

Programski jezici

<http://www.programskijezici.matf.bg.ac.rs/>

**Univerzitet u Beogradu
Matematički fakultet**

Programske paradigme

Materijali za vežbe

**Nastavnik: Milena Vujošević Janičić
Asistent: Branislava Živković**

**Beograd
2016.**

Priprema materijala:

dr Milena Vujošević Jančić, docent na Matematičkom fakultetu u Beogradu

Marjana Šolajić, asistent na Matematičkom fakultetu u Beogradu

Branislava Živković, asistent na Matematičkom fakultetu u Beogradu

Sadržaj

1 Skript programiranje	3
1.1 Uvod, kolekcije, matematičke funkcije	3
1.1.1 Uvodni primeri	3
1.1.2 Zadaci za samostalni rad sa rešenjima	7
1.1.3 Zadaci za vežbu	8
1.2 Datoteke, niske, JSON format, datum	8
1.2.1 Uvodni primeri	8
1.2.2 Zadaci za samostalni rad sa rešenjima	10
1.2.3 Zadaci za vežbu	11
1.3 Argumenti komandne linije, sortiranje, obilazak direktorijuma	12
1.3.1 Uvodni primeri	12
1.3.2 Zadaci za samostalni rad sa rešenjima	14
1.3.3 Zadaci za vežbu	15
1.4 Rešenja	15
2 Programiranje ograničenja - Python	21
2.1 Programiranje ograničenja	21
2.1.1 Uvodni primeri	21
2.1.2 Zadaci za samostalni rad sa rešenjima	22
2.1.3 Zadaci za vežbu	24
2.2 Rešenja	25
3 Funkcionalno programiranje	35
3.1 Uvod	35
3.1.1 Uvodni primeri	35
3.1.2 Zadaci za samostalni rad sa rešenjima	37
3.1.3 Zadaci za vežbu	38
3.2 Liste	38
3.2.1 Uvodni primeri	38
3.2.2 Zadaci za samostalni rad sa rešenjima	39
3.2.3 Zadaci za vežbu	40
3.3 Funkcije	40
3.3.1 Uvodni primeri	40
3.3.2 Zadaci za samostalni rad sa rešenjima	43
3.3.3 Zadaci za vežbu	43
3.4 Rešenja	44
4 Konkurentno programiranje	49
4.1 Scala	49
4.1.1 Uvod u jezik Scala	49
4.1.2 Zadaci	52
4.1.3 Zadaci za vežbu sa rešenjima	54
4.1.4 Zadaci za vežbu	55
4.2 Rešenja	56

5	Distribuirano programiranje	71
5.1	Scala	71
5.1.1	Uvod	71
5.1.2	Zadaci sa rešenjima	73
5.1.3	Zadaci za vezbu	74
5.2	Rešenja	75
6	Logičko programiranje	83
6.1	Uvod	83
6.1.1	Uvodni primeri	83
6.1.2	Zadaci za samostalni rad sa rešenjima	87
6.1.3	Zadaci za vezbu	88
6.2	Liste	89
6.2.1	Uvodni primeri	89
6.2.2	Zadaci za samostalni rad sa rešenjima	90
6.2.3	Zadaci za vezbu	90
6.3	Razni zadaci	91
6.3.1	Zadaci sa rešenjima	91
6.3.2	Zadaci za vezbu	92
6.4	Rešenja	93
7	Programiranje ograničenja - Prolog	99
7.1	Programiranje ograničenja	99
7.1.1	Uvodni primeri	99
7.1.2	Zadaci za samostalni rad sa rešenjima	100
7.1.3	Zadaci za vezbu	102
7.2	Rešenja	103

1

Skript programiranje

Potrebno je imati instaliran Python 2.7 na računaru.

Literatura:

- (a) <https://www.python.org/>
- (b) <http://www.tutorialspoint.com/python>
- (c) <https://wiki.python.org/moin/>

1.1 Uvod, kolekcije, matematičke funkcije

1.1.1 Uvodni primeri

Zadatak 1.1 Ispisivanje pozdravne poruke, komentari.

```
1 # Ovako se pisu komentari
#
3 # Pokretanje programa iz terminala:
# $python hello.py
5 #
print "Hello world! :)"
```

Zadatak 1.2 Promenljive, niske, formatiran ispis, učitavanje sa standardnog ulaza, aritmetičke i logičke operacije, naredbe grananja.

```
2 # Promenljive se dinamički tipiziraju
a = 45
4 b = 67.45
istina = True
6 # Niske su konstantne tj. nisu promenljive.
# To znaci da se menjanjem nekog karaktera u niski
# pravi nova niska u memoriji.
8 niska = "I believe i can fly!"
10
# Ispis na standardni izlaz
12 print a
print b
14 print a, b, istina
16
# Formatiran ispis
17 print "\n-----Formatiran ispis-----\n"
18 print "Ceo broj: {0:d} \nBroj u pokretnom zarezu {1:f}\nBulovska vrednost: {2:b}\nNiska: {3:s}\n".format(a,b,istina,niska)
20
# Ucitavanje niske sa standardnog ulaza
22 print "\n-----Ucitavanje sa standardnog ulaza-----\n"
string_broj = raw_input("Unesite ceo broj: ")
24 broj = int(string_broj) # vrsi se konverzija stringa u ceo broj, slicno: float, str
```



```
26 # Osnovne aritmetičke operacije:
   # +, -, *, /, %, ** (stepenovanje)
28 print "\n-----Osnovne aritmetičke operacije-----\n"
   print broj+4
30
   # Osnovne logičke operacije:
32 # not, and, or
   print "\n-----Osnovne logičke operacije-----\n"
34 print istina or False
36
   # Blokovi se ne ograničavaju viticastim zagradama kao što je u C-u
   # već moraju biti uvučeni tabulatorom.
38
   # Naredba grananja
40 print "\n-----Naredba grananja-----\n"
   if broj%2 == 0:
42     print "Unet je paran broj \n"
   elif broj%3 == 0:
44     print "Unet je broj deljiv sa 3\n"
   # Naredbi <<elif>> može biti više
46 else:
   print "Unet je broj koji nije ni paran ni deljiv sa 3\n"
48
   # Naredba <<switch>> ne postoji
50
   # Petlja
52 print "\n-----Petlja <<while>>-----\n"
   i=1
54 while i<=10:
   print i
56   i=i+1 # i++ ne postoji, može ili ovako ili i+=1
58
   # Naredba <<break>> iskace iz bloka, isto kao i u C-u
   # Naredba <<pass>> je ista kao naredba <<continue>> u C-u
60
   # Funkcije
62 #
   # def ime_funkcije(argumenti):
64 #     telo funkcije
   #
66
   def f1(x,y):
68     return x+y
70 print f1(2,2)
72 def f2(a):
   return [a,2*a,3*a]
74
   print f2(11.1)
```

Zadatak 1.3 Moduli math i random.

```
1 # Matematicke funkcije
3 # Uključujemo modul <<math>>
   import math
5
   # U ovom modulu se nalaze brojne funkcije kao što su:
7 #
   # math.sqrt(broj)
9 # math.log(broj, osnova)
   # math.sin(ugao_u_radijanima), math.cos(), ...
11 # math.exp(stepen)
   # math.factorial(broj)
13 # i druge...
   print "\n-----Matematicke funkcije-----\n"
15 print math.factorial(6)
   print math.log(125, 5)
17
   # Pseudo slučajni brojevi
19
   # Uključujemo modul <<random>>
```

```

21 import random

23 # Funkcija random() vraća pseudo slučajan broj tipa float iz opsega [0.0, 1.0)
print "\n-----Pseudo slučajni brojevi-----\n"
25 print "Pseudo slučajan broj iz opsega [0.0,1.0)\n"
print random.random()

27 # Korisne funkcije:
#
29 # randint(a,b) - vraća pseudo slučajan ceo broj n iz opsega [a,b]
31 # choice(lista) - vraća pseudo slučajan element iz liste
#

```

Zadatak 1.4 Liste.

```

# LISTA
#
# Notacija: [element1, element2, ...]
#
# Liste mogu sadržati različite tipove podataka
6 lista = [1,2,3.4, "Another brick in the wall", True, [5, False, 4.4, 'Layla']]

8 print "\n-----Lista-----\n"
print lista

10 # Prazna lista
12 prazna = []

14 # Pristupanje elementima liste
print "\n-----Pristupanje elementima liste-----\n"
16 # Indeksiranje elemenata liste
print lista[0]
18 print lista[3][1]
print lista[0:3]
20 # Možemo indeksirati liste unazad, pozicija -1 odgovara poslednjem elementu
print lista[-1]
22 # Ukoliko pokušamo da pristupimo elementu liste
# koji se nalazi na poziciji van opsega interpreter će nam prijaviti gresku
24 # IndexError: list index out of range
# print lista[100]

26 print "\n-----Provera da li se element nalazi u listi-----\n"
28 if 1 in lista:
    print "1 se nalazi u listi\n"

30 print "\n-----Korisne funkcije za rad sa listama-----\n"
32 # Ubacivanje elementa na kraj
print "Ubacivanje elementa na kraj liste\n"
34 lista.append(3.14)
print lista

36 # Ubacivanje elementa na određenu poziciju u listi
38 print "\nUbacivanje elementa na određenu poziciju u listi\n"
# list.insert(pozicija, element)
40 lista.insert(2, "Jana")
print lista

42 #
#
44 # Korisne funkcije:
#
46 # list.remove(x) - izbacuje prvo pojavljivanje elementa x iz liste
# list.count(x) - vraća broj koliko puta se element x nalazi u listi
48 # list.index(x) - vraća indeks prvog pojavljivanja elementa x u listi
# len(lista) - vraća broj elemenata liste
50 # del lista[a:b] - briše elemente liste od pozicije a do b
#
# Nadovezivanje dve liste
52 print "\n-----Nadovezivanje dve liste-----\n"
54 lista = lista+["Plava", "Zuta", "Crna"]
print lista

56 #
58 # Prolazak kroz listu

```

```

print "\n-----Prolazak kroz listu petljom <<for>>-----\n"
60 for i in lista:
    print i
62
# Poredjenje listi
64 #
# Dve liste se porede tako sto se njihovi elementi porede redom leksikografski
66 print "\n-----Poredjenje listi-----\n"
print "[1,2,3] < [1,2,5]"
68 print [1,2,3] < [1,2,5]
print "\n['abc','abc','abc'] < ['abc', 'ab', 'abcd']"
70 print ['abc','abc','abc'] < ['abc', 'ab', 'abcd']
print "\n['a','b','c'] > ['a', 'b']"
72 print ['a','b','c'] > ['a', 'b']

74
# Koriscenje liste kao stek strukture podataka
76 stek = [9,8,7]
# Operacija push je implementirana funkcijom append
78 stek.append(6)
stek.append(5)
80 print "\n-----Ispisujemo stek-----\n"
print stek
82 # Operacija pop je implementirana funkcijom pop
print "\n-----Ispisujemo element dobijem funkcijom pop-----\n"
84 print stek.pop()
print "\n-----Ispisujemo znanje nakon pozivanja funkcije pop-----\n"
86 print stek

```

Zadatak 1.5 Skup, katalog, uredene n-torke.

```

2 # SKUP
#
# Pravljenje skupa od liste
4 print "\n-----Pravljenje skupa od liste-----\n"
6 lista1 = [4,56,34,2,5,6,4,4,6]
skup = set(lista1)
8
for i in skup:
10     print i
12
# Funkcija <<range>>
14 #
# range(kraj)
16 # range(pocetak, kraj[, korak])
print "\n-----Funkcija <<range>>-----\n"
18 brojevi = range(10)
for i in brojevi:
20     print i
22
# KATALOG
#
24 # Katalog je kolekcija uredjenih parova oblika (kljuc, vrednost)
#
# Notacija: {kljuc:vrednost, kljuc:vrednost, ...}
26 print "\n-----Katalog-----\n"
prazna_mapa = {} # prazna mapa
28
30 mapa1 = {'a' : 3, 'b' : 4, 'c' : 5}
32
print mapa1
34
mapa = {"kljuc1":67.7, 6:"Vrednost 2"}
36
# Pristupanje elementima u mapi
print "\n-----Pristupanje elementima u katalogu-----\n"
38 print mapa['kljuc1']
40
# Prolazak kroz mapu
print "\n-----Prolazak kroz katalog-----\n"
42 for kljuc in mapa:

```

```

    print "{0:s} => {1:s}\n".format(str(kljuc),str(mapa[kljuc]))
44
# Korisne funkcije
46 #
# map.keys() - vraca listu kljuceva iz kataloga
48 # map.values() - vraca listu vrednosti iz kataloga
# map.has_key(kljuc) - vraca True/False u zavisnosti od toga da li se element
50 # sa kljucem kljuc nalazi u katalogu

52 # Uredjene N-TORKE
print "\n-----Torke-----\n"
54 torka = ("Daffy","Duck",11)

56 # Pristupanje elementima u torci
print "\n-----Pristupanje elementima u torci-----\n"
58 print torka[1]

60 print "\n-----Ispisivanje torke-----\n"
print torka

62
# Poredjenje torki
64 #
# Dve torke se porede tako sto se njihovi elementi porede redom leksikografski
66 print "\n-----Poredjenje torki-----\n"
print "(1,2,'a') < (1,2,'b')"
68 print (1,2,'a') < (1,2,'b')
print "\n([1,2,3], 'Bugs', 4) < ([1,1,1], 'Bunny', 6)"
70 print ([1,2,3], 'Bugs', 4) < ([1,1,1], 'Bunny', 6)
# Ukoliko torke ne sadrže elemente istog tipa na istim pozicijama, i dalje ih mozemo
porediti,
72 # ali poredjenje se vrši na osnovu imena tipa elementa leksikografski
# npr. element tipa List < element tipa String < element tipa Tuple i slicno
74 print "\n(1,2,['a','b']) < (1,2,'ab')"
print (1,2,['a','b']) < (1,2,'ab')
```

1.1.2 Zadaci za samostalni rad sa rešenjima

Zadatak 1.6 Pogodi broj Napisati program koji implementira igricu "Pogodi broj". Na početku igre računar zamišlja jedan slučajan broj u intervalu [0,100]. Nakon toga igrač unosi svoje ime i započinje igru. Igrač unosi jedan po jedan broj sve dok ne pogodi koji broj je računar zamislio. Svaki put kada igrač unese broj, u zavisnosti od toga da li je broj koji je unet veći ili manji od zamišljenog broja ispisuje se odgovarajuća poruka. Igra se završava u trenutku kada igrač pogodio zamišljen broj.

[Rešenje 1.6]

Zadatak 1.7 Aproksimacija broja PI metodom Monte Karlo Napisati program koji aproksimira broj PI koriscenjem metode Monte Karlo. Sa standardnog ulaza unosi se broj N. Nakon toga N puta se bira tačka na slučajan način tako da su obe koordinate tačke iz intervala [0,1]. Broj PI se računa po sledecoj formuli:

$$PI = 4 * A/B$$

- A je broj slučajno izabranih tačaka koje pripadaju krugu poluprečnika 0.5, sa centrom u tački (0.5,0.5)
- B je broj slučajno izabranih tačaka koje pripadaju kvadratu čija temena su tačke (0,0), (0,1), (1,1), (1,0).

[Rešenje 1.7]

Zadatak 1.8 X-O Napisati program koji implementira igricu X-O sa dva igrača.

[Rešenje 1.8]

1.1.3 Zadaci za vežbu

Zadatak 1.9 Anjc Napisati program koji implementira igricu Anjc sa jednim igračem. Igra se sa špilom od 52 karte. Na početku igrač unosi svoje ime nakon čega računar deli dve karte igraču i dve karte sebi. U svakoj sledećoj iteraciji računar deli po jednu kartu igraču i sebi. Cilj igre je sakupiti karte koje u zbiru imaju 21 poen. Karte sa brojevima nose onoliko bodova koliki je broj, dok žandar, dama, kralj nose 10 bodova. Karta As može da nosi 1 ili 10 bodova, u zavisnosti od toga kako igraču odgovara. Igrač koji sakupi 21 je pobedio. Ukoliko igrač premaši 21 bod, porednik je njegov protivnik. <https://en.wikipedia.org/wiki/Blackjack>

Zadatak 1.10 4 u liniji Napisati program koji implementira igricu 4 u nizu sa dva igrača. Tabla za igru je dimenzije 8x8. Igrači na početku unose svoja imena, nakon čega računar nasumično dodeljuje crvenu i žutu boju igračima. Igrač sa crvenom bojom igra prvi i bira kolonu u koju ce da spusti svoju lopticu. Cilj igre je da se sakupe 4 loptice iste boje u liniji. Prvi igrač koji sakupi 4 loptice u liniji je pobedio. https://en.wikipedia.org/wiki/Connect_Four

1.2 Datoteke, niske, JSON format, datum

1.2.1 Uvodni primeri

Zadatak 1.11 Funkcije za rad sa niskama.

```

1 # Niske
2 #
3 # Mozemo ih pisati izmedju jednostrukih i dvostrukih navodnika
4
5 niska1 = 'Ovo je neka niska.'
6 niska2 = "People are strange when you're a stranger ."
7
8 print "\n-----Niske-----\n"
9 print niska1
10 print niska2
11
12 # Karakterima u niski mozemo pristupati koristeći notaciju [] kao kod listi
13 print "\n-----Pristupanje karakterima u niski-----\n"
14 print niska2[4]
15 print niska2[6:10]
16
17 # Duzinu niske racunamo koristeći funkciju len(niska)
18 print "\n-----Duzina niske-----\n"
19 print len(niska1)
20
21 # Funkcija count
22 # niska.count(podniska [, pocetak [, kraj]]) - vraca broj koliko se puta
23 # podniska nalazi u niski (u intervalu od pocetak do kraj)
24 print "\n-----Funkcija <<count>>-----\n"
25 print niska2.count("strange")
26
27 # Funkcija find
28 # niska.find(podniska [, pocetak [, kraj]]) - vraca poziciju prvog pojavljivanja
29 # podniska u niski (u intervalu od pocetak do kraj), -1 ukoliko se podniska ne nalazi
30 # u niski
31 print "\n-----Funkcija <<find>>-----\n"
32 print niska2.find("are")
33
34 # Funkcija join
35 # niska_separator.join([niska1,niska2,niska3,...]) - spaja listu niski separatorom
36 print " ".join(["Olovka", 'pise', 'srcem.'])
37
38 # Korisne funkcije za rad sa niskama:
39 #
40 # niska.isalnum()
41 #     isalpha()
42 #     isdigit()
43 #     islower()

```

```

#         isspace()
46 #         isupper()
# niska.split(separator) - razlaze nisku u listu koristeći separator
48 # niska.replace(stara, nova [, n]) - zamenjuje svako pojavljivanje niske stara
# niskom nova (ukoliko je zadat broj n, onda zamenjuje najviše n pojavljivanja)

```

Zadatak 1.12 Datoteke.

```

1 # Datoteke
#
3 # Datoteku otvaramo koristeći funkciju
#
5 # open(ime_datoteke, mod)
#
7 # mod: "r" -> read, "w" -> write, "a" -> append, "r+" -> read + append
#
9 # Datoteku zatvaramo koristeći funkciju
#
11 # datoteka.close()

13 f = open("dat1.txt", "r")

15 # f.read(n) cita n karaktera iz datoteke
print "\n-----Funkcija <<read>>-----\n"
17 while True:
    c = f.read(2)
19     if c == '':
        break
21     print c

23 # f.readline() cita jednu liniju iz Datoteke
f.close()

25 g = open("dat2.txt", "r")

27 # Liniju po liniju mozemo ucitavati koristeći petlju
29 # tako sto 'iteriramo' kroz Datoteku
print "-----Iteriranje kroz datoteku <<for>> petljom-----\n"
31 for linija in g:
    print linija

33 g.close()
35 # f.readlines() i list(f)
# vraćaju listu linija datoteke
#
37 # f.write(niska) upisuje nisku u datoteku
39 print "-----Upisivanje u datoteku-----\n"
h = open("dat3.txt", "r+")
41 h.write("water\n")

43 print h.readlines()

45 h.close()

```

Zadatak 1.13 Modul datetime.

```

1 # Datumi

3 # Uključujemo klasu datetime iz modula datetime

5 from datetime import datetime

7 # Nov objekat datuma:
#
9 # datetime.datetime(godina, mesec, dan [, sat [, minut [, sekund]])
#
11 # Korisne funkcije:
#
13 # datetime.now() - vraća trenutno vreme odnosno datum
# datetime.strptime(datum_niska, format)
15 # datetime.year, datetime.month, datetime.day, datetime.hour, datetime.minute,
    datetime.second,

```

```
# datetime.strptime(format) - vraća string reprezentaciju objekta datuma na osnovu
#   zadanog formata
17 # datetime.strptime(niska, format) - vraća objekat datetime konstruisan na osnovu
#   niske u zadanom formatu
# datetime.time([sat [, minut [, sekund]]) - vraća objekat koji predstavlja vreme
19 # datetime.date(dan, mesec, godina) - vraća objekat datuma
# format:
21 # %A - dan u nedelji (Monday, Tuesday,...)
# %w - dan u nedelji (0, 1, 2,..., 6)
23 # %d - dan (01, 02, 03,...)
# %B - mesec (January, February,...)
25 # %m - mesec (01, 02, ...)
# %Y - godina (1992, 1993,...)
27 # %H - sat (00, 01, ..., 23)
# %M - minut (00, 01, ..., 59)
29 # %S - sekund (00, 01, ..., 59)
#
31
33
35 print "\n-----Datumi-----\n"
print datetime.now().strftime("Dan u nedelji: %a/%w, Dan: %d, Mesec: %b/%m, Godina: %
y, Vreme: %H:%M:%S\n")
print datetime.now().time()
37 print datetime.now().date()
```

Zadatak 1.14 JSON format.

```
1 # JSON format
#
3 # Funkcije za rad sa JSON formatom se nalaze u modulu json
import json
5
# json.dumps(objekat) vraća string koji sadrži JSON reprezentaciju objekta x
7
9 print "\n-----JSON reprezentacija objekta-----\n"
junak = {"Ime": "Dusko", "Prezime": "Dugousko", "Godine": 11}
print json.dumps(junak)
11
# json.dump(x,f) upisuje string sa JSON reprezentacijom objekta x u datoteku f
13
f = open("dat4.json", "w")
15 json.dump(junak, f)
f.close()
17
# json.load(f) učitava iz datoteke string koji sadrži JSON format objekta i vraća
#   objekat
19 print "\n-----Učitavanje objekta iz datoteke-----\n"
f = open("dat4.json", "r")
21 x = json.load(f)
print x['Ime']
23 print x['Prezime']
print x['Godine']
25 f.close()
```

1.2.2 Zadaci za samostalni rad sa rešenjima

Zadatak 1.15 Napisati program koji sa standardnog ulaza učitava ime datoteke i broj n i računa broj pojavljivanja svakog n -grama u datoteci koji su sačinjeni od proizvoljnih karaktera i rezultat upisuje u datoteku `rezultat.json`.

Na primer:

Listing 1.1: `dat.txt`

```
1 Ovo je datoteka dat
```

Listing 1.2: `rezultat.json`

```

1 {
2   'a' : 1, 'ka' : 1, 'ot' : 1, 'ek' : 1,
3   'd' : 2, 'j' : 1, 'da' : 2, 'e' : 1,
4   'o' : 1, 'to' : 1, 'at' : 2, 'je' : 1,
5   'Ov' : 1, 'te' : 1, 'vo' : 1
6 }

```

[Rešenje 1.15]

Zadatak 1.16

U datoteci `korpa.json` se nalazi spisak kupljenog voća u json formatu:

```
1 [ { 'ime' : ime_voca, 'kolicina' : broj_kilograma } , ... ]
```

U datotekama `maxi_cene.json`, `idea_cene.json`, `shopngo_cene.json` se nalaze cene voća u json formatu:

```
1 [ { 'ime' : ime_voca, 'cena' : cena_po_kilogramu } , ... ]
```

Napisati program koji izračunava ukupan račun korpe u svakoj prodavnici i ispisuje cene na standardni izlaz.

[Rešenje 1.16]

1.2.3 Zadaci za vežbu

Zadatak 1.17 Napisati program koji iz datoteke `ispiti.json` učitava podatke o ispitima i njihovim datumima. Ispisati na standardni izlaz za svaki ispit njegovo ime i status "Prosao" ukoliko je ispit prosao, odnosno "Ostalo je jos n dana.", gde je n broj dana od trenutnog datuma do datuma ispita.

Listing 1.3: `ispiti.json`

```

1 [ { 'ime': 'Relacione baze podataka',
2   'datum': '21.09.2016.' },
3   { 'ime': 'Vestacka inteligencija',
4   'datum': '17.06.2017.' },
5   { 'ime': 'Linearna algebra i analiticka geometrija',
6   'datum': '08.02.2017.' } ]

```

Zadatak 1.18 Napisati program koji izdvaja sve jednolinijske i višelinijске komentare iz `.c` datoteke čije ime se unosi sa standardnog ulaza, listu jednih i drugih komentara upisuje u datoteku `komentari.json`. Jednolinijski komentari se navode nakon `//` a višelinijски između `/*` i `*/`.

Listing 1.4: `program.c`

```

#include <stdio.h>
2
// Primer jednolinijskog komentara
4
int main(){
6 /*
   Na ovaj nacin ispisujemo tekst
   na standardni izlaz koristeći jezik C.
8 */
   printf("Hello world!");
10
   // Na ovaj nacin se ispisuje novi red
12   printf("\n");
14 /*
   Ukoliko se funkcija uspesno završila
16   vratamo 0 kao njen rezultat.

```



```
18  */  
    return 0;  
}
```

Listing 1.5: *komentari.json*

```
1 {  
2   'jednolinijski' : ['Primer jednolinijskog komentara',  
3                     'Na ovaj nacin se ispisuje novi red'],  
4   'viselinejski' : ['Na ovaj nacin ispisujemo tekst na standardni  
5                     izlaz koristeći jezik C.',  
6                     'Ukoliko se funkcija uspesno završila  
7                     vracamo 0 kao njen rezultat.'],  
8 }
```

Zadatak 1.19 Napisati program upoređuje dve datoteke čija imena se unose sa standardnog ulaza. Rezultat upoređivanja je datoteka *razlike.json* koja sadrži broj linija iz prve datoteke koje se ne nalaze u drugoj datoteci i obratno. *Napomena* Obratiti pažnju na efikasnost.

Listing 1.6: *dat1.txt*

```
2 //netacno  
   same=1;  
4  
   for(i=0;s1[i]!='\0' && s2[i]!='\0';i++) {  
6     if(s1[i]!=s2[i]) {  
       same=0;  
8       break;  
     }  
10  }  
   return same;
```

Listing 1.7: *dat2.txt*

```
1 //tacno  
3  
   for(i=0;s1[i]!='\0' && s2[i]!='\0';i++){  
5     if(s1[i]!=s2[i])  
       return 0;  
7   }  
   return s1[i]==s2[i];
```

Listing 1.8: *razlike.json*

```
1 {  
2   'dat1.txt' : 7,  
3   'dat2.txt' : 4  
4 }
```

1.3 Argumenti komandne linije, sortiranje, obilazak direktorijuma

1.3.1 Uvodni primeri

Zadatak 1.20 Modul `sys` i argumenti komandne linije

```
1 # modul sys ima definisan objekat argv koji predstavlja listu argumenata komandne  
   linije (svi argumenti se cuvaju kao niske karaktera)  
2
```

```
import sys
4
if len(sys.argv)==1:
6     print "Niste naveli argumente komandne linije"
    # funkcija exit() iz modula sys prekida program
8     # (ukoliko se ne prosledi argument, podrazumevano
    # se salje None objekat)
10    exit()

12 # ispisujemo argumente komandne linije
# prvi argument, tj. sys.argv[0] je uvek ime skript fajla koji se pokrece
14 for item in sys.argv:
    print item

16
# korisnik moze da zada ime datoteke kao prvi argument komandne linije
18 # u tom slucaju datoteku otvaramo sa f = open(sys.argv[1], "r")
```

Zadatak 1.21 Modul os

```
# Prolazak kroz direktorijume
2
import os
4 # ispisuje oznaku za tekuci direktorijum
print os.getcwd()
6 # ispisuje oznaku za roditeljski direktorijum tekuceg direktorijuma
print os.pardir
8 # ispisuje separator koji koristi za pravljenje putanja
print os.sep
10

12 # funkcija za prosledjenu putanju direktorijuma vraca listu imena svih fajlova u tom
    direktorijumu, . je zamena za putanju tekuceg direktorijuma
14 print os.listdir(".")

16 #
# os.walk() - vraca listu torki (trenutni_direktorijum, poddirektorijumi, datoteke)
18 # os.path.join(putanja, ime) - pravi putanju tako sto nadovezuje na prosledjenu
    putanju zadato ime odvojeno /

20 print "\n-----Prolazak kroz zadati direktorijum-----\n"
for (trenutni_dir, poddirektorijumi, datoteke) in os.walk("."):
22     print trenutni_dir
    for datoteka in datoteke:
24         print os.path.join(trenutni_dir, datoteka)

26 # os.path.abspath(path) - vraca apsolutnu putanju za zadatu relativnu putanju nekog
    fajla
# os.path.isdir(path) - vraca True ako je path putanja direktorijuma, inace vraca
    False
28 # os.path.isfile(path) - vraca True ako je path putanja regularnog fajla, inace vraca
    False

30 print "\n-----Regularni fajlovi zadatag direktorijuma-----\n"
for ime in os.listdir("."):
32     # ako je regularan fajl u pitanju ispisujemo njegovu apsolutnu putanju
    if os.path.isfile(os.path.join(".", ime)):
34         print os.path.abspath(os.path.join(".", ime))
```

Zadatak 1.22 Sortiranje

```
# Sortiranje
#
3 # sorted(kolekcija [, poredi [, kljuc [, obrni]]) - vraca sortiranu kolekciju
#
5 # kolekcija - kolekcija koju zelimo da sortiramo
# poredi - funkcija poredjenja
7 # kljuc - funkcija koja vraca kljuc po kome se poredi
# obrni - True/False (opadajuce/rastuce)
9 #
# za poziv sorted(kolekcija) koristi se funkcija cmp za poredjenje
11 # cmp(x, y) -> integer
```

```

# vraca negativnu vrednost za x<y, 0 za x==y, pozitivnu vrednost za x>y
13 # ako su x i y niske, cmp ih leksikografski poredi
#
15
import json
17 import math

19 l = ["A", "C", "D", "5", "1", "3"]
print l
21 print "sortirana lista: ", sorted(l)

23 # u sledecem primeru je neophodno da definisemo svoje funkcije za poredjenje i
# vracanje kljucja jer je kolekcija lista recnika i za to cmp nema definisano
# ponasanje
tacke = [{"teme": "A", "koordinata": [10.0, 1.1]}, {"teme": "B", "koordinata": [1.0,
15.0]}, {"teme": "C", "koordinata": [-1.0, 5.0]}]
25

27 # funkcija koja tacke x i y poredi po njihovoj udaljenosti od koordinatnog pocetka
def poredi(x,y):
29     if (x[0]*x[0] + x[1]*x[1]) > (y[0]*y[0] + y[1]*y[1]):
        return 1
31     else:
        return -1
33 # funkcija kljuc kao argument ima element kolekcije koja se poredi, u ovom slucaju je
# to jedan recnik
# povratna vrednost funkcije kljuc je u stvari tip argumenata funkcije poredi
35 def kljuc(x):
    return x["koordinata"]
37
sortirane_tacke = sorted(tacke, poredi, kljuc) # ili sorted(tacke, poredi, kljuc,
True) ako zelimo opadajuce da se sortira
39 print "Tacke pre sortiranja:"
for item in tacke:
41     print item["teme"],
print "\nTacke nakon sortiranja: "
43 for item in sortirane_tacke:
    print item["teme"],
45 print

```

1.3.2 Zadaci za samostalni rad sa rešenjima

Zadatak 1.23 Napisati program koji računa odnos kardinalnosti skupova duže i šire za zadati direktorijum. Datoteka pripada skupu duže ukoliko ima više redova od maksimalnog broja karaktera po redu, u suprotnom pripada skupu šire. Sa standardnog ulaza se unosi putanja do direktorijuma. Potrebno je obići sve datoteke u zadatom direktorijumu i njegovim poddirektorijumima (koristiti funkciju `os.walk()`) i ispisati odnos kardinalnosti skupova duže i šire.

[Rešenje 1.23]

Zadatak 1.24 Napisati program koji obilazi direktorijume rekurzivno i računa broj datoteka za sve postojeće ekstenzije u tim direktorijumima. Sa standardnog ulaza se unosi putanja do početnog direktorijuma, a rezultat se ispisuje u datoteku `rezultat.json`. Na primer:

Listing 1.9: `rezultat.txt`

```

1 {
2   'txt' : 14,
3   'py'  : 12,
4   'c'   : 10
5 }
6

```

[Rešenje 1.24]

Zadatak 1.25 U datoteci `radnici.json` nalaze se podaci o radnom vremenu zaposlenih u preduzeću u sledecem formatu:

```

1  [ { 'ime' : 'Pera Peric',
2    'odmor' : ['21.08.2016.', '31.08.2016.'],
3    'radno_vreme' : ['08:30', '15:30'] }, ...]

```

Napisati program koji u zavisnosti od unete opcije poslodavcu ispisuje trenutno dostupne radnike odnosno radnike koji su na odmoru. Moguće opcije su 'd' - trenutno dostupni radnici i 'o' - radnici koji su na odmoru. Radnik je dostupan ukoliko nije na odmoru i trenutno vreme je u okviru njegovog radnog vremena.

[Rešenje 1.25]

Zadatak 1.26 Napisati program koji učitava ime datoteke sa standardnog ulaza i na standardni izlaz ispisuje putanje do svih direktorijuma u kojima se nalazi ta datoteka.

[Rešenje 1.26]

1.3.3 Zadaci za vežbu

Zadatak 1.27 Napisati program koji ispisuje na standardni izlaz putanje do lokacija svih Apache virtuelnih hostova na računaru. Smatrati da je neki direktorijum lokacija Apache virtuelnog hosta ukoliko u sebi sadrži `index.html` ili `index.php` datoteku.

Zadatak 1.28 Napisati program koji realizuje autocomplete funkcionalnost. Sa standardnog ulaza korisnik unosi delove reči sve dok ne unese karakter !. Nakon svakog unetog dela reči ispisuju se reči koje počinju tim karakterima. Spisak reči koje program može da predloži se nalazi u datoteci `reci.txt`.

1.4 Rešenja

Rešenje 1.6 Pogodi broj

```

1  # Pogodi broj
3  import random
5  print "----- IGRA: Pogodi broj -----\n"
7  zamisljen_broj = random.randint(0,100)
9  ime = raw_input("Unesite Vase ime: ")
11 print "Zdravo {0:s}. :) \nZamisljio sam neki broj od 1 do 100. Da li mozes da pogodis
    koji je to broj?".format(ime)
13 pogodio = 0;
    while not pogodio:
15     print "Unesi broj:"
        broj = int(raw_input())
17     if broj == zamisljen_broj:
            pogodio = 1
19     elif broj > zamisljen_broj:
            print "Broj koji sam zamisljio je MANJI od {0:d}.".format(broj)
21     else:
            print "Broj koji sam zamisljio je VECI od {0:d}.".format(broj)
23 print "BRAVO!!! Pogodio si! Zamisljio sam {0:d}. Bilo je lepo igrati se sa tobom. :)".
    format(zamisljen_broj)

```

Rešenje 1.7 Aproksimacija broja PI metodom Monte Karlo


```

41     while True:
42         x,y = učitaj_koordinate(igrac[0])
43         if tabla[x][y] == "-":
44             tabla[x][y] = igrac[1]
45             ispisi_tablu(tabla)
46             break
47         else:
48             print tabla[x][y]
49             print "Uneto polje je popunjeno!\n"
51 print "IGRA: X-O pocinje\n"
53 ime1 = raw_input("Unesite ime prvog igraca: ")
54 print "Zdravo {0:s}!\n".format(ime1)
55 ime2 = raw_input("Unesite ime drugog igraca: ")
56 print "Zdravo {0:s}!\n".format(ime2)
57
58 indikator = random.randint(1,2)
59 if indikator == 1:
60     prvi_igrac = (ime1, "X")
61     drugi_igrac = (ime2, "O")
62 else:
63     prvi_igrac = (ime2, "X")
64     drugi_igrac = (ime1, "O")
65
66 print "Igrac {0:s} igra prvi. \n".format(prvi_igrac)
67 print "X : {0:s}\n".format(prvi_igrac)
68 print "O : {0:s}\n".format(drugi_igrac)
69
70 tabla = [['-', '-', '-'], ['-', '-', '-'], ['-', '-', '-']]
71
72 print "Zapocnimo igru \n"
73
74 ispisi_tablu(tabla)
75
76 na_redu = 0
77 iteracija = 0
78 igraci = [prvi_igrac, drugi_igrac]
79 while iteracija < 9:
80     korak(igraci[na_redu])
81     if pobedio(tabla) == True:
82         print "BRAVO!!!!!! {0:s} je pobedio!\n".format(igraci[na_redu][0])
83         break
84     na_redu = (na_redu+1)%2
85     iteracija = iteracija + 1
87
88 if iteracija == 9:
89     print "NERESENO! Pokusajte ponovo.\n"

```

Rešenje 1.15

```

# dat.txt:
2 # Ovo je datoteka dat
#
4 # rezultat.json:
#
6 # {"a ": 1, "ka": 1, "ot": 1, "ek": 1, " d": 2, " j": 1, "da": 2, "e ": 1, "o ": 1, "
   to": 1, "at": 2, "je": 1, "Ov": 1, "te": 1, "vo": 1}
8 import json
10 ime_datoteke = raw_input("Unesite ime datoteke: ")
11 n = int(raw_input("Unesite broj n: "))
12
13 # Otvaramo datoteku i citamo njen sadrzaj
14 f = open(ime_datoteke, "r")
15 sadrzaj = f.read()
16 f.close()
17
18 recnik = {}
19 i = 0
20 # Prolazimo kroz sadrzaj i uzimamo jedan po jedan n-gram

```

```
22 while i < len(sadrzaj) - n:  
    ngram = sadrzaj[i : i+n]  
    # Ukoliko se n-gram vec nalazi u recniku,  
    # povecavamo mu broj pojavljivanja  
    if ngram in recnik:  
        recnik[ngram] = recnik[ngram]+1  
    # Dodajemo n-gram u recnik i postavljamo mu broj na 1  
    else:  
        recnik[ngram] = 1  
    i = i + 1  
32 f = open("rezultat.json", "w")  
    json.dump(recnik, f)  
34 f.close()
```

Rešenje 1.16

```
1 import json  
  
3 def cena_voca(prodavnica, ime_voca):  
    for voce in prodavnica:  
        if voce['ime'] == ime_voca:  
            return voce['cena']  
7  
# Ucitavamo podatke iz datoteka  
9 f = open('korpa.json', "r")  
korpa = json.load(f)  
11 f.close()  
  
13 f = open('maxi_cene.json', "r")  
maxi_cene = json.load(f)  
15 f.close()  
  
17 f = open('idea_cene.json', "r")  
idea_cene = json.load(f)  
19 f.close()  
  
21 f = open('shopngo_cene.json', "r")  
shopngo_cene = json.load(f)  
23 f.close()  
  
25 maxi_racun = 0  
idea_racun = 0  
27 shopngo_racun = 0  
i = 0  
29 # Za svako voce u korpi dodajemo njegovu cenu u svaki racun posebno  
while i < len(korpa):  
31     ime_voca = korpa[i]['ime']  
    maxi_racun = maxi_racun + korpa[i]['kolicina']*cena_voca(maxi_cene, ime_voca)  
33     idea_racun = idea_racun + korpa[i]['kolicina']*cena_voca(idea_cene, ime_voca)  
    shopngo_racun = shopngo_racun + korpa[i]['kolicina']*cena_voca(shopngo_cene,  
        ime_voca)  
35     i += 1  
  
37 print "Maxi: " + str(maxi_racun) + " dinara"  
print "Idea: " + str(idea_racun) + " dinara"  
39 print "Shopngo: " + str(shopngo_racun) + " dinara"
```

Rešenje 1.23

```
1 import os  
  
3 dat_u_duze = 0  
5 dat_u_sire = 0  
  
7 # Funkcija koja obilazi datoteku i vraca 1 ukoliko datoteka pripada skupu duze  
# odnosno 0 ukoliko datoteka pripada skupu sire  
9 def obilazak(ime_datoteke):  
    br_linija = 0
```

```

11     najduza_linija = 0
12     f = open(ime_datoteke, "r")
13     for linija in f:
14         br_linija = br_linija + 1
15         if len(linija) > najduza_linija:
16             najduza_linija = len(linija)
17     f.close()
18     if br_linija > najduza_linija:
19         return 1
20     else:
21         return 0
22
23 ime_direktorijuma = raw_input("Unesite putanju do direktorijuma: ")
24
25 for (tren_dir, pod_dir, datoteke ) in os.walk(ime_direktorijuma):
26     for dat in datoteke:
27         if obilazak(os.path.join(tren_dir, dat)) == 0:
28             dat_u_sire += 1
29         else:
30             dat_u_duze += 1
31
32 print "Kardinalnost skupa duze: kardinalnost skupa sire"
33 print str(dat_u_duze)+" "+str(dat_u_sire)

```

Rešenje 1.24

```

1     import os
2     import json
3
4     ime_direktorijuma = raw_input("Unesite putanju do direktorijuma: ")
5
6     ekstenzije = {}
7
8     for (tren_dir, pod_dir, datoteke ) in os.walk(ime_direktorijuma):
9         for dat in datoteke:
10            pozicija = dat.find(".")
11            # Ukoliko datoteka ima ekstenziju, pretpostavljamo da su datoteke imenovane
12            tako da posle . ide ekstenzija u ispravnom obliku
13            if pozicija >= 0:
14                # Ukoliko ekstenzija postoji u mapi, povecavamo njen broj
15                if dat[pozicija:] in ekstenzije:
16                    ekstenzije[dat[pozicija:]] += 1
17            else:
18                # Dodajemo novu ekstenziju u mapu i postavljamo njen broj na 1
19                ekstenzije[dat[pozicija:]] = 1
20
21 f = open("rezultat.json", "w")
22 json.dump(ekstenzije, f)
23 f.close()

```

Rešenje 1.25

```

1     import json, os
2     from datetime import datetime
3
4     f = open("radnici.json", "r")
5     radnici = json.load(f)
6     f.close()
7
8     opcija = raw_input("Unesite opciju koju zelite (d - dostupni radnici, o - radnici na
9     odmoru): \n")
10
11 if opcija != "d" and opcija != "o":
12     print "Uneta opcija nije podrzana."
13     exit()
14
15 tren_dat = datetime.now()

```



```
17 # funkcija datetime.strptime(string, format) pravi objekat tipa datetime na osnovu
    # zadatih podataka u stringu i odgovarajuceg formata, na primer ako je datum
    # zapisan kao "21.08.2016" odgovarajuci format je "%d.%m.%Y." pa se funkcija poziva
    # sa datetime.strptime("21.08.2016", "%d.%m.%Y.")
19 for radnik in radnici:
    kraj_odmora = datetime.strptime(radnik['odmor'][1], "%d.%m.%Y.").date()
    pocetak_odmora = datetime.strptime(radnik['odmor'][0], "%d.%m.%Y.").date()
    kraj_rad_vrem = datetime.strptime(radnik['radno_vreme'][1], "%H:%M").time()
    pocetak_rad_vrem = datetime.strptime(radnik['radno_vreme'][0], "%H:%M").time()
23 if opcija == "o":
    # Ukoliko je radnik trenutno na odmoru ispisujemo ga
    if pocetak_odmora < tren_dat.date() < kraj_odmora:
    25     print radnik["ime"]
27 else:
    # Ukoliko je radnik trenutno dostupan i nije na odmoru, ispisujemo ga
    29     if not (pocetak_odmora < tren_dat.date() < kraj_odmora) and pocetak_rad_vrem
        < tren_dat.time() < kraj_rad_vrem:
        print radnik["ime"]
```

Rešenje 1.26

```
import os
2
ime_datoteke = raw_input("Unesite ime datoteke: ")
4
# pretražujemo ceo fajl sistem, odnosno pretragu krecemo od root direktorijuma /
6 # imajte u vidu da ce vreme izvršavanja ovog programa biti veliko posto se pretražuje
    ceo fajl sistem, mozete ga prekinuti u svakom trenutku sa CTRL+C
for (tren_dir, pod_dir, datoteke) in os.walk("/"):
8     # objekat datoteke predstavlja listu imena datoteka iz direktorijuma
    # ta imena poredimo sa zadatim
10     for dat in datoteke:
        # ako smo naisli na trazenu datoteku, pravimo odgovarajucu putanju
    12     if dat == ime_datoteke:
        print os.path.join(os.path.abspath(tren_dir), ime_datoteke)
```

2

Programiranje ograničenja - Python

Potrebno je imati instaliran Python 2.7 i biblioteku python-constraint. Na Ubuntu 14.04 operativnom sistemu, biblioteka python-constraint se može instalirati pomoću Pip alata:

```
sudo apt-get -y install python-pip
sudo pip install python-constraint
```

Korisni linkovi i literatura:

<http://labix.org/doc/constraint/>
<https://pypi.python.org/pypi/python-constraint>
http://www.hakank.org/constraint_programming_blog/

2.1 Programiranje ograničenja

2.1.1 Uvodni primeri

Zadatak 2.1 Modul constraint, osnovne funkcije

```
1 # Programiranje ogranicenja
3 # Uključujemo modul za rad sa ogranicenjima
import constraint
5
# Definiramo problem
7 problem = constraint.Problem()
# Dodajemo promenljive
9 #
# problem.addVariable(ime_promenljive, domen_lista)
11 # problem.addVariables(lista_imena_promenljivih, domen_lista)
problem.addVariable('x',[1,2,3])
13 problem.addVariable('y',['a','b','c'])
# Ispisujemo rešenja
15 # print problem.getSolutions()
17
problem.addVariable('z',[0.1,0.2,0.3])
# Dodajemo ogranicenja
19 #
# problem.addConstraint(ogranicenje [, redosled_promenljivih])
21 #
# ogranicenje može biti:
23 # constraint.AllDifferentConstraint() - različite vrednosti svih promenljivih
# constraint.AllEqualConstraint() - iste vrednosti svih promenljivih
25 # constraint.MaxSumConstraint(s [,tezine]) - suma vrednosti promenljivih (pomnožena
sa tezina) ne prelazi s
# constraint.MinSumConstraint(s [,tezine]) - suma vrednosti promenljivih (pomnožena
sa tezina) nije manja od s
27 # constraint.ExactSumConstraint(s [,tezine]) - suma vrednosti promenljivih (
pomnožena sa tezina) je s
# constraint.InSetConstraint(skup) - vrednosti promenljivih se nalaze u skupu skup
```

```
29 # constraint.NotInSetConstraint(skup) - vrednosti promenljivih se ne nalaze u skupu
    skup
    # constraint.SomeInSetConstraint(skup) - vrednosti nekih promenljivih se nalaze u
    skupu skup
31 # constraint.SomeNotInSetConstraint(skup) - vrednosti nekih promenljivih se ne
    nalaze u skupu skup
    #
33 # redosled_promenljivih predstavlja listu promenljivih
    # i zadaje se zbog definisanja tacnog redosleda
35 # ogranicenja koja se primenjuju na promenljive
    #
37 # Mozemo napraviti i svoju funkciju ogranicenja
def ogranicenje(x,z):
39     if x / 10.0 == z:
        return True
41
    # Prosledjujemo funkciju ogranicenja i redosled promenljivih koji treba da odgovara
    redosledu argumenata funkcije ogranicenja
43 problem.addConstraint(ogranicenje,['x','z'])
    resenja = problem.getSolutions()
45 print "\n-----Resenja-----\n"
    for resenje in resenja:
47     print resenje
```

2.1.2 Zadaci za samostalni rad sa rešenjima

Zadatak 2.2 Napisati program koji pronalazi trocifren broj ABC tako da je količnik $ABC / (A + B + C)$ minimalan i A, B i C su različiti brojevi.

[Rešenje 2.2]

Zadatak 2.3 Dati su novčići od 1, 2, 5, 10, 20 dinara. Napisati program koji pronalazi sve moguće kombinacije tako da zbir svih novčića bude 50.

[Rešenje 2.3]

Zadatak 2.4 Napisati program koji reda brojeve u magičan kvadrat. Magičan kvadrat je kvadrat dimenzija 3x3 takav da je suma svih brojeva u svakom redu, svakoj koloni i svakoj dijagonali jednak 15 i svi brojevi različiti. Na primer:

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

[Rešenje 2.4]

Zadatak 2.5 Napisati program koji pronalazi sve vrednosti promenljivih X, Y i Z za koje važi da je $X \geq Z$ i $X * 2 + Y * X + Z \leq 34$ pri čemu promenljive pripadaju narednim domenima $X \in \{1, 2, \dots, 90\}$, $Y \in \{2, 4, 6, \dots, 60\}$ i $Z \in \{1, 4, 9, 16, \dots, 100\}$

[Rešenje 2.5]

Zadatak 2.6 Napisati program koji dodeljuje različite vrednosti različitim karakterima tako da suma bude zadovoljena:

```
TWO
+TWO
-----
FOUR
```

[Rešenje 2.6]

Zadatak 2.7 Napisati program koji pronalazi sve vrednosti promenljivih X , Y , Z i W za koje važi da je $X \geq 2 * W$, $3 + Y \leq Z$ i $X - 11 * W + Y + 11 * Z \leq 100$ pri čemu promenljive pripadaju narednim domenima $X \in \{1, 2, \dots, 10\}$, $Y \in \{1, 3, 5, \dots, 51\}$, $Z \in \{10, 20, 30, \dots, 100\}$ i $W \in \{1, 8, 27, \dots, 1000\}$.

[Rešenje 2.7]

Zadatak 2.8 Napisati program koji raspoređuje brojeve 1-9 u dve linije koje se seku u jednom broju. Svaka linija sadrži 5 brojeva takvih da je njihova suma u obe linije 25 i brojevi su u rastućem redosledu.

```

1   3
  2  4
   5
  6  8
 7   9

```

[Rešenje 2.8]

Zadatak 2.9 Pekara *Kiftica* proizvodi hleb i kifle. Za mešenje hleba potrebno je 10 minuta, dok je za kiflu potrebno 12 minuta. Vreme potrebno za pečenje ćemo zanemariti. Testo za hleb sadrži 300g brašna, a testo za kiflu sadrži 120g brašna. Zarada koja se ostvari prilikom prodaje jednog hleba je 7 dinara, a prilikom prodaje jedne kifle je 9 dinara. Ukoliko pekara ima 20 radnih sati za mešenje peciva i 20kg brašna, koliko komada hleba i kifli treba da se umesi kako bi se ostvarila maksimalna zarada (pod pretpostavkom da će pekara sve prodati)?

[Rešenje 2.9]

Zadatak 2.10 Napisati program pronalazi vrednosti $A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S$ (svako slovo predstavlja različit broj) koje su poredane u heksagon na sledeći način:

```

  A, B, C
D, E, F, G
H, I, J, K, L
  M, N, O, P
    Q, R, S

```

tako da zbir vrednosti duž svake horizontalne i dijagonalne linije bude 38 ($A+B+C = D+E+F+G = \dots = Q+R+S = 38$, $A+D+H = B+E+I+M = \dots = L+P+S = 38$, $C+G+L = B+F+K+P = \dots = H+M+Q = 38$).

[Rešenje 2.10]

Zadatak 2.11 Kompanija Start ima 250 zaposlenih radnika. Rukovodstvo kompanije je odlučilo da svojim radnicima obezbedi dodatnu edukaciju. Da bi se radnik obučio programskom jeziku Elixir potrebno je platiti 100 evra po osobi za kurs, ali bi njegovo produktivno znanje ovog programskog jezika donelo 150 projekat/sati mesečno, što bi za kompaniju značilo dobit od 5 evra po projekat/satu. Da bi se radnik obučio programskom jeziku Dart potrebno je platiti 105 evra po osobi za kurs, ali bi njegovo produktivno znanje ovog programskog jezika donelo 170 projekat/sati mesečno, koji bi za kompaniju značili dobit od 6 evra po satu. Ukoliko Start ima na raspolaganju 26000 evra za obuku i maksimalan broj 51200 mogućih projekat/sati mesečno, odrediti na koji način kompanija treba da obuči svoje zaposlene kako bi ostvarila maksimalnu dobit.

[Rešenje 2.11]

Zadatak 2.12 Napisati program koji raspoređuje 8 topova na šahovsku tablu tako da se nikoja dva topa ne napadaju.

[Rešenje 2.12]

Zadatak 2.13 Napisati program koji raspoređuje 8 dama na šahovsku tablu tako da se nikoje dve dame ne napadaju.

[Rešenje 2.13]

Zadatak 2.14 Napisati program koji učitava tablu za Sudoku iz datoteke čije ime se zadaje sa standardnog ulaza i korišćenjem ograničenja rešava Sudoku zagonetku.

[Rešenje 2.14]

2.1.3 Zadaci za vežbu

Zadatak 2.15 Za svaku narednu zagonetku, napisati program koji dodeljuje različite vrednosti različitim karakterima tako da suma bude zadovoljena:

```
GREEN + ORANGE = COLORS
MANET + MATISSE + MIRO + MONET + RENOIR = ARTISTS
COMPLEX + LAPLACE = CALCULUS
THIS + IS + VERY = EASY
CROSS + ROADS = DANGER
FATHER + MOTHER = PARENT
WE + WANT + NO + NEW + ATOMIC = WEAPON
EARTH + AIR + FIRE + WATER = NATURE
SATURN + URANUS + NEPTUNE + PLUTO = PLANETS
SEE + YOU = SOON
NO + GUN + NO = HUNT
WHEN + IN + ROME + BE + A = ROMAN
DONT + STOP + THE = DANCE
HERE + THEY + GO = AGAIN
OSAKA + HAIKU + SUSHI = JAPAN
MACHU + PICCHU = INDIAN
SHE + KNOWS + HOW + IT = WORKS
COPY + PASTE + SAVE = TOOLS
```

Zadatak 2.16 Za svaku narednu zagonetku, napisati program koji dodeljuje različite vrednosti različitim karakterima tako da suma bude zadovoljena:

```
THREE + THREE + ONE = SEVEN
NINE + LESS + TWO = SEVEN
ONE + THREE + FOUR = EIGHT
THREE + THREE + TWO + TWO + ONE = ELEVEN
SIX + SIX + SIX = NINE + NINE
SEVEN + SEVEN + SIX = TWENTY
ONE + ONE + ONE + THREE + THREE + ELEVEN = TWENTY
EIGHT + EIGHT + TWO + ONE + ONE = TWENTY
ELEVEN + NINE + FIVE + FIVE = THIRTY
NINE + SEVEN + SEVEN + SEVEN = THIRTY
TEN + SEVEN + SEVEN + SEVEN + FOUR + FOUR + ONE = FORTY
TEN + TEN + NINE + EIGHT + THREE = FORTY
FOURTEEN + TEN + TEN + SEVEN = FORTYONE
NINETEEN + THIRTEEN + THREE + TWO + TWO + ONE + ONE + ONE = FORTYTWO
FORTY + TEN + TEN = SIXTY
SIXTEEN + TWENTY + TWENTY + TEN + TWO + TWO = SEVENTY
SIXTEEN + TWELVE + TWELVE + TWELVE + NINE + NINE = SEVENTY
TWENTY + TWENTY + THIRTY = SEVENTY
FIFTY + EIGHT + EIGHT + TEN + TWO + TWO = EIGHTY
FIVE + FIVE + TEN + TEN + TEN + TEN + THIRTY = EIGHTY
SIXTY + EIGHT + THREE + NINE + TEN = NINETY
ONE + NINE + TWENTY + THIRTY + THIRTY = NINETY
```

Zadatak 2.17 Za svaku narednu zagonetku, napisati program koji dodeljuje različite vrednosti različitim karakterima tako da jednakost bude zadovoljena:

MEN * AND = WOMEN
 COGITO = ERGO * SUM
 ((JE + PENSE) - DONC) + JE = SUIS
 FERMAT * S = LAST + THEOREM.
 WINNIE / THE = POOH
 TWO * TWO + EIGHT = TWELVE

Zadatak 2.18 Uraditi sve zadatke koji su pobrojani ovde:
<http://www.primepuzzle.com/leeslatest/alphameticpuzzles.html>

Zadatak 2.19 Napisati program koji učitava ceo broj n i ispisuje magičnu sekvencu S brojeva od 0 do $n - 1$. $S = (x_0, x_1, \dots, x_{n-1})$ je magična sekvenca ukoliko postoji x_i pojavljivanja broja i za $i = 0, 1, \dots, n - 1$.

Zadatak 2.20 Čistačica Mica sređuje i čisti kuće i stanove. Da bi sredila i počistila jedan stan potrebno joj je 1 sat, dok joj je za kuću potrebno 1.5 sati. Prilikom čišćenja, Mica potroši neku količinu deterdženta, 120ml po stanu, odnosno 100ml po kući. Mica zaradi 1000 dinara po svakom stanu, odnosno 1500 dinara po kući. Ukoliko Mica radi 40 sati nedeljno i ima 5l deterdženta na raspolaganju, koliko stanova i kuća je potrebno da očisti kako bi imala najveću zaradu?

Zadatak 2.21 Marija se bavi grnčarstvom i pravi šolje i tanjire. Da bi se napravila šolja, potrebno je 6 minuta, dok je za tanjir potrebno 3 minuta. Pri pravljenju šolje potroši se 75 gr, dok se za tanjir potroši 100 gr gline. Ukoliko ima 20 sati na raspolaganju za izradu svih proizvoda i 250 kg gline, a zarada koju ostvari iznosi 2 evra po svakoj šolji i 1.5 evra po tanjiru, koliko šolja i tanjira treba da napravi kako bi ostvarila maksimalnu zaradu?

Zadatak 2.22 Jovanin komšija preprodaje računare i računarsku opremu. Očekuje isporuku računara i štampača. Pri tom, računari su spakovani tako da njihova kutija zauzima 360 kubnih decimetara prostora, dok se štampači pakuju u kutijama koje zauzimaju 240 kubnih decimetara prostora. Komšija se trudi da mesečno proda najmanje 30 računara i da taj broj bude bar za 50% veći od broja prodatih štampača. Računari koštaju 200 evra po nabavnoj ceni, a prodaju se po ceni od 400 evra, dok štampači koštaju u nabavci 60 evra i prodaju se za 140 evra. Magacin kojim komšija raspolaže ima svega 30000 kubnih decimetara prostora i mesečno može da nabavi robu u iznosu od najviše 14000 evra. Koliko računara, a koliko štampača komšija treba da proda kako bi se maksimalno obogatilo?

2.2 Rešenja

Rešenje 2.2

```

1  import constraint
3
4  problem = constraint.Problem()
5  # Definisemo promenljive i njihove vrednosti
6  problem.addVariable('A',range(1,10))
7  problem.addVariable('B',range(10))
8  problem.addVariable('C',range(10))
9  # Dodajemo ogranicenje da su vrednosti svih promenljivih razlicite
10 problem.addConstraint(constraint.AllDifferentConstraint())
11 resenja = problem.getSolutions()
12 # Znamo da minimalni kolicnik mora biti manji od 999
13 min_kolicnik = 999
14 min_resenje = {}
15 for resenje in resenja:
16     a = resenje['A']
17     b = resenje['B']
18     c = resenje['C']
19     kolicnik = (float)(a*100 + b*10 + c) / (a+b+c)
20     if kolicnik < min_kolicnik:
21         min_kolicnik = kolicnik
22         min_resenje = resenje

```

```
23 print min_resenje['A']*100 + min_resenje['B']*10 + min_resenje['C']
```

Rešenje 2.3

```
1 import constraint
3 problem = constraint.Problem()
# Definiseimo promenljive za svaki novcic
5 # ako bi se zahtevalo da u kombinaciji bude od svake vrednosti bar po jedan novcic
  samo treba promeniti da domen za svaku promenljivu krece od 1
problem.addVariable("1 din",range(0,51))
7 problem.addVariable("2 din",range(0,26))
problem.addVariable("5 din",range(0,11))
9 problem.addVariable("10 din",range(0,6))
problem.addVariable("20 din",range(0,3))
11
# Problem koji je uocen pri ispisu resenja je sledeci, redosled u kom ce biti dodate
  promenljive problemu ne mora uvek da odgovara redosledu kojim smo mi definisali
  promenljive, u konkretnom primeru (videti oblik u kom ispisuje resenje),
  promenljive ce se dodati u sledecem redosledu: '1 din', '2 din', '10 din', '20 din',
  '5 din' (nacin na koji se kljucevi organizuju u recniku nije striktno
  definisan, primetimo da niske nisu sortirane)
13 # posledica je da postavljanje ogranicenja
# problem.addConstraint(constraint.ExactSumConstraint(50,[1,2,5,10,20]))
15 # nece ispravno dodeliti tezine, na primer, tezinu 5 dodeli promenljivoj '10 din'
  umesto '5 din' kako bismo ocekivali
17 # I nacin da se resi ovaj problem je da redosled promenljivih koji odgovara redosledu
  tezina za ExactSumConstraint prosledimo kao dodatni argument za funkciju
  addConstraint
19 problem.addConstraint(constraint.ExactSumConstraint(50,[1,2,5,10,20]), ["1 din", "2
  din", "5 din", "10 din", "20 din"])
21 # II nacin je da definiseimo svoju funkciju koja predstavlja ogranicenje, samo ce sada
  solver nesto sporije da radi posto ugradjene funkcije imaju optimizovanu
  pretragu i brze dolaze do resenja
#
23 #def o(a, b, c, d, e):
# if a + 2*b + 5*c + 10*d + 20*e == 50:
25 #     return True
#
27 #problem.addConstraint(o, ["1 din", "2 din", "5 din", "10 din", "20 din"])
#
29 resenja = problem.getSolutions()
31 for r in resenja:
  print r
33 # Provera da je suma bas 50
  print r["1 din"] + r["2 din"]*2 + r["5 din"]*5 + r["10 din"]*10 + r["20 din"]*20
```

Rešenje 2.4

```
# 4 9 2
2 # 3 5 7
# 8 1 6
4 #
6 import constraint
8 def o(x,y,z):
  if x+y+z == 15:
10     return True
12 problem = constraint.Problem()
# Promenljive:
14 # a b c
# d e f
16 # g h i
problem.addVariables("abcdefghi", range(1,10))
```

```

18 problem.addConstraint(constraint.AllDifferentConstraint())
# Dodajemo ogranicenja za svaku vrstu
20 problem.addConstraint(o,"abc")
problem.addConstraint(o,"def")
22 problem.addConstraint(o,"ghi")
# Dodajemo ogranicenja za svaku kolonu
24 problem.addConstraint(o,"adg")
problem.addConstraint(o,"beh")
26 problem.addConstraint(o,"cfi")
#Dodajemo ogranicenja za dijagonale
28 problem.addConstraint(o,"aei")
problem.addConstraint(o,"ceg")
30
resenja = problem.getSolutions()
32 for r in resenja:
    print "-----"
34     print "| {0:d} {1:d} {2:d} |".format(r['a'],r['b'],r['c'])
    print "| {0:d} {1:d} {2:d} |".format(r['d'],r['e'],r['f'])
36     print "| {0:d} {1:d} {2:d} |".format(r['g'],r['h'],r['i'])
    print "-----"

```

Rešenje 2.5

```

1 # X,Y,Z
#
3 # X >= Z
# X*2 + X*Y + Z <= 34
5 #
# X <- {1,2,3,...90}
7 # Y <- {2,4,6,...60}
# Z <- {1,4,9,16,...100}
9 #
#
11 import constraint
13
14 problem = constraint.Problem()
15
# Dodajemo promenljivu X i definisemo njen domen
17 problem.addVariable('X', range(1,91))
18
# Dodajemo promenljivu Y i definisemo njen domen
19 problem.addVariable('Y', range(2,61,2))
20
21 domenZ = [];
23 for i in range(1,11):
    domenZ.append(i*i)
25
# Dodajemo promenljivu Z i definisemo njen domen
27 problem.addVariable('Z', domenZ)
28
29 def o1(x,z):
    if x >= z:
31         return True
32
33 def o2(x,y,z):
    if x*2 + x*y + z <= 34:
35         return True;
36
37 # Dodajemo ogranicenja
problem.addConstraint(o1, 'XZ')
39 problem.addConstraint(o2, 'XYZ')
40
41 resenja = problem.getSolutions()
42
43 for r in resenja:
    print "-----"
45     print "X = {0:d} , Y = {1:d} , Z = {2:d}".format(r['X'],r['Y'],r['Z'])
    print "-----"

```


Rešenje 2.6

```

2 # TWO
3 # +TWO
4 # -----
5 # FOUR
6 #
7
8 import constraint
9
10 problem = constraint.Problem()
11 # Definisemo promenljive i njihove vrednosti
12 problem.addVariables("TF", range(1,10))
13 problem.addVariables("WOUR", range(10))
14
15 # Definisemo ogranicenje za cifre
16 def o(t, w, o, f, u, r):
17     if 2*(t*100 + w*10 + o) == f*1000 + o*100 + u*10 + r:
18         return True
19
20 # Dodajemo ogranicenja za cifre
21 problem.addConstraint(o, "TWOFUR")
22 # Dodajemo ogranicenje da su sve cifre razlicite
23 problem.addConstraint(constraint.AllDifferentConstraint())
24
25 resenja = problem.getSolutions()
26
27 for r in resenja:
28     print "-----"
29     print "  "+str(r['T'])+str(r['W'])+str(r['O'])
30     print " "+str(r['T'])+str(r['W'])+str(r['O'])
31     print "="+str(r['F'])+str(r['O'])+str(r['U'])+str(r['R'])

```

Rešenje 2.7

```

2 # Dati sistem nejednacina nema resenje, tj. metog getSolutions() vraca praznu listu
3 # Ukoliko se za promenljivu W domen promeni na {1,...,100} sistem ce imati resenje
4 import constraint
5
6 problem = constraint.Problem()
7
8 # Dodajemo promenljivu X i definisemo njen domen
9 problem.addVariable('X', range(1,11))
10
11 # Dodajemo promenljivu Y i definisemo njen domen
12 problem.addVariable('Y', range(1,52,2))
13
14 domenZ = []
15 domenW = []
16
17 for i in range(1,11):
18     domenZ.append(i*10)
19     domenW.append(i**3)
20
21 # Dodajemo promenljivu Z i definisemo njen domen
22 problem.addVariable('Z', domenZ)
23
24 # Dodajemo promenljivu W i definisemo njen domen
25 problem.addVariable('W', domenW)
26
27 # Za ovako definisan domen za promenljivu W sistem ce imati resenje
28 # problem.addVariable('W', range(1,101))
29
30 def o1(x,w):
31     if x >= 2*w:
32         return True
33
34 def o2(y,z):
35     if 3 + y <= z:
36         return True

```

```

38 def o3(x,y,z,w):
    if x - 11*w + y + 11*z <= 100:
40         return True;

42 # Dodajemo ogranicenja
problem.addConstraint(o1, 'XW')
44 problem.addConstraint(o2, 'YZ')
problem.addConstraint(o3, 'XYZW')

46
resenja = problem.getSolutions()
48 # Proveravamo da li postoji resenje za sistem nejednacina
if resenja==[]:
50     print "Sistem nema resenje."
else:
52     for r in resenja:
        print "-----"
54         print "X = {0:d} , Y = {1:d} , Z = {2:d}, W = {3:d}".format(r['X'],r['Y'],r['
Z'], r['W'])
        print "-----"

```

Rešenje 2.8

```

1 #      1   3
#      2   4
3 #      5
#      6   8
5 #      7   9
#

7
import constraint
9
# Definiseмо ogranicenje za jednu dijagonalu
11 def o(a,b,c,d,e):
    if a<b<c<d<e and a+b+c+d+e==25:
13         return True

15 problem = constraint.Problem()
# Definiseмо promenljive za svaku poziciju
17 problem.addVariables('abcdeABCDE',range(1,10))
# Dodajemo ogranicenja za obe dijagonale
19 problem.addConstraint(o,'abcde')
problem.addConstraint(o,'ABCDE')
21 # Dodajemo ogranicenje da su vrednosti svih promenljivih razlicite
problem.addConstraint(constraint.AllDifferentConstraint())

23
resenja = problem.getSolutions()
25 for r in resenja:
    print "-----"
27     print "{0:d}   {1:d}".format(r['a'],r['A'])
    print " {0:d} {1:d} ".format(r['b'],r['B'])
29     print "  {0:d} ".format(r['c'])
    print " {0:d} {1:d} ".format(r['D'],r['d'])
31     print "{0:d}   {1:d}".format(r['E'],r['e'])
    print "-----"

```

Rešenje 2.9

```

#
2 # Potrebno je napraviti H komada hleba i K komada kifli
#
4 # Zarada iznosi:
# - 7din/hleb, tj. zarada za H komada hleba bice 7*H
6 # - 9din/kifla tj. zarada za K komada kifli bice 9*K
#
8 # Ukupna zarada iznosi:
# 7*H + 9*K - funkcija koju treba maksimizovati
10 #
# Ogranichenja vremena:
12 # - vreme potrebno za mesenje jednog hleba je 10min,
#   tj. za mesenje H komada hleba potrebno je 10*H minuta

```

2 Programiranje ograničenja - Python

```
14 # - vreme potrebno za mesenje jedne kifle je 12min,  
15 #   tj. za mesenje K komada kifli potrebno je 12*K minuta  
16 #  
17 # Ukupno vreme koje je na raspolaganju iznosi 20h, tako da je:  
18 #  $10*H + 12*K \leq 1200$   
19 #  
20 # Ogranicenje materijala:  
21 # - za jedan hleb potrebno je 300g brasna, a za H komada hleba potrebno je H*300  
22 #   grama  
23 # - za jednu kifli potrebno je 120g brasna, a za K komada kifli potrebno je K*120  
24 #   grama  
25 #  
26 # Ukupno, na raspolaganju je 20kg brasna, tako da je:  
27 #  $300*H + 120*K \leq 20000$   
28 #  
29 # Broj kifli i hleba je najmanje 0, tako da:  
30 #  $H \geq 0$   
31 #  $K \geq 0$   
32 #  
33 # S obzirom na to da imamo 20kg brasna na raspolaganju, mozemo napraviti:  
34 # - najvise 20000/120 kifli  
35 # - najvise 20000/300 hleba  
36 #  
37 #  $H \leq 20000/120 \sim 167$   
38 #  $K \leq 20000/300 \sim 67$   
39 #  
40 # S obzirom na to da imamo 20h na raspolaganju, mozemo napraviti:  
41 # - najvise 1200/12 kifli  
42 # - najvise 1200/10 hleba  
43 #  
44 #  $H \leq 1200/10 = 120$   
45 #  $K \leq 1200/12 = 100$   
46 #  
47 # najoptimalnije je za gornju granicu domena postaviti minimum od dobijenih vrednosti  
48 #   , tj. sve ukupno  $H \leq 120$ ,  $K \leq 67$   
49  
50 import constraint  
51  
52 problem = constraint.Problem()  
53  
54 # Dodajemo promenljivu H i definisemo njen domen  
55 problem.addVariable('H', range(0,121))  
56  
57 # Dodajemo promenljivu K i definisemo njen domen  
58 problem.addVariable('K', range(0,68))  
59  
60 def ogranicenje_vremena(h,k):  
61     if 10*h + 12*k <= 1200:  
62         return True  
63  
64 def ogranicenje_materijala(h,k):  
65     if 300*h + 120*k <= 20000:  
66         return True;  
67  
68 # Dodajemo ogranicenja vremena i materijala  
69 problem.addConstraint(ogranicenje_vremena, 'HK')  
70 problem.addConstraint(ogranicenje_materijala, 'HK')  
71  
72 resenja = problem.getSolutions()  
73  
74 # Pronalazimo maksimalnu vrednost funkcije cilja  
75 max_H = 0  
76 max_K = 0  
77  
78 for r in resenja:  
79     if 7*r['H'] + 9*r['K'] > 7*max_H + 9*max_K:  
80         max_H = r['H']  
81         max_K = r['K']  
82  
83 print "-----"  
84 print "Maksimalna zarada je {0:d}, komada hleba je {1:d}, a komada kifli {2:d}".  
85     format(7*max_H + 9*max_K, max_H, max_K)  
86 print "-----"
```

Rešenje 2.10

```

1 import constraint
3
4 def o1(x,y,z):
5     if x+y+z == 38:
6         return True
7 def o2(x,y,z,w):
8     if x+y+z+w == 38:
9         return True
10 def o3(x,y,z,w,h):
11     if x+y+z+w+h == 38:
12         return True
13
14 problem = constraint.Problem()
15 problem.addVariables("ABCDEFGHJKLMNOPQRST", range(1,38))
16 problem.addConstraint(constraint.AllDifferentConstraint())
17 # Dodajemo ogranicenja za svaku horizontalnu liniju
18 # A,B,C
19 # D,E,F,G
20 #H,I,J,K,L
21 # M,N,O,P
22 # Q,R,S
23 problem.addConstraint(o1,"ABC")
24 problem.addConstraint(o2,"DEFG")
25 problem.addConstraint(o3,"HIJKL")
26 problem.addConstraint(o2,"MNOP")
27 problem.addConstraint(o1,"QRS")
28
29 # Dodajemo ogranicenja za svaku od glavnih dijagonala
30 # A,B,C
31 # D,E,F,G
32 #H,I,J,K,L
33 # M,N,O,P
34 # Q,R,S
35
36 problem.addConstraint(o1,"HMQ")
37 problem.addConstraint(o2,"DINR")
38 problem.addConstraint(o3,"AEJOS")
39 problem.addConstraint(o2,"BFKP")
40 problem.addConstraint(o1,"CGL")
41
42 # Dodajemo ogranicenja za svaku od sporednih dijagonala
43 # A,B,C
44 # D,E,F,G
45 #H,I,J,K,L
46 # M,N,O,P
47 # Q,R,S
48
49 problem.addConstraint(o1,"ADH")
50 problem.addConstraint(o2,"BEIM")
51 problem.addConstraint(o3,"CFJNQ")
52 problem.addConstraint(o2,"GKOR")
53 problem.addConstraint(o1,"LPS")
54
55 resenja = problem.getSolutions()
56 for r in resenja:
57     print " -----"
58     print " {0:d},{1:d},{2:d}".format(r['A'],r['B'],r['C'])
59     print " {0:d},{1:d},{2:d},{3:d}".format(r['D'],r['E'],r['F'],r['G'])
60     print " {0:d},{1:d},{2:d},{3:d},{4:d}".format(r['H'],r['I'],r['J'],r['K'],r['L'])
61     print " {0:d},{1:d},{2:d},{3:d}".format(r['M'],r['N'],r['O'],r['P'])
62     print " {0:d},{1:d},{2:d}".format(r['Q'],r['R'],r['S'])
63     print " -----"

```

Rešenje 2.11

```

1 import constraint
3
4 problem = constraint.Problem()
5 # Kompanija ima 250 zaposlenih radnika

```

```

5 # za sve njih organizuje dodatnu obuku
# ako je E promenljiva za Elixir, a D za Dart
7 # mora da vaziti E<=250, D<=250 i E + D = 250
#
9 # Dodajemo promenljivu E i definisemo njen domen
problem.addVariable('E', range(0,251))
11
13 # Dodajemo promenljivu D i definisemo njen domen
problem.addVariable('D', range(0,251))
15
17 def ukupno_radnika(e,d):
    if e+d == 250:
        return True
19
21 def ogranicenje_projekat_sati(e,d):
    if 150*e + 170*d <= 51200:
        return True
23
25 def ogranicenje_sredstava(e,d):
    if 100*e + 105*d <= 26000:
        return True;
27
29 # Dodajemo ogranicenja za broj projekat/sati i ukupna sredstva
# na raspolaganju kao i za broj radnika u firmi
31 problem.addConstraint(ogranicenje_projekat_sati, 'ED')
problem.addConstraint(ogranicenje_sredstava, 'ED')
problem.addConstraint(ukupno_radnika, 'ED')
33
35 resenja = problem.getSolutions()
# Pronalazimo maksimalnu vrednost funkcije cilja
37 max_E = 0
max_D = 0
# Od ostvarene dobiti preko broja projekat/sati oduzimamo gubitak za placanje kurseva
radnicima
39 for r in resenja:
    if 150*5*r['E'] + 170*6*r['D'] - (100*r['E'] + 105*r['D']) > 150*5*max_E + 170*6*
max_D - (100*max_E + 105*max_D) :
41         max_E = r['E']
max_D = r['D']
43
45 print "Maksimalna zarada je {0:d}, broj radnika koje treba poslati na kurs Elixir je
{1:d}, a broj radnika koje treba poslati na kurs Dart je {2:d}.".format(170*6*
max_E + 150*5*max_D - (100*max_E + 105*max_D) , max_E, max_D)

```

Rešenje 2.12

```

1 # Jedan od mogucih rasporeda
# sva ostala resenja su permutacije
3 # takve da je u svakoj vrsti i koloni samo jedan top
#
5 # 8 T - - - - -
# 7 - T - - - - -
7 # 6 - - T - - - -
# 5 - - - T - - -
9 # 4 - - - - T - -
# 3 - - - - - T -
11 # 2 - - - - - T -
# 1 - - - - - - T
13 # 1 2 3 4 5 6 7 8
#
15 import constraint
17
19 problem = constraint.Problem()
# Dodajemo promenljive za svaku kolonu i njihove vrednosti 1-8
problem.addVariables("12345678", range(1,9))
21 # Dodajemo ogranicenje da se topovi ne napadaju medjusobno po svakoj vrsti
problem.addConstraint(constraint.AllDifferentConstraint())
23 resenja = problem.getSolutions()

```

```

25 # Broj svih mogućih permutacija je 8! = 40320
# Za prikaz svih najbolje pozvati program sa preusmerenjem izlaznih podataka
27 # python 2_12.py > izlaz.txt
print "Broj resenja je: {0:d}.".format(len(resenja))
29
for r in resenja:
31     print "-----"
    for i in "12345678":
33         for j in range(1,9):
            if r[i] == j:
35                 print "T",
            else:
37                 print "-",
        print ""
39     print "-----"

```

Rešenje 2.13

```

1 # 8 D - - - - -
# 7 - - - - D - - -
3 # 6 - D - - - - -
# 5 - - - - - D - -
5 # 4 - - D - - - -
# 3 - - - - - D -
7 # 2 - - - D - - -
# 1 - - - - - D
9 # 1 2 3 4 5 6 7 8
#
11
import constraint
13 import math
15
problem = constraint.Problem()
# Dodajemo promenljive za svaku kolonu i njihove vrednosti 1-8
17 problem.addVariables("12345678", range(1,9))
# Dodajemo ogranicenje da se dame ne napadaju medjusobno po svakoj vrsti
19 problem.addConstraint(constraint.AllDifferentConstraint())
21
for k1 in range(1,9):
    for k2 in range(1,9):
23         if k1 < k2:
            # Definisemo funkciju ogranicenja za dijagonale
25             def o(vrsta1, vrsta2, kolona1=k1, kolona2=k2):
                if math.fabs(vrsta1 - vrsta2) != math.fabs(kolona1 - kolona2):
27                     return True
            problem.addConstraint(o, [str(k1),str(k2)])
29
resenja = problem.getSolutions()
31 # Za prikaz svih najbolje pozvati program sa preusmerenjem izlaznih podataka
# python 2_13.py > izlaz.txt
33 print "Broj resenja je: {0:d}.".format(len(resenja))
for r in resenja:
35     print "-----"
    for i in "12345678":
37         for j in range(1,9):
            if r[i] == j:
39                 print "D",
            else:
41                 print "-",
        print ""
43     print "-----"

```

Rešenje 2.14

```

1 # 9 - - - - -
# 8 - - - - -
3 # 7 - - - - -
# 6 - - - - -
5 # 5 - - - - -
# 4 - - - - -

```

```

7 # 3 - - - - -
9 # 2 - - - - -
9 # 1 - - - - -
# 1 2 3 4 5 6 7 8 9
11 #
13 import constraint
import json
15
16 problem = constraint.Problem()
17
18 # Dodajemo promenljive za svaki red i njihove vrednosti 1-9
19 for i in range(1, 10):
20     problem.addVariables(range(i * 10 + 1, i * 10 + 10), range(1, 10))
21
22 # Dodajemo ogranicenja da se u svakoj vrsti nalaze razlicite vrednosti
23 for i in range(1, 10):
24     problem.addConstraint(constraint.AllDifferentConstraint(), range(i * 10 + 1, i *
25     10 + 10))
26
27 # Dodajemo ogranicenja da se u svakoj koloni nalaze razlicite vrednosti
28 for i in range(1, 10):
29     problem.addConstraint(constraint.AllDifferentConstraint(), range(10 + i, 100 + i,
30     10))
31
32 # Dodajemo ogranicenja da svaki podkvadrat od 3x3 promenljive ima razlicite vrednosti
33 for i in [1,4,7]:
34     for j in [1,4,7]:
35         pozicije = [10*i+j,10*i+j+1,10*i+j+2,10*(i+1)+j,10*(i+1)+j+1,10*(i+1)+j
36         +2,10*(i+2)+j,10*(i+2)+j+1,10*(i+2)+j+2]
37         problem.addConstraint(constraint.AllDifferentConstraint(),pozicije)
38
39 ime_datoteke = raw_input("Unesite ime datoteke sa tablom za sudoku: ")
40 f = open(ime_datoteke, "r")
41 tabla = json.load(f)
42 f.close()
43
44 # Dodajemo ogranicenja za svaki broj koji je zadat na tabli
45 for i in range(9):
46     for j in range(9):
47         if tabla[i][j] != 0:
48             def o(vrednost_promenljive, vrednost_na_tabli = tabla[i][j]):
49                 if vrednost_promenljive == vrednost_na_tabli:
50                     return True
51
52         problem.addConstraint(o, [((i+1)*10 + (j+1))])
53
54 resenja = problem.getSolutions()
55
56 for r in resenja:
57     print "======"
58     for i in range(1,10):
59         print "|",
60         for j in range(1,10):
61             if j%3 == 0:
62                 print str(r[i*10+j])+" |",
63             else:
64                 print str(r[i*10+j]),
65         print ""
66     if i%3 == 0 and i!=9:
67         print "-----"
68     print "======"

```

3

Funkcionalno programiranje

Potrebno je imati instaliran GHC kompajler na računaru.

Literatura:

- (a) <https://www.haskell.org/>
- (b) <https://wiki.haskell.org/Haskell>

3.1 Uvod

3.1.1 Uvodni primeri

Zadatak 3.1 Korišćenje kompilatora i interpretera.

```
2  -- Ovako pisemo jednolinijske komentare
3  {-
4     Ovako pisemo
5     viselinijске
6     komentare
7  -}
8
9  {-
10     Interpreter pokrecemo iz terminala komandom:
11
12     ghci
13
14     i otvorice nam se interaktivni interpreter:
15
16     GHCi, version 8.0.1: http://www.haskell.org/ghc/  :? for help
17     Prelude>
18
19     Sa interpreterom mozemo direktno komunicirati, npr
20     Prelude> print "Zdravo! :)"
21     "Zdravo! :)"
22
23     Interpretatoru dodajemo mogucnost izvorsavanja funkcija iz izvorne
24     ime_programa.hs komandom:
25
26     :load ime_programa.hs
27
28     Nakon toga mozemo pokrenuti sve funkcije koje su u toj datoteci
29     definisane, npr.
30
31     Prelude> main
32     "Zdravo! :)"
33
34     Iz interpretera se izlazi komandom :quit
35     Prelude> :quit
36
37     Haskell programe mozemo kompajlirati komandom:
38
39     ghc ime_programa.hs
```



```
40     koja ce napraviti izvrsni program koji pokrecemo sa
42     ./ime_programa
44 -}
46 main = print "Zdravo! :)"
```

Zadatak 3.2 Tipovi podataka. Tipski razredi. Funkcije.

```
1 {-
2   Osnovni tipovi:
3   Bool
4   Char
5   String
6   Int
7   Integer
8   Float
9   Double
11
12  Tipski razredi (definisu funkcije koje neki tip mora da implementira):
13  Eq - tipovi sa jednakoscu (==,/=)
14  Ord - tipovi sa uredjenjem (<,<=,>,>=,...)
15  Num - numericki tipovi (+,-,*,...)
16  Integral - celobrojni tipovi (div, mod,...)
17  Fractional - razlomacki tipovi (/ , recip,...)
18
19  Funkcije preslikavaju vrednosti jednog tipa (ili tipskog razreda) u vrednost drugog
20  tipa (ili tipskog razreda):
21
22  duplo :: Int -> Int
23
24  Informacije o bilo kom objektu mozete dobiti naredbom:
25  :info ime_objekta
26 -}
27
28 duplo :: Int -> Int
29 duplo x = x+x
30
31 {-
32   Aritmeticke operacije nad tipovima Int, Integer, Float:
33   +, -, *, /,
34   ^^ - stepen (tip Int)
35   ** - stepen (tip Float)
36
37   Funkcije:
38   mod, div, log, sqrt, sin, cos, tan
39 -}
40
41 ostatak3 x = mod x 3
42 -- ekvivalentno sa x `mod` 3
43
44 -- ukoliko koristimo funkcije koje su definisane za realne tipove nad argumentima
45 -- koji su celobrojnog tipa potrebno je prevesti vrednost u nadklasu Num koriscenjem
46 -- funkcije fromIntegral, nece se izvršiti eksplicitno kastovanje
47 -- primer: funkcija racuna vrednost korena samo za celobrojni argument
48 korenCeli :: Int -> Double
49 korenCeli n = sqrt (fromIntegral n)
50
51 -- Blokovi se u Haskelu odvajaju tabulatorom.
52 -- Uslovni izraz (mora imati else granu)
53
54 jedan n = if n == 1
55           then True
56           else False
57
58 -- Ogradjene jednačine (guarded equations) - mozemo ih koristiti umesto uslovnih
59 -- izraza, bitan je redosled navodjenja
60
61 sumaPrvih n
62   | n == 0 = 0
```

```
59 | otherwise = n + sumaPrvih(n-1)
```

Zadatak 3.3 Liste. N-torke.

```
1 {-
  Lista je niz vrednosti istog tipa:
3
  [element1,element2,...]
5
  Operacije:
7
  x : lista - dodaje element x na pocetak liste
9  lista1 ++ lista1 - nadovezuje dve liste
  lista !! pozicija - vraca element liste koji se nalazi na poziciji pozicija
11 [a..b] - konstruise listu sa elementima od a do b (u zavisnosti od tipa)
-}
13
lista = [1..5]
15 obrnutaLista = [5,4..1]
17 {-
  Korisne funkcije:
19 head lista - vraca prvi element liste
  tail lista - vraca rep liste
21 length lista - vraca broj elemenata liste
  take n lista - vraca listu prvih n elemenata
23 drop n lista - vraca listu bez prvih n elemenata
  null lista - vraca True ukoliko je lista prazna, False inace
25 -}
27 -- Niske su liste karaktera
29 niska1 = ['A','l','a','d','d','i','n']
  niska2 = "Aladdin"
31
33 {-
  N-torke su nizovi vrednosti razlicitog tipa:
  ('a',1,[1.2, 2.3],"torka")
35
  Funkcije definisane za parove (n=2):
37 fst () - vraca prvi element
  snd () - vraca drugi element
39 -}
```

3.1.2 Zadaci za samostalni rad sa rešenjima

Zadatak 3.4 Napisati sledeće funkcije:

- proizvodPrvih n - računa proizvod prvih n pozitivnih brojeva (rekurzivno, bez korišćenja formule)
- prost n - vraća True ako je broj prost, False inače
- nzd a b - računa najmanji zajednički delilac brojeva a i b (koristiti Euklidov algoritam)
- tipJednacine a b c - vraća tip kvadratne jednačine $a * x^2 + b * x + c = 0$ (”degenerisana”, ”jedno resenje”, ”dva resenja”, ”bez resenja”)
- izDekadne x osn - prebacuje broj x iz dekadne u osnovu osn (pretpostaviti da je $osn > 1$ i $osn < 10$)
- uDekadne x osn - prebacuje broj x iz osnove osn u dekadnu osnovu (pretpostaviti da je $osn > 1$ i $osn < 10$)
- ceoDeo x - računa ceo deo korena broja x (bez korišćenja ugrađenih funkcija za koren i/ili stepen)

[Rešenje 3.4]

Zadatak 3.5 Napisati sledeće funkcije koristeći liste:

- `harm n` - vraća listu prvih `n` elemenata harmonijskog reda
- `delioci n` - vraća listu svih delilaca pozitivnog broja `n`
- `nadovezi lista1 lista2 n` - nadovezuje na `lista1` `n` puta `lista2`

[Rešenje 3.5]

3.1.3 Zadaci za vežbu

Zadatak 3.6 Napisati sledeće funkcije:

- `sumaKvadrata n` - računa sumu kvadrata prvih `n` brojeva (rekurzivno, bez korišćenja formule)
- `brojDelilaca n` - vraća broj delilaca broja `n`
- `fib n` - računa `n`-ti element Fibonačijevog reda
- `osnova x osn1 osn2` - prebacuje broj `x` iz osnove `osn1` u osnovu `osn2` (pretpostaviti da su `osn1` i `osn2` brojevi veći od 1 i manji od 10)

Zadatak 3.7 Napisati sledeće funkcije koristeći liste:

- `parni n` - vraća listu prvih `n` parnih brojeva
- `fibLista n` - vraća listu prvih `n` elemenata Fibonačijevog niza
- `jednocifreniDelioci n` - vraća listu svih jednocifrenih delilaca broja `n`

3.2 Liste

3.2.1 Uvodni primeri

Zadatak 3.8

```
1 {-
2   Uparivanje sablona (pattern matching)
3
4   not :: Bool -> Bool
5   not False = True
6   not True = False
7
8   Ako pozovemo not True uparice se prvi sablon,
9   a ako pozovemo not False uparice se drugi sablon
10
11  Dzoker (wildcard)
12
13  prvi nacin (ako bismo zamenili redosled sablona, javlja pattern match overlapped)
14  (&&) :: Bool -> Bool -> Bool
15  True && True = True
16  _ && _ = False    --- moglo bi i: n && m = False, bitno je samo da su razlicito
17                        imenovani
18
19  drugi nacin, efikasniji:
20  (&&) :: Bool -> Bool -> Bool
21  True && x = x
22  False && _ = False
23
24  _ predstavlja dzoker, tj. sablon koji odgovara bilo kojoj vrednosti (obratiti
25  paznju na redosled definisanja sablona!)
26
27  Sabloni lista:
28
29  length [_] = 1 - [_] predstavlja listu sa jednim elementom
```

```

length x = 1 + length (tail x) - x predstavlja listu
29 -}
31
length1 [] = 0
33 length1 x = 1 + length1 (tail x)

35 -- (x:xs) - predstavlja sablon liste, x je glava, xs je rep

37 length2 [] = 0
length2 (_:xs) = 1 + length2 xs
39

-- preslikava iz liste elemenata u jedan element
41 glava :: [a] -> a
glava (x:_) = x
43

-- preslikava iz liste u listu
45 rep :: [a] -> [a]
rep (_:xs) = xs
47

-- restrikcija na liste elemenata koji pripadaju numerickim tipovima
49 glavNum :: (Num a) => [a] -> a
glavNum (x:_) = x
51

-- Generatori liste (list comprehension)
53

-- even proverava da li je argument paran, a odd da li je neparan
55 -- elementi listi pripadaju skupu od 1 do 50
parni = [x | x<-[1..50], even x]
57 neparni = [x | x<-[1..50], odd x]

59 deljiviPet = [x | x<-[1..50], mod x 5 == 0] -- elementi liste pripadaju skupu od 1 do
50 i deljivi su sa 5

61 kvadrati = [x^2 | x<-[1..10]]

63 -- Mozemo imati vise generatora

65 torke1 = [(x,y) | x<-[1..5], y<-[6..9]]

67 -- obratiti paznju kako utice zamena mesta generatora
torke2 = [(x,y) | y<-[6..9], x<-[1..5]]
69

-- sa negativnim korakom
71 torke3 = [(x,y) | y<-[9,8..6], x<-[1..5]]

73 -- Generatori mogu zavisiti jedni od drugih

75 -- kasniji generatori mogu zavisiti samo od promenljivih koje su uvedene ranijim
generatorima citajuci sa leva na desno
torke4 = [(x,y) | x<-[1..10], y<-[x..10]]
77

{-
79 Sekcija where

81 where ime = vrednost
- imena koja definisemo u where sekciji su vidljiva samo u okviru funkcije
83 u kojoj su definisana
-}

85 bmiIzracunaj tezina visina
87 | bmi <= 18.5 = "Mrsavko"
| bmi <= 25.0 = "Sportista"
89 | bmi > 25.0 = "Bucko"
where bmi = tezina / visina ^ 2

```

3.2.2 Zadaci za samostalni rad sa resenjima

Zadatak 3.9 Napisati sledeće funkcije:

- bezbedanRep lista - ukoliko je lista prazna vraća praznu listu, inače vraća rep liste, kori-

steći: a) uslovne izraze b) ograđene jednačine c) uparivanje šablona

- **savršeni n** - vraća listu savršenih brojeva manjih od n (broj je savršen ako je jednak sumi svojih faktora (tj. delilaca), ne uključujući taj broj)
- **zbirPar n** - vraća listu parova (a,b) takvih da su a i b prirodni brojevi i da je zbir a i b jednak n
- **polovina lista** - deli listu na dve polovine
- **poslednji lista** - vraća poslednji element liste
- **spoji lista** - spaja listu listi u jednu listu
- **sufiksi lista** - vraća listu svih sufiksa liste
- **obrni lista** - obrće listu
- **izbaci k lista** - izbacuje k-ti element iz liste
- **duplira lista** - duplira svaki element iz liste
- **ubaci k x lista** - ubacuje u listu na poziciju k element x
- **izbaciSvaki n lista** - izbacuje svaki n-ti element iz liste (broji se od 1)

[Rešenje 3.9]

3.2.3 Zadaci za vežbu

Zadatak 3.10 Napisati sledeće funkcije:

- **prosti n** - vraća listu prvih n prostih brojeva
- **pitagorineTrojke n** - vraća listu Pitagorinih trojki (a,b,c) takvih da su $a, b, c \leq n$
- **proizvodPar n** - vraća listu parova (a,b) takvih da su a i b prirodni brojevi i da je proizvod a i b manji od n
- **podeli n lista** - deli listu na dve liste, prva lista sadrži prvih n elemenata, a druga lista sadrži ostatak
- **pretposlednji lista** - vraća pretposlednji element liste
- **izbrisiPrvi x lista** - briše prvo pojavljivanje broja x u listi
- **palindrom lista** - ispituje da li je lista palindrom
- **pozNeg lista** - vraća par (a, b) takav da a predstavlja broj True elemenata u listi, a b predstavlja broj False elemenata u listi
- **izmedju a b lista** - vraća listu elemenata koji su veći od a, a manji od b

3.3 Funkcije

3.3.1 Uvodni primeri

Zadatak 3.11

```

1  {-
   Karijeve funkcije (uzimaju jedan po jedan element)
3
   add :: (Int, Int) -> Int
5  Karijeva verzija:
   add1 :: Int -> Int -> Int
7  -}

9  add1 :: Int -> Int -> Int
10 {-
11   -> je desno asocijativna
   add1 :: Int -> (Int -> Int)
13 -}
14 add1 x y = x + y
15
16 {-
17   Primena funkcije je levo asocijativna
19
   add1 x y === ((add1 x) y)
21
   Polimorfne funkcije - sadrže jednu ili više tipskih promenljivih
23
   length1 :: [a] -> Int (a je tipska promenljiva, može da bude bilo kog tipa)
25
   Preopterećene funkcije (overloaded) - sadrže jednu ili više tipskih promenljivih
   koje propadaju nekom tipskom razredu
27
   sum :: Num a => [a] -> a (a mora biti numerickog tipa)
29 -}
30
31 sum1 :: Num a => [a] -> a
32 sum1 [x] = x
33 sum1 (x:xs) = x + sum1 xs
34
35 {-
36   Lambda (anonimne) funkcije
37
   \ x = x + x
39 -}
40
41 add2 = \x -> (\y -> x + y )
42
43 {-
44   Sekcije
45
   Operatore možemo zapisati kao Karijeve funkcije:
46   (+) x y <-> x+y
47   (+2) x <-> x+2
48
49   Generalno ako je op operator
   (op), (x op), (op y) nazivamo sekcijama
51 -}
52
53 duplo x = (*2) x
54
55 {-
56   Funkcija zip pravi listu torki od dve liste
   tako što spaja elemente prve liste sa elementima druge liste
59 -}
60
61 spojeno1 = zip ['a', 'b', 'c'] [1,2,3]
62 spojeno2 = zip ['a', 'b', 'c'] [1,2,3,4,5,6]
63 spojeno3 = zip ['a', 'b', 'c'] [1,2]

```

Zadatak 3.12

```

1  {-
   Funkcije viseg reda
3   -funkcije koje uzimaju funkciju kao argument ili vraćaju funkciju kao rezultat
5
   Funkcija map

```

3 Funkcionalno programiranje

```
7 - kao prvi argument uzima funkciju koju primenjuje na sve elemente liste
  koju uzima kao drugi argument
9 map :: (a -> b) -> [a] -> [b]
11 -}
13 povecaj lista = map (+1) lista
15 {-
  Funkcija filter
17 - kao prvi argument uzima funkciju uslova (funkcija koja vraca logicku vrednost)
  i izdvaja iz liste koju uzima kao drugi argument
19 sve elemente koji zadovoljavaju zadat uslov
21 filter :: (a -> Bool) -> [a] -> [a]
23 -}
25 pozitivni lista = filter (>0) lista
27 {-
  Jos neke funkcije viseg reda za rad sa listama:
29
  any uslov lista - vraca True ako postoji element u listi koji zadovoljava uslov,
  False inace
31 all uslov lista - vraca True ako svi elementi u listi zadovoljavaju uslov, False
  inace
  takeWhile uslov lista - izdvaja jedan po jedan element liste sve dok ne stigne do
  elementa koji ne zadovoljava uslov
33 dropWhile uslov lista - izbacuje jedan po jedan element liste sve dok ne stigne do
  elementa koji ne zadovoljava uslov
  sum lista - sabira elemente liste
35 product lista - mnozi elemente liste
  reverse lista - obrce listu
37 unzip - spaja listu parova u dve liste
  concat lista - spaja liste iz liste listi u jednu listu
39 elem e lista - vraca True ako e postoji u listi, False inace
  replicate n x - kopira broj x n puta
41 -}
43 prviNePozitivni lista = takeWhile (<=0) lista
45 {-
  Funkcija foldr
47 - enkapsulira sledeci sablon rekurzije
49
  f [] = v - f preslikava praznu listu u vrednost v
51 f (x:xs) = x op (f xs) - nepraznu listu preslikava u funkciju op
  primenjenu na glavu liste (x) i f od repa liste (f xs)
53
  Npr:
55 -}
57 sum1 [] = 0 -- v = 0
  sum1 (x:xs) = x + sum1 xs -- op = +
59 sum2 lista = foldr (+) 0 lista
61 {-
  Kompozicija funkcija
63
  (f . g) x <-> f (g x)
65 -}
67 negativneSume liste = map (negate . sum) liste
69 -- negativneSume [[1..5], [6..10], [11..15]]
71 {-
  Cesto nam je potrebno da listu zadamo na sledeci nacin:
73
  lista = [f(x) | x<-D, p(x)]
```

```

75     to mozemo jednostavno uraditi koristeći funkcije map i filter
77     map f(filter p xs)
79 -}
81 povecajPozitivne [] = []
   povecajPozitivne lista = map (+1) (filter (>0) lista)

```

3.3.2 Zadaci za samostalni rad sa rešenjima

Zadatak 3.13 Napisati sledeće funkcije:

- `pozicije x lista` - vraća listu pozicija broja `x` u listi
- `qsort lista` - sortira listu algoritmom `qsort`
- `brisiPonavljanja lista` - briše sva uzastopna ponavljanja elemenata u listi
- `podlistePonavljanja lista` - pakuje sva uzastopna ponavljanja nekog elementa u podlistu tako da rezultat bude lista listi
- `imaDuplikata lista` - ispituje da li u listi postoje duplikati
- `uzastopniParovi lista` - vraća listu uzastopnih parova
- `broj lista` - vraća broj određen ciframa koje se nalaze u listi u obrnutom poretku
- `broj1 lista` - vraća broj određen ciframa koje se nalaze u listi

[Rešenje 3.13]

Zadatak 3.14 Napisati sledeće funkcije (koristiti funkcije `map`, `foldr` ili `filter`):

- `sumaKvNeg lista` - računa sumu kvadrata negativnih elemenata iz liste (koristiti funkciju
- `malaSlova lista` - računa broj malih slova unutar liste
- `sume lista` - vraća listu suma elemenata svake podliste
- `spoji lista` - spaja sve podliste u jednu listu
- `listaUPar lista` - pretvara listu parova u par dve liste, tako da prva lista sadrži sve prve elemente parova a druga sve druge (implementacija funkcije `unzip`)
- `parOdListi lista1 lista2` - pravi listu parova od dve liste, tako da prvi element svakog para bude iz prve liste, a drugi element svakog para bude iz druge liste (implementacija funkcije `zip`)
- `ucesljaj lista1 lista2` - pravi listu naizmeničnim učešljavanjem elemenata listi

[Rešenje 3.14]

3.3.3 Zadaci za vežbu

Zadatak 3.15 Napisati sledeće funkcije:

- `paroviPonavljanja lista` - pravi listu parova (x,k) takvih da je `x` element iz liste, a `k` broj koliko se puta `x` uzastopno pojavljuje u listi
- `sortirana lista` - ispituje da li je lista sortirana
- `permutacija lista1 lista2` - ispituje da li je `lista1` permutacija `lista2`
- `izdvoji a b lista` - izdvaja elemente liste od pozicije `a` do pozicije `b`

- izbrisi x lista - brise sva pojavljivanja broja x u listi
- ponovi n lista - svaki element iz liste ponavlja n puta

Zadatak 3.16 Napisati sledeće funkcije (koristiti funkcije map, foldr ili filter):

- brojPoz lista - računa broj pozitivnih elemenata liste
- eliminišiProste lista - eliminiše sve proste brojeve iz listi
- saberiAbs lista - sabira apsolutne vrednosti elemenata iz liste
- nadoveziProizvod lista - svakoj podlisti nadovezuje proizvod njenih elemenata
- prosek lista - računa prosek liste
- brojPojavljivanja x lista - vraća broj pojavljivanja broja x u listi

3.4 Rešenja

Rešenje 3.4

```
1  -- Pretpostavljamo da funkciju koristimo samo za n>=1
   proizvodPrvih n
3     | n == 1 = 1
     | otherwise = n * proizvodPrvih (n-1)
5
6  -- Ispitujemo da li je broj prost tako sto pozovemo pomocnu funkciju
7  -- koja ispituje da li postoji neki broj pocev od 2 do n koji deli broj n
   prost n = prost1 n 2
9
10 -- Proveravamo da li postoji broj koji deli n (pocev od broja 2 - poziv iz prethodne
     funkcije)
11 -- ukoliko je k == n to znaci da smo proverili za svaki broj od 2 do n-1
    -- i ustanovili da nijedan od njih ne deli n tako da vracamo True kao indikator da
        broj n jeste prost
13 -- ukoliko je n deljivo sa k, vracamo False (n nije prost)
    -- ukoliko n nije deljivo sa k, pozovemo funkciju rekurzivno za sledeci broj (k+1)
15 prost1 n k
     | n == k = True
     | n `mod` k == 0 = False
     | otherwise = prost1 n (k+1)
17
19 nzd a b
21 | b == 0 = a
   | otherwise = nzd b (a `mod` b)
23
24 tipJednacine a b c
25 | a == 0 = "Degenerisana"
   | (b*b - 4*a*c) == 0 = "Jedno resenje"
   | (b*b - 4*a*c) > 0 = "Dva resenja"
   | otherwise = "Nema resenja"
27
29 uDekadnu x osn = if x==0 then 0 else uDekadnu (x `div` 10) osn * osn + (mod x 10)
31 izDekadne x osn = if x==0 then 0 else izDekadne (x `div` osn) osn * 10 + (mod x osn)
33
34 -- Trazimo ceo deo korena broja x
35 -- trazimo prvi broj (pocev od 1) ciji je kvadrat veci od kvadrata naseg broja x
    -- i kao rezultat vracamo broj koji je za jedan manji
37 ceoDeo x = ceoDeo1 x 1
39
40 ceoDeo1 x i
     | (i*i) > x = (i-1)
     | otherwise = ceoDeo1 x (i+1)
41
```

Rešenje 3.5

```

1 -- Pravimo listu prvih n elemenata harmonijskog reda (n>0)
2 -- ukoliko je n = 1, vracamo listu koja sadrzi prvi element harmonijskog reda
3 -- inace pozovemo funkciju rekurzivno za n-1
4 -- i na rezultat nadovezemo reciprocnu vrednost broja n
5 harm n
6   | n == 1 = [1.0]
7   | otherwise = harm (n-1) ++ [recip n]
8
9 -- Pravimo listu delioca broja n
10 -- ukoliko je n = 1, vracamo listu koja sadrzi samo 1
11 -- inace pozivamo pomocnu funkciju sa jos jednim parametrom
12 -- koji nam govori do kog potencijalnog delioca smo stigli
13 delioci n
14   | n == 1 = [1]
15   | otherwise = delioci1 n 2
16
17 -- Pravimo listu delioca broja n pocev od broja k
18 -- ukoliko smo stigli do broja n (k==n) vracamo praznu listu
19 -- inace proveravamo da li je k delioc broja n
20 -- ako jeste, stavljamo ga u listu i pozivamo funkciju rekurzivno
21 -- za k+1 (sledeci potencijalni delilac)
22 -- inace samo pozivamo funkciju rekurzivno za k+1
23 delioci1 n k
24   | k == n = []
25   | n `mod` k == 0 = [k] ++ (delioci1 n (k+1))
26   | otherwise = delioci1 n (k+1)
27
28 -- Nadovezujemo listu2 na listu1 n puta
29 -- ukoliko je n == 1 na listu1 nadovezemo listu2 operatorom ++
30 -- inace pozovemo funkciju rekurzivno za n-1 i na rezultat nadovezemo listu2
31 nadovezi lista1 lista2 n
32   | n == 1 = lista1 ++ lista2
33   | otherwise = (nadovezi lista1 lista2 (n-1)) ++ lista2

```

Rešenje 3.9

```

1 bezbedanRep1 lista = if lista == [] then [] else tail lista
2
3 bezbedanRep2 lista
4   | lista == [] = []
5   | otherwise = tail lista
6
7 bezbedanRep3 [] = []
8 bezbedanRep3 (x:xs) = xs
9 -- bezbedanRep3 lista = tail lista
10
11 savrseni n = [x | x<-[1..n], sum(faktori x) == x]
12 faktori x = [i | i<-[1..x-1], x `mod` i ==0]
13
14 zbirPar n = [(a,b) | a<-[1..n], b<-[1..n], a+b==n]
15 -- drugi, efikasniji nacin
16 zbirPar2 n = [(a,b) | a<-[1..n], b<-[n-a], b/=0]
17
18 polovina [] = ([],[ ])
19 polovina lista = (take n lista, drop n lista)
20   where n = length lista `div` 2
21
22 poslednji lista = lista !! poz
23   where poz = length lista - 1 -- moze i: length(tail lista) -1
24
25 spoji [] = [] -- nije neophodno, prolazi drugi sablon i za praznu listu
26 spoji lista = [x | podlista <- lista, x <- podlista]
27
28 sufiksi [] = [[]]
29 -- lista je sama svoj sufiks, a ostali sufiksi su sufiksi njenog repa
30 sufiksi (x:xs) = (x:xs) : sufiksi xs
31
32 obrni [] = []
33 obrni (x:xs) = obrni xs ++ [x]
34

```

```

izbaci _ [] = []
36 izbaci 0 (x:xs) = xs
izbaci k (x:xs) = x : (izbaci (k-1) xs)
38
duplira [] = []
40 duplira (x:xs) = [x,x] ++ duplira xs

42 ubaci 0 n lista = n : lista
ubaci k n [] = [n]
44 ubaci k n (x:xs) = x : (ubaci (k-1) n xs)

46 izbaciSvaki n [] = []
izbaciSvaki n lista = take (n-1) lista ++ izbaciSvaki n (drop n lista)

```

Rešenje 3.13

```

-- Racunamo sve pozicije broja x u listi
2 -- tako sto spajamo listu sa listom brojeva od 0 do n
-- i od liste parova (broj, pozicija)
4 -- izdvajamo one pozicije kod kojih je broj = x
pozicije :: Eq a => a -> [a] -> [Int]
6 pozicije x [] = []
pozicije x lista = [i | (x1,i) <- zip lista [0..n], x == x1]
8     where n = length lista - 1

10 qsort :: Ord a => [a] -> [a]
qsort [] = []
12 qsort (x:xs) = qsort manji ++ [x] ++ qsort veci
     where manji = [a | a<-xs, a <= x]
           veci = [b | b<-xs, b > x]

14
16 -- Pravimo listu bez uzastopnih ponavljanja
-- tako na glavu nadovezemo rezultat rekurzivnog poziva funkcije nad korigovanim
   repom
18 -- sa pocetka repa izbacujemo sve elemente koji su jednaki glavi
-- [1,1,1,1,1,2]
20 -- 1 : brojPonavljanja [2]
-- [1,2]
22 brisiPonavljanja :: Eq a => [a] -> [a]
brisiPonavljanja [] = []
24 brisiPonavljanja (x:xs) = x : brisiPonavljanja(dropWhile (==x) xs)

26 -- Pravimo listu koja sadrzi sva uzastopna pojavljivanja elemenata u posebnim listama
-- [1,1,1,2,2,3] -> [[1,1,1], [2,2], [3]]
28 -- tako sto pravimo listu uzastopnih pojavljivanja glave
-- i pozivamo funkciju rekurzivno pocev od elementa koji nije jednak glavi
30 -- [[1,1,1], podlistePonavljanja [2,2,3]]
-- [[1,1,1], [2,2], podlistePonavljanja [3]]
32 -- [[1,1,1], [2,2], [3], podlistePonavljanja []]
podlistePonavljanja :: Eq a => [a] -> [[a]]
34 podlistePonavljanja [] = []
podlistePonavljanja (x:xs) = (x : (takeWhile (==x) xs)) : podlistePonavljanja(
   dropWhile (==x) xs)
36

-- Ispitujemo da li lista sadrzi duplikate
38 -- ukoliko je lista prazna vracamo False
-- inace, ukoliko se glava nalazi jos negde u repu vracamo True
-- odnosno pozivamo funkciju rekurzivno za rep
40 imaDuplikata :: Eq a => [a] -> Bool
42 imaDuplikata [] = False
imaDuplikata (x:xs) = elem x xs || imaDuplikata xs
44

-- Pravimo listu uzastopnih parova tako sto spajamo svaki element liste
46 -- sa elementima iz repa
-- [1,2,3,4] [2,3,4] -> [(1,2), (2,3), (3,4)]
48 uzastopniParovi :: [a] -> [(a,a)]
uzastopniParovi [] = []
50 uzastopniParovi lista = zip lista (tail lista)

52 -- [1,2,3] -> 321
broj :: Num a => [a] -> a
54 broj [] = 0

```

```

broj (x:xs) = (broj xs)*10 + x
56 -- [1,2,3] -> 123
58 broj [] = 0
broj1 (x:xs) = x*10^(length xs) + broj xs
60 -- Drugi nacin koriscenjem funkcije broj
62 -- broj1 lista = broj (reverse lista)

```

Rešenje 3.14

```

-- Racunamo sumu kvadrata negativnih elemenata liste
2 -- tako sto izdvajamo negativne elemente funkcijom filter u listu negativni
-- i primenjujemo operaciju kvadriranja nad svim elementima liste negativni
4 -- koristeći funkciju map i rezultat nazivamo kvadrati
-- nakon toga sabiramo elemente liste kvadrati u akumulator cija je pocetna vrednost
= 0
6 -- Podsetnik: celobrojni stepen - ^ , realni stepen - ^^
sumaKvNeg1 :: (Ord a, Fractional a) => [a] -> a
8 sumaKvNeg1 [] = 0
sumaKvNeg1 lista = foldr (+) 0 kvadrati
10   where kvadrati = map (^2) negativni
         negativni = filter (<0) lista
12
-- Drugi nacin: pravimo kompoziciju funkcija filter, map i fold
14 sumaKvNeg2 :: [Double] -> Double
sumaKvNeg2 [] = 0
16 sumaKvNeg2 lista = suma lista
suma = (foldr (+) 0) . (map (^2)) . (filter (<0))
18
-- Racunamo broj malih slova u listi tako sto primenjujemo kompoziciju funkcija
20 -- filter - filtriramo karaktere tako da nam ostanu samo mala slova
-- i nakon toga racunamo duzinu rezultujuce liste - funkcija length
22 malaSlova :: [Char] -> Int
malaSlova [] = 0
24 malaSlova lista = (length . (filter (\x -> if x>='a' && x<='z' then True else False)
)) lista
26
-- Pravimo listu suma svih elemenata podliste
-- tako sto primenjujemo funkciju sum nad svakom podlistom koristeći funkciju map
28 sume :: Num a => [[a]] -> [a]
sume [] = []
30 sume lista = map (sum) lista
32
-- Spajamo podliste u jednu listu
-- tako sto nadovezujemo svaku podlistu na akumulator cija je pocetna vrednost
jednaka []
34 -- koristeći funkciju foldr
-- [[1,1], [2,2,2], [3,3,3]] - []
36 -- [[2,2,2], [3,3,3]] - [1,1]
-- [[3,3,3]] - [1,1,2,2,2]
38 -- [] - [1,1,2,2,2,3,3,3]
spoji :: [[a]] -> [a]
40 spoji [] = []
spoji lista = foldr (++) [] lista
42
-- Pravimo par listi od liste parova kao sto radi funkcija unzip
44 -- [(1,2), (1,2), (1,2)] -> ([1,1,1], [2,2,2])
-- tako sto svaki par iz liste dodajemo u akumulator cija je pocetna vrednost par
praznih listi
46 -- [(1,2), (1,2), (1,2)] - ([],[])
-- [(1,2), (1,2)] - ([1],[2])
48 -- [(1,2)] - ([1,1],[2,2])
-- [] - ([1,1,1],[2,2,2])
50 listaUPar :: [(a,b)] -> ([a],[b])
listaUPar [] = ([],[])
52 listaUPar lista = foldr (\ (a,b) (c,d) -> (a:c, b:d)) ([],[]) lista
54 {- Drugi nacin bez foldr:
listaUPar [] = ([],[])
56 listaUPar ((a,b):xs) = (a:l1, b:l2)
   where

```

```
58     l1 = fst rep
59     l2 = snd rep
60     rep = listaUPar xs
61 -}
62
63 -- Implementacija funkcije zip
64 -- [1,2,3] [4,5,6] -> [(1,4), (2,5), (3,6)]
65 -- Da bi radila po duzini krace liste neophodno je dodati sablone za takve situacije
66 parOdListi [] _ = []
67 parOdListi _ [] = []
68 -- Od glava listi pravimo par koji dodajemo na pocetak rezultujuce liste za rep
69 parOdListi (x:xs) (y:ys) = ( (x, y) : (parOdListi xs ys) )
70
71 -- [1,1,1] [2,2,2] -> [1,2,1,2,1,2]
72 -- [1,1] [2,2,2,2] -> [1,2,1,2,2,2]
73 -- Treba obuhvatiti slucajeve listi razlicitih duzina
74 -- Sledeca linija koda ne radi jer nije dozvoljeno koriscenje dzoker promenljive u
75 -- izrazu kojim definisete vrednost funkcije
76 -- ucesljaj [] _ = _
77 ucesljaj [] lista = lista
78 ucesljaj lista [] = lista
79 ucesljaj (x:xs) (y:ys) = [x]++[y]++(ucesljaj xs ys)
```

4

Konkurentno programiranje

4.1 Scala

Koristicemo jezik Scala-u (verzija 2.11) <http://www.scala-lang.org/>.
Potrebno je imati instalirano:

- (a) Jezik Scala-u verziju 2.11
- (b) Eclipse Neon razvojno okruzenje <https://www.eclipse.org/downloads/>
- (c) ScalaIDE dodatak za Eclipse razvojno okruzenje <http://scala-ide.org/>

Korisni linkovi:

<https://www.scala-lang.org/documentation/>
<http://docs.scala-lang.org/tutorials/scala-for-java-programmers.html> <http://scalatutorials.com/>

4.1.1 Uvod u jezik Scala

Zadatak 4.1 HelloWorld

```
/**
2  * Viselinijske komentare u Scali pisemo ovako (isto kao i u C-u).
3  *
4  * */
6 // Jednolinijske komentare pisemo ovako
8 object HelloWorld {
9     def main(args: Array[String]){
10 // Tacka zapeta na kraju svake naredbe je opciona
11     println("Hello world! :)")
12     }
13 }
```

Zadatak 4.2 Promenljive, niske, petlje, funkcije

```
1 object Uvod {
2     def main(args: Array[String]){
3
4         println("----- Promenljive -----")
5         // Promenljive mozemo deklarirati kao konstantne
6         // ili promenljive cija se vrednost moze menjati.
7         // Kljucna rec val deklarise konstantu a var promenljivu.
8         //
9         // kljucnaRec imePromenljive : tipPromenljive = vrednost
10        //
11        val c : Int = 42
12        var x : Int = 6
13    }
```

```
15     x += 6
17 //   Naredba:
18 //
19 //   c += 4
20 //
21 //   prijavljuje gresku.
22
23     println("Konstanta: " + c)
24     println("Promenljiva: " + x)
25
26     if(x > 6)
27         println("Promenljiva x je veca od 6.")
28     else if(x == 6)
29         println("Promenljiva x je jednaka 6.")
30     else
31         println("Promenljiva x je manja od 6.")
32
33     println("----- Niske -----")
34
35     val crtani : String = "Maza i Lunja"
36     println("Duzina niske: '" + crtani + "' je " + crtani.length())
37
38 //   Viselinijske niske pisemo izmedju trostrukih navodnika:
39     println(""" Likovi:
40             -Alisa
41             -Vendi
42             -Bambi
43             -Had
44             """)
45 //   Interpolacija niski
46
47 //   Scala nam dozvoljava da ugnjezdavamo vrednosti promenljivih u niske
48 //   dodavanjem karaktera s na pocetak niske (npr: s"Moja niska") omogucavamo
49 //   ugnjezdavanje vrednosti promenljivih u nisku dodavajuci karakter $ pre imena
50 //   promenljive
51 //   (npr s"Kolicina: $kol grama")
52
53     var trajanje : Int = 76
54     println(s"Crtani film: '$crtani' - $trajanje minuta.")
55
56     println("----- Petlje -----")
57
58     var sat = 0
59
60     while(trajanje >= 60){
61         sat += 1
62         trajanje -= 60
63     }
64
65     println(s"'$crtani' traju $sat sat i $trajanje minuta.")
66
67 //   For petljom mozemo iterirati kroz kolekcije koristeći sintaksu
68 //
69 //   for(element <- kolekcija)
70 //
71 //   1 to 5 - pravi kolekciju brojeva [1,5]
72
73     println("FOR - 1 to 5 ")
74     for(i <- 1 to 5)
75         println(i)
76
77 //   1 until 5 - pravi kolekciju brojeva [1,5)
78
79     println("FOR - 1 until 5 ")
80     for(i <- 1 until 5)
81         println(i)
82
83 //   Range(pocetak, kraj, korak) - pravi kolekciju [pocetak, kraj) sa zadatik
84 //   korakom.
85 //   Korak moze biti i negativan npr. Range(10,0,-2)
```

```

87     println("FOR - Range(0,6,2) ")
      for(i <- Range(0,6,2))
        println(i)
89
91     println("----- Nizovi -----")
    // Pravimo ih na sledeci nacin:
93     // var niz : Array[tip] = new Array[tip](brojElemenata)
      var crtaci : Array[String] = new Array[String](5)
95     crtaci(0) = "Petar Pan"
      crtaci(1) = "Mulan"
97     crtaci(2) = "Aladdin"
      crtaci(3) = "Herkules"
99     crtaci(4) = "Pocahontas"
101
      println("Crtaci: ")
      for(crtac <- crtaci)
103         println(crtac)
105
      println("----- Funkcije -----")
      ispisiSortirano(crtaci)
107  }
109 // def imeFunkcije([listaArgumenata]) : povratnaVrednost = {
    //     teloFunkcije
111 // }
113 // Povratna vrednost Unit je ekvivalentna void vrednosti
    def ispisiSortirano(crtaci : Array[String]) : Unit = {
115     println("Sortirani crtaci:")
      for(crt <- crtaci.sortWith(poredi))
117         println(crt)
119 // Anonimne funkcije
    //
121 // (listaArgumenata) => {teloFunkcije}
123
      println("Sortirani crtaci koristeći anonimnu funkciju:")
      for(crt <- crtaci.sortWith((c1 , c2) => { if(c1.compareTo(c2) < 0) false else
125 true} ))
        println(crt)
127  }
129
    def poredi(c1 : String, c2 : String) : Boolean = {
      if(c1.compareTo(c2) < 0)
131         return true
      else
133         return false
    }
135 }

```

Zadatak 4.3 Klase, objekti, nasleđjivanje

```

2 // Klase se u Scali konstruisu na sledeci nacin:
  //
4 // class ImeKlase {
  //     teloKlase
6 // }
  // ili
8 // class ImeKlase (argumentiKonstruktor) {
  //     teloKlase
10 // }
  //
12
    class Film {
14     var naslov : String = ""
      var trajanje : Int = 0
16     var godina : Int = 0
18 // Konstruktor
      def this(nas : String, traj : Int, god : Int) = {

```



```
20     this()
21     this.naslov = nas
22     this.trajanje = traj
23     this.godina = god
24 }
25
26 // Metodi klase
27
28 def getNaslov() : String = {
29     return this.naslov
30 }
31
32 def getTrajanje() : Int = {
33     return this.trajanje
34 }
35
36 def getGodina() : Int = {
37     return this.godina
38 }
39
40 override def toString() : String = {
41     return "Film " + this.naslov + ", traje " + this.trajanje + " minuta, napravljen
42     je " + this.godina + " godine"
43 }
44 }
45
46 // Nasledjivanje
47 class CrtaniFilm extends Film {
48     var animator : String = ""
49
50     def this(nas : String, traj : Int, god : Int, anim : String) = {
51         this()
52         this.naslov = nas
53         this.trajanje = traj
54         this.godina = god
55         this.animator = anim
56     }
57
58     def getAnimator() : String = {
59         return this.animator
60     }
61
62     override def toString() : String = {
63         return "Crtani " + super.toString() + ", animator je " + this.animator
64     }
65 }
66
67 // U Scali mozemo definisati takozvane 'singleton' objekte kljucnom reci object.
68 // Garantuje se da ce postojati tacno jedan 'singleton' objekat na nivou naseg
69 // programa
70 // i on se najcesce koristi za implementaciju main metoda
71 object Program {
72     def main(args: Array[String]) {
73
74         val assassinsCreed = new Film("Assassin's Creed", 115, 2016)
75         val tarzan = new CrtaniFilm("Tarzan", 88 , 1999, "Walt Disney")
76
77         println(assassinsCreed)
78         println(tarzan)
79
80         // Scala nudi brojne korisne kontejnerske i nizovske klase
81         // kao sto su Array, ArrayBuffer, Map, HashMap, Queue, Tuple1, Tuple2, i druge
82         // sa kojima cemo se susretati u narednim primerima
83         // gde ce biti vise objasnjene
84     }
85 }
```

4.1.2 Zadaci

Zadatak 4.4 Restoran Pet konobara radi u restoranu, i nakon što dođu na posao, poslovođa im dodeli broj stolova koje moraju da usluže. Napisati program koji učitava sa standardnog ulaza broj neusluženih stolova u restoranu i raspoređuje ih konobarima. Nakon što usluže jedan sto, konobari šalju poruku tako da poslovođa u svakom trenutku ima uvid u brojčano stanje. Konobare implementirati kao posebne niti koje uslužuju sto (spavaju nasumično izabran broj sekundi), ispisuju redni broj stola koji su uslužili i po završetku ispisuju poruku da su završili.

[Rešenje 4.4]

Zadatak 4.5 Nastavno osoblje Koristeći biblioteku `HtmlCleaner` <http://htmlcleaner.sourceforge.net/index.php> napisati program koji dohvata imena, prezimena i email naloge nastavnog osoblja Matematičkog fakulteta sa sledećih stranica:

- <http://www.matf.bg.ac.rs/redovni/>
- <http://www.matf.bg.ac.rs/vanredni/>
- <http://www.matf.bg.ac.rs/docenti/>
- <http://www.matf.bg.ac.rs/asistenti/>
- <http://www.matf.bg.ac.rs/saradnici/>

i upisuje ih u odgovarajuće datoteke. Svaku stranicu obraditi u posebnoj niti.

[Rešenje 4.5]

Zadatak 4.6 Množenje matrica Napisati program koji konkurentno množi dve matrice čiji se elementi nalaze u datotekama `matrica1.txt` i `matrica2.txt` i rezultat upisuje u datoteku `matrica3.txt`. Svaka nit treba da računa vrednosti za jednu vrstu rezultujuće matrice. Format podataka u datotekama je sledeći:

```

1  n m
2  a11 a12 a13 ... a1m
3  a21 a22 a21 ... a2m
4  ...
5  an1 an2 an3 ... anm

```

[Rešenje 4.6]

Zadatak 4.7 Broj karaktera DNK Napisati program koji konkurentno prebrojava koliko puta se koja baza (A, C, G, T) pojavljuje u DNK sekvenci koja se nalazi u višelinijskoj datoteci `bio_podaci.txt`. Sa standardnog ulaza učitati broj niti. Svakoj niti dodeliti određen broj linija koji će da obrađuje a rezultate čuvati u deljenoj mapi (pogodno je koristiti klasu `ConcurrentHashMap`).

[Rešenje 4.7]

Zadatak 4.8 Krediti Bogoljub otvara banku i poseduje određeni kapital. Kao svaki dobar ekonomista on odlučuje da zaradi na davanju kredita. Napisati program koji demonstrira rad službenica i davanje kredita u Bogoljubovoj banci. Sa standardnog ulaza učitati njegov početni kapital, kamatnu stopu i broj zaposlenih službenica. U datoteci `red_klijenata.txt` se nalazi spisak klijenata koji čekaju na red za razgovor sa službenicom u sledećem formatu:

```

1  imeKlijenta potrebnaPozajmica

```

Posao svake službenice se izvršava u posebnoj niti. Službenica poziva sledećeg klijenta iz reda na razgovor. Nakon što ustanovi koliko je novca klijentu potrebno, službenica proverava da li banka trenutno poseduje tu količinu novca i daje klijentu kredit tako što umanjuje kapital banke i klijentu računa vrednost duga u skladu sa kamatom. Ukoliko banka nije u mogućnosti da izda kredit, ispisati odgovarajuću poruku. Nakon što službenice završe sa svim klijentima iz reda ispisati poruku o ukupnoj zaradi banke.

[Rešenje 4.8]

Zadatak 4.9 Kladionica Napisati program koji simulira proces kladjenja. Kladioničari se klade na ishod pet fudbalskih utakmica uplaćujući određenu količinu novca na tiket. U datoteci `kladionici.txt` se nalazi spisak kladioničara i njihovih tiketa u sledećem formatu:

```
1 imeKladionicara svotaNovca
2 utakmica1
3 rezultat
4 ...
5 utakmica5
6 rezultat
7 ...
```

Ishod može biti 1 - prva ekipa je pobedila, x - nerešeno, 2 - druga ekipa je pobedila. Svaki ishod nosi određenu kvotu kojom se množi novac koji je kladioničar uplatio. U datoteci `utakmice.txt` se nalazi spisak utakmica sa njihovim kvotama u formatu:

```
1 imeUtakmice
2 kvota1 kvotaX kvota2
3 ...
```

Učitati podatke o utakmicama i kladioničarima. Svaki kladioničar čeka na ishod utakmica u posebnoj niti. Kladionica nakon što dobije ishod utakmica (implementirati nasumično biranje ishoda) obaveštava kladioničare da su utakmice završene. Kladioničari nakon toga računaju sopstvenu zaradu kao zbir kvota pogođenih utakmica pomnoženih sa odgovarajućim delom novca i ispisuju poruku o pogodjenom rezultatu. Na kraju ispisati ukupnu količinu novca koju kladionica treba da isplati kladioničarima.

[Rešenje 4.9]

4.1.3 Zadaci za vežbu sa rešenjima

Zadatak 4.10 Zbir vektora Napisati program koji konkurentno sabira dva vektora. Sa standardnog ulaza se učitava dimenzija vektora, elementi oba vektora i broj niti. Svakoj niti dodeliti indeks početka i kraja vektora nad kojim računa zbir a rezultat smeštati u prvi vektor. Indekse računati na osnovu dimenzije vektora i broja niti. Rezultujući vektor ispisati na standardni izlaz.

[Rešenje 4.10]

Zadatak 4.11 Broj petocifrenih brojeva Napisati program koji konkurentno računa broj pojavljivanja petocifrenih brojeva koji se nalaze u datotekama čija imena se učitavaju sa standardnog ulaza. Svakoj niti dodeliti po jednu datoteku nad kojom će računati. Rezultate brojanja smeštati u lokalne promenljive unutar niti i po završetku računanja za svaku datoteku ispisati broj pojavljivanja petocifrenih brojeva.

[Rešenje 4.11]

Zadatak 4.12 Berba Milovan ima voćnjak u kome gaji trešnje, kajsije, kruške i šljive. Došlo je vreme berbe i mora uposliti mlade studente da obru voćnjak. U datoteci `drvoredi.txt` se nalaze podaci o drvoredima voćnjaka u sledećem formatu:

```
1 vrstaVoća brojStabala
```

Sa standardnog ulaza učitati broj zaposlenih studenata a iz datoteke učitati podatke o drvoredima. Svaka nit predstavlja jednog studenta. Student odlazi do jednog drvoreda skidajući ga iz reda drvoreda koje je potrebno obrati i počinje branje. Ako pretpostavimo da jedno stablo voća može roditi od 30-50 kilograma voća, student za svako stablo iz drvoreda nasumično računa broj kilograma voća koji je obran i dodaje ih u skladište. Ukoliko su svi drvoredi obrani studenti prestaju sa radom i na standardni izlaz se ispisuje ukupna količina obranog voća svake vrste.

[Rešenje 4.12]

Zadatak 4.13 Turistička agencija Turistička agencija FlyProgrammer organizuje nagradnu igru i daje pet vaučera od 20% popusta na cenu kupljenje karte svojim klijentima. U datoteci `ucesnici.txt` se nalaze podaci o klijentima i cenama u sledećem formatu:

```
1 ime prezime
2 cena
```

Napisati program koji simulira nagradnu igru. Svaka nit čeka na rezultate nagradne igre za jednog klijenta. Turistička agencija izvlači dobitnike slučajnom selekcijom i nakon završenog izvlačenja obaveštava niti. Niti proveravaju da li je izvučen odgovarajući klijent i ispisuje poruku o ishodu.

[Rešenje 4.13]

4.1.4 Zadaci za vežbu

Zadatak 4.14 Množenje vektora skalarom Napisati program koji konkurentno množi vektor skalarom. Sa standardnog ulaza učitati dimenziju i elemente vektora, skalar i broj niti. Svakoj niti dodeliti deo vektora (indekse početnog i krajnjeg elementa). Nit računa proizvod vektora skalarom za elemente u zadatom opsegu i rezultat smešta u isti vektor. Na kraju ispisati rezultat na standardni izlaz.

Zadatak 4.15 Množenje matrice vektorom Napisati program koji konkurentno množi matricu vektorom. Sa standardnog ulaza učitati dimenziju i elemente vektora, dimenzije i elemente matrice i broj niti. Svakoj niti dodeliti deo matrice (indekse početne i krajnje vrste). Nit računa proizvod matrice vektorom za vrste u zadatom opsegu i rezultat smešta u nov vektor. Na kraju ispisati rezultat na standardni izlaz.

Zadatak 4.16 Transponovanje matrice Napisati program koji konkurentno transponuje matricu. U datoteci `matrica.txt` se nalaze dimenzije matrice nakon čega slede elementi matrice. Svaka nit transponuje po jednu vrstu matrice i smešta rezultat u rezultujuću matricu. Na kraju izračunavanja rezultujuću matricu upisati u datoteku `transponovana_matrica.txt`.

Zadatak 4.17 Matrice (dodatak) Implementirati zadatke 4.6 i 4.16 tako da se broj niti učitava sa standardnog ulaza, i svakoj niti dodeliti broj vrsta koje će obrađivati.

Zadatak 4.18 Seminari konkursi projekti Koristeći biblioteku `HtmlCleaner` napisati program koji konkurentno dohvata podatke o seminarima, konkursima i projektima Matematičkog fakulteta sa sledećih stranica:

- <http://www.matf.bg.ac.rs/nauka-konkursi/>
- <http://www.matf.bg.ac.rs/seminari/>
- <http://www.matf.bg.ac.rs/projekti/>

i upisuje ih u odgovarajuće datoteke. Svaku stranicu obraditi u posebnoj niti.

Zadatak 4.19 Funkcije nad vektorima Napisati program koji konkurentno računa proizvod, sumu, prosečnu vrednost, broj negativnih i broj pozitivnih elemenata vektora. Sa standardnog ulaza učitavati imena datoteka u kojima se nalaze vektori. Implementirati konkurentno računanje na dva načina, koristeći paralelizaciju zadataka i paralelizaciju podataka i uporediti vremena izvršavanja (obratiti pažnju na to da je korišćenjem paralelizacije zadataka najveće moguće ubrzanje jednako najsporijem zadatku dok kod paralelizacije podataka ubrzanje zavisi od broja procesora, broja niti,...). Na kraju ispisati rezultate svih izračunavanja na standardni izlaz.

Zadatak 4.20 Broj petocifrenih brojeva (dodatak) Implementirati zadatak 4.11 uz sledeće izmene. Brojeve učitati iz jedne datoteke `brojevi.txt` u niz a broj niti učitati sa standardnog ulaza. Svakoj niti dodeliti deo niza brojeva koji će obrađivati a rezultat čuvati u deljenoj promenljivoj tipa `AtomicLong`.

Zadatak 4.21 Broj karaktera DNK (dodatak) Implementirati zadatak 4.7 tako da se rezultati brojanja baza čuvaju u deljenim promenljivama tipa `AtomicLong` i na kraju ispisati procenat pojavljivanja svake baze u lancu.

Zadatak 4.22 Hotel Hotel Nightlife brine o svojim klijentima i trudi se da im smanji vreme čekanja dok se sređuje soba koju su rezervisali. U datoteci `sobe.txt` nalaze se brojevi soba koje je potrebno srediti. Sa standardnog ulaza učitati broj trenutno dostupnih čistačica. Svaka nit predstavlja jednu čistačicu. Čistačica skida sobe sa reda soba koje je potrebno srediti i odlazi do nje da je sredi (ispisati redni broj sobe u koju je ušla čistačica a nit uspavati na slucajan broj sekundi). Nakon što je sredila sobu, čistačica pronalazi ostavljen bakšiš (slučajan broj od 0-500 dinara) i po dogovoru bakšiš ubacuje u zajedničku kutiju. Kada završi sa jednom sobom, čistačica odlazi do sledeće sobe (skida je sa reda) i nastavlja sve dok red ne postane prazan. Nakon što završe sa čišćenjem, čistačice otvaraju kutiju sa bakšišem i dele novac na ravne časti tj. ispisuju na standardan izlaz dobijeni bakšiš.

Zadatak 4.23 Loto Napisati program koji simulira izvlačenje na Loto-u. Izvlače se tri broja iz opsega [1,37] a ukupna vrednost nagradnog fonda se unosi sa standardnog ulaza. Učesnici uplaćuju srećke i pokušavaju da pogode tri broja. U datoteci `ucesnici.txt` se nalaze informacije o uplaćenim srećkama u sledećem formatu:

```
1 ime
2 broj1 broj2 broj3
3 ...
```

Učitati podatke o učesnicima. Svaki učesnik čeka na kraj izvlačenja u posebnoj niti. Izvlačenje se odvija u glavnoj niti tako što se nasumično biraju tri broja iz opsega [1,37] i po završetku se obaveštavaju niti učesnika. Niti učesnika nakon toga porede izvučene brojeve i ukoliko se sva tri poklapaju ispisuju poruku i u ukupnu sumu novca koji je potrebno isplatiti dodaju vrednost nagradnog fonda. Ukoliko se dva od tri broja poklapaju u ukupnu sumu se dodaje 40% nagradnog fonda. Ukoliko se samo jedan broj poklapa u ukupnu sumu se dodaje 10% nagradnog fonda. Na kraju ispisati količinu novca koju je potrebno isplatiti i količinu novca koju je lutrija zaradila (ukoliko zarada postoji).

4.2 Rešenja

Rešenje 4.4 Restoran

```
1 import java.util.concurrent._
import java.util.Scanner
3
4 // Pravljenje niti
5 //
6 // Da bismo napravili nit potrebno je da definisemo klasu koja nasledjuje klasu
   Thread
7 // i implementiramo metod run cije izvršavanje pocinje kada nad instancom nase klase
   // pozovemo metod start.
8 //
9 // Drugi nacin je da nasa klasa implementira interfejs Runnable
10 // i implementira metod run.
11 // Medjutim, da bismo naglasili da zelimo da se metod run nase instance izvršava kao
   posebna nit
12 // potrebno je da napravimo instancu tipa Thread kojoj u konstruktoru treba da
   prosledimo instancu nase klase
13 // (koja implementira metod run.
14 //
15 //
16
17 class Konobar(ime : String, brStolova : Int) extends Thread {
   override def run(){
18     for(i <- 0 until brStolova){
19       // Klasa ThreadLocalRandom predstavlja generator slucajnih brojeva
20       // jedinstven na nivou jedne niti.
21       // Metod current() vraca objekat ove klase za trenutnu nit.
22       Thread.sleep(ThreadLocalRandom.current().nextInt(1,10)*1000)
23       println("Konobar " + ime + " je uslužio " + i + ". sto.")
24     }
25     println("Konobar " + ime + " je završio sa posluživanjem.")
26   }
27 }
```

```

29 object Restoran {
31   def main(args : Array[String]) {
32     val sc : Scanner = new Scanner(System.in)
33
34     println("Unesite broj neusluzenih stolova u restoranu: ")
35     val brojStolova = sc.nextInt()
36
37     val korak = Math.ceil(brojStolova/5.0).toInt
38     // Pravimo instance niti
39     val stefan = new Konobar("Stefan", korak)
40     val nikola = new Konobar("Nikola", korak)
41     val filip = new Konobar("Filip", korak)
42     val nebojsa = new Konobar("Nebojsa", korak)
43     val djordje = new Konobar("Djordje", brojStolova - 4*korak)
44
45     // Ukoliko nasa klasa implementira interfejs Runnable
46     //
47     // val stefan = new Thread(new Konobar("Stefan", 10))
48
49     // Metod start zapocinje izvršavanje metoda run
50     stefan.start()
51     nikola.start()
52     filip.start()
53     nebojsa.start()
54     djordje.start()
55
56   }
57 }

```

Rešenje 4.5 Nastavno osoblje

```

import java.net.URL
import java.util.concurrent._
import java.io.PrintWriter
import java.io.File

// Dodajemo biblioteku za parsiranje HTML stranica
// http://htmlcleaner.sourceforge.net/index.php
import org.htmlcleaner._

class ParserOsoblja(url : String, datoteka : String) extends Thread {

  override def run() {
    // Pravimo novi objekat tipa HtmlCleaner koji parsira HTML datoteku sa zadatim url-om
    // Neki od korisnih metoda klase HtmlCleaner su:
    // - clean(url) - vraca koreni cvor tipa TagNode sadrzaja datog url-a
    // - getInnerHtml(cvor) - vraca string - sadrzaj HTML koda datog cvora
    //
    // Neki od korisnih metoda klase TagNode su:
    // - getChildTags() - vraca niz dece cvorova tipa TagNode
    // - hasChildren() - vraca true ukoliko cvor ima dece, false inace
    // - getAttributeByName(a) - vraca vrednost atributa a
    // - getElementsByName(ime, rekurzivno) - vraca niz dece sa zadatim imenom
    // - getElementsByAttValue(a, v, rekurzivno, caseSensitive)
    // - vraca niz dece koja imaju atribut a sa vrednoscu v
    // - getElementsHavingAttribute(a, rekurzivno)
    // - vraca niz dece koja imaju atribut a
    // ...
    //
    val cleaner = new HtmlCleaner()
    val pw : PrintWriter = new PrintWriter(new File(datoteka))
    // Dohvatamo korenu etiketu - html
    val root = cleaner.clean(new URL(url))
    // Dohvatamo niz etiketa koje imaju klasu employer
    val elements = root.getElementsByAttValue("class", "employer", true, false)
    for(el <- elements){
    // Dohvatamo link- a etikete jer se u prvoj nalazi ime osobe
    val linkovi = el.getElementsHavingAttribute("href", true)
    // Dohvatamo div etikete jer se u prvoj nalazi korisnicko ime

```

```

40     val divovi = el.getElementsByName("div", true)
41     if(linkovi.length != 0 && divovi.length != 0)
42         pw.append(linkovi(0).getText + " - " + divovi(0).getText + "@matf.bg.ac.rs \n")
43     }
44     pw.close()
45 }
46
47 object NastavnoOsoblje {
48     def main(args : Array[String]) {
49         // Pravimo niz niti koje ce da izvlace email adrese nastavnog osoblja na fakultetu
50         val niti = new Array[ParserOsoblja](5)
51         niti(0) = new ParserOsoblja("http://www.matf.bg.ac.rs/redovni/", "
52         redovni_profesori.txt")
53         niti(1) = new ParserOsoblja("http://www.matf.bg.ac.rs/vanredni/", "
54         vanredni_profesori.txt")
55         niti(2) = new ParserOsoblja("http://www.matf.bg.ac.rs/docenti/", "docenti.txt")
56         niti(3) = new ParserOsoblja("http://www.matf.bg.ac.rs/asistenti/", "asistenti.txt")
57         niti(4) = new ParserOsoblja("http://www.matf.bg.ac.rs/saradnici/", "saradnici.txt")
58
59         // Pokrecemo niti
60         for(i <- 0 until 5)
61             niti(i).start()
62     }
63 }

```

Rešenje 4.6 Množenje matrica

```

1  import java.util.concurrent._
2  import java.util.Scanner
3  import java.io.PrintWriter
4  import java.io.File
5  import scala.Array._
6
7  class Mnozilac(vrst1 : Array[Int],
8                matrica2 : Array[Array[Int]],
9                rezultat : Array[Int])
10     extends Thread {
11
12     var k = matrica2.length*matrica2(1).length / vrst1.length
13     var m = vrst1.length
14
15     override def run() {
16         for(i <- 0 until k)
17             rezultat(i) = skProizvod(i)
18     }
19
20     def skProizvod(j : Int) : Int = {
21         var res = 0
22         for(i <- 0 until m)
23             res += vrst1(i)*matrica2(i)(j)
24         return res
25     }
26 }
27
28 object MnozenjeMatrica {
29     def main(args : Array[String]) {
30
31         val sc1 : Scanner = new Scanner(new File("matrica1.txt"))
32         val sc2 : Scanner = new Scanner(new File("matrica2.txt"))
33         val pw : PrintWriter = new PrintWriter(new File("matrica3.txt"))
34
35         // Ucitavamo dimenzije matrica
36         val n = sc1.nextInt()
37         val m1 = sc1.nextInt()
38         val m2 = sc2.nextInt()
39         val k = sc2.nextInt()
40
41         if(m1 != m2){

```

```

43     println("Greska! Dimenzije matrica se moraju poklapati!")
44     return
45 }
46 // Funkcija ofDim[Tip](n,m) pravi videdimenzioni niz dimenzija mxn
47 val matrica1 = ofDim[Int](n,m1)
48 val matrica2 = ofDim[Int](m2,k)
49 val rezultat = ofDim[Int](n,k)
50
51 // Ucitavamo elemente prve matrice
52 for(i <- 0 until n)
53     for(j <- 0 until m1)
54         matrica1(i)(j) = sc1.nextInt()
55
56 // Ucitavamo elemente druge matrice
57 for(i <- 0 until m2)
58     for(j <- 0 until k)
59         matrica2(i)(j) = sc2.nextInt()
60
61 val mnozioci = new Array[Mnozilac](n)
62
63 // Pravimo niz niti koje ce da racunaju i-tu vrstu rezultata mnozenja matrica
64 for(i <- 0 until n)
65     mnozioci(i) = new Mnozilac(matrica1(i), matrica2, rezultat(i))
66
67 // Zapocinjemo izvršavanje niti
68 for(i <- 0 until n)
69     mnozioci(i).start()
70
71 // Cekamo da niti zavrse sa izracunavanjem
72 for(i <- 0 until n)
73     mnozioci(i).join()
74
75 // Upisujemo rezultujuću matricu u datoteku
76 pw.append(s"$n $k \n")
77 for(i <- 0 until n){
78     for(j <- 0 until k)
79         pw.append(s"${rezultat(i)(j)} ")
80     pw.append("\n")
81 }
82
83 pw.close()
84 }
85 }

```

Rešenje 4.7 Broj karaktera DNK

```

1 import java.util.concurrent._
2 import java.util.concurrent.atomic._
3 import java.util.Scanner
4 import java.io.File
5 import scala.collection.mutable.ArrayBuffer
6
7 class Brojac(poc : Int, kraj : Int,
8             linije : ArrayBuffer[String],
9             mapaKaraktera : ConcurrentHashMap[Char, Int])
10     extends Thread {
11
12     override def run() {
13         for(i <- poc until kraj){
14             // Racunamo broj svakog karaktera u liniji
15             val a = linije(i).count(_=='a')
16             val c = linije(i).count(_=='c')
17             val g = linije(i).count(_=='g')
18             val t = linije(i).count(_=='t')
19
20             // Synchronized ključna rec obeležava kritičnu sekciju
21             // i garantuje se da u svakom trenutku tačno jedna nit može izvršavati naredbe iz bloka.
22             // Synchronized se može koristiti na više načina:
23             //
24             // Metodi klase

```



```
25 //
26 //   def f() = synchronized { teloFunkcije }
27 //
28 //   Na ovaj nacin smo naglasili da je metod f jedne instance nase klase kriticna
29 //   sekciya
30 //   i u svakom trenutku tacno jedna nit moze izvršavati ovaj metod te instance.
31 //
32 //   Blok instance
33 //
34 //   instanca.synchronized { blok }
35 //
36 //   Na ovaj nacin smo naglasili da za datu instancu u svakom trenutku
37 //   tacno jedna nit moze izvršavati naredbe bloka
38 //   Ovakvo ponasanje mozemo posmatrati i iz drugacijeg ugla.
39 //   Instanca predstavlja monitor objekat.
40 //   U trenutku kada nit pozeli da izvršava blok kriticne sekcije,
41 //   ona zaključa instancu, izvrši kritičnu sekciju i otključa instancu.
42 //
43 //   Treba teziti ka tome da kritična sekcija bude sto manja
44 //   kako bismo sto vise iskoristili prednosti konkurentnog izvršavanja
45 //
46 //   TODO: Zakomentarisati synchronized blok i videti sta se desava prilikom
47 //   pokretanja programa
48 //   sa razlicitim brojem niti.
49 //   Moguci scenario je da jedna nit procita vrednost iz mape, druga nit nakon toga
50 //   azurira mapu,
51 //   a prva nit i dalje drzi vrednost koju je procitala pre azuriranja tako da kada
52 //   ona azurira mapu
53 //   rezultat azuriranja nece biti ispravan.
54 //
55 //   mapaKaraktera.synchronized {
56 //       mapaKaraktera.replace('a', mapaKaraktera.get('a')+a)
57 //       mapaKaraktera.replace('c', mapaKaraktera.get('c')+c)
58 //       mapaKaraktera.replace('g', mapaKaraktera.get('g')+g)
59 //       mapaKaraktera.replace('t', mapaKaraktera.get('t')+t)
60 //   }
61 // }
62 // }
63 // }
64 // }
65 // }
66 // }
67 // }
68 // }
69 // }
70 // }
71 // }
72 // }
73 // }
74 // }
75 // }
76 // }
77 // }
78 // }
79 // }
80 // }
81 // }
82 // }
83 // }
84 // }
85 // }
86 // }
87 // }
88 // }
89 // }
90 // }
91 // }
92 // }
93 // }
94 // }
95 // }
96 // }
97 // }
98 // }
99 // }
100 // }
101 // }
102 // }
103 // }
104 // }
105 // }
106 // }
107 // }
108 // }
109 // }
110 // }
111 // }
112 // }
113 // }
114 // }
115 // }
116 // }
117 // }
118 // }
119 // }
120 // }
121 // }
122 // }
123 // }
124 // }
125 // }
126 // }
127 // }
128 // }
129 // }
130 // }
131 // }
132 // }
133 // }
134 // }
135 // }
136 // }
137 // }
138 // }
139 // }
140 // }
141 // }
142 // }
143 // }
144 // }
145 // }
146 // }
147 // }
148 // }
149 // }
150 // }
151 // }
152 // }
153 // }
154 // }
155 // }
156 // }
157 // }
158 // }
159 // }
160 // }
161 // }
162 // }
163 // }
164 // }
165 // }
166 // }
167 // }
168 // }
169 // }
170 // }
171 // }
172 // }
173 // }
174 // }
175 // }
176 // }
177 // }
178 // }
179 // }
180 // }
181 // }
182 // }
183 // }
184 // }
185 // }
186 // }
187 // }
188 // }
189 // }
190 // }
191 // }
192 // }
193 // }
194 // }
195 // }
196 // }
197 // }
198 // }
199 // }
200 // }
201 // }
202 // }
203 // }
204 // }
205 // }
206 // }
207 // }
208 // }
209 // }
210 // }
211 // }
212 // }
213 // }
214 // }
215 // }
216 // }
217 // }
218 // }
219 // }
220 // }
221 // }
222 // }
223 // }
224 // }
225 // }
226 // }
227 // }
228 // }
229 // }
230 // }
231 // }
232 // }
233 // }
234 // }
235 // }
236 // }
237 // }
238 // }
239 // }
240 // }
241 // }
242 // }
243 // }
244 // }
245 // }
246 // }
247 // }
248 // }
249 // }
250 // }
251 // }
252 // }
253 // }
254 // }
255 // }
256 // }
257 // }
258 // }
259 // }
260 // }
261 // }
262 // }
263 // }
264 // }
265 // }
266 // }
267 // }
268 // }
269 // }
270 // }
271 // }
272 // }
273 // }
274 // }
275 // }
276 // }
277 // }
278 // }
279 // }
280 // }
281 // }
282 // }
283 // }
284 // }
285 // }
286 // }
287 // }
288 // }
289 // }
290 // }
291 // }
292 // }
293 // }
294 // }
295 // }
296 // }
297 // }
298 // }
299 // }
300 // }
301 // }
302 // }
303 // }
304 // }
305 // }
306 // }
307 // }
308 // }
309 // }
310 // }
311 // }
312 // }
313 // }
314 // }
315 // }
316 // }
317 // }
318 // }
319 // }
320 // }
321 // }
322 // }
323 // }
324 // }
325 // }
326 // }
327 // }
328 // }
329 // }
330 // }
331 // }
332 // }
333 // }
334 // }
335 // }
336 // }
337 // }
338 // }
339 // }
340 // }
341 // }
342 // }
343 // }
344 // }
345 // }
346 // }
347 // }
348 // }
349 // }
350 // }
351 // }
352 // }
353 // }
354 // }
355 // }
356 // }
357 // }
358 // }
359 // }
360 // }
361 // }
362 // }
363 // }
364 // }
365 // }
366 // }
367 // }
368 // }
369 // }
370 // }
371 // }
372 // }
373 // }
374 // }
375 // }
376 // }
377 // }
378 // }
379 // }
380 // }
381 // }
382 // }
383 // }
384 // }
385 // }
386 // }
387 // }
388 // }
389 // }
390 // }
391 // }
392 // }
393 // }
394 // }
395 // }
396 // }
397 // }
398 // }
399 // }
400 // }
401 // }
402 // }
403 // }
404 // }
405 // }
406 // }
407 // }
408 // }
409 // }
410 // }
411 // }
412 // }
413 // }
414 // }
415 // }
416 // }
417 // }
418 // }
419 // }
420 // }
421 // }
422 // }
423 // }
424 // }
425 // }
426 // }
427 // }
428 // }
429 // }
430 // }
431 // }
432 // }
433 // }
434 // }
435 // }
436 // }
437 // }
438 // }
439 // }
440 // }
441 // }
442 // }
443 // }
444 // }
445 // }
446 // }
447 // }
448 // }
449 // }
450 // }
451 // }
452 // }
453 // }
454 // }
455 // }
456 // }
457 // }
458 // }
459 // }
460 // }
461 // }
462 // }
463 // }
464 // }
465 // }
466 // }
467 // }
468 // }
469 // }
470 // }
471 // }
472 // }
473 // }
474 // }
475 // }
476 // }
477 // }
478 // }
479 // }
480 // }
481 // }
482 // }
483 // }
484 // }
485 // }
486 // }
487 // }
488 // }
489 // }
490 // }
491 // }
492 // }
493 // }
494 // }
495 // }
496 // }
497 // }
498 // }
499 // }
500 // }
501 // }
502 // }
503 // }
504 // }
505 // }
506 // }
507 // }
508 // }
509 // }
510 // }
511 // }
512 // }
513 // }
514 // }
515 // }
516 // }
517 // }
518 // }
519 // }
520 // }
521 // }
522 // }
523 // }
524 // }
525 // }
526 // }
527 // }
528 // }
529 // }
530 // }
531 // }
532 // }
533 // }
534 // }
535 // }
536 // }
537 // }
538 // }
539 // }
540 // }
541 // }
542 // }
543 // }
544 // }
545 // }
546 // }
547 // }
548 // }
549 // }
550 // }
551 // }
552 // }
553 // }
554 // }
555 // }
556 // }
557 // }
558 // }
559 // }
560 // }
561 // }
562 // }
563 // }
564 // }
565 // }
566 // }
567 // }
568 // }
569 // }
570 // }
571 // }
572 // }
573 // }
574 // }
575 // }
576 // }
577 // }
578 // }
579 // }
580 // }
581 // }
582 // }
583 // }
584 // }
585 // }
586 // }
587 // }
588 // }
589 // }
590 // }
591 // }
592 // }
593 // }
594 // }
595 // }
596 // }
597 // }
598 // }
599 // }
600 // }
601 // }
602 // }
603 // }
604 // }
605 // }
606 // }
607 // }
608 // }
609 // }
610 // }
611 // }
612 // }
613 // }
614 // }
615 // }
616 // }
617 // }
618 // }
619 // }
620 // }
621 // }
622 // }
623 // }
624 // }
625 // }
626 // }
627 // }
628 // }
629 // }
630 // }
631 // }
632 // }
633 // }
634 // }
635 // }
636 // }
637 // }
638 // }
639 // }
640 // }
641 // }
642 // }
643 // }
644 // }
645 // }
646 // }
647 // }
648 // }
649 // }
650 // }
651 // }
652 // }
653 // }
654 // }
655 // }
656 // }
657 // }
658 // }
659 // }
660 // }
661 // }
662 // }
663 // }
664 // }
665 // }
666 // }
667 // }
668 // }
669 // }
670 // }
671 // }
672 // }
673 // }
674 // }
675 // }
676 // }
677 // }
678 // }
679 // }
680 // }
681 // }
682 // }
683 // }
684 // }
685 // }
686 // }
687 // }
688 // }
689 // }
690 // }
691 // }
692 // }
693 // }
694 // }
695 // }
696 // }
697 // }
698 // }
699 // }
700 // }
701 // }
702 // }
703 // }
704 // }
705 // }
706 // }
707 // }
708 // }
709 // }
710 // }
711 // }
712 // }
713 // }
714 // }
715 // }
716 // }
717 // }
718 // }
719 // }
720 // }
721 // }
722 // }
723 // }
724 // }
725 // }
726 // }
727 // }
728 // }
729 // }
730 // }
731 // }
732 // }
733 // }
734 // }
735 // }
736 // }
737 // }
738 // }
739 // }
740 // }
741 // }
742 // }
743 // }
744 // }
745 // }
746 // }
747 // }
748 // }
749 // }
750 // }
751 // }
752 // }
753 // }
754 // }
755 // }
756 // }
757 // }
758 // }
759 // }
760 // }
761 // }
762 // }
763 // }
764 // }
765 // }
766 // }
767 // }
768 // }
769 // }
770 // }
771 // }
772 // }
773 // }
774 // }
775 // }
776 // }
777 // }
778 // }
779 // }
780 // }
781 // }
782 // }
783 // }
784 // }
785 // }
786 // }
787 // }
788 // }
789 // }
790 // }
791 // }
792 // }
793 // }
794 // }
795 // }
796 // }
797 // }
798 // }
799 // }
800 // }
801 // }
802 // }
803 // }
804 // }
805 // }
806 // }
807 // }
808 // }
809 // }
810 // }
811 // }
812 // }
813 // }
814 // }
815 // }
816 // }
817 // }
818 // }
819 // }
820 // }
821 // }
822 // }
823 // }
824 // }
825 // }
826 // }
827 // }
828 // }
829 // }
830 // }
831 // }
832 // }
833 // }
834 // }
835 // }
836 // }
837 // }
838 // }
839 // }
840 // }
841 // }
842 // }
843 // }
844 // }
845 // }
846 // }
847 // }
848 // }
849 // }
850 // }
851 // }
852 // }
853 // }
854 // }
855 // }
856 // }
857 // }
858 // }
859 // }
860 // }
861 // }
862 // }
863 // }
864 // }
865 // }
866 // }
867 // }
868 // }
869 // }
870 // }
871 // }
872 // }
873 // }
874 // }
875 // }
876 // }
877 // }
878 // }
879 // }
880 // }
881 // }
882 // }
883 // }
884 // }
885 // }
886 // }
887 // }
888 // }
889 // }
890 // }
891 // }
892 // }
893 // }
894 // }
895 // }
896 // }
897 // }
898 // }
899 // }
900 // }
901 // }
902 // }
903 // }
904 // }
905 // }
906 // }
907 // }
908 // }
909 // }
910 // }
911 // }
912 // }
913 // }
914 // }
915 // }
916 // }
917 // }
918 // }
919 // }
920 // }
921 // }
922 // }
923 // }
924 // }
925 // }
926 // }
927 // }
928 // }
929 // }
930 // }
931 // }
932 // }
933 // }
934 // }
935 // }
936 // }
937 // }
938 // }
939 // }
940 // }
941 // }
942 // }
943 // }
944 // }
945 // }
946 // }
947 // }
948 // }
949 // }
950 // }
951 // }
952 // }
953 // }
954 // }
955 // }
956 // }
957 // }
958 // }
959 // }
960 // }
961 // }
962 // }
963 // }
964 // }
965 // }
966 // }
967 // }
968 // }
969 // }
970 // }
971 // }
972 // }
973 // }
974 // }
975 // }
976 // }
977 // }
978 // }
979 // }
980 // }
981 // }
982 // }
983 // }
984 // }
985 // }
986 // }
987 // }
988 // }
989 // }
990 // }
991 // }
992 // }
993 // }
994 // }
995 // }
996 // }
997 // }
998 // }
999 // }
1000 // }
```

```

91 // Konstruktor prima tri argumenta:
92 // - inicijalni kapacitet mape
93 // - faktor povećavanja mape
94 // - broj niti koji se pretpostavlja da će konkurentno pristupati objektu mape
95 //
96 // Neki od korisnih metoda klase ConcurrentHashMap su:
97 // - get(kljuc) - vraća element sa zadatim ključem, odnosno null ukoliko takav ne
    postoji
98 // - put(kljuc, vrednost) - dodaje element sa zadatim parametrima
99 // - remove(kljuc) - uklanja element sa zadatim ključem
100 // - replace(kljuc, vrednost) - postavlja vrednost elementu sa zadatim ključem
101 // - size() - vraća veličinu mape
102 // - isEmpty() - vraća true ukoliko je mapa prazna, false inace
103 // ...
104 //
105 val mapaKaraktera = new ConcurrentHashMap[Char, Int](4,4,brojNiti)
106 mapaKaraktera.put('a', 0)
107 mapaKaraktera.put('c', 0)
108 mapaKaraktera.put('g', 0)
109 mapaKaraktera.put('t', 0)
110
111 val korak = Math.ceil(brojLinija.toDouble/brojNiti.toDouble).toInt
112
113 for(i <- 0 until brojNiti)
114     brojaci(i) = new Brojac(i*korak, Math.min((i+1)*korak, brojLinija), linije,
115     mapaKaraktera)
116
117     for(b <- brojaci)
118         b.start()
119
120     for(b <- brojaci)
121         b.join()
122
123     println("Rezultati konkurentnog izvršavanja")
124     println("A: " + mapaKaraktera.get('a'))
125     println("C: " + mapaKaraktera.get('c'))
126     println("G: " + mapaKaraktera.get('g'))
127     println("T: " + mapaKaraktera.get('t'))
128
129     println("Pravi rezultati \nA: 1761 \nC: 1577 \nG: 1589 \nT: 1913")
130 }
131 }

```

Rešenje 4.8 Krediti

```

1 import java.util.concurrent.atomic._
2 import java.util.concurrent._
3 import java.util.Scanner
4 import java.io.File
5
6 class Sluzbenica(kamata : Int,
7     kapital : AtomicLong,
8     redKlijenata : ConcurrentLinkedQueue[Klijent],
9     zaduzeniKlijenti : ConcurrentLinkedQueue[Klijent])
10     extends Thread {
11
12     override def run() {
13         while(true){
14             // Dohvatamo sledeceg klijenta iz reda
15             var k : Klijent = redKlijenata.poll()
16             // Ukoliko takav ne postoji završavamo
17             if(k == null)
18                 return
19
20             println("Klijent " + k.getIme() + " razgovara sa sluzbenicom.")
21             Thread.sleep(ThreadLocalRandom.current().nextInt(1,10)*1000)
22             // Iako je kapital objekat AtomicLong i garantuje se atomicnost operacija
23             // azuriranja
24             // mogu nastati problemi prilikom konkurentnog pristupanja i azuriranja,
25             // i zbog toga je potrebno operacije sa ovim objektom obmotati synchronized
26             // blokom.

```

```

25 //      Problem moze nastati u delu koda (*****).
//      Pretpostavimo da dve niti izvrsavaju ovaj deo koda konkurentno.
27 //      Prva nit procita vrednost kapitala, nakon toga druga nit procita vrednost
kapitala
//      pre nego sto je prva nit izmenila vrednot, odmah posle prva nit promeni
vrednost kapitala
29 //      i postavi novu vrednot (staraVrednost - prvaPozajmica)/
//      U ovom trenutku druga nit ima vrednost kapitala koja nije ispravna sa kojom
dalje operise.
31 //      Druga nit promeni vrednost kapitala i postavi novu vrednost (staraVrednost -
drugaPozajmica)
//      sto nije realna vrednost (staraVrednost - prvaPozajmica - drugaPozajmica)
33      kapital.synchronized {
          if(k.getPozajmica() > kapital.get())
35              println("Klijent " + k.getIme() + " ne moze dobiti kredit.")
          else{
37              k.setDug(k.getPozajmica()*((100+kamata.toFloat)/100))
//              *****
39              val novKapital = kapital.get() - k.getPozajmica()
kapital.set(novKapital)
41              *****
43              println("Klijent " + k.getIme() + " je dobio kredit u iznosu od "
+ k.getPozajmica() + "e odnosno sa kamatom " + k.getDug() + "e.")
          zaduzeniKlijenti.add(k)
45          }
47      }
49 }

51 class Klijent(ime : String, pozajmica : Int) {
53     var dug : Float = 0
55     def getIme() : String = {
        return ime
57     }
59     def getPozajmica() : Int = {
        return pozajmica
61     }
63     def getDug() : Float = {
        return dug
65     }
67     def setDug(d : Float) = {
        dug = d
69     }
71 }

object Banka {
73     def main(args : Array[String]) {
//      Klasa AtomicLong predstavlja enkapsulaciju long integer vrednosti
75 //      nad kojom se operacije azuriranja izvrsavaju atomicno.
//
77 //      Neki od korisnih metoda ove klase su:
//      - get() - vraca trenutnu vrednost
79 //      - set(v) - postavlja vrednost v
//      - getAndAdd(v) - atomicki dodaje vrednost v i vraca prethodnu vrednost
81 //      - addAndGet(v) - atomicki dodaje vrednost v i vraca novu vrednost
//      - getAndIncrement() - atomicki inkrementira vrednost i vraca prethodnu vrednost
83 //      - incrementAndGet() - atomicki inkrementira i vraca novu vrednost
//      - getAndDecrement() - atomicki dekrementira vrednost i vraca prethodnu vrednost
85 //      - decrementAndGet() - atomicki dekrementira i vraca novu vrednost
//      - compareAndSet(ocekivanaVrednost, novaVrednost) - postavlja novu vrednost
87 //      ukoliko je stara jednaka ocekivanoj
//
89 //      U paketu java.util.concurrent.atomic postoje i druge korisne klase kao sto su
//      AtomicBoolean, AtomicIntegerArray, AtomicInteger itd.
91     val sc1 : Scanner = new Scanner(System.in)
93     println("Unesite pocetni kapital banke: ")

```

```

95     val kapital = new AtomicLong(sc1.nextLong())
    val sacuvanKapital : Float = kapital.get()

97     println("Unesite kamatnu stopu: ")
    val kamata = sc1.nextInt()

99

101    println("Unesite broj sluzbenica u ekspozituri: ")
    val sluzbenice = new Array[Sluzbenica](sc1.nextInt())

103    val sc2 : Scanner = new Scanner(new File("red_klijenata.txt"))

105    // Klasa ConcurrentLinkedQueue predstavlja implementaciju reda
    // cije su operacije bezbedne u kontekstu konkurentnog izvršavanja.
107    //
    // Neki od korisnih metoda su:
109    // - add(e) - dodaje element u red
    // - poll() - skida element sa pocetka reda i vraća ga kao rezultat
111    // - peek() - vraća element sa pocetka reda (ne skida ga)
    // - remove(e) - uklanja element e iz reda
113    // - isEmpty() - vraća true ukoliko je red prazan
    // ...
115    //

117    val redKlijenata = new ConcurrentLinkedQueue[Klijent]()
    val zaduzeniKlijenti = new ConcurrentLinkedQueue[Klijent]()

119    while(sc2.hasNextLine())
        redKlijenata.add(new Klijent(sc2.next(), sc2.nextInt()))

121

123    for(i <- 0 until sluzbenice.length)
        sluzbenice(i) = new Sluzbenica(kamata, kapital, redKlijenata, zaduzeniKlijenti)

125    for(s <- sluzbenice)
        s.start()

127

129    for(s <- sluzbenice)
        s.join()

131    // Iteriramo kroz red zaduzenih klijenata i racunamo ukupno zaduzenje
    var ukupnoZaduzenje : Float = 0
133    val iterator = zaduzeniKlijenti.iterator()
    while(iterator.hasNext())
135        ukupnoZaduzenje += iterator.next().getDug()

137    println(s"Banka je zaradila ${ukupnoZaduzenje-sacuvanKapital}e.")
    }
139 }

```

Rešenje 4.9 Kladionica

```

1 import java.util.concurrent.atomic._
  import java.util.concurrent._
3 import java.util.Scanner
  import java.io.File
5 import scala.collection.mutable.HashMap
  import scala.collection.mutable.ArrayBuffer
7
  class Kladionicar(ime : String,
9                      novac : Int,
                      tiket : HashMap[String, Char],
11                     utakmice : HashMap[String, Tuple4[Float,Float,Float,Char]])
                      extends Thread {
13
15     var zarada : Float = 0

17     override def run(){
        // Cekamo da se odigraju sve utakmice
        // Funkcije wait(), notify() i notifyAll()
19        // moraju biti zakljucane unutar bloka synchronized
        utakmice.synchronized {
21            utakmice.wait()
        }
23    }

```

```

25     var pogodjeno = 0
26     var ukupnaKvota : Float = 0
27     // Racunamo ukupnu zaradu
28     for(t <- tiket)
29         if(t._2 == utakmice(t._1)._4){
30             println(ime + " je pogodio utakmicu " + t._1 + " - " + utakmice(t._1)._4)
31             pogodjeno += 1
32             if(utakmice(t._1)._4 == '1')
33                 ukupnaKvota += utakmice(t._1)._1
34             else if(utakmice(t._1)._4 == 'x')
35                 ukupnaKvota += utakmice(t._1)._2
36             else if(utakmice(t._1)._4 == '2')
37                 ukupnaKvota += utakmice(t._1)._3
38         }
39         if(pogodjeno != 0)
40             zarada = ukupnaKvota * novac/pogodjeno
41     }
42
43     def getIme() : String = {
44         return ime
45     }
46
47     def getZarada() : Float = {
48         return zarada
49     }
50 }
51
52 object Kladionica {
53     def main(args : Array[String]) {
54         val sc1 : Scanner = new Scanner(new File("utakmice.txt"))
55         val sc2 : Scanner = new Scanner(new File("kladionicari.txt"))
56         // Klasa HashMap iz paketa scala.collection.mutable
57         // predstavlja implementaciju mape koja se moze azurirati (eng. mutable)
58         //
59         // Neke od korisnih funkcija su:
60         // -put(k,v) - dodaje vrednost u mapu sa zadatim kljucem
61         // -size - vraca velicinu mape
62         // -contains(k) - vraca true ukoliko postoji element sa zadatim kljucem, false
63         //   inace
64         // ...
65         // Takodje mozemo iterirati kroz elemente mape for petljom
66         //
67         // Klasa Tuple4 je jedna u nizu klasa koje implementiraju torke (Tuple1, Tuple2,
68         //   Tuple3,...)
69         // koje se mogu azurirati.
70         // Elementima torke pristupamo na sledeci nacin - torka._1, torka._2, torka._3,
71         //   torka._4
72         //
73         val utakmice = new HashMap[String, Tuple4[Float,Float,Float,Char]]()
74
75         // Rezultate utakmica postavljamo da budu karakter '-'
76         // kako bismo naglasili da se utakmice jos nisu odigrale
77         while(sc1.hasNextLine()){
78             utakmice.put(sc1.nextLine(),(sc1.nextFloat(), sc1.nextFloat(), sc1.nextFloat(),
79             '-'))
80             sc1.nextLine()
81         }
82         val kladionicari = new ArrayBuffer[Kladionicar]()
83
84         while(sc2.hasNextLine()){
85             val ime = sc2.next()
86             val novac = sc2.nextInt()
87             val tiket = new HashMap[String, Char]()
88             for(i <- 0 until 5){
89                 sc2.nextLine()
90                 tiket.put(sc2.nextLine(), sc2.next()(0))
91             }
92             kladionicari.append(new Kladionicar(ime, novac, tiket, utakmice))
93         }
94
95         for(k <- kladionicari)
96             k.start()

```

```

93     println("Cekamo da se utakmice odigraju.")
94     Thread.sleep(5000)
95
96 // Racunamo rezultate utakmica
97     val res = Array('1', 'x', '2')
98     for(u <- utakmice)
99         utakmice(u._1) = (u._2._1,
100                          u._2._2,
101                          u._2._3,
102                          res(ThreadLocalRandom.current().nextInt(0, 3))
103                          )
104
105 // Ulazimo u kriticnu sekciju
106 // i obavestavamo niti koje cekaju
107     utakmice.synchronized {
108         utakmice.notifyAll()
109     }
110
111     for(k <- kladionicari)
112         k.join()
113
114     var isplata : Float = 0
115     for(k <- kladionicari){
116         isplata += k.getZarada()
117         println(k.getIme() + " cekna na isplatu " + k.getZarada() + " dinara.")
118     }
119     println("Ukupno kladionica treba da isplati " + isplata + " dinara.")
120 }
121 }

```

Rešenje 4.10 Zbir vektora

```

import java.util.Scanner
2
3 class Sabirac(poc : Int,
4               kraj : Int ,
5               vektor1 : Array[Float],
6               vektor2 : Array[Float])
7     extends Thread {
8 // Svaka nit racuna zbir svog dela vektora [poc, kraj]
9 // i rezultat smesta u prvi vektor.
10    override def run() {
11        for( i <- poc until kraj)
12            vektor1(i) += vektor2(i)
13    }
14 }
15
16 object ZbirVektora {
17     def main(args : Array[String]){
18
19         val sc = new Scanner(System.in)
20
21         println("Unesite dimenziju vektora: ")
22         val n = sc.nextInt()
23
24         val vektor1 : Array[Float] = new Array(n)
25         val vektor2 : Array[Float] = new Array(n)
26
27         println("Unesite elemente prvog vektora: ")
28         for(i <- 0 until n)
29             vektor1(i) = sc.nextFloat()
30
31         println("Unesite elemente drugog vektora: ")
32         for(i <- 0 until n)
33             vektor2(i) = sc.nextFloat()
34
35         println("Unesite broj niti: ")
36         val brojNiti = sc.nextInt()
37
38         val niti = new Array[Sabirac](brojNiti)

```

```

40     val korak = Math.ceil(n/brojNiti.toDouble).toInt
42 // Pravimo niti i zadajemo im indekse - granice
43     for(i <- 0 until brojNiti)
44         niti(i) = new Sabirac(i*korak, Math.min((i+1)*korak,n),vektor1,vektor2)
46 // Pokrecemo racunanje
47     for(i <- 0 until brojNiti)
48         niti(i).start()
50 // Cekamo da niti zavrse sa racunanjem
51     for(i <- 0 until brojNiti)
52         niti(i).join()
54 // Kada sve niti zavrse sa racunanjem ispisujemo rezultat
55     print("Zbir vektora je: \n[")
56     for(i <- 0 until n-1)
57         print(vektor1(i) + ", ")
58     println(vektor1(n-1) + "]"")
59 }
60 }

```

Rešenje 4.11 Broj petocifrenih brojeva

```

import java.util.concurrent._
import java.io._
import java.util.Scanner
import scala.collection.mutable.ArrayBuffer

class BrojacPetocifrenih(dat : String) extends Thread {

    var rezultat : Int = 0
    // Citamo brojeve iz datoteke i povecavamo lokalni brojac
    // ukoliko je broj petocifren
    override def run() {
        val sc : Scanner = new Scanner(new File(dat))

        while(sc.hasNextInt()){
            var broj = sc.nextInt()
            if(broj >= 10000 && broj <= 99999)
                this.rezultat+=1
        }
    }

    def getRezultat() : Int = {
        return rezultat
    }

    def getDatoteka() : String = {
        return dat
    }
}

object BrojPetocifrenih {
    def main(args : Array[String]){

        val sc = new Scanner(System.in)

        val brojaci = ArrayBuffer[BrojacPetocifrenih]()
        var kraj = false
        var odg = ""
        var dat = ""

        while(!kraj) {
            println("Da li zelite da zadate ime datoteke koja ce biti obradjena (y/n)?")
            odg = sc.next()
            if(odg.toLowerCase() == "n")
                kraj = true
            else{
                println("Unesite ime datoteke: ")
                dat = sc.next()
                brojaci.append(new BrojacPetocifrenih(dat))
            }
        }
    }
}

```

```

    }
50 }
52 // Zapocinjemo izvršavanje
   for(brojac <- brojaci)
54     brojac.start()
56 // Pozivom metoda join cekamo sve brojace da zavrse sa izracunavanjem
   for(brojac <- brojaci)
58     brojac.join()
60 // Citamo rezultate brojanja svake niti
   for(brojac <- brojaci)
62     println(s"Datoteka ${brojac.getDatoteka()} sadrzi ${brojac.getRezultat()}
   petocifrenih brojeva.")
   }
64 }

```

Rešenje 4.12 Berba

```

1  import java.util.concurrent.atomic._
   import java.util.concurrent._
3  import java.util.Scanner
   import java.io.File
5
   class Berac(drvoredi : ConcurrentLinkedQueue[Tuple2[String,Int]],
7         skladiste : AtomicIntegerArray) extends Thread {
9
   override def run() {
11
       while(true) {
13           // Dohvatamo drvored za berbu iz reda
           val drvored = drvoredi.poll()
15           // Ukoliko nema vise drvoreda za berbu završavamo
           if(drvored == null)
               return
17           println("Berac bere drvo " + drvored._1 )
           Thread.sleep(ThreadLocalRandom.current().nextInt(1,10)*1000)
19           // Dodajemo kilograme voca koje smo obrali u skladiste
           for(i <- 0 until drvored._2){
21               val obrano = ThreadLocalRandom.current().nextInt(30, 50)
               if(drvored._1 == "tresnje")
23                   skladiste.getAndAdd(0, obrano)
               else if(drvored._1 == "kruske")
25                   skladiste.getAndAdd(1, obrano)
               else if(drvored._1 == "kajsije")
27                   skladiste.getAndAdd(2, obrano)
               else if(drvored._1 == "sljive")
29                   skladiste.getAndAdd(3, obrano)
           }
31     }
   }
33 }
35 object Berba {
   def main(args : Array[String]) {
37       val sc1 : Scanner = new Scanner(new File("drvoredi.txt"))
       val sc2 : Scanner = new Scanner(System.in)
39
       // Pravimo skladiste voca koje imamo u vocnjaku
41       // i postavljamo inicijalne kolicine voca.
       // Klasa AtomicIntegerArray sadzi niz integer vrednosti
43       // nad kojima se operacije izvršavaju atomicno.
       // Slicno kao kod klasa AtomicInteger, AtomicLong i dr.
45       val skladiste = new AtomicIntegerArray(4)
       skladiste.set(0,0)
47       skladiste.set(1,0)
       skladiste.set(2,0)
49       skladiste.set(3,0)
       // Pravimo red drvoreda za berbu
51       // Svaki drvored je jedan par (voce, brojStabala).
       val drvoredi = new ConcurrentLinkedQueue[Tuple2[String, Int]]()

```



```

53     while(sc1.hasNextLine())
        drvoredi.add((sc1.next(), sc1.nextInt()))
55
56     println("Unesite broj beraca: ")
57     val brojBeraca = sc2.nextInt()
58
59     val beraci = new Array[Berac](brojBeraca)
60     for(i <- 0 until brojBeraca)
61         beraci(i) = new Berac(drvoredi, skladiste)
62
63     for(b <- beraci)
64         b.start()
65
66     for(b <- beraci)
67         b.join()
68
69     println("Tresanja je obrano: " + skladiste.get(0) + " kilograma.")
70     println("Krusaka je obrano: " + skladiste.get(1) + " kilograma.")
71     println("Kajsija je obrano: " + skladiste.get(2) + " kilograma.")
72     println("Sljiva je obrano: " + skladiste.get(3) + " kilograma.")
73 }
}

```

Rešenje 4.13 Turistička agencija

```

import java.util.concurrent._
2 import java.util.Scanner
import java.io.File
4
5 class Ucesnik(ime : String,
6             cena : Int,
7             dobitnici : Array[String])
8             extends Thread {
9
10     override def run() {
11         // Cekamo dok se ne zavrshi izvlacenje.
12         //
13         // Metod wait() suspenduje nit, oslobadja kriticku sekciju obmotanu synchronized
14         // blokom
15         // i dozvoljava drugim nitima koje su zakljucale isti objekat da udju u kriticku
16         // sekciju
17         // sve dok neka druga nit ne pozove nad istim objektom metod notifyAll()
18         // cime se obavestavaju sve niti koje cekaju sa metodom wait()
19         // da mogu nastaviti sa radom
20         //
21         dobitnici.synchronized {
22             dobitnici.wait()
23         }
24         for(d <- dobitnici)
25             if(d == ime){
26                 println("Cestitamo " + ime +
27                     "!!! Osvojili ste popust od 20% na cenu karte. Vasa karta sada kosta
28                     " + cena*0.8 + "e.")
29                 return
30             }
31         println("Nazalost " + ime +
32             " niste osvojili popust, vise sreće drugi put. Vasa karta kosta " + cena
33             + "e.")
34     }
35
36     def getIme() : String = {
37         return ime
38     }
39 }
40
41 object TuristickaAgencija {
42     def main(args : Array[String]) {
43         val sc : Scanner = new Scanner(new File("ucesnici.txt"))
44
45         val dobitnici = new Array[String](5)
46         val n = sc.nextInt()

```

```
44     sc.nextLine()
45     val ucesnici = new Array[Ucesnik](n)
46     for(i <- 0 until n){
47         ucesnici(i) = new Ucesnik(sc.nextLine(), sc.nextInt(), dobitnici)
48         sc.nextLine()
49     }
50
51     for(u <- ucesnici)
52         u.start()
53
54     println("Izvlacenje je u toku.")
55     Thread.sleep(5000)
56 // Ulazimo u kriticnu sekciju, i racunamo dobitnike nagradnih popusta
57 dobitnici.synchronized {
58     val izvuceniIndeksi = ThreadLocalRandom.current().ints(0, n).distinct().limit
59     (5).toArray()
60     for(j <- 0 until izvuceniIndeksi.length)
61         dobitnici(j) = ucesnici(izvuceniIndeksi(j)).getIme()
62 // Kada su izracunati dobitnici, obavestavamo niti koje cekaju
63 // i izlazimo iz kriticne sekcije
64     dobitnici.notifyAll()
65 }
66 }
67 }
68 }
```


5

Distribuirano programiranje

5.1 Scala

Koristimo jezik Scalu (verzija 2.10) <http://www.scala-lang.org/> i Apache Spark alat <http://spark.apache.org/>

Potrebno je imati instalirano:

- (a) Jezik Scalu verziju 2.10 i/ili 2.11
- (b) Eclipse Neon razvojno okruženje <https://www.eclipse.org/downloads/>
- (c) ScalaIDE dodatak za Eclipse razvojno okruženje <http://scala-ide.org/>

Korisni linkovi:

<http://spark.apache.org/docs/latest/index.html>

https://www.youtube.com/watch?v=7k_9sdT0dX4

5.1.1 Uvod

Apache Spark je radni okvir (eng. framework) za distribuirano programiranje. Pruža interfejs za programiranje (eng. API) u jezicima Java, Scala, Python i R.

Svaka Spark aplikacija se sastoji od glavnog (eng. *driver*) programa koji pokreće funkciju `main` i izvršava paralelne operacije na povezanom klaster računaru. Pristupanje klaster računaru se vrši pomoću objekta kontekst tipa `SparkContext`. Prilikom konstrukcije kontekst objekta, potrebno je definisati koji klaster računar će se koristiti. Spark aplikacija može koristiti lokalni računar kao simulaciju klaster računara (svaka procesorska jedinica će simulirati jedan čvor klaster računara) ili neki udaljeni klaster računar.

Spark koristi posebne kolekcije podataka koje se mogu obrađivati paralelno (eng. *RDD - Resilient Distributed Datasets*). Paralelne kolekcije se mogu napraviti od već postojećih kolekcija u programu ili se mogu učitati iz spoljašnjeg sveta. Nad paralelnim kolekcijama možemo izvršavati dva tipa operacija *transformacije* i *akcije* (slika 5.1). Transformacije transformišu kolekciju na klaster računaru i prave novu paralelnu kolekciju. Sve transformacije su lenje, što znači da rezultat izračunavaju u trenutku kada on postane potreban.

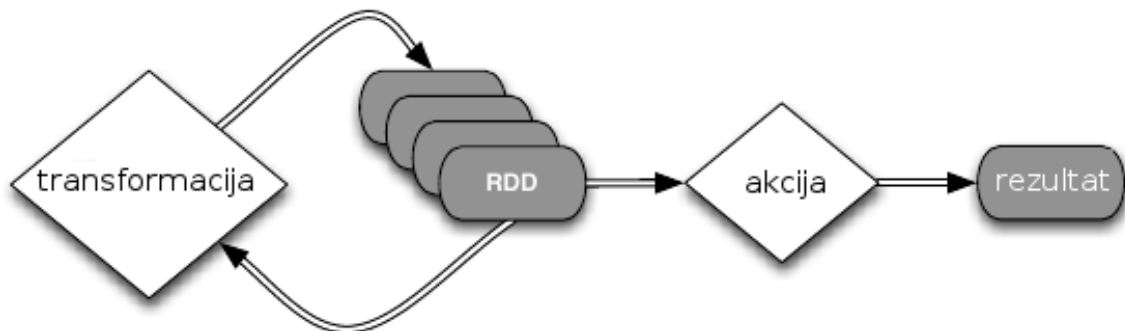
Akcije su operacije koje se izvršavaju na klaster računaru nad paralelnim kolekcijama i njihov rezultat (izračunata vrednost) se vraća na lokalni računar.

Spark može da sačuva paralelne kolekcije u memoriji čvorova klaster računara i na taj način ubrzati izvršavanje narednih operacija.

Spark pruža mogućnost korišćenja deljenih podataka u vidu emitovanih promenljivih (eng. *broadcast variables*) i akumulatora (eng. *accumulators*).

Neke od funkcija transformacija su:

- `map(f)` - vraća novu kolekciju koja se dobija tako što se primeni funkcija `f` nad svakim elementom postojeće kolekcije
- `filter(f)` - primenjuje funkciju `f` nad svim elementima kolekcije i vraća novu kolekciju koja sadrži one elemente za koje je funkcija `f` vratila `true`



Slika 5.1: Operacije nad paralelnim podacima

- `flatMap(f)` - slična je funkciji `map`, razlika je to što primena funkcije `f` nad nekim elementom kolekcije može da vrati 0 ili više novih elemenata koji se smeštaju u rezultujuću kolekciju
- `groupByKey()` - poziva se nad kolekcijom parova (ključ, vrednost) i vraća kolekciju parova (ključ, `Iterable<vrednost>`) tako što grupiše sve vrednosti sa istim ključem i smešta ih u drugi element rezultujućeg para
- `reduceByKey(f)` - poziva se nad kolekcijom parova (ključ, vrednost) i vraća kolekciju parova (ključ, nova_vrednost), nova_vrednost se dobija agregiranjem svih vrednosti sa istim ključem koristeću zadatu funkciju agregacije `f`
- `aggregateByKey(pocetna_vrednost)(f1, f2)` - poziva se nad kolekcijom parova (ključ, vrednost) i vraća kolekciju parova (ključ, nova_vrednost), nova_vrednost se dobija agregiranjem početne vrednosti i svih vrednosti sa istim ključem koristeću zadatu funkciju agregacije `f1` u svakom čvoru klaster računara, a funkcija `f2` agregira vrednosti izračunate u čvorovima klaster računara u jednu vrednost - nova_vrednost
- `sortByKey()` - poziva se nad kolekcijom parova (ključ, vrednost) i vraća novu kolekciju sortiranu po ključu
- `cartesian(druga_kolekcija)` - spaja kolekciju sa drugom kolekcijom i vraća kolekciju svih parova (vrednost_iz_prve_kolekcije, vrednost_iz_druge_kolekcije)
- `zip(druga_kolekcija)` - spaja kolekciju sa drugom kolekcijom spajajući elemente na istim pozicijama i vraća kolekciju parova (vrednost_iz_prve_kolekcije, vrednost_iz_druge_kolekcije)

Neke od funkcija akcija su:

- `reduce(f)` - agregira elemente kolekcije koristeći funkciju `f` i vraća rezultat agregacije
- `collect()` - pretvara paralelnu kolekciju u niz (koji se nalazi na lokalnom računaru)
- `count()` - vraća broj elemenata kolekcije
- `countByKey()` - poziva se nad kolekcijom parova (ključ, vrednost), za svaki ključ broji koliko ima elemenata sa tim ključem i vraća neparalelnu kolekciju (ključ, broj_elementa)
- `first()` - vraća prvi element kolekcije
- `take(n)` - vraća prvih `n` elemenata kolekcije
- `takeSample(sa_vracanjem, n, seed)` - vraća prvih `n` nasumično izabranih elemenata kolekcije (`sa` ili bez vraćanja), `seed` predstavlja početnu vrednost generatora slučajnih brojeva
- `takeOrdered(n[, poredak])` - vraća prvih `n` elemenata sortirane kolekcije (koristeći prirodan poredak kolekcije ili zadati poredak)
- `saveAsTextFile(ime_direktorijuma)` - upisuje kolekciju u datoteke koje se nalaze u zadatom direktorijumu

- `foreach(f)` - poziva funkciju `f` nad svim elementima kolekcije (uglavnom se koristi kada funkcija `f` ima neke sporedne efekte kao što je upisivanja podataka u datoteku ili slično)

Konfiguraciju Spark aplikacije možemo podešavati dinamički prilikom pokretanja aplikacije na klaster računaru. Potrebno je upakovati aplikaciju zajedno sa svim njenim bibliotekama u `.jar` datoteku koristeći neki od alata (Maven ¹, SBT ² i sl.) i instalirati Spark upravljač na klaster računaru (Standalone ³, Mesos ⁴, Yarn ⁵ i sl.). Aplikaciju možemo pokrenuti pomoću `spark-submit` skripta koji se nalazi u `bin` direktorijumu instaliranog Spark alata i konfigurirati dinamički. Na primer:

```
1 ./bin/spark-submit --class Main --master local --num-executors 20
   Aplikacija.jar
```

Parametri koji se najčešće koriste prilikom konfiguracije su:

- `--master url` - URL klaster računara
- `--class ime_klase` - glavna klasa naše aplikacije
- `--num-executors n` - broj čvorova koji izvršavaju našu aplikaciju (eng. executors)
- `--executor-cores n` - broj zadataka koje jedan čvor može izvršavati istovremeno
- `--executor-memory n` - veličina hip memorije svakog čvora

5.1.2 Zadaci sa rešenjima

Zadatak 5.1 Parni kvadrati Napisati program koji učitava ceo broj `n` veći od 2 i ispisuje sve kvadrate parnih brojeva počev od broja 2 do `n`.

[Rešenje 5.1]

Zadatak 5.2 Broj petocifrenih Napisati program koji ispisuje broj petocifrenih brojeva koji se nalaze u datoteci `brojevi.txt` (svaka linija sadrži jedan broj).

[Rešenje 5.2]

Zadatak 5.3 Skalarni proizvod Napisati program koji računa skalarni proizvod dva vektora (pretpostavimo da su vektori uvek zadati ispravno tj. iste su dužine) i ispisuje ih na izlaz. Vektori se nalaze u datotekama `vektor1.txt` i `vektor2.txt` u formatu:

```
1 a1, a2, a3, a4, ... an
```

[Rešenje 5.3]

Zadatak 5.4 Broj pojavljivanja reči Napisati program koji za svaku reč iz knjige (datoteka `knjiga.txt`) broji koliko se puta ona pojavljuje i rezultat upisuje u datoteku.

[Rešenje 5.4]

Zadatak 5.5 Uređaji transakcije U datoteci `uredjaji.txt` se nalaze podaci o kupljenim uređajima u formatu:

```
1 marka_uredjaja ostali_podaci
```

Napisati program koji izdvaja podatke o svim transakcijama jedne marke i upisuje ih u posebnu datoteku sa nazivom `ime_marke.txt`.

¹<http://maven.apache.org/>

²<http://www.scala-sbt.org/>

³<http://spark.apache.org/docs/latest/spark-standalone.html>

⁴<http://spark.apache.org/docs/latest/running-on-mesos.html>

⁵<http://spark.apache.org/docs/latest/running-on-yarn.html>

[Rešenje 5.5]

Zadatak 5.6 Log poruke Datoteka *log.txt* sadrži podatke koji su generisani pokretanjem Java programa. Napisati program koji izdvaja poruke koje se odnose na pakete jezika Java grupisane po tipu poruke i ispisuje ih na izlaz. Poruke mogu biti informacione, upozorenja ili greške. Format poruka je:

```
1 tip ostatak_poruke
```

Tip može biti [warn], [info] ili [error].

[Rešenje 5.6]

Zadatak 5.7 Uspešna preuzimanja U datoteci *mavenLog.txt* se nalaze podaci o započetim/uspešnim preuzimanjima paketa prilikom pokretanja Maven alata (upravljач zavisnostima) u formatu:

```
1 Downloading: ostatak_poruke
```

ili

```
1 Downloaded: ostatak_poruke
```

Napisati program koji računa procenat uspešnih preuzimanja paketa u odnosu na započeta preuzimanja.

[Rešenje 5.7]

Zadatak 5.8 Pokloni Bliži se Božić i firma želi da pokloni svojim zaposlenim programerima tri paketića. Sin direktora firme je budući programer i želi da napravi program koji će simulirati izvlačenje troje dobitnika paketića. Kako još uvek nije dobro savladao programiranje, pomozimo mu tako što ćemo napisati program koji nasumično bira tri programera i ispisuje njihova imena, prezimena i email.

Podaci o zaposlenima se nalaze u datoteci *zaposleni.txt* u formatu:

```
1 ime prezime pol identifikator IP_adresa_racunara datum_zaposlenja
   sifra_pozicije plata
```

Šifra pozicije programera je IT_PROG.

[Rešenje 5.8]

Zadatak 5.9 Prosečna temperatura U datoteci *temperaturaBoston.txt* se nalaze podaci o prosečnim temperaturama u Bostonu od 1995 do 2016 godine u Farenhajtima. Napisati program koji ispisuje prosečne temperature u Bostonu za svaku godinu od 1995 do 2016 godine posebno u Celzijusima.

Format podataka je:

```
1 mesec dan godina temperatura
```

[Rešenje 5.9]

5.1.3 Zadaci za vežbu

Zadatak 5.10 Napisati program koji učitava ceo broj n i ispisuje faktorijske svih brojeva od 1 do n .

Zadatak 5.11 Napisati program koji učitava ceo broj n i ispisuje sumu prvih n elemenata harmonijskog reda (podsetimo se, harmonijski red je red oblika $1 + 1/2 + 1/3 + 1/4 \dots$).

Zadatak 5.12 Napisati program koji racuna zbir dva vektora (pretpostavimo da su vektori uvek zadati ispravno tj. iste su dužine) i ispisuje ih na izlaz.

Vektori se nalaze u datotekama *vektor1.txt* i *vektor2.txt* u formatu:

```
1 a1, a2, a3, a4, ... an
```

Zadatak 5.13 Napisati program koji racuna broj pojavljivanja svake cifre 0-9 u datoteci *knjiga.txt* i ispisuje rezultat sortiran po ciframa.

Zadatak 5.14 Napisati program koji prebrojava sve poruke o greškama koje se odnose na Spark alat i rezultat ispisuje na izlaz. Poruke o greškama se nalaze u datoteci *log.txt* u formatu:

```
1 tip ostatak_poruke
```

Tip poruke o grešci je [error].

Zadatak 5.15 Napisati program koji izdvaja dane sa najvišom temperaturom u Bostonu za svaku godinu posebno, počev od 1995 do 2016 i rezultat upisuje u direktorijum *MaxTemp*. Podaci o temperaturama se nalaze u datoteci *temperaturaBoston.txt* u formatu:

```
1 mesec dan godina temperatura
```

Zadatak 5.16 Napisati program koji izdvaja podatke o veličini preuzetih paketa koji se odnose na Apache server i rezultat upisuje u direktorijum *ApacheDownloaded*. Podaci o preuzimanjima se nalaze u datoteci *mavenLog.txt* u formatu:

```
1 Downloaded: putanja podaci_o_velicini
```

Zadatak 5.17 Ekonomski analitičari žele da analiziraju tržište tehničke robe sa nekim rasponom cena. Napisati program koji učitava ime marke i bira nasumično pet poruka o transakcijama koje se odnose na tu marku. Podaci o transakcijama se nalaze u datoteci *uredjaji.txt* u formatu:

```
1 marka_uredjaja ostali_podaci
```

Zadatak 5.18 Napisati program koji računa prosečnu platu programera u firmi. Podaci o zaposlenima se nalaze u datoteci *zaposleni.txt* u formatu:

```
1 ime prezime pol identifikator IP_adresa_racunara datum_zaposlenja
   sifra_pozicije plata
```

Šifra pozicije programera je IT_PROG.

5.2 Rešenja

Rešenje 5.1 Parni kvadrati

```
1 import org.apache.spark.SparkConf
2 import org.apache.spark.SparkContext
3 import org.apache.spark.rdd.RDD._
4
5 object ParniKvadrati {
6
7     def main(args: Array[String]) = {
8
9         println("Unesite broj n:")
10        val n = Console.readInt()
11        /**
12         * Podesavamo konfiguraciju Spark okruzenja
13         * tako sto dajemo ime aplikaciji
14         * i dodeljujemo joj potencijalno 4 cvora
15         * (u nasem slucaju procesorska jezgra)
16     */
17    }
```



```

17     * */
18     val konf = new SparkConf()
19         .setAppName("ParniKvadrati")
20         .setMaster("local[4]")
21
22     /**
23     * Pravimo objekat Spark konteksta
24     * koji pokrece i upravlja Spark okruzenjem
25     * */
26     val sk = new SparkContext(konf)
27
28     /**
29     * Ukoliko zelimo da podesavamo parametre dinamicki
30     * (broj cvorova koji izvrsavaju nasu aplikaciju,
31     * velicina hip memorije i sl.)
32     * potrebno je da inicijalizujemo spark kontekst na sledeci nacin
33     *
34     * val sk = new SparkContext(new SparkConf());
35     *
36     * cime naznacavamo da ce se parametri konfiguracije
37     * podesiti dinamicki (koriscenjem spark-submit skripte).
38     * */
39
40     val niz = (2 to n by 2).toArray
41
42     /**
43     * Pravimo niz tipa RDD[Integer] od niza tipa Array[Integer]
44     * */
45     val nizRDD = sk.parallelize(niz)
46
47     /**
48     * Pravimo niz kvadrata parnih brojeva (uzimamo prvih 10)
49     * i rezultat pretvaramo u niz tipa Array[Integer]
50     * */
51     val nizKvadrata = nizRDD.map(x => x*x)
52                             /* .take(10) */
53                             .collect()
54
55     /**
56     * Zaustavljamo Spark okruzenje
57     * */
58     sk.stop()
59
60     /**
61     * Ispisujemo rezultujuci niz
62     * */
63     println("Niz kvadrata parnih brojeva: ")
64     println(nizKvadrata.mkString(", "))
65 }
66 }

```

Rešenje 5.2 Broj petocifrenih

```

1 import org.apache.spark.SparkConf
2 import org.apache.spark.SparkContext
3 import org.apache.spark.rdd.RDD._
4
5
6 object BrojPetocifrenih {
7
8     def main(args: Array[String]){
9
10        val konf = new SparkConf()
11            .setAppName("BrojPetocifrenih")
12            .setMaster("local[4]")
13
14        val sk = new SparkContext(konf)
15
16        /**
17        * Otvaramo datoteku i njen sadrzaj cuvamo
18        * u nizu tipa RDD[String].

```

```

20     * Elementi niza su pojedinačne linije iz datoteke.
21     * */
22     val datRDD = sk.textFile("brojevi.txt")
23
24     /**
25     * Filtriramo niz tako da nam ostanu samo petocifreni brojevi
26     * i prebrojavamo ih.
27     * */
28     val brojPetocifrenihBrojeva = datRDD.filter(_.length() == 5)
29                                         .count()
30
31     sk.stop()
32
33     println("Petocifrenih brojeva ima: ")
34     println(brojPetocifrenihBrojeva)
35 }
36 }

```

Rešenje 5.3 Skalarni proizvod

```

1  import org.apache.spark.SparkConf
2  import org.apache.spark.SparkContext
3  import org.apache.spark.rdd.RDD._
4
5  object SkalarniProizvod {
6
7      def main(args: Array[String]){
8
9          val konf = new SparkConf()
10             .setAppName("SkalarniProizvod")
11             .setMaster("local[4]")
12
13         val sk = new SparkContext(konf)
14
15         /**
16         * Otvaramo datoteku i učitavamo vektor.
17         * */
18         val vek1RDD = sk.textFile("vektor1.txt")
19             /**
20             * Razdvajamo elemente vektora iz niske koristeći separator ", "
21             * */
22             .flatMap(_.split(", "))
23             /**
24             * Koristimo niske u tip Integer.
25             * */
26             .map(_.toInt)
27
28         val vek2RDD = sk.textFile("vektor2.txt")
29             .flatMap(_.split(", "))
30             .map(_.toInt)
31
32         /**
33         * Spajamo nizove vektora A i B funkcijom zip
34         * i pravimo jedan niz parova (tipa Tuple)
35         * tako da svaki par sadrži element vektora A i element vektora B
36         * (a1,b1), (a2,b2), ... (an, bn).
37         * */
38         val skProizvod = vek1RDD.zip(vek2RDD)
39             /**
40             * Množimo elemente para.
41             * */
42             .map(par => par._1 * par._2)
43             /**
44             * Pomnožene elemente parova sabiramo.
45             * */
46             .reduce((a, b) => a+b)
47
48         sk.stop()
49
50         println("Skalarni proizvod je: ")
51         println(skProizvod)
52     }
53 }

```

Rešenje 5.4 Broj pojavljivanja reči

```

import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.rdd.RDD._

object BrojPojavljivanjaReci {

  def main(args: Array[String]){

    val konf = new SparkConf()
      .setAppName("BrojPojavljivanjaReci")
      .setMaster("local[4]")

    val sk = new SparkContext(konf)

    val knjigaRDD = sk.textFile("knjiga.txt")

    /**
     * Ucitavamo linije i razlazemo ih separatorom " " tako da dobijemo niz reci
     */
    val reciBr = knjigaRDD.flatMap(_.split(" "))
      /**
       * Od svake reci pravimo par (rec, 1).
       */
      .map(rec => (rec, 1))
      /**
       * Sabiramo sve vrednosti drugog elementa para
       * grupisane po prvom elementu para
       * koji nam predstavlja kljuc.
       */
      .reduceByKey((+_))
      /**
       * Sortiramo reci leksikografski.
       */
      .sortByKey()
      /**
       * Cuvamo ih u datotekama koje se nalaze u direktorijumu
       * BrojPojavljivanjaReci
       */
      .saveAsTextFile("BrojPojavljivanjaReci")

    sk.stop()
  }
}

```

Rešenje 5.5 Uredjaji transakcije

```

import org.apache.spark.SparkConf
import org.apache.spark.SparkContext
import org.apache.spark.rdd.RDD._
import java.io._

object UredjajiTransakcije {

  def main(args: Array[String]){

    val konf = new SparkConf()
      .setAppName("UredjajiTransakcije")
      .setMaster("local[4]")

    val sk = new SparkContext(konf)

    val transakcije = sk.textFile("uredjaji.txt")
    /**
     * Razdvajamo podatke o uredjajima
     * i pravimo parove transakcija
     * (marka, ostali_podaci)
     */
    .map(linija => {
      val niz = linija.split(" ")
      (niz(0), niz.drop(1).mkString(" "))
    })
  }
}

```

```

25         })
26         /**
27         * Grupisemo ih po marki tako da za svaku marku
28         * cuvamo niz obavljenih transakcija
29         * (tj. niz koji sadrzi ostale podatke za svaku transakciju)
30         */
31         .groupByKey()
32         /**
33         * Prolazimo kroz niz parova (marka, niz_transakcija)
34         * i u datoteku ime_marke.txt
35         * upisujemo podatke o transakcijama.
36         *
37         * Parametar t u foreach konstrukciji predstavlja jedan par
38         * (marka, niz_transakcija).
39         */
40         .foreach(t => {
41             val dat = new PrintWriter(new File(t._1.toLowerCase() +
42                 ".txt" ))
43
44             dat.write("---" + t._1 + "---\n")
45
46             t._2.foreach(por => {
47                 dat.append(por + "\n")
48             })
49
50             dat.close()
51         })
52
53     sk.stop()
54 }
55 }

```

Rešenje 5.6 Log poruke

```

1 import org.apache.spark.SparkConf
2 import org.apache.spark.SparkContext
3 import org.apache.spark.rdd.RDD._
4
5 object LogPoruke {
6
7     def main(args: Array[String]){
8
9         val konf = new SparkConf()
10            .setAppName("LogPoruke")
11            .setMaster("local[4]")
12
13         val sk = new SparkContext(konf)
14
15         val poruke = sk.textFile("log.txt")
16             /**
17             * Filtriramo podatke tako da nam ostanu
18             * samo linije koje predstavljaju
19             * upozorenja, informacije ili greske i odnose se na javu.
20             */
21         .filter(linija =>
22             (linija.contains("[warn]")
23             || linija.contains("[info]")
24             || linija.contains("[error]"))
25             && (linija.contains("java")))
26             /**
27             * Pravimo parove (tip_poruke, poruka).
28             */
29         .map(linija => {
30             val niz = linija.split(" ")
31             (niz(0), niz.drop(1).mkString(" "))
32         })
33         /**
34         * Grupisemo poruke po njihovom tipu (kljuc).
35         * tako da dobijemo niz parova (tip_poruke, niz_poruka).
36         */
37         .groupByKey()
38         /**

```

```

39         * Za svaki tip racunamo broj poruka tog tipa
40         * tako sto od parova (tip_poruke, niz_poruka)
41         * pravimo par (tip_poruke, broj_poruka)
42         * */
43         .map( por => (por._1, por._2.size))
44         .collect()
45
46     println("Informacije o log porukama koje se odnose na Javu: ")
47
48     poruke.foreach( por => println(" " + por._1 + ": " + por._2 ))
49
50     sk.stop()
51 }
}

```

Rešenje 5.7 Uspešna preuzimanja

```

1 import org.apache.spark.SparkConf
2 import org.apache.spark.SparkContext
3 import org.apache.spark.rdd.RDD._
4
5 object UspesnaPreuzimanja {
6
7     def main(args: Array[String]){
8
9         val konf = new SparkConf()
10            .setAppName("UspesnaPreuzimanja")
11            .setMaster("local[4]")
12
13        val sk = new SparkContext(konf)
14
15        /**
16         * Ucitavamo podatke i smestamo ih u kes memoriju radi brzog pristupanja.
17         * */
18        val preuzimanja = sk.textFile("mavenLog.txt")
19            .cache()
20
21        /**
22         * Racunamo broj zapocetih preuzimanja.
23         * */
24        val zapoceta = preuzimanja.filter(_.contains("Downloading:"))
25            .count()
26
27        /**
28         * Racunamo broj završenih preuzimanja.
29         * */
30        val završena = preuzimanja.filter(_.contains("Downloaded:"))
31            .count()
32
33        sk.stop()
34
35        println("%.2f".format(završena*100.0/zapoceta) + " procenata zapocetih
36        preuzimanja je završeno.")
37    }
38 }

```

Rešenje 5.8 Pokloni

```

1 import org.apache.spark.SparkConf
2 import org.apache.spark.SparkContext
3 import org.apache.spark.rdd.RDD._
4 import scala.compat._
5
6 object RandomZaposleni {
7
8     def main(args: Array[String]){
9
10        val konf = new SparkConf()
11            .setAppName("RandomZaposleni")
12            .setMaster("local[4]")
13
14        val sk = new SparkContext(konf)

```

```

15
16
17     /**
18     * Pronalazimo liniju koja sadrzi radnika programera,
19     * razdvajamo podatke o jednom radniku
20     * i cuvamo njegovo ime, prezime i email nalog.
21     * Nakon toga nasumicno biramo 3 programera.
22     *
23     * Funkcija takeSample kao argumente prihvata:
24     * - indikator da li zelimo izbora sa vraćanjem
25     * - broj uzoraka
26     * - pocetnu vrednost (seed) za slucajni generator
27     */
28     val triProgramera = sk.textFile("zaposleni.txt")
29     /**
30     * Pronalazimo liniju koja sadrzi radnika programera,
31     * */
32     .filter(_.contains("IT_PROG"))
33     /**
34     * Razdvajamo podatke o jednom radniku
35     * i cuvamo njegovo ime, prezime i email nalog.
36     * */
37     .map(linija => {
38         val niz = linija.split(" ")
39         (niz(0), niz(1), niz(3))
40     })
41     /**
42     * Nasumicno biramo 3 programera.
43     * Prvi parametar (false) oznacava
44     * izbor bez vraćanja a poslednji parametar
45     * predstavlja pocetnu vrednost (seed)
46     * generatora slucajnih brojeva.
47     */
48     .takeSample(false, 3, Platform.currentTime)
49
50     println("Tri zaposlena radnika u IT sektoru su: ")
51     triProgramera.foreach(prog => {
52         println("Ime i prezime: " + prog._1 + " " + prog._2 + "\n Email: " + prog._3
53             .toLowerCase()+"@firma.com")
54     })
55
56     sk.stop()
57 }

```

Rešenje 5.9 Prosečna temperatura

```

1 import org.apache.spark.SparkConf
2 import org.apache.spark.SparkContext
3 import org.apache.spark.rdd.RDD._
4
5 object ProsečnaTemperatura {
6
7     def main(args: Array[String]){
8
9         val konf = new SparkConf()
10            .setAppName("ProsečnaTemperatura")
11            .setMaster("local[4]")
12
13         val sk = new SparkContext(konf)
14
15         val tempRDD = sk.textFile("temperatureBoston.txt")
16         /**
17         * Pravimo torke (kljuc, vrednost) takve da je
18         * kljuc = godina
19         * vrednost = (mesec, dan, temperatura)
20         */
21         .map(linija => {
22             val niz = linija.split(" ")
23             (niz(3), (niz(1), niz(2), niz(4).toFloat))
24         })
25         /**
26         * Grupisemo torke po njihovom kljucu i za svaki kljuc racunamo

```

```

27         * sumu svih temperatura i broj temperatura koje smo sabrali.
28         *
29         * Funkcija aggregateByKey(pocetnaVrednostAkumulatora)(f1, f2)
30         * obradjuje niz parova (kljuc, vrednost)
31         * grupise ih po kljucu, akumulira vrednosti
32         * (inicijalna vrednost akumulatora
33         * se prosledjuje kao parametar)
34         * i kao rezultat vraca niz parova (k, akumuliranaVrednost).
35         *
36         * Funkcija f1(akumulator, vrednost) se primenjuje
37         * nad svim vrednostima koje se nalaze u jednom cvoru
38         * i rezultat se smesta u akumulator tog cvora
39         * Funkcija f2(akumulator1, akumulator2) se primenjuje
40         * nad svim izracunatim akumulatorima pojedinačnih cvorova
41         * i rezultat se smesta u globalni akumulator.
42         *
43         * U našem slučaju, akumulator će da sadrži dve vrednosti
44         * (sumaTemp, brojTemp) sa nulom kao početnom vrednošću.
45         * Funkcija f1(akumulator, vrednost)
46         * sabira vrednost (temperaturu)
47         * sa sumaTemp iz akumulatora, a brojTemp povećava za 1.
48         * Funkcija f2(akumulator1, akumulator2)
49         * sabira obe vrednosti ova dva akumulatora.
50         *
51         * Kao rezultat primene ove funkcije dobićemo niz parova
52         * (kljuc, (sumaTemp, brojTemp))
53         * u kojoj želimo da izračunamo prosečnu temperaturu.
54         * */
55         .aggregateByKey((0.0, 0))((ak, vr) => (ak._1 + vr._3, ak._2 +
56         1),
57                                     (a1, a2) => (a1._1 + a2._1, a1._2 + a2.
58         _2))
59         /**
60         * Pravimo niz parova (kljuc, prosečnaTemperatura)
61         * */
62         .map(st => (st._1, st._2._1/st._2._2))
63         .sortByKey()
64         .collect()
65         .foreach(st => println("Godine "
66                               + st._1
67                               + " prosečna temperatura je iznosila "
68                               + "%.2f".format(((st._2-32)/1.8))
69                               + " celzijusa. "))
70     sk.stop()
71 }

```

6

Logičko programiranje

Potrebno je imati instaliran B-Prolog na računaru.

Literatura:

- (a) <http://www.picat-lang.org/bprolog/>
- (b) <http://www.picat-lang.org/bprolog/download/manual.pdf>

6.1 Uvod

6.1.1 Uvodni primeri

Zadatak 6.1 U bazu znanja uneti informacije o životinjama i njihovim odnosima po pitanju veličine. Napisati pravilo koje omogućava da se proveri koja je od dve životinje veća, kao i da generiše sve životinje za koje je neka životinja veća.

```
1 % ovo je jednolinijski komentar
3 /*
4  ovo je viselinijijski
5  komentar
6  */
7
8 /*
9  Programi se cuvaju sa ekstenzijom .pro ili .pl (na sistemima gde ekstenzije imaju
10     globalno znacenje, kao sto je MS-Windows da ne bi bilo mesanja sa Perl programima
11     treba uvek koristiti .pro).
12  Interpreter se pokrece komandom bp. Naredbe:
13  help -- pomoc
14  compile('ime_programa') -- prevodi program i pravi izvrsni fajl ukoliko nema gresaka
15  load('ime_izvrsnog_fajla') -- uvozi izvrsni fajl
16  cl('ime_programa') -- compile + load
17  halt ili control+D -- za izlazak iz interpretera
18
19  Termovi: konstante, promenljive ili kompozitni termovi.
20  --- Konstante: atomi i brojevi.
21  ----- Atomi: stringovi od najvise 1000 karaktera koji pocinju malim slovom ('abc', '
22     a01', 'b_cd', 'l122k', ...).
23  ----- Brojevi: celi i realni.
24  --- Promenljive: imena pocinju velikim slovom ili podvlakom (_). Specijalna, anonimna
25     promenljiva: '_'.
26  --- Kompozitni (slozeni) termovi ili strukture: oblika f(t1, ..., tn) gde je f neka
27     funkcija arnosti n ( 0<n<32768), a t1, ..., tn termovi.
28
29  Program: sekvenca Hornovih klauza. Postoje tri tipa Hornovih klauza: cinjenice,
30     pravila i upiti.
31  --- Cinjenice: atomicka formula oblika p(t1, ..., tn) gde je p predikat arnosti n, a
32     t1, ..., tn termi. One opisuju svojstva i relacije izmedju objekata. Primer:
33     zivotinja(slon).
34     veci(zebra,vuk).
35  --- Pravila: imaju sledecu formu
```



```

29 H :- B1, ..., Bn. (n>0)
H, B1, ..., Bn su atomicne formule. H se zove GLAVA pravila, dok je sve sa desne
strane :- TELO pravila. Citamo ih kao implikaciju sa desna na levo: vazi H, ako
vaze B1, ..., Bn (" u pravilu zamenjuje logicko "i").
31 --- Cinjenice i pravila cine BAZU ZNANJA.
--- Upiti: konstrukcije kojima korisnik komunicira sa bazom znanja. Za ovo je
neophodan interpetator kome se postavlja upit. Primer:
33 ?- veci(slom, zec).
true.
35 ?- zivotinja(veverica).
false.
37 */

39 /* cinjenice, svojstva */
zivotinja(slom).
41 zivotinja(vuk).
zivotinja(zec).
43 zivotinja(zebra).

45 /* cinjenice, odnosi */
veci(slom,vuk).
47 veci(vuk,zec).
veci(slom,zebra).
49 veci(zebra,vuk).
veci(slom,zec).
51

53 /*
upiti sa promenljivama:
-- daje jedno resenje, ako zelimo da prikaze jos resenja kucamo ; nakon prikazanog ?
a za prekid control+C
55
?- veci(slom, X).
57 X = vuk ? ^C

59 | ?- veci(slom, X).
X = vuk ?;
61 X = zebra ?;
X = zec
63 yes

65 | ?- veci(X, Y).
X = slom
67 Y = vuk ?;
X = vuk
69 Y = zec ?;
X = slom
71 Y = zebra ?;
X = zebra
73 Y = vuk ?;
X = slom
75 Y = zec
yes
77 */

79 /*
pravilo : za neko X i Y vazi je_veci(X,Y) ako postoji Z tako da vazi veci(X,Z) i veci
(Z,Y)
81 */
je_veci(X,Y):-veci(X,Z),veci(Z,Y).
83 /*
| ?- je_veci(X,Y)
85 X = slom
Y = zec ?;
87 X = slom
Y = vuk ?;
89 X = zebra
Y = zec ?;
91 no
*/

```

Zadatak 6.2 Unifikacija. Jednakost.

/*

```

2
Provera tipa:
4
atom(X) - da li je term X atom
6
atomic(X) - da li je term X atom ili broj
number(X) - da li je term X broj
8
float(X) ili real(X) - da li je term X realan broj
integer(X) - da li je term X ceo broj
10
var(X) - da li je term X slobodna promenljiva
nonvar(X) - da li term X nije promenljiva
12
Primeri:
14
| ?- atom('abc')
yes
16
| ?- atomic(5)
yes
18
Unifikacija:
20
= unifikabilni
22
\= nisu unifikabilni
== identicno jednaki termovi
24
\== nisu identicno jednaki termovi
26
*/
28
uni(X, Y):- X = Y.
30
/*
32
| ?- uni(4,5)
no
34
| ?- uni(4,X)
X = 4
yes
36
*/
38
jed(X, Y):- X == Y.
40
/*
42
| ?- jed(4,X)
no
44
| ?- jed(4,5)
no
46
| ?- jed(4,4)
yes
*/

```

Zadatak 6.3 Aritmetički operatori. Operatori *is* i *cut*.

```

/*
2
is aritmeticko izracunavanje
3
:= aritmeticki jednaki
4
=\= aritmeticki nisu jednaki
5
<, =<, >, >=
6
+, -, *, /, // (celobrojno deljenje), div, mod, ** (stepenovanje)
7
*/
8
/*
10
Ako je X promenljiva, tada se njoj dodeljuje vrednost koju ima term Y (mora biti
    poznata vrednost), a ukoliko X nije promenljiva, X is Y se svodi na X := Y
11
*/
12
op1(X, Y):- X is Y.
13
/*
14
Termovima X i Y moraju biti poznate vrednosti, inace ce prijaviti gresku.
15
*/
16
op2(X, Y):- X := Y.
17
18
/*
19
| ?- op1(3,4)
20
no
21
| ?- op1(4,4)
22
yes
23
| ?- op1(X,4)

```

```

24 X = 4
   yes
26 | ?- op1(4,X)
   *** error
28 | ?- op2(4,4)
   yes
30 | ?- op2(4,2)
   no
32 | ?- op2(4,X)
   *** error
34 | ?- op2(X,4)
   *** error
36 */

38 /*
   apsolutna vrednost, prvi argument je broj za koji trazimo apsolutnu vrednost, a drugi
   promenljiva gde se smesta rezultat

40 losa implementacija, za pozitivne brojeve oba pravila prolaze

42 | ?- abs1(1,X)
44 X = 1 ?;
46 X = -1
48 yes
   | ?- abs1(-1,X)
48 X = 1
   yes
50 */

52 abs1(X, X):- X >= 0.
   abs1(X, Y):- Y is -X.
54

56 /*
   dobre implementacije abs2 i abs3

58 | ?- abs2(1,X)
60 X = 1 ?
   yes
   | ?- abs2(-1,X)
62 X = 1
   yes
64 */

66 abs2(X, X):- X >= 0.
   abs2(X, Y):- X < 0, Y is -X.

68 /*
   korisćenje operatora "cut" koji se oznacava sa "!" uklanjamo alternativne klauze,
   tako da se u slucaju pozitivnih neće primeniti drugi predikat cim uspe prvi

70 | ?- abs3(1,X)
72 X = 1
   yes
74 | ?- abs3(-1,X)
76 X = 1
   yes
78 */

80 abs3(X, X):- X >= 0, !.
   abs3(X, Y):- Y is -X.

```

Zadatak 6.4 Rekurzivni predikat, primer porodičnog stabla.

```

% porodicno stablo
2
% svojstva
4 musko(mihajlo).
   musko(stevan).
6 musko(petar).
   musko(mladen).
8 musko(rajko).
   zensko(milena).
10 zensko(milica).
   zensko(jelena).

```

```

12 zensko(senka).
   zensko(mina).
14 zensko(maja).

16 % odnosi
   roditelj(mihajlo,milica).
18 roditelj(mihajlo,senka).
   roditelj(milena,rajko).
20 roditelj(maja,petar).
   roditelj(maja,mina).
22 roditelj(stevan,mladen).
   roditelj(stevan,jelena).
24 roditelj(milica,mladen).
   roditelj(milica,jelena).
26

28 % pravila
   majka(X,Y):- roditelj(X,Y), zensko(X).
   otac(X,Y):- roditelj(X,Y), musko(X).
30 brat(X,Y):- musko(X), majka(Z,X), majka(Z,Y), X\==Y.
   sestra(X,Y):- zensko(X), majka(Z,X), majka(Z,Y), X\==Y.
32 ujak(X,Y):- brat(X,Z), majka(Z,Y).
   tetka(X,Y):- sestra(X,Z), majka(Z,Y).
34

36 % rekurzivno pravilo
36 % roditelj je predak
   predak(X,Y):- roditelj(X,Y).
38 % roditelj pretka je takodje predak
   predak(X,Y):- roditelj(X,Z), predak(Z,Y).

```

Zadatak 6.5 Anonimna promenljiva, primer prestupne godine.

```

1 % godina je prestupna ako je deljiva sa 4 i nije deljiva sa 100 ili je deljiva sa 400
   prestupna(X):- X mod 4 == 0, X mod 100 /= 0.
3 prestupna(X):- X mod 400 == 0.

5 /*
   anonimna promenljiva _ se koristi da oznaci da nam vrednost koja se prosledi za
   godinu nije bitna, moze biti bilo sta, ali tu vrednost ne koristimo
7
9 | ?- brdana(januar, _, X)
   X = 31
   yes
11 | ?- brdana(januar, 2017, X)
   X = 31
   yes
13 */
15 brdana(januar, _, 31).
   brdana(februar, X, 28):- not(prestupna(X)).
17 brdana(februar, X, 29):- prestupna(X).
   brdana(mart,_,31).
19 brdana(april,_,30).
   brdana(maj,_,31).
21 brdana(jun,_,30).
   brdana(jul,_,31).
23 brdana(avgust,_,31).
   brdana(septembar,_,30).
25 brdana(oktobar,_,31).
   brdana(novembar,_,30).
27 brdana(decembar,_,31).

```

6.1.2 Zadaci za samostalni rad sa rešenjima

Zadatak 6.6 Napisati sledeće predikate:

- maksimum(A, B, M) - određuje maksimum za dva broja A i B
- suma(N, S) - za dati prirodan broj N računa sumu prvih N brojeva
- sumaParnih(N, S) - za dati paran prirodan broj N računa sumu parnih brojeva od 2 do N

- d) $\text{proizvod}(N, P)$ - za dati prirodan broj N računa proizvod prvih N prirodnih brojeva
- e) $\text{proizvodNeparnih}(N, P)$ - za dati neparan prirodan broj N računa proizvod neparnih brojeva od 1 do N
- f) $\text{cifre}(N)$ - ispisuje cifre prirodnog broja N rečima

[Rešenje 6.6]

6.1.3 Zadaci za vežbu

Zadatak 6.7 Napisati sledeće predikate:

- a) $\text{sumaCifara}(N, SC)$ - određuje sumu cifara prirodnog broja N
- b) $\text{brojCifara}(N, BC)$ - određuje broj cifara prirodnog broja N
- c) $\text{maxCifra}(N, MC)$ - određuje maksimalnu cifru prirodnog broja N
- d) $\text{sumaKvadrata}(N, SK)$ - računa sumu kvadrata prvih N prirodnih brojeva
- e) $\text{fakt}(N, F)$ - računa faktorijel prirodnog broja N
- g) $\text{sumaDel}(X, D)$ - računa sumu pravih delilaca broja X

Zadatak 6.8 Ako su date činjenice oblika:

- $\text{ucenik}(\text{SifraUcenika}, \text{ImeUcenika}, \text{Odeljenje})$
- $\text{ocene}(\text{SifraUcenika}, \text{SifraPredmeta}, \text{Ocena})$
- $\text{predmet}(\text{SifraPredmeta}, \text{NazivPredmeta}, \text{BrojCasova})$

Napisati sledeće predikate:

- a) $\text{bar2PeticeSifra}(S)$ - određuje šifru S učenika koji ima bar dve petice iz različitih predmeta
- b) $\text{bar2PeticeIme}(X)$ - određuje ime X učenika koji ima bar dve petice iz različitih predmeta
- c) $\text{odeljenjePetice}(X, Y)$ - određuje odeljenje X u kome postoje bar dve petice iz predmeta sa šifrom Y

Zadatak 6.9 Ako su date činjenice oblika:

- $\text{film}(\text{NazivFilma}, \text{ZanrFilma}, \text{ImeReditelja}, \text{SifraGlumca})$
- $\text{glumac}(\text{SifraGlumca}, \text{ImeGlumca}, \text{GodRodj}, \text{MestoRodj})$

Napisati sledeće predikate:

- a) $\text{filmskiUmetnik}(X)$ - X je filmski umetnik ako je reditelj nekog filma i igra u nekom filmu
- b) $\text{glumacBarDva}(X)$ - određuje ime glumca X koji igra u bar dva različita filma
- c) $\text{opstiGlumac}(X)$ - određuje ime glumca X koji igra u bar dva filma različitog žanra
- d) $\text{zanrovskiGlumac}(X, Y)$ - određuje ime glumca X koji igra u filmu žanra Y

6.2 Liste

6.2.1 Uvodni primeri

Zadatak 6.10 Osnovni pojmovi i predikati za rad sa listama.

```

1  /*
   Lista - niz uredjenih elemenata, tj. termova.
3  Lista moze biti:
   [] - prazna
5  .(G,R) - struktura, gde je G ma koji term i naziva se glava liste, a R lista i
   naziva se rep liste

7  Primeri:
   [] - prazna
9  .(a, []) - jednoclana lista, gde je a bilo koji term
   .(a, .(b, [])) - dvoclana lista, gde su a i b termi ...

11 Zapis pomocu zagrada (prvi element predstavlja glavu, a ostali cine listu koja je rep
   ):
13 [a,b,c] <=> .(a, .(b, [c]))

15 Zapis liste u kom su jasno razdvojeni glava i rep (pogodan za unifikaciju): [G|R].

17 Primeri unifikacije listi:
[X, Y, Z] [jabuka, kruska, banana] -----> X = jabuka, Y = kruska, Z = banana
19 [racunar] [X|Y] -----> X = racunar, Y = []
[maja, ana, jovana] [X, Y|Z] -----> X = maja, Y = ana, Z = [jovana]

21 */

23 % predikat proverava da li element pripada listi (ako joj pripada jednak je glavi ili
   nekom elementu iz repa liste)
25 sadrzi(X, [X|_]) :- !.
   sadrzi(X, [G|R]) :- G \== X, sadrzi(X, R).

27 % drugi nacin, predikat kao disjunkcija:
29 % sadrzi(X, [G|R]) :- G == X; sadrzi(X, R).

31 % predikat koji racuna duzinu liste (prazna je duzine nula, nepraznu dekomponujemo na
   glavu i rep, pa je duzina liste = 1 + duzina repa)
duzina([], 0).
33 duzina([G|R], L) :- duzina(R,L1), L is L1+1.

35 % predikat racuna sumu elemenata liste brojeva
suma([], 0).
37 suma([G|R], S) :- number(G), suma(R, S1), S is S1+G.

39 % predikat racuna aritmeticku sredinu elemenata liste brojeva
arsr([],0).
41 % ako ne koristimo sablon za nepraznu listu [G|R], moramo proveriti da li je K nula
   jer se sada L moze unifikovati sa []
arsr(L, A) :- duzina(L, K), K \= 0, suma(L, S), A is S/K.

43 % predikat ucitava listu duzine N ciji elementi mogu biti proizvoljni termovi
45 % za negativno N ne ucitava listu
ucitaj(N,_) :- N < 0, !.
47 % za nulu vraca praznu listu
ucitaj(0, []).
49 % read(X) ucitanu vrednost sa ulaza dodeljuje promenljivoj X
   % ako je N > 1, lista ima glavu i rep, ucitamo glavu, pa pozovemo predikat za
   ucitavanje repa
51 ucitaj(N, [G|R]) :- N > 1, write('unesi element '), read(G), nl, M is N-1, ucitaj(M,R)

53 /*
   prilikom unosa vrednosti obavezna je tacka kao oznaka kraja ulaza za read
55 | ?- ucitaj(3,L)
   unesi element | 5.
57
   unesi element | 4.
59

```

```
61 unesi element | 3.  
L = [5,4,3]  
63 yes  
65 | ?- ucitaj(3,L)  
unesi element | [1,2,3].  
67  
69 unesi element | 4.  
unesi element | [].  
71  
73 L = [[1,2,3],4,[]]  
yes  
75 */
```

6.2.2 Zadaci za samostalni rad sa rešenjima

Zadatak 6.11 Napisati sledeće predikate:

- a) dodajPocetak(X, L, NL) - dodaje X na početak liste L
- b) dodajKraj(X, L, NL) - dodaje X na kraj liste L
- c) obrisiPrvi(L, NL) - briše prvi element, tj. glavu liste
- d) obrisiPoslednji(L, NL) - briše poslednji element liste
- e) obrisi(X, L, NL) - briše sva pojavljivanja elementa X u listi L
- f) obrisiPrvo(X, L, NL) - briše samo prvo pojavljivanje elementa X u listi L
- g) obrisiK(L, K, NL) - briše K -ti element liste L

[Rešenje 6.11]

Zadatak 6.12 Napisati predikat `podeli(L, L1, L2)` koji deli listu L na dve liste, listu pozitivnih elemenata $L1$ i listu negativnih elemenata $L2$.

[Rešenje 6.12]

Zadatak 6.13 Napisati predikat `pretvori(L, X)` koji za datu listu cifara L formira broj određen tim ciframa.

[Rešenje 6.13]

Zadatak 6.14 Napisati predikat `maxEl(L, X)` koji određuje maksimalni element liste L .

[Rešenje 6.14]

Zadatak 6.15 Napisati predikat `spoji(L1, L2, L)` koji spaja dve rastuće sortirane liste $L1$ i $L2$ u treću tako da i ona bude sortirana rastuće.

[Rešenje 6.15]

6.2.3 Zadaci za vežbu

Zadatak 6.16 Napisati predikat `parNepar(L, L1, L2)` koji deli listu L na dve liste, listu parnih elemenata $L1$ i listu neparnih elemenata $L2$.

Zadatak 6.17 Napisati predikat `podeli(L, N, L1, L2)` koji deli listu L na dve liste $L1$ i $L2$, pri čemu je zadata dužina prve liste $L1$.

Zadatak 6.18 Napisati predikat `ogledalo(L1, L2)` koji proverava da li je lista L1 jednaka obrnutoj listi liste L2.

Zadatak 6.19 Napisati predikat `interval(X, Y, L)` koji kreira listu L koja sadži sve cele brojeve iz intervala zadanog sa prva dva argumenta.

Zadatak 6.20 Napisati predikat `skalar(L1, L2, S)` koji određuje skalarni proizvod dva vektora, tj. listi brojeva L1 i L2.

6.3 Razni zadaci

6.3.1 Zadaci sa rešenjima

Zadatak 6.21 Ako su date činjenice oblika:

- `stan(Porodica, KvadraturaStana)`
- `clan(Porodica, BrojClanova)`

Napisati predikat `poClanu(Porodica, Prosek)` koji određuje prosečan broj kvadrata stana po članu porodice koja živi u njemu.

[Rešenje 6.21]

Zadatak 6.22 Ako je data baza znanja:

- `automobil(SifraAutomobila, NazivAutomobila)`
- `vlasnik(ImeVlasnika, SifraAutomobila)`
- `brziSifra(SX, SY)` - automobil šifre SX je brži od automobila šifre SY

Napisati predikate:

- a) `brziNaziv(X, Y)` - automobil naziva X je brži od automobila naziva Y
- b) `imaAutomobil(X)` - X je vlasnik nekog automobila
- c) `imaBrzi(X, Y)` - X je vlasnik bržeg automobila od onog čiji je vlasnik Y.

[Rešenje 6.22]

Zadatak 6.23 Napisati program koji rešava sledeću zagonetku. Postoji pet kuća, svaka različite boje u kojoj žive ljudi različitih nacionalnosti koji piju različita pića, jedu različita jela i imaju različite kućne ljubimce. Važi sledeće:

- Englez živi u crvenoj kući
- Španac ima psa
- kafa se pije u zelenoj kući
- Ukrajinac pije čaj
- zelena kuća je odmah desno uz belu
- onaj koji jede špagete ima puža
- pica se jede u žutoj kući
- mleko se pije u srednjoj kući
- Norvežanin živi u prvoj kući s leva
- onaj koji jede piletinu živi pored onoga koji ima lisicu

- pica se jede u kući koja je pored kuće u kojoj je konj
- onaj koji jede brokoli pije sok od narandže
- Japanac jede suši
- Norvežanin živi pored plave kuće

Čija je zebra, a ko pije vodu?

[Rešenje 6.23]

Zadatak 6.24 Napisati program koji rešava sledeću zagonetku. Svakog vikenda, Milan čuva petoro komšijske dece. Deca se zovu Kata, Lazar, Marko, Nevenka i Ognjen, a prezivaju Filipović, Grbović, Hadžić, Ivanović i Janković. Svi imaju različit broj godina od dve do šest. Važi sledeće:

- jedno dete se zove Lazar Janković
- Kata je godinu dana starija od deteta koje se preziva Ivanović koje je godinu dana starije od Nevenke
- dete koje se preziva Filipović je tri godine starije od Marka
- Ognjen je duplo stariji od deteta koje se preziva Hadžić

Kako se ko zove i koliko ima godina?

[Rešenje 6.24]

6.3.2 Zadaci za vežbu

Zadatak 6.25 Napisati predikat `uzastopni(X, Y, Z, L)` koji proverava da li su prva tri argumenta uzastopni elementi u listi L.

Zadatak 6.26 Napisati predikat `kompresuj(L, KL)` koji u datoj listi L eliminiše uzastopne duplikate.

Zadatak 6.27 Napisati predikat `prefiksi(L, P)` koji određuje sve liste koje su prefiksi date liste L.

Zadatak 6.28 Napisati predikat `sufiksi(L, S)` koji određuje sve liste koje su sufiksi date liste L.

Zadatak 6.29 Napisati predikat `opadajuće(N, L)` koji za dat prirodan broj N formira listu brojeva od N do 1.

Zadatak 6.30 Napisati predikat `form(N, L)` kojim se formira lista od prirodnih brojeva deljivih sa 5 i manjih od datog prirodnog broja N.

Zadatak 6.31 Napisati program koji rešava sledeću zagonetku. Četiri žene se zovu Petra, Milica, Lenka i Jovana, a prezivaju Perić, Mikić, Lazić i Jović. One imaju četiri kćerke koje se takodje zovu Petra, Milica, Lenka i Jovana. Važi sledeće:

- nijedna majka nema prezime koje počinje istim slovom kao ime
- nijedna kćerka nema prezime koje počinje istim slovom kao ime
- nijedna kćerka se ne zove kao majka
- majka koja se preziva Perić se zove isto kao Miličina kćerka
- Lenkina kćerka se zove Petra

Određiti imena majki i kćerki.

Zadatak 6.32 Napisati program koji rešava sledeću zagonetku. Četiri para je došlo na maskenbal:

- Markova žena se maskirala kao macka
- dva para su stigla pre Marka i njegove žene, a jedan muskarac je bio maskiran u medveda
- prvi koji je stigao nije bio Vasa, ali je stigao pre onoga koji je bio maskiran u princa
- žena maskirana u vešticu (nije Bojana) je udata za Peru, koji se maskirao kao Paja patak
- Marija je došla posle Laze, a oboje su stigli pre Bojane
- žena maskirana u Ciganku je stigla pre Ane, pri čemu nijedna od njih nije udata za muškarca maskiranog u Betmena
- žena maskirana u Snežanu je stigla posle Ivane

Odrediti kako je bio obučen koji par.

Zadatak 6.33 Izračunavanje vrednosti aritmetičkog izraza korišćenjem listi možete pogledati ovde:

https://rosettacode.org/wiki/Arithmetic_evaluation#Prolog

Zadatak 6.34 Implementacije raznih problema u Prologu možete pogledati ovde:

<https://rosettacode.org/wiki/Category:Prolog>

6.4 Rešenja

Rešenje 6.6

```

1  % maksimum dva broja
   % I nacin:
3  maksimum(A,B,M):- A>=B, M is A.
   maksimum(A,B,M):- A<B, M is B.
5  % II nacin bez trece promenjive:
   % maksimum(A,B,A):- A>=B.
7  % maksimum(A,B,B):- A<B.

9  % suma prvih N prirodnih brojeva
   suma(1,1).
11 suma(N,S):- N>1, N1 is N-1, suma(N1,S1), S is S1+N.

13 % suma parnih prirodnih brojeva od 2 do N
   % moze se dodati provera N mod 2 := 0 u pravilu, ali i bez toga sam prepoznaje za
   % neparne da je netacan upit jer rekurzijom dodje do sumaParnih(1,S) sto je netacna
   % cinjenica u bazi
15 sumaParnih(2,2).
   sumaParnih(N,S):- N>2, N1 is N-2, sumaParnih(N1,S1), S is S1+N.

17 % proizvod prvih N prirodnih brojeva
19 proizvod(1,1).
   proizvod(N,P):- N>1, N1 is N-1, proizvod(N1,P1), P is P1*N.

21 % proizvod neparnih prirodnih brojeva od 1 do N
23 proizvodNeparnih(1,1).
   proizvodNeparnih(N,P):- N>1, N1 is N-2, proizvodNeparnih(N1,P1), P is P1*N.

25 % ispis cifara unetog prirodnog broja N
27 cifra(0, nula).
   cifra(1, jedan).
29 cifra(2, dva).
   cifra(3, tri).
31 cifra(4, cetiri).
   cifra(5, pet).
33 cifra(6, sest).
   cifra(7, sedam).
35 cifra(8, osam).
   cifra(9, devet).

37 % ukoliko nije prirodan broj, cut operatorom sprecavamo poziv poslednjeg predikata

```

```

39 cifre(N):- N < 1, !.
41 % ukoliko je jednocifren svodi se na poziv predikata cifra
42 % write(t) gde je t neki term, ispisuje term
43 % nl (newline) - ispisuje se novi red
44 % obratiti paznju na upotrebu cut operatora ! - sprečavamo poziv poslednjeg predikata
45 % za jednocifrene
46 cifre(N):- N > 1, N < 10, cifra(N, C), write(C), nl, !.
47 % ukoliko nije jednocifren, racunamo tekucu cifru koju ispisujemo i ostatak broja za
48 % koji se ponovo poziva predikat
49 cifre(N):- N1 is (N // 10), cifre(N1), N2 is (N mod 10), cifra(N2, C), write(C), nl.

```

Rešenje 6.11

```

1 % dodaje element na pocetak liste
2 dodajPocetak(X, L, [X|L]).
3
4 % dodaje element na kraj liste
5 dodajKraj(X, [], [X]).
6 dodajKraj(X, [G|R], [G|LR]):- dodajKraj(X, R, LR).
7
8 % brise prvi element liste
9 % za praznu listu ce uvek vratiti no jer ne moze unifikovati sa sablonom [G|R], a
10 % mozemo napraviti i sami za taj slucaj da je predikat netacan
11 % fail je uvek netacan pa ce nam ovako definisan predikat za slucaj prazne liste
12 % vratiti no
13 obrisiPrvi([], _):- fail.
14 obrisiPrvi([_|R], R).
15
16 % brise poslednji element liste
17 obrisiPoslednji([],_):- fail.
18 % bitan cut operator jer se jednoclana moze upariti sa sablonom [G|R]
19 obrisiPoslednji([_|R],[_]):- !.
20 obrisiPoslednji([G|R], [G|R1]):- obrisiPoslednji(R, R1).
21
22 % brise element X iz liste ako postoji (svako pojavljivanje elementa X)
23 obrisi(_, [], []).
24 obrisi(X, [X|R], R1):- obrisi(X, R, R1), !.
25 obrisi(X, [G|R], [G|R1]):- G \== X, obrisi(X, R, R1).
26
27 % brise element X iz liste ako postoji (samo prvo pojavljivanje elementa X)
28 obrisiPrvo(X, [], []).
29 obrisiPrvo(X, [X|R], R):- !.
30 obrisiPrvo(X, [G|R], [G|R1]):- G \== X, obrisiPrvo(X, R, R1).
31
32 % brise K-ti element liste, brojimo od 1, ako je K vece od duzine liste, treci
33 % argument je jednak prvom
34 obrisiK([], K, []):- K > 0.
35 obrisiK([G|R], 1, R):- !.
36 obrisiK([G|R], K, [G|R1]):- K>1, K1 is K-1, obrisiK(R, K1, R1).
37
38 % druga varijanta predikata: brise K-ti element liste, broji od 1, ali ukoliko zadata
39 % lista nema K-ti element, predikat vraca no kao odgovor i nema unifikacije za
40 % treci argument
41 % obrisiK([], K, []):- fail.
42 % obrisiK([G|R], 1, R):- !.
43 % obrisiK([G|R], K, [G|R1]):- K>1, K1 is K-1, obrisiK(R, K1, R1).

```

Rešenje 6.12

```

1 % podeli1 - deli listu na dve liste - listu pozitivnih i listu negativnih elemenata
2 % L1 - lista pozitivnih, L2 - lista negativnih
3 podeli([], [], []).
4 podeli([G|R], [G|R1], L2):- G >= 0, podeli(R, R1, L2), !.
5 podeli([G|R], L1, [G|R2]):- G < 0, podeli(R, L1, R2).

```

Rešenje 6.13

```

2 % potreban nam je dodatni predikat za izdvajanje poslednjeg elementa liste
  izdvojPoslednji([G], G, []):- !.
  izdvojPoslednji([G|R], X, [G|R1]):- izdvojPoslednji(R, X, R1).
4
  % formira broj od date liste cifara
6  pretvori([], 0):- !.
  pretvori(L, X):- izdvojPoslednji(L, Poslednji, Ostatak),
8     pretvori(Ostatak, Y),
     X is Poslednji + 10 * Y.
10
/*
12 | ?- pretvori([7,0,7], X)
  X = 707
14 yes
  | ?- pretvori([0,2,3], X)
16 X = 23
  yes
18 | ?- pretvori([], X)
  X = 0
20 yes
22 */

```

Rešenje 6.14

```

2 % maksimalni element liste
  maxEl([X], M):- M is X, !.
  % pozivamo za rep (idemo u dubinu), pa poredimo maksimalni element repa i glavu liste
4  maxEl([G|R], X):- maxEl(R, Y), G < Y, X is Y, !.
  maxEl([G|R], X):- maxEl(R, Y), G >= Y, X is G.

```

Rešenje 6.15

```

2 % poredjenjem glava listi zakljucujemo u kom redosledu dodajemo elemente u spojenu
  listu
  spoji([], L, L):- !.
  spoji(L, [], L):- !.
4  spoji([G1|R1], [G2|R2], [G1|R]):- G1<G2, spoji(R1, [G2|R2], R), !.
  spoji([G1|R1], [G2|R2], [G2|R]):- G1>=G2, spoji([G1|R1], R2, R).

```

Rešenje 6.21

```

1 % cinjenice
  stan(petrovic, 76).
3  stan(ciric, 93).
  stan(aleksic, 55).
5  stan(lisic, 123).
  stan(peric, 67).
7
  clan(ciric, 3).
9  clan(peric, 5).
  clan(aleksic, 2).
11 clan(lisic, 3).
  clan(petrovic, 4).
13
  % koristimo operator za realno deljenje / (// je za celobrojno, ne mesati ova dva
  operatora) da bi se dobio tacan rezultat
15 poClanu(Porodica, Prosek):- stan(Porodica, X), clan(Porodica, Y), Prosek is X/Y.

```

Rešenje 6.22

```

2 % baza znanja
  automobil(a1, audi).
  automobil(h1, honda).
4  automobil(m1, mercedes).
  automobil(m2, mercedes).

```

```

6 automobil(c1, citroen).
  automobil(c2, citroen).

8
10 vlasnik(milan, h1).
  vlasnik(maja, m1).
  vlasnik(nemanja, m2).
12 vlasnik(aleksandar, a1).
  vlasnik(andjela, c1).
14 vlasnik(petar, c2).

16 brziSifra(a1, c1).
  brziSifra(m1, c1).
18 brziSifra(m2, h1).
  brziSifra(a1, c2).

20 brziNaziv(X, Y):- automobil(SX, X), automobil(SY, Y), brziSifra(SX, SY).

22 imaAutomobil(X):- vlasnik(X, _).

24 imaBrzi(X, Y):- vlasnik(X, S1), vlasnik(Y, S2), brziSifra(S1, S2).

```

Rešenje 6.23

```

/*
2 strukturama oblika k(boja, nacionalnost, jelo, pice, kucniLjubimac) opisujemo date
  cinjenice, a u listi L su kuće poredjane jedna pored druge, tako da po redosledu
  u listi imamo informaciju da li je kuća desno od neke druge kuće i da li su kuće
  jedna pored druge
*/
4
/*
6 pomocni predikat koji proverava da li je X clan liste, njemu prosledjujemo strukturu
  k sa poznatim vrednostima iz teksta i opisujemo kakva kuća treba da bude u listi
*/
8 clan(X, [X|_]).
  clan(X, [_|R]):- clan(X,R).

10 % predikat smesta u listu L kuće koje zadovoljavaju uslove iz teksta, tj. predikat L
  unifikuje sa rešenjem zagonetke
12 % u listu ubacujemo cinjenice koje su vezane za raspored kuća i to samo one koje
  jednoznacno odredjuju poziciju kuće u listi
kuce(L):- L = [ k(_,norvezanin,_,_,_),
14   k(plava,_,_,_,_),
   k(,_,_,mleko,_,),
16   k(,_,_,_,_),
   k(,_,_,_,_) ],
18   % dodajemo kakve sve kuće treba da budu u listi
   clan(k(crvena, englez,_,_,_),L),
   clan(k(, spanac,_,_,pas),L),
20   clan(k(zelena,_,_,kafa,_,),L),
   clan(k(, ukrajinac,_,caj,_,),L),
22   % kada informacija daje relaciju za neke dve kuće iz liste koristimo predikate
   desno i pored
24   desno(k(zelena,_,_,kafa,_,),k(bela,_,_,_,_),L),
   clan(k(,_,spagete,_,puz),L),
26   clan(k(zuta,_,pica,_,_),L),
   pored(k(,_,piletina,_,_),k(,_,_,_,lisica),L),
28   pored(k(,_,pica,_,_),k(,_,_,_,konj),L),
   clan(k(,_,brokoli,narandza,_,),L),
30   clan(k(, japanac,susi,_,_),L),
   % medju datim informacijama se ne pominje zebra niti voda, ali posto je krajnje
   pitanje vezano za ove pojmove, moramo dodati da takvi clanovi treba da postoje u
   resenju zagonetke
32   clan(k(,_,_,_,zebra),L),
   clan(k(,_,_,voda,_,),L).

34
36 % proverava da li su kuće X i Y jedna pored druge u listi L
  pored(X,Y,[X,Y|_]).
  pored(X,Y,[Y,X|_]).
38 pored(X,Y,[_|R]):- pored(X,Y,R).

40 % proverava da li je kuća X desno od kuće Y u listi L

```

```

desno(X,Y,[Y,X|_]).
42 desno(X,Y,[_|R]):- desno(X,Y,R).

44 % predikat zagonetka daje odgovor na pitanje cija je zebra, a ko pije vodu, tako sto
    prvo trazi resenje zagonetke pa iz njega izdvaja samo potrebne clanove
zagonetka(X,Y):- kuce(L), clan(k(_,X,_,_,zebra),L), clan(k(_,Y,_,voda,_),L) .

46
/*
48 resenje:
50 | ?- kuce(L)
    L = [k(zuta,norvezanin,pica,voda,lisica),k(plava,ukrajinac,piletina,caj,konj),k(
        crvena,englez,spagete,mleko,puz),k(bela,spanac,brokoli,narandza,pas),k(zelena,
        japanac,susi,kafa,zebra)]
52
    odgovor na pitanje:
54 | ?- zagonetka(X,Y)
56 X = japanac
    Y = norvezanin
58
*/

```

Rešenje 6.24

```

/*
2 resenje je lista L sa 5 struktura oblika: d(ime, prezime, godine)
*/

4
% predikat kojim se proverava da li je X clan liste
6 clan(X, [X|_]).
clan(X, [_|R]):- clan(X,R).

8 % predikat smesta u listu L decu koja zadovoljavaju uslove iz teksta
% redosled dece u resenju nije bitan
10 % direktno u listi mozemo naznaciti ili broj godina, ili data imena ili data
    prezimena, ovde su izabrane godine, a ostale informacije dodate preko predikata
    clan
deca(L):- L = [d(_,_,2), d(_,_,3), d(_,_,4), d(_,_,5), d(_,_,6)],
12     clan(d(lazar,jankovic,_),L),
    clan(d(kata,_,G1),L),
14     clan(d(_,ivanovic,G2),L),
    clan(d(nevenka,_,G3),L),
16     clan(d(_,filipovic,G4),L),
    clan(d(marko,_,G5),L),
18     clan(d(ognjen,_,G6),L),
    clan(d(_,hadzic,G7),L),
20     clan(d(_,grbovic,_),L),
    % medju informacijama imamo odnos izmedju broja godina odredjene dece, ali ne
    znamo tacno koliko godina imaju ta deca, zbog toga koristimo promenljive kojima
    opisujemo date relacije, obratiti paznju da je potrebno koristiti aritmeticko
    poredjenje =:=, a ne poredjenje na identicnost (==), jer ne zelimo da se porede
    termini oblika Gi i Gj+N vec njihove brojevne vrednosti prilikom izgradjivanja
    resenja
22     G1:=G2+1, G2:=G3+1, G4:=G5+3, G6:=2*G7.

24
/*
26 resenje:
28 | ?- deca(L)
    L = [d(marko,hadzic,2),d(nevenka,grbovic,3),d(ognjen,ivanovic,4),d(kata,filipovic,5),
        d(lazar,jankovic,6)] ?yes
30
*/

```


7

Programiranje ograničenja - Prolog

Potrebno je imati instaliran B-Prolog na računaru.

Literatura:

- (a) <http://www.picat-lang.org/bprolog/>
- (b) <http://www.picat-lang.org/bprolog/download/manual.pdf>

7.1 Programiranje ograničenja

7.1.1 Uvodni primeri

Zadatak 7.1 Osnovni pojmovi i postavka problema.

```
1 /*
3 Programiranje ogranicenja nad konacnim domenom:
4 1) generisanje promenljivih i njihovih domena
5 2) generisanje ogranicenja nad promenljivima
6 3) instanciranje promenljivih ili obelezavanje
7
8 Definisanje domena (D) promenljivih:
9 -- X in D ili X :: D -> promenljiva X uzima samo vrednosti iz konacnog domena D
10 -- Vars in D ili Vars :: D -> sve promenjive iz liste Vars uzimaju samo vrednosti iz
11 konacnog domena D
12 Domen se definise kao interval u obliku Pocetak..Korak..Kraj (Korak je opcion i
13 ukoliko se ne navede, podrazumeva se da je Korak = 1)
14 Primeri za domen:
15 1..10 -> 1,2,3,4,5,6,7,8,9,10
16 1..2..10 -> 1,3,5,7,9
17
18 Osnovni predikati za ogranicenja
19
20 1) opsta:
21 -- alldifferent(Vars) ili alldistinct(Vars) - sve vrednosti razlicite u listi termova
22 Vars
23 -- atmost(N,L,V) - najvise N elemenata iz skupa L jednako sa V (N mora biti Integer,
24 V term, a L lista termova)
25 -- atleast(N,L,V) - najmanje N elemenata iz skupa L jednako sa V (N mora biti Integer
26 , V term, a L lista termova)
27 -- excatly(N,L,V) - tacno N elemenata iz skupa L jednako sa V (N mora biti Integer, V
28 term, a L lista termova)
29
30 2) aritmeticka:
31 -- E1 R E2 gde su E1 i E2 aritmeticki izrazi, a R iz skupa {#=#, #\=#, #>=#, #>, #=<,
32 #<}
33 -- min(L) - minimalni element iz liste termova L
34 -- max(L) - maksimalni element iz liste termova L
```



```

29 -- max(E1, E2)/min(E1, E2) -- manji/veci od izraza E1 i E2
-- sum(L) - suma elemenata liste termova L

31 Instanciranje i prikazivanje promenljivih: labeling(Vars).
Funkcija labeling se moze pozvati i sa razlicitim opcijama u obliku labeling(Options,
    Vars)
33 gde je Options lista opcija. Ukoliko se pozove sa labeling(Vars), podrazumevano je
    Options = []. Neke od opcija:
-- minimize(E) - trazi instance za Vars pri kojima je vrednost celobrojnog izraza E
    minimalna
35 -- maximize(E) - trazi instance za Vars pri kojima je vrednost celobrojnog izraza E
    maksimalna

37 */
39 /*
Primer: X pripada skupu {1,2,3}, Y skupu {2,4,6,8,10}, Z skupu {5,6,7,8} i vazi Z>=Y
41 */

43 primer(Vars) :- Vars = [X, Y, Z], % generisanje promenljivih
    X :: 1..3, % definisanje domena
45 Y :: 2..2..10,
    Z :: 5..8,
47 Z #>= Y, % ogranicenje
    labeling(Vars). % instanciranje

49 /*
51 | ?- primer(Vars).
Vars = [1,2,5] ?;
53 Vars = [1,2,6] ?;
Vars = [1,2,7] ?;
55 Vars = [1,2,8] ?;
Vars = [1,4,5] ?;
57 Vars = [1,4,6] ?;
Vars = [1,4,7] ?;
59 Vars = [1,4,8] ?;
Vars = [1,6,6] ?;
61 Vars = [1,6,7] ?;
Vars = [1,6,8] ?;
63 Vars = [1,8,8] ?;
...
65 */

67 /*
Primer: ispisati sve brojeve od 1..100 koji su puni kvadrati
69 */

71 puni(Vars) :- Vars = [X],
    Vars :: 1..100,
73 Y*Y #= X ,
    labeling(Vars).

75 /*
77 | ?- puni(Vars)
Vars = [1] ?;
79 Vars = [4] ?;
Vars = [9] ?;
81 Vars = [16] ?;
Vars = [25] ?;
83 Vars = [36] ?;
Vars = [49] ?;
85 Vars = [64] ?;
Vars = [81] ?;
87 Vars = [100] ?;
no
89 */
*/

```

7.1.2 Zadaci za samostalni rad sa rešenjima

Zadatak 7.2 Napisati program koji pronalazi petocifren broj ABCDE za koji je izraz $A+2*B-3*C+4*D-5*E$

minimalan i A, B, C, D i E su različite cifre.

[Rešenje 7.2]

Zadatak 7.3 Dati su novčići od 1, 2, 5, 10, 20 dinara. Napisati program koji pronalazi sve moguće kombinacije tako da zbir svih novčića bude 50 i da se svaki novčić pojavljuje bar jednom u kombinaciji.

[Rešenje 7.3]

Zadatak 7.4 Napisati program koji reda brojeve u magičan kvadrat. Magičan kvadrat je kvadrat dimenzija 3x3 takav da je suma svih brojeva u svakom redu, svakoj koloni i svakoj dijagonali jednak 15 i svi brojevi različiti. Na primer:

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

[Rešenje 7.4]

Zadatak 7.5 Napisati program koji pronalazi sve vrednosti promenljivih X, Y, Z za koje važi da je $X \geq Z$ i $X * 2 + Y * X + Z \leq 34$ pri čemu promenljive pripadaju narednim domenima $X \in \{1, 2, \dots, 90\}$, $Y \in \{2, 4, 6, \dots, 60\}$ i $Z \in \{1, 10, 20, \dots, 100\}$

[Rešenje 7.5]

Zadatak 7.6 Napisati program koji dodeljuje različite vrednosti različitim karakterima tako da suma bude zadovoljena:

```
TWO
+TWO
-----
FOUR
```

[Rešenje 7.6]

Zadatak 7.7 Napisati program koji pronalazi sve vrednosti promenljivih X, Y, Z i W za koje važi da je $X \geq 2 * W$, $3 + Y \leq Z$ i $X - 11 * W + Y + 11 * Z \leq 100$ pri čemu promenljive pripadaju narednim domenima $X \in \{1, 2, \dots, 10\}$, $Y \in \{1, 3, 5, \dots, 51\}$, $Z \in \{10, 20, 30, \dots, 100\}$ i $W \in \{1, 8, 15, 22, \dots, 1000\}$.

[Rešenje 7.7]

Zadatak 7.8 Napisati program koji raspoređuje brojeve 1-9 u dve linije koje se seku u jednom broju. Svaka linija sadrži 5 brojeva takvih da je njihova suma u obe linije 25 i brojevi su u rastućem redosledu.

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

[Rešenje 7.8]

Zadatak 7.9 Pekara *Kiflica* proizvodi hleb i kifle. Za mešenje hleba potrebno je 10 minuta, dok je za kiflu potrebno 12 minuta. Vreme potrebno za pečenje ćemo zanemariti. Testo za hleb sadrži 300g brašna, a testo za kiflu sadrži 120g brašna. Zarada koja se ostvari prilikom prodaje jednog hleba je 7 dinara, a prilikom prodaje jedne kifle je 9 dinara. Ukoliko pekara ima 20 radnih sati za mešenje peciva i 20kg brašna, koliko komada hleba i kifli treba da se umesi kako bi se ostvarila maksimalna zarada (pod pretpostavkom da će pekara sve prodati)?

[Rešenje 7.9]

Zadatak 7.10 Napisati program pronalazi vrednosti A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S (svako slovo predstavlja različit broj) koje su poređane u heksagon na sledeći način:

A, B, C
D, E, F, G
H, I, J, K, L
M, N, O, P
Q, R, S

tako da zbir vrednosti duž svake horizontalne i dijagonalne linije bude 38 ($A+B+C = D+E+F+G = \dots = Q+R+S = 38$, $A+D+H = B+E+I+M = \dots = L+P+S = 38$, $C+G+L = B+F+K+P = \dots = H+M+Q = 38$).

[Rešenje 7.10]

Zadatak 7.11 Kompanija Start ima 250 zaposlenih radnika. Rukovodstvo kompanije je odlučilo da svojim radnicima obezbedi dodatnu edukaciju. Da bi se radnik obučio programskom jeziku Elixir potrebno je platiti 100 evra po osobi za kurs, ali bi njegovo produktivno znanje ovog programskog jezika donelo 150 projekat/sati mesečno, što bi za kompaniju značilo dobit od 5 evra po projekat/satu. Da bi se radnik obučio programskom jeziku Dart potrebno je platiti 105 evra po osobi za kurs, ali bi njegovo produktivno znanje ovog programskog jezika donelo 170 projekat/sati mesečno, koji bi za kompaniju značili dobit od 6 evra po satu. Ukoliko Start ima na raspolaganju 26000 evra za obuku i maksimalan broj 51200 mogućih projekat/sati mesečno, odrediti na koji način kompanija treba da obuči svoje zaposlene kako bi ostvarila maksimalnu dobit.

[Rešenje 7.11]

7.1.3 Zadaci za vežbu

Zadatak 7.12 Za svaku narednu zagonetku, napisati program koji dodeljuje različite vrednosti različitim karakterima tako da suma bude zadovoljena:

GREEN + ORANGE = COLORS
MANET + MATISSE + MIRO + MONET + RENOIR = ARTISTS
COMPLEX + LAPLACE = CALCULUS
THIS + IS + VERY = EASY
CROSS + ROADS = DANGER
FATHER + MOTHER = PARENT
WE + WANT + NO + NEW + ATOMIC = WEAPON
EARTH + AIR + FIRE + WATER = NATURE
SATURN + URANUS + NEPTUNE + PLUTO = PLANETS
SEE + YOU = SOON
NO + GUN + NO = HUNT
WHEN + IN + ROME + BE + A = ROMAN
DONT + STOP + THE = DANCE
HERE + THEY + GO = AGAIN
OSAKA + HAIKU + SUSHI = JAPAN
MACHU + PICCHU = INDIAN
SHE + KNOWS + HOW + IT = WORKS
COPY + PASTE + SAVE = TOOLS

Zadatak 7.13 Za svaku narednu zagonetku, napisati program koji dodeljuje različite vrednosti različitim karakterima tako da suma bude zadovoljena:

THREE + THREE + ONE = SEVEN
NINE + LESS + TWO = SEVEN
ONE + THREE + FOUR = EIGHT
THREE + THREE + TWO + TWO + ONE = ELEVEN

SIX + SIX + SIX = NINE + NINE
 SEVEN + SEVEN + SIX = TWENTY
 ONE + ONE + ONE + THREE + THREE + ELEVEN = TWENTY
 EIGHT + EIGHT + TWO + ONE + ONE = TWENTY
 ELEVEN + NINE + FIVE + FIVE = THIRTY
 NINE + SEVEN + SEVEN + SEVEN = THIRTY
 TEN + SEVEN + SEVEN + SEVEN + FOUR + FOUR + ONE = FORTY
 TEN + TEN + NINE + EIGHT + THREE = FORTY
 FOURTEEN + TEN + TEN + SEVEN = FORTYONE
 NINETEEN + THIRTEEN + THREE + TWO + TWO + ONE + ONE + ONE = FORTYTWO
 FORTY + TEN + TEN = SIXTY
 SIXTEEN + TWENTY + TWENTY + TEN + TWO + TWO = SEVENTY
 SIXTEEN + TWELVE + TWELVE + TWELVE + NINE + NINE = SEVENTY
 TWENTY + TWENTY + THIRTY = SEVENTY
 FIFTY + EIGHT + EIGHT + TEN + TWO + TWO = EIGHTY
 FIVE + FIVE + TEN + TEN + TEN + TEN + THIRTY = EIGHTY
 SIXTY + EIGHT + THREE + NINE + TEN = NINETY
 ONE + NINE + TWENTY + THIRTY + THIRTY = NINETY

Zadatak 7.14 Za svaku narednu zagonetku, napisati program koji dodeljuje različite vrednosti različitim karakterima tako da jednakost bude zadovoljena:

MEN * AND = WOMEN
 COGITO = ERGO * SUM
 ((JE + PENSE) - DONC) + JE = SUIS
 FERMAT * S = LAST + THEOREM.
 WINNIE / THE = POOH
 TWO * TWO + EIGHT = TWELVE

Zadatak 7.15 Uraditi sve zadatke koji su pobrojani ovde:
<http://www.primepuzzle.com/leeslatest/alphameticpuzzles.html>

Zadatak 7.16 Čistačica Mica sređuje i čisti kuće i stanove. Da bi sredila i počistila jedan stan potrebno joj je 1 sat, dok joj je za kuću potrebno 1.5 sati. Prilikom čišćenja, Mica potroši neku količinu deterdženta, 120ml po stanu, odnosno 100ml po kući. Mica zaradi 1000 dinara po svakom stanu, odnosno 1500 dinara po kući. Ukoliko Mica radi 40 sati nedeljno i ima 5l deterdženta na raspolaganju, koliko stanova i kuća je potrebno da očisti kako bi imala najveću zaradu?

Zadatak 7.17 Marija se bavi grnčarstvom i pravi šolje i tanjire. Da bi se napravila šolja, potrebno je 6 minuta, dok je za tanjir potrebno 3 minuta. Pri pravljenju šolje potroši se 75 gr, dok se za tanjir potroši 100 gr gline. Ukoliko ima 20 sati na raspolaganju za izradu svih proizvoda i 250 kg gline, a zarada koju ostvari iznosi 2 evra po svakoj šolji i 1.5 evra po tanjiru, koliko šolja i tanjira treba da napravi kako bi ostvarila maksimalnu zaradu?

Zadatak 7.18 Jovanin komšija preprodaje računare i računarsku opremu. Očekuje isporuku računara i štampača. Pri tom, računari su spakovani tako da njihova kutija zauzima 360 kubnih decimetara prostora, dok se štampači pakuju u kutijama koje zauzimaju 240 kubnih decimetara prostora. Komšija se trudi da mesečno proda najmanje 30 računara i da taj broj bude bar za 50% veći od broja prodatih štampača. Računari koštaju 200 evra po nabavnoj ceni, a prodaju se po ceni od 400 evra, dok štampači koštaju u nabavci 60 evra i prodaju se za 140 evra. Magacin kojim komšija raspolaže ima svega 30000 kubnih decimetara prostora i mesečno može da nabavi robu u iznosu od najviše 14000 evra. Koliko računara, a koliko štampača komšija treba da proda kako bi se maksimalno obogatio?

7.2 Rešenja

Rešenje 7.2

```

% funkciji labeling prosledjujemo zahtev za minimizaciju trazenog izraza u listi
2 petocifren :- Vars = [A,B,C,D,E],
    % definisemo domen
4     A :: 1..9,
    B :: 0..9,
6     C :: 0..9,
    D :: 0..9,
8     E :: 0..9,
    % dodajemo uslov
10    alldifferent(Vars),
    % prilikom obelezavanja prosledjujemo i uslov minimizacije
12    labeling([minimize(A+2*B-3*C+4*D-5*E)],Vars),
    % prevodimo dobijene vednosti u broj
14    Broj is 10000*A+1000*B+100*C+10*D+E,
    % ispisujemo resenje
16    write(Broj), nl.

```

Rešenje 7.3

```

% promenljive oznacavaju broj novcica u kombinaciji redom za A - 1 din, B - 2 din, C
- 5 din, D - 10 din, E - 20 din.
2 kombinacije(Vars) :- Vars = [A,B,C,D,E],
    % definisemo domen
4     A :: 1..50,
    B :: 1..25,
6     C :: 1..10,
    D :: 1..5,
8     E :: 1..2,
    % dodajemo uslov
10    A+2*B+5*C+10*D+20*E #= 50,
    labeling(Vars),
12    % ispisujemo resenje
    write(A+2*B+5*C+10*D+20*E = 50), nl.

```

Rešenje 7.4

```

magicni(Vars):- Vars=[X1,X2,X3,X4,X5,X6,X7,X8,X9],
2     % domen za sve je isti
    Vars :: 1..38,
4     % uslov razlicitosti
    alldifferent(Vars),
6     % uslovi za zbirove
    X1+X2+X3#=15,
8     X4+X5+X6#=15,
    X7+X8+X9#=15,
10    X1+X4+X7#=15,
    X2+X5+X8#=15,
12    X3+X6+X9#=15,
    X1+X5+X9#=15,
14    X3+X5+X7#=15,
    labeling(Vars),
16    write(X1), write(' '), write(X2), write(' '), write(X3), nl,
    write(X4), write(' '), write(X5), write(' '), write(X6), nl,
18    write(X7), write(' '), write(X8), write(' '), write(X9), nl.
/*
20 | ?- magicni(Vars).
21 | 2 7 6
22 | 9 5 1
23 | 4 3 8
24 | Vars = [2,7,6,9,5,1,4,3,8] ?;
25 | 2 9 4
26 | 7 5 3
27 | 6 1 8
28 | Vars = [2,9,4,7,5,3,6,1,8] ?;
29 | 4 3 8
30 | 9 5 1
31 | 2 7 6
32 | ...
*/

```

Rešenje 7.5

```

1  pronadji(Vars):- Vars=[X,Y,Z],
   % definisemo domen
3  X :: 1..90,
   Y :: 2..2..60,
5  Z :: 1..10..100,
   % definisemo ogranicenja
7  Z #=< X,
   2*X+Y*X+Z #=< 34,
9  % instanciramo promenljive
   labeling(Vars).
11
/*
13 | ?- pronadji(Vars)
   Vars = [1,2,1] ?;
15 Vars = [1,4,1] ?;
   Vars = [1,6,1] ?;
17 Vars = [1,8,1] ?;
   Vars = [1,10,1] ?;
19 ...
21 */

```

Rešenje 7.6

```

1  jednakost(Vars):- Vars=[T,W,O,F,U,R],
   Vars :: 0..9,
   % iskljuccujemo za T i F nulu iz domena
4  T#\=0,
   F#\=0,
6  alldifferent(Vars),
   2*(T*100+W*10+O) #=
8  F*1000+O*100+U*10+R,
   labeling(Vars),
10 write(' '), write(T), write(W), write(O), nl,
   write('+'), write(T), write(W), write(O), nl,
12 write('-----'), nl,
   write(F), write(O), write(U), write(R), nl.
14
/*
16 | ?- jednakost(Vars)
   734
18 +734
   -----
20 1468
   Vars = [7,3,4,1,6,8] ?;
22 765
   +765
   -----
24 1530
   ...
26 */

```

Rešenje 7.7

```

1  pronadji(Vars):- Vars=[X,Y,Z,W],
   % definisemo domen
3  X :: 1..10,
   Y :: 1..2..51,
5  Z :: 10..10..100,
   W :: 1..7..1000,
7  % definisemo ogranicenja
   2*W #=< X,
9  3+Y #=< Z,
   X-11*W+Y+11*Z #=< 100,
11 % instanciramo promenljive
   labeling(Vars).
13
/*
   Ovo je primer sistema nejednacina bez resenja:

```

```

15 | ?- pronadji(Vars)
17 | no
19 | */

```

Rešenje 7.8

```

1 | % A1, B1, C1, D1, E1 predstavljaju brojeve na glavnoj dijagonali
2 | % A2, B2, C1, D2, E2 predstavljaju brojeve na glavnoj dijagonali
3 | dijagonale(Vars):- Vars=[A1,B1,C1,D1,E1,A2,B2,D2,E2],
4 |     % domen za sve je isti
5 |     Vars :: 1..9,
6 |     % uslov razlicitosti
7 |     alldifferent(Vars),
8 |     % uslovi za zbirove
9 |     A1+B1+C1+D1+E1#=25,
10 |    A2+B2+C1+D2+E2#=25,
11 |    % uslovi za poredak
12 |    A1#<B1, B1#<C1, C1#<D1, D1#<E1,
13 |    A2#<B2, B2#<C1, C1#<D2, D2#<E2,
14 |    labeling(Vars),
15 |    write(A1), write(' '), write(A2), nl,
16 |    write(' '), write(B1), write(' '), write(B2), nl,
17 |    write(' '), write(C1), nl,
18 |    write(' '), write(D2), write(' '), write(D1), nl,
19 |    write(E2), write(' '), write(E1), nl.

```

Rešenje 7.9

```

1 | /*
2 | H - broj komada hleba, K - broj komada kifli
3 |
4 | H>=0
5 | K>=0
6 |
7 | S obzirom na to da imamo 20kg brasna na raspolaganju, mozemo napraviti:
8 | - najvise 20000/120 kifli
9 | - najvise 20000/300 hleba
10 |
11 | H <= 20000/120 ~ 167
12 | K <= 20000/300 ~ 67
13 |
14 | S obzirom na to da imamo 20h na raspolaganju, mozemo napraviti:
15 | - najvise 1200/12 kifli
16 | - najvise 1200/10 hleba
17 |
18 | H <= 1200/10 = 120
19 | K <= 1200/12 = 100
20 |
21 | najoptimalnije je za gornju granicu domena postaviti minimum od dobijenih vrednosti,
22 | tj. sve ukupno H <= 120, K <= 67
23 | */
24 |
25 | pekara(Vars) :- Vars = [H, K],
26 |     H :: 0..120,
27 |     K :: 0..67,
28 |
29 | /*
30 | Ogranicenja vremena:
31 | - vreme potrebno za mesenje jednog hleba je 10min,
32 |   tj. za mesenje H komada hleba potrebno je 10*H minuta
33 | - vreme potrebno za mesenje jedne kifle je 12min,
34 |   tj. za mesenje K komada kifli potrebno je 12*K minuta
35 |
36 | Ukupno vreme koje je na raspolaganju iznosi 20h, tako da je:
37 | 10*H + 12*K <= 1200
38 | */
39 |

```

```

10*H + 12*K #=< 1200,
41
/*
43 Ogranichenje materijala:
- za jedan hleb potrebno je 300g brasna, a za H komada hleba potrebno je H*300 grama
45 - za jednu kifli potrebno je 120g brasna, a za K komada kifli potrebno je K*120
   grama
47 Ukupno, na raspolaganju je 20kg brasna, tako da je:
300*H + 120*K <= 20000
49 */
300*H + 120*K #=< 20000,
51
53 /*
Zarada iznosi:
55 - 7din/hleb, tj. zarada za H komada hleba bice 7*H
- 9din/kifla tj. zarada za K komada kifli bice 9*K
57
Ukupna zarada iznosi:
59 7*H + 9*K - funkcija koju treba maksimizovati - ovo dodajemo prilikom obelezavanja
*/
61 labeling([maximize(7*H+9*K)], Vars),
Zarada is 7*H+9*K,
63 write('Maksimalna zarada od '), write(Zarada), write(' dinara se ostvaruje za '),
write(H), write(' komada hleba i '), write(K), write(' komada kifli. '), nl.

```

Rešenje 7.10

```

heksagon(Vars):- Vars=[A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S],
2 % domen za sve je isti
Vars :: 1..38,
4 % uslov razlicitosti
alldifferent(Vars),
6
/*
Dodajemo ogranichenja za svaku horizontalnu liniju
8 A,B,C
D,E,F,G
10 H,I,J,K,L
M,N,O,P
12 Q,R,S
*/
14 A+B+C#=38,
D+E+F+G#=38,
16 H+I+J+K+L#=38,
M+N+O+P#=38,
18 Q+R+S#=38,
% Dodajemo ogranichenja za svaku od glavnih dijagonala
20 H+M+Q#=38,
D+I+N+R#=38,
22 A+E+J+O+S#=38,
B+F+K+P#=38,
24 C+G+L#=38,
% Dodajemo ogranichenja za svaku od sporednih dijagonala
26 A+D+H#=38,
B+E+I+M#=38,
28 C+F+J+N+Q#=38,
G+K+O+R#=38,
30 L+P+S#=38,
32
labeling(Vars),
write(' '), write(A), write(' '), write(B), write(' '), write(C), nl,
34 write(' '), write(D),write(' '), write(E), write(' '),write(F),write(' '), write(
G), nl,
write(H), write(' '),write(I),write(' '), write(J),write(' '), write(K),write(' '
), write(L), nl,
36 write(' '), write(M),write(' '), write(N), write(' '),write(O),write(' '), write(
P), nl,
write(' '), write(Q),write(' '), write(R), write(' '),write(S), nl.

```


Rešenje 7.11

```
1  /*
2  kompanija ima 250 zaposlenih radnika
3  za sve njih organizuje dodatnu obuku
4  ako je E promenjiva za Elixir, a D za Dart
5  mora da vazi E<=250, D<=250 i E + D = 250
6  */
7
8  kompanija(Vars) :- Vars = [E, D],
9      E :: 0..250,
10     D :: 0..250,
11
12     % dodajemo uslov za ukupan broj radnika
13     E+D#=250,
14
15     % dodajemo ogranicenje za broj projekat sati
16     150*E+170*D#=<51200,
17
18     % dodajemo ogranicenje za raspoloziva sredstva
19     100*E+105*D#=<26000,
20
21 /*
22 ukupna zarada se dobija kada od ostvarene dobiti preko broja projekat/sati oduzmemo
23 gubitak za placanje kurseva radnicima, tj. funkcija koju treba maksimizovati je:
24
25 150*5*E + 170*6*D - (100*E + 105*D) --> ovo dodajemo kod obelezavanja
26 */
27 labeling([maximize(150*5*E + 170*6*D - (100*E + 105*D)),Vars),
28     Zarada is (150*5*E + 170*6*D - (100*E + 105*D)),
29     write('Maksimalna zarada je '), write(Zarada),
30     write(', broj radnika koje treba poslati na kurs Elixir je '), write(E),
31     write(', a broj radnika koje treba poslati na kurs Dart je '), write(D),nl.
```