INFORMATIKA - Elektrijada 2016
Zadaci sa komentarima, uputstvima i rešenjima

Zadaci

U prvom delu dokumenta, nalaze se zadaci sa objašnjenjima i diskusijom. U tekstu se poziva na knjigu Lasla Krausa (Programski jezik C sa rešenim zadacima), pa je preporuka da se tu potraže objašnjenja, koristeći indeks, sadržaj ili ovde navedenu referencu. Rezultat izvršavanja dat je u sivom polju, ispod čega sledi tekst objašnjenja.

U drugom delu se nalaze opšte preporuke.
Tekst svakog zadatka glasi: Odrediti izlaz sledećeg programa.

Zadatak 1.

```
#include <stdio.h>
#include <stdlib.h>
#define SIZEOF(arr) (sizeof(arr)/sizeof(arr[0]))
#define PrintInt(expr) printf("%s:%d\n",#expr,(expr))
int main() {
  int pot[] = {0001, 0010, 0100, 1000};
  int i;
  for(i=0;i<SIZEOF(pot);i++) PrintInt(pot[i]);
  return 0;
}
```

```
pot[i]:1
pot[i]:8
pot[i]:64
pot[i]:1000
```

Operator `sizeof` primenjen na *statički* alocirani niz kao rezultat daje broj bajtova koje taj niz zauzima. Prema tome, makro `SIZEOF` rezultuje brojem elemenata niza. Makro `PrintInt` koristi `#expr`, čime se, prilikom ekspanzije makroa, na mestu gde je `#expr` naveden, postavlja tekst naveden pri pozivanju makroa, kao string.

Zadatak 2.

```
#include <stdio.h>
#define S(t,m,n) {t p=m; m=n; n=p; b++;}
int b;
void fsj(int d, int g, int a[]) {
  int i,j;
  if (d>=g) return;
  if (d==g-1) {
    if (a[d]>a[g]) S(int,a[d],a[g]);
    return;
  }
  for (i=d, j=g; i<j; i++, j--)
    if (a[i]>a[j]) S(int,a[i],a[j]);
  fsj(j,g,a);
  for (j--; j>=d; j--)
   for (i=j; a[i]>a[i+1]; i++)
     S(int,a[i],a[i+1]);
}
main() {
  int niz[]={4,0,3,2,8,7,6,1,9,5}, i;
  fsj(1,9,niz); printf("%d",b);
  for (i=1; i<5; i++)
  printf(" %d",niz[i]);
}
```

```
11 0 1 2 3
```

Pri ekspanziji makroa `S`, parametar `t` biva zamenjen identifikatorom tipa (literal koji se preda pri pozivu makroa – `int` u našem slučaju). Prema tome, `S` vrši zamenu vrednosti promenljivim čiji se identifikatori navedu na mesto drugog i trećeg parametra pri pozivu, a pritom uvećava vrednost globalne promenljive `b`. Za sada se može pretpostaviti da se time broji broj zamena koje se izvrše putem ovog makroa.

Funkcija `fsj` je rekurzivna funkcija. Rekurzija se ne produbljuje u sledećim situacijama: (1) `d>=g`; (2) `d==g-1`. Iz analize ostatka koda, zaključuje se da je d donji indeks podniza koji pripada nizu `a`, a `g` gornji indeks podniza koji treba obraditi. U drugom slučaju, vrši se zamena elemenata `a[d]`, `a[g]`, u slučaju da je zadovoljena relacija. Dakle, u slučaju dvoelementnog niza, biće uređen nerastuće. Ukoliko je niz duži od 2 elementa, prva `for` petlja vrši zamenu elemenata koji se u podnizu nalaze na simetričnim pozicijama (prvi i poslednji, drugi i pretposlednji itd.). Potom sledi rekurzivni poziv ove funkcije, ali se njome obrađuje *gornja* polovina podniza (indeksi `j..g`). Posle rekurzivnog poziva, sprovodi se bubble sort nad *donjom* polovinom niza `a`. Prema tome, obrada koja se rekurzivno izvrši i nema uticaja na to šta će se naći u donjoj polovini niza `a`, pa je nije potrebno ni "simulirati".

Svrha ovog zadatka je da pokaže da pažljiva analiza pre brute force rada može da uštedi dosta vremena. Naravno, brute force rad mora da se izvrši u nekom trenutku.

