Dodatne naloge pri predmetu Programiranje I

Predgovor

Pred vami je komplet 60 nalog za utrjevanje snovi pri predmetu Programiranje I. Dodatnih nalog — za razliko od domačih — ne bomo pregledovali in ocenjevali, saj so namenjene izključno utrjevanju snovi. Programiranje je namreč obrt, ki se je lahko priučite samo z rednim in sprotnim pisanjem programov. Zato vam priporočamo, da se takoj po predavanjih lotite nalog na temo odpredavane snovi. Naloge so urejene po sklopih, znotraj posameznih sklopov pa (v grobem) po težavnosti. Naloge, označene z zvezdico, so nekoliko težje, a se jih nikar ne ustrašite. Vse je možno rešiti s programskimi konstrukti in tehnikami, ki jih bomo spoznali pri predmetu Programiranje I.

Poleg besedila nalog boste na spletni učilnici našli tudi paket, ki vsebuje testne primere in izhodiščne datoteke za posamezne naloge. Vsaka dodatna naloga je opremljena z desetimi testnimi primeri, s katerimi lahko vaše rešitve preizkusite. Če smo natančnejši:

- Pri nalogah tipa vhod-izhod imate na voljo 10 parov vhodnih datotek `vhod*.txt` in pripadajočih izhodnih datotek `izhod*.txt`. Testni primer i se šteje kot pravilno obravnavan natanko tedaj, ko vaš program pri branju vhoda iz datoteke `vhod`i`.txt` proizvede izpis, ki je enak vsebini datoteke `izhod`i`.txt`.
- Pri nalogah, ki se preverjajo s testnimi razredi, smo pripravili 10 parov testnih razredov (datotek `Test*.java`) in pripadajočih izhodnih datotek `izhod*.txt`. Vsak testni razred praviloma ustvari vsaj en objekt razreda, ki ga želite preizkusiti, nato pa na objektu kliče vsaj eno metodo in izpiše njen rezultat. Testni primer i se šteje kot pravilno obravnavan natanko tedaj, ko po zagonu razreda `Test`i`.java` dobimo izpis, ki je enak vsebini datoteke `izhod`i`.txt`.
- Pri nalogah na temo risanja v grafičnem oknu je na voljo 10 parov testnih razredov (datotek `Test*.java`) in pripadajočih izhodnih slik (datotek `izhod*.png`). Vsak testni razred na podlagi vašega razreda ustvari slikovno datoteko. Testni primer i se šteje kot pravilno obravnavan natanko tedaj, ko po zagonu razreda `Test`i`.java` dobimo sliko, ki je enaka sliki v datoteki `izhod`i`.png`.

Pri nalogah, ki zahtevajo dopolnitev delno že napisanega razreda, boste v paketu na spletni učilnici našli tudi ustrezne izhodiščne razrede.

Veliko užitkov pri reševanju nalog!

Kazalo

1 Osnovni konstrukti I

Splošna navodila

Pri vseh nalogah v tem sklopu lahko svoje rešitve preverite z množico vhodnih datotek in pripadajočih izhodnih datotek.

1.1 Urejanje trojice

Naloga

Napišite program, ki prebere tri cela števila in jih izpiše v naraščajočem vrstnem redu.

Vhod

Na vhodu so podana tri cela števila z intervala $[-10^9, 10^9]$. Med seboj so ločena s po enim presledkom.

Izhod

Na izhod izpišite vhodna števila v naraščajočem vrstnem redu. Med seboj jih ločite s po enim presledkom.

Primer 1

Testni vhod:

```
5 7 2
```

Pričakovani izhod:

```
2 5 7
```

Primer 2

Testni vhod:

```
-6 -9 -6
```

Pričakovani izhod:

```
-9 -6 -6
```

1.2 Zaporedja

Naloga

Napišite program, ki prebere cela števila a, b in k, nato pa izpiše zaporedje števil od a do b s korakom k. V primeru $a \leqslant b$ naj se izpis zaključi pri največjem številu, ki ni večje od b, v primeru $a > b$ pa pri najmanjšem številu, ki ni manjše od b.

Pred izpisom zaporedja naj program preveri, ali vhod zadošča sledečima pogojema:

- korak k ni enak 0;
- korak je pozitiven v primeru $a < b$ oziroma negativen v primeru $a > b$.

Če vhod ne izpolnjuje katerega od pogojev, naj program izpiše zgolj besedo `NAPAKA`.

Vhod

Na vhodu so po vrsti podana cela števila a, b in k. Vsa tri števila so z intervala $[-10^9, 10^9]$. Med seboj so ločena s po enim presledkom.

Izhod

V primeru nepravilnega vhoda izpišite besedo `NAPAKA`, v primeru pravilnega vhoda pa izpišite ustrezno zaporedje. Vsako število izpišite v svojo vrstico.

Primer 1

Testni vhod:

```
10 30 0
```

Pričakovani izhod:

```
NAPAKA
```

Primer 2

Testni vhod:

```
10 30 -2
```

Pričakovani izhod:

```
NAPAKA
```

Primer 3

Testni vhod:

```
10 30 3
```

Pričakovani izhod:

```
10
13
16
19
22
25
28
```

Primer 4

Testni vhod:

```
30 -20 -5
```

Pričakovani izhod:

```
30
25
20
15
10
5
0
-5
-10
-15
-20
```

1.3 Poštevanka I

Naloga

Napišite program, ki prebere celo število a in pozitivno celo število b ter izpiše poštevanko števila a s faktorji od 1 do vključno b.

Vhod

Na vhodu sta po vrsti zapisani celo število $a \in [-10^6, 10^6]$ in celo število $b \in [1, 10^3]$. Števili sta ločeni s presledkom.

Izhod

Vsako enačbo poštevanke izpišite v svoji vrstici. Posamezno enačbo izpišite v sledeči obliki (znak ␣ predstavlja presledek):

`p␣*␣q␣=␣r`

Primer 1

Testni vhod:

```
5 6
```

Pričakovani izhod:

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
```

Primer 2

Testni vhod:

```
-20 3
```

Pričakovani izhod:

```
-20 * 1 = -20
-20 * 2 = -40
-20 * 3 = -60
```

1.4 Poštevanka II

Naloga

Napišite program, ki prebere pozitivni celi števili a in b ter izpisuje poštevanko števila a tako dolgo, dokler rezultat ni večji od b.

Vhod

Na vhodu sta po vrsti zapisani celo število $a \in [1,\ 10^9]$ in celo število $b \in [1,\ 10^9]$. Števili sta ločeni s presledkom.

Izhod

Vsako enačbo poštevanke izpišite v svoji vrstici. Posamezno enačbo izpišite v sledeči obliki (znak ␣ predstavlja presledek):

`p␣*␣q␣=␣r`

V primeru $b < a$ naj program ne izpiše ničesar.

Primer 1

Testni vhod:

```
6 25
```

Pričakovani izhod:

```
6 * 1 = 6
6 * 2 = 12
6 * 3 = 18
6 * 4 = 24
```

Primer 2

Testni vhod:

```
5 25
```

Pričakovani izhod:

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
```

Primer 3

Testni vhod:

```
10 9
```

Pričakovani izhod:

(V tem primeru program ne sme izpisati ničesar!)

1.5 Potenca

Naloga

Napišite program, ki prebere pozitivno celo število a in ne-negativno celo število b ter izpiše vrednost potence a^b. Nalogo rešite s pomočjo zaporednih množenj, ne z metodo `Math.pow`.

Vhod

Na vhodu sta po vrsti zapisani celo število $a \in [1,\ 10^9]$ in celo število $b \in [0,\ 10^9]$, tako da velja $a^b \leqslant 10^9$. Števili sta ločeni s presledkom.

Izhod

Na izhod izpišite vrednost potence a^b.

Primer 1

Testni vhod:

```
3 4
```

Pričakovani izhod:

```
81
```

Primer 2

Testni vhod:

```
6 0
```

Pričakovani izhod:

```
1
```

2 Osnovni konstrukti II

Splošna navodila

Pri vseh nalogah v tem sklopu lahko svoje rešitve preverite z množico vhodnih datotek in pripadajočih izhodnih datotek.

2.1 Delitelji

Naloga

Napišite program, ki prebere pozitivno celo število, nato pa izpiše vse njegove delitelje.

Vhod

Na vhodu je podano pozitivno celo število, manjše od 10^6.

Izhod

Na izhod v naraščajočem vrstnem redu izpišite vse delitelje prebranega števila. Vsak delitelj izpišite v svoji vrstici.

Primer 1

Testni vhod:

```
30
```

Pričakovani izhod:

```
1
2
3
5
6
10
15
30
```

Primer 2

Testni vhod:

```
31
```

Pričakovani izhod:

```
1
31
```

Primer 3

Testni vhod:

```
1
```

Pričakovani izhod:

```
1
```

2.2 EvroŠop®

Naloga

V trgovini EvroŠop® so vsi izdelki naprodaj za 1 evro. Denimo, da vsaka stranka kupi samo po en izdelek, plača pa ga bodisi s kovancem za 1 evro ali pa s kovancem za 2 evra. V prvem primeru blagajničarka stranki seveda ne vrne ničesar (saj izdelek stane 1 evro), v drugem primeru pa ji vrne kovanec za 1 evro. Denimo, da je blagajna na začetku prazna.

Napišite program, ki prebere zaporedje podatkov o tem, s katerim kovancem je posamezna stranka plačala izdelek, nato pa izpiše končno število kovancev v blagajni. Lahko se zgodi, da blagajničarka stranki, ki je izdelek plačala s kovancem za 2 evra, ne more vrniti kovanca za 1 evro, ker jih v blagajni preprosto ni. V tem primeru naj se program takoj zaključi z ustreznim sporočilom.

Vhod

Na vhodu je zapisano zaporedje števil 1 in 2, ločenih s po enim presledkom. Dolžina zaporedja ni znana vnaprej.

Izhod

Če blagajničarka neki stranki ne more vrniti denarja, izpišite samo besedo `BANKROT`. V nasprotnem primeru pa v prvi vrstici izpišite končno število kovancev za 1 evro v blagajni, v drugi pa končno število kovancev za 2 evra v blagajni.

Primer 1

Testni vhod:

```
1 1 1 1 1 2 1 1 1
```

Pričakovani izhod:

```
7
1
```

V tem primeru najprej prejmemo 5 kovancev za 1 evro, nato pa enega vrnemo stranki, ki nam je dala kovanec za 2 evra. Nato prejmemo še 3 kovance za 1 evro.

Primer 2

Testni vhod:

```
1 2 2 1 1 2 2 2
```

Pričakovani izhod:

```
BANKROT
```

V tem primeru že pri tretji stranki bankrotiramo.

2.3 Smučanje

Naloga

Na smučarskem tekmovanju nastopa n tekmovalcev. Vsak tekmovalec odsmuča progo dvakrat, njegov rezultat pa je seštevek obeh časov. Če tekmovalca diskvalificirajo, se njegov rezultat ne upošteva. Če ga diskvalificirajo že v prvem teku, potem v drugem teku sploh ne bo nastopal.

Napišite program, ki najprej prebere število n, nato pa za vsakega tekmovalca prebere njegov rezultat v prvem in drugem teku. Rezultat je podan bodisi kot pozitivno celo število, ki podaja čas vožnje, ali pa kot število 0, ki pomeni diskvalifikacijo. Če so vse tekmovalce diskvalificirali, naj program to sporoči, sicer pa naj izpiše zaporedno številko tekmovalca z najboljšim skupnim časom in njegov skupni čas. Če je najboljših tekmovalcev več, naj program izbere tistega z najmanjšo zaporedno številko.

Vhod

V prvi vrstici vhoda je podano celo število $n \in [1, 10^6]$, nato pa sledi še n vrstic vhoda. V vsaki od teh n vrstic je podano bodisi samo število 0 ali pa dvoje celih števil, ločenih s presledkom. Prvo od teh dveh števil je z intervala $[1, 10^9]$, drugo pa z intervala $[0, 10^9]$.

Izhod

Če so vse tekmovalce diskvalificirali, izpišite samo niz `NIHCE`, v nasprotnem primeru pa v prvi vrstici izpišite številko tekmovalca z najmanjšim skupnim časom, v drugem pa njegov skupni čas.

Primer 1

Testni vhod:

```
5
70 65
40 0
55 59
0
50 72
```

Pričakovani izhod:

```
3
114
```

Tekmovalca 2 in 4 so diskvalificirali, zato ju ne upoštevamo, od preostalih pa je najboljši tekmovalec 3.

Primer 2

Testni vhod:

```
3
0
70 0
0
```

Pričakovani izhod:

```
NIHCE
```

2.4 Vozni red

Naloga

Avtobus vozi v enakomernih časovnih presledkih. Napišite program, ki prebere čas (uro in minuto, h_z in m_z) začetka dnevne vožnje, čas (uro in minuto, h_k in m_k) konca dnevne vožnje in interval v minutah (d), nato pa izpiše dnevni vozni red. Prva vožnja se izvrši natanko ob času začetka vožnje, zadnja pa ob času, ki je kvečjemu enak času konca vožnje.

Vhod

Na vhodu je zapisanih pet celih števil, med seboj ločenih s po enim presledkom: h_z, m_z, h_k, m_k in d. Velja $0 \leqslant h_z \leqslant 23$, $0 \leqslant h_k \leqslant 23$, $0 \leqslant m_z \leqslant 59$, $0 \leqslant m_k \leqslant 59$, $1 \leqslant d \leqslant 1440$ ter bodisi $h_z < h_k$ ali pa $h_z = h_k$ in $m_z < m_k$.

Izhod

Na izhod v naraščajočem vrstnem redu izpišite vse odhode avtobusa v obliki `HH:MM`. Vsak čas naj bo zapisan v svoji vrstici.

Primer 1

Testni vhod:

```
10 0 15 0 30
```

Pričakovani izhod:

```
10:00
10:30
11:00
11:30
12:00
12:30
13:00
13:30
14:00
14:30
15:00
```

Primer 2

Testni vhod:

```
8 50 14 10 35
```

Pričakovani izhod:

```
08:50
09:25
10:00
10:35
11:10
11:45
12:20
12:55
13:30
14:05
```

Primer 3

Testni vhod:

```
5 55 23 59 142
```

Pričakovani izhod:

```
05:55
08:17
10:39
13:01
15:23
17:45
20:07
22:29
```

2.5 Sprotna statistika

Naloga

Napišite program, ki bere cela števila in sproti izpisuje največje in najmanjše dotlej vnešeno število ter povprečje in standardni odklon vseh dotlej vnešenih števil. Standardni odklon števil $a_1, \ldots, a_n$ izračunamo po sledeči formuli:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} a_i^2}{n} - \left(\frac{\sum_{i=1}^{n} a_i}{n}\right)^2}$$

Vhod

Na vhodu je podano zaporedje celih števil z intervala $[-10^6, 10^6]$. Vsako število je zapisano v svoji vrstici. Dolžina zaporedja ni znana vnaprej.

Izhod

Izhod naj bo sestavljen iz enakega števila vrstic kot vhod. V i-ti vrstici izhoda po vrsti izpišite maksimum, minimum, povprečje in standardni odklon vseh prebranih števil od prvega do vključno i-tega. Podatke, izpisane znotraj posamezne vrstice, med seboj ločite s po enim presledkom. Povprečje in standardni odklon izpišite na 2 decimalki natančno.

Primer

Testni vhod:

```
30
-16
10
-450
450
-24
700
-140
-90
-61
```

Pričakovani izhod:

```
30 30 30.00 0.00
30 -16 7.00 23.00
30 -16 8.00 18.83
30 -450 -106.50 198.99
450 -450 4.80 285.01
450 -450 0.00 260.39
700 -450 100.00 343.68
700 -450 70.00 331.14
700 -450 52.22 316.23
700 -450 40.90 301.91
```

3

Osnovni konstrukti III

Splošna navodila

Pri vseh nalogah v tem sklopu lahko svoje rešitve preverite z množico vhodnih datotek in pripadajočih izhodnih datotek.

3.1 Predvolilni golaž

Naloga

Politiku Gvidu[1] je podpora pred volitvami nevarno padla, zato se odloči, da bo izbranim skupinam volilcev plačeval kosila v dobrih gostilnah tako dolgo, dokler mu ne zmanjka denarja. Vsak dan povabi določeno skupino ljudi v izbrano gostilno. Cena pogostitve se v osnovi izračuna kot zmnožek števila kosil in cene kosila, izbrano vino pa ceno pogostitve poveča za navzgor zaokroženo polovico (če cena brez vina znaša 45 evrov, je cena z vinom enaka $45 + 23 = 68$ evrov, če pa bi brez vina odšteli 46 evrov, bi z vinom $46 + 23 = 69$ evrov). Napišite program, ki najprej prebere podatek o začetni zalogi Gvidovega denarja, nato pa zaporedoma bere podatke o pogostitvah ter sproti izpisuje njihove cene in preostalo zalogo Gvidovega denarja. Program naj se zaključi, ko zmanjka vhoda ali pa Gvidovega denarja.

Vhod

V prvi vrstici vhoda je zapisana začetna količina Gvidovega denarja (celo število z intervala $[0, 10^9]$), v vseh ostalih vrsticah pa so zapisani podatki o posameznih pogostitvah. Število pogostitev ni znano vnaprej. Vsaka pogostitev je opredeljena s tremi števili, ki so med seboj ločena s po enim presledkom:

- cena enega kosila (celo število z intervala $[0, 10^3]$);

[1] Po istoimenski skladbi Iztoka Mlakarja

- število kosil (celo število z intervala $[1, 10^3]$);
- podatek o tem, ali so gostje pili tudi izbrano vino (1: da, 0: ne).

Izhod

V i-ti vrstici izhoda izpišite dve števili, ločeni s presledkom: ceno i-te pogostitve in zalogo Gvidovega denarja po i-ti pogostitvi. Če Gvidu zmanjka denarja, namesto drugega števila izpišite znak - (minus).

Primer 1

V tem testnem primeru Gvido ostane brez denarja še pred koncem vhoda.

Testni vhod:

```
500
30 5 0
30 6 1
15 5 1
20 7 0
50 4 1
```

Pričakovani izhod:

```
150 350
270 80
113 -
```

Primer 2

V tem testnem primeru Gvido »vzdrži« do konca vhoda.

Testni vhod:

```
1000
30 5 0
30 6 1
15 5 1
20 7 0
50 4 1
```

Pričakovani izhod:

```
150 850
270 580
113 467
140 327
300 27
```

3.2 Piramida števil

Naloga

Napišite program, ki prebere pozitivno celo število n, nato pa nariše »piramido« števil višine n, kot jo prikazujeta primera v nadaljevanju.

Vhod

Na vhodu je podano zgolj celo število $n \in [1, 100]$.

Izhod

Na izhod izpišite »piramido« po zgledu sledečih primerov. Ne izpisujte odvečnih presledkov in praznih vrstic.

Primer 1

Testni vhod:

```
5
```

Pričakovani izhod:

```
    1
   234
  34567
 4567890
567890123
```

Pričakovani izhod s prikazanimi presledki:

```
␣␣␣␣1
␣␣␣234
␣␣34567
␣4567890
567890123
```

Primer 2

Testni vhod:

```
11
```

Pričakovani izhod:

```
          1
         234
        34567
       4567890
      567890123
```

```
     67890123456
    7890123456789
   890123456789012
  90123456789012345
 0123456789012345678
12345678901234567901
```

3.3 Igorjevi bloki

Naloga

Napišite program, ki prebere tri enomestna neničelna cela števila in nato nariše vzorec, kot ga prikazujeta primera v nadaljevanju.

Vhod

Na vhodu so podana tri cela števila z intervala $[1, 9]$. Med seboj so ločena s po enim presledkom.

Izhod

Na izhod izpišite vzorec po zgledu sledečih primerov. Ne izpisujte odvečnih presledkov in praznih vrstic.

Primer 1

Testni vhod:

```
3 7 4
```

Pričakovani izhod:

```
333 7777777 4444
333 7777777 4444
333 7777777 4444
    7777777 4444
    7777777
    7777777
    7777777
```

Pričakovani izhod s prikazanimi presledki:

```
333␣7777777␣4444
333␣7777777␣4444
333␣7777777␣4444
␣␣␣␣7777777␣4444
␣␣␣␣7777777
␣␣␣␣7777777
␣␣␣␣7777777
```

Primer 2

Testni vhod:

```
2 2 3
```

Pričakovani izhod:

```
22 22 333
22 22 333
      333
```

3.4 Šahovnica

Naloga

Napišite program, ki najprej prebere pozitivna cela števila v, s in d, nato pa nariše vzorec v obliki šahovnice z v vrsticami in s stolpci, pri čemer ima vsako polje obliko kvadrata velikosti $d \times d$. Šahovnica naj bo tudi obrobljena. Zgledujte se po primerih v nadaljevanju.

Vhod

Na vhodu so podana cela števila v, s in d z intervala $[1, 20]$. Med seboj so ločena s po enim presledkom.

Izhod

Na izhod izpišite vzorec po zgledu sledečih primerov. Ne izpisujte odvečnih presledkov in praznih vrstic.

