**Algoritmi i strukture podataka**

*5. srpnja 2012.*

*Nije dopušteno korištenje globalnih i statičkih varijabli te naredbe* ***goto****.*

## 1. zadatak (12 bodova)

Jedan zapis datoteke organizirane po načelu raspršenog adresiranja definiran je strukturom:

```
typedef struct{
        int sifra;
        char naziv[70+1];
        double cijena;
} zapis;
```

Zapis je prazan ako je na mjestu šifre vrijednost nula. Parametri za raspršeno adresiranje nalaze se u datoteci parametri.h i oni su:

- BLOK..... veličina bloka na disku
- MAXZAP ..... broj zapisa
- C..... broj zapisa u jednom pretincu
- M ..... broj pretinaca

Preljevi su realizirani ciklički, upisom u prvi sljedeći slobodni pretinac. Ključ zapisa je šifra artikla, a transformacija ključa u adresu obavlja se zadanom funkcijom **`int adresa(int sifra).`** Napišite funkciju koja će za dani redni broj pretinca **N** provjeriti postoje li zapisi koji su trebali biti zapisani u njega, ali su zbog preljeva zapisani u neki drugi pretinac. Vratiti redni broj pretinca koji ima najviše takvih zapisa te vratiti ukupan broj takvih zapisa. Funkcija vraća 0 ako takvi zapisi ne postoje.

## 2. zadatak (13 bodova)

Zadana je jednostruko povezana lista sljedećom strukturom:

```
typedef struct at{
        int element;
        struct at *sljed;
} atom;
```

Napisati rekurzivnu funkciju koja će promjenom pokazivača na sljedeće elemente obrnuti listu. Nerekurzivno rješenje neće se priznavati.

## 3. zadatak (15 bodova)

Binarno stablo sadrži podatke o artiklima koji su predstavljeni šifrom (pozitivan cijeli broj) i cijenom (struktura **el**), dok su čvorovi binarnog stabla definirani strukturom **`cvor`** :

```
typedef struct{
        int sifra;
        float cijena;
} el;
```

```
typedef struct s {
        el element;
        struct s *lijevo, *desno;
} cvor;
```

Uz pretpostavku da je stablo sortirano po šifri artikla: lijevo artikli s manjom šifrom, desno oni s većom te da u stablu ne postoje dva artikla s istom šifrom:

a) Napišite rekurzivnu funkciju **`razina`** koja će za zadanu šifru ispisati na kojoj razini unutar stabla se nalazi artikl s tom šifrom. Funkcija vraća 0 ako ne postoji artikl sa zadanom šifrom. Prototip funkcije je:

```
int razina(int sifra, cvor *korijen)
```

b) Napišite **rekurzivnu** funkciju **`zbrojCijenaNaRazini`** koja vraća zbroj cijena svih artikala koji se nalaze na zadanoj razini. Prototip funkcije je:

```
float zbrojCijenaNaRazini(int razina, cvor *korijen)
```

**4. zadatak (11 bodova)**

Zadan je tip podatka Stog za koji su definirane funkcije za inicijalizaciju stoga, dodavanje elementa na stog te za brisanje elementa sa stoga. Prototipovi navedenih funkcija su

```
void init_stog(Stog *stog);
int dodaj(int element, Stog *stog);
int skini(int *element, Stog *stog);
```

Funkcije dodaj i skini vraćaju 1 ako je operacija dodavanja, odnosno skidanja uspjela, a 0 inače.
Napišite funkciju **bezEkstrema** koja će sa zadanog stoga izbaciti sva pojavljivanja najmanjeg i najvećeg elementa. Međusobni poredak ostalih elemenata ne smije se mijenjati. Rješenje treba biti neovisno o implementaciji apstraktnog tipa podataka Stog. Smiju se koristiti samo zadane funkcije i eventualno pomoćni stogovi.

**5. zadatak (11 bodova)**

Za tip podatka `Red` koji je realiziran jednostruko povezanom listom definirane su funkcije za inicijalizaciju reda, dodavanje elementa u red i skidanje elementa iz reda:

```
void init_red(Red *red);
int dodaj (Kupac element, Red *red);
int skini (Kupac *element, Red *red);
```

Funkcije `dodaj` i `skini` vraćaju 1 ako je operacija dodavanja ili skidanja uspjela, a 0 inače.
Elementi reda su podaci tipa `Kupac` (šifra kupca (cijeli broj) i godina rođenja (cijeli broj)).

Korištenjem gore navedenih funkcija napišite funkciju **najstarijiNaprijed** koja će na početak reda staviti najstariju osobu u redu (prema godini rođenja). Ako takvih osoba ima više od jedne, one sve trebaju doći na početak reda u istom poretku u kojem su bile prije promjene. Poredak ostalih osoba u redu mora ostati nepromijenjen. Smiju se koristiti samo zadane funkcije i eventualno pomoćni redovi. Dan je primjer:

Prije promjene: izlaz->12;1988->75;**1945**->13;1946->87;1968->97,1968->23;**1945**<-ulaz
Nakon promjene: izlaz->75;**1945**->23;**1945**->12;1988->13;1946->87;1968->97,1968<-ulaz

**6. zadatak (8 bodova)**

a) (4 boda) Odredite apriornu složenost programskog odsječka i detaljno obrazložite odgovor. O kojem algoritmu je riječ?

```
void StoJaRadim (int A [], int N) {
  int i, j;
  int pom;
  for (i = 1; i < N; i++) {
    pom = A[i];
    for (j = i; j >= 1 && A[j-1] > pom; j--)
      A[j] = A[j-1];
    A[j] = pom;
  }
}
```

b) (4 boda) Odredite apriornu složenost programskih odsječaka i detaljno obrazložite odgovore:

```
1) int nekaFunkcija(int n) {
     int br = 0;
     int nasao = 0;
     do {
       br++;
       if (br == korak)
         nasao = 1;
       n /= 2;
     } while (n > 0);
     return br;
   }
```

```
2)   nasao = 0;
   for (double i=0; i<n; i++)
     for (double j=0; j<n-i; j++)
       if (i==j && a[i][j]==trazeni)
         nasao = 1;
```

