Programiranje 1 (vježbe)

Vedran Šego

2. prosinca 2017.

Sadržaj

Poglavlje 1

Brojevni sustavi

Brojimo: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Kako dalje? Pošto nemamo dodatnih znamenaka, prelazimo na dvoznamenkaste brojeve: na prvo mjesto stavljamo znamenku 1, a zadnju znamenku postavljamo na najmanju moguću vrijednost (nula), te dobijamo broj 10. Ponovno brojimo kao i do sada, mijenjajući samo posljednju znamenku: 11, 12, 13, 14, 15, 16, 17, 18, 19. Kad nam ponovno ponestane znamenaka, prvu znamenku povećamo za jedan, a posljednju opet vraćamo na nulu i tako dobijemo 20. Postupak nastavljamo dok god je potrebno. Opisani način zapisivanja brojeva zovemo "prikaz brojeva u dekadskoj bazi" (ili "u bazi deset").

Kako bismo brojali kad bi nam netko zabranio upotrebu znamenaka 8 i 9?

Krenimo istom logikom: 0, 1, 2, 3, 4, 5, 6, 7. Sada više nemamo znamenaka, pa na prelazimo na dvoznamenkaste brojeve: 10. Kao i malo prije, nastavljamo: 11, 12, 13, 14, 15, 16, 17. Ponovno nam nedostaju znamenke, pa povećavamo prvu, dok drugu vraćamo na nulu: 20. Ovakav zapis zovemo "prikaz brojeva u oktalnoj bazi" (ili "u bazi osam").

Napomena 1.0.1. *Iako tako izgleda, ovdje nismo preskočili brojeve. Naime, kad napišemo "*17*", to je* ***prikaz*** *nekog broja na određeni način, jednako kao što su* `XVII`*,* `MMVII` *i sl. prikazi nekih brojeva (*17 *i* 2007*). Oznaka "*12*" u bazi* 10 *predstavlja broj dvanaest; no, ta ista oznaka ("*12*") u bazi osam predstavlja broj deset. Naime, ako brojimo po redu, iza "*7*" dolazi "*10*" (a znamo da iza sedam dolazi osam); zatim imamo "*11*" (tj. devet), pa "*12*" (tj. deset).*

Možemo koristiti i 16 znamenaka. Tada, uz uobičajenih deset (od 0 do 9), upotrebljavamo i 6 dodatnih, koje označavamo slovima A, B, C, D, E i F. U tom slučaju, oznaka "A" predstavlja broj deset, oznaka "B" broj jedanaest itd. Oznaka "F" će predstavljati broj petnaest, pa će oznaka "10" zapravo označavati broj šesnaest. Takav prikaz zovemo "prikaz brojeva u heksadecimalnoj bazi" (ili "u bazi šesnaest").

Da bismo znali u kojoj je bazi broj zapisan, u sam zapis dodajemo oznaku baze:

$$(broj)_{baza},$$

npr. $(10)_{10} = (12)_8 = (A)_{16}$.

Baza	Jedan	Dva	Tri	Četiri	Sedamnaest	Dvije tisuće i sedam
2	$(1)_2$	$(10)_2$	$(11)_2$	$(100)_2$	$(10001)_2$	$(11111010111)_2$
3	$(1)_3$	$(2)_3$	$(10)_3$	$(11)_3$	$(122)_3$	$(2202100)_3$
8	$(1)_8$	$(2)_8$	$(3)_8$	$(4)_8$	$(21)_8$	$(3727)_8$
10	$(1)_{10}$	$(2)_{10}$	$(3)_{10}$	$(4)_{10}$	$(17)_{10}$	$(2007)_{10}$
16	$(1)_{16}$	$(2)_{16}$	$(3)_{16}$	$(4)_{16}$	$(11)_{16}$	$(7D7)_{16}$

Tablica 1.1: Primjeri prikaza nekih brojeva u raznim brojevnim sustavima

Računalo koristi sustav sa samo dvije znamenke (tzv. binarni zapis), što znači da ono broji 0, 1, 10, 11, 100,...

U tablici 1 prikazani su primjeri prikaza nekih brojeva u binarnom, ternarnom, oktalnom, dekadskom i heksadecimalnom sustavu.

1.1 Pretvaranje zapisa brojeva među bazama

Zapis broja a u bazi b označavamo s

$$a = (a_n a_{n-1} \dots a_1 a_0)_b,$$

što možemo raspisati:

$$a = a_n b^n + a_{n-1} b^{n-1} + \dots + a_1 b^1 + a_0 b^0 = \sum_{i=0}^{n} a_i b^i.$$

Na taj način dobijamo o kojem je broju riječ, tj. zapis u dekadskoj bazi.

Primijetite da uvrštavanjem baze $b = 10$ dobijamo upravo "uobičajenu" interpretaciju zapisa po znamenkama.

Zadatak 1.1.1. *Pretvorite u bazu* 10 *sljedeće brojeve:*
a) $(13)_{16}$ *b)* $(123)_4$ *c)* $(101001000)_2$ *d)* $(101001000)_4$

Rješenje. Raspisujemo po definiciji:

a) $(13)_{16} = 1 \cdot 16^1 + 3 \cdot 16^0 = 16 + 3 = 19$,

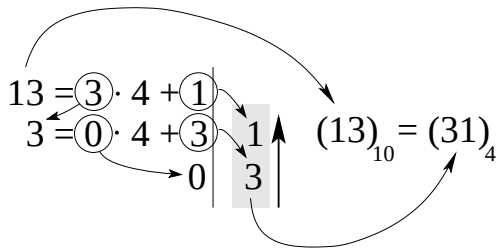
b) $(123)_4 = 1 \cdot 4^2 + 2 \cdot 4^1 + 3 \cdot 4^0 = 16 + 8 + 3 = 27$,

c) $(101001000)_2 = 1 \cdot 2^8 + 1 \cdot 2^6 + 1 \cdot 2^3 = 256 + 64 + 8 = 328$.

d) $(101001000)_4 = 1 \cdot 4^8 + 1 \cdot 4^6 + 1 \cdot 4^3 = 65536 + 4096 + 64 = 69696$.

□

Prikaz broja a u bazi b dobijamo uzastopnim dijeljenjem a s b, te zapisivanjem unatrag svih ostataka pri dijeljenju. Postupak je prikazan na slici 1.1, pri čemu je konačni rezultat ("$(31)_4$") označen sivom bojom.



Slika 1.1: Pretvaranje $(13)_{10}$ u bazu 4 (rezultat je $(31)_4$)

Zadatak 1.1.2. *Napišite prikaz u bazi b broja a (zapisanog u dekadskoj bazi):*
a) $a = 31$, $b = 4$ *b)* $a = 123$, $b = 16$ *c)* $a = 1719$, $b = 13$

Rješenje.

a)
$$\begin{array}{rl|l} 31 = 7 \cdot 4 + 3 & & \\ 7 = 1 \cdot 4 + 3 & & 3 \\ 1 = 0 \cdot 4 + 1 & & 3 \\ 0 & & 1 \end{array}$$
Rješenje: $(31)_{10} = (133)_4$.

b)
$$\begin{array}{rl|l} 123 = 7 \cdot 16 + 11 & & \\ 7 = 0 \cdot 16 + 7 & & \mathrm{B}\ (= 11) \\ 0 & & 7 \end{array}$$
Rješenje: $(123)_{10} = (7\mathrm{B})_{16}$.

c)
$$\begin{array}{rl|l} 1719 = 132 \cdot 13 + 3 & & \\ 132 = 10 \cdot 13 + 2 & & 3 \\ 10 = 0 \cdot 13 + 10 & & 2 \\ 0 & & \mathrm{A}\ (= 10) \end{array}$$
Rješenje: $(123)_{10} = (\mathrm{A}23)_{13}$.

□

Pretvaranje zapisa broja iz baze $b_1 \neq 10$ u bazu $b_2 \neq 10$ obično radimo u dva koraka:

1. Pretvorimo zapis broja iz baze b_1 u bazu 10,
2. Pretvorimo zapis broja iz baze 10 u bazu b_2.

Zadatak 1.1.3. *Za brojeve sa zapisom a u bazi b_1 napišite prikaz u bazi b_2, pri čemu je:*
a) $a = 1719$, $b_1 = 11$, $b_2 = 9$ *b)* $a = 1311$, $b_1 = 5$, $b_2 = 8$

Rješenje. Pretvaranja među bazama rade se kao u rješenjima prethodna dva zadatka. Ovdje navodimo samo rješenja s međukorakom (prikazom broja u bazi 10):

a) $(1719)_{11} = (2198)_{10} = (3012)_9$

b) $(1311)_5 = (206)_{10} = (316)_8$

□

Broj a iz baze b_1 u bazu b_2 možemo izvesti direktno, ako postoje $b, n_1, n_2 \in \mathbb{N}$ takvi da je $b_1 = b^{n_1}$ i $b_2 = b^{n_2}$. Neka je zapis broja a u bazi b_1 dan s

$$a = (a_n a_{n-1} \dots a_1 a_0)_{b_1}.$$

Po koracima:

1. Svaku pojedinu znamenku pretvaramo iz baze b_1 u bazu b:

$$a_n a_{n-1} \dots a_1 a_0 \longrightarrow |(a_n)_b|(a_{n-1})_b| \dots |(a_1)_b|(a_0)_b|,$$

   pri čemu svaku znamenku u početnom zapisu zamjenjujemo s n_1 znamenaka. Ako zapis znamenke a_i u bazi b ima manje od n_1 znamenaka, na početak dopisujemo nule.

2. Dobiveni zapis dijelimo u grupice od po n_2 znamenaka s desna na lijevo (ako nam za najlijeviju grupu ponestane znamenaka, na početak dopišemo nule). Tako dobijemo:

$$|(a_n)_b|(a_{n-1})_b| \dots |(a_1)_b|(a_0)_b| \longrightarrow |a'_k|a'_{k-1}| \dots |a'_1|a'_0|,$$

   pri čemu smo samo regrupirali postojeće znamenke (nema nikakve konverzije).

3. Svaku grupu znamenaka a'_i zasebno pretvorimo u bazu b_2 (od svake mora nastati točno jedna znamenka u bazi b_2; ako ne ispadne, postoji greška u postupku!).

Primjer 1.1.1. *Želimo broj* $(3701)_8$ *pretvoriti u bazu* 16. *Očito je* $b = 2$, $n_1 = 3$ *(jer je* $8 = 2^3$*) i* $n_2 = 4$ *(jer je* $16 = 2^4$*).*

1. $3701 \longrightarrow |011|111|000|001|$

2. $|011|111|000|001| = 011111000001 \longrightarrow |0111|1100|0001|$

3. $|0111|1100|0001| \longrightarrow 7C1$,
jer je $(111)_2 = (7)_{16}$, $(1100)_2 = (12)_{10} = (C)_{16}$ *i* $(1)_2 = (1)_{16}$.

Rješenje: $(3701)_8 = (7\text{C}1)_{16}$.

Naravno, ako je $b_2 = b_1^n$, tada preskačemo prvi korak (jer je onda $b_2 = b$, $n_1 = 1$ i $n_2 = n$). Ako je pak $b_1 = b_2^n$, preskačemo drugi i treći korak (jer je onda $b_1 = b$, $n_1 = n$ i $n_2 = 1$).

Zadatak 1.1.4. *Bez pretvaranja u dekadski sustav, popunite praznine:*

a) $(17)_8 = (\quad)_{16}$

b) $(313)_4 = (\quad)_{16}$

c) $(1412)_9 = (\quad)_{27}$

d) $(256)_{25} = (\quad)_5$

Rješenje.

a) $(17)_8 \overset{b=2,n_1=3}{\longrightarrow} |001|111| \overset{n_2=4}{\longrightarrow} |0000|1111| = (F)_{16}$

b) $(313)_4 \overset{b=4,n_1=1}{\longrightarrow}$ preskačemo 1. korak $\overset{n_2=2}{\longrightarrow} |03|13| = (37)_{16}$ ili
$(313)_4 \overset{b=2,n_1=2}{\longrightarrow} |11|01|11| \overset{n_2=4}{\longrightarrow} |0011|0111| = (37)_{16}$

c) $(1412)_9 \overset{b=3,n_1=2}{\longrightarrow} |01|11|01|02| \overset{n_2=3}{\longrightarrow} |001|110|102| = (1\text{CB})_{27}$,
jer je $(110)_3 = 1 \cdot 3^2 + 1 \cdot 3^1 = 9 + 3 = 12 = (\text{C})_{27}$.

d) $(256)_{25} \overset{b=5,n_1=2}{\longrightarrow} |02|10|11| = (21011)_5$ (preskačemo korake 2 i 3 jer je $n_2 = 1$).

□

Zadatak 1.1.5. *Popunite praznine:*

a) $(17)_{16} = (\quad)_{17} = (\quad)_3$

b) $(314)_8 = (\quad)_2$

c) $(1412)_5 = (\quad)_2 = (\quad)_{11}$

d) $(121)_3 = (\quad)_9$

e) $(\text{DEDA})_{16} = (\quad)_7 = (\quad)_3$

f) $(321)_9 = (\quad)_{27}$

Zadatke b), d) i f) riješite na oba načina (i direktno i pomoću pretvaranja u dekadsku bazu).

Rješenje.

a) $(17)_{16} = (16)_{17} = (212)_3$

b) $(314)_8 = (11001100)_2$

c) $(1412)_5 = (11101000)_2 = (1A1)_{11}$

d) $(121)_3 = (17)_9$

e) $(\text{DEDA})_{16} = (325220)_7 = (2220020222)_3$

f) $(321)_9 = (9J)_{27}$

□

Ovakve zadatke možete si i sami zadavati (odredite broj i baze između kojih ga želite prebaciti; pri tome pazite da sve znamenke broja moraju biti strogo manje od baze u kojoj je broj zapisan!), a rješenje možete provjeriti na službenoj stranici kolegija:
http://degiorgi.math.hr/prog1/apps/brsust.php

1.2 Računske operacije

Brojeve, naravno, možemo zbrajati, oduzimati, množiti i dijeliti bez obzira na to u kojoj su bazi zapisani. Jedan način je pretvaranjem u dekadsku bazu, izvođenjem operacija, te pretvaranjem u polaznu bazu, no moguće je i direktno. Sve operacije se rade analogno onima u dekadskom sustavu, uz tu razliku da "1 dalje" i sl. ne radimo za 10 nego za b.

Napomena 1.2.1. *Računske operacije možemo izvršavati samo na brojevima koji su prikazani u istom brojevnom sustavu. Ako su brojevi prikazani s različitim bazama, potrebno je jednog od njih (ili oba) pretvoriti u neku bazu, preporučljivo u onu u kojoj treba biti zapisan rezultat.*

Primjer 1.2.1. *Tražimo zbroj, razliku, produkt i kvocijent brojeva* $x = (11110)_2$ *i* $y = (101)_2$. *Pretvaranjem lako dobijemo da su to* $x = (30)_{10}$ *i* $y = (5)_{10}$, *te dobijamo:*

$$x + y = (35)_{10} = (100011)_2, \qquad x - y = (25)_{10} = (11001)_2,$$
$$x \cdot y = (150)_{10} = (10010110)_2, \qquad x : y = (6)_{10} = (110)_2.$$

Izračunajmo vrijednost tih izraza bez pretvaranja u dekadsku bazu.

1. ***zbroj***:
   Primijetite:

$$(0)_2 + (0)_2 = (0)_2 \ i$$
$$(1)_2 + (0)_2 = (1)_2 \ i$$
$$(0)_2 + (1)_2 = (1)_2, \ ali$$
$$(1)_2 + (1)_2 = (1)_{10} + (1)_{10} = (2)_{10} = (10)_2,$$

   tj. rezultat zbrajanja $1 + 1$ *u bazi* 2 *je "nula i jedan dalje".*

$$\begin{array}{rr} & {\scriptstyle 1\,1\,1} \\ & 11110 \\ + & 101 \\ \hline & 100011 \end{array}$$

2. ***razlika***:
   Kad oduzimamo veću znamenku od manje, npr. $(0)_2 - (1)_2$*, potrebno je "posuditi"* 1 *od lijeve znamenke. Time dobijamo oduzimanje jednoznamenkastog broja od dvoznamenkastog, npr.* $(10)_2 - (1)_2$*.*
   Oduzimamo kao i u dekadskoj bazi, samo što "posuđenih" 1 *ne daje* 10 *nego* 2 *(pa* $0 - 1$ *daje rezultat* $2 - 1 = 1$*, uz "pamtimo* 1*" koji zatim oduzmemo od iduće znamenke):*

$$\begin{array}{rr} & 11110 \\ - & 101 \\ & {\scriptstyle -1} \\ \hline & 11001 \end{array}$$

3. ***produkt***:
   Množenje se obavlja slično zbrajanju, samo treba paziti jer nemamo nužno samo 1 *dalje, nego može biti i više. U binarnoj bazi, taj dio se ne primjećuje; dapače, nema niti klasičnog množenja (jer množimo samo s nulom ili jedinicom). No, potrebno je paziti na potpisivanje i ispravno zbrojiti sve rezultate:*

$$\begin{array}{rr} 11110 & \cdot\ 101 \\ \hline {\scriptstyle 1\,1\,1\,1} & \\ 11110 & \\ 0 & \\ 11110 & \\ \hline 10010110 & \end{array}$$

   Primijetite: $(1)_2 + (1)_2 + (1)_2 = (10)_2 + (1)_2 = (11)_2$*. Zato, ako već imamo "*1 *dalje", izraz* $(1)_2 + (1)_2$ *daje "rezultat* 1 *i* 1 *dalje".*

4. ***kvocijent***:
   Ponovno, princip je jednak onome iz dekadskog sustava. Dapače, jedini koraci

koji čine razliku su množenje i oduzimanje. Dakle, potrebno je "pogoditi" kvocijent početka djeljenika s djeliteljem. Zatim se djelitelj množi s "pogođenim" rezultatom, te se rezultat oduzima od početka djeljenika. Ako je dobiveni rezultat nenegativan i strogo manji od djelitelja, znači da je kvocijent dobro pogođen. "Spuštamo" iduću znamenku djeljenika i nastavljamo postupak:

$$\begin{array}{r} 11110 \;:\; 101 = 110 \\ \hline 111 \\ -101 \\ \hline 101 \\ -101 \\ \hline 00 \\ -0 \\ \hline 0 \end{array}$$

Napomena 1.2.2. *Za svaku bazu b vrijedi:*

$$(a_n a_{n-1} \dots a_1 a_0)_b \cdot b = (a_n a_{n-1} \dots a_1 a_0)_b \cdot (10)_b = (a_n a_{n-1} \dots a_1 a_0 0)_b.$$

Tako, na primjer, množenje s 2 u binarnoj bazi radimo dopisivanjem nule na kraj.

Zadatak 1.2.1. *Izračunajte zbroj, razliku, produkt i kvocijent brojeva $x = (7347)_8$ i $y = (155)_8$ bez pretvaranja u dekadsku bazu.*

Rješenje.

1. **zbroj**:
   Vrijedi: $(7)_8 + (5)_8 = (7)_{10} + (5)_{10} = (12)_{10} = 1 \cdot 8 + 4$, pa je rezultat takvog zbrajanja 4 i "1 dalje". Nakon toga imamo $(1)_8 + (4)_8 + (5)_8$ (tih "1 dalje" zbrojeno s predzadnje dvije znamenke), što daje $(10)_{10} = 1 \cdot 8 + 2$, tj. rezultat 2 i "1 dalje". Analogno dalje.

$$\begin{array}{rr} & \overset{1\,1}{} \\ & 7347 \\ + & 155 \\ \hline & 7524 \end{array}$$

2. **razlika**:
   Kod oduzimanja predzadnjih znamenaka imamo: $(4)_8 - (5)_8$, što daje negativan rezultat. Zbog toga jepotrebno "posuditi" 1 od treće znamenke s desna. Tada je potrebno izračunati $(14)_8 - (5)_8 = (12)_{10} - (5)_{10} = (7)_{10} = (7)_8$. Pri tome "posuđenu" jedinicu treba oduzeti od znamenke od koje je "posuđena".

$$\begin{array}{rr} & 7347 \\ - & 155 \\ & {\scriptstyle -1} \\ \hline & 7172 \end{array}$$

3. **produkt**:
   Množenje s jednom znamenkom, na primjer $(7347)_8$ s 5, možemo obaviti po koracima:

   1. $7 \cdot 5 = (35)_{10} = (43)_8 \to 3$
   2. $4 \cdot 5 = (20)_{10} = (24)_8$; ovome još treba pribrojiti "4 dalje" (prva znamenka iz prethodnog koraka), što za rezultat daje $(24)_8 + (4)_8 = (30)_8 \to 0$
   3. $3 \cdot 5 = (15)_{10} = (17)_8$; dodajemo još "3 dalje" iz prethodnog koraka, pa je konačni rezultat $(17)_8 + (3)_8 = (22)_8 \to 2$
   4. $7 \cdot 5 = (35)_{10} = (43)_8$; dodajemo još "2 dalje" iz prethodnog koraka, pa je konačni rezultat $(43)_8 + (2)_8 = (45)_8$

   Konačni rezultat dobijemo slaganjem rezultata po koracima, od zadnjeg prema prvom:

   $$(7347)_8 \cdot (5)_8 = (45|2|0|3)_8 = (45203)_8$$

$$\begin{array}{r} 7347 \quad \cdot 155 \\ \hline {\scriptstyle 1\,1\,1\,1} \\ 7347 \\ 45203 \\ \quad 45203 \\ \hline 1454133 \end{array}$$

4. **kvocijent**:

$$\begin{array}{r} 7347 \;:\; 155 = 43 \\ \hline 734 \\ -664 \\ \hline 507 \\ -507 \\ \hline 0 \end{array}$$

Provjere radi, lako vidimo da je $x = (7347)_8 = (3815)_{10}$ i $y = (155)_8 = (109)_{10}$, pa imamo:

$$x + y = (3924)_{10} = (7524)_8, \qquad x - y = (3706)_{10} = (7172)_8,$$
$$x \cdot y = (415835)_{10} = (1454133)_8, \qquad x : y = (35)_{10} = (43)_8,$$

što pokazuje da su dobiveni rezultati točni. □

Napomena 1.2.3 (Česte greške)**.** *Prilikom računanja u raznim nedekadskim bazama, pripazite na česte greške primjenjivanja uobičajenih pravila iz dekadskog sustava! Na primjer:*

- *Izraze* $(x)_b \pm (y)_b$, $(x)_b \cdot (y)_b$ *i* $(x)_b : (y)_b$ ***ne smijete*** *prvo izračunati kao da su* x *i* y *dekadski brojevi, pa zatim rezultat pretvoriti u bazu* b*, nego* ***morate*** *ili računati direktno u bazi* b *(kako je opisano) ili pretvoriti u dekadsku bazu! Na primjer,* $(10)_2 + (11)_2$ ***ne možemo*** *proglasiti za* 21 *i onda pretvoriti u bazu* 2*, jer će rezultat biti* $(10101)_2$ *(točan rezultat je* $(101)_2$*).*

- *Izraze ne smijete niti samo evaluirati kao da su dekadski brojevi! Na primjer,* $(28)_{16} + (37)_{16}$ ***nije*** $(65)_{16}$ *(*$= (101)_{10}$*), nego* $(5\mathrm{F})_{16}$ *(*$= (95)_{10}$*).*

- *Ako je zapis broja u bazi* b *dan s*

$$(a_n a_{n-1} \ldots a_1 a_0)_b,$$

onda za svaki $i \in \{0, 1, \ldots, n\}$ *mora vrijediti:*

$$a_i \in \{0, 1, \ldots, b-1\}.$$

Drugim riječima, sve znamenke broja prikazanog u bazi b *moraju biti nenegativne i* ***strogo manje*** *od* b*! Na primjer, ako dobijete rezultat poput* $(678)_8$ *ili* $(73)_5$*, on je* ***besmislen*** *i samim tim pogrešan.*

- *Pažljivo pročitajte zadatak i pazite koja je baza zadana. Na primjer, broj* $(10110101)_8$ *može izgledati kao binarni broj, no on je, u stvari oktalni!*

Zadatak 1.2.2. *Izračunajte bez pretvaranja u druge baze:*

a) $(123)_4 + (231)_4$

b) $(1234)_5 + (2341)_5 + (3412)_5 + (4123)_5$

c) $(\mathrm{BABA})_{16} + (\mathrm{DEDA})_{16}$

d) $(624)_8 - (555)_8$

e) $(147\mathrm{AD})_{15} - (\mathrm{EEEE})_{15}$

f) $(\mathrm{DEDA})_{16} - (\mathrm{BABA})_{16}$

g) $(505)_6 \cdot (23)_6$

h) $(314)_8 \cdot (72)_8$

i) $(\mathrm{BABA})_{16} \cdot (9\mathrm{A})_{16}$

j) $(111011010)_3 : (110)_3$

k) $(7170)_9 : (120)_9$

l) $(\mathrm{F1FA})_{16} : (293)_{16}$

Rješenje.

a) $(123)_4 + (231)_4 = (1020)_4$

b) $(1234)_5 + (2341)_5 + (3412)_5 + (4123)_5 = (22220)_5$

c) $(\mathrm{BABA})_{16} + (\mathrm{DEDA})_{16} = (19994)_{16}$

d) $(624)_8 - (555)_8 = (47)_8$

e) $(147\mathrm{AD})_{15} - (\mathrm{EEEE})_{15} = (47AE)_{15}$

f) $(\mathrm{DEDA})_{16} - (\mathrm{BABA})_{16} = (2420)_{16}$

g) $(505)_6 \cdot (23)_6 = (20503)_6$

h) $(314)_8 \cdot (72)_8 = (27070)_8$

i) $(\mathrm{BABA})_{16} \cdot (9\mathrm{A})_{16} = (7053E4)_{16}$

j) $(111011010)_3 : (110)_3 = (1002121)_3$

k) $(7170)_9 : (120)_9 = (58)_9$

l) $(\mathrm{F1FA})_{16} : (293)_{16} = (5\mathrm{E})_{16}$

□

Kao i s pretvaranjem zapisa brojeva između različitih brojevnih sustava, zadatke s računanjem u određenoj bazi možete sami sastaviti, a rezultate možete provjeriti na službenoj stranici kolegija: http://degiorgi.math.hr/prog1/apps/brsust.php.

Rezultate možete provjeriti i "na ruke", pretvaranjem u dekadsku bazu.

Napomena 1.2.4. *Ukoliko Vam na kolokviju ostane dovoljno vremena, za provjeru riješite zadatke s računskim operacijama na zasebnom papiru i s pretvaranjem u dekadski sustav. Ukoliko se rezultati ne poklapaju, znači da postoji greška (ona može biti i u originalnom i u kontrolnom računanju)!*

1.3 Traženje baze

Ponekad se u zadacima traži baza b u kojoj vrijedi neko svojstvo. U pravilu, takve zadatke rješavamo raspisom po definiciji (tj. pretvaranjem u dekadsku bazu). Pri tome treba paziti da $x + a_i$ i $x \cdot a_i$, gdje su x i a_i jednoznamenkasti nenegativni brojevi, ne moraju više biti jednoznamenkasti, kao i da $x - a_i$ i $a_i - x$ ne moraju više biti nenegativni. Također, korisno je dobivena rješenja uvrstiti u početne uvjete i tako provjeriti jesu li dobra.

Napomena 1.3.1 (Česte greške)**.**

- *Sve dobivene baze moraju biti strogo veće od* 1. *Na primjer, brojevi* -17, 0, 1 *i sl. nisu dobre baze brojevnih sustava.*
- *Baza zapisa svakog broja mora biti strogo veća i od svih znamenaka tog broja.*
- *Baza, naravno, mora biti cijeli broj, kao i sve znamenke. Drugim riječima, ako dobijete da je neka baza, na primjer,* $\sqrt{5}$*, to rješenje morate odbaciti kao nezadovoljavajuće.*
- *Pazite što je baza, jer ona ne mora nužno biti varijabla* b*, nego može biti i neka druga varijabla ili čak neki izraz. Na primjer, ako imamo zadan broj* $(123)_{b+7}$ *za koji mora vrijediti neko svojstvo, tada mora biti* $b+7>3$*, tj.* $b>-4$*. U takvom zadatku* $b=-3$ *može (ali i ne mora) biti dobro rješenje, jer je baza* $b+7$*, a ne* b*.*

Zadatak 1.3.1. *U kojoj bazi* b *je broj* $(123)_b$ *trostruko veći od broja* $(21)_b$*, a u kojoj je trostruko veći od broja* $(25)_b$*?*

Rješenje. Raspisujemo po definiciji:

$$\begin{aligned}(123)_b &= 3\cdot(21)_b\\ 1\cdot b^2+2\cdot b+3 &= 3\cdot(2\cdot b+1)=6\cdot b+3\\ b^2-4b &= 0\\ b(b-4) &= 0\end{aligned}$$

Kandidati za rješenja su $b_1=0$ i $b_2=4$. No, iz zadatka vidimo da mora biti $b>3$, pa b_1 ne može biti rješenje. Za kraj, uvrštavamo rješenje $b=4$ u uvjet zadatka:

$$(123)_4 = 4^2+2\cdot 4+3 = 16+8+3 = 27 = 3\cdot 9 = 3\cdot(21)_4.$$

Zaključujemo da je $b=4$ jedino rješenje prvog dijela zadatka.

Riješimo sada drugi dio zadatka (na isti način):

$$\begin{aligned}(123)_b &= 3\cdot(25)_b\\ 1\cdot b^2+2\cdot b+3 &= 3\cdot(2\cdot b+5)=6\cdot b+15\\ b^2-4b-12 &= 0\\ (b-6)(b+2) &= 0\end{aligned}$$

Kandidati za rješenja su $b_1=6$ i $b_2=-2$. No, iz zadatka vidimo da mora biti $b>5$, pa b_2 ne može biti rješenje. Za kraj, uvrštavamo rješenje $b=6$ u uvjet zadatka:

$$(123)_6 = 6^2+2\cdot 6+3 = 36+12+3 = 51 = 3\cdot 17 = 3\cdot(25)_6.$$

Zaključujemo da je $b=6$ jedino rješenje drugog dijela zadatka. □

U oba slučaja u prethodnom zadatku mogli smo jednostavno utvrditi donju granicu za b (4 u prvom dijelu, odnosno 6 u drugom dijelu) i onda uvrštavanjem u uvjet zadatka provjeravati kandidate za rješenja: 4, 5, 6,... U oba slučaja bismo odmah pogodili rješenja, jer su ona jednaka donjim granicama.

Ipak, takav pristup nikako nije dobar. Kao prvo, za pogođeno rješenje treba dokazati da je jedino. Kao drugo, rješenje može biti puno veće od donje granice i pogađanje može potrošiti mnogo vremena.

Zadatak 1.3.2 (Pismeni ispit iz "Uvoda u računarstvo", 28. 11. 2005)**.** *Nađite sve b-ove takve da je* $(123)_{2b+3} = (501)_b$

Rješenje. Raspisujemo po definiciji:

$$\begin{aligned}(123)_{2b+3} &= (501)_b\\ 1\cdot(2b+3)^2+2\cdot(2b+3)+3 &= 5b^2+1\\ 4b^2+16b+18 &= 5b^2+1\\ b^2-16b-17 &= 0\\ (b-17)(b+1) &= 0\end{aligned}$$

Kandidati za rješenja su $b_1 = 17$ i $b_2 = -1$. Zbog uvjeta nenegativnosti, b_2 ne može biti rješenje. S druge strane, mora vrijediti $2b + 3 > 3$ i $b > 5$. Lako se vidi da $b = 17$ zadovoljava oba uvjeta. Za kraj, uvrštavamo rješenje $b = 17$ u uvjet zadatka:

$$\begin{aligned}(123)_{2\cdot17+3} = (123)_{37} &= 37^2+2\cdot37+3 = 1369+74+3\\ &= 1446 = 5\cdot17^2+1 = (501)_{17}.\end{aligned}$$

Zaključujemo da je $b = 17$ jedino rješenje zadatka. □

Zadatak 1.3.3 (Pismeni ispit iz "Uvoda u računarstvo", 14. 2. 2005)**.** *Nađite sve b-ove takve da je*

$$(321)_b + (321)_{b+1} + (321)_{b+2} = (840)_{b+3}.$$

Rješenje. Raspisujemo po definiciji:

$$\begin{aligned}(321)_b + (321)_{b+1} + (321)_{b+2} &= (840)_{b+3}\\ 9b^2+24b+24 &= 8b^2+52b+84\\ b^2-28b-60 &= 0\\ (b-30)(b+2) &= 0\end{aligned}$$

Kandidati za rješenja su $b_1 = 30$ i $b_2 = -2$. Zbog uvjeta nenegativnosti, b_2 ne može biti rješenje. S druge strane, mora vrijediti $b > 3$, $b+1 > 3$, $b+2 > 3$ i $b+3 > 8$. Lako se vidi da $b = 30$ zadovoljava sve te uvjete. Za kraj, uvrštavamo rješenje $b = 30$ u uvjet zadatka:

$$\begin{aligned}(321)_{30} + (321)_{30+1} + (321)_{30+2} &= (321)_{30} + (321)_{31} + (321)_{32}\\ &= 2761+2946+3137 = 8844\\ &= (840)_{33}.\end{aligned}$$

Zaključujemo da je $b = 30$ jedino rješenje zadatka. □

Zadatak 1.3.4 (Pismeni ispit iz "Uvoda u računarstvo", 27. 6. 2005)**.** *Nađite sve b-ove takve da je*

$$(101)_b + (101)_{b+2} + (101)_{b+4} = (271)_{b+6}.$$

Rješenje. Zadatak rješavamo analogno prethodnom. Jedino rješenje je $b = 23$. □

Zadatak 1.3.5 (Pismeni ispit iz "Uvoda u računarstvo", 31. 8. 2005)**.** *Nađite sve b-ove takve da je*

$$(131)_b + (131)_{b+2} + (131)_{b+4} + (131)_{b+6} = (360)_{b+8}.$$

Rješenje. Ponovno, raspisom po definiciji dobijamo da je jedino rješenje $b = 24$. □

Zadatak 1.3.6. *Za koje b vrijedi*

$$(123)_b = (321)_{b-12}?$$

Rješenje. Raspisom po definiciji dobijemo:

$$\begin{aligned}
(123)_b &= (321)_{b-12} \\
b^2 + 2b + 3 &= 3(b-12)^2 + 2(b-12) + 1 \\
&= 3b^2 - 70b + 409 \\
2b^2 - 72b + 406 &= 0 \\
b^2 - 36b + 203 &= 0 \\
(b-29)(b-7) &= 0
\end{aligned}$$

Kandidati za rješenja su $b_1 = 29$ i $b_2 = 7$. No, iz zadatka vidimo da mora vrijediti:

$$b > 3 \text{ i } b - 12 > 3,$$

što ukupno daje $b > 15$. Zbog toga odbacujemo b_2, a uvrštavanjem $b = 29$ (primijetimo: $b - 12 = 17$) u formulu zadatka dobijamo:

$$(123)_{29} = (902)_{10} = (321)_{17},$$

pa je konačno (i jedino) rješenje zadatka $b = 29$. □

Zadatak 1.3.7. *Za koji b vrijedi*

$$(101010)_b = (10101)_{b+1}?$$

Rješenje.

$$\begin{aligned}
(101010)_b &= (10101)_{b+1} \\
b^5 + b^3 + b &= (b+1)^4 + (b+1)^2 + 1 \\
&= b^4 + 4b^3 + 6b^2 + 4b + 1 + b^2 + 2b + 1 + 1 \\
(b-3)(b^2 + b + 1)^2 &= 0
\end{aligned}$$

Jedino realno rješenje jednadžbe je $b = 3$. Uvrštavanjem u formulu zadatka dobijamo:

$$(101010)_3 = (273)_{10} = (10101)_4.$$

□

Uvjet može biti i naizgled složeniji, no rješavanje se ponovno svodi na raspis po definiciji i provjeravanje svih potrebnih uvjeta:

Zadatak 1.3.8 (Pismeni ispit iz "Uvoda u računarstvo", 5. 7. 2006)**.** *Neka je zadana baza $b \in \mathbb{N}$. Za koje dvoznamenkaste brojeve $(xy)_b$ vrijedi da je $(xy)_b \cdot (11)_b$ palindrom (u ovisnosti o parametru b)?*

Palindrom je niz znakova (ili znamenaka) koji se jednako čita s lijeva na desno i s desna na lijevo. Na primjer: 11, 616, ABBA i sl.

Rješenje. Označimo traženi broj s $a = (xy)_b = x \cdot b + y$. Tada je

$$(11)_b \cdot a = x \cdot b^2 + (x + y) \cdot b + y$$

(za svaki b, tj. u svim brojevnim sustavima). Imamo sljedeće slučajeve:

1. $x + y < b \Rightarrow (11)_b \cdot a = (x(x + y)y)_b \Rightarrow x = y < \frac{b}{2}$
2. $x + y \geq b \Rightarrow 0 \leq x + y - b < 2 \cdot b$, pa broj – rastavljen na znamenke – izgleda ovako: $(x + 1) \cdot b^2 + (x + y - b) \cdot b + y$. Ponovno imamo dva slučaja:
   1. $x + 1 = b \Rightarrow (11)_b \cdot a = (10cy)_b$, gdje je $c = x + y - b$, pa (zato jer je $(11)_b \cdot a$ palindrom), zaključujemo: $y = 1$, $c = 0$. Provjerom uvjeta $c = 0$ dobijemo:

      $$0 = c = x + y - b = b - 1 + 1 - b = 0,$$

      pa je dobiveno rješenje $a = ((b - 1)1)_b$ dobro.
   2. $x + 1 < b$ (tj. $x < b - 1$, što znači da je na početku izraza jedna znamenka). Tada je $y = x + 1$ (što je manje od b, pa je y dobra znamenka).

Konačno rješenje zadatka, dobiveno spajanjem rješenja i uvjeta svih slučajeva, je:

$$\begin{aligned} a \in & \left\{ (xx)_b,\ x < \frac{b}{2} \right\} \cup \\ & \{((b-1)1)_b\} \cup \\ & \left\{ (x(x+1))_b,\ x < b-1,\ x \geq \frac{(b-1)}{2} \right\}. \end{aligned}$$

□

Neka svojstva dekadskog sustava vrijede i općenitije. Dva ćemo pokazati u sljedećim zadacima:

Zadatak 1.3.9. *Broj zapisan u bazi b je djeljiv s brojem $b-1$ (tj. s najvećom znamenkom u bazi b) ako i samo ako je njegova suma znamenaka djeljiva s $b-1$.*

Primjer ove tvrdnje je djeljivost s 9 u dekadskom sustavu.

Rješenje. Neka je zadan broj

$$a = (a_n a_{n-1} \dots a_0)_b.$$

Tada je

$$\begin{aligned} a &= \sum_{i=0}^{n} a_i \cdot b^i = \sum_{i=0}^{n} a_i \cdot b^i - \sum_{i=0}^{n} a_i + \sum_{i=0}^{n} a_i \\ &= \sum_{i=0}^{n} a_i \cdot (b^i - 1) + \sum_{i=0}^{n} a_i. \end{aligned}$$

Primijetimo:

$$b^i - 1 = (b-1) \cdot (b^{i-1} + b^{i-2} + \dots + 1),$$

pa je $b^i - 1$ djeljivo s $b-1$. Zaključujemo da je a djeljiv s $b-1$ ako i samo ako je s $b-1$ djeljivo i $\sum_{i=0}^{n} a_i$, što je upravo suma znamenaka broja a zapisanog u bazi b. □

Zadatak 1.3.10. *Neka je $k \in \mathbb{N}_0$, te neka je $b = 2(2k+1)$ paran prirodni broj takav da je $\frac{b}{2}$ neparan. Dokažite da tada broj*

$$\left(\frac{b}{2} + 1\right)^i,$$

prikazan u bazi b ima zadnju znamenku $\frac{b}{2} + 1$ za sve $i > 0$.

