Programiranje II

Beleške sa vežbi

Smer *Informatika*

Matematički fakultet, Beograd

Sana Stojanović

05.06.07.

Sadržaj

1 Sortiranje niza - selection sort

1. Napisati funkciju koja sortira niz brojeva *selection sort* algoritmom.

```
/* Sortiranje niza celih brojeva - Selection sort algoritam */
#include <stdio.h>

/* Funkcija sortira dati niz brojeva koji ima n elemenata */
void selection_sort(int a[], int n)
{
    int i, j;

    /* Na svaku poziciju od 0 do pretposlednje */
    for (i = 0; i < n-1; i++)
        /* Dovodimo minimalni element ostatka niza */
        for(j = i+1; j < n; j++)
            if (a[i]>a[j])
            {
                int tmp = a[i];
                a[i] = a[j];
                a[j] = tmp;
            }
}

main()
{
    int a[] = {5, 8, 2, 4, 1, 9, 3, 7, 6};
    int n = sizeof(a)/sizeof(int);
    int i;

    selection_sort(a, n);

    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");
}
```

2 Sortiranje liste

1. Prepraviti prethodni program tako da radi sortiranje liste.

```
#include <stdio.h>

typedef struct cvor
```

```
{
    int br;
    struct cvor * sl;
} cvor;

cvor * napravi_cvor(int br)
{
    cvor * novi = (cvor *)malloc(sizeof(cvor));
    if (novi == NULL)
    {
        fprintf(stderr, "greska pri alok mem");
        exit(1);
    }

    novi->br = br;
    return novi;
}

cvor * ubaci_na_kraj_rekurzivno(cvor * l, int br)
{
   if (l==NULL)
   {
       cvor * novi = napravi_cvor(br);
       novi->sl = NULL;
       return novi;
   }

   l->sl = ubaci_na_kraj_rekurzivno(l->sl, br);
   return l;
}

void ispisi_listu_i(cvor * l)
{
   cvor * t;
   for(t=l; t!=NULL; t=t->sl)
      printf("%d ", t->br);
}

void oslobodi_listu(cvor *l)
{
    cvor *tmp;
    while (l)
    {
        tmp = l->sl;
        free(l);
        l = tmp;
```

```
    }
}

/* Modifikovali smo selection-sort da odgovara aritmetici liste. */
void selection_sort_liste(cvor *l)
{
    cvor * i ,*j;
    /* Na svaku poziciju od prve do pretposlednje */
    for (i = l; i !=NULL; i=i->sl)
       /* Dovodimo minimalni element ostatka liste */
       for(j = i->sl; j !=NULL; j=j->sl)
            if (i->br > j->br)
            {
                int tmp = i->br;
                i->br = j->br;
                j->br = tmp;
            }
}

int main()
{
    cvor * l = NULL;
    int i;

    printf("Unesi sledeci element liste (0 za kraj):\n");
    while(1)
    {
        scanf("%d", &i);
        if (!i) break;
        l= ubaci_na_kraj_rekurzivno(l, i);
    }

    printf("Uneli ste: ");
    ispisi_listu_i(l);
    printf("\n");

    selection_sort_liste(l);
    printf("Lista nakon sortiranja: ");
    ispisi_listu_i(l);

    oslobodi_listu(l);

    return 0;
}
```

3 Formiranje generičke funkcije sortiranja, sistemska funkcija *qsort*

Jedan od načina da sortiramo niz celih brojeva je korišćenjem *selection-sort* algoritma. Ovaj algoritam možemo izdvojiti u funkciju:

```
void sort_int(int a[], int n)
{
    int i, j;

    for(i=0; i<n-1; i++)
        for(j=i+1; j<n; j++)
            if(a[i]<a[j])
            {
                int pom=a[i];
                a[i]=a[j];
                a[j]=pom;
            }
}
```

