

## **PROGRAMIRANJE 2**



**Milena Vujošević Janičić, Jelena Graovac,  
Nina Radojičić, Ana Spasić,  
Mirko Spasić, Anđelka Zečević**

**PROGRAMIRANJE 2**  
**Zbirka zadataka sa rešenjima**

**Beograd  
2016.**

Autori:

*dr Milena Vujošević Janičić*, docent na Matematičkom fakultetu u Beogradu

*dr Jelena Graovac*, docent na Matematičkom fakultetu u Beogradu

*Nina Radojičić*, asistent na Matematičkom fakultetu u Beogradu

*Ana Spasić*, asistent na Matematičkom fakultetu u Beogradu

*Mirko Spasić*, asistent na Matematičkom fakultetu u Beogradu

*Andelka Zečević*, asistent na Matematičkom fakultetu u Beogradu

## PROGRAMIRANJE 2

Zbirka zadataka sa rešenjima

Izdavač: Matematički fakultet Univerziteta u Beogradu. Studentski trg 16, Beograd.

Za izdavača: *prof. dr Zoran Rakić*, dekan

Recenzenti:

*dr Gordana Pavlović-Lažetić*, redovni profesor na Matematičkom fakultetu u Beogradu

*dr Dragan Urošević*, naučni savetnik na Matematičkom institutu SANU

Obrada teksta, crteži i korice: *autori*.

Štampa: Copy Centar, Beograd. Tiraž 200.

СIP Каталогизација у публикацији

Народна библиотека Србије, Београд

004.4(075.8)(076)

004.432.2C(075.8)(076)

PROGRAMIRANJE 2 : zbirka zadataka sa rešenjima / Milena Vujošević

Jančić ... [et al.]. - Beograd : Matematički fakultet, 2016

(Beograd : Copy Centar). - VII, 361 str. ; 24 cm

Tiraž 200.

ISBN 978-86-7589-107-9

1. Вујошевић Јаничић, Милена 1980- [аутор]

а) Програмирање - Задаци б) Програмски језик "C"- Задаци

COBISS.SR-ID 221508876

©2016. Milena Vujošević Jančić, Jelena Graovac, Nina Radojičić, Ana Spasić, Mirko Spasić, Andelka Zečević

Ovo delo zaštićeno je licencom Creative Commons CC BY-NC-ND 4.0 (Attribution-NonCommercial-NoDerivatives 4.0 International License). Detalji licence mogu se videti na veb-adresi <http://creativecommons.org/licenses/by-nc-nd/4.0/>. Dozvoljeno je umnožavanje, distribucija i javno saopštavanje dela, pod uslovom da se navedu imena autora. Upotreba dela u komercijalne svrhe nije dozvoljena. Prerada, preoblikovanje i upotreba dela u sklopu nekog drugog nije dozvoljena.



# Sadržaj

<b>1</b>	<b>Uvodni zadaci</b>	<b>1</b>
1.1	Podela koda po datotekama . . . . .	1
1.2	Algoritmi za rad sa bitovima . . . . .	5
1.3	Rekurzija . . . . .	10
1.4	Rešenja . . . . .	18



# Predgovor

U okviru kursa *Programiranje 2* na Matematičkom fakultetu vežbaju se zadaci koji imaju za cilj da studente nauče rekurzivnom pristupu rešavanja problema, ispravnom radu sa pokazivačima i dinamički alociranom memorijom, osnovnim algoritmima pretraživanja i sortiranja, kao i radu sa dinamičkim strukturama podataka, poput listi i stabala. Zadaci koji se nalaze u ovoj zbirci predstavljaju objedinjen skup zadataka sa vežbi i praktikuma ovog kursa, kao i primere zadataka sa održanih ispita. Elektronska verzija zbirke i propratna rešenja u elektronskom formatu, dostupna su besplatno u okviru strane kursa [www.programiranje2.matf.bg.ac.rs](http://www.programiranje2.matf.bg.ac.rs) u skladu sa navedenom licencom.

U prvom poglavlju zbirke obrađene su uvodne teme koje obuhvataju osnovne tehnike koje se koriste u rešavanju svih ostalih zadataka u zbirci: podela koda po datotekama i rekurzivni pristup rešavanju problema. Takođe, u okviru ovog poglavlja dati su i osnovni algoritmi za rad sa bitovima. Drugo poglavlje je posvećeno pokazivačima: pokazivačkoj aritmetici, višedimenzionim nizovima, dinamičkoj alokaciji memorije i radu sa pokazivačima na funkcije. Treće poglavlje obrađuje algoritme pretrage i sortiranja, a četvrto dinamičke strukture podataka: liste i stabla. Dodatak sadrži najvažnije ispitne rokove iz jedne akademske godine. Većina zadataka je rešena, a teži zadaci su obeleženi zvezdicom.

Autori velikog broja zadataka ove zbirke su ujedno i autori same zbirke, ali postoje i zadaci za koje se ne može tačno utvrditi ko je originalni autor jer su zadacima davali svoje doprinose različiti asistenti koji su držali vežbe iz ovog kursa u prethodnih desetak godina. Zbog toga smatramo da je naš osnovni doprinos što smo objedinili, precizno formulisali, rešili i detaljno iskomentarisali sve najvažnije zadatke koji su potrebni za uspešno savlađivanje koncepata koji se obrađuju u okviru kursa. Takođe, formulacije zadataka smo obogatili primerima koji upotpunjuju razumevanje zahteva zadataka i koji omogućavaju čitaocu zbirke da proverí sopstvena rešenja. Primeri su dati u obliku testova i interakcija sa programom. Testovi su svedene prirode i obuhvataju samo jednostavne ulaze i izlaze iz programa. Interakcija sa programom obuhvata naizmeničnu interakciju čovek-računar u kojoj su ulazi i izlazi isprepleteni. U zadacima koji zahtevaju

rad sa argumentima komandne linije, navedeni su i primeri poziva programa, a u zadacima koji demonstriraju rad sa datotekama, i primeri ulaznih ili izlaznih datoteka. Test primeri koji su navedeni uz ispitne zadatke u dodatku su oni koji su korišćeni za početno testiranje (koje prethodi ocenjivanju) studentskih radova na ispitima.

Neizmerno zahvaljujemo recenzentima, Gordani Pavlović Lažetić i Draganu Uroševiću, na veoma pažljivom čitanju rukopisa i na brojnim korisnim sugestijama. Takođe, zahvaljujemo studentima koji su svojim aktivnim učešćem u nastavi pomogli i doprineli uobličavanju ovog materijala.

Svi komentari i sugestije na sadržaj zbirke su dobrodošli i osećajte se slobodnim da ih pošaljete elektronskom poštom bilo kome od autora<sup>1</sup>.

*Autori*

---

<sup>1</sup>Adrese autora su: milena, jgraovac, nina, aspasic, mirko, andjelkaz, sa nastavkom @matf.bg.ac.rs



# 1

## Uvodni zadaci

### 1.1 Podela koda po datotekama

**Zadatak 1.1** Napisati program za rad sa kompleksnim brojevima.

- (a) Definirati strukturu `KompleksanBroj` koja opisuje kompleksan broj zadan njegovim realnim i imaginarnim delom.
- (b) Napisati funkciju `void ucitaj_kompleksan_broj(KompleksanBroj * z)` koja učitava kompleksan broj `z` sa standardnog ulaza.
- (c) Napisati funkciju `void ispisi_kompleksan_broj(KompleksanBroj z)` koja ispisuje kompleksan broj `z` na standardni izlaz u odgovarajućem formatu.
- (d) Napisati funkciju `float realan_deo(KompleksanBroj z)` koja vraća vrednost realnog dela broja `z`.
- (e) Napisati funkciju `float imaginaran_deo(KompleksanBroj z)` koja vraća vrednost imaginarnog dela broja `z`.
- (f) Napisati funkciju `float moduo(KompleksanBroj z)` koja vraća moduo kompleksnog broja `z`.
- (g) Napisati funkciju `KompleksanBroj konjugovan(KompleksanBroj z)` koja vraća konjugovano-kompleksni broj broja `z`.
- (h) Napisati funkciju `KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)` koja vraća zbir dva kompleksna broja `z1` i `z2`.

## 1 Uvodni zadaci

---

- (i) Napisati funkciju `KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)` koja vraća razliku dva kompleksna broja  $z1$  i  $z2$ .
- (j) Napisati funkciju `KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)` koja vraća proizvod dva kompleksna broja  $z1$  i  $z2$ .
- (k) Napisati funkciju `float argument(KompleksanBroj z)` koja vraća argument kompleksnog broja  $z$ .

Napisati program koji testira prethodno napisane funkcije. Sa standardnog ulaza uneti dva kompleksna broja  $z1$  i  $z2$ , a zatim ispisati realni deo, imaginarni deo, moduo, konjugovano-kompleksan broj i argument broja koji se dobija kao zbir, razlika ili proizvod brojeva  $z1$  i  $z2$  u zavisnosti od znaka ('+', '-', '\*') koji se unosi sa standardnog ulaza.

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite realni i imaginarni deo kompleksnog broja: 1 -3
(1.00 - 3.00 i)
Unesite realni i imaginarni deo kompleksnog broja: -1 4
(-1.00 + 4.00 i)
Unesite znak (+,-,*): -
(1.00 - 3.00 i) - (-1.00 + 4.00 i) = (2.00 - 7.00 i)
Realni_deo: 2
Imaginarni_deo: -7.000000
Moduo: 7.280110
Konjugovano kompleksan broj: (2.00 + 7.00 i)
Argument kompleksnog broja: - 1.292497
```

**Zadatak 1.2** Uraditi prethodni zadatak tako da su sve napisane funkcije za rad sa kompleksnim brojevima zajedno sa definicijom strukture `KompleksanBroj` izdvojene u posebnu biblioteku. Napisati program koji testira ovu biblioteku. Sa standardnog ulaza uneti kompleksan broj, a zatim na standardni izlaz ispisati njegov polarni oblik.

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite realni i imaginarni deo kompleksnog broja: -5 2
Polarni oblik kompleksnog broja je 5.39 * e^i * 2.76
```

**Zadatak 1.3** Napisati biblioteku za rad sa polinomima.

- (a) Definisati strukturu `Polinom` koja opisuje polinom stepena najviše 20 koji je zadat nizom svojih koeficijenata tako da se na  $i$ -toj poziciji u nizu nalazi koeficijent uz  $i$ -ti stepen polinoma.

- (b) Napisati funkciju `void ispisi(const Polinom * p)` koja ispisuje polinom `p` na standardni izlaz, od najvišeg ka najnižem stepenu. Ipisati samo koeficijente koji su različiti od nule.
- (c) Napisati funkciju `Polinom ucitaj()` koja učitava polinom sa standardnog ulaza. Za polinom najpre uneti stepen, a zatim njegove koeficijente.
- (d) Napisati funkciju `double izracunaj(const Polinom * p, double x)` koja vraća vrednosti polinoma `p` u datoj tački `x` koristeći Hornerov algoritam.
- (e) Napisati funkciju `Polinom saberi(const Polinom * p, const Polinom * q)` koja vraća zbir dva polinoma `p` i `q`.
- (f) Napisati funkciju `Polinom pomnozi(const Polinom * p, const Polinom * q)` koja vraća proizvod dva polinoma `p` i `q`.
- (g) Napisati funkciju `Polinom izvod(const Polinom * p)` koja vraća izvod polinoma `p`.
- (h) Napisati funkciju `Polinom n_izvod(const Polinom * p, int n)` koja vraća `n`-ti izvod polinoma `p`.

Napisati program koji testira prethodno napisane funkcije. Sa standardnog ulaza učitati polinome `p` i `q`, a zatim ih ispisati na standardni izlaz u odgovarajućem formatu. Izračunati i ispisati zbir `z` i proizvod `r` unetih polinoma `p` i `q`. Sa standardnog ulaza učitati realni broj `x`, a zatim na standardni izlaz ispisati vrednost polinoma `z` u tački `x` zaokruženu na dve decimale. Na kraju, sa standardnog ulaza učitati broj `n` i na izlaz ispisati `n`-ti izvod polinoma `r`.

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite polinom p (prvo stepen, pa zatim koeficijente od najvećeg stepena do nultog):
3 1.2 3.5 2.1 4.2
Unesite polinom q (prvo stepen, pa zatim koeficijente od najvećeg stepena do nultog):
2 2.1 0 -3.9
Zbir polinoma je polinom z:
1.20x^3+5.60x^2+2.10x+0.30
Prozvod polinoma je polinom r:
2.52x^5+7.35x^4-0.27x^3-4.83x^2-8.19x-16.38
Unesite tacku u kojoj racunate vrednost polinoma z:
0
Vrednost polinoma z u tacki 0.00 je 0.30
Unesite izvod polinoma koji zelite:
3
3. izvod polinoma r je: 151.20x^2+176.40x-1.62
```

**Zadatak 1.4** Napisati biblioteku za rad sa razlomcima.

- (a) Definirati strukturu `Razlomak` koja opisuje razlomak.
- (b) Napisati funkciju `Razlomak ucitaj()` za učitavanje razlomka.
- (c) Napisati funkciju `void ispisi(const Razlomak * r)` koja ispisuje razlomak `r`.
- (d) Napisati funkciju `int brojilac(const Razlomak * r)` koja vraća brojilac razlomka `r`.
- (e) Napisati funkciju `int imenilac(const Razlomak * r)` koja vraća imenilac razlomka `r`.
- (f) Napisati funkciju `double realna_vrednost(const Razlomak * r)` koja vraća odgovarajuću realnu vrednost razlomka `r`.
- (g) Napisati funkciju `double recipročna_vrednost(const Razlomak * r)` koja vraća recipročnu vrednost razlomka `r`.
- (h) Napisati funkciju `Razlomak skрати(const Razlomak * r)` koja vraća skraćenu vrednost datog razlomka `r`.
- (i) Napisati funkciju `Razlomak saberi(const Razlomak * r1, const Razlomak * r2)` koja vraća zbir dva razlomka `r1` i `r2`.
- (j) Napisati funkciju `Razlomak oduzmi(const Razlomak * r1, const Razlomak * r2)` koja vraća razliku dva razlomka `r1` i `r2`.
- (k) Napisati funkciju `Razlomak pomnozi(const Razlomak * r1, const Razlomak * r2)` koja vraća proizvod dva razlomka `r1` i `r2`.
- (l) Napisati funkciju `Razlomak podeli(const Razlomak * r1, const Razlomak * r2)` koja vraća količnik dva razlomka `r1` i `r2`.

Napisati program koji testira prethodne funkcije. Sa standardnog ulaza učitati dva razlomka `r1` i `r2`. Na standardni izlaz ispisati skraćene vrednosti zbira, razlike, proizvoda i količnika razlomaka `r1` i recipročne vrednosti razlomka `r2`.

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite imenilac i brojilac prvog razlomka: 1 2
Unesite imenilac i brojilac drugog razlomka: 2 3
1/2 + 3/2 = 2
1/2 - 3/2 = -1
1/2 * 3/2 = 3/4
1/2 / 3/2 = 1/3
```

## 1.2 Algoritmi za rad sa bitovima

**Zadatak 1.5** Napisati biblioteku `stampanje_bitova` za rad sa bitovima. Biblioteka treba da sadrži funkcije `stampanje_bitova`, `stampanje_bitova_short` i `stampanje_bitova_char` za štampanje bitova u binarnom zapisu celog broja tipa `int`, `short` i `char`, koji se zadaje kao argument funkcije. Napisati program koji testira napisanu biblioteku. Sa standardnog ulaza učitati u heksadekadnom formatu cele brojeve tipa `int`, `short` i `char` i na standardni izlaz ispisati njihovu binarnu reprezentaciju.

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj tipa int: 0xf4f4f4f
Binarna reprezentacija: 01001111010011110100111101001111
Unesite broj tipa short: 0xf4f
Binarna reprezentacija: 0100111101001111
Unesite broj tipa char: 0xf
Binarna reprezentacija: 01001111
```

**Zadatak 1.6** Napisati funkcije `_bitove_1` i `prebroj_bitove_2` koje vraćaju broj jedinica u binarnom zapisu označenog celog broja  $x$  koji se zadaje kao argument funkcije. Prebrojavanje bitova ostvariti na dva načina:

- formiranjem odgovarajuće maske i njenim pomeranjem (funkcija `prebroj_bitove_1`)
- formiranjem odgovarajuće maske i pomeranjem promenljive  $x$  (funkcija `prebroj_bitove_2`).

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati ceo broj u heksadekadnom formatu i redni broj funkcije koju treba primeniti (1 ili 2), a zatim na standardni izlaz ispisati broj jedinica u binarnom zapisu učitano broja pozivom izabrane funkcije. Ukoliko korisnik ne unese ispravnu vrednost za redni broj funkcije, prekinuti izvršavanje programa i ispisati odgovarajuću poruku na standardni izlaz za greške.

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj: 0x7F
Unesite redni broj funkcije: 1
Poziva se funkcija prebroj_bitove_1
Broj jedinica u zapisu je 7
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj: -0x7F
Unesite redni broj funkcije: 2
Poziva se funkcija prebroj_bitove_2
Broj jedinica u zapisu je 26
```

*Primer 3*

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj: 0x00FF00FF
Unesite redni broj funkcije: 2
Poziva se funkcija prebroj_bitove_2
Broj jedinica u zapisu je 16
```

*Primer 4*

```
INTERAKCIJA SA PROGRAMOM:
Unesite broj: 0x00FF00FF
Unesite redni broj funkcije: 3
IZLAZ ZA GREŠKE:
Greska: Neodgovarajući redni broj funkcije.
```

**Zadatak 1.7** Napisati funkcije `unsigned najveci(unsigned x)` i `unsigned najmanji(unsigned x)` koje vraćaju najveći, odnosno najmanji neoznačen ceo broj koji se može zapisati istim binarnim ciframa kao broj `x`.

Napisati program koji testira prethodno napisane funkcije. Sa standardnog ulaza učitati neoznačen ceo broj u heksadekadnom formatu, a zatim ispisati binarnu reprezentaciju najvećeg i najmanjeg broja koji se može zapisati istim binarnim ciframa kao učitani broj.

### Test 1

```
ULAZ:
0x7F
IZLAZ:
Najveci:
11111100000000000000000000000000
Najmanji:
00000000000000000000000001111111
```

### Test 2

```
ULAZ:
0x80
IZLAZ:
Najveci:
10000000000000000000000000000000
Najmanji:
00000000000000000000000000000001
```

### Test 3

```
ULAZ:
0x00FF00FF
IZLAZ:
Najveci:
11111111111111111000000000000000
Najmanji:
00000000000000001111111111111111
```

### Test 4

```
ULAZ:
0xFFFFFFFF
IZLAZ:
Najveci:
11111111111111111111111111111111
Najmanji:
11111111111111111111111111111111
```

**Zadatak 1.8** Napisati funkcije za rad sa bitovima.

- Napisati funkciju `unsigned postavi_0(unsigned x, unsigned n, unsigned p)` koja vraća broj koji se dobija kada se `n` bitova datog broja `x`, počevši od pozicije `p`, postave na 0.
- Napisati funkciju `unsigned postavi_1(unsigned x, unsigned n, unsigned p)` koja vraća broj koji se dobija kada se `n` bitova datog broja `x`, počevši od pozicije `p`, postave na 1.
- Napisati funkciju `unsigned vrati_bitove(unsigned x, unsigned n, unsigned p)` koja vraća broj u kome se `n` bitova najmanje težine poklapa sa `n` bitova broja `x` počevši od pozicije `p`, dok su mu ostali bitovi postavljeni na 0.
- Napisati funkciju `unsigned postavi_1_n_bitova(unsigned x, unsigned n, unsigned p, unsigned y)` koja vraća broj koji se dobija upisivanjem poslednjih `n` bitova najmanje težine broja `y` u broj `x`, počevši od pozicije `p`.
- Napisati funkciju `unsigned invertuj(unsigned x, unsigned n, unsigned p)` koja vraća broj koji se dobija invertovanjem `n` bitova broja `x` počevši od pozicije `p`.



## 1 Uvodni zadaci

---

na poziciju najmanje težine. Analogno, rotiranje bitova udesno podrazumeva pomeranje svih bitova za jednu poziciju udesno, s tim što se bit sa pozicije najmanje težine pomera na poziciju najveće težine.

- (a) Napisati funkciju `unsigned rotiraj_ulevo(unsigned x, unsigned n)` koja vraća broj koji se dobija rotiranjem `n` puta ulevo datog celog neoznačenog broja `x`.
- (b) Napisati funkciju `unsigned rotiraj_udesno(unsigned x, unsigned n)` koja vraća broj koji se dobija rotiranjem `n` puta udesno datog celog neoznačenog broja `x`.
- (c) Napisati funkciju `int rotiraj_udesno_oznaceni(int x, unsigned n)` koja vraća broj koji se dobija rotiranjem `n` puta udesno datog celog broja `x`.

Napisati program koji sa standardnog ulaza učitava neoznačene cele brojeve `x` i `n` koji se unose u heksadekaskom formatu, zatim ispisuje binarnu reprezentaciju vrednosti dobijene pozivanjem tri prethodno napisane funkcije sa argumentima `x` i `n`, a na kraju ispisuje binarnu reprezentaciju vrednosti dobijene pozivanjem funkcije `rotiraj_udesno_oznaceni` za argumente `-x` i `n`.

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite neoznaceni broj x: ba11a7
Unesite neoznaceni broj n: 5
x = 00000000101110100001000110100111
rotiraj_ulevo(ba11a7, 5)           = 00010111010000100011010011100000
rotiraj_udesno(ba11a7, 5)          = 00111000000001011101000010001101
rotiraj_udesno_oznaceni(ba11a7, 5) = 00111000000001011101000010001101
rotiraj_udesno_oznaceni(-ba11a7, 5) = 11000111111101000010111101110010
```

**Zadatak 1.10** Napisati funkciju `unsigned ogledalo(unsigned x)` koja vraća broj čiji binarni zapis predstavlja sliku u ogledalu binarnog zapisa broja `x`. Napisati program koji testira datu funkciju za broj koji se sa standardnog ulaza zadaje u heksadekadnom formatu. Najpre ispisati binarnu reprezentaciju unetog broja, a zatim i binarnu reprezentaciju broja dobijenog kao njegova slika u ogledalu.

### Test 1

```
ULAZ:
255
IZLAZ:
00000000000000000000000000000000
10101010010000000000000000000000
```

### Test 2

```
ULAZ:
-15
IZLAZ:
11111111111111111111111111101011
11010111111111111111111111111111
```



**Zadatak 1.11** Napisati funkciju `int broj_01(unsigned int n)` koja za dati broj `n` vraća 1 ako u njegovom binarnom zapisu ima više jedinica nego nula, a inače vraća 0. Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza.

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
ULAZ: 10	ULAZ: 2147377146	ULAZ: 1111111115
IZLAZ: 0	IZLAZ: 1	IZLAZ: 0

**Zadatak 1.12** Napisati funkciju `int broj_parova(unsigned int x)` koja vraća broj pojava dve uzastopne jedinice u binarnom zapisu celog neoznačenog broja `x`. Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza. NAPOMENA: *Tri uzastopne jedinice sadrže dve uzastopne jedinice dva puta.*

<i>Test 1</i>	<i>Test 2</i>	<i>Test 3</i>
ULAZ: 11	ULAZ: 1024	ULAZ: 2147377146
IZLAZ: 1	IZLAZ: 0	IZLAZ: 22

\* **Zadatak 1.13** Napisati program koji sa standardnog ulaza učitava pozitivan ceo broj, a na standardni izlaz ispisuje vrednost tog broja sa razmenjenim vrednostima bitova na pozicijama *i* i *j*. Pozicije *i* i *j* učitati kao parametre komandne linije. Pri rešavanju nije dozvoljeno koristiti ni pomoćni niz ni aritmetičke operatore `+`, `-`, `/`, `*`, `%`.

<i>Primer 1</i>	<i>Primer 2</i>	<i>Primer 2</i>
POKRETANJE: ./a.out 1 2	POKRETANJE: ./a.out 1 2	POKRETANJE: ./a.out 12 12
INTERAKCIJA SA PROGRAMOM:	INTERAKCIJA SA PROGRAMOM:	INTERAKCIJA SA PROGRAMOM:
ULAZ: 11	ULAZ: 1024	ULAZ: 12345
IZLAZ: 13	IZLAZ: 1024	IZLAZ: 12345

\* **Zadatak 1.14** Napisati funkciju `void prevod(unsigned int x, char s[])` koja na osnovu neoznačenog broja `x` formira nisku `s` koja sadrži heksadekadni zapis broja `x` koristeći algoritam za brzo prevođenje binarnog u heksadekadni zapis (svake 4 binarne cifre se zamenjuju jednom odgovarajućom heksade-

## 1 Uvodni zadaci

---

kadnom cifrom). Napisati program koji tu funkciju testira za broj koji se zadaje sa standardnog ulaza.

Test 1	Test 2	Test 3
ULAZ: 11 IzLAZ: 0000000B	ULAZ: 1024 IzLAZ: 00000400	ULAZ: 12345 IzLAZ: 00003039

\* **Zadatak 1.15** Napisati funkciju koja za data dva neoznačena broja  $x$  i  $y$  invertuje one bitove u broju  $x$  koji se poklapaju sa odgovarajućim bitovima u broju  $y$ . Ostali bitovi treba da ostanu nepromenjeni. Napisati program koji testira tu funkciju za brojeve koji se zadaju sa standardnog ulaza.

Test 1	Test 2	Test 3
ULAZ: 123 10 IzLAZ: 4294967285	ULAZ: 3251 0 IzLAZ: 4294967295	ULAZ: 12541 1024 IzLAZ: 4294966271

**Zadatak 1.16** Napisati funkciju koja vraća broj petica u oktalnom zapisu neoznačenog celog broja  $x$ . Napisati program koji testira tu funkciju za broj koji se zadaje sa standardnog ulaza. NAPOMENA: *Zadatak rešiti isključivo korišćenjem bitskih operatora.*

Test 1	Test 2	Test 3
ULAZ: 123 IzLAZ: 0	ULAZ: 3245 IzLAZ: 2	ULAZ: 100328 IzLAZ: 1

## 1.3 Rekurzija

**Zadatak 1.17** Napisati rekurzivnu funkciju koja izračunava  $x^k$ , za dati ceo broj  $x$  i prirodan broj  $k$

- (a) tako da rešenje bude linearne složenosti,
- (b) tako da rešenje bude logaritamske složenosti.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati redni broj funkcije koju treba primeniti ('1' ili '2'), ceo broj  $x$  i prirodan broj  $k$ ,

a zatim na standardni izlaz ispisati rezultat primene izabrane funkcije na unete brojeve. Ukoliko se na ulazu unese pogrešan redni broj funkcije, ispisati odgovarajuću poruku o grešci na standardni izlaz i prekinuti izvršavanje programa.

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
Unesite redni broj funkcije (1/2):
1
Unesite broj x:      2
Unesite broj k:      10
1024
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:
Unesite redni broj funkcije (1/2):
2
Unesite broj x:      9
Unesite broj k:      4
6561
```

**Zadatak 1.18** Koristeći uzajamnu (posrednu) rekurziju napisati:

- (a) funkciju `unsigned paran(unsigned n)` koja proverava da li je broj cifara broja `x` paran i vraća 1 ako jeste, a 0 inače;
- (b) i funkciju `unsigned neparan(unsigned n)` koja proverava da li je broj cifara broja `x` neparan i vraća 1 ako jeste, a 0 inače.

Napisati program koji testira napisane funkcije tako što za heksadekadni broj koji se unosi sa standardnog ulaza ispisuje da li je broj njegovih cifara paran ili neparan.

*Test 1*

```
ULAZ:
11
IZLAZ:
Uneti broj ima paran broj cifara.
```

*Test 2*

```
ULAZ:
123
IZLAZ:
Uneti broj ima neparan broj cifara.
```

**Zadatak 1.19** Napisati repno-rekurzivnu funkciju koja izračunava faktorijel broja  $n$ . Napisati program koji testira napisanu funkciju za proizvoljan broj  $n$  ( $n \leq 12$ ) unet sa standardnog ulaza. NAPOMENA: Gornja vrednost za  $n$  je postavljena na 12 zbog ograničenja veličine broja koji može da stane u promenljivu tipa `int` i činjenice da niz faktorijela brzo raste.

*Primer 1*

```
INTERAKCIJA SA PROGRAMOM:
Unesite n (<= 12): 5
5! = 120
```

*Primer 2*

```
INTERAKCIJA SA PROGRAMOM:
Unesite n (<= 12): 0
0! = 1
```

**Zadatak 1.20** Napisati funkciju koja vraća  $n$ -ti element u nizu Fibonačijevih brojeva. Elementi niza Fibonačijevih brojeva  $F$  izračunavaju se na osnovu

## 1 Uvodni zadaci

---

sledećih rekurentnih relacija:

$$F(0) = 0$$

$$F(1) = 1$$

$$F(n) = F(n-1) + F(n-2)$$

Napisati program koji testira napisanu funkciju. Sa standardnog ulaza učitati prirodan broj  $n$  i na standardni izlaz ispisati rezultat primene napisane funkcije na prirodan broj  $n$ .

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite koji clan niza se racuna: 5
F(5) = 5
```

### Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesite koji clan niza se racuna: 8
F(8) = 21
```

**Zadatak 1.21** Elementi niza  $F$  izračunavaju se na osnovu sledećih rekurentnih relacija:

$$F(0) = 0$$

$$F(1) = 1$$

$$F(n) = a \cdot F(n-1) + b \cdot F(n-2)$$

Napisati funkciju koja računa  $n$ -ti element u nizu  $F$

- (a) iterativno,
- (b) tako da funkcija bude rekurzivna i da koristi navedene rekurentne relacije,
- (c) tako da funkcija bude rekurzivna, ali da se problemi manje dimenzije rešavaju samo jedan put.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati redni broj funkcije koju treba primeniti ('1','2','3'), vrednosti koeficijenata  $a$  i  $b$  i prirodan broj  $n$ . Na standardni izlaz ispisati rezultat primene odabrane funkcije nad učitanim podacima, a u slučaju unosa pogrešnog rednog broja funkcije ispisati odgovarajuću poruku i prekinuti izvršavanje programa. NAPOMENA: *Niz  $F$  definisan na ovaj način predstavlja uopštenje Fibonačijevih brojeva.*

### Primer 1

```
INTERAKCIJA SA PROGRAMOM:
Unesite redni broj funkcije:
1 - iterativna
2 - rekurzivna
3 - rekurzivna napredna
1
Unesite koeficijente: 2 3
Unesite koji clan niza se racuna: 5
F(5) = 61
```

### Primer 2

```
INTERAKCIJA SA PROGRAMOM:
Unesite redni broj funkcije:
1 - iterativna
2 - rekurzivna
3 - rekurzivna napredna
3
Unesite koeficijente: 4 2
Unesite koji clan niza se racuna: 8
F(8) = 31360
```

**Zadatak 1.22** Napisati rekurzivnu funkciju koja sabira dekadne cifre datog celog broja  $x$ . Napisati program koji testira ovu funkciju za broj koji se unosi sa standardnog ulaza.

<p><i>Test 1</i></p> <pre> ULAZ: 123 IZLAZ: 6 </pre>	<p><i>Test 2</i></p> <pre> ULAZ: 23156 IZLAZ: 17 </pre>	<p><i>Test 3</i></p> <pre> ULAZ: 1432 IZLAZ: 10 </pre>
<p><i>Test 4</i></p> <pre> ULAZ: 1 IZLAZ: 1 </pre>	<p><i>Test 5</i></p> <pre> ULAZ: 0 IZLAZ: 0 </pre>	

**Zadatak 1.23** Napisati rekurzivnu funkciju koja sumira elemente niza celih brojeva

- (a) sabirajući elemente počev od početka niza ka kraju niza,
- (b) sabirajući elemente počev od kraja niza ka početku niza.

Napisati program koji testira napisane funkcije. Sa standardnog ulaza učitati redni broj funkcije ('1' ili '2'), zatim dimenziju  $n$  ( $0 < n \leq 100$ ) celobrojnog niza, a potom i elemente niza. Na standardni izlaz ispisati rezultat primene odabrane funkcije nad učitanim nizom, a u slučaju unosa pogrešnog rednog broja funkcije ispisati odgovarajuću poruku i prekinuti izvršavanje programa.

<p><i>Primer 1</i></p> <pre> INTERAKCIJA SA PROGRAMOM: Unesite redni broj funkcije (1 ili 2): 1 Unesite dimenziju niza: 5 Unesite elemente niza: 1 2 3 4 5 Suma elemenata je 15 </pre>	<p><i>Primer 2</i></p> <pre> INTERAKCIJA SA PROGRAMOM: Unesite redni broj funkcije (1 ili 2): 2 Unesite dimenziju niza: 4 Unesite elemente niza: -5 2 -3 6 Suma elemenata je 0 </pre>
--	---

**Zadatak 1.24** Napisati rekurzivnu funkciju koja određuje maksimum niza celih brojeva. Napisati program koji testira ovu funkciju za niz koji se unosi sa standardnog ulaza. Elementi niza se unose sve do kraja ulaza (EOF). Pretpostaviti da niz neće imati više od 256 elemenata.

### Test 1

