1. Auditorne vježbe

Raspored nastave

Izvodit će se 3 tjedna po 1h kao auditorne za cijelu grupu, u tjednu iza toga grupa se dijeli na 2 podgrupe i jedna podgrupa ima po 2h laboratorijskih vježbi u tom tjednu, a druga u sljedećemu. Cijeli se ciklus ponavlja 3 puta.

1. tjedan	2. tjedan	3. tjedan	4. tjedan	5. tjedan	
A 1h Auditorne	A 1h Auditorne	A 1h Auditorne	A C E G I K 1h Priprema + Blitz	B D F H J L 2h Priprema + Blitz	x 3

Pokazivači

Zadatak:

Što će se ispisati izvođenjem sljedećeg programskog odsječka?

```
int a=4;
int *b;

b=&a;
*b=8;

printf("%d %d",a,*b);
```

Rješenje:

```
int a=4;
/* prilikom izvođenja programa, simbolička oznaka a dobiva
određenu adresu – npr. 2000 */

int *b;
/* pokazivač b pokazuje na adresu na kojoj se nalazi cijeli
broj
pretpostavimo da je b oznaka za adresu 2004. */
```

Adresa u memoriji:	2000 (a)	2004 (b)
Sadržaj memorije:	4	?

```
b=&a; // inicijalizacija
```

Adresa u memoriji:	2000	2004
Sadržaj memorije:	4	2000

```
*b=8;
```

Adresa u memoriji:	2000	2004
Sadržaj memorije:	8	2000

Ispisati će se **8 8**, jer se vrijednost na koju pokazuje varijabla b nalazi na istoj adresi (`b=&a`), gdje i vrijednost varijable a.

Pitanje:
Što bi se dogodilo da izostavimo liniju `b=&a;` ?

Odgovor:
Budući da pokazivač `b` prije pridruživanja vrijednosti pokazuje na nedefiniranu adresu, može doći do pokušaja upisivanja vrijednosti 8 na adresu koja je rezervirana za pohranu drugih varijabli ili koda, što može izazvati neočekivano ponašanje ili pogrešku pri izvođenju programa zbog neovlaštenog pristupa dijelu memorije. Stoga je uvijek potrebno inicijalizirati vrijednost pokazivača prije upotrebe.

Zadatak:

Napisati funkciju koja će iz zadanog JMBG-a vratiti datum rođenja u obliku DD.MM.GG.

Rješenje:

```
char * datum(char *jmbg) {
   int d,m,g; // lokalne varijable – vrijede samo unutar
funkcije
   char *p; //pokazivač je deklariran, ali nije inicijaliziran!
   char c;
   sscanf(jmbg,"%2d%2d%c%2d",&d,&m,&c,&g);

   //JMBG ima oblik DDMMYYYXXXXXX
```

```
   p=(char *) malloc(9 * sizeof(char));

/* pokazivač je inicijaliziran i naredbom malloc zauzeta je
memorija potrebna za pohranu datuma u formatu DD.MM.GG */

   sprintf(p,"%02d.%02d.%02d",d,m,g);

   // datum se sprema na adresu na koju pokazuje pokazivač p

   return p;

}
```

Polja

Zadatak:

Napisati funkciju koja računa zbroj pozitivnih elemenata dvodimenzionalnog polja, te prikazati poziv funkcije iz glavnog programa.

```
float zbroj_poz(int brred, int brstup, int maxstup, float *p )

/* brred, brstup, int maxstup – ulazni parametri funkcije;
njihova eventualna promjena u funkciji neće biti sačuvana
nakon povratka u glavni program. Budući da se polje uvijek
prenosi u funkciju kao pokazivač, vrijednost članova polja
moguće je u funkciji mijenjati */

{
   int i,j;
   float  suma;

   suma = 0.0;
   for( i=0; i<brred; i++ )
      for( j=0; j<brstup; j++ )
         if (p[i*maxstup + j]>0) suma += p[i*maxstup + j];

         /* može i suma += *(p + i*maxstup + j)
          nije dozvoljeno p(i,j), p[i][j] i slično – vidi
objašnjenje */

   return suma;
}
```

Objašnjenje: Dvodimenzionalno polje se u funkciju prenosi kao jednodimenzionalno. Pohranjeno je po retcima, pa se elementu p(i,j) pristupa kao p[i*maxstup+j].

Glavni program:

```
#define MAXRED 100
#define MAXSTUP  20

void main() {
int    red, stup;
   float zbroj, mat[MAXRED][MAXSTUP];

/* ovdje slijedi postavljanje stvarnog broja redova i stupaca
(red<=MAXRED, stup<=MAXSTUP, punjenje polja */

zbroj = zbroj_poz(red, stup, MAXSTUP,(float *) mat);  // poziv
f-e
}
```

Zapisi i Datoteke

Zadatak:

Zadana je direktna datoteka `studenti.dat` i struktura zapisa (vidi rješenje). Napisati funkciju koja dohvaća podatke o studentu sa zadanim matičnim brojem, te ga potom briše iz datoteke. Šifra studenta odgovara rednom broju zapisa. Prazan zapis sadrži šifru jednaku nuli.

Rješenje:

```
struct zapis {
  int mbr;
  char ime_prezime[40];
  char spol;
};

struct zapis dohvati_brisi(int mbr) {  // funkcija vraća dohvaćeni zapis
  FILE *f;
  struct zapis z; // z je varijabla tipa 'struct zapis'
  struct zapis *z1;  // z1 je pokazivač na strukturu

  if ((f=fopen("studenti.dat","r+b"))==NULL) exit(1);

   /* ukoliko nije zadano ime datoteke, pretpostaviti da je ona već
   otvorena u glavnom programu, te se funkciji predaje pokazivač FILE *f
   */

   fseek(f,(mbr-1)*sizeof(zapis),SEEK_SET)

   /* direktan pristup zapisu jer je zadano da šifra odgovara rednom
   broju zapisa */

   fread(&z,sizeof(z),1,f);
   z1=&z; // inicijalizacija pokazivača z1
   z1->mbr=0; // isto što i (*z1).mbr=0

   /* zapisi se iz direktne datoteke ne brišu fizički. U zadatku je
   zadano da vrijednost 0 označava prazan zapis */

   fseek(f,-sizeof(zapis),SEEK_CUR);
   /* nakon čitanja potrebno se vratiti na početak zapisa, da bi se zapis
   upisao na isto mjesto */

   fwrite(z1,sizeof(*z1),1,f);

  fclose(f);
  return z;
}
```

2. Auditorne vježbe

Zadatak:

U poreznoj upravi u nekoj neformatiziranoj datoteci postoje podaci s poreznih kartica poreznih obveznika. Bitni podaci su *JMBG* (13+1 znamenka), *ime i prezime* (40+1 znak), *ukupni prijavljeni prihod* (float) i *iznos poreza koji još moraju platiti* (float). Na početku datoteke je upisan jedan podatak tipa long koji govori koliko ima zapisa u datoteci. Potrebno je napisati glavni program sa sljedećim dijelovima:

a) funkcijom koja učitava sadržaj datoteke u dinamički alocirano polje struktura

b) funkcijom koja nalazi poreznog obveznika koji mora platiti najveći porez

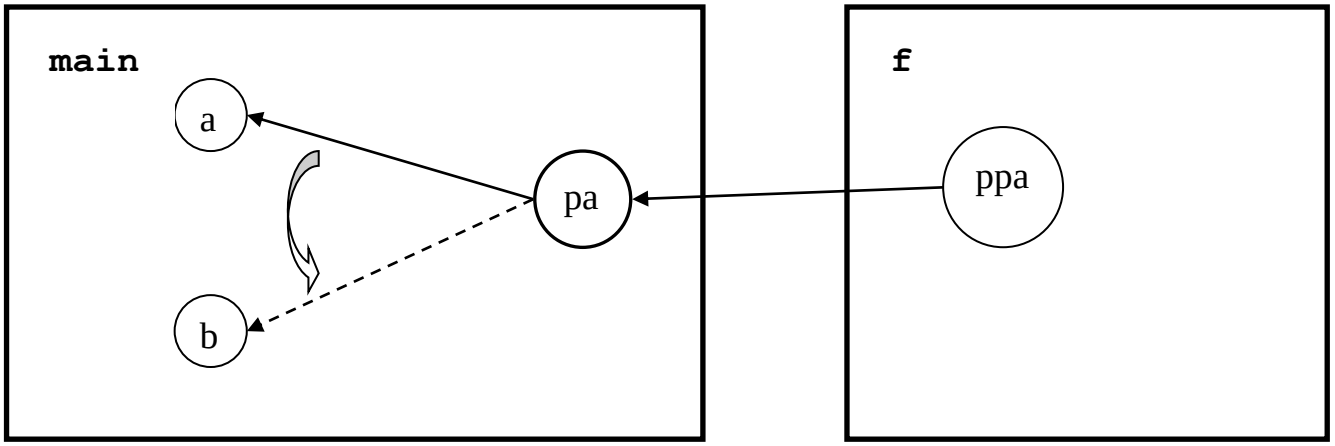
```
struct element {
   char JMGB[13+1];
   char ImePrezime[40+1];
   float  Prihod;
   float  PlatitiPorez;
};
typedef struct element zapis;

zapis* Ucitaj(char *FileName, int *BrElem);
int    MaxPorez( zapis *Polje, int BrElem );

int main()
{
   int    BrElem, MaxInd; //lokalne varijable
   char FileName[40];
   zapis* Polje;

   printf("Unesite ime datoteke : ");
   gets(FileName);

   Polje = Ucitaj(FileName, &BrElem);

   if( Polje != NULL ) {
      MaxInd = MaxPorez(Polje, BrElem);

      printf("Najviše poreza treba platiti %s, u iznosu od %f",
      Polje[MaxInd].ImePrezime,
      Polje[MaxInd].PlatitiPorez );
   }
   free(Polje);
}
```

```
zapis* Ucitaj( char *FileName, int *BrElem ) {
   int    i; //lokalne varijable
   long Br;
   FILE   *fp;
   zapis  *Polje = NULL;
   zapis  Elem;

   if( ( fp = fopen(FileName, "rb") ) == NULL ) {
      printf("Ne mogu otvoriti datoteku");
      return NULL;
   }

   fread( &Br, sizeof(long), 1, fp );

   if( Br > 0 ) {
      Polje = (zapis *) malloc( Br * sizeof(zapis) );
      /* inicijalizacija varijable polje – pokazuje na adresu
      na kojoj je slobodno kontinuirano područje od Br *
      sizeof(zapis) byte-ova */
   }
   else {
      printf("Neispravan broj elemenata");
      return NULL;
   }

   i = 0;
   while( fread(&Elem, sizeof(zapis), 1, fp) == 1) {
      Polje[i++] = Elem;      /*   obavlja se pridruživanje
      cijelog sadržaja jedne strukture drugoj na način da se
      iskopira sadržaj memorije !!! */
   }
   // moglo je i: fread(Polje, sizeof(zapis), Br, fp);
   *BrElem = Br;//uz pretpostavku da smo sve uspješno pročitali

   fclose(fp);
   return Polje;
}
```

```
int  MaxPorez( zapis *Polje, int BrElem ) {
   int    MaxInd = 0;
   float  Max;

   Max = Polje[0].PlatitiPorez;

   for( i=1; i<BrElem; i++ ) {
      if( Polje[i].PlatitiPorez > Max ) {
         MaxInd = i;
         Max = Polje[i].PlatitiPorez;
      }
   }
  return MaxInd;
}
```

Dvostruki pokazivači

Primjer.
Potrebno je napisati funkciju koja će promijeniti vrijednost pokazivača (kamo pokazivač pokazuje). U primjeru ćemo "preusmjeriti" pokazivač pa s varijable a na varijablu b:




```
int f(int **ppa, int *novaAdresa){
   *ppa = novaAdresa;
}
void main(){
   int a=1, b=2;
   int *pa = &a;
   printf("\n &pa=%d pa=%d *pa=%d", &pa, pa, *pa);
   f(&pa, &b);
   printf("\n &pa=%d pa=%d *pa=%d", &pa, pa, *pa);
}
(ispisuje:)
 &pa=1245044 pa=1245052 *pa=1
 &pa=1245044 pa=1245048 *pa=2
```

Zadatak:
Kako bi se napisala funkcija koja će iz danog polja izbaciti zadnji element ?

Problem: potrebno je promijeniti veličinu polja, i to se mora napraviti u funkciji, a ne u glavnom programu. Moguće rješenje je korištenje globalne varijable tipa `zapis * Polje`, koju bi onda funkcija promijenila te bi tu promjenu registrirao i glavni program.
Međutim, kako korištenje globalnih varijabli nije dopušteno, mogu se primijeniti dva pristupa:

a) korištenje dvostrukih pokazivača
b) funkcija vraća adresu novog polja

Prvo rješenje je elegantnije jer dopušta da povratni parametar funkcije bude nekakav cijeli broj koji će signalizirati da li je funkcija uspješno obavila zadatak ili ne. U drugom zadatku se taj kôd mora vraćati *by reference* preko parametra funkcije.

---

a) prvo rješenje

```
int  Izbaci( zapis **Polje, int *BrElem ) {
   *Polje = (zapis *) realloc(*Polje, (*BrElem - 1) *
   sizeof(zapis) );

   if( *Polje == NULL ) {
      /* nije uspjelo realociranje memorije */
      return 0;
   else {
      *BrElem -= 1; // treba smanjiti i brojac elemenata polja
      return 1;
   }
}
```

---

b) drugo rješenje

```
zapis* Izbaci( zapis *Polje, int *BrElem, int *KodGreske )
{
   Polje = (zapis *) realloc(Polje, (*BrElem - 1) *
   sizeof(zapis) );

   if( Polje == NULL ) {
      // nije uspjelo realociranje memorije
      *KodGreske = 0;
   else {
      *KodGreske = 1;
      *BrElem -= 1; // treba smanjiti i brojac elemenata polja
   }

   return Polje;
}
```

```
int main() {
   int    BrElem, KodGreske;
   char FileName[40];
   zapis* Polje;

   printf("Unesite ime datoteke : ");
   gets(FileName);

   Polje = Ucitaj(FileName, &BrElem);

   // prva varijanta
   if( Polje != NULL ) {
      KodGreske = Izbaci( &Polje, &BrElem)
   }

   // druga varijanta
   if( Polje != NULL ) {
      Polje = Izbaci( Polje, &BrElem, &KodGreske)
   }
}
```

3. Auditorne vježbe

Rekurzija

Što je rekurzija?

