a;;
123
456
7 8
# slide a 4 2;;
# print_tile a;;
123
456
7 8
# slide a 2 2;;
# print_tile a;;
123
4 6
758
```

3 Briškula (35%)

Briškula je lokalna igra s kartami katere barve so Špada, Kope, Baštoni in Denari (Swords, Cups, Clubs, Coins). Poleg barve ima vsaka karto ali figuro ali

številko. Figure so Kralj, Kaval in Fant (King, Rider, Jack) in številke so med 1 in 7.

1. Naredi tip *suit* za barve kart.

2. Naredi tip *briscola*, ki je lahko katera koli opisana karta (ali glej sliko).

3. Napiši funkcijo points

   ```
   val points : briscola -> int = <fun>
   ```

   ki za vhodno karto vrne koliko točk je vredna (1 → 11; 3→ 10; King → 4; Rider → 3; Jack → 2; 7,6,5,4,2 → 0) in ignorira barvo.

4. Napiši funkcijo color

   ```
   val color : briscola -> suit = <fun>
   ```

   ki za vhodno karto vrne njeno barvo.

5. Napiši funkcijo higherfigure

   ```
   val higherfigure : briscola -> briscola -> bool = <fun>
   ```

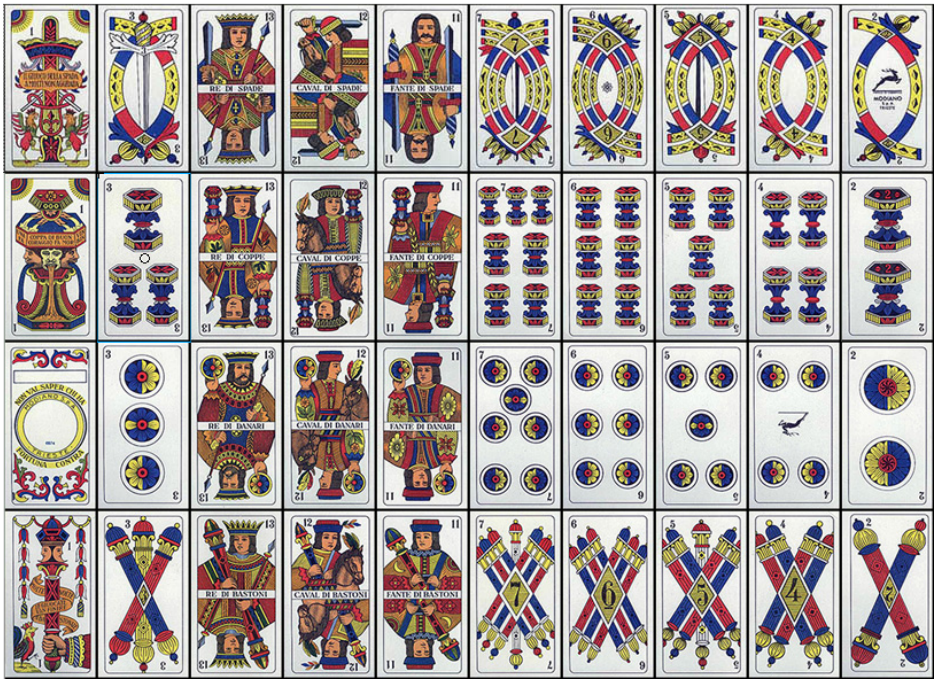
   ki za dve vhodni karti, vrne true če je prva karta večja ali enaka drugi glede samo na figuro/številko in vrne false drugače (1 > 3 > King > Rider > Jack > 7 > 6> 5 > 4> 2).

6. Napiši funkcijo highercard

   ```
   val highercard : suit -> briscola -> briscola -> bool = <fun>
   ```

   ki ima tri vhodne parametre: trumpsuit, card1, card2 in vrne true če je prva karta (card1) močnejša od druge karte (card2) in vrne false drugače. Če pa sta obe karti enake barve, potem zmaga tista z močnejšo figuro. Če sta drugačne barve potem zmaga prva karta razen, če je barva druga karta adut. [2].

[2]Opomba: ta funkcija ni simetrična za fiksen adut. (i.e. mišljeno je da je prva karta igrana prva in je njena barva močnejša od drugi barv ki niso adut. Na primer, dve karti, ki sta v dveh barvah drugačnih od aduta vrneta true v tudi v drugem vhodnem vrstnem redu)




```
# points (Number (1,Club));;
- : int = 11
# points (Jack Sword);;
- : int = 2
# color (Jack Sword);;
- : suit = Sword
# higherfigure (King Coin) (Rider Cup);;
- : bool = true
# highercard Cup (King Coin) (Rider Cup);;
- : bool = false
# highercard Sword (Number (7,Club)) (Number (3,Club));;
- : bool = false
# highercard Club (Number (1,Club)) (Number (6,Club)) ;;
- : bool = true
```

Bonus naloga (dodatnih 10%)

Napiši funkcijo play

```
val play : suit -> briscola -> briscola -> briscola ->
briscola -> string * int =<fun>
```

z 5. vhodnimi parametri (trumpsuit, player1card, player2card, player3card, player4card), ki demonstrira eno rundo Briškule. Igre se igra s štirimi igralci,

od sta 1. in 3. igralec skupaj (Team1), ter 2. in 4. igrata skupaj (Team2). Če noben ni igral aduta potem zmaga najvišja karta v barvi karte prvega igralca. V nasprotnem primeru zmaga najvišji adut. Funkcija vrne katera skupina zmaga in koliko točk so pobrali. Število točk je enako vsoti točk igranih kart. Lahko prepostaviš, da so vse igrane karte bile različne. Več o pravilih na:

`https://en.wikipedia.org/wiki/Briscola`.

Ta izziv ne prinese nobenih dodatnih točk.

```
# play Cup (Number (4,Club)) (Rider Club) (Number (7,Coin)) (Number (1, Cup));;
# (*4th card is best card because trump*)
- : string * int = ("Team2", 14)
# play Sword (Number (4,Club)) (Number (3,Club)) (Number (7,Coin)) (Number (1, Cup));;
# (*2nd card is best, because it is highest in 1st card's suit and there are no trumps *)
- : string * int = ("Team2", 21)
# play Sword (Number (4,Club))  (Number (7,Coin)) (Rider Club) (Number (1, Cup));;
# (*3rd card is best*)
- : string * int = ("Team1", 14)
# play Club (Number (2,Coin)) (Number (4,Club))  (Jack Club) (Number (6, Cup));;
# (*3rd card is best, because it is highest among trumps*)
- : string * int = ("Team1", 2)
```