```

|| ULAZ:
|| 3 2 1 4 21
|| IZLAZ:
|| 21

```

### Test 2

```

|| ULAZ:
|| 2 -1 0 -5 -10
|| IZLAZ:
|| 2

```

### Test 3

```

|| ULAZ:
|| 1 11 3 5 8 1
|| IZLAZ:
|| 11

```

**Zadatak 1.25** Napisati rekurzivnu funkciju koja izračunava skalarni proizvod dva vektora celih brojeva. Napisati program koji testira ovu funkciju za nizove (vektore) koji se unose sa standardnog ulaza. Prvo treba uneti dimenziju nizova, a zatim i njihove elemente. Na standardni izlaz ispisati skalarni proizvod unetih nizova. Pretpostaviti da nizovi neće imati više od 256 elemenata.

### Primer 1

```

|| INTERAKCIJA SA PROGRAMOM:
|| Unesite dimenziju nizova: 3
|| Unesite elemente prvog niza:
|| 1 2 3
|| Unesite elemente drugog niza:
|| 1 2 3
|| Skalarni proizvod je 14

```

### Primer 2

```

|| INTERAKCIJA SA PROGRAMOM:
|| Unesite dimenziju nizova: 2
|| Unesite elemente prvog niza:
|| 3 5
|| Unesite elemente drugog niza:
|| 2 6
|| Skalarni proizvod je 36

```

**Zadatak 1.26** Napisati rekurzivnu funkciju koja vraća broj pojavljivanja elementa  $x$  u nizu  $a$  dužine  $n$ . Napisati program koji testira ovu funkciju za broj  $x$  i niz  $a$  koji se unose sa standardnog ulaza. Prvo se unosi  $x$ , a zatim elementi niza sve do kraja ulaza. Pretpostaviti da nizovi neće imati više od 256 elemenata.

### Primer 1

```

|| INTERAKCIJA SA PROGRAMOM:
|| Unesite ceo broj:
|| 4
|| Unesite elemente niza:
|| 1 2 3 4
|| Broj pojavljivanja je 1

```

### Primer 2

```

|| INTERAKCIJA SA PROGRAMOM:
|| Unesite ceo broj:
|| 11
|| Unesite elemente niza:
|| 3 2 11 14 11 43 1
|| Broj pojavljivanja je 2

```

### Primer 3

```

|| INTERAKCIJA SA PROGRAMOM:
|| Unesite ceo broj:
|| 1
|| Unesite elemente niza:
|| 3 21 5 6
|| Broj pojavljivanja je 0

```

**Zadatak 1.27** Napisati rekurzivnu funkciju kojom se proverava da li su tri data cela broja uzastopni članovi datog celobrojnog niza. Sa standardnog ulaza

```
INTERAKCIJA SA PROGRAMOM:
Unesite tri cela broja:
1 2 3
Unesite elemente niza:
4 1 2 3 4 5
Uneti brojevi jesu uzastopni
clanovi niza.
```

```
INTERAKCIJA SA PROGRAMOM:
Unesite tri cela broja:
1 2 3
Unesite elemente niza:
11 1 2 4 3 6
Uneti brojevi jesu uzastopni
clanovi niza.
```

```
ULAZ:
    0x7F
IZLAZ:
    7
```

```
ULAZ:
    0x00FF00FF
IZLAZ:
    16
```

```
ULAZ:
    0xFFFFFFFF
IZLAZ:
    32
```

```
ULAZ:
    10
IzLAZ:
      000000000000000000000000000001010
```

```
ULAZ:  
0  
IZLAZ:  
00000000000000000000000000000000
```

## 1 Uvodni zadaci

---

Test 1	Test 2	Test 3
ULAZ: 5    IZLAZ: 5	ULAZ: 125    IZLAZ: 7	ULAZ: 8    IZLAZ: 1

**Zadatak 1.31** Napisati rekurzivnu funkciju za određivanje (dekadne vrednosti) najveće cifre u heksadekadnom zapisu neoznačenog celog broja korišćenjem bitskih operatora. UPUTSTVO: *Binarne cifre grupisati u podgrupe od po četiri cifre, počev od bitova najmanje težine.*

Test 1	Test 2	Test 3
ULAZ: 5    IZLAZ: 5	ULAZ: 16    IZLAZ: 1	ULAZ: 18    IZLAZ: 2

**Zadatak 1.32** Napisati rekurzivnu funkciju `int palindrom(char s[], int n)` koja ispituje da li je data niska `s` palindrom. Napisati program koji testira ovu funkciju za nisku koja se zadaje sa standardnog ulaza. Pretpostaviti da niska neće imati više od 31 karaktera.

Test 1	Test 2	Test 3
ULAZ: a    IZLAZ: da	ULAZ: aa    IZLAZ: da	ULAZ: aba    IZLAZ: da

  

Test 4	Test 5
ULAZ: programiranje    IZLAZ: ne	ULAZ: anavolimilovana    IZLAZ: da

\* **Zadatak 1.33** Napisati rekurzivnu funkciju koja prikazuje sve permutacije skupa  $\{1, 2, \dots, n\}$ . Napisati program koji testira napisanu funkciju za proizvoljan prirodan broj  $n$  ( $n \leq 15$ ) unet sa standardnog ulaza.



*Test 1*

```

ULAZ:
  2
IZLAZ:
  1 2
  2 1

```

*Test 2*

```

ULAZ:
  3
IZLAZ:
  1 2 3
  1 3 2
  2 1 3
  2 3 1
  3 1 2
  3 2 1

```

*Test 3*

```

ULAZ:
  -5
IZLAZ ZA GREŠKE:
  Greska: Duzina
  permutacije
  mora biti
  broj iz
  intervala
  [0, 15]!

```

\* **Zadatak 1.34** Paskalov trougao sadrži brojeve čije se vrednosti računaju tako što svako polje ima vrednost zbira dve vrednosti koje su u susedna dva polja iznad. Izuzetak su jedinice na krajevima. Vrednosti brojeva Paskalovog trougla odgovaraju binomnim koeficijentima, tj. vrednost polja  $(n, k)$ , gde je  $n$  redni broj hipotenuze, a  $k$  redni broj elementa u tom redu (na toj hipotenuzi) odgovara binomnom koeficijentu  $\binom{n}{k}$ , pri čemu brojanje počinje od nule. Na primer, vrednost polja  $(4, 2)$  je 6.

- Napisati rekurzivnu funkciju koja izračunava vrednost binomnog koeficijenta  $\binom{n}{k}$  koristeći osobine Paskalovog trougla.
- Napisati rekurzivnu funkciju koja izračunava  $d_n$  kao sumu elemenata  $n$ -te hipotenuze Paskalovog trougla.

Napisati program koji za unetu veličinu Paskalovog trougla i redni broj hipotenuze najpre iscertava Paskalov trougao, a zatim štampa sumu elemenata hipotenuze.

*Test 1*

```

ULAZ:
  5 3
IZLAZ:
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 1
8

```

*Test 2*

```

ULAZ:
  6 5
IZLAZ:
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 1
1 6 15 20 15 6 1
32

```

\* **Zadatak 1.35** Napisati rekurzivnu funkciju koja prikazuje sve varijacije sa ponavljanjem dužine  $n$  skupa  $\{a, b\}$ , i program koji je testira, za  $n$  koje se unosi sa standardnog ulaza.

### Test 1

```
ULAZ:
2
IZLAZ:
a a
a b
b a
b b
```

### Test 2

```
ULAZ:
3
IZLAZ:
a a a
a a b
a b a
a b b
b a a
b a b
b b a
b b b
```

\* **Zadatak 1.36** *Hanojske kule*: Data su tri vertikalna štapa. Na jednom od njih se nalazi  $n$  diskova poluprečnika 1, 2, 3,... do  $n$ , tako da se najveći nalazi na dnu, a najmanji na vrhu. Ostala dva štapa su prazna. Potrebno je premestiti diskove sa jednog na drugi štap tako da budu u istom redosledu, pri čemu se ni u jednom trenutku ne sme staviti veći disk preko manjeg. Preostali štap koristiti kao pomoćni štap prilikom premeštanja.

Napisati program koji za proizvoljnu vrednost  $n$ , koja se unosi sa standardnog ulaza, prikazuje proces premeštanja diskova.

\* **Zadatak 1.37** *Modifikacija Hanojskih kula*: Data su četiri vertikalna štapa. Na jednom se nalazi  $n$  diskova poluprečnika 1, 2, 3,... do  $n$ , tako da se najveći nalazi na dnu, a najmanji na vrhu. Ostala tri štapa su prazna. Potrebno je premestiti diskove na drugi štap tako da budu u istom redosledu, premestajući jedan po jedan disk, pri čemu se ni u jednom trenutku ne sme staviti veći disk preko manjeg. Preostala dva štapa koristiti kao pomoćne štapove prilikom premeštanja. Napisati program koji za proizvoljnu vrednost  $n$ , koja se unosi sa standardnog ulaza, prikazuje proces premeštanja diskova.

## 1.4 Rešenja

### Rešenje 1.1

```
#include <stdio.h>
2 #include <stdlib.h>
#include <math.h>

4
/* Struktura kojom je predstavljan kompleksan broj sadrzi realan i
6   imaginaran deo kompleksnog broja */
typedef struct {
8   float real;
   float imag;
```

```
10 } KomplexsanBroj;

12 /* Funkcija ucitava sa standardnog ulaza realan i imaginara deo
    kompleksnog broja i smesta ih u strukturu cija je adresa
14 argument funkcije */
void ucitaj_kompleksan_broj(KompleksanBroj * z)
16 {
    /* Ucitava se vrednost sa standardnog ulaza */
18    printf("Unesite realni i imaginarni deo kompleksnog broja: ");
    scanf("%f", &z->real);
20    scanf("%f", &z->imag);
}

22 /* Funkcija ispisuje na standardan izlaz zadati kompleksni broj u
    obliku (x + i y). Ovoj funkciji se argument prenosi po vrednosti
24 jer se u samoj funkciji ne menja njegova vrednost */
void ispisi_kompleksan_broj(KompleksanBroj z)
26 {
    /* Zapocinje se sa ispisom */
28    printf("(");

30    /* Razlikuju se dva slucaja: 1) realni deo kompleksnog broja
    razlicit od nule: tada se realni deo ispisuje na standardni
32 izlaz, nakon cega se ispisuje znak + ili - u zavisnosti da li
    je imaginarni deo pozitivan ili negativan, a potom i apsolutna
34 vrednost imaginarnog dela kompleksnog broja 2) realni deo
    kompleksnog broja je nula: tada se samo ispisuje imaginaran
36 deo, s tim sto se ukoliko su oba dela nula ispisuje samo 0, bez
    decimalnih mesta */
38    if (z.real != 0) {
        printf("%.2f", z.real);

40        if (z.imag > 0)
            printf(" + %.2f i", z.imag);
        else if (z.imag < 0)
            printf(" - %.2f i", -z.imag);
42    } else {
        if (z.imag == 0)
            printf("0");
        else
            printf("%.2f i", z.imag);
44    }

46    /* Završava se sa ispisom */
48    printf(")");
50 }

52 /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
54 float realan_deo(KompleksanBroj z)
56 {
    return z.real;
60 }
```

```
62  /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
64  float imaginaran_deo(KompleksanBroj z)
65  {
66      return z.imag;
67  }
68
69  /* Funkcija vraca vrednost modula zadatog kompleksnog broja */
70  float moduo(KompleksanBroj z)
71  {
72      return sqrt(z.real * z.real + z.imag * z.imag);
73  }
74
75  /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
76     odgovara kompleksnom broju argumentu */
77  KompleksanBroj konjugovan(KompleksanBroj z)
78  {
79      /* Konjugovano kompleksan broj z se dobija tako sto se promeni
80         znak imaginarnom delu kompleksnog broja */
81      KompleksanBroj z1 = z;
82      z1.imag *= -1;
83
84      return z1;
85  }
86
87  /* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
88     argumenata funkcije */
89  KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
90  {
91      /* Rezultat sabiranja dva kompleksna broja z1 i z2 je kompleksan
92         broj ciji je realan deo zbir realnih delova kompleksnih
93         brojeva z1 i z2, a imaginaran deo zbir imaginarnih delova
94         kompleksnih brojeva z1 i z2 */
95      KompleksanBroj z = z1;
96      z.real += z2.real;
97      z.imag += z2.imag;
98
99      return z;
100 }
101
102 /* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
103     argumenata funkcije */
104 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)
105 {
106     /* Rezultat oduzimanja dva kompleksna broja z1 i z2 je kompleksan
107        broj ciji je realan deo razlika realnih delova kompleksnih
108        brojeva z1 i z2, a imaginaran deo razlika imaginarnih delova
109        kompleksnih brojeva z1 i z2 */
110     KompleksanBroj z = z1;
111     z.real -= z2.real;
112     z.imag -= z2.imag;
```

```
114     return z;
115 }
116
117 /* Funkcija vraca kompleksan broj cija vrednost je jednaka
118    proizvodu argumenata funkcije */
119 KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
120 {
121     /* Rezultat mnozenja dva kompleksna broja z1 i z2 je kompleksan
122        broj ciji se realan i imaginaran deo racunaju po formuli za
123        mnozenje kompleksnih brojeva z1 i z2 */
124     KompleksanBroj z;
125     z.real = z1.real * z2.real - z1.imag * z2.imag;
126     z.imag = z1.real * z2.imag + z1.imag * z2.real;
127
128     return z;
129 }
130
131 /* Funkcija vraca argument zadatog kompleksnog broja */
132 float argument(KompleksanBroj z)
133 {
134     /* Argument kompleksnog broja z se racuna pozivanjem funkcije
135        atan2 iz biblioteke math.h */
136     return atan2(z.imag, z.real);
137 }
138
139 int main()
140 {
141     char c;
142
143     /* Deklaracija 3 promenljive tipa KompleksanBroj */
144     KompleksanBroj z1, z2, z;
145
146     /* Ucitava se prvi kompleksni broj, koji se potom ispisuje na
147        standardni izlaz */
148     ucitaj_kompleksan_broj(&z1);
149     ispisi_kompleksan_broj(z1);
150     printf("\n");
151
152     /* Ucitava se drugi kompleksni broj, koji se potom ispisuje na
153        standardni izlaz */
154     ucitaj_kompleksan_broj(&z2);
155     ispisi_kompleksan_broj(z2);
156     printf("\n");
157
158     /* Ucitavase znak na osnovu koga korisnik bira aritmeticku
159        operaciju koja ce se izvorsiti nad kompleksnim brojevima, a
160        zatim se vrsi provera da li je unet neki od dozvoljenih
161        aritmetickih znakova. */
162     getchar();
163     printf("Unesite znak (+,-,*): ");
164     scanf("%c", &c);
165     if (c != '+' && c != '-' && c != '*') {
```

## 1 Uvodni zadaci

```
166     fprintf(stderr, "Greska: Nedozvoljena vrednost operatora.\n");
167     exit(EXIT_FAILURE);
168 }
169
170 /* Analizira se uneti operator */
171 if (c == '+') {
172     /* Racuna se zbir */
173     z = saberi(z1, z2);
174 } else if (c == '-') {
175     /* Racuna se razlika */
176     z = oduzmi(z1, z2);
177 } else {
178     /* Racuna se proizvod */
179     z = mnozi(z1, z2);
180 }
181
182 /* Ispisuje se rezultat */
183 ispisi_kompleksan_broj(z1);
184 printf(" %c ", c);
185 ispisi_kompleksan_broj(z2);
186 printf(" = ");
187 ispisi_kompleksan_broj(z);
188
189 /* Ispisuje se realan, imaginaran deo i moduo prvog kompleksnog
190    broja */
191 printf("\nRealni_deo: %.f\nImaginarni_deo: %.f\nModuo: %.f\n",
192        realan_deo(z), imaginaran_deo(z), moduo(z));
193
194 /* Izracunava se i ispisuje konjugovano kompleksan broj drugog
195    kompleksnog broja */
196 printf("Konjugovano kompleksan broj: ");
197 ispisi_kompleksan_broj(konjugovan(z));
198 printf("\n");
199
200 /* Testira se funkcija koja racuna argument kompleksnog broja */
201 printf("Argument kompleksnog broja: %.f\n", argument(z));
202
203 exit(EXIT_SUCCESS);
204 }
```

### Rešenje 1.2

*kompleksan\_broj.h*