Primjer ove tvrdnje je znamenka 6 u dekadskom sustavu (jer je $10 = 2(2 \cdot 2 + 1)$).

Rješenje. Uvedimo oznaku: $x = \frac{b}{2} + 1$. Dokaz provodimo indukcijom. Za $i = 1$, istinitost tvrdnje je očita. Za $i = 2$ imamo:

$$\begin{aligned} x^2 &= \left(\frac{b}{2} + 1\right)^2 = \frac{b^2}{4} + b + 1 = \frac{b^2}{4} + \frac{b}{2} + \frac{b}{2} + 1 \\ &= \frac{b+2}{4} \cdot b + x = \frac{2(2k+1)+2}{4} \cdot b + x = (k+1)b + x, \end{aligned}$$

što pokazuje da je zadnja znamenka broja x^2 prikazanog u bazi b upravo x.

Pretpostavimo da tvrdnja vrijedi za sve $i < j$, te provjerimo vrijedi li i za j. Kako za $(x)_b^{j-1}$ tvrdnja vrijedi (po pretpostavci), možemo uvesti oznaku

$$(x)_b^{j-1} = yb + x, \quad y \in \mathbb{N}_0.$$

Vrijedi:

$$\begin{aligned}(x)_b^j &= (x)_b^{j-1} \cdot (x)_b = (yb+x)\cdot x = xyb + x^2 \\ &= xyb + (k+1)b + x = (xy+k+1)b + x,\end{aligned}$$

čime smo pokazali da je posljednja znamenka $(x)_b^j$ zaista x, što znači da tvrdnja vrijedi i za j. Prema principu matematičke indukcije, to znači da tvrdnja vrijedi za sve $i \in \mathbb{N}$. □

Tvrdnja može biti i općenitija, jer posljednja znamenka od proizvoljnog x^i (za $i \geq 1$) ovisi samo o posljednjoj znamenci od x. Dakle, ako su zadovoljeni uvjeti tvrdnje iz zadatka, i-ta potencija svakog broja sa zadnjom znamenkom $\frac{b}{2}+1$ imat će opet zadnju znamenku $\frac{b}{2}+1$ (naravno, ako je broj zapisan u bazi b).

Zadatak 1.3.11. *Dokažite da tvrdnja zadatka 1.3.9 ne vrijedi samo za $b-1$, nego i za sve djelitelje od $b-1$ (npr. djeljivost s 3 u dekadskoj bazi).*

Zadatak 1.3.12. *Broj $a = (a_n a_{n-1} \ldots a_0)$ zapisan u bazi b je djeljiv s brojem $b+1$ (tj. s $(11)_b$) ako i samo ako je broj*

$$\sum_{i=0}^{n} (-1)^n a_i = a_0 - a_1 + a_2 - a_3 + \ldots$$

također djeljiv s $b+1$.

Rješenje. Tvrdnju je jednostavno dokazati upotrebom kongruencija, ali može i direktno, primjenom sljedećih identiteta:

$$\begin{aligned}x^n + y^n &= (x+y)(x^{n-1} - x^{n-2}y + \cdots - xy^{n-1} + y^{n-1}), \quad n \text{ paran}, \\ x^n - y^n &= (x+y)(x^{n-1} - x^{n-2}y + \cdots + xy^{n-1} - y^{n-1}), \quad n \text{ neparan}.\end{aligned}$$

Pretpostavimo da je n neparan (tj. da broj ima parno mnogo znamenaka). Tada je

$$\begin{aligned}a &= \sum_{i=0}^{n} a_i b^i = \sum_{i=0}^{(n-1)/2} (a_{2i} b^{2i} + a_{2i+1} b^{2i+1}) \\ &= \sum_{i=0}^{(n-1)/2} \big((a_{2i} b^{2i} + a_{2i+1} b^{2i+1}) + (a_{2i} - a_{2i+1})\big) - \sum_{i=0}^{(n-1)/2} (a_{2i} - a_{2i+1}) \\ &= \sum_{i=0}^{(n-1)/2} (a_{2i}(b^{2i}+1) + a_{2i+1}(b^{2i+1}-1)) - \sum_{i=0}^{(n-1)/2} (a_{2i} - a_{2i+1}) \\ &= \sum_{i=0}^{(n-1)/2} (a_{2i}(b+1)p_i + a_{2i+1}(b+1)q_i) - \sum_{i=0}^{(n-1)/2} (a_{2i} - a_{2i+1}) \\ &= (b-1) \sum_{i=0}^{(n-1)/2} (a_{2i} p_i + a_{2i+1} q_i) - \underbrace{\sum_{i=0}^{(n-1)/2} (a_{2i} - a_{2i+1})}_{=\sum_{i=0}^{n} (-1)^n a_i},\end{aligned}$$

pri čemu su p_i i q_i ostaci nakon izlučivanja $b+1$ iz $b^{2i}+1$ i $b^{2i+1}-1$ (respektivno). Ti ostaci su cijeli brojevi, prema formulama za rastavljanje x^n+y^n i x^n-y^n.

Ako je n paran (tj. $n-1$ je neparan), onda iz broja a izdavajamo vodeću znamenku, te na preostalih $n-1$ znamenaka primjenjujemo prethodni rezultat:

$$\begin{aligned} a &= \sum_{i=0}^{n} a_i b^i = a_n + \sum_{i=0}^{n-1} a_i b^i \\ &= (b-1) \sum_{i=0}^{(n-2)/2} (a_{2i} p_i + a_{2i+1} q_i) + \underbrace{a_n - \sum_{i=0}^{(n-2)/2} (a_{2i} - a_{2i+1})}_{=\sum_{i=0}^{n} (-1)^n a_i}. \end{aligned}$$

Alternativno, broj s neparno mnogo znamenaka (n paran) možemo promatrati i kao broj s jednom znamenkom više, tako da definiramo $a_{n+1} := 0$. □

Primjer ove tvrdnje je djeljivost s 11 u dekadskom sustavu.

Poglavlje 2

Račun sudova

Sud ili izjava je svaka (smislena) rečenica čiju istinitost možemo utvrditi, tj. koja je ili istinita ili lažna. Na primjer:

- $17 + 19 = 36$ (sud, istinit)
- $17 > 19$ (sud, lažan)
- $17 > x$ (nije sud, dok god ne znamo koliki je x)
- Popocatépetl je aktivni vulkan u Čileu (sud, lažan; Popocatépetl se nalazi u Meksiku).
- Hoću li uspjeti položiti ovaj kolegij? (nije sud, jer ne iskazuje tvrdnju čiju bismo istinitost mogli provjeriti)

Jednostavne sudove zamjenjujemo sudovnim varijablama (najčešće ih označavamo velikim ili malim latiničnim slovima), te od njih slažemo složene izjave pomoću logičkih operatora. Uz varijable, imamo i dvije konstante: 1 (istina, true, $\top$) i 0 (laž, false, $\bot$). Operatori koje obično koristimo prikazani su u tablici 2.1.

Operator negacije ima najveći prioritet; zatim dolazi operator konjunkcije, te zatim operator disjunkcije i isključivo ILI (oni imaju jednake prioritete). Nakon njih dolaze implikacija i, nakon nje, ekvivalencija.

Ako niste sigurni koji su prioriteti operatora, pišite zagrade!

2.1 Tablice istinitosti

Istinitost pojedine formule možemo jednostavno provjeriti tzv. tablicom istinitosti.

Zadatak 2.1.1. *Za koje vrijednosti a i b je formula $a + \bar{b}$ istinita?*

Naziv			
hrv.	engl.	Simboli	Opis
I	AND	$\cdot$, $\wedge$	$a \cdot b$ je istina ako i samo ako su i a i b istiniti
ILI	OR	$+$, $\vee$	$a + b$ je istina ako i samo ako je a istinit ili b istinit (ili oba)
NE	NOT	$\overline{a}$, $\neg a$	$\overline{a}$ je istina ako i samo ako je a laž
isključivo ILI	XOR	$\oplus$	$a \oplus b$ je istina ako i samo ako je ili a istinit ili b istinit (ali ne oba)

Tablica 2.1: Opisi nekih logičkih operatora

Rješenje. Tablicu istinitosti slažemo navođenjem svih kombinacija vrijednosti varijabli a i b, te računanjem vrijednosti formule za svaku kombinaciju:

Tijekom postupka, preporučljivo je računati i podizraze, na primjer $\overline{b}$.

a	b	$\overline{b}$	$a + \overline{b}$
0	0	1	1
0	1	0	0
1	0	1	1
1	1	0	1

Dakle, formula $a + \overline{b}$ je istinita ako je a istinit ili b lažan. □

Logičke operatore možemo zadati pomoću tablice istinitosti (v. tablicu 2.1). Iz priložene tablice istinitosti odmah vidimo da vrijedi:

$$(a \Leftrightarrow b) = \overline{a \oplus b}.$$

a	b	$a + b$	$a \cdot b$	$\overline{a}$	$a \Rightarrow b$	$a \Leftrightarrow b$	$a \oplus b$
0	0	0	0	1	1	1	0
0	1	1	0	1	1	0	1
1	0	1	0	0	0	0	1
1	1	1	1	0	1	1	0

Tablica 2.2: Logički operatori

Napomena 2.1.1 (Varijable u tablici istinitosti)**.** *Tablica istinitosti za k varijabli ima 2^k redaka. Pri tome, najlakše je prvi stupac popuniti s 2^{k-1} nula, te s 2^{k-1} jedinica, drugi*

stupac s 2^{k-2} nula i 2^{k-2} jedinica, te opet 2^{k-2} nula i 2^{k-2} jedinica, idući opet s dvostruko manjim grupicama nula i jedinica itd. U stupcu posljednje varijable trebaju ispasti nule i jedinice naizmjence.

Zadatak 2.1.2. *Dokažite:*

$$a + \overline{a} = 1, \quad a \cdot \overline{a} = 0.$$

Uputa. Složite tablicu istinitosti za varijablu a (trebala bi imati dva retka). □

Zadatak 2.1.3. *Složite tablicu istinitosti za izraz*

$$f = (a + b) \cdot (c + d).$$

Rješenje. Kako ovdje imamo četiri varijable, tablica istinitosti imat će $2^4 = 16$ redaka (ovdje rastavljeno u $8 + 8$ redaka):

a	b	c	d	$a+b$	$c+d$	**f**
0	0	0	0	0	0	**0**
0	0	0	1	0	1	**0**
0	0	1	0	0	1	**0**
0	0	1	1	0	1	**0**
0	1	0	0	1	0	**0**
0	1	0	1	1	1	**1**
0	1	1	0	1	1	**1**
0	1	1	1	1	1	**1**

a	b	c	d	$a+b$	$c+d$	**f**
1	0	0	0	1	0	**0**
1	0	0	1	1	1	**1**
1	0	1	0	1	1	**1**
1	0	1	1	1	1	**1**
1	1	0	0	1	0	**0**
1	1	0	1	1	1	**1**
1	1	1	0	1	1	**1**
1	1	1	1	1	1	**1**

□

Zadatak 2.1.4. *Pomoću tablica istinitosti dokažite sljedeće važne identitete:*

a) De Morganove formule:

$$\overline{a + b} = \overline{a} \cdot \overline{b}, \quad \overline{a \cdot b} = \overline{a} + \overline{b},$$

b) Negacija implikacije:

$$\overline{a \Rightarrow b} = a \cdot \overline{b},$$

c) Obrat po kontrapoziciji:

$$(a \Rightarrow b) = (\overline{b} \Rightarrow \overline{a})$$

d) $(a \Leftrightarrow b) = (a \Rightarrow b) \cdot (b \Rightarrow a)$.

Rješenje. Pokazat ćemo da vrijede De Morganove formule:

a	b	$a+b$	$\overline{\mathbf{a+b}}$	$\overline{a}$	$\overline{b}$	$\overline{\mathbf{a}}\cdot\overline{\mathbf{b}}$	$a\cdot b$	$\overline{\mathbf{a\cdot b}}$	$\overline{\mathbf{a}}+\overline{\mathbf{b}}$
0	0	0	**1**	1	1	**1**	0	**1**	**1**
0	1	1	**0**	1	0	**0**	0	**1**	**1**
1	0	1	**0**	0	1	**0**	0	**1**	**1**
1	1	1	**0**	0	0	**0**	1	**0**	**0**

Istinitost ostalih tvrdnji provjerite za vježbu. □

Zadatak 2.1.5. *Političar N. T. (Nema Takvog) nikada ne laže, osim srijedom kada samo laže. Koje dane u tjednu on može reći:*

a) "Jučer sam govorio istinu i lagat ću sutra"?

b) "Prekjučer sam lagao, a danas i sutra govorit ću istinu"?

c) "Ako nisam lagao jučer, lagat ću sutra"?

d) "Ako danas i sutra govorim istinu, prekosutra ću lagati".

e) "Ako danas i sutra lažem, prekosutra ću govoriti istinu".

Je li izjava pod d) negacija izjave pod e)?

Rješenje. Ovdje nas ne zanima kad je neka izjava istinita, nego kad se podudara s onime što N. T. priča. Drugim riječima, zanima nas kad je

$$\text{izjava} \Leftrightarrow p,$$

gdje je p istina sve dane osim srijede (kad je laž).

Riješimo zadatak pod a). Počinjemo uvođenjem pomoćnih varijabli:

$$a = \text{"Jučer sam govorio istinu"}$$
$$b = \text{"Lagat ću sutra"}$$

Zanima nas kad se istinitost od p poklapa s istinitošću od $a \cdot b$.

Dan	a	b	p	$a\cdot b$	$p \Leftrightarrow (a+b)$
pon	1	0	1	0	0
uto	1	1	1	1	1
sri	1	0	0	0	1
čet	0	0	1	0	0
pet	1	0	1	0	0
sub	1	0	1	0	0
ned	1	0	1	0	0

Dakle, u tablici pratimo kad je formula $a \cdot b$ istinita, a na kraju (posljednji stupac) provjeravamo kad se njena istinitost poklapa s istinitošću tvrdnji koje taj dan govori N. T, jer on istinu može reći svaki dan osim srijede, a srijedom može reći samo laž. Zbog toga je konačno rješenje: **utorak i srijeda**, jer je u utorak izjava istinita (a on utorkom govori istinu), dok je u srijedu izjava lažna (a on srijedom laže). Ostale dane, izjava je lažna, a on bi trebao govoriti istinu, pa zato takvu izjavu ne može dati.

Ostale tvrdnje provjerite za vježbu. Rješenja su:

a) utorak i srijeda,

b) srijeda i petak,

c) utorak, srijeda i četvrtak

d) ponedjeljak i utorak

e) svi dani osim srijede (jer je lijeva strana inkluzije – "danas i sutra (dva uzastopna dana) lažem" – uvijek lažna, pa je inkluzija uvijek istinita).

Iz priloženog je jasno i da izjave pod d) i pod e) nisu jedna drugoj negacije. Izjava koja bi bila negacija izjavi pod d), za rješenje bi imala "svi dani osim ponedjeljka i utorka". □

2.2 Normalne forme

Zadatak 2.2.1. *Dokažite da se svaka formula računa sudova može prikazati samo pomoću negacije (operator NE), konjunkcije (operator I) i disjunkcije (operator ILI).*

Rješenje. Ostale operacije (implikacija, ekvivalencija, isključivo ILI) prikazat ćemo na taj način:

$$\begin{aligned}
(a \Rightarrow b) &= \overline{a} \cdot \overline{b} + \overline{a} \cdot b + a \cdot b, \\
(a \Leftrightarrow b) &= \overline{a} \cdot \overline{b} + a \cdot b, \\
(a \oplus b) &= \overline{a} \cdot b + a \cdot \overline{b}, \\
(a \Rightarrow b) &= \overline{a} + b, \\
(a \Leftrightarrow b) &= (a + \overline{b}) \cdot (\overline{a} + b), \\
(a \oplus b) &= (a + b) \cdot (\overline{a} + \overline{b}).
\end{aligned}$$

Za vježbu provjerite ispravnost navedenih formula.

Primijetite da, primjenom De Morganovih pravila, možemo eliminirati i sve disjunkcije ili sve konjunkcije, tj. da sve formule računa sudova možemo prikazati pomoću negacije i samo konjunkcije ili samo disjunkcije. □

Kako dobiti formulu ako je zadana tablica istinitosti? Postoji više načina.

Algoritam 2.2.1 (Konjunktivna normalna forma)**.** *U tablici istinitosti gledamo sve retke koji sadrže nulu (laž) u rezultatu. Reci su oblika:*

a_1	a_2	$\dots$	a_n	f
$\vdots$	$\vdots$		$\vdots$	$\vdots$
$\mathbf{v_1}$	$\mathbf{v_2}$	$\dots$	$\mathbf{v_n}$	$\mathbf{0}$
$\vdots$	$\vdots$		$\vdots$	$\vdots$

Za svaki takav redak kreiramo izraz:

$$B_i = b_1 + b_2 + \cdots + b_n,$$

gdje je

$$b_j = \begin{cases} \overline{a_j},\ v_j = 1 \\ a_j,\ v_j = 0 \end{cases}.$$

Konjunktivna normalna forma (KNF) izraza f je

$$B_1 \cdot B_2 \cdot \dots \cdot B_k,$$

gdje je k broj nula u stupcu koji odgovara izrazu f u tablici istinitosti.

Pojednostavljeno rečeno: gledamo nule u rezultatu, negiramo varijable koje imaju vrijednost jedan (ostale ne mijenjamo), "zbrajamo" te varijable (i negirane i nepromijenjene), te radimo "umnožak" tako dobivenih "suma".

Algoritam 2.2.2 (Disjunktivna normalna forma)**.** *U tablici istinitosti gledamo sve retke koji sadrže jedinicu (istinu) u rezultatu. Reci su oblika:*

a_1	a_2	$\dots$	a_n	f
$\vdots$	$\vdots$		$\vdots$	$\vdots$
$\mathbf{v_1}$	$\mathbf{v_2}$	$\dots$	$\mathbf{v_n}$	$\mathbf{1}$
$\vdots$	$\vdots$		$\vdots$	$\vdots$

Za svaki takav redak kreiramo izraz:

$$B_i = b_1 \cdot b_2 \cdot \dots \cdot b_n,$$

gdje je

$$b_j = \begin{cases} a_j,\ v_j = 1 \\ \overline{a_j},\ v_j = 0 \end{cases}.$$

Disjunktivna normalna forma (DNF) izraza f je

$$B_1 + B_2 + \cdots + B_k,$$

gdje je k broj jedinica u stupcu koji odgovara izrazu f u tablici istinitosti.

Pojednostavljeno rečeno: gledamo jedinice u rezultatu, negiramo varijable koje imaju vrijednost nula (ostale ne mijenjamo), "množimo" te varijable (i negirane i nepromijenjene), te radimo "zbroj" tako dobivenih "produkata".

Rješenje zadatka 2.2.1 pomoću normalnih formi. Da bismo riješili zadatak, potrebno je prvo složiti tablicu istinitosti za operatore implikacije, ekvivalencije i isključivog ILI. Iznimno, označit ćemo i brojeve redaka (prvi stupac) za potrebe objašnjavanja postupka:

	a	b	$a \Rightarrow b$	$a \Leftrightarrow b$	$a \oplus b$
1.	0	0	1	1	0
2.	0	1	1	0	1
3.	1	0	0	0	1
4.	1	1	1	1	0

Konjunktivnu normalnu formu za formulu $a \Rightarrow b$ dobijemo promatranjem trećeg retka (jer se samo u njemu nalazi nula u stupcu koji odgovara formuli $a \Rightarrow b$). U tom retku, varijabla a ima vrijednost 1, a varijabla b ima vrijednost 0. Prema algoritmu 2.2.1, trebamo negirati varijablu a i varijablu b ostaviti nepromijenjenu, te zbrojiti dobiveno:

$$B_1 = \overline{a} + b.$$

Kako se nula nalazi samo u trećem retku, to je B_1 jedini u konjunktivnoj normalnoj formi za $a \Rightarrow b$, pa je konačno rješenje

$$(a \Rightarrow b) = \overline{a} + b.$$

Disjunktivnu normalnu formu za formulu $a \Rightarrow b$ dobijemo promatranjem redaka koji sadrže jedinice (to su reci 1, 2 i 4). U prvom retku, i varijabla a i varijabla b imaju vrijednost nula, pa ih – prema algoritmu 2.2.2 – negiramo, što znači da je

$$B_1 = \overline{a} \cdot \overline{b}.$$

U drugom retku, varijabla a ima vrijednost nula (pa ju negiramo), dok varijabla b ima vrijednost 1 (pa ostaje nepromijenjena). Dakle,

$$B_2 = \overline{a} \cdot b.$$

Analogno, za treći redak imamo:

$$B_3 = a \cdot b.$$

Konačno rješenje je:

$$(a \Rightarrow b) = (\overline{a} \cdot \overline{b}) + (\overline{a} \cdot b) + (a \cdot b).$$

Na sličan način riješite formule $a \Leftrightarrow b$ i $a \oplus b$ (rješenja, do na poretke operanada, moraju biti jednaka onima iz originalnog rješenja zadatka). □

Napomena 2.2.1. *Primijetite: ako imamo n varijabli,*

- *tablica istinitosti imat će 2^n redaka (ne računajući naslovni redak).*
- *ukupni broj B-ova u konjunktivnoj i disjunktivnoj normalnoj formi bit će također 2^n.*

Napomena 2.2.2. *Generalno, ako je moguće birati koju normalnu formu koristiti, dobro je koristiti konjunktivnu ako ima malo jedinica, a disjunktivnu ako ima malo nula. Ipak, ako se očekuje i pojednostavljivanje izraza (poglavlje 2.3), često je praktičnije koristiti disjunktivnu normalnu formu jer kod nje nije potrebno "izmnožiti" elemente u zagradama.*

Zadatak 2.2.2. *Napišite formule računa sudova za izraze f, g i h određene sljedećom tablicom istinitosti (zbog preglednosti je podijeljena je u dva dijela):*

a	b	c	f	g	h
0	0	0	1	1	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	1	1	1	0

a	b	c	f	g	h
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	0	0	0
1	1	1	1	1	0

Rješenje. U zadatku ne piše koju normalnu formu treba napisati (dapače, ne piše niti da trebaju biti normalne forme), pa možemo birati. Za izraz f je svejedno koju normalnu formu izaberemo jer tom izrazu odgovara jednag broj jedinica i nula. Napisat ćemo obje normalne forme za f:

$$\begin{aligned} &\text{KNF:} \quad f = (a + \overline{b} + c) \cdot (\overline{a} + b + c) \cdot (\overline{a} + b + \overline{c}) \cdot (\overline{a} + \overline{b} + c), \\ &\text{DNF:} \quad f = \overline{a} \cdot \overline{b} \cdot \overline{c} + \overline{a} \cdot \overline{b} \cdot c + \overline{a} \cdot b \cdot c + a \cdot b \cdot c. \end{aligned}$$

Izrazu g odgovaraju dvije nule i čak šest jedinica, pa je lakše napisati konjunktivnu normalnu formu:

$$g = (a + \overline{b} + c) \cdot (\overline{a} + \overline{b} + c).$$

Izrazu h odgovara čak šest nula i samo dvije jedinice, pa je lakše napisati disjunktivnu normalnu formu:

$$h = \overline{a} \cdot b \cdot \overline{c} + a \cdot \overline{b} \cdot c.$$

□

Interaktivnu vježbalicu za normalne forme možete naći na adresi
http://degiorgi.math.hr/prog1/apps/knfdnf.pl

2.3 Pojednostavljivanje logičkih izraza

Formule izražene u normalnim formulama u pravilu su komplicirane. Primjenom jednostavnih identiteta (poput De Morganovih pravila), formule možemo značajno pojednostaviti, što je izuzetno važno u primjenama. Na primjer, logički izraz može predstavljati uvjet koji neki program treba izvršiti (v. zadatak 7.1.4), pa njegova složenost može značajno usporiti ili ubrzati izvođenje programa. Također, logički izraz može predstavljati i elektronički sklop koji treba sastaviti. Kompliciraniji sklop (koji je rezultat kompliciranijeg izraza) ima više elemenata, što povlači veću cijenu, veću potrošnju energije, sporije izvođenje operacija i veći konačni sklop.

U praksi, većina elektroničkih sklopova (iznimka je DRAM) sastoji se od NAND-vrata (NOT AND) i NOR-vrata (NOT OR) s tri ili više ulaza.

Neke od formula koje mogu pomoći prilikom pojednostavljivanja logičkih izraza navedene su u tablici 2.3.

1.	$a+b=b+a$	$a\cdot b=b\cdot a$
2.	$a+(b+c)=(a+b)+c$	$a\cdot(b\cdot c)=(a\cdot b)\cdot c$
3.	$a+0=a$	$a\cdot 0=0$
4.	$a+1=1$	$a\cdot 1=a$
5.	$a+a=a$	$a\cdot a=a$
6.	$a\cdot(b+c)=a\cdot b+a\cdot c$	$a+b\cdot c=(a+b)\cdot(a+c)$
7.	$\bar{\bar{a}}=a$	
8.	$\bar{1}=0$	$\bar{0}=1$
9.	$\bar{a}+a=1$	$\bar{a}\cdot a=0$
10.	$a+\bar{a}\cdot b=a+b$	$a\cdot(\bar{a}+b)=a\cdot b$
11.	$\overline{a+b}=\bar{a}\cdot\bar{b}$	$\overline{a\cdot b}=\bar{a}+\bar{b}$
12.	$a+a\cdot b=a$	$a\cdot(a+b)=a$
13.	$a\cdot b+\bar{a}\cdot c+b\cdot c=a\cdot b+\bar{a}\cdot c$	
	$(a+b)\cdot(\bar{a}+c)\cdot(b+c)=(a+b)\cdot(\bar{a}+c)$	

Tablica 2.3: Neke formule za pojednostavljivanje logičkih izraza

Napomena 2.3.1 (Česta greška)**.** *U računu sudova* ***NE VRIJEDI*** $1+1=2$*; štoviše, u računu sudova* 2 *uopće nema nikakvo značenje!*

Zadatak 2.3.1. *Dokažite formule iz tablice 2.3 pomoću tablica istinitosti.*

Zadatak 2.3.2. *Pojednostavite formule računa sudova:*

a) $f=a\cdot b\cdot c+a\cdot\bar{b}\cdot c+a\cdot b\cdot\bar{c}$,

b) $g = \overline{a} \cdot b \cdot c + a \cdot \overline{b} \cdot c + a \cdot b \cdot \overline{c} + a \cdot b \cdot c$,

c) $h = c \cdot a + \overline{\overline{\overline{c} \cdot b} \cdot \overline{\overline{c} \cdot a}} + \overline{\overline{c} + \overline{b}}$,

d) $k = a \cdot \overline{b} \cdot (a + b) + \overline{a \cdot \overline{b}} \cdot \overline{b + a}$.

Općenito, formule je najlakše pojednostaviti tako da ih se svede na disjunktivnu formu (ne nužno normalnu, tj. ne mora svaki sumand sadržavati sve varijable). Zatim se spretnim izlučivanjem i sličnim pravilima (v. tablicu 2.3) formula pojednostavljuje.

Rješenje.

a) Počinjemo izlučivanjem zajedničkog podizraza $(a{\cdot}c)$ iz prva dva elementa disjunkcije:

$$\begin{aligned} f &= a \cdot b \cdot c + a \cdot \overline{b} \cdot c + a \cdot b \cdot \overline{c} \\ &= a \cdot c \cdot (\underbrace{b + \overline{b}}_{=1}) + a \cdot b \cdot \overline{c} \\ &= a \cdot c + a \cdot b \cdot \overline{c} = a \cdot (\underbrace{c + b \cdot \overline{c}}_{=b+c \text{ (pr. 10)}}) \\ &= a \cdot (b + c). \end{aligned}$$

b) Primijetimo:

$$g = f + \overline{a} \cdot b \cdot c \text{ i } b + c = b + c + b \cdot c.$$

Sada imamo:

$$\begin{aligned} g &= f + \overline{a} \cdot b \cdot c = a \cdot (b + c) + \overline{a} \cdot b \cdot c \\ &= a \cdot (b + c + b \cdot c) + \overline{a} \cdot b \cdot c \\ &= a \cdot (b + c) + a \cdot b \cdot c + \overline{a} \cdot b \cdot c \\ &= a \cdot (b + c) + (a + \overline{a}) \cdot b \cdot c \\ &= a \cdot (b + c) + b \cdot c. \end{aligned}$$

Možemo i direktno:

$$\begin{aligned} g &= \overline{a} \cdot b \cdot c + a \cdot \overline{b} \cdot c + a \cdot b \cdot \overline{c} + a \cdot b \cdot c \\ &= (\overline{a} + a) \cdot b \cdot c + a \cdot \overline{b} \cdot c + a \cdot b \cdot \overline{c} \\ &= b \cdot c + a \cdot \overline{b} \cdot c + a \cdot b \cdot \overline{c} \\ &= (b + a \cdot \overline{b}) \cdot c + a \cdot b \cdot \overline{c} \\ &= (a + b) \cdot c + a \cdot b \cdot \overline{c} = a \cdot c + b \cdot c + a \cdot b \cdot \overline{c} \\ &= a \cdot (c + b \cdot \overline{c}) + b \cdot c = a \cdot (b + c) + b \cdot c \\ &= a \cdot b + a \cdot c + b \cdot c. \end{aligned}$$

Primijetite da formula $a \cdot b + a \cdot c + b \cdot c$ ima 5 operatora, dok prethodna formula, $a \cdot (b+c) + b \cdot c$, ima samo 4. Zbog toga kao rješenje uzimamo formulu $a \cdot (b+c) + b \cdot c$.

Inače, ovisno o izboru izlučivanja, mogli smo dobiti i druga, jednako dobra rješenja. Na primjer:

$$a \cdot b + (a + b) \cdot c.$$

c) Ovdje krećemo s višestrukom primjenom De Morganovih pravila kako bismo se riješili složenih izraza ispod negacija (jer s njima najčešće ne možemo ništa napraviti):

$$\begin{aligned} h &= c \cdot a + \overline{\overline{\overline{c} \cdot b} \cdot \overline{\overline{c} \cdot a}} + \overline{\overline{c} + \overline{b}} \\ &= c \cdot a + \overline{\overline{\overline{c} \cdot b}} + \overline{\overline{\overline{c} \cdot a}} + \overline{\overline{c}} \cdot \overline{\overline{b}} \\ &= c \cdot a + \overline{c} \cdot b + \overline{c} \cdot a + c \cdot b \\ &= a \cdot (c + \overline{c}) + (\overline{c} + c) \cdot b = a + b. \end{aligned}$$

Kao što vidimo, konačna formula ne mora sadržavati sve varijable.

d) Ponovno krećemo primjenom De Morganovih pravila, te "množenjem" zagrada:

$$\begin{aligned} k &= a \cdot \overline{b} \cdot (a + b) + \overline{a \cdot \overline{b}} \cdot \overline{b + a} \\ &= \underbrace{a \cdot \overline{b} \cdot a}_{=a \cdot \overline{b}} + \underbrace{a \cdot \overline{b} \cdot b}_{=0} + (\overline{a} + b) \cdot (\overline{a} \cdot \overline{b}) \\ &= a \cdot \overline{b} + \overline{a} \cdot \overline{a} \cdot \overline{b} + \underbrace{b \cdot \overline{a} \cdot \overline{b}}_{=0} = a \cdot \overline{b} + \overline{a} \cdot \overline{b} \\ &= (a + \overline{a}) \cdot \overline{b} = \overline{b}. \end{aligned}$$

□

Prilikom pojednostavljivanja, svaka dobivena formula mora biti ekvivalentna početnoj. Jednakost formula možete provjeriti na službenoj stranici kolegija:
http://degiorgi.math.hr/prog1/apps/logika_sudova.pl

Zadatak 2.3.3 (Pismeni ispit iz "Uvoda u računarstvo", 14.2.2005)**.** *Definirajte formulu računa sudova koja kao ulaz uzima binarne znamenke x_0, x_1 i x_2, te vraća 1 ako se broj $(x_2x_1x_0)_2$ može prikazati kao umnožak dva prosta broja. Inače vraća 0. Pojednostavite dobiveni izraz!*

Napomena 2.3.2 (Česta greška)**.** *Broj 1 nije ni prost ni složen!*

Rješenje. Najprije slažemo tablicu istinitosti:

x_2	x_1	x_0	$(x_2x_1x_0)_2$	f
0	0	0	0	0
0	0	1	1	0
0	1	0	2	0
0	1	1	3	0

x_2	x_1	x_0	$(x_2x_1x_0)_2$	f	
1	**0**	**0**	**4**	**1**	$(4 = 2 \cdot 2)$
1	0	1	5	0	
1	**1**	**0**	**6**	**1**	$(6 = 2 \cdot 3)$
1	1	1	7	0	

Sada vidimo i zašto je ovakav raspored varijabli dobar: brojevi koje one daju (ako ih interpretiramo kao binarne znamenke) idu po redu od 0 do $2^n - 1$, gdje je n broj varijabli (ovdje je $n = 3$).

Problem je očito lakše riješiti pomoću disjunktivne, nego pomoću konjunktivne normalne forme jer imamo samo dvije jedinice i čak šest nula:

$$f = x_2 \cdot \overline{x_1} \cdot \overline{x_0} + x_2 \cdot x_1 \cdot \overline{x_0} = x_2 \cdot \overline{x_0}.$$

□

Zadatak 2.3.4 (Pismeni ispit iz "Uvoda u računarstvo", 22.4.2002)**.** *Napišite formulu računa sudova (koja se sastoji samo od operatora ILI i negacije) koja za ulaz $x = (x_2x_1x_0)_2$ provjerava je li x nultočka funkcije $f(x) = x^3 - x^2 - 4x + 4$.*

Upute. Izračunajte nultočke funkcije ili uvrstite sve moguće vrijednosti x (cijeli brojevi od 0 do 7), te napišite tablicu istinitosti. Zatim izvedite konjunktivnu ili disjunktivnu normalnu formu, te primjenom De Morganovih pravila eliminirajte sve konjunkcije. □

2.4 Osnovni sklopovi

Kao što smo naveli, logički izrazi često se koriste prilikom izrade elektroničkih sklopova. Izraze možemo i nacrtati, na način da svaki operator zamijenimo odgovarajućim sklopom, koje zatim povezujemo u cjelinu. Sklopovi koji odgovaraju osnovnim operatorima su dani su u tablici 2.4.

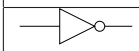
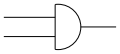
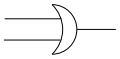
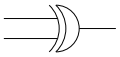
Sklop gradimo tako da svakoj varijabli dodijelimo jednu ulaznu "žicu" (liniju), te ju povezujemo sa sklopovima koji odgovaraju operatorima koje ti sklopovi predstavljaju. Na isti način nastavljamo gradnju s izlazima iz sklopova.

Primjer 2.4.1. *Nacrtajmo sklop koji odgovara izrazu*

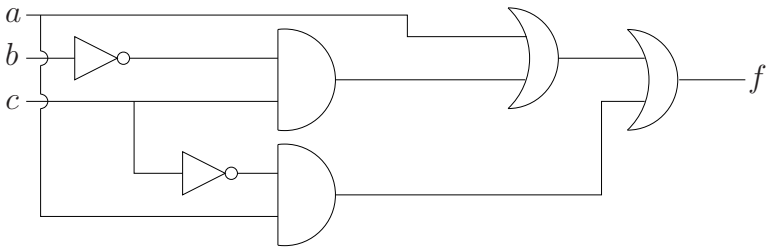
$$f = a + \overline{b} \cdot c + a \cdot \overline{c}.$$

Rješenje je priloženo na slici 2.1.

"Mostići" pokraj varijabli b i c služe da se naglasi da se "žica" od a ne siječe sa "žicama" od b i c.

Simbol	Naziv	Oznake
	negacija	$\overline{a}$, $\neg a$
	konjunkcija	$a \cdot b$, $a \wedge b$
	disjunkcija	$a + b$, $a \vee b$
	isključivo ILI	$a \oplus b$

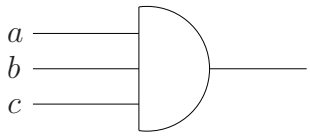
Tablica 2.4: Osnovni sklopovi



Slika 2.1: Sklop $f = a + \overline{b} \cdot c + a \cdot \overline{c}$

Zadatak 2.4.1. *Nacrtajte sklopove koji odgovaraju originalnim i pojednostavljenim izrazima iz zadatka 2.3.2.*

Napomena 2.4.1. *Ponekad se u operatore ubacuje više ulaznih "žica" nego je to uobičajeno, kao – na primjer – kod sklopa AND na slici 2.2. Time se želi skraćeno prikazati izraz*



Slika 2.2: Sklop AND s tri ulazne "žice"

$a \cdot b \cdot c$. Ipak, iako je nacrtan samo jedan sklop, ***on je samo pokrata i zato se broji kao dva sklopa!*** *Ovakvim "zahvatima" u crtanju ne smanjujete broj operatora/sklopova u rješenju zadatka!*

2.5 Poluzbrajalo i potpuno zbrajalo

Zadatak 2.5.1 (Poluzbrajalo)**.** *Konstruirajte (nacrtajte) poluzbrajalo (engl. half adder), tj. sklop koji ima ulaze x i y (jednoznamenkasti binarni brojevi), te izlaze s (sum) i c (carry) za koje vrijedi:*

$$s = \text{znamenka jedinica u sumi } x + y$$
$$c = \text{znamenka desetica u sumi } x + y$$

Očito, s predstavlja "sumu", a c "jedan dalje" prilikom zbrajanja dva binarna broja.

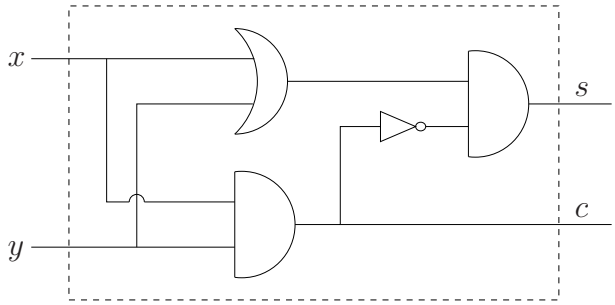
Rješenje. Sastavimo tablicu istinitosti za s i c:

x	y	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Primijetite: $x + y = (sc)_2$.
Izraze nalazimo pomoću normalnih formi:

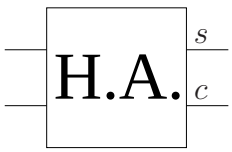
$$\begin{aligned} &\text{DNF:} \quad c = x \cdot y \\ &\text{KNF:} \quad s = (x + y) \cdot (\overline{x} + \overline{y}) = (x + y) \cdot \overline{x \cdot y} \end{aligned}$$

Konačno, možemo nacrtati poluzbrajalo (v. sliku 2.3). □



Slika 2.3: Poluzbrajalo

Poluzbrajalo označavamo kao na slici 2.4. Kako izrazi c i s ne komutiraju, prilikom



Slika 2.4: Simbol poluzbrajala

crtanja poluzbrajala **nužno je naglasiti koji je koji** (tj. na svaku izlaznu "žicu" napisati je li ona s ili c).

