

Algoritmi i strukture podataka

Zadaci za vježbu za 1. međuispit

Dinamička alokacija memorije

Zadatak 1.

Napišite funkciju koja će iz ulaznog polja cijelih brojeva izbaciti sve parne brojeve, a zatim i dinamički smanjiti veličinu polja na osnovu novog broja članova. Funkcija kao rezultat vraća pokazivač na navedeno polje. Prototip funkcije je:

```
int *izbaci_parne(int *polje, int *br_clanova)
```

Također, napisati i glavni program u kojem će se polje dinamički alocirati, napuniti slučajno odabranim vrijednostima, pozvati funkcija i ispisati novo polje te osloboditi dinamički stvorenu memoriju.

Napomena: međusobni poredak neparnih članova nakon izbacivanja nije bitan.

Zadatak 2.

Napišite funkciju čiji je prototip:

```
int *stvari_polje(int a, int b, int n)
```

koja će stvoriti polje od n slučajno odabranih cijelih brojeva u intervalu [a,b]. Funkcija kao rezultat vraća pokazivač na navedeno polje.

Također, napisati i glavni program u kojem će se pozvati funkcija, ispisati polje te osloboditi dinamički stvorena memorija.

Zadatak 3.

Napišite funkciju čiji je prototip:

```
int *stvari_polje(int a, int b, int *br_clanova)
```

koja će stvoriti polje i napuniti ga slučajno odabranim cijelim brojevima. Broj članova polja je slučajan broj u intervalu [a,b]. Funkcija kao rezultat vraća pokazivač na navedeno polje.

Također, napisati i glavni program u kojem će se pozvati funkcija, ispisati polje te osloboditi dinamički stvorena memorija.

Zadatak 4.

Napišite funkciju čiji je prototip:

```
int *proširi(int *polje, int br_clanova, int n)
```

koja će proširiti ulazno polje od br_clanova sa n slučajno odabranih cijelih brojeva u intervalu [50,100]. Funkcija kao rezultat vraća pokazivač na navedeno polje.

Također, napisati i glavni program u kojem će se pozvati funkcija, ispisati polje te osloboditi dinamički stvorena memorija.

Zadatak 5.

Napišite funkciju čiji je prototip:

```
int *nadodaj(int *polje, int *br_clanova)
```

koja će proširiti ulazno polje od br_clanova tako da na kraj nadopíše sve parne brojeve.

Funkcija kao rezultat vraća pokazivač na navedeno polje.

Također, napisati i glavni program u kojem će se pozvati funkcija, ispisati polje te osloboditi dinamički stvorena memorija.

Zadatak 6.

Napišite funkciju čiji je prototip:

```
int *kopiraj(int *polje, int br1, int *br2)
```

koja će stvoriti novo polje koje će sadržavati sve neparne brojeve iz ulaznog polja. Broj elemenata u početnom polju je br1. Funkcija kao rezultat vraća pokazivač na navedeno polje. Ako u početnom polju nema neparanih brojeva funkcija vraća NULL.

Također, napisati i glavni program u kojem će se pozvati funkcija, ispisati polje te osloboditi dinamički stvorena memorija.

Raspršeno adresiranje

Zajednički dio za sve zadatke:

Jedan zapis datoteke organizirane po principu raspršenog adresiranja je definiran sljedećom strukturom:

```
typedef struct{
    int sifra;
    char naziv[50+1];
    int kolicina;
    float 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

Ključ zapisa je šifra artikla, a transformacija ključa u adresu se obavlja zadanom funkcijom:

```
int adresa(int sifra);
```

Zadatak 1.

Napisati funkciju koja će vratiti broj pretinaca u kojima se nalazi više od n zapisa koji su upisani kao preljev. Funkcija treba imati prototip:

```
int broji(FILE *f, int n);
```

Zadatak 2.

Napisati funkciju koja će u datoteci pronaći i vratiti zapis o artiklu sa zadanom šifrom. Funkcija preko imena mora vratiti 1 ako se zapis nalazi u svom pretincu, -1 ako je upisan kao preljev, a 0 ako se ne nalazi u datoteci. Funkcija mora imati prototip:

```
int pronadji(FILE *f, int sifra, zapis *z);
```

Zadatak 3.

Napisati funkciju koja će u datoteku upisati novi zapis. Funkcija treba vratiti 1 ako je zapis upisan u svoj pretinac, -1 ako je upisan kao preljev, a 0 ako zapis nije upisan u datoteku (nema mjesta). Funkcija treba imati prototip:

```
int upisi_zapis (FILE *f, zapis z);
```

Napomena: možete pretpostaviti da zapis z ne postoji u datoteci prije poziva funkcije upis.

Zadatak 4.

Napisati funkciju koja će vratiti broj zapisa koji su upisani kao preljev. Funkcija treba imati prototip:

```
int broji_preljeve(FILE *f);
```

Zadatak 5.

Napisati funkciju koja će pronaći prosječnu „udaljenost“ zapisