ELEKTROTEHNIČKI FAKULTET UNIVERZITETA U BEOGRADU

PROGRAMIRANJE 2
MATERIJAL ZA VEŽBE NA TABLI I PRIPREMU ISPITA

verzija: 26.06.2008.

Detaljna objašnjenja vezana za bilo koji deo gradiva vezanog za teoriju dostupna su u beleškama sa predavanja, kao i u knjizi koja opisuje jezik C dovoljno precizno:

- Laslo Kraus – Programski jezik C.

Više zadataka se može pronaći u odgovarajućoj zbirci prof. Krausa:

- Laslo Kraus, Zbirka zadataka iz programskog jezika C.

Sve ove knjige su dostupne u skriptarnici ili u biblioteci fakulteta.

Zadaci koji su tipa ispitnih pitanja su priloženi u celosti, uključujući i obrazloženje rešenja tamo gde je to potrebno.

U materijal je uključeno i nekoliko ispitnih zadataka, koji su rešeni u potpunosti i komentarisani do odgovarajućeg stepena opširnosti.

Materijal će biti stalno dorađivan. Nove verzije će redovno biti dodavane na Internet stranicu predmeta:

http://rti.etf.rs/ir1p2

Sugestije, primedbe i uočene greške poslati putem elektronske pošte na adresu **`oo1p2@etf.rs`**.

Ove materijale treba koristiti isključivo kao podsetnik a nipošto kao jedini izvor znanja.

Nemojte NIKAD učiti programski kod napamet.

Svako ponavljanje bez razumevanja programskog koda je štetno i na ispitu će biti kažnjavano oduzimanjem poena.

SADRŽAJ

PREDSTAVLJANJE REALNIH BROJEVA

Standard za aritmetiku realnih brojeva

- **Raznolikost** predstavljanja realnih brojeva u računarima.
- Problem **portabilnosti** numeričkog softvera.
- **Standard** za *binarnu aritmetiku realnih brojeva u pokretnom zarezu*:
  ANSI/IEEE Std 754-1985.
  (IEEE – The Institute of Electrical and Electronics Engeneers
  ANSI – American National Standard Institute)

Standard specificira:

1. Osnovni i prošireni format realnih brojeva
2. Operacije +, –, *, /, ostatka (rem), kvadratnog korena($\sqrt{\ }$) i komparaciju realnih brojeva
3. Konverzije: celi brojevi ⇔ realni brojevi
4. Konverzije: realni brojevi ⇔ realni brojevi (različiti formati brojeva)
5. Konverzije: osnovni format realnih brojeva ⇔ decimalni niz znakova
6. Rukovanje kodovima grešaka

Osnovna osobina standarda: niz bitova, generisan kao rezultat aritmetičke operacije, može nositi dvojaku informaciju:

1. ispravno obavljena operacija ⇒ niz bitova je rezultat operacije
2. detektovana je neka greška ⇒ niz bitova je binarno kodirani signal greške
   (tzv. NaN = **N**ot **a** **N**umber)

Postoje dva tipa NaN–a:

1. signalni NaN (signal NaN) – signalizira neispravnu operaciju kad god se pojavi u ulozi operanda
2. tihi (mirni) NaN (quiet NaN) – za skoro sve aritmetičke operacije ne signalizira neispravnost

U skladu sa IEEE standardom, realni brojevi se u memorijske reči smeštaju na sledeći način:

memorijska reč	s	eeeeeee	mmmmmmmmmmm
broj bita u datom polju	1	k	p
smisao polja	znak	eksponent	mantisa

Ukupan broj bita: n = 1 + k + p

Vrednost tako predstavljenog realnog broja se računa na sledeći način:

$$R = (-1)^{s} \cdot 2^{E} \cdot M$$

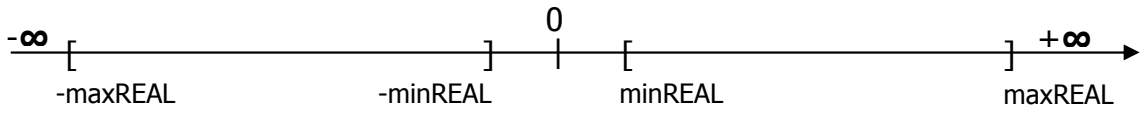
gde vrednosti **s**, **E** i **M** imaju sledeća tumačenja**:**

- **s** predstavlja znak datog broja (s=0 ⇒ broj je pozitivan; s=1 ⇒ broj je negativan)
- **E** predstavlja celobrojni eksponent, čija se vrednost računa u kôdu sa viškom, kao **E = e-v**, gde je
  - **e** vrednost neoznačenog celog broja čija se vrednost dobija na osnovu bita eee...e, polja *eksponent*
  - **v** se naziva *višak*, a predstavlja celobrojnu vrednost koja se računa: $v=2^{k-1}-1$, gde je k broj bita polja *eksponent*
- **M** predstavlja vrednost mantise sa skrivenim bitom (videti objašnjenje u nastavku)

U zavisnosti vrednosti bita polja *eksponent*, postoje četiri različita načina tumačenja sadržaja memorijske reči u kojoj je smešten realan broj.

1. Kada biti polja *eksponent* nisu sve 0 ili sve 1 (**e≠0000…0** i **e≠1111…1**), vrednost eksponenta **E** se nalazi u opsegu [**Emin**, **Emax**] i tada memorijska reč sadrži uobičajen realni broj sa **normalizovanom** mantisom. Vrednost mantise **M** se računa dodavanjem skrivenog bita vrednosti **1** i decimalnog zareza ispred niza bita polja *mantisa*: **M**=**1**.mmmmm...m. Skriveni bit se podrazumeva pa zbog toga nije potrebno posebno ga skladištiti u memorijskoj reči.
2. Kada su biti polja *eksponent* sve 0 (**e=0000…0**), ali biti polja *mantisa* nisu sve 0 (**m≠0000…0**), tada memorijska reč sadrži realan broj sa **nenormalizovanom** mantisom. Vrednost eksponenta **E** se u ovom specijalnom slučaju računa kao **E=e-v+1**. Vrednost mantise **M** se računa dodavanjem skrivenog bita vrednosti **0** i decimalnog zareza ispred niza bita polja *mantisa*: **M**=**0**.mmmm...m. Kao i u prethodnom slučaju, ovaj skriveni bit se podrazumeva. Smisao realnih brojeva sa nenormalizovanom mantisom je proširenje opsega za realne brojeve veoma male apsolutne vrednosti.
3. Kada su biti polja *eksponent* i *mantisa* sve 0 (**e=0000…0**, **m=0000…0**), tada memorijska reč sadrži realan broj čija je vrednost 0. Treba primetiti da zbog znaka (bit s), standard dozvoljava postojanje pozitivne i negativne nule (+0 i -0) ali između njih ne pravi razliku (tj. imaju istu vrednost).
4. Kada su biti polja *eksponent* sve 1 (**e=1111…1**), vrednost eksponenta **E** je veća od **Emax** i razlikuju se dva slučaja:
   a. vrednosti bita polja *mantisa* su sve 0 (**m=0000…0**) : realni broj ima vrednost ±**∞** (u zavisnosti od vrednosti bita s)
   b. vrednosti bita polja *mantisa* nisu sve 0 (**m≠0000…0**) : radi se o NaN-u, koji označava matematički nedefinisanu vrednost

Pozitivni realni brojevi, manji od najmanjeg realnog broja (minREALl) koji je moguće predstaviti sa zadatim brojem bita se zaokružuju na vrednost 0 (tzv. *potkoračenje*, eng. underflow), dok se pozitivni realni brojevi veći od najvećeg realnog broja (maxREAL) zaokružuju na vrednost +**∞** (tzv. *prekoračenje*, eng. overflow). Slično važi za negativne realne brojeve.



Standardni formati realnih brojeva

IEEE Standard uvodi 4 formata realnih brojeva (format se odnosi na broj bitova pojedinih polja):
(1) jednostruki (2) jednostruki prošireni (3) dvostruki (4) dvostruki prošireni

FORMAT	w	k	p	v	E_{max}	E_{min}
jednostruki	32	8	23	+127	+127	-126
jednostruki prošireni	≥ 43	≥ 11	≥ 32	nedef.	≥ +1023	≤ -1022
dvostruki	64	11	52	+1023	+1023	-1022
dvostruki prošireni	≥ 79	≥ 15	≥ 64	nedef.	≥ +16383	≤ -16382

Jednostruki format

1	k = 8	p = 23
s	e e e e e e e e	m m m m m m m m m m m m m m m m m m m m m m m
31	30 23	22 0

Opseg brojeva:

1. Najmanji nenormalizovani: e=0 ⇒ $R = (-1)^s \cdot 2^{-126} \cdot (0.00000\ldots\ldots\ldots1)$
2. Najveći nenormalizovani: e=0 ⇒ $R = (-1)^s \cdot 2^{-126} \cdot (0.11111\ldots\ldots\ldots1)$
3. Najmanji normalizovani: e=1 ⇒ $R = (-1)^s \cdot 2^{-126} \cdot (1.00000\ldots\ldots\ldots0)$
4. Najveći realan broj: e=254⇒ $R = (-1)^s \cdot 2^{127} \cdot (1.11111\ldots\ldots\ldots1)$

$minREAL = (-1)^s \cdot 2^{-126} \cdot 2^{-23} = (-1)^s \cdot 2^{-149} \approx (-1)^s \cdot 1.4 \cdot 10^{-45}$

$maxREAL = (-1)^s\ 2^{127} \cdot (1.111...1) \approx (-1)^s \cdot 2^{128} = (-1)^s \cdot 3.4 \cdot 10^{38}$

Zaokruživanje

Zbog ograničenog broja bita polja *mantisa*, najčešće nije moguće prezicno smeštanje realnog broja u memorijsku reč, već se pristupa zaokruživanju. Podrazumeva se da je posmatrani broj, pre smeštanja u memorijsku reč, beskonačno precizan, pa se zaokruživanje vrši sa ciljem da se uklopi u odredišni format. Skoro sve operacije nad realnim brojevima daju najpre "beskonačno precizan" rezultat koji se potom zaokružuje prema odredišnom formatu.

1. Osnovni način zaokruživanja: prema najbližoj vrednosti.
2. Ako su dve zaokružene vrednosti podjednako udaljene od "beskonačno precizne" vrednosti – bira se ona vrednost kod koje je bit najmanjeg značaja 0.
3. Ako je apsolutna vrednost "beskonačno precizne" vrednosti:

$|R| \geq 2^{Emax} \cdot (2 - 2^{-p})$, vrši se zaokruživanje na $(-1)^s \cdot (\infty)$

p i E_{max} se određuju iz odredišnog formata.

U praksi, ovo znači sledeće (desno od vertikalne crte se nalaze biti koji se odbacuju):

	1	2	3	4
binarna predstava broja	x.xx | 0zzz...zzz	x.xx | 1yyy...yyy (makar jedan y je 1)	x.x0 | 100...00	x.x1 | 100...00
vrednost na koju se zaokružuje	x.xx	x.xx + 0.01	x.x0	x.x1 + 0.01

Ako se pažljivije pogleda, ovakav način zaokruživanja odgovara uobičajenom načinu zaokruživanja realnih brojeva.

Gubitak tačnosti ima za posledicu da kod računarskog sabiranja realnih brojeva ne važi uvek zakon asocijativnosti.

Primer1: decimalni računar sa 4 značajne cifre:
R=123.4, r=0.049 ⇒ R+r=123.449 ≈ 123.4
Z1 = ((R+r)+r)+r = 123.4
Z2 = ((r+r)+r)+R = 123.4+0.147 = 123.547 ≈ 123.5 ≠ Z1

Opšte pravilo: početi sumiranje od brojeva manje apsolutne vrednosti.

Zadatak Z12

Za smeštanje realnih brojeva koriste se 8 bitova od kojih 3 bita predstavljaju eksponent u kodu sa viškom. Mantisa se smešta sa skrivenim bitom. Za brojeve veće od nule odrediti najmanju i najveću vrednost koja se može predstaviti na gornji način.

Zadatak IZ3

Napomena: ovaj zadatak ilustruje razliku između savremenog IEEE standarda za reprezentaciju realnih brojeva i standarda VAX koji mu je prethodio.

Format predstave realnih brojeva je **seeeemmmmm** – s je bit za znak broja, eeee su bitovi za eksponent broja (kôd sa viškom), a mmmmm su bitovi normalizovane mantise (sa skrivenim bitom). U računaru A – predstava realnih brojeva je sa viškom 8, a u računaru B – sa viškom 7. U računaru A – mantisa je $0.5 \le M_A < 1$, a u računaru B – mantisa je $1 \le M_B < 2$. Izgled jednog broja u računaru B, u skladu sa opisanim formatom, je $3DF_{16}$. Kako je predstavljen realan broj iste vrednosti u računaru A?

A) $3DF_{16}$
B) $3FF_{16}$
C) $3FE_{16}$

Rešenje

Prvi način: Moguće je reći, da je predstava realnih brojeva u računaru B u skladu sa duhom IEEE standarda, imajući u vidu da je višak $7 = 2^{4-1} - 1$, a mantisa je oblika $M_B = 1.mmmmm$.
Zbog toga, broj $3DF_{16} = 1|1110|11111_2$ predstavlja maxREAL, kad je računar B u pitanju.

Moguće je zaključiti, da je predstava realnih brojeva u računaru A – u skladu sa starom (VAX) predstavom brojeva ($8 = 2^{4-1}$, $M_A = 0.1mmmmm$).

Kako je $maxREAL_{IEEE} = 2 \cdot maxREAL_{VAX}$, može se zaključiti da ovaj broj nije moguće predstaviti u računaru A, jer je $maxREAL_{VAX}$ dva puta manji od posmatranog broja.

Drugi način:
B: $3DF_{16} = 1|1110|11111_2$ $E_B = eeee_B - 7 = 7$ $M_B = 1.mmmmm_B = 1.11111_2$

$X_B = -1.11111_2 \cdot 2^7 = -11111100_2$ $X_A = X_B = -0.111111_2 \cdot 2^8$

$M_A = 0.1m_A$ $mA = 11111_2$

$E_A = e_A + 8 = 16_{10} = 10000_2$ a to nije moguće predstaviti u polju e_A koje je širine 4 bita.

Odgovor: N

Zadatak IZ4

Format predstave realnih brojeva pomoću reči širine 10 bita je: bit najveće težine za znak broja, četiri bita za eksponent u kôdu sa viškom 7, i pet bitova najmanje težine za normalizovanu mantisu sa skrivenim bitom ($1 \le M < 2$). Rezultat sabiranja brojeva čiji je izgled $A=1606_8$ i $B=0515_8$ je:

A) 1633_8
B) 1566_8
C) 1766_8

Rešenje

A: 1|1100|00110
$S_A=1 \Rightarrow A<0$
$E_A=12-7=5$
$M_A=1.00110$

B: 0|1010|01101
$S_B=0 \Rightarrow B>0$
$E_B=10-7=3$
$M_B=1.01101$

$E_A > E_B \Rightarrow B = M_B \cdot 2^E B = M_B \cdot 2^{E B - E A} \cdot 2^E A = M_B \cdot 2^{-2} \cdot 2^E A = M_B' \cdot 2^E A;\ M_B' = M_B / 2^2$

$M_B' = 0.01011|01$ $\quad |M_A| > |M_B'| \Rightarrow A + B = -(M_A - M_B') \cdot 2^{EA} = -M_{A+B} \cdot 2^{EA}$

$$\begin{array}{rcl} M_A & = & 1.00110 \\ -M_B' & = & 0.01011 \\ \hline M_{A+B} & = & 0.11011 \end{array}$$

ovde se mora izvršiti normalizacija.