```
1 /* Zaglavlje kompleksan_broj.h sadrzi definiciju tipa
2    KomplexanBroj i deklaracije funkcija za rad sa kompleksnim
3    brojevima. Zaglavlje nikada ne treba da sadrzi definicije
4    funkcija. Da bi neki program mogao da koristi ove brojeve i
5    funkcije iz ove biblioteke, neophodno je da ukljuci ovo
6    zaglavlje. */
```

```
7  /* Ovim pretprocesorskim direktivama se zakljucava zaglavlje i
9  onemogucava se da se sadrzaj zaglavlja vise puta ukljuci. Niska
   posle kljucne reci ifndef je proizvoljna, ali treba da se ponovi
11  u narednoj pretprocesorskoj define direktivi. */
   #ifndef _KOMPLEKSAN_BROJ_H
13  #define _KOMPLEKSAN_BROJ_H

15  /* Struktura KompleksanBroj */
   typedef struct {
17     float real;
     float imag;
19 } KompleksanBroj;

21 /* Deklaracije funkcija za rad sa kompleksnim brojevima. Sve one su
   definisane u kompleksan_broj.c */

23 /* Funkcija ucitava sa standardnog ulaza realan i imaginara deo
25 kompleksnog broja i smesta ih u strukturu cija je adresa
   argument funkcije */
27 void ucitaj_kompleksan_broj(KompleksanBroj * z);

29 /* Funkcija ispisuje na standardan izlaz zadati kompleksni broj u
   obliku (x + i y) */
31 void ispisi_kompleksan_broj(KompleksanBroj z);

33 /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
   float realan_deo(KompleksanBroj z);

35 /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
   float imaginaran_deo(KompleksanBroj z);

37 /* Funkcija vraca vrednost modula zadatog kompleksnog broja */
   float moduo(KompleksanBroj z);

41 /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
43 odgovara kompleksnom broju argumentu */
   KompleksanBroj konjugovan(KompleksanBroj z);

45 /* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
47 argumenata funkcije */
   KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2);

49 /* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
51 argumenata funkcije */
   KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2);

53 /* Funkcija vraca kompleksan broj cija vrednost je jednaka
55 proizvodu argumenata funkcije */
   KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2);

57 /* Funkcija vraca argument zadatog kompleksnog broja */
```

## 1 Uvodni zadaci

---

```
59 float argument(KompleksanBroj z);  
61 /* Kraj zakljucanog dela */  
#endif
```

*kompleksan\_broj.c*

```
#include <stdio.h>  
2 #include <math.h>  
  
4 /* Ukljucuje se zaglavlje za rad sa kompleksnim brojevima, jer je  
   neophodno da bude poznata definicija tipa KompleksanBroj. */  
6 #include "kompleksan_broj.h"  
  
8 void ucitaj_kompleksan_broj(KompleksanBroj * z)  
{  
10 /* Ucitavanje vrednosti sa standardnog ulaza */  
   printf("Unesite realan i imaginaran deo kompleksnog broja: ");  
12   scanf("%f", &z->real);  
   scanf("%f", &z->imag);  
14 }  
  
16 void ispisi_kompleksan_broj(KompleksanBroj z)  
{  
18 /* Zapocinje se sa ispisom */  
   printf("(");  
20  
   /* Razlikuju se dva slucaja: 1) realni deo kompleksnog broja  
   razlicit od nule: tada se realni deo ispisuje na standardni  
   izlaz, nakon cega se ispisuje znak + ili - u zavisnosti da li  
   je imaginarni deo pozitivan ili negativan, a potom i apsolutna  
   vrednost imaginarnog dela kompleksnog broja 2) realni deo  
   kompleksnog broja je nula: tada se samo ispisuje imaginaran  
   deo, s tim sto se ukoliko su oba dela nula ispisuje samo 0, bez  
   decimalnih mesta */  
28   if (z.real != 0) {  
30     printf("%.2f", z.real);  
  
32     if (z.imag > 0)  
       printf(" + %.2f i", z.imag);  
34     else if (z.imag < 0)  
       printf(" - %.2f i", -z.imag);  
36   } else {  
     if (z.imag == 0)  
       printf("0");  
38     else  
       printf("%.2f i", z.imag);  
40   }  
42  
   /* Zavrшава se sa ispisom */  
44   printf(")");
```



```

}
46
float realan_deo(KompleksanBroj z)
48 {
    /* Vraca se vrednost realnog dela kompleksnog broja */
50     return z.real;
}
52
float imaginaran_deo(KompleksanBroj z)
54 {
    /* Vraca se vrednost imaginarnog dela kompleksnog broja */
56     return z.imag;
}
58
float moduo(KompleksanBroj z)
60 {
    /* Koriscenjem funkcije sqrt racuna se moduo kompleksnog broja */
62     return sqrt(z.real * z.real + z.imag * z.imag);
}
64
KompleksanBroj konjugovan(KompleksanBroj z)
66 {
    /* Konjugovano kompleksan broj se dobija od datog broja z tako
68     sto se promeni znak imaginarnom delu kompleksnog broja */
    KompleksanBroj z1 = z;
70     z1.imag *= -1;
    return z1;
72 }
74
KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
{
76     /* Rezultat sabiranja dva kompleksna broja z1 i z2 je kompleksan
    broj ciji je realan deo zbir realnih delova kompleksnih
78     brojeva z1 i z2, a imaginaran deo zbir imaginarnih delova
    kompleksnih brojeva z1 i z2 */
80     KompleksanBroj z = z1;
    z.real += z2.real;
82     z.imag += z2.imag;

84     return z;
}
86
KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)
88 {
    /* Rezultat oduzimanja dva kompleksna broja z1 i z2 je kompleksan
90     broj ciji je realan deo razlika realnih delova kompleksnih
    brojeva z1 i z2, a imaginaran deo razlika imaginarnih delova
92     kompleksnih brojeva z1 i z2 */
    KompleksanBroj z = z1;
94     z.real -= z2.real;
    z.imag -= z2.imag;
96

```

## 1 Uvodni zadaci

---

```
    return z;
98 }

KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2)
{
100     /* Rezultat mnozenja dva kompleksna broja z1 i z2 je kompleksan
102        broj ciji se realan i imaginaran deo racunaju po formuli za
104        mnozenje kompleksnih brojeva z1 i z2 */
    KompleksanBroj z;
106     z.real = z1.real * z2.real - z1.imag * z2.imag;
    z.imag = z1.real * z2.imag + z1.imag * z2.real;
108
    return z;
110 }

112 float argument(KompleksanBroj z)
{
114     /* Argument kompleksnog broja z se racuna pozivanjem funkcije
        atan2 iz biblioteke math.h */
116     return atan2(z.imag, z.real);
}
```

*main.c*

```
1  /*****
   Ovaj program koristi korektno definisanu biblioteku kompleksnih
3  brojeva. U zaglavlju kompleksan_broj.h nalazi se definicija
   kompleksnog broja i popis deklaracija podrzanih funkcija, a u
5  kompleksan_broj.c se nalaze njihove definicije.

7  Kompilacija programa se najjednostavnije postize naredbom
   gcc -Wall -lm -o kompleksan_broj kompleksan_broj.c main.c
9
   Kompilacija se moze uraditi i postupno, na sledeci nacin:
11 gcc -Wall -c -o kompleksan_broj.o kompleksan_broj.c
   gcc -Wall -c -o main.o main.c
13 gcc -lm -o kompleksan_broj kompleksan_broj.o main.o

15 Napomena: Prethodne komande se koriste kada se sva tri navedena
   dokumenta nalaze u istom direktorijumu. Ukoliko se biblioteka (npr.
17 kompleksan_broj.c kompleksan_broj.h) nalazi u direktorijumu sa imenom
   header_dir prevodjenje se vrsi dodavanjem opcije -I header_dir
19 gcc -I header_dir -Wall -lm -o kompleksan_broj kompleksan_broj.c
   main.c
21 *****/

23 #include <stdio.h>
   /* Ukljucuje se zaglavlje za rad sa kompleksnim brojevima */
25 #include "kompleksan_broj.h"

27 /* U glavnoj funkciji se za uneti kompleksan broj ispisuje njegov
```

```

    polarni oblik */
29 int main()
    {
31     KompleksanBroj z;

33     /* Ucitava se kompleksan broj */
    ucitaj_kompleksan_broj(&z);

35     /* Ispisuje se polarni oblik kompleksnog broja */
37     printf("Polarni oblik kompleksnog broja je %.2f * e~i * %.2f\n",
        moduo(z), argument(z));

39     return 0;
41 }

```

### Rešenje 1.3

*polinom.h*

```

1  #ifndef _POLINOM_H
   #define _POLINOM_H
3
   #include <stdio.h>
5  #include <stdlib.h>

7  /* Maksimalni stepen polinoma */
   #define MAKS_STEPEN 20
9
11 /* Polinomi se predstavljaju strukturom koja cuva koeficijente
    (koef[i] je koeficijent uz clan x^i) i stepen polinoma */
   typedef struct {
13     double koef[MAKS_STEPEN + 1];
        int stepen;
15 } Polinom;

17 /* Funkcija koja ispisuje polinom na standardni izlaz u citljivom
    obliku. Polinom se prenosi po adresi da bi se uštedela memorija:
19     ne kopira se cela struktura, vec se samo prenosi adresa na kojoj
    se nalazi polinom koji ispisujemo */
21 void ispisi(const Polinom * p);

23 /* Funkcija koja ucitava polinom sa standardnog ulaza */
   Polinom ucitaj();

25
27 /* Funkcija racuna i vraca vrednost polinoma p u tacki x Hornerovim
    algoritmom */
   double izracunaj(const Polinom * p, double x);

29
31 /* Funkcija koja sabira dva polinoma */
   Polinom saberi(const Polinom * p, const Polinom * q);

```

## 1 Uvodni zadaci

---

```
33 /* Funkcija koja mnozi dva polinoma p i q */
   Polinom pomnozi(const Polinom * p, const Polinom * q);
35
   /* Funkcija koja racuna izvod polinoma p */
37 Polinom izvod(const Polinom * p);
39
   /* Funkcija koja racuna n-ti izvod polinoma p */
   Polinom n_izvod(const Polinom * p, int n);
41 #endif
```

*polinom.c*

```
1 #include <stdio.h>
   #include <stdlib.h>
3 #include "polinom.h"

5 void ispisi(const Polinom * p)
   {
7     int nulaPolinom = 1;
       int i;
9     /* Ispisivanje polinoma pocinje od najviseg stepena ka najnižem
       da bi polinom bio ispisan na prirodan nacin. Ispisuju se samo
11     oni koeficijenti koji su razliciti od nule. Ispred pozitivnih
       koeficijenata je potrebno ispisati znak + (osim u slucaju
13     koeficijenta uz najvisi stepen). */
       for (i = p->stepen; i >= 0; i--) {
15
           if (p->koef[i]) {
17               /* Polinom nije nula polinom, cim je neki od koeficijenata
                  razlicit od nule */
19               nulaPolinom = 0;
               if (p->koef[i] >= 0 && i != p->stepen)
21                   putchar('+');
               if (i > 1)
23                   printf("%.2fx~%d", p->koef[i], i);
               else if (i == 1)
25                   printf("%.2fx", p->koef[i]);
               else
27                   printf("%.2f", p->koef[i]);
           }
29     }
       /* U slucaju nula polinoma indikator ce imati vrednost 1 i tada
       se ispisuje nula. */
31     if (nulaPolinom)
33         printf("0");
       putchar('\n');
35 }

37 Polinom ucitaj()
   {
```

```

39  int i;
    Polinom p;

41

    printf("Unesite polinom p (prvo stepen, pa zatim koeficijente");
43  printf(" od najveceg stepena do nultog):\n");

45  /* Ucitava se stepena polinoma */
    scanf("%d", &p.stepen);

47

    /* Ponavlja se učitavanje stepena sve dok se ne unese stepen iz
49  dozvoljenog opsega */
    while (p.stepen > MAKS_STEPEN || p.stepen < 0) {
51        printf("Stepen polinoma pogresno unet, pokušajte ponovo: ");
        scanf("%d", &p.stepen);
53    }

55    /* Unose se koeficijenti polinoma */
    for (i = p.stepen; i >= 0; i--)
57        scanf("%lf", &p.koef[i]);

59    /* Vraca se procitani polinom */
    return p;
61 }

63 double izracunaj(const Polinom * p, double x)
{
65     /* Primer: Hornerov algoritam za polinom  $x^4+2x^3+3x^2+2x+1$ :
         $x^4+2x^3+3x^2+2x+1 = ((x+2)*x + 3)*x + 2)*x + 1$  */

67     double rezultat = 0;
69     int i = p->stepen;

71     /* Rezultat se na pocetku inicijalizuje na nulu, a potom se u
        svakoj iteraciji najpre mnozi sa x, a potom i uvecava za
73     vrednost odgovarajuceg koeficijenta */
    for (; i >= 0; i--)
75        rezultat = rezultat * x + p->koef[i];
    return rezultat;
77 }

79 Polinom saberi(const Polinom * p, const Polinom * q)
{
81     Polinom r;
    int i;

83

    /* Stepen rezultata ce odgovarati stepenu polinoma sa vecim
85     stepenom */
    r.stepen = p->stepen > q->stepen ? p->stepen : q->stepen;

87

    /* Racunaju se svi koeficijenti rezultujucega polinoma tako sto se
89     sabiraju koeficijenti na odgovarajucim pozicijama polinoma koje
        sabiramo. Ukoliko je pozicija za koju se racuna koeficijent veca

```

```

91     od stepena nekog od polaznih polinoma podrazumeva se da je
    koeficijent jednak koeficijentu uz odgovarajuci stepen iz drugog
93     polinoma */
    for (i = 0; i <= r.stepen; i++)
95         r.koef[i] =
            (i > p->stepen ? 0 : p->koef[i]) +
97             (i > q->stepen ? 0 : q->koef[i]);

99     /* Vraca se izracunati polinom */
    return r;
101 }

103 Polinom pomnozi(const Polinom * p, const Polinom * q)
{
105     int i, j;
    Polinom r;

107     /* Stepen rezultata odgovara zbiru stepena polaznih polinoma */
    r.stepen = p->stepen + q->stepen;
109     if (r.stepen > MAKS_STEPEN) {
        fprintf(stderr, "Stepen proizvoda polinoma izlazi iz opsega\n");
111         exit(EXIT_FAILURE);
    }
113 }

115 /* Svi koeficijenti rezultujuceg polinoma se inicijalizuju na 0 */
    for (i = 0; i <= r.stepen; i++)
117         r.koef[i] = 0;

119     /* U svakoj iteraciji odgovarajuci koeficijent rezultata se
        uvecava za proizvod odgovarajucih koeficijenata iz polaznih
121         polinoma */
    for (i = 0; i <= p->stepen; i++)
123         for (j = 0; j <= q->stepen; j++)
            r.koef[i + j] += p->koef[i] * q->koef[j];

125     /* Vraca se izracunati polinom */
127     return r;
}

129 Polinom izvod(const Polinom * p)
{
131     int i;
    Polinom r;

133     /* Izvod polinoma ce imati stepen za jedan stepen manji od
        stepena polaznog polinoma. Ukoliko je stepen polinoma p vec
135         nula, onda je rezultujuci polinom nula (izvod od konstante je
        nula). */
137     if (p->stepen > 0) {
        r.stepen = p->stepen - 1;
139
        /* Racunanje koeficijenata rezultata na osnovu koeficijenata
141

```

```

143     polaznog polinoma */
144     for (i = 0; i <= r.stepen; i++)
145         r.koef[i] = (i + 1) * p->koef[i + 1];
146 } else
147     r.koef[0] = r.stepen = 0;