- Funkcija poziva samu sebe
- Mora postojati završetak
- Programi su kraći, ali se izvode duže
- Koristi se stog za pohranjivanje rezultata i povratak iz rekurzije

1. zadatak: *računanje potencije*
Napisati funkciju koja prima dva argumenta **x** i **y**, cjelobrojnog tipa, i vraća preko imena vrijednost $\mathbf{x}^{\mathbf{y}}$.

Rješenje:

```
long pot(long x, long y) {
   if (y <= 0) return 1;
   else return x * pot(x, y - 1);
}
```

Poziv funkcije:

```
k = pot(2,5);
     = 2*pot(2,4)
          = 2*pot(2,3)
               = 2*pot(2,2)
                    = 2*pot(2,1)
                         = 2*pot(2,0)
                              = 1
```

STOG:

pot(2,5)	pot(2,4)	pot(2,3)	pot(2,2)	pot(2,1)	pot(2,0)	return 1	return 2*1	return 2*2	return 2*4	return 2*8	return 2*16
					(2,0)	1					
				(2,1)	(2,1)	(2,1)	2				
			(2,2)	(2,2)	(2,2)	(2,2)	(2,2)	4			
		(2,3)	(2,3)	(2,3)	(2,3)	(2,3)	(2,3)	(2,3)	8		
	(2,4)	(2,4)	(2,4)	(2,4)	(2,4)	(2,4)	(2,4)	(2,4)	(2,4)	16	
(2,5)	(2,5)	(2,5)	(2,5)	(2,5)	(2,5)	(2,5)	(2,5)	(2,5)	(2,5)	(2,5)	32

Pitanje:
Što bi se dogodilo kada bi bila izostavljena linija:

```
    if (y <= 0) return 1;
```

tj. kad bi funkcija izgledala ovako:

```
long pot(long x, long y) {
   return x * pot(x, y - 1);
}
```

Odgovor:
Funkcija bi samu sebe pozivala beskonačno puta i nikada ne bi vratila neku vrijednost u glavni program. Npr. u gornjem primjeru dogodilo bi se slijedeće:

```
pot(2,5);
 = 2*pot(2,4)
      = 2*pot(2,3)
           = 2*pot(2,2)
                = 2*pot(2,1)
                     = 2*pot(2,0)
                          = 2*pot(2,-1)
                               = 2*pot(2,-2)
                                    = 2*pot(2,-3)
                                         ...
```

Rješenje bez rekurzije:

```
long pot(long x, long y) {
   long retval = 1;
   int i;
   for (i = 0; i < y; i++) retval *= x;
   return retval;
}
```

2. zadatak: *aritmetički niz*
Napisati rekurzivnu funkciju koja računa n-ti član aritmetičkog niza.

Rješenje:

```
long aniz(long a0, long d, long n) {
   if (n == 0) return a0;
   else return d + aniz(a0, d, n-1);
}
```

3. zadatak: *ispis niza brojeva* (često pitanje na usmenom ispitu)
Napisati funkciju koja ispisuje sve brojeve od 1 do 100.

Rješenja s dva ulazna argumenta:

```
// Ispisuje uzlazno (od 1 do 100)
void ispis(int n1, int n2) {
   if (n1 <= n2) {
      printf(˝\n%d˝, n1);
      ispis(n1 + 1, n2);
   }
}
```

```
// Ispisuje uzlazno (od 1 do 100)
void ispis(int n1, int n2) {
   if (n1 <= n2) {
      ispis(n1, n2 - 1);
      printf(˝\n%d˝, n2);
   }
}
```

```
// Ispisuje silazno (od 100 do 1)
void ispis(int n1, int n2) {
   if (n1 <= n2) {
      printf(˝\n%d˝, n2);
      ispis(n1, n2 - 1);
    }
}
```

```
// Ispisuje silazno (od 100 do 1)
void ispis(int n1, int n2) {
   if (n1 <= n2) {
      ispis(n1 + 1, n2);
      printf(˝\n%d˝, n1);
   }
}
```

Poziv funkcije:

```
ispis(1,100);
```

Rješenja s jednim ulaznim argumentom:

```
// Ispisuje uzlazno (od 1 do 100)
void ispis(int n) {
   if (n >= 1) {
      ispis(n - 1);
      printf(˝\n%d˝, n);
   }
}

// Ispisuje silazno (od 100 do 1)
void ispis(int n) {
   if (n >= 1) {
      printf(˝\n%d˝, n);
      ispis(n - 1);
   }
}
```

Poziv funkcije:

```
ispis(100);
```

4. Zadatak: *kamatni račun*

Zadana suma novaca oročena je u banci na zadani broj godina uz zadanu godišnju kamatnu stopu. Napisati program koji računa dobivenu sumu nakon isteka oročenja.

Rješenje:

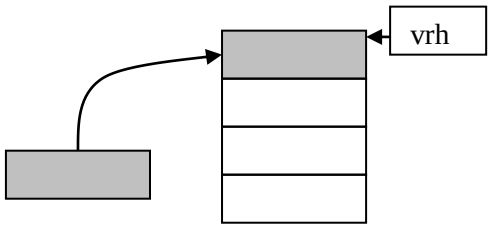
```
// g – glavnica
// n – trajanje oročenja u godinama
// p – kamatna stopa u postotcima
float kamrac(float g, int n, float p) {
   if (n <= 0) return g;
   else return (1 + p / 100) * kamrac(g, n - 1, p);
}
```

Stog

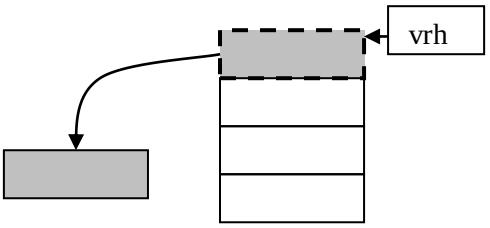
- struktura podataka kod koje se posljednji pohranjeni podatak obrađuje prvi
  - LIFO (*Last In First Out*)
- osnovni postupci:
  - dodavanje (stavljanje, engl. *push*) elemenata na vrh stoga
  - brisanje (skidanje, engl. *pop*) elemenata s vrha stoga
  - zahtijevaju jednako vrijeme izvođenja bez obzira na broj elemenata pohranjenih na stogu: O(1)
- realizacija:
  - statička struktura podataka (polje)
  - dinamička struktura podataka (lista)

1. zadatak (REALIZACIJA STOGA POLJEM):
Na stogu se nalaze podaci o rezultatima ispita : **šifra ispita** (`int`), **šifra studenta** (`long`), **ocjena** (`int`). Realizirati funkcije za stavljanje elementa na stog i uzimanje elementa sa stoga, čiji je maksimalni kapacitet *n*. Stog realizirati poljem. Napisati program koji će sa stoga izbaciti sve rezultate s neprolaznom ocjenom. Elementima stoga pristupati samo pomoću realiziranih funcija.
Zapisi o rezultatima ispita se učitavaju dok se za ocjenu ne učita 0.

Rješenje:
```
#include <stdio.h>
#define MAXSTOG 100
struct s {
  int  sifraIspit;
  long sifraStudent;
  int  ocjena;
};
typedef struct s zapis;

int dodaj(zapis stavka, zapis stog[], int n,
  int *vrh) {
  // ako je stog pun
  if (*vrh >= n-1) return 0;
  // ako ima mjesta na stogu
  (*vrh)++;
  stog [*vrh] = stavka;
  return 1;
}
```



```
int skini (zapis *stavka, zapis stog[], int *vrh) {
   // ako je stog prazan
   if (*vrh < 0) return 0;
   // ako ima zapisa na stogu
   *stavka = stog[*vrh];
  (*vrh)--;
  return 1;
}
```




```
int main() {
   zapis z, stog[MAXSTOG], pomStog[MAXSTOG];
   int i = 0, vrh = -1, pomVrh = -1;

   // ucitavanje elemenata stoga
   do {
      scanf("%d %ld %d", &z.sifraIspit,
                         &z.sifraStudent, &z.ocjena);
     // prekini ucitavanje,
      // ako je za ocjenu ucitana 0
      //ili ako je stog pun
   } while (z.ocjena &&
            dodaj (z, stog, MAXSTOG, &vrh));

   // brisanje zapisa s neprolaznim ocjenama
   // koristenjem pomocnog stoga:
   // 1. premjesti sve zapise s
   // prolaznom ocjenom na pomocni stog
   while (skini(&z, stog, &vrh)) {
      if (z.ocjena > 1)
         dodaj(z, pomStog, MAXSTOG, &pomVrh);
   }

   // 2. premjesti nazad na pocetni stog
   // zapise s pomocnog stoga
   while (skini(&z, pomStog, &pomVrh)) {
      dodaj(z, stog, MAXSTOG, &vrh);
      // kontrolni ispis
      printf("z.ocjena = %d\n", z.ocjena);
   }
}
```

2. zadatak (REALIZACIJA STOGA DATOTEKOM):
Na magnetskom disku postoji neformatizirana datoteka *stog.dat* koja je organizirana kao stog. Na početku datoteke upisana je adresa zadnje upisanog elementa na stogu (`long`). Ako je ta vrijednost negativna, stog je prazan. Element stoga je zapis o položenom ispitu studenta:

- matični broj (`long`)
- ime i prezime (24+1 znak)
- šifra predmeta (`int`)
- ocjena (`short`)

Napisati funkciju za ispis sadržaja stoga uz njegovo pražnjenje.

Za vježbu:
Napisati funkciju koji stavlja zapise na stog pohranjen u datoteci.

Rješenje:

```
#include <stdio.h>
#include <stdlib.h>
struct s {
  long  maticniBroj;
  char  imePrezime[24+1];
  int   sifraPredmet;
  short ocjena;
};
typedef struct s zapis;


int fatal(char *poruka) {
   puts(poruka);
   exit(1);
}
```

a) *Prednost:* učinkovitije - datoteka se otvara i zatvara samo jednom u funkciji *stog*, a adresa zadnjeg zapisa u datoteci se piše samo jednom u funkciji *stog*
Nedostatak: nije crna kutija - ako dođe do nepredviđenog završetka izvođenja programa, u datoteci će biti zapisana pogrešna adresa zadnjeg zapisa na stogu; nedovoljno općenito

```
int skini(zapis *stavka, long *adresaZadnji, FILE *f) {
   // ako se ne moze pozicionirati (adresaZadnji < 0)
   // u datoteci, znaci da je stog prazan
   if (fseek(f, *adresaZadnji, SEEK_SET)) return 0;

   // procitaj zapis s kraja datoteke,
   // tj. s vrha stoga
   fread(stavka, sizeof(*stavka), 1, f);

   *adresaZadnji -= sizeof(zapis);
   return 1;
}

void stog () {
   FILE *f;
   long adresaZadnji = -1L;
   zapis z;
   if ((f = fopen("stog.dat", "r+b")) == NULL)
      fatal("Datoteka se ne moze otvoriti!");

   // prvo procitaj adresu zadnjeg
   fread(&adresaZadnji, sizeof(adresaZadnji), 1, f);

   // isprazni stog i ispisi sve podatke
   while (skini(&z, &adresaZadnji, f)) {
      printf("%ld %s %d %hd\n",
               z.maticniBroj, z.imePrezime,
               z.sifraPredmet, z.ocjena);
   }

   // nakon sto procitas sve zapise,
   // zapisi na pocetak datoteke
   // novu adresu zadnjeg zapisa
   fseek(f, 0L, SEEK_SET); // ILI: rewind(f);
   fwrite(&adresaZadnji, sizeof(adresaZadnji), 1, f);
   fclose(f);
}
```

b) *Prednost:* radi po principu crne kutije - zatvorene cjeline koja funkcionira sama za sebe
Nedostatak: slabija učinkovitost (svakim pozivom funkcije se otvara i zatvara datoteka i svaki put se ponovo piše adresa zadnjeg zapisa)

```
int skini(zapis *stavka) {
//JOS OPCENITIJE:
// int skini(void *stavka, size_t velicinaStavka)

   FILE *f;
   long adresaZadnji;

   if ((f = fopen("stog.dat", "r+b")) == NULL)
      fatal("Datoteka se ne moze otvoriti!");

   // prvo procitaj adresu zadnjeg
   fread(&adresaZadnji, sizeof(adresaZadnji), 1, f);

   // ako se ne moze pozicionirati (adresaZadnji < 0)
   // u datoteci, znaci da je stog prazan
   if (fseek(f, adresaZadnji, SEEK_SET)) return 0;

   // procitaj zapis s kraja datoteke,
   // tj. s vrha stoga
   fread(stavka, sizeof(*stavka), 1, f);

   // zapisi na pocetak datoteke
   // novu adresu zadnjeg zapisa
   adresaZadnji -= sizeof(*stavka);
   fseek(f, 0L, SEEK_SET); // ILI: rewind(f);
   fwrite(&adresaZadnji, sizeof(adresaZadnji), 1, f);
   fclose(f);
   return 1;
}

void stog () {
   zapis z;
   // isprazni stog i ispisi sve podatke
   while (skini(&z)) {
      printf("%ld %s %d %hd\n",
               z.maticniBroj, z.imePrezime,
               z.sifraPredmet, z.ocjena);
   }
}
```

3. zadatak:
Napisati funkciju koja će korištenjem zadanog stoga zamijeniti sadržaje varijabli `a` i `b`.
Funkcija ima prototip:

```
int zamijeni (int *a, int *b, int stog[],
              int *vrh, int N)
```

Funkcija `zamijeni` ne smije koristiti dodatne varijable. Ukoliko je zamjena uspjela, funkcija `zamijeni` treba vratiti 1, a ukoliko nije, vraća se 0.