$M_{A+B} = 1.10110 \cdot 2^{-1}$

$A + B = -M_{A+B} \cdot 2^E A = -1.10110 \cdot 2^{-1} \cdot 2^E A = -1.10110 \cdot 2^E A^{-1} = -M_{A+B}' \cdot 2^E A + B'$

$S_{A+B} = 1$ $\quad M_{A+B}' = 1.10110$ $\quad E_{A+B}' = E_A - 1 = 4$ $\quad e_{A+B} = 4 + 7 = 11$

s = 1, eeee = 1011, mmmmm = 10110

(A + B) : 1|1011|10110 = 1|101|110|110 $=1566_8$

Odgovor: B

Zadatak IZ5

U nekom računaru, celi brojevi su predstavljeni u drugom komplementu pomoću reči širine 10 bita, a za predstavljanje realnih brojeva je takođe predviđeno 10 bita, i to tako da bit najveće težine određuje znak broja, sledeća 4 bita su za eksponent broja u kôdu sa viškom 7, a preostalih 5 za normalizovanu mantisu sa skrivenim bitom ($1 \leq M < 2$). Ako je izgled realnog broja na lokaciji X: 397_{16}, a izgled celog broja na lokaciji J: 310_{16}, koji će biti izgled lokacije realne promenljive Y nakon izvršene operacije Y=X+J? Računanje obavljati upotrebljavajući realne brojeve.

A) $3FA_{16}$
B) $3E5_{16}$
C) $3BA_{16}$

Rešenje

X: 1|1100|10111

$S_X = 1, X<0$
$M_X = 1.10111$
$E_X = 12 - 7 = 5$

J: 1100010000
$-J = 0011110000 = 1.111 \cdot 2^7$
$S_J = 1, J<0$
$M_J = 1.11100$
$E_J = 7$

$E_J > E_X$
$X = - M_X \cdot 2^{-2} \cdot 2^7$
$M_X' = 0.01101|11 \approx 0.01110$

$Y = - (M_J + M_X') \cdot 2^{Ej}$

$M_J = 1.11100$
$+M_X' = 0.01110$
———————
$M_Y = 10.01010 = 1.00101 \cdot 2^1$

$Y = - M_Y' \cdot 2^{E}Y^{+1} = -1.00101 \cdot 2^8$

$s = 1, e = 8 + 7 = 15_{10} = 1111_2$, eeee = 1111, mmmmm = 00101

Y: 1|1111|00101 = $3E5_{16}$

Odgovor: B

Zadatak IZ34A (integralni ispit, 09.10.1998. godine)

Realni brojevi se predstavljaju sa 11 bita formatom **seeeeemmmmm** gde je **s** bit za predznak broja, **eeeee** (5) biti eksponenta u kodu sa viškom 8, a **mmmmm** (5) biti normalizovane mantise sa skrivenim bitom 0.5<=M<1. Celi brojevi predstavljaju se sa 10 bita u drugom komplementu. Ceo broj I ima izgled 177_{16} a ceo broj J ima izgled 325_{16}. Najpre se izvrši celobrojno sabiranje N=I+J. Zatim se izvrši konverzija brojeva I i J u realne brojeve, i sabiranjem tih realnih brojeva dobije broj B. Kolika će biti apsolutna vrednost razlike brojeva N i B? (Napomena: Sva zaokruživanja vrše se prema pravilima IEEE standarda, a svi koraci u sabiranju realnih brojeva vrše se na širini određenoj formatom mantise.)

A) 4 B) 8 C) 2.125

Rešenje

Prvo ćemo izvršiti celobrojno sabiranje.
$I = 177_{16} = 01\ 0111\ 0111$
$J = 325_{16} = 11\ 0010\ 0101$
$I+J = 100\ 1001\ 1100$
Pošto u memoriji imamo samo 10 bita sa predstavljanje celih brojeva, jedanaesti bit koji je generisan u procesu sabiranja otpada i dobijamo $0010011100_2 = 9C_{16} = 156_{10}$ kao rešenje.
Realno sabiranje se vrši tako što prvo celobrojne vrednosti konvertujemo u realne, uz potrebna zaokruživanja, a potom dobijene realne vrednosti saberemo.
I: $0101110111 = 0.101110111 \cdot 2^9 \approx 0.101111 \cdot 2^9$ s
J: 11 0010 0101, što znači da je J<0.
-J: $00\ 1101\ 1011 = 0.11011011 \cdot 2^8 \approx 0.110111 \cdot 2^8$

Sada je potrebno svesti ova dva broja na isti eksponent i pri tome izvršiti zaokruživanja, ako se za tim ukaže potreba:
$J = (-1) \cdot 0.110111 \cdot 2^8 = (-1) \cdot 0.0110111 \cdot 2^8 \approx (-1) \cdot 0.011100 \cdot 2^8$.
Pošto su brojevi različitog znaka, oduzimamo mantisu manjeg od mantise većeg broja:
M_I: 0.101111
$-M_J$: - 0.011100
$M_I - M_J$: 0.100110
Traženi broj izgleda ovako: $0.100110 \cdot 2^8$, što je jednako pozitivnom celom broju $0010011000_2 = 98_{16} = 152_{10}$.
Odavde se može zaključiti da je apsolutna razlika ova dva zbira jednaka 4.
Napomena: Kada se vrši konverzija iz celobrojnog u realni broj, vrši se zaokruživanje na po vrednosti najbliži realni broj koji se može predstaviti u memoriji datog računara. Konverzija u obrnutom smeru podrazumeva ili zaokruživanje početnog realnog broja na po vrednosti najbliži celi broj ili prosto odsecanje razlomljenog dela realnog broja. Ako nije posebno naglašeno, podrazumeva se prvi pristup.

Odgovor: A

Zadatak Z14

Brojevi sa pokretnom tačkom u računaru imaju binarni oblik **seeeeeemmmmmmmmm**, gde je **s** bit za znak broja, **m** su bitovi normalizovane mantise sa skrivenim bitom ($0.5 \le M < 1$), **e** su bitovi eksponenta u kodu sa viškom 2^5. Naći zbir brojeva **x** i **y** čiji su oblici u računaru zadati u brojnom sistemu sa osnovom **q**. Rezultate proveriti pretvaranjem zadatih vrednosti i rezultata u decimalni brojni sistem.

PROGRAMSKI JEZIK C

Zadatak C5

Sledeći program za određivanje rešenja linearne jednačine Ax+B=0, za A<>0, napisan na programskom jeziku C, sadrži više grešaka. Ispraviti sve greške!

```
main                    main predstavlja funkciju – treba da stoji main()
  Float _x,a,b          C je CaseSensitive jezik (razlikuje mala i velika slova)
                        deklaracija treba da stoji unutar programskog bloka
{
  scanf('%f %f',a,b);   niz znakova u jeziku C se ogranicava parom znakova "
                        scanf zahteva adresu, a ne vrednost; Prevodilac za jezik C ne
                        prijavljuje ovo kao gresku.
  _x=-b/a;
  printf('%f',_X)       _X nije dobro, jer smo deklarisali _x
}
```

Rešenje:

Ispravljeni program izgleda ovako:

```
#include <stdio.h>          /* na početku svakog programa dolaze direktive */
main()
{
   float _x,a,b;            /* _x ne smeta (Promenljiva moze sadrzati znak _ ) */
   scanf("%f %f",&a,&b);
   _x=-b/a;
   printf("%f",_x);         /* svaka naredba jeziku C završava se znakom tacka-zarez; */
}
```

Komentar:

Neispravno napisan program je testiran i kompajliran u programskom paketu Visual Studio.net. Poruke kompajlera su sledeće:

```
Compiling...
c5.c
c5.c(5) : error C2061: syntax error : identifier 'Float'
c5.c(5) : error C2059: syntax error : ';'
c5.c(7) : error C2143: syntax error : missing ';' before '{'
c5.c(7) : error C2449: found '{' at file scope (missing function header?)
c5.c(8) : error C2015: too many characters in constant
c5.c(14) : error C2059: syntax error : '}'
```

Na početku svakog programa koji nešto učitava ili ispisuje mora postojati direktiva **include** sa parametrom **<stdio.h>.** Ovo predstavlja uputstvo prevodiocu da iz datoteke **stdio.h** treba da pročita osobine bibliotečkih funkcija za ulaz i izlaz podataka.

Zadatak C10

Sastaviti program na programskom jeziku C za ispisivanje tablice ASCII kodova za sve štampajuće znake.

Rešenje

ASCII kod sa brojem 32 je blanko simbol ili razmak. Prva 32 ASCII koda (od 0 do 31) i svi ASCII kodovi veći ili jednaki 127 predstavljaju specijalne simbole koji se koriste u različite svrhe i najčešće nemaju neko jasno slovno značenje. Koristili su se za iscrtavanje DOS prozora, i na taj način simulirali grafiku u tekstualnom režimu rada. Konkretno, prozori razvojnog okruženja Turbo C su napravljeni na ovaj način! ASCII kodovi između 32 i 126 su štampajući znaci, i ima ih ukupno 95.

Znaci se prikazuju po kolonama. U jednoj koloni se prikazuje po 19 znakova, a u jednom redu se prikazuje po 5 znakova (5x19=95). Prikazaćemo sve ASCII kodove od 32 do 126.

Program za prikazivanje tablice ASCII kodova može se realizovati na više načina.

I način

```
#include <stdio.h>
main()
{
char c;
int i;
   printf ("\t\tTablica ASCII kodova \n \n");
   for(c = ' '; c < ' ' + 19; c++)
   {
      for(i = 0; i < 95; i += 19)
         printf("%3d%c   ", c+i, c+i);
      putchar('\n');
   }
}
```

II način – ovako NE TREBA raditi!!!

```
#include <stdio.h>
main()
{
char c=' ';
int i;

   printf ("\t\tTablica ASCII kodova \n \n");
linija:
   i=0;
   znak:
   printf("%3d %c     ", c+i, c+i);
   i=i+19;
   if (i<95) goto znak;
   printf("\n");
   c=c+1;
   if (c<' '+19) goto linija;
}
```

Komentar:

U drugom rešenju se koristi skok sa proizvoljnim odredištem: **goto**. **Korišćenje naredbe goto treba izbegavati** (bilo koji algoritam koji sadrži **goto** može se ostvariti na neki drugi način, što i pokazuje prvo rešenje). Glavni razlog za izbegavanje naredbe **goto** je taj što ta naredba ima veliki stepen slobode u izboru odredišta skoka tako da omogućava sastavljanje nestrukturiranih, vrlo nepreglednih programa, koji jako često sadrže mnogo grešaka uzrokovanih upravo lošom strukturom.

Operatori: prioritet i redosled primene

PRIORITET	BROJ OPERANADA	OPERATORI	SMER GRUPISANJA
15	2	[] () . ->	→
14	1	! - ++ -- + - * & (tip) sizeof	←
13	2	* / %	→
12	2	+ -	→
11	2	<< >>	→
10	2	< <= > >=	→
9	2	== !=	→
8	2	&	→
7	2	^	→
6	2	|	→
5	2	&&	→
4	2	||	→
3	3	?:	←
2	2	= += -= *= /= %= &= ^= |= <<= >>=	←
1	2	,	→

Primeri

Relacijski operatori

Izraz	Objašnjenje (rezultat)
5 > 7	0
10 <= 20	1
8 == 13 > 5	8==(13>5) -> 8==1 -> 0
14 > 5 < 3	(14>5)<3 -> 1<3 -> 1
a < b < 5	(a<b)<5 -> (0/1)<5 -> 1
a+5 >= c-1.0/e	(a+5)>=(c-(1.0/e))

Operatori za dodelu vrednosti

Izraz	Objašnjenje
y = a * x + b	
d *= e + f	d = (d*(e+f))
d = d * e + f	d = ((d*e)+f)
a=b=c=d+5	c=d+5,b=c,a=b
a = b++ + 3*(c=d<<3)	c=d<<3,u=b,b=b+1,a=u+(3*c)
a = b++ + 3*c = d<<3	greška !

Redosled računanja i bočni efekti

((a<<5)+4/b)*(d-=c-2) * ++e

Standard garantuje da će se pre množenja izračunati vrednosti operanada u zagradama. Standard ne precizira kojim će se redom računati (prvo levi ili prvo desni operand) – implementaciona zavisnost.
Ako rezultat zavisi od redosleda izračunavanja operanada - loš stil pisanja izraza.
Bočni efekti: uzgredna promena vrednosti jedne ili više promenljivih. Rezultat zavisi od redosleda izračunavanja operanada ako se bar jedna promenljiva, koja podleže uticaju nekog bočnog efekta, koristi na više od jednog mesta u izrazu. Operatori koji proizvode bočne efekte su ++, -- i svi operatori dodele vrednosti.

Izraz	Objašnjenje
a=b*c + b++	u=b*c, v=b, b++, a=u+v
	v=b, b++, u=b*c, a=u+v
a=b*c+(b=d/e)	u=b*c, v=(b=d/e), a=u+v
	v=(b=d/e), u=b*c, a=u+v
a[i] = ++i	Indeksiranje je bin.op.
	u=i, ++i, v=i, a[u]=v
	i++, v=i, u=i, a[u]=v

Zadatak C15

Odrediti čemu su ekvivalentni sledeći izrazi (koristiti zagrade da bi se eksplicitno odredio redosled izračunavanja)

```
1. x+=y-=m
2. n%=y+m
3. m++ - --j
4. x=j * j++
5. ++j==m!=y*2
```

Odgovor:

```
1. x=(x+(y=(y-m))
2. n=(n % (y+m))
3. m-(j-1); m=m+1; j=j-1
4. x=j * j++ ;  ovo zavisi od toga kojim se redom racuna, sa leva na desno ili obrnuto
       Odgovor: implementaciono zavisno
5. ((j+1)==m)!=(y*2); j=j+1
```

Primer – Bočni efekti (autor: Jelica Protić)

Sledeći primer ilustruje manifestovanje bočnih efekata u 8 različitih izraza, za različite prevodioce jezika C. Inicijalna vrednost promenljive **j** je 5.

	izraz	VS 6.0	VS .NET 2003	TC 2.0	GNU GCC	ARM C
1.	`j * j++`	25	25	30	25	30
2.	`j++ * j`	25	25	30	25	30
3.	`++j * j`	36	36	36	36	36
4.	`j * ++j`	36	36	36	36	36
5.	`(j=3) * (j++)`	9	9	9	9	9
6.	`(j=3) * (++j)`	16	16	12	12	12
7.	`(j++) * (j=3)`	9	9	15	9	15
8.	`(++j) * (j=3)`	9	9	18	9	18

Pitanje: zašto VS dobija 16 kao rezultat u primeru 6?

Zadatak C20

Sastaviti program na programskom jeziku C koji učitava srednje temperature po mesecima za 12 meseci i na osnovu njih izračuna i ispiše srednju temperaturu za celu godinu.