149 /* Vraca se izracunati polinom */
150 return r;
151 }

153 Polinom n_izvod(const Polinom * p, int n)
154 {
155     int i;
156     Polinom r;

157     /* Provera da li je n nenegativna vrednost */
158     if (n < 0) {
159         fprintf(stderr, "U n-tom izvodu polinoma, n mora biti >=0 \n");
160         exit(EXIT_FAILURE);
161     }

162     /* Multi izvod je bas taj polinom */
163     if (n == 0)
164         return *p;

165     /* Za n>=1, n-ti izvod se racuna tako sto se n puta pozove
166        funkcija za racunanje prvog izvoda polinoma */
167     r = izvod(p);
168     for (i = 1; i < n; i++)
169         r = izvod(&r);

170     /* Vraca se dobijeni polinom */
171     return r;
172 }

```

*main.c*

```

#include <stdio.h>
#include "polinom.h"

int main(int argc, char **argv)
{
    Polinom p, q, z, r;
    double x;
    int n;

    /* Unosi se polinom p */
    p = ucitaj();

    /* Ispisuje se polinom p */
    ispisi(&p);

```

```
16  /* Unosi se polinom q */
    q = ucitaj();

18

20  /* Polinomi se sabiraju i ispisuje se izracunati zbir */
    z = saberi(&p, &q);
    printf("Zbir polinoma je polinom z:\n");
    ispisi(&z);

22

24  /* Polinomi se mnoze i ispisuje se izracunati proizvod */
    r = pomnozi(&p, &q);
    printf("Prozvod polinoma je polinom r:\n");
    ispisi(&r);

26

28

30  /* Ispisuje se vrednost polinoma u unetoj tacki */
    printf("Unesite tacku u kojoj racunate vrednost polinoma z:\n");
    scanf("%lf", &x);
    printf("Vrednost polinoma z u tacki %.2f je %.2f\n", x,
           izracunaj(&z, x));

32

34

36  /* Racuna se n-ti izvod polinoma i ispisuje se rezultat */
    printf("Unesite izvod polinoma koji zelite:\n");
    scanf("%d", &n);
    r = n_izvod(&r, n);
    printf("%d. izvod polinoma r je: ", n);
    ispisi(&r);

38

40

42  exit(EXIT_SUCCESS);
}
```

### Rešenje 1.5

*stampanje\_bitova.h*

```
1  #ifndef _STAMPANJE_BITOVA_H
2  #define _STAMPANJE_BITOVA_H
3
4  #include <stdio.h>
5
6  /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
7   celog broja u memoriji. Bitove koji predstavljaju binarnu
8   reprezentaciju broja treba ispisati sa leva na desno, tj. od
9   bita najvece tezine ka bitu najmanje tezine */
10 void stampaj_bitove(unsigned x);
11
12 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
13 celog broja tipa 'short' u memoriji. */
14 void stampaj_bitove_short(short x);
15
16 /* Funkcija prikazuje na standardni izlaz binarnu reprezentaciju
```



```

17     karaktera u memoriji. */
void stampaj_bitove_char(char x);
19
#endif

```

*stampanje\_bitova.c*

```

#include <stdio.h>
2 #include "stampanje_bitova.h"

4 void stampaj_bitove(unsigned x)
{
6     /* Broj bitova celog broja */
    unsigned velicina = sizeof(unsigned) * 8;

8     /* Maska koja se koristi za ocitavanje bitova celog broja */
    unsigned maska;

10     /* Pocetna vrednost maske se postavlja na broj ciji binarni zapis
        na mestu bita najvece tezine sadrzi jedinicu, a na svim ostalim
        mestima sadrzi nulu. U svakoj iteraciji maska se menja tako sto
12     se jedini bit jedinica pomera udesno, kako bi se odredio
        naredni bit broja x koji je argument funkcije. Zatim se
        odgovarajuca cifra, ('0' ili '1'), ispisuje na standardni
14     izlaz. Neophodno je da promenljiva maska bude deklarirana kao
        neoznacena ceo broj kako bi se pomeranjem u desno vrsilo logicko
        pomeranje (popunjavanje nulama), a ne aritmeticko pomeranje
        (popunjavanje znakom broja). */
16     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
        putchar(x & maska ? '1' : '0');

22     putchar('\n');
24 }

26 void stampaj_bitove_short(short x)
{
28     /* Broj bitova celog broja tipa short */
    unsigned velicina = sizeof(short) * 8;

30     /* Maska koja se koristi za ocitavanje bitova broja tipa short */
    unsigned short maska;

32     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
        putchar(x & maska ? '1' : '0');

34     putchar('\n');
36 }

38 void stampaj_bitove_char(char x)
{
40     /* Broj bitova karaktera */
42     /* Broj bitova karaktera */
44

```

## 1 Uvodni zadaci

---

```
    unsigned velicina = sizeof(char) * 8;
46
    /* Maska koja se koristi za ocitavanje bitova jednog karaktera */
48    unsigned char maska;

50    for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
        putchar(x & maska ? '1' : '0');

52    putchar('\n');
54 }
```

*main.c*

```
#include <stdio.h>
2 #include "stampanje_bitova.h"

4 int main()
{
6     int broj_int;
    short broj_short;
8     char broj_char;

10    /* Ucitava se broj tipa int */
    printf("Unesite broj tipa int: ");
12    scanf("%x", &broj_int);

14    /* Ispisuje se binarna reprezentacija unetog broja */
    printf("Binarna reprezentacija: ");
16    stampaj_bitove(broj_int);

18    /* Ucitava se broj tipa short */
    printf("Unesite broj tipa short: ");
20    scanf("%hx", &broj_short);

22    /* Ispisuje se binarna reprezentacija unetog broja */
    printf("Binarna reprezentacija: ");
24    stampaj_bitove_short(broj_short);

26    /* Ucitava se broj tipa char */
    printf("Unesite broj tipa char: ");
28    scanf("%hcx", &broj_char);

30    /* Ispisuje se binarna reprezentacija unetog broja */
    printf("Binarna reprezentacija: ");
32    stampaj_bitove_char(broj_char);

34    return 0;
}
```

## Rešenje 1.6

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
5     kreiranjem odgovarajuće maske i njenim pomeranjem */
6  int prebroj_bitove_1(int x)
7  {
8      int br = 0;
9      unsigned broj_pomeranja = sizeof(unsigned) * 8 - 1;
10
11     /* Formiranje se maska čija binarna reprezentacija izgleda
12        100000...0000000, koja služi za očitavanje bita najveće
13        težine. U svakoj iteraciji maska se pomera u desno za 1 mesto,
14        i očitava se sledeći bit. Petlja se završava kada više nema
15        jedinica tj. kada maska postane nula. */
16     unsigned maska = 1 << broj_pomeranja;
17     for (; maska != 0; maska >>= 1)
18         x & maska ? br++ : 1;
19
20     return br;
21 }
22
23 /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
24     formiranjem odgovarajuće maske i pomeranjem promenljive x */
25 int prebroj_bitove_2(int x)
26 {
27     int br = 0;
28     unsigned broj_pomeranja = sizeof(int) * 8 - 1;
29
30     /* Kako je argument funkcije označen ceo broj x naredba x>>=1
31        vrši aritmetičko pomeranje u desno, tj. popunjavanje bita
32        najveće težine bitom znaka. U tom slučaju, za negativnu
33        vrednost promenljive x, nikad ne bi bio ispunjen uslov x!=0 i
34        program bi bio zarobljen u beskonacnoj petlji. Zbog toga se
35        koristi pomeranje broja x ulevo i maska koja očitava bit
36        najveće težine. */
37     unsigned maska = 1 << broj_pomeranja;
38     for (; x != 0; x <<= 1)
39         x & maska ? br++ : 1;
40
41     return br;
42 }
43
44 int main()
45 {
46     int x, i;
47
48     /* Učitava se broj sa ulaza */
49     printf("Unesite broj:\n");
50     scanf("%x", &x);
```

## 1 Uvodni zadaci

```
51  /* Dozvoljava se korisniku da bira na koji nacin ce biti
53     izracunat broj jedinica u zapisu broja */
printf("Unesite redni broj funkcije:\n");
55  scanf("%d", &i);

57  /* Ispisuje se odgovarajuci rezultat na osnovu unesenog rednog
     broja funkcije */
59  if (i == 1) {
    printf("Poziva se funkcija prebroj_bitove_1\n");
    printf("Broj jedinica u zapisu je %d\n", prebroj_bitove_1(x));
61  } else if (i == 2) {
    printf("Poziva se funkcija prebroj_bitove_2\n");
    printf("Broj jedinica u zapisu je %d\n", prebroj_bitove_2(x));
63  } else {
    fprintf(stderr,
65         "Greska: Neodgovarajuci redni broj funkcije.\n");
    exit(EXIT_FAILURE);
67  }
69

71  exit(EXIT_SUCCESS);
}
```

### Rešenje 1.7

NAPOMENA: Rešenje koristi biblioteku za štampanje bitova iz zadatka 1.5.

```
1  #include <stdio.h>
   #include "stampanje_bitova.h"

3
   /* Funkcija vraća najveći neoznačeni broj sastavljen od istih
5     bitova koji se nalaze u binarnoj reprezentaciji vrednosti
     promenljive x */
7  unsigned najveći(unsigned x)
   {
9     unsigned velicina = sizeof(unsigned) * 8;

11    /* Formira se maska 100000...0000000 */
    unsigned maska = 1 << (velicina - 1);

13
    /* Rezultat se inicijalizuje vrednošću 0 */
15    unsigned rezultat = 0;

17    /* Promenljiva x se pomera u levo sve dok postoje jedinice u
     njenoj binarnoj reprezentaciji (tj. sve dok je promenljiva x
19     različita od nule). */
    for (; x != 0; x <<= 1) {
21        /* Za svaku jedinicu koja se korišćenjem maske detektuje na
         poziciji najveće težine u binarnoj reprezentaciji promenljive
23         x, potiskuje se jedna nova jedinica sa leva u rezultat */
        if (x & maska) {
```

```
25     rezultat >>= 1;
26     rezultat |= maska;
27 }
28 }
29
30 /* Vraca se izracunata vrednost */
31 return rezultat;
32 }
33
34 /* Funkcija vraca najmanji neoznaceni broj sastavljen od istih
35 bitova koji se nalaze u binarnoj reprezentaciji vrednosti
36 promenjive x */
37 unsigned najmanji(unsigned x)
38 {
39     /* Rezultat se inicijalizuje vrednoscu 0 */
40     unsigned rezultat = 0;
41
42     /* Promenljiva x se pomera u desno sve dok postoje jedinice u
43 njenoj binarnoj reprezentaciji (tj. sve dok je promenljiva x
44 razlicita od nule). */
45 for (; x != 0; x >>= 1) {
46     /* Za svaku jedinicu koja se koriscenjem vrednosti 1 za masku
47 detektuje na poziciji najmanje tezine u binarnoj
48 reprezentaciji promenjive x, potiskuje se jedna nova
49 jedinicu sa desna u rezultat */
50     if (x & 1) {
51         rezultat <<= 1;
52         rezultat |= 1;
53     }
54 }
55
56 /* Vraca se izracunata vrednost */
57 return rezultat;
58 }
59
60 int main()
61 {
62     int broj;
63
64     /* Ucitava se broj sa ulaza */
65     scanf("%x", &broj);
66
67     /* Ispisuju se, redom, najveći i najmanji broj formirani od
68 bitova unetog broja */
69     printf("Najveci:\n");
70     stampaj_bitove(najveci(broj));
71
72     printf("Najmanji:\n");
73     stampaj_bitove(najmanji(broj));
74
75     return 0;
76 }
```

### Rešenje 1.8

NAPOMENA: Rešenje koristi biblioteku za štampanje bitova iz zadatka 1.5.

```
1  #include <stdio.h>
2  #include "stampanje_bitova.h"
3
4  /* Funkcija postavlja na nulu n bitova pocev od pozicije p */
5  unsigned postavi_0(unsigned x, unsigned n, unsigned p)
6  {
7      /******
8       Formira se maska cija binarna reprezentacija ima n bitova
9       postavljenih na 0 pocev od pozicije p, dok su svi ostali
10      postavljeni na 1. Na primer, za n=5 i p=10 formira se maska oblika
11      1111 1111 1111 1111 1111 1000 0011 1111
12      To se postize na sledeci nacin:
13      ~0                      1111 1111 1111 1111 1111 1111 1111 1111
14      (~0 << n)                1111 1111 1111 1111 1111 1111 1110 0000
15      ~(~0 << n)              0000 0000 0000 0000 0000 0000 0001 1111
16      ~(~0 << n) << (p-n+1)) 0000 0000 0000 0000 0000 0111 1100 0000
17      ~(~0 << n) << (p-n+1)) 1111 1111 1111 1111 1111 1000 0011 1111
18      *****/
19      unsigned maska = ~(~0 << n) << (p - n + 1));
20
21      return x & maska;
22  }
23
24  /* Funkcija postavlja na jedinicu n bitova pocev od pozicije p */
25  unsigned postavi_1(unsigned x, unsigned n, unsigned p)
26  {
27      /******
28       Formira se maska kod koje je samo n bitova pocev od pocev od
29       pozicije p jednako 1, a ostali su 0.
30       Na primer, za n=5 i p=10 formira se maska oblika
31       0000 0000 0000 0000 0000 0111 1100 0000
32      *****/
33      unsigned maska = ~(~0 << n) << (p - n + 1);
34
35      return x | maska;
36  }
37
38  /* Funkcija vraca celobrojno polje bitova, desno poravnato, koje
39     predstavlja n bitova pocev od pozicije p u binarnoj
40     reprezentaciji broja x */
41  unsigned vrati_bitove(unsigned x, unsigned n, unsigned p)
42  {
43      /******
44       Kreira se maska kod koje su poslednjih n bitova 1, a ostali su 0.
45       Na primer, za n=5
46       0000 0000 0000 0000 0000 0000 0001 1111
47      *****/
48      unsigned maska = ~(~0 << n);
```

```

50  /* Najpre se vrednost promenljive x pomera u desno tako da
    trazen polje bude uz desni kraj. Zatim se maskiraju ostali
52  bitovi, sem zeljenih n i funkcija vraca tako izracunatu
    vrednost */
54  return maska & (x >> (p - n + 1));
    }

56  /* Funkcija vraca broj x kome su n bitova pocev od pozicije p
58  postavljeni na vrednosti n bitova najmanje tezine binarne
    reprezentacije broja y */
60  unsigned postavi_1_n_bitova(unsigned x, unsigned n, unsigned p,
                                unsigned y)
62  {
    /* Kreira se maska kod koje su poslednjih n bitova 1, a
64    ostali su 0. */
    unsigned poslednjih_n_1 = ~(~0 << n);

66    /* Kao i kod funkcije postavi_0, i ovde se kreira maska koja ima
68    n bitova postavljenih na 0 pocevsi od pozicije p, dok su
    ostali bitovi 1 */
70    unsigned srednjih_n_0 = ~(~(~0 << n) << (p - n + 1));

72    /* U promenljivu x_postavi_0 se smesta vrednost dobijena kada se
    u binarnoj reprezentaciji vrednosti promenljive x postavi na 0
74    n bitova na pozicijama pocev od p */
    unsigned x_postavi_0 = x & srednjih_n_0;

76    /* U promenljivu y_pomeri_srednje se smesta vrednost dobijena od
    binarne reprezentacije vrednosti promenljive y ciji je n
78    bitova najnize tezine pomera tako da stoje pocev od pozicije
    p. Ostali bitovi su nule. Sa (y & poslednjih_n_1) postave na
80    0 svi bitovi osim najnižih n */
82    unsigned y_pomeri_srednje = (y & poslednjih_n_1) << (p - n + 1);

84    return x_postavi_0 ^ y_pomeri_srednje;
    }

86  /* Funkcija invertuje bitove u zapisu broja x pocevsi od pozicije p
88    njih n */
    unsigned invertuj(unsigned x, unsigned n, unsigned p)
90  {
    /* Formira se maska sa n jedinica pocev od pozicije p */
92    unsigned maska = ~(~0 << n) << (p - n + 1);

94    /* Operator ekskluzivno ili invertuje sve bitove gde je
    odgovarajuci bit maske 1. Ostali bitovi ostaju nepromenjeni. */
96    return maska ^ x;
    }

98  int main()
100 {
    unsigned x, p, n, y;