Primer 1

Testni vhod:

```
3 4 5
```

Pričakovani izhod:

```
+ - - - - - - - - - - - - - - - - - - - - - +
|           * * * * *           * * * * * |
|           * * * * *           * * * * * |
|           * * * * *           * * * * * |
|           * * * * *           * * * * * |
|           * * * * *           * * * * * |
| * * * * *           * * * * *           |
| * * * * *           * * * * *           |
| * * * * *           * * * * *           |
| * * * * *           * * * * *           |
| * * * * *           * * * * *           |
|           * * * * *           * * * * * |
|           * * * * *           * * * * * |
|           * * * * *           * * * * * |
|           * * * * *           * * * * * |
|           * * * * *           * * * * * |
+ - - - - - - - - - - - - - - - - - - - - - +
```

Primer 2

Testni vhod:

```
6 5 2
```

Pričakovani izhod:

```
+ - - - - - - - - - - +
|     * *     * *     |
|     * *     * *     |
| * *     * *     * * |
| * *     * *     * * |
|     * *     * *     |
|     * *     * *     |
| * *     * *     * * |
| * *     * *     * * |
|     * *     * *     |
|     * *     * *     |
| * *     * *     * * |
| * *     * *     * * |
+ - - - - - - - - - - +
```

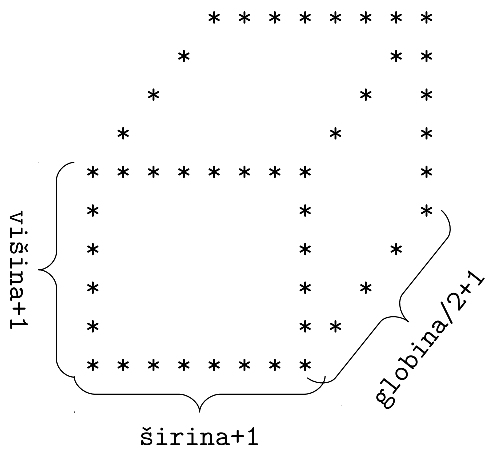
Pričakovani izhod s prikazanimi presledki:

```
+␣-␣-␣-␣-␣-␣-␣-␣-␣-␣-␣+
|␣␣␣␣␣*␣*␣␣␣␣␣*␣*␣␣␣␣␣|
|␣␣␣␣␣*␣*␣␣␣␣␣*␣*␣␣␣␣␣|
|␣*␣*␣␣␣␣␣*␣*␣␣␣␣␣*␣*␣|
|␣*␣*␣␣␣␣␣*␣*␣␣␣␣␣*␣*␣|
|␣␣␣␣␣*␣*␣␣␣␣␣*␣*␣␣␣␣␣|
|␣␣␣␣␣*␣*␣␣␣␣␣*␣*␣␣␣␣␣|
|␣*␣*␣␣␣␣␣*␣*␣␣␣␣␣*␣*␣|
|␣*␣*␣␣␣␣␣*␣*␣␣␣␣␣*␣*␣|
|␣␣␣␣␣*␣*␣␣␣␣␣*␣*␣␣␣␣␣|
|␣␣␣␣␣*␣*␣␣␣␣␣*␣*␣␣␣␣␣|
|␣*␣*␣␣␣␣␣*␣*␣␣␣␣␣*␣*␣|
|␣*␣*␣␣␣␣␣*␣*␣␣␣␣␣*␣*␣|
+␣-␣-␣-␣-␣-␣-␣-␣-␣-␣-␣+
```

3.5 Kvader

Naloga

Napišite program, ki prebere pozitivna cela števila s, v in g, nato pa na tekstovni zaslon nariše trirazsežno iluzijo kvadra širine s, višine v in globine g. Sliko narišite po vzoru sledečega primera, ki prikazuje kvader s širino 7, višino 5 in globino 8:



Vhod

Na vhodu so po vrsti podana cela števila $s \in [1, 100]$, $v \in [1, 100]$ in $g \in [2, 100]$. Med seboj so ločena s po enim presledkom.

Izhod

Na izhod izpišite vzorec po zgledu sledečih primerov. Ne izpisujte odvečnih presledkov in praznih vrstic.

Primer 1

Testni vhod:

```
3 7 4
```

Pričakovani izhod:

```
    * * * *
  *     * *
* * * *   *
*     *   *
*     *   *
*     *   *
*     *   *
*     *   *
*     * *
* * * *
```

Pričakovani izhod s prikazanimi presledki:

```
␣␣␣␣*␣*␣*␣*
␣␣*␣␣␣␣␣*␣*
*␣*␣*␣*␣␣␣*
*␣␣␣␣␣*␣␣␣*
*␣␣␣␣␣*␣␣␣*
*␣␣␣␣␣*␣␣␣*
*␣␣␣␣␣*␣␣␣*
*␣␣␣␣␣*␣␣␣*
*␣␣␣␣␣*␣*
*␣*␣*␣*
```

Primer 2

Testni vhod:

```
5 3 15
```

Pričakovani izhod:

```
              * * * * * *
            *           * *
          *           *   *
        *           *     *
      *           *     *
    *           *     *
  *           *     *
* * * * * *     *
*           *   *
*           * *
* * * * * *
```

Primer 3

Testni vhod:

```
5 5 5
```

Pričakovani izhod:

```
    * * * * * *
  *           * *
* * * * * *   *
*           *   *
*           *   *
*           *   *
*           * *
* * * * * *
```

3.6 Anžetove ledene sveče

Naloga

Napišite program, ki prebere pozitivno celo število n in nariše vzorec višine n, kot ga prikazujejo primeri v nadaljevanju.

Vhod

Na vhodu je podano celo število $n \in [2, 20]$.

Izhod

Na izhod izpišite vzorec po zgledu sledečih primerov. Ne izpisujte odvečnih presledkov in praznih vrstic.

Primer 1

Testni vhod:

```
5
```

Pričakovani izhod:

```
*****************
* * * * * * * * *
*   *   *   *   *
*       *       *
*               *
```

Primer 2

Testni vhod:

```
6
```

Pričakovani izhod:

```
*********************************
* * * * * * * * * * * * * * * * *
*   *   *   *   *   *   *   *   *
*       *       *       *       *
*               *               *
*                               *
```

Primer 3

Testni vhod:

```
7
```

Pričakovani izhod:

```
***************************************************************
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *   *
*       *       *       *       *       *       *       *       *
*               *               *               *               *
*                               *                               *
*                                                               *
```

3.7 Metaprogram

Naloga

S sledečo zanko po vrsti izpišemo vse velike črke angleške abecede od A do Z:

```
for (char c1 = 'A';  c1 <= 'Z';  c1++) {
    System.out.println("" + c1);
}
```

Sedaj pa bi želeli po abecednem vrstnem redu izpisati vse nize (besede), sestavljene iz n velikih črk angleške abecede. Na primer, če je $n = 2$, bi želeli izpisati nize `AA`, `AB`, ..., `AZ`, `BA`, `BB`, ..., `BZ`, ..., `ZA`, `ZB`, ..., `ZZ`. Če je $n = 3$, bi želeli izpisati nize `AAA`, `AAB`, ..., `ZZZ`. Da bi lahko nalogo rešili za poljuben n, bi potrebovali nekoliko več znanja, kot ga imamo sedaj. Za fiksen n pa lahko nalogo rešimo s pomočjo n vgnezdenih zank. Vaša naloga je napisati *metaprogram* za rešitev opisanega problema, tj. program, ki izpiše program, ki rešuje problem.

Napišite program, ki prebere število n in izpiše program, ki problem izpisa vseh nizov z n črkami reši s pomočjo n vgnezdenih zank. Natančno se zgledujte po primeru, prikazanem v nadaljevanju.

Vhod

Na vhodu je podano celo število $n \in [1, 100]$.

Izhod

Na izhod izpišite program po zgledu sledečega testnega primera. Natančno se držite števila presledkov med posameznimi elementi izpisanega programa. Ne izpisujte tabulatorjev, odvečnih presledkov in praznih vrstic.

Primer

Testni vhod:

```
4
```

Pričakovani izhod:

```
public class Nizi {
    public static void main(String[] args) {
        for (char c1 = 'A';  c1 <= 'Z';  c1++) {
            for (char c2 = 'A';  c2 <= 'Z';  c2++) {
                for (char c3 = 'A';  c3 <= 'Z';  c3++) {
                    for (char c4 = 'A';  c4 <= 'Z';  c4++) {
                        System.out.println("" + c1 + c2 + c3 + c4);
                    }
                }
            }
        }
    }
}
```

Pričakovani izhod s prikazanimi presledki:

```
public␣class␣Nizi␣{
␣␣␣␣public␣static␣void␣main(String[]␣args)␣{
␣␣␣␣␣␣␣␣for␣(char␣c1␣=␣'A';␣␣c1␣<=␣'Z';␣␣c1++)␣{
␣␣␣␣␣␣␣␣␣␣␣␣for␣(char␣c2␣=␣'A';␣␣c2␣<=␣'Z';␣␣c2++)␣{
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣for␣(char␣c3␣=␣'A';␣␣c3␣<=␣'Z';␣␣c3++)␣{
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣for␣(char␣c4␣=␣'A';␣␣c4␣<=␣'Z';␣␣c4++)␣{
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣System.out.println(""␣+␣c1␣+␣c2␣+␣c3␣+␣c4);
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣}
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣}
␣␣␣␣␣␣␣␣␣␣␣␣}
␣␣␣␣␣␣␣␣}
␣␣␣␣}
}
```

Napotek

Če želite *izpisati* enojne ali dvojne navednice s pomočjo ukazov `System.out.print*`, uporabite zaporedje `\"` oz. `\'`. Na primer, ukaz

```
System.out.println("Znak \'a\' nastopa v nizu \"miza\".");
```

izpiše besedilo

```
Znak 'a' nastopa v nizu "miza".
```

3.8 Razbijanje števil

Naloga

Napišite program, ki prebere pozitivni celi števili n in m, nato pa po vrsti izpiše posamezne dele (zaporedja števk) števila n, pri čemer je dolžina vsakega dela določena s pripadajočo števko v številu m. Dolžina prvega dela je tako enaka prvi števki števila m, dolžina drugega dela je enaka drugi števki števila m itd.

Nalogo rešite zgolj z operacijami nad celimi števili. Uporaba realnoštevilskih operacij ter nizov, tabel ipd. ni dovoljena.

Vhod

Na vhodu sta podani celi števili $n \in [1, 10^{18}]$ in $m \in [1, 10^{18}]$, pri čemer je vsota števk števila m enaka številu števk števila n. Število m se ne konča z ničlo. Števili sta ločeni s presledkom.

Izhod

Na izhodu izpišite toliko vrstic, kolikor je števk števila m. V prvi vrstici izpišite začetnih a_1 števk števila n (pri čemer je a_1 prva števka števila m), v drugi sledečih a_2 števk števila n (pri čemer je a_2 druga števka števila m) itd.

Primer 1

Testni vhod:

```
362903157 2313
```

Pričakovani izhod:

```
36
290
3
157
```

Primer 2

Testni vhod:

```
123456789087654321 72315
```

Pričakovani izhod:

```
1234567
89
87
6
54321
```

Tretji del števila n je zapisan kot `87`, ne kot `087`, saj začetno ničlo pri izpisu celih števil izpuščamo.

4 Osnovni konstrukti IV

Splošna navodila

Pri vseh nalogah v tem sklopu lahko svoje rešitve preverite z množico vhodnih datotek in pripadajočih izhodnih datotek.

4.1 Trojice števil

Naloga

Napišite program, ki prebere pozitivni celi števili n in d ter izpiše vse trojice števil od 1 do vključno n, katerih največji skupni delitelj (GCD) je enak d. Vsaka trojica naj se izpiše samo enkrat. GCD treh števil je definiran s formulo $\gcd(a, b, c) = \gcd(\gcd(a, b), c)$.

Vhod

Na vhodu sta podani celi števili $n \in [1, 10^6]$ in $d \in [1, n]$. Med seboj sta ločeni s presledkom.

Izhod

Na izhod izpišite vse iskane trojice. Vsako trojico izpišite v svojo vrstico, med elementi trojice pa izpišite po en presledek. Elementi trojice naj bodo urejeni naraščajoče, trojice med seboj pa leksikografsko. Trojica (a_1, b_1, c_1) je leksikografsko manjša od trojice (a_2, b_2, c_2), če velja

$$(a_1 < a_2) \vee (a_1 = a_2 \wedge b_1 < b_2) \vee (a_1 = a_2 \wedge b_1 = b_2 \wedge c_1 < c_2)$$

Primer

Testni vhod:

```
10 3
```

Pričakovani izhod:

```
3 3 3
3 3 6
3 3 9
3 6 6
3 6 9
3 9 9
6 6 9
6 9 9
```

4.2 Vraževerni Boris

Naloga

Boris vsak dan ponavlja sledeči obred za odganjanje zlih duhov: igralno kocko meče tako dolgo, dokler trikrat (v celotnem obredu) ne pade liho število pik. Ob nedeljah obred zaključi šele, ko liho število pik pade petkrat. Napišite program, ki prebere seme naključnega generatorja s in pozitivno celo število n, nato pa simulira Borisov obred za n dni, začenši s ponedeljkom. Za vsak dan naj program izpiše še skupno število metov kocke.

Navodila za uporabo semena naključnega generatorja so podana v razdelku *Napotek* ob koncu te naloge.

Vhod

Na vhodu sta podani celi števili $s \in [1, 10^9]$ in $n \in [1, 10^4)$. Med seboj sta ločeni s presledkom.

Izhod

Na izhod izpišite n vrstic. Vsaka vrstica naj bo zapisana v formatu

D␣`(`T`):`␣M␣`[`S`]`

pri čemer je

- D zaporedna številka dneva, zapisana na 4 mesta (npr. ␣␣␣`3` ali ␣`735`);
- T oznaka dneva v tednu (`D`: delovnik ali sobota, `N`: nedelja);
- M zaporedje rezultatov metov kocke, med seboj ločenih s po enim presledkom;
- S skupno število metov v tekočem dnevu.

Primer

Testni vhod:

```
123456 22
```

Pričakovani izhod:

```
  1 (D): 4 6 2 2 5 2 4 5 1 [9]
  2 (D): 2 4 3 4 1 1 [6]
  3 (D): 4 5 3 2 2 4 1 [7]
  4 (D): 4 3 1 4 5 [5]
  5 (D): 6 3 1 4 1 [5]
  6 (D): 6 3 4 5 5 [5]
  7 (N): 4 4 6 6 6 5 3 3 5 6 3 [11]
  8 (D): 3 6 2 3 4 1 [6]
  9 (D): 4 3 6 2 1 3 [6]
 10 (D): 6 1 5 1 [4]
 11 (D): 3 4 1 5 [4]
 12 (D): 5 1 2 4 1 [5]
 13 (D): 3 6 4 6 1 6 6 4 4 3 [10]
 14 (N): 3 6 6 4 3 3 6 4 4 3 4 4 1 [13]
 15 (D): 2 5 3 6 2 3 [6]
 16 (D): 4 4 6 1 1 2 4 6 5 [9]
 17 (D): 6 2 2 1 3 4 4 6 4 3 [10]
 18 (D): 6 5 1 4 4 2 2 3 [8]
 19 (D): 6 2 6 2 3 1 2 1 [8]
 20 (D): 1 2 3 5 [4]
 21 (N): 3 6 5 6 3 1 1 [7]
 22 (D): 4 4 5 3 3 [5]
```

Napotek

Naključne rezultate metov tvorite s pomočjo razreda `Random` iz paketa `java.util`. Natančno se držite sledeče sheme, sicer se vaši izhodi ne bodo ujemali s testnimi:

```
// na začetku programa
Random rand = new Random(s);

// pri tvorbi posameznih rezultatov metov
int stPik = rand.nextInt(a) + b;
```

Pri tem je s prebrano seme, a in b pa sta celi števili, ki morata seveda imeti ustrezni vrednosti. (Oglejte si dokumentacijo metode `nextInt` razreda `Random`.)

4.3 Zdolgočasena Mojca

Naloga

Mojca preganja dolgčas z metanjem igralnih kock. V vsakem metu hkrati vrže k kock. Vsak dan izvaja mete tako dolgo, dokler ni vsota vseh k kock v tekočem metu praštevilo. Napišite program, ki s pomočjo generatorja naključnih števil simulira Mojčino igro za d dni.

Vhod

Na vhodu so v sledečem zaporedju podana tri cela števila, ločena s po enim presledkom:

- seme generatorja naključnih števil (v intervalu $[1, 10^9]$);
- $k \in [1, 10^3]$
- $d \in [1, 10^3]$

Izhod

Za vsak dan simulacije naj program najprej izpiše vrstico oblike

D`.␣dan`

kjer je D zaporedna številka tekočega dne. Nato pa naj izpiše podatke o posameznih metih. Za vsak met naj se izpiše vrstica oblike

␣␣␣␣M`.␣met:␣`*kocke*`␣|␣vsota␣=␣`V

kjer je M zaporedna številka meta, *kocke* zaporedje števil, ločenih s po enim presledkom, ki podajajo število pik na posameznih kockah, V pa vsota pik za tekoči met.

Primer

Testni vhod:

```
12345 5 7
```

Pričakovani izhod:

```
1. dan:
    1. met: 2 5 4 1 2 | vsota = 14
    2. met: 5 2 1 2 4 | vsota = 14
    3. met: 6 1 5 5 3 | vsota = 20
    4. met: 5 6 2 3 4 | vsota = 20
    5. met: 6 2 5 5 3 | vsota = 21
    6. met: 4 6 6 6 4 | vsota = 26
    7. met: 4 6 4 2 6 | vsota = 22
    8. met: 6 3 4 5 5 | vsota = 23
2. dan:
```

```
    1. met: 1 5 4 3 3 | vsota = 16
    2. met: 5 1 2 3 1 | vsota = 12
    3. met: 1 5 1 5 3 | vsota = 15
    4. met: 4 3 4 5 4 | vsota = 20
    5. met: 1 4 2 5 6 | vsota = 18
    6. met: 2 1 2 1 2 | vsota = 8
    7. met: 3 6 5 5 5 | vsota = 24
    8. met: 1 4 5 3 2 | vsota = 15
    9. met: 4 5 1 1 4 | vsota = 15
    10. met: 5 4 6 3 5 | vsota = 23
3. dan:
    1. met: 2 2 2 4 2 | vsota = 12
    2. met: 4 2 3 1 2 | vsota = 12
    3. met: 5 5 4 6 5 | vsota = 25
    4. met: 2 3 4 1 3 | vsota = 13
4. dan:
    1. met: 3 4 3 3 1 | vsota = 14
    2. met: 3 3 4 3 4 | vsota = 17
5. dan:
    1. met: 3 2 5 6 2 | vsota = 18
    2. met: 1 6 2 2 4 | vsota = 15
    3. met: 3 1 4 5 6 | vsota = 19
6. dan:
    1. met: 6 3 2 6 6 | vsota = 23
7. dan:
    1. met: 3 3 2 1 3 | vsota = 12
    2. met: 3 4 3 3 6 | vsota = 19
```

Napotek

Generator naključnih števil uporabite na enak način kot v nalogi Vraževerni Boris.

4.4 Šahovski popoldnevi

Naloga

Andrej in Branko vsak dan šahirata. Vsaka njuna partija se z verjetnostjo $a\%$ zaključi z zmago Andreja, z verjetnostjo $b\%$ z zmago Branka in s preostalo verjetnostjo z remijem. Njun dnevni dvoboj se zaključi po p partijah oziroma takrat, ko eden od njiju nabere z zmag (odvisno od tega, kaj se zgodi prej).

Napišite program, ki s pomočjo generatorja naključnih števil izvede simulacijo opisanega šahovskega dvoboja za d dni. Program naj za vsak dan izpiše zaporedje izidov in število odigranih partij.

Vhod

Na vhodu je v sledečem zaporedju zapisanih šest celih števil, ločenih s po enim presledkom:

- seme generatorja naključnih števil (v intervalu $[1, 10^9]$);
- $a \in [0, 100]$
- $b \in [0, 100 - a]$
- $p \in [1, 10^4]$
- $z \in [0, 10^4]$
- $d \in [1, 10^3]$

Izhod

Za vsak dan izpišite po eno vrstico v obliki

D`.␣dan:␣`*zaporedje*`␣(`N`)`

kjer je D številka tekočega dneva, *zaporedje* zaporedje znakov `A`, `B` in `r`, ki ponazarjajo izide posameznih partij v tekočem dnevu (`A`: zmaga Andreja; `B`: zmaga Branka; `r`: remi), N pa število odigranih partij v tekočem dnevu.

Primer

Testni vhod:

```
12345 50 30 8 3 10
```

Pričakovani izhod:

```
1. dan: BrAABrB (7)
2. dan: AArA (4)
3. dan: ArAA (4)
4. dan: rrAABA (6)
5. dan: BrAArA (6)
6. dan: ABAA (4)
7. dan: rAAA (4)
8. dan: BBAAA (5)
9. dan: rrrBArAr (8)
10. dan: BABAB (5)
```

V 9. dnevu sta Andrej in Branko zaključila po skupno p (osmih) odigranih partijah, v vseh ostalih dneh pa po k (treh) zmagah enega od njiju.

Napotek

Verjetnostno pogojene izide generirajte tako, da tvorite naključno celo število med 0 in vključno 99 (s pomočjo metode `nextInt` razreda `Random`), nato pa upoštevate sledeča pravila:

- če je dobljeno število manjše od a, zmaga Andrej;
- če je dobljeno število v intervalu $[a,\ a + b)$, zmaga Branko;
- sicer se partija zaključi z remijem.

5 Razredi in objekti

Splošna navodila

V tem sklopu nalog uporabljamo nekoliko drugačen pristop kot pri prejšnjih. Pri vseh nalogah zahtevamo realizacijo nekega razreda in njegovih javno dostopnih konstruktorjev in metod. Glede ostalih komponent (atributi, morebitne privatne metode oz. konstruktorji itd.) ste svobodni, seveda pa upoštevajte splošne smernice objektno usmerjenega programiranja. Svoje rešitve boste lahko preverili z množico testnih razredov in pripadajočih izhodnih datotek. Testni razredi so zasnovani tako, da učinkovitost vaše rešitve ni bistven dejavnik. Glavno merilo naj bo tokrat »eleganca« rešitve, ne njena optimalnost v smislu prostorsko-časovne učinkovitosti. Pri elegantni rešitvi se, denimo, metode »rade« kličejo med seboj.

5.1 Razreda `Posta` in `Pismo`

Naloga

V skladu s sledečimi navodili napišite razreda `Posta` in `Pismo`.

Razred `Posta`

Razred `Posta` definirajte tako, da bo vsak njegov objekt predstavljal neko pošto s poštno številko (npr. 1000) in nazivom (npr. Ljubljana). Razred naj vsebuje sledeče konstruktorje/metode:

- `public Posta(int stevilka, String naziv)`

  Ustvari nov objekt tipa `Posta`, ki predstavlja pošto s podano poštno številko in nazivom.

- `public int vrniStevilko()`

Vrne poštno številko pošte `this`.

- `public String vrniNaziv()`

  Vrne naziv pošte `this`.

- `public String toString()`

  Za pošto `this` vrne niz sledeče oblike:

  stevilka␣*naziv*

  Na primer:

  `1000␣Ljubljana`

Razred `Pismo`

Razred `Pismo` definirajte tako, da bo vsak njegov objekt predstavljal neko pismo s sledečimi podatki:

- izvorna pošta (npr. 1000 Ljubljana);
- ciljna pošta (npr. 2000 Maribor);
- podatek o tem, ali je pismo priporočeno ali navadno;
- razdalja (v kilometrih) med izvorno in ciljno pošto.

V razredu definirajte sledeče javno dosopne konstruktorje/metode:

- `public Pismo(Posta izvorna, Posta ciljna, boolean jePriporoceno, int razdalja)`

  Ustvari nov objekt tipa `Pismo`, ki predstavlja pismo s podano izvorno in ciljno pošto, »priporočenostjo« (`true`: priporočeno; `false`: navadno) ter razdaljo (v kilometrih) med izvorno in ciljno pošto.

- `public String toString()`

  Za pismo `this` vrne niz oblike

  izvorna_pošta␣`->`␣*ciljna_pošta*␣`(`*razdalja*␣`km)`␣`[`*vrsta*`]`

  pri čemer je *vrsta* bodisi `P` (priporočeno) ali pa `N` (navadno). Na primer:

  `1000␣Ljubljana␣->␣2000␣Maribor␣(130␣km)␣[P]`

- `public boolean izviraOd(Posta posta)`

  Vrne `true` natanko v primeru, če je pošta `posta` izvorna pošta za pismo `this`.

- `public boolean staIzvorInCiljIsta()`

  Vrne `true` natanko v primeru, če ima pismo `this` isto izvorno in ciljno pošto (npr. če je pismo poslano s pošte 1000 Ljubljana na pošto 1000 Ljubljana).