Rešenje:

```
#include <stdio.h>
#define BROJ_MESECI 12
main()
{
   enum meseci {JAN=1,FEB,NAR,APR,MAJ,JUN,JUL,AVG,SEP,OKT,NOV,DEC};
   enum meseci mesec=JAN;
/*
   Moguce je umesto gornje dve linije napisati samo jednu:
   enum meseci {JAN=1,FEB,NAR,APR,MAJ,JUN,JUL,AVG,SEP,OKT,NOV,DEC} mesec=JAN;
*/
   float temperature[BROJ_MESECI];
   float srednja_temp=0;

   while (1)
   {
      printf("Temperatura za mesec %2d: ",mesec);
      scanf("%f",&temperature[mesec - 1]);  /* niz u C-u uvek ide od 0 */
      srednja_temp+=temperature[mesec - 1];
      if (mesec==DEC) break;
      mesec++;
   }

   srednja_temp/=BROJ_MESECI;
   printf("Srednja temperatura je %.2f\n",srednja_temp);
}

/*
   Jasno je da nam niz nije potreban za racunanje srednje vrednosti
   dovoljno je da mesecnu temperaturu ucitavamo u neku pomocnu (float)
   promenljivu i da nju dodajemo na srednja_temp
*/
```

Komentar

Prikaz nekih vrednosti adresa i niza temperature:

```
&temperature,p: 22FC(SS):0FCC
&temperature[0],p: 22FC(SS):0FCC      - adresa clana 0
&temperature[1],p: 22FC(SS):0FD0      - adresa clana 1
temperature: { 3.0,8.0,12.0,15.0,20.0,26.0,29.0,30.0,26.0,20.0,13.0,7.0 }
                                      - prikaz celog niza
temperature[0]: 3.0                   - vrednost clana sa indeksom 0
```

Ovakav prikaz se može dobiti pomoću Watch prozora razvojnog okruženja Turbo C. Najpre se navodi izraz čija se vrednost posmatra, a onda način kako ona treba da bude ispisana. Uz pomoć ovog dodatka, može se isti izraz posmatrati na više načina.

izraz,d – kao broj

izraz,c – kao karakter

izraz,p – kao pokazivač (adresa)

Zadatak C25

Sastaviti program na programskom jeziku C za određivanje broja velikih slova, malih slova i cifara u tekstu koji se iz proizvoljnog broja redova učitava preko glavne ulazne jedinice. Tekst se završava znakom za kraj datoteke.

Rešenje

```
#include <stdio.h>
#include <ctype.h> /* treba za funkcije vezane za ispitivanje slova */
main()
{
   int znak, vel_sl=0, mal_sl=0,cifra=0;

   printf("Unesite zeljeni tekst \n");

   while ((znak=getchar())!=EOF)
   {
      vel_sl += isupper(znak) != 0; /*  ctype.h */
      mal_sl += islower(znak) != 0; /*  ctype.h */
      cifra  += isdigit(znak) != 0; /*  ctype.h */
   }
   printf("Velikih slova ima %d\n",vel_sl);
   printf("Malih slova ima %d\n",mal_sl);
   printf("Cifara ima %d\n",cifra);
}
```

Komentar

U zaglavlju **ctype.h** se nalaze prototipovi funkcija za ispitivanje znakova. U ovom programu su upotrebljene sledeće funkcije:

- **isupper(znak)**: ispituje da li je zadovoljen uslov (znak>='A') && (znak<='Z')

- **islower(znak)** : ispituje da li je zadovoljen uslov (znak>='a') && (znak<='a')

- **isdigit(znak)** : ispituje da li je zadovoljen uslov (znak>='0')&&(znak<='9')

Sve **is**_ funkcije vraćaju 0 za logičku neistinu, a različito od 0 (ne obavezno 1) za logičku istinu

Zadatak C30

Sastaviti program na programskom jeziku C koji učitava decimalan pozitivan celi broj u obliku niza znakova i ispisuje njegovu vrednost u binarnom obliku. Pretpostaviti da se za interno predstavljanje celih brojeva koristi 16 bitova.

Rešenje

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
main()
{
char dec[10];
short int bin, i;

  printf("Unesite decimalan broj: ");
  scanf("%s",dec);   /* ucitava string sa standardnog ulaza (dec=&dec[0]) */
  /*
  atoi vraca 0 ukoliko nije uspela konverzija.
  Ukoliko je strlen(dec)=0 onda se drugi deo USLOVA (iza && operatora)
  nece ni proveravati. Po postavci zadatka ocekuje se pozitivan broj,
  i takav uslov se jednostavno, ovim putem, moze proveriti.
  */
  if (strlen(dec) && (bin=atoi(dec)))
  {
    printf("Binarni broj: ");
    i=-1;
    while (++i<16)
    {
      putchar((bin & 0x8000) ? '1' : '0');
      /*
      0x8000 ima jedinicu na najvisem 15-om bitu
      Gornja naredba ce ispisati 15-i bit broja bin!
      Najpre ispisujemo najvise bitove, jer takav prikaz zelimo na ekranu
      bit15 bit14 ... bit2 bit1 bit0
      */
      bin <<= 1 ; /* bin = bin shl 1 ; pomeramo bin ulevo za 1 bit */
      if (i%4 == 3)
        putchar(' ');  /* prikaz razmaka izmedju svake polovine bajta */
    }
  printf("\n");
  }
  else
    printf("Neispravan broj ili nula\n");
}
```

Komentar

16-bitni binarni broj može da ima najveću vrednost 65535, ukoliko se tretira kao **unsigned**, odnosno 32767, ukoliko se tretira kao **signed**. To konkretno znači da je za ovaj 16-bitni prikaz dovoljno koristiti niz brojeva **char dec[6]** (5 za cifre i 1 za terminator). Sa druge strane, ako se u ovom zadatku unese neki broj veći od 65535/32767 rezultat neće biti ispravan (biće prikazano samo najnižih 16-bitova zadatog broja ako je short int na datom računaru duži od 16 bita).

Funkcija **atoi** se može koristiti i za konverziju negativnih vrednosti.

Treba imati na umu i sledeću činjenicu: ako se unese više od 10 znakova nepredvidiv je ishod izvršavanja ovog programa, jer niz 'dec' ima dodeljeno svega 10 znakova (bajtova) u memoriji i ništa preko toga! Ostatak memorije pripada drugim podacima ili drugom programskom kodu, tako da se upisivanjem van granica niza može izazvati nepredvidivo ponašanje.

Zadatak C35

Koja od datih konstrukcija na programskom jeziku C predstavlja ekvivalent iskaza na programskom jeziku Pascal: **`if (a>b) then begin a:=1; b:=1 end`**

A) **`if (a>b) { a=1; b=1; }`**
B) **`if (a>b) a=1, b=1;`**
C) **`if (a>b) a=1; b=1;`**

Komentar

Odgovor će biti A ili V (A i B), zavisno od toga kako se definiše ekvivalentnost dva iskaza.
A) Ovo je upravo prepisana selekcija sa Pascala na C.
B) Ukoliko je ispunjen uslov biće izvršena jedna instrukcija u then grani, pri čemu se ona sastoji od 2 izraza. Efekat izvršavanja ova dva izraza je u ovom slučaju identičan slučaju pod A).
C) b=1 se izvršava u svakom slučaju, što ne važi za originalni segment.

Zadatak C40

Šta ispisuju sledeći programi?

a)

```
#include <stdio.h>
main()
{
  int x;
  for (x=0;x<100;x++)
  {
    /* ako x nije deljivo sa dva
    vrati se na pocetak tela ciklusa */
    if (x%2)
      continue;
    /* do ovde dolazi samo ako je
    x deljivo sa 2 (x mod 2 = 0) */
    printf("%d\n",x);
  }
}
```

A) Sve cele brojeve od 0 .. 99
B) Sve parne brojeve od 0 .. 99
C) Sve neparne brojeve od 0 .. 99

Odgovor

B

b)

```
#include <stdio.h>
main()
{
  int x=0,i;
  for (i=0;i<5;i++)
    switch (i)
    {
    case 1: x+=1;
    /* ukoliko je i=1 izvrsava se deo
    iza case 1: i case 2: */
    case 2: x+=2; break;
    /* Ako je i=2 izvrsava se samo deo
    iza case 2: (zbog break;) */
    case 4: x+=3;
    /* Ako je i=4 izvrsava se samo deo
    iza case 4: */
    }
  printf("d=%d\n",x);
}
```

A) 5
B) 6
C) 9

Odgovor

N

Komentar

Najpre je x=0;
Za i=0, x se ne menja (x=0);
Za i=1, x+=1; x+=2; => x=3;
Za i=2, x+=2; => x=5;
Za i=3, x se ne menja;
Za i=4, x+=3; => x=8;

Zadatak C45

Sastaviti program na programskim jeziku C koji formira slučajan celobrojni niz sastavljen od jednocifrenih brojeva i izvrši uređivanje niza po neopadajućem redosledu vrednosti brojeva.

Rešenje:

Za sortiranje brojeva postoji niz algoritama. Više detalja o ovoj oblasti se može pronaći u knjizi "Strukture podataka" profesora Tomaševića. Ovde je predstavljen najjednostavniji algoritam pod nazivom **Selection sort**, odnosno sortiranje metodom izbora. Ideja je sledeća: u i-tom prolazu se odabere i-ti (u ovom slučaju) najmanji broj, pa se smesti na i-to mesto.

```
#include <stdio.h>
#include <stdlib.h>
#define DIM 50

main()
{
  int n,a[DIM],i,j,b;
  for (;;)
  {
    printf("\n\n Duzina niza (max %d): ",DIM); scanf("%d", &n);
    if ((n <= 0) || (n > DIM)) break;
    /* break prekida for ciklus. Ovo je ciklus sa izlazom u sredini */
    printf("\nPocetni niz: \n\n");
    for (i=0;i<n;i++)
      printf("%d%c", a[i] = rand()/((double)RAND_MAX+1.0)*10,
                    (i % 30 == 29 || i == n-1) ? ('\n') : (' '));
      /* Generisace se slucajni brojevi u intevalu od 0 .. 10.
      rand() generise slucajan broj od 0 .. RAND_MAX
      (RAND_MAX je u slucaju TurboC-a 32767 ) */

  /* sledi  sortiranje */
  for (i = 0; i < n-1; i++)
    for (j = i+1; j < n; j++)
      if (a[i] > a[j]) b = a[i],a[i] = a[j],a[j] = b;     /* zamena 2 promenljive */

  printf("\nSortirani niz:\n\n");
  for (i = 0; i < n; i++)
    printf("%d%c", a[i], (i % 30 == 29 || i == n-1) ? ('\n') : (' '));

/* Deo (i%30==29 || i==n-1) ? ('\n') : (' ') predstavlja karakter
koji treba da se napise iza broja (Clana) niza. Ukoliko se nalazimo
na kraju niza (poslednji element i==n-1), ispisujemo znak za novi red.
Takodje, radi jasnijeg prikaza posle svakog 30-tog clana niza ispisujemo
isto znak za novi red. U svakom drugom slucaju ispisuje se blanko (razmak).*/
  }
}
```

Primer sortiranja 10 slučajnih brojeva:

Prolaz 1: **a: { 2, 4, 1, 6, 5, 0, 1, 8, 6, 7 }**

Prolaz 2: **a: { 0, 4, 2, 6, 5, 1, 1, 8, 6, 7 }**

Prolaz 3: **a: { 0, 1, 4, 6, 5, 2, 1, 8, 6, 7 }**

Prolaz 4: **a: { 0, 1, 1, 6, 5, 4, 2, 8, 6, 7 }**

Prolaz 5: **a: { 0, 1, 1, 2, 6, 5, 4, 8, 6, 7 }**

Prolaz 6: **a: { 0, 1, 1, 2, 4, 6, 5, 8, 6, 7 }**

Prolaz 7: **a: { 0, 1, 1, 2, 4, 5, 6, 8, 6, 7 }**

Prolaz 8: **a: { 0, 1, 1, 2, 4, 5, 6, 6, 8, 7 }**

Prolaz 9: **a: { 0, 1, 1, 2, 4, 5, 6, 6, 7, 8 }**

Zadatak C47

Koji od programskih segmenata na programskom jeziku C daju isti izlaz kao dati programski segment na programskom jeziku Pascal?

```
j:= -2;
for i:= -2 to j*j do
  begin write(j:3); j:=j+1 end;
```

A) `for (i=j=-1;i<=j*j;i++,j++) printf(" %3d ",j);`

B) `i=-2; while (printf(" %3d ",i),i++ -4);`

C) `i=j=-2; while (i++ == 4 ? 0 : 4) printf(" %3d ",j++);`

Obrazloženje:

U programskom jeziku Pascal se interval for ciklusa određuje neposredno po ulasku u ciklus i ne menja se u toku izvršavanja ciklusa, dok se u programskom jeziku C uslov kod for ciklusa računa pre svakog ponavljanja.

Ovaj programski segment na programskom jeziku Pascal ispisuje -2,-1,0,1,2,3,4.

Programski segment (A) na programskom jeziku C je logički neispravan, jer **i** nikada neće prestići vrednost **j*j**. Dodirna tačka im je za i==0,j==0 i i==1,j==1, ali posle toga zbog obostranog inkrementiranja **i** će biti uvek manje od **j*j**. U realnosti, u nekom trenutku će doći do prekoračenja opsega celih brojeva, i **j*j** će postati manje od **i**, za vrednost **j** takvu da vrednost izraza **j*j** prekoračuje maksimalni celi broj.

Programski segment (B) na programskom jeziku C je logički ispravan i ispisuje iste vrednosti kao i dati programski segment na programskom jeziku Pascal. Zadnji ispisani broj je 4, tada se ne postiže uslov za ulaz u petlju (i-4 == 0, što predstavlja logičku neistinu), a kao bočni efekat i=5.

Programski segment (C) na programskom jeziku C je logički ispravan. Međutim, on ne ispisuje iste vrednosti kao i programski segment na programskom jeziku Pascal. Ispisuje jedan broj manje (tj. sve od -2 do 3). Iz ciklusa se izlazi kada je i==4. Broj koji se ispisao pre toga je bio 3, a bočni efekat je bio j=4.

Zadatak C48

Data su tri segmenta programa na programskom jeziku C:

I

```
for (i=x=y=0; ; ++i)
  {x++; if (i==n) break; y++; }
printf("%ld %ld",x,y);
```

II

```
x=0, y=0;
for (i=0; i<=n; i++)
  {x=x+1; if (i<n) continue; y=y+1;}
printf("%ld %ld",x,y);
```

III

```
x=i=0;
while (i<=n)
  { i++; ++x; y=x>n;}
printf("%ld %ld",x,y);
```

Ako su sve promenljive celi brojevi koja dva segmenta daju isti izlaz za n>0?

A) I i II B) II i III C) I i III

Obrazloženje:

I uvek ispisuje x=n+1, y=n,a II i III uvek ispisuju x=n+1, y=1 (n ne utiče na vrednost y).

Zadatak C50

Napisati na programskom jeziku C program koji analizira tekst koji se unosi sa standardnog ulaza, prepoznaje realne brojeve po formatu **cc...c.ddd...d** i ispisuje ih po formatu **c.ddE+ee**, a sve ostale učitane znake ispisuje bez izmene.

Rešenje

```
#include <stdio.h>
#include <ctype.h>
#define DECIMALNA_TACKA '.'
#define ZNAK_ZA_KRAJ '!'

main()
{
int c, j, broj_cifara;
double vrednost, decimala;

  do
  {
    /* sve dok se ne pojavi neka cifra, ispisuju se karakteri sa ulaza */
    while (! ( isdigit(c = getchar()) || (c==ZNAK_ZA_KRAJ) ) )
      putchar(c);
    if (c == ZNAK_ZA_KRAJ)
      break;  /* Ukoliko je nadjen znak ! onda se prekida glavna petlja */

    vrednost = 0.;
    broj_cifara = 0;
    /* sve dok se ne naidje na tacku, dodaje se vrednosti pritisnuta cifra na kraj */
    while ( isdigit(c) )
      vrednost = vrednost * 10 + (c-'0'), c = getchar();
    if (c == DECIMALNA_TACKA)
    /* sve dok se unose cifre, dodaju se na kraj decimalnog dela */
    {
      c=getchar();
      while ( isdigit(c) )
      {
        broj_cifara++;
        decimala = c - '0';
        for (j=0; j < broj_cifara; j++)
          decimala/=10;
        vrednost += decimala;
        c = getchar();
      }
    }
    ungetc(c,stdin);
    printf("%1.2E",vrednost);
  } while (c != ZNAK_ZA_KRAJ); /* '!' je trenutno znak za kraj */
}
```

Obrazloženje

Iz programa se izlazi ukoliko se bilo gde u liniji upiše znak '!'

Od unete cifre, broj koji ona predstavlja dobija se kada se od njenog ASCII koda oduzme ASCII kod znaka '0'.

Pitanje

Šta će se dobiti na izlazu za sledeći red?

123.123.123

Odgovor:

1.23E+02.1.23E+02

Zadatak C55

Šta ispisuje sledeći program?

a)

```
/*C55a*/
#include <stdio.h>
main()
{
  int a[]={1,0}, *p;
  p=a;
  printf("%d, ", *p++);
  printf("%d\n", *p);
}
```

A) 1, 2 B) 1, 0 C) 0, 0

Komentar

Odgovor je B. Prioritet * i ++ je isti, ali je asocijativnost ove grupe operatora sdesna ulevo.

b)

```
/*C55b, J.Protic*/
#include <stdio.h>
main()
{
  int a[]={0, 1, 2, 3, 4};
  int i, *p;
  for (p=a, i=0; p<=&a[4]; p++)
    printf("%d", *p);
  printf("\n");
  for (p=&a[0], i=0; p+i<a+4; p++, i++)
    printf("%d", *(p+i));
  printf("\n");
  for (p=a+4, i=0; i<=4; i++)
    printf("%d", p[-i]);
  printf("\n");
}
```

Rešenje

```
01234
02
43210
```

c)

```
/*C55c, L.Kraus, primer, modifikacija J.Protic, I.Tartalja, dopuna A.Bosnjakovic*/
#include <stdio.h>
#include <stdlib.h>
main()
{
  int **a, n, i, j;
  printf("N="); scanf("%d", &n);
  a = calloc(n, sizeof(int*)); /*alocira memoriju za n pokazivaca na vrste*/
  for (i=0; i<n; i++)
  {
    *(a+i)= calloc(n, sizeof(int));
    /*alocira memoriju za n elemenata vrste koji su tipa int*/
    for (j=0; j<n; j++) *(*(a+i)+j) = rand()/((double)RAND_MAX + 1) * 10;
/* C55c.c(44) : warning C4244: '=' : conversion from 'double ' to 'int ', possible loss
of data */
  }
  for (i=0; i<n; printf("\n"), i++)
    for (j=0; j<n; j++)
      printf("%d", *(*(a+i)+j));
  for (i=n-1; i>=0; i--) printf("%d", *(*(a+i)+n-1-i));
  putchar('\n');
}
```

A) sadržaj matrice vrstu po vrstu, a potom sporednu dijagonalu, sleva-udesno
B) sadržaj matrice vrstu po vrstu, a potom glavnu dijagonalu, sdesna-ulevo
C) sadržaj matrice vrstu po vrstu, a potom sporednu dijagonalu, sdesna-ulevo

Odgovor:

A

Zadatak C57

Da bi se učitani znakovni niz (string) precizno ispisao «unatrag» (samo znaci koji su unošeni sa standardnog ulaza), koja naredba se može staviti umesto ***?

```
scanf(%s%, string1); a = string1; *** while (a>string1) putchar(*(--a));
```

A) `while (*a) a++;`

B) `for (; *a; a++);`

C) `while (*a++);`

Odgovor

V (A i B)

Komentar

Umesto *** mora da stoji naredba koja će postaviti pointer a tako da pokazuje na NULL **('\0')** karakter pointera **string1,** jer se u narednoj while petlji pointer **a** najpre dekrementira pa se tek tada ispisuje odgovarajući znak. Odgovor C nije dobar jer postavlja (zbog bočnog efekta) pointer **a** na znak iza **'\0'** karaktera.

Zadatak C60

Napisati program na programskom jeziku C koji učitava jedan znakovni niz (**string**) **S1** i jedan ceo broj **M**, a zatim formira novi znakovni niz **S2** samo od onih znakova na čijim su pozicijama odgovarajući bitovi broja **M** jednaki 1. Smatrati da se ceo broj predstavlja u 16-bitnoj lokaciji, a da znakovni niz može imati najviše 16 znakova. Na kraju, program ispisuje novi znakovni niz **S2**.

Rešenje

```
/* C60.c, 9.4.1994., II parcijalni ispit */
#include <stdio.h>

main()
{
short int M, maska;
char S1[17], *s1, S2[17], *s2;

   printf("Unesi string S1 i broj M:\n");
   scanf("%s%d", S1, &M);
   s1 = S1;
   s2 = S2;
   maska = 1;
   while (maska && *s1)
   {
      if (M & maska) {*s2 = *s1; s2++; };
      s1++; maska<<=1;
   }
   *s2 = '\0';
   printf("Novi string S2: %s\n", S2);
}
```

Komentar

U ovom programu promenjiva **maska** služi da ispitamo gde se nalaze jedinice u broju **M** koji zadaje korisnik. Broj bitova broja **M** (16) odgovara broju znakova stringa **S1**. Na osnovu pozicije tih jedinica određuju se znakovi koji će biti u stringu **S2**. Svaki od stringova **S1** i **S2** zauzima ukupno 17 bajtova u memoriji – 16 za same znakove i 1 za završni nulti znak.

Uslov **(maska && *s1)** je neophodan da bi se dobio ispravan niz **s2**.

Zadatak C65

Koji od sledećih izraza je ekvivalentan izrazu **ar[1][2],** ako je data sledeća deklaracija:
int ar[][3]={{0,1,2}, {3,4,5}}
A) ***(int*) (((char*)ar + (1*3*4)) + (2*4))**
B) ***(ar[1]+2)**
C) ***((*ar+1)+2)**

Odgovor

B

Obrazloženje

Watch

Name	Value
⊟ ar	0x0012ff68
⊟ [0x0]	0x0012ff68
[0x0]	0
[0x1]	1
[0x2]	2
⊟ [0x1]	0x0012ff74
[0x0]	3
[0x1]	4
[0x2]	5
⊞ *ar	0x0012ff68
**ar	0
⊞ ar+1	0x0012ff74
⊟ ar[1]	0x0012ff74
[0x0]	3
[0x1]	4
[0x2]	5
⊞ ar[1]+2	0x0012ff7c
*(ar[1]+2)	5
⊞ *ar+1	0x0012ff6c
⊞ (*ar+1)+2	0x0012ff74
*((*ar+1)+2)	3
⊟ (*ar+3)	0x0012ff74
[0x0]	3
[0x1]	4
[0x2]	5
⊟ (*ar+2)	0x0012ff70
[0x0]	2
[0x1]	3
[0x2]	4
⊟ (*ar+1)	0x0012ff6c
[0x0]	1
[0x1]	2
[0x2]	3
⊞ (char*)ar	0x0012ff68 ""
⊞ (char*)ar+(1*3*4)	0x0012ff74 "□"
⊞ ((char*)ar+(1*3*4))+(2*4)	0x0012ff7c "□"
⊞ (int*)(((char*)ar+(1*3*4))+(2*4))	0x0012ff7c
(int)(((char*)ar+(1*3*4))+(2*4))	5

Watch1 Watch2 Watch3 Watch4

A) zavisi od implementacije tipa **int**, u prikazanom slučaju je u redu
C) bilo bi ispravno da stoji ***(*(ar+1)+2),** ovako je u pitanju ***(ar[0]+3)**

Zadatak C70

Napisati program na programskom jeziku C koji učitava dva znakovna niza, izvrši nadovezivanje drugog na prvi, okrene "naopako" dobijeni niz i ispiše ga na standardnom izlaznom uređaju.

Rešenje

```
#include <stdio.h>
#include <string.h>
main()
{
  int n;
  char c, *d, *p, *prvi, *drugi;
  printf("Maksimalna duzina: ");
  scanf("%d\n", &n);
  p = calloc(2*n+1, sizeof(char)) ;
  d = calloc(n+1, sizeof(char));
  if ((NULL == d) || (NULL == p))
    printf("Nije moguca alokacija!\n");
  else
  {
    prvi = p; drugi = d;
    while ((*p = getchar()) != '\n') p++;
    *p = '\0';
    while ((*d = getchar()) != '\n') d++;
    *d = '\0';
    p = prvi; d = drugi;
    /* konkatenacija */
    while (*p) p++;
    while (*p++=*d++);
    /* okretanje */
    p = prvi;
    for (d=p+(strlen(p)-1); p < d; p++, d--)
      c=*p, *p=*d, *d=c;
    printf("%s\n", prvi);

    free(p); free(d);
  }

}
```

Komentar

Za string **p** se alocira dvostruko više memorije zbog nadovezivanja nizova. OBAVEZNO treba proveriti da li je uspela alokacija memorije.

Nakon alokacije memorije, vrši se učitavanje stringova, znak po znak, korišćenjem bibliotečke funkcije **getchar()**. Zatim se vrši konkatenacija, tako što se drugi string nadoveže na prvi.

Okretanje dobijenog niza u datom rešenju radi se tako što prvi član menja mesto sa poslednjim, drugi sa pretposlednjim, itd.

S obzirom na to da postoji dinamička alokacija memorije, svu alociranu memoriju OBAVEZNO treba osloboditi kada ona više nije potrebna (u ovom slučaju pre završetka programa).

Zadatak C75

Napisati program na programskom jeziku C koji ponavlja sledeću sekvencu operacija:

1. učitava informaciju o tipu podataka sa kojim će raditi (ceo broj ili znak);
2. učitava dužinu niza podataka;
3. učitava niz podataka zadatog tipa i dužine u dinamički alociranu memoriju;
4. ispisuje u heksadecimalnom obliku sadržaj učitanih podataka, bajt po bajt.

Rešenje

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

main()
{
void *blok_p, *bajt_p;
int lnt, n;
char o, *format_p;

  while(1)
  {
    o = 0;
    while (o != 'c' && o != 'Z' && o != 'k')
    {
      printf("Tip brojeva: (c)eli/(z)naci, ili (k)raj?");
      scanf("%c", &o);
      switch(o)
      {
      case 'c' :
           lnt = sizeof(int);
           format_p = "%d";
           break ;
      case 'z' :
           lnt = sizeof(char) ;
           format_p = "%c" ;
           break ;
      case 'k' :
           break ;
      default: printf("Neispravan unos! Ponovite.\n");
      }
    }

    if (o == 'k') break ;
    printf("Broj podataka?");
    scanf("%d", &n);
    blok_p = malloc(lnt*n) ;
    printf("Podaci:\n");
    for (bajt_p=blok_p; bajt_p<(char*)blok_p+n*lnt; (char*)bajt_p+=lnt)
    /*konverzija tipa, pretvaramo generički pokazivac u pokazivač na char */
    { printf("?"); scanf(format_p, bajt_p); }
    printf("Bajtovi:\n");
    for (bajt_p = blok_p; bajt_p<(char*)blok_p+n*lnt; ((char*)bajt_p)++)
      printf("%x", *(char *)bajt_p);
    printf("\n");
    free(blok_p);
  }
}
```

Komentar

Bajt_p i blok_p su generički pokazivači; pošto kod njih nije određen tip pokazivanih podataka, takav pokazivač ne može se koristiti za pristup podacima, već samo za čuvanje informacije o adresi nekog podatka. Pre pristupa nekom podatku, neophodno je generički pokazivač, upotrebom operatora za konverziju tipa, pretvoriti u pokazivač na željeni tip podataka. Posledice su nepredvidive ako pokazivani podatak nije tipa koji je naznačen u operatoru za konverziju tipa.

Zadatak C80b

Sastaviti program na jeziku C za učitavanje imena gradova uz njihovo uređivanje po abecednom redosledu i ispisivanje rezultata. U svakom redu se učitava po jedno ime sve dok se ne učita prazan red.

```
/* program za sortiranje imena gradova */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX_GRAD 100
#define MAX_DUZ 30

main()
{
  char *adrese[MAX_GRAD], *adresa;
  int znak, br_grad = 0, duz, i;

  do
  {
    adresa = calloc(MAX_DUZ, sizeof(char));
    for (duz = 0; duz < MAX_DUZ - 1; duz++)
      if ((znak = getchar()) != '\n')
        *(adresa + duz) = znak;
      else
        break;
    *(adresa + duz) = '\0';
    if (!duz)
    {
      free(adresa); break;
    }
    adresa = realloc(adresa, duz + 1);
    for (i = br_grad - 1; i >= 0; i--)
      if (strcmp(adrese[i], adresa) > 0)
        adrese[i + 1] = adrese[i];
      else
        break;
    adrese[i+1] = adresa;
  } while (br_grad++ < MAX_GRAD);

  for (i = 0; i < br_grad; i++)
    printf("%s\n", adrese[i]);
}
```

Komentar:

Ova tema je detaljno obrađena u knjizi profesora Krausa. Savetuje se studentima da detaljno prouče primere iz knjige.

Zadatak C90

Napisati program koji ispisuje redom binarne vrednosti brojeva koji su zadati putem komandne linije. Glavnu obradu izdvojiti u potprogram, a potprogram smestiti u posebnu datoteku.

raspak.h

```
/* limits.h - implementation dependent values
   limits.h sadrzi vrednosti koje se ticu maksimalnih i minimalnih velicina podataka,
   kao i broja bitova koriscenih za neke tipove podataka, a koje su zavisne od konkretnog
   ostvarenja programskog jezika C */

#include <limits.h>

#define NUM_BITS( Type ) ( sizeof( Type ) * CHAR_BIT )

void raspak( unsigned, int [] );
```

raspak.c

```
#include "raspak.h"
void raspak( unsigned k, int bit[] )
{
  int i = NUM_BITS( unsigned );
  while ( i ) bit [ --i ] = k & 1, k >>= 1;
}
```

c90.c

```
/*
Napisati program koji ispisuje binarne vrednosti brojeva koji
su zadati na komandnoj liniji. Glavnu obradu izdvojiti u potprogram
*/

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

static const BITS_IN_GROUP =4;

int main ( int argc, char *argv[] )
{
  int i, j;
  int niz[ NUM_BITS( unsigned ) ];
  for( i = 1;i < argc; i++ )
  {
    raspak( atoi( argv[ i ] ), niz );
    printf("Bitovi broja %s: ", argv[i]);
    for( j =0; j < NUM_BITS( unsigned ); j++ )
    {
      printf( "%d", niz[ j ] );
      if ( j % BITS_IN_GROUP == BITS_IN_GROUP -1 )
        putchar ( ' ' );
    }
    putchar( '\n' );
  }
  return 0;
}
```

Komentar

raspak.h – definiciona datoteka u kojoj su deklarisani svi potrebni tipovi podataka, simboličke konstante, definicije makroa, kao i prototipovi korišćenih funkcija.

raspak.c – sadrži funkciju za rad sa binarnim brojevima (to je ono što se traži u zadatku).

c90.c – ovde je smešten je glavni program, koji koristi definicije iz **raspak.h** i poziva funkciju čija je implementacija u **raspak.c**

Zadatak C95

Navesti tip identifikatora **x** u sledećim definicijama:

char **x; - pokazivač na... pokazivač na... **char**

int (*x)[10]; - (pokazivač na...) niz od 10... **int**

int *x[10]; - niz od 10... pokazivača na... **int**

int *x(); - funkcija koja vraća... pokazivač na... **int**

int (*x)(); - (pokazivač na...) funkciju koja vraća... **int**

char (* (* x())[]) ();

funkcija koja vraća

pokazivač na

niz

pokazivača na

funkciju koja vraća

char

char (* (* x[3])()) [5];

niz od tri

pokazivača na

funkciju koja vraća

pokazivač na

niz od 5

char

Obrazloženje

Deklaracije se čitaju počev od identifikatora ka spoljašnjosti. Operatori **()** i **[]** imaju prioritet 15 i smer grupisanja s leva udesno, dok operator ***** ima prioritet 14 i smer grupisanja s desna ulevo (videti tablicu prioriteta). Zbog toga, kada se deklariše (definiše) pokazivač na niz ili funkciju, moraju da se upotrebe zagrade. Ovo znači da se prvo uoči identifikator, a zatim se, ako postoje obuhvatajuće zagrade, unutar tih zagrada čita po sledećem pravilu: prvo se čitaju modifikatori desno od identifikatora, i to redom s leva udesno; zatim se čitaju modifikatori levo od identifikatora, i to redom sdesna ulevo. Postupak se ponavlja dok se ne obrade sve obuhvatajuće zagrade, ako postoje. Na kraju se čita tip podatka koji stoji na početku deklaracije.

char *((*x)[](char **));

pokazivač na niz funkcija koje kao argument primaju pokazivač na pokazivač na karakter, a vraćaju pokazivač na karakter. **Ova definicija NIJE VALJANA jer u C-u nije dozvoljeno praviti niz funkcija, pa samim tim ni pokazivač na niz funkcija.**

Zadatak C100

Napisati funkciju na programskom jeziku C koja realizuje algoritam QuickSort. Funkcija treba da sortira niz podataka proizvoljnog tipa. Kriterijum sortiranja definisan je funkcijskim parametrom.