```

```

102  /* Ucitavaju se vrednosti sa standardnog ulaza */
104  printf("Unesite neoznaceni broj x:\n");
106  scanf("%u", &x);
108  printf("Unesite neoznaceni broj n:\n");
110  scanf("%u", &n);
112  printf("Unesite neoznaceni broj p:\n");
114  scanf("%u", &p);
116  printf("Unesite neoznaceni broj y:\n");
118  scanf("%u", &y);

120  /* Ispisuju se binarne reprezentacije broja x i broja koji se
122     izracunava funkcijom postavi_0 sa argumentima x, n i p */
124  printf("x = %10u %36s = ", x, "");
126  stampaj_bitove(x);
128  printf("postavi_0(%10u,%6u,%6u)%16s = ", x, n, p, "");
130  stampaj_bitove(postavi_0(x, n, p));
132  printf("\n");

134  /* Ispisuju se binarne reprezentacije broja x i broja koji se
136     izracunava funkcijom postavi_1 sa argumentima x, n i p */
138  printf("x = %10u %36s = ", x, "");
140  stampaj_bitove(x);
142  printf("postavi_1(%10u,%6u,%6u)%16s = ", x, n, p, "");
144  stampaj_bitove(postavi_1(x, n, p));
146  printf("\n");

148  /* Ispisuju se binarne reprezentacije broja x i broja koji se
150     izracunava funkcijom vrati_bitove sa argumentima x, n i p */
152  printf("x = %10u %36s = ", x, "");
154  stampaj_bitove(x);
156  printf("vrati_bitove(%10u,%6u,%6u)%13s = ", x, n, p, "");
158  stampaj_bitove(vrati_bitove(x, n, p));
160  printf("\n");

162  /* Ispisuju se binarne reprezentacije brojeva x, y i broja koji
164     se izracunava funkcijom postavi_1_n_bitova sa argumentima x,
166     n, p */
168  printf("x = %10u %36s = ", x, "");
170  stampaj_bitove(x);
172  printf("y = %10u %36s = ", y, "");
174  stampaj_bitove(y);
176  printf("postavi_1_n_bitova(%10u,%4u,%4u,%10u) = ", x, n, p, y);
178  stampaj_bitove(postavi_1_n_bitova(x, n, p, y));
180  printf("\n");

182  /* Ispisuju se binarne reprezentacije broja x i broja koji se
184     izracunava funkcijom invertuj sa argumentima x, n i p */
186  printf("x = %10u %36s = ", x, "");
188  stampaj_bitove(x);
190  printf("invertuj(%10u,%6u,%6u)%17s = ", x, n, p, "");
192  stampaj_bitove(invertuj(x, n, p));

```



```

154     return 0;
155 }

```

### Rešenje 1.9

NAPOMENA: Rešenje koristi biblioteku za štampanje bitova iz zadatka 1.5.

```

#include <stdio.h>
#include "stampanje_bitova.h"

/* Funkcija ceo broj x rotira ulevo za n mesta. */
unsigned rotiraj_ulevo(int x, unsigned n)
{
    unsigned bit_najvece_tezine;

    /* Maska koja ima samo bit na poziciji najvece tezine postavljen
       na 1, neophodna je da bi pre pomeranja x ulevo za 1, bit na
       poziciji najvece tezine bio sacuvan. */
    unsigned bit_najvece_tezine_maska =
        1 << (sizeof(unsigned) * 8 - 1);
    int i;

    /* n puta se vrši rotacija za jedan bit ulevo. U svakoj iteraciji
       se odredi bit na poziciji najvece tezine, a potom se pomera
       binarna reprezentacija trenutne vrednosti promenljive x ulevo
       za 1. Nakon toga, bit na poziciji najmanje tezine se postavlja
       na vrednost koju je imao bit na poziciji najvece tezine koji je
       istisnut pomeranjem */
    for (i = 0; i < n; i++) {
        bit_najvece_tezine = x & bit_najvece_tezine_maska;
        x = x << 1 | (bit_najvece_tezine ? 1 : 0);
    }

    /* Vraca se izracunata vrednost */
    return x;
}

/* Funkcija neoznaceni broj x rotira udesno za n mesta. */
unsigned rotiraj_udesno(unsigned x, unsigned n)
{
    unsigned bit_najmanje_tezine;
    int i;

    /* n puta se ponavlja rotacija udesno za jedan bit. U svakoj
       iteraciji se odredjuje bit na poziciji najmanje tezine broja
       x, zatim tako odredjeni bit se pomera ulevo tako da bit na
       poziciji najmanje tezine dodje do pozicije najvece tezine.
       Zatim, nakon pomeranja binarne reprezentacije trenutne
       vrednosti promenljive x za 1 udesno, bit na poziciji najvece
       tezine se postavlja na vrednost vec zapamcenog bita koji je

```

## 1 Uvodni zadaci

```
44     bio na poziciji najmanje tezine. */
45     for (i = 0; i < n; i++) {
46         bit_najmanje_tezine = x & 1;
47         x = x >> 1 | bit_najmanje_tezine << (sizeof(unsigned) * 8 - 1);
48     }

50     /* Vraca se izracunata vrednost */
51     return x;
52 }

54 /* Verzija funkcije koja broj x rotira udesno za n mesta, gde je
   argument funkcije x oznaceni ceo broj */
55 int rotiraj_udesno_oznaceni(int x, unsigned n)
56 {
57     unsigned bit_najmanje_tezine;
58     int i;

60     /* U svakoj iteraciji se odredjuje bit na poziciji najmanje
61     tezine i smesta u promenljivu bit_najmanje_tezine. Kako je x
62     oznacen ceo broj, tada se prilikom pomeranja udesno vrši
63     aritmeticko pomeranje i cuva se znak broja. Dakle, razlikuju
64     se dva slucaja u zavisnosti od znaka broja x. Nije dovoljno da
65     se ova provera izvrši pre petlje, s obzirom da rotiranjem
66     udesno na poziciju najveće težine može doći i 0 i 1, nezavisno
67     od početnog znaka broja smestenog u promenljivu x. */
68     for (i = 0; i < n; i++) {
69         bit_najmanje_tezine = x & 1;

71         if (x < 0)
72             /******
73             Siftovanjem udesno broja koji je negativan dobija se 1 kao bit
74             na poziciji najveće težine. Na primer ako je x
75             1010 1011 1100 1101 1110 0001 0010 001b
76             (sa b je oznacen ili 1 ili 0 na poziciji najmanje težine)
77             Onda je sadržaj promenljive bit_najmanje_tezine:
78             0000 0000 0000 0000 0000 0000 0000 000b
79             Nakon siftovanja sadržaja promenljive x za 1 udesno
80             1101 0101 1110 0110 1111 0000 1001 0001
81             Kako bi umesto 1 na poziciji najveće težine u trenutnoj binarnoj
82             reprezentaciji x bilo postavljeno b nije dovoljno da se pomeri na
83             poziciju najveće težine jer bi se time dobile 0, a u ovom slučaju
84             su potrebne jedinice zbog bitovskog & zato se prvo vrši
85             komplementiranje, a zatim pomeranje
86             ~bit_najmanje_tezine << (sizeof(int)*8 -1)
87             B000 0000 0000 0000 0000 0000 0000 0000
88             gde B oznacava ~b.
89             Potom se ponovo vrši komplementiranje kako bi se b nalazilo na
90             poziciji najveće težine i sve jedinice na ostalim pozicijama
91             ~(~bit_najmanje_tezine << (sizeof(int)*8 -1))
92             b111 1111 1111 1111 1111 1111 1111 1111
93             *****/
94             x = (x >> 1) & ~(~bit_najmanje_tezine <<
```

```

96         (sizeof(int) * 8 - 1));
97     else
98         x = (x >> 1) | bit_najmanje_tezine << (sizeof(int) * 8 - 1);
99 }
100
101 /* Vraca se izracunata vrednost */
102 return x;
103 }
104
105 int main()
106 {
107     unsigned x, n;
108
109     /* Ucitavaju se vrednosti sa standardnog ulaza */
110     printf("Unesite neoznaceni broj x:");
111     scanf("%x", &x);
112     printf("Unesite neoznaceni broj n:");
113     scanf("%x", &n);
114
115     /* Ispisuje se binarna reprezentacija broja x */
116     printf("x\t\t\t\t\t= ");
117     stampaj_bitove(x);
118
119     /* Testiraju se napisane funkcije */
120     printf("rotiraj_ulevo(%x,%u)\t\t= ", x, n);
121     stampaj_bitove(rotiraj_ulevo(x, n));
122
123     printf("rotiraj_udesno(%x,%u)\t\t= ", x, n);
124     stampaj_bitove(rotiraj_udesno(x, n));
125
126     printf("rotiraj_udesno_oznaceni(%x,%u)\t\t= ", x, n);
127     stampaj_bitove(rotiraj_udesno_oznaceni(x, n));
128
129     printf("rotiraj_udesno_oznaceni(-%x,%u)\t\t= ", x, n);
130     stampaj_bitove(rotiraj_udesno_oznaceni(-x, n));
131
132     return 0;
133 }

```

### Rešenje 1.10

NAPOMENA: Rešenje koristi biblioteku za štampanje bitova iz zadatka 1.5.

```

1  #include <stdio.h>
2  #include "stampanje_bitova.h"
3
4  /* Funkcija vraca vrednost cija je binarna reprezentacija slika u
5   ogledalu binarne reprezentacije broja x. */
6  unsigned ogledalo(unsigned x)
7  {
8      unsigned najnizi_bit;

```

## 1 Uvodni zadaci

---

```
9   unsigned rezultat = 0;

11  int i;
   /* U svakoj iteraciji najnizi bit u binarnoj reprezentaciji
13     tekuće vrednosti broja x se određuje i pamti u promenljivoj
       najnizi_bit, nakon čega se na promenljivu x primeni pomeranje
15     u desno */
   for (i = 0; i < sizeof(x) * 8; i++) {
17       najnizi_bit = x & 1;
       x >>= 1;
19       /* Potiskivanjem trenutnog rezultata ka levom kraju svi
           prethodno postavljeni bitovi dobijaju veću poziciju. Novi
21       bit se postavlja na najnižu poziciju */
       rezultat <<= 1;
23       rezultat |= najnizi_bit;
   }

25
   /* Vraca se izracunata vrednost */
27   return rezultat;
}

29
int main()
31 {
   int broj;

33
   /* Ucitava se broj sa ulaza */
35   scanf("%x", &broj);

37   /* Ispisuje se njegova binarna reprezentacija */
   stampaj_bitove(broj);

39
   /* Ispisuje se i binarna reprezentacija broja izracunatog pozivom
41     funkcije ogledalo */
   stampaj_bitove(ogledalo(broj));

43
   return 0;
45 }
```

### Rešenje 1.11

```
#include <stdio.h>

2
   /* Funkcija vraća 1 ukoliko je u binarnoj reprezentaciji broja n
4     broj jedinica veći od broja nula. U suprotnom funkcija vraća 0 */
int broj_01(unsigned int n)
6 {
   int broj_nula, broj_jedinica;
8   unsigned int maska;

10
   broj_nula = 0;
   broj_jedinica = 0;
```

```

12  /* Maska je inicijalizovana tako da moze da analizira bit najvece
14  tezine */
16  maska = 1 << (sizeof(unsigned int) * 8 - 1);

18  /* Cilj je proci kroz sve bitove broja x, zato se maska u svakoj
   iteraciji pomera u desno pa ce jedini bit koji je postavljen
   na 1 biti na svim pozicijama u binarnoj reprezentaciji maske */
20  while (maska != 0) {
22      /* Proverava se da li se na poziciji koju odredjuje maska
        nalazi 0 ili 1 i uvecava se odgovarajuci brojac. */
24      if (n & maska) {
26          broj_jedinica++;
28      } else {
30          broj_nula++;
32      }

34      /* Pomera se maska u desnu stranu */
36      maska = maska >> 1;
38  }

40  /* Ako je broj jedinica veci od broja nula funkcija vraca 1, u
   suprotnom vraca 0 */
42  return (broj_jedinica > broj_nula) ? 1 : 0;
44 }

46 int main()
48 {
49     unsigned int n;

51     /* Ucitava se neoznaceni broj */
53     scanf("%u", &n);

55     /* Ispisuje se rezultat */
57     printf("%d\n", broj_01(n));

60     return 0;
62 }

```

### Rešenje 1.12

```

1  #include <stdio.h>

2  /* Funkcija broji koliko se puta dve uzastopne jedinice pojavljuju
   u binarnom zapisu celog čneoznaenog broja x */
4  int broj_parova(unsigned int x)
6  {
8      int broj_parova;
9      unsigned int maska;

10     /* Vrednost promenljive koja predstavlja broj parova se

```

## 1 Uvodni zadaci

```
12     inicijalizuje na 0 */
    broj_parova = 0;

14     /* Postavlja se maska tako da moze da procita da li su dva
        najmanja bita u zapisu broja x 11 */
16     /* Binarna reprezentacija broja 3 je 000....00011 */
    maska = 3;

18
20     while (x != 0) {
        /* Proverava se da li se na najmanjim pozicijama broja x nalazi
           par bitova 11 */
22         if ((x & maska) == maska) {
            broj_parova++;
24         }

26         /* Pomera se broj u desnu stranu da bi se u narednoj iteraciji
           proveravao sledeci par bitova. Pomeranjem u desno bit
28             najvece tezine se popunjava nulom jer je x neoznaceni broj */
        x = x >> 1;
30     }

32     /* Vraca se izracunata vrednost */
    return broj_parova;
34 }

36 int main()
{
38     unsigned int x;

40     /* Ucitava se neoznaceni broj */
    scanf("%u", &x);

42
44     /* Ispisuje se rezultat */
    printf("%d\n", broj_parova(x));

46     return 0;
}
```

### Rešenje 1.14

```
#include <stdio.h>

2
/* Niska koja se formira je duzine (sizeof(unsigned int)*8)/4 +1
4   jer su za svaku heksadekadnu cifru potrebne 4 binarne cifre i
   jedna dodatna pozicija za terminirajucu nulu. Prethodni izraz se
6   moze zapisati kao sizeof(unsigned int)*2+1. */
#define MAKS_DUZINA sizeof(unsigned int)*2 +1

8
/* Funkcija za neoznaceni broj x formira nisku s koja sadrzi njegov
10   heksadekadni zapis */
void prevod(unsigned int x, char s[])
```

```

12 {
13     int i;
14     unsigned int maska;
15     int vrednost;
16
17     /* Heksadekadni zapis broja 15 je 000...0001111 - odgovarajuca
18        maska za citanje 4 uzastopne cifre */
19     maska = 15;
20
21     /******
22        Broj se posmatra od pozicije najmanje tezine ka poziciji najvece
23        tezine. Na primer za broj cija je binarna reprezentacija
24        00000000001101000100001111010101
25        u prvom koraku se citaju bitovi izdvojeni sa <...>:
26        0000000000110100010000111101<0101>
27        u drugom koraku:
28        000000000011010001000011<1101>0101
29        u trecem koraku:
30        00000000001101000100<0011>11010101 i tako redom...
31
32        Indeks i oznacava poziciju na koju se smesta vrednost.
33        *****/
34     for (i = MAKS_DUZINA - 2; i >= 0; i--) {
35         /* Vrednost izdvojene cifre */
36         vrednost = x & maska;
37
38         /* Ako je vrednost iz opsega od 0 do 9 odgovarajuci karakter se
39            dobija dodavanjem ASCII koda '0'. Ako je vrednost iz opsega
40            od 10 do 15 odgovarajuci karakter se dobija tako sto se prvo
41            oduzme 10 (time se dobiju vrednosti od 0 do 5) pa se na tako
42            izracunatu vrednost dodaje ASCII kod 'A' (time se dobija
43            odgovarajuce slovo 'A', 'B', ... 'F') */
44         if (vrednost < 10) {
45             s[i] = vrednost + '0';
46         } else {
47             s[i] = vrednost - 10 + 'A';
48         }
49
50         /* Promenljiva x se pomera za 4 bita u desnu stranu i time se u
51            narednoj iteraciji posmatraju sledeca 4 bita */
52         x = x >> 4;
53     }
54
55     /* Upisuje se terminirajuca nula */
56     s[MAKS_DUZINA - 1] = '\0';
57 }
58
59 int main()
60 {
61     unsigned int x;
62     char s[MAKS_DUZINA];

```

## 1 Uvodni zadaci

---

```
64  /* Ucitava se broj sa ulaza */
    scanf("%u", &x);
66
    /* Poziva se funkcija za prevodjenje */
68  prevod(x, s);
70
    /* I stampa se formirana niska */
    printf("%s\n", s);
72
    return 0;
74 }
```