Za sortiranje realnih brojeva možemo koristiti sličnu funkciju:

```
void sort_float(float a[], int n)
{
    int i, j;

    for(i=0; i<n-1; i++)
        for(j=i+1; j<n; j++)
            if(a[i]<a[j])
            {
                float pom=a[i];
                a[i]=a[j];
                a[j]=pom;
            }
}
```

pri čemu vidimo da se ove dve funkcije razlikuju na sledeći način:

- Tip prvog argumenta funkcije (u prvom slučaju *int* a u drugom *float*)
- Tip pomoćne promenljive
- Operacija poređenja (u prvom slučaju poređenje celih brojeva a u drugom poređenje realnih brojeva)

Da bismo naglasili ove razlike posmatrajmo sortiranje niza struktura. Neka nam je data struktura *student*:

```
typedef struct _student
{
    char ime[MAX_IME];
    char prezime[MAX_IME];
    int ocena;
} student;
```

Sada možemo zadati niz studenata sortirati na tri načina: po oceni, po imenu i po prezimenu i dobijamo sledeće funkcije:

1. Sortiranje studenata po oceni:

```
void sort_po_oceni(student a[], int n)
{
    int i, j;

    for(i=0; i<n-1; i++)
       for(j=i+1; j<n; j++)
           if(a[i].ocena < a[j].ocena)
           {
                student pom=a[i];
                a[i]=a[j];
                a[j]=pom;
           }
}
```

2. Sortiranje studentata po imenu:

```
void sort_po_imenu(student a[], int n)
{
    int i, j;

    for(i=0; i<n-1; i++)
       for(j=i+1; j<n; j++)
          if(strcmp(a[i].ime, a[j].ime)<0)
          {
               student pom=a[i];
               a[i]=a[j];
               a[j]=pom;
          }
}
```

3. Sortiranje studenata po prezimenu:

```
void sort_po_prezimenu(student a[], int n)
{
```

```
    int i, j;

    for(i=0; i<n-1; i++)
        for(j=i+1; j<n; j++)
            if(strcmp(a[i].prezime, a[j].prezime)<0)
            {
                student pom=a[i];
                a[i]=a[j];
                a[j]=pom;
            }
}
```

Prvo što možemo da primetimo jeste da se ove tri funkcije razlikuju samo po funkciji koja poredi dva studenta.

Izdvojimo funkcije poređenja:

```
int poredi_po_oceni(student st1, student st2)
{
    return st1.ocena - st2.ocena;
}

int poredi_po_imenu(student st1, student st2)
{
    return strcmp(st1.ime, st2.ime);
}

int poredi_po_prezimenu(student st1, student st2)
{
    return strcmp(st1.prezime, s2.prezime);
}
```

Vidimo da sve tri funkcije imaju isti potpis i jos jedno bitno svojstvo, a to je da sve tri vraćaju vrednost manju od nule ukoliko je prvi element manji od drugog, nula ukoliko su elementi jednaki i vrednost veću od nule ukoliko je drugi element veći od prvog.

Sada funkcija koja sortira niz izgleda ovako:

```
void sort_po_imenu(student a[], int n)
{
    int i, j;

    for(i=0; i<n-1; i++)
       for(j=i+1; j<n; j++)
           /*if(poredi_po_prezimenu(a[i], a[j])<0)*/
           /*if(poredi_po_oceni(a[i], a[j])<0)*/
           if(poredi_po_imenu(a[i], a[j])<0)
           {
```

```
                student pom=a[i];
                a[i]=a[j];
                a[j]=pom;
            }
}
```

I vidimo da možemo da dodamo funkciju poređenja kao argument funkciji sortiranja i da na taj način dobijemo jednu funkciju umesto tri:

```
void sort_studente(student a[], int n, int (*f)(student, student))
{
    int i, j;

    for(i=0; i<n-1; i++)
        for(j=i+1; j<n; j++)
            if((*f)(a[i], a[j])<0)
            {
                student pom=a[i];
                a[i]=a[j];
                a[j]=pom;
            }
}
```

Sada smo rešili problem sortiranja niza struktura bez obzira na uslov poređenja, ali još uvek treba da rešimo sledeće probleme:

- Razmena mesta elemenata zavisi od tipa elemenata niza
- Potpis funkcije poređenja zavisi od tipa elemenata niza
- Prvi argument funkcije zavisi od tipa elemenata niza

Ove probleme ćemo rešiti korišćenjem pokazivača na *void*. Ostaje pitanje kako ćemo razmeniti vrednosti dva elementa niza ako ne znamo kog tipa će biti pomoćna promenljiva?