Za stavljanje i skidanje podataka sa stoga koristiti postojeće funkcije:

```
int stavi(int podatak, int stog[], int *vrh, int N)
int skini(int *podatak, int stog[], int *vrh)
```

Funkcije `stavi` i `skini` vraćaju 1, ako je podatak uspješno stavljen/skinut sa stoga, a 0 inače.

Rješenje:

```
int zamijeni(int *a, int *b, int stog[],
             int *vrh, int N) {
  int ok = 0;
  // elementi se skidaju sa stoga
  // obratnim redoslijedom
  // od onog kojim se stavljaju na stog

  if (stavi(*a, stog, vrh, N)) {
     if (stavi(*b, stog, vrh, N)) {
        // ako su se obadvije varijable
        // uspjele staviti na stog, znaci da stog
        // nije prazan, pa ce se uspjeti nesto
        // i skinuti s njega
        skini(a, stog, vrh);
        skini(b, stog, vrh);
        ok = 1;
     } else {
        // ako se druga varijabla
        // nije uspjela staviti na stog,
        // obrisi i prvu sa stoga
        skini(a, stog, vrh);
     }
  }
  return ok;
}
```

DODATAK:

Zadatak (REKURZIJA): *hanojski tornjevi*
Napisati program koji rješava problem hanojskih tornjeva.

- Štapovi S (*source*, izvor), D (*destination*, odredište), T(*temp*, pomoćni)
- Na prvom štapu (S) ima n diskova različite veličine postavljenih tako da veći nikad ne dolazi iznad manjeg. Preseliti sve diskove na D, **jedan po jedan**, uvijek postavljajući manji na veći

Algoritam rješenja:

- Ignorirati donji (najveći) disk i riješiti problem za n-1 disk, ali sa štapa S na štap T koristeći D kao pomoćni.
- Sada se najveći disk nalazi na S, a ostalih n-1 na T.
- Preseliti najveći disk sa S na D.
- Preseliti n-1 disk sa T na D koristeći S kao pomoćni (problem je već riješen za n-1 disk).

Rješenje:

```
void hanoii(char src, char dest, char tmp, int n) {
   if (n > 0) {
      hanoii(src, tmp, dest, n - 1);
      printf(˝\nPrebacujem element %d s tornja
                %c na toranj %c˝, n, src, dest);
      hanoii(tmp, dest, src, n - 1);
   }
}
int main() {
   int n = 3;
   printf(˝\nHanojski tornjevi (%d elementa)˝, n);
   hanoii('S', 'D', 'T', n);
   return 0;
}
```

Rezultat izvođenja:

```
Hanojski tornjevi (3 elementa)
Prebacujem element 1 sa tornja S na toranj D.
Prebacujem element 2 sa tornja S na toranj T.
Prebacujem element 1 sa tornja D na toranj T.
Prebacujem element 3 sa tornja S na toranj D.
Prebacujem element 1 sa tornja T na toranj S.
Prebacujem element 2 sa tornja T na toranj D.
Prebacujem element 1 sa tornja S na toranj D.
```

Zadatak (REKURZIJA): *problem 8 kraljica*
Napisati funkciju koja će pronaći položaj 8 kraljica na šahovskoj ploči, tako da se one međusobno ne napadaju.

Napomene:
- šahovska ploča je veličine 8 * 8
- postoje 92 različita rješenja
- u svakom retku i svakom stupcu mora se nalaziti samo jedna kraljica

Algoritam rješenja:
- promatramo stupce na šahovskoj ploči od prvog prema zadnjem
- u svaki postavljamo jednu kraljicu
- promatramo ploču u situaciji kada je već postavljeno ***i*** kraljica (u ***i*** različitih stupaca) koje se međusobno ne napadaju
- želimo postaviti ***i*** + 1 kraljicu tako da ona ne napada niti jednu od već postavljenih kraljica i da se ostale kraljice mogu postaviti uz uvjet nenapadanja

Rješenje:
```
//   funkcija   koja   govori   da   li   se   dvije   kraljice,
// postavljene na polja (x1, y1) i (x2, y2)
// međusobno napadaju
int nenapadaju(int x1, int y1, int x2, int y2) {
   int retval = 1;
   if (x1 == x2) retval = 0;
   if (y1 == y2) retval = 0;
   if (x1 + y2 == x2 + y1) retval = 0;
   if (x1 - y2 == x2 - y1) retval = 0;
   return retval;
}

int K8(int *k, int i, int n) {
   int a, b;
   int dobar;
   if (i == n) return 1;
   for (a = 0; a < n; a++) {
      dobar = 1;
      for (b = 0; b < i; b++) {
         if (!nenapadaju(b + 1, k[b] + 1,
                         i + 1, a + 1)) {
            dobar = 0;
            break;
         }
      }
      if (dobar) {
```

```
            k[i] = a;
            if (K8(k,i+1,n) == 1) return 1;
         }
      }
      return 0;
}

#include <stdio.h>
int main() {
   int k[8] = {0}, i;
   K8(k, 0, 8);
   for (i = 0; i < 8; i++)
      printf("(%d, %d)\n", i + 1, k[i] + 1);
  return 0;
}
```

Rezultati izvođenja:

```
(1, 1)
(2, 5)
(3, 8)
(4, 6)
(5, 3)
(6, 7)
(7, 2)
(8, 4)
```

Zadatak (STOG):
Postoje dva stoga čiji su elementi cijeli brojevi. Jedan stog je popunjen, a drugi (pomoćni) je prazan. Napisati funkciju koja će iz prvog stoga izbaciti najveći element koristeći samo pomoćni stog i dvije pomoćne varijable. Funkcija vraća izbačeni element. Stogovi su realizirani kao polja.

Napomena:
Koriste se prototipovi funkcija za skidanje i stavljanje elemenata na stog (definicija funkcija je navedena u 1. zadatku):

```
int dodaj (int stavka, int stog[], int n, int *vrh);
int skini(int *stavka, int stog[], int *vrh);
```

Rješenje:

```
int izbaciNajveci(int stog[],int pomStog[],
                  int n, int *vrh) {
  int pomVrh = -1, elemMax, stavka;

  // prebaci element s vrha stoga na pomocni stog,
  // a ujedno ga oznaci kao inicijalno
  // najveceg elementa na stogu
  skini(&elemMax, stog, vrh);
  dodaj(elemMax, pomStog, n, &pomVrh);

  // nadji najveci element na stogu,
  // a pritom prebacuj sve stavke na pomocni stog
  while (skini(&stavka, stog, vrh)) {
     dodaj(stavka, pomStog, n, &pomVrh);
     if (stavka > elemMax) elemMax = stavka;
  }

  // skidaj s pomocnog stoga sve elemente
  // i sve, osim najveceg,
  // premjestaj nazad na pocetni stog
  // (Ako ima vise najvecih, sve ce ih izbaciti!)
  while (skini(&stavka, pomStog, &pomVrh)) {
     if (stavka < elemMax) dodaj(stavka, stog, n, vrh);
  }
  return elemMax;
}
```

4. Auditorne vježbe

1. Napisati funkciju koja će ispisati presjek dvaju skupova cijelih brojeva, te ustanoviti složenost algoritma.

a) nizovi nisu sortirani - uspoređujemo svaki element iz skupa **a** sa svakim elementom iz skupa **b**

```
int presjek1(int n, int *a, int *b, int **p) {
   int i, j, k = 0; // k – broj zajedničkih elemenata
   for (i = 0; i < n; i ++)
      for (j = 0; j < n; j ++)
         if (a[i] == b[j]) {
            *p = realloc(*p, (k + 1) * sizeof(int));
            (*p)[k] = a[i];
                    // paziti: *p[k] znači *(p[k])!!
            ++ k;
         }
   return k;
}
```

Funkcija `presjek1` ima složenost O (n^2).

Glavni program bi izgledao ovako:

```
#include <stdio.h>
#include <stdlib.h>
#define MAXCLAN 5

void main() {
   int a[MAXCLAN] = {1, 8, 2, 22, 19};
   int b[MAXCLAN] = {2, 19, 3, 34, 35};
   int *p = NULL, i, n;

   n = presjek1(MAXCLAN, a, b, &p);
   printf ("Presjek: ");
   for (i = 0; i < n; i++)
      printf ("%d ", p[i]);
}
```

Ispis programa:

```
Presjek: 2, 19
```

b) nizovi su sortirani – uspoređujemo element iz skupa **a** s elementom iz skupa **b** i ujedno pomičemo pokazivač kod onog skupa čiji je element manji

```
int presjek2(int n, int *a, int *b, int **p) {
   int i = 0, j = 0, k = 0;
   while (i < n && j < n) {
      if (a[i] == b[j]) {
         *p = realloc(*p, (k + 1) * sizeof(int));
         (*p)[k] = a[i];
         ++ k;
                ++ i;
                ++ j;
      } else if (a[i] < b[j]) {
         ++ i;
      } else {
         ++ j;
      }
   }
   return k;
}
```

Funkcija `presjek2` ima složenost O (n), međutim `presjek2` radi samo ako su nizovi sortirani. Ukoliko nizovi nisu sortirani, oni se mogu sortirati pomoću ugrađene funkcije **qsort**.
Glavni program i pomoćna funkcija `compare` izgledaju ovako:

```
int compare (const void *arg1, const void *arg2){
     if (* (int *) arg1 == * (int *) arg2)
      return 0;
     else if (* (int *) arg1 < * (int *) arg2)
      return -1;
     else
      return 1;
}

void main() {
   int a[MAXCLAN] = {1, 8, 2, 22, 19};
   int b[MAXCLAN] = {2, 19, 3, 34, 35};
   int *p = NULL, i, n;

     // void qsort (void *polje, size_t brElemenata, size_t velicinaElemenata,
     //       int (*compare)(const void *elem1, const void *elem2 ) );

   qsort (a, MAXCLAN, sizeof(int), compare);
   qsort (b, MAXCLAN, sizeof(int), compare);

   printf("Polje a nakon sortiranja: ");
   for (i = 0; i < MAXCLAN; i ++)
         printf ("%d ", a[i]);
   printf ("\n");
```

```
   printf("Polje b nakon sortiranja: ");
   for (i = 0; i < MAXCLAN; i ++)
     printf ("%d ", b[i]);
   printf ("\n");

   n = presjek2(MAXCLAN, a, b, &p);

   printf ("Presjek: ");
   for (i = 0; i < n; i ++)
     printf ("%d ", p[i]);
}
```

Ispis programa

```
Polje a nakon sortiranja: 1 2 8 19 22
Polje b nakon sortiranja: 2 3 19 34 35
Presjek: 2, 19
```

Složenost qsorta je n * $\log_2 n$, pa je složenost cijelog programa (ne računajući ispis):
$f(n) = n + 2 * n * \log_2 n = O(n*\log_2 n)$ (konstante i pribrojnici nižeg reda se zanemaruju)

Diskusija: ukoliko polja nisu jednakih dimenzija, nego su na primjer duljina **m** i **n**, tada je složenost za rješenje pod
a) O (m * n)
b) O (MAX(m, n))

2. Napisati funkciju koja računa sumu članova polja.

a) trivijalno

```
int suma1 (int polje[], int n) {
     int i;
     int suma = 0;
     for (i = 0; i < n; i ++) {
        suma += polje[i];
     }
     return suma;
}
```

složenost: O (n)

b) sumiramo prvi član s ostatkom polja (rekurzivno)

```
int suma2 (int polje[], int n) {
     if (n <= 0) return 0;
     return polje[n - 1] + suma2(polje, n - 1);
}
```

Apriorna složenost algoritma je O(n), a prosječno asimptotsko vrijeme izvođenja je 2n.

c) niz dijelimo u polovice i sumiramo polovice

```
int suma3 (int polje[], int l, int d) {
     int p;
     // l - indeks lijevog ruba polja
     // d - indeks desnog ruba polja
     if (d < l) return 0;
     if (d == l) return polje[d];
     p = (d + l) / 2;
      return suma3 (polje, l, p) + suma3 (polje, p + 1, d);
}
```

Poziv funkcije iz glavnog programa:

```
     s = suma3 (polje, 0, MAXCLAN - 1)
```

Apriorna složenost je O(n).

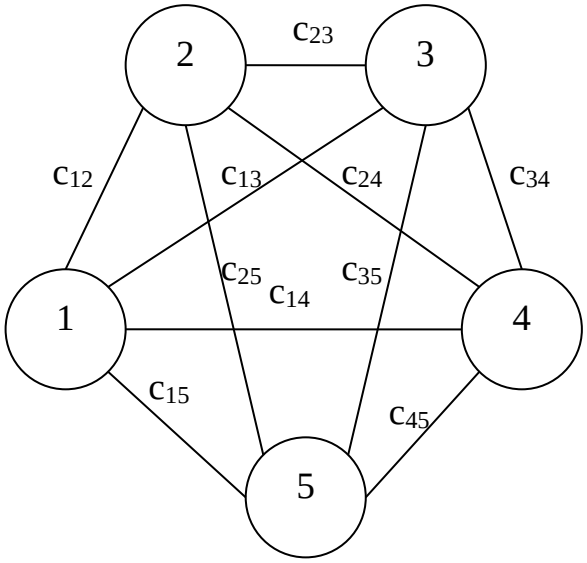
3. Problem trgovačkog putnika (TSP; *Traveling Salesman Problem*): zadan je skup gradova G i cijene c_{ij} putovanja iz grada i u grad j.

Potrebno je, krenuvši iz zadanog grada, obići sve gradove točno jedanput tako da je ukupni trošak puta najmanji.

$$c_{ij} = c_{ji}$$

$$\mathrm{TSP}(G_i, G) = \min_j (c_{ij} + \mathrm{TSP}(G_j, G \setminus G_j))$$

$$\mathrm{TSP}(G_i, \{G_j\}) = c_{ij}$$



Složenost je O((n-1)!).