Zadatak 3.

```
#include <stdio.h>
enum Roman {I,II,III,V=5,IV=4,VI,VII=VI+1};
main(){
   int rom[3][3]={{I,II,III},{IV,V,VI},{VII}};
   int i,j,s=0,t=0;
   for(i=0; i<2; i++)
   for(j=0; j<3; s+=rom[i][j],j++);
   for(i=0; i<2; i++)
   for(j=0; j<3; t+=*rom[i,j],j++);
   printf("%d\n%d", s, t);
}
```

```
17
20
```

Obrazloženje: Vrednosti enum konstanti `Roman` su sledeće: 0, 1, 2, 5, 4, 5, 6, respektivno redosledu navođenja.

Statička matrica `rom` izgleda kao u priloženoj tabeli.

`I (0)`	`II (1)`	`III (2)`
`IV (4)`	`V (5)`	`VI (5)`
`VII(6)`	`0`	`0`

Razlog: pri statičkoj inicijalizaciji matrice se podrazumeva da je dužina jedne vrste jednaka dužini najdužeg inicijalizacionog niza; nedostajuće komponente dobijaju vrednost 0. Nakon toga, izvršavaju se navedene `for` petlje. Obratiti pažnju na smer grupisanja (*asocijativnost*) operatora `,` - i redosled izračunavanja izraza koje ovaj operator razdvaja; zato unutrašnja `for` petlja `for(j=0; j<3; s+=rom[i][j],j++);` sumira elemente u kolonama 1 i 2.

Takođe, izraz `*rom[i,j]` evaluira na `*(rom[j])`, zbog načina na koji operator ulančavanja funkcioniše (poslednji ulančani izraz određuje vrednost celog lanca izraza); `rom[j]` je j-ta vrsta matrice, a čitav izraz je, prema tome, ekvivalentan sa `rom[j][0]`.

Zadatak 4.

```
#include <stdio.h>

#define T1(x,y) { x+=y; y=x-y; x-=y; }
#define T2(x) (x&1 ? '*' : ' ')
#define T3(x) ((x>>1)&1 ? '*':' ')

char p[]={0167,044,0135,0155,056,0153,0173,045,0177,0157};
void ps(int b, int r) {
  int c;
  c=p[b]>> (r%2 ? r/2*3:3+(r-3)*4/r);
  if (r%2) printf(" %c ", T2(c));
  else printf("%c %c", T2(c),T3(c));
  printf(" ");
}
void f() {
int i,j,r,k,uc,ruc,c[2]={8};
  char a,b;
  for (i=0; i<10; i++) {
    for (j=i+1,r=i,ruc=0; j<10; j++) {
      for (a=p[r],b=p[j],uc=0,k=7;k;k--) {
        uc+=(a&1)-(b&1); a>>=1; b>>=1; }
      if (uc<ruc) { ruc=uc; r=j; }
    }
    if (r-i) T1(p[i],p[r])
  }
}
void main() {
 int r;
 for (f(), r=1; r<=3; r++) {
   ps(0,r);
   printf("\n");
 }
}
```

```
 *
* *
 *
```

Ovakav zadatak je čist brute force. Potrebno je paziti na bočne efekte makroa, kao i na problematično napisane makroe (oni kod kojih postoje aritmetičke operacije, argumenti su izrazi, a u definiciji makroa nisu upotrebljene zagrade pri navođenju parametara makroa). Primer problema: `T1(p[i]=i, p[j]=j)`. Sasvim je legalno, a ekspanduje se u `p[i]=i+= p[j]=j`. Preporuka pri rešavanju zadataka sa makroima je da pored poziva makroa u kodu zapišete ekspanziju koja se dešava zapravo, i pazite na prioritete operatora!!!

Zadatak 5.

```
#include <stdio.h>
union unija {
     int i; float f; unsigned char c[4];
} u;
char b01(char n){  return n ? b01(n<<1)<<1|n<0 : 0;  }
int a01(int i) {
  union unija u;
  static int k=sizeof(int);
  u.i=i;
  while (--k>=0) u.c[k] = b01(u.c[k]);
  return u.i;
}
main () {
  union unija u;
  u.f = -2011;  printf("%x", a01(u.i));
}
```

```
23df0600
```

Podatak sadržan u uniji je predstavljen na onoliko bajtova koliko I najduži podatak. Prema tome, bajtovi sadržani u nizu `c` su najviša četiri bajta celobrojnog `i`, odnosno najviša 4 bajta realnog `f`, koliko god bajtova taj broj sadržao.

Funkcija `b01` je rekurzivna. U svakom koraku rekurzije, za nenulto `n`, izraz `n<0` evaluira na 1 ukoliko je taj broj negativan, 0 ako je pozitivan. Prema tome, izraz desno od operatora | je zapravo najviši bit broja `n`. Tako određeni bit se postavlja kao najniži bit broja koji se dobija pomeranjem rezultata rekurzivno pozvane funkcije `b01`, nad brojem koji se dobija pomeranjem broja `n` za jedno mesto ulevo. U samom rekurzivnom pozivu, isto se dogodi: najviši bit broja `n<<1` se postavi kao najniži bit rezultata funkcije `b01`, pozvane nad (efektivno!) `n<<2`. Primer, uz pretpostavku da se celi brojevi predstavljaju na 4 bita:

Poziv (“niz” rekurzivno stablo)	Transformacija nad rezultatom	Rezultat posmatranog poziva funkcije
`b01(1001)`	`1001`	
`b01(0010)<<1|1`	`100<<1|1`	`1001`
`b01(0100)<<1|0`	`10<<1|0`	`100`
`b01(1000)<<1|0`	`1<<1|0`	`10`
`b01(0000)<<1|1`	`0<<1|1`	`1`
`--kraj--`	`--`	`0`

Efektivno, ova funkcija obavlja zamenu bita koji su na simetričnim pozicijama (*mirroring*).

Funkcija `a01`, za zadati celi broj `i`, koristeći osobine unija, obrađuje, bajt po bajt, bajtove od kojih je broj `i` sačinjen. Zbog dodele `u.i=i`, vrednost argumenta `i` se u prostor koji unija zauzima smešta kao celi broj predstavljen u drugom komplementu.

Obrada se svodi na poziv gore opisane funkcije `b01`, pa se, prema tome, svaki od bajtova broja `i` zameni svojom *mirrored* slikom. Iako unapred nije poznat broj bajtova koje sadrži celi broj `i`, obrada preko niza `c` funkcioniše, jer C ne vrši proveru toga da li je indeks u okviru statički definisane dužine niza (u našem slučaju 4). Čak i da je navedeno c[1], ovo bi funkcionisalo, jer je c pokazivač na početak niza; budući da je taj niz član unije, `c` sadrži adresu prvog bajta broja `i` odnosno `f`, a koristeći adresnu aritmetiku (indeksiranje) moguće je pristupiti *bilo kom* bajtu broja `i` odnosno `f`.

U glavnom programu se takođe koriste osobine unija. Prvo, broj -2011 se smešta u polje `f`. Prema tome, prostor te unije je popunjen floating point reprezentacijom broja -2011. Uz pretpostavku da se floating point brojevi predstavljaju po IEEE 754 standardu (ovo pitati na takmičenju!), imamo 1 bit za znak, 8 bita za eksponent i 23 bita za mantisu (bilo normalizovanu ili nenormalizovanu, po Standardu). Broj -2011 se predstavlja sa skrivenim bitom;
`2011: 11111011011`, pa je mantisa 1.`1111011011`. Vrednost eksponenta, koju treba kodirati, iznosi 10 (broj značajnih binarnih cifara). Prilikom kodiranja, kodirana vrednost eksponenta se izračunava kao
`E = e+v = 10+127 = 137`, budući da pomeraj (*eng. Bias*) iznosi $2^{k/2} - 1 = 127$. Prema tome, polje `f` biva

popunjeno sledećim binarnim sadržajem: `1|10001001|111101101100000`, odnosno `c4 fb 60 00` u heksadecimalnoj formi, a funkcija `a01` izvrši opisanu transformaciju nad svakim bajtom ponaosob.

Obratiti posebnu pažnju na definiciju promenljive `k` u funkciji `a01`. To je `static` promenljiva unutar funkcije, što znači da se njena inicijalizacija dešava samo jedanput, pri prvom pozivu funkcije!!! Kasniji pozivi koriste tu staru vrednost. U ovom zadatku, to nema značaja, ali može napraviti probleme.
Pogledati poglavlje 4.6 „Trajsnost podataka“, kao i poglavlje 8.6 „Deklaracija i definicija“ knjige Lasla Krausa.

Zadatak 6.

```
#include <stdio.h>
double f(int i){
  double data = 1;
  while(i--) data/=2;
  return (data);
}
union unija {double d; float f; char c[8];} u;
main(){
  int i, j, br;
  for (br=0, i=126; i<1022; i++)
  for (u.f=f(i), j=3; j>=0; j--)
  if (u.c[j]) br++; printf("%d", br);
}
```

24

Funkcija `f` izračunava i-ti negativni stepen broja 2. Obratiti pažnju da se u drugoj for petlji dodela vrši polju `f` unije, a ne polju `d`. Stoga, vrednost odgovarajućeg stepena dvojke se "prepakuje" tako da može da se prikaže kao float broj (nije ista bitska reprezentacija rezultata funkcije `f` i polja `u.f`, jer se implicitna konverzija stara da se vrednost očuva).