- `public boolean imaIstiCiljKot(Pismo pismo)`

  Vrne `true` natanko v primeru, če ima pismo `this` isto ciljno pošto kot pismo `pismo`.

- `public static boolean imataIstiCilj(Pismo p1, Pismo p2)`

  Vrne `true` natanko v primeru, če imata obe podani pismi isto ciljno pošto.

- `public int cena()`

Vrne ceno (v stotinih) oddaje pisma `this`. Za navadno pismo se cena izračuna glede na razdaljo: za razdaljo od 0 do vključno $(r-1)$ km je cena enaka k stotinov, za razdaljo od r km do vključno $(2r-1)$ km je cena enaka $2k$ stotinov, za razdaljo od $2r$ km do vključno $(3r-1)$ km je cena enaka $3k$ stotinov itd. Ceno priporočenega pisma izračunamo tako, da ceni navadnega pisma prištejemo priporočnino p stotinov, ki je neodvisna od razdalje. Konstante k, r in p se nastavijo z metodo, ki jo predstavljamo v naslednji alineji.

- `public static void nastaviKonstanteZaCeno(int enotaRazdalje, int enotaCene, int priporocnina)`

  Konstante r, k in p, ki se uporabljajo za izračun cene oddaje pisma (gl. prejšnjo alinejo), nastavi na vrednosti `enotaRazdalje`, `enotaCene` in `priporocnina` (v tem vrstnem redu).

- `public boolean jeDrazjeOd(Pismo pismo)`

  Vrne `true` natanko v primeru, če je cena pisma `this` večja od cene pisma `pismo`.

- `public static Pismo vrniDrazje(Pismo p1, Pismo p2)`

  Vrne tisto pismo izmed `p1` in `p2`, ki ima večjo ceno. Če imata obe pismi enako ceno, naj vrne pismo `p2`.

- `public Pismo izdelajPovratno()`

  Ustvari in vrne nov objekt tipa `Pismo`, ki predstavlja povratnico pisma `this`. Povratnica ima enake podatke kot pismo `this`, le izvorna in ciljna pošta sta med seboj zamenjani.

5.2 Razred `Ulomek`

Naloga

Napišite razred `Ulomek` tako, da bodo njegovi objekti predstavljali posamezne okrajšane ulomke. Razred `Ulomek` naj vsebuje sledeče konstruktorje/metode:

- `public Ulomek(int a, int b)`

  Ustvari nov objekt tipa `Ulomek`, ki predstavlja okrajšano različico ulomka `a`/`b`. Ulomek p/q je okrajšan natanko tedaj, ko velja $q > 0$ in $\gcd(p,q) = 1$. Na primer, okrajšana različica ulomka 15/5 je ulomek 3/1, okrajšana različica ulomka $10/(-20)$ pa je ulomek $-1/2$. Lahko predpostavite, da sta števili `a` in `b` različni od 0.

- `public String toString()`

  Vrne okrajšani ulomek `this` v obliki niza *števec*/*imenovalec*, npr. `3/1` ali `-1/2`.

- `public boolean jeEnakKot(Ulomek u)`

  Vrne `true` natanko v primeru, če sta ulomka `this` in `u` enaka.

- `public Ulomek negacija()`

  Ustvari in vrne nov objekt, ki predstavlja nasprotno vrednost ulomka `this` (torej $-x$, če je x dani ulomek).

- `public Ulomek obrat()`

  Ustvari in vrne nov objekt, ki predstavlja obratno vrednost ulomka `this` (torej $1/x$, če je x dani ulomek).

- `public Ulomek vsota(Ulomek u)`
  `public Ulomek razlika(Ulomek u)`
  `public Ulomek zmnozek(Ulomek u)`
  `public Ulomek kolicnik(Ulomek u)`

  Ustvari in vrne nov objekt, ki predstavlja vsoto, razliko, zmnožek oziroma količnik ulomka `this` in ulomka `u`.

- `public Ulomek potenca(int eksponent)`

  Vrne potenco ulomka `this` na podani eksponent. Eksponent ni nujno pozitiven!

- `public boolean jeManjsiOd(Ulomek u)`

  Vrne `true` natanko v primeru, če je ulomek `this` manjši od ulomka `u`.

Nespremenljivi objekti

Metode, ki jih morate realizirati, so zasnovane tako, da ne spreminjajo stanja objekta `this`. Če boste to načelo uporabili za vse metode vašega razreda `Ulomek`, bodo objekti tipa `Ulomek` *nespremenljivi* (angl. immutable). Stanja nespremenljivih objektov po izdelavi ne moremo več spreminjati. Nespremenljivi objekti imajo vrsto dobrih lastnosti in lahko programerju prihranijo marsikatero skrb:

`http://www.javapractices.com/topic/TopicAction.do?Id=29`

Na primer, sledeči potencialno zahrbtni pojavi so možni zgolj pri spremenljivih objektih, saj nespremenljivi po definiciji sploh ne morejo imeti metod vrste »setter«:

```
Delavec d = new Delavec(12345, "Gorišek", "Jože", 150);
Delavec d1 = d;
d.nastaviStUr(200);
d.izpisi();     // Jože Gorišek (12345), 200 ur
d1.izpisi();    // Jože Gorišek (12345), 200 ur (past!)
```

5.3 Razred `Datum`

Naloga

Napišite razred `Datum` tako, da bodo njegovi objekti predstavljali veljavne datume po gregorijanskem koledarju. Hraniti želimo datume od 1. januarja 1583 (torej od leta, ko se je uveljavil gregorijanski koledar) do 31. decembra 2999. Razred `Datum` naj vsebuje sledeče metode:

- `public static Datum ustvari(int dan, int mesec, int leto)`

  Če podani parametri predstavljajo veljaven datum (`leto` med 1583 in 2999, `mesec` od 1 do 12, `dan` od 1 do števila dni v izbranem mesecu izbranega leta), naj metoda ustvari in vrne nov objekt razreda `Datum`, sicer pa naj vrne `null`. Pri preverjanju

veljavnosti datuma upoštevajte pravilo za prestopna leta, ki se glasi takole: leto je prestopno, če je deljivo s 400 ali pa če je deljivo s 4, vendar ni deljivo s 100. Leta 1700, 1800, 1900, 2100, 2200, 2300, 2500, ... tako niso prestopna, leta 1600, 2000, 2400, ... pa so.

Zakaj smo se v tem primeru odločili za statično konstrukcijsko metodo namesto za konstruktor? Ali je konstruktor kljub tej metodi smiselno napisati? Če da, naj bo javno dostopen ali privaten? (Namig: radi bi, da objekti tipa `Datum` predstavljajo izključno veljavne datume.)

- `public String toString()`

  Vrne predstavitev datuma v obliki niza `DD.MM.LLLL`, npr. `15.07.2014` ali `05.11.1975`.

- `public boolean jeEnakKot(Datum datum)`

  Vrne `true` natanko v primeru, če objekt `this` predstavlja isti datum kot objekt `datum`.

- `public boolean jePred(Datum datum)`

  Vrne `true` natanko v primeru, če je datum `this` kronološko pred datumom `datum`. Na primer, datum 30.09.2014 je pred datumom 27.10.2014, ta pa je pred datumom 20.01.2015.

- `public Datum naslednik()`

  Ustvari in vrne nov objekt razreda `Datum`, ki predstavlja neposrednega naslednika datuma `this`. Na primer, naslednik datuma 28.02.2012 je datum 29.02.2012, njegov naslednik pa je datum 01.03.2012. Naslednik datuma 28.02.2014 je datum 01.03.2014. Če datum nima veljavnega naslednika (edini tak datum je 31.12.2999), naj metoda vrne `null`.

- `public Datum predhodnik()`

  Ustvari in vrne nov objekt razreda `Datum`, ki predstavlja neposrednega predhodnika datuma `this`. Na primer, predhodnik datuma 01.03.2012 je datum 29.02.2012, predhodnik datuma 01.03.2014 pa je datum 28.03.2014. Če datum nima veljavnega predhodnika (edini tak datum je 01.01.1583), naj metoda vrne `null`.

- `public Datum cez(int stDni)`

  Izračuna (in vrne kot nov objekt tipa `Datum`) datum, ki je za `stDni` oddaljen od datuma `this`. Parameter `stDni` je lahko tudi negativen; v tem primeru metoda izračuna datum, ki je `-stDni` pred datumom `this`. Če ciljni datum pade pred datum 01.01.1583 ali za datum 31.12.2999, naj metoda vrne `null`.

- `public int razlika(Datum datum)`

  Vrne razliko (v številu dni) med datumoma `this` in `datum`. Če je datum `this` pred datumom `datum`, je razlika seveda negativna.

Tovarna

Statična konstrukcijska metoda se imenuje *tovarna* (angl. factory method). Tovarne so namenjene nadzorovani izdelavi objektov in se zato uporabljajo v kombinaciji s privatnimi konstruktorji. V tej nalogi smo tovarno izdelali zato, ker želimo zagotoviti, da objekti razreda `Datum` predstavljajo samo veljavne datume. Samo s konstruktorjem tega ne bi mogli doseči, saj konstruktor ne more preprečiti izdelave objekta.

5.4 Dopolnitve razreda `Oseba` (⋆)

Naloga

V razred `Oseba`, ki smo ga napisali na vajah in ki ga najdete tudi v mapi z izhodiščnimi datotekami za to nalogo, dodajte sledeče metode:

- `public int ocetovskaGeneracijskaRazlika(Oseba os)`

  Vrne očetovsko generacijsko razliko med osebama `this` in `os` ($OGR(\texttt{this}, \texttt{os})$). Vrednost $OGR(A, B)$ za osebi A in B je definirana takole:

  - Če sta osebi A in B identični (če gre za isto osebo), velja $OGR(A, B) = 0$.
  - Če je oseba A očetovski prednik osebe B, potem $OGR(A, B)$ izračunamo kot število očetov na liniji od B do A. (Če je oseba A oče osebe B, velja $OGR(A, B) = 1$, če je oseba A oče očeta osebe B, velja $OGR(A, B) = 2$ itd.)
  - Če je oseba B očetovski prednik osebe A, velja $OGR(A, B) = -OGR(B, A)$.
  - Če nobeden od zgornjih pogojev ni izpolnjen, naj metoda namesto vračila vrednosti (torej namesto stavka `return`) sproži izjemo tipa `IllegalArgumentException`:

    ```
    throw new IllegalArgumentException();
    ```

- `public boolean jePrednikOd(Oseba os)`

  Vrne `true` natanko v primeru, če je oseba `this` prednik osebe `os`. Oseba A je prednik osebe B, če je izpolnjen eden od sledečih pogojev:

  - $A = B$ (vsaka oseba je sam svoj prednik).
  - A je prednik B-jevega očeta.
  - A je prednik B-jeve matere.

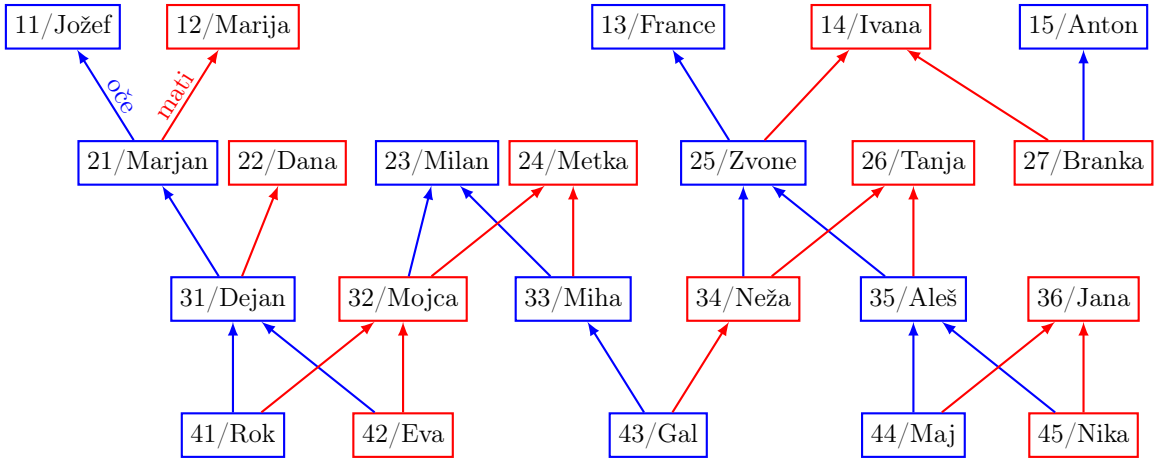
- `public void nastejPrednike()`

  Izpiše vse prednike osebe `this`. Vsak prednik naj se izpiše v svoji vrstici, in sicer na sledeči način:

  veriga`:␣`*prednik*

  Pri tem je *veriga* niz, ki ponazarja starševsko verigo od osebe `this` do prednika, izpisanega v tekoči vrstici, *prednik* pa niz, ki ga za danega prednika vrne metoda `toString`. Niz *veriga* izdelajte po sledečem zgledu: niz `this` predstavlja osebo `this`, niz `this.oce` predstavlja očeta osebe `this`, niz `this.oce.mati` predstavlja mater očeta osebe `this` itd. Najprej naj se na opisani način izpišejo vsi predniki po očetovi strani, nato pa še vsi predniki po materini strani. Enako pravilo naj se rekurzivno uporabi za posamezne prednike. Za primer s slike 5.1 bi klic `os43.nastejPrednike()` izpisal sledeče:

  ```
  this: Gal Smole [M] (2009)
  this.oce: Miha Smole [M] (1978)
  this.oce.oce: Milan Smole [M] (1953)
  this.oce.mati: Metka Smole [Z] (1953)
  this.mati: Neža Smole [Z] (1980)
  ```



Slika 5.1: *Starševski odnosi med osebami v testnem razredu.*

```
this.mati.oce: Zvone Kotnik [M] (1956)
this.mati.oce.oce: France Kotnik [M] (1932)
this.mati.oce.mati: Ivana Kotnik [Z] (1931)
this.mati.mati: Tanja Kotnik [Z] (1954)
```

- `public boolean jeSorodnikOd(Oseba os)`

  Vrne `true` natanko v primeru, če sta osebi `this` in `os` sorodnika. Osebi A in B sta sorodnika, če obstaja oseba C, ki je prednik tako osebe A kot osebe B. (Tudi sorodstvo je možno definirati na eleganten rekurzivni način. Odkrijte ga sami!)

  V primeru z vaj (slika 5.1) sta osebi `os41` in `os43` (Rok in Gal) sorodnika. Sorodnika sta tudi osebi `os24` in `os42` (Metka in Eva), osebi `os33` in `os34` (Miha in Neža) pa nista.

6 Enorazsežne tabele

Splošna navodila

Pri vseh nalogah v tem sklopu lahko svoje rešitve preverite z množico vhodnih datotek in pripadajočih izhodnih datotek.

6.1 Najbližje povprečju

Naloga

Napišite program, ki prebere zaporedje celih števil in izpiše *indeks* tistega števila, ki je od povprečja zaporedja najmanj oddaljeno (ni pomembno, ali v pozitivno ali v negativno smer). Če je takih števil več, naj program izpiše indeks prvega od njih.

Vhod

V prvi vrstici vhoda je podano celo število $n \in [1, 10^4]$, v drugi pa zaporedje n števil z intervala $[-10^5, 10^5]$, ločenih s po enim presledkom.

Izhod

Na izhodu izpišite samo iskani indeks (z metodo `System.out.println`, kot smo že navajeni).

Primer

Testni vhod:

```
5
3 9 6 4 -1
```

Pričakovani izhod:

```
3
```

Od povprečja, ki znaša 4.2, je najmanj oddaljeno število 4 (torej število z indeksom 3).

6.2 Digitalne črtice

Naloga

Napišite program, ki prebere zaporedje pozitivnih celih števil in izpiše, katero od njih bi bilo na kalkulatorju s klasičnim digitalnim prikazovalnikom zapisano z največ črticami. Če je takih števil v zaporedju več, naj izpiše prvo od njih.

Sledeča preglednica podaja število črtic, iz katerih so sestavljene posamezne števke:

Števka	0	1	2	3	4	5	6	7	8	9
Število črtic	6	2	5	5	4	5	6	3	7	6

Vhod

V prvi vrstici vhoda je zapisano celo število $n \in [1, 10^3]$, v drugi pa zaporedje n celih števil z intervala $[1, 10^9]$, med seboj ločenih s po enim presledkom.

Izhod

Na izhodu izpišite samo iskano število v zaporedju.

Primer 1

Testni vhod:

```
8
4567 888 1113111 90 2352 211 9 63
```

Pričakovani izhod:

```
888
```

Primer 2

Testni vhod:

```
3
35 61 127
```

Pričakovani izhod:

```
35
```

Števili 35 in 127 sta obe zapisani z 10 črticami, program pa izpiše 35, ker se v zaporedju pojavi prej.

6.3 Pascalov trikotnik

Naloga

Napišite program, ki prebere celo število n, nato pa izpiše Pascalov trikotnik višine $n + 1$. V Pascalovem trikotniku je i-ta vrstica (za $i \in \{1, \ldots, n + 1\}$) sestavljena iz i števil. V vsaki vrstici sta prvo in zadnje število enaki 1. Za $1 < j < i$ pa se število $p_{i,\,j}$ (j-to število v i-ti vrstici) izračuna po formuli $p_{i,\,j} = p_{i-1,\,j-1} + p_{i-1,\,j}$.

Vhod

Na vhodu je podano samo celo število $n \in [0, 30]$.

Izhod

Na izhodu izpišite Pascalov trikotnik višine $n + 1$. Vsako vrstico trikotnika izpišite v svoji vrstici. Števila znotraj iste vrstice naj bodo ločena s po enim presledkom. Na koncu vrstic ne sme biti presledkov!

Primer 1

Testni vhod:

```
4
```

Pričakovani izhod:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

Pričakovani izhod s prikazanimi presledki:

```
1
1␣1
1␣2␣1
1␣3␣3␣1
1␣4␣6␣4␣1
```

Primer 2

Testni vhod:

```
10
```

Pričakovani izhod:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
```

6.4 Telefonski imenik

Naloga

Napišite program, ki najprej prebere n osebnih imen in pripadajočih telefonskih številk, nato pa prebere še k osebnih imen in za vsako izpiše pripadajočo telefonsko številko. Za imena, ki se ne pojavljajo med n imeni s pripisanimi telefonskimi številkami, naj program izpiše niz `NEZNANA`. V seznamu imen s pripisanimi telefonskimi številkami se lahko imena tudi ponavljajo; v tem primeru velja zadnja telefonska številka.

Vhod

V prvi vrstici vhoda je podano celo število $n \in [0, 10^3]$. Nato sledi n vrstic, od katerih je v vsaki najprej zapisano ime v obliki niza do 20 črk angleške abecede, zatem presledek, nazadnje pa pripadajoča telefonska številka v obliki niza do 20 števk ter znakov `+` in `-`. V naslednji vrstici je zapisano celo število $k \in [1, 10^3]$. Nato sledi k vrstic, od katerih je v vsaki zapisano samo ime v obliki niza do 20 črk angleške abecede.

Izhod

Na izhodu izpišite k vrstic. V i-ti vrstici izpišite telefonsko številko, ki pripada imenu v i-ti vrstici znotraj skupine vrstic brez pripisanih telefonskih številk. Če ime ne nastopa v skupini vrstic s pripisanimi telefonskimi številkami, izpišite niz `NEZNANA`.

Primer

Testni vhod:

```
7
Mojca 01-234-567
Peter 041-317-650
Ivan +386-31-55-72-08
Helga +49-11-22-33-44-55
Polona 059-456-789
Helga +49-66-77-88-99-00
Ivan 031-78-56-34
6
Polona
Mirko
Helga
Iva
Ivan
Mojca
```

Pričakovani izhod:

```
059-456-789
NEZNANA
+49-66-77-88-99-00
NEZNANA
031-78-56-34
01-234-567
```

Napotek

Nize berete zelo podobno kot cela števila, le da namesto klica `sc.nextInt()` (kjer je `sc` objekt tipa `Scanner`) uporabite klic `sc.next()`. Za primerjanje nizov uporabite metodo `equals` (`niz1.equals(niz2)`), ne dvojnega enačaja.

6.5 Zlata sredina

Naloga

Napišite program, ki prebere zaporedje $(2k + 1)$ medsebojno različnih celih števil in izpiše tisto število, od katerega je k števil manjših in k večjih.

Vhod

V prvi vrstici je podano celo število $k \in [0, 10^5]$, v drugi vrstici pa je nanizanih $(2k + 1)$ medsebojno različnih celih števil z intervala $[-10^9, 10^9]$, ločenih s po enim presledkom.

Izhod

Na izhodu izpišite samo iskano število.

Primer

Testni vhod:

```
5
2 10 8 4 9 -6 6 1 3 -4 -2
```

Pričakovani izhod:

```
3
```

6.6 Vsi različni I

Naloga

Napišite program, ki prebere zaporedje celih števil in izpiše `RAZLICNI`, če so vsi elementi v njem medsebojno različni. V nasprotnem primeru naj izpiše najmanjše število, ki v zaporedju nastopa najmanj dvakrat.

Vhod

V prvi vrstici vhoda je podano celo število $n \in [1, 10^4]$, v drugi pa zaporedje n celih števil z intervala $[-10^9, 10^9]$, ločenih s po enim presledkom.

Izhod

Na izhodu izpišite samo niz `RAZLICNI` oziroma najmanjše število, ki se v zaporedju ponovi.

Primer 1

Testni vhod:

```
6
3 9 8 10 6 2
```

Pričakovani izhod:

```
RAZLICNI
```

Primer 2

Testni vhod:

```
7
3 10 9 8 9 2 10
```

Pričakovani izhod:

```
9
```

6.7 Vsi različni II

Naloga

Napišite program, ki za zaporedje celih števil, tvorjeno z naključnim generatorjem, izpiše `RAZLICNI`, če so vsi elementi v njem medsebojno različni. V nasprotnem primeru naj izpiše število, ki v zaporedju največkrat nastopa. Če je takih števil več, naj izpiše najmanjše izmed njih.

Vhod

Na vhodu sta podani dve celi števili, ločeni s presledkom:

- seme naključnega generatorja (v intervalu $[1, 10^9]$);
- dolžina zaporedja (v intervalu $[1, 10^7]$).

Naključni generator uporabite tako:

```
// na vrhu datoteke
import java.util.Random;

// na vrhu razreda
private static final int MAKS_STEVILO = 10000;

// pred pričetkom tvorbe zaporedja
Random random = new Random(seme);

// vsakokrat, ko tvorite člen zaporedja
int clen = random.nextInt(2 * MAKS_STEVILO + 1) - MAKS_STEVILO;
```

Vsi členi zaporedja bodo torej cela števila z intervala $[-10^4, 10^4]$.

Izhod

Na izhodu izpišite samo niz `RAZLICNI` oziroma število, ki se v zaporedju največkrat ponovi.

Primer

Testni vhod:

```
12345 10
```

Pričakovani izhod:

```
RAZLICNI
```

V tem primeru se tvori sledeče zaporedje:

```
-2595 3978 -1933 359 -9393 8040 -3678 -583 8214 9596
```

6.8 Izstopajoči element

Naloga

Dano je zaporedje najmanj treh celih števil, večjih od 1. Definirajmo pojem *izstopajočega elementa* zaporedja na sledeči način: element x *izstopa*, če je GCD (največji skupni delitelj) vseh ostalih elementov v zaporedju večji od 1, sam element x pa s tem GCD-jem ni deljiv. Na primer, v zaporedju $\{25, 40, 15, 36, 30\}$ izstopa element 36, saj je GCD ostalih elementov (25, 40, 15 in 30) enak 5, element 36 pa s tem GCD-jem ni deljiv. Napišite program, ki prebere zaporedje in po vrsti izpiše vse izstopajoče elemente v zaporedju. Če takih elementov ni, naj program izpiše `NIC`.

Vhod

V prvi vrstici je podano število $n \in [3, 10^3]$, v drugi pa n celih števil z intervala $[2, 10^9]$, med seboj ločenih s po enim presledkom.

Izhod

Na izhodu izpišite vse izstopajoče elemente v istem vrstnem redu, kot nastopajo v zaporedju. Vsak izstopajoči element izpišite v svoji vrstici. Če tovrstnih elementov ni, izpišite `NIC`.

Primer 1

Testni vhod:

```
7
24 60 36 18 54 40 48
```

Pričakovani izhod:

```
40
```

Primer 2

Testni vhod:

```
7
24 60 36 18 54 42 48
```

Pričakovani izhod:

```
NIC
```

Primer 3

Testni vhod:

```
3
15 20 18
```

Pričakovani izhod:

```
15
20
18
```

(V tem primeru vsi trije elementi izstopajo.)

6.9 Kombinacije (⋆)

Naloga

Napišite program, ki prebere celi števili n in k, nato pa izpiše vsa strogo naraščajoča zaporedja k števil med 1 in n.

Vhod

Na vhodu sta podani celi števili $n \in [1, 15]$ in $k \in [1, n]$. Med seboj sta ločeni s presledkom.

Izhod

Na izhodu izpišite vsa iskana zaporedja, vsako v svoji vrstici. Zaporedja izpišite v leksikografskem vrstnem redu: najprej naraščajoče po prvem členu, nato (v okviru skupine zaporedij z isto vrednostjo prvega člena) naraščajoče po drugem členu itd. Vsako zaporedje naj bo izpisano v obliki, kot jo proizvede metoda `Arrays.toString`.

Primer 1

Testni vhod:

```
5 3
```

Pričakovani izhod:

```
[1, 2, 3]
[1, 2, 4]
[1, 2, 5]
[1, 3, 4]
[1, 3, 5]
[1, 4, 5]
[2, 3, 4]
[2, 3, 5]
```

```
[2, 4, 5]
[3, 4, 5]
```

Primer 2

Testni vhod:

```
7 6
```

Pričakovani izhod:

```
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 7]
[1, 2, 3, 4, 6, 7]
[1, 2, 3, 5, 6, 7]
[1, 2, 4, 5, 6, 7]
[1, 3, 4, 5, 6, 7]
[2, 3, 4, 5, 6, 7]
```

Primer 3

Testni vhod:

```
2 2
```

Pričakovani izhod:

```
[1, 2]
```

Napotek

Možnih je več pristopov, najelegantnejši pa je gotovo z uporabo rekurzivne metode. Na i-tem nivoju rekurzije obravnavajte vsa možna števila na i-tem mestu v zaporedju. Na prvem nivoju boste tako obravnavali vsa števila med 1 in n, na i-tem nivoju (pri $i \geqslant 2$) pa vsa števila, večja od trenutno izbranega števila na mestu $i - 1$. Za vsako izbrano število na i-tem nivoju rekurzivno obravnavajte vsa števila na nivoju $i + 1$. Na zadnjem nivoju rekurzije izpišite trenutno generirano zaporedje.

6.10 Politična nasprotja (⋆)

Naloga

Na politično konferenco je povabljenih l levičarjev, d desničarjev in c centristov. Organizatorji jih morajo razmestiti na $l + d + c$ zaporedno postavljenih sedežev tako, da levičar in desničar nikjer ne bosta soseda. Napišite program, ki prebere število levičarjev, desničarjev in centristov ter izpiše število vseh sedežnih redov, ki ustrezajo navedenemu pogoju.

Vhod

Na vhodu so podana cela števila $l \in [0, 10]$, $d \in [0, 10]$ in $c \in [0, 5]$. Med seboj so ločena s po enim presledkom.

Izhod

Na izhod izpišite samo število možnih razporeditev. To število bo zagotovo manjše od 10^9.

Primer 1

Testni vhod:

```
2 3 2
```

Pričakovani izhod:

```
15
```

V tem primeru so možne sledeče razporeditve:

```
LLCDDDC
LLCDDCD
LLCDCDD
LLCCDDD
LCLCDDD
LCDDDCL
DDDCLLC
DDDCLCL
DDDCCLL
DDCLLCD
DDCDCLL
DCLLCDD
DCDDCLL
CLLCDDD
CDDDCLL
```

Primer 2

Testni vhod:

```
2 3 1
```

Pričakovani izhod:

```
2
```

V tem primeru sta možni le razporeditvi `LLCDDD` in `DDDCLL`.

7 Splošne tabele

Splošna navodila

Pri nalogah 7.1–7.7 lahko svoje rešitve preverite z množico vhodnih datotek in pripadajočih izhodnih datotek, pri nalogi 7.8 pa z množico testnih razredov in pripadajočih izhodnih datotek.

7.1 Maksimumi po stolpcih I

Naloga

Napišite program, ki prebere celoštevilsko matriko in izpiše maksimalne elemente po posameznih stolpcih.

Vhod

V prvi vrstici vhoda sta podani celi števili $v \in [1, 100]$ in $s \in [1, 100]$, nato pa sledi v vrstic. V vsaki od njih je zapisanih po s celih števil z intervala $[-10^9, 10^9]$, ki tvorijo vsebino pripadajoče vrstice matrike.

Izhod

Na izhodu izpišite maksimalne elemente posameznih stolpcev matrike, in sicer v obliki, kot jo proizvede metoda `Arrays.toString`.

Primer

Testni vhod:

```
5 4
10 -6 -5 1
0 -1 -7 -3
9 5 -4 7
8 3 -3 7
4 2 -6 9
```

Pričakovani izhod:

```
[10, 5, -3, 9]
```

7.2 Maksimumi po stolpcih II

Naloga

Napišite program, ki prebere **ne nujno pravokotno** celoštevilsko matriko in izpiše maksimalne elemente po posameznih stolpcih. Število izpisanih elementov naj bo enako dolžini najdaljše vrstice matrike.

Vhod

V prvi vrstici vhoda je podano celo število $n \in [1, 100]$, nato pa sledi n vrstic. Vsaka od njih se prične s celim številom $d \in [1, 100]$, nato pa sledi d celih števil z intervala $[-10^9, 10^9]$, ki tvorijo vsebino pripadajoče vrstice matrike.

Izhod

Na izhodu izpišite maksimalne elemente posameznih stolpcev matrike, in sicer v obliki, kot jo proizvede metoda `Arrays.toString`.

Primer

Testni vhod:

```
5
5 9 5 3 -2 -4
2 3 -4
5 10 2 -5 -9 8
6 -5 6 -2 -7 -3 -4
4 6 -2 6 -3
```

Pričakovani izhod:

```
[10, 6, 6, -2, 8, -4]
```

V tem primeru vsebuje prvi stolpec matrike števila 9, 3, 10, -5 in 6, drugi števila 5, -4, 2, 6 in -2, ..., zadnji pa samo število -4.

7.3 Pravilni trikotniki

Naloga

Napišite program, ki prebere zaporedje parov celoštevilskih koordinat ravninskih točk in poišče vse trojice točk, ki tvorijo pravilne trikotnike v okviru določene tolerance. Trikotnik proglasimo za pravilnega, če je razlika med dolžino njegove najdaljše stranice in dolžino njegove najkrajše stranice manjša od (10^{-d})-kratnika dolžine njegove najkrajše stranice, kjer je d neko pozitivno celo število. Če pravilnih trikotnikov ni, naj program to sporoči.

Vhod

V prvi vrstici sta podani celi števili $d \in [1, 10]$ in $n \in [1, 100]$, nato pa sledi n vrstic, ki podajajo koordinate posameznih točk. V vsaki vrstici sta zapisani celi števili z intervala $[-2 \cdot 10^4, 2 \cdot 10^4]$, ki po vrsti predstavljata koordinati x in y.

Izhod

Na izhodu izpišite vse trojice indeksov (i, j in k) točk, ki tvorijo pravilne trikotnike. Vsaka trojica naj se izpiše samo enkrat, in sicer v obliki

$i␣j␣k$

pri čemer velja $i < j < k$. Trojice izpišite v leksikografskem vrstnem redu (najprej naraščajoče po indeksih i, nato po indeksih j, nazadnje pa po indeksih k).

Če nobena trojica ne tvori pravilnega trikotnika, naj program izpiše `BREZ`.

Primer 1

Testni vhod:

```
2 10
100 100
-23 287
473 373
300 200
163 163
437 237
400 100
250 360
337 337
200 300
```

Pričakovani izhod:

```
0 1 9
0 6 7
2 5 8
2 6 9
3 4 9
3 5 6
```

```
3 5 8
3 8 9
4 6 8
```

Primer 2

Testni vhod:

```
2 4
10 10
30 20
50 70
10 80
```

Pričakovani izhod:

```
BREZ
```

7.4 Leksikografsko urejanje

Naloga

Vektor $(a_0, a_1, \ldots, a_{n-1})$ je *leksikografsko manjši* od vektorja $(b_0, b_1, \ldots, b_{n-1})$, če obstaja indeks $i \geqslant 0$, tako da velja (1) $a_i < b_i$ in (2) $a_j = b_j$ za vsak $j < i$. Na primer, vektor $a = (5, 3, 4)$ je leksikografsko manjši od vektorjev $b = (5, 3, 8)$ ($a_2 < b_2$, $a_1 = b_1$, $a_0 = b_0$), $c = (5, 6, 1)$ ($a_1 < c_1$, $a_0 = c_0$) in $d = (6, 2, 7)$ ($a_0 < d_0$), ne pa od vektorja $e = (3, 6, 7)$. Napišite program, ki prebere n vektorjev dolžine d, nato pa jih izpiše v leksikografskem vrstnem redu.

Opomba: Povsem enak način urejanja se uporablja pri nizih (npr. pri priimkih, slovarskih geslih itd.), le da tam v vlogi vektorjev nastopajo nizi, v vlogi posameznih elementov vektorjev pa znaki.

Vhod

V prvi vrstici vhoda sta zapisani celi števili $n \in [1, 100]$ in $d \in [1, 100]$. Nato sledi n vrstic po d celih števil z intervala $[-10^9, 10^9]$.

Izhod

Na izhodu izpišite iste vektorje, urejene v leksikografskem vrstnem redu. Vsak vektor izpišite v svoji vrstici v obliki, kot jo proizvede metoda `Arrays.toString`.

Primer 1

Testni vhod:

```
5 3
6 2 7
3 6 7
5 6 1
5 3 4
5 3 8
```

Pričakovani izhod:

```
[3, 6, 7]
[5, 3, 4]
[5, 3, 8]
[5, 6, 1]
[6, 2, 7]
```

Primer 2

Testni vhod:

```
7 5
-3 5 -7 2 -6
-3 4 -2 8 -4
-3 4 -2 8 -9
-5 10 6 10 5
-3 4 2 8 -9
-3 4 -2 5 -4
-3 4 -2 8 -4
```

Pričakovani izhod:

```
[-5, 10, 6, 10, 5]
[-3, 4, -2, 5, -4]
[-3, 4, -2, 8, -9]
[-3, 4, -2, 8, -4]
[-3, 4, -2, 8, -4]
[-3, 4, 2, 8, -9]
[-3, 5, -7, 2, -6]
```

7.5 Šahovski turnir

Naloga

Na šahovskem turnirju nastopa n igralcev. Napišite program, ki prebere rezultate posameznih partij (v obliki »igralec A, ki je vodil bele figure, je proti igralcu B, ki je vodil črne figure, zmagal/izgubil/remiziral«) in izpiše turnirsko lestvico. Turnirska lestvica je seznam igralcev, padajoče urejen po skupnem številu točk. Za potrebe te naloge privzemimo, da

vsaka zmaga prinese po 2 točki, remi po 1 točko, poraz pa po 0 točk. (Dejansko šahovsko točkovanje je $1/\frac{1}{2}/0$, vendar pa bi se radi izognili decimalkam.)

Vhod

V prvi vrstici vhoda je podano celo število $n \in [2, 100]$, nato pa sledi vnaprej neznano število vrstic, ki podajajo rezultate posameznih partij. Vsaka vrstica je sledeče oblike:

štBelega␣*štČrnega*␣*izid*

Pri tem je *štBelega* zaporedna številka igralca z belimi figurami, *štČrnega* zaporedna številka igralca s črnimi figurami, *izid* pa je enak 1 (zmagal je beli), -1 (zmagal je črni) ali 0 (remi). Zaporedne številke so seveda cela števila z intervala $[1, n]$, velja pa tudi *štBelega* $\neq$ *štČrnega*.

Izhod

Na izhodu izpišite n vrstic sledeče oblike:

zapŠtIgralca␣*točke*

Pri tem je *zapŠtIgralca* zaporedna številka igralca, *točke* pa njegova skupna vsota točk. Zaporedje vrstic naj bo urejeno po padajočih točkah, v primeru enakega števila točk pa po naraščajočih zaporednih številkah.

Primer

Testni vhod:

```
5
3 2 1
2 4 0
2 4 -1
5 1 -1
3 5 0
1 3 -1
5 2 1
3 1 -1
```

Pričakovani izhod:

```
3 5
1 4
4 3
5 3
2 1
```

7.6 Determinanta

Naloga

Napišite program, ki prebere kvadratno celoštevilsko matriko in izračuna njeno determinanto po sledeči definiciji (za $n > 1$):

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & a_{34} & \dots & a_{3n} \\ a_{41} & a_{42} & a_{43} & a_{44} & \dots & a_{4n} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \dots & a_{nn} \end{vmatrix} = (-1)^0 \, a_{11} \begin{vmatrix} a_{22} & a_{23} & a_{24} & \dots & a_{2n} \\ a_{32} & a_{33} & a_{34} & \dots & a_{3n} \\ a_{42} & a_{43} & a_{44} & \dots & a_{4n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n2} & a_{n3} & a_{n4} & \dots & a_{nn} \end{vmatrix}$$

$$+ (-1)^1 \, a_{12} \begin{vmatrix} a_{21} & a_{23} & a_{24} & \dots & a_{2n} \\ a_{31} & a_{33} & a_{34} & \dots & a_{3n} \\ a_{41} & a_{43} & a_{44} & \dots & a_{4n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n3} & a_{n4} & \dots & a_{nn} \end{vmatrix}$$

$$+ (-1)^2 \, a_{13} \begin{vmatrix} a_{21} & a_{22} & a_{24} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{34} & \dots & a_{3n} \\ a_{41} & a_{42} & a_{44} & \dots & a_{4n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n4} & \dots & a_{nn} \end{vmatrix}$$

$$+ \dots$$

$$+ (-1)^{n-1} \, a_{1n} \begin{vmatrix} a_{21} & a_{22} & a_{23} & \dots & a_{2,\,n-1} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3,\,n-1} \\ a_{41} & a_{42} & a_{43} & \dots & a_{4,\,n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{n,\,n-1} \end{vmatrix}$$

Determinanta matrike velikosti 1×1 je kar enaka edinemu elementu te matrike.

Opomba: Determinante v praksi ne računamo po gornji definiciji, saj vodi do neučinkovite in numerično nestabilne kode. Z vidika učenja programiranja pa je ta definicija odlična, zato jo v vašem programu striktno upoštevajte.

Vhod

V prvi vrstici vhoda je podano celo število $n \in [1, 8]$, nato pa sledi n vrstic. V vsaki od njih je zapisanih po n celih števil z intervala $[-100, 100]$, ki podajajo vsebino pripadajoče vrstice matrike.

Izhod

Na izhodu izpišite samo determinanto. Determinanta bo po absolutni vrednosti zanesljivo manjša od 10^9.

Primer 1

Testni vhod:

```
1
-15
```

Pričakovani izhod:

```
-15
```

Primer 2

Testni vhod:

```
2
5 6
2 8
```

Pričakovani izhod:

```
28
```

Primer 3

Testni vhod:

```
3
3 4 -2
5 7 8
-3 10 5
```

Pričakovani izhod:

```
-473
```

7.7 Politična nasprotja II (⋆)

Naloga

Opomba: Ta naloga je enaka nalogi *Politična nasprotja* iz sklopa 6, le vhodni podatki so večji.

Na politično konferenco je povabljenih l levičarjev, d desničarjev in c centristov. Organizatorji jih morajo razmestiti na $l + d + c$ zaporedno postavljenih sedežev tako, da levičar in

desničar nikjer ne bosta soseda. Napišite program, ki prebere število levičarjev, desničarjev in centristov ter izpiše število vseh sedežnih redov, ki ustrezajo navedenemu pogoju.

Vhod

Na vhodu so podana cela števila $l \in [0, 20]$, $d \in [0, 20]$ in $c \in [0, 20]$. Med seboj so ločena s po enim presledkom.

Izhod

Na izhod izpišite samo število možnih razporeditev. To število bo zagotovo manjše od 10^{18}.

Primer 1

Testni vhod:

```
2 3 2
```

Pričakovani izhod:

```
15
```

V tem primeru so možne sledeče razporeditve:

```
LLCDDDC
LLCDDCD
LLCDCDD
LLCCDDD
LCLCDDD
LCDDDCL
DDDCLLC
DDDCLCL
DDDCCLL
DDCLLCD
DDCDCLL
DCLLCDD
DCDDCLL
CLLCDDD
CDDDCLL
```

Primer 2

Testni vhod:

```
2 3 1
```

Pričakovani izhod:

```
2
```

V tem primeru sta možni le razporeditvi `LLCDDD` in `DDDCLL`.

7.8 Tabela s poljubnim številom dimenzij (⋆)

Naloga

Napišite in preizkusite razred `Ptabela`, čigar objekt predstavlja neko (hiper-)pravokotno celoštevilsko tabelo s *poljubnim* številom dimenzij (»p-tabela«). Razred naj ponuja sledeče konstruktorje in metode:

- `public Ptabela(int[] dimenzije)`

  Ustvari p-tabelo s podanimi velikostmi dimenzij in jo napolni z ničlami. Lahko predpostavite, da je dolžina tabele `dimenzije` enaka najmanj 1.

  Na primer, stavek

  ```
  Ptabela p = new Ptabela(new int[]{3, 4, 2});
  ```

  bi ustvaril p-tabelo velikosti $3 \times 4 \times 2$.

- `public void nastavi(int[] indeksi, int vrednost)`

  Element p-tabele `this`, določen s podano tabelo indeksov, nastavi na podano vrednost. Lahko predpostavite, da je dolžina tabele `indeksi` enaka številu dimenzij p-tabele `this` in da so vsi indeksi veljavni.

  Na primer, stavek

  ```
  p.nastavi(new int[]{1, 3, 0}, 25);
  ```

  bi nastavil element na poziciji `[1][3][0]` (druga »stran«, četrta vrstica, prvi element znotraj vrstice) na vrednost 25.

- `public int vrni(int[] indeksi)`

  Vrne vrednost elementa p-tabele `this`, določenega s podano tabelo indeksov. Lahko predpostavite, da je dolžina tabele `indeksi` enaka številu dimenzij p-tabele `this` in da so vsi indeksi veljavni.

- `public Ptabela podtabela(int[] indeksi)`

  Vrne nov objekt tipa `Ptabela`, ki predstavlja kopijo pod-p-tabele p-tabele `this`, določene s podano tabelo indeksov. Lahko predpostavite, da je dolžina tabele `indeksi` manjša od števila dimenzij p-tabele `this` in da so vsi indeksi veljavni.

  Na primer, če objekt `p` predstavlja p-tabelo velikosti $3 \times 4 \times 2$, potem bi klic

  ```
  p.podtabela(new int[]{2})
  ```

  vrnil kopijo tretje »strani« p-tabele `p`. Vrnjeni objekt bi torej predstavljal p-tabelo velikosti 4×2. Klic

  ```
  p.podtabela(new int[]{2, 0})
  ```

  pa bi vrnil kopijo prve vrstice tretje »strani« p-tabele `p`.

- `public String toString()`

  Vrne predstavitev p-tabele `this` v obliki niza. Enodimenzionalna p-tabela z d_1 elementi naj bo predstavljena z nizom `[`a_0`, `a_1`, ..., `a_{d_1-1}`]`, kjer so a_0, a_1, ..., a_{d_1-1} elementi p-tabele. P-tabela velikosti $d_1 \times d_2 \times d_3 \times \ldots \times d_n$ naj bo predstavljena z nizom `[`S_0`, `S_1`, ..., `S_{d_1-1}`]`, kjer so S_0, S_1, ..., S_{d_1-1} nizi, ki predstavljajo posamezne pod-p-tabele velikosti $d_2 \times d_3 \times \ldots \times d_n$.

Primer

Testni razred:

```
import java.util.Random;

public class Test1 {

    public static void main(String[] args) {
        // delali bomo s tabelo 3 × 4 × 2
        int a = 3, b = 4, c = 2;

        System.out.println("Inicializacija:");
        Ptabela p = new Ptabela(new int[]{a, b, c});
        System.out.println(p.toString());
        System.out.println("----------");

        System.out.println("Polnjenje z elementi od -99 do 99:");
        Random random = new Random(12345);
        for (int i = 0;  i < a;  i++) {
            for (int j = 0;  j < b;  j++) {
                for (int k = 0;  k < c;  k++) {
                    p.nastavi(new int[]{i, j, k}, random.nextInt(199) - 99);
                }
            }
        }
        System.out.println(p.toString());
        System.out.println("----------");

        System.out.println("Elementi in podtabele:");
        System.out.println("p[2][0][1] = " + p.vrni(new int[]{2, 0, 1}));
        System.out.println("p[2][1] = " + p.podtabela(new int[]{2, 1}).toString());
        System.out.println("p[1] = " + p.podtabela(new int[]{1}).toString());
        System.out.println("p = " + p.podtabela(new int[]{}).toString());
    }
}
```

Pričakovani izhod:

```
Inicializacija:
[[[0, 0], [0, 0], [0, 0], [0, 0]],
 [[0, 0], [0, 0], [0, 0], [0, 0]],
 [[0, 0], [0, 0], [0, 0], [0, 0]]]
----------
Polnjenje z elementi od -99 do 99:
[[[-95, 17], [24, 57], [-43, -72], [-2, 43]],
 [[27, 13], [52, 69], [64, -31], [60, 73]],
 [[1, 61], [-8, 23], [-56, 36], [45, -71]]]
----------
Elementi in podtabele:
p[2][0][1] = 61
p[2][1] = [-8, 23]
p[1] = [[27, 13], [52, 69], [64, -31], [60, 73]]
```

```
p = [[[-95, 17], [24, 57], [-43, -72], [-2, 43]],
     [[27, 13], [52, 69], [64, -31], [60, 73]],
     [[1, 61], [-8, 23], [-56, 36], [45, -71]]]
```

Opomba: Zavoljo večje preglednosti smo izpis tridimenzionalnih p-tabel umetno prelomili. Metoda `toString` naj tovrstnih prelomov ne vstavlja.

Namig

Bi lahko p-tabelo simulirali s pomočjo enodimenzionalne tabele?

8 Tabele objektov

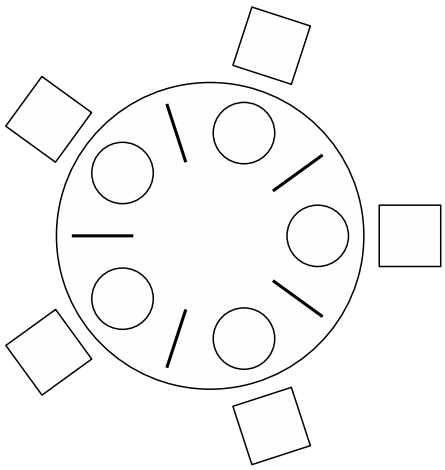
Splošna navodila

Pri vseh nalogah v tem sklopu lahko svoje rešitve preverite z množico testnih razredov in pripadajočih izhodnih datotek. Pri nalogah 8.2–8.4 izhajajte iz razredov, ki jih boste našli v pripadajočih mapah.

8.1 Obedujoči filozofi

Naloga

Skupina kitajskih filozofov sedi za okroglo mizo. Vsak filozof ima svoj krožnik s hrano, po dva soseda pa si delita jedilno paličico:



Filozofi večino časa razmišljajo, občasno pa kdo od njih postane lačen in želi pričeti jesti. Filozof za to opravilo potrebuje obe paličici, levo in desno. Paličici lahko vzame samo, če sta prosti — torej, če njegov levi sosed ne drži svoje desne paličice in če njegov desni sosed ne drži svoje leve paličice. Ko filozof prične jesti, prime obe paličici in ju drži, dokler

ne konča s prehranjevanjem. V tem času seveda preprečuje svojemu levemu in desnemu sosedu, da bi pričela jesti. Ko filozof preneha jesti, odloži (in s tem sprosti) obe paličici.[1]

Za predstavitev posameznih filozofov napišite razred `Filozof` s sledečimi konstruktorji in metodami:

- `public Filozof()`:

  Ustvari objekt, ki predstavlja filozofa za mizo. Filozof ob svoji »stvaritvi« ne drži niti leve niti desne paličice.

- `public void nastaviSoseda(Filozof levi, Filozof desni)`:

  Nastavi oba soseda filozofa `this`.

- `public int kolikoPalicicDrzi()`:

  Vrne število paličic, ki jih drži filozof `this`. To število je lahko samo 0, 1 ali 2.

- `public void primiLevo()`:

  Simulira dogodek, ko filozof `this` prime paličico na svoji levi. To lahko stori samo v primeru, če je prosta (če je ne drži njegov levi sosed). Če filozof `this` svojo levo paličico drži že pred klicem metode, se ne zgodi nič.

- `public void primiDesno()`:

  Simulira dogodek, ko filozof `this` prime paličico na svoji desni. To lahko stori samo v primeru, če je prosta (če je ne drži njegov desni sosed). Če filozof `this` svojo desno paličico drži že pred klicem metode, se ne zgodi nič.

- `public void izpustiLevo()`:

  Simulira dogodek, ko filozof `this` izpusti paličico na svoji levi. Če filozof `this` svoje leve paličice že pred klicem metode ne drži, se ne zgodi nič.

- `public void izpustiDesno()`:

  Simulira dogodek, ko filozof `this` izpusti paličico na svoji desni. Če filozof `this` svoje desne paličice že pred klicem metode ne drži, se ne zgodi nič.

- `public int steviloFilozofov()`:

  Ob predpostavki, da je filozof `this` eden od filozofov za mizo, vrne število filozofov za mizo. Lahko predpostavite, da so relacije sosednosti med filozofi za mizo pravilno vzpostavljene. Če za mizo sedi en sam filozof, je sam svoj levi in desni sosed.

  Namig: Pričnite s filozofom `this` in potujte po verigi desnih (ali levih) sosedov, dokler ne pridete spet do filozofa `this`.

Pri vseh metodah razen `nastaviSoseda` lahko predpostavite, da ima filozof `this` že pravilno nastavljenega levega in desnega soseda. To pomeni, da vam ni treba preverjati, ali levi oz. desni sosed sploh obstaja.

[1] Gre za znan problem iz teorije operacijskih sistemov. Filozofi predstavljajo procese, ki tečejo v računalniku, paličice pa vire (npr. datoteke, naprave, procesorski čas, ...), ki jih procesi potrebujejo. Problem obedujočih filozofov tako ilustrira borbo procesov za omejene vire.

8.2 Krožki

Naloga

Učenci obiskujejo različne krožke. Člani krožkov se sestajajo enkrat tedensko po dve polni uri. Vsak krožek ima svoj termin izvajanja, ki je določen z dnevom v tednu in uro pričetka. Učence predstavimo z objekti tipa `Ucenec`, krožke pa z objekti tipa `Krozek`:

```
public class Ucenec {
    private String ip;          // ime in priimek učenca
    private int razred;         // šolski razred, ki ga obiskuje učenec
    private Krozek[] krozki;    // krožki, v katere je učenec včlanjen

    public Ucenec(String ip, int razred) {
        this.ip = ip;
        this.razred = razred;
    }

    public void nastaviKrozke(Krozek[] krozki) {
        this.krozki = krozki;
    }
}

public class Krozek {
    private String dejavnost;   // dejavnost krožka, npr. šah
    private int dan;            // dan, ko se krožek izvaja (1: poned., 2: torek, ...)
    private int zacUra;         // ura, ko se krožek prične
    private Ucenec[] clani;     // člani krožka

    public Krozek(String dejavnost, int dan, int zacUra) {
        this.dejavnost = dejavnost;
        this.dan = dan;
        this.zacUra = zacUra;
    }

    public void nastaviClane(Ucenec[] clani) {
        this.clani = clani;
    }
}
```

Razreda `Ucenec` in `Krozek`, ki ju boste našli v mapi z izhodiščnimi datotekami za to nalogo, dopolnite s sledečimi metodami:

- `public String toString()` v razredu `Ucenec`:

  Vrne niz s podatki o učencu `this`. Niz naj ima takšno obliko:

  ip`,␣`*razred*`.␣razred`

  Na primer:

  `Mojca Jamnik,␣6.␣razred`

- `public boolean imaCasZa(Krozek k)` v razredu `Ucenec`:

Vrne `true` natanko v primeru, če se termin krožka `k` ne prekriva z nobenim od krožkov, ki jih učenec `this` že obiskuje. Ne pozabite, da vsak krožek traja dve uri!

- `public int steviloSosolcevVKrozku(Krozek k)` v razredu `Ucenec`:

  Vrne število članov krožka `k`, ki hodijo v isti razred kot učenec `this`. Če je učenec `this` tudi član krožka `k`, ga seveda ne smete šteti poleg.

- `public int steviloRazlicnihDejavnosti()` v razredu `Krozek`:

  Vrne skupno število različnih dejavnosti, ki jih obiskujejo člani krožka `this`. Upoštevajte možnost, da imata dva različna krožka (dva različna objekta tipa `Krozek`) isto dejavnost.

  Za primer vzemimo, da sta v krožek `this` včlanjena Ana in Bojan. Ana obiskuje šahovski, rokodelski in dramski krožek, Bojan pa šahovski, dramski in planinski krožek. Člana krožka `this` tako obiskujeta štiri različne dejavnosti: šah, rokodelstvo, dramo in planinstvo.

8.3 Dopolnitve naloge *Prijatelji*

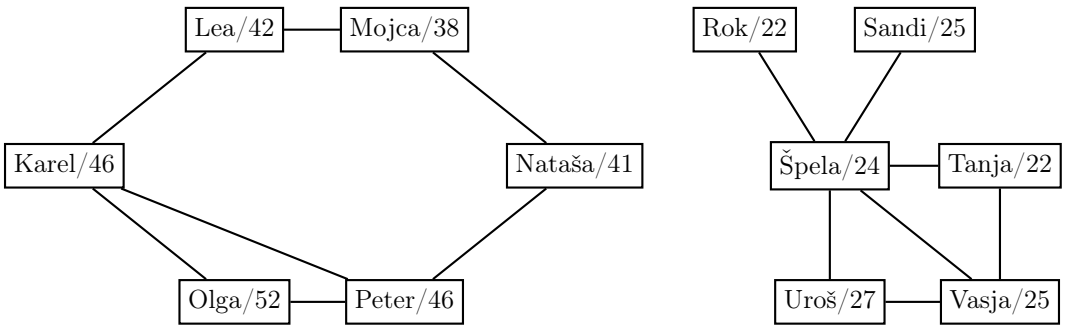
Naloga

Razred `Oseba`, ki smo ga napisali na vajah in ki ga najdete tudi v mapi z izhodiščnimi datotekami za to nalogo, dopolnite s sledečimi metodami:

- `public int steviloPrijateljevPrijateljev()`:

  Vrne število prijateljev prijateljev osebe `this`, ki niso obenem neposredni prijatelji osebe `this`. Tudi osebe `this` ne štejemo med prijatelje prijateljev osebe `this`.

  Slika 8.1 prikazuje primer množice oseb (za vsako osebo je podano njeno ime in njena starost) in prijateljstev med njimi. Osebi sta na sliki povezani natanko tedaj, ko sta prijatelja. V tem primeru so vsa prijateljstva vzajemna. Karel ima dva prijatelja prijateljev: Mojco (preko Lee) in Natašo (preko Petra). Peter je Karlov neposredni prijatelj, zato ga ne štejemo poleg. Uroš ima tri prijatelje prijateljev: Roka, Sandija in Tanjo.



Slika 8.1: *Primer vzajemnih prijateljstev.*

- `public Oseba[] prijateljiPoStarosti()`:

  Vrne novo tabelo, v kateri so prijatelji osebe `this` padajoče urejeni po starosti. Če je več prijateljev enako starih, naj bodo ti v tabeli urejeni po naraščajočem abecednem

vrstnem redu glede na »IP-je« (imena in priimke). Za primerjanje nizov po abecedi uporabite metodo `compareTo` iz razreda `String`.

V primeru na sliki 8.1 bi metoda za osebo Špela vrnila tabelo, v kateri bi bili v tem vrstnem nanizane sledeče osebe: Uroš, Sandi, Vasja, Rok in Tanja.

- (⋆) `public static boolean[][] povezanost(Oseba[] osebe)`:

  Definirajmo relacijo *povezanosti* takole: osebi A in B sta povezani, če velja $A = B$ ali pa če obstaja oseba C, tako da je A prijatelj osebe C ali C prijatelj osebe A, osebi C in B pa sta povezani. (Učeno pravimo, da je relacija povezanosti *refleksivno-simetrično-tranzitivna ovojnica* relacije prijateljstva.)

  Metoda `povezanost` naj vrne matriko velikosti $n \times n$ (kjer je n dolžina tabele `osebe`), v kateri ima element (i, j) vrednost `true` natanko v primeru, če sta osebi i in j povezani.

  V primeru na sliki 8.1 bi metoda `povezanost` za podano tabelo {Karel, Lea, Mojca, Nataša, Olga, Peter, Rok, Sandi, Špela, Tanja, Uroš, Vasja} vrnila sledečo matriko ($1 \rightarrow$ `true`, $0 \rightarrow$ `false`):

```
1 1 1 1 1 1 0 0 0 0 0 0
1 1 1 1 1 1 0 0 0 0 0 0
1 1 1 1 1 1 0 0 0 0 0 0
1 1 1 1 1 1 0 0 0 0 0 0
1 1 1 1 1 1 0 0 0 0 0 0
1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1
0 0 0 0 0 0 1 1 1 1 1 1
0 0 0 0 0 0 1 1 1 1 1 1
0 0 0 0 0 0 1 1 1 1 1 1
0 0 0 0 0 0 1 1 1 1 1 1
0 0 0 0 0 0 1 1 1 1 1 1
```

8.4 Graf (⋆)

Naloga

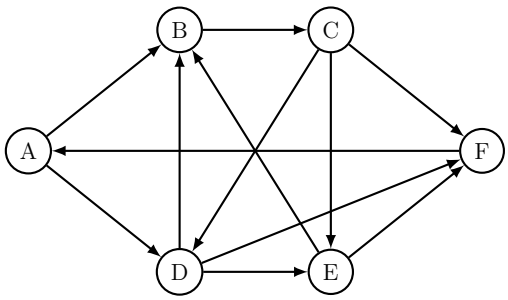
V tej nalogi bomo namesto tabel uporabljali fleksibilnejšo javansko podatkovno strukturo: *seznam*. Seznami so prav tako kot tabele uporabni za shranjevanje zaporedij elementov istega tipa, vendar pa se v nasprotju s tabelami samodejno »raztezajo«. Seznami so objekti tipa `List` iz paketa `java.util`. Delo s seznami je enostavno:

```
// na začetku datoteke
import java.util.*;

// ustvari prazen seznam z elementi tipa String
List<String> seznam = new ArrayList<>();

// dodaj tri elemente na konec seznama
seznam.add("Anja");
```

```
seznam.add("Boris");
seznam.add("Dejan");

// dodaj element na pozicijo 2 (pred elementom Dejan)
seznam.add(2, "Cvetka");

// izpiši vsebino seznama
System.out.println(seznam.toString());
// ali pa kar System.out.println(seznam);

// odstrani element z indeksom 1 (Boris)
seznam.remove(1);

// pridobi element z indeksom 2 (Dejan) in ga izpiši
String element = seznam.get(2);
System.out.println(element);

// Anja, Cvetka, Dejan
System.out.println(seznam);

// izpiši število elementov seznama (3)
System.out.println(seznam.size());

// izprazni seznam
seznam.clear();

// (prazno)
System.out.println(seznam);
```

Več o vmesniku `List` ter njegovih implementacijah `ArrayList` in `LinkedList` boste našli v javanski dokumentaciji.

V tej nalogi bomo sezname potrebovali pri realizaciji podatkovne strukture *graf*. Graf je množica *vozlišč* in *povezav* med njimi. Obstaja veliko različnih definicij grafa. Grafi, s kakršnimi se bomo ukvarjali v tej nalogi, so *enostavni* in *usmerjeni*. V usmerjenem grafu so vse povezave usmerjene, kar pomeni, da ima vsaka povezava svoje izvorno in ponorno vozlišče. Enostaven graf ne sme vsebovati zank (povezav $u \to u$) in vzporednih povezav (npr. več povezav $u \to v$). Primer enostavnega usmerjenega grafa prikazuje slika 8.2.



Slika 8.2: *Enostaven usmerjen graf.*

Enostaven usmerjen graf lahko definiramo s sledečo programsko kodo:

```
class Graf {
    private List<Vozlisce> vozlisca;   // seznam vozlišč grafa
}

class Vozlisce {
    private String oznaka;             // oznaka vozlišča
    private List<Povezava> povezave;   // seznam povezav, ki izvirajo v vozlišču this
}

class Povezava {
    private Vozlisce izvor;   // izvorno vozlišče povezave
    private Vozlisce ponor;   // ponorno vozlišče povezave
}
```

V objektu tipa `Graf` nam ni treba hraniti seznama povezav, saj so povezave dostopne preko posameznih vozlišč.

Definirajmo še razred, čigar objekt predstavlja neko usmerjeno pot po grafu:

```
class Pot {
    private List<Vozlisce> vozlisca;   // zaporedje vozlišč na poti
}
```

V nadaljevanju bomo definirali zgolj funkcionalnost, ki jo mora uporabniku ponujati razred `Graf`. Razrede `Vozlisce`, `Povezava` in `Pot` po potrebi opremite z (dodatnimi) konstruktorji in metodami.

Razred `Graf` mora ponujati sledeče konstruktorje in metode:

- `public Graf()`

  Ustvari prazen graf (graf brez vozlišč in povezav).

- `public void dodajVozlisce(Vozlisce vozlisce)`

  Doda podano vozlišče v graf.

- `public static void dodajPovezavo(Vozlisce izvor, Vozlisce ponor)`

  Ustvari povezavo med podanim izvornim in ponornim vozliščem. (Kam boste shranili povezavo? Zakaj je ta metoda lahko statična?)

- `public String toString()`

  Vrne niz, ki podaja zgradbo grafa. Pri gradnji niza se zgledujte po primeru, ki je prikazan v nadaljevanju.

- `public static Pot pot(Vozlisce zacetek, Vozlisce konec)`

  Poišče neko aciklično pot (katerokoli) od vozlišča `zacetek` do vozlišča `konec`. Če med vozliščema ni nobene poti, naj metoda vrne `null`.

  Upoštevajte, da graf lahko vsebuje cikle! Kako boste zagotovili, da se metoda ne bo ujela v katerem od njih?

- `public static List<Pot> vsePoti(Vozlisce zacetek, Vozlisce konec)`

  Poišče vse aciklične poti od vozlišča `zacetek` do vozlišča `konec` in rezultat vrne v obliki seznama poti.

Primer

Sledeča koda ustvari graf s slike 8.2, izpiše njegovo vsebino ter poišče in izpiše neko pot od vozlišča D do vozlišča A in vse poti od vozlišča A do vozlišča F:

```
public class Test {

    public static void main(String[] args) {
        Graf graf = new Graf();

        String[] oznake = {"A", "B", "C", "D", "E", "F"};
        int[][] vezi = {
            {0, 1}, {0, 3}, {1, 2}, {2, 3}, {2, 4}, {2, 5},
            {3, 1}, {3, 4}, {3, 5}, {4, 1}, {4, 5}, {5, 0}
        };

        Vozlisce[] vozlisca = new Vozlisce[oznake.length];
        for (int i = 0;  i < oznake.length;  i++) {
            vozlisca[i] = new Vozlisce(oznake[i]);
            graf.dodajVozlisce(vozlisca[i]);
        }
        int stPovezav = vezi.length;
        for (int i = 0;  i < stPovezav;  i++) {
            Graf.dodajPovezavo(vozlisca[vezi[i][0]], vozlisca[vezi[i][1]]);
        }

        System.out.println("Graf:");
        System.out.println(graf.toString());

        System.out.printf("Ena od poti od vozlišča %s do vozlišča %s:%n",
                vozlisca[3].toString(), vozlisca[0].toString());
        System.out.println(Graf.pot(vozlisca[3], vozlisca[0]));
        System.out.println();

        System.out.printf("Vse poti od vozlišča %s do vozlišča %s:%n",
                vozlisca[0].toString(), vozlisca[5].toString());
        System.out.println(Graf.vsePoti(vozlisca[0], vozlisca[5]));
        System.out.println();
    }
}
```

Gornja koda bi morala ustvariti sledeči izpis (pri klicu metode `pot` lahko seveda dobimo tudi drugačen izpis):

```
Graf:
A --> [B, D]
B --> [C]
C --> [D, E, F]
D --> [B, E, F]
E --> [B, F]
F --> [A]
```

```
Ena od poti od vozlišča D do vozlišča A:
[D, B, C, E, F, A]

Vse poti od vozlišča A do vozlišča F:
[[A, B, C, D, E, F], [A, B, C, D, F], [A, B, C, E, F],
 [A, B, C, F], [A, D, B, C, E, F], [A, D, B, C, F],
 [A, D, E, B, C, F], [A, D, E, F], [A, D, F]]
```

Opomba: V izpisu seznama poti od vozlišča A do vozlišča F so prelomi vrstic ročno vstavljeni.

9 Dedovanje

Splošna navodila

Pri nalogah 9.1 in 9.2 lahko svoje rešitve preverite z množico vhodnih datotek in pripadajočih izhodnih datotek, pri nalogah 9.3 in 9.4 pa z množico testnih razredov in pripadajočih izhodnih datotek.

9.1 Poštne pošiljke

Naloga

Na poštnem uradu razpošiljajo tri vrste poštnih pošiljk: *navadna pisma*, *priporočena pisma* in *telegrame*. Za vsako pošiljko sta podana naslovnik in vsebina. Za navadna pisma je poleg tega podana še razdalja do naslovnika, za priporočena pisma pa (poleg naslovnika, vsebine in razdalje) še pošiljatelj. Za telegram sta podana samo naslovnik in vsebina. Naslovniki, pošiljatelji in vsebine pošiljk so podani kot nizi, sestavljeni iz črk angleške abecede, števk in podčrtajev.

Cena oddaje navadnega pisma je odvisna zgolj od razdalje: za razdaljo od 0 do vključno $(R-1)$ dolžinskih enot znaša cena Z denarnih enot, za razdaljo od R do vključno $(2R-1)$ znaša cena $(Z + D)$, za razdaljo od $2R$ do vključno $(3R - 1)$ je cena enaka $(Z + 2D)$ itd. Cena oddaje priporočenega pisma se izračuna tako, da se cena oddaje navadnega pisma pomnoži s faktorjem P. Cena telegrama je enaka cT, kjer je c število črk v vsebini telegrama (števke in podčrtaji ne štejejo).

Napišite program, ki prebere zaporedje poštnih pošiljk in ukaz (celo število med 1 in vključno 3), nato pa na podlagi ukaza proizvede ustrezni izpis:

- Če je ukaz enak 1, naj se izpišejo podatki o vseh pošiljkah. Za vsako pošiljko mora izpis vsebovati njeno vrsto, naslovnika, vsebino, morebitne dodatne podatke (odvisno od vrste pošiljke) in ceno.
- Če je ukaz enak 2, naj se izpišejo podatki o najdražji pošiljki (o prvi od njih, če jih je več).

- Če je ukaz enak 3, naj se za vsako priporočeno pismo izpišejo podatki o njegovi povratnici. Povratnica priporočenega pisma je priporočeno pismo, pri katerem je razdalja enaka kot pri izhodiščnem pismu, naslovnik in pošiljatelj sta med seboj zamenjana, vsebina pa je niz `povratnica`.

Vrstni red izpisa pošiljk na izhodu mora biti enak vrstnemu redu pošiljk na vhodu.

Vhod

V prvi vrstici vhoda je v tem vrstnem redu nanizanih pet celih števil z intervala $[1, 1000]$: Z, R, D, P in T. V drugi vrstici je podano celo število $n \in [1, 100]$. Sledi n vrstic s podatki o pošiljkah. V vsaki vrstici je najprej podana vrsta pošiljke (`navadnoPismo`, `priporocenoPismo` oziroma `telegram`), nato pa sledita naslovnik in vsebina. Pri navadnem pismu sledi še razdalja, pri priporočenem pa razdalja in pošiljatelj. V zadnji vrstici vhoda je zapisano število $U \in \{1, 2, 3\}$, ki predstavlja ukaz.

Vsi nizi vsebujejo od 1 do 100 znakov. Vse razdalje so cela števila z intervala $[0, 1000]$. Vsi podatki znotraj iste vrstice so med seboj ločeni s po enim presledkom.

Izhod

Na izhodu izpišite podatke, ki jih zahteva ukaz. Zgledujte se po sledečih primerih. Zadnji podatek v vsaki vrstici je cena pošiljke.

Primer 1

Testni vhod:

```
30 50 10 3 9
7
telegram Ana_Antolinc vsebina_telegrama_1
priporocenoPismo Branko_Bizjak vsebina_pp_1 150 Cvetka_Cevc
navadnoPismo Danica_Dobravc vsebina_np_1 75
telegram Eva_Erker zelo_dolga_VSEBINA_s_5t3v1lk4m1
navadnoPismo Franci_Furman vsebina_np_2 49
navadnoPismo Gorazd_Gaber vsebina_np_2 50
priporocenoPismo Hinko_Hojc vsebina_pp_2 50 Iva_Intihar
1
```

Pričakovani izhod:

```
T | Ana_Antolinc | vsebina_telegrama_1 | 144
PP | Branko_Bizjak | vsebina_pp_1 | 150 | Cvetka_Cevc | 180
NP | Danica_Dobravc | vsebina_np_1 | 75 | 40
T | Eva_Erker | zelo_dolga_VSEBINA_s_5t3v1lk4m1 | 198
NP | Franci_Furman | vsebina_np_2 | 49 | 30
NP | Gorazd_Gaber | vsebina_np_2 | 50 | 40
PP | Hinko_Hojc | vsebina_pp_2 | 50 | Iva_Intihar | 120
```

Primer 2

Testni vhod:

```
30 50 10 3 9
7
telegram Ana_Antolinc vsebina_telegrama_1
priporocenoPismo Branko_Bizjak vsebina_pp_1 150 Cvetka_Cevc
navadnoPismo Danica_Dobravc vsebina_np_1 75
telegram Eva_Erker zelo_dolga_VSEBINA_s_5t3v1lk4m1
navadnoPismo Franci_Furman vsebina_np_2 49
navadnoPismo Gorazd_Gaber vsebina_np_2 50
priporocenoPismo Hinko_Hojc vsebina_pp_2 50 Iva_Intihar
2
```

Pričakovani izhod:

```
T | Eva_Erker | zelo_dolga_VSEBINA_s_5t3v1lk4m1 | 198
```

Primer 3

Testni vhod:

```
30 50 10 3 9
7
telegram Ana_Antolinc vsebina_telegrama_1
priporocenoPismo Branko_Bizjak vsebina_pp_1 150 Cvetka_Cevc
navadnoPismo Danica_Dobravc vsebina_np_1 75
telegram Eva_Erker zelo_dolga_VSEBINA_s_5t3v1lk4m1
navadnoPismo Franci_Furman vsebina_np_2 49
navadnoPismo Gorazd_Gaber vsebina_np_2 50
priporocenoPismo Hinko_Hojc vsebina_pp_2 50 Iva_Intihar
3
```

Pričakovani izhod:

```
PP | Cvetka_Cevc | povratnica | 150 | Branko_Bizjak | 180
PP | Iva_Intihar | povratnica | 50 | Hinko_Hojc | 120
```

Napotki

Nize beremo s klicem `sc.next()`, pri čemer je `sc` objekt razreda `Scanner`. Če je `s` niz (objekt tipa `String`), potem njegovo dolžino (število znakov) pridobimo s klicem `s.length()`, znak z indeksom `i` pa s klicem `s.charAt(i)`. Na vprašanje, ali je podani znak `c` (spremenljivka tipa `char`) črka, odgovorimo s klicem `Character.isLetter(c)`.

9.2 Geometrijska telesa

Naloga

Na vhodu so zapisani osnovni podatki o različnih geometrijskih telesih: kvadrih, kockah in kroglah. Kvader je določen s tremi stranicami, kocka z eno samo, krogla pa s polmerom. Napišite program, ki prebere podatke o telesih ter izpiše njihove prostornine, površine in mreže. Prostornine in površine naj bodo zaokrožene na najbližje celo število. Pri krogli naj se izpišeta samo prostornina in površina.

Na primer, mreža kvadra s stranicami $a = 2$, $b = 3$ in $c = 4$ naj se nariše tako:

```
        * * *
        * * *
        * * *
        * * *
+ + + + o o o + + + +
+ + + + o o o + + + +
        * * *
        * * *
        * * *
        * * *
        o o o
        o o o
```

oziroma (s presledki)

```
␣␣␣␣␣␣␣␣*␣*␣*
␣␣␣␣␣␣␣␣*␣*␣*
␣␣␣␣␣␣␣␣*␣*␣*
␣␣␣␣␣␣␣␣*␣*␣*
+␣+␣+␣+␣o␣o␣o␣+␣+␣+␣+
+␣+␣+␣+␣o␣o␣o␣+␣+␣+␣+
␣␣␣␣␣␣␣␣*␣*␣*
␣␣␣␣␣␣␣␣*␣*␣*
␣␣␣␣␣␣␣␣*␣*␣*
␣␣␣␣␣␣␣␣*␣*␣*
␣␣␣␣␣␣␣␣o␣o␣o
␣␣␣␣␣␣␣␣o␣o␣o
```

V izpisu morajo biti telesa urejena po *padajočih prostorninah*. Urejanje mora biti stabilno: telesa z enakimi prostorninami morajo biti na izhodu izpisana v istem medsebojnem vrstnem redu, kot so podana na vhodu.

Vhod

V prvi vrstici vhoda je podano celo število $n \in [1, 100]$, nato pa sledi n vrstic s podatki o posameznih telesih. Na začetku vsake vrstice je zapisano število 1 (to število predstavlja kvader), 2 (kocka) oziroma 3 (krogla), nato pa sledijo dolžine stranic (pri kvadru), dolžina

stranice (pri kocki) oziroma polmer (pri krogli). Vsi navedeni podatki so cela števila z intervala $[1, 100]$.

Izhod

Pri izpisu na izhod se zgledujte po primeru v nadaljevanju. Ne izpisujte odvečnih presledkov ali praznih vrstic!

Primer

Testni vhod:

```
6
2 4
1 4 2 8
3 4
2 3
3 2
1 8 4 2
```

Pričakovani izhod:

```
krogla
r = 4
V = 268
P = 201
======
kocka
a = 4
V = 64
P = 96
        * * * *
        * * * *
        * * * *
        * * * *
+ + + + o o o o + + + +
+ + + + o o o o + + + +
+ + + + o o o o + + + +
+ + + + o o o o + + + +
        * * * *
        * * * *
        * * * *
        * * * *
        o o o o
        o o o o
        o o o o
        o o o o
======
kvader
a = 4
b = 2
```

```
c = 8
V = 64
P = 112
                * *
                * *
                * *
                * *
                * *
                * *
                * *
                * *
+ + + + + + + + o o + + + + + + + + +
+ + + + + + + + o o + + + + + + + + +
+ + + + + + + + o o + + + + + + + + +
+ + + + + + + + o o + + + + + + + + +
                * *
                * *
                * *
                * *
                * *
                * *
                * *
                * *
                o o
                o o
                o o
                o o
======
kvader
a = 8
b = 4
c = 2
V = 64
P = 112
    * * * *
    * * * *
+ + o o o o + +
+ + o o o o + +
+ + o o o o + +
+ + o o o o + +
+ + o o o o + +
+ + o o o o + +
+ + o o o o + +
+ + o o o o + +
    * * * *
    * * * *
    o o o o
    o o o o
    o o o o
    o o o o
    o o o o
```

```
    o o o o
    o o o o
    o o o o
======
krogla
r = 2
V = 34
P = 50
======
kocka
a = 3
V = 27
P = 54
      * * *
      * * *
      * * *
+ + + o o o + + +
+ + + o o o + + +
+ + + o o o + + +
      * * *
      * * *
      * * *
      o o o
      o o o
      o o o
======
```

9.3 Seznam (⋆)

Naloga

Seznam je podatkovna struktura, ki predstavlja zaporedje elementov. Obstajata dve vrsti seznamov:

Prazen seznam: To je seznam, ki ne vsebuje nobenih elementov.

Neprazen seznam: To je seznam, ki je sestavljen iz *glave* in *repa*. Glava je prvi element seznama (v tej nalogi bo to vedno neko celo število), rep pa je seznam (prazen ali neprazen). Seznam z elementi 3, 5 in 7 je sestavljen iz glave 3 in repa, ki je seznam, sestavljen iz glave 5 in repa, ki je seznam, sestavljen iz glave 7 in praznega seznama. Če prazen seznam zapišemo kot `[]`, neprazen pa kot `[`*glava* `|` *rep*`]`, bi lahko seznam z elementi 3, 5 in 7 zapisali kot `[3 | [5 | [7 | []]]]`.

To definicijo lahko prevedemo v hierarhijo razredov, ki jo sestavljajo abstrakten razred `Seznam` ter njegova podrazreda `Prazen` in `Neprazen`. Realizirajte vse tri razrede. Razred `Prazen` mora vsebovati javno dostopen konstruktor brez argumentov, ki izdela objekt, ki predstavlja prazen seznam. Razred `Neprazen` pa mora ponujati konstruktor z glavo

```
public Neprazen(int glava, Seznam rep)
```

ki izdela objekt, ki predstavlja seznam s podano glavo in repom. Na primer, objekt, ki predstavlja seznam z elementi 3, 5 in 7, bi s pomočjo navedenih konstruktorjev ustvarili takole:

```
Seznam s = new Neprazen(3, new Neprazen(5, new Neprazen(7, new Prazen())));
```

V nadaljevanju bomo podali množico javno dostopnih metod, ki jih mora ponujati razred `Seznam`. Premislite, katere od teh metod naj bodo v razredu `Seznam` abstraktne (in definirane v obeh podrazredih), katere pa je možno definirati že v razredu `Seznam`. Nobena metoda ne spremeni seznama `this` in nobena ni daljša od nekaj vrstic.

Razred `Seznam` mora ponujati sledeče metode:

- `public int glava()`:

  Če je seznam `this` neprazen, vrne njegovo glavo, sicer pa sproži izjemo:

  ```
  throw new java.util.NoSuchElementException("Prazen seznam nima glave!");
  ```

- `public Seznam rep()`:

  Če je seznam `this` neprazen, vrne njegov rep, sicer pa sproži izjemo.

- `public boolean jePrazen()`:

  Vrne `true` natanko v primeru, če je seznam prazen.

- `public Seznam dodajZ(int element)`:

  Vrne nov seznam, ki ima v glavi podani element, njegov rep pa je seznam `this`.

- `public Seznam dodajK(int element)`:

  Vrne nov seznam, ki je enak seznamu `this`, le da ima na koncu dodan še podani element.

  Pri praznem seznamu ta metoda deluje enako kot metoda `dodajZ`, pri nepraznem seznamu pa metoda `dodajK` izdela in vrne nov neprazen seznam, čigar glava je enaka glavi seznama `this`, rep pa je enak seznamu, ki ga dobimo, če na konec repa seznama `this` dodamo podani element.

- `public Seznam dodajU(int element)`:

  Ob predpostavki, da je seznam `this` naraščajoče urejen, ta metoda vrne nov seznam, ki je enak seznamu `this`, le da ima podani element vstavljen na mestu, kamor spada po velikosti. Na primer, če seznam `s` po vrsti vsebuje elemente 3, 5 in 7, potem klic `s.dodajU(6)` vrne seznam [3, 5, 6, 7], klic `s.dodajU(2)` pa seznam [2, 3, 5, 7].

  Pri nepraznem seznamu upoštevajte dve možnosti: (1) podani element je manjši ali enak glavi seznama `this`; (2) podani element je večji od glave.

- `public boolean vsebuje(int element)`:

  Vrne `true` natanko v primeru, če seznam vsebuje podani element. Prazen seznam elementa gotovo ne vsebuje, pri nepraznem seznamu pa moramo najprej preveriti glavo, če ta ni enaka iskanemu elementu, pa preiščemo še rep.

- `public Seznam odstrani(int element)`:

  Vrne nov seznam, ki je enak seznamu `this`, le da je v njem odstranjen prvi element, ki je enak podanemu elementu. Na primer, če je seznam `s` enak [7, 3, 6, 3, 2, 3],

potem klic `s.odstrani(3)` vrne seznam [7, 6, 3, 2, 3]. Če seznam `this` podanega elmenta ne vsebuje, naj bo rezultat metode enak seznamu `this`.

- `public Seznam uredi()`:

  Vrne naraščajoče urejeno kopijo seznama `this`. Na primer, klic metode nad seznamom [7, 6, 3, 2, 3] bi vrnil seznam [2, 3, 3, 6, 7].

  Namig: pri nepraznem seznamu najprej uredimo rep.

- `public Seznam odstraniDuplikate()`:

  Vrne nov seznam, ki je enak seznamu `this`, le da v njem vsak element nastopa le enkrat. Ohraniti se mora le *zadnja* pojavitev elementa. Na primer, klic metode nad seznamom [7, 3, 6, 3, 2, 3, 7, 2, 9] bi vrnil seznam [6, 3, 7, 2, 9].

- `public String toString()`:

  Vrne niz oblike

  $[a_1,\sqcup a_2,\sqcup \ldots,\sqcup a_n]$

  kjer so a_1, a_2, ..., a_n elementi seznama `this`. Na primer, klic metode nad seznamom [7, 6, 3, 2, 3] bi vrnil sledeči niz:

  ```
  [7, 6, 3, 2, 3]
  ```

  Pri realizaciji metode `toString` vam bo morda koristila pomožna metoda, ki izdela niz brez oklepajev.

9.4 Naravno število (⋆)

Naloga

Cilj te naloge je predstaviti naravna števila (1, 2, 3, ...) in realizirati nekatere operacije nad njimi s pomočjo dedovanja in rekurzije. Ker bi lahko nalogo brez posebnih težav razširili na celotno množico celih števil, se izkaže, da celoštevilskih podatkovnih tipov pravzaprav sploh ne potrebujemo! Ta drzna trditev pa velja le na konceptualni ravni, saj časovna in prostorska učinkovitost operacij, ki jih boste realizirali v tej nalogi, ne bo ravno najboljša. Vendar pa naj tokrat estetske vrednote prevladajo nad materialnimi!

Množico naravnih števil lahko definiramo na sledeči način (Peanovi aksiomi):

- Število 1 je naravno število.
- Naslednik naravnega števila je tudi naravno število.

To definicijo lahko prevedemo v hierarhijo razredov, ki jo sestavljajo abstrakten razred `NaravnoStevilo` ter njegova podrazreda `Ena` in `Naslednik`. Realizirajte vse tri razrede. Razred `Ena` mora vsebovati javno dostopen konstruktor brez argumentov, ki izdela objekt, ki predstavlja naravno število 1. Razred `Naslednik` pa mora ponujati konstruktor z glavo

```
public Naslednik(NaravnoStevilo stevilo)
```

ki izdela objekt, ki predstavlja naslednika podanega naravnega števila. Na primer, objekt, ki predstavlja naravno število 3, bi s pomočjo navedenih konstruktorjev ustvarili takole:

```
NaravnoStevilo tri = new Naslednik(new Naslednik(new Ena()));
```

V nadaljevanju bomo podali množico javno dostopnih metod, ki jih mora ponujati razred `NaravnoStevilo`. Premislite, katere od teh metod naj bodo v razredu `NaravnoStevilo` abstraktne (in definirane v obeh podrazredih), katere pa je možno definirati že v razredu `NaravnoStevilo`. Nobena metoda ne spremeni naravnega števila `this` in nobena ni daljša od nekaj vrstic.

Razred `NaravnoStevilo` mora ponujati sledeče metode:

- `public boolean jeEna()`:

  Vrne `true` natanko v primeru, če objekt `this` predstavlja število 1.

- `public NaravnoStevilo naslednik()`:

  Vrne naravno število, ki je naslednik naravnega števila `this`.

- `public NaravnoStevilo predhodnik()`:

  Vrne naravno število, ki je predhodnik naravnega števila `this`. Upoštevajte, da je predhodnik števila `new Naslednik(`n`)` kar število n. Ker število 1 nima predhodnika, naj metoda zanj vrže izjemo tipa `java.util.NoSuchElementException`:

  ```
  throw new java.util.NoSuchElementException("Enica nima predhodnika!");
  ```

- `public NaravnoStevilo vsota(NaravnoStevilo stevilo)`:

  Vrne naravno število, ki je vsota naravnega števila `this` in števila `stevilo`. Namig: primer $1 + n$ sodi v razred `Ena`, primer $m + n$ za $m > 1$ pa v razred `Naslednik`.

- `public NaravnoStevilo razlika(NaravnoStevilo stevilo)`:

  Vrne naravno število, ki je razlika naravnega števila `this` in števila `stevilo`. Če rezultat razlike ni naravno število, naj metoda sproži izjemo tipa `java.util.NoSuchElementException`.

- `public NaravnoStevilo zmnozek(NaravnoStevilo stevilo)`:

  Vrne naravno število, ki je zmnožek naravnega števila `this` in števila `stevilo`.

- `public String toString()`:

  Če objekt `this` predstavlja število 1, vrne niz `1`, sicer pa vrne niz `s(`t`)`, kjer je t niz, ki ga metoda `toString` vrne za predhodnika števila `this`. Na primer, za število 3 naj metoda vrne niz `s(s(1))`.

- `public int toInt()`:

  Vrne celo število, ki ga predstavlja objekt `this`.

- `public static NaravnoStevilo ustvariIzInt(int n)`:

  Ustvari objekt, ki predstavlja podano pozitivno celo število `n`. To metodo morate seveda definirati v razredu `NaravnoStevilo`.

Primer

Za lažjo predstavo sledi primer testne kode in izpis, ki bi ga morala proizvesti.

Koda:

```
NaravnoStevilo a = new Naslednik(new Naslednik(new Naslednik(new Ena())));
NaravnoStevilo b = NaravnoStevilo.ustvariIzInt(9);

System.out.printf("a: %s [%d]%n", a, a.toInt());
System.out.printf("b: %s [%d]%n", b, b.toInt());

NaravnoStevilo an = a.naslednik();
NaravnoStevilo ap = a.predhodnik();

System.out.printf("a.naslednik(): %s [%d]%n", an, an.toInt());
System.out.printf("a.predhodnik(): %s [%d]%n", ap, ap.toInt());

System.out.println("a + b = " + a.vsota(b).toInt());
System.out.println("b + a = " + b.vsota(a).toInt());

try {
    System.out.println("a - b = " + a.razlika(b).toInt());
} catch (java.util.NoSuchElementException ex) {
    System.out.println("Razlika ne obstaja!");
}
try {
    System.out.println("b - a = " + b.razlika(a).toInt());
} catch (java.util.NoSuchElementException ex) {
    System.out.println("Razlika ne obstaja!");
}

System.out.println("a * b = " + a.zmnozek(b).toInt());
System.out.println("b * a = " + b.zmnozek(a).toInt());
```

Pričakovani izpis:

```
a: s(s(s(1))) [4]
b: s(s(s(s(s(s(s(s(s(1))))))))) [9]
a.naslednik(): s(s(s(s(1)))) [5]
a.predhodnik(): s(s(1)) [3]
a + b = 13
b + a = 13
Razlika ne obstaja!
b - a = 5
a * b = 36
b * a = 36
```

10 Grafika

Splošna navodila

Pri vseh nalogah v tem sklopu lahko svoje rešitve preverite z množico testnih razredov in pripadajočih izhodnih slik. Izhajajte iz razredov, ki jih boste našli v pripadajočih mapah. Dopolnite metodo `narisi` v razredu, ki je naveden pri posamezni nalogi, seveda pa lahko dodate še poljubno število lastnih metod in razredov. Razreda `OsnovaZaRisanje` ne spreminjajte.

10.1 Barvni krog

Naloga

Metodo `narisi` v razredu `BarvniKrog` dopolnite tako, da bo ta ob zagonu narisal krog, sestavljen iz n ($n \geqslant 2$) enako velikih zaporednih krožnih izsekov. Vsak krožni izsek naj bo pobarvan z barvo, ki ima v modelu HSV/HSB (Hue, Saturation, Value/Brightness) komponenti S in V oz. B enaki 1 (maksimalna možna vrednost). Komponenta H naj bo pri prvem krožnem izseku (izseku, ki se prične pri kotu $0°$) enaka 0, pri vsakem naslednjem izseku pa naj bo za $1/n$ večja. Krog naj zavzema celotno krajšo stranico platna, po daljši stranici pa naj bo prikazan na sredini.

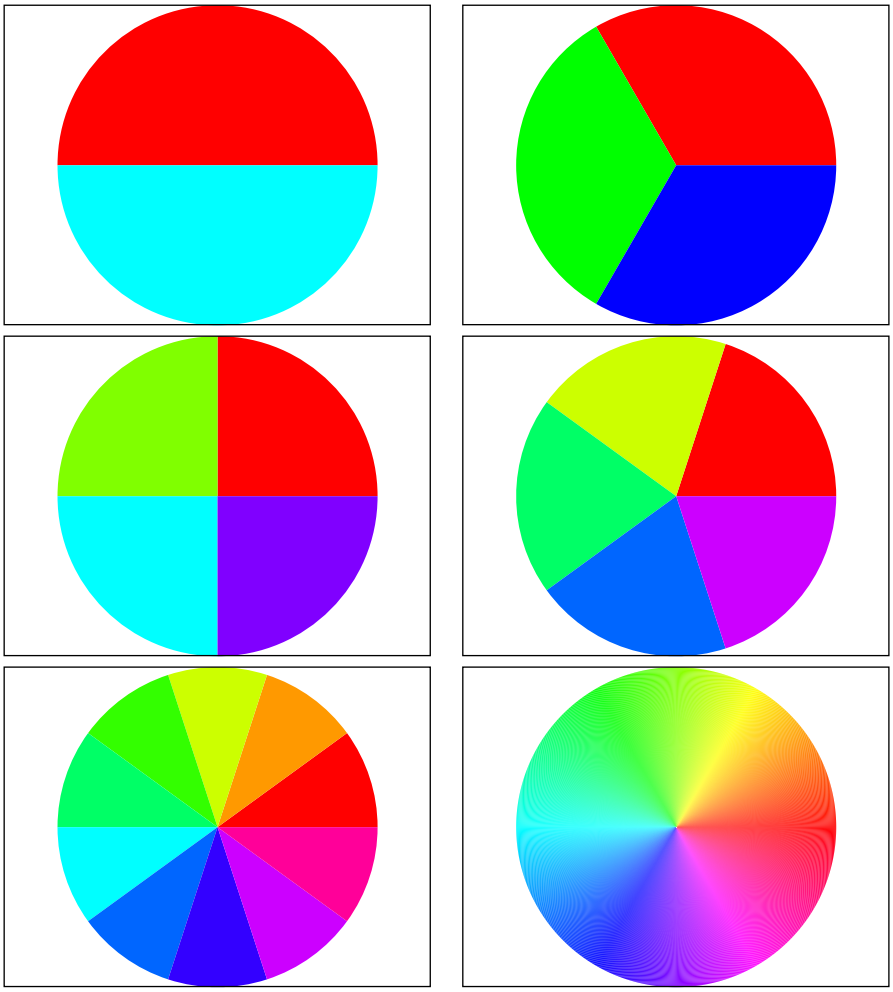
Število n se ob klicu konstruktorja razreda `BarvniKrog` prenese v atribut `stIzsekov`.

Primeri

Slika 10.1 prikazuje vsebino platna za $n = 2$, $n = 3$, $n = 4$, $n = 5$, $n = 10$ in $n = 1000$. Obrob ne rišite, saj predstavljajo zgolj robove platna.

Napotki

- Pomagajte si z metodo `getHSBColor` iz razreda `java.awt.Color`.



Slika 10.1: *Vsebina platna za različne vrednosti n.*

- Metoda `g.fillArc` sprejme samo celoštevilske kote, kar je predvsem pri večjem številu izsekov premalo za natančen izris. Zato izseke raje rišite takole:

```
g.fill(new Arc2D.Double(..., ..., ..., ..., ..., ..., Arc2D.PIE));
```

  Prvih šest parametrov konstruktorja razreda `Arc2D.Double` je enakih kot pri metodi `g.fillArc`, le da so lahko tipa `double`.

10.2 Kvadratna spirala

Naloga

Metodo `narisi` v razredu `Spirala` dopolnite tako, da bo ta ob zagonu narisala kvadratno spiralo, sestavljeno iz n polnih zavojev. Zgledujte se po prikazanih primerih. Število n se ob klicu konstruktorja razreda `Spirala` prenese v atribut `stZavojev`.

Primeri

Slika 10.2 prikazuje vsebino platna za $n = 1$, $n = 2$, $n = 3$ in $n = 4$. Obrobe in kvadratne mreže ne rišite, saj sta namenjeni zgolj za lažjo določitev ključnih mer. Navidezna kvadratna mreža naj zavzema celotno krajšo stranico platna, po daljši stranici pa naj bo prikaza na sredini.

Spiralo narišite z barvo `Color.BLUE`. Uporabite privzeto debelino črt; črte na spodnji sliki so odebeljene zgolj zaradi boljše vidnosti.

10.3 Drevo (⋆)

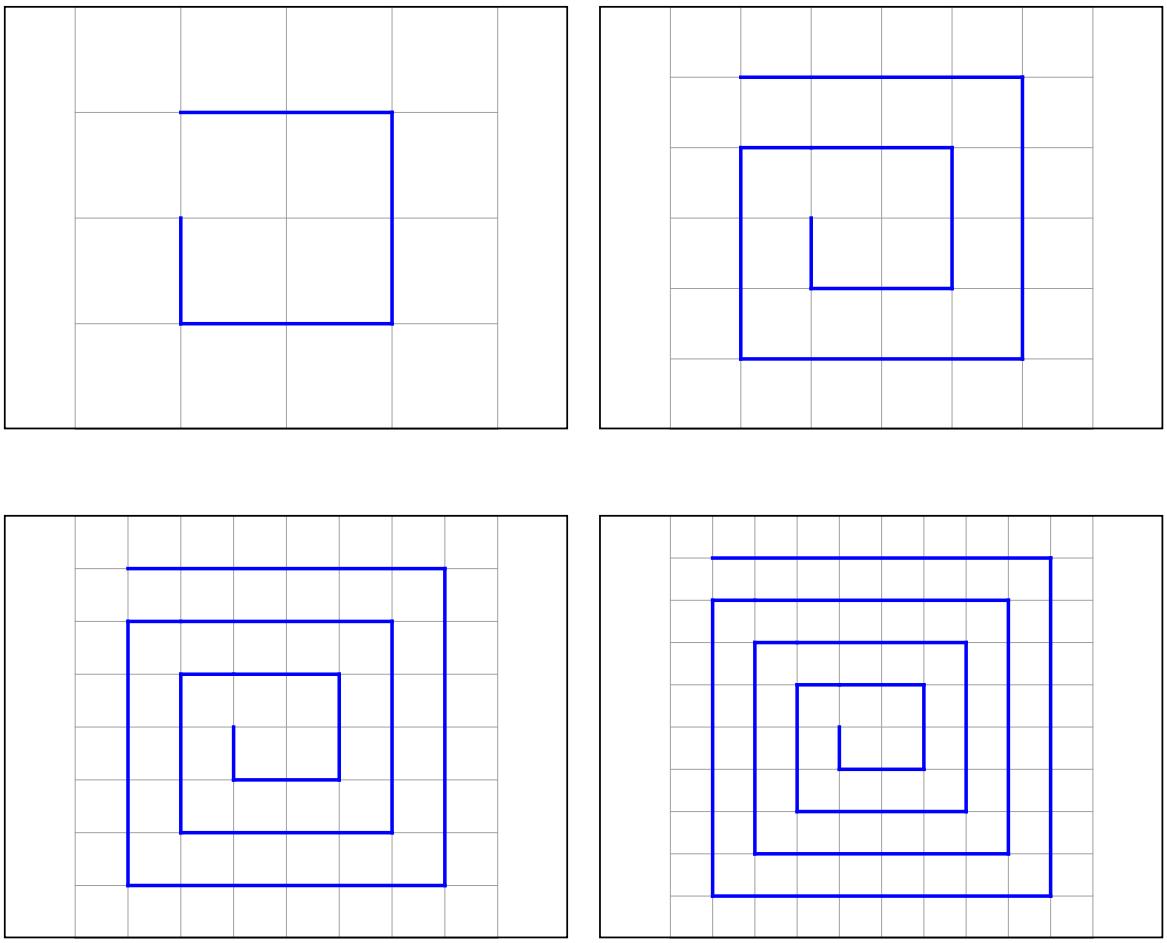
Naloga

Metodo `narisi` v razredu `Drevo` dopolnite tako, da bo ta ob zagonu narisal drevo, predstavljeno s podanim opisom. Opis drevesa je neprazen niz, sestavljen iz ne-negativnih celih števil, ki so med seboj ločena s po enim presledkom. Na začetku opisa je podano število otrok korena drevesa, nato pa so na enak način (rekurzivno) po vrsti opisani posamezni otroci. Na primer, opis

```
4 2 0 0 1 2 0 0 0 3 0 3 0 0 0 0
```

predstavlja drevo, v katerem ima koren 4 otroke. Prvi otrok korena ima dva otroka; oba od njiju sta brez otrok. Drugi otrok korena ima enega otroka, ta pa ima dva otroka; oba od njiju sta brez otrok. Tretji otrok korena nima nobenega otroka. Četrti otrok korena ima tri otroke: prvi je brez otrok, drugi ima tri otroke (vsi so brez otrok), tretji pa je prav tako brez otrok. Za lažje razumevanje bomo dele opisa, ki pripadajo posameznim poddrevesom korena, označili z različnimi barvami:

```
4 2 0 0 1 2 0 0 0 3 0 3 0 0 0 0
```



Slika 10.2: *Vsebina platna za različne vrednosti n.*

Drevo naj po višini in širini zavzema celotno površino platna. Vsakemu vozlišču naj pripada (navidezna) pravokotna celica. Vse celice naj bodo enako visoke, njihove širine pa so določene s sledečima praviloma:

- Celica s korenom drevesa zavzema celotno širino platna.
- Če ima neko vozlišče drevesa n otrok, potem je širina celice vsakega njegovega otroka enaka $1/n$ širine celice starševskega vozlišča.

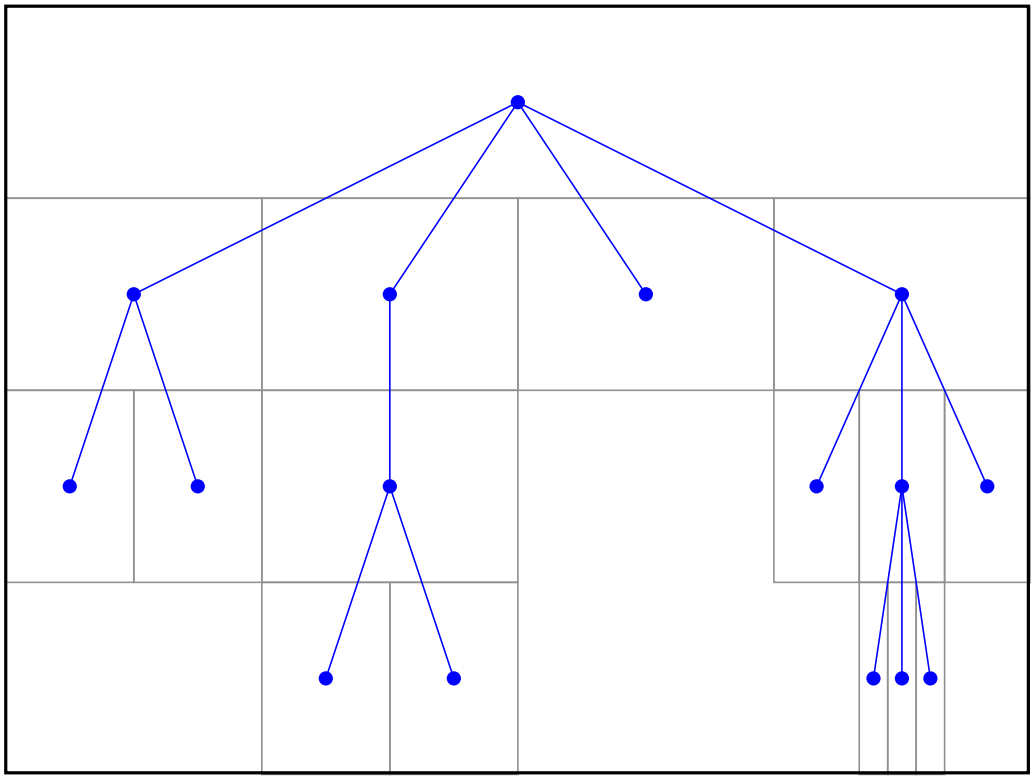
Vozlišča naj bodo prikazana kot enako veliki polni krogi v središčih posameznih celic. Premer vozlišča naj bo enak polovici širine oz. višine (vzame naj se manjša od obeh dimenzij) najožje celice drevesa. Vozlišča naj bodo brez obrobe, zapolnjena pa naj bodo z barvo `Color.BLUE`. Isto barvo uporabite za povezave.

Lahko predpostavite, da je opis drevesa pravilen. Obrobe celic, ki so prikazane na sledečih vzorčnih slikah, so namenjene zgolj lažji predstavi, zato jih ne rišite.

Opis drevesa se ob klicu konstruktorja razreda `Drevo` prenese v atribut `opis`.

Primera

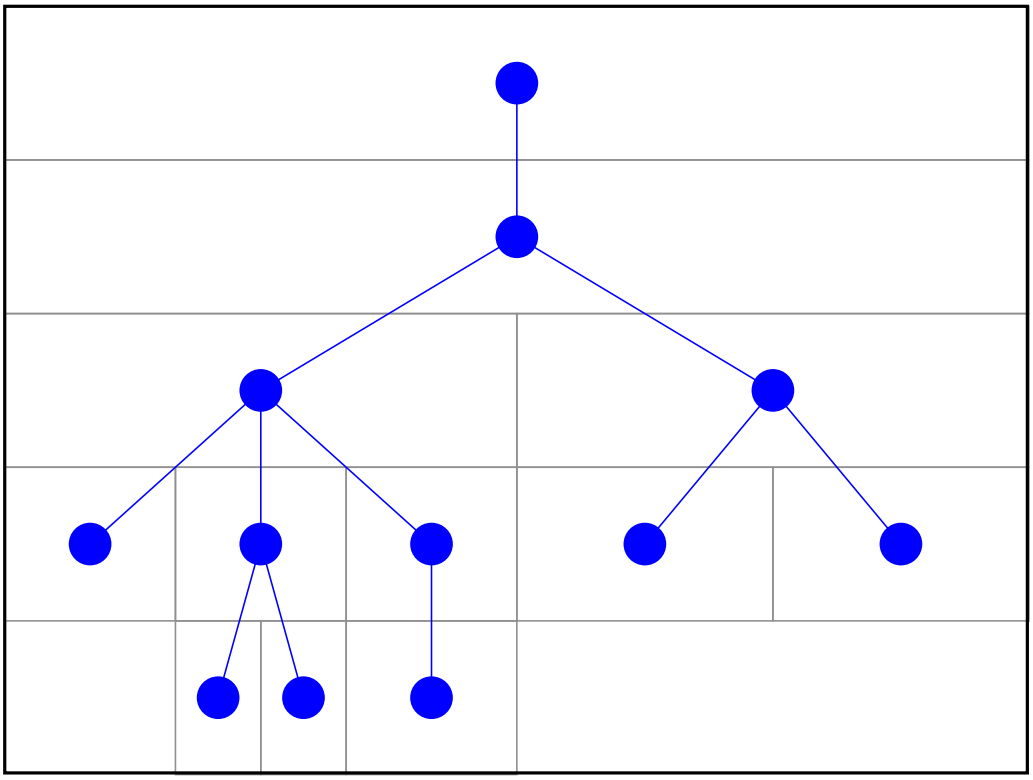
Sliki 10.3 in 10.4 prikazujeta dve različni drevesi.



Slika 10.3: *Drevo z opisom* `4 2 0 0 1 2 0 0 0 3 0 3 0 0 0 0`

Napotek

Opis drevesa lahko na posamezne komponente razbijete s pomočjo metode `split` iz razreda `String`. Druga možnost pa je, da opis drevesa podate konstruktorju razreda `Scanner` in se



Slika 10.4: *Drevo z opisom* `1 2 3 0 2 0 0 1 0 2 0 0`

potem z objektom tega tipa »sprehodite« po nizu na enak način, kot bi se po standardnem vhodu.

10.4 Kochova snežinka (⋆)

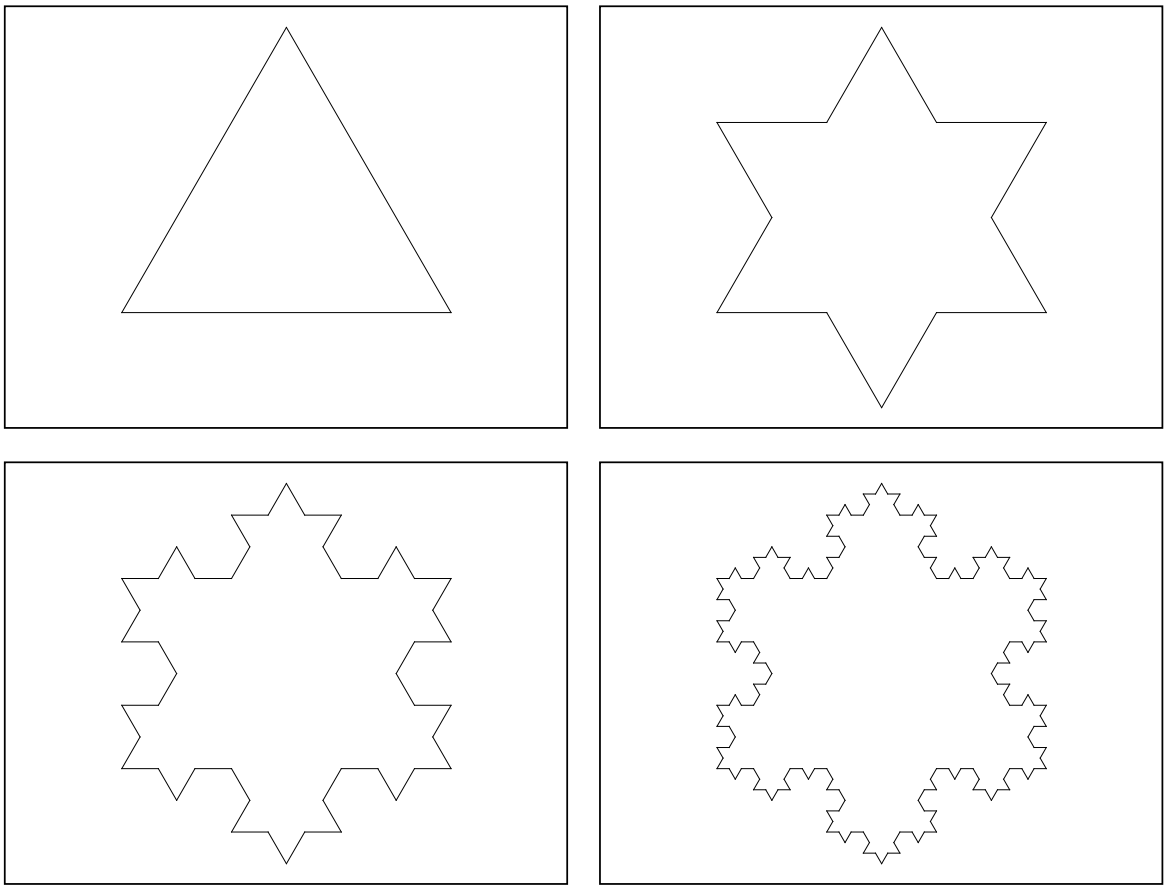
Naloga

Metodo `narisi` v razredu `KochovaSnezinka` dopolnite tako, da bo ta ob zagonu narisal *Kochovo snežinko* po n iteracijah ($n \geqslant 1$). Kochova snežinka po prvi iteraciji je enakostranični trikotnik, vsako naslednjo iteracijo pa dobimo tako, da vsako daljico iz trenutne iteracije razdelimo na tri enake dele, nad srednjim delom narišemo enakostranični trikotnik, nato pa srednji del daljice pobrišemo. Slika 10.5 prikazuje Kochove snežinke po prvih štirih iteracijah.

V vašem programu Kochove snežinke ne boste risali po opisani iterativni shemi, pač pa boste kar neposredno (brez vmesnih iteracij) narisali snežinko po podanem številu iteracij. Še več: **snežinko narišite v eni potezi**, torej kot sklenjeno lomljenko. Pričnite v nekem oglišču snežinke, povlecite daljico do naslednjega oglišča, od tam povlecite daljico do naslednjega oglišča itd. Vse daljice narišite z barvo `Color.BLACK`.

Stranico enakostraničnega trikotnika, ki predstavlja Kochovo snežinko v iteraciji 1, določite s formulo

$$d = \frac{9}{10} \min(w_p, \frac{\sqrt{3}}{2} h_p),$$



Slika 10.5: *Kochove snežinke po prvih štirih iteracijah.*

kjer w_p in h_p označujeta širino in višino platna. Trikotnik naj bo po širini prikazan na sredini platna, vse nadaljnje Kochove snežinke pa naj ležijo na sredini platna tudi po višini.

Namig

Kochovo snežinko je možno razdeliti na tri enake dele, ki izvirajo iz posameznih stranic izhodiščnega enakostraničnega trikotnika. Ali lahko posamezni del iteracije i sestavite s pomočjo posameznih delov iteracije $i - 1$?

11 Grafični uporabniški vmesnik

Splošna navodila

Pri nalogah v tem sklopu boste morali shajati brez testnih primerov in vnaprej pripravljenih ogrodij razredov.

11.1 Pretvornik med številskimi sistemi

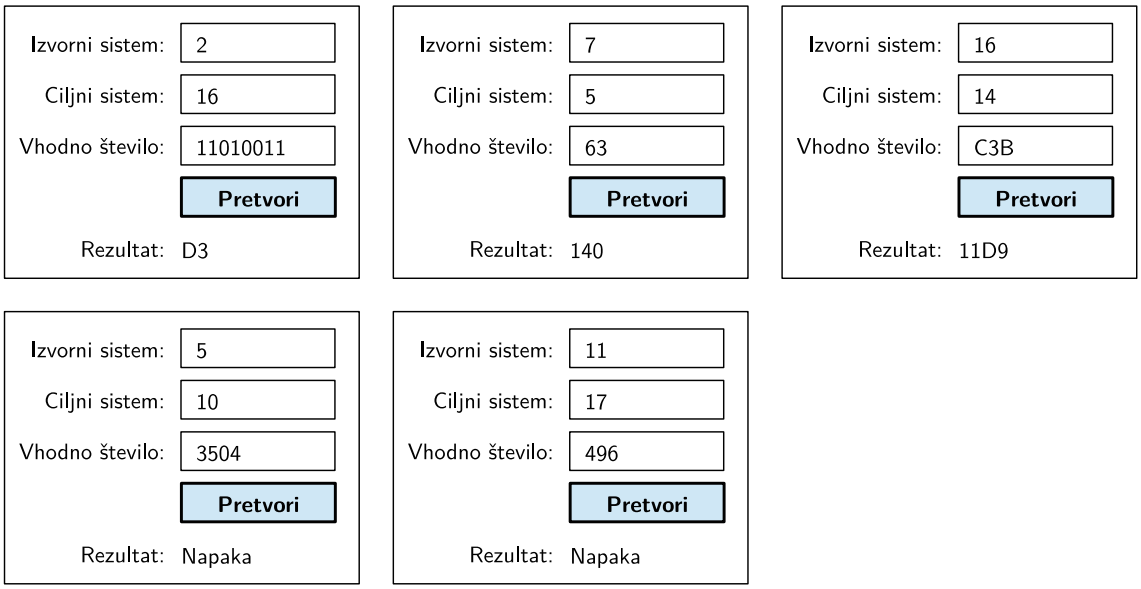
Naloga

Napišite program za pretvarjanje med številskimi sistemi v grafičnem oknu. Program naj uporabniku omogoča vnos izvornega in ciljnega številskega sistema ter pozitivnega celega števila, zapisanega v izvornem številskem sistemu. Po uporabnikovem vnosu podatkov in pritisku na gumb naj program vnešeno število pretvori v podani ciljni številski sistem in prikaže rezultat v obliki oznake (`JLabel`). Na primer, če dvojiško število 11010011 pretvorimo v šestnajstiški sistem, dobimo rezultat D3. (Šestnajstiška števila so sestavljena iz števk 0, 1, ..., 9, A, B, C, D, E in F.)

Program naj deluje za številske sisteme od dvojiškega do vključno šestnajstiškega. Lahko predpostavite, da bo desetiška vrednost podanega števila vedno enaka kvečjemu `Integer.MAX_VALUE`. (Z drugimi besedami: vaš program ima »pravico«, da pri prevelikih vhodnih številih daje napačne rezultate.) Z izjemo te dopustne pomanjkljivosti pa naj program deluje karseda robustno: v primeru neveljavnega vnosa naj ne prikaže obvestila o neulovljeni izjemi, pač pa naj namesto rezultata izpiše niz `Napaka`.

Primeri

Na sliki 11.1 je prikazanih nekaj primerov izvajanja programa po vnosu podatkov in pritisku na gumb za pretvorbo. V četrtem primeru gre za napako v vhodnem številu (petiška števila lahko vsebujejo zgolj števke med 0 in 4), v petem pa za napako v ciljnem številskem sistemu (možne vrednosti so med 2 in 16).



Slika 11.1: *Primeri delovanja pretvornika med številskimi sistemi za različne vnose podatkov.*

Namig

Pri tej nalogi vam bo koristilo poznavanje osnovnih operacij za delo z nizi in znaki. Dolžino niza pridobimo s pomočjo metode (ne atributa!) `length`, posamezne znake niza pa s pomočjo metode `charAt`. Znake lahko med seboj primerjamo in odštevamo. Razlika dveh znakov je enaka razliki njunih kod ASCII. Ker so kode ASCII za znake od `'0'` do `'9'` ter za znake od `'A'` do `'Z'` zaporedne (od 48 do 57 za znake od `'0'` do `'9'` in od 65 do 90 za znake od `'A'` do `'Z'`), lahko znak med `'0'` in `'9'` pretvorimo v število med 0 in 9 s pomočjo izraza `(znak - '0')`. Število med 0 in 9 lahko v znak med `'0'` in `'9'` pretvorimo s pomočjo izraza `(char) (stevilo + '0')`. Podobno lahko obravnavamo znake med `'A'` in `'F'`.

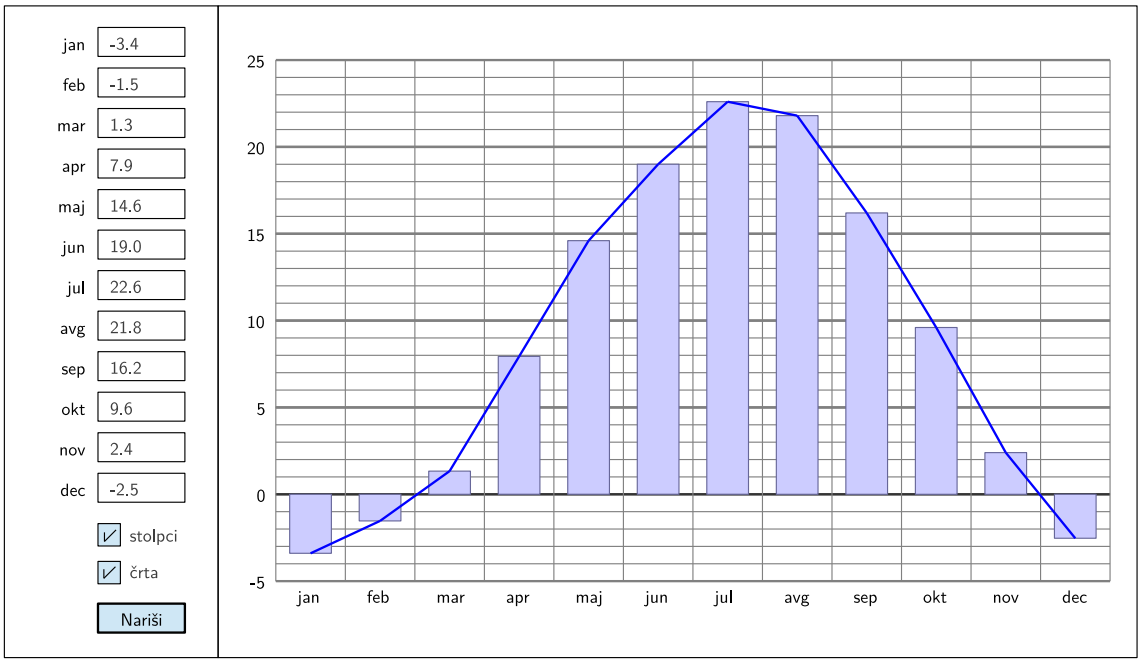
11.2 Diagram temperatur

Naloga

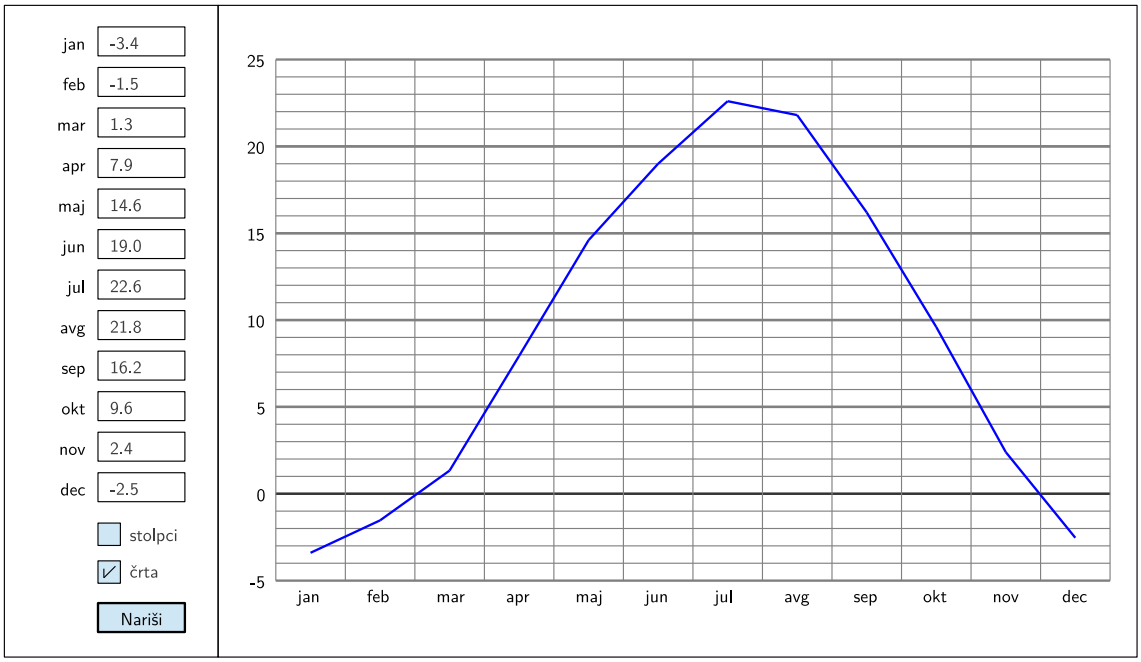
Napišite program za prikaz stolpičnega in/ali črtnega diagrama povprečnih mesečnih temperatur. Program naj se izvaja v oknu, ki vsebuje vnosna polja za vnos posameznih temperatur, potrditveni polji (`JCheckBox`) za določitev načina prikaza diagrama (stolpični diagram, črtni diagram ali oboje), gumb za sprožitev oz. osvežitev izrisa in ploščo za prikaz diagrama. Program naj smiselno deluje tudi v primeru, ko uporabnik ne vnese vseh podatkov.

Primeri

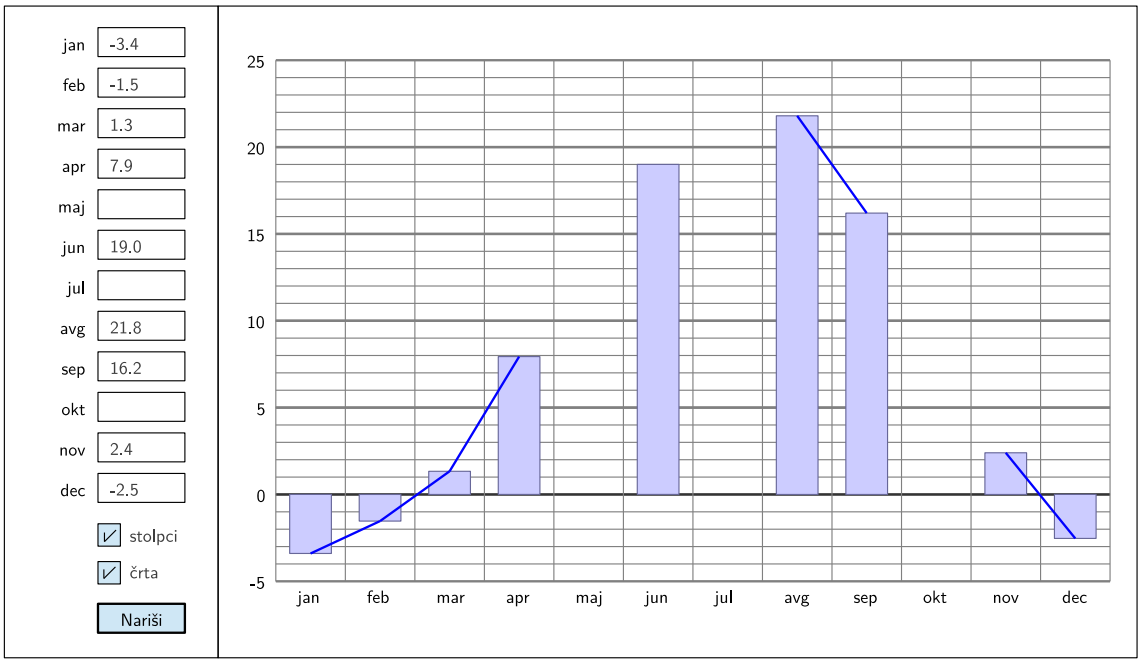
Na slikah 11.2–11.4 so prikazani trije primeri izvajanja programa po vnosu podatkov in pritisku na gumb za izris.



Slika 11.2: *Prikaz gibanja temperatur s stolpičnim in črtnim diagramom.*



Slika 11.3: *Prikaz gibanja temperatur samo s črtnim diagramom.*



Slika 11.4: *Prikaz gibanja temperatur s stolpičnim in črtnim diagramom pri manjkajočih podatkih.*

11.3 Minolovec (⋆)

Naloga

Napišite program za igro Minolovec. Igra se odvija na pravokotni površini z $A \times B$ polji ($A \geqslant 4$, $B \geqslant 4$), pod katerimi je naključno razporejenih M min ($2 \leqslant M \leqslant AB-10$); števila A, B in M naj uporabnik poda kot parametre ob zagonu programa. Naloga uporabnika je določiti položaje vseh M min.

Na začetku igre so vsa polja *zaprta*; uporabnik ne ve, kaj se skriva pod njimi. Nato uporabnik z levim miškinim gumbom *odpira* posamezna polja, dokler ne odpre miniranega polja (v tem primeru takoj izgubi) ali pa dokler ne odpre vseh neminiranih polj (v tem primeru zmaga). Če uporabnik odpre neminirano polje, naj program na tem polju prikaže številko, ki pove, koliko min se nahaja v neposredni okolici (3×3) pravkar odprtega polja. (Največje možno število min v neposredni okolici znaša 8.) Če se v neposredni okolici ne nahaja nobena mina, naj program po enakem pravilu (torej rekurzivno) odpre vsa sosednja polja.

Uporabnik lahko mine tudi aktivno označuje — s pomočjo *zastavic*. Zastavice naj bo možno postavljati in odstranjevati z desnim miškinim gumbom: klik na prazno polje postavi zastavico, klik na polje z zastavico pa zastavico odstrani. Zastavice ne vplivajo na cilj igre (odpreti vsa neminirana polja), ampak služijo zgolj kot pomoč uporabniku. Če uporabnik izgubi (odpre minirano polje), naj program prikaže položaje vseh še ne označenih min in vseh napačno označenih min.

Če uporabnik z levim gumbom klikne na že odprto polje, na katerem se nahaja neka številka n, in če je uporabnik z zastavicami označil natanko n domnevnih min v neposredni okolici,

naj program odpre vsa zaprta polja v okolici tega polja. Če je uporabnik katero od n zastavic postavil napačno, bo igro izgubil, saj bo eno od zaprtih polj v okolici gotovo vsebovalo mino. Ta funkcija sicer ni nujno potrebna za igranje Minolovca, vendar pa igro znatno pospeši.

Mine naj se razporedijo šele po prvem uporabnikovem kliku z levim gumbom, in sicer tako, da v neposredni okolici polja, na katerega klikne uporabnik, ni nobene mine. (Uporabnik tako na začetku vedno klikne na polje s številko 0, kar pomeni, da se bo poleg izbranega polja odprlo še najmanj vseh 8 sosednjih polj.) Na ta način se velikokrat vzpostavi situacija, ki uporabniku omogoča smiselno nadaljevanje igre.

Primer

1. Denimo, da uporabnik izbere igro na plošči 7×7 z 10 minami. Denimo, da najprej klikne na polje (5, 5) (6. vrstica, 6. stolpec). Sedaj mora program naključno razporediti 10 min, upoštevajoč dejstvo, da na polju (5, 5) in v njegovi neposredni okolici — torej na poljih (4, 4), (4, 5), (4, 6), (5, 4), (5, 6), (6, 4), (6, 5) in (6, 6) — ne sme biti nobene mine. Denimo, da program mine razporedi takole:

$$\begin{bmatrix} - & - & * & * & - & - & - \\ * & - & - & - & - & - & - \\ * & - & - & * & * & - & * \\ - & - & * & * & - & - & - \\ - & - & - & - & - & - & - \\ - & - & - & - & - & - & - \\ * & - & - & - & - & - & - \end{bmatrix}$$

2. Ker okolica polja (5, 5) ne vsebuje nobene mine, bo program po kliku na to polje rekurzivno odprl še vsa sosednja polja. Dobimo sledečo sliko (recimo, da program številk 0 zaradi preglednosti ne označuje):

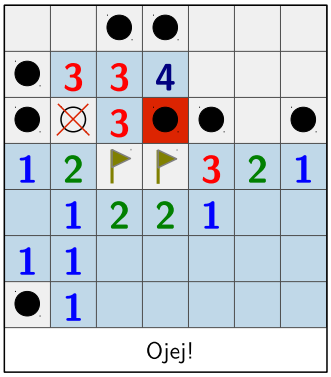
3. Uporabnik zlahka ugotovi, da morata biti na poljih (3, 2) in (3, 3) mini, zato ju označi z zastavicama (desni klik):

4. Ker je sedaj okrog enice na polju (4, 1) označena natanko ena mina, lahko uporabnik klikne na to enico, nakar program odpre vsa zaprta polja v njeni okolici (če pri tem naleti na polje, ki v neposredni okolici nima nobene mine, rekurzivno odpre vse njegove sosede):

5. Mina mora biti bodisi na polju (2, 0) bodisi (2, 1), zato je na polju (2, 2) gotovo ni:

6. Recimo, da uporabnik tvega in označi polje (2, 1) kot minirano:

7. Sedaj lahko klikne na trojko na polju (2, 2). Program bo odprl vsa zaprta polja v okolici tega polja:



Žal se je uporabnik zmotil: na polju (2, 1) mine ni, je pa na polju (2, 3). Ta mina eksplodira ob kliku na trojko na polju (2, 2). Igre je konec!

8. Če bi uporabnik v koraku 6 namesto polja (2, 1) z zastavico označil polje (2, 3), bi po kliku na trojko na polju (2, 2) nastala takšna slika:

9. Sedaj ni težko ugotoviti, da morajo na poljih (1, 0), (2, 0), (2, 4) in (2, 6) biti mine:

10. Polje (2, 5) je potemtakem prazno:

11. Dvojki na polju (2, 5) smo že zadostili, zato lahko nanjo kliknemo:

12. Sedaj lahko kliknemo na dvojko na polju (1, 5):

13. Polji (0, 2) in (0, 3) sta očitno minirani:

14. Sedaj lahko kliknemo na trojko na polju (1, 1):

Po odprtju polj (0, 0) in (0, 1) se na vseh zaprtih poljih (takšno je le še polje (6, 0)) nahajajo mine, zato program sam dopolni manjkajoče zastavice (v našem primeru je ena sama) in čestita uporabniku.

11.4 Elektronska preglednica (⋆)

Naloga

Napišite preprosto elektronsko preglednico. Okno programa naj prikazuje dva stolpca po N vnosnih polj ($1 \leqslant N \leqslant 26$; število N naj vnese uporabnik kot parameter ob zagonu programa), oznake in gumb »Izračunaj!«. Vnosna polja naj bodo označena z zaporednimi črkami angleške abecede. Vnosna polja v levem stolpcu naj bodo namenjena vnosu aritmetičnih izrazov, v vnosnih poljih v desnem stolpcu pa naj se prikazujejo rezultati po pritisku na gumb »Izračunaj!«. Ker so vnosna polja v desnem stolpcu namenjena zgolj prikazu rezultatov, onemogočite vnašanje podatkov vanje (nastavite `setEditable(false)` in `setFocusable(false)`).

V vnosna polja v levem stolpcu naj bo mogoče vnašati izraze, ki jih lahko definiramo na sledeči način. Veljaven *izraz* je eno od sledečega:

- ne-negativno celo število
- oznaka vnosnega polja (A, B, C, ...)

- *izraz* + *izraz*
- *izraz* − *izraz*
- *izraz* ∗ *izraz*
- *izraz*/*izraz* (celoštevilsko deljenje)
- (*izraz*)

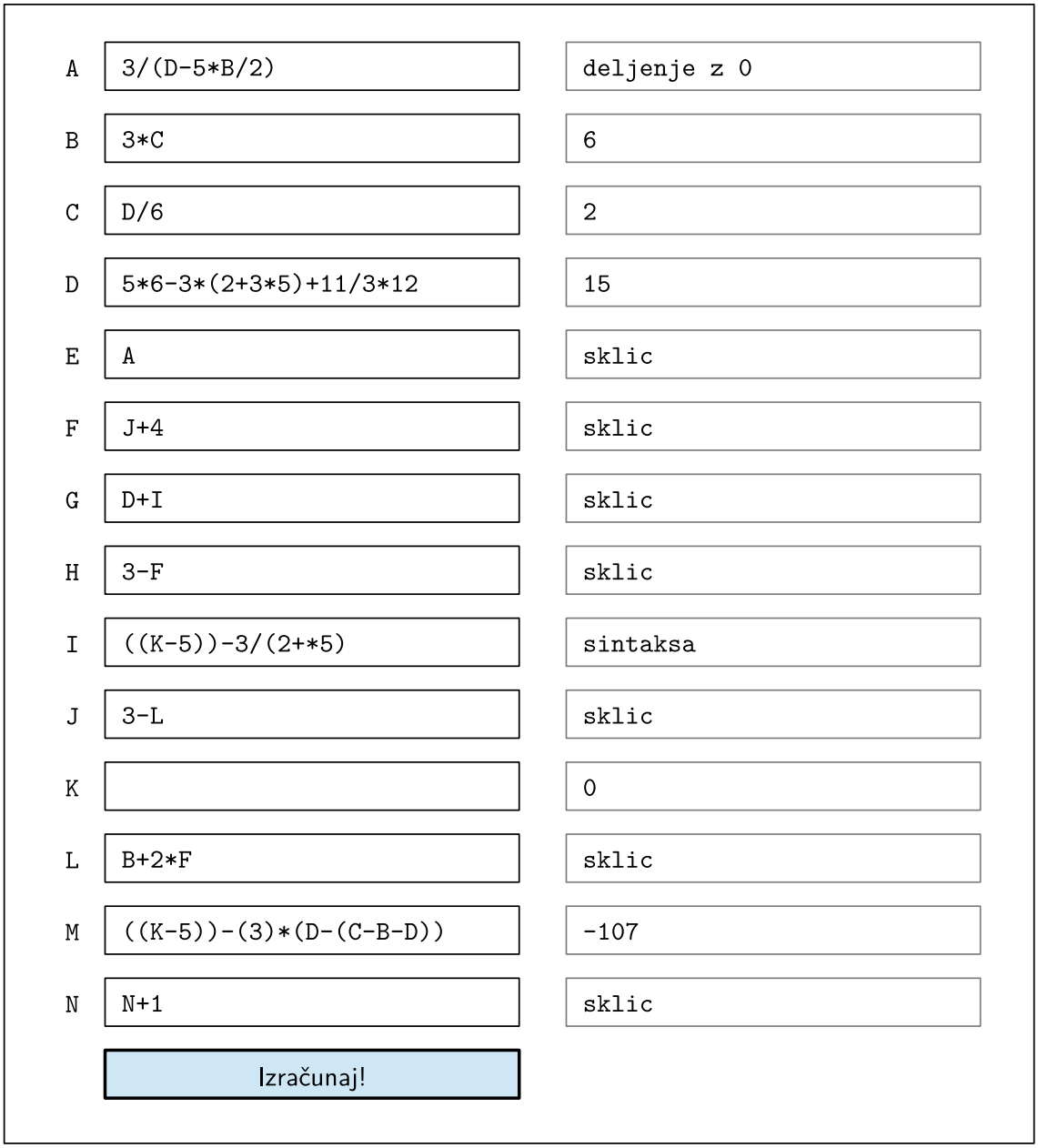
Primer veljavnega izraza je $32 * (0 - 3 * (C + D + 15)/3)$. Rezultat izraza je vedno neko celo število (lahko tudi negativno). Lahko predpostavite, da so podatki in rezultati vedno v mejah od `Integer.MIN_VALUE` do `Integer.MAX_VALUE`.

Po pritisku na gumb »Izračunaj!« naj se izračunajo vrednosti izrazov v vseh vnosnih poljih v levem stolpcu, rezultati pa naj se zapišejo v istoležna vnosna polja v desnem stolpcu. Rezultat praznega vnosnega polja je enak 0. Če izraza v nekem vnosnem polju ni mogoče izračunati, naj program namesto rezultata izpiše sledeče:

- `sintaksa`, če je izraz sintaktično napačen (neujemajoči oklepaji, operatorji brez operandov itd.; izraz -3, denimo, je sintaktično napačen, saj predpostavljamo, da so vsa števila ne-negativna, zato operatorju '−' v tem izrazu manjka levi operand);
- `sklic`, če je izraz sintaktično pravilen, vendar pa vsebuje krožni sklic (npr. sklic na vnosno polje p v vnosnem polju p ali pa sklic na vnosno polje q v vnosnem polju p, pri čemer vnosno polje q neposredno ali posredno vsebuje sklic na vnosno polje p) ali pa sklic na neko vnosno polje, čigar rezultata ni mogoče izračunati.
- `deljenje z 0`, če je izraz sintaktično pravilen in ne vsebuje krožnih sklicev, vendar pa zahteva deljenje z ničlo.

Primer

Slika 11.5 prikazuje vsebino preglednice za $N = 14$ po vnosu izrazov v vnosna polja v levem stolpcu in pritisku na gumb »Izračunaj!«. Ker vnosno polje I vsebuje sintaktično napako (zaporedna operatorja + in ∗), vsebuje vnosno polje G, ki se sklicuje na I, napako sklica. Vnosna polja F, J in L se druga na drugo krožno sklicujejo, zato vsa tri vsebujejo napako sklica. Napako sklica vsebuje tudi vnosno polje H, ki se sklicuje na neizračunljivo vnosno polje F, in vnosno polje E, ki se sklicuje na vnosno polje A, ki vsebuje deljenje z ničlo. Vnosno polje N vsebuje sklic samo nase. Prazno vnosno polje K ima po definiciji vrednost 0. Vrednosti ostalih vnosnih polj se izračunajo po običajnih matematičnih pravilih.



Slika 11.5: *Primer delovanja preglednice.*