

PROGRAMIRANJE 2

**Milena Vujošević Janićić, 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

Anđelka 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 i crteži: *autori*. Dizajn korica: *Anđelka Zečević*

Štampa: Copy Centar, Beograd

CIP Каталогизација у публикацији

Народна библиотека Србије, Београд

ISBN 978-86-7589-107-9

©2016. Milena Vujošević Janičić, Jelena Graovac, Nina Radojičić, Ana Spasić, Mirko Spasić, Anđelka 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

1	Uvodni zadaci	1
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 rešavanju 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, dostupni su besplatno u okviru strane kursa 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.

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 zadatke i rešenja zbirke su dobrodošli i osećajte se slobodno da ih pošaljete elektronskom poštom bilo kome od autora¹.

Autori

¹Adrese 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.

<p><i>Test 1</i></p> <pre> ULAZ: 0x7F IZLAZ: Najveci: 111111000000000000000000000000 Najmanji: 000000000000000000000000111111 </pre>	<p><i>Test 2</i></p> <pre> ULAZ: 0x80 IZLAZ: Najveci: 100000000000000000000000000000 Najmanji: 000000000000000000000000000001 </pre>
<p><i>Test 3</i></p> <pre> ULAZ: 0x00FF00FF IZLAZ: Najveci: 11111111111111110000000000000000 Najmanji: 00000000000000001111111111111111 </pre>	<p><i>Test 4</i></p> <pre> ULAZ: 0xFFFFFFFF IZLAZ: Najveci: 11111111111111111111111111111111 Najmanji: 11111111111111111111111111111111 </pre>

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`.

Napisati program koji testira prethodno napisane funkcije za neoznačene cele brojeve x , n , p , y koji se unose sa standardnog ulaza. Na standardni izlaz ispisati binarne reprezenatacije brojeva x i y , a zatim i binarne reprezentacije brojeva koji se dobijaju pozivanjem prethodno napisanih funkcija. NAPOMENA: *Bit najmanje težine je krajnji desni bit i njegova pozicija se označava nultom dok se pozicije ostalih bitova uvećavaju za jedan, sa desna na levo.*

[illegible]

```

INTERAKCIJA SA PROGRAMOM:
  Unesite neoznaceni ceo broj x: 2882398951
  Unesite neoznaceni ceo broj n: 5
  Unesite neoznaceni ceo broj p: 10
  Unesite neoznaceni ceo broj y: 35156526

  x = 2882398951 = 10101011110011011110101011100111
  postavi_0(2882398951, 5, 10) = 10101011110011011110100000100111

  x = 2882398951 = 10101011110011011110101011100111
  postavi_1(2882398951, 5, 10) = 10101011110011011110111111100111

  x = 2882398951 = 10101011110011011110101011100111
  vrati_bitove(2882398951, 5, 10) = 00000000000000000000000000000101

  x = 2882398951 = 10101011110011011110101011100111
  y = 35156526 = 00000010000110000111001000101110
  postavi_1_n_bitove(2882398951, 5, 10, 35156526) = 10101011110011011110101110100111

  x = 2882398951 = 101010111100110111101010111100111
  invertuj(2882398951, 5, 10) = 10101011110011011110101100100111

```

Zadatak 1.9 Pod rotiranjem bitova ulevo podrazumeva se pomeranje svih bitova za jednu poziciju ulevo, s tim što se bit sa pozicije najveće težine pomera 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, tatim 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) = 1100011111110100010111101110010
```

Zadatak 1.10 Napisati funkciju `unsigned ogledalo(unsigned x)` koja vraća ceo 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:
0000000000000000000000001001010101
1010101001000000000000000000000000
```

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:  
    000000000000000000000000000000001010
```

```
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	Test 2	Test 3
<pre> ULAZ: 2 IZLAZ: 1 2 2 1 </pre>	<pre> ULAZ: 3 1 2 3 1 3 2 2 1 3 2 3 1 3 1 2 3 2 1 </pre>	<pre> ULAZ: -5 Duzina permutacije mora biti broj iz intervala [0, 15]! </pre>

* **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.

- (a) Napisati rekurzivnu funkciju koja izračunava vrednost binomnog koeficijenta $\binom{n}{k}$ koristeći osobine Paskalovog trougla.
- (b) 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 iscrtava Paskalov trougao, a zatim štampa sumu elemenata hipotenuze.

Test 1	Test 2
<pre> 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 </pre>	<pre> 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 </pre>

* **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;
9   float imag;
10  } KompleksanBroj;

12  /* Funkcija ucitava sa standardnog ulaza realan i imaginara deo
13     kompleksnog broja i smesta ih u strukturu cija je adresa argument
14     funkcije */
15  void ucitaj_kompleksan_broj(KompleksanBroj * z)
16  {
17      /* Ucitavanje vrednosti sa standardnog ulaza */
18      printf("Unesite realni i imaginarni deo kompleksnog broja: ");
19      scanf("%f", &z->real);
20      scanf("%f", &z->imag);
21  }

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

29      /* Razlikuju se dva slucaja: 1) realni deo kompleksnog broja
30         razlicit od nule: tada se realni deo ispisuje na standardni
31         izlaz, nakon cega se ispisuje znak + ili - u zavisnosti da li je
32         imaginarni deo pozitivan ili negativan, a potom i apsolutna
33         vrednost imaginarnog dela kompleksnog broja 2) realni deo
34         kompleksnog broja je nula: tada se samo ispisuje imaginaran deo,
35         s tim sto se ukoliko su oba dela nula ispisuje samo 0, bez
36         decimalnih mesta */

37      if (z.real != 0) {
38          printf("%.2f", z.real);

39          if (z.imag > 0)
40              printf(" + %.2f i", z.imag);
41          else if (z.imag < 0)
42              printf(" - %.2f i", -z.imag);
43      } else {
44          if (z.imag == 0)
45              printf("0");
46          else
47              printf("%.2f i", z.imag);
48      }

49      /* Zavrшава se sa ispisom */
50      printf(")");
51  }

52  /* Funkcija vraca vrednosti realnog dela kompleksnog broja */

```

1 Uvodni zadaci

```
60 float realan_deo(KompleksanBroj z)
61 {
62     return z.real;
63 }
64
65 /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
66 float imaginaran_deo(KompleksanBroj z)
67 {
68     return z.imag;
69 }
70
71 /* Funkcija vraca vrednost modula zadatog kompleksnog broja */
72 float moduo(KompleksanBroj z)
73 {
74     return sqrt(z.real * z.real + z.imag * z.imag);
75 }
76
77 /* Funkcija vraca vrednost konjugovano kompleksnog broja koji
78    odgovara kompleksnom broju argumentu */
79 KompleksanBroj konjugovan(KompleksanBroj z)
80 {
81     /* Konjugovano kompleksan broj z se dobija tako sto se promeni znak
82        imaginarnom delu kompleksnog broja */
83
84     KompleksanBroj z1 = z;
85
86     z1.imag *= -1;
87
88     return z1;
89 }
90
91 /* Funkcija vraca kompleksan broj cija vrednost je jednaka zbiru
92    argumenata funkcije */
93 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
94 {
95     /* Rezultat sabiranja dva kompleksna broja z1 i z2 je kompleksan
96        broj ciji je realan deo zbir realnih delova kompleksnih brojeva
97        z1 i z2, a imaginaran deo zbir imaginarnih delova kompleksnih
98        brojeva z1 i z2 */
99
100    KompleksanBroj z = z1;
101
102    z.real += z2.real;
103    z.imag += z2.imag;
104
105    return z;
106 }
107
108 /* Funkcija vraca kompleksan broj cija vrednost je jednaka razlici
109    argumenata funkcije */
110 KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2)
111 {
```

```
112  /* Rezultat oduzimanja dva kompleksna broja z1 i z2 je kompleksan
113     broj ciji je realan deo razlika realnih delova kompleksnih
114     brojeva z1 i z2, a imaginaran deo razlika imaginarnih delova
115     kompleksnih brojeva z1 i z2 */
116
117     KomplexanBroj z = z1;
118
119     z.real -= z2.real;
120     z.imag -= z2.imag;
121
122     return z;
123 }
124
125 /* Funkcija vraca kompleksan broj cija vrednost je jednaka proizvodu
126     argumenata funkcije */
127 KomplexanBroj mnozi(KomplexanBroj z1, KomplexanBroj z2)
128 {
129     /* Rezultat mnozenja dva kompleksna broja z1 i z2 je kompleksan
130        broj ciji se realan i imaginaran deo racunaju po formuli za
131        mnozenje kompleksnih brojeva z1 i z2 */
132
133     KomplexanBroj z;
134
135     z.real = z1.real * z2.real - z1.imag * z2.imag;
136     z.imag = z1.real * z2.imag + z1.imag * z2.real;
137
138     return z;
139 }
140
141 /* Funkcija vraca argument zadatog kompleksnog broja */
142 float argument(KomplexanBroj z)
143 {
144     /* Argument kompleksnog broja z se racuna pozivanjem funkcije atan2
145        iz biblioteke math.h */
146
147     return atan2(z.imag, z.real);
148 }
149
150 int main()
151 {
152     char c;
153
154     /* Deklaracija 3 promenljive tipa KomplexanBroj */
155     KomplexanBroj z1, z2, z;
156
157     /* Ucitavanje prvog kompleksnog broja, a potom i njegovo
158        ispisivanje na standardni izlaz */
159     ucitaj_komplexan_broj(&z1);
160     ispisi_komplexan_broj(z1);
161     printf("\n");
162
163     /* Ucitavanje drugog kompleksnog broja, a potom njegovo ispisivanje
```