Korišćenjem funkcije *malloc* možemo odvojiti prostor u memoriji za smeštanje pomoćne promenljive. Jedino što treba da znamo jeste koliko nam prostora treba. Kako funkcija sortiranja ne zna tip elementa niza pa samim tim ni njegovu veličinu vidimo da kao još jedan argument funkciji sortiranja moramo dodati veličinu jednog elementa niza.

Sada, pod pretpostavkom da nam je data *size* - veličina jednog elementa, razmenu vršimo na sledeći način:

```
void* tmp = malloc(size);
memcpy(tmp, adresa_itog, size);
memcpy(adresa_itog, adresa_jtog, size);
memcpy(adresa_jtog, tmp, size);
free(tmp);
```

Potpis funkcije poređenja ne sme da zavisi od tipa elemenata koji se porede. To ćemo rešiti korišćenjem pokazivača na tip void.

Na primer, funkcija koja poredi dva cela broja će tada izgledati ovako:

```
int poredi_br(void* a, void* b)
{
    int br_a = *(int*)a;
    int br_b = *(int*)b;

    return br_a-br_b;
}
```

Funkcija koja poredi dva realna broja:

```
int poredi_float(void* a, void* b)
{
    float br_a = *(float*)a;
    float br_b = *(float*)b;

    if (br_a > br_b) return 1;
    else if (br_a < br_b) return -1;
    else return 0;
}
```

Funkcija koja poredi dva studenta po oceni:

```
int poredi_br(void* a, void* b)
{
    student student1 = *(studnet*)a;
    student student2 = *(studnet*)b;

    return student1.ocena-student2.ocena;
}
```

Sada funkcija poređenja ima uvek isti potpis:

```
int poredi(void* a, void* b)
```

i može se kao parametar proslediti funkciji sortiranja.

1. Sortiranje niza celih brojeva: implementacija generičke funkcije sortiranja - funkcija je nezavisna od tipa elemenata niza koji se sortira.

```
#include <stdlib.h>

/* Funkcija sortira niz elemenata proizvoljnog tipa koriscenjem
   selection-sort-a. Argumenti funkcije su :
```

```
   -adresa pocetka niza data kao void*,
   -broj elemenata niza,
   -velicina jednog elementa niza,
   -funkcija poredjenja dva elementa niza.
*/
void sort(void* a, int n, int size, int (*poredi)(void*,void*))
{
    int i, j;
    for (i = 0; i<n-1; i++)
        for (j = i+1; j<n; j++)
        {
            void* adresa_itog = (char*)a+i*size;
            void* adresa_jtog = (char*)a+j*size;

            if (poredi(adresa_itog, adresa_jtog))
            {
                void* tmp = malloc(size);
                memcpy(tmp, adresa_itog, size);
                memcpy(adresa_itog, adresa_jtog, size);
                memcpy(adresa_jtog, tmp, size);
                free(tmp);
            }

        }
}


int poredi_br(void* a, void* b) {
    int br_a = *(int*)a;
    int br_b = *(int*)b;

    return br_a>br_b;
}

int poredi_float(void* a, void* b) {
    float br_a = *(float*)a;
    float br_b = *(float*)b;

    if (br_a > br_b) return 1;
    else if (br_a < br_b) return -1;
    else return 0;
}

main() {
    int a[] = {8, 2, 1, 9, 3, 7, 6, 4, 5};
    float b[] = {0.3, 2, 5, 5.8, 8};
```

```
    int n = sizeof(a)/sizeof(int);
    int nf = sizeof(b)/sizeof(float);
    int i;

    /* Sortiranje niza celih brojeva */
    sort(a, n, sizeof(int), poredi_br);

    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
    putchar('\n');

    /* Sortiranje niza realnih brojeva */
    sort(b, nf, sizeof(float), poredi_float);

    for (i = 0; i < n; i++)
        printf("%f ", b[i]);
    putchar('\n');
}
```

2. Sortiranje niza celih brojeva - koriscenje ugradjene funkcije *qsort*.

   Sa standardnog ulaza se prvo učitava dužina niza n pa zatim n celih brojeva. Pozivom funkcije *qsort* sortirati dobijeni niz i ispisati ga na standardni izlaz.