Rješenja:

1. **(12 bodova)**

```
int broj_preljeva(FILE *f, int N, int *rbr){
	zapis pretinac[C];
	int i,j, ukupno = 0, max = 0, brojac = 0;

	for (i = 0; i < M; i++) {
		brojac =0;
		fseek (f, i*BLOK, SEEK_SET);
		fread (pretinac, sizeof (pretinac), 1, f);
		if (i == N) continue;
		for (j = 0; j < C; j++) {
			if (( pretinac[j].sifra != 0) &&
			adresa(pretinac[j].sifra) == N)) {
				ukupno++;
				brojac++;
			}
		}
		if (brojac > max) {
			max = brojac;
			*rbr = i;
		}

	}
	return ukupno;
}
```

2. **(13 bodova)**

```
void obrniListu(atom** glava) {
	atom* prvi;
	atom* pom;

	if (*glava == NULL) return;

	prvi = *glava;
	pom = prvi->sljed;

	if (pom == NULL) return;

	obrniListu(&pom);

	prvi->sljed->sljed = prvi;
	prvi->sljed = NULL;

	*glava = pom;
}
```

3. **(15 bodova)**

a) (9 bodova)

```
int razina(int sifra, cvor *korijen)
{
	int d;
	if (!(korijen)) return 0;
	if (sifra==korijen->element.sifra){
		return 1;
	}
	else{
		if (sifra<korijen->element.sifra){
			d=razina(sifra, korijen->lijevo);
		}
		else d=razina(sifra,korijen->desno);
		if (d) return 1+d;
		return 0;
	}
}
```

b) (6 bodova)

```
float zbrojCijenaNaRazini (int razina, cvor *korijen){
	if (!korijen) return 0;
	if (razina<1) return 0;
	if (razina==1) return korijen->element.cijena;
	return zbrojRazina(razina -1,korijen->lijevo)
	     +zbrojRazina(razina -1,korijen->desno);
}
```

4. (11 bodova)

```
void bezEkstrema(Stog *stog) {
	int elemMin, elemMax, stavka;
	Stog pomStog;

	if(!skini(&stavka, stog)) return; // nema se što raditi, stog je prazan

	// proglasi prvi element stoga trenutno najvećim i najmanjim
	elemMin=stavka; elemMax=stavka;

	init_stog(&pomStog);   // iniciraj pomoćni stog i stavi prvi element na njega
	dodaj(stavka,&pomStog);

	while(skini(&stavka,stog)){ // dok je elemenata
		if (stavka>elemMax) elemMax=stavka;  // provjeri je li novi možda najveći
		if (stavka<elemMin) elemMin=stavka;  // ili najmanji
		dodaj(stavka,&pomStog); // i stavi ga na pomoćni stog
	}

	while(skini(&stavka,&pomStog)){ // s pomoćnog stoga prepiši na originalni
		 // sve osim ekstrema
		if ((stavka<elemMax) && (stavka>elemMin)) dodaj(stavka,stog);
	}
}
```

5. (11 bodova)

```
void starijiNaprijed(Red *red){
  Red pom1, pom2;
  Kupac k;
  int najstariji, prvi=1;
  init_red (&pom1);
  init_red (&pom2);
  while (skini (&k, red)){
    if (prvi) {
      prvi=0;
      najstariji=k.godinarodj;
    }
    else if (k.godinarodj <najstariji) najstariji=k.godinarodj;
    dodaj (k,&pom1);
    dodaj (k,&pom2);
  }
  while (skini (&k, &pom1)){
    if (k.godinarodj == najstariji) dodaj (k,red);
  }
  while (skini (&k, &pom2)){
    if (k.godinarodj !=najstariji) dodaj (k,red);
  }
}
```

## *Zadatak 6. (8 bodova)*

a) (4 boda)

**$O(n^2)$**

Ovo je sortiranje umetanjem („Insertion sort“). Postoje dva dijela ulaznog statičkog polja A: sortirani i nesortirani. U svakom koraku algoritma sortirani dio se proširuje tako da se u njega na ispravno mjesto ubaci prvi element iz nesortiranog dijela polja. Vanjska petlja služi za određivanje granice sortiranog dijela, dok unutarnja ubacuje element u sortirani niz i pomiče ostale elemente. Složenost i u najgorem slučaju (kad je niz sortiran naopako) je $O(n^2)$.

b) (2 boda)

**$O(\log_2 n)$**

Koristi se do-while petlja i provjeravanje uvjeta (if naredba) nema utjecaja na duljinu izvođenja algoritma. Algoritam uzastopce cjelobrojno dijeli ulazni broj s 2 dok god je rezultat veći od 0. Dakle, petlja će se izvršiti onoliko puta koliko je n potencija broja 2, odnosno točnije, $\log_2 n + 1$ puta. Naredbe prije i poslije do-while petlje imaju konstantu složenost.

c) (2 boda)

**$O(n^2)$**

Kroz petlje se prolazi uvijek, bez obzira je li uvjet unutar petlje zadovoljen ili ne, jer nema break naredbe. Broj izvršavanja 1 (n=1), 55 (n=10), 5050 (n=100), 500500 (n=1000), itd. Točniji opis složenosti je $O(½\ n^2)$.