```
164     na_standardni_izlaz */
165     ucitaj_kompleksan_broj(&z2);
166     ispisi_kompleksan_broj(z2);
167     printf("\n");
168
169     /* Ucitavanje i provera znaka na osnovu koga korisnik bira
170        aritmeticku operaciju koja ce se izvršiti nad kompleksnim
171        brojevima */
172     getchar();
173     printf("Unesite znak (+,-,*): ");
174     scanf("%c", &c);
175     if (c != '+' && c != '-' && c != '*') {
176         printf("Greska: nedozvoljena vrednost operatora!\n");
177         exit(EXIT_FAILURE);
178     }
179
180     /* Analizira se uneti operator */
181     if (c == '+') {
182         /* Racuna se zbir */
183         z = saberi(z1, z2);
184     } else if (c == '-') {
185         /* Racuna se razlika */
186         z = oduzmi(z1, z2);
187     } else {
188         /* Racuna se proizvod */
189         z = mnozi(z1, z2);
190     }
191
192     /* Ispisuje se rezultat */
193     ispisi_kompleksan_broj(z1);
194     printf(" %c ", c);
195     ispisi_kompleksan_broj(z2);
196     printf(" = ");
197     ispisi_kompleksan_broj(z);
198
199     /* Ispisuje se realan, imaginaran deo i moduo prvog kompleksnog
200        broja */
201     printf("\nRealni_deo: %.f\nImaginarni_deo: %.f\nModuo: %.f\n",
202           realan_deo(z), imaginaran_deo(z), moduo(z));
203
204     /* Izracunava se i ispisuje konjugovano kompleksan broj drugog
205        kompleksnog broja */
206     printf("Konjugovano kompleksan broj: ");
207     ispisi_kompleksan_broj(konjugovan(z));
208     printf("\n");
209
210     /* Testira se funkcija koja racuna argument kompleksnog broja */
211     printf("Argument kompleksnog broja: %.f\n", argument(z));
212
213     exit(EXIT_SUCCESS);
214 }
```

Rešenje 1.2

kompleksan_broj.h

```

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

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

15 /* Zaglavlja standardne biblioteke koje sadrže deklaracije funkcija
   koje se koriste u definicijama funkcija navedenim u
   kompleksan_broj.c */
17 #include <stdio.h>
   #include <math.h>

19 /* Struktura KompleksanBroj */
   typedef struct {
21     float real;
       float imag;
23 } KompleksanBroj;

25 /* Deklaracije funkcija za rad sa kompleksnim brojevima. Sve one su
   definisane u kompleksan_broj.c */
27

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

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

37 /* Funkcija vraca vrednosti realnog dela kompleksnog broja */
   float realan_deo(KompleksanBroj z);
39

41 /* Funkcija vraca vrednosti imaginarnog dela kompleksnog broja */
   float imaginaran_deo(KompleksanBroj z);
43

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

   /* Funkcija vraca vrednost konjugovano kompleksnog broja koji

```

1 Uvodni zadaci

```
47     odgovara kompleksnom broju argumentu */
KompleksanBroj konjugovan(KompleksanBroj z);
49
/* Funkcija vraća kompleksan broj čija vrednost je jednaka zbiru
51 argumenata funkcije */
KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2);
53
/* Funkcija vraća kompleksan broj čija vrednost je jednaka razlici
55 argumenata funkcije */
KompleksanBroj oduzmi(KompleksanBroj z1, KompleksanBroj z2);
57
/* Funkcija vraća kompleksan broj čija vrednost je jednaka proizvodu
59 argumenata funkcije */
KompleksanBroj mnozi(KompleksanBroj z1, KompleksanBroj z2);
61
/* Funkcija vraća argument zadatog kompleksnog broja */
63 float argument(KompleksanBroj z);
65
/* Kraj zaključanog dela */
#endif
```

kompleksan_broj.c

```
/* Uključuje se zaglavlje za rad sa kompleksnim brojevima, jer je
2 neophodno da bude poznata definicija tipa KompleksanBroj. Takođe,
time su uključena zaglavlja standardne biblioteke koja su navedena
4 u kompleksan_broj.h */
#include "kompleksan_broj.h"
6
8 void ucitaj_kompleksan_broj(KompleksanBroj * z)
{
10     /* Učitavanje vrednosti sa standardnog ulaza */
    printf("Unesite realan i imaginarni 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
22     /* Razlikuju se dva slučaja: 1) realni deo kompleksnog broja
    različit od nule: tada se realni deo ispisuje na standardni
    izlaz, nakon čega se ispisuje znak + ili - u zavisnosti da li je
24 imaginarni deo pozitivan ili negativan, a potom i apsolutna
    vrednost imaginarnog dela kompleksnog broja 2) realni deo
26 kompleksnog broja je nula: tada se samo ispisuje imaginarni deo,
    s tim što se ukoliko su oba dela nula ispisuje samo 0, bez
    decimalnih mesta */
28     printf("%f", z.real);
    if (z.imag != 0)
    {
        printf("%f", z.imag);
        if (z.imag > 0)
            printf(" + ");
        else
            printf(" - ");
        printf("%f", abs(z.imag));
    }
    printf(")");
}
```



```
30     if (z.real != 0) {
31         printf("%.2f", z.real);
32
33         if (z.imag > 0)
34             printf(" + %.2f i", z.imag);
35         else if (z.imag < 0)
36             printf(" - %.2f i", -z.imag);
37     } else {
38         if (z.imag == 0)
39             printf("0");
40         else
41             printf("%.2f i", z.imag);
42     }
43
44     /* Završava se sa ispisom */
45     printf("\n");
46 }
47
48 float realan_deo(KompleksanBroj z)
49 {
50     /* Vraca se vrednost realnog dela kompleksnog broja */
51     return z.real;
52 }
53
54 float imaginaran_deo(KompleksanBroj z)
55 {
56     /* Vraca se vrednost imaginarnog dela kompleksnog broja */
57     return z.imag;
58 }
59
60 float moduo(KompleksanBroj z)
61 {
62     /* Koriscenjem funkcije sqrt racuna se moduo kompleksnog broja */
63     return sqrt(z.real * z.real + z.imag * z.imag);
64 }
65
66 KompleksanBroj konjugovan(KompleksanBroj z)
67 {
68     /* Konjugovano kompleksan broj se dobija od datog broja z tako sto
69        se promeni znak imaginarnom delu kompleksnog broja */
70     KompleksanBroj z1 = z;
71     z1.imag *= -1;
72     return z1;
73 }
74
75 KompleksanBroj saberi(KompleksanBroj z1, KompleksanBroj z2)
76 {
77     /* Rezultat sabiranja dva kompleksna broja z1 i z2 je kompleksan
78        broj ciji je realan deo zbir realnih delova kompleksnih brojeva
79        z1 i z2, a imaginaran deo zbir imaginarnih delova kompleksnih
80        brojeva z1 i z2 */
```

1 Uvodni zadaci

```
82     KomplexsanBroj z = z1;
84     z.real += z2.real;
86     z.imag += z2.imag;
88     return z;
89 }
90 KomplexsanBroj oduzmi(KomplexsanBroj z1, KomplexsanBroj z2)
91 {
92     /* Rezultat oduzimanja dva kompleksna broja z1 i z2 je kompleksan
93        broj ciji je realan deo razlika realnih delova kompleksnih
94        brojeva z1 i z2, a imaginaran deo razlika imaginarnih delova
95        kompleksnih brojeva z1 i z2 */
96     KomplexsanBroj z = z1;
97     z.real -= z2.real;
98     z.imag -= z2.imag;
99     return z;
100 }
101 KomplexsanBroj mnozi(KomplexsanBroj z1, KomplexsanBroj z2)
102 {
103     /* Rezultat mnozenja dva kompleksna broja z1 i z2 je kompleksan
104        broj ciji se realan i imaginaran deo racunaju po formuli za
105        mnozenje kompleksnih brojeva z1 i z2 */
106     KomplexsanBroj z;
107
108     z.real = z1.real * z2.real - z1.imag * z2.imag;
109     z.imag = z1.real * z2.imag + z1.imag * z2.real;
110
111     return z;
112 }
113 float argument(KomplexsanBroj z)
114 {
115     /* Argument kompleksnog broja z se racuna pozivanjem funkcije atan2
116        iz biblioteke math.h */
117     return atan2(z.imag, z.real);
118 }
```

main.c

```
1  /*****
2  Ovaj program koristi korektno definisanu biblioteku kompleksnih
3  brojeva. U zaglavlju kompleksan_broj.h nalazi se definicija
4  kompleksnog broja i popis deklaracija podrzanih funkcija, a u
5  kompleksan_broj.c se nalaze njihove definicije.
6
7  Kompilacija programa se najjednostavnije postize naredbom
8  gcc -Wall -lm -o kompleksan_broj kompleksan_broj.c main.c
9  *****/
```

```

Kompilacija se može uraditi i na sledeći način:
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 vrši 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>
25 /* Uključuje se zaglavlje neophodno za rad sa kompleksnim brojevima
*/
#include "kompleksan_broj.h"

27 /* U glavnoj funkciji se za uneti kompleksan broj ispisuje njegov
polarni oblik */
29 int main()
31 {
    KompleksanBroj z;

33     /* Učitavamo kompleksan broj */
35     učitaj_kompleksan_broj(&z);