Zadatak 2.5.2 (Potpuno zbrajalo)**.** *Konstruirajte (nacrtajte) potpuno zbrajalo (engl. full adder), tj. sklop koji ima ulaze x, y i z (jednoznamenkasti binarni brojevi), te izlaze s (sum) i c (carry) za koje vrijedi:*

$$s = \text{znamenka jedinica u sumi } x + y + z$$
$$c = \text{znamenka desetica u sumi } x + y + z$$

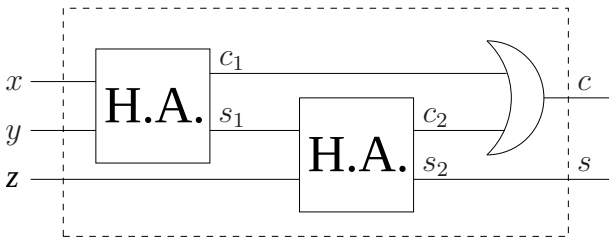
Očito, s predstavlja "sumu", a c "jedan dalje" prilikom zbrajanja tri binarna broja.

Rješenje. Sastavimo tablicu istinitosti za s i c:

x	y	z	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Primijetite: $x + y + z = (sc)_2$.

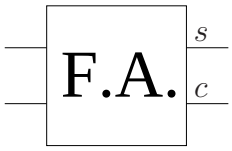
Izraze možemo lako dobiti pomoću normalnih formi (napravite to sami za vježbu). No, potpuno zbrajalo možemo nacrtati i koristeći poluzbrajala, simulirajući time zbrajanje "znamenku po znamenku" (v. sliku 2.5). Suma $x + y + z$ ima jednaku zadnju znamenku



Slika 2.5: Potpuno zbrajalo

kao i suma $s_1 + z$ (pri čemu je s_1 zadnja znamenka sume $x + y$), a prenosimo "jedan dalje" ako imamo "jedan dalje" u bilo kojem od ta dva zbrajanja (tj. ako je c_1 jednak 1 ili ako je c_2 jednak 1). □

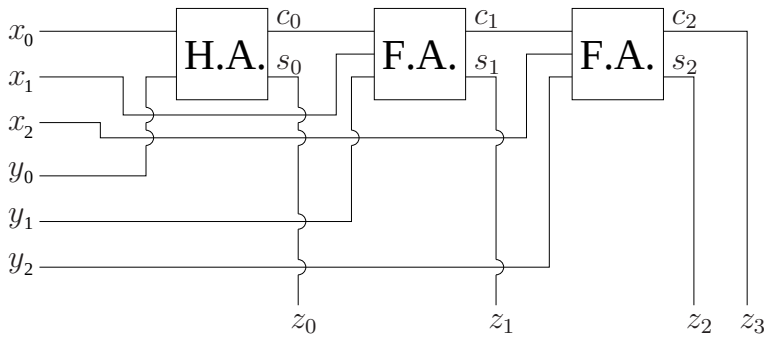
Potpuno zbrajalo označavamo kao na slici 2.6. Kako izrazi c i s ne komutiraju, prilikom crtanja potpunog zbrajala (kao i kod poluzbrajala) **nužno je naglasiti koji je koji** (tj. na svaku izlaznu "žicu" napisati je li ona s ili c).



Slika 2.6: Simbol potpunog zbrajala

Zadatak 2.5.3. *Konstruirajte sklop koji zbraja dva* 3*-bitna binarna broja* $x = (x_2x_1x_0)_2$ *i* $y = (y_2y_1y_0)_2$.

Rješenje. Tablica istinitosti bi imala $2^6 = 64$ retka, što je velik posao. Zbog toga ćemo simulirati zbrajanje "na ruke" (slika 2.7). Prvo zbrajamo zadnje znamenke brojeva: $x_0 +$



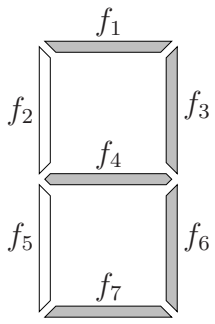
Slika 2.7: Sklop za zbrajanje troznamenkastih binarnih brojeva

y_0. Rezultat tog zbroja daje posljednju znamenku konačnog rezultata z_0 i, eventualno, "jedan dalje" (označimo ga s c_0). U drugom koraku zbrajamo $c_0 + x_1 + y_1$; rezultar će biti znamenka z_1 i, eventualno, "jedan dalje" (označimo ga s c_1). Konačno, zbrajamo $c_1 + x_2 + y_2$; rezultat tog zbrajanja je $(c_2s_2)_2$, što su prve dvije znamenke konačnog rezultata (tj. $c_2 = z_3$, $s_2 = z_2$). □

2.6 Dodatni zadaci

Zadatak 2.6.1. *Konstruirajte logičke sklopove za* 7*-segmentni indikator s* 3*-bitnim ulazom (tj. za brojeve od* 0 *do* 7*).*

Uputa. U ovom zadatku je sadržano sedam istovjetnih zadataka. Svaki broj od 0 do 7 treba nacrtati kao što ga iscrtava kalkulator (pri tome se broj 1 može nacrtati na dva načina; izaberite jedan). Za svaki od brojeva, neke od dioda $f_1 - f_7$ će biti upaljene (stanje 1), a neke će biti ugašene (stanje 0). Potrebno je sastaviti tablicu istinitosti za sve brojeve, te pomoću konjunktivnih ili disjunktivnih normalnih formi dobiti formule za pojedine f_i. Na kraju, te formule treba pojednostaviti i nacrtati pripadne sklopove.



Slika 2.8: 7-segmentni indikator s prikazanim brojem 3

Pazite: tablicu je najlakše popuniti po recima (tj. za svaki $x = (x_2x_1x_0)_2$ popuniti nulama i jedinicama stupce $f_1 - f_7$ ovisno o tome koja dioda svjetli za prikaz broja x), ali normalne forme moramo konstruirati po stupcima (tj. jednu za f_1, jednu za f_2, itd).

Neka od rješenja koje trebate dobiti (uz pretpostavku da smo broj 1 dobili paljenjem f_3 i f_6, a ne f_2 i f_5):

$$\begin{aligned}
f_1 &= (\overline{x_0} + x_1 + x_2) \cdot (x_0 + x_1 + \overline{x_2}) \\
&= \overline{x_0} \cdot x_0 + \overline{x_0} \cdot x_1 + \overline{x_0} \cdot \overline{x_2} + x_1 \cdot x_0 + x_1 \cdot x_1 + x_1 \cdot \overline{x_2} + x_2 \cdot x_0 + \\
&\quad x_2 \cdot x_1 + x_2 \cdot \overline{x_2} \\
&= x_1 \cdot (\overline{x_0} + x_0 + 1 + \overline{x_2} + x_2) + x_0 \cdot x_2 + \overline{x_0} \cdot \overline{x_2} \\
&= x_1 + x_0 \cdot x_2 + \overline{x_0} \cdot \overline{x_2} = x_1 + x_0 \cdot x_2 + \overline{x_0 + x_2}, \\
f_2 &= \overline{x_0} \cdot \overline{x_1} \cdot \overline{x_2} + \overline{x_0} \cdot \overline{x_1} \cdot x_2 + x_0 \cdot \overline{x_1} \cdot x_2 + \overline{x_0} \cdot x_1 \cdot x_2 \\
&= \overline{x_0} \cdot \overline{x_1}(\overline{x_2} + x_2) + x_0 \cdot \overline{x_1} \cdot x_2 + \overline{x_0} \cdot x_1 \cdot x_2 \\
&= \overline{x_1} \cdot (\overline{x_0} + x_0 \cdot x_2) + \overline{x_0} \cdot x_1 \cdot x_2 = \overline{x_1} \cdot (\overline{x_0} + x_2) + \overline{x_0} \cdot x_1 \cdot x_2 \\
&= \overline{x_1} \cdot \overline{x_0} + \overline{x_1} \cdot x_2 + \overline{x_0} \cdot x_1 \cdot x_2 = \overline{x_0} \cdot \overline{x_1} + (\overline{x_1} + \overline{x_0} \cdot x_1) \cdot x_2 \\
&= \overline{x_0} \cdot \overline{x_1} + (\overline{x_1} + \overline{x_0}) \cdot x_2 = \overline{x_0} \cdot \overline{x_1} + \overline{x_1} \cdot x_2 + \overline{x_0} \cdot x_2 \\
&= \overline{x_0 + x_1} + (\overline{x_0} + \overline{x_1}) \cdot x_2 = \overline{x_0 + x_1} + \overline{x_0 \cdot x_1} \cdot x_2.
\end{aligned}$$

□

Zadatak 2.6.2. *Konstruirajte logičke sklopove za* 7*-segmentni indikator s* 4*-bitnim ulazom (tj. za heksadecimalne brojeve od* 0 *do* $(F)_{16}$*).*

Uputa. I ovaj zadatak sadrži 7 podzadataka (tj. trebate složiti 7 sklopova), no ovaj put sklopovi ovise o 4 ulazne varijable, što znači da će tablica istinitosti sadržavati $2^4 = 16$ redaka. □

Zadatak 2.6.3 (Kolokvij iz "Uvoda u računarstvo", 30. 11. 2001)**.** *Napišite konjunktivnu*

i disjunktivnu normalnu formu formule f računa sudova zadane pomoću tablice:

a	b	c	f
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1

a	b	c	f
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Zadatak 2.6.4. *Pojednostavite koliko god je moguće formulu računa sudova:*

$$f = a \cdot \overline{b} + \overline{a \cdot b} + a \cdot b \cdot \overline{c} + \overline{a \cdot c} \cdot b.$$

Rješenje. $f = \overline{a \cdot b \cdot c}$. □

Zadatak 2.6.5 (Pismeni ispit iz "Uvoda u računarstvo", 22. 4. 2002)**.** *Napišite formulu računa sudova koja se sastoji od varijabli x_0, x_1 i x_2. Formula poprima vrijednost* 1 *ako je broj $x = (x_2x_1x_0)_2$ nultočka funkcije*

$$f(x) = x^3 - x^2 - 4x + 4,$$

a 0 *inače. Obavezno pojednostavite formulu!*

Zadatak 2.6.6 (Pismeni ispit iz "Uvoda u računarstvo", 10. 7. 2002)**.** *Formula računa sudova f sadrži varijable x, y i z, te poprima vrijednost* 1 *ako su barem dvije od varijabli isitinite, a* 0 *inače. Napišite f tako da sadrži samo operatore I i NE.*

Zadatak 2.6.7 (Pismeni ispit iz "Uvoda u računarstvo", 18. 6. 2001)**.** *Konstruirajte logički sklop s tri ulaza x, y i z, te izlazom f koji treba biti* 1 *ako je broj $(xyz)_2$ prost, a* 0 *inače. Obavezno minimizirajte sklop!*

Uputa. Zadatak rješavamo klasično, kreiranjem tablice istinitosti. Konačno rješenje je

$$f = \overline{x} \cdot y + x \cdot z.$$

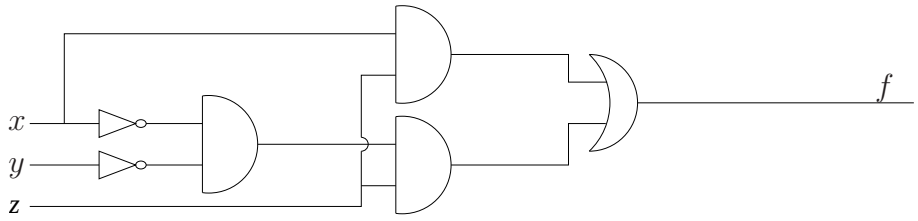
□

Napomena 2.6.1 (Česte greške)**.** *Kod opisnih zadataka u računu sudova, potrebno je paziti kako je uvjet zadan. Na primjer, u prethodnom zadatku treba paziti da je prost broj $(xyz)_2$, a ne $(zyx)_2$.*

Dodatno, poštujte nazive varijabli iz zadatka. Na primjer, ako je zadatrak zadan s varijablama x_0, x_1 i x_2, onda i postupak i rješenje trebaju biti s tim varijablama (a ne, recimo, a, b i c). Ako postoji potreba za izmjenom naziva varijabli, onda treba jasno naznačiti o kakvoj je supstituciji riječ (npr. $x_0 = a$, $x_1 = b$ i sl).

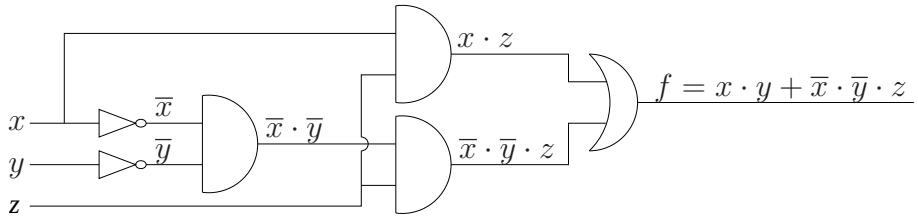
Nije greška, ali je nepotrebno: nemojte rješavati ono što se u zadatku ne traži. Ako se ne traži crtanje sklopa, onda za crtež nećete dobiti bodove, nego ćete samo izgubiti vrijeme. Ako se ne traži pojednostavljivanje formule, ono vam neće donijeti dodatne bodove. Dapače, ako pogriješite, konačni rezultat će biti pogrešan, pa ćete izgubiti ponešto bodova.

Zadatak 2.6.8 (Pismeni ispit iz "Uvoda u računarstvo", 11. 7. 2001)**.** *Logički sklop s tri ulaza x, y, z i izlazom f zadan je pomoću sheme:*



Sastavite ekvivalentan logički sklop koristeći samo jedna NE-vrata, jedna ILI-vrata i jedna I-vrata.

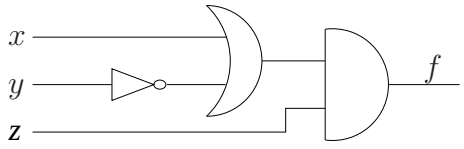
Rješenje. Da bismo riješili zadatak, potrebno je prvo iz sklopa rekonstruirati izraz f, što radimo kao na priloženoj slici:



Sada možemo pojednostaviti izraz, pazeći na ispunjavanje uvjeta zadatka (tj. uz upotrebu po jednog od operatora negacije, disjunkcije i konjunkcije):

$$f = \overline{x} \cdot \overline{y} \cdot z + x \cdot z = (\overline{x} \cdot \overline{y} + x) \cdot z = (x + \overline{y}) \cdot z.$$

Treba još samo nacrtati sklop koji odgovara tako pojednostavljenom izrazu f:



□

Zadatak 2.6.9. *Konstruirajte logički sklop s tri ulaza x_0, x_1 i x_2, te izlazom f koji treba biti* 1 *ako je broj $x = (x_2x_1x_0)_2$ nultočka polinoma*

$$p(x) = x^3 - 3x^2 - 6x + 8,$$

a 0 *inače. Obavezno minimizirajte sklop!*

Zadatak 2.6.10. *Konstruirajte logički sklop s četiri ulaza x_0, x_1, x_2 i x_3, te izlazom f koji treba biti* 1 *ako se broj $x = (x_3x_2x_1x_0)_2$ može prikazati kao suma barem dva uzastopna prirodna broja, a* 0 *inače. Obavezno minimizirajte sklop!*

Poglavlje 3

Konačni automati i regularni izrazi

Pri matematičkom modeliranju računala možemo koristiti razne modele. U ovom poglavlju osvrnut ćemo se na konačne automate i regularne jezike, s naglaskom na regularne izraze koji su često dostupno (i izuzetno korisno!) pomagalo u naprednijim tekst editorima. Konačni automati su posebno korisni za modeliranje računala s vrlo malo memorije (poput automatskih vrata, dizala, mikrovalnih pećnica i sl).

3.1 Konačni automati

Pogledajmo kako bismo opisali ponašanje klima-uređaja specijaliziranog za vinske podrume. Kao što je poznato, idealna temperatura za čuvanje vina je otprilike 12.2°C. Razmotrit ćemo pojednostavljeni model, jer se stvarni kompleksniji modeli ne razlikuju suštinski. Pretpostavljamo da uređaj zna hladiti i grijati, oboje s dvije raspoložive snage ("slabo" i "jako"), te pokušava cijelo vrijeme održavati temperaturu prostorije na željenih 12.2°C. Pri tome ne smije prenaglo mijenjati temperaturu, što znači da je "jače" hlađenje ili zagrijavanje prostora poželjno samo ako "slabije" nije dostatno (dakle, uređaj neće "žuriti" i uvijek raditi što jače).

Uređaj započinje rad u stanju "održavanje", te pomoću senzora u pravilnim vremenskim intervalima očitava temperaturu zraka u prostoriji, te ju uspoređuje sa zadanom vrijednošću. Ako je očitana temperatura niža od željene, uređaj ulazi u stanje "slabo grijanje" u kojem zagrijava prostor. Ako je uređaj već bio u stanju "slabo grijanje", a prošla očitana temperatura je viša od upravo očitane (dakle, prostor se i dalje hladi, što znači da "slabo grijanje" nije dostatno), stroj prelazi u stanje "jakog grijanja". Slično i za visoke temperature i hlađenje. Kad je postignuta idealna temperatura, uređaj će prijeći u stanje "održavanje" u kojem samo prati temperaturu.

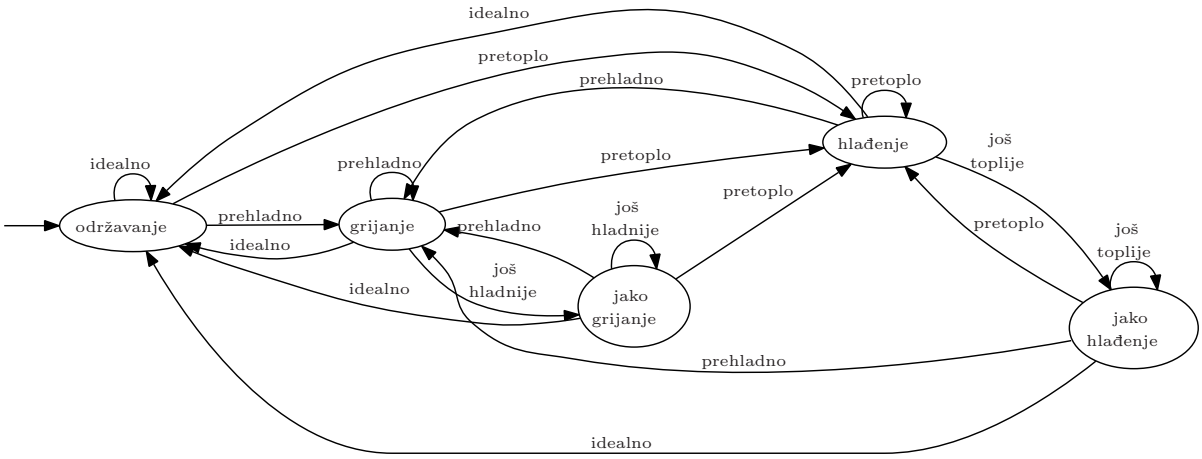
Slično bi radio i ovlaživač zraka koji bi držao vlažnost na, za vinski podrum idealnih 70%.

Prikazani uređaj je jednostavan, no pokazuje što konačni automati rade: u memoriji pamte vrlo ograničenu količinu podataka (u opisanom primjeru je to jedno od pet

ponuđenih stanja), te prima nekakve ulazne podatke (u opisanom primjeru: očitanja senzora). U ovisnosti o tim podacima (i upamćenima i primljenima), uređaj mijenja stanje ili zadržava postojeće. Uz stanja se mogu vezati neke akcije koje će obaviti neki komad hardwarea (u našem primjeru sam klima uređaj kojim upravljamo).

U samom primjeru, uz stanje, pamtimo i prošlo očitanje temperature, no to možemo smatrati i ulaznim podatkom (npr. senzor temperature može pamtiti zadnju očitanu vrijednost, te slati nju i aktualnu vrijednost).

Konačne automate možemo prikazati grafički, no to često ispada nepregledno (v. Sliku 3.1). U takvim prilikama su prikladniji tablični i funkcijski prikaz, no njih ovdje nećemo detaljnije obrađivati.



Slika 3.1: Konačni automat za upravljanje klima-uređajem

Pogledajmo formalnu definiciju matematičkog modela konačnog automata.

Definicija 3.1.1 (Konačni automat)**.** *Konačni automat je uređena petorka* $(Q, \Sigma, \delta, q_0, F)$*, pri čemu je*

- Q *konačan* skup stanja*,*
- Σ *konačan skup koji zovemo* abeceda*,*
- $\delta : \ Q \times \Sigma \to Q$ funkcija prijelaza*,*
- $q_0 \in Q$ početno stanje*, te*
- $F \subseteq Q$ skup konačnih stanja*.*

Iz prikazanog primjera je jasno što je skup stanja ("održavanje", te po dva "grijanja" i "hlađenja"), kao i koje je stanje početno ("održavanje"). Abeceda se na prvi pogled čini velika (skup svih mogućih očitanja temperature), no zapravo je dovoljno da senzori šalju podatke "idealna temperatura", "prehladno", "prehladno i hladnije nego maloprije",

"pretoplo", te "pretoplo i toplije nego prije". Nakon što smo popisali stanja i članove abecede, jasno je kako je definirana funkcija δ. Na primjer,

$$\delta(\text{"hlađenje"}, \text{"pretoplo"}) = \text{"hlađenje"},$$
$$\delta(\text{"hlađenje"}, \text{"pretoplo i toplije nego prije"}) = \text{"jako hlađenje"}.$$

Konačno stanje ovdje nije zadano jer želimo da stroj radi unedogled.

3.2 Regularni izrazi

Kako smo vidjeli u prethodnom poglavlju, osnovna ideja konačnog automata je da "čita" nekakve podatke, te mijenja stanja u kojima se nalazi. Primijetimo da je takav model idealan za prepoznavanje nekakvih riječi. Npr. kad bismo željeli prepoznati sve riječi koje imaju barem jedno slovo "e", te završavaju na "ca", stroj bi pomoću stanja morao pamtiti je li pročitao slovo "e", te – ako je – koja su zadnja dva slova. Stanje u kojem bi "e" bilo pročitano, te bi zadnja dva slova bila "c" i "a", proglasili bismo za završno stanje.

Stroj bi tako čitao zadanu riječ i "preskakao" iz stanja u stanje. Kad "potrošimo" sva slova, provjerili bismo je li stroj u završnom stanju. Ako je, riječ je kakvu tražimo; inače nije.

Skup A svih riječi koje prihvaća konačni automat M zovemo *jezik automata* M, u oznaci $L(M) = A$. Za taj jezik kažemo da ga automat M *prihvaća* ili *prepoznaje*. Kao što smo vidjeli, konačni automat može prihvaćati i pojedine riječi, pa – da bismo izbjegli zabune – za jezike obično kažemo da ih "prepoznaje".

Konačni automat može prihvaćati više riječi, no uvijek prepoznaje samo jedan jezik.

Definicija 3.2.1 (Regularni jezik)**.** *Kažemo da je jezik* regularan *ako postoji konačni automat koji ga prepoznaje.*

Izrazi kojima opisujemo regularne jezike zovu se *regularni izrazi*. Da bismo mogli precizno definirati regularne izraze, potrebne su nam prvo operacije na regularnim jezicima.

Definicija 3.2.2 (Regularne operacije)**.** *Neka su A i B dva regularna jezika. Za njih definiramo sljedeće* regularne operacije*:*

- Unija*:* $A \cup B := \{x : \ x \in A \text{ ili } x \in B\}$,
- Konkatenacija *(spajanje):* $A \circ B := \{xy : \ x \in A \text{ i } y \in B\}$,
- Zvjezdica*:* $A^* := \{x_1 x_2 \dots x_k : \ k \geqslant 0 \text{ i svaki } x_i \in A\}$.

Primijetite da zvjezdica producira beskonačni jezik koji sadrži i praznu riječ! Pomoću regularnih operacija prirodno definiramo regularne izraze.

Definicija 3.2.3 (Regularni izraz)**.** *Kažemo da je R* regularni izraz *ako je R jedno od navedenog:*

- $a \in \Sigma$ *(jedan znak zadanog alfabeta),*
- ε *(prazna riječ, tj. riječ duljine nula),*
- $\emptyset$ *(prazni jezik),*
- $(R_1 \cup R_2)$*, pri čemu su R_1 i R_2 regularni izrazi,*
- $(R_1 \circ R_2)$*, pri čemu su R_1 i R_2 regularni izrazi, ili*
- (R_1^*)*, pri čemu je R_1 regularni izraz.*

Napomena 3.2.1. *Razlikujte ε (prazna riječ, odnosno – u terminima jezika – jezik koji sadrži jednu (praznu) riječ) i $\emptyset$ (prazni jezik, odnosno jezik koji ne sadrži niti jednu riječ)!*

Regularni izrazi samo su dio priče o konačnim automatima i regularnim izrazima. Matematička teorija je puno šira, no u nastavku ćemo se baviti samo regularnim izrazima jer su oni direktno primjenjivi u svakodnevnom korištenju računala. Zainteresirani čitatelji mogu više pronaći u knjizi M. Sipsera, "Introduction to the theory of computation".

Iako je prikazana definicija matematički korektna, te se koristi u znanstvenoj i stručnoj literaturi, odabrani simboli su nespretni za upotrebu na računalu (npr. u tekst editorima i programskim jezicima). Zbog toga koristimo prilagođene oznake (zajedničke većini implementacija regularnih izraza).

Definicija 3.2.4. *Oznake korištene za zapis regularnih izraza:*

- `R|S` $= R \cup S$ *(unija izraza R i S),*
- `RS` $= R \circ S$ *(konkatenacija izraza R i S),*
- `(R)*` $= (R)^*$,
- `(R)+` $= R(R)^*$ *(izraz R ponovljen jednom ili više puta),*
- `(R)?` $= (|R)$ *(prazna riječ ili izraz R),*
- `(R){n}` $\equiv$ *izraz R ponovljen točno n puta,*
- `(R){n,m}` $\equiv$ *izraz R ponovljen barem n, a najviše m puta,*
- `(R){n,}` $=$ `(R){n}(R)*` $\equiv$ *izraz R ponovljen barem n puta,*
- `^` $\equiv$ *početak linije,*
- `$` $\equiv$ *kraj linije,*

- `[znakovi]` ≡ *točno jedan od znakova pobrojanih unutar zagrada,*
- `[^znakovi]` ≡ *točno jedan znak, različit od svih znakova pobrojanih unutar zagrada,*
- `.` ≡ *bilo koji znak,*
- `\n`, $\mathtt{n} \in \mathbb{N}$ ≡ `n`*-ta grupacija (tj. ono čemu odgovara* `n`*-ta zagrada),*
- `\x` ≡ `x` *ako je* `x` *neka od specijalnih oznaka (npr.* `\[` *prihvaća uglatu zagradu, tj.* ***NE predstavlja*** *početak izraza* `[znakovi]`*, dok* `\.` *prihvaća točku, a ne bilo koji znak; slično,* `\\` *prihvaća "naopaku kosu crtu"* `\`*),*
- `\s` ≡ *jedan razmak, tabulator ili neki drugi znak za razmak,*
- `\d` ≡ *jedna znamenka,*
- `\w` ≡ *jedan alfanumerički znak (slovo ili znamenka) ili* `_`*,*
- `\S`, `\D` *i* `\W` ≡ *suprotno od* `\s`, `\d` *i* `\w`*, respektivno,*
- `\b` ≡ *rub riječi, tj. mjesto između* `\w` *i* `\W` *(nije znak!). U nekim verzijama se koriste oznake* `\<` *i* `\>` *za početak, odnosno kraj riječi.*

Oznaka `.` *jako često ne uključuje skok u novi redak, no mi ćemo uključiti i skok u novi redak, jer definicija skoka u novi red također ovisi o dodatnim okolnostima (npr. o operativnom sustavu i programu u kojem koristimo regularni izraz).*

U prethodnoj definiciji dodani su novi simboli, proizašli iz same upotrebe regularnih izraza. Konkretno, u matematičkoj teoriji računalstva regularni izraz provjerava cijele riječi. U praktičnim primjenama, on traži neku riječ (općenito govoreći, "riječ" u kontekstu regularnih izraza može sadržavati i razmake!), obično liniju po liniju teksta. Zbog toga, kad želimo osigurati da se pronađena riječ nalazi na početku ili kraju retka, koristimo znakove `^` i `$`.

Oznaka `\n` nam služi kada se želimo osigurati da se neki dio riječi ponavlja. Na primjer, izraz `([ab])c\1` prihvaća riječi "aca" i "bcb" (ali **NE** i "acb").

Izraze `[znakovi]` i `[^znakovi]` ponekad možemo kraće zapisati korištenjem raspona. Tako `[ag-m]` označava bilo koji znak iz skupa $\{\mathtt{a}, \mathtt{g}, \mathtt{h}, \mathtt{i}, \mathtt{j}, \mathtt{k}, \mathtt{l}, \mathtt{m}\}$.

Važno je naglasiti da kvantifikatori `*`, `+` i `{·}` djeluju samo na prethodni znak ili zagradu. Dakle, izraz `ab+` prihvatit će riječi "ab", "abbbbb" i sl. (ali **NE** i "abab", kao niti "a"). S druge strane, operator `|` djeluje na najšire susjedne podizraze, što znači da je

$$L(\mathtt{a(bc|def)g}) = \{\text{“abcg”}, \text{“adefg”}\},$$

tj. jezik koji izraz `a(bc|def)g` prepoznaje ima točno dvije riječi ("abcg" i "adefg").

Postoji nekoliko standarda za zapis regularnih izraza na računalima. Najrašireniji su POSIX[1] Basic (BRE) i POSIX Extended (ERE). Najnapredniji, no manje rašireni su Perl[2] Compatibile Regular Expression (PCRE). Ovdje smo opisali podskup PCRE-a (zapravo ERE s dodacima koje podržava većina programa s podrškom za regularne izraze).

Neki od programa i jezika koji podržavaju takve ili slične regularne izraze su:

- `(e)grep` – program za traženje teksta u datotekama.

  Besplatna komandnolinijska verzija za Windows nalazi se na http://gnuwin32.sourceforge.net/packages/grep.htm, a postoje i "jače" (ali komercijalne) verzije,

- `(g)vim` – editor teksta.

  Dostupan na http://www.vim.org/download.php,

- `emacs` – editor teksta.

  Dostupan na http://www.gnu.org/software/emacs/,

- PSPad – editor teksta.

  Samo za Windows, na http://www.pspad.com/en/,

- Notepad++ – editor teksta sa skromnom podrškom za regularne izraze.

  Postoji samo za Windows, a možete ga "skinuti" na http://notepad-plus.sourceforge.net/uk/download.php,

- OpenOffice.org – besplatan uredski paket.

  Dostupno na http://download.openoffice.org/.

Uz pobrojane programe, opisane ili slične regularne izraze podržavaju i neki SQL sustavi, te jezici Perl, PHP[3], Python, Ruby, JavaScript, .NET jezici itd., a postoje i dodaci koji regularne izraze dodaju i jezicima koji ih *a priori* ne podržavaju, poput Delphija (v. http://www.regular-expressions.info/delphi.html). Regularni izrazi dostupni su i u GNU C-u, pomoću funkcija u biblioteci `regex`.

Neke implementacije regularnih izraza prepoznaju neke od specijalnih znakova doslovno, a opisana značenja im daju tek ako ispred dodamo `\`. Na primjer, u `gvim`-u, oznaka `+` jednostavno označava znak "plus", dok je kvantifikator za "jedan ili više izraza" specificiran oznakom `\+`.

Točnu sintaksu regularnih izraza provjerite u uputama za program ili jezik u kojem koristite regularne izraze.

[1] Portable Operating System Interface for uniX

[2] Perl je, službeno, akronim od Practical Extraction and Report Language, iako je naziv "Perl" nastao prije. Humoristična verzija L. Walla, autora Perla, kaže da "Perl" znači "Pathologically Eclectic Rubbish Lister".

[3] PHP podržava i ERE i PCRE, preko dva odvojena seta funkcija.

Napomena 3.2.2. *Na ispitu smijete koristiti bilo koji od uobičajenih standarda, no ako koristite pravila različita od ovdje opisanih obavezno napišite koji standard koristite! Također, standarde ne smijete miješati, nego se morate držati jednog standarda!*

Napomena 3.2.3. *Povratne reference (*`\1` *i sl)* ***NISU*** *regularni izrazi u matematičkom smislu (tj. ne opisuju regularne jezike), ali su zbog praktičnosti čest dio implementiranih regularnih izraza u raznim tražilicama.*

Kvantifikatori `*`, `+` i `{·}` rade "pohlepno", što znači da će "pokupiti" najveći mogući dio teksta. Na primjer, izraz `[qrts].*i` će u tekstu "skočimiš" prepoznati riječ "skočimi". Da je riječ bila "skočimš", izraz bi prepoznao "skoči". Ova razlika je bitna kad se regularni izraz koristi za zamjene dijelova teksta. Opisano ponašanje može se u PCRE (ili njihovim dijelovima) zamijeniti tzv. "lijenim" prepoznavanjem.

"Pohlepnost" algoritma "ide" u smjeru slijeva na desno. To znači da će izraz `^a*.` "pokupiti" sve uzastopne znakove "`a`" (podziraz `a*`) na početku linije i prvi znak različit od "`a`" (podizraz `.`). No, ako linija sadrži samo n znakova "`a`" (i ništa više nakon tih `a`-ova) za neki $n > 0$, onda će podizrazu `a*` odgovarati prvih $n - 1$ znakova "`a`", dok će posljednji znak "`a`" odgovarati podizrazu "`.`".

Primjer 3.2.1. *U tekstu je potrebno urediti "višak" razmaka. Kako to napraviti u editoru koji podržava regularne izraze?*

Za početak, potrebno je odrediti što je to "višak razmaka". Ovdje ćemo uzeti da su to svi razmaci na početu i na kraju linija, te oni kojih ima dva ili više za redom.

Da bismo maknuli sve razmake na počecima svih linija, tražimo

```
^\s+
```

te zadajemo editoru da pronađeno zamijeni s praznim stringom. Analogno, za razmake na kraju redaka tražimo

```
\s+$
```

i opet zamjenjujemo s praznim stringom. Ako neka linija nema razmake na početku/kraju, prikazani izrazi ju neće prepoznati, pa se zamjena za njih neće izvršiti, baš kao što bismo i očekivali.

Ako editor podržava povratne reference, ove dvije stvai možemo zamijeniti jednim traženjem

```
^\s*(.*\S)\s*$
```

prilikom kojeg ćemo pronađeni tekst zamijeniti s povratnom referencom `\1`*. Pri tome, opisani izraz neće obrisati linije koje sadrže samo razmake (zbog* `\S`*). Primijetite i da, zbog pohlepnosti kvantifikatora, izraz* `^\s*(.*)\s*$` *ne bi bio dobar, jer nikad ne bi brisao razmake na kraju linije. Ako želimo "počistiti" i takve linije, tražimo*

`^\s*(.*\S|)\s*$`

Dakle, tražimo razmake, zatim niz znakova koji završava ne-razmakom ili prazan niz znakova (u zagradi), te na kraju razmake do kraja linije.

Višestruke razmake (uključivo i tabulatore i sl.) unutar teksta želimo zamijeniti s po jednim običnim razmakom. Dakle, tražimo dva ili više razmaka:

`\s{2,}` *ili* `\s\s+` *(ako editor ne podržava* `{·}`*)*

te pronađeni tekst zamjenjujemo s običnim razmakom (␣). Mogli smo tražiti i samo `\s+`*, no to opterećuje editor s puno nepotrebnih zamjena (zamjena jednog razmaka s tim istim razmakom). Ipak, takva zamjena može biti korisna ako želimo i pojedinačne tabulatore (ili druge "nevidljive" znakove) zamijeniti razmacima.*

Iako razmak možemo označavati kao ␣, općenito je bolje koristiti `\s` (v. prethodni primjer) jer ono što u tekstu izgleda kao razmak može biti tabulator ili nešto slično. Naravno, iznimka su formatirane datoteke kad zaista želimo prepoznavati baš razmake, ali ne i ostale znakove koji samo liče na razmak.

Naravno, korištenje `\s` kao onoga s čime zamjenjujemo pronađeni tekst je besmisleno, jer `\s` znači "razmak ili tabulator ili…", a ne jedan točno određeni znak.

U primjeru smo pokazali jednu praktičnu primjenu regularnih izraza. U zadacima ćemo se baviti samo regularnim izrazima, no ne i time čime ih treba zamjenjivati, jer zamjene ovise o konkretnim primjenama, a lagane su za napraviti, kad znamo ispravno tražiti.

Zadatak 3.2.1. *U datoteci su pohranjeni opisi fotografija i filmova napravljenih digitalnim fotoaparatom. U svakoj liniji datoteke nalazi se opis jedne slike ili filma i to tako da je navedeno ime datoteke (bez ekstenzije), zatim "␣-␣" (razmak, minus i još jedan razmak), te opis. Neke slike i filmovi nemaju opisa.*

Napišite regularni izraz koji prepoznaje sve opisane separatore iza kojih nema opisa (kako biste ih u nekom editoru obrisali, pa da u tim linijama ostanu samo nazivi datoteka).

Rješenje. Osnovno prepoznavanje je lagano: `\s-\s` ili ␣-␣. No, ovakav izraz prepoznat će **SVA** pojavljivanja separatora "␣-␣", dakle i one koji iza sebe imaju opis. Traženi regularni izraz ima samo jedan znak više – onaj za označavanje kraja retka:

`\s-\s$` ili ␣-␣$

Općenito, prvo rješenje je bolje, jer drugo prejudicira da nam tražimo samo razmake, a ne i druge "nevidljive" znakove. □

Zadatak 3.2.2. *Napišite regularni izraz koji prepoznaje sve linije teksta koje sadrže*

1. *barem jedan znak "y",*

2. *točno jedan znak "y".*

Rješenje.

1. Sam pronalazak linije je jednostavan: `y` (regularni izraz koji će pronaći znak "y"). No, ovakav izraz neće prepoznati cijelu liniju nego samo pojedini znak (što će stvoriti neželjene posljedice kod izmjene teksta). Umjesto toga, moramo zadati izraz koji prepoznaje sve znakove od početka do kraja linije:

   ```
   ^.*y.*$
   ```

   Ako u liniji postoji više znakova "y", njih će pokupiti prvi podizraz `.*` (zbog "pohlepnog" traženja).

2. Na osnovu prethodnog podzadatka slažemo ovo rješenje. Pošto želimo točno jedan znak "y" (koji već imamo), moramo se osigurati da niti jedan znak koji podizrazi `.*` prihvate ne bude "y". Dakle:

   ```
   ^[^y]*y[^y]*$
   ```

   □

Zadatak 3.2.3. *Napišite regularni izraz koji prepoznaje sve linije teksta u kojima se neposredno iza svakog znaka "a" nalazi znak "b" ili znak "c".*

Rješenje. Pošto želimo provjeravati cijele linije, izraz moramo omeđiti znakovima `^` i `$`. Često, ali **POGREŠNO** rješenje kod ovakvih zadataka je

```
^.*(a[bc].*)*$
```

Problem ovdje je da točka odgovara bilo kojem znaku, pa tako i znaku "a". Drugim riječima, podizraz `.*` može odgovarati cijeloj liniji (bilo kojoj!), pa `(a[bc].*)*` jednostavno može biti prazna riječ. Ispravno je osigurati ne samo gdje će se "a" pojaviti (ispred "b" ili "c"), nego i gdje neće:

```
^[^a]*(a[bc][^a]*)*$
```

□

Zadatak 3.2.4. *Napišite regularni izraz koji u tekstu prepoznaje*

1. *pozitivne binarne brojeve,*
2. *ispravno zapisane pozitivne binarne brojeve (bez vodećih nula),*
3. *ispravno zapisane binarne brojeve (bez vodećih nula, sa ili bez negativnog predznaka),*
4. *ispravno zapisane binarne brojeve s točno* 3 *znamenke,*
5. *ispravno zapisane binarne brojeve s barem* 3 *znamenke,*

6. ispravno zapisane binarne brojeve s barem 3 *i najviše* 5 *znamenaka.*

Rješenje.

1. Pozitivni binarni brojevi su nizovi nula i jedinica, pri čemu čitani niz znakova mora sadržavati barem jedan znak (tj. prazna riječ nije broj), pa je traženi izraz, "na prvu loptu",

   ```
   [01]+
   ```

   No, ovo nije u potpunosti točno, jer će kao pozitivni broj prepoznati i text koji sadrži samo nule (npr. "`0`", "`0000`" i sl.). Nama treba da barem jedna znamenka (za sada ne nužno vodeća, jer još ne inzistiramo na "ispravnom" zapisu) bude različita od nule (tj. 1).

   ```
   [01]*1[01]*
   ```

   Primijetite da smo kvantifikator `+` zamijenili kvantifikatorom `*`. Naime, `+` nam je trebao osigurati da imamo barem jednu znamenku. Ovdje smo tu znamenku izdvojili ("`1`" u sredini), pa lijevo i desno od nje smiju biti i prazne riječi.

2. Ako želimo osigurati da nema vodećih nula, prepoznata riječ mora **započeti** sa znamenkom koja nije nula. Dakle, traženi izraz sličan je prethodnom, s time da znak "`1`" mora biti na početku:

   ```
   1[01]*
   ```

   Primijetite: izraz `1[01]+` bi odgovarao pozitivnim binarnim brojevima strogo većim od 1 (počinju sa znamenkom 1 i imaju još barem jednu znamenku iza).

3. Da bismo prepoznali minus koji se može pojaviti najviše jednom (ali se i ne mora pojaviti), upotrijebit ćemo kvantifikator `?`. Dodatno, više ne ograničavamo brojeve na pozitivne, pa treba prepoznati i broj 0 (jedini binarni broj koji ne počinje znamenkom 1). Dakle, traženi izraz je

   ```
   -?1[01]*|0
   ```

   Da smo kao ispravan broj željeli prihvatiti i -0, izraz bi glasio

   ```
   -?(1[01]*|0)
   ```

4. Ovdje kvantifikator `*` zamjenjujemo kvantifikatorom `{·}`. Pri tome treba uzeti u obzir da je jedna znamenka ona prva jedinica, pa ostatak broja treba imati dvije (a ne tri) znamenke. Konačno, imamo izraz

```
-?1[01]{2}
```

5. Ako kvantifikatoru `{·}` dodamo zarez, mijenjamo mu značenje iz "točno koliko piše" u "barem koliko piše". Novi izraz je

```
-?1[01]{2,}
```

6. Ako kvantifikator `{·}` zamijenimo s `{·,·}`, mijenjamo mu značenje iz "točno koliko piše" u "barem koliko je prvi parametar i najviše koliko je drugi" (opet smanjujemo oba broja za jedan jer jednu znamenku provjeravamo prije `[·]`). Novi izraz je

```
-?1[01]{2,4}
```

□

U rješenjima prethodnog zadatka prepoznajemo binarne brojeve bez obzira na to što se nalazi oko njih. Tako će, na primjer, u tekstu "`Pero101x`" podstring "`101`" biti prepoznat kao binarni broj, iako je samo dio riječi (npr. nečijeg korisničkog imena). Kako prepoznati samo brojeve odvojene od teksta, ovisi donekle o tome što točno želimo. Najčešći način vidjet ćemo kasnije, npr. u rješenjima Zadatka 3.2.7.

Kvantifikator `{·}` nije moguće koristiti u varijanti `{,·}`, tj. prva vrijednost uvijek mora biti zadana. To nije problem: jednostavno, ako želimo reći "najviše n ponavljanja, umjesto nepostojećeg `{,`n`}` koristimo `{0,`n`}`.

Zadatak 3.2.5. *Riješite prethodni zadatak (svih šest varijanti) prepoznavajući*

a) oktalne brojeve,

b) heksadecimalne brojeve i

c) parne dekadske brojeve.

Uputa. Parni brojevi su oni koji završavaju znamenkom 0, 2, 4, 6 ili 8. Tu znamenku izdvojite, no pripazite da ispravno prepoznate i jednoznamenkaste brojeve (jer je prva znamenka već izdvojena). □

Zadatak 3.2.6. *Napišite regularni izraz koji prepoznaje ispravno napisane e-mail adrese.*

Rješenje. Osnovno na što ovdje želimo paziti je da niz znakova mora sadržavati točno jedan znak "`@`", te barem jednu točku:

```
[^@]+@[^@.]+(\.[^@.]+)+
```

Točan izraz, koji uzima u obzir i koji su znakovi gdje dozvoljeni, je

```
[a-z0-9._%-]+@([a-z0-9-]+\.)+[a-z]{2,4}
```

no to nije potrebno znati na ispitu. Ipak, primijetite u podizrazu `[a-z0-9-]` završni minus: on označava znak "minus", a ne raspon (jer nema znak do kojeg bi raspon trebao ići). □

Zadatak 3.2.7. *Napišite regularni izraz koji prepoznaje*

1. *bilo kakvu riječ (u klasičnom smislu, dakle niz slova omeđenih razmacima),*
2. *nenaglašenu riječ (prvo slovo može biti malo ili veliko; ostala moraju biti mala),*

Prvo prikazujemo rješenje u kojem ćemo tražiti odgovarajući raspored razmaka u okolini traženih riječi.

Rješenje pomoću razmaka. Bilo koje slovo prepoznajemo podizrazom `[a-zA-Z]`. Bilo koje malo slovo prepoznajemo podizrazom `[a-z]`, dok veliko slovo prepoznajemo podizrazom `[A-Z]`. U svim podzadacima moramo prepoznati "višak", tj. bližu okolinu riječi. Zato ćemo samo riječ dodatno grupirati u zagrade, a uz izraz treba naglasiti i koja grupa zagrada daje samu riječ (jer nju zadatak traži).

1. Riječ omeđena razmacima:

   ```
   \s([A-Za-z]+)\s
   ```

   Riječ odgovara prvoj grupaciji (zagradi).

   Ovdje treba uzeti u obzir da se riječ može nalaziti i na početku ili na kraju retka u tekstu, kad ispred ili iza nje nema razmaka. Pošto želimo i takve riječi prepoznati, pišemo

   ```
   (^|\s)([A-Za-z]+)(\s|$)
   ```

   Dakle, ispred riječi može biti početak retka (ovo nije znak!) ili razmak, a nakon riječi može biti razmak ili kraj retka (koji također nije znak).

   Riječ odgovara drugoj grupaciji (zagradi), dok prva i treća služe za ispravno grupiranje podizraza oko operatora `|`.

2. Slično kao kod prepoznavanja brojeva, treba izdvojiti prvo slovo od ostalih:

   ```
   (^|\s)([A-Za-z][a-z]*)(\s|$)
   ```

   Riječ koju tražimo odgovara drugoj grupaciji (zagradi).

Da bismo u potpunosti ispravno prepoznavali početak rečenice, tekst je potrebno tretirati kao cjelinu (a ne redak po redak). To se radi ovisno o programu/jeziku, pa ovdje nećemo razmatrati. □

Ovakvo rješenje se **NEĆE** ponašati kako bismo željeli. Naime ako imamo tekst

`Kupus␣sadi␣ciklu`

onda će opisani regularni izrazi prepoznati svaku drugu riječ, jer kad prepoznamo "`Kupus␣`", prepoznavanje iduće riječi kreće nakon već prepoznatog niza znakova, tj. od slova "s" u "sadi". Zbog toga prikazani regularni izraz neće prepoznati da ispred riječi "sadi" postoji razmak. Da bismo izbjegli takve probleme, koristimo rub riječi `\b` (ili, u nekim izvedbama, poput `(g)vim`-a, `\<` i `\>`).

Rješenje pomoću ruba riječi.

1. Riječ:

   `\b[A-Za-z]+\b` ili `\<[A-Za-z]+\>`

   Pošto `\b` nije znak, onda će prikazani regularni izraz prihvatiti točno jednu riječ, što je upravo ono što želimo. Zato nam ne treba grupiranje, kao niti posebna provjera jesmo li na početku ili kraju retka.

2. Opet ćemo izdvojiti prvo slovo od ostalih:

   `\b[A-Za-z][a-z]*\b` ili `\<[A-Za-z][a-z]*\>` □

Prikazano rješenje (pomoću ruba riječi) ispravno će prepoznati i riječi koje nisu omeđene samo razmacima, nego i interpunkcijskim i sličnim znakovima.

Napomena 3.2.4. *Ako izraz prepoznaje "višak", obavezno naglasite koja grupacija (ili zagrada) odgovara rješenju zadatka. Ako takvog naglašavanja nema, smatra se da cijeli izraz odgovara onome što se u zadatku traži.*

Zadatak 3.2.8. *Napišite regularni izraz koji prepoznaje riječi koje sadrže nehrvatska slova engleskog alfabeta ("q", "w", "x", "y").*

Zadatak 3.2.9. *Napišite regularni izraz koji prepoznaje sve jednostavne LaTeX naredbe oblika*

`\naredba{glavni parametar}` *ili*
`\naredba[pomocni parametar]{glavni parametar}`

pri čemu treba odvojiti naredbu i parametre (u tri odvojene grupacije). Možete pretpostaviti da se `naredba` *sastoji samo od slova (barem jednog), dok parametri u sebi ne sadrže uglate, odnosno vitičaste zagrade, ali mogu biti i prazni.*

Rješenje. Ovdje koristimo nekoliko specijalnih znakova (\, [,], {, }) ispred kojih treba dodati \ kako bismo prihvaćali upravo njih, bez specijalnih značenja koja inače imaju u regularnim izrazima. Također, podizraz koji prihvaća [pomocni parametar] treba označiti kao nešto što se može i ne mora pojaviti. To radimo dodatnim grupiranjem, uz kvantifikator ?.

```
\\([a-zA-Z]+)(\[([^]]*)\])?\{([^}]*)\}
```

Naredba će se nalaziti u \1, pomoćni parametar (ako postoji) bit će u \3 (ako ne postoji, \3 će biti prazan), dok će se glavni parametar naći u \4. □

Zadatak 3.2.10. *Napišite regularni izraz koji prepoznaje sve složenije LaTeX naredbe oblika*

\naredba{gp1}{gp2}...{gpn} *ili* \naredba[pp]{gp1}{gp2}...{gpn}

pri čemu treba odvojiti naredbu i parametre slično prethodnom zadatku. Možete pretpostaviti da postoji barem jedan glavni parametar, tj. $n \geq 1$ *te da niti jedan od njih ne sadrži vitičaste zagrade.*

*Pokušajte u jednu grupaciju (napišite i koju!) "uhvatiti" sve glavne parametre (*gp1*,* gp2*,..., *gpn*), zajedno s pripadnim vitičastim zagradama.*

Rješenje.

```
\\([a-zA-Z]+)(\[([^]]*)\])?(\{([^}]*)\})+
```

Sve glavne parametre ćemo "uloviti" dodatnim grupiranjem:

```
\\([a-zA-Z]+)(\[([^]]*)\])?((\{([^}]*)\})+)
```

Glavni parametri bit će pospremljeni u \4. □

Što će se nalaziti u \6 ako je tekst koji pretražujemo

```
Jedna \slozena{La}{TeX} naredba
```

pošto će ta grupacija prepoznati i "La" i "TeX"? Što će se nalaziti u ostalim grupacijama (\1,..., \5)?

Napomena 3.2.5. *U prethodnim zadacima vidimo relativno zbunjujući podizraz* [^]]. *Kad bi prvi znak*] *zatvaro izraz, tj. kad bismo pod izraz gledali kao* [^] *i*] *posebno, onda bi* [^] *značilo "bilo koji znak osim ničega", što je besmisleno (postiže se točkom). Zato, kad želimo prihvatiti neki znak koji ne uključuje zatvorenu uglatu zagradu (i, eventualno, još neke znakove), nju postavljamo na prvo mjesto:*

```
[^]...]
```

Slično, ako želimo prihvatiti neki znak koji može biti zatvorena uglata zagrada (i, eventualno, još neki znak):

`[]...]`

Ako želimo prihvatiti samo `^`*, pišemo*

`\^`

Ako želimo prihvatiti `^` *i još neke znakove, npr.* `a` *ili* `b`*, znak* `^` *ne smijemo staviti na početak (da ne ispadne negacija), pa pišemo*

`[a^b]` *ili* `[ab^]` *ili* `[a-b^]`

Zadatak 3.2.11. *Napišite regularni izraz koji prepoznaje sve palindrome (riječi koje se jednako čitaju slijeva i sdesna) od točno* 5 *slova.*

Rješenje. Ovdje nam trebaju povratne reference, te rubovi riječi:

```
\b([A-Za-z])([A-Za-z])[A-Za-z]\2\1\b
```

□

Zadatak 3.2.12. *Napišite regularni izraz koji prepoznaje brojeve telefona u formatu*

`+kod1-kod2-broj`

gdje `kod1` *označava kôd države,* `kod2` *označava kôd grada/regije, a broj je lokalni broj telefona. Kodovi države i grada/regije sadrže između jedne i tri znamenke, dok se lokalni broj telefona sastoji od znamenaka (barem tri) i najviše jedne crtice (koja ne smije biti na početku i kraju broja).*

Kôdove i lokalni broj izdvojite u zasebne grupacije (i, naravno, napišite što je u kojoj).

Kako biste prepoznali linije koje se sastoje samo od takvih brojeva (po jedan u svakoj liniji), tj. koje sadrže takve brojeve i ništa više?

Rješenje. Ovdje imamo znak `+` koji ima specijalno značenje, pa ispred njega dodajemo `\`:

```
\b\+(\d{1,3})-(\d{1,3})-(\d{2,}-?\d+|\d+-?\d{2,})\b
```

Kôdu države odgovara `\1`, kôdu regije `\2`, a lokalnom broju telefona `\3`. Kad bismo htjeli prepoznavati linije (kako je opisano na kraju zadatka), izraz bi bio:

```
^\+(\d{1,3})-(\d{1,3})-(\d{2,}-?\d+|\d+-?\d{2,})$
```

Inače, na ispitu bi se prihvatilo i rješenje

```
\b\+(\d{1,3})-(\d{1,3})-(\d+-?\d{2,}|\d{2,}-?\d+)\b,
```

iako se ono ne bi ponašalo kako bismo očekivali. Naime, u stringu "`+123-45-678-9`" taj bi izraz prepoznao samo "`+123-45-678`" (zašto?). □

Zadatak 3.2.13. *Napišite regularni izraz koji prepoznaje datume u formatu*

1. `dd.mm.yyyy.`
2. `dd. mjesec yyyy.`
3. `yyyy/mm/dd`

pri čemu je `dd` *dan,* `mm` *mjesec, a* `yyyy` *godina (prirodni brojevi s najviše dvije, odnosno četiri znamenke). "*`mjesec`*" označava tekstualni naziv mjeseca. Ne morate paziti koliko koji mjesec ima dana, no izbjegnite očito nebulozne datume (*`dd` > 31*,* `mm` > 12*, više od jedne riječi u nazivu mjeseca).*

Dobivene dijelove datuma (dan, mjesec i godinu) izdvojite u grupacije (i, naravno, napišite što je u kojoj).

Zadatak 3.2.14. *Napišite regularni izraz koji prepoznaje linije s ispravnim zapisom vremena u* `.srt` *datotekama (SubRip Text, za pohranu titlova). Traženi format je*

`vrijeme1␣-->␣vrijeme2`

pri čemu je svako od vremena zapisano kao

`H:M:S,ms`

gdje `H` *označava sate,* `M` *minute,* `S` *sekunde, a* `ms` *milisekunde (nenegativni cijeli brojevi, sa ili bez vodećih nula, s tim da su milisekudne uvijek zapisane kao troznamenkasti broj). Pazite da minute i sekunde ograničite na dozvoljene vrijednosti.*

Rješenje. Prepoznajmo najprije jedno vrijeme:

```
\d+:[0-5]?\d:[0-5]?\d,\d{3}
```

Sada ovaj izraz treba zapisati dva puta, te između postaviti separator "␣-->␣". Usput, treba pripaziti i na oznake za početak i kraj linije teksta:

```
^\d+:[0-5]?\d:[0-5]?\d,\d{3}␣-->␣\d+:[0-5]?\d:[0-5]?\d,\d{3}$
```

Grupiranjem bismo mogli dobiti pojedine dijelove iz prvog ili drugog vremena (npr. sate i minute). □

Napomena 3.2.6. *U prethodnom zadatku sljedeće rješenje bilo bi* ***KRIVO****:*

```
^(\d+:[0-5]?\d:[0-5]?\d,\d{3})␣-->␣\1$
```

jer bi odgovaralo isključivo linijama kod kojih su oba zadana vremena potpuno jednaka.

Zadatak 3.2.15. *Napišite regularni izraz koji prepoznaje ispravne linije* `.sub` *datoteka (MicroDVD, za pohranu titlova). Svaka linija datoteke sadržava točno jedan titl čiji je format*

```
{frame1}{frame2}Tekst
```

Parametri `frame1` *i* `frame2` *su nenegativni cijeli brojevi, a* `Tekst` *može biti bilo što. Izdvojite te tri cjeline u zasebne grupacije.*

Zadatak 3.2.16. *Napišite regularni izraz koji prepoznaje sve linije u tekstu koje započinju validnim JMBAGom (prirodni deseteroznamenkasti broj) iza kojeg se mogu (ali i ne moraju) nalaziti podaci o vlasniku JMBAGa (ako se nalaze, odvojeni su barem jednim razmakom). JMBAG izdvojite u jednu grupaciju, a podatke (bez početnih razmaka) u drugu.*

Zadatak 3.2.17. *Napišite regularni izraz koji prepoznaje sve linije u tekstu koje započinju validnim JMBGom (prirodni broj koji započinje s datumom (format* `ddmmyyy`*: dvoznamenkasti dan i mjesec i troznamenkasta godina), te ima ukupno* 13 *znamenaka) iza kojeg se mogu (ali i ne moraju) nalaziti podaci o vlasniku JMBGa (ako se nalaze, odvojeni su barem jednim razmakom). JMBG izdvojite u jednu grupaciju, a podatke (bez početnih razmaka) u drugu.*

Zadatak 3.2.18. *Napišite regularni izraz koji prepoznaje rimske brojeve u originalnoj notaciji (dozvoljeno je samo "zbrajanje", tj. IIII, a ne IV, je* 4*). Koristite oznake I (jedan), V (pet), X (deset), L (pedeset), C (sto), D (petsto) i M (tisuću). Pazite da ne dozvolite prazne riječi, te da ne prepoznate dijelove većih riječi (npr. DVI u "PRE**DVI**DJETI").*

Zadatak 3.2.19. *Napišite regularni izraz koji prepoznaje rečenice s najmanje 3 riječi.*

Riječi su nizovi slova odvojeni razmacima, a rečenice završavaju točkom, upitnikom ili uskličnikom i ne prelamaju se kroz retke.

Rješenje.

```
[A-Za-z]+(\s+[A-Za-z]+){2,}\.
```

□

U prethodnom zadatku, varijante s

1. točno 3 riječi i
2. najviše 3 riječi

ne možemo jednostavno prepoznati. Zašto?

Zadatak 3.2.20. *Zadan je tekst*

Alan Mathison Turing (1912–1954), nositelj ordena Reda britanskog carstva i počasni član Kraljevskog društva znanosti, bio je engleski matematičar, logičar, kriptoanalitičar i računalni znanstvenik. Izvršio je veliki utjecaj na razvoj računalne znanosti i pridonio je formalizaciji koncepta algoritma i izračunavanja s tzv. Turingovim strojem. Godine 1999. Time Magazine nazvao je Turinga jednim od 100 najvažnijih ljudi dvadesetog stoljeća za ulogu koju je odigrao u stvaranju modernih računala – njegov Turingov test bio je značajan i karakteristično provokativan doprinos raspravi o umjetnoj inteligenciji.

Tijekom Drugog svjetskog rata Turing je radio za britansku obavještajnu agenciju odgovornu za razbijanje obavještajnih signala i osiguranje podataka britanskoj Vladi i oružanim snagama. Neko je vrijeme bio i šef Odjeljka 8 koji je bio odgovoran za analiziranje njemačkih mornaričkih kriptograma i glavni sudionik u naporima za razbijanje njemačkih šifri. Osmislio je brojne tehnike za razbijanje njemačkih šifri, uključujući elektromehanički stroj koji bi mogao pronaći postavke Enigma stroja. U prosincu 1940. Turing je riješio pomorski Enigma sustav, koji je bio daleko složeniji matematički sustav od svih korištenih od strane ostalih službi. Također je izumio statističke tehnike za pomoć u razbijanju Enigma koda. U srpnju 1942. izmislio je tehniku za korištenje protiv nove njemačke, Lorenzove šifre.

Koji će sve dijelovi teksta odgovarati sljedećim regularnim izrazima, te koje će vrijednosti vratiti njihove grupacije?

1. `[A-Z]\w+`
2. `[a-z]\w+`
3. `\d{4}`
4. `\d{3}`
5. `\d{3,}`
6. `[^A-Za-z␣]+`
7. `\w+␣\d`
8. `U\s(.+u)`
9. `(.)(.\1){2,}`
10. `T((.)*)ing`
11. `^(\D+)(\d+)`
12. `[aeiou]{2,}`

13. `\w+-$`

Rješenje.

1. “Alan”, “Mathison”, “Turing”, “Reda”, “Kraljevskog”,...
2. “lan”, “athison”, “uring”, “nositelj”, “ordena”,...
3. “1912”, “1954”, “1999”, “1940” i “1942”
4. “191”, “195”, “199”, “100”, “194” i “194”
5. “1912”, “1954”, “1999”, “100”, “1940” i “1942”
6. “(1912–1954),”, “-” (prekid riječi na kraju prve linije), “,”,...
7. “ne 1”, “od 1”, “Odjeljka 8”,...
8. “U prosincu 1940. Tu” (**NE** samo “U prosincu”, zbog pohlepnog principa rada algoritma), “U srpnju 1942. izmislio je tehniku” (**NE** samo “U srpnju”, zbog pohlepnog principa rada algoritma)

   U oba slučaja, `\1` će biti sve osim prva dva znaka (početno “U” i razmak)
9. “ačaja” (`\1` = “a”, `\2` = “ja”), “ataka” (`\1` = “a”, `\2` = “ka”), “anagama” (`\1` = “a”, `\2` = “ma”), “a␣ana” (`\1` = “a”, `\2` = “na”), “učuju” (`\1` = “u”, `\2` = “ju”)

Preostale riješite samostalno za vježbu, računajući da je svaka linija teksta zasebna upravo onako kako su prelomljene. □

Primjere rješenja iz ovog poglavlja, kao i druge regularne izraze pisane po opisanom standardu, možete isprobati na službenoj stranici kolegija, preciznije ovdje:
http://degiorgi.math.hr/prog1/apps/regex.php.

Poglavlje 4

Uvod u algoritme i osnovni program

U prethodnim poglavljima vidjeli smo različite upute kako nešto napraviti. Na primjer, kako se u raznim brojevnim sustavima zbraja, oduzima, množi i dijeli, kako od tablice istinitosti složiti konjunktivnu i disjunktivnu normalnu formu i sl.

Opisani procesi su primjeri algoritama. Precizna definicija pojma "algoritam" je komplicirana, ali opisno možemo reći da je algoritam postupak za rješavanje nekog problema.

Neke važne karakteristike algoritama su:

1. Jednim algoritmom možemo riješiti **samo točno određenu klasu problema**. Na primjer, algoritam za zbrajanje u bazi b treba ispravno računati sumu dva broja zapisana u bazi b, ali nije određeno što mora raditi ako brojevi nisu dobro zapisani (npr. neki ima znamenku veću ili jednaku b).

   To znači da se možete osloniti na to da će ulazni podaci biti onakvi kakvi su navedeni u zadatku. Na primjer, ako zadatak kaže da treba učitati prirodan broj, niste dužni provjeravati je li učitani broj zaista veći od nule (osim ako se takva provjera eksplicitno traži u zadatku).

2. Algoritam **uzima ulazne i vraća izlazne podatke** koji predstavljaju rješenje problema. To znači da programi moraju ispisati rezultat, a potprogrami ga moraju ili ispisati ili nekako vratiti glavnom programu (obično će u zadatku biti precizirano što treba).

   Ponekad će u zadatku biti navedeno da treba napisati samo dio programa koji nešto radi; tada nije potrebno učitavati ulazne podatke i ispisivati rezultat (osim ako se u zadatku to eksplicitno traži).

3. Algoritam završava nakon **konačno mnogo koraka**, tj. ne smije se izvršavati u nedogled!

4. Svaki korak mora biti **precizno i jednoznačno opisan**, tako da ga može izvršiti i računalo.

Odakle potječe riječ "algoritam"? Arapi su skupljali i prevodili klasična grčka djela, te znanost iz Indije i Kine. Muammad ibn Mūsā al-Khwārizmī napisao je knjigu o "indijskim brojevima" (precizni sustav sa znamenkama $0, 1, \ldots, 9$), te je u njoj opisao postupke zbrajanja, oduzimanja, množenja, dijeljenja i vađenja korijena. Danas postoji sačuvan latinski prijevod iz dvanaestog stoljeća pod naslovom "Algoritmi de numero Indorum", pri čemu je riječ "Algoritmi" bila predstavljala latinizirano ime autora. Kasnije je to misinterpretirano kao množina na latinskom, pa je riječ "algoritam" preuzeta kao "postupak za računanje".

Najpoznatije al-Khwārizmījevo djelo je "al-Kitāb al-mukhtaar fī isāb al-jabr wa-l-muqābala" u kojem je opisao postupke rješavanja jednadžbi prvog i drugog reda. Prema tom djelu, od riječi "al-jabr" (uspostaviti, dovršiti) potječe riječ "algebra".

U starom značenju, "algoritmi" su samo postupci za računanje, dok je moderno značenje bitno šire. Pojam algoritma centralni je pojam računarstva, a postaje i sve važniji u matematici.

4.1 Tko "programira"?

Danas je uvriježena definicija da su programeri "oni koji programiraju", no što to znači? U osnovi, "one koji programiraju" dijelimo u dvije grupe:

1. **projektanti** – osobe koje osmišljavaju algoritme i strukture podataka kojima se rješava neki problem te

2. **programeri** – osobe koje pišu programski kôd u određenom programskom jeziku, prema uputama koje daju projektanti.

Kod manjih projekata, to dvoje je često isto. Međutim, kod zahtjevnijih projekata, ta razlika je jako bitna. Naime, za realizaciju velikih projekata često je potrebno mnogo "onih koji programiraju". Bilo bi izuzetno loše za konačno rješenje kad bi svatko to radio na svoj način. Zbog toga je potreban manji tim projektanata koji će osmisliti rješenje i dati upute programerima.

Ovdje (na računarskim kolegijima na PMF-MO) prvenstveno obrazujemo projektante, dakle osobe sposobne "algoritamski razmišljati". Pri tome, nužno je poznavanje nekog programskog jezika (zbog provjere rješenja i poznavanja mogućnosti računala), ali se ne stavlja naglasak na apsolutno poznavanje sintakse odabranog jezika (u našem slučaju C-a), nego na poznavanje principa rada i konstrukcija jezika u onoj mjeri koja je nužna za zapisivanje algoritama koje ćemo raditi.

Gornja podjela na projektante i programere je simplificirana. Veliki projekti rade se u više razvojnih faza, a glavnu ulogu u tom procesu imaju software inženjeri – vrlo slično onome što ovdje zovemo projektantima. Više o tom dijelu posla uči se na kolegiju "Softversko inženjerstvo", a riječ je o izuzetno dobro plaćenim stručnjacima. Dakle, isplati

se učiti kolegije "Programiranje 1" i "Programiranje 2", kao uvertiru u ono što slijedi u nastavku studija. ☺

4.2 Osnovni program

Pod pojmom "osnovni program" u svakom programskom jeziku podrazumijevamo program koji ispisuje neku jednostavnu poruku i završava s radom. Svrha takvog programa je demonstracija osnovne sintakse: zaglavlja, način navođenja naredbi i sl.

Najčešće se ispisuje poruka "hello, world", pa se i takvi programi često zovu "hello world" programi. Veliku arhivu takvih programa u raznim jezicima možete vidjeti ovdje: http://www2.latech.edu/~acm/HelloWorld.shtml

U C-u, takav program izgleda ovako:

Primjer 4.2.1 ("Hello world" program pisan u C-u)**.**

```
1 #include <stdio.h>
2
3 int main(void) {
4   printf("hello, world\n");
5   return 0;
6 }
```

Lako se vidi koja naredba ispisuje tekst, ali vidimo da je (čak i za ovako jednostavan program!) potrebno napisati i neke "dodatke".

Osnovni program u C-u sastoji se od:

1. `#include <stdio.h>`
   Direktiva compileru da uključi u program sadržaj datoteke `stdio.h`. Ta datoteka sadrži sve što je potrebno za ispravan ispis teksta na standardni izlaz (najčešće ekran računala).

2. `int main(void)`
   Zaglavlje glavnog dijela programa. Kako ćemo kasnije vidjeti, `main()` je običan potprogram koji se od ostalih potprograma razlikuje samo po tome što upravo njega računalo izvodi prilikom pokretanja programa.
   Kad negdje navedemo programski kod bez funkcije `main()`, to znači da prikazujemo samo dio programa! Za uspješno compiliranje i izvođenje takvog koda, potrebno je dopisati elemente koji nedostaju!

3. `printf("hello, world\n");`
   Sama naredba za ispis. Tekst koji će se ispisati navodimo između dvostrukih navodnika. Često se dodaju i druge vrijednosti, što ćemo također vidjeti malo kasnije.
   Specijalni znak `\n` (engl. *backslash n*) označava skok u novi red.

4. `return 0;`
   Povratna vrijednost funkcije; kod funkcije `main()` označava vrijednost koju program vraća operacijskom sustavu. Ako je program ispravno obavio posao, potrebno je vratiti nulu; inače se navodi neka druga (cjelobrojna) vrijednost.

5. `{}`
   Vitičaste zagrade služe za grupiranje pojedinih cjelina programa. Ovdje je to upotrijebljeno za grupiranje dvije naredbe u funkciju `main()`.

U C-u se ne navode brojevi linija (kao, na primjer, u BASICu). Ovdje te brojeve navodimo samo za lakše snalaženje u kodu.

Program iz primjera 4.2.1 je ispravno napisani program u jeziku C koji će ispisati željenu poruku. Većina današnjih C compilera će prihvatiti i jednostavnije verzije, poput

```
1 main()␣{
2 ␣␣printf("hello,␣world\n");
3 }
```

Ipak, mi ćemo pisati korektne programe, jer prilikom "skraćivanja" kompliciranijih programa može doći do problema: neobična ponašanja programa ili nemogućnost prevođenja na nekom računalu ili compileru.

4.3 Ispis

U prethodnom poglavlju vidjeli smo osnovni program koji prilikom svakog izvršavanja ispisuje isti tekst. Nakon izvršavanja naredbe `printf()`, kursor ostaje točno tamo gdje je ispis stao. Idući poziv naredbe `printf()` će nastaviti pisati upravo na tom mjestu. Zbog toga, isti ispis kao u primjeru 4.2.1 daje i sljedeći program:

```
1 #include␣<stdio.h>
2
3 int␣main(void)␣{
4 ␣␣printf("hell");
5 ␣␣printf("o,");
6 ␣␣printf("␣world");
7 ␣␣printf("\n");
8 ␣␣return␣0;
9 }
```

Bilo kakvo drugačije cjepkanje niza znakova "`hello, world`" bi polučilo isti rezultat. Primijetimo da je potrebno ispisati sve znakove (tj. compiler neće ništa sam dodati). Zbog toga smo treći `printf()` započeli razmakom jer je to prvi znak koji želimo ispisati iza zareza (njega ispisuje drugi `printf()`).

Ispisati možemo i brojčane vrijednosti, jednostavnim uvrštavanjem u niz znakova:

```
1  ␣␣printf("Prog1␣cu␣poloziti␣s␣ocjenom␣4␣ili␣5.\n");
```

Ponekad želimo neke vrijednosti navesti odvojeno. Tada u nizu znakova koji ispisujemo treba navesti gdje će se dodatne vrijednosti pojaviti, te kojeg su one tipa (cijeli broj, realni broj, znak, niz znakova,...).

Primjer 4.3.1 (Ispis dodatnih vrijednosti)**.**

```
1  ␣␣printf("Ovdje␣cemo␣ispisati␣%s␣vrijednosti:␣",
2  ␣␣␣␣"razlicite");
3  ␣␣printf(
4  ␣␣␣␣"cijeli␣broj␣%d,␣realni␣broj␣%g␣i␣znak␣%c.\n",
5  ␣␣␣␣17,
6  ␣␣␣␣17.19,␣'Q'
7  ␣␣);
```

Ova naredba ispisuje sljedeći tekst:

```
Ovdje cemo ispisati razlicite vrijednosti: cijeli broj 17,
realni broj 17.19 i znak Q.
```

Tekst će biti ispisan u jednoj liniji (između zareza i riječi "realni" nalazit će se razmak). Ovdje je razlomljen samo zato da bi stao na stranicu.

Nakon izvršavanja naredbi `printf()`*, kursor će biti na početku sljedećeg retka (zbog znaka* `\n` *na kraju).*

Podizrazi koji se navode u naredbi `printf()`, a počinju znakom `%` zovu se *formati za ispis*. Njih ćemo koristiti u svim primjerima i zadacima. Značenje samih formata je detaljno objašnjeno na predavanjima.

Primijetimo da naredbe smijemo "lomiti" između parametara (npr. novi redak između `17` i `17.19`). To je poželjno raditi da bi programi bili pregledniji, no ne treba pretjerivati.

Čemu služe formati? Očito, program koji ispisuje uvijek istu poruku nije naročito koristan. Potrebno je omogućiti učitavanje vrijednosti, računanje, snimanje i mnoge druge operacije da bi program mogao rješavati određene probleme. U tu svrhu koristimo varijable, čije vrijednosti ćemo onda ispisivati upravo pomoću formata u naredbi `printf()`.

Poglavlje 5

Varijable

U ovom poglavlju bavimo se spremanjem podataka u memoriju, bez čega bi svaki program radio uvijek isto, bez mogućnosti da mu zadajemo podatke. Prvo ćemo prikazati s kakvim sve podacima možemo baratati, a zatim i na koji način to radimo.

5.1 Uvod

Varijable su simbolička imena za memorijske lokacije u koje pohranjujemo vrijednosti. Svaka varijabla ima tip koji treba navesti prije upotrebe varijable. Osnovni tipovi su:

- brojevi (cijeli, realni,...)
- znakovi
- pokazivači (sadrže adrese drugih varijabli)

Neki jezici imaju i logički tip (varijable koje mogu imati samo dvije vrijednosti: istina i laž), no u C-u tog tipa nema. Umjesto logičkog tipa koristi se cjelobrojni, gdje 0 predstavlja laž, a sve ostale vrijednosti predstavljaju istinu (obično koristimo vrijednost 1).

Napomena 5.1.1. *Način učitavanja varijabli ovisi o tipu varijable! Varijable nekog od osnovnih tipova možemo učitati direktno (pomoću naredbe* `scanf()` *ili neke slične), dok za složene tipove treba posebno složiti dio programa za učitavanje.*

Naredba `scanf()` *prihvaća formate slične onima koje prima naredba* `printf()`*. Ti formati se moraju slagati s tipom varijable koju učitavamo.*

Napomena 5.1.2. *Funkcije u C-u* ***ne mogu mijenjati vrijednosti varijabli koje navedemo kao ulazne parametre*** *(pri pozivu funkcije)! Zbog toga funkciji* `scanf()` *zadajemo adrese varijabli u koje spremamo učitane vrijednosti. Adresu varijable dobijamo operatorom* `&`*.*

Dakle, ako imamo varijablu `x`, onda njenu adresu možemo dobiti izrazom `&x`.

Zadatak 5.1.1. *Napišite program koji učitava tri cijela broja i ispisuje njihovu sumu.*

Rješenje.

```
1 #include␣<stdio.h>
2
3 int␣main(void)␣{
4 ␣␣int␣a,␣b,␣c;␣␣␣␣␣␣␣␣␣␣␣␣␣/*␣Deklaracija␣varijabli␣*/
5 ␣␣scanf("%d",␣&a);␣␣␣␣␣␣␣␣/*␣Ucitavanje␣jedne␣var.␣*/
6 ␣␣scanf("%d␣%d",␣&b,␣&c);␣/*␣Ucitavanje␣dvije␣var.␣*/
7 ␣␣printf("%d␣+␣%d␣+␣%d␣=␣%d\n",␣a,␣b,␣c,␣a␣+␣b␣+␣c);
8 ␣␣return␣0;
9 }
```

U naredbama `scanf()` (linije 5 i 6), kao i u naredbi `printf()`, prvo navodimo formate vrijednosti koje učitavamo. Pri tome, `%d` označava cijeli broj. Nakon formata (navedenih unutar dvostrukih navodnika) popisujemo **adrese varijabli** koje učitavamo.

U liniji 7 vidimo klasičan ispis tri varijable i jednog izraza. □

Napomena 5.1.3 (Komentari)**.** *Tekst naveden između* `/*` *i* `*/` *je komentar, što znači da ga compiler ignorira. Koristimo ga kao zabilješke za lakše snalaženje u programskom kodu.*

Na sličan način učitavamo znakove (tip `char`), ali upotrebom formata `%c`. Bitna razlika je u tome što prilikom učitavanja više brojeva, pripadne formate odvajamo razmakom (v. liniju 6 u rješenju prethodnog zadatka), dok kod znakova taj razmak izostavljamo. Razlozi takvog navođenja formata objašnjeni su na predavanjima.

Zadatak 5.1.2. *Napišite program koji učitava tri znaka, te ih ispisuje obrnutim redoslijedom.*

Učitavanje niza znakova je složenije i to ćemo kasnije obraditi. Ako želimo ispisati niz znakova, to radimo pomoću formata `%s`. Sam niz znakova navodimo pomoću dvostrukih navodnika (v. primjer 4.3.1). Na primjer, dio koda

```
printf("Danas %sje lijep dan.\n", "ni");
```

će ispisati

```
Danas nije lijep dan.
```

Za realne brojeve koristimo tipove `float` (format `%f`) i `double` (format `%lg`). Potonji omogućuje veći raspon vrijednosti.

Zadatak 5.1.3. *Napišite program koji učitava tri realna broja (jedan tipa* `float` *i dva tipa* `double`*), te ispisuje njihov produkt.*

Rješenje.

```
#include <stdio.h>

int main(void) {
  float a;
  double b, c;
  scanf("%f", &a);
  scanf("%lg %lg", &b, &c);
  printf("%g * %g * %g = %g\n", a, b, c, a * b * c);
  return 0;
}
```

Prilikom ispisa varijabli tipa `double` ne treba navoditi `l` u formatu. □

5.2 Još o formatima

Pomoću formata možemo dodatno definirati izgled ispisanog teksta.

Primjer 5.2.1.

```
#include <stdio.h>

int main (void) {
  int i;

  /* ucitavanje var. tipa int u dekadskom zapisu */
  scanf("%d", &i);

  /* ispisujemo varijablu i u razlicitim formatima: */

  /* ----dekadskom ---> %d ili %i */
  printf("dekadski zapis: %d\n", i);

  /* ----oktalnom ---> %o */
  printf("oktalni zapis: %o\n", i);

  /* ----hexadecimalnom s velikim slovima ---> %X */
  printf("hexadec. zapis s velikim slovima: %X\n", i);

  /* ----hexadecimalnom s malim slovima ---> %x */
  printf("hexadec. zapis s malim slovima: %x\n", i);

```

```
23  ␣␣/*␣----␣min.␣10␣mjesta␣,␣desno␣poravnavanje␣->␣%10d␣*/
24  ␣␣printf("10␣mjesta␣desno␣poravnavanato:␣%10d\n",␣i);
25
26  ␣␣/*␣----␣min.␣7␣mjesta␣,␣lijevo␣poravnavanje␣->␣%-7d␣*/
27  ␣␣printf("7␣mjesta␣s␣lijevo␣poravnavnato:␣%-7d\n",␣i);
28
29  ␣␣/*␣ucitavanje␣u␣oktalnom␣ili␣hex-formatu:␣*/
30  ␣␣printf("unesite␣cijeli␣broj␣u␣hex-formatu␣");
31  ␣␣printf("(mala␣slova):␣");
32  ␣␣scanf("%x",␣&i);
33  ␣␣printf("dekadska␣vrijednost␣od␣%x␣je␣%d\n",␣i,␣i);
34
35  ␣␣/*␣ucitamo␣li␣int␣sa␣%i␣format␣se␣automatski␣*/
36  ␣␣/*␣prepoznaje␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣*/
37  ␣␣/*␣0123␣->␣oktalni␣zapis;␣0x123␣->␣hex-zapis␣*/
38  ␣␣scanf("%i",␣&i);
39  ␣␣printf("dec:␣%d;␣oct:␣%o;␣hex:␣%x",␣i,␣i,␣i);
40
41  ␣␣return␣0;
42  }
```

Formati `%o`, `%x` *i* `%X` *rade samo s nenegativnim cijelim brojevima!*

Formati kod ispisa realnih brojeva:

- `%f` – obicni decimalni zapis, npr. 3.14159
- `%e` – e-notacija, npr. -3.14e-3; koristi slovo e
- `%E` – e-notacija, npr. -3.14E-3; koristi slovo E
- `%g` – automatski se prepoznaje o kojem se tipu zapisa radi; ako je potrebna e-notacija, koristi se slovo e
- `%G` – automatski se prepoznaje o kojem se tipu zapisa radi; ako je potrebna e-notacija, koristi se slovo E

Napomena 5.2.1. *Znak "%" ispisujemo navođenjem formata* `%%`.

Primjer 5.2.2.

```
1  #include␣<stdio.h>
2
3  int␣main␣(void)␣{
4  ␣␣double␣dbl;
5  ␣␣float␣flt;
6
7  ␣␣scanf("%f",␣&flt);
```

```
8  ␣␣printf("%%f␣-->␣%f\n",␣flt);
9  ␣␣printf("%%e␣-->␣%e\n",␣flt);
10 ␣␣printf("%%g␣-->␣%g\n",␣flt);
11
12 ␣␣/*␣da␣bi␣ucitali␣double␣NUZNO␣je␣koristiti␣prefix␣l
13 ␣␣/*␣za␣ispis␣ne␣treba␣*/
14 ␣␣scanf("%lg",␣&dbl);
15
16 ␣␣/*␣%.5f␣-->␣zaokruzeno␣na␣5␣decimalnih␣mjesta␣*/
17 ␣␣printf("zaokruzeno␣na␣5␣dec.␣mjesta:␣%.5f\n",␣dbl);
18
19 ␣␣/*␣%.0f␣-->␣zaokruzeno␣na␣0␣dec.␣mjesta␣*/
20 ␣␣/*␣␣␣␣␣␣␣␣␣␣decimalna␣tocka␣se␣ne␣ispisuje␣*/
21 ␣␣printf("zaokruzeno␣na␣0␣dec.␣mjesta:␣%.0f\n",␣dbl);
22
23 ␣␣/*␣%10.3f␣-->␣zaokruzeno␣na␣3␣dec.␣mjesta␣,␣trosi␣*/
24 ␣␣/*␣␣␣␣␣␣␣␣␣␣␣ukupno␣najmanje␣10␣znakova␣*/
25 ␣␣printf("zaokruzeno␣na␣3␣dec.␣mjesta,␣ukupno␣bar␣");
26 ␣␣printf("10␣znakova:␣\'%10.3f\'\n",␣dbl);
27
28 ␣␣/*␣Na␣isti␣nacin␣zaokruzujemo␣i␣s␣%e␣i␣%g␣*/
29 ␣␣printf("2␣dec.␣mjesta,␣ukupno␣bar␣12␣znakova,␣");
30 ␣␣printf("poravnato␣lijevo:␣\'%-12.2e\'\n",␣dbl);
31
32 ␣␣return␣0;
33 }
```

Pokazivači (engl. *pointeri*) sadrže, kako smo naveli, memorijske adrese. Računalo sve, pa tako i adrese, pamti kao brojeve. Zbog toga pokazivače možemo ispisati kao prirodne brojeve (`unsigned int`):

```
1 int␣x␣=␣17;
2 printf("x␣=␣%d,␣&x␣=␣%u\n",␣x,␣&x);
```

Napomena 5.2.2. *U nastavku ćemo uglavnom navoditi samo tijela ili dijelove programa, bez zaglavlja. Naravno, inače* ***programe treba pisati cijele, osim ako je u zadatku naglašeno da se piše samo dio programa!*** *Da biste na računalu testirali dijelove programa, potrebno je dopisati one dijelove koji nedostaju (u pravilu zaglavlja, deklaracije i učitavanja varijabli, ispisi,...).*

Također, više nećemo posebno označavati razmake u programima.

Poglavlje 6

Osnovne računske operacije

Vrijednosti nije dovoljno pohraniti. Očito, potrebno nam je i nekakvo mijenjanje tih vrijednosti koje se obično svodi na računanje i pohranjivanje novih vrijednosti u nove ili već korištene varijable.

6.1 Uvod

Cijele brojeve možemo zbrajati (operator `+`), oduzimati (operator `-`), množiti (operator `*`) i dijeliti (operatori `/` i `%`). Operatori `+`, `-` i `*` se ponašaju identično kao u matematici (s tim da treba paziti da rezultat nije prevelik, tj. da ga spremamo u varijablu s dovoljno memorije).

Operator dijeljenja, `/`, vraća **cjelobrojni** rezultat dijeljenja dva cijela broja, dok operator ostatka, `%`, vraća **ostatak** pri dijeljenju dva cijela broja.

Ukoliko je jedan od operanada realan broj, operacije se vrše realno (a ne cjelobrojno).

Primjer 6.1.1.

```
1 int a = 17, b = 7;
2 double c = 17.0, d = 7.0;
3 printf("a+b=%d, a-b=%d, a*b=%d, a/b=%d, a%%b=%d\n",
4   a+b, a-b, a*b, a/b, a%b);
5 printf("a+d=%g, a-d=%g, a*d=%g, a/d=%g",
6   a+d, a-d, a*d, a/d);
7 printf("c+d=%g, c-d=%g, c*d=%g, c/d=%g",
8   c+d, c-d, c*d, c/d);
```

Ispis ovog dijela programa bit će

```
a+b=24, a-b=10, a*b=119, a/b=2, a%b=3
a+d=24, a-d=10, a*d=119, a/d=2.42857
c+d=24, c-d=10, c*d=119, c/d=2.42857
```

Primijetite razliku u rezultatima cjelobrojnog (prva linija ispisa) i realnog dijeljenja (druga i treća linija ispisa).

Napomena 6.1.1 (Česta greška)**.** *Razlikujte znakove "/", "|" i "\"! Znak "/" označava dijeljenje koje* ***ne smijete*** *pisati "*`a \ b`*", dok znak "\" služi za specijalne znakove, tj. skok u novi red* ***ne smijete*** *pisati "*`/n`*"! Znak "|" označava operator bitwise-OR i ne može biti zamjena niti za jedan od preostala dva znaka.*

6.2 Pridruživanje

Varijabli pridružujemo vrijednost tako da iza nje stavimo znak `=` i desno od njega navedemo vrijednost koju pridružujemo. Na primjer, izraz

```
x = 17;
```

će varijabli `x` pridružiti vrijednost 17. Na sličan način varijabli možemo pridružiti i rezultat nekog izraza. Na primjer, izraz

```
x = a + b;
```

će varijabli `x` pridružiti rezultat zbrajanja varijabli `a` i `b`.

Svi izrazi u C-u vraćaju neku vrijednost, pa tako i izraz `x = A`. Rezultat koji taj izraz vraća je upravo vrijednost izraza `A`. Zbog toga smijemo napisati i

```
y = (x = 17);
```

što će varijablama `x` i `y` pridružiti vrijednost 17. Dapače, ovakvo pridruživanje možemo navesti i bez zagrada (iz razloga objašnjenih na predavanjima). Na primjer, izraz

```
y = x = a + b;
```

će varijablama `x` i `y` pridružiti rezultat zbrajanja vrijednosti koje se nalaze u varijablama `a` i `b`.

Vrijednost koju ćemo pridružiti nekoj varijabli smijemo računati pomoću te iste varijable. Na primjer:

```
p = p * 2;
```

Prilikom pridruživanja izraza varijabli, računalo prvo računa vrijednost desne strane, te tu vrijednost pridružuje varijabli s lijeve strane znaka jednakosti. Dakle, ovaj izraz će udvostručiti vrijednost varijable `p`.

Navedeni izraz smo mogli i kraće zapisati:

`p *= 2.`

Na taj način možemo skratiti i ostale izraze oblika

`a = a` op `b`,

gdje je "op" neki operator. Dakle, sljedeći izrazi su ekvivalentni:

`x = x + d`	`x += d`
`x = x - d`	`x -= d`
`x = x * d`	`x *= d`
`x = x / d`	`x /= d`
`x = x % d`	`x %= d`

Isti način skraćivanja može se primijeniti i na operatore koje ćemo tek obraditi (logičke, bitovne,...).

Postoje još i sljedeće posebne pokrate:

`x++` i `++x` za inkrement, te
`x--` i `--x` za dekrement.

Pri tome `x++` i `++x` povećavaju vrijednost varijable `x` za jedan, a razlikuju se u povratnoj vrijednosti izraza. Naime, `x++` vraća **staru** vrijednost varijable `x` (tj. onu vrijednost koju je varijabla imala prije inkrementa), dok `++x` vraća **novu** vrijednost varijable `x` (tj. onu vrijednost koju je poprimila inkrementom).

Na sličan način, izraz `x--` smanjuje vrijednost od `x` za jedan i vraća staru vrijednost, dok `--x` smanjuje vrijednost od `x` za jedan, ali vraća novu vrijednost.

Primjer 6.2.1. *Pogledajmo dio programa:*

```
1  int i = 17, d = 17;
2
3  printf("i = %d, ", i);
4  printf("(i++) = %d, ", i++);
5  printf("i = %d\n", i);
6
7  printf("i = %d, ", i);
8  printf("(++i) = %d, ", ++i);
9  printf("i = %d\n", i);
10
11 printf("d = %d, ", d);
12 printf("(d--) = %d, ", d--);
13 printf("d = %d\n", d);
14
```

```
15 printf("d = %d, ", d);
16 printf("(--d) = %d, ", --d);
17 printf("d = %d\n", d);
```

Ispis ovog dijela programa će biti:

```
i = 17, (i++) = 17, i = 18
i = 18, (++i) = 19, i = 19
d = 17, (d--) = 17, d = 16
d = 16, (--d) = 15, d = 15
```

Kao što vidimo, izraz `i++` *je povećao vrijednost varijable* `i` *sa* 17 *na* 18*, ali je vrijednost izraza jednaka* 17 *(dakle, vrijednost koju je varijabla* `i` *imala prije inkrementa). S druge strane, izraz* `++i` *je također povećao vrijednost varijable* `i` *(ovaj put s* 18 *na* 19*), ali je njegova povratna vrijednost jednaka novoj vrijednosti varijable* `i` *(dakle,* 19*).*

Analogno vrijedi i za operator `--`*.*

Napomena 6.2.1. *Na sljedeći način možete jednostavno zapamtiti "koji je koji" (na primjeru operatora za inkrement):* `++x` *ima operator* ***prije*** *varijable, pa se zato* prvo poveća vrijednost varijable, a zatim vrati. *S druge strane, izraz* `x++` *ima operator* ***nakon*** *varijable, pa se zato* prvo vrati vrijednost varijable, pa tek onda poveća.

Tehnički, ovo nije ispravno, jer se vrijednost izraza vraća uvijek nakon što je sve ostalo računanje gotovo, no dobro je kao sugestija prilikom pamćenja operatora.

Poglavlje 7

Grananje

Često želimo da se, ovisno o vrijednostima varijabli, izvrše različite naredbe. U tu svrhu koristimo obično grananje (`if`), uvjetni operator (`?:`) ili višestruko grananje (koje ćemo obraditi u poglavlju 9).

7.1 Obično grananje

Zadatak 7.1.1. *Napišite dio programa koji učitava cijeli broj i ispisuje poruku je li taj broj negativan.*

Rješenje.

```
1 int x;
2 scanf("%d", &x);
3 if (x < 0)
4   printf("Broj %d je negativan.\n", x);
5 else
6   printf("Broj %d nije negativan.\n", x);
```

□

Grananje ima nekoliko komponenti:

1. Ključna riječ `if` i uvjet koji provjeravamo.
2. Programski kod koji se izvodi ako je uvjet istinit.
3. Ključna riječ `else` i programski kod koji se izvršava ako uvjet nije istinit. Ovaj dio nije nužno navesti ako u slučaju neispunjenja uvjeta ne treba ništa izvršiti.

Napomena 7.1.1. *Ako se u nekom slučaju (npr. kad je uvjet ispunjen) treba izvršiti više naredbi, onda ih grupiramo pomoću vitičastih zagrada!*

Zadatak 7.1.2. *Napišite dio programa koji učitava cijeli broj i ispisuje poruku je li taj broj paran.*

Uputa. Upotrijebite operator `%`. □

Uvjete možemo kombinirati kao i u logici sudova.

Zadatak 7.1.3. *Napišite dio programa koji učitava cjelobrojne varijable* `a`, `b` *i* `c`, *te ispisuje je li a veći od b i c.*

Rješenje.

```
int a, b, c;
scanf("%d %d %d", &a, &b, &c);
if (a > b && a > c)
  printf("Broj %d je veci od %d i %d.\n", a, b, c);
else
  printf("Broj %d nije veci od %d i %d.\n", a, b, c);
```

□

Operator `&&` označava "logičko I", operator `||` označava "logičko ILI", dok operator `!` označava logičku negaciju. Dakle izraz

$$(a > b) \text{ i } (c \not> d \text{ ili } e = d)$$

C-ovski zapisujemo kao

`(a > b) && (!(c > d) || e == d).`

Napomena 7.1.2 (Česta greška)**.** *Operator za provjeru jednakosti dva broja je* `==` *(**dvostruko** "jednako"). Treba paziti da ga se ne zamijeni s* `=` *(jednostruko "jednako"), što je uvijek pridruživanje!*

Napomena 7.1.3. *U C-u ne postoje znakovi* $\leq$ *i* $\geq$*. Umjesto njih, koristimo* `<=` *i* `>=` *respektivno (**NE*** `=<` *i* `=>`*!).*

Operator za provjeru "različitosti" je `!=` ("uskličnik" i "jednako", **bez** razmaka između). Dakle, uvjet `x != y` znači $x \neq y$.

Napomena 7.1.4 (Česta greška)**.** *Operatori* `&&` *i* `||` *povezuju **logičke izraze**. Drugim riječima, izraz*

$$a > b, c$$

ne smijemo[1] *zapisati kao*

[1] U stvari, smijemo napisati i `a > b && c` i `a > b, c`, ali će rezultat u oba slučaja biti pogrešan (evaluirat će se drugačije od očekivanog). Pokušajte otkriti kako C interpretira takve izraze.

`a > b && c` *ili* `a > b, c`.

Ispravan *način za zapisati zadani izraz je*

`a > b && a > c`.

Zadatak 7.1.4. *Napišite dio programa koji učitava cjelobrojne varijable* `a`, `b` *i* `c` *koje imaju vrijednosti* 0 *ili* 1 *(potrebno je provjeriti da je dobro upisano i, ako nije, prijaviti grešku). Potom, program treba reći je li izraz*

$$\bar{a}bc + a\bar{b}c + ab\bar{c} + abc$$

istinit ili lažan.

Uputa. Pojednostavite izraz, "pretvorite" ga u C-ovski zapis i upotrijebite `if`. □

7.2 Uvjetni operator

U C-u postoji posebni operator `?:` koji služi za jednostavnije ubacivanje uvjeta u izraze. Osnovni poziv operatora izgleda ovako:

```
x = (UVJ ? A : B);
```

Pri tome, varijabla `x` poprima vrijednost izraza `A` ako je uvjet `UVJ` zadovoljen, odnosno vrijednost izraza `B` ako uvjet `UVJ` nije zadovoljen. Zapisano pomoću `if`, taj izraz izgleda ovako:

```
if (UVJ) x = A; else x = B;.
```

Primjer 7.2.1. *Učitavamo cijeli broj* `x` *i ispisujemo prvi parni broj strogo veći od njega. Pri tome koristimo činjenicu da je idući parni broj* `x+1` *ako je* `x` *neparan, odnosno* `x+2` *ako je* `x` *paran.*

```
1 scanf("%d", &x);
2 printf("Prvi parni broj stro\-go veci od %d je %d.",
3   x, (x % 2 == 1 ? x + 1 : x + 2));
```

Napomena 7.2.1. *Podsjetimo se: ako broj promatramo kao logički izraz, onda* 0 *označava laž, a sve ostalo istinu. Zbog toga, gornji dio koda smo mogli zapisati i bez "*`== 1`*":*

```
1 scanf("%d", &x);
2 printf("Prvi parni broj stro\-go veci od %d je %d.",
3   x, (x % 2 ? x + 1 : x + 2));
```

Poglavlje 8

Petlje

Gotovo svaki program neke dijelove koda izvodi više puta, u pravilu svaki put s drugim vrijednostima nekih varijabli. U tu svrhu koristimo razne petlje. U C-u postoje `while()`, `for()` i `do...while()`, a izbor one "prave" ovisi o tome što se točno traži, kao i o ukusu programera. Same petlje su ekvivalentne i svaka može manje ili više jednostavno zamijeniti ostale dvije.

Nakon pregleda petlji u C-u, objasnit ćemo što je to složenost algoritama i kako uspoređujemo različite algoritme koji rješavaju isti problem.

8.1 `while()`-petlja

Zadatak 8.1.1. *Napišite dio programa koji učitava prirodni broj x i ispisuje sve potencije broja 2 strogo manje od x.*

Rješenje.

```
1 unsigned long int x, p = 1;
2
3 printf("Unesite x: ");
4 scanf("%lu", &x);
5 while (p < x) {
6   printf("%u\n", p);
7   p *= 2;
8 }
```

□

U prethodnom rješenju imamo nekoliko novosti:

- U deklaraciji (linija 1) deklariramo varijable `x` i `p`, te za njihov tip navodimo `unsigned long int`.

Kad brojevnom tipu dodamo atribut `unsigned`, compileru dajemo do znanja da ćemo u tu varijablu pohranjivati isključivo nenegativne vrijednosti (engl. *unsigned* znači "bez predznaka"). Zbog toga što ne trebamo pamtiti negativne brojeve, umjesto njih možemo pamtiti duplo više pozitivnih brojeva. Varijable s atributom `unsigned` se u računalu pamte kao obični binarni brojevi, jer nema potrebe za dvojnim komplementima.
Atribut `long` znači da će varijabla zauzeti dvostruko više memorije. Posljedica je opet mogućnost memoriranja više vrijednosti. Kako ovdje radimo s potencijama, vrijednosti će brzo rasti, pa je dobro imati varijable koje mogu pamtiti što više vrijednosti.

- Također u deklaraciji, imamo izraz `p = 1`. Naime, možemo zadati vrijednost varijable odmah prilikom deklaracije. Tako smo ovdje deklarirali varijablu `x` kojoj **nismo** zadali nikakvu vrijednost (pa će ona biti neka slučajna) i varijablu `p` kojoj dajemo vrijednost 1.

- U liniji 4 učitavamo cjelobrojnu vrijednost u varijablu `x`. Format `%u` najavljuje učitavanje varijable tipa `unsigned int`, dok slovo `l` dodajemo prilikom učitavanja bilo koje varijable tipa `long`.
Primijetite da smo slovo `l` dodali i prilikom učitavanja varijable `dbl` u primjeru 5.2.2. To je zato jer je `double` zapravo `long float`.

- U liniji 5 vidimo zaglavlje `while` petlje: ključna riječ `while` i uvjet (naveden u oblim zagradama) do kada se petlja izvršava.

Nakon svakog izvršavanja tijela petlje, provjerava se uvjet. Ako je on istinit, tijelo petlje se ponovno izvršava; ako nije, izvršavanje programa se nastavlja nakon petlje.

U praksi, prirodne brojeve možemo spremati i u "običan" `int` (dakle, ne nužno u `unsigned int`). Mi ćemo uglavnom koristiti `int`, neovisno o tome radi li se s prirodnim ili s cijelim brojevima, jer udvostručenje raspona nije naročito veliko, a može prouzročiti probleme (kako ćemo vidjeti u jednom od zadataka).

Napomena 8.1.1. *Uvjet petlje se **ne provjerava** u toku izvođenja tijela petlje!*

Zadatak 8.1.2. *Napišite dio programa koji učitava prirodni broj* `x`*, te ispisuje je li on prost ili nije.*

Rješenje.

```
1 int x, p = 2, prost = 1;
2
3 printf("Unesite x: ");
4 scanf("%d", &x);
5 if (x < 2) prost = 0;
6 while (prost && p < x) {
7   if (x % p == 0) prost = 0;
```

```
8     p++;
9   }
10  if (prost) printf("Broj je prost.\n"); else
11    printf("Broj nije prost.\n");
```

□

8.2 for()-petlja

Zadatak 8.2.1. *Napišite dio programa koji učitava prirodni broj* `n`*, te ispisuje koliko je* $k!$ *za sve prirodne* k *manje ili jednake* `n`*.*

Rješenje.

```
1  int k, n, fakt = 1;
2
3  printf("Unesite n: "); scanf("%d", &n);
4  for (k = 1; k <= n; k++) {
5    fakt *= k;
6    printf("%d! = %d\n", k, fakt);
7  }
```

□

Navedeni programski isječak je ekvivalentan sljedećem kodu:

```
1  int k, n, fakt = 1;
2
3  printf("Unesite n: "); scanf("%d", &n);
4  k = 1;
5  while (k <= n) {
6    fakt *= k;
7    printf("%d! = %d\n", k, fakt);
8    k++;
9  }
```

Prikazana konverzija `for()` u `while()` se uvijek može provesti.

Kod `for()` petlje imamo tri polja:

`for(INICIJALIZACIJA; UVJET; INKREMENT).`

Nije nužno navesti sva polja. Na primjer, prethodni zadatak smo mogli riješiti i ispuštanjem (tj. premještanjem) inicijalizacije i inkrementa:

```
1  int k, n, fakt = 1;
2
```

```
3 printf("Unesite n: "); scanf("%d", &n);
4 k = 1;
5 for ( ; k <= n ; ) {
6   fakt *= k;
7   printf("%d! = %d\n", k, fakt);
8   k++;
9 }
```

Primijetite da separatori između polja (;) moraju ostati!

8.3 do...while()-petlja

Ponekad želimo da se dio programa izvrši nekoliko puta, ali **bezuvjetno** barem jednom. Kod `while()` i `for()`, uvjet se provjerava prije prvog izvršavanja tijela petlje. Da bismo tu (prvu) provjeru izbjegli, koristimo `do...while` petlju.

Zadatak 8.3.1. *Napišite dio programa koji učitava prirodne brojeve i ispisuje njihovu sumu. Učitavanje treba prekinuti kad korisnik upiše vrijednost* 0*.*

Rješenje.

```
1 int sum = 0, x;
2
3 do {
4   printf("Unesite broj: ");
5   scanf("%d", &x);
6   sum += x;
7 } while (x != 0);
8
9 printf("Suma ucitanih brojeva je %d.\n", sum);
```

□

Ovaj kod trivijalno možemo pretvoriti u `while()` ili `for()` na sljedeći način

```
1 int sum = 0, x = 1;
2
3 while (x != 0) {
4   printf("Unesite broj: ");
5   scanf("%d", &x);
6   sum += x;
7 };
8
9 printf("Suma ucitanih brojeva je %d.\n", sum);
```

Primijetite da smo varijabli `x` pridružili vrijednost 1 (mogla je biti bilo koja vrijednost različita od nule!). Bez toga, program bi bio neispravan jer **ne smijemo čitati vrijednost**

varijable prije nego joj pridružimo neku vrijednost (npr. pomoću pridruživanja ili `scanf()`).

Ovdje smo iskoristili pomoćnu varijablu `x`, no postoji i "recept" koji uvijek vrijedi.

Primjer 8.3.1. *Pretvorite programski kod*

```
do {
  TIJELO;
} while (UVJET);
```

u `while()` *i* `for()`.

To radimo korištenjem pomoćne varijable koja će biti istinita samo prilikom prve provjere uvjeta (te će tako osigurati da se tijelo petlje barem jednom izvrši):

```
prvi_korak = 1;
while (prvi_korak || UVJET) {
  prvi_korak = 0;
  TIJELO;
}
```

Analogno, slažemo `for()` *petlju:*

```
for (prvi_korak = 1; prvi_korak || UVJET; ) {
  prvi_korak = 0;
  TIJELO;
}
```

Ovdje, `for()` *petlja nema inkrement, ali ima inicijalizaciju.*

Zadatak 8.3.2. *Pretvorite sljedeće programske isječke u* `do...while` *petlju:*

```
for (INICIJALIZACIJA; UVJET; INKREMENT) {
  TIJELO;
}
```

i

```
while (UVJET) {
  TIJELO;
}
```

8.4 Zadaci

Zadatak 8.4.1. *Napišite dio programa koji učitava cijele brojeve dok korisnik ne upiše nulu. Program treba ispisati produkt svih brojeva koje je korisnik učitao.*

Rješenje. U zadatku 8.3.1, učitanu nulu smo pribrojili sumi. To smijemo raditi u slučaju sume jer ona ostaje nepromijenjena. No, u slučaju produkta, nula bi poništila rezultat, pa na to treba paziti.

```
int prod = 1, x;

do {
  printf("Unesite broj: ");
  scanf("%d", &x);
  if (x != 0) prod *= x;
} while (x != 0);

printf("Produkt ucitanih brojeva je %d.\n", prod);
```

□

Zadatak 8.4.2. *Napišite dio programa koji učitava prirodne brojeve dok korisnik ne upiše nulu. Program treba ispisati vrijednost najvećeg učitanog broja, te koji je on (po redu) učitan.*

Rješenje. Pošto učitavamo **prirodne** brojeve, znamo da će svi biti nenegativni. Dakle, možemo pretpostaviti da je maksimum neki negativni broj (npr. -1), pa će prvi učitani broj sigurno biti veći i onda će postaviti ispravnu vrijednost maksimuma.

Nadalje, moramo pamtiti i koji broj (po redu) učitavamo, te koji je redni broj samog maksimuma. Za to će služiti varijable `i` (kao *index*) i `im` (kao *index maksimuma*).

```
int max = -1, x, i = 0, im;

do {
  printf("Unesite broj: ");
  scanf("%d", &x);
  i++;
  if (x > max) {
    max = x;
    im = i;
  }
} while (x != 0);

printf("Najveci ucitani broj je %d, ucitan", max);
printf(" %d. po redu.\n", im);
```

□

Zadatak 8.4.3. *Napišite dio programa koji učitava **cijele** brojeve dok korisnik ne upiše nulu. Program treba ispisati vrijednost najvećeg učitanog broja, te koji je on (po redu) učitan.*

Rješenje. Na žalost, sada ne možemo krenuti od najmanjeg broja jer takav ne postoji u skupu cijelih brojeva. Zbog toga je bolje prvi broj učitati posebno i odmah tu vrijednost pridružiti varijabli `max`. Također, treba zadati i početnu vrijednost varijabli `i` i `im`.

```
int max, x, i = 1, im = 1;

printf("Unesite broj: ");
scanf("%d", &x);
max = x;

while (x != 0) {
  printf("Unesite broj: ");
  scanf("%d", &x);
  i++;
  if (x && x > max) {
    max = x;
    im = i;
  }
}

printf("Najveci ucitani broj je %d, ucitan", max);
printf(" %d. po redu.\n", im);
```

Prikazani način djeluje općenito (dakle, mogao je biti ispravno rješenje prethodnog zadatka). Drugi mogući način je izoliranje prvog koraka (u ovisnosti o parametru `i`):

```
int max, x = 1, i = 0, im;

while (x != 0) {
  printf("Unesite broj: ");
  scanf("%d", &x);
  i++;
  if (i == 1 || (x && x > max)) {
    max = x;
    im = i;
  }
}

printf("Najveci ucitani broj je %d, ucitan", max);
printf(" %d. po redu.\n", im);
```

Primijetite da smo sada varijabli `x` morali zadati inicijalnu vrijednost koja osigurava ulazak u petlju u prvom koraku. To nije morala biti vrijednost 1, nego bilo koja koja zadovoljava uvijet `while()` petlje (`x != 0`). □

Zadatak 8.4.4. *Napišite dio programa koji učitava prirodni broj* `n`*, te učitava* `n` *realnih brojeva. Program treba ispisati sumu najvećeg i najmanjeg učitanog broja.*

Uputa. Kad petlja treba imati unaprijed određeni broj koraka, praktično je upotrijebiti `for()`. Preporuča se uzeti inicijalizaciju `i = 0` i uvjet `i < n`, jer je tako lakše raditi s poljima (koja ćemo kasnije obraditi). □

Zadatak 8.4.5. *Napišite dio programa koji učitava prirodni broj* `n`*, te učitava* `n` *cijelih brojeva. Program treba ispisati sumu svih parnih i produkt svih neparnih brojeva.*

Ako nije učitan niti jedan paran broj, pripadna suma je 0*; ako nije učitan niti jedan neparan broj, pripadni produkt je* 1*.*

Zadatak 8.4.6. *Napišite dio programa koji učitava prirodne brojeve* `n` *i* `k`*, te ispisuje koliko je* $\binom{n}{k}$*.*

Rješenje. Prema definiciji vrijedi:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

No, faktorijele su često veliki brojevi, što može izazvati *overflow*. Zbog toga je puno bolje računati

$$\frac{n \cdot (n-1) \cdot \cdots \cdot (k+1)}{(n-k)!} \text{ ili } \frac{n \cdot (n-1) \cdot \cdots \cdot (n-k+1)}{k!},$$

ovisno o tome kakvi su n i k. Prva formula je "bolja" ako je $n - k < k$, tj. $n < 2k$.

Slično prethodnom programu, opet je praktično upotrijebiti `for()`:

```
1  int i, n, k, povrh = 1;
2  printf("Unesite n: "); scanf("%d", &n);
3  printf("Unesite k: "); scanf("%d", &k);
4  if (n < 2 * k) {
5    for (i = n; i > k; i--) povrh *= i;
6    for (i = 1; i <= n - k; i++) povrh /= i;
7  } else {
8    for (i = n; i > n - k; i--) povrh *= i;
9    for (i = 1; i <= k; i++) povrh /= i;
10 }
11 printf("Rezultat: %d\n", povrh);
```

Primijetimo da promjenom inkrementa (i, shodno tome, inicijalizacije i uvjeta), `for()` petlja može ići i "prema gore" i "prema dolje". □

Napomena 8.4.1. *Primijetimo da su dijeljenja u linijama* 6 *i* 9 *u rješenju prethodnog zadatka* ***cjelobrojna****. To smijemo napraviti zato jer znamo da je* $\binom{n}{k}$ *uvijek cijeli broj! Inače, morali bismo paziti da dijeljenje bude realno.*

Zadatak 8.4.7. *Napišite dio program koji za zadani prirodni broj* `n` *ispisuje vrijednost izraza*

$$\sum_{i=1}^{n} a_i \cdot i,$$

gdje je

$$a_i = \begin{cases} 1, \, i \text{ paran} \\ 3, \, i \text{ neparan} \end{cases}.$$

Rješenje. Uz `for()` petlju, ovdje će nam pomoći uvjetni operator:

```
int i, res = 0;
for (i = 1; i <= n; i++)
  res += i * (i % 2 ? 3 : 1);
printf("%d\n", res);
```

□

Napomena 8.4.2. *Podsjetimo se: izraz* `i % 2` *vraća ostatak pri dijeljenju* `i` *s* 2*. Pošto ga koristimo kao uvjet, taj uvjet će biti istinit ako je ostatak različit od* 0 *(tj. ako je broj neparan). Ako je broj paran, tj. rezultat je nula, uvjet je lažan.*

Petlje se mogu smještati jedna u drugu. Pri tome treba paziti kako koristimo varijable, da jedna petlja ne bi narušila vrijednosti o kojima ovisi uvjet druge petlje.

Za razliku od dijelova programa, napišimo i jedan **cijeli** program:

Zadatak 8.4.8. *Napišite program koji učitava* `n` *prirodnih brojeva, te ispisuje produkt prostih.*

Rješenje. U zadatku 8.1.2 smo vidjeli kako se provjerava je li broj prost. Sad ćemo taj programski kod kombinirati s rješenjem zadatka 8.4.5:

```
#include <stdio.h>

int main(void) {
  int n, i, x, p, prost, prod = 1;

  printf("Unesite n: ");
  scanf("%d", &n);

  for (i = 0; i < n; i++) {
    printf("Unesite broj: ");
    scanf("%d", &x);
    p = 2;
    prost = (x > 1);
    while (prost && p < x) {
      if (x % p == 0) prost = 0;
      p++;
    }
    if (prost) prod *= x;
  }
  printf("%d\n", prod);
  return 0;
}
```

□

Zadatak 8.4.9. *Napišite dio programa koji za zadane prirodne brojeve* `n` *i* `b` *(*`b` ≥ 2*) ispisuje koliko znamenaka ima dekadski broj* `n` *ako ga zapišemo u bazi* `b`*.*

Rješenje.

```
int i = 0;
while (n > 0) {
  n /= b;
  i++;
}
printf("Rezultat: %d\n", i);
```

□

Zadatak 8.4.10. *Primijetite da bi u rješenju prethodnog zadaka ispis*

```
printf("Broj %d u bazi %d ima %d znamenaka.\n", n, b, i);
```

bio pogrešan. U čemu je greška i kako ju ispraviti (a da poruka ostane kako smo ju upravo naveli)?

Uputa. Isprobajte program na računalu i primijetite da će ispis uvijek započinjati s "`Broj 0 u bazi...`". Vrijednost koju `n` ima prije petlje treba sačuvati u pomoćnoj varijabli jer uzastopnim dijeljenjima tu vrijednost gubimo! □

Zadatak 8.4.11. *Napišite dio programa koji učitava prirodne brojeve dok ne učita nulu. Za svaki broj (osim nule) treba ispisati sumu njegovih znamenaka u bazi* 7*.*

Rješenje.

```
int x, sum, t;

do {
  printf("Unesite broj: "); scanf("%d", &x);
  sum = 0;
  if (x) {
    t = x;
    while (t > 0) {
      sum += (t % 7);
      t /= 7;
    }
    printf("Suma znamenaka: %d\n", sum);
  }
} while (x);
```

□

Napomena 8.4.3. *Kao što vidimo, ako zadatak zahtijeva, rezultat se može ispisati i u tijelu petlje; ne nužno izvan nje. Nema recepta gdje ide ispis rezultata; to treba shvatiti iz teksta zadatka!*

Sljedeći zadatak je izuzetno važan za razumjeti, zajedno s čestim greškama koje navodimo odmah nakon ispravnog rješenja.

Zadatak 8.4.12. *Napišite dio programa koji učita prirodni broj* `b`*, a zatim učitava prirodne brojeve dok ne učita nulu. Program treba ispisati sumu najvećih znamenaka svih učitanih brojeva zapisanih u bazi* `b`*.*

Rješenje.

```
int x, sum, t, max;

sum = 0;
do {
  printf("Unesite broj: "); scanf("%d", &x);
  max = 0;
  t = x;
  while (x > 0) {
    if (max < (x % b)) max = x % b;
    x /= b;
  }
  sum += max;
} while (t);
printf("Suma najvecih znamenaka: %d\n", sum);
```

□

Ovakvim zadacima treba pristupiti oprezno. Primijetimo da tražimo sumu – **jednu za cijeli program** – što znači da nju treba inicijalizirati izvan svih petlji. Tražimo i neki maksimum – ali **po jedan za svaki učitani broj** – pa njega moramo inicijalizirati unutar petlje u kojoj učitavamo brojeve! Pogledajmo dvije česte **greške**:

1. Ako sumu inicijaliziramo unutar `do...while` petlje, onda će se ona za svaki učitani broj "vraćati" na nulu, pa će na kraju imati vrijednost jednaku najvećoj znamenci posljednjeg učitanog broja (a to je 0):

```
do {
  sum = 0;
  printf("Unesite broj: "); scanf("%d", &x);
  max = 0;
  t = x;
  while (x > 0) {
    if (max < (x % b)) max = x % b;
    x /= b;
```

```
9     }
10    sum += max;
11  } while (t);
```

Isprobajte za $b = 10$ i učitane brojeve 17, 19, 31 i 0 (suma bi trebala biti $7+9+3 = 19$, a ispast će 0).

2. Ako `max` inicijaliziramo izvan `do...while`, onda se najveća znamenka brojeva učitanih nakon prvog neće dobro računati:

```
1  sum = 0;
2  max = 0;
3  do {
4     printf("Unesite broj: "); scanf("%d", &x);
5     t = x;
6     while (x > 0) {
7        if (max < (x % b)) max = x % b;
8        x /= b;
9     }
10    sum += max;
11 } while (t);
```

Isprobajte za $b = 10$ i učitane brojeve 17, 19, 31 i 0. Suma bi opet trebala biti $7 + 9 + 3 = 19$, no umjesto toga program će izračunati najveću znamenku broja 17 i ona će (ispravno) ispasti 7. To će biti vrijednost varijable `max` nakon prvog izvođenja tijela petlje. Zatim će taj 7 (nepromijenjen, jer se `max` postavlja na nulu samo prije tijela `do...while` petlje!) uspoređivati s 1 i 9 (od broja 19) i dobit će (ispravno) najveću znamenku drugog broja: 9. No, za treći broj, uspoređivat će znamenke 3 i 1 s maksimumom, a to će (opet, od prije) biti 9. Kako je su i 3 i 1 manji od 9, vrijednost varijable `max` će i za treći broj ostati 9, pa će ispasti suma $7 + 9 + 9 = 25$ što je pogrešno!

Dodatno, u rješenju ovog zadatka, uvjet u vanjskoj `while()` petlji ovisi o `t`, dok je u prethodnom zadatku ovisio o `x`. Ta razlika proizlazi iz načina na koji te varijable koristimo. Naime, unutarnja petlja dijeli jednu od tih varijabli dok njena vrijednost ne padne na nulu, pa nam ta varijabla više nema nikakvu korisnu vrijednost (nakon izvršavanja petlje, vrijednost je nužno nula!). S druge strane, uvjet u vanjskoj petlji mora ovisiti o originalnom učitanom broju. Zato u prethodnom zadatku u unutrašnjoj petlji "trošimo" pomoćnu varijablu `t`, pa uvjet ovisi o nepromijenjenoj varijabli `x`, dok u ovom zadatku u unutrašnjoj petlji "trošimo" varijablu `x`, pa uvjet ovisi o njenoj početnoj vrijednosti pospremljenoj u varijabli `t`. Naravno, ovi načini su ekvivalentni, dakle oba ispravna.

8.5 Malo o složenosti

Većinu zadataka moguće je riješiti na više načina, od kojih su neki "brži", a neki "sporiji". Složenost algoritama bavi se uspoređivanjem algoritama, pri tome zanemarujući dobitke u brzini koji se mogu dobiti jednostavnom kupnjom bržeg hardwarea. To znači da program koji radi 2 puta brže od drugog programa nije interesantan (u terminima složenosti), dok onaj koji za ulaz $n \in \mathbb{N}$ (što god n bio) radi u vremenu n sekundi umjesto n^2 sekundi smatramo velikim poboljšanjem. Dapače, poboljšanje je i svaki onaj program koji radi u vremenu $c \cdot n$ sekundi, gdje je $c \in \mathbb{R}^+$ neka konstanta, bez obzira na veličinu same konstante c.

Zadatak 8.5.1. *Napišite dio programa koji učitava prirodni broj* `n`*, te ispisuje sve djelitelje broja* `n` *(proizvoljnim redoslijedom, ali svakog točno jednom).*

Rješenje (jednostavno).

```
1 int i, n;
2
3 scanf("%d", &n);
4 for (i = 1; i <= n; i++)
5   if (!(n % i)) printf("%d\n", i);
```

□

U ovom rješenju provjeravamo djeljivost sa svim brojevima od 1 do n. No, ako pogledamo rastav

$$n = p \cdot q,$$

primijetit ćemo da ne može biti $p, q > \frac{n}{2}$, jer bismo onda imali

$$n = p \cdot q > \frac{n^2}{4} \Rightarrow n < 4.$$

Dakle, dana situacija je moguća samo za brojeve 1, 2 i 3. No, to nisu složeni brojevi, tj. djeljivi su samo s 1 i sa samim sobom, pa niti kod njih ograničenje $i \leq \frac{n}{2}$ ne uvodi probleme. Drugim riječima, brži algoritam je:

Rješenje (dvostruko brže).

```
1 int i, n, lmt;
2
3 scanf("%d", &n);
4 lmt = n / 2;
5 if (n > 1)
6   for (i = 1; i <= lmt; i++)
7     if (!(n % i)) printf("%d\n%d\n", i, n / i);
8 printf("%d\n", n);
```

□

Ovaj program će se izvoditi dvostruko brže od prethodnog, no to znači da smo jednako ubrzanje mogli dobiti i kupnjom dvostruko bržeg računala. U terminima složenosti algoritama, algoritmi su jednako složeni (oba su linearni!).

Još gore, neki djelitelji će se ispisati dva puta (npr. 2 i 4 za $n = 16$). To se može riješiti dodatnim uvjetom:

Rješenje (ispravak prethodnog).

```
int i, n, lmt, t;

scanf("%d", &n);
lmt = n / 2;
for (i = 1; i <= lmt; i++)
  if (!(n % i))
    if ((t = n / i) <= lmt)
      printf("%d\n", i);
    else
      printf("%d\n%d\n", i, t);
```

□

U ovim rješenjima smo uveli pomoćnu varijablu `lmt` koja služi tome da se izraz `n / 2` ne računa u svakom koraku petlje, nego samo jednom. Takva uvođenja pomoćnih varijabli su uvijek poželjna, i zbog brzine izvršavanja i zbog preglednosti programa: riječ "lmt" je izabrano kao pokrata engleske riječi *limit*, dok slovo "t" označava privremenu varijablu, engl. *temporary*. Za *limit* nismo uzeli jednoslovnu pokratu jer je slovo "l" presličnо broju 1, pa može zbuniti kod čitanja programa (prilikom učenja ili traženja grešaka).

Dodatni uvjeti znače i dodatno računanje, pa ostaje upitno je li (i koliko) ovakav program uopće brži od prvog rješenja (računalo obavlja različite zadaće različitom brzinom). Zbog toga se, prilikom računanja složenosti algoritama, i inzistira da se algoritmi razlikuju za više od konstantnog faktora.

Ako iskoristimo činjenicu da ne postoji $x \in \mathbb{N}$ takav da je n djeljiv s x i $\frac{n}{2} < x < n$, onda zadatak možemo riješiti i ovako:

Rješenje (poboljšanje prethodnog).

```
int i, n, lmt;

scanf("%d", &n);
lmt = n / 2;
for (i = 1; i <= lmt; i++)
  if (!(n % i))
    printf("%d\n", i);
printf("%d\n", n);
```

□

Postignuto ubrzanje je i dalje dvostruko (dakle, konstantno, tj. neovisno o n).

Primijetimo da iz

$$n = p \cdot q$$

možemo zaključiti da je

$$p \leq \sqrt{n} \text{ ili } q \leq \sqrt{n},$$

što nas navodi na zaključak da je granica (varijabla `l` u programima) mogla biti jednaka $\sqrt{n}$. Takvo rješenje zahtijeva i poziv matematičke funkcije `sqrt()`:

Rješenje (manja složenost).

```
#include <stdio.h>
#include <math.h>

int main() {
  int i, n, lmt;
  scanf("%d", &n);
  lmt = sqrt(n);
  for (i = 1; i <= lmt; i++)
    if (!(n % i))
      printf("%d\n%d\n", i, n / i);
  return 0;
}
```

□

Na početku je bilo potrebno dodati `#include<math.h>` zbog korištenja matematičke funkcije koja radi s realnim vrijednostima. Prilikom compiliranja u komandnoj liniji, u ovom je slučaju nužno dodati i opciju `-lm`.

Ovo rješenje je zaista manje složeno. Na primjer, za $n = 16$ petlja se vrti 4 umjesto 16 (u prvom rješenju) ili 8 puta (u drugom rješenju), što je četverostruko, odnosno dvostruko ubrzanje. No, za $n = 64$, petlja se vrti 8 puta, umjesto starih 64 odnosno 32, što je ubrzanje od 8, odnosno 4 puta. Dakle, dobiveno ubrzanje raste s veličinom ulaza n, pa je ovo rješenje zaista manje složeno!

Granicu $\sqrt{n}$ ne možemo dodatno smanjiti, što se pokazuje jednostavnim traženjem protuprimjera (npr. za $n = 9$ moramo izvršiti provjere od 1 do 3).

Zadatak 8.5.2. *Modificirajte prethodni program tako da ne koristi funkciju* `sqrt()`*, ali da i dalje radi provjere samo za* $i \leq \sqrt{n}$.

Uputa. Modificirajte uvjet petlje. □

Zadatak 8.5.3. *Napišite dio programa koji za zadani prirodni broj* `n` *ispisuje sve njegove proste djelitelje (svakog točno jednom).*

Rješenje. Naizgled jednostavan način za napisati ovaj program je traženje svih djelitelja (kao u prethodnim zadacima), te provjera za svakoga je li prost ili nije. Takvo rješenje ostavljamo čitateljima za vježbu.

Umjesto opisanog pristupa, možemo tražiti najmanji broj x (veći od 1) s kojim je n djeljiv. Takav broj nužno mora biti prost, jer kad bi postojali $a > 1$ i $b > 1$ takvi da je $x = a \cdot b$, onda bi nužno bilo $a, b < x$ i n bi bio djeljiv s a i b, što je kontradiktorno pretpostavci da je x najmanji takav broj.

Idući korak je dijeljenje n s pronađenim x dok god je to moguće. Nakon toga se vraćamo na traženje idućeg najmanjeg djelitelja dobivenog broja. To ponavljamo dok god ima smisla tražiti njegove djelitelje (tj. dok god je $n > 1$).

```
int x = 2;
while (n > 1) {
  if (!(n % x)) {
    printf("%d\n", x);
    while (!(n % x)) n /= x;
  }
  x++;
}
```

□

Sljedeća modifikacija zadatka, iako naizgled kompliciranija, zapravo ima jednostavnije rješenje:

Zadatak 8.5.4. *Napišite dio programa koji za zadani prirodni broj* `n` *ispisuje sve njegove proste djelitelje (svakog točno onoliko puta koliko se pojavljuje u rastavu broja* `n` *na proste faktore).*

Rješenje.

```
int x = 2;
while (n > 1) {
  while (!(n % x)) {
    printf("%d\n", x);
    n /= x;
  }
  x++;
}
```

□

Zadatak 8.5.5. *Napišite dio programa koji za zadani prirodni broj* `n` *ispisuje sve njegove proste djelitelje (svakog točno jednom), te uz svakoga napiše koliko puta se pojavljuje u rastavu broja* `n` *na proste faktore. Npr. za* $n = 96 = 2^5 \cdot 3^1$ *program treba ispisati:*

```
2^5
3^1
```

Zadatak 8.5.6. *Napišite dio programa koji učitava realni broj* `x` *i prirodni broj* `n`*, te ispisuje koliko je* x^n*.*

Rješenje.

```
int i, n;
double x, res = 1;

printf("Unesite n: "); scanf("%d", &n);
printf("Unesite x: "); scanf("%lg", &x);
for (i = 0; i < n; i++) res *= x;
printf("(%g)^%d = %g", x, n, res);
```

□

Primijetimo da izraz `res *= n` izvršavamo n puta, što znači da je složenost programa linearna. Može li bolje?

Odgovor je (krajnje (ne)očekivnano): može. Primijenimo uzastopno kvadriranje. Na primjer:

$$x^8 = ((x^2)^2)^2, \text{ tj. } x^2 = x \cdot x,\ x^4 = x^2 \cdot x^2,\ x^8 = x^4 \cdot x^4,$$

pa imamo samo 3 množenja (umjesto 8). Na taj način možemo dobiti x^n ako je n potencija od 2. Što ako nije?

Pogledajmo sljedeće primjere:

$$n = 9 \Rightarrow n = 8 + 1 \Rightarrow x = x^8 \cdot x^1$$
$$n = 12 \Rightarrow n = 8 + 4 \Rightarrow x = x^8 \cdot x^4$$

Dakle, broj n treba prikazati kao zbroj nekih potencija broja 2. Možemo li to izvesti za svaki prirodni n?

Naravno. Upravo takav rastav čini prikaz broja u binarnoj bazi! Dakle, broj n treba prikazati u binarnom sustavu. Za prethodne primjere:

$$(9)_{10} = \overset{3\,2\,1\,0}{(1001)_2} \Rightarrow x^9 = x^{2^3} \cdot x^{2^0}$$
$$(12)_{10} = \overset{3\,2\,1\,0}{(1100)_2} \Rightarrow x^{12} = x^{2^3} \cdot x^{2^2}$$

Vidjeli smo kako se prebrojavaju i ispisuju znamenke. Sada samo, umjesto ispisivanja binarnih znamenki, treba te znamenke iskoristiti kao uvjet u množenju. Preciznije, kad god je binarna znamenka 1 množimo rezultat s x^k, gdje je x^k odgovarajuća potencija od x (prvo x^1, pa x^2, pa x^4, pa x^8 itd.).

```
int n, on;
double x, res = 1, pot;

```

```
4  printf("Unesite n: "); scanf("%d", &n);
5  printf("Unesite x: "); scanf("%lg", &x);
6  pot = x; on = n;
7
8  while (n > 0) {
9    if (n % 2) res *= pot;
10   pot *= pot;
11   n /= 2;
12 }
13
14 printf("(%g)^%d = %g", x, on, res);
```

U najgorem slučaju (za $n = 2^{\text{nešto}} - 1$) imamo 2 puta više množenja nego n ima znamenaka i točno onoliko dijeljenja koliko n ima binarnih znamenaka, a to je $\log_2 n$ (dokažite!). Dakle, složenost ovakvog potenciranja je **logaritamska**, što je puno bolje od linearne.

8.6 Promjena toka petlje

Ponekad "usred izvršavanja" želimo prekinuti petlju ili "skočiti" natrag na uvjet. Vratimo se na zadatak 8.1.2 gdje je trebalo provjeriti je li zadani broj prost. Petlja kojom smo vršili provjeru izgledala je ovako:

```
5  while (prost && p < x) {
6    if (x % p == 0) prost = 0;
7    p++;
8  }
```

Dakle, u svakom koraku petlje, kao dio uvjeta se provjerava stanje varijable `prost`. Također, nakon što varijabli `prost` zadamo vrijednost 0, program će i dalje izvesti liniju 7 (`p++;`) iako je ona nepotrebna.

Prirodan način za provesti provjeru bi, opisno, bio:

```
5  dok (p nije prevelik) {
6    ako je (x djeljiv s p) {
7      zapamti da x nije prost
8      prekini provjeru
9    }
10   povecaj p za jedan
11 }
```

Petlju prekidamo naredbom `break`. Prikazani opisni kod možemo doslovno prevesti u C liniju po liniju:

```
5  while (p < x) {
6    if (x % p == 0) {
```

```
7       prost = 0;
8       break;
9     }
10    p++;
11  }
```

U trenutku kad se izvrši naredba `break`, program **bezuvjetno** prekida izvršavanje tijela petlje. Izvršavanje programa se nastavlja nakon tijela petlje (na naredbi koja bi se nalazila u liniji 12).

Slično, u zadatku 8.4.11 imali smo petlju:

```
3  do {
4    printf("Unesite broj: "); scanf("%d", &x);
5    sum = 0;
6    if (x) {
7      t = x;
8      while (t > 0) {
9        sum += (t % 7);
10       t /= 7;
11     }
12     printf("Suma znamenaka: %d\n", sum);
13   }
14 } while (x);
```

Bilo bi daleko prirodnije jednostavno prekinuti izvršavanje petlje onda kad znamo da smo gotovi (umjesto da veći dio tijela petlje "zapakiramo" u `if()`):

```
3  do {
4    printf("Unesite broj: "); scanf("%d", &x);
5    sum = 0;
6    if (!x) break;
7    t = x;
8    while (t > 0) {
9      sum += (t % 7);
10     t /= 7;
11   }
12   printf("Suma znamenaka: %d\n", sum);
13 } while (x);
```

Dapače, ako je `x == 0`, petlja će prekinuti odmah u liniji 6. Ako je `x != 0`, onda je uvjet u liniji 13 istinit, pa nema potrebe da ga provjeravamo. To znači da liniju 13 možemo zamijeniti s:

```
13 } while (1);
```

No, sada nam više ne treba varijabla `x`, pa ju ne moramo "spašavati" upotrebom pomoćne varijable `t`. Konačno, cijelu petlju možemo zapisati ovako:

```
3  do {
```

```
4     printf("Unesite broj: "); scanf("%d", &x);
5     sum = 0;
6     if (!x) break;
7     while (x > 0) {
8       sum += (x % 7);
9       x /= 7;
10    }
11    printf("Suma znamenaka: %d\n", sum);
12  } while (1);
```

Naredbu `break` možemo čitati kao "skoči iza tijela petlje".

Na sličan način, naredba `continue` prekida izvršavanje petlje i nastavlja:

- s inkrementom, te provjerom uvjeta (kod `for()` petlje) ili
- s provjerom uvjeta (kod `while()` i `do...while` petlje).

Pogledajmo zadatak:

Zadatak 8.6.1. *Napišite program koji učitava prirodni broj* `n`*, te zatim učitava još* `n` *prirodnih brojeva. Program treba ispisati sumu svih znamenaka parnih učitanih brojeva (neparne jednostavno ignorira) zapisanih u bazi* 10*.*

Rješenje.

```
1  int n, i, x, sum;
2
3  sum = 0;
4  printf("Unesite n: "); scanf("%d", &n);
5  for (i = 0; i < n; i++) {
6    printf("Unesite broj: "); scanf("%d", &x);
7    if (x % 2) continue;
8    while (x > 0) {
9      sum += (x % 10);
10     x /= 10;
11   }
12 }
13 printf("Suma znamenaka: %d\n", sum);
```

□

Očito, problem smo mogli riješiti i pomoću `else`:

```
5  for (i = 0; i < n; i++) {
6    printf("Unesite broj: "); scanf("%d", &x);
7    if (x % 2); else
8      while (x > 0) {
9        sum += (x % 10);
```

```
10        x /= 10;
11      }
12  }
```

ili pomoću negiranog uvjeta:

```
5   for (i = 0; i < n; i++) {
6     printf("Unesite broj: "); scanf("%d", &x);
7     if (!(x % 2))
8       while (x > 0) {
9         sum += (x % 10);
10        x /= 10;
11      }
12  }
```

Naredbe `break` i `continue` se uvijek mogu "zaobići", ali kod složenijih problema to može biti izuzetno kompliciranо (mnogo `if...else` blokova jedan unutar drugog, te dodavanje pomoćnih varijabli). Kompliciranje često dovodi do grešaka, pa se preporuča korištenje ovih naredbi kad god je to prirodno.

Ponekad koristimo "beskonačne petlje". To su, kako naziv kaže, petlje čiji je uvjet uvijek zadovoljen. Kako svaki algoritam mora stati nakon konačno mnogo koraka (prema definiciji algoritma!), takve petlje je nužno prekinuti "na silu", upravo naredbom `break`.

Pogledajmo zadatak sličan zadatku 8.3.1:

Zadatak 8.6.2. *Napišite dio programa koji učitava prirodne brojeve i ispisuje njihov produkt. Učitavanje treba prekinuti kad korisnik upiše vrijednost* 0.

Rješenje.

```
1   int prod = 1, x;
2
3   do {
4     printf("Unesite broj: ");
5     scanf("%d", &x);
6     if (x) prod *= x;
7   } while (x != 0);
8
9   printf("Produkt ucitanih brojeva je %d.\n", prod);
```

Petlju smo mogli i drugačije napisati. Najprije opisno:

```
3   radi {
4     ucitaj x
5     ako je x jednak nuli, prekini petlju
6     povecaj produkt
7   };
```

Prisjetimo se: istina je svaka vrijednost različita od nule. Obično se koristi vrijednost 1.

C-ovski, prikazani opisni kod bi izgledao ovako:

```
3 do {
4   printf("Unesite broj: "); scanf("%d", &x);
5   if (x == 0) break;
6   prod *= x;
7 } while (1);
```

ili

```
3 while (1) {
4   printf("Unesite broj: "); scanf("%d", &x);
5   if (x == 0) break;
6   prod *= x;
7 }
```

ili

```
3 for ( ; 1; ) {
4   printf("Unesite broj: "); scanf("%d", &x);
5   if (x == 0) break;
6   prod *= x;
7 }
```

Ovdje to ne izgleda naročito korisno, ali kod složenih uvjeta može biti jako praktično. Npr. ako imamo nekoliko uvjeta, "razbacanih" po tijelu petlje, onda ih je teško objediniti, pa je lakše svakome zadati da prekine petlju, a onda sam uvjet petlje "izostaviti" (tj. staviti da je uvijek istinit). □

Ako uvjet `for()`-petlje nije naveden, smatra se da je on istinit. Dakle, liniju 3 iz zadnjeg prikazanog programskog isječka možemo jednostavnije zapisati ovako

```
3 for ( ; ; ) {
```

Napomena 8.6.1. *Svaku petlju obavezno treba nekako prekinuti! Ako uvjet ne osigurava prekidanje petlje za sve moguće početne vrijednosti varijabli, onda prekidanje izvršavanja petlje* ***nužno*** *treba osigurati naredbom* `break`*.*

Naredbe `break` i `continue` utječu samo na izvršavanje najdublje petlje u kojoj se nalaze.

Primjer 8.6.1. *Programski kod*

```
1 for (i = 0; i < 3; i++) {
2   for (j = i + 1; j < 999; j++) {
3     printf("Unutrasnja petlja: i = %d, j = %d\n",i,j);
4     break;
5   }
6   printf("Vanjska petlja: i = %d\n", i);
7 }
```

će ispisati:

```
Unutrasnja petlja: i = 0, j = 1
Vanjska petlja: i = 0
Unutrasnja petlja: i = 1, j = 2
Vanjska petlja: i = 1
Unutrasnja petlja: i = 2, j = 3
Vanjska petlja: i = 2
```

Dakle, tijelo unutrašnje `for` *petlje (linije* 3 *i* 4*) će se izvršiti točno jednom za svaki* `i`*, jer će naredba* `break` *prekinuti unutrašnju petlju odmah nakon ispisa vrijednosti. S druge strane, linija* 6 *će se izvršiti za svaki* `i` *jer* `break` *"skače iza (unutrašnje) petlje", pa se izvršavanje programa (nakon* `break`*) nastavlja upravo na liniji* 6*.*

Zadatak 8.6.3. *Napišite program koji učitava prirodne brojeve dok ne učita nulu. Program treba ispisati produkt znamenaka (u bazi* 8*) svih učitanih prostih brojeva.*

Uputa. Petlju je dobro prekinuti pomoću `break` odmah nakon učitavanja broja ako je broj jednak nuli. Zatim treba provjeriti je li broj prost. Ako nije, treba se vratiti (pomoću `continue`) na početak petlje. Očito, sva potrebna prekidanja petlje će se izvršiti pomoću `break`, pa uvjet petlje može uvijek biti istinit. □

Zadatak 8.6.4. *Prethodni zadatak riješite bez* `break` *i* `continue`*.*

Zadatak 8.6.5. *Napišite program koji učitava prirodni broj* `n`*, te pomoću zvjezdica (*`"*"`*) iscrtava pravokutni trokut katete duljine* `n` *znakova. Na primjer, za* `n = 4` *treba ispisati:*

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

Rješenje.

```
1 int n, i, j;
2
3 printf("Unesite n: "); scanf("%d", &n);
4
5 for (i = 0; i < n; i++) {
6   for (j = 0; j <= i; j++)
7     printf("*");
8   printf("\n");
9 }
```

□

Zadatak 8.6.6. *Napišite program koji učitava prirodni broj* `n`*, te pomoću zvjezdica (“*`*`*”) iscrtava pravokutni trokut kojem je vertikalna kateta duljine* `n` *znakova, a horizontalna duljine* `2 * n - 1` *znak. Na primjer, za* `n = 4` *treba ispisati:*

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

Zadatak 8.6.7 (Šlag na kraju)**.** *Napišite program koji učitava prirodni broj* `n`*, te pomoću zvjezdica (“*`*`*”) iscrtava krug radijusa* `n` *znakova.*

Poglavlje 9

Višestruko grananje

Ponekad imamo više uvjeta koji ovise o istom izrazu. Tada ulančavanje `if()`-ova ispada nespretno i nezgodno, pogotovo ako blokovi koji se izvršavaju moraju imati i neke zajedničke dijelove. Zbog toga pribjegavamo upotrebi višestrukog grananja.

9.1 Naredba `switch()`

Primjer 9.1.1. *Napišite dio programa koji za zadani prirodni broj* $n \in \{1, 2, \ldots, 365\}$ *ispisuje na koji dan u tjednu pada* n*-ti dan u godini (pretpostavite da godina počinje ponedjeljkom).*

Klasično rješenje, pomoću `if()` *bi bilo:*

```
if (n % 7 == 0) printf("Nedjelja\n"); else
if (n % 7 == 1) printf("Ponedjeljak\n"); else
if (n % 7 == 2) printf("Utorak\n"); else
if (n % 7 == 3) printf("Srijeda\n"); else
if (n % 7 == 4) printf("Cetvrtak\n"); else
if (n % 7 == 5) printf("Petak\n"); else
  printf("Subota\n");
```

ili, da se smanji količina računanja:

```
int pom = n % 7;
if (pom == 0) printf("Nedjelja\n"); else
if (pom == 1) printf("Ponedjeljak\n"); else
if (pom == 2) printf("Utorak\n"); else
if (pom == 3) printf("Srijeda\n"); else
if (pom == 4) printf("Cetvrtak\n"); else
if (pom == 5) printf("Petak\n"); else
  printf("Subota\n");
```

Pomoću `switch()`*, rješenje izgleda ovako:*

```
switch (n % 7) {
  case 0: printf("Nedjelja\n");
          break;
  case 1: printf("Ponedjeljak\n");
          break;
  case 2: printf("Utorak\n");
          break;
  case 3: printf("Srijeda\n");
          break;
  case 4: printf("Cetvrtak\n");
          break;
  case 5: printf("Petak\n");
          break;
  default: printf("Subota\n");
}
```

Prilikom izvršavanja, računalo ulazi u prvu granu koja ima vrijednost jednaku vrijednosti izraza, te nastavlja dalje. Ukoliko želimo prekinuti izvršavanje naredbi unutra `switch()` (tj. želimo "izaći" iz `switch()`), zadajemo narebu `break` (jednako kao kod petlji). Grana `default` (nije obavezna!) se izvršava ako niti jedan `case` nema vrijednost kao zadani izraz (nešto kao `else` kod `if()` petlji).

Napomena 9.1.1. *Ako ne zadamo* `break`*, izvršavanje prelazi u iduću granu!*

Primjer 9.1.2. *Prethodni primjer, ali bez upotrebe* `break`*:*

```
switch (n % 7) {
  case 0: printf("Nedjelja\n");
  case 1: printf("Ponedjeljak\n");
  case 2: printf("Utorak\n");
  case 3: printf("Srijeda\n");
  case 4: printf("Cetvrtak\n");
  case 5: printf("Petak\n");
  default: printf("Subota\n");
}
```

će za $n = 4$ *ispisati:*

```
Cetvrtak
Petak
Subota
```

Zadatak 9.1.1. *Napišite dio programa koji za zadane prirodne brojeve* $p \in \{1, 2, \ldots, 7\}$ *i* $n \in \{1, 2, \ldots, 365 - p\}$ *ispisuje na koji dan u tjednu pada* n*-ti dan u godini (pretpostavite da godina počinje ponedjeljkom ako je* $p = 1$*, utorkom ako je* $p = 2$*, itd.). Ukoliko je dan koji ispisujete utorak ili petak, program treba ispisati i idući dan.*

Rješenje.

```
switch ((n + p - 1) % 7) {
  case 0: printf("Nedjelja\n");
          break;
  case 1: printf("Ponedjeljak\n");
          break;
  case 2: printf("Utorak\n");
  case 3: printf("Srijeda\n");
          break;
  case 4: printf("Cetvrtak\n");
          break;
  case 5: printf("Petak\n");
  default: printf("Subota\n");
}
```

□

Zadatak 9.1.2. *Riješite prethodni program bez upotrebe* `switch()`.

Ukoliko dvije grane rade točno jednak posao, nabrajamo ih jednu iza druge.

Zadatak 9.1.3. *Napišite dio programa koji za zadani prirodni broj n ispisuje je li zadnja dekadska znamenka broja* 3^n *prost broj ili ne.*

Rješenje.

```
/* Racunanje n-te potencije bro\-ja 3 */
pot = 1;
while (n--) pot *= 3;
/* Da li je zadnja znamenka prost broj */
printf("Zadnja znamenka ");
switch (pot % 10) {
  case 2:
  case 3:
  case 5:
  case 7: printf("je");
          break;
  default: printf("nije");
}
printf(" prost broj.\n");
```

ili, kraće:

```
/* Racunanje n-te potencije bro\-ja 3 */
pot = 1;
while (n--) pot *= 3;
/* Da li je zadnja znamenka prost broj */
printf("Zadnja znamenka ");
```

```
6  switch (pot % 10) {
7    case 1: case 4: case 6: case 8:
8    case 9: printf("ni");
9  }
10 printf("je prost broj.\n");
```

□

Kao što vidimo, "prazne grane" možemo pobrojati u jedan red.

Napomena 9.1.2. *U prethodnom programskom isječku, računanje zadnje znamenke n-te potencije broja* 3 *mogli smo napisati i točnije:*

```
1  pot = 1;
2  while (n--) pot = (3 * pot) % 10;
```

Zašto je ovo "točnije", objašnjeno je na predavanjima (overflow).

Zadatak 9.1.4. *Napišite sljedeći dio koda bez upotrebe* `switch()` *(tj. samo upotrebom* `if()`*).*

```
1  switch (a) {
2    case 1: printf("A");
3    case 2: printf("B");
4            break;
5    case 3: if (x % 2)
6              printf("1");
7            else
8              printf("?");
9    case 4: break;
10   case 5: printf("C");
11   default: printf("D");
12 }
```

Poglavlje 10

Funkcije

Često neki dio kôda treba izvršiti više puta, a jedina razlika u tim izvršavanjima su vrijednosti na kojima se taj kôd izvršava. Kako ne bismo dva puta pisali isto, eventualno uz mijenjanje naziva varijabli (što lako može dovesti do grešaka), koristimo funkcije – dijelove koda izdvojene u zasebne cjeline.

Funkcije mogu pozivati jedne druge, pa čak i same sebe. Ove druge zovu se rekurzivne funkcije i njih ćemo obraditi u drugom semestru.

10.1 Jednostavne funkcije

Zadatak 10.1.1 (Euklidov algoritam)**.** *Napišite program koji učitava dva cijela broja* $x, y \in \mathbb{Z}$*. Program treba ispisati* $\mathrm{GCD}(x, y)$ *(najveći zajednički djelitelj brojeva* x *i* y*).*

Rješenje. Zadatak rješavamo Euklidovim algoritmom. On se svodi na dijeljenje većeg broja s manjim dok ostatak pri dijeljenju ne postane jednak nuli. Tada je rezultat dijeljenja upravo najveća zajednička mjera početnih brojeva (dokažite!).

```
1  #include <stdio.h>
2
3  int main(void) {
4    int x, y;
5
6    printf("Ucitajte x: "); scanf("%d", &x);
7    printf("Ucitajte y: "); scanf("%d", &y);
8
9    printf("GCD(%d, %d) = ", x, y);
10
11   if (x < 0) x = -x;
12   if (y < 0) y = -y;
13   if (x < y) { int t = x; x = y; y = t; }
14
```

```
15    while (y) {
16      int t = x % y;
17      x = y;
18      y = t;
19    }
20
21    printf("%d\n", x);
22
23    return 0;
24  }
```

□

Zadatak 10.1.2. *Bez upotrebe nizova napišite program koji učitava različite cijele brojeve $a, b, c \in \mathbb{Z}$. Program treba ispisati brojeve x, y i $\mathrm{GCD}(x, y)$, pri čemu je*

$$x = \min_{p,q \in \{a,b,c\}} \mathrm{GCD}(p, q) \quad \text{i} \quad y = \max_{p,q \in \{a,b,c\}} \mathrm{GCD}(p, q).$$

Pokušajte zadatak riješiti bez upotrebe funkcija (potprograma), replicirajući rješenje prethodnog zadatka.

Rješenje. Idealno, rješenje zadatka bi izgledalo ovako (skica):

```
x = y = gcd(a,b);
p = gcd(a,c);
if (p < x) x = p;
if (p > y) y = p;
p = gcd(b,c);
if (p < x) x = p;
if (p > y) y = p;
printf("x = %d, y = %d, gcd(%d, %d) = %d\n",
  x, y, x, y, gcd(x, y));
```

No, u C-u ne postoji funkcija `gcd()`, što znači da ili moramo višestruko prepisati rješenje prethodnog zadatka (uz prikladnu promjenu vrijednosti varijabli) ili moramo sami definirati funkciju `gcd()`. Mi ćemo, naravno, složiti svoju funkciju:

```
#include <stdio.h>

int gcd(int x, int y) {
  if (x < 0) x = -x;
  if (y < 0) y = -y;
  if (x < y) { int t = x; x = y; y = t; }

  while (y) {
    int t = x % y;
    x = y;
```

```
11      y = t;
12    }
13
14    return x;
15  }
16
17  int main(void) {
18    int a, b, c, p, x, y;
19
20    printf("Ucitajte a: "); scanf("%d", &a);
21    printf("Ucitajte b: "); scanf("%d", &b);
22    printf("Ucitajte c: "); scanf("%d", &c);
23
24    x = y = gcd(a,b);
25    p = gcd(a,c);
26    if (p < x) x = p;
27    if (p > y) y = p;
28    p = gcd(b,c);
29    if (p < x) x = p;
30    if (p > y) y = p;
31    printf("x = %d, y = %d, gcd(%d, %d) = %d\n",
32      x, y, x, y, gcd(x, y));
33
34    return 0;
35  }
```

Funkcija se sastoji od sljedećih dijelova:

- **zaglavlje** (linija 3), u kojem su redom navedeni tip vrijednosti koju funkcija vraća (pišemo ključnu riječ "`void`" ako funkcija ne vraća vrijednost), naziv funkcije, te popis formalnih argumenata (i njihovih tipova) koje funkcija prima. Ako funkcija ne prima nikakve vrijednosti, navode se prazne zagrade (dakle "`ime_funkcije()`").
- **tijelo** (linije 4–14), u kojima (kao i inače u programu) opisujemo što funkcija radi.
- **povrat vrijednosti** (linija 14), kojim prekidamo izvršavanje funkcije i zadajemo koju vrijednost ona vraća. Ovaj dio nije obavezan ako funkcija ne vraća vrijednost. Želimo li negdje prekinuti izvršavanje funkcije koja ne vraća vrijednost, pozivamo samo `return;`.

Varijable `x` i `y` u funkciji *gcd*() su lokalne i **nemaju veze** s istoimenim varijablama u glavnom programu! U njih se spremaju **kopije vrijednosti** varijabli s kojima je funkcija pozvana (npr. kod poziva `gcd(a,c)`, varijabla `x` poprima vrijednost `a`, a varijabla `y` poprima vrijednost `c`). Promjene tih lokalnih varijabli ne utječu na vrijednosti varijabli s kojima je funkcija pozvana (tj. promjena vrijednosti varijabli `x` i `y` neće dovesti do izmjene vrijednosti varijabli `a` i `c`). □

Kao što smo vidjeli u rješenju prethodnog zadatka, potprogramima smanjujemo količinu pisanja i povećavamo preglednost programa, što direktno utječe i na smanjivanje broja grešaka. Rješenje prethodnog zadatka moglo se dodatno raščlaniti na funkcije, te (uz dodatni zahvat na ispisu[1]) skratit na sljedeći oblik:

```
#include <stdio.h>

int gcd(int x, int y) {
  if (x < 0) x = -x;
  if (y < 0) y = -y;
  if (x < y) { int t = x; x = y; y = t; }

  while (y) {
    int t = x % y;
    x = y;
    y = t;
  }

  return x;
}

int min(int x, int y) {
  return (x < y ? x : y);
}

int max(int x, int y) {
  return (x > y ? x : y);
}

int main(void) {
  int a, b, c, p, x, y;

  printf("Ucitajte a: "); scanf("%d", &a);
  printf("Ucitajte b: "); scanf("%d", &b);
  printf("Ucitajte c: "); scanf("%d", &c);

  x = y = gcd(a,b);
  p = gcd(a,c); x = min(x, p); y = max(y, p);
  p = gcd(b,c); x = min(x, p); y = max(y, p);
  printf("x = %d, y = %d, gcd(%1$d, %2$d) = %d\n",
    x, y, gcd(x, y));
```

[1] Za one koji žele znati više: format `%m$f`, pri čemu je `m` broj, a `f` neki od uobičajenih formata (`d`, `e`, `3x` i sl.), odgovara formatu `%f`, ali kao vrijednost uzima m-ti parametar. Zbog toga prikazani poziv funkcije `printf()` ispisuje vrijednosti varijabli `x` i `y` na dva mjesta (na mjesta prva dva `%d` i na mjesta gdje je zadano `%1$d` i `%2$d`). Ovo radi na Linuxu i drugim POSIX-kompatibilnim sustavima, ali ne i na Microsoft Windows (gdje radi pod Cygwinom i sličnim podsustavima).

```
37
38   return 0;
39 }
```

Da je riječ o zaista kraćem rješenju, možete se uvjeriti tako da pokušate riješiti modifikaciju zadatka koja radi s 5 varijabli (umjesto njih 3).

Pogledajte strukturu glavnog programa. U C-u, on je samo jedna od funkcija koja se od ostalih razlikuje isključivo po specijalnom nazivu (`main()`)!

Zadatak 10.1.3. *Napišite funkciju koja kao argument uzima jedan prirodni broj $n \in \mathbb{N}$, a kao povratnu vrijednost vraća $n! := 1 \cdot 2 \cdot \cdots \cdot n$.*

Rješenje. Funkcija prima argumente i vraća vrijednost. Zbog toga ona ne mora ništa niti učitavati niti ispisivati, ali ju ipak možemo smatrati algoritmom!

```
1 long int fakt(int n) {
2   int i;
3   long int res = 1;
4
5   for (i = 2; i <= n; i++) res *= i;
6
7   return res;
8 }
```

Funkciju smo mogli i kraće napisati, koristeći činjenicu da je varijabla `n` lokalna, tj. da njenom promjenom nećemo izazvati promjenu varijable koja je pri pozivu funkcije navedena kao argument:

```
1 long int fakt(int n) {
2   long int res = 1;
3
4   while (n > 1) res *= (n--);
5
6   return res;
7 }
```

□

Napomena 10.1.1 (Česte greške)**.**

1. *Ako funkcija prima neku vrijednost, onda tu vrijednost ne smijete u funkciji učitavati, jer učitavanjem gubite zadanu vrijednost argumenta. Dakle, ovo je* ***KRIVO****:*

```
1 long int fakt(int n) {
2   long int res = 1;
3
4   scanf("%d", &n);
5
```

```
6   while (n > 1) res *= (n--);
7
8   return res;
9 }
```

2. *Iza* `return` *nema smisla ništa pisati jer se to neće izvršiti (*`return` *prekida izvršavanje funkcije). Zbog toga, u sljedećem kodu nikad neće doći do ispisa:*

```
1 int f(int x) {
2   return 2 * x;
3   printf("%d\n", x);
4 }
```

Naravno, možete pomoću `return` *prekinuti izvršavanje funkcije samo u nekim slučajevima. Na primjer, iduća funkcija vraća najmanji parni broj veći ili jednak x:*

```
1 int f(int x) {
2   if (x % 2 == 0) return x;
3   x++;
4   return x;
5 }
```

Istu stvar rade i sljedeće dvije funkcije:

```
1 int g(int x) {
2   if (x % 2 == 0) return x;
3   return x + 1;
4 }
5 int h(int x) {
6   return (x % 2 ? x + 1 : x);
7 }
```

Dakle, naredbe iza `return` *ima smisla zadavati samo ako je do njih moguće doći bez izvršavanja samog* `return`*-a (npr. u različitim granama* `if()`*-a).*

Zadatak 10.1.4. *Napišite funkciju koja kao argument uzima jedan prirodni broj $n \in \mathbb{N}$, a kao povratnu vrijednost vraća*

$$(2n)!! := 2 \cdot 4 \cdot 6 \cdot \cdots \cdot (2n) = 2^n n!.$$

Zadatak 10.1.5. *Napišite funkciju koja kao argument uzima jedan prirodni broj $n \in \mathbb{N}$, a kao povratnu vrijednost vraća*

$$(2n-1)!! := 1 \cdot 3 \cdot 5 \cdot \cdots \cdot (2n-1).$$

Zadatak 10.1.6. *Napišite funkciju koja kao argument uzima jedan prirodni broj $n \in \mathbb{N}$, a kao povratnu vrijednost vraća*

$$n!! := n \cdot (n-2) \cdot (n-4) \cdot \cdots = \prod_{0 \le i < \frac{n}{2}} (n-2i).$$

Zadatak 10.1.7. *Napišite funkciju koja prima jedan cjelobrojni parametar $x \in \mathbb{Z}$, te vraća njegovu apsolutnu vrijednost.*

Zadatak 10.1.8. *Napišite funkciju koja prima jedan cjelobrojni parametar $x \in \mathbb{Z}$, te vraća 1 ako je $|x|$ prost, odnosno 0 ako $|x|$ nije prost.*

Rješenje.

```
1 int abs_is_prime(int x) {
2   int i = 2;
3   if (x < 0) x = -x;
4   if (x <= 1) return 0;
5   while (i < x)
6     if (!(x % i++)) return 0;
7   return 1;
8 }
```

U liniji 6, ako je `x` djeljiv s `i`, prekidamo izvršavanje funkcije i ona vraća 0. U protivnom, `while()` petlja nastavlja s izvršavanjem. Vrijednost varijable `i` povećava se za jedan, ali se djeljivost provjerava sa starom vrijednošću od `i` (jer `i` povećavamo post-inkrement operatorom). Dakle, u prvom koraku `i` poprimi vrijednost 3, ali se provjerava djeljivost `x`-a s 2, u drugom koraku `i` postaje 4, ali se djeljivost `x`-a s 3 itd.

Za učitani broj `x`, funkciju možemo pozvati na sljedeći način:

```
1   printf("Broj |%d| %sje prost.\n",
2     x, abs_is_prime(x) ? "" : "ni");
```

□

Zadatak 10.1.9. *Napišite funkciju koja prima jedan prirodni parametar $n \in \mathbb{N}$, te ispisuje sve njegove proste faktore i to svakog onoliko puta kolika mu je kratnost, tj. za*

$$n = \prod_i p_i^{n_i}, \quad p_i \text{ prost}, \quad n_i \in \mathbb{N}$$

treba svaki p_i ispisati n_i puta. Redoslijed ispisa nije bitan.

Rješenje. Funkcija ne vraća nikakvu vrijednost, pa je njen tip `void`. Osnovni algoritam je već prikazan u zadatku 8.5.3, pa ćemo ovdje samo podsjetiti kako on radi.

Tvrdnja: Neka je p_i najmanji broj (strogo veći od 1) takav da je n djeljiv s p_i, tada je p_i prost (dokažite!).

Korištenjem te tvrdnje možemo složiti izuzetno brz algoritam za rastav broja na proste faktore. Najprije inicijaliziramo pomoćnu varijablu `p` na najmanju vrijednost strogo veću od 1 (tj. na vrijednost 2). Zatim provjeravamo djeljivost `n` s `p`. Dok god je `n` djeljiv s `p`, ispisujemo `p`, te `n` dijelimo s `p`. Ako `n` (više) nije djeljiv s `p`, povećavamo `p` za jedan, kako bismo u idućem koraku petlje probali za idući broj. Petlju vrtimo dok god `n` ima prostih faktora, tj. dok god je strogo veći od 1.

```
void print_primes(int n) {
  int p = 2;
  while (n > 1) {
    while (!(n % p)) {
      printf("%d\n", p);
      n /= p;
    }
    p++;
  }
}
```

□

Zadatak 10.1.10. *Napišite funkciju koja prima jedan prirodni parametar $n \in \mathbb{N}$, te ispisuje sve njegove proste faktore, svakog točno jednom.*

Zadatak 10.1.11. *Napišite funkciju koja prima jedan prirodni parametar $n \in \mathbb{N}$, te ispisuje sve njegove proste faktore i uz svakog njegovu kratnost. Funkcija treba vratiti broj različitih prostih faktora koje sadrži broj n.*

10.2 Varijabilni argumenti funkcija

Kod funkcija u C-u, parametri se prenose "po vrijednosti". To znači da funkcija ima lokalnu varijablu u kojoj se nalazi kopija vrijednosti s kojom se funkcija poziva. Zbog toga se promjene argumenata u funkciji ne odražavaju na vrijednosti varijabli s kojima je funkcija pozvana.

Primjer 10.2.1.

```
#include <stdio.h>

void f(int a) {
  printf("Funkcija 1: a = %d\n", a);
  a++;
  printf("Funkcija 2: a = %d\n", a);
}

int main(void) {
  int x = 1;

  printf("Gl. prog. 1: x = %d\n", x);
  f(x);
  printf("Gl. prog. 2: x = %d\n", x);

  return 0;
}
```

No, što se događa ako u funkciju proslijedimo adresu varijable i onda pristupamo direktno toj adresi?

Primjer 10.2.2.

```
#include <stdio.h>

void f(int *a) {
  printf("Funkcija 1: *a = %d\n", *a);
  (*a)++;
  printf("Funkcija 2: *a = %d\n", *a);
}

int main(void) {
  int x = 1;

  printf("Gl. prog. 1: x = %d\n", x);
  f(&x);
  printf("Gl. prog. 2: x = %d\n", x);

  return 0;
}
```

Primjer 10.2.1 ispisat će

```
Gl. prog. 1: x = 1
Funkcija 1: a = 1
Funkcija 2: a = 2
Gl. prog. 1: x = 1
```

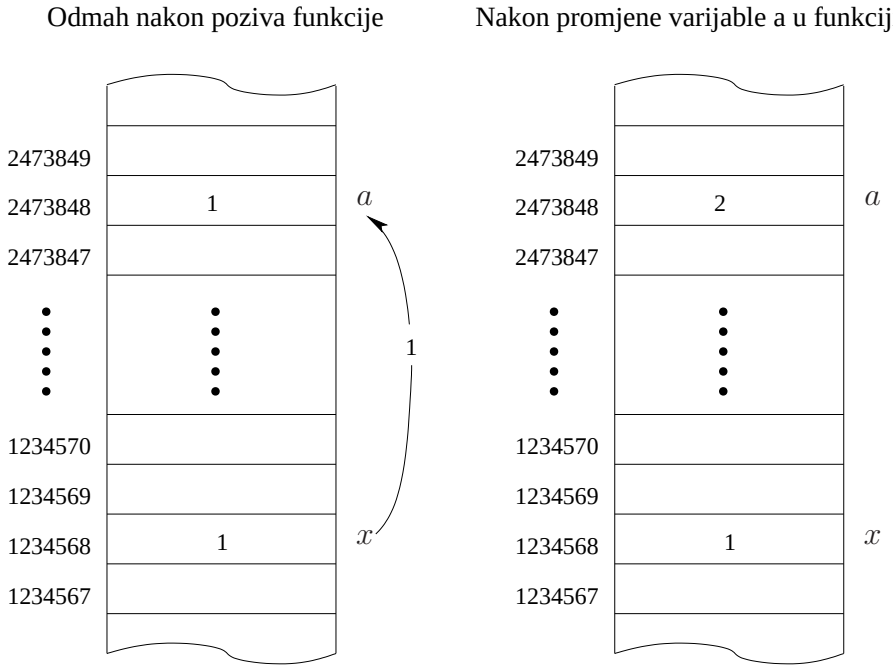
dok će primjer 10.2.2 ispisati

```
Gl. prog. 1: x = 1
Funkcija 1: *a = 1
Funkcija 2: *a = 2
Gl. prog. 1: x = 2
```

Iako radimo naizgled istu stvar, promijenili smo varijablu koja je bila pozivni argument funkcije. Zašto? Pogledajmo što se događa u memoriji (slike 10.1 i 10.2).

Obje slike prikazuju dijelove memorije u kojima se nalaze varijable `x` i `a`. Lijevo od memorije su popisane memorijske lokacije (ovdje *ad hoc* izmišljene; u stvarnosti ih dodjeljuje računalo, bez utjecaja korisnika na stvarne vrijednosti). Bitno je da te lokacije idu po redu, jedna za drugom.

U primjeru 10.2.1, varijabla `a` je tipa `int` i prilikom poziva funkcije poprima vrijednost parametra `x`. To znači da je vrijabla `a` kopija varijable `x`. Evaluiranjem izraza `a++` mi



Slika 10.1: Memorija u primjeru 10.2.1

mijenjamo vrijednost te kopije, no varijabla `x` ostaje nepromijenjena. Naravno, prilikom izlaska iz funkcije, varijabla `a` se briše i promjena "nestaje".

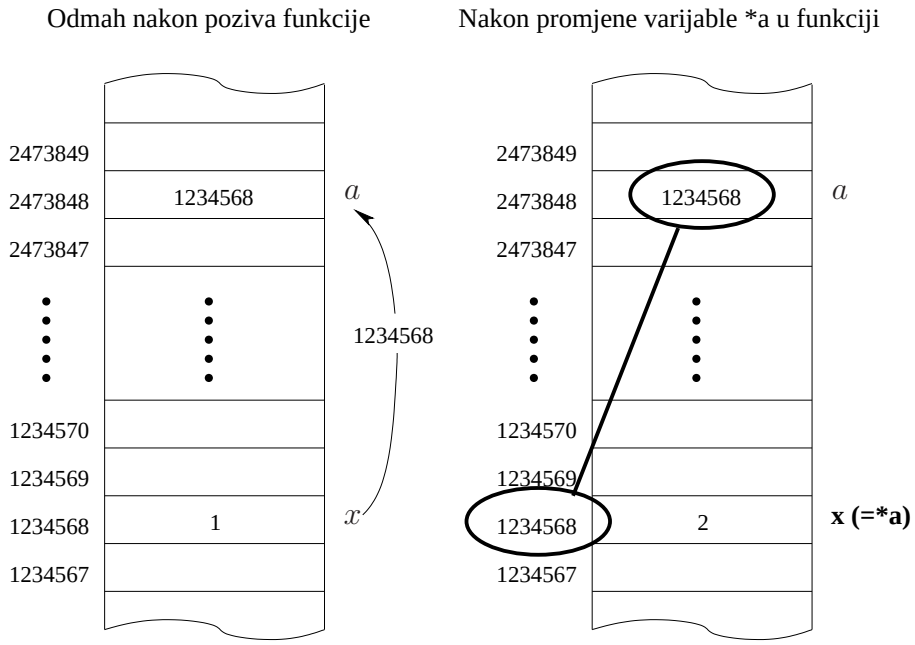
S druge strane, u primjeru 10.2.2, varijabla `a` je tipa `int*` (pointer na `int`, odnosno adresa memorijske lokacije na kojoj se nalazi neka vrijednost tipa `int`). Slično prethodnom primjeru, prilikom poziva funkcije varijabla `a` poprima vrijednost parametra, no ovaj put to je `&x`. Podsjetimo se: operator "`&`" vraća adresu varijable koja se nalazi iza njega, pa `&x` vraća adresu varijable `x`. Zbog toga, na slici 10.2 varijabla `a` ima vrijednost 1234568, što je adresa varijable `x`. Analogno tome, izraz `*a` označava memorijsku lokaciju na adresi `a`, pa promjenom vrijednosti `*a` (izraz `(*a)++`) efektivno mijenjamo vrijednost varijable `x`.

Napomena 10.2.1. *Riječ je o jednoj od najvažnijih stvari u C-u. Proučite dobro, dok niste apsolutno sigurni da Vam je potpuno jasno, te* ***na računalu*** *riješite sve zadatke iz ovog poglavlja!*

Zadatak 10.2.1. *Što ispisuje sljedeći program?*

```
#include <stdio.h>

void f(int *a, int b) {
  int *c;
  c = &b;
  (*a)++; b++; (*c)++;
}
```



Slika 10.2: Memorija u primjeru 10.2.2

```
8
9  int main(void) {
10   int a = 1, b = 10, c = 100;
11
12   printf("a = %d, b = %d, c = %d\n", a, b, c);
13   f(&a, b);
14   printf("a = %d, b = %d, c = %d\n", a, b, c);
15
16   return 0;
17 }
```

Rješenje. Program ispisuje

```
a = 1, b = 10, c = 100
a = 2, b = 10, c = 100
```

Iako imamo izraz `(*c)++` koji je sličan izrazu `(*a)++`, on neće izmijeniti niti varijablu c iz glavnog programa (jer s njom nema veze), niti varijablu `b` iz glavnog programa, jer je `c` pointer na varijablu `b` iz funkcije, a ona sama je kopija varijable `b` iz glavnog programa! Zbog toga izraz `(*c)++` radi isto što i `b++` – mijenja lokalnu kopiju varijable `b` iz glavnog programa. □

Zadatak 10.2.2. *Napišite funkciju koja omogućuje zamjenu vrijednosti dva realna broja.*

Napomena 10.2.2 (Česta greška)**.** *Dio programa koji* ***ne zamjenjuje*** *dvije varijable, ali se često – greškom – pojavljuje u kolokvijima i zadaćama:*

```
x = y;
y = x;
```

U trenutku kad je računalo izvršilo naredbu `x = y`*, obje varijable će poprimiti vrijednost koju je na početku imala varijabla* `y`*, pa će originalna vrijednost varijable* `x` *biti izgubljena. Naredba* `y = x` *pridružuje varijabli* `y` ***novu*** *vrijednost varijable* `x`*, što je vrijednost koju* `y` *ionako već ima od prije.*

*Postoje jezici (npr. ASM – **ne** Assembler!) u kojima prikazana konstrukcija zaista mijenja vrijednosti dvije varijable. No, u "klasičnim" jezicima to ne vrijedi.*

Rješenje (ispravno). Potrebna nam je pomoćna varijabla istog tipa kao `x` i `y` (potrebno ju je deklarirati na početku bloka u kojem se nalazi zamjena!):

```
double pomocna = x;
x = y;
y = pomocna;
```

Funkciju za zamjenu dva realna broja možemo složiti pomoću varijabilnih parametara, pri čemu funkcija ne treba vraćati nikakvu vrijednost (tj. njen povratni tip je `void`):

```
void swap(double *x, double *y) {
  double temp;
  temp = *x;
  *x = *y;
  *y = temp;
}
```

□

Primijetimo da je varijabla `temp` "obični" `double`, tj. nije pointer na `double`. Kad bi ona bila pointer na `double`, značilo bi da – kao i `x` i `y` – pokazuje na neku memorijsku lokaciju u kojoj se nalazi nešto tipa `double`. No, za razliku od `x` i `y`, nemamo ni jednu ćeliju na koju bi `temp` pokazivala, pa bi kod izvršavanja došlo do greške. Više o ovome bit će rečeno u poglavlju o dinamičkim varijablama.

Napomena 10.2.3 (Česta greška)**.** *To što je varijabilni parametar pointer i dalje ne znači da možemo njega mijenjati tako da promjena afektira parametar s kojim je funkcija pozvana. Možemo mijenjati isključivo ono na što varijabilni parametar pokazuje! Na primjer, sljedeća funkcija neće napraviti ništa:*

```
void f(double *x, double *y) {
  double *temp;
  temp = x;
  x = y;
  y = temp;
}
```

Preciznije, ona će zamijeniti privremene pointere `x` *i* `y`*, ali ne i vrijednosti ćelija na koje oni pokazuju. Nakon izvršavanja funkcije, pointeri* `x` *i* `y` *se gube, pa njihova zamjena ne utječe na odgovarajuće varijable iz glavnog programa.*

Zadatak 10.2.3. *Napišite funkciju koja preko varijabilnog parametra poništava pokazivač na cijeli broj, tj. postavlja ga na vrijednost* `NULL`*.*

Rješenje. Princip je isti kao i do sada, ali sintaksa može djelovati zbunjujuće. Ovdje želimo mijenjati varijablu koja je tipa `int*` (pointer na `int`). To znači da varijablini parametar mora biti tipa `int**` (pointer na pointer na `int`).

```
1 void nullify(int **x) {
2   *x = NULL;
3 }
```

□

Zadatak 10.2.4. *Napišite funkciju* `shift()` *koja prima tri cijela broja, te ih cirkularno "pomiče" u desno na način da trećem pridijeli vrijednost drugog, drugom vrijednost prvog i prvom vrijednost trećeg.*

Uputa. Ovo je, u osnovi, malo modificirani `swap()`. □

Zadatak 10.2.5. *Napišite funkciju koja prima tri argumenta: prirodni broj* $n \in \mathbb{N}$*, te pointere* `x` *i* `y` *na prirodne brojeve. Funkcija treba naći najmanji i najveći prosti faktor broja* n*, te ih preko parametara* `x` *i* `y` *vratiti u glavni program. Sama funkcija ne vraća nikakvu vrijednost (tj. ona je tipa* `void`*).*

Zadatak 10.2.6. *Napišite funkciju koja za zadani prirodni broj* $n \in \mathbb{N}$ *vraća sumu i produkt svih prirodnih brojeva strogo manjih od* n*.*

Uputa. Funkcija ne može vratiti više od jedne vrijednosti, pa je zadatak potrebno riješiti pomoću varijablinih parametara. □

Poglavlje 11

Nizovi (polja)

Nizovi (polja ili indeksirane varijable) su varijable koje u sebi sadrže više vrijednosti istog tipa. Deklariramo ih navođenjem tipa vrijednosti i "duljine niza", tj. broja vrijednosti koliko **najviše** možemo pospremiti u niz.

11.1 Uvod

Primjer 11.1.1. *Nizove* `x` *od (najviše)* 17 *cijelih brojeva i* `y` *od (najviše)* 19 *realnih brojeva deklariramo na sljedeći način:*

```
1 int x[17];
2 double y[19];
```

Napomena 11.1.1. *U deklaraciji niza, npr.* `int x[n]`*, duljina (u ovom primjeru* `n`*) mora biti zadana konstanta, a nikako ne varijabla ili izraz čija vrijednost nije jednoznačno određena u trenutku compiliranja!*

Pojedinim vrijednostima pristupamo navođenjem imena varijable i indeksa.

Napomena 11.1.2. *Indeksi "kreću" od nule i moraju biti strogo manji od broja elemenata niza!*

Na primjer, za niz `int x[5]` *indeksi kojima smijemo pristupati su* 0, 1, 2, 3 *i* 4*, ali* ***ne*** *i* 5 *(ili bilo koji broj veći od* 5*). Također, ne smijemo pristupati niti negativnim indeksima.*

Primjer 11.1.2. *Sljedeći dio programa postavlja treći element varijable* `x` *na vrijednost* 13*, ispisuje prvi element varijable* `y`*, te učitava sedmi element varijable* `x`*:*

```
1 x[2] = 13;
2 printf("%g", y[0]);
3 scanf("%d", &x[6]);
```

Nizove možemo deklarirati i tako da ih odmah inicijaliziramo (navedemo početne vrijednosti njihovih elemenata).

Primjer 11.1.3.

```
int a[17] = {1, 3, 7};
int b[] = {1, 3, 7};
```

Varijable `a` *i* `b` *razlikuju se u duljini:* `a` *je niz sa* 17 *elemnata od koji prva tri imaju vrijednosti* 1, 3 *i* 7 *(respektivno), dok su ostale vrijednosti nula;* `b` *je niz od tri elementa (ponovno s vrijednostima* 1, 3 *i* 7 *respektivno). Ovaj primjer identičan je (do na dodanu varijablu* `i`*) sljedećem kodu:*

```
int a[17], b[3], i;
a[0] = 1; a[1] = 3; a[2] = 7;
for (i = 3; i < 17; i++) a[i] = 0;
b[0] = 1; b[1] = 3; b[2] = 7;
```

Napomena 11.1.3. *Ako nizu nije zadana početna vrijednost, njegovi elementi* ***neće*** *imati vrijednost nula, nego će ta vrijednost biti slučajna (kao i kod "običnih" varijabli kad ih neinicijaliziramo)!*

Primjer 11.1.4.

```
int x[17];
int y[17] = {0};
```

Nakon izvršavanja ovog dijela programa, varijabla `x` *će biti niz od* 17 *cijelih brojeva nepoznate vrijednosti, dok će varijabla* `y` *biti niz od* 17 *cijelih brojeva koji svi imaju vrijednost nula.*

Napomena 11.1.4. *Općenito, nizove* ***ne možemo*** *odjednom ispisati ili učitati:*

```
int x[17];
scanf("...", &x);
printf("...", x);
```

nego moramo raditi sa svakim pojedinim članom niza (npr. pomoću `for()`*-petlje). Iznimka su nizovi znakova (tzv. stringovi) koje ćemo naknadno obraditi.*

Zadatak 11.1.1. *Napišite program koji učitava prirodni broj* $n \leq 20$*, te* n *prirodnih brojeva. Nakon učitavanja, brojeve treba ispisati u obrnutom redoslijedu.*

Rješenje. Ovdje ćemo upotrijebiti `unsigned int`, kako bismo pokazali da kod `unsigned` tipova treba dodatno pripaziti.

```
#include <stdio.h>

int main(void) {
  unsigned int n, x[20];
```

```
5   int i;
6   printf("Unesite n: ");
7   scanf("%u", &n);
8   if (n > 0) {
9     for (i = 0; i < n; i++) {
10      printf("Unesite x[%d]: ", i);
11      scanf("%u", &x[i]);
12    }
13    printf("Ispis brojeva u obrnutom redoslijedu:\n");
14    for (i = n - 1; i >= 0; i--)
15      printf("%u\n", x[i]);
16  } else {
17    printf("Odabrali ste unos nula brojeva, ");
18    printf("pa ja nemam sto raditi...\n");
19  }
20  return 0;
21 }
```

□

Napomena 11.1.5. *Da je tip varijable* `i` *u prethodnom programu bio* `unsigned int`*, uvjet* `for()`*-petlje bio bi uvijek istinit. Moguće je i definirati varijablu* `i` *tako da bude tipa* `unsigned int`*, no onda moramo izmijeniti* `for()`*-petlju (naravno, na način da daje jednaki ispis). Na primjer:*

```
14    for (i = 1; i <= n; i++)
15      printf("%u\n", x[n - i]);
```

ili

```
14    for (i = n; i > 0; i--)
15      printf("%u\n", x[i - 1]);
```

Ispis iz prethodnog zadatka, izveden u jednoj liniji uz zareze između brojeva, možemo napraviti ovako:

```
1 printf("Ispis brojeva u obrnutom redoslijedu: %u",
2   x[n-1]);
3 for (i = n - 2; i >= 0; i--) printf(", %u", x[i]);
4 printf("\n");
```

ili, pomoću `while()`-petlje (i bez pomoćne varijable `i`):

```
1 printf("Ispis brojeva u obrnutom redoslijedu: %u",
2   x[--n]);
3 while (n) printf(", %u", x[--n]);
4 printf("\n");
```

Zadatak 11.1.2. *Napišite dio programa koji za zadani prirodni broj n ispisuje njegove znamenke "ispravnim" redoslijedom (od lijeve prema desnoj).*

Rješenje koje smo prije vidjeli:

```
while (n > 0) {
  printf("%d\n", n % 10);
  n /= 10;
}
```

ispisuje znamenke u "pogrešnom" redoslijedu (od desne prema lijevoj). Da bismo dobili "ispravni" redoslijed, znamenke broja smještamo u niz, te ga ispisujemo unatrag. No, prilikom deklaracije niza potrebno je utvrditi koliko niz mora biti dugačak?

Moguće je precizno izračunati najveći mogući broj znamenaka (neovisno o računalu na kojem se program compilira), no to zahtijeva dodatna znanja, pa ćemo samo pretpostaviti da broj n ima manje od 100 znamenaka.

Rješenje.

```
int znam[100], i = 0;
while (n > 0) {
  znam[i++] = n % 10;
  n /= 10;
}
while (i) printf("%d\n", znam[--i]);
```

□

Zadatak 11.1.3. *Napišite dio programa koji učitava cijele brojeve dok ne učita nulu ili* 17 *brojeva. Potrebno je ispisati indeks i vrijednost broja koji ima najveću apsolutnu vrijednost (bez korištenja funkcija iz* `stdlib.h`*, tj. bez korištenja funkcije* `abs()` *i sličnih).*

Rješenje.

```
int x[17], n, i, max = 0, maxi = 0;

for (n = 0; n < 17; n++) {
  scanf("%d", &x[n]);
  if (!x[n]) break;
}
for (i = 0; i < n; i++) {
  int abs = (x[i] < 0 ? -x[i] : x[i]);
  if (abs > max) {
    max = abs;
    maxi = i;
  }
}
printf("Index: %d; Vrijednost: %d\n", maxi, x[maxi]);
```

□

Zadatak 11.1.4. *Napišite program koji učitava prirodne brojeve dok ne učita nulu. Za svaku znamenku od* 0 *do* 9 *program treba ispisati koliko se puta pojavila (u svim učitanim brojevima zajedno, ne računajući nulu kojom se prekida učitavanje).*

Rješenje.

```
#include <stdio.h>

int main(void) {
  int zn[10] = {0}, i = 0;
  int x;

  while(1) {
    printf("Ucitajte broj #%d: ", ++i);
    scanf("%d", &x);
    if (!x) break;
    while (x) {
      zn[x % 10]++;
      x /= 10;
    }
  }

  for (i = 0; i < 10; i++)
    printf("%d: %d\n", i, zn[i]);

  return 0;
}
```

□

Zadatak 11.1.5. *Riješite prethodni zadatak bez upotrebe nizova!*

Zadatak 11.1.6. *Napišite dio programa koji učitava n realnih brojeva ($n < 19$ je zadan). Potrebno je ispisati indeks i vrijednost broja koji je najudaljeniji (ima najveću apsolutnu razliku) od svog sljedbenika. Ako takvih ima više, ispišite samo prvog.*

Rješenje.

```
double x[18], max = 0;
int i, maxi = 0;

for (i = 0; i < n; i++)
  scanf("%lg", &x[i]);
for (i = 0; i < n - 1; i++) {
  double razlika = x[i] - x[i+1];
  if (razlika < 0) razlika = -razlika;
```

```
9     if (razlika > max) {
10      max = razlika;
11      maxi = i;
12    }
13  }
14  printf("Index: %d; Vrijednost: %g\n", maxi, x[maxi]);
```

□

Zadatak 11.1.7. *Promijenite prethodno rješenje tako da ako traženih brojeva ima više, program ispiše zadnjeg.*

Uputa. Prethodnom rješenju je dovoljno dodati točno jedan znak. Naravno, smijete to riješiti i "kompliciranije". □

11.2 Nizovi kao funkcijski argumenti

Napomena 11.2.1 (Ekvivalencija polja i pointera)**.** *U C-u je niz (polje) ekvivalentan pointeru na nulti element tog polja, tj. vrijedi:*

$$\texttt{p} \Leftrightarrow \texttt{\&p[0]}$$

Štoviše, vrijedi i općenitije:

$$\texttt{p + i} \Leftrightarrow \texttt{\&p[i]}$$

tj.

$$\texttt{*(p + i)} \Leftrightarrow \texttt{p[i]}$$

Više riječi o ovome bit će u poglavlju o dinamičkim nizovima. Ono što nama za sada treba je činjenica da se niz proslijeđen kao argument funkcije ponaša jednako kao pointer. To znači da ako smo funkciji proslijedili niz `niz` i izvršimo pridruživanje

```
p[0] = 17;
```

to je jednako kao da smo napisali

```
*p = 17;
```

što znači da ne mijenjamo varijablu `p` nego ono na što ona pokazuje. Zbog toga, promjena će se odraziti i na "originalni" niz u glavnom programu.

Dapače, kad "pošaljemo" niz u funkciju, zapravo se u memoriji kopira samo adresa nultog elementa niza, dok sam niz ostaje nepromijenjen u memoriji. **Zato su svi elementi takvog niza varijabilni argumenti funkcije!**

Niz, kao argument funkcije, navodimo jednako kao i u deklaraciji varijable u koju pohranjujemo niz, osim što možemo izostaviti duljinu niza (upravo zato jer ta deklaracija ne alocira cijeli niz, nego samo jedan pointer).

Zadatak 11.2.1. *Napišite funkciju koja prima niz realnih brojeva, te vraća sumu njegovih elemenata.*

Rješenje. Niz je određen elementima i duljinom. Kad se u zadatku kaže da funkcija prima niz nekakvih podataka, onda se obično podrazumijeva da zadajemo dva argumenta: sam niz i prirodni broj koji predstavlja njegovu duljinu! Duljinu ne prosljeđujemo kao argument ako ona nije potrebna (sljedeći zadatak).

```
double sum(double niz[], int n) {
  double s = 0;
  int i;
  for (i = 0; i < n; i++) s += niz[i];
  return s;
}
```

Varijabla `n` je obični argument (nije niti niz niti pointer), pa ju smijemo mijenjati bez posljedica po odgovarajuću varijablu s kojom je funkcija pozvana. Zbog toga (i zbog komutativnosti zbrajanja) funkciju možemo i ovako napisati:

```
double sum(double niz[], int n) {
  double s = 0;
  while (n--> 0) s+= niz[n];
  return s;
}
```

□

Zadatak 11.2.2. *Napišite funkciju koja prima niz cijelih brojeva i cjelobrojne argumente* `i` *i* `x`*. Funkcija treba postaviti vrijednost* `i`*-tog elementa niza na* `x`*. Napišite i program za testiranje funkcije.*

Rješenje. Ovdje je potrebno nešto napraviti na nekom elementu niza, što ne ovisi o duljini niza (ne piše da treba provjeriti ispravnost parametra `i`, tj. da je on nenegativan i strogo manji od duljine niza). Zbog toga nam ovdje ne treba duljina niza kao argument funkcije. Očito, funkcija ne mora vratiti nikakvu vrijednost.

Da bismo testirali funkciju, potrebno je dva puta ispisati niz (prije i nakon poziva funkcije). Zato ćemo ispis niza smjestiti u zasebnu funkciju (`print_niz()`). Također, kad samo testiramo funkciju, ne moramo učitavati brojeve; možemo ih zadati kao dio programa.

```
#include <stdio.h>

void set(int niz[], int i, int x) {
```

```
4   niz[i] = x;
5 }
6
7 void print_niz(int niz[], int n) {
8   int i;
9
10  printf("Elementi niza: %d", niz[0]);
11  for (i = 1; i < n; i++) printf(", %d", niz[i]);
12  printf("\n");
13 }
14
15 int main(void) {
16  int n = 5, x[] = {17, 19, 23, 29, 31};
17
18  print_niz(x, n);
19  set(x, 3, 17);
20  print_niz(x, n);
21
22  return 0;
23 }
```

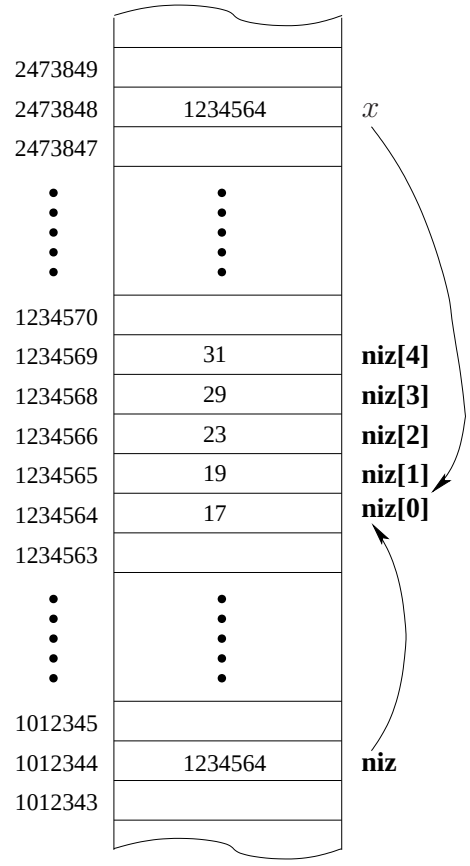
Na slici 11.1 možete vidjeti skicu memorije prilikom poziva funkcije `set()`, uz zanemarivanje veličina memorijskih lokacija (u stvarnosti, ćelije su različitih veličina; mi ovdje, zbog jednostavnosti, uzimamo veličinu 1). □

Zadatak 11.2.3. *Napišite funkcije za traženje minimuma i maksimuma niza realnih brojeva.*

Rješenje. "Klasično" rješenje:

```
1 double min(double niz[], int n) {
2   int i;
3   double res = niz[0];
4   for (i = 1; i < n; i++)
5     if (niz[i] < res) res = niz[i];
6   return res;
7 }
8
9 double max(double niz[], int n) {
10  int i;
11  double res = niz[0];
12  for (i = 1; i < n; i++)
13    if (niz[i] > res) res = niz[i];
14  return res;
15 }
```

Rješenje se može napisati i malo kompaktnije:



Slika 11.1: Prikaz memorije prilikom poziva funkcije `set()` u rješenju zadatka 11.2.2

```
double min(double niz[], int n) {
  double res = niz[--n];
  while (n--) if (niz[n] < res) res = niz[n];
  return res;
}

double max(double niz[], int n) {
  double res = niz[--n];
  while (n--) if (niz[n] > res) res = niz[n];
  return res;
}
```

Oba rješenja su podjednako dobra (ušteda od jedne cjelobrojne varijable je zanemariva).

Primijetite da oba rješenja pretpostavljaju da je $n > 0$, tj. da niz nije prazan. Kad bi niz bio prazan, minimum i maksimum ne bi bili definirani, a zadatak ništa ne govori o tom slučaju. □

Zadatak 11.2.4. *Napišite funkcije* `min()` *i* `max()` *tako da primaju još jedan parametar* `x` *tipa* `double`*. Funkcije trebaju vraćati minimum (odnosno maksimum) niza, osim za*

prazan niz za kojeg trebaju vratiti vrijednost `x`.

Zadatak 11.2.5. *Napišite funkciju koja prima niz realnih brojeva. Funkcija treba vratiti sumu elemenata niza, te – pomoću varijabilnih parametara – najmanji i najveći element niza. Pri tome smijete koristiti funkcije iz zadatka 11.2.3.*

Za vježbu, možete bilo koji od zadataka iz predhodnih poglavlja o programiranju pretočiti u jednu ili čak više funkcija.

11.3 Hornerov algoritam

Zadatak 11.3.1. *Napišite dio programa koji za zadani niz realnih brojeva* `a` *duljine* `n+1` *(*`n` *je također zadan) i za zadani realni broj* `x` *ispisuje vrijednost polinoma*

$$p(x) = \sum_{i=0}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n.$$

Očito, ali loše rješenje je:

Rješenje.

```
double p = 0;
int i, j;
for (i = 0; i <= n; i++) {
  double pot = 1;
  for (j = 0; j < i; j++) pot *= x;
  p += a[i] * pot;
}
printf("p(%g) = %g\n", x, p);
```

□

Ovo rješenje ima daleko previše množenja. Npr. x^3 računamo za svaki indeks $i \geq 3$. Donekle bolje rješenje je:

Rješenje.

```
double p = 0, pot = 1;
int i, j;
for (i = 0; i <= n; i++) {
  p += a[i] * pot;
  pot *= x;
}
printf("p(%g) = %g\n", x, p);
```

□

Iako ima puno manje množenja, ovo rješenje još uvijek nije dovoljno dobro zbog numeričke nestabilnosti (potencije mogu biti jako velike, pa se javlja *overflow* ili *underflow*). Daleko bolje rješenje možemo dobiti izlučivanjem zajedničkih potencija iz dijelova polinoma:

$$\begin{aligned} p(x) &= a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + \cdots + a_{n-1}x^{n-1} + a_nx^n \\ &= a_0 + x(a_1 + a_2x + a_3x^2 + a_4x^3 + \cdots + a_{n-1}x^{n-2} + a_nx^{n-1}) \\ &= a_0 + x(a_1 + x(a_2 + a_3x + a_4x^2 + \cdots + a_{n-1}x^{n-3} + a_nx^{n-2})) \\ &= a_0 + x(a_1 + x(a_2 + x(a_3 + a_4x + \cdots + a_{n-1}x^{n-4} + a_nx^{n-3}))) \\ &= a_0 + x(a_1 + x(a_2 + x(a_3 + x(a_4 + \cdots + x(a_{n-1} + xa_n)\dots)) \end{aligned}$$

Sada, po koracima, možemo računati:

$$\begin{aligned} p_0 &= a_n \\ p_1 &= a_{n-1} + xp_0 \\ p_2 &= a_{n-2} + xp_1 \\ p_3 &= a_{n-3} + xp_2 \\ &\dots \\ p_{n-1} &= a_1 + xp_{n-2} \\ p_n &= a_0 + xp_{n-1}, \end{aligned}$$

pa je $p(x) = p_n$. Opisani postupak zove se Hornerov algoritam.

Hornerov algoritam.

```
p = 0;
for (i = n; i >= 0; i--)
  p = p * x + a[i];
printf("p(%g) = %g\n", x, p);
```

□

Zadatak 11.3.2. *Napišite dio programa koji učitava realni broj x, te za zadani niz* `a` *duljine* `2*n` *ispisuje vrijednost $p_1(x) \cdot p_2(x)$ gdje je*

$$\begin{aligned} p_1(x) &= a_0 + a_2x + a_4x^2 + \cdots + a_{2n-2}x^{n-1} \\ p_2(x) &= a_1 + a_3x + a_5x^2 + \cdots + a_{2n-1}x^{n-1} \end{aligned}$$

Rješenje. Pomoću dvije primjene Hornerovog algoritma računamo $p_1(x)$ i $p_2(x)$, a zatim dobivene vrijednosti množimo.

```
scanf("%g", &x);
p1 = 0;
```

```
for (i = 2 * n - 2; i >= 0; i -= 2)
  p1 = p1 * x + a[i];
p2 = 0;
for (i = 2 * n - 1; i >= 1; i -= 2)
  p2 = p2 * x + a[i];
printf("p1(%g) * p2(%g) = %g\n", x, x, p1 * p2);
```

Primijetimo da oba polinoma možemo računati u istoj petlji:

```
scanf("%g", &x);
p1 = 0;
p2 = 0;
for (i = 2 * n - 1; i >= 1; i -= 2) {
  p1 = p1 * x + a[i - 1];
  p2 = p2 * x + a[i];
}
printf("p1(%g) * p2(%g) = %g\n", x, x, p1 * p2);
```

ili

```
scanf("%lg", &x);
p1 = 0;
p2 = 0;
for (i = n - 1; i >= 0; i--) {
  p1 = p1 * x + a[2 * i];
  p2 = p2 * x + a[2 * i + 1];
}
printf("p1(%g) * p2(%g) = %g\n", x, x, p1 * p2);
```

□

Napomena 11.3.1. *U posljednja dva rješenja prethodnog zadatka, vrijednost polinoma računamo u istoj* `for()`*-petlji jer polinomi imaju jednak broj koeficijenata. Da nije tako, morali bismo svakoga računati u zasebnoj petlji jer se petlja "vrti" onoliko puta koliko polinom ima koeficijenata.*

Zadatak 11.3.3. *Napišite funkciju koja kao parametre prima realni broj x, niz realnih brojeva* `a` *i cijeli broj* `n`*, pri čemu je niz* `a` *duljine* `2*(n+1)`*. Funkcija treba Hornerovim algoritmom izračunati i vratiti vrijednost $p_1(x) + p_2(x)$ gdje je*

$$p_1(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$
$$p_2(x) = a_{n+1} + a_{n+2} x + a_{n+3} x^2 + \cdots + a_{2n+1} x^n$$

Rješenje.

```
double p(double x, double a[], int n) {
  double p1 = 0, p2 = 0;
  for (i = n; i >= 0; i--) {
```

```
4       p1 = p1 * x + a[i];
5       p2 = p2 * x + a[i + n + 1];
6     }
7     return p1 + p2;
8   }
```

□

Zadatak 11.3.4. *Napišite dio programa koji učitava realni broj x, te za zadani niz* `a` *duljine* `m+n+2` *ispisuje vrijednost $p_1(x) - p_2(x)$ gdje je*

$$p_1(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$
$$p_2(x) = a_{n+1} + a_{n+2} x + a_{n+3} x^2 + \cdots + a_{m+n+1} x^m$$

Općenito, probleme s Hornerovim algoritmom je najlakše riješiti svođenjem polinoma na oblik

$$r(x) \cdot \sum_{i=0}^{f(n)} a_{g(i)} h(x)^i,$$

tj. preslagivanjem polinoma na način da potencije argumenta (ovdje je to $h(x)$) idu redom od nule (bez "preskakanja" potencija). Tada programski isječak koji računa vrijednost polinoma izgleda ovako:

```
p = 0;
for (i = f(n); i >= 0; i--)
  p = p * h(x) + a[g(i)];
p = p * r(x);
```

pri čemu za `r(x)`, `f(n)`, `g(i)` i `h(x)` treba uvrstiti stvarne formule.

Primjer 11.3.1.