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

#define MAXGRAD 15       // i to je previše s obzirom na n!
// c - matrica udaljenosti između gradova (i, j)
int c[MAXGRAD][MAXGRAD]; // da ne kompliciramo s prijenosom 2D
polja
```

```
// operacija \ (skup \ element)
void minus (int skup[], int n, int element){
     int i, j;

     for (i = 0; i < n; i ++) {
        if (skup[i] == element) break;
      }

     for (j = i; j < n - 1; j ++) {
        skup[j] = skup[j + 1];
   }
}

int TSP (int IzGrada, int *gradovi, int n) {
     int *lgradovi, minTSP, pomTSP, i;

     lgradovi = malloc (n * sizeof(int));
   memcpy (lgradovi, gradovi, n * sizeof(int));
     minus (lgradovi, n, IzGrada);
   -- n;

   if (n == 1) {
        minTSP = c[IzGrada][lgradovi[0]];
   }
   else {
        minTSP = c[IzGrada][lgradovi[0]]
                  + TSP (lgradovi[0], lgradovi, n);

        for (i = 1; i < n; i++) {
          pomTSP = c[IzGrada][lgradovi[i]]
                      + TSP (lgradovi[i], lgradovi, n);
          if (pomTSP < minTSP) minTSP = pomTSP;
        }
   }
     free (lgradovi);
     return minTSP;
}
```

```
main () {
   int gradovi[MAXGRAD], n, i, j, minTSP;
     srand (time(NULL));

     do {
        printf ("Unesite broj gradova: ");
        scanf ("%d", &n);
   } while (n < 2 || n > MAXGRAD);

   // generiranje matrice
   for (i = 0; i < n; i++) {
        gradovi[i] = i;
        c[i][i] = 0;

        for (j = i + 1; j < n; j ++) {
          c[i][j] = rand() + 1;
          c[j][i] = c[i][j];
          printf ("c[%d][%d] = %d\n", i, j, c[i][j]);
        }
   }

   minTSP = TSP (0, gradovi, n);
   printf ("Najmanji trošak je: %d\n", minTSP);
}
```

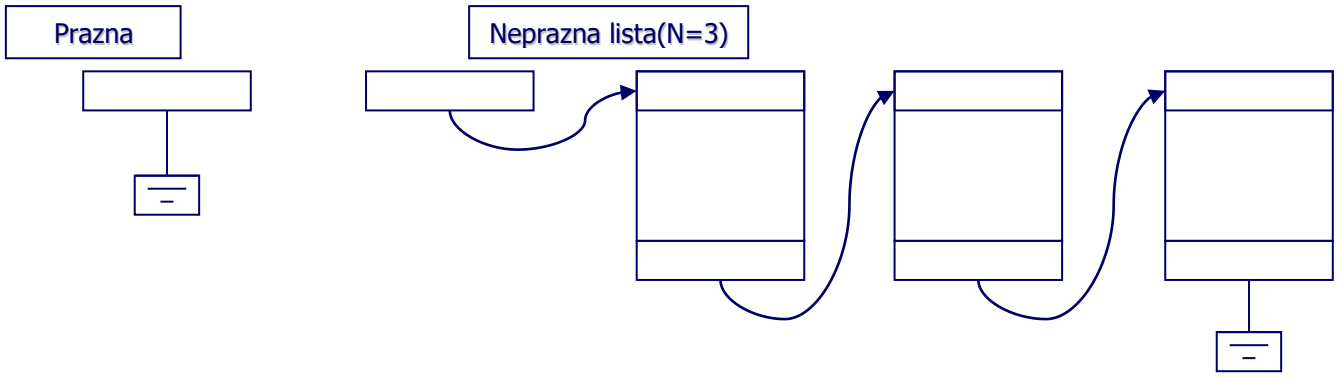
4. Napisati funkciju koja računa zbroj znamenki zadanog prirodnog broja. Kolika je složenost funkcije?

```
int zbrojZnamenki (int N) {
      long zbroj = 0;

      while (N > 0) {
            zbroj += N % 10;
            N /= 10;
      }
}
```

RJEŠENJE: $O(\log_{10}N)$.

5. Auditorne vježbe

Jednostruko povezane liste



Zadatak:

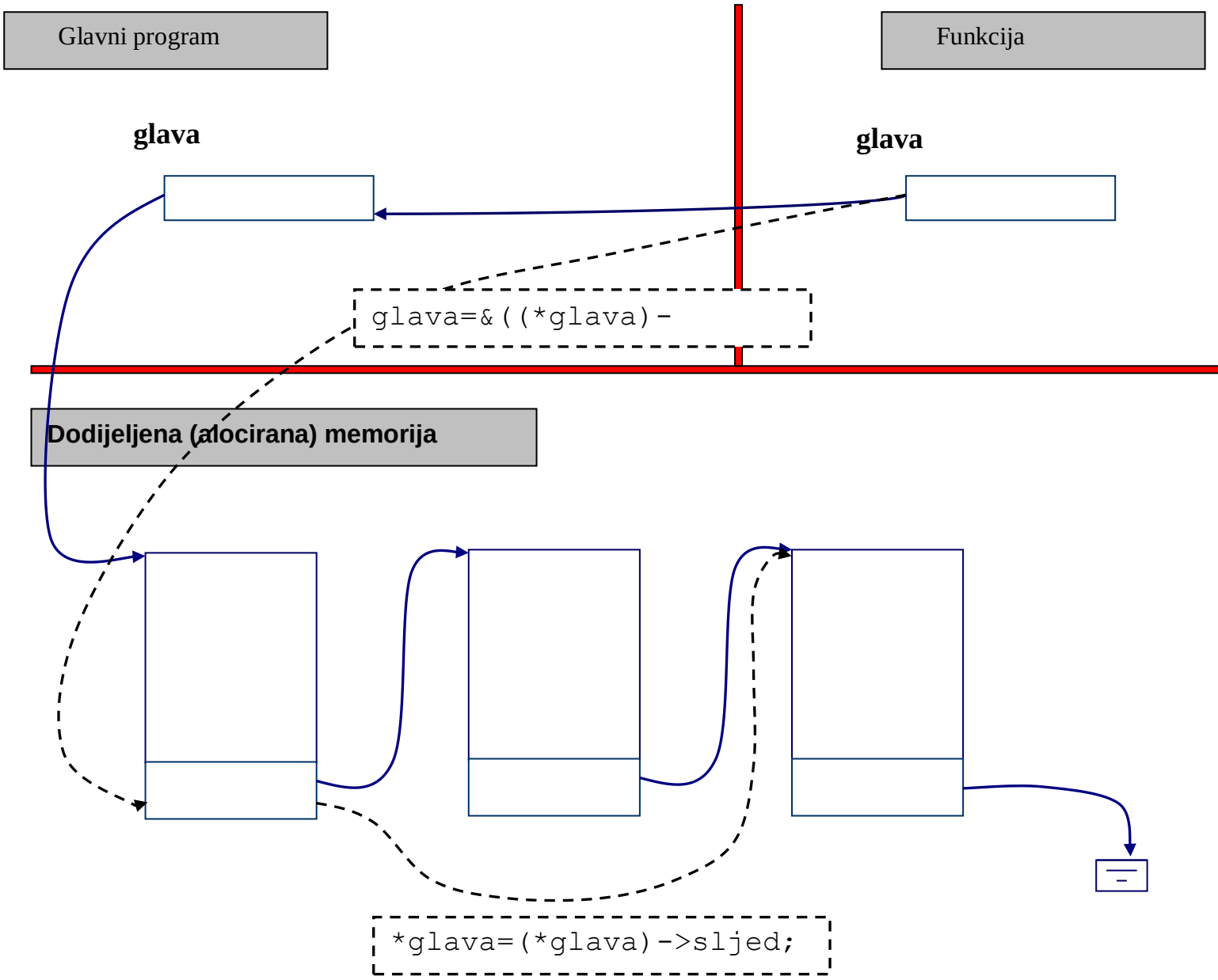
U jednostruko povezanu listu spremaju se podaci o rezultatima ispita. Jedan element liste sadrži slijedeće podatke: **ime i prezime studenta** (50+1 znak), **matični broj studenta** (`long int`), **šifru predmeta** (`long`) i **ocjenu** (`int`). Lista nije sortirana. Napisati funkciju koja će iz liste izbaciti negativno ocijenjene studente (one koji imaju ocjenu 1). Funkcija vraća broj izbačenih članova liste.

```
struct s1{
   char   imeprezime[50+1];
   long   matbr;
   long   sifpred;
   int    ocjena;
   struct s1 *sljed;
};
typedef struct s1 cvor;
```

```
int izbacineg(cvor **glava){
   cvor *p, *pp;
  int izb = 0;

  p = *glava;
  pp = NULL;
  while(p) {
    if (p->ocjena == 1) {
      izb++;
      if (p == *glava){ // ili pp == NULL,  brisanje glave
        *glava = p->sljed;
        free(p);
        p = *glava;
      } else { // brisanje unutar liste
        pp->sljed = p->sljed;
        free(p);
        p = pp->sljed;
      }
    } else {
      pp = p;
      p = p->sljed;
    }
  }
   return izb;
}
```

Drugo rješenje (manje koda):

```
int izbacineg(cvor **glava){
  int izb = 0;
  cvor *tmp;
  while(*glava) {
    if ((*glava)->ocjena == 1) {
      izb++;
      tmp = *glava;
      *glava = (*glava)->sljed;
      free(tmp);
    } else {
      glava = &((*glava)->sljed);
    }
  }
  return izb;
}
```

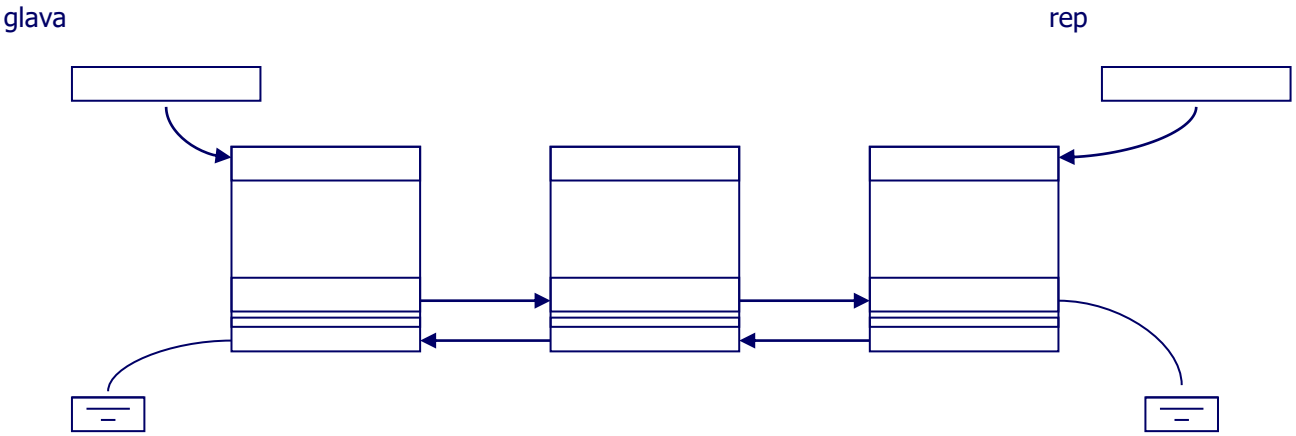


Glava iz funkcije prvo pokazuje na glavu iz glavnog programa, a zatim redom na pokazivače sljed, koji su članovi strukture. Ukoliko prvi čvor ima negativnu ocjenu naredba:

```
*glava=(*glava)->sljed;
```

promijenit će pokazivač glava iz glavnog programa, inače mijenja pokazivač sljed koji je član strukture.

Dvostruko povezane liste



Zadatak:

U čvorovima dvostruko povezane liste nalaze se slijedeći podaci: **šifra poduzeća** (`int`), **naziv poduzeća** (40+1 znakova) i **broj žiro računa poduzeća** (15+1 znakova). Lista je sortirana po šifri poduzeća. Napisati funkciju koja će u listu, tako da ona ostane sortirana, ubaciti podatke o novom poduzeću (šifra, naziv i broj žiro računa zadaju se preko argumenata). Ako poduzeće s istom šifrom već postoji u listi, novi se podaci prepisuju preko starih. Funkcija vraća 1, ako je ubacivanje uspješno obavljeno, odnosno 0, ako ubacivanje nije obavljeno.

```
typedef struct {
   char   naziv[40+1];
   char   ziro[15+1];
   long   sifpod ;
} element;

typedef struct s2{
   element podaci;
   struct s2 *sljed;
   struct s2 *preth;
} cvor;
```

```
int ubaci(cvor **glava, cvor **rep, element novipodaci) {
  cvor *novi, *p;
  if (*glava == NULL || (*glava)->podaci.sifpod >
novipodaci.sifpod) {
    // Dodavanje na pocetak liste
    if ((novi = (cvor *) malloc(sizeof(cvor)))==NULL) return 0;
    novi->podaci = novipodaci;
    novi->sljed = *glava;
    novi->preth = NULL;
    *glava = novi;
    if (*rep == NULL)
      *rep = novi;
    else
      (novi->sljed)->preth = novi;
  } else {
    for (p = *glava; p->sljed && (p->sljed)->podaci.sifpod <=
novipodaci.sifpod; p = p->sljed);
    if (p->podaci.sifpod == novipodaci.sifpod) {  // postojeći
element
      p->podaci = novipodaci;
    } else {
      if ((novi = (cvor *) malloc(sizeof(cvor))) == NULL) return
0;
      novi->podaci = novipodaci;
      novi->sljed = p->sljed;
      p->sljed = novi;
      novi->preth = p;
      if (novi->sljed) {
        (novi->sljed)->preth = novi; // ako nije dodan kao
zadnji
      } else {
        *rep = novi;
      }
    }
  }
  return 1;
}
```

Stog realiziran listom

Zadatak:
Stog realiziran jednostruko povezanom listom sadrži podatke cjelobrojnog tipa (int). Napisati funkcije za skidanje elementa sa stoga i stavljanje novog elementa na stog.