Rešenje

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
void swap( void **p1, void **p2 )
{
   void *temp = *p1;
   *p1 = *p2;
   *p2 = temp;
}
/* rekurzivna varijanta */
void quicksort(void *v[], int left, int right, int (*comp)(const void *, const void * ))
{
   int i, last;
   if( left >= right ) return;
   last = left;
   for ( i = left + 1; i <= right; i++ )
      if ( ( *comp )( v[ i ], v[ left ] ) < 0)
         swap( &v[ ++last ],&v[ i ] );
   swap( &v[ left ], &v[ last ] );
   quicksort( v, left, last - 1, comp );
   quicksort( v, last + 1, right, comp );
}
#define MAX_STR 20
#define MAX_DIM 10

void main()
{
   char *niz[ MAX_DIM ];
   int i,n;
   for ( i = 0; i< MAX_DIM; i++ )
   {
      niz[ i] = calloc( MAX_STR, 1 );
      if ( ( niz[ i ] == NULL )
         || ( fgets( niz[ i ], MAX_STR, stdin ) == NULL )
         || ( *( niz[ i ] ) == '\n' )
         ) break;
   }
   n = i - 1;
   quicksort( niz, 0, n, strcmp );
   for ( i = 0; i <= n; i++ ) fputs( niz[ i ], stdout );
}
```

Komentar

Funkcija **swap** ima dva argumenta; ovom funkcijom se postiže da ****p1** pokazuje tamo gde je pokazivao ****p2** i obrnuto. Funkcija **quicksort** se zasniva na algoritmu particijskog sortiranja, realizovana je rekurzivno, i treba da uredi neuređen niz.

Deklaracijom int (*comp)(const void*, const void*) definiše se pokazivač na funkcije čija je povratna vrednost tipa **int** i koje imaju dva argumenta tipa **const void*.** Tamo gde se u programskom kodu nalazi **(*comp)(v[i], v[left])** imamo indirektno adresiranje koje se primenjuje na promenljivu **comp**, što je praktično poziv funkcije koja se u memoriji nalazi na adresi na koju pokazuje pokazivač **comp**. Da bi ovo bilo ispravno, funkcija na koju **comp** pokazuje se mora slagati po tipu/tipovima argumenata navedenim pored **(*comp)**.

Zadatak C104

Napisati program u programskom jeziku C za rad sa kompleksnim brojevima. Preko argumenata programa zadaju se dva kompleksna broja i to tako da se prvo navodi realan deo prvog broja, zatim imaginarni deo prvog broja, zatim realni i imaginarni deo drugog kompleksnog broja, respektivno. Ako korisnik ne unese tačan broj argumenata potrebno je prekinuti izvršavanje programa. Takođe treba napisati potprogram koji proverava da li znakovni niz sadrži broj i pomoću tog potprograma treba ispitati da li su svi navedeni argumenti zaista brojevi. Ako bar jedan argument nije broj, potrebno je prekinuti izvršavanje programa. Program treba da sadrži i potprogram za sabiranje dva kompleksna broja koji vraća kompleksni broj kao rezultat kao i potprogram za konjugovanje kompleksnog broja koji nema povratnu vrednost. Glavni program treba da na standardnom izlazu ispiše sve argumente programa, zbir kompleksnih brojeva i konjugovane vrednosti oba kompleksna broja. Ispis treba vršiti pozivom odgovarajućeg potprograma koji takođe treba napisati. Kompleksne brojeve predstaviti strukturom.

Rešenje

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

#define TRUE 1
#define FALSE 0

/* istovremeno definisemo i strukturu sa imenom (SComplex) i tip podatka (TComplex) */
typedef struct SComplex
{
  double Re, Im;
} TComplex;

/* parsiramo string da bismo utvrdili da li predstavlja realni broj */
int DaLiJeBroj(const char *pstr)
{
  int bTacka = FALSE;
  if (*pstr=='+' || *pstr=='-')
    ++pstr;

  while(*pstr)
  {
    if (!isdigit(*pstr) && *pstr!='.')
      return FALSE;
    if (*pstr == '.')
    {
      if (bTacka)
        return FALSE;
      bTacka = TRUE;
    }
    pstr++;
  }
  return TRUE;
}

/* sabiramo dva broja i zbir vracamo kao rezultat funkcije */
TComplex Zbir(TComplex z1, TComplex z2)
{
  TComplex cplx;
  cplx.Re = z1.Re + z2.Re;
  cplx.Im = z1.Im + z2.Im;
  return cplx;
}
```

```
/* konjugujemo kompleksni broj, operacija se obavlja na samom primerku strukture */
void Konjug(TComplex *pZ)
{
  pZ->Im = - pZ->Im;
}

/* ispisujemo broj, u formatu x + jy */
void Ispis(TComplex z)
{
  if (z.Im >= 0)
    printf("%5.4f + j%5.4f", z.Re, z.Im);
  else
    printf("%5.4f - j%5.4f", z.Re, -z.Im);
}

/* glavni program, ulazne podatke zadajemo putem komandne linije */
int main(int argc, char *argv[])
{
TComplex z1, z2, zbir;
int i;

  if (argc != 5)
  {
    printf("Unesite tacno cetiri argumenata pri startu programa!\n");
    return -1;
  }

  printf("Uneli ste sledece argumente preko komandne linije:\n");
  for (i=1; i < argc; i++)
    printf("%d %s\n", i, argv[i]);

  for (i=1; i < argc; i++)
  {
    if (!DaLiJeBroj(argv[i]))
    {
      printf("Uneli ste argument koji nije broj!\n");
      return -1;
    }
  }

  z1.Re = atof(argv[1]);
  z1.Im = atof(argv[2]);

  z2.Re = atof(argv[3]);
  z2.Im = atof(argv[4]);

  printf("\nPrvi broj: "); Ispis(z1);
  printf("\nDrugi broj: "); Ispis(z2);
  zbir = Zbir(z1, z2);
  Konjug(&z1);  Konjug(&z2);

  printf("\nZbir: "); Ispis(zbir);
  printf("\nKonjug(z1): "); Ispis(z1);
  printf("\nKonjug(z2): "); Ispis(z2);
  printf("\n\nHvala na paznji!\n");

  return 0;
}
```

Komentar

Zadavanje argumenta putem komandne linije znači da glavni program ima dva argumenta. Argumenti komandne linije su stringovi, te ih moramo konvertovati pre korišćenja. Ti argumenti omogućavaju prenošenje parametara glavnom programu koji se zadaju u sastavu komande operativnog sistema za izvršavanje programa.

Zadatak C110

Date su sledeće deklaracije:

```
typedef int CeoBroj;
typedef int *PokazivacNaCeoBroj;
typedef int NizCelihBrojeva[100];
CeoBroj *pokazA;
PokazivacNaCeoBroj pokazB;
NizCelihBrojeva niz;

struct {
   int x,y;
} zapisA;

struct {
   int x,y;
} zapisB, zapisC;
```

Koje od sledećih naredbi dodele su semantički ispravne?

```
pokazA = pokazB; /* OK */
pokazA = niz; /* OK */
niz = pokazA; /* GREŠKA */

zapisB = zapisC; /* OK */
zapisA = zapisB; /* GREŠKA */
```

Obrazloženje

Prvi primer je ispravan zato što je moguća dodela između dve promenljive koje su istog tipa – **pokazA** i **pokazB** su pokazivači na **int**. Drugi primer je ispravan jer **pokazA** sada pokazuje tamo gde pokazuje **niz**. Tip imena niza je tip nepromenljivog pokazivača na nulti član, zato je treći primer pogrešan, jer se ne može nepromenljivom pokazivaču dodeliti nova vrednost.

U programskom jeziku C se koristi strukturna ekvivalencija tipova (**typedef** imena se pri tome zamenjuju odgovarajućim tipom) osim u slučaju struktura i unija. Za strukture i unije koristi se ekvivalencija po imenu. Svaka neimenovana struktura je poseban tip, čak i u slučaju da su sva polja neke dve strukture međusobno identična.

Zadatak C115

Sastaviti program na programskom jeziku C za spajanje sadržaja nekoliko sekvencijalnih tekst datoteka u jednu izlaznu datoteku. Imena datoteka se zadaju kao parametri u komandnoj liniji. Ime izlazne datoteke je prvi parametar. Ukoliko je prvi parametar "-", izlazna datoteka je stdout.

Rešenje

```
#include <stdio.h>
#include <string.h>
main (int argc, char *argv[])
{
  void prepis (FILE *, FILE*);
  FILE *ulaz, *izlaz;
  char *prog = argv[0];

  izlaz = stdout;
  if (argc > 1)
  {
    argc--;
    if (strcmp(*++argv, "-") != 0)
      if ((izlaz = fopen(*argv, "w")) == NULL)
      {
        fprintf(stderr,"%s:greska u otvaranju datoteke %s\n",prog,*argv);
        exit(1);
      }
  }
  while(--argc > 0)
    if ((ulaz = fopen(*++argv, "r")) != NULL)
      prepis(ulaz, izlaz), fclose(ulaz);
    else
    {
      fprintf(stderr,"%s: ne postoji datoteka %s\n", progr, *argv);
      exit(2);
    }

  fclose(izlaz);
  exit (0);
}

void prepis(FILE *ulaz, FILE *izlaz)
{
  int c;
  while((c=getc(ulaz)) != EOF) putc(c, izlaz);
}
```

Komentar

Elementi **char *argv[]** (niz pod imenom **argv** čiji su elementi pokazivači na **char**, gde operativni sistem smešta stringove iz komandne linije) se mogu tretirati u kodu sa ***argv** zbog promocije – niz se ne može inkrementirati kao što se to radi u datom primeru, ali ako se kao parametar funkcije navede neki niz, C to automatski tretira kao pokazivač na tip podataka kome pripadaju elementi niza, a za pokazivače je inkrementiranje dozvoljeno.

Zadatak C120

U binarnoj datoteci se nalaze zapisi o automobilima i vozačima. Svaki zapis sadrži podatak o tipu entiteta, samom entitetu i celobrojnu vrednost sledećeg zapisa u logički organizovanoj listi. Prvi zapis u datoteci je i prvi zapis liste, a poslednji zapis liste je zapis u čijem polju ukazatelja na sledeći zapis stoji 0. Svaki zapis u datoteci ima polje koje sadrži broj logički sledećeg zapisa u datoteci.

Napisati:

1. potprogram koji čita navedenu datoteku i formira listu zapisa u dinamički dodeljenoj memoriji, pri čemu zapise povezuje u suprotnom smeru od smera u datoteci;
2. potprogram koji ispisuje na standardnom izlazu listu iz tačke a) i
3. glavni program koji poziva gornje potprograme

cz120.h

```
#include <stdio.h>

enum tip {AUTO=1, VOZAC};

struct automobil {enum tip t; char reg_br[10], boja[10], model[10]; };
struct vozac {enum tip t; char pol[2], br_dozvole[10], ime[10], prezime[10]; };

union zapis {struct automobil a; struct vozac v; };

struct element_dat {union zapis z; int b; };
struct element_lis {union zapis z; struct element_lis * s; };

void pisi_auto( struct element_lis *);
void pisi_vozac( struct element_lis *);
void formiraj_listu( FILE*, struct element_lis **);
void pisi_listu( struct element_lis *);
void kreiraj_datoteku(char*);
```

cz120.c

```
#include <stdio.h>
#include "cz120.h"

void main(int argc, char *argv[]) {
   FILE *av_bin;
   struct element_lis *lista;

   if (argc > 2)
      kreiraj_datoteku(argv[1]);
   if (av_bin = fopen(argv[1],"rb"))
   {
      formiraj_listu(av_bin,&lista);
      pisi_listu(lista);
   }
}
```

pisi_av.c

```
#include <stdio.h>
#include "cz120.h"

void pisi_auto(struct element_lis *e)
{
   printf("Registarski broj: %s; Boja: %s; Model: %s\n",
      e->z.a.reg_br, e->z.a.boja, e->z.a.model);
}

void pisi_vozac(struct element_lis *e)
{
   printf("Pol: %s; Ime i prezime: %s %s; Broj dozvole: %s\n",
      e->z.v.pol, e->z.v.ime, e->z.v.prezime, e->z.v.br_dozvole);
}
```

formlist.c

```
#include <stdlib.h>
#include <stdio.h>
#include "cz120.h"
void formiraj_listu(FILE *av_bin, struct element_lis **ppLista)
{
  struct element_lis *preth=NULL;
  struct element_dat bafer;
  *ppLista = NULL;
  bafer.b = 1;
  while (bafer.b > 0)
  {
    fread(&bafer, sizeof(bafer), 1, av_bin);
    fseek(av_bin, sizeof(bafer)*(bafer.b-1), SEEK_SET);
    *ppLista = calloc(sizeof(struct element_lis), 1);
    (*ppLista)->z=bafer.z;
    (*ppLista)->s=preth;
    preth = *ppLista;
  };
}
```

pisilist.c

```
#include "cz120.h"
void pisi_listu(struct element_lis *lista)
{
  while(lista)
  {
    switch (lista->z.a.t)
    {
    case AUTO: pisi_auto(lista); break;
    case VOZAC: pisi_vozac(lista); break;
    default: printf("Neispravan element liste!\n");
    }
    lista = lista->s;
  }
}
```

kr_dat.c

```
#include "cz120.h"
void kreiraj_datoteku(char* filename)
{
  struct element_dat bafer;
  int izbor = 1;
  FILE *file = fopen(filename, "wb");
  while(izbor)
  {
    printf("Auto[1] ili vozac[2] ili kraj unosa[bilo sta]? \n");
    scanf("%d", &izbor);
    switch(izbor)
    {
    case AUTO:
      printf("Unesite redom sledece podatke: boja, model, registarski broj, KLJUC: \n");
      scanf("%s%s%s%d", bafer.z.a.boja, bafer.z.a.model, bafer.z.a.reg_br, &bafer.b);
      bafer.z.a.t = AUTO; break;
    case VOZAC:
      printf("Unesite redom sledece podatke: ime, prezime, pol, broj dozvole: \n");
      scanf("%s%s%s%s%d", bafer.z.v.ime, bafer.z.v.prezime, bafer.z.v.pol,
                          bafer.z.v.br_dozvole, &bafer.b);
      bafer.z.v.t = VOZAC; break;
    default: izbor = 0;
    }
    if (izbor)
      fwrite(&bafer, sizeof(struct element_dat), 1, file);
  }
  fclose(file);}
```

Komentar

Ovaj zadatak ilustruje korišćenje struktura i unija, kao i konstrukciju programskog sistema iz više datoteka. Shodno logičkom ustrojstvu celog programskog sistema, programer treba da odluči koje će funkcije imati u sistemu i kako će ih rasporediti po datotekama.