37     /* Ispisivanje polarnog oblika kompleksnog broja */
printf("Polarni oblik kompleksnog broja je %.2f * e~i * %.2f\n",
39     moduo(z), argument(z));

41     return 0;
}

```

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 čuva koeficijente

```

1 Uvodni zadaci

```
13      (koef[i] je koeficijent uz clan xi) i stepen polinoma */
14  typedef struct {
15      double koef[MAKS_STEPEN + 1];
16      int stepen;
17  } Polinom;
18
19  /* Funkcija koja ispisuje polinom na standardni izlaz u citljivom
20     obliku. Polinom se prenosi po adresi da bi se uštedela memorija:
21     ne kopira se cela struktura, vec se samo prenosi adresa na kojoj
22     se nalazi polinom koji ispisujemo */
23  void ispisi(const Polinom * p);
24
25  /* Funkcija koja ucitava polinom sa tastature */
26  Polinom ucitaj();
27
28  /* Funkcija racuna i vraca vrednost polinoma p u tacki x Hornerovim
29     algoritmom */
30  double izracunaj(const Polinom * p, double x);
31
32  /* Funkcija koja sabira dva polinoma */
33  Polinom saberi(const Polinom * p, const Polinom * q);
34
35  /* Funkcija koja mnozi dva polinoma p i q */
36  Polinom pomnozi(const Polinom * p, const Polinom * q);
37
38  /* Funkcija koja racuna izvod polinoma p */
39  Polinom izvod(const Polinom * p);
40
41  /* Funkcija koja racuna n-ti izvod polinoma p */
42  Polinom n_izvod(const Polinom * p, int n);
43  #endif
```

polinom.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "polinom.h"
4
5  void ispisi(const Polinom * p)
6  {
7      int nulaPolinom = 1;
8      int i;
9      /* Ispisivanje polinoma pocinje od najviseg stepena ka najnižem da
10         bi polinom bio ispisan na prirodan nacin. Ispisuju se samo oni
11         koeficijenti koji su razliciti od nule. Ispred pozitivnih
12         koeficijenata je potrebno ispisati znak + (osim u slucaju
13         koeficijenta uz najvisi stepen). */
14     for (i = p->stepen; i >= 0; i--) {
15
16         if (p->koef[i]) {
17             /* Polinom nije nula polinom, cim je neki od koeficijenata
```

```

18         razlicit od nule */
        nulaPolinom = 0;
20     if (p->koef[i] >= 0 && i != p->stepen)
        putchar('+');
22     if (i > 1)
        printf("%.2fx~%d", p->koef[i], i);
24     else if (i == 1)
        printf("%.2fx", p->koef[i]);
26     else
        printf("%.2f", p->koef[i]);
28 }
    }
30 /* U slucaju nula polinoma indikator ce imati vrednost 1 i tada se
    ispisuje nula. */
32 if(nulaPolinom)
    printf("0");
34 putchar('\n');
}

36 Polinom ucitaj()
37 {
38     int i;
40     Polinom p;

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

44     /* Ponavlja se ucitavanje stepena sve dok se ne unese stepen iz
    dozvoljenog opsega */
46     while (p.stepen > MAKS_STEPEN || p.stepen < 0) {
48         printf("Stepen polinoma pogresno unet, pokusajte ponovo: ");
        scanf("%d", &p.stepen);
50     }

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

56     /* Vraca se procitani polinom */
    return p;
58 }

60 double izracunaj(const Polinom * p, double x)
61 {
62     /* Rezultat se na pocetku inicijalizuje na nulu, a potom se u
    svakoj iteraciji najpre mnozi sa x, a potom i uvecava za
64     vrednost odgovarajuceg koeficijenta */

66     /* Primer: Hornerov algoritam za polinom  $x^4+2x^3+3x^2+2x+1$ :
     $x^4+2x^3+3x^2+2x+1 = ((x+2)*x + 3)*x + 2)*x + 1$  */
68     double rezultat = 0;

```

```
70     int i = p->stepen;
71     for (; i >= 0; i--)
72         rezultat = rezultat * x + p->koef[i];
73     return rezultat;
74 }

76 Polinom saberi(const Polinom * p, const Polinom * q)
77 {
78     Polinom rez;
79     int i;
80
81     /* Stepen rezultata ce odgovarati stepenu polinoma sa vecim
82        stepenom */
83     rez.stepen = p->stepen > q->stepen ? p->stepen : q->stepen;
84
85     /* Racunaju se svi koeficijenti rezultujucega polinoma tako sto se
86        sabiraju koeficijenti na odgovarajucim pozicijama polinoma koje
87        sabiramo. Ukoliko je pozicija za koju se racuna koeficijent veca
88        od stepena nekog od polaznih polinoma podrazumeva se da je
89        koeficijent jednak koeficijentu uz odgovarajuci stepen iz drugog
90        polinoma */
91     for (i = 0; i <= rez.stepen; i++)
92         rez.koef[i] =
93             (i > p->stepen ? 0 : p->koef[i]) +
94             (i > q->stepen ? 0 : q->koef[i]);
95
96     /* Vraca se dobijeni polinom */
97     return rez;
98 }

100 Polinom pomnozi(const Polinom * p, const Polinom * q)
101 {
102     int i, j;
103     Polinom r;
104
105     /* Stepen rezultata ce odgovarati zbiru stepena polaznih polinoma
106        */
107     r.stepen = p->stepen + q->stepen;
108     if (r.stepen > MAKS_STEPEN) {
109         fprintf(stderr, "Stepen proizvoda polinoma izlazi iz opsega\n");
110         exit(EXIT_FAILURE);
111     }
112
113     /* Svi koeficijenti rezultujucega polinoma se inicijalizuju na nulu
114        */
115     for (i = 0; i <= r.stepen; i++)
116         r.koef[i] = 0;
117
118     /* U svakoj iteraciji odgovarajuci koeficijent rezultata se uvecava
119        za proizvod odgovarajucih koeficijenata iz polaznih polinoma */
120     for (i = 0; i <= p->stepen; i++)
```

```
120     for (j = 0; j <= q->stepen; j++)
121         r.koef[i + j] += p->koef[i] * q->koef[j];
122
123     /* Vraca se dobijeni polinom */
124     return r;
125 }
126
127 Polinom izvod(const Polinom * p)
128 {
129     int i;
130     Polinom r;
131
132     /* Izvod polinoma ce imati stepen za jedan stepen manji od stepena
133        polaznog polinoma. Ukoliko je stepen polinoma p vec nula, onda
134        je rezultujuci polinom nula (izvod od konstante je nula). */
135     if (p->stepen > 0) {
136         r.stepen = p->stepen - 1;
137
138         /* Racunanje koeficijenata rezultata na osnovu koeficijenata
139            polaznog polinoma */
140         for (i = 0; i <= r.stepen; i++)
141             r.koef[i] = (i + 1) * p->koef[i + 1];
142     } else
143         r.koef[0] = r.stepen = 0;
144
145     /* Vraca se dobijeni polinom */
146     return r;
147 }
148
149 Polinom n_izvod(const Polinom * p, int n)
150 {
151     int i;
152     Polinom r;
153
154     /* Provera da li je n nenegativna vrednost */
155     if (n < 0) {
156         fprintf(stderr, "U n-tom izvodu polinoma, n mora biti >=0 \n");
157         exit(EXIT_FAILURE);
158     }
159
160     /* Multi izvod je bas taj polinom */
161     if (n == 0)
162         return *p;
163
164     /* Za n>=1, n-ti izvod se racuna tako sto se n puta pozove funkcija
165        za racunanje prvog izvoda polinoma */
166     r = izvod(p);
167     for (i = 1; i < n; i++)
168         r = izvod(&r);
169
170     /* Vraca se dobijeni polinom */
171     return r;
```

1 Uvodni zadaci

172 | }

main.c

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

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

10     /* Unos polinoma p */
    printf
12     ("Unesite polinom p (prvo stepen, pa zatim koeficijente od
        najveceg stepena do nultog):\n");
    p = ucitaj();

14     /* Ispis polinoma p */
    ispisi(&p);

16     /* Unos polinoma q */
    printf
20     ("Unesite drugi polinom q (prvo stepen, pa zatim koeficijente
        od najveceg stepena do nultog):\n");
    q = ucitaj();

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

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

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

38     /* Racuna se n-ti izvoda polinoma i ispisuje se dobijeni polinoma
        */
    printf("Unesite izvod polinoma koji zelite:\n");
    scanf("%d", &n);
42     r = n_izvod(&r, n);
    printf("%d. izvod polinoma r je: ", n);
```



```

44     ispisi(&r);
46     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 bita
9   najveće težine ka bitu najmanje težine */
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. */
18 void stampaj_bitove_char(char x);
19
20 #endif

```

stampanje_bitova.c

```

1  #include <stdio.h>
2  #include "stampanje_bitova.h"
3
4  void stampaj_bitove(unsigned x)
5  {
6      /* Broj bitova celog broja */
7      unsigned velicina = sizeof(unsigned) * 8;
8
9      /* Maska koja se koristi za "ocitavanje" bitova celog broja */
10     unsigned maska;
11
12     /* Pocetna vrednost maske se postavlja na broj ciji binarni zapis
13     na mestu bita najveće težine sadrži jedinicu, a na svim ostalim
14     mestima sadrži nulu. U svakoj iteraciji maska se menja tako sto
15     se jedini bit jedinica pomera udesno, kako bi se odredio naredni
16     bit broja x koji je argument funkcije. Zatim se odgovarajuca