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

/* Potrebno je napraviti funkciju poredjenja */
int poredi(const void* a, const void* b)
{
    return  *((int*)a)-*((int*)b);
}

main()
{
    int *a;
    int n;
    int i;

    printf("Unesite dimenziju niza: ");
    scanf("%d", &n);

    a = (int *)malloc(n*sizeof(int));
    if (a==NULL)
```

```
        exit (1);

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

    /* Pocetak niza, broj elemenata, velicina jednog elementa, adresa
       funkcije poredjenja */
    qsort((void*)a, n, sizeof(int), poredi);

    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");

    free(a);
}
```

4 Primeri korišćenja funkcije *qsort*

1. Napisati program koji sortira niz reči (dat u okviru funkcije *main*) sortira prvo leksikografski pa zatim po dužini i ispisuje oba načina sortiranja na standardni izlaz.

```
/* Sortiranje niza reci - leksikografski odnosno po duzini -
koriscenje ugradjene qsort funkcije*/

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

/* Funkcija koja vrsi leksikografsko poredjenje dve reci */
int lexicographic_compare(const void* a, const void* b)
{
    return strcmp(*(char**)a,*(char**)b);
}

/* Funkcija koja vrsi poredjenje po duzini dve reci, opadajuce*/
int length_compare(const void* a, const void* b)
{
    return strlen(*(char**)b)-strlen(*(char**)a);
}

main()
{
    int i;
```

```
    char* a[] = {"Jabuka", "Kruska", "Sljiva", "Dinja", "Lubenica"};
    int n = sizeof(a)/sizeof(int);

    /* Sortiramo leksikografski i ispisujemo rezultat */
    qsort((void*)a, n, sizeof(char*), lexicographic_compare);
    for (i=0; i<n; i++)
        printf("%s ", a[i]);
    putchar('\n');


    /* Sortiramo po duzini i ispisujemo rezultat */
    qsort((void*)a, n, sizeof(char*), length_compare);
    for (i=0; i<n; i++)
        printf("%s ", a[i]);
    putchar('\n');


}
```

2. Napisati program kojim se niz struktura sortira po jednom polju. Na primer, neka je data struktura artikli sa dva polja: ime artikla (karakteska niska od 20 karaktera) i cena artikla (ceo broj). Sa standardnog ulaza se prvo učitava broj artikala pa zatim redom ime artikla pa cena artikla za sve artikle. Pozivom funkcije *qsort* sortirati dati niz struktura leksikografski po nazivima artikla a zatim i prema cenama. Ispisati sortirani niz.

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

typedef struct artikal{
    char ime[20];
    int cena;
}artikal;

int leksikografsko_poredjenje(const void* a, const void* b)
{
    return strcmp(((artikal*)a)->ime,((artikal*)b)->ime);
}

int poredjenje_po_cenama(const void* a, const void* b)
{
    return ((artikal*)a)->cena - ((artikal*)b)->cena;
}
```

```
main()
{
    artikal* niz;   //Niz u kome cemo cuvati artikle
    int n;          //Broj elemenata niza
    int i;

    printf("Unesite broj artikala: \n");
    scanf("%d", &n);

    niz = (artikal *)malloc(n*sizeof(artikal));
    if (niz==NULL) exit(1);

    //Ucitavanje artikala
    for(i=0; i<n; i++)
    {
        scanf("%s", niz[i].ime);
        scanf("%d", &niz[i].cena);
    }

    //Sortiranje niza artikala po imenima
    qsort((void *)niz, n, sizeof(artikal), leksikografsko_poredjenje);

    //Ispis niza artikala sortiranog po imenima
    for(i=0; i<n; i++)
        printf("\n%s %d", niz[i].ime, niz[i].cena);

    putchar('\n');

    //Sortiranje niza artikala po ceni
    qsort((void *)niz, n, sizeof(artikal), poredjenje_po_cenama);

    //Ispis niza artikala sortiranog po cenama
    for(i=0; i<n; i++)
        printf("\n%s %d", niz[i].ime, niz[i].cena);

    putchar('\n');
    free(niz);
}
```