### Rešenje 1.17

```
#include <stdio.h>
2  #include <stdlib.h>

4  /*****
    Resenje linearne slozenosti:
6   x^0 = 1
    x^k = x * x^(k-1)
8   *****/
int stepen(int x, int k)
10 {
    if (k == 0)
12     return 1;

14     return x * stepen(x, k - 1);
    /* kraci zapis: return k == 0 ? 1 : x * stepen(x,k-1); */
16 }

18 /*****
    Resenje logaritamske slozenosti:
20     x^0 =1;
    x^k = x * (x^2)^(k/2) - za neparno k
22     x^k = (x^2)^(k/2) - za parno k
    Ovom resenju ce biti potrebno manje rekurzivnih poziva da bi
24     se doslo do rezultata, i stoga je efikasnije.
    *****/
26 int stepen_2(int x, int k)
    {
28         if (k == 0)
            return 1;
30
        /* Ukoliko je stepen k paran */
32         if ((k % 2) == 0)
            return stepen_2(x * x, k / 2);
34
        /* Ukoliko je stepen k neparan */
36         return x * stepen_2(x * x, k / 2);
    }
```



```

38 int main()
40 {
    int x, k, ind;

    /* Ucitavanje rednog broja funkcije koja ce se primeniti */
44 printf("Unesite redni broj funkcije (1/2):\n");
    scanf("%d", &ind);

    /* Ucitavanje vrednosti za x i k */
48 printf("Unesite broj x:\n");
    scanf("%d", &x);
50 printf("Unesite broj k:\n");
    scanf("%d", &k);

    /* Ispisuje se vrednost koju vraca odgovarajuca funkcija */
54 if (x == 1)
    printf("%d\n", stepen(x, k));
56 else if (x == 2)
    printf("%d\n", stepen_2(x, k));
58 else {
    fprintf(stderr,
60         "Greska: Neodgovarajuci redni broj funkcije.\n");
    exit(EXIT_FAILURE);
62 }

64 exit(EXIT_SUCCESS);
}

```

### Rešenje 1.18

```

#include <stdio.h>
2
/* NAPOMENA: Ovaj problem je iskoriscen da ilustruje uzajamnu
4 (posrednu) rekurziju */

6 /* Deklaracija funkcije neparan mora da bude navedena jer se ta
   funkcija koristi u telu funkcije paran, tj. koristi se pre svoje
8 definicije. Funkcija je mogla biti deklarirana i u telu funkcije
   paran. */
10 unsigned neparan(unsigned n);

12 /* Funkcija paran vraca 1 ako broj n ima paran broj cifara, inace
   vraca 0 */
14 unsigned paran(unsigned n)
{
16     if (n <= 9)
        return 0;
18     else
        return neparan(n / 10);
20 }

```

## 1 Uvodni zadaci

---

```
22 /* Funkcija neparan vraća 1 ako broj n ima neparan broj cifara,
    inace vraća 0 */
24 unsigned neparan(unsigned n)
25 {
26     if (n <= 9)
27         return 1;
28     else
29         return paran(n / 10);
30 }
31
32 int main()
33 {
34     int n;
35
36     /* Ucitava se ceo broj */
37     scanf("%d", &n);
38
39     /* Ispisuje se rezultat */
40     printf("Uneti broj ima %sparan broj cifara.\n",
41           (paran(n) == 1 ? "p" : "ne"));
42
43     return 0;
44 }
```

### Rešenje 1.19

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 /* Pomocna funkcija koja izracunava n! * rezultat. Koristi repnu
5   rekurziju. rezultat je argument u kome se akumulira do tada
6   izracunatu vrednost faktoriijela. Kada dodje do izlaza iz
7   rekurzije iz rekurzije potrebno je da vratimo rezultat. */
8 int faktoriijel_repna(int n, int rezultat)
9 {
10     if (n == 0)
11         return rezultat;
12
13     return faktoriijel_repna(n - 1, n * rezultat);
14 }
15
16 /* U sledecim funkcijama je prikazan postupak oslobadjanja od repne
17   rekurzije koja postoji u funkciji faktoriijel_repna. */
18
19 /* Funkcija se transformise tako sto se rekurzivni poziv zemeni sa
20   naredbama kojima se vrednost argumenta funkcije postavlja na
21   vrednost koja bi se prosledjivala rekurzivnom pozivu i
22   navodjenjem goto naredbe za vracanje na pocetak tela funkcije. */
23 int faktoriijel_repna_v1(int n, int rezultat)
24 {
```

```

25 pocetak:
    if (n == 0)
27         return rezultat;

29     rezultat = n * rezultat;
    n = n - 1;
31     goto pocetak;
}

33
/* Pisanje bezuslovnih skokova (goto naredbi) nije dobra
35     programerska praksa i prethodna funkcija se koristi samo kao
    medjukorak. Sledi iterativno resenje bez bezuslovnih skokova */
37 int faktorijel_repna_v2(int n, int rezultat)
{
39     while (n != 0) {
        rezultat = n * rezultat;
41         n = n - 1;
    }

43     return rezultat;
45 }

47 /* Prilikom poziva prethodnih funkcija pored prvog argumenta celog
    broja n, mora da se salje i 1 za vrednost drugog argumenta u
49     kome ce se akumulirati rezultat. Funkcija faktorijel(n) je ovde
    radi udobnosti korisnika, jer je sasvim prirodno da za
51     faktorijel zahteva samo 1 parametar. Funkcija faktorijel
    izracunava n!, tako sto odgovarajucoj gore navedenoj funkciji
53     koja zaista racuna faktorijel, salje ispravne argumente i vraca
    rezultat koju joj ta funkcija vrati. Za testiranje, zameniti u
55     telu funkcije faktorijel poziv faktorijel_repna sa pozivom
    faktorijel_repna_v1, a zatim sa pozivom funkcije
57     faktorijel_repna_v2. */
int faktorijel(int n)
59 {
    return faktorijel_repna(n, 1);
61 }

63 int main()
{
65     int n;

67     /* Ucitava se ceo broj */
    printf("Unesite n (<= 12): ");
69     scanf("%d", &n);
    if (n > 12) {
71         fprintf(stderr, "Greska: Nedozvoljena vrednost za n.\n");
        exit(EXIT_FAILURE);
73     }

75     /* Ispisuje se rezultat poziva funkcije faktorijel */
    printf("%d! = %d\n", n, faktorijel(n));

```

## 1 Uvodni zadaci

---

```
77     exit(EXIT_SUCCESS);
79 }
```

### Rešenje 1.21

```
1  #include <stdio.h>

3  /* a) Funkcija racuna n-ti element u nizu F - iterativna verzija */
4  int f_iterativna(int n, int a, int b)
5  {
6      int i;
7      int f_0 = 0;
8      int f_1 = 1;
9      int tmp;

11     if (n == 0)
12         return 0;

13     for (i = 2; i <= n; i++) {
14         tmp = a * f_1 + b * f_0;
15         f_0 = f_1;
16         f_1 = tmp;
17     }

18     return f_1;
19 }

21 /* b) Funkcija racuna n-ti element u nizu F - rekurzivna verzija */
22 int f_rekurzivna(int n, int a, int b)
23 {
24     /* Izlaz iz rekurzije */
25     if (n == 0 || n == 1)
26         return n;

27     /* Rekurzivni pozivi */
28     return a * f_rekurzivna(n - 1, a, b) +
29           b * f_rekurzivna(n - 2, a, b);
30 }

32 /* NAPOMENA: U slucaju da se rekurzijom problem svodi na vise
33    manjih potproblema koji se mogu preklapati, postoji opasnost da
34    se pojedini potproblemi manjih dimenzija resavaju veci broj
35    puta. Npr.  $F(20) = a \cdot F(19) + b \cdot F(18)$ , a  $F(19) = a \cdot F(18) +$ 
36     $b \cdot F(17)$ , tj. problem  $F(18)$  se resava dva puta! Problemi manjih
37    dimenzija ce se resavati jos veci broj puta. Resenje za ovaj
38    problem je kombinacija rekurzije sa dinamicnim programiranjem.
39    Potproblemi se resavaju samo jednom, a njihova resenja se pamte
40    u memoriji (obicno u nizovima ili matricama), odakle se koriste
41    ako tokom resavanja ponovo budu potrebni.
42
43
44
45
```

```
47     U narednoj funkciji vec izracunati clanovi niza se cuvaju u
49     statickom nizu celih brojeva, jer se staticki niz ne smesta na
    stek, kao sto je to slucaj sa lokalnim promenljivama, vec na
    segment podataka, odakle je dostupan svim pozivima rekurzivne
    funkcije. */
51
52     /* c) Funkcija racuna n-ti broj niza F - efikasnija rekurzivna
53     verzija */
    int f_napredna(int n, int a, int b)
54     {
55         /* Niz koji cuva resenja potproblema. Kompajler inicijalizuje
57         staticke promenljive na potrazumevane vrednosti. Stoga,
58         elemente celobrojnog niza inicijalizuje na 0 */
59         static int f[20];
60
61         /* Ako je potproblem vec ranije resen, koristi se resenje koje je
62         vec izracunato */
63         if (f[n] != 0)
64             return f[n];
65
66         /* Izlaz iz rekurzije */
67         if (n == 0 || n == 1)
68             return f[n] = n;
69
70         /* Rekurzivni pozivi */
71         return f[n] =
            a * f_napredna(n - 1, a, b) + b * f_napredna(n - 2, a, b);
72     }
73
74     int main()
75     {
76         int n, a, b, ind;
77
78         /* Unosi se redni broj funkcije koja ce se primeniti */
79         printf("Unesite redni broj funkcije:\n");
80         printf("1 - iterativna\n");
81         printf("2 - rekurzivna\n");
82         printf("3 - rekurzivna napredna\n");
83         scanf("%d", &ind);
84
85         /* Ucitaju se koeficijenti a i b */
86         printf("Unesite koeficijente:\n");
87         scanf("%d%d", &a, &b);
88
89         /* Ucitava se broj n */
90         printf("Unesite koji clan niza se racuna:\n");
91         scanf("%d", &n);
92
93         /* Na osnovu vrednosti promenljive ind ispisuje se rezultat
94         poziva funkcije f_iterativna, f_rekurzivna ili f_napredna */
95         if (ind == 0)
96             printf("F(%d) = %d\n", n, f_iterativna(n, a, b));
97     }
```

## 1 Uvodni zadaci

---

```

    else if (ind == 1)
99     printf("F(%d) = %d\n", n, f_rekurzivna(n, a, b));
    else
101     printf("F(%d) = %d\n", n, f_napredna(n, a, b));
103     return 0;
}
```

### Rešenje 1.22

```

#include <stdio.h>
2
/* Funkcija odredjuje zbir cifara zadatog broja x */
4 int zbir_cifara(unsigned int x)
{
6     /* Izlazak iz rekurzije: ako je broj jednocifren */
    if (x < 10)
8         return x;

10     /* Zbir cifara broja jednak je zbiru svih njegovih cifara osim
        poslednje cifre + poslednja cifra tog broja */
12     return zbir_cifara(x / 10) + x % 10;
}

14
16 int main()
{
    unsigned int x;

18     /* Ucitava se ceo broj */
20     scanf("%u", &x);

22     /* Ispisuje se zbir cifara ucitanog broja */
    printf("%d\n", zbir_cifara(x));

24     return 0;
26 }
```

### Rešenje 1.23

```

#include <stdio.h>
2 #include <stdlib.h>
#define MAKS_DIM 100
4
/* Ako je n<=0, onda je suma niza jednaka nuli. Ako je n>0, onda je
6 suma niza jednaka sumi prvih n-1 elementa uvecenoj za poslednji
   element niza. */
8 int suma_niza_1(int *a, int n)
{
10     if (n <= 0)
```

```
12     return 0;
13
14     return suma_niza_1(a, n - 1) + a[n - 1];
15 }
16
17 /* Funkcija napisana na drugi nacin: Ako je n<=0, onda je suma niza
18    jednaka nuli. Ako je n>0, suma niza je jednaka zbiru prvog
19    elementa niza i sume preostalih n-1 elementa. */
20 int suma_niza_2(int *a, int n)
21 {
22     if (n <= 0)
23         return 0;
24
25     return a[0] + suma_niza_2(a + 1, n - 1);
26 }
27
28 int main()
29 {
30     int a[MAKS_DIM];
31     int n, i = 0, ind;
32
33     /* Ucitava se redni broj funkcije */
34     printf("Unesite redni broj funkcije (1 ili 2):\n");
35     scanf("%d", &ind);
36
37     /* Ucitava se broj elemenata niza */
38     printf("Unesite dimenziju niza:\n");
39     scanf("%d", &n);
40
41     /* Ucitava se n elemenata niza. */
42     printf("Unesite elemente niza:\n");
43     for (i = 0; i < n; i++)
44         scanf("%d", &a[i]);
45
46     /* Na osnovu vrednosti promenljive ind ispisuje se rezultat
47        poziva funkcije suma_niza_1, ondosno suma_niza_2 */
48     if (ind == 1)
49         printf("Suma elemenata je %d\n", suma_niza_1(a, n));
50     else if (ind == 2)
51         printf("Suma elemenata je %d\n", suma_niza_2(a, n));
52     else {
53         fprintf(stderr,
54             "Greska: Neodgovarajuci redni broj funkcije.\n");
55         exit(EXIT_FAILURE);
56     }
57
58     exit(EXIT_SUCCESS);
59 }
```

### Rešenje 1.24

```
1 #include <stdio.h>
2 #define MAKS_DIM 256
3
4 /* Rekurzivna funkcija koja odredjuje maksimum celobrojnog niza niz
   dimenzije n */
5
6 int maksimum_niza(int niz[], int n)
7 {
8     /* Izlazak iz rekurzije: ako je niz dimenzije jedan, najveći je
       ujedno i jedini element niza */
9     if (n == 1)
10         return niz[0];
11
12     /* Resava se problem manje dimenzije */
13     int max = maksimum_niza(niz, n - 1);
14
15     /* Na osnovu poznatog resenja problema dimenzije n-1, resava se
       problem dimenzije n */
16     return niz[n - 1] > max ? niz[n - 1] : max;
17 }
18
19
20 int main()
21 {
22     int brojevi[MAKS_DIM];
23     int n;
24
25     /* Sve dok se ne stigne do kraja ulaza, brojevi se ucitavaju u
       niz. Promenljiva i predstavlja indeks tekuceg broja. U niz se
       ne moze ucitati vise od MAKS_DIM brojeva, pa se u slucaju da
       promenljiva i dostigne vrednost MAKS_DIM prekida unos novih
       brojeva. */
26     int i = 0;
27     while (scanf("%d", &brojevi[i]) != EOF) {
28         i++;
29         if (i == MAKS_DIM)
30             break;
31     }
32     n = i;
33
34     /* Stampa se maksimum unetog niza brojeva */
35     printf("%d\n", maksimum_niza(brojevi, n));
36
37     return 0;
38 }
```

### Rešenje 1.25

```
1 #include <stdio.h>
2 #include <stdlib.h>
```



```

3  #define MAKS_DIM 256

5  /* Funkcija koja izracunava skalarni proizvod dva data vektora */
int skalarno(int a[], int b[], int n)
7  {
    /* Izlazak iz rekurzije: vektori su duzine 0 */
9    if (n == 0)
        return 0;

11   /* Na osnovu resenja problema dimenzije n-1, resava se problem
13   dimenzije n primenom definicije skalarnog proizvoda a*b =
        a[0]*b[0] + a[1]*b[1] + ... + a[n-2]*a[n-2] + a[n-1]*a[n-1]
15   Dakle, skalarni proizvod dva vektora duzine n se dobija kada
        se na skalarni proizvod dva vektora duzine n-1 koji se dobiju
17   od polazna dva vektora otklanjanjem poslednjih elemenata, doda
        proizvod poslednja dva elementa polaznih vektora. */
19   else
        return skalarno(a, b, n - 1) + a[n - 1] * b[n - 1];
21 }

23 int main()
{
25     int i, a[MAKS_DIM], b[MAKS_DIM], n;

27     /* Unosi se dimenzija nizova */
    printf("Unesite dimenziju nizova:");
29     scanf("%d", &n);

31     /* Provera da li je dimenzija niza odgovarajuca */
    if (n < 0 || n > MAKS_DIM) {
33         fprintf(stderr,
35             "Greska: Dimenzija mora biti prirodan broj <= %d!\n",
                MAKS_DIM);
        exit(EXIT_FAILURE);
37     }