```

1 Uvodni zadaci

```
18         cifra, ('0' ili '1'), ispisuje na standardnom izlazu. Neophodno
20         je da promenljiva maska bude deklarirana kao neoznaceni broj
22         kako bi se pomeranjem u desno vrsilo logicko pomeranje
24         (popunjavanje nulama), a ne aritmeticko pomeranje (popunjavanje
26         znakom broja). */
28     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
29         putchar(x & maska ? '1' : '0');
31     putchar('\n');
32 }
33
34 void stampaj_bitove_short(short x)
35 {
36     /* Broj bitova celog broja tipa short */
37     unsigned velicina = sizeof(short) * 8;
38
39     /* Maska koja se koristi za "ocitavanje" bitova broja tipa short */
40     unsigned short maska;
41
42     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
43         putchar(x & maska ? '1' : '0');
44
45     putchar('\n');
46 }
47
48 void stampaj_bitove_char(char x)
49 {
50     /* Broj bitova karaktera */
51     unsigned velicina = sizeof(char) * 8;
52
53     /* Maska koja se koristi za "ocitavanje" bitova jednog karaktera */
54     unsigned char maska;
55
56     for (maska = 1 << (velicina - 1); maska != 0; maska >>= 1)
57         putchar(x & maska ? '1' : '0');
58
59     putchar('\n');
60 }
```

main.c

```
1 #include <stdio.h>
2 #include "stampanje_bitova.h"
3
4 int main()
5 {
6     int broj_int;
7     short broj_short;
8     char broj_char;
9
10    /* Ucitavanje broja tipa int */
```

```

12 printf("Unesite broj tipa int: ");
   scanf("%x", &broj_int);

14 /* Ispisivanje binarne reprezentacije unetog broja */
   printf("Binarna reprezentacija: ");
16   stampaj_bitove(broj_int);

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

22 /* Ispisivanje binarne reprezentacije unetog broja */
   printf("Binarna reprezentacija: ");
24   stampaj_bitove_short(broj_short);

26 /* Ucitavanje broja tipa char */
   printf("Unesite broj tipa char: ");
28   scanf("%hxx", &broj_char);

30 /* Ispisivanje binarne reprezentacije unetog broja */
   printf("Binarna reprezentacija: ");
32   stampaj_bitove_char(broj_char);

34   return 0;
   }

```

Rešenje 1.6

```

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

3

5  /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x
   kreiranjem odgovarajuće maske i njenim pomeranjem */
   int prebroj_bitove_1(int x)
7  {
   {
   int br = 0;
9   unsigned broj_pomeranja = sizeof(unsigned) * 8 - 1;

11  /* Formiranje se maska čija binarna reprezentacija izgleda
   100000...0000000, koja služi za očitavanje bita najveće težine.
13   U svakoj iteraciji maska se pomera u desno za 1 mesto, i
   očitavamo sledeći bit. Petlja se završava kada više nema
15   jedinica tj. kada maska postane nula. */
   unsigned maska = 1 << broj_pomeranja;
17   for (; maska != 0; maska >>= 1)
       x & maska ? br++ : 1;

19   return br;
21 }

23 /* Funkcija vraca broj jedinica u binarnoj reprezentaciji broja x

```

```
    formiranjem odgovarajuće maske i pomeranjem promenljive x */
25 int prebroj_bitove_2(int x)
{
27     int br = 0;
    unsigned broj_pomeranja = sizeof(int) * 8 - 1;

29     /* Kako je argument funkcije oznacen ceo broj x naredba x>>=1 bi
31     vrsila aritmeticko pomeranje u desno, tj. popunjavanje bita
    najveće težine bitom znaka. U tom slučaju nikad ne bi bio
33     ispunjen uslov x!=0 i program bi bio zarobljen u beskonacnoj
    petlji. Zbog toga se koristi pomeranje broja x ulevo i maska
35     koja očitava bit najveće težine. */

37     unsigned maska = 1 << broj_pomeranja;
    for (; x != 0; x <= 1)
39         x & maska ? br++ : 1;

41     return br;
}

43 int main()
45 {
    int x, i;

47     /* Ucitava se broj sa ulaza */
    printf("Unesite broj:\n");
    scanf("%x", &x);

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

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

69     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 vraca najveći neoznačeni broj sastavljen od istih bitova
   5     koji se nalaze u binarnoj reprezentaciji vrednosti promenjive x */
   unsigned najveći(unsigned x)
7   {
       unsigned velicina = sizeof(unsigned) * 8;
9
       /* Formira se maska 100000...0000000 */
11      unsigned maska = 1 << (velicina - 1);
13
       /* Rezultat se inicijalizuje vrednoscu 0 */
       unsigned rezultat = 0;
15
       /* Promenljiva x se pomera u levo sve dok postoje jedinice u njenoj
17         binarnoj reprezentaciji (tj. sve dok je promenljiva x razlicita
         od nule). */
19      for (; x != 0; x <<= 1) {
          /* Za svaku jedinicu koja se koriscenjem maske detektuje na
21             poziciji najveće težine u binarnoj reprezentaciji promenjive
             x, potiskuje se jedna nova jedinicu sa leva u rezultat */
23         if (x & maska) {
             rezultat >>= 1;
25             rezultat |= maska;
           }
27     }
29
       /* Vraca se dobijena vrednost */
       return rezultat;
31 }
33
   /* Funkcija vraca najmanji neoznačeni broj sastavljen od istih bitova
   koji se nalaze u binarnoj reprezentaciji vrednosti promenjive x */
35 unsigned najmanji(unsigned x)
   {
37     /* Rezultat se inicijalizuje vrednoscu 0 */
       unsigned rezultat = 0;
39
       /* Promenljiva x se pomera u desno sve dok postoje jedinice u
41         njenoj binarnoj reprezentaciji (tj. sve dok je promenljiva x
         razlicita od nule). */
43     for (; x != 0; x >>= 1) {
          /* Za svaku jedinicu koja se koriscenjem vrednosti 1 za masku
45             detektuje na poziciji najmanje težine u binarnoj
             reprezentaciji promenjive x, potiskuje se jedna nova jedinicu
47             sa desna u rezultat */
           if (x & 1) {
29             rezultat <<= 1;
49             rezultat |= 1;
           }
51     }
```

```

    }
53
    /* Vraca se dobijena vrednost */
55    return rezultat;
    }
57
int main()
59 {
    int broj;
61
    /* Ucitava se broj sa ulaza */
63    scanf("%x", &broj);
65
    /* Ispisuju se, redom, najveći i najmanji broj formirani od bitova
       unetog broja */
67    printf("Najveći:\n");
    stampaj_bitove(najveći(broj));
69
    printf("Najmanji:\n");
71    stampaj_bitove(najmanji(broj));
73
    return 0;
}

```

Rešenje 1.8

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

```

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

4  /* Funkcija postavlja na nulu n bitova pocev od pozicije p. */
unsigned postavi_0(unsigned x, unsigned n, unsigned p)
6  {
    /******
8     Formira se maska cija binarna reprezentacija ima n bitova
       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
       1111 1111 1111 1111 1111 1000 0011 1111
12    To se postize na sledeci nacin:
       ~0                1111 1111 1111 1111 1111 1111 1111 1111
14    (~0 << n)           1111 1111 1111 1111 1111 1111 1110 0000
       ~(~0 << n)         0000 0000 0000 0000 0000 0000 0001 1111
16    ~(~0 << n) << (p-n+1) 0000 0000 0000 0000 0000 0111 1100 0000
       ~(~0 << n) << (p-n+1) 1111 1111 1111 1111 1111 1000 0011 1111
18    *****/
    unsigned maska = ~(~0 << n) << (p - n + 1);
20
    return x & maska;
22 }

```

```

24 /* Funkcija postavlja na jedinicu n bitova pocev od pozicije p. */
    unsigned postavi_1(unsigned x, unsigned n, unsigned p)
26 {
28     /******
        Formira se maska kod koje je samo n bitova pocev od
30     pozicije p jednako 1, a ostali su 0.
        Na primer, za n=5 i p=10 formira se maska oblika
32     0000 0000 0000 0000 0000 0111 1100 0000
        *****/
34     unsigned maska = ~(~0 << n) << (p - n + 1);

36     return x | maska;
    }

38 /* Funkcija vraca celobrojno polje bitova, desno poravnato, koje
40     predstavlja n bitova pocev od pozicije p u binarnoj
        reprezentaciji broja x. */
42     unsigned vrati_bitove(unsigned x, unsigned n, unsigned p)
    {
44         /******
            Kreira se maska kod koje su poslednjih n bitova 1, a ostali su 0.
            Na primer, za n=5
46             0000 0000 0000 0000 0000 0000 0001 1111
            *****/
50         unsigned maska = ~(~0 << n);

52         /* Najpre se vrednost promenljive x pomera u desno tako da trazeno
            polje bude uz desni kraj. Zatim se maskiraju ostali bitovi, sem
54             zeljenih n i funkcija vraca tako dobijenu vrednost */
        return maska & (x >> (p - n + 1));
56     }

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

66     /* Kao i kod funkcije postavi_0, i ovde se kreira maska koja ima n
        bitova postavljenih na 0 pocevsi od pozicije p, dok su
68         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 n
74         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
78     binarne reprezentacije vrednosti promenljive y cijih je n bitova
    najnize tezine pomera tako da stoje pocev od pozicije p. Ostali
80     bitovi su nule. Sa (y & poslednjih_n_1) postavimo na 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");
    scanf("%u", &x);
106     printf("Unesite neoznaceni broj n:\n");
    scanf("%u", &n);
108     printf("Unesite neoznaceni broj p:\n");
    scanf("%u", &p);
110     printf("Unesite neoznaceni broj y:\n");
    scanf("%u", &y);
112

    /* Ispisuju se binarne reprezentacije broja x i broja koji se
114     dobije kada se primeni funkcija postavi_0 za x, n i p*/
    printf("x = %10u %36s = ", x, "");
116     stampaj_bitove(x);
    printf("postavi_0(%10u,%6u,%6u)%16s = ", x, n, p, "");
118     stampaj_bitove(postavi_0(x, n, p));
    printf("\n");
120

    /* Ispisuju se binarne reprezentacije broja x i broja koji se
122     dobije kada se primeni funkcija postavi_1 za x, n i p*/
    printf("x = %10u %36s = ", x, "");
124     stampaj_bitove(x);
    printf("postavi_1(%10u,%6u,%6u)%16s = ", x, n, p, "");
126     stampaj_bitove(postavi_1(x, n, p));
```



```

printf("\n");

/* Ispisuju se binarne reprezentacije broja x i broja koji se
dobiye kada se primeni funkcija vrati_bitove za x, n i p*/
printf("x = %10u %36s = ", x, "");
stampaj_bitove(x);
printf("vrati_bitove(%10u,%6u,%6u)%13s = ", x, n, p, "");
stampaj_bitove( vrati_bitove(x, n, p));
printf("\n");

/* Ispisuju se binarne reprezentacije brojeva x, y i broja koji se
dobiye kada se primeni funkcija postavi_1_n_bitova za x, n i p*/
printf("x = %10u %36s = ", x, "");
stampaj_bitove(x);
printf("y = %10u %36s = ", y, "");
stampaj_bitove(y);
printf("postavi_1_n_bitova(%10u,%4u,%4u,%10u) = ", x, n, p, y);
stampaj_bitove( postavi_1_n_bitova(x, n, p, y));
printf("\n");

/* Ispisuju se binarne reprezentacije broja x i broja koji se
dobiye kada se primeni funkcija invertuj za x, n i p*/
printf("x = %10u %36s = ", x, "");
stampaj_bitove(x);
printf("invertuj(%10u,%6u,%6u)%17s = ", x, n, p, "");
stampaj_bitove( invertuj(x, n, p));

return 0;
}

```

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 u levo 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 je neophodna da bi pre pomeranja u levo 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 rotaciju za jedan bit u levo. U svakoj iteraciji
    se odredi bit na poziciji najvece tezine, a potom se pomera

```

1 Uvodni zadaci

```
18     binarna reprezentacija trenutne vrednosti promenljive x u levo
20     za 1. Nakon toga, bit na poziciji najmanje tezine se postavlja
    na vrednost koju je imao bit na poziciji najveće težine koji je
    istisnut pomeranjem */
22     for (i = 0; i < n; i++) {
        bit_najvece_tezine = x & bit_najvece_tezine_maska;
        x = x << 1 | (bit_najvece_tezine ? 1 : 0);
24     }

26     /* Vraca se dobijena vrednost */
    return x;
28 }

30 /* Funkcija neoznaceni broj x rotira u desno za n mesta. */
unsigned rotiraj_udarno(unsigned x, unsigned n)
32 {
    unsigned bit_najmanje_tezine;
    int i;
34

36     /* n puta se ponavlja rotacija u desno za jedan bit. U svakoj
    iteraciji se određuje bit na poziciji najmanje težine broja x,
    zatim tako određeni bit se pomera u levo tako da bit na
    poziciji najmanje težine dodje do pozicije najveće težine.
    Zatim, nakon pomeranja binarne reprezentacije trenutne vrednosti
    promenljive x za 1 u desno, bit na poziciji najveće težine se
    postavlja na vrednost već zapamćenog bita koji je bio na
    poziciji najmanje težine. */
44     for (i = 0; i < n; i++) {
        bit_najmanje_tezine = x & 1;
        x = x >> 1 | bit_najmanje_tezine << (sizeof(unsigned) * 8 - 1);
46     }

48     /* Vraca se dobijena vrednost */
    return x;
50 }

52 /* Verzija funkcije koja broj x rotira u desno za n mesta, gde je
    argument funkcije x oznaceni ceo broj */
54 int rotiraj_udarno_oznaceni(int x, unsigned n)
56 {
    unsigned bit_najmanje_tezine;
    int i;
58

60     /* U svakoj iteraciji se određuje bit na poziciji najmanje težine
    i smesta u promenljivu bit_najmanje_tezine. Kako je x oznaceni
    ceo broj, tada se prilikom pomeranja u desno vrši aritmetičko
    pomeranje i čuva se znak broja. Dakle, razlikuju se dva slučaja
    u zavisnosti od znaka broja x. Nije dovoljno da se ova provera
    izvrši pre petlje, s obzirom da rotiranjem u desno na poziciju
    najveće težine može doći i 0 i 1, nezavisno od početnog znaka
    broja smestenog u promenljivu x. */
68     for (i = 0; i < n; i++) {
```

```

    bit_najmanje_tezine = x & 1;

70
    if (x < 0)
72 /*****
    Siftovanjem u desno broja koji je negativan dobija se 1 kao bit
74 na poziciji najveće težine. Na primer ako je x
    1010 1011 1100 1101 1110 0001 0010 001b
76 (sa b je oznacen ili 1 ili 0 na poziciji najmanje težine)
    Onda je sadržaj promenljive bit_najmanje_tezine:
78 0000 0000 0000 0000 0000 0000 0000 000b
    Nakon siftovanja sadržaja promenljive x za 1 u desno
80 1101 0101 1110 0110 1111 0000 1001 0001
    Kako bi umesto 1 na poziciji najveće težine u trenutnoj binarnoj
82 reprezentaciji x bilo postavljeno b nije dovoljno da se pomeri na
    poziciju najveće težine jer bi se time dobile 0, a u ovom slučaju
84 su potrebne jedinice zbog bitovskog & zato se prvo vrsi
    komplementiranje, a zatim pomeranje
86 ~bit_najmanje_tezine << (sizeof(int)*8 -1)
    B000 0000 0000 0000 0000 0000 0000 0000
88 gde B oznacava ~b.
    Potom se ponovo vrsi komplementiranje kako bi se b nalazilo na
90 poziciji najveće težine i sve jedinice na ostalim pozicijama
    ~(~bit_najmanje_tezine << (sizeof(int)*8 -1))
92 b111 1111 1111 1111 1111 1111 1111 1111
    *****/
94 x = (x >> 1) & ~(~bit_najmanje_tezine << (sizeof(int) * 8 - 1))
    ;
    else
96 x = (x >> 1) | bit_najmanje_tezine << (sizeof(int) * 8 - 1);
    }

    /* Vraca se dobijena vrednost */
100 return x;
}

102 int main()
104 {
    unsigned x, n;

106
    /* Ucitavanje vrednosti sa standardnog ulaza */
108 printf("Unesite neoznaceni broj x:");
    scanf("%x", &x);
110 printf("Unesite neoznaceni broj n:");
    scanf("%x", &n);
112
    /* Ispisivanje binarne reprezentacije broja x */
114 printf("x\t\t\t\t\t= ");
    stampaj_bitove(x);
116
    /* Testiranje rada napisanih funkcija */
118 printf("rotiraj_ulevo(%x,%u)\t\t\t= ", x, n);
    stampaj_bitove(rotiraj_ulevo(x, n));

```

1 Uvodni zadaci

```
120     printf("rotiraj_udesno(%x,%u)\t\t= ", x, n);
122     stampaj_bitove(rotiraj_udesno(x, n));

124     printf("rotiraj_udesno_oznaceni(%x,%u)\t= ", x, n);
126     stampaj_bitove(rotiraj_udesno_oznaceni(x, n));

128     printf("rotiraj_udesno_oznaceni(-%x,%u)\t= ", x, n);
130     stampaj_bitove(rotiraj_udesno_oznaceni(-x, n));

130     return 0;
}
```

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 vraća vrednost cija je binarna reprezentacija slika u
5   *ogledalu binarne reprezentacije broja x. */
6  unsigned ogledalo(unsigned x)
7  {
8      unsigned najnizi_bit;
9      unsigned rezultat = 0;
10
11     int i;
12     /* U svakoj iteraciji najnizi bit u binarnoj reprezentaciji tekuće
13      *vrednosti broja x se određuje i pamti u promenljivoj
14      *najnizi_bit, nakon čega se na promenljivu x primeni pomeranje u
15      *desno */
16     for (i = 0; i < sizeof(x) * 8; i++) {
17         najnizi_bit = x & 1;
18         x >>= 1;
19         /* Potiskivanjem trenutnog rezultata ka levom kraju svi prethodno
20          *postavljeni bitovi dobijaju veću poziciju. Novi bit se
21          *postavlja na najnizu poziciju */
22         rezultat <<= 1;
23         rezultat |= najnizi_bit;
24     }
25
26     /* Vraća se dobijena vrednost */
27     return rezultat;
28 }
29
30 int main()
31 {
32     int broj;
33
34     /* Učitava se broj sa ulaza */
```

```

35     scanf("%x", &broj);
37     /* Ispisuje se njegova binarna reprezentacija */
    stampaj_bitove(broj);
39
    /* Ispisuje se i binarna reprezentacija broja dobijenog pozivom
41     funkcije ogledalo */
    stampaj_bitove(ogledalo(broj));
43
44     return 0;
45 }