`cz120.h` – definiciona datoteka u kojoj su deklarisani svi potrebni tipovi podataka, simboličke konstante, definicije makroa, kao i prototipovi korišćenih funkcija.

`cz120.c` – ovde se nalazi glavni program, koji poziva odgovarajuće potprograme. U slučaju da datoteka sa podacima ne postoji, glavni program poziva funkciju koja će je napraviti, a potom čita iz te datoteke u listu zapisa, koju na kraju na standardnom izlazu.

`pisi_av.c` – ovde su smeštene funkcije **`pisi_auto()`** i **`pisi_vozac()`**, koje služe za ispis podataka o automobilima i o vozačima.

`formlist.c` – sadrži funkciju koja izvršava čitanje datoteke, koju smo formirali pozivom funkcije **`kreiraj_datoteku()`**, i formiranje lista zapisa u dinamički dodeljenoj memoriji.

`pisilist.c` – ovde se nalazi funkcija za ispis već formirane liste zapisa, koja zavisno od tipa zapisa poziva jednu od funkcija **`pisi_auto()`** i **`pisi_vozac()`**.

`kr_dat.c` – ovde se nalazi funkcija **`kreiraj_datoteku()`** koja kreira binarnu datoteku u kojoj će se nalaziti zapisi o automobilima i vozačima.

Zadatak C122

Potrebno je realizovati u programskom jeziku C skup funkcija za rad sa listom čije je ponašanje isto kao ponašanje podatka tipa liste iz pseudojezika. Funkcije za rad sa listom treba da se zovu isto kao u pseudojeziku (find_bolp, insert, move_forward...) i treba da budu realizovane u zasebnoj implementacionoj .c datoteci. Takođe je potrebno napraviti odgovarajuću definicionu .h (header) datoteku u kojoj će biti deklarisani svi potrebni tipovi podataka i prototipovi korišćenih funkcija. Podaci se u listi skladište preko generičkog pokazivača (**void ***), a potrebno je obezbediti brisanje podataka sadržanih u listi kroz call-back mehanizam. Napraviti inicializacionu funkciju koja inicijalizuje listu i koja postavlja odgovarajuću funkciju koja se poziva kroz call-back mehanizam pri uništavanju liste. Pointer na ovu funkciju se prosleđuje kao argument funkcije za inicijalizaciju liste. U listu je iz tekstualne datoteke potrebno pročitati podatke o studentima. Podaci su u fajl složeni tako da je u jednom redu ime i prezime studenta, u drugom broj indeksa u formatu Broj/GodinaUpisa i u trećem prosek studenta. Nakon čitanja iz datoteke, upisati sve podatke o studentima iz liste u novu binarnu datoteku i to tako da se upisuju samo podaci o studentima sa prosekom većim od 8.5.

pj_lista.h

```
#define TRUE 1
#define FALSE 0

typedef enum { lsBOLP = 0, lsEOLP, lsCURRENT } TListStatus;

struct SListElement             /* element liste */
{
  void *pData;       /* pointer u kome se skladisti adresa podatka */
  struct SListElement *pNext;   /* pokazivac na sledeci element u listi */
};

struct SList
{
  TListStatus lsStatus;           /* status - trenutni polzaj u listi */
  struct SListElement *pHead;   /* pokazivac na glavu liste */
  struct SListElement *pCurrent;  /* pokazivac na tekuci element */
  void (*destructor)(void *); /* pokazivac na funkciju koja ima argument tipa void*   */
};

typedef struct SListElement TListElement;
typedef struct SList TList;

void initialize_list(TList *pList, void (*destructor)(void *));
void find_bolp(TList *pList);
int move_forward(TList *pList);
int insert(TList *pList, void *pData);
void *get(TList *pList);
void destroy_list(TList *pList);
int eolp(TList *pList);
```

pj_lista.c

```
#include <stdlib.h>
#include "pj_lista.h"

/* Argumenti: pList - pointer na promenljivau tipa Tlist, destructor - pointer na
   funkciju koja se poziva u toku unistavanja liste za svaki njen podatak */
void initialize_list(TList *pList, void (*destructor)(void *))
{
  pList->pHead = NULL;
  pList->lsStatus = lsBOLP;
  pList->pCurrent = NULL;
  pList->destructor = destructor;
}
```

```
void find_bolp(TList *pList)
{
  pList->lsStatus = lsBOLP;
  pList->pCurrent = NULL;
}

int move_forward(TList *pList)
{
  if (pList->lsStatus == lsEOLP)
    return FALSE;

  if (pList->lsStatus == lsBOLP)
  {
    pList->pCurrent = pList->pHead;
    if (pList->pCurrent == NULL)
      pList->lsStatus = lsEOLP;
    else
      pList->lsStatus = lsCURRENT;
  }
  else
  {
    pList->pCurrent = pList->pCurrent->pNext;

    if (pList->pCurrent == NULL)
      pList->lsStatus = lsEOLP;
  }

  return TRUE;
}

/* pList - pokazivac na tip Tlist, pData - podatak koji ce biti ubacen u listu. */
int insert(TList *pList, void *pData)
{
  TListElement *pNewElem;

  if (pList->lsStatus == lsEOLP)
    return FALSE;

  pNewElem = (TListElement *)calloc(1, sizeof(TListElement));

  if (pNewElem == NULL)
    return FALSE;

  pNewElem->pData = pData;
  pNewElem->pNext = NULL;

  if (pList->lsStatus == lsBOLP)
  {
    pNewElem->pNext = pList->pHead;
    pList->pHead = pNewElem;
    pList->pCurrent = pNewElem;
    pList->lsStatus = lsCURRENT;
  }
  else
  {
    pNewElem->pNext =  pList->pCurrent->pNext;
    pList->pCurrent->pNext = pNewElem;
    pList->pCurrent = pNewElem;
  }
  return TRUE;
}

void *get(TList *pList)
{
  if (pList->lsStatus == lsCURRENT)
    return pList->pCurrent->pData;
```

```
  return NULL;
}
void destroy_list(TList *pList)
{
  TListElement *pElem, *pHelpElem;
  for (pElem = pList->pHead; pElem != NULL; )
  {
    pHelpElem = pElem->pNext;
    if (pList->destructor)
      (*pList->destructor)(pElem->pData); /* na ovom mestu pointer na funkciju */
             /* se dereferencira i zatim se poziva ta funkcija */
             /* sa argumentom pElem->pData */
    free(pElem);
    pElem = pHelpElem;
  }
}

int eolp(TList *pList)
{
  return pList->lsStatus == lsEOLP;
}
```

cz122.c

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include "pj_lista.h"

typedef struct {   /* struktura u kojoj se cuvaju podaci o */
  char strIme[256];    /* studentu */
  int nGodUpisa;
  int nBroj;
  float fProsek;
} TStudent;

void UnistiStudenta(void *pStudent)        /* funkcija neophodna za call-back */
{                         /* pri unistavanju liste */
  if (pStudent != NULL)
  {
    printf("UnistiStudenta %s\n", ((TStudent*)pStudent)->strIme);
    free(pStudent);
  }
}

void main()
{
  FILE *f1, *f2;
  TList list;
  TStudent *pStudent;
  char *pKrajLinije;

  f1 = fopen("studenti.txt", "r"); /* otvaramo tekstualnu datoteku za citanje */
  if (f1 == NULL)
  {
    printf("Datoteka ne postoji!\n"); /* otvaramo binarnu datoteku za upis */
    return;
  }

  f2 = fopen("studenti.bin", "wb");
  if (f2 == NULL)
  {
    printf("Datoteka za upis nece da se otovori!\n");
    fclose(f1);
    return;
  }
```

```
  initialize_list(&list, UnistiStudenta);
  /* identifikator funkcije je ujedno i pokazivac na funkciju */

  while (!feof(f1))
  {
    pStudent = (TStudent *)calloc(sizeof(TStudent), 1);
    fgets(pStudent->strIme, 255, f1);
/*
Funkcija fgets cita string sve do kraja linije, ali ukljucuje i kraj linije u string!
Zato cemo koristiti strchr funkciju koja vraca adresu prvog karaktera koji se zadaje kao
drugi argument funkcije u okviru stringa koji se zadaje kao prvi argument funkcije.
Ako karakter nije pronadjen strchr vraca NULL. Kada pronadjemo karakter za kraj linije,
prosto cemo ga izbaciti postavljajuci ga na '\0' sto oznacava kraj stringa.
*/
    if ((pKrajLinije = strchr(pStudent->strIme, '\n')) != NULL)
      *pKrajLinije = '\0';

    fscanf(f1, "%d/%d \n", &pStudent->nBroj, &pStudent->nGodUpisa);
    fscanf(f1, "%f \n", &pStudent->fProsek);
    if (!insert(&list, (void *)pStudent))
    {
      destroy_list(&list);
      printf("Greska pri ubacivanju u listu!\nProgram prekinut!\n");
      return;
    }
  }

  fclose(f1);
  find_bolp(&list);
  while (TRUE)
  {
    move_forward(&list);
    if (eolp(&list))
      break;
    pStudent = (TStudent*)get(&list);
    printf("Ime i prezime: %s\n", pStudent->strIme);
    printf("Broj indeksa:  %d/%02d\n", pStudent->nBroj, pStudent->nGodUpisa);
    printf("Prosek:        %4.2f\n", pStudent->fProsek);
    if (pStudent->fProsek > 8.5)
    {
      if (fwrite(pStudent, sizeof(TStudent), 1, f2) == 1)
        printf("Kandidat uspesno upisan u novi fajl...\n\n");
      else
      {
        printf("Greska pri upisu! Program prekinut.\n");
        break;
      }
    }
    else
      printf("Kandidat nije zadovoljio uslove za upis u novi fajl!\n\n");
  }

  fclose(f2);
  destroy_list(&list);
  printf("** KRAJ **\n");
}
```

Zadatak C125

Pod pretpostavkom da su zapisi u datoteci uređeni u leksikografski uređeno stablo, šta ispisuje dati program za zadatu datoteku (prvi argument komandne linije) i zadati ključ (drugi argument)?

```
#include <stdio.h>
#include <string.h>
struct tree_elem
{
  char key[7];
  unsigned long left, right;
};
void create(char* filename)
{
  struct tree_elem buffer;
  FILE *file = fopen(filename, "wb");
  while (!ferror(file))
  {
    printf("\nKEY LEFT RIGHT: ");
    scanf("%6s %lu %lu", buffer.key, &buffer.left, &buffer.right);
    if (strcmp(buffer.key, "END") == 0) break;
    fwrite(&buffer, sizeof(struct tree_elem), 1, file);
  }
  fclose(file);
}
void traverse( FILE *file, char key[], unsigned long node)
{
  struct tree_elem buffer;
  static int ukupno_aktivacija = 0;
  int ova_akt;
  printf("Pocetak aktivacije %d\n", ova_akt = ++ukupno_aktivacija);
  if ( node == 0 )
    { printf("Kraj aktivacije %d\n", ova_akt); return;   }
  fseek( file, ( node - 1 ) * sizeof( struct tree_elem ), SEEK_SET);
  fread( &buffer, sizeof(struct tree_elem ), 1, file );
  if ( ferror( file ) ) return;
  traverse( file, key, buffer.left );
  if ( strcmp( key, buffer.key ) >= 0)
  {
    printf( "%s\n", buffer.key );
    traverse( file, key, buffer.right );
  }
  printf("Kraj aktivacije %d\n", ova_akt);
}
void main( int argc, char *argv[] )
{
  FILE *file;                    /* ovako odredjujemo da li kreiramo datoteku ili */
  if (argc > 3) create(argv[1]); /* koristimo vec napravljenu nekom ranijom prilikom */
  if ( ( file = fopen( argv[1], "rb" ) ) == NULL )
    fprintf( stderr, "Greska pri otvaranju datoteke %s\n", argv[1] );
  else
    traverse( file, argv[2], 1 );
}
```

A) ključeve svih zapisa u datoteci u leksikografskom poretku
B) ključeve svih zapisa manjih do jednakih zadatom, u leksikografskom poretku
C) ključeve svih zapisa manjih do jednakih zadatom, u obrnutom leksikografskom poretku

Odgovor

B

Komentar

Promenljive **ukupno_aktivacija** i **ova_akt** služe za pojašnjavanje rednog broja aktivacije posmatrane rekurzivne funkcije **traverse**, i nemaju uticaja na suštinu zadatka.

Zadatak CI-2007-Jan-2

Napisati potprogram **`unsigned in zbir (char *a, char *b)`** na programskom jeziku C koji određuje decimalnu vrednost zbira brojeva **a** i **b**. Odgovarajući stvarni argumenti su znakovni nizovi (stringovi) od maksimalno 3 znaka i predstavljaju pozitivne cele brojeve u heksadecimalnom brojnom sistemu. Pri tome, znak koji predstavlja cifru najveće težine je prvi u znakovnom nizu. Napisati glavni program na programskom jeziku C koji sa standardnog ulaza učitava dva niza od po 3 znaka koji treba da predstavljaju heksadecimalne cifre, potom poziva potprogram **`zbir`** i na standardnom izlazu ispisuje rezultat zbira ta dva broja.

Nemojte NIKAD programski kod učiti napamet.

```
#include <stdio.h>
unsigned int zbir (char *a, char *b)
{
unsigned int vred_a = 0, vred_b = 0;
      while (*a)
      {
            vred_a *= 16;
            if (*a >= '0' && *a <= '9') vred_a += (*a - '0');
            else if (*a >= 'a' && *a <= 'f') vred_a += (*a - 'a' + 10);
            else if (*a >= 'A' && *a <= 'F') vred_a += (*a - 'A' + 10);
            a++;
      }
      sscanf(b, "%x", &vred_b);
      printf("%d\t%x\n", vred_a, vred_a); /* nije traženo u postavci */
      printf("%d\t%x\n", vred_b, vred_b); /* nije traženo u postavci */
      return vred_a + vred_b;
}
```

Svako ponavljanje bez razumevanja programskog koda je štetno i na ispitu će biti kažnjavano oduzimanjem poena.