3. Napisati program koji sa standradnog ulaza učitava niz reči. Među unetim rečima pronaći sve anagrame i ispisati ih tako da se u jednom redu nalaze sve reči koje su anagrami.

   Program pozivati sa preusmeravanjem ulaza. Ako se u datoteci *reci.txt* nalaze sve reči koje želimo da obradimo i ako je naš program (recimo) *./a.out* onda ga treba pozivati sa:

./a.out ¡ reci.txt

```
/* Sortiranje : program pronalazi sve anagrame medju unetim recima */
/* Dve reci su anagrami ako i samo ako se sastoje od istog broja
   istih slova */

/* Algoritam se zasniva na odredjivanju kanonske forme svake reci.
   Kanonska forma se odredjuje sortiranjem slova svake reci.
   Reci su anagrami ako i samo ako su im kanonske forme jednake.
   Sortiranjem niza reci na osnovu kanonskih formi anagrami se pojave
   jedan uz drugi u nizu */

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


/* Pomocna funkcija koja ucitava rec. Pod recju se smatra niz slova (pa
   koristimo funkciju isalpha za proveru da li je karatker slovo).
   Funkcija sem niske u koju treba da upise rec dobija i duzinu te niske
   (da ne bismo prekoracili rezervisanu memoriju) */
int getword(char s[], int lim)
{
    int c;
    int i = 0;
    while (!isalpha(c = getchar()))
        if (c == EOF)
            return 0;
    do
    {
        s[i++] = c;
    }
    while (i<lim-1 && isalpha(c = getchar()));

    s[i] = '\0';
    return i;
}



/* Maksimalna duzina reci */
#define MAX_WORD 50

/* Struktura reci je odredjena originalnom recju i njenim
   kanonskim predstavnikom */
struct word {
```

```
    char original[MAX_WORD];
    char cannonical[MAX_WORD];
};


/* Funkcija koja vrsi poredjenje dva slova u reci */
int letter_compare(const void* a, const void* b)
{
    return *(char*)a - *(char*)b;
}

/* Funkcija koja vrsi poredjenje po dve reci, na osnovu njihovih
   kanonskih formi*/
int cannonical_compare(const void* a, const void* b)
{
    return strcmp(((struct word*)a)->cannonical,((struct word*)b)->cannonical);
}

/* Dve reci su anagrami ako imaju istu kanonsku formu a razlicite su */
int is_anagram(struct word a, struct word b)
{
    return strcmp(a.cannonical, b.cannonical) == 0
    && strcmp(a.original, b.original) != 0;
}

main()
{
    int i, n;
    struct word words[1000];

    /* Ucitavamo reci sa standardnog ulaza dok ih ima */
    for (i = 0; getword(words[i].original, MAX_WORD) != 0; i++)
    {
        /* Odredjujemo kanonsku formu svake reci */
        strcpy(words[i].cannonical, words[i].original);
        qsort(words[i].cannonical, strlen(words[i].cannonical),
            sizeof(char), letter_compare);
    }

    /* Broj ucitanih reci */
    n = i;
    printf("Procitano %d reci\n", n);

    /* Sortiramo niz reci na osnovu kanonske forme */
    qsort(words, n, sizeof(struct word), cannonical_compare);
```

```
    /* Pronalazimo anagrame kao susedne elemente niza i ispisujemo ih */
    for (i = 1; i<n; i++)
        if (is_anagram(words[i-1], words[i]))
        {   printf("Anagrami : %s - %s",
                words[i-1].original, words[i].original);
            while(i+1 < n && is_anagram(words[i], words[i+1]))
            {   printf(" - %s", words[i+1].original);
                i++;
            }
            printf("\n");
        }

}
```