```
int dodaj (cvor **vrh, int element) {
  cvor *novi;
  if ((novi= (cvor *) malloc(sizeof(cvor))) == NULL) return 0;
  novi->element = element;
  novi->sljed = *vrh;
  *vrh = novi;
  return 1;
}

int skini (cvor **vrh, int *element) {
  cvor *pom;
  if (*vrh == NULL) return 0; // ako je stog prazan
  *element = (*vrh)->element;
  pom = *vrh;
  *vrh = (*vrh)->sljed; // pomicanje pokazivača na vrh stoga
  free (pom);
  return 1; // uspješno skinut element s vrha stoga
}
```

Rekurzivno pretraživanje liste:

Jednostruko povezana lista u memoriji sadrži statističke podatke o svim dosadašnjim nastupima jednog tenisača na turnirima. Element liste sadrži **šifru turnira** (int) i **broj pobjeda na tom turniru** (int). Napisati ***rekurzivnu*** funkciju koja će vratiti ukupan broj pobjeda tog tenisača na svim turnirima.

Prototip funkcije je zadan i nije ga dozvoljeno mijenjati:

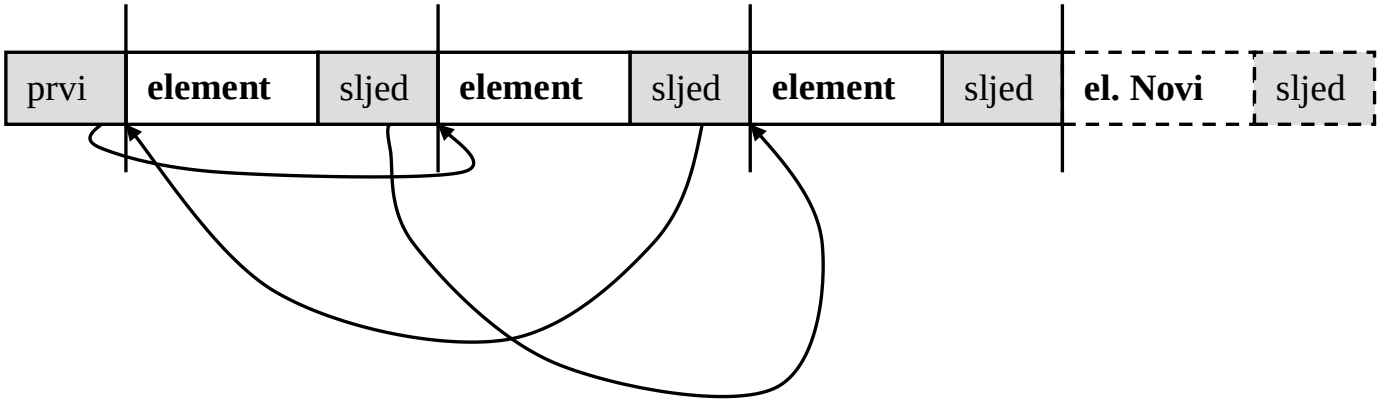
```
int broj_pobjeda(cvor *glava);
```

Rješenje:

```
struct s1 {
   int sifturn;
   int brpobj;
   struct s1 *sljed;
};
typedef struct s1 cvor;

int broj_pobjeda(cvor *glava){
  if (glava)
    return glava->brpobj + broj_pobjeda(glava->sljed);
  else
    return 0;
}
```

Lista u datoteci

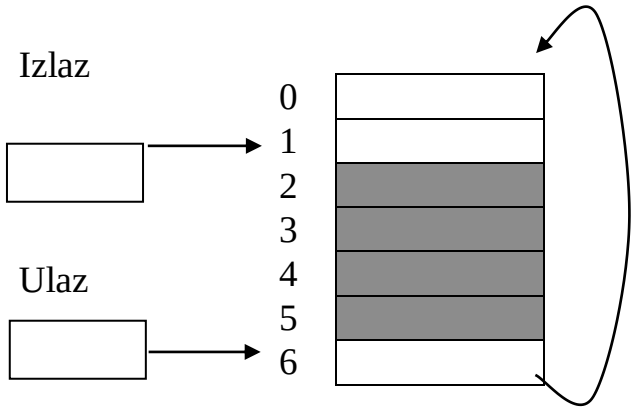
U datoteci se nalazi jednostruko povezana lista uređena po rastućoj vrijednosti matičnog broja. Potrebno je napisati funkciju koja će omogućiti dodavanje novog člana u listu.



```
typedef struct {
  int kljuc;
} tip;

typedef struct {
  tip element;
  long sljed;
} cvor;

int dodaj (tip element, FILE *fl) {
  // Dodavanje u listu sortiranu po rastucoj
  // vrijednosti maticnog broja
  long noviadr, prethodni, sljedeci;
  cvor c, novi;

  // Fizički upis i dodavanje po maticnom broju
  fseek(fl, 0L, SEEK_END);
  noviadr = ftell(fl);
  novi.sljed = 0;
  novi.element = element;
  prethodni = 0;
  fseek (fl, 0L, SEEK_SET);
  fread (&sljedeci, sizeof(sljedeci), 1, fl);
  while (sljedeci) {
    fseek (fl, sljedeci, SEEK_SET);
    fread (&c, sizeof (c), 1, fl);
    if (c.element.kljuc > element.kljuc) {
      // novi se ubacuje ispred
      novi.sljed = sljedeci;
      break;
    }
    prethodni = sljedeci + sizeof (c.element);
    sljedeci = c.sljed;
  }
  // Povezivanje prethodnog
  fseek (fl, prethodni, SEEK_SET);
  if (fwrite (&noviadr, sizeof (noviadr), 1, fl) != 1)
     return 0;
  fseek (fl, noviadr, SEEK_SET);
  if (fwrite (&novi, sizeof(novi), 1, fl) != 1) return 0;
  return 1;
}
```

6. Auditorne vježbe

Red realiziran cikličkim poljem

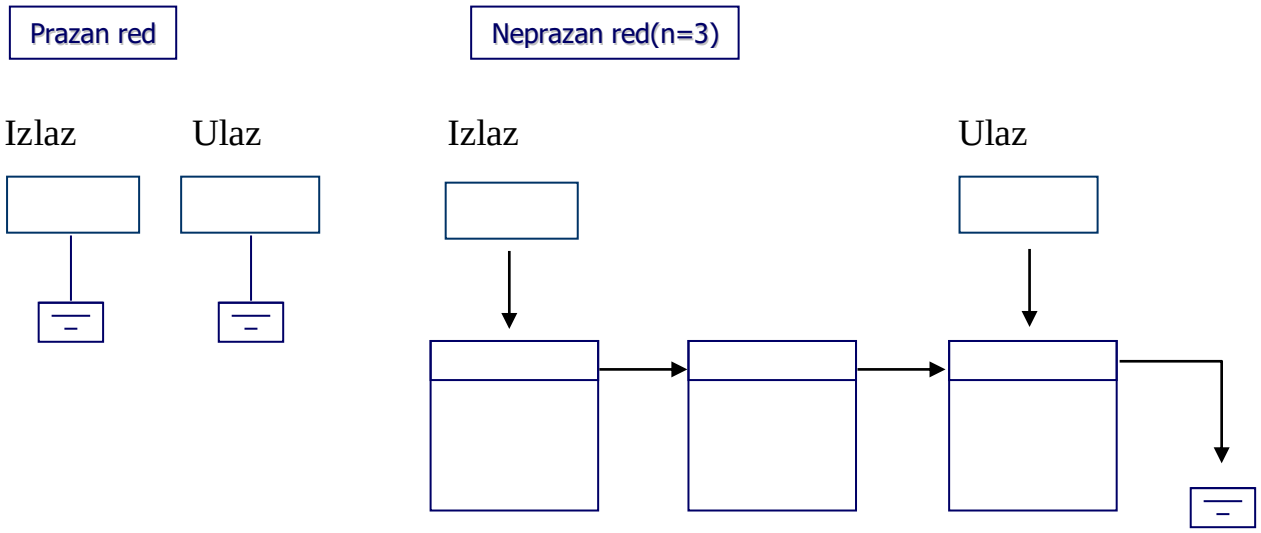


Zadatak:

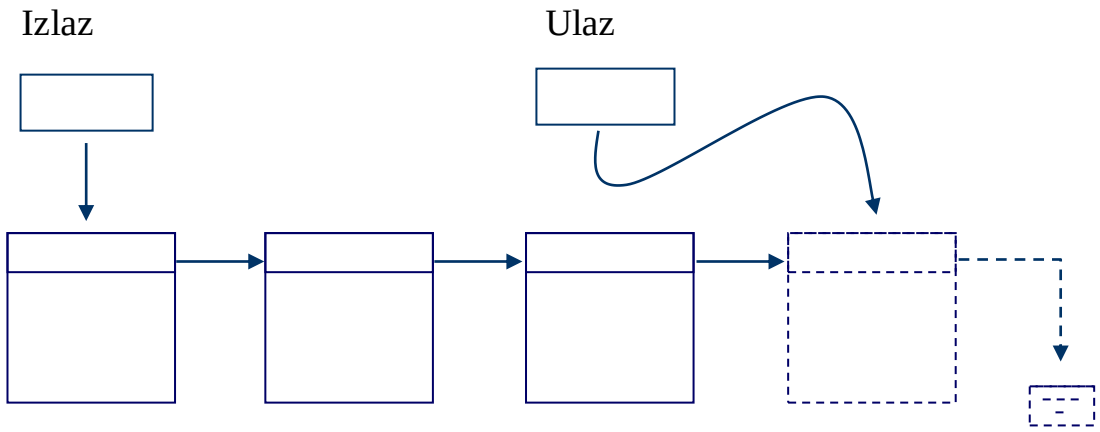
Napisati funkciju koja dodaje element u red i funkciju koja skida element iz reda realiziranog cikličkim poljem. Napisati funkciju koja za zadani red (realiziran cikličkim poljem) vraća postotak popunjenosti tog reda.

```
int dodajURed(zapis *red, int *ulaz, int izlaz, zapis element, int
velicinaReda) {
   //red je pun
   if((*ulaz+1)%velicinaReda == izlaz) return 0;
     *ulaz+=1;
     *ulaz%=velicinaReda;
     red[*ulaz]=element;
   return 1;
}
int SkiniIzReda (zapis *element, zapis red[], int velicinaReda, int
*izlaz, int ulaz) {
  if (ulaz == *izlaz) return 0;
  (*izlaz) ++;
  *izlaz %= velicinaReda;
  *element = red[*izlaz];
  return 1;
}
float Popunjenost(int ulaz, int izlaz, int velicinaReda) {
   int brElemenata;
   if(ulaz >= izlaz) {
      brElemenata = ulaz - izlaz;
   } else {
      brElemenata = velicinaReda - (izlaz - ulaz);
   }
   return (float)brElemenata/velicinaReda;
}
```

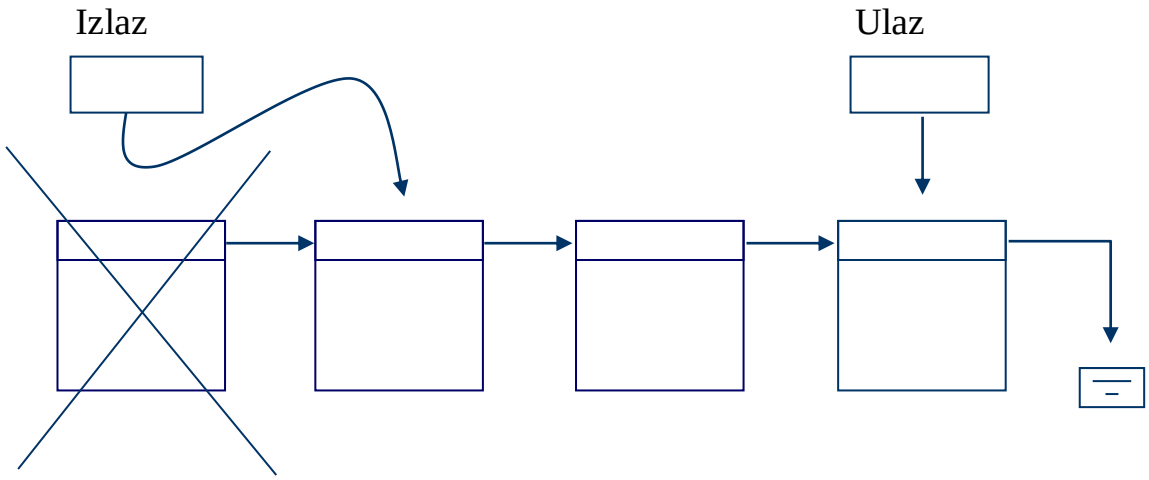
Red realiziran jednostruko povezanom listom



Dodavanje u red



Brisanje iz reda

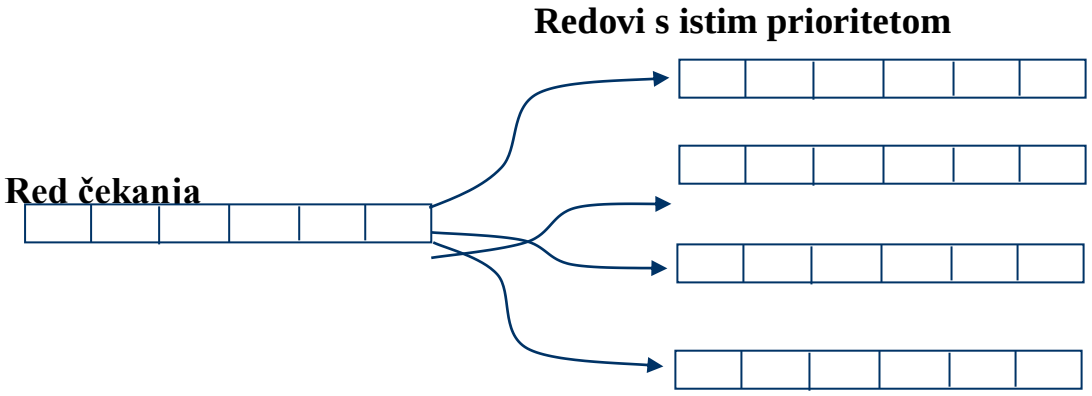


Funkcije za stavljanje elementa u red i skidanje elementa iz reda realziranog jednostruko povezanom listom:

```
struct cv {
   int element;
   struct cv *sljed;
};
typedef struct cv cvor;

int DodajURed (int element, cvor **ulaz, cvor **izlaz) {
   cvor *novi;
   if (novi = malloc (sizeof (cvor))) {
      novi->element = element;
      novi->sljed = NULL;
      if (*izlaz == NULL) {
         *izlaz = novi;          // Ako je red bio prazan
      } else {
         (*ulaz)->sljed = novi;  // inace, stavi na kraj
      }
      *ulaz = novi;                // Zapamti zadnjeg
      return 1;
   }
   return 0;
}

int SkiniIzReda (int *element, cvor **ulaz, cvor **izlaz) {
   cvor *stari;
   if (*izlaz) {
      *element = (*izlaz)->element;
      stari = *izlaz;
      *izlaz = (*izlaz)->sljed;
      free (stari);
      if (*izlaz == NULL) *ulaz = NULL;
      return 1;
   }
   return 0;
}
```

Zadatak

Procesi (**identifikator procesa**, **adresa dodijeljenog memorijskog prostora** i **prioritet procesa**) dolaze u red čekanja i nakon toga se raspoređuju u odgovarajuće redove s istim prioritetom. Napisati funkciju za stavljanje procesa u red čekanja i funkciju za stavljanje procesa u red s istim prioritetom. Koristeći funkciju za stavljanje procesa u redove s istim prioritetom realizirajte funkciju za skidanje iz reda čekanja. Svi redovi su realizirani dvostruko povezanom listom. Pokazivači na ulaz i izlaz redova s istim prioritetom čuvaju se u polju.



Dvodimenzionalno polje čiji su elementi pokazivači na ulaz i izlaz iz reda s istim prioritetom:

0	*ulaz[0]	*izlaz[0]	-> pokazivači na ulaz i izlaz reda s prioritetom 1
1	*ulaz[1]	*izlaz[1]	
2	*ulaz[2]	*izlaz[2]	
3	*ulaz[3]	*izlaz[3]	

```
struct pr {
   long PID;
   long adresa;
   char prioritet;
   struct pr *sljed;
   struct pr *preth;
};
typedef pr proc;

void dodajUPrioritetni(proc *novi, proc **ulaz, proc **izlaz){
   novi->sljed=NULL;
   novi->preth=NULL;
   if(*izlaz==NULL) {
      *izlaz=novi;
      *ulaz=novi;
   } else {
      (*ulaz)->sljed=novi;
      novi->preth=*ulaz;
      *ulaz=novi;
   }
}
int dodajURedCekanja(long PID, long adresa, char
prioritet,proc **ulaz,proc **izlaz) {

   proc *novi;
   if(novi=(proc *)malloc(sizeof(proc))) {
      novi->PID=PID;
      novi->adresa=adresa;
      novi->prioritet=prioritet;
      novi->sljed=NULL;
      novi->preth=NULL;
      if(*izlaz==NULL) {
         *izlaz=novi;
         *ulaz=novi;
      } else {
         (*ulaz)->sljed=novi;
         novi->preth=*ulaz;
         *ulaz=novi;
      }
      return 1;
   }
   return 0;
}
```

```
int skiniSRedaCekanja(proc **ulazRed, proc **izlazRed
                    , proc *prioritetniRedovi[]) {

proc *prebaci;
if(*izlazRed) {
  prebaci = *izlazRed;
  if(*ulazRed==*izlazRed) {
    *ulazRed=NULL;
    *izlazRed=NULL;
  } else {
    ((*izlazRed)->sljed)->preth=NULL;
    *izlazRed=(*izlazRed)->sljed;
  }
//memorija se ne oslobađa jer se element reda seli iz jednog
//reda u drugi red
  dodajUPrioritetni(prebaci
             , &prioritetniRedovi[prebaci->prioritet*2]
             , &prioritetniRedovi[prebaci->prioritet*2+1]);
    return 1;
}
return 0;
}
```

Zadatak sa 1. KZ 2002/2003 ak.godine

Red je realiziran kao polje, koje može primiti najviše MAXZAPIS zapisa. Svaki zapis sadrži sljedeće podatke: **šifru procesa** (`long`) i **opis procesa** (40 + 1 znak). Potrebno je napisati funkciju koja dodaje zapis u red. Funkcija treba vratiti 1, ako je zapis dodan u red, a inače 0.

Rješenje:

```
struct sProces {
   long sifra;
   char opis[40+1];
};
typedef struct sProces proces;

int dodajURed (proces element, proces red[], int n, int
izlaz, int *ulaz) {
   if (((*ulaz+1) % n) == izlaz) return 0;
   (*ulaz)++;
   *ulaz %= n;
   red [*ulaz] = element;
   return 1;
}
```

složenost: O(1)

primjer poziva iz glavnog programa:

```
#define MAXZAPIS 10
void main() {
   proces element, red[MAXZAPIS];
int izlaz, ulaz, i;
   ...
   i = dodajURed (element, red, MAXZAPIS, izlaz, &ulaz);
...
}
```

Zadatak sa 1.KZ 2001/2002 ak.godine

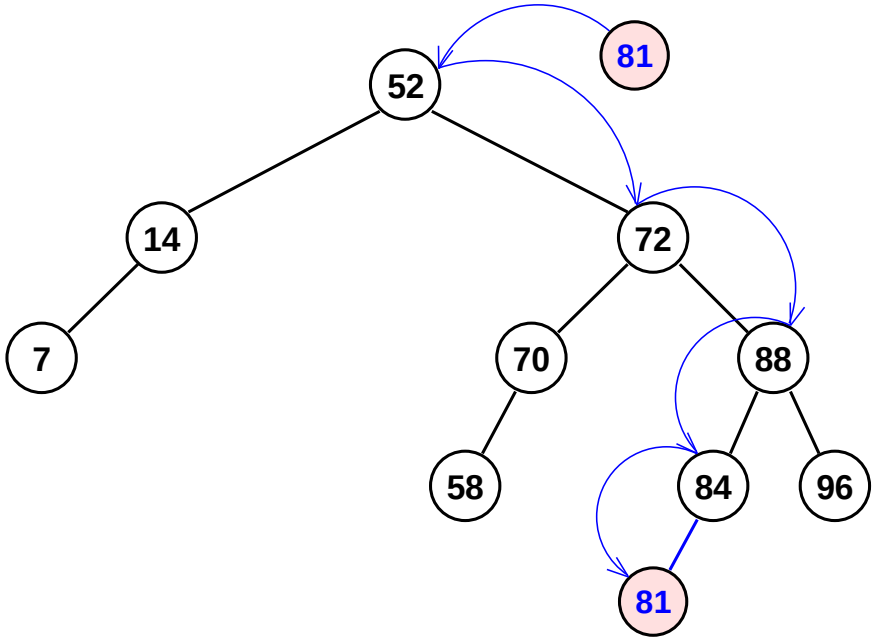
U red realiziran jednostruko povezanom listom spremaju se zapisi o studentima. Svaki zapis sadrži **matični broj** (10+1 znak), **ime i prezime studenta** (40+1 znak), **prosjek ocjena** (float) te **godinu prvog upisa u prvu godinu studija** (int). Napisati funkciju za skidanje elementa iz reda. Ako je operacija uspješno obavljena funkcija vraća 1, a ako operacija nije obavljena funkcija vraća 0. Definirati strukture potrebne za rješenje ovog zadatka. Odrediti apriornu složenost funkcije. Funkcija mora imati prototip:

```
int skini(zapis **ulaz, zapis **izlaz, zapis *element);

typedef struct {
   char   mbr[10+1];
   char   imepr[40+1];
   float    pr_ocjena;
   int    god1upis;
} zapis1;

typedef struct s{
   zapis1   podaci;
   struct s  *sljed;
} zapis;

int skini( zapis **ulaz, zapis **izlaz, zapis *element) {
  zapis *stari;
  if (*izlaz != NULL) {
    element->podaci  = (*izlaz)->podaci;
    stari = *izlaz;
    *izlaz = (*izlaz)->sljed;
    free (stari);
    if (*izlaz == NULL) *ulaz = NULL;
      return 1;
  }else{
    return 0;
  }
}
```

složenost: O(1)

7. Auditorne vježbe

Zadatak: Nacrtati izgled binarnog sortiranog stabla nakon dodavanja sljedećih podataka: 52, 72, 14, 88, 70, 88, 84, 7, 58 i 96. (Pitanje: Što se zbiva s drugim podatkom 88?)
Nacrtati postupak dodavanja podatka 81 u tako već formirano binarno stablo.



Zadatak: Ukoliko se stablo prikazuje jednodimenzionalnim poljem, s koliko se članova mora dimenzionirati u najgorem i najboljem slučaju ako je potrebno pohraniti 52 različita elementa?

Općenito u stablo dubine *k* stane 2^k-1 članova.

Najbolji slučaj – stablo će biti **potpuno,** pa je za pohranu potrebna dubina koja je jednaka najmanjoj potenciji broja 2 koja je veća od 52, a to je $64=2^6$. Dakle, u stablo dubine 6 stane 63 elementa te je to dovoljno za pohranu 52 elementa. U najboljem slučaju potrebno je polje dimenzionirati na 63.

Najgori slučaj – stablo će biti potpuno **koso** i dubina stabla će biti 52. Za pohranu je potrebno polje dimenzionirati s 2^{52}-1= 4,503,599,627,370,495

Zadatak: Pretražuje se stablo u kojem se nalazi 27 različitih elemenata. Potrebno je odrediti koliko se operacija uspoređivanja obavi u najboljem i najgorem slučaju za potpuno i koso stablo.

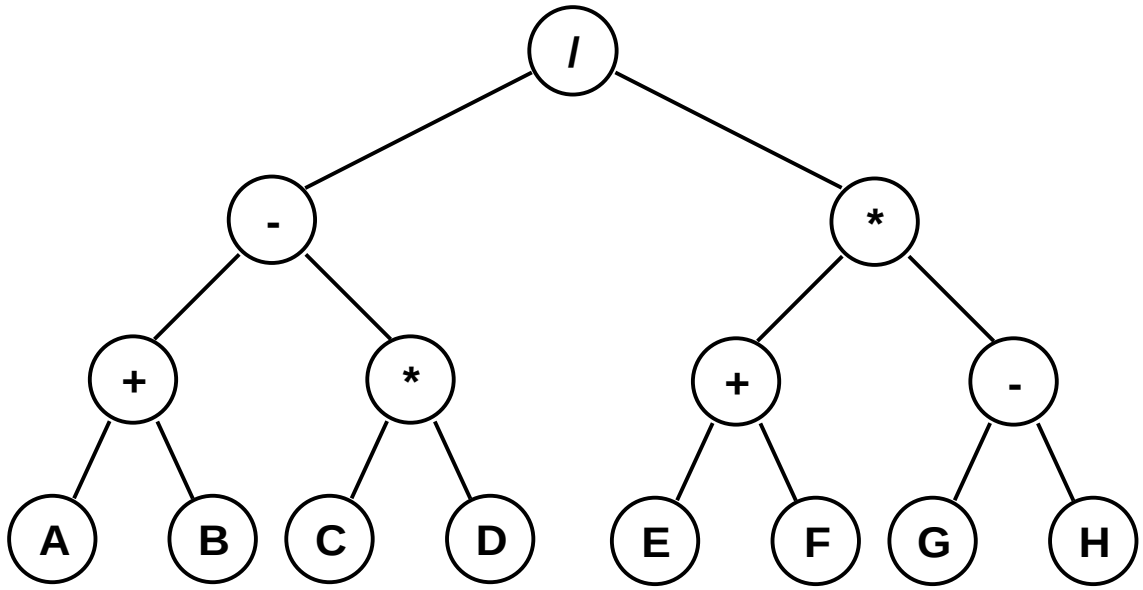
Potpuno stablo:

a) najbolji slučaj – pri prvoj usporedbi nađe se element ⇒ obavljena je **jedna** usporedba
b) najgori slučaj – podatak se nalazi u "najdubljem" *listu*. Najmanja potencija broja 2 koja je veća od 27 je $32=2^5$. Dakle stablo ima dubinu 5 te je potrebno obaviti najviše **pet** operacija uspoređivanja.

Koso stablo (u potpunosti koso):

a) najbolji slučaj – pri prvoj usporedbi nađe se element ⇒ obavljena je **jedna** usporedba
b) najgori slučaj – podatak se nalazi u "najdubljem" *listu*. Budući da je u najgorem slučaju stablo u potpunosti koso, razina najdubljeg čvora je 27. Dakle, u najgorem slučaju potrebno je obaviti **27** operacija uspoređivanja.

Zadatak: Za već formirano binarno stablo potrebno je napisati što se dobije *inorder* i *postorder* obilaskom. Stablo izgleda na sljedeći način:



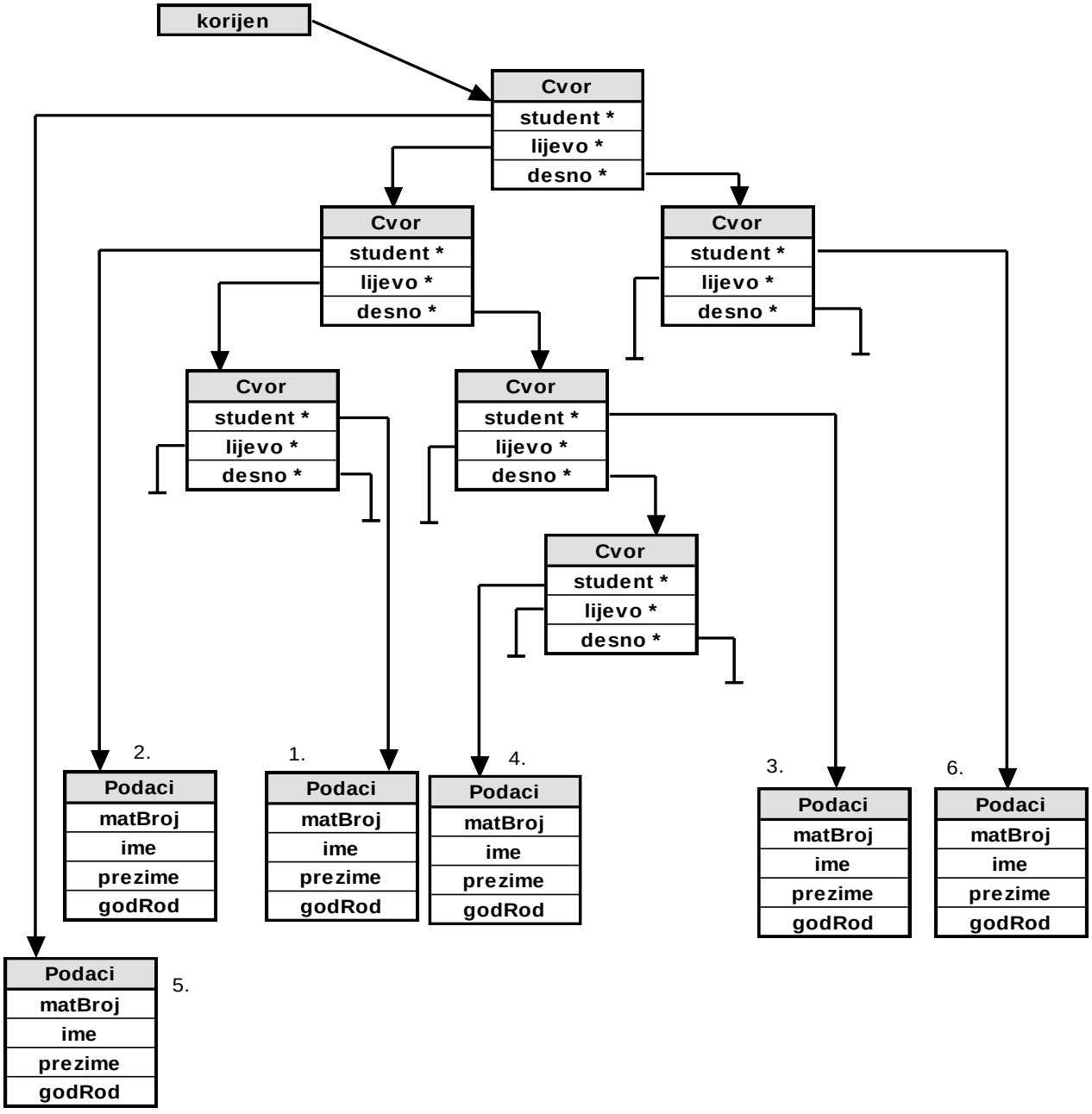
Inorder obilazak stabla daje matematički izraz:
A + B - C * D / E + F * G – H

Postorder obilazak također daje matematički izraz ali u tzv. RPN notaciji:
A B + C D * - E F + G H - * /

Zadatak: U SORTIRANOM (lijevo manji, desno veći) binarnom stablu realiziranom dinamičkom strukturom pohranjuju se matični brojevi (long). Napisati funkciju koja će pronaći element sa zadanim matičnim brojem. Koja je složenost takve funkcije?