```
main()
{
char a[4], b[4];
      printf("Unesite brojeve: ");
      scanf("%s%s", a, b);
      printf("Zbir ovih brojeva je %d\n", zbir(a, b));
}
```

Komentar

U funkciji **`zbir()`** su prikazana dva različita pristupa dobijanja vrednosti broja zapisanog u heksadecimalnom brojnom sistemu. Najpre se vrši konverzija broja a, tako što se u ciklusu analizira znak po znak, sve dok se ne dođe do kraja znakovnog niza. Primetiti da nigde nije pretpostavljena dužina nizova a i b, pa je ovo rešenje univerzalno (u onoj meri u kojoj opseg predstavljanja celih brojeva dozvoljava). Za praktične primene, ovom rešenju nedostaje provera da li znakovni nizovi a i b zaista predstavljaju cele brojeve u heksadecimalnom zapisu. To se, međutim, nije tražilo - pa nije ni prikazano u rešenju. Nakon ciklusa za konverziju broja a, vrši se konverzija broja b pozivom funkcije **`sscanf()`**. Ova funkcija se ponaša slično scanf() odnosno fscanf(), s tim što je izvor koji treba čitati u datom formatu zadat znakovnim nizom koji se zadaje kao prvi argument. U ovom slučaju radi se o nizu b. Naziv **`sscanf()`** zapravo potiče *string scanf*. U ostalim aspektima, može se smatrati da se ova funkcija ponaša kao **`scanf()`**.

Zadatak CI-2006-Okt-1

Napisati program na programskom jeziku C koji izračunava broj osvojenih bodova na šampionatu trka Formule 1. Podaci o vozačima i timovima se nalaze u tekst datoteci vozaci.txt prema sledećem formatu: u svakom redu datoteke redom su zapisani prezime i ime vozača i naziv tima za koji vozač vozi (svako polje po 30 karaktera najviše). Poznato je da ima tačno 28 vozača koji učestvuju u trkama i da se na šampionatu vozi 20 trka. Podaci o rezultatima trka smešteni su tekst datoteci trke.txt tako da se u svakom redu datoteke nalaze redni brojevi pozicija jednog vozača po završetku svake trke (20 pozicija, za svaku trku po jedna). U slučaju da vozač ne prođe cilj (odustane od trke), broj pozicije za tu trku je 0. Podaci u prvom redu datoteke trke.txt odgovaraju vozaču iz prvog reda datoteke vozaci.txt, itd. Bodovi za vozače za jednu trku se računaju na sledeći način: prvih 8 vozača dobija 10, 8, 6, 5, 4, 3, 2 i 1 bod respektivno, ostali 0 bodova. Program treba da izračuna i na standardnom izlazu ispiše osvojene bodove tako da u jednom redu stoje prezime i ime vozača, osvojen broj bodova i naziv tima.

```
#include <stdio.h>
main() {
FILE *vozaci, *trke;
char prezime[31], ime[31], tim[31];
int v,p,pozicija,poena;

       vozaci = fopen("vozaci.txt", "r");
       trke = fopen("trke.txt", "r");
       if( ! vozaci || ! trke ) {
              if( ! vozaci ) printf("Neuspelo otvaranje datoteke vozaci.txt\n");
              else           fclose(vozaci);
              if( ! trke )   printf("Neuspelo otvaranje datoteke vozaci.txt\n");
              else           fclose(trke);
              exit(0);
       }
       for(v=0; v<28; v++) {
              fscanf(vozaci, "%s %s %s", prezime, ime, tim);
              poena = 0;
              for(p=0; p < 20; p++) {
                     fscanf(trke, "%d", &pozicija);
                     switch(pozicija) {
                            case 1: poena += 10; break;
                            case 2: poena += 8; break;
                            case 3: poena += 6; break;
                            case 4: poena += 5; break;
                            case 5: poena += 4; break;
                            case 6: poena += 3; break;
                            case 7: poena += 2; break;
                            case 8: poena += 1; break;
                     }
              }
              printf("%s %s %d %s\n", prezime, ime, poena, tim);
       }
       fclose(vozaci);
       fclose(trke);
}
```

Nemojte NIKAD programski kod učiti napamet.

Svako ponavljanje bez razumevanja programskog koda je štetno i na ispitu će biti kažnjavano oduzimanjem poena.

Komentar

Provera da li su datoteke uspešno otvorene nije najjezgrovitije zapisana, ali je verovatno najpribližnija načinu na koji se razmišlja prilikom donošenja odluke. Najpre je izvršeno otvaranje obe datoteke. Ako bilo koja od njih nije uspešno otvorena, potrebno je prekinuti program, a prethodno prethodno ispisati koja zapravo nije uspešno otvorena (moguće – obe). Za onu koja je uspešno otvorena se ne vrši ispisivanje već njeno zatvaranje. Iako obaveštenje o nastalim greškama pri radu nije traženo u postavci, ovde je to prikazano iz edukativnih razloga.

Zadatak CI-2007-Okt-2

Napraviti program na programskom jeziku C koji obrađuje datoteku koja sadrži izvorni program na programskom jeziku C. Potrebno je datoteku prepisati u novu datoteku, tako da se izostave svi komentari i očuva uređenost teksta po redovima. Primer:

Ulazna datoteka	Odgovarajuća izlazna datoteka

`#include <stdio.h>`
`/* ispis proizvoda i kolicnika dva broja`
`*/`
`main()`
`{`
`int a, b;`
`scanf("%d%d", &a, &b);`
`printf("%d", a*b); /* ispis a*b */`
`/* ispis a/b */ printf("%d", a/b);`
`}`
	`#include <stdio.h>`
	`main()`
	`{`
	`int a, b;`
	`scanf("%d%d", &a, &b);`
	`printf("%d", a*b);`
	`printf("%d", a/b);`
	`}`

Imena ulazne i izlazne datoteke se zadaju kao prvi i drugi parametar komandne linije i imaju najviše 30 znakova. U slučaju da dođe do greške pri otvaranju neke od datoteka, ispisati poruku o greški i prekinuti izvršavanje programa.

```
#include <stdio.h>
int main(int argc, char *argv[]) {
char znak;
enum StanjeObrade { VAN_K, POC_K, UNUTAR_K, KRAJ_K } stanje = VAN_K;
FILE *ul, *izl;
	if (NULL == (ul = fopen(argv[1], "r"))) {
		fprintf(stderr, "Doslo je do greske pri otvaranju %s\n", argv[1]);
		return 1;
	}
	if (NULL == (izl = fopen(argv[2], "w"))) {
		fclose(ul);
		fprintf(stderr, "Doslo je do greske pri otvaranju %s\n", argv[2]);
		return 2;
	}
	while ((znak = fgetc(ul)) != EOF) {
		switch(stanje) {
			case VAN_K:
				if ('/' == znak) stanje = POC_K;
				else fputc (znak, izl);
				break;
			case POC_K:
				if ('*' == znak) stanje = UNUTAR_K;
				else stanje = VAN_K, fputc('/', izl), fputc(znak, izl);
				break;
			case UNUTAR_K:
				if ('*' == znak) stanje = KRAJ_K;
				break;
			case KRAJ_K:
				if ('/' == znak)  stanje = VAN_K;
				else                   stanje = UNUTAR_K;
				break;
		}
	}
	fclose(ul);
	fclose(izl);
	return 0;
}
```

Nemojte NIKAD programski kod učiti napamet.

Svako ponavljanje bez razumevanja programskog koda je štetno i na ispitu će biti kažnjavano oduzimanjem poena.

Komentar

Zadatak se bavi nešto složenijom obradom ulazne datoteke u odnosu na jednostavno prepisivanje podataka. U programu se uvodi pojam "stanja" na osnovu kojeg se odlučuje kako treba tretirati ulazni tekst. Na primer, ako se trenutna obrada vrši unutar komentara, ništa se ne prepisuje u izlaznu datoteku i obrnuto. S obzirom na to da se početak i kraj komentara označava pomoću dva karaktera (/ i *), uvodi se četiri stanja: **van komentara**, **početak komentara**, **unutar komentara** i **kraj komentara**. U stanju **van komentara**, svi znaci se prepisuju, osim kada se pojavi znak /. Tada je potrebno pročitati naredni znak i utvrditi da li je on * (počinje komentar) ili nije (treba / poslati u izlaznu datoteku). Dakle, kada u stanju **van komentara** naiđe znak /, potrebno je preći u novo stanje, u kojem se očekuje dolazak znaka *. Slično se vrši izlazak iz komentara.

stderr je izlazni uređaj koji se automatski stvara i koji služi kao "kanal" za ispis grešaka. Njegovo korišćenje nije zahtevano u postavci zadatka – njegova uloga u ovom rešenju je edukativna.

Zadatak CI-2006-Sep-2

Napisati program koji jednu tekst datoteku prepisuje u drugu tekst datoteku, uz očuvanje uređenosti teksta po linijama. Linije u kojima postoji neparan broj znakova razmaka se prepisuju tako da sva slova budu mala, a ostale tako da sva slova budu velika. Linije ne mogu biti duže od 80 znakova. Imena tekst datoteka se zadaju putem standardnog ulaza; ne mogu biti duža od 30 znakova. U slučaju da ne uspe otvaranje neke od datoteka, ispisati poruku korisniku u kojoj se navodi sa kojom datotekom je došlo do problema i prekinuti izvršenje programa.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
main() {
char ime_ul_dat[30+1], ime_izl_dat[30+1], linija[80+1+1];
FILE *ul_dat, *izl_dat;
int duz_lin, i, br_raz;

	scanf("%s", ime_ul_dat);
	scanf("%s", ime_izl_dat);
	ul_dat = fopen(ime_ul_dat, "r");
	if (NULL == ul_dat) {
		fprintf(stderr, "Doslo je do greske pri radu sa %s\n", ime_ul_dat);
		exit(EXIT_FAILURE);
	}
	izl_dat = fopen(ime_izl_dat, "w");
	if (NULL == izl_dat) {
		fclose(ul_dat);
		fprintf(stderr, "Doslo je do greske pri radu sa %s\n", ime_izl_dat);
		exit(EXIT_FAILURE);
	}
	while (fgets(linija, 82, ul_dat) != NULL) {
		duz_lin = strlen(linija);
		br_raz = 0;
		for (i = 0; i < duz_lin; i++)
			br_raz += isspace(linija[i]) != 0;
		for (i = 0; i < duz_lin; i++)
			if (br_raz % 2)
				linija[i] = tolower(linija[i]);
			else
				linija[i] = toupper(linija[i]);
		fputs(linija, izl_dat);
	}
	fclose(ul_dat);
	fclose(izl_dat);
}
```

Nemojte NIKAD programski kod učiti napamet.

Svako ponavljanje bez razumevanja programskog koda je štetno i na ispitu će biti kažnjavano oduzimanjem poena.

Komentar

U postavci je rečeno da imena datoteka nisu duža od 30 znakova. Zbog toga se kreiraju dva niza dužine 31 znak, gde je namerno naglašeno da se rezerviše prostor za jedan znak više od maksimalne dužine zbog znaka za kraj znakovnog niza ('\0'). Za liniju se rezerviše jedno mesto više: pod pretpostavkom da niz može imati najviše 80 znakova, potrebno je još jedno mesto za kraj znakovnog niza **i još jedno mesto za eventualni znak za novi red** koji će funkcija **fgets**() smestiti u niz *linija*. Kasnije u programu se funkciji fgets() zadaje vrednost 82 kao jedan od argumenata. To je uputstvo da je za čitanje rezervisano tačno 82 znaka, od kojih jedan mora biti znak za kraj znakovnog niza.

Simbol EXIT_FAILURE je definisan u zaglavlju *stdlib.h* i služi kao argument funkciji **exit**() za označavanje prekida programa zbog greške. U postavci se to nije tražilo, ali je u rešenju prikazano iz edukativnih razloga.

Funkcija **isspace**() određuje da li je dati znak razmak, a funkcije **tolower**() i **toupper**() konvertuju dati znak u malo odnosno veliko slovo respektivno. Ni njihova upotreba nije tražena u postavci. Osim iz edukativnih razloga, ovde su prikazane jer povećavaju čitljivost programa, pa se njihova upotreba preporučuje.

Zadatak CI-2006-Jan-1

U cilju proglašenja najboljeg igrača(MVP) na odbojkaškom turniru, organizacioni odbor turnira je doneo sledeću odluku o načinu bodovanja učinka igrača: poen iz servisa vredi 1 poen, poen iz smeća vredi 0.5 poena, a blokada 0.2 poena. Podaci o učinku svih igrača za vreme celog turnira nalaze se u tekstualnoj datoteci **odbojka.txt**. U jednom redu datoteke nalaze se podaci za jednog igrača. Za svakog igrača se pamti njegovo ime i prezime (niz od tačno 30 karaktera), visina (ceo broj), težina (ceo broj), broj ostvarenih poena iz servisa na turniru (ceo broj), broj poena iz smeća (ceo broj) i broj blokada (ceo broj). Napisati program na programskom jeziku C koji iz datoteke **odbojka.txt** čita podatke o igračima i pronalazi prvog igrača sa najvećim brojem ostvarenih poena i na standardnom izlazu ispisuje ime tog igrača i broj ostvarenih poena.

Rešenje sa fgets

```
#include<stdio.h>
#include<string.h>
main() {
double poen_serv = 1.0, poen_smec = 0.5, poen_blok = 0.2;
double max_poeni = 0.0, poeni;
char ime_i_prez[31], max_ime_i_prez[31];
unsigned visina, tezina, br_serv, br_smec, br_blok;
FILE *ul_dat = fopen("odbojka.txt", "r");

	if( ul_dat == NULL )
		exit(0);

	while (fgets(ime_i_prez, 31, ul_dat) != NULL) {
		fscanf(ul_dat, "%d%d%d%d%d", &visina, &tezina, &br_serv, &br_smec, &br_blok);
		poeni = poen_serv * br_serv + poen_smec * br_smec + poen_blok * br_blok;
		if (max_poeni < poeni) {
			max_poeni = poeni;
			strcpy(max_ime_i_prez, ime_i_prez);
		}
	}

	fclose(ul_dat);

	if (max_poeni > 0.0)
		printf("MVP je %s, sa ostvarenih %lf poena\n", max_ime_i_prez, max_poeni);
}
```

Nemojte NIKAD programski kod učiti napamet.

Svako ponavljanje bez razumevanja programskog koda je štetno i na ispitu će biti kažnjavano oduzimanjem poena.

Komentar

Ovaj zadatak je rešen na dva veoma slična načina. U rešenju prikazanom na ovoj strani koristi se funkcija **fgets**() da pročita ime i prezime igrača kao jednu celinu (tj. kao jedan red teksta) i smesti ih u znakovni niz **ime_i_prez**.

Drugo rešenje, prikazano na sledećoj strani, koristi funkciju **fscanf**() za čitanje imena i prezimena kao dva nezavisna znakovna niza, zbog čega je naknadno potrebno izvršiti njihovo spajanje u jedan znakovni niz: najpre se pročita ime i smesti u znakovni niz **ime_i_prez**. Nakon toga se u poseban niz **prezime** smešta pročitano prezime. Zatim se na ranije pročitano ime dodaje jedan znak razmaka a posle toga i pročitano prezime. Dodavanje jednog znakovnog niza na kraj drugog se vrši funkcijom **strcat**().

Treba primetiti sledeće. Očekivano da oba rešenja daju isti rezultat u smislu čitanja imena i prezimena. Ipak, ne mora biti tako: ako ulazna datoteka sadrži više od jednog razmaka između imena i prezimena, ti razmaci će ostati nakon čitanja kada se koristi funkcija **fgets**(). To se neće desiti kada se za čitanje imena i prezimena koristi funkcija **fscanf**() jer ona automatski ignoriše sve razmake koji razdvajaju dva niza znakova.