5 Sistemska funkcija *bsearch*

1. Binarna pretraga niza celih brojeva : korišćenje ugrađene funkcije *bsearch*.

```
#include <stdlib.h>

/* Potrebno je izgraditi funkciju poredjenja */
int poredi(const void* a, const void *b)
{
    return *(int*)a - *(int*)b;
}

main()
{
    int *a;
    int n;
    int x, *e;

    printf("Unesite dimenziju niza: ");
    scanf("%d", &n);

    a = (int *)malloc(n*sizeof(int));
    if (a==NULL)
        exit (1);

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

    //PRE POZIVA FUNKCIJE bsearch NIZ KOJI PRETRAZUJEMO MORA BITI
```

```
    //SORTIRAN PA ZATO PRVO POZIVAMO FUNKCIJU qsort

    qsort((void*)a, n, sizeof(int), poredi);

    printf("Unesi element koji trazimo: ");
    scanf("%d",&x);

    /* Adresa elementa koji trazimo, pocetak niza, broj elemenata,
       velicina jednog elementa, funkcija poredjenja */
    e = (int*)bsearch((void*)&x,
              (void*)a,
              sizeof(a)/sizeof(int),
              sizeof(int),
              poredi);

    if (e == NULL)
        printf("Elementa %d nema\n", x);
    else
        printf("Pronadjen na poziciji %d\n", e-a);

    free(a);
}
```

2. Napisati program koji sa standardnog ulaza unosi podatke o artiklima (ime artikla i cena), sortira ih po imenu (pozivom funkcije *qsort* i nakon toga pozivom funkcije *bsearch* određuje cenu konkretnog artikla (čije ime takođe unosimo sa standardnog ulaza).

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

typedef struct artikal{
    char ime[20];
    int cena;
}artikal;

/* Funkcija koja poredi dva artikla po imenima, prosledjuje se
   kao parametar funkciji qsort */
int leksikografsko_poredjenje(const void* a, const void* b)
{
    return strcmp(((artikal*)a)->ime,((artikal*)b)->ime);
}

//OBRATITI PAZNJU: FUNKCIJA KOJA SE PROSLEDNJUJE FUNKCIJI bsearch NIJE
//ISTA KAO FUNKCIJA KOJA SE PROSLEDJUJE FUNKCIJI qsort KADA SU U
```

```
//PITANJU NIZOVI STRUKTURA
/* Funkcija koja poredi karaktersku nisku sa poljem strukture,
   prosledjuje se kao parametar funkciji bsearch */
int poredi_b(const void* a, const void* b)
{
    return strcmp((char*)a,((artikal*)b)->ime);
}


main()
{
    artikal* niz;   //Niz u kome cemo cuvati artikle
    int n;          //Broj elemenata niza
    int i;
    char s[20];
    artikal *e;

    printf("Unesite broj artikala: \n");
    scanf("%d", &n);

    niz = (artikal *)malloc(n*sizeof(artikal));
    if (niz==NULL) exit(1);

    //Ucitavanje artikala
    for(i=0; i<n; i++)
    {
        scanf("%s", niz[i].ime);
        scanf("%d", &niz[i].cena);
    }

    //Sortiranje niza artikala po imenima
    qsort((void *)niz, n, sizeof(artikal), leksikografsko_poredjenje);

    //Ispis niza artikala sortiranog po imenima
    for(i=0; i<n; i++)
        printf("\n%s %d", niz[i].ime, niz[i].cena);

    putchar('\n');

    //AKO PRETRAZUJEMO NIZ ARTIKALA PO IMENU ONDA TAJ NIZ MORA
    //PRETHODNO BITI SORTIRAN UPRAVO PO IMENIMA ARTIKALA
    printf("Unesite naziv artikla cija vas cena interesuje: \n");
    scanf("%s", s);

    e = (artikal*)bsearch((void*)s,(void*)niz,n,sizeof(artikal),poredi_b);
```

```
    if (e == NULL)
        printf("%s se ne nalazi u nizu\n", s);
    else
        printf("Trazena cena je: %d\n", e->cena);

    putchar('\n');
    free(niz);
}
```

3. [1] Binarna pretraga niza struktura - pretraga studenata po broju indeksa.