39     /* Unose se elementi nizova */
    printf("Unesite elemente prvog niza:");
41     for (i = 0; i < n; i++)
        scanf("%d", &a[i]);

43     printf("Unesite elemente drugog niza:");
45     for (i = 0; i < n; i++)
        scanf("%d", &b[i]);

47     /* Ispisuje se rezultat skalarnog proizvoda dva ucitana niza */
49     printf("Skalarni proizvod je %d\n", skalarno(a, b, n));

51     exit(EXIT_SUCCESS);
}

```

### Rešenje 1.26

```
#include <stdio.h>
2 #define MAKS_DIM 256

4 /* Funkcija koja racuna broj pojavljivanja elementa x u nizu a
   duzine n */
6 int br_pojave(int x, int a[], int n)
{
8     /* Izlazak iz rekurzije: za niz duzine jedan broj pojava broja x
       u nizu je 1 ukoliko je jedini element a[0] bas x ili 0 inace */
10     if (n == 1)
        return a[0] == x ? 1 : 0;

12     /* U promenljivu bp se smesta broj pojava broja x u prvih n-1
       elemenata niza a. Ukupan broj pojavljivanja broja x u celom
       nizu a je jednak bp uvecanom za jedan ukoliko je se na
       poziciji n-1 u nizu a nalazi broj x */
14     int bp = br_pojave(x, a, n - 1);
16     return a[n - 1] == x ? 1 + bp : bp;
18 }

20
22 int main()
{
24     int x, a[MAKS_DIM];
26     int n, i = 0;

28     /* Ucitava se ceo broj */
    printf("Unesite ceo broj:");
    scanf("%d", &x);

30     /* Sve dok se ne stigne do kraja ulaza, ucitavaju se brojevi u
       niz. Promenljiva i predstavlja indeks tekućeg broja. U niz se
       ne moze ucitati vise od MAKS_DIM brojeva, pa se u slucaju da
       promenljiva i dostigne vrednost MAKS_DIM prekida unos novih
       brojeva. */
32     printf("Unesite elemente niza:");
34     i = 0;
    while (scanf("%d", &a[i]) != EOF) {
36         i++;
38         if (i == MAKS_DIM)
40             break;
    }
42     n = i;

44     /* Ispisuje se broj pojavljivanja */
    printf("Broj pojavljivanja je %d\n", br_pojave(x, a, n));
46
48     return 0;
}
```

## Rešenje 1.27

```
1  #include <stdio.h>
2  #define MAKS_DIM 256
3
4  /* Funkcija koja proverava da li su tri zadata broja uzastopni
5     clanovi niza */
6  int tri_uzastopna_clana(int x, int y, int z, int a[], int n)
7  {
8      /* Ako niz ima manje od tri elementa izlazi se iz rekurzije i
9         vraca se 0 jer nije ispunjeno da su x, y i z uzastopni clanovi
10        niza */
11     if (n < 3)
12         return 0;
13
14     /* Da bi bilo ispunjeno da su x, y i z uzastopni clanovi niza a
15        dovoljno je da su oni poslednja tri clana niza ili da se oni
16        rekuzivno tri uzastopna clana niza a bez poslednjeg elementa */
17     return ((a[n - 3] == x) && (a[n - 2] == y) && (a[n - 1] == z))
18         || tri_uzastopna_clana(x, y, z, a, n - 1);
19 }
20
21 int main()
22 {
23     int x, y, z, a[MAKS_DIM], n;
24
25     /* Ucitavaju se tri cela broja za koje se ispituje da li su
26        uzastopni clanovi niza */
27     printf("Unesite tri cela broja:");
28     scanf("%d%d%d", &x, &y, &z);
29
30     /* Sve dok se ne stigne do kraja ulaza, brojeve se ucitavaju u
31        niz. Promenljiva i predstavlja indeks tekuceg broja. U niz se
32        ne moze ucitati vise od MAKS_DIM brojeva, pa se u slucaju da
33        promenljiva i dostigne vrednost MAKS_DIM prekida unos novih
34        brojeva. */
35     printf("Unesite elemente niza:");
36     int i = 0;
37     while (scanf("%d", &a[i]) != EOF) {
38         i++;
39         if (i == MAKS_DIM)
40             break;
41     }
42     n = i;
43
44     /* Na osnovu rezultata poziva funkcije tri_uzastopna_clana
45        ispisuje se odgovarajuca poruka */
46     if (tri_uzastopna_clana(x, y, z, a, n))
47         printf("Uneti brojevi jesu uzastopni clanovi niza.\n");
48     else
49         printf("Uneti brojevi nisu uzastopni clanovi niza.\n");
```

## 1 Uvodni zadaci

---

```
51     return 0;
    }
```

### Rešenje 1.28

```
#include <stdio.h>

2
/* Funkcija koja broji bitove postavljene na 1. */
4 int prebroj(int x)
{
6     /* Izlaz iz rekurzije */
    if (x == 0)
8         return 0;

10     /* Ukoliko vrednost promenljive x nije 0, neki od bitova broja x
        je postavljen na 1. Koriscenjem odgovarajuce maske proverava
12     se vrednost bita na poziciji najvece tezine i na osnovu toga
        se razlikuju dva slucaja. Ukoliko je na toj poziciji nula,
14     onda je broj jedinica u zapisu x isti kao broj jedinica u
        zapisu broja x<<1, jer se pomeranjem u levo sa desne stane
16     dopisuju 0. Ako je na poziciji najvece tezine jedinica,
        rezultat dobijen rekurzivnim pozivom funkcije za x<<1 treba
18     uvecati za jedan. Za rekurzivni poziv se salje vrednost koja
        se dobija kada se x pomeri u levo. Napomena: argument funkcije
20     x je oznacen ceo broj, usled cega se ne koristi pomeranje
        udesno, jer funkciji moze biti prosledjen i negativan broj. Iz
22     tog razloga, odlucujemo se da proveramo najvisi, umesto
        najnizeg bita */
24     if (x & (1 << (sizeof(x) * 8 - 1)))
        return 1 + prebroj(x << 1);
26     else
        return prebroj(x << 1);
28     /******
        Krace zapisano
30     return ((x& (1<<(sizeof(x)*8-1))) ? 1 : 0) + prebroj(x<<1);
        *****/
32 }

34 int main()
{
36     int x;

38     /* Ucitava se ceo broj */
    scanf("%x", &x);

40
42     /* Ispisuje se rezultat */
    printf("%d\n", prebroj(x));

44     return 0;
}
```

## Rešenje 1.30

```
2  #include <stdio.h>
3
4  /* Rekurzivna funkcija za odredjivanje najveće oktalne cifre */
5  int maks_oktalna_cifra(unsigned x)
6  {
7      /* Izlazak iz rekurzije: ako je vrednost broja 0, onda je i
8         vrednost najveće oktalne cifre u broju 0 */
9      if (x == 0)
10         return 0;
11
12     /* Odredjuje se poslednja oktalna cifra u broju */
13     int poslednja_cifra = x & 7;
14
15     /* Odredjuje se maksimalna oktalna cifra u broju kada se iz njega
16        izbrise poslednja oktalna cifra */
17     int maks_bez_poslednje_cifre = maks_oktalna_cifra(x >> 3);
18
19     return poslednja_cifra >
20         maks_bez_poslednje_cifre ? poslednja_cifra :
21         maks_bez_poslednje_cifre;
22 }
23
24 int main()
25 {
26     unsigned x;
27
28     /* Ucitava se neoznaceni ceo broj */
29     scanf("%u", &x);
30
31     /* Ispisuje se vrednost najveće oktalne cifre unetog broja */
32     printf("%d\n", maks_oktalna_cifra(x));
33
34     return 0;
35 }
```

## Rešenje 1.31

```
2  #include <stdio.h>
3
4  /* Rekurzivna funkcija za odredjivanje najveće heksadekadne cifre */
5  int maks_heksadekadna_cifra(unsigned x)
6  {
7      /* Izlazak iz rekurzije: ako je vrednost broja 0, onda je i
8         vrednost najveće heksadekadne cifre u broju 0 */
9      if (x == 0)
10         return 0;
11
12     /* Odredjuje se poslednja heksadekadna cifra u broju */
13 }
```

## 1 Uvodni zadaci

---

```
12  int poslednja_cifra = x & 15;

14  /* Odredjuje se maksimalna heksadekadna cifra broja kada se iz
    njega izbrise poslednja heksadekadna cifra */
16  int maks_bez_poslednje_cifre = maks_heksadekadna_cifra(x >> 4);

18  return poslednja_cifra >
    maks_bez_poslednje_cifre ? poslednja_cifra :
20  maks_bez_poslednje_cifre;
}

22
24  int main()
25  {
26      unsigned x;

28      /* Ucitava se neoznaceni ceo broj */
29      scanf("%u", &x);

30      /* Ispisivanje vrednosti najveće heksadekadne cifre unetog broja */
31      printf("%d\n", maks_heksadekadna_cifra(x));
32
33      return 0;
34  }
```

### Rešenje 1.32

```
#include <stdio.h>
2 #include <string.h>

4 /* Niska može imati najviše 31 karaktera + 1 za terminalnu nulu */
#define MAKS_DIM 32

6 /* Funkcija ispituje da li je zadana niska dužine n palindrom */
8 int palindrom(char s[], int n)
9 {
10     /* Izlaz iz rekurzije - trivijalno, niska dužine 0 ili 1 je
        palindrom */
12     if ((n == 1) || (n == 0))
13         return 1;

14     /* Da bi niska bila palindrom potrebno je da se poklapaju prvi i
        poslednji karakter i da je palindrom niska koja nastaje kada
        se polaznoj nisci otklone prvi i poslednji karakter */
16     return (s[n - 1] == s[0]) && palindrom(s + 1, n - 2);
18 }

20
22 int main()
23 {
24     char s[MAKS_DIM];
25     int n;
```

```

26  /* Ucitava se niska sa standardnog ulaza */
    scanf("%s", s);
28
    /* Odredjuje se duzina niske */
30  n = strlen(s);

    /* Ispisuje se da li je niska palindrom ili nije */
32  if (palindrom(s, n))
34      printf("da\n");
    else
36      printf("ne\n");

38  return 0;
}

```

### Rešenje 1.33

```

1  #include <stdio.h>
    #include <stdlib.h>
3  #define  MAKS_DUZINA_PERMUTACIJE 15

5  /* Funkcija koja ispisuje elemente niza a duzine n */
    void ispisi_niz(int a[], int n)
7  {
    int i;

9      for (i = 1; i <= n; i++)
11         printf("%d ", a[i]);
        printf("\n");
13 }

15 /* Funkcija koja vraca 1, ako se broj x nalazi u nizu a duzine n,
    inace vraca 0 */
17 int koriscen(int a[], int n, int x)
    {
19     int i;

21     /* Obilaze se svi elementi niza */
        for (i = 1; i <= n; i++)
23         /* Ukoliko se naidje na trazenu vrednost, pretraga se prekida */
            if (a[i] == x)
25                 return 1;

27     /* Zakljucuje se da broj nije pronadjen */
        return 0;
29 }

31 /* Funkcija koja ispisuje sve permutacije od skupa {1,2,...,n}
    dobija kao argument niz a[] u koji se smesta permutacija, broj m
33     oznacava da se u okviru tog poziva funkcije na m-tu poziciju u
    permutaciji smesta jedan od preostalih celih brojeva, n je

```

```
35     velicina skupa koji se permutuje. Funkciju se inicijalno poziva
36     sa argumentom m = 1, jer formiranje permutacije pocinje od
37     pozicije broj 1. Stoga, a[0] se ne koristi. */
void permutacija(int a[], int m, int n)
39 {
40     int i;
41
42     /* Izlaz iz rekurzije: Ako je pozicija na koju treba smestiti
43     broj premasila velicinu skupa, onda se svi brojevi vec nalaze
44     u permutaciji i ispisuje se permutacija. */
45     if (m > n) {
46         ispisi_niz(a, n);
47         return;
48     }
49
50     /* Ideja: pronalazi se prvi broj koji moze da se postavi na m-to
51     mesto u nizu (broj koji se do sada nije pojavio u
52     permutaciji). Zatim, rekurzivno se pronalaze one permutacije
53     koje odgovaraju ovako postavljenom pocetku permutacije. Kada
54     se to zavrshi, vrsi se provera da li postoji jos neki broj koji
55     moze da se stavi na m-to mesto u nizu (to se radi u petlji).
56     Ako ne postoji, funkcija zavrшава sa radom. Ukoliko takav broj
57     postoji, onda se ponovo poziva rekurzivno pronalazenje
58     odgovarajucih permutacija, ali sada sa drugacije postavljenim
59     prefiksom. */
60     for (i = 1; i <= n; i++) {
61         /* Ako se broj i nije do sada pojavio u permutaciji od 1 do m-1
62         pozicije, onda se on postavlja na poziciju m i poziva se
63         ponovo funkcija da dopuni ostatak permutacije posle
64         upisivanja i na poziciju m. Inace, nastavlja se dalje, trazi
65         se broj koji se nije pojavio do sada u permutaciji. */
66         if (!koriscen(a, m - 1, i)) {
67             a[m] = i;
68             permutacija(a, m + 1, n);
69         }
70     }
71 }
72
73 int main(void)
74 {
75     int n;
76     int a[MAKS_DUZINA_PERMUTACIJE + 1];
77
78     /* Ucitava se broj n iz odgovarajuceg opsega */
79     scanf("%d", &n);
80     if (n < 0 || n > MAKS_DUZINA_PERMUTACIJE) {
81         fprintf(stderr,
82             "Greska: Duzina permutacije mora biti broj iz intervala
83             [0, %d]!\n",
84             MAKS_DUZINA_PERMUTACIJE);
85         exit(EXIT_FAILURE);
86     }
87 }
```



```

87  /* Ispisuju se permutacije duzine n */
    permutacija(a, 1, n);
89
    exit(EXIT_SUCCESS);
91 }

```

### Rešenje 1.34

```

#include <stdio.h>
#include <stdlib.h>

/* Rekurzivna funkcija za racunanje binomnog koeficijenta */
int binomni_koeficijent(int n, int k)
{
    /* Ukoliko je k=0 ili k=n, onda je binomni koeficijent 0. Ukoliko
       je k strogo izmedju 0 i n, onda se koristi formula  $b_k(n,k) = b_{k-1}(n,k-1) + b_k(n-1,k)$  koja se moze izvesti iz definicije
       binomnog koeficijenata */
    return (0 < k && k < n) ?
        binomni_koeficijent(n - 1, k - 1) +
        binomni_koeficijent(n - 1, k) : 1;
}

/***** Iterativno izracunavanje binomnog koeficijenta *****/

int binomni_koeficijent (int n, int k) {
    int i, j, b;

    for (b=i=1, j=n; i<=k; b =b * j-- / i++);

    return b;
}

Iterativno resenje je efikasnije i preporucuje se. Rekurzivno
resenje je navedeno u cilju demonstracije rekurzivnih tehnika.
*****/

/* Svaki element n-te hipotenuze (osim ivicnih jedinica) dobija kao
   zbir 2 elementa iz n-1 hipotenuze. Ukljucujuci i pomenute dve
   ivicne jedinice suma elemenata n-te hipotenuze je tacno 2 puta
   veca od sume elemenata prethodne hipotenuze. */
int suma_elementata_hipotenuze(int n)
{
    return n > 0 ? 2 * suma_elementata_hipotenuze(n - 1) : 1;
}

int main()
{
    int n, k, i, d, r;

```

```
44  /* Ucitavaju se brojevi d i r */
    scanf("%d %d", &d, &r);
46
    /* Ispisuje se Paskalov trougao */
48  putchar('\n');
    for (n = 0; n <= d; n++) {
50      for (i = 0; i < d - n; i++)
          printf(" ");
52      for (k = 0; k <= n; k++)
          printf("%4d", binomni_koeficijent(n, k));
54      putchar('\n');
    }
56
    /* Proverava se da li je r nenegativan */
58  if (r < 0) {
        fprintf(stderr,
60             "Greska: Redni broj hipotenuze mora biti veci ili jednak
              od 0!\n");
        exit(EXIT_FAILURE);
62  }

64  /* Ispisuje se suma elemenata hipotenuze */
    printf("%d\n", suma_elementa_hipotenuze(r));
66
    exit(EXIT_SUCCESS);
68 }
```