```
typedef struct s{
  long      mbr;
  struct s  *lijevo;
  struct s  *desno;
} cvor;

cvor *trazi (cvor *glava, long mbr) {
  if (glava) {
    if ( glava->mbr > mbr ) {
      return trazi (glava->lijevo, mbr);
    } else if (glava->mbr > mbr) {
      return trazi (glava->desno, mbr);
    }
  }
  return glava; // ili je pronadjen ili NULL;
}
```

Zadatak: Zadani su podaci o studentu: matični broj long, ime 25 znakova, prezime 25 znakova i godina rođenja short.Uz uporabu dinamičke strukture stabla potrebno je napisati funkcije za:
a) pohranu podataka o studentima uz uvjet da je omogućeno brže pronalaženje studenata po prezimenu
b) pronalaženje studenta po prezimenu
c) ispis čvorova na zadanoj razini
d) izračunavanje dubine stabla i najmanje razine listova
e) ispis listova stabla
Realizacija:



Napomena: brojevi uz podatke označavaju redoslijed *inorder* obilaska stabla

Podatke je moguće pohraniti i u čvorove stabla (kao što je prikazano na predavanju), no u tom se slučaju isti podaci pohranjuju na dva mjesta (ili se komplicira izrada drugog binarnog stabla nad istim podacima).

Strukture potrebne pri takvoj realizaciji

```
struct podaci {
   long matBroj;
   char ime[25];
   char prezime[25];
   short godRod;
};
typedef struct podaci Podaci;

struct cvor {
   Podaci *student;
   struct cvor *lijevo;
   struct cvor *desno;
};
typedef struct cvor Cvor;
```

Funkcija dodaje u binarno sortirano stablo s ključem *prezime*:

```
Cvor *dodajPrezime (Cvor *korijen, Podaci *student) {
  int s;  // smjer
  if (korijen == NULL) {
    korijen = (Cvor *) malloc (sizeof(Cvor));
    if (korijen) {
      korijen->student = student;
      korijen->lijevo = korijen->desno = NULL;
    }
  } else if ((s=strcmp(student->prezime,korijen->student-
>prezime)) < 0)
            korijen->lijevo = dodajPrezime (korijen->lijevo,
student);
         else if (s > 0)
                korijen->desno = dodajPrezime (korijen->desno,
student);
              else
                printf("Podatak %s vec postoji!\n", student-
>prezime);
  return korijen;
}
```

Funkcija pretražuje binarno sortirano stablo po ključu *prezime*:

```
Cvor *potraziPrezime (Cvor *korijen, char *prezime) {
  int smjer;
  if (korijen) {
    if ((smjer = strcmp(prezime, korijen->student->prezime)) <
0)
       return potraziPrezime (korijen->lijevo, prezime);
    else if (smjer > 0)
            return potraziPrezime (korijen->desno, prezime);
  }
  return korijen;
}
```

Funkcija ispisuje čvorove zadane razine:

```
// poziv ispis(korijen, 1, n)
// gdje je n trazena dubina
void ispisi (Cvor *korijen, int trenutnaDubina, int
trazenaDubina) {
  if (korijen) {
    if (trenutnaDubina == trazenaDubina)
      printf("%s %s\n",
                    korijen->student->ime, korijen->student-
>prezime);
    ispisi (korijen->lijevo, trenutnaDubina + 1,
trazenaDubina);
    ispisi (korijen->desno, trenutnaDubina + 1,
trazenaDubina);
  }
}
```

Može li se prethodna funkcija napisati bolje (tako da ne ide dublje od tražene dubine)?

Funkcija izračunava najveću i najmanju dubinu stabla (razinu listova stabla):

```
// poziv balansirano (korijen, 1, &maxDub, &minDub);
void balansirano (Cvor *korijen, int trenDub,
                                        int *maxDub, int *minDub)
{
  if (korijen) {
    if (!korijen->lijevo & !korijen->desno) {
      if (*maxDub == 0 || trenDub > *maxDub)
        *maxDub = trenDub;
      if (*minDub == 0 || trenDub < *minDub)
        *minDub = trenDub;
    }
   else {
      balansirano (korijen->lijevo, trenDub + 1, maxDub,
minDub);
      balansirano (korijen->desno, trenDub + 1, maxDub,
minDub);
    }
  }
}
```

Funkcija ispisuje listove stabla:

```
void ispisiListove(Cvor *korijen) {
  if (korijen) {
    if (!korijen->lijevo & !korijen->desno)
      printf("%s %s; ",
                korijen->student->ime, korijen->student-
>prezime);
     ispisiListove(korijen->lijevo);
     ispisiListove(korijen->desno);
    }
}
```

Dodatak: Kompletan 1. program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct podaci {
  long matBroj;
  char ime[25];
  char prezime[25];
  short godRod;
};
typedef struct podaci Podaci;

struct cvor {
  Podaci * student;
  struct cvor * lijevo;
  struct cvor * desno;
};
typedef struct cvor Cvor;

// funkcija dodaje u binarno sortirano stablo
// kljuc je prezime
Cvor * dodajPrezime (Cvor * korijen, Podaci * student) {
  int smjer;
  if (korijen == NULL) {
    korijen = (Cvor *) malloc (sizeof(Cvor));
    if (korijen) {
      korijen->student = student;
      korijen->lijevo = korijen->desno = NULL;
    }
  } else if ((smjer = strcmp(student->prezime, korijen->student-
>prezime)) < 0)
            korijen->lijevo = dodajPrezime (korijen->lijevo, student);
         else if (smjer > 0)
                 korijen->desno = dodajPrezime (korijen->desno,
student);
              else
                 printf("Podatak %s vec postoji!\n", student->prezime);
  return korijen;
}

// funkcija dodaje u binarno sortirano stablo
// kljuc je ime
Cvor * dodajIme (Cvor * korijen, Podaci * student) {
  int smjer;
  if (korijen == NULL) {
    korijen = (Cvor *) malloc (sizeof(Cvor));
    if (korijen) {
      korijen->student = student;
      korijen->lijevo = korijen->desno = NULL;
    }
```

```
  } else if ((smjer = strcmp(student->ime, korijen->student->ime)) < 0)
            korijen->lijevo = dodajIme (korijen->lijevo, student);
          else if (smjer > 0)
                  korijen->desno = dodajIme (korijen->desno, student);
               else
                  printf("Podatak %s vec postoji!\n", student->ime);
  return korijen;
}

// funkcija pretrazuje binarno sortirano stablo po kljucu prezime
Cvor * potraziPrezime (Cvor * korijen, char * prezime) {
  int smjer;
  if (korijen) {
    if ((smjer = strcmp(prezime, korijen->student->prezime)) < 0)
       return potraziPrezime (korijen->lijevo, prezime);
    else if (smjer > 0)
            return potraziPrezime (korijen->desno, prezime);
  }
  return korijen;
}

// funkcija inorder ispisuje zadano binarno stablo
void inOrder (Cvor * korijen) {
  if (korijen) {
    inOrder (korijen->lijevo);
    printf("%s, %s;", korijen->student->prezime, korijen->student->ime);
    inOrder (korijen->desno);
  }
}

// funkcija ispisuje sve cvorove zadane dubine
// poziv ispis(korijen, 1, n)
// gdje je n trazena dubina
void ispisi (Cvor * korijen, int trenutnaDubina, int trazenaDubina) {
  if (korijen) {
    if (trenutnaDubina == trazenaDubina)
      printf("%s %s\n", korijen->student->ime, korijen->student-
>prezime);
    ispisi (korijen->lijevo, trenutnaDubina + 1, trazenaDubina);
    ispisi (korijen->desno, trenutnaDubina + 1, trazenaDubina);
  }
}

// funkcija ispisuje sve cvorove na zadanoj dubini a ispred
// podataka stavlja odgovarajuci broj praznina
// poziv ispis2(korijen, 1, n)
// gdje je n trazena dubina
void ispisi2 (Cvor * korijen, int trenutnaDubina, int trazenaDubina) {
  int i;
  if (korijen) {
    if (trenutnaDubina == trazenaDubina) {
```

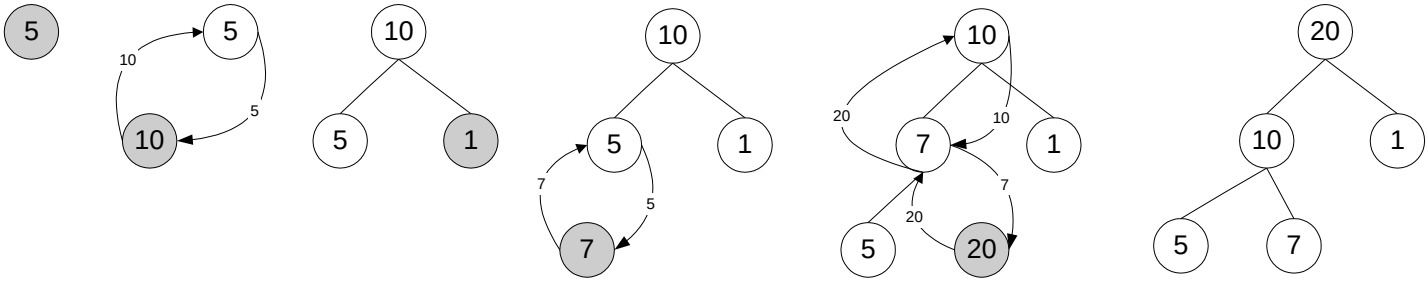
```
      for (i=0; i<trazenaDubina; i++) printf(" ");
      printf("%s %s\n", korijen->student->ime, korijen->student-
>prezime);
    }
    ispisi2 (korijen->lijevo, trenutnaDubina + 1, trazenaDubina);
    ispisi2 (korijen->desno, trenutnaDubina + 1, trazenaDubina);
  }
}

// funkcija trazi listove s najvecom i najmanjom razinom
// poziv balansirano (korijen, 1, &maxDub, &minDub);
void balansirano (Cvor * korijen, int trenDub,
                                      int * maxDub, int * minDub) {
  if (korijen) {
    if (!korijen->lijevo & !korijen->desno) {
      if (*maxDub == 0 || trenDub > *maxDub)
        *maxDub = trenDub;
      if (*minDub == 0 || trenDub < *minDub)
        *minDub = trenDub;
    }
    else
      balansirano (korijen->lijevo, trenDub + 1, maxDub, minDub);
      balansirano (korijen->desno, trenDub + 1, maxDub, minDub);
    }
  }
}

// funkcija za zadano binarno stablo ispisuje listove
void ispisiListove(Cvor * korijen) {
  if (korijen) {
    if (!korijen->lijevo & !korijen->desno)
      printf("%s %s; ", korijen->student->ime, korijen->student-
>prezime);
    ispisiListove(korijen->lijevo);
    ispisiListove(korijen->desno);
  }
}
```

```
// funkcija u zadanom binarnom stablu zbraja broj cvorova (argument
// funkcije) i ukupne godine starosti podataka u stablu (podatak se
// prenosi preko imena funkcije)
int zbroj(Cvor * korijen, int * broj) {
  if (korijen) {
    (*broj)++;
    return 2001 - korijen->student->godRod +
         zbroj(korijen->lijevo, broj) +
         zbroj(korijen->desno, broj);
  }
  return 0;
}

// funkcija prebroji sve cvorove stabla
int prebroji(Cvor * korijen) {
  if (korijen)
    return 1 + prebroji(korijen->lijevo) + prebroji(korijen->desno);
  else
    return 0;
}

main () {

  FILE * fUl;
  char buf[256];
  Podaci * student;
  Cvor * korijenPrezime = NULL;
  Cvor * korijenIme = NULL;
  Cvor * trazeni;
  int ukupno, broj = 0;
  int minDubina = 0, maxDubina = 0;
  int i;

  if ((fUl = fopen("stud.txt", "r")) == NULL) {
    fprintf(stderr, "Ne mogu otvoriti 'stud.txt'\n");
    exit(1);
  }
  while (fgets(buf, 256, fUl)) {
    student = (Podaci *) malloc(sizeof(Podaci));
    sscanf(buf, "%ld;%[^;];%[^;];%d", &(student->matBroj), student-
>prezime,
      student->ime, &(student->godRod));
    korijenPrezime = dodajPrezime(korijenPrezime, student);
    korijenIme = dodajIme(korijenIme, student);
  }
  fclose(fUl);
  inOrder(korijenPrezime);
  printf("\n");
  if(trazeni = potraziPrezime(korijenPrezime, "Maric"))
    printf("Pronasao: %s, %s %ld %d\n", trazeni->student->prezime,
```