1. $\sum_{i=0}^{n} a_i x^{n-i} = \sum_{i=0}^{n} a_{n-i} x^i$
   Dakle: $r(x) = 1$, $f(n) = n$, $g(i) = n - i$, $h(x) = x$, pa Hornerov algoritam izgleda ovako:

```
1 p = 0;
2 for (i = n; i >= 0; i--)
3   p = p * x + a[n - i];
```

2. $\sum_{i=0}^{n} a_i x^{i+1} = x \cdot \sum_{i=0}^{n} a_i x^i$
   Dakle: $r(x) = x$, $f(n) = n$, $g(i) = i$, $h(x) = x$, pa Hornerov algoritam izgleda ovako:

```
1 p = 0;
2 for (i = n; i >= 0; i--)
3   p = p * x + a[i];
4 p *= x;
```

3. $\sum_{i=2}^{n} a_{2\cdot i}x^{2\cdot i+1} = \sum_{i=0}^{n-2} a_{2\cdot(i+2)}x^{2\cdot(i+2)+1} = x^5 \cdot \sum_{i=0}^{n-2} a_{2i+4}(x^2)^i$
Dakle: $r(x) = x^5$, $f(n) = n-2$, $g(i) = 2i+4$, $h(x) = x^2$, *pa Hornerov algoritam izgleda ovako:*

```
p = 0;
for (i = n - 2; i >= 0; i--)
  p = p * (x * x) + a[2 * i + 4];
p *= x * x * x * x * x;
```

ili

```
p = 0;
for (i = n - 2; i >= 0; i--)
  p = p * (x * x) + a[2 * i + 4];
for (i = 0; i < 5; i++) p *= x;
```

Zadatak 11.3.5. *Napišite program koji učitava prirodne brojeve m i n takve da je $m+n < 100$, te $m+n+1$ realnih brojeva $(a_i)_{i=0}^{m+n}$. Program treba ispisati koliko je $p(a_0)$, pri čemu je*

$$p(x) = \frac{\sum_{i=1}^{m} a_i x^i}{\sum_{i=m+1}^{m+n} a_i x^{i-m}}.$$

Na primjer, za `m=3`, `n=2` *i niz brojeva* `a[] = {2,3,5,7,11,13}` *program treba ispisati* `1.10811`, *jer je*

$$p(2) = \frac{3\cdot 2 + 5\cdot 2^2 + 7\cdot 2^3}{11\cdot 2 + 13\cdot 2^2} = \frac{6+20+56}{22+52} = \frac{82}{74} = \frac{41}{37} = 1.10811.$$

Zadatak 11.3.6. *Napišite program koji učitava prirodne brojeve m i n takve da je $m+n < 100$, te $m+n+1$ realnih brojeva $(a_i)_{i=0}^{m+n}$. Program treba ispisati koliko je $p(a_m)$, pri čemu je*

$$p(x) = \frac{\sum_{i=0}^{m-1} a_i x^i}{\sum_{i=m+1}^{m+n} a_i x^{i-m}}.$$

Na primjer, za `m=3`, `n=2` *i niz brojeva* `a[] = {2,3,5,7,11,13}` *program treba ispisati* `0.37535`, *jer je*

$$p(7) = \frac{2\cdot 7^0 + 3\cdot 7^1 + 5\cdot 7^2}{11\cdot 7^1 + 13\cdot 7^2} = \frac{2+21+245}{77+637} = \frac{268}{714} = 0.37535.$$

Zadatak 11.3.7. *Napišite program koji učitava prirodni broj n, te n^2+1 realnih brojeva $(a_i)_{i=0}^{n^2}$. Program treba ispisati koliko je $p(a_{n^2})$, pri čemu je*

$$p(x) = \prod_{i=0}^{n-1} \sum_{j=n\cdot i}^{n\cdot(i+1)-1} a_j x^{j-n\cdot i}.$$

Na primjer, za `n=3` *i niz brojeva*
`a[] = {2, 0.3, 0.5, 2, 0.3, 0.5, -2, 3, -7, 0.7}`
program treba ispisati `-20.07`*, jer je*

$$\begin{aligned} p(0.7) &= (2+0.3\cdot 0.7+0.5\cdot 0.7^2)\cdot \\ &\quad (2+0.3\cdot 0.7+0.5\cdot 0.7^2)\cdot \\ &\quad (-2+3\cdot 0.7-7\cdot 0.7^2) \\ &= 2.455\cdot 2.455\cdot (-3.33) = -20.06999325 \\ &\approx -20.07. \end{aligned}$$

Zadatak 11.3.8. *Napišite program koji učitava prirodni broj n, te $n+1$ realnih brojeva $(a_i)_{i=0}^n$. Program treba ispisati koliko je $p(a_0)$, pri čemu je*

$$p(x)=\prod_{i=0}^{n-1}\sum_{j=1}^{i+1} a_j x^{j-1}.$$

Na primjer, za `n=3` *i niz brojeva* `a[] = {0.7, 2, 0.3, 0.5}` *program treba ispisati* `10.8511`*, jer je*

$$p(0.7)=2\cdot(2+0.3\cdot 0.7)\cdot(2+0.3\cdot 0.7+0.5\cdot 0.7^2)=2\cdot 2.21\cdot 2.455=10.8511.$$

11.4 Sortiranje

Da bismo u nekom nizu mogli što brže pronaći neki podatak (ili utvrditi da ga u nizu nema), korisno je poredati elemente niza po nekom kriteriju (npr. po veličini). Osnovna operacija koja nam pri tome treba je zamjena vrijednosti dviju varijabli, što je obrađeno u zadatku 10.2.2.

Nadalje, treba nam preraspoređivanje elemenata niza.

Zadatak 11.4.1. *Napišite dio programa koji učitani niz realnih brojeva* `x` *duljine n invertira (poreda tako da mu elementi idu obrnutim redoslijedom: $x_{n-1}, x_{n-2}, \ldots, x_1, x_0$). U rješenju smijete koristiti funkciju* `swap()` *iz zadatka 10.2.2.*

Rješenje (pogrešno). Zamjenjujemo i-ti element s lijeva i i-ti element s desna:

```
1 int i;
2 for (i = 0; i < n; i++) swap(&x[i], &x[n-1-i]);
```

Ovakva zamjena će dva puta zamijeniti svaki par (x_i, x_{n-1-i}) elemenata, što znači da će niz na kraju ostati nepromijenjen! □

Rješenje. Zamjenu i-tog elementa s lijeva i i-tog elementa s desna treba napraviti samo u jednoj polovici niza:

```
int i;
for (i = 0; i < n/2; i++) swap(&x[i], &x[n-1-i]);
```

ili, ako uzmemo u obzir ekvivalentnost nizova i pointera u C-u:

```
int i;
for (i = 0; i < n/2; i++) swap(x+i, x+n-1-i);
```

Bez korištenja funkcije `swap()`, rješenje izgleda ovako:

```
int i;
for (i = 0; i < n/2; i++) {
  double temp = x[i];
  x[i] = x[n-1+i];
  x[n-1+i] = temp;
}
```

□

Sada smo oboružani s dovoljno znanja da možemo poredati elemente niza po veličini.

Zadatak 11.4.2. *Napišite program koji učitava broj $n \leq 1719$, te niz od n cijelih brojeva. Brojeve treba sortirati uzlazno po veličini, te ispisati sortirani niz.*

Primijetimo da je niz $(x_i)_{i=0}^{n-1}$ sortiran ako i samo ako vrijedi:

$$\forall i, j : (i < j) \Rightarrow x_i \leq x_j.$$

Dakle, potrebno je "protrčati" po svim parovima indeksa i i j takvima da je $i < j$ i provjeriti jesu li x_i i x_j u ispravnom poretku. Ako nisu, potrebno ih je zamijeniti.

Rješenje.

```
#include <stdio.h>

int main(void) {
  int niz[1719];
  int n, i, j;

  printf("koliko elemenata ima niz? ");
  scanf("%d", &n);
  for (i = 0; i < n; i++) {
    printf("unesite %d. broj: ", i + 1);
    scanf("%d", &niz[i]);
  }

  for (i = 0; i < n - 1; i++)
    for (j = i + 1; j < n; j++)
      if (niz[i] > niz[j]) {
```

```
17          int temp = niz[i];
18          niz[i] = niz[j];
19          niz[j] = temp;
20        }
21
22    printf("sortirani niz:\n");
23    for (i = 0; i < n; i++)
24      printf(" %d\n", niz[i]);
25
26    return 0;
27  }
```

□

Prikazani algoritam se zove "klasični sort". Preciznije, riječ je o "lošoj" varijanti tog sorta, jer se radi previše zamjena za svaki element niza. Puno je bolje prvo naći najmanji element niza u dijelu niza s indeksima od i do $n-1$, pa onda njega zamijeniti s i-tim elementom niza (pogledati predavanja):

```
for (i = 0; i < n - 1; i++) {
  int temp;
  int mini;
  mini = i;
  for (j = i + 1; j < n; j++)
    if (niz[mini] > niz[j]) mini = j;
  temp = niz[i];
  niz[i] = niz[mini];
  niz[mini] = temp;
}
```

ili još malo bolje (ne radi zamjenu elementa sa samim sobom):

```
for (i = 0; i < n - 1; i++) {
  int mini;
  mini = i;
  for (j = i + 1; j < n; j++)
    if (niz[mini] > niz[j]) mini = j;
  if (mini > i) {
    int temp = niz[i];
    niz[i] = niz[j];
    niz[j] = temp;
  }
}
```

Zadatak 11.4.3. *Riješite prethodni zadatak tako da niz sortirate silazno (od najvećeg broja prema najmanjem).*

Uputa. Dovoljno je izmijeniti jedan znak u prethodnom rješenju. □

Zadatak 11.4.4. *Napišite dio programa koji učitava prirodni broj $n < 1317$, te niz od n cijelih brojeva. Brojeve treba sortirati silazno po sumi znamenaka, te ispisati sortirani niz.*

Rješenje.

```
int x[1316];
int n, i, j;

printf("Koliko elemenata ima niz? ");
scanf("%d", &n);

for (i = 0; i < n; i++) {
  printf("Unesite %d. broj: ", i + 1);
  scanf("%d", &x[i]);
}
for (i = 0; i < n - 1; i++) {
  /* Suma znamenaka elementa x[i] */
  int suma1 = 0,
      x1 = (x[i] < 0 ? -x[i] : x[i]);
  while (x1 > 0) {
    suma1 += x1 % 10;
    x1 /= 10;
  }
  for (j = i + 1; j < n; j++) {
    /* Suma znamenaka elementa x[j] */
    int suma2 = 0,
        x2 = (x[j] < 0 ? -x[j] : x[j]);
    while (x2 > 0) {
      suma2 += x2 % 10;
      x2 /= 10;
    }
    /* Usporedba suma znamenaka brojeva x[i] i x[j] */
    if (suma1 < suma2) {
      /* Zamjena x[i] i x[j] */
      int temp = x[i];
      x[i] = x[j];
      x[j] = temp;
      /* Sada x[i] ima sumu znamenaka suma2 */
      suma1 = suma2;
    }
  }
}

printf("Sortirani niz:\n");
for (i = 0; i < n; i++)
```

```
41    printf("  %d\n", x[i]);
```

Primijetimo da se mijenja samo uvjet zamjene (uspoređujemo sume, a ne same brojeve), dok sama zamjena ostaje nepromijenjena (i dalje zamjenjujemo elemente niza)!

Prilikom računanja suma znamenaka gledali smo apsolutne vrijednosti elemenata niza jer za negativne vrijednosti operator % vraća negativne vrijednosti, što bi dalo pogrešan rezultat!

Ovdje imamo dva puta napisan praktički isti kod (onaj za računanje zbroja znamenaka). To je puno lakše riješiti upotrebom funkcije koja će zbrojiti znamenke broja i vratiti tu vrijednost:

```
1  #include <stdio.h>
2
3  int suma_znamenaka(int x) {
4    int suma = 0;
5    if (x < 0) x = -x;
6    while (x > 0) {
7      suma += x % 10;
8      x /= 10;
9    }
10   return suma;
11 }
12
13 int main(void) {
14   int x[1316];
15   int n, i, j;
16
17   printf("Koliko elemenata ima niz? ");
18   scanf("%d", &n);
19   for (i = 0; i < n; i++) {
20     printf("Unesite %d. broj: ", i + 1);
21     scanf("%d", &x[i]);
22   }
23
24   for (i = 0; i < n - 1; i++) {
25     int suma1 = suma_znamenaka(x[i]);
26     for (j = i + 1; j < n; j++) {
27       if (suma1 < suma_znamenaka(x[j])) {
28         /* Zamjena x[i] i x[j] */
29         int temp = x[i];
30         x[i] = x[j];
31         x[j] = temp;
32         /* Sada x[i] ima sumu znamenaka koju je
33            prije imao x[j]. Dakle, treba azurirati
34            stanje varijable suma1. */
35         suma1 = suma_znamenaka(x[i]);
```

```
36        }
37      }
38    }
39
40    printf("Sortirani niz:\n");
41    for (i = 0; i < n; i++)
42      printf("  %d\n", x[i]);
43
44    return 0;
45  }
```

□

Napomena 11.4.1. *Suma znamenaka će se za većinu brojeva računati mnogo puta. Na žalost, ovo možemo izbjeći samo uvođenjem pomoćnog niza, što je često nepoželjno (i zabranjeno u zadacima) zbog povećane potrošnje memorije.*

Zadatak 11.4.5. *Riješite prethodni zadatak uvođenjem pomoćnog niza u kojem ćete čuvati sume znamenaka elemenata početnog niza tako da se te sume računaju točno jednom za svaki broj.*

Uputa. Pazite da prilikom zamjene treba zamjenjivati elemente **oba niza**! □

Primijetimo da je niz $(x_i)_{i=0}^{n-1}$ sortiran ako i samo ako vrijedi:

$$i \in \{0, 1, \ldots, n-2\} \Rightarrow x_i \leq x_{i+1}.$$

Drugim riječima, sortiranje možemo provesti tako da uspoređujemo samo susjedne elemente niza. Algoritam za sortiranje koji koristi ovo svojstvo zove se Bubble sort.

Zadatak 11.4.6. *Napišite dio programa koji uzlazno sortira zadani niz* `a` *s* `n` *realnih brojeva primjenom Bubble sorta.*

Ono što trebamo raditi je "protrčati" nizom i zamijeniti svaka dva susjedna elementa koji nisu u dobrom poretku. Na taj način ćemo donekle ispraviti niz, ali ga, na žalost, nećemo u potpunosti sortirati. "Trčanje" je potrebno raditi dok god niz ne postane sortiran, tj. dok god ne "protrčimo" kroz niz bez i jedne zamjene.

Bubble sort.

```
1  int nered;
2  do {
3    nered = 0;
4    for (i = 0; i < n - 1; i++)
5      if (a[i] > a[i+1]) {
6        double temp = a[i];
7        a[i] = a[i + 1];
```

```
8         a[i + 1] = temp;
9         nered = 1;
10      }
11  } while (nered);
```

□

Primijetimo da će već u prvom prolazu najveći element niza doći na zadnje mjesto (jer je veći od svih, pa će se stalno raditi zamjene). Dakle, njega više ne moramo provjeravati. Zbog toga možemo malo smanjiti obim posla koji računalo mora obaviti:

Malo brži Bubble sort.

```
1  int nered;
2  int zadnji = n;
3  do {
4    nered = 0;
5    zadnji--;
6    for (i = 0; i < zadnji; i++)
7      if (a[i] > a[i+1]) {
8        double temp = a[i];
9        a[i] = a[i + 1];
10       a[i + 1] = temp;
11       nered = 1;
12     }
13 } while (nered);
```

□

Iz ovog rješenja je očito i koliko puta će se izvršavati tijelo `for()` petlje u najgorem slučaju: u prvom koraku `while()`-petlje $n-1$ put, u drugom $n-2$ puta, ..., u zadnjem koraku jednom. Dakle, ukupno:

$$(n-1)+(n-2)+\cdots+2+1=\frac{n(n-1)}{2}$$

što znači da i ovaj algoritam, slično klasičnom sortu, ima kvadratnu složenost. Ipak, ta složenost se postiže u najgorem i u prosječnom slučaju, ali u idealnom slučaju (kad je niz već sortiran), algoritam samo $n-1$ puta izvršava `if()` (i provjeru uvjeta i inkrement `for()`-petlje), te tri puta pridruživanje (`zadnji = n`, `nered = 0` i `zadnji--`) i jednom provjeru uvjeta `do...while` petlje. Ukupno, složenost (u idealnom slučaju!) je samo linearna, što je ipak brže od klasičnog sortiranja.

11.5 Pretraživanje

Zadatak 11.5.1. *Neka je učitan niz cijelih brojeva* `x` *duljine* `n`*. Napišite dio programa koji će provjeriti nalazi li se u nizu* `x` *zadani cijeli broj* `y`*. Ako se nalazi, potrebno je*

ispisati njegov indeks; u protivnom treba ispisati odgovarajuću poruku.

Rješenje. "Trčimo" po svim elementima niza. Ako nađemo `y`, prekidamo "trčanje". Nakon petlje provjeravamo jesmo li iz petlje izašli jer je indeks "izletio" iz niza (tj. postao prevelik). Ako je, znači da nismo našli traženi broj; u protivnom, broj se nalazi na indeksu na kojem smo prekinuli petlju:

```
int i;
for (i = 0; i < n; i++)
  if (x[i] == y) break;
if (i < n)
  printf("Broj %d se nalazi u nizu i ima indeks %d\n",
    y, i);
else
  printf("Broj %d se ne nalazi u nizu.\n", y, i);
```

□

Zadatak 11.5.2. *Neka je učitan* ***uzlazno sortirani*** *niz cijelih brojeva* `x` *duljine* `n`*. Napišite dio programa koji će provjeriti nalazi li se u nizu* `x` *zadani cijeli broj* `y`*. Ako se nalazi, potrebno je ispisati njegov indeks; u protivnom treba ispisati odgovarajuću poruku.*

Očito, ovaj zadatak možemo riješiti jednako kao i prethodni, no možemo i efikasnije, korištenjem podatka da je niz (uzlazno) sortiran. Osnovna ideja je iskoristiti sljedeću činjenicu (koja vrijedi za sortirane nizove):

$$x < a_i \Rightarrow (\forall j \geq i : x < a_j)$$
$$x > a_i \Rightarrow (\forall j \leq i : x > a_j)$$

Dakle, prvi "sumnjivi" element niza, tj. onaj koji provjeravamo, nalazi se u sredini (ima indeks $\lfloor \frac{n}{2} \rfloor$). Ako je to element niza koji tražimo, problem je riješen. Ako nije, onda – ovisno o tome je li taj element veći ili manji od traženog broja – znamo da se traženi broj ne nalazi u prvih ili u zadnjih $\lfloor \frac{n}{2} \rfloor + 1$ elemenata niza. U svakom slučaju, odjednom smo "riješili" barem pola niza.

Korak algoritma koji pretražuje elemente niza od indeksa `lijevi` do indeksa `desni` možemo zapisati opisno:

- $srednji = \frac{lijevi+desni}{2}$
- Ako je $x = a_{srednji}$:
  Broj je nađen ⇒ STOP
- Ako je $x < a_{srednji} \leq a_{srednji+1} \leq \ldots$:
  Broj se ne nalazi desno od srednjeg, pa u idućem koraku ne treba gledati ništa od

sadašnjeg srednjeg na desno. Drugim riječima, nova desna granica je prethodnik sadašnjeg srednjeg, tj.
$desni = srednji - 1$

- Ako je $x > a_{srednji} \geq a_{srednji-1} \geq \ldots$:
  Broj se ne nalazi lijevo od srednjeg, pa u idućem koraku ne treba gledati ništa prije sadašnjeg srednjeg (niti njega). Drugim riječima, nova lijeva granica je sljedbenik sadašnjeg srednjeg, tj.
  $lijevi = srednji + 1$

Ovaj postupak ponavljamo dok ne nađemo broj ili dok `lijevi` ne postane veći od `desnog` (jer je tada dio niza koji gledamo prazan).

Početne vrijednosti `lijevog` i `desnog` su početni prvi i zadnji indeks niza, tj. nula i $n - 1$.

Rješenje.

```
int lijevi = 0, desni = n - 1, srednji;

while (lijevi <= desni) {
  srednji = (lijevi + desni) / 2;
  if (x == a[srednji]) break;
  if (x < a[srednji])
    desni = srednji - 1;
  else
    lijevi = srednji + 1;
}
if (lijevi <= desni)
  printf("Broj %d se nalazi u nizu i ima indeks %d\n",
    x, srednji);
else
  printf("Broj %d se ne nalazi u nizu.\n", x);
```

□

Napomena 11.5.1. *Prikazani algoritam zovemo **binarno pretraživanje** i možemo ga primijeniti kad god imamo niz koji je sortiran po istom kriteriju po kojem vršimo traženje. Inače, potrebno je primijeniti klasično pretraživanje (ono s početka ovog poglavlja).*

Klasično pretraživanje općenito nesortiranih nizova ima linearnu složenost, dok binarno pretraživanje ima logaritamsku složenost. Kao ilustraciju razlike ova dva algoritma, zamislite kako biste u telefonskom imeniku našli neku osobu kad ne biste uzeli u obzir da je imenik abecedno sortiran, nego biste prezimena pretraživali po redu.

Zadatak 11.5.3. *Neka je učitan **silazno** sortirani niz cijelih brojeva* `x` *duljine* `n`*. Napišite dio programa koji će provjeriti nalazi li se u nizu* `x` *zadani cijeli broj* `y`*. Ako se nalazi, potrebno je ispisati njegov indeks; u protivnom treba ispisati odgovarajuću poruku.*

Ponekad tražimo sve elemente koji odgovaraju nekom kriteriju. Tada nam sortiranost niza, u pravilu, ne pomaže.

Zadatak 11.5.4. *Napišite dio programa koji učitava prirodni broj $n \leq 1719$ i niz* `x` *od n realnih brojeva. Program treba izračunati aritmetičku sredinu niza i ispisati sve elemente niza koji su strogo manji od aritmetičke sredine.*

Podsjetnik s predavanja. Aritmetičku sredinu niza definiramo formulom:

$$AS(a) = \frac{\sum_{i=0}^{n-1} a_i}{n} = \frac{a_0 + a_1 + \cdots + a_{n-1}}{n},$$

a računamo ju tako da "protrčimo" kroz cijeli niz, zbrojimo sve elemente i na kraju dobivenu sumu podijelimo s brojem elemenata. □

Rješenje.

```
double x[1719], as = 0;
int n, i;

printf("Koliko elemenata ima niz? ");
scanf("%d", &n);

for (i = 0; i < n; i++) {
  printf("Unesite %d. broj: ", i + 1);
  scanf("%lg", &x[i]);
  as += x[i]; /* odmah racunamo sumu elemenata niza */
}
as /= n; /* sumu jos treba podijeliti s n */
/* "Trcimo" po nizu i ispisujemo elemente koji su */
/* manji od aritmeticke sredine */
for (i = 0; i < n; i++)
  if (x[i] < as) printf("%g\n", x[i]);
```

□

Zadatak 11.5.5. *Slično zadatku 11.1.6: Napišite dio programa koji učitava n realnih brojeva ($n < 19$ je zadan). Potrebno je ispisati indeks i vrijednost broja koji je najudaljeniji (ima najveću razliku) od svog sljedbenika. Ako takvih ima više, ispišite sve.*

Uputa. Ova modifikacija nije jednostavna kao ona od zadatka 11.1.7. Ovdje je potrebno napraviti dvije petlje: jednu koja će tražiti `max` i drugu koja će ispisati sve one brojeve čija je udaljenost od sljedbenika jednaka `max`. □

Zadatak 11.5.6. *Napišite program koji u niz učitava cijele brojeve dok god ne učita nulu (ili dok ne učita 616 brojeva). Program treba ispisati indekse onih elemenata niza kojima je prva znamenka manja od zadnje.*

Zadatak 11.5.7 (Šlag na kraju)**.** *Napišite program koji u niz učitava prirodne brojeve dok god ne učita prost broj (ili dok ne učita* 616 *brojeva). Program treba ispisati one elemente niza kojima je suma znamenka prost broj veći od* 17*.*

11.6 Dodavanje i brisanje elemenata

Ponekad je potrebno u nizove dodati nove ili izbrisati neke od starih elemenata. To se, naravno, može rješavati kreiranjem pomoćnih nizova, no takva rješenja su često memorijski zahtjevna i kompliciranа za izvođenje. Dodavanje i brisanje elemenata možemo jednostavno raditi na postojećim nizovima.

11.6.1 Dodavanje elemenata

Dodavanje elemenata smo već vidjeli u onoj najjednostavnijoj formi: učitavanje (svaki učitani broj dodajemo na kraj do tada učitanog niza). No, ponekad elemente treba dodati na početak ili drugdje.

Zadatak 11.6.1. *Napišite dio programa koji će u učitani niz* `x` *cijelih brojeva dodati broj* 17 ***ispred*** *prvog neparnog broja. Duljina niza pohranjena je u varijabli* `n`*.*

Rješenje. Dodavanje radimo u dva koraka. Prvo je potrebno naći prvi neparni broj, a zatim – ako takav postoji – sve elemente iza njega treba pomaknuti za jedno mjesto udesno i broj `n` povećati za 1.

```
1  /* Trazenje prvog neparnog */
2  for (i = 0; i < n; ++i)
3    if (x[i] % 2) break;
4
5  if (i < n) {
6    /* Pomicanje svih iza i-tog */
7    for (j = n-1; j >= i; --j)
8      x[j+1] = x[j];
9    /* Smanjimo n */
10   ++n;
11   /* Zapisujemo novi element */
12   x[i] = 17;
13 }
```

Primijetite: ako u nizu nema neparnih brojeva, niz će ostati nepromijenjen. □

Napomena 11.6.1. *Prilikom pomicanja elemenata udesno važno je da petlja po varijabli* `j` *ide s desna nalijevo. Sljedeće pomicanje je* ***pogrešno*** *jer će "pregaziti" sve elemente iza* `i`*-tog (pokušajte za niz* 2, 3, 4, 5*):*

```
6       /* Pomicanje svih iza i-tog */
7       for (j = i; j < n; ++j)
8         x[j+1] = x[j];
```

Zadatak 11.6.2. *Napišite dio programa koji će u učitani niz* `x` *cijelih brojeva dodati broj* 17 ***iza*** *prvog neparnog broja. Duljina niza pohranjena je u varijabli* `n`*.*

Rješenje. Rješenje je identično prethodnom, uz jedinu razliku u petlji za pomak udesno (linija 7) i zadavanju vrijednosti 17 (linija 12).

```
1   /* Trazenje prvog neparnog */
2   for (i = 0; i < n; ++i)
3     if (x[i] % 2) break;
4
5   if (i < n) {
6     /* Pomicanje svih iza i-tog */
7     for (j = n-1; j > i; --j)
8       x[j+1] = x[j];
9     /* Smanjimo n */
10    ++n;
11    /* Zapisujemo novi element */
12    x[i+1] = 17;
13  }
```

Alternativno, mogli smo samo prije ubacivanja (tj. prije linije 6 ili 7) povećati vrijednost varijable `i` za 1 i u potpunosti primijeniti rješenje prethodnog zadatka. □

Zadatak 11.6.3. *Napišite dio programa koji će u učitani niz* `x` *cijelih brojeva dodati brojeve* 17 *i* 19

a) ispred

b) iza

c) i ispred i iza

prvog neparnog broja. Duljina niza pohranjena je u varijabli `n`*.*

Zadatak 11.6.4. *Napišite dio programa koji će u učitani niz* `x` *cijelih brojeva dodati brojeve* 17 *i* 19

a) ispred

b) iza

c) i ispred i iza

zadnjeg *neparnog broja. Duljina niza pohranjena je u varijabli* `n`*.*

Uputa. Jedino što se mijenja je dio programa koji traži referentni član. Petlja u liniji 2 rješenja zadatka 11.6.1 treba ići unatrag (od `n` − 1 do 0). □

Zadatak 11.6.5. *Napišite dio programa koji će u učitani niz* `x` *cijelih brojeva dodati brojeve* 17 *i* 19

a) ispred

b) iza

c) i ispred i iza

trećeg *neparnog broja. Duljina niza pohranjena je u varijabli* `n`.

Uputa. Jedino što se mijenja je dio programa koji traži referentni član. U petlju u liniji 2 rješenja zadatka 11.6.1 treba dodati brojač koji se povećava sa svakim pronađenim neparnim brojem. Petlja se prekida kad taj brojač dostigne vrijednost 3. □

Zadatak 11.6.6. *Napišite dio programa koji će u učitani niz* `x` *cijelih brojeva dodati broj* $2x_k$ *ispred* ***svakog*** *neparnog broja* x_k. *Duljina niza pohranjena je u varijabli* `n`.

Rješenje. Ovaj bismo zadatak mogli riješiti višestrukim ponavljanjem rješenja zadatka 11.6.1, no takvo rješenje bilo bi kompliciranо i sporo. Umjesto toga, zadatak ćemo riješiti jednim prolazom kroz cijeli niz, prepisujući elemente na njihova nova mjesta.

Kroz niz je potrebno prolaziti s desna na lijevo, kao i kod pomicanja elemenata za jedno mjesto (iz istog razloga koji je objašnjen u napomeni 11.6.1). Postavlja se pitanje odakle krenuti? Da bismo to znali, potrebno je prvo izračunati novu duljinu niza.

```
1  #include <stdio.h>
2
3  void ispis(int x[], int n) {
4    int i;
5    printf("%d", x[0]);
6    for (i = 1; i < n; ++i)
7      printf(", %d", x[i]);
8    printf("\n");
9  }
10
11 int main(void) {
12   int x[1719], n, n2, i, j, br_dodanih = 0;
13
14   printf("Unesite duljinu niza: ");
15   scanf("%d", &n);
16
17   for (i = 0; i < n; ++i) {
18     printf("Unesite %d. clan niza: ", i+1);
```

```
19      scanf("%d", &x[i]);
20    }
21
22    printf("\nPrije dodavanja: ");
23    ispis(x, n);
24
25    /* Ra\v{c}unanje nove duljine niza */
26    for (i = 0; i < n; ++i)
27      if (x[i] % 2 == 1) ++br_dodanih;
28    n2 = n + br_dodanih;
29
30    /* Dodavanje novih elemenata */
31    j = n2 - 1;
32    for (i = n - 1; i >= 0; --i) {
33      x[j] = x[i];
34      --j;
35      if (x[i] % 2 == 1) {
36        x[j] = 2 * x[i];
37        --j;
38      }
39    }
40    n = n2;
41
42    printf("Nakon dodavanja: ");
43    ispis(x, n);
44
45    return 0;
46  }
```

□

11.6.2 Brisanje elemenata

Zadatak 11.6.7. *Napišite dio programa koji iz učitanog niza* `x` *cijelih brojeva briše prvi parni broj. Duljina niza pohranjena je u varijabli* `n`.

Napomena 11.6.2 (Česte greške)**.** *Elemente koje želimo obrisati nije dovoljno zamijeniti s nulom, jer je to samo zamjena elemenata drugom vrijednošću, a broj elemenata u nizu ostaje nepromijenjen.*

Također, nije dovoljno niti samo ispisati niz bez elemenata koje želimo obrisati, jer se samim ispisom niz uopće ne mijenja i ti elementi i dalje ostaju njegovi članovi.

Rješenje zadatka 11.6.7. Slično rješenju zadatka 11.6.1, brisanje radimo u dva koraka. Prvo je potrebno naći prvi parni broj, a zatim – ako takav postoji – sve elemente iza njega treba pomaknuti za jedno mjesto ulijevo i broj `n` smanjiti za 1.

```
1  /* Trazenje prvog parnog */
2  for (i = 0; i < n; ++i)
3    if (x[i] % 2 == 0) break;
4
5  if (i < n) {
6    /* Pomicanje svih iza i-tog */
7    while (i+1 < n) {
8      x[i] = x[i+1];
9      ++i;
10   }
11   /* Smanjimo n */
12   --n;
13 }
```

□

Zadatak 11.6.8. *Napišite dio programa koji iz učitanog niza* `x` *cijelih brojeva briše* ***zadnji*** *parni broj. Duljina niza pohranjena je u varijabli* `n`*.*

Zadatak 11.6.9. *Napišite dio programa koji iz učitanog niza* `x` *cijelih brojeva briše prvi parni broj i iduća dva elementa iza njega. Duljina niza pohranjena je u varijabli* `n`*.*

Napomena 11.6.3. *Pripazite postoje li tražena dva elementa niza. Na primjer, ako je prvi parni broj u nizu predzadnji element, onda iza njega postoji samo jedan element niza, pa će ukupni broj obrisanih elemenata biti dva, a ne tri.*

Zadatak 11.6.10. *Napišite program koji učitava prirodni broj* $n \leq 1719$ *i niz* `x` *od* n *realnih brojeva. Program treba iz niza obrisati sve elemente koji su strogo manji od aritmetičke sredine svih elemenata niza, te ispisati tako nastali niz i novu aritmetičku sredinu niza.*

Rješenje. Ovaj zadatak možemo riješiti višestrukim ponavljanjem rješenja prethodnog zadatka, no to znači da ćemo mnogo elemenata pomicati više puta po jedno mjesto, što je sporo.

Zato ovakvo brisanje (linije 35–44) radimo, kao i kod višestrukog dodavanja u zadatku 11.6.6, na način da po redu čitamo sve elemente (indeks `i`). One koje **ne treba** obrisati, prepisujemo na nove pozicije te povećavamo odgovarajući indeks (`j`) kako bi bio spreman za zapisivanje idućeg broja. Za elemente koje treba obrisati, pamtimo koliko je obrisanih (varijabla `br_obrisanih`), ali **ne mijenjamo** indeks `j`.

Na kraju je potrebno smanjiti vrijednost u varijabli `n` koja drži duljinu niza.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void provjeri_n(int n) {
5    /* aritmeticka sredina nije definirana za prazne nizove */
```

```
6   if (n == 0) {
7     printf("Niz je prazan.");
8     exit(0);
9   }
10 }
11
12 double arsr(double niz[], int n) {
13   int i;
14   double sum = 0;
15   for (i = 0; i < n; ++i) sum += niz[i];
16   return sum / n;
17 }
18
19 int main(void) {
20   double x[1719], as;
21   int n, i, j = 0, br_obrisanih = 0;
22
23   printf("Unesite duljinu niza: ");
24   scanf("%d", &n);
25   provjeri_n(n);
26
27   for (i = 0; i < n; ++i) {
28     printf("Unesite %d. clan niza: ", i+1);
29     scanf("%lf", &x[i]);
30   }
31
32   as = arsr(x, n);
33   printf("Aritmeticka sredina prije brisanja: %g\n", as);
34
35   /* Brisanje */
36   for (i = 0; i < n; ++i) {
37     if (x[i] < as) {
38       ++br_obrisanih;
39       continue;
40     }
41     if (i > j) x[j] = x[i];
42     ++j;
43   }
44   n -= br_obrisanih;
45
46   provjeri_n(n);
47
48   as = arsr(x, n);
49   printf("Aritmeticka sredina nakon brisanja: %g\n", as);
50   printf("%g", x[0]);
51   for (i = 1; i < n; ++i)
```

```
52      printf(", %g", x[i]);
53    printf("\n");
54
55    return 0;
56  }
```

Kod višestrukog dodavanja kroz niz prolazimo slijeva na desno. Utoliko je brisanje jednostavnije od dodavanja jer nije potrebno unaprijed izračunati novu duljinu niza (v. rješenje zadatka 11.6.6). □

Zadatak 11.6.11. *Napišite program koji učitava prirodni broj $n \leq 1719$ i niz* `x` *od n realnih brojeva. Program treba iz niza obrisati sve elemente kojima je index između 7 i 19, te ispisati tako nastali niz.*

Zadatak 11.6.12. *Napišite program koji učitava prirodni broj $n \leq 1719$ i niz* `x` *od n realnih brojeva. Program treba iz niza obrisati sve elemente kojima je index prost broj, te ispisati tako nastali niz.*

Zadatak 11.6.13. *Napišite program koji učitava prirodni broj $n \leq 1719$ i niz* `x` *od n realnih brojeva. Program treba iz niza obrisati sve elemente kojima je index neka potencija broja 2, te ispisati tako nastali niz.*

Zadatak 11.6.14. *Napišite program koji učitava prirodni broj $n \leq 1719$ i niz* `x` *od n realnih brojeva. Program treba iz niza obrisati sve elemente koji su strogo veći od svog prvog susjeda slijeva, te ispisati tako nastali niz.*

Napomena 11.6.4. *Pripazite odakle čitate niz, jer prvi element niza nema lijevih susjeda.*

Zadatak 11.6.15. *Napišite funkciju koja prima niz realnih brojeva, te iz njega briše najmanji element. Ako se taj pojavljuje u nizu više puta, treba obrisati samo najljevijeg od njih.*

Rješenje. Podsjetimo se: niz je određen s dva podatka, što znači da funkcija mora primiti barem dva argumenta (polje i duljinu niza). Prilikom brisanja elemen(a)ta, potrebno je smanjiti duljinu niza, te ju nekako vratiti u glavni program. To možemo napraviti ili preko povratne vrijednosti funkcije ili preko varijabilnih argumenata. Prvi način je obično jednostavniji, no nije primjenjiv kad funkcija mora vratiti još neku vrijednost.

Kao što znamo, polja se uvijek prenose kao varijabilni argumenti, pa njih možemo mijenjati bez posebnih "zahvata".

```
1  void brisi(double x[], int *n) {
2    double min;
3    int i, j;
4    /* Ako je niz prazan, nemamo sto raditi */
5    if (n == 0) return;
6    /* Trazimo najmanji element niza */
7    min = x[0];
```

```
  for (i = 1; i < *n; ++i)
    if (x[i] < min) min = x[i];
  /* Brisemo prvo pojavljivanje najmanjeg elementa niza */
  for (i = 0; i < *n; ++i) {
    if (x[i] == min) {
      for (j = i+1; j < *n; ++j) x[j-1] = x[j];
      --(*n);
      return;
    }
  }
}
```

Primjer poziva funkcije:

```
  double x[1719];
  int n;
  ...
  brisi(x, &n);
```

Ista funkcija bez varijabilnih argumenata izgleda ovako:

```
int brisi(double x[], int n) {
  double min;
  int i, j;
  /* Ako je niz prazan, nemamo sto raditi */
  if (n == 0) return 0;
  /* Trazimo najmanji element niza */
  min = x[0];
  for (i = 1; i < n; ++i)
    if (x[i] < min) min = x[i];
  /* Brisemo prvo pojavljivanje najmanjeg elementa niza */
  for (i = 0; i < n; ++i) {
    if (x[i] == min) {
      for (j = i+1; j < n; ++j) x[j-1] = x[j];
      --n;
      return n;
    }
  }
}
```

Primjer poziva funkcije:

```
  double x[1719];
  int n;
  ...
  n = brisi(x, n);
```

Ovakvo rješenje neće biti primjenjivo u zadatku 11.6.18. □

Zadatak 11.6.16. *Napišite funkciju koja prima niz realnih brojeva, te iz njega briše najmanji element. Ako se taj pojavljuje u nizu više puta, treba obrisati samo* ***najdesnijeg*** *od njih.*

Uputa. Rješenje je slično prvom iz zadatka 11.6.15, uz promjenu smjera vanjske petlje kod brisanja (linija 11). □

Zadatak 11.6.17. *Napišite funkciju koja prima niz realnih brojeva, te iz njega briše najmanji element. Ako se taj pojavljuje u nizu više puta, treba obrisati* ***sve*** *njih.*

Uputa. Rješenje je najlakše napraviti po uzoru na ono iz zadatka 11.6.10, no može se napraviti i slično prvom rjesenju zadatka 11.6.15, uz brisanje naredbe `return` u liniji 15 i premještanje inkrementa `++i` iz `for`-petlje u `else` (ako je element s indeksom `i` obrisan, tada na njegovo mjesto dolazi onaj koji je prije bio na indeksu `i+1`, što znači da još nije provjeren i u tom slučaju se `++i` ne smije izvršiti). □

Zadatak 11.6.18. *Napišite funkciju koja prima niz realnih brojeva, te iz njega briše najmanji element. Ako se taj pojavljuje u nizu više puta, treba obrisati* ***sve*** *njih. Funkcija kao povratnu vrijednost treba vratiti broj obrisanih elemenata.*

Kazalo