```

Rešenje 1.11

```

#include <stdio.h>
2
/* Funkcija vraca 1 ukoliko je u binarnoj reprezentaciji broja n broj
4     jedinica veci od broja nula. U suprotnom funkcija vraca 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 */
    maska = 1 << (sizeof(unsigned int) * 8 - 1);
16
    /* Cilj je proci kroz sve bitove broja x, zato se maska u svakoj
18     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         /* Provera da li se na poziciji koju odredjuje maska nalazi 0 ili
            1 i uveca se odgovarajuci brojac */
24         if (n & maska) {
            broj_jedinica++;
26         } else {
            broj_nula++;
28         }
30         /* Pomera se maska u desnu stranu */
        maska = maska >> 1;
32     }

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

```

1 Uvodni zadaci

```
38 int main()
40 {
    unsigned int n;

    /* Ucitavanje broja */
44     scanf("%u", &n);

    /* Ispisivanje rezultata */
46     printf("%d\n", broj_01(n));

48     return 0;
50 }
```

Rešenje 1.12

```
#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     int broj_parova;
8     unsigned int maska;

10     /* Vrednost promenljive koja predstavlja broj parova se
       inicijalizuje na 0 */
12     broj_parova = 0;

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

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

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

28     }

30     /* Vraca se dobijena vrednost */
32     return broj_parova;
34 }
```

```

int main()
36 {
    unsigned int x;
38
    /* Ucitavanje broja */
40    scanf("%u", &x);

    /* Ispisivanje rezultata */
42    printf("%d\n", broj_parova(x));

44    return 0;
46 }

```

Rešenje 1.14

```

#include <stdio.h>
2
/* Niska koja se formira je duzine (sizeof(unsigned int)*8)/4 +1 jer
4 su za svaku heksadekadnu cifru potrebne 4 binarne cifre i jedna
dodatna pozicija za terminirajucu nulu.
6 Prethodni izraz se 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 {
    int i;
14    unsigned int maska;
    int vrednost;
16
    /* Heksadekadni zapis broja 15 je 000...0001111 - odgovarajuca
18 maska za citanje 4 uzastopne cifre */
    maska = 15;
20
    /*****
22     Broj se posmatra od pozicije najmanje tezine ka poziciji najvece
    tezine. Na primer za broj cija je binarna reprezentacija
24     00000000001101000100001111010101
    u prvom koraku se citaju bitovi izdvojeni sa <...>:
26     0000000000110100010000111101<0101>
    u drugom koraku:
28     000000000011010001000011<1101>0101
    u trecem koraku:
30     00000000001101000100<0011>11010101 i tako redom...

32     Indeks i oznacava poziciju na koju se smesta vrednost.
    *****/
34    for (i = MAKS_DUZINA - 2; i >= 0; i--) {
        /* Vrednost izdvojene cifre */
36        vrednost = x & maska;

```

1 Uvodni zadaci

```
38      /* Ako je vrednost iz opsega od 0 do 9 odgovarajuci karakter se
39         dobija dodavanjem ASCII koda '0'. Ako je vrednost iz opsega od
40         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         dobijenu vrednost doda ASCII kod 'A' (time se dobija
43         odgovarajuće slovo 'A', 'B', ... 'F') */
44     if (vrednost < 10) {
45         s[i] = vrednost + '0';
46     } else {
47         s[i] = vrednost - 10 + 'A';
48     }
49
50     /* Primenljiva 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];
63
64     /* Ucitava se broj sa ulaza */
65     scanf("%u", &x);
66
67     /* Poziva se funkcija za prevodjenje */
68     prevod(x, s);
69
70     /* I stampa se dobijena niska */
71     printf("%s\n", s);
72
73     return 0;
74 }
```

Rešenje 1.17

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /*****
5   Resenje linearne slozenosti:
6   x^0 = 1
7   x^k = x * x^(k-1)
8   *****/
9  int stepen(int x, int k)
10 {
```



```

12     if (k == 0)
        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;
22     x^k = x * (x^2)^(k/2) , za neparno k
    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     /* Ako je stepen paran */
32     if ((k % 2) == 0)
        return stepen_2(x * x, k / 2);

34     /* Inace (ukoliko je stepen neparan) */
36     return x * stepen_2(x * x, k / 2);
    }

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

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

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

52     /* 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, "Greska: Neodgovarajuci redni broj funkcije.\n");
60         exit(EXIT_FAILURE);
    }

62

```

1 Uvodni zadaci

```
    exit(EXIT_SUCCESS);  
64 }
```

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 vraca 1 ako broj n ima paran broj cifara, inace vraca 0  
   */  
unsigned paran(unsigned n)  
14 {  
    if (n <= 9)  
16        return 0;  
    else  
18        return neparan(n / 10);  
}  
  
20 /* Funkcija vraca 1 ako broj n ima neparan broj cifara, inace vraca  
22   0 */  
unsigned neparan(unsigned n)  
24 {  
    if (n <= 9)  
26        return 1;  
    else  
28        return paran(n / 10);  
}  
  
30 int main()  
32 {  
    int n;  
  
34    /* Ucitava se ceo broj */  
36    scanf("%d", &n);  
  
38    /* Ispisuje se rezultat */  
    printf("Uneti broj ima %s paran broj cifara.\n",  
40          (paran(n) == 1 ? "" : "ne"));  
  
42    return 0;  
}
```

Rešenje 1.19

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Pomocna funkcija koja izracunava n! * result. Koristi repnu
5     rekurziju. Result je argument u kome se akumulira do tada
6     izracunatu vrednost faktoriijela. Kada dodje do izlaza iz rekurzije
7     iz rekurzije potrebno je da vratimo result. */
8  int faktoriijel_repna(int n, int result)
9  {
10     if (n == 0)
11         return result;
12
13     return faktoriijel_repna(n - 1, n * result);
14 }
15
16 /* U sledecim funkcijama je prikazan postupak oslobadjanja od
17     repne 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 navodjenjem
22     goto naredbe za vraćanje na pocetak tela funkcije. */
23 int faktoriijel_repna_v1(int n, int result)
24 {
25     pocetak:
26         if (n == 0)
27             return result;
28
29     result = n * result;
30     n = n - 1;
31     goto pocetak;
32 }
33
34 /* Pisanje bezuslovnih skokova (goto naredbi) nije dobra programerska
35     praksa i prethodna funkcija se koristi samo kao medjukorak. Sledi
36     iterativno resenje bez bezuslovnih skokova */
37 int faktoriijel_repna_v2(int n, int result)
38 {
39     while (n != 0) {
40         result = n * result;
41         n = n - 1;
42     }
43
44     return result;
45 }
46
47 /* Prilikom poziva prethodnih funkcija pored prvog argumenta celog
48     broja n, mora da se salje i 1 za vrednost drugog argumenta u kome
49     ce se akumulirati rezultat. Funkcija faktoriijel(n) je ovde radi
50     udobnosti korisnika, jer je sasvim prirodno da za faktoriijel
```

1 Uvodni zadaci

```
51     zahteva samo 1 parametar. Funkcija faktorijel izracunava n!, tako
53     sto odgovarajucoj gore navedenoj funkciji koja zaista racuna
55     faktorijel, salje ispravne argumente i vraca rezultat koju joj ta
57     funkcija vrati. Za testiranje, zameniti u telu funkcije faktorijel
59     poziv faktorijel_repna sa pozivom faktorijel_repna_v1, a zatim sa
61     pozivom funkcije faktorijel_repna_v2. */
63 int faktorijel(int n)
64 {
65     return faktorijel_repna(n, 1);
66 }
67
68 int main()
69 {
70     int n;
71
72     /* Ucitava se ceo broj */
73     printf("Unesite n (<= 12): ");
74     scanf("%d", &n);
75     if (n > 12) {
76         fprintf(stderr, "Greska: nedozvoljena vrednost za n!\n");
77         exit(EXIT_FAILURE);
78     }
79
80     /* Ispisuje se rezultat */
81     printf("%d! = %d\n", n, faktorijel(n));
82
83     exit(EXIT_SUCCESS);
84 }
```

Rešenje 1.21

```
1 #include <stdio.h>
2
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;
10
11     if (n == 0)
12         return 0;
13
14     for (i = 2; i <= n; i++) {
15         tmp = a * f_1 + b * f_0;
16         f_0 = f_1;
17         f_1 = tmp;
18     }
19
20     return f_1;
21 }
```

```

21 }
23 /* b) Funkcija racuna n-ti element u nizu F - rekurzivna verzija */
24 int f_rekurzivna(int n, int a, int b)
25 {
26     /* Izlaz iz rekurzije */
27     if (n == 0 || n == 1)
28         return n;
29
30     /* Rekurzivni pozivi */
31     return a * f_rekurzivna(n - 1, a, b) +
32         b * f_rekurzivna(n - 2, a, b);
33 }
34
35 /* NAPOMENA: U slucaju da se rekurzijom problem svodi na vise manjih
36 podproblema koji se mogu preklapati, postoji opasnost da se
37 pojedini podproblemi manjih dimenzija resavaju veci broj puta.
38 Npr.  $F(20) = a \cdot F(19) + b \cdot F(18)$ , a  $F(19) = a \cdot F(18) + b \cdot F(17)$ , tj.
39 problem  $F(18)$  se resava dva puta! Problemi manjih
40 dimenzija ce se resavati jos veci broj puta. Resenje za ovaj
41 problem je kombinacija rekurzije sa dinamicnim programiranjem.
42 Podproblemi se resavaju samo jednom, a njihova resenja se pamte u
43 memoriji (obicno u nizovima ili matricama), odakle se koriste ako
44 tokom resavanja ponovo budu potrebni.
45
46 U narednoj funkciji vec izracunati clanovi niza se cuvaju u
47 statickom nizu celih brojeva, jer se staticki niz ne smesta
48 na stek, kao sto je to slucaj sa lokalnim promenljivama, vec na
49 segment podataka, odakle je dostupan svim pozivima
50 rekurzivne funkcije. */
51
52 /* c) Funkcija racuna n-ti broj niza F - napredna rekurzivna
53 verzija */
54 int f_napredna(int n, int a, int b)
55 {
56     /* Niz koji cuva resenja podproblema. Kompajler inicijalizuje
57 staticke promenljive na podrazumevane vrednosti. Stoga, elemente
58 celobrojnog niza inicijalizuje na 0 */
59     static int f[20];
60
61     /* Ako je podproblem vec ranije resen, koristi se resenje koje je
62 vec izracunato i */
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);

```

1 Uvodni zadaci