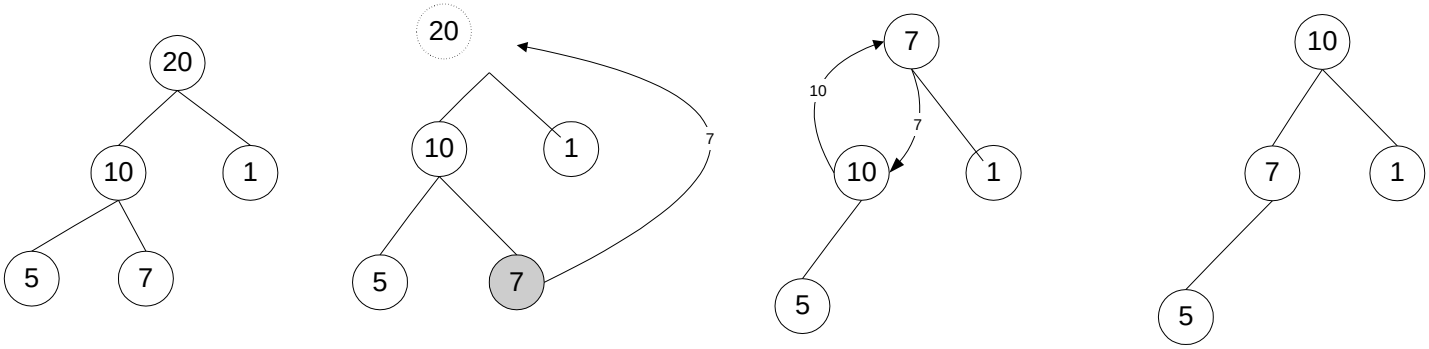
```
      trazeni->student->ime, trazeni->student->matBroj, trazeni-
>student->godRod);
  inOrder(korijenIme);
  printf("\nPrezime, listovi:\n");
  ispisiListove(korijenPrezime);
  printf("\nIme, listovi:\n");
  ispisiListove(korijenIme);
  ukupno = zbroj(korijenPrezime, &broj);
  printf ("\nZbroj godina=%d Podataka u listi=%d prosjek=%5.2f\n",
    ukupno, broj, (float)ukupno / broj);
  broj = 0;
  ukupno = zbroj(korijenIme, &broj);
  printf ("\nZbroj godina=%d Podataka u listi=%d prosjek=%5.2f\n",
    ukupno, broj, (float)ukupno / broj);
  printf("Prebroji po prezimenu=%d\n", prebroji(korijenPrezime));
  printf("Prebroji po imenu=%d\n", prebroji(korijenIme));
  balansirano (korijenPrezime, 1, &maxDubina, &minDubina);
  printf("MaxDubina=%d  MinDubina=%d Balansirano=%s\n", maxDubina,
minDubina,
    maxDubina - minDubina > 1 ? "false":"true");
  for (i=1; i<=maxDubina; i++) {
    printf("%d razina:\n", i);
    ispisi2(korijenPrezime, 1, i);
  }
  maxDubina = minDubina = 0;
  balansirano (korijenIme, 1, &maxDubina, &minDubina);
  printf("MaxDubina=%d  MinDubina=%d Balansirano=%s\n", maxDubina,
minDubina,
    maxDubina - minDubina > 1 ? "false":"true");
  for (i=1; i<=maxDubina; i++) {
    printf("%d razina:\n", i);
    ispisi2(korijenIme, 1, i);
  }
}
```

8. Auditorne vježbe

Stvaranje gomile za niz : 5, 10, 1, 7, 20



Skidanje elementa iz gomile:



1. Napisati program za ubacivanje i skidanje procesa iz prioritetnog reda. Za prioritet procesa i odabir operacije (ubacivanje ili skidanje) koristi se generator slučajnih brojeva. Ako generator načini neparni broj, proces najvećeg prioriteta vadi se iz prioritetnog reda. Ako generator načini parni broj, stvara se novi proces sa slučajno generiranim prioritetom i stavlja u prioritetni red.

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

```
#define MAXPRIOR 100
typedef int tip;
```

```
void podesi (tip A[], int i, int n) {
   // potpuna binarna stabla s korijenima A[2 * i] i A[2 * i + 1]
   // kombiniraju se s A[i] formirajuci jedinstvenu gomilu
   // 1 <= i <= n
   int j;
   tip stavka;

   j = 2 * i; stavka = A[i];
   while (j <= n) {
      // Usporedi lijevo i desno dijete (ako ga ima)
      if ((j < n) && (A[j] < A[j + 1])) j++;
      // j pokazuje na vece dijete
      if (stavka >= A[j]) break;  // stavka je na dobrom mjestu
      A[j / 2] = A[j]; // vece dijete podigni za razinu
      j *= 2;
   }
   A[j / 2] = stavka;
}
```

```
void ubaci (tip A[], int k) {
   // ubacuje vrijednost iz A[k] na gomilu pohranjenu u A[1 : k -
1]
   int i, j;
   tip novi;

   j = k;
   i = k / 2;
   novi = A[k];
   while ((i > 0) && (A[i] < novi)) {
      A[j] = A[i]; // smanji razinu za roditelja
      j = i;
      i /= 2;    // roditelj od A[i] je na A[i/2]
   }
   A[j] = novi;
}
```

```
tip skini (tip A[], int *k) {
   // izbacuje vrijednost iz A[k] sa prvog mjesta
   // ako je red prazan vraca -1
   tip retVal = -1;

   if (*k <= 1) return retVal;
   retVal = A[1];
   (*k) --;
   A[1] = A[*k];
   podesi (A, 1, *k);
   return retVal;
}



int main() {
   int prior, i, j, k = 1;
   tip A[MAXPRIOR];

   srand((unsigned) time(NULL));
   while(1) {
      if (rand() % 2) {
         if (k >= MAXPRIOR)
            printf("Red prioriteta pun!\n");
         else {
            printf("Dodavanje u red prioriteta: %d\n",
            prior=(int)(rand()/(RAND_MAX + 1.) * 99 + 1));
            A[k] = prior;
            ubaci(A, k);
            k++;
         }
      } else {
         if ((prior = skini(A, &k)) == -1)
            printf("Red prioriteta prazan!\n");
         else
            printf("Skidanje iz reda prioriteta: %d\n", prior);
      }
      for (i = 1, j = 1; i < k; j++) {
         for (; i <= pow (2, j) - 1 && i < k; i++) {
            printf(" %d ", A[i]);
         }
         printf ("\n");
      }
      printf("Jos (d/n)? ");
      if (getch() == 'n') break;
   }
}
```

2.Napisati program koji će od dvije sortirane slijedne datoteke napraviti treću sortiranu slijednu datoteku.

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

void MergeF(FILE *f1, FILE *f2, FILE *fsort){
   char buf1[4096], buf2[4096], *pb1, *pb2;

   pb1 = fgets(buf1, 4096, f1);
   pb2 = fgets(buf2, 4096, f2);
   while (pb1 || pb2) {
      /* ako u obje datoteke još ima zapisa i
         zapis iz prve datoteke manji je od zapisa iz druge ili
          u prvoj datoteci još ima, a u drugoj više nema zapisa */
      if ((pb1 && pb2 && strcmp(pb1, pb2) <= 0) || (pb1 && !pb2)) {
         fputs(pb1, fsort);
         pb1 = fgets(buf1, 4096, f1);
      }
      /* ako u obje datoteke još ima zapisa i
         zapis iz prve datoteke veći je od zapisa iz druge ili
          u prvoj datoteci nema, a u drugoj još ima zapisa */
      if ((pb1 && pb2 && strcmp(pb1, pb2) > 0) || (!pb1 && pb2)) {
         fputs(pb2, fsort);
         pb2 = fgets(buf2, 4096, f2);
      }
   }
}

int main() {
   FILE *f1,*f2,*fsort;

   if ((f1 = fopen ("dat1.txt", "r"))==NULL){
      printf("Pogreška kod otvaranja datoteke dat1.txt");
      exit (1);
   }
   if ((f2 = fopen ("dat2.txt", "r"))==NULL){
      printf("Pogreška kod otvaranja datoteke dat2.txt");
      exit (1);
   }
   if ((fsort = fopen ("sort.txt", "w"))==NULL){
      printf("Pogreška kod otvaranja datoteke sort.txt");
      exit (1);
   }

   MergeF(f1, f2, fsort);
   fclose(f1);   fclose(f2);fclose(fsort);
   return 0;
}
```

3. Napisati program za sortiranje velike slijedne datoteke. Ideja je da se od velike datoteke načini više manjih sortiranih datoteka koje se potom spoje u jednu sortiranu datoteku.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAXREDAKA 100
#define MAXZNAKOVA 512

void MergeNFile (FILE **fpolje, FILE *fout, int brdat);

int compare (const void *redak1, const void *redak2);

int main() {
   FILE *f, *fout, **fpolje = NULL;
   int brdat, brojaczapisa, i;
   char *flag, imedat[30], *redak[MAXREDAKA];
   char linija[MAXZNAKOVA + 1];

   if ((f = fopen ("dat1.txt", "r")) == NULL) {
      printf("Pogreška kod otvaranja datoteke dat1.txt");
      exit (1);
   }
   for (brdat = 0; ;) {
      // dok imamo manje od veličine bloka i imamo što čitati
  for (brojaczapisa = 0; brojaczapisa < MAXREDAKA &&


     (flag = fgets (linija, MAXZNAKOVA, f)); brojaczapisa ++) {


         redak [brojaczapisa]= (char *) malloc(strlen (linija) + 1);
         if (redak[brojaczapisa] == NULL){
            fprintf (stderr, "Nedovoljno memorije!");
            exit(1);
         }
         strcpy (redak[brojaczapisa], linija);
      }
      if (brojaczapisa > 0) {
         //treba sortirati podatke
  qsort((void *)redak, brojaczapisa, sizeof(char *), compare);
         //treba zapisati u novu datoteku
         fpolje = (FILE **)realloc(fpolje,(brdat + 1)* sizeof(FILE*));
         sprintf (imedat, "%03d.txt", brdat);
         if (( fpolje[brdat] = fopen (imedat, "w+")) == NULL){
            printf("Pogreška kod otvaranja datoteke %s", imedat);
            exit (1);
         }
         for(i = 0; i < brojaczapisa; i++)
            fputs(redak[i], fpolje[brdat]);
         brdat ++;
      }
```

```
      if (!flag)  break;
   }
   if ((fout = fopen ("out.txt", "w")) == NULL){
        fprintf (stderr, "Pogreška kod otvaranja datoteke out.txt");
        exit(1);
   }
   MergeNFile(fpolje, fout, brdat);
   for(i = 0; i < brdat; i++)
      fclose(fpolje[i]);

   fclose(f);
   fclose(fout);
   free(fpolje);
   return 0;
}

void MergeNFile (FILE **fpolje, FILE *fout, int brdat){
   int i, ind;
   char **plinija;

   plinija =(char**)malloc(brdat * sizeof(char*));
   for(i = 0; i < brdat; i++){
      rewind(fpolje[i]);  //premotati na pocetak
      plinija[i]=(char *)malloc(MAXZNAKOVA + 1);
      if (!fgets(plinija[i], MAXZNAKOVA, fpolje[i]))
         *plinija[i] = '\0';
   }

   while (1){
      ind = -1;
      for(i = 0; i < brdat; i++){
         if (*plinija[i]) {
           if (ind == -1 || strcmp(plinija[i], plinija[ind]) < 0)
            ind = i;
         }
      }
      if (ind == -1) break;
      fputs (plinija[ind], fout);
      if (!fgets(plinija[ind], MAXZNAKOVA, fpolje[ind])) *plinija[ind] =
         '\0';
   }
   for(i = 0; i < brdat; i++) free(plinija[i]);
   free(plinija);
}


int compare( const void *redak1, const void *redak2 ) {
  return strcmp( * (char **) redak1, * (char **) redak2 );
}
```