```
/* Datoteka sadrzi podatke o uspehu studenata na kolokvijumima iz
osnova
   programiranja. Prva linija datoteke sadrzi broj studenata, a zatim svaka
   sledeca linija sadri ime i prezime odredjenog studenta, njegov broj
   indeksa (u obliku korisnickog imena na alas-u npr. mr01123) i broj poena
   na prvom i na drugom kolokvijumu. Redosled studenata u datoteci je
   odredjen na osnovu njihovog broja indeksa, i to tako da su na pocetku
   datoteke navedeni stariji studenti sa manjim brojevima indeksa (npr.
   mv02234 je ispred mn03123 jer je stariji, a mr03123 je ispred ml03234
   jer ima manji broj indeksa).

   a) Definisati strukturu podataka za cuvanje podataka o studentima
   b) Napraviti funkciju koja poredi dva indeksa u skladu sa poretkom
      opisanim u uvodu zadatka
   c) Napsati rekurzivnu funckiju

            int binary_search_recursive(char* indeks, .....)

      koja izracunava broj poena studenta sa datim brojem indeksa, ukoliko je
      student polagao kolokvijum, odnosno -1 ukoliko nije.
   d) Napisati iterativnu varijantu prethodne funkcije

            int binary_search_iterative(char* indeks, ......)

   e) Napisati funkciju

            int binary_search_stdlib(char* indeks, .....)

      koja prethodnu funkcionalnost ostvaruje kroz poziv bibliotecke funkcije
      bsearch.
   f) Napisati program koji ucitava podatke o studentima iz datoteke cije
      je ime navedeno kao argument komandne linije, zatim koristeci funkcije
      iz prethodnog dela zadatka ispisuje ukupan broj poena za svakog studenata
```

[1] Obratiti pažnju na strukturu funkcije *poredi* u pozivu funkcije *bsearch*. Razlikuje se od funkcije *poredi* koju koristimo u *qsort*!

```
      ciji indeks korisnik unese sa standardnog ulaza, sve dok ne unese rec
      kraj za kraj.
*/

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

/* Definicija strukture za cuvanje podataka o jednom studentu */
typedef struct _student {
    char ime[15];
    char prezime[15];
    char indeks[8];
    int  kol_1;
    int  kol_2;
} student;

/* Studenti se smestaju u niz koji ce se dinamicki alocirati.
   Pretpostavlja se da je niz sve vreme sortiran kako bi se
   moglo vrsiti binarno pretrazivanje */
student* studenti;
int br_stud;

/* Funkcija vrsi poredjenje indeksa. Manjim se smatraju stariji studenti
   sa manjim brojem indeksa. Funkca vraca negativnu vrednost ukoliko je
   prvi indeks manji, pozitivnu ukoliko je veci, a 0 ukoliko su indeksi
   jednaki. Informacije o smerovima se u potpunosti zanemaruju. */
int poredi_indekse(char* indeks_1, char* indeks_2)
{
    /* Izdvaja se informacija o godini upisa i
       broju indeksa za oba studenta*/
    char s_godina_1[3], s_godina_2[3];
    int  godina_1, godina_2;
    char s_broj_1[4], s_broj_2[4];
    int  broj_1, broj_2;

    /* Godina upisa je zapisana u 3 i 4 karakteru
       indeksa. Za nultu godinu uzimamo 1900. i smatramo da su
       studenti ciji je broj indeksa ispod 50 upisani sto godina posle. */

    strncpy(s_godina_1, indeks_1+2, 2);
    godina_1 = atoi(s_godina_1);
    if (godina_1 < 50) godina_1 += 100;

    strncpy(s_godina_2, indeks_2+2, 2);
    godina_2 = atoi(s_godina_2);
```

```
    if (godina_2 < 50) godina_2 += 100;