```
73 }
75 int main()
76 {
77     int n, a, b, ind;
79     /* Unosi se redni broj funkcije koja ce se primeniti */
80     printf("Unesite redni broj funkcije:\n");
81     printf("1 - iterativna\n");
82     printf("2 - rekurzivna\n");
83     printf("3 - rekurzivna napredna\n");
84     scanf("%d", &ind);
85
86     /* Ucitavaju se koeficijenti a i b */
87     printf("Unesite koeficijente:\n");
88     scanf("%d%d", &a, &b);
89
90     /* Ucitava se broj n */
91     printf("Unesite koji clan niza se racuna:\n");
92     scanf("%d", &n);
93
94     /* Na osnovu vrednosti promenljive ind ispisuje se rezultat poziva
95     funkcije f_iterativna, f_rekurzivna ili f_napredna */
96     if (ind == 0)
97         printf("F(%d) = %d\n", n, f_iterativna(n, a, b));
98     else if (ind == 1)
99         printf("F(%d) = %d\n", n, f_rekurzivna(n, a, b));
100     else
101         printf("F(%d) = %d\n", n, f_napredna(n, a, b));
103     return 0;
104 }
```

Rešenje 1.22

```
1 #include <stdio.h>
2
3 /* Funkcija odredjuje zbir cifara zadatog broja x */
4 int zbir_cifara(unsigned int x)
5 {
6     /* Izlazak iz rekurzije: ako je broj jednocifren */
7     if (x < 10)
8         return x;
9
10    /* Zbir cifara broja jednak je zbiru svih njegovih cifara osim
11    poslednje cifre + poslednja cifra tog broja */
12    return zbir_cifara(x / 10) + x % 10;
13 }
14
15 int main()
16 {
```

```
18 unsigned int x;
20 /* Ucitava se ceo broj */
22 scanf("%u", &x);
24 /* Ispisivanje zbira cifara ucitanog broja */
26 printf("%d\n", zbir_cifara(x));
return 0;
}
```

Rešenje 1.23

```
#include <stdio.h>
#include <stdlib.h>
#define MAKS_DIM 100

/* Ako je n<=0, onda je suma niza jednaka nuli. Ako je n>0, onda je
suma niza jednaka sumi prvih n-1 elementa uvecenoj za poslednji
element niza. */
int suma_niza_1(int *a, int n)
{
    if (n <= 0)
        return 0;

    return suma_niza_1(a, n - 1) + a[n - 1];
}

/* Funkcija napisana na drugi nacin: Ako je n<=0, onda je suma niza
jednaka nuli. Ako je n>0, suma niza je jednaka zbiru prvog
elementa niza i sume preostalih n-1 elementa. */
int suma_niza_2(int *a, int n)
{
    if (n <= 0)
        return 0;

    return a[0] + suma_niza_2(a + 1, n - 1);
}

int main()
{
    int a[MAKS_DIM];
    int n, i = 0, ind;

    /* Ucitava se redni broj funkcije */
    printf("Unesite redni broj funkcije (1 ili 2):\n");
    scanf("%d", &ind);

    /* Ucitava se broj elemenata niza */
    printf("Unesite dimenziju niza:\n");
    scanf("%d", &n);
}
```

```
40  /* Ucitava se n elemenata niza. */
    printf("Unesite elemente niza:\n");
42  for (i = 0; i < n; i++)
        scanf("%d", &a[i]);

44

    /* Na osnovu vrednosti promenljive ind ispisuje se rezultat poziva
    funkcije suma_niza_1, odnosno suma_niza_2 */
46  if (ind == 1)
        printf("Suma elemenata je %d\n", suma_niza_1(a, n));
48  else if (ind == 2)
        printf("Suma elemenata je %d\n", suma_niza_2(a, n));
50  else{
52      fprintf(stderr, "Greska: Neodgovarajuci redni broj funkcije.\n");
        exit(EXIT_FAILURE);
54  }

56  exit(EXIT_SUCCESS);
}
```

Rešenje 1.24

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

4 /* Rekurzivna funkcija koja odredjuje maksimum celobrojnog niza niz
   dimenzije n */
6 int maksimum_niza(int niz[], int n)
{
8     /* Izlazak iz rekurzije: ako je niz dimenzije jedan, najveći je
       ujedno i jedini element niza */
10    if (n == 1)
        return niz[0];

12

    /* Resavanje problema manje dimenzije */
14    int max = maksimum_niza(niz, n - 1);

16    /* Na osnovu poznatog resenja problema dimenzije n-1, resava se
       problem dimenzije n */
18    return niz[n - 1] > max ? niz[n - 1] : max;
}

20
22 int main()
{
    int brojevi[MAKS_DIM];
24    int n;

26    /* Sve dok se ne stigne do kraja ulaza, brojeve 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. */
28
```



```

30  int i = 0;
    while (scanf("%d", &brojevi[i]) != EOF) {
32      i++;
      if (i == MAKS_DIM)
34          break;
    }
36  n = i;

38  /* Stampa se maksimum unetog niza brojeva */
    printf("%d\n", maksimum_niza(brojevi, n));
40
    return 0;
42 }

```

Rešenje 1.25

```

1  #include <stdio.h>
   #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
      dimenzije n primenom definicije skalarnog proizvoda
13   a*b = a[0]*b[0] + a[1]*b[1] +...+ a[n-2]*a[n-2] + a[n-1]*a[n-1]
      Dakle, skalarni proizvod dva vektora duzine n se dobija kada se
15   na skalarni proizvod dva vektora duzine n-1 koji se dobiju od
      polazna dva vektora otklanjanjem poslednjih elemenata, doda
17   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         printf("Dimenzija mora biti prirodan broj <= %d!\n", MAKS_DIM);
            exit(EXIT_FAILURE);
35     }

```

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

41
    printf("Unesite elemente drugog niza:");
43  for (i = 0; i < n; i++)
        scanf("%d", &b[i]);

45
    /* Ispisivanje rezultata skalarnog proizvoda dva ucitana niza */
47  printf("Skalarni proizvod je %d\n", skalarno(a, b, n));

49  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 */
5  int br_pojave(int x, int a[], int n)
6  {
7      /* Izlazak iz rekurziije: 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()
23 {
24     int x, a[MAKS_DIM];
25     int n, i = 0;

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

30     /* Sve dok se ne stigne do kraja ulaza, ucitavaju se brojevi 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. */
32 }
```

```

34 printf("Unesite elemente niza:");
   i = 0;
36 while (scanf("%d", &a[i]) != EOF) {
       i++;
38     if (i == MAKS_DIM)
           break;
40 }
   n = i;
42
   /* Ispisuje se broj pojavljivanja */
44 printf("Broj pojavljivanja je %d\n", br_pojave(x, a, n));
46
   return 0;
}

```

Rešenje 1.27

```

1  #include <stdio.h>
   #define MAKS_DIM 256
3
   /* Funkcija koja proverava da li su tri zadata broja uzastopni
   clanovi niza */
5  int tri_uzastopna_clana(int x, int y, int z, int a[], int n)
7  {
       /* Ako niz ima manje od tri elementa izlazi se iz rekurzije i vraca
       se 0 jer nije ispunjeno da su x, y i z uzastopni clanovi niza */
9      if (n < 3)
           return 0;
11
13     /* Da bi bilo ispunjeno da su x, y i z uzastopni clanovi niza a
       dovoljno je da su oni poslednja tri clana niza ili da se oni
       rekuzivno tri uzastopna clana niza a bez poslednjeg elementa */
15     return ((a[n - 3] == x) && (a[n - 2] == y) && (a[n - 1] == z))
           || tri_uzastopna_clana(x, y, z, a, n - 1);
17 }
19
21 int main()
   {
       int x, y, z, a[MAKS_DIM];
23     int n;

25     /* Ucitavaju se tri cela broja za koje se ispituje da li su
       uzastopni clanovi niza */
27     printf("Unesite tri cela broja:");
       scanf("%d%d%d", &x, &y, &z);
29
       /* Sve dok se ne stigne do kraja ulaza, brojeve 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. */
31     printf("Unesite elemente niza:");
33

```

1 Uvodni zadaci

```
35  int i = 0;
36  while (scanf("%d", &a[i]) != EOF) {
37      i++;
38      if (i == MAKS_DIM)
39          break;
40  }
41  n = i;
42
43  /* Na osnovu rezultata poziva funkcije tri_uzastopna_clana ispisuje
44     se odgovarajuca poruka */
45  if (tri_uzastopna_clana(x, y, z, a, n))
46      printf("Uneti brojevi jesu uzastopni clanovi niza.\n");
47  else
48      printf("Uneti brojevi nisu uzastopni clanovi niza.\n");
49
50  return 0;
51 }
```

Rešenje 1.28