Resenje sa fscanf

```
#include<stdio.h>
#include<string.h>
main() {
double poen_serv = 1.0, poen_smec = 0.5, poen_blok = 0.2;
double max_poeni = 0.0, poeni;
char ime_i_prez[31], prezime[31], max_ime_i_prez[31];
unsigned visina, tezina, br_serv, br_smec, br_blok;
FILE *ul_dat = fopen("odbojka.txt", "r");

	if( ul_dat == NULL )
		exit(0);

	while( fscanf(ul_dat, "%s", ime_i_prez) != EOF) {
		fscanf(ul_dat, "%s", prezime);
		strcat(ime_i_prez, " ");
		strcat(ime_i_prez, prezime);
		fscanf(ul_dat, "%d%d%d%d%d", &visina, &tezina, &br_serv, &br_smec, &br_blok);
		poeni = poen_serv * br_serv + poen_smec * br_smec + poen_blok * br_blok;
		if (max_poeni < poeni) {
			max_poeni = poeni;
			strcpy(max_ime_i_prez, ime_i_prez);
		}
	}

	fclose(ul_dat);

	if (max_poeni > 0.0)
		printf("MVP je %s, sa ostvarenih %lf poena\n", max_ime_i_prez, max_poeni);
}
```

Nemojte NIKAD programski kod učiti napamet.

Komentar

Videti komentar dat uz prethodno rešenje zadatka.

Svako ponavljanje bez razumevanja programskog koda je štetno i na ispitu će biti kažnjavano oduzimanjem poena.

Zadatak CI-2006-Jan-2

Napisati potprogram **`double bin2dbl(char * num)`** na programskom jeziku C koji na osnovu znakovnog niza (stringa) **num** koji predstavlja binarnu reprezentaciju realnog broja izračunava i kao rezultat vraća brojnu vrednost tipa double, koja će se ispisati u decimalnom brojnom sistemu. String **num** ima najviše 10 binarnih cifara uključujući i decimalnu tačku koja može a ne mora postojati. Napisati glavni program na programskom jeziku C koji sa standardnog ulaza učitava string **num,** poziva potprograma **bin2dbl** i na standardnom izlazu ispisuje dobijeni rezultat. (npr. za unos "1011.101" program treba da ispiše rezultat 11.625).

```
#include <stdio.h>
#include <string.h>
double bin2dbl(char *num) {
unsigned num_length = strlen(num), i = 0;
double celi_deo = 0, razl_deo = 0;

	/* ako prvi znak nije decimalna tacka, računa se celi deo broja */
	if (num[i] != '.') {
		celi_deo = num[0] - '0';
		/* dok se ne stigne do kraja broja ili do decimalne tacke */
		while (++i < num_length && num[i] != '.')
			celi_deo = celi_deo * 2 + (num[i] - '0');
	}

	/* ako postoji razlomljeni deo, kreće se od kraja */
	if (i < num_length)
	/* dok se ne stigne do decimalne tacke */
		while (i < --num_length)
			razl_deo = (razl_deo + num[num_length] - '0')/2;

	return celi_deo + razl_deo;
}

main() {
char ul_num[11];
	printf("Unesite binarni broj koji treba prevesti u decimalni brojni sistem: ");
	scanf("%s", ul_num);
	printf("%7.3lf\n", bin2dbl(ul_num));
}
```

Nemojte NIKAD programski kod učiti napamet.

Svako ponavljanje bez razumevanja programskog koda je štetno i na ispitu će biti kažnjavano oduzimanjem poena.

Komentar

Tražena funkcija izračunava vrednost realnog broja zapisanog u binarnom brojnom sistemu i od dobijene vrednosti pravi podatak tipa double. Obrada se vrši na sledeći način: ako niz ne počinje znakom '.', onda je u pitanju celobrojni deo tog realnog broja. Pravi se ciklus koji se ponavlja sve dok se ne obrade svi znakovi u nizu (slučaj kada je zadat ceo broj) ili dok se ne naiđe na decimalnu tačku (realni broj sa razlomljenim delom). Do nastavka funkcije se dakle stiže u tri situacije: niz je počeo decimalnom tačkom ili obrada niza je završena ili tokom obrade celobrojnog dela se stiglo do decimalne tačke. S obzirom na to da se u slučaju prve i treće situacije postupa isto (određuje se razlomljeni deo), najpre se eliminiše druga situacija (završena obrada) proverom da li je i manje od num_length. Ako je taj uslov ispunjen, u novom ciklusu se određuje vrednost razlomljenog dela, s tim što obrada ide od kraja niza, zbog prirode problema (krajnje desna pozicija je pozicija cifre najmanje težine).

Ova funkcija nije prilagođena praktičnoj upotrebi, jer bi to zahtevalo da se pre određivanja vrednosti broja izvrši provera da li se dati znakovni niz zaista sastoji od cifara 0 i 1 i najviše jednog decimalnog zareza. Preporučuje se pisanje funkcije koja to utvrđuje za samostalnu vežbu.

Zadatak CI-2006-Sep-1

Na programskom jeziku C napisati potprograme i glavni program koji vrše obradu niza realnih brojeva. Unos podataka, obrada, ispis rezultata i oslobađanje memorije se vrši u potprogramima. Potprogram za unos sa standardnog ulaza učitava broj elemenata niza realnih brojeva tipa **float**, alocira potrebnu dinamičku memoriju sa smeštanje tog niza i učitava elemente niza. Ovaj potprogram treba da proveri uspešnost poziva funkcije za dodelu dinamičke memorije i u slučaju neuspeha prekida izvršavanje programa. Potprogram za obradu, ispis i oslobađanje memorije štampa sve brojeve iz unetog niza koji su u opsegu +/- 10% od vrednosti aritmetičke sredine niza. Nakon ispisa, potprogram oslobađa dinamički alociranu memoriju. Napisati glavni program koji redom poziva opisane potprograme. Potprogrami sa glavnim programom smeju razmenjivati podatke samo preko liste argumenata i povratne vrednosti.

```
#include <stdio.h>
#include <stdlib.h>
#define FAILURE 0
#define SUCCESS 1
int Unos(float **niz_Adr, int *br_el_Adr) {
float *niz;
int i, br_el;
	printf("Unesite broj elemenata: ");
	scanf("%d", &br_el);
	niz = calloc(br_el, sizeof(float));
	if (NULL == niz) return FAILURE;
	for (i = 0; i < br_el; i++)
		scanf("%f", &niz[i]);
	*niz_Adr = niz;
	*br_el_Adr = br_el;
	return SUCCESS;
}
void ObradaIspisDealociranje(float *niz, int br_el) {
float arit_sred = 0.0;
int i;
	for(i = 0; i < br_el; i++) arit_sred += niz[i];
	if (0 != br_el) arit_sred /= br_el;
	printf("%f\n", arit_sred);
	for(i = 0; i < br_el; i++)
		if (niz[i] >= 0.9 * arit_sred && niz[i] <= 1.1 * arit_sred)
			printf("%f\t", niz[i]);
	printf("\n");
	free(niz);
}
main() {
float *niz = 0;
int br_el;
	if (Unos(&niz, &br_el))
		ObradaIspisDealociranje(niz, br_el);
	else
		printf("Greska pri unosu\n");
	niz = 0;
}
```

Nemojte NIKAD programski kod učiti napamet.

Svako ponavljanje bez razumevanja programskog koda je štetno i na ispitu će biti kažnjavano oduzimanjem poena.

Komentar

U postavci se eksplicitno traži da glavni program komunicira sa potprogramima putem argumenata i povratnih vrednosti. Takav pristup u pisanju programa se preporučuje čak i ako se posebno ne naglasi. Takođe, nije navedeno - **ali bi trebalo podrazumevati**, međusobna komunikacija potprograma treba da se odvija na isti način.

Funkcija za unos i alokaciju memorije vraća vrednost celobrojnu vrednost - FAILURE odnosno SUCCES u zavisnosti od uspeha. S obzirom na to da ta funkcija alocira memoriju za niz koji treba učitati – dakle menja vrednost pokazivača, u potprogram se prenosi adresa pokazivača na dati niz brojeva. U suprotnom nikakva izmena ne bi bila vidljiva van potprograma. Kod dealokacije memorije, ne prosleđuje se adresa pokazivača jer se ne očekuje da se promena vrednosti tog pokazivača vidi sa mesta poziva (u ovom slučaju iz glavnog programa). Ipak, u toj situaciji programer treba da vodi računa da nakon poziva funkcije za dealokaciju memorije postavi vrednost to pokazivača na 0. U ovom zadatku to nije neophodno jer se taj pokazivač više ne koristi (kraj programa), ali je dobra praksa.

Zadatak C2008-A1

Šta ispisuje sledeći program na programskom jeziku C, ukoliko je pozvan sa **`prog 3 5 7 8`** ?

```
#include <stdio.h>
#include <stdlib.h>
union { struct { int a,b;} c; struct { int a,b,c;} d; } e;
void main(int argc, char* argv[])
{
  int s=atoi(argv[1]), br = atoi(argv[2]);

  for(e.c.a=0;e.c.a<br;e.c.a++)
    for(e.d.a=0;e.d.a<5;e.d.a++)
      for(e.c.a=0;e.c.a<5;e.c.a++)
        for(e.c.b=0;e.c.b<5;e.c.b++) s++;

  printf("%d", s);
}
```

A) 30 (**B**) 28 C) 628

Komentar

S obzirom na to da je kod elemenata unije memorijski prostor deljen, polja a i b strukture c dele iste lokacije sa poljima a i b strukture d. Drugim rečima, svaka promena nad poljima a i b strukture c istovremeno predstavlja i promenu nad poljima a i b strukture d i obrnuto.

Na početku promenljiva s dobija vrednost 3 a br 5. Dakle, spoljni ciklus (e.c.a) se na prvi pogled ponavlja 5 puta. Medjutim, vec naredni unutrašnji ciklus (e.d.a) se vrši nad istim podatkom, tako da kada on bude završen, e.d.a će imati vrednost 5, a to automatski važi i za e.c.a. Sledeći for ciklus se odvija ponovo nad e.c.a. Na osnovu prethodnog zaključka, kada on bude završen, automatski će biti ispunjen uslov za prekid prva dva ciklusa. Prema tome, "koristan efekat" u programu imaju naredna dva ugneždena ciklusa. Ukupan broj izvršavanja operacije s++ je zbog toga 25. Kako je početna vrednost s bila 3, konačna vrednost u trenutku ispisivanja će biti 28.

Zadatak C2008-S11

Napisati program na programskom jeziku C koji sa standardnog ulaza čita znak po znak do kraja linije. Znakovi se čitaju jedan po jedan. U početku se dinamički alocira niz od 10 elemenata. Nadalje, svaki put kada u nizu treba više mesta, niz se proširi za još 10 znakova.

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

void main(){
  char *p=NULL, c;
  int i = 0;
  while( (c=getchar()) != '\n') {
    if (i%10 == 0) p = realloc( p, (i+10) * sizeof(char) );
    p[i++] = c;
  }
  if (i%10 == 0) p = realloc( p, (i+1) * sizeof(char) );
  p[i++] = '\0';
  printf("%s\n",p);
  free(p);
}
```

Svako ponavljanje bez razumevanja programskog koda je štetno i na ispitu će biti kažnjavano oduzimanjem poena.

Zadatak C2008-S12

Napisati segment programa na programskom jeziku C koji menja dimenziju kvadratne matrice sa staroN na N. Ako se matrica smanji, tada elementi koji ostanu u novoj matrici treba da budu neizmenjeni u odnosu na staru matricu. Ako se matrica poveća, elementi koji su postojali u staroj matrici treba da ostanu neizmenjeni.

```
if (N < staroN) {
  for (j = 1; j < N; j++)
    for (i = 0; i < N; i++) m[j*N+i] = m[j*staroN+i];
}
if (N != staroN) m = realloc(m,N*N*sizeof(unsigned int));
if (N > staroN){
  for (j = staroN-1; j > 0; j--)
    for (i = staroN-1; i >= 0; i--)
      m[j*N+i] = m[j*staroN+i];
}
```

Zadatak C2008-S21

Napisati program na programskom jeziku C koji sa standardnog ulaza čita znak po znak do kraja linije. Znakovi se čitaju jedan po jedan. U početku se dinamički alocira niz od 10 elemenata. Nadalje, svaki put kada u nizu više nema mesta, niz se proširi za još 10 znakova. U slučaju da realokacija bude neuspešna, završiti čitanje i ispisati ono što može da stane u već alocirani prostor.

Rešenje

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

void main(){
  char *staro_p=NULL, *novo_p, c;
  int i = 0;
  while( (c=getchar()) != '\n') {
    if (i%10 == 0) {
      novo_p = realloc( staro_p, (i+10) * sizeof(char) );
      if (novo_p == NULL) {
        printf("Neuspesno realociranje\n");
        break;
      }
      else staro_p = novo_p;
    };
    staro_p[i++] = c;
  }
  if (i%10 == 0) {
    novo_p = realloc( staro_p, (i+1) * sizeof(char) );
    if (novo_p == NULL) {
      printf("Neuspesno realociranje\n");
      i--;
    }
    else staro_p = novo_p;
  };
  staro_p[i] = '\0';
  printf("%s\n",staro_p);
  free(staro_p);
}
```

Nemojte NIKAD programski kod učiti napamet.

Svako ponavljanje bez razumevanja programskog koda je štetno i na ispitu će biti kažnjavano oduzimanjem poena.

Zadatak C2008-S22

Napisati program na programskom jeziku C koji učita broj vrsta i broj kolona i potom učita matricu tih dimenzija. Program potom učita broj kolone koju treba izbaciti i izbacuje tu kolonu iz matrice. Zatim čita redni broj vrste na koji umeće vrstu koju će učitati sa standardnog ulaza.

Rešenje

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

void main(){
  unsigned int **m, **novo_m;
  int M, N, i, j, k;
  scanf("%d %d", &M, &N);
  m = malloc(M * sizeof(unsigned int*));
  for (j = 0; j < M; j++){
    m[j] = malloc( N * sizeof(unsigned int) );
    for (i = 0; i < N; i++)scanf("%u", &m[j][i] );
  }

  printf("Unesite broj kolone koju izbacujete: ");
  scanf("%d", &k);
  if ( k >= 0 && k < N) {
    for (j = 0; j < M; j++){
      for (i = k+1; i < N; i++)
        m[j][i-1] = m[j][i];
      m[j] = realloc( m[j], (N-1) * sizeof(unsigned int) );
    }
    N--;
  }

  for (j = 0; j < M; j++){
    for (i = 0; i < N; i++)
      printf("%d ",m[j][i]);
    printf("\n");
  }
  printf("\n\n");

  printf("Unesite red broj na koji umecete vrstu: ");
  scanf("%d", &k);
  if ( k >= 0 && k < M) {
    m = realloc( m, (M+1) * sizeof(unsigned int*) );
    M++;
    for( j = M-1; j > k; j--)
      m[j] = m[j-1];
    m[k] = malloc( N * sizeof(unsigned int) );
    for( i = 0; i < N; i++)
      scanf("%u", &m[k][i]);
  }

  for (j = 0; j < M; j++){
    for (i = 0; i < N; i++)
      printf("%u ",m[j][i]);
    free(m[j]);
    printf("\n");
  }
  printf("\n\n");
  free(m);
}
```

Nemojte NIKAD programski kod učiti napamet.

Svako ponavljanje bez razumevanja programskog koda je štetno i na ispitu će biti kažnjavano oduzimanjem poena.

Komentar

Zbog čitljivosti programa, u ovom rešenju su izostavljene provere da li je alokacija (realokacija) uspešno izvršena. Pri pisanju bilo kakvog programa ove provere NE SMEJU biti izostavljene. Izostavljanjem tih provera, ponašanje programa u slučaju nedostatka prostora nije predvidivo i može dovesti do katastrofalnih posledica.