    /* Prvo poredimo godine */
    if (godina_1 < godina_2)
        return -1;

    if (godina_1 > godina_2)
        return 1;

    /* Ukoliko su godine jednake, izdvajamo brojeve indeksa
       koji su zapisani u 5. 6. i 7. karakteru indeksa */
    strncpy(s_broj_1, indeks_1+4, 3);
    broj_1 = atoi(s_broj_1);
    strncpy(s_broj_2, indeks_2+4, 3);
    broj_2 = atoi(s_broj_2);

    /* Umesto ovoga, prolazi i return strcmp(indeks_1+4, indeks_2+4); */

    /* Poredimo godine upisa */
    return broj_1-broj_2;
}

/* Najjednostavnija rekurzivna implementacija algoritma
   binarne pretrage */
int binary_search_recursive(char* indeks, int l, int d) {
    int s, cmp;

    if (l>d)
        return -1;

    s = (l+d)/2;
    cmp = poredi_indekse(studenti[s].indeks, indeks);
    if (cmp == 0)
        return studenti[s].kol_1 + studenti[s].kol_2;
    else if (cmp < 0)
        return binary_search_recursive(indeks, s+1, d);
    else
        return binary_search_recursive(indeks, l, s-1);
}

/* Iterativna varijanta prethodnog algoritma */
int binary_search_iterative(char* indeks) {
    int l = 0, d = br_stud - 1;
    while (l <= d)
    {
        int s = (l+d)/2;
```

```
        int cmp = poredi_indekse(studenti[s].indeks, indeks);
        if (cmp == 0)
            return studenti[s].kol_1 + studenti[s].kol_2;
        else if (cmp < 0)
            l = s+1;
        else
            d = s-1;
    }
    return -1;
}

/* Funkcija poredjenja koja je potrebna prilikom poziva
   ugradjene funkcije bsearch */
int poredi(const void* indeks, const void* indeks_2) {
    return poredi_indekse((char*)indeks, ((student*)indeks_2)->indeks);
}

int binary_search_stdlib(char* indeks) {
    student* s = bsearch(indeks, studenti, br_stud, sizeof(student), poredi);
    return s==NULL ? -1 : s->kol_1+s->kol_2;
}

/* Glavni program */
main(int argc, char* argv[])
{
    FILE* datoteka;
    int i;
    char indeks[8];

    /* Proveravamo prisutnost argumenata komandne linije */
    if (argc<2)
    {   printf("Upotreba %s : ime_datoteke\n",argv[0]);
        return -1;
    }


    /* Otvaramo datoteku */
    if ((datoteka = fopen(argv[1], "r")) == NULL)
    {   printf("Greska prilikom otvaranja datoteke %s\n",argv[1]);
        return -1;
    }

    /* Ucivamo broj studenata i alociramo niz */
    fscanf(datoteka, "%d", &br_stud);
    studenti = (student*)malloc(br_stud*sizeof(student));
```

```
    /* Ucitavamo podatke o studentima */
    for (i = 0; i < br_stud; i++)
    {   fscanf(datoteka, "%s",studenti[i].ime);
        fscanf(datoteka, "%s",studenti[i].prezime);
        fscanf(datoteka, "%s",studenti[i].indeks);
        fscanf(datoteka, "%d", &studenti[i].kol_1);
        fscanf(datoteka, "%d", &studenti[i].kol_2);
    }

    /* Ispisujemo poene za svakog studenta kojeg korisnik trazi */
    do
    {   printf("Unesi broj indeksa : ");
        scanf("%s",indeks);
        printf("Broj poena : %d\n",binary_search_stdlib(indeks));
    } while (strcmp(indeks,"kraj") != 0);


    /* Oslobadjamo alociranu memoriju */
    free(studenti);

    return 0;

}
```