```
#include <stdio.h>
2
/* Funkcija koja broji bitove postavljene na 1. */
4 int prebroj(int x)
{
6     /* Izlaz iz rekurzije */
7     if (x == 0)
8         return 0;
9
10    /* Ukoliko vrednost promenljive x nije 0, neki od bitova broja x je
11       postavljen na 1. Koriscenjem odgovarajuce maske proverava se
12       vrednost bita na poziciji najvece tezine i na osnovu toga se
13       razlikuju dva slucaja. Ukoliko je na toj poziciji nula, onda je
14       broj jedinica u zapisu x isti kao broj jedinica u zapisu broja
15       x<<1, jer se pomeranjem u levo sa desne stane dopisuju 0. Ako je
16       na poziciji najvece tezine jedinica, rezultat dobijen
17       rekurzivnim pozivom funkcije za x<<1 treba uvecati za jedan.
18       Za rekurzivni poziv se salje vrednost koja se dobija kada se x
19       pomeri u levo. Napomena: argument funkcije x je oznacen ceo
20       broj, usled cega se ne koristi pomeranje udesno, jer funkciji
21       moze biti prosledjen i negativan broj. Iz tog razloga,
22       odlucujemo se da proveramo najvisi, umesto najnizeg bita */
23    if (x & (1 << (sizeof(x) * 8 - 1)))
24        return 1 + prebroj(x << 1);
25    else
26        return prebroj(x << 1);
27
28    /* *****
29       Krace zapisano
30       return ((x & (1<<(sizeof(x)*8-1))) ? 1 : 0) + prebroj(x<<1);
31       ***** */
32 }
```

```
32 int main()
33 {
34     int x;
35
36     /* Ucitava se ceo broj */
37     scanf("%x", &x);
38
39     /* Ispisivanje rezultata */
40     printf("%d\n", prebroj(x));
41
42     return 0;
43 }
44
```

Rešenje 1.30

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

Rešenje 1.31

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

Rešenje 1.32

```
1  #include <stdio.h>
2  #include <string.h>
3
4  /* Niska može imati najviše 31 karaktera + 1 za terminalnu nulu */
5  #define MAKS_DIM 32
6
7  /* Funkcija ispituje da li je zadata niska dužine n palindrom */
8  int palindrom(char s[], int n)
9  {
10     /* Izlaz iz rekurzije - trivijalno, niska dužine 0 ili 1 je
11        palindrom */
12     if ((n == 1) || (n == 0))
```

```

    return 1;
14
    /* Da bi niska bila palindrom potrebno je da se poklapaju prvi i
16    poslednji karakter i da je palindrom niska koja nastaje kada se
    polaznoj nisci otklone prvi i poslednji karakter */
18    return (s[n - 1] == s[0]) && palindrom(s + 1, n - 2);
}
20
int main()
22 {
    char s[MAKS_DIM];
24    int n;

    /* Ucitavanje niske sa standardnog ulaza */
26    scanf("%s", s);

    /* Odredjuje se duzina niske */
28    n = strlen(s);

    /* Ispisivanje da li je niska palindrom ili nije */
30    if (palindrom(s, n))
        printf("da\n");
32    else
        printf("ne\n");
34
36    return 0;
38 }

```

Rešenje 1.33

```

#include <stdio.h>
2 #include <stdlib.h>
#define MAKS_DUZINA_PERMUTACIJE 15
4
/* Funkcija koja ispisuje elemente niza a duzine n */
6 void ispisi_niz(int a[], int n)
{
8     int i;

    for (i = 1; i <= n; i++)
10         printf("%d ", a[i]);
    printf("\n");
12 }
14
/* Funkcija koja vraca 1 ako se broj x nalazi u nizu a duzine n, a
16    inace 0 */
int koriscen(int a[], int n, int x)
18 {
    int i;
20
    /* Obilaze se svi elementi niza */

```

```
22     for (i = 1; i <= n; i++)
23         /* Ukoliko se naidje na trazenu vrednost, pretraga se prekida */
24         if (a[i] == x)
25             return 1;
26
27     /* Zakljucuje se da broj nije pronadjen */
28     return 0;
29 }
30
31 /* Funkcija koja ispisuje sve permutacije od skupa {1,2,...,n} dobija
32    kao argument niz a[] u koji se smesta permutacija, broj m oznacava
33    da se u okviru tog poziva funkcije na m-tu poziciju u permutaciji
34    smesta jedan od preostalih celih brojeva, n je velicina skupa koji
35    se permutuje. Funkciju se inicijalno poziva sa argumentom m = 1,
36    jer formiranje permutacije pocinje od pozicije broj 1. Stoga, a[0]
37    se ne koristi. */
38 void permutacija(int a[], int m, int n)
39 {
40     int i;
41
42     /* Izlaz iz rekurzije: Ako je pozicija na koju treba smestiti broj
43        premasila velicinu skupa, onda se svi brojevi vec nalaze u
44        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 permutaciji).
52        Zatim, rekurzivno se pronalaze one permutacije koje odgovaraju
53        ovako postavljenom pocetku permutacije. Kada se to zavrshi, vrsi
54        se provera da li postoji jos neki broj koji moze da se stavi na
55        m-to mesto u nizu (to se radi u petlji). Ako ne postoji,
56        funkcija zavrшава sa radom. Ukoliko takav broj postoji, onda se
57        ponovo poziva rekurzivno pronalazenje odgovarajucih permutacija,
58        ali sada sa drugacije postavljenim prefiksom. */
59     for (i = 1; i <= n; i++) {
60         /* Ako se broj i nije do sada pojavio u permutaciji od 1 do m-1
61            pozicije, onda se on postavlja na poziciju m i poziva se
62            ponovo funkcija da dopuni ostatak permutacije posle upisivanja
63            i na poziciju m. Inace, nastavlja se dalje, trazeci broj koji
64            se nije pojavio do sada u permutaciji. */
65         if (!koriscen(a, m - 1, i)) {
66             a[m] = i;
67             permutacija(a, m + 1, n);
68         }
69     }
70 }
71
72 int main(void)
73 {
```



```

74  int n;
75  int a[MAKS_DUZINA_PERMUTACIJE+1];
76
77  /* Ucitava se broja n i proverava se da li je u odgovarajucem opsegu
78  */
79  scanf("%d", &n);
80  if (n < 0 || n > MAKS_DUZINA_PERMUTACIJE) {
81      fprintf(stderr,
82          "Duzina permutacije mora biti broj iz intervala [0, %d]!\n",
83          MAKS_DUZINA_PERMUTACIJE);
84      exit(EXIT_FAILURE);
85  }
86
87  /* Ispisuju se permutacije duzine n */
88  permutacija(a, 1, n);
89
90  exit(EXIT_SUCCESS);
91  }

```

Rešenje 1.34

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* Rekurzivna funkcija za racunanje binomnog koeficijenta */
5  int binomni_koeficijent(int n, int k)
6  {
7      /* Ukoliko je k=0 ili k=n, onda je binomni koeficijent 0.
8       Ukoliko je k strogo izmedju 0 i n, onda se koristi formula
9       bk(n,k) = bk(n-1,k-1) + bk(n-1,k)
10      koja se moze izvesti iz definicije binomnog koeficijenata */
11      return (0 < k && k < n) ?
12          binomni_koeficijent(n - 1, k - 1) + binomni_koeficijent(n - 1,
13          k) : 1;
14  }
15
16  /******
17   Iterativno izracunavanje datog binomnog koeficijenta
18   *****/
19
20  int binomni_koeficijent (int n, int k) {
21      int i, j, b;
22
23      for (b=i=1, j=n; i<=k; b =b * j-- / i++);
24
25      return b;
26  }
27
28  Iterativno resenje je efikasnije i preporucuje se. Rekurzivno
29  resenje je navedeno u cilju demonstracije rekurzivnih tehnika.
30  *****/

```

```
30  /* Svaki element n-te hipotenuze (osim ivicnih jedinica) dobija kao
32     zbir 2 elementa iz n-1 hipotenuze. Ukljucujuci i pomenute dve
34     ivicne jedinice suma elemenata n-te hipotenuze je tacno 2 puta
36     veca od sume elemenata prethodne hipotenuze. */
37  int suma_elementa_hipotenuze(int n)
38  {
39      return n > 0 ? 2 * suma_elementa_hipotenuze(n - 1) : 1;
40  }
41
42  int main()
43  {
44      int n, k, i, d, r;
45
46      /* Ucitavaju se brojevi d i r */
47      scanf("%d %d", &d, &r);
48
49      /* Ispisivanje Paskalovog trougla */
50      putchar('\n');
51      for (n = 0; n <= d; n++) {
52          for (i = 0; i < d - n; i++)
53              printf(" ");
54          for (k = 0; k <= n; k++)
55              printf("%4d", binomni_koeficijent(n, k));
56          putchar('\n');
57      }
58
59      /* Provera da li je r nenegativan */
60      if (r < 0) {
61          fprintf(stderr,
62              "Redni broj hipotenuze mora biti veci ili jednak od 0!\n"
63          );
64          exit(EXIT_FAILURE);
65      }
66
67      /* Ispisivanje sume elemenata hipotenuze */
68      printf("%d\n", suma_elementa_hipotenuze(r));
69
70      exit(EXIT_SUCCESS);
71  }
```