Unutar if-a se broji broj bajtova od kojih je sačinjeno polje `f`, pošavši od poslednjeg, ka prvom, koji imaju bitski sadržaj različit od 0.

Spoljašnja for petlja iterira putem `i` u opsegu `126..1022`. Budući da je, za IEEE float brojeve predstavljene na 32 bita moguće da je najmanja *vrednost* eksponenta koristeći *normalizovanu* mantisu E_{min} = -126. Binarna predstava broja 2^{-r} je oblika $0.\underbrace{00\ldots0}_{r-1}1\ldots0$, takav broj se predstavlja kao $1.0 \cdot 2^{-r}$. Uzevši obe činjenice u obzir, zaključuje se da samo za vrednost `i=126` može da se predstavi broj 2^{-126} koristeći *normalizovanu* mantisu, i ta mantisa iznosi 1.00…0, a zbog skrivenog bita, svi bitovi u predstavi mantise biće 0!

Za ostale vrednosti `i`, koristi se nenormalizovana mantisa (`E=0`). Kod nenormalizovane mantise, vrednost mantise se uzima "bukvalno" kako je kodirana (nema skrivenog bita), pa je 2^{-127} prikazano kao 0|00000000|10000000000000000000000. Ostali negativni stepeni dvojke se dobijaju pomerajući jedinicu kroz mantisu (2^{-128} -> 01000000000000000000000 itd.). Prema tome, najveći stepen koji se može predstaviti je $2^{-(126+23)} = 2^{-149}$. Ostale vrednosti bivaju zaokružene na 0.

Uzevši to u obzir, i radnju koja se obavlja u `if`, zaključuje se da će:

- Za *normalizovanu* mantisu biti samo 1 bajt aktivan (bajt koji sadrži kodirani eksponent), dok svi bitovi mantise imaju vrednost 0.
- Za *nenormalizovanu* mantisu, samo 1 bajt je aktivan (onaj u kome se nalazi jedinica koja "šeta"). Takvih brojeva ima 23

Ukupno: 24.

Zadatak 7.

```
#include <stdio.h>
void mov(int *s, int *t, int *p, int n) {
  if (n==0) return;
  mov(s,p,t,n-1); (*s)++;
  (*t)=((*t)<=(*p))?(*t)++:(*t)--;
  (*p)=((*p)>=(*t))?(*p)++:(*p)--;
  mov(p,t,s,n-1);
  (*s)++;
}
main() {
  int L=0,C=011,D=0;
  mov(&L,&C,&D,3);
  printf("%d\n%d\n%d",L,C,D);
}
```

```
8
11
4
```

I ovaj zadatak se svodi na brute force i pažljivo razmatranje stanja pokazivača. Budući da je funkcija `mov` rekurzivna, najbolje bi bilo rešavati je crtanjem dijagrama stanja promenljivih.

Zadatak 8.

```
#include <stdio.h>
void main () {
  int i;
  int t[4][3] = {{0, 1, 2}, {3, 4, 5}, {6, 7, 8}};
  int *p[5], *q;
  int (*r)[3] = t;
  p[0] = &(*r++)[2];
  *(p+1) = &((*r)[r-t]);
  q = &(*r)[1];
  for(i=3;i>1;i--) {
       *(p+i) = &(*++r)[i%2];
       t[i][2]=++*q++;
  }
  p[4] = q;
  for (i=0; i<5; i++) printf("%d", *p[i]);
}
```

```
25076
```

Zadatak 9.

```
#include <stdio.h>
#include <string.h>
#define MAX 10
main()
{  int mat[MAX][MAX],b,i,j,k,m,max,n;
   m=strlen("Elektrijada")-1;
   n=strlen("2003");
   for (i=1,b=1; i<2*m-i; i++)
   {  j=(i-m+1>1)?i-m+1:1;
      max=(i<m)?i:m;
      for ( ;j<=max; j++) mat[j][i-j+1]=b++;
   }
   for (i=1; i<n; i++)
   {   for (j=i,k=1; j<n; k++,j++) printf("%d",mat[i][k]);
       printf("\n");
   }
}
```

```
124
35
6
```

Zadatak 10.

```
#include <stdio.h>
main()
{   unsigned int p1=0125252, p2=0052525;
    p1&=~(0xF<<8); p2|=p1>>8;
    printf("%o",~(~p1^~p2));
}
```

```
37777605240
```

Relativno prost zadatak u kome se radi sa bitskim operatorima primenjenim nad oktalnim brojevima.

Zadatak 11.

```
#include <stdio.h>
main() {   struct tacka
              {int x,y,smjer:3;
               unsigned int brzina:4;
               short int boja;} a;
    a.x=0x100; a.y=0x200; a.smjer=3; a.brzina=-19; a.boja=0x500;
    printf("%d %d ",a.brzina,a.smjer);
    a.brzina = a.smjer+a.brzina; printf("%d ",a.brzina);
}
```

```
13 3 0
```

U ovom zadatku se koriste tzv. bit-polja struktura. Pogledati o strukturama i bitskim poljima u knjizi. Takođe, pogledati o implicitnoj konverziji i mehanizmu konverzije podataka u C-u. Pogledati primer u poglavlju 3.8. Konverzija tipa, gde se porede `1L` i `1UL`.

Dodela `a.x=0x100;` se ostvaruje tako što se samo 4 bita „zapamte“, ali tako da dođe do neke vrste zaokruživanja (`a.x=7`). Dodela `a.brzina` se realizuje tako što se binarni oblik broja -19 *neizmenjen(!)* preda neoznačenom polju `brzina`. Kako je predstava broja -19 zapravo `111…11|1011`, u polje `brzina` biva upisan sadržaj *najniža* 4 bita te predstave, pa je `a.brzina=13`.

Preporuke za podsećanje/učenje

Dobar deo dole navedenih stvari moguće je pronaći u knjizi Lasla Krausa. Koristiti indeks i sadržaj da biste pronašli odgovarajuće tekstove:

- Predstavljanje konstanti različitih tipova (literali I sl.)
- Enumeracije - način funkcionisanja
- Pokazivači na funkcije, Pokazivači i nizovi
- Rad sa višedimenzionalnim nizovima: moguća adresiranja (mešovito indeksiranje pomoću `[]` i čiste adresne aritmetike), statički višedimenzionalni nizovi i statička inicijalizacija elemenata niza (uz nenavedene izraze za pojedine elemente).
- Strukture, bit-fields (strukture čije su komponente tačno definisane dužine u bitovima)
- Unije (struktura sa samo jednim fizičkim podatkom, u smislu memorijskog zauzeća, a čiji se binarni sadržaj tumači čas kao jedan tip, čas kao drugi, zavisno od komponente kojom mu se pristupa)
- Static promenljive: `static` promenljive se mogu definisati unutar funkcije ili izvan nje. Ova druga situacija nije od insteresa, budući da ima na značaju samo kada se radi sa više .c fajlova; prva je bitna, jer se na taj način definiše trajni podatak (kao globalni) sa dosegom ograničenim na BLOK u kom je definisan, a njegova inicijalizacija se dešava samo pri prvom izvršavanju definicije tog podatka (prvi put kada se uđe u taj blok, v. Program 4.2 LK).
- Inicijalizacija promenljivih i nedostatak iste. Koje su inicijalne vrednosti statičkih, globalnih i lokalnih promenljivih, ako se ne navede inicijalizator?
- Endianess – koji bajt se smešta na koju adresu. Little endian: niži bajt je na nižoj adresi; bigendian: niži je na višoj adresi. Primer za littleendian: `0x1234` -> adresa 100h: `34` adresa 101h: `12`.
- Rad sa funkcijama: prenos argumenata po vrednosti. Drugačije se argument ne prenosi u C-u. Pascal-ski prenos po reference se ostvaruje na taj način što funkcija prihvata argument tipa pokazivač na podatak (i time se pokazivač prenese po vrednosti, a uslovno govoreći, podatak po referenci). Izmene nad argumentom su nevidljive izvan funkcije; izmene nad pokazanim podatkom ostaju vidljive.
- Makroi i simboličke konstante. Posebnu pažnju obratiti na makroe koji nisu korektno napisani (kao `#define SQR(X) X*X`), i na način na koji se pozivaju. Takođe, obratiti pažnju na upotrebu argumenta makroa u vidu stringa (#X se zameni tekstom sa kojim je makro pozvan, pa se `M(X) #X` eskpanduje u YY za poziv `M(YY)`), kao i konkatenaciju (`M(X) X ##promenljiva` se ekspanduje u `YYpromenljiva`, dok se `N(X) #X##prom` ekspanduje u `“YY”prom`, kada se pozovu sa `M(YY)`, `N(YY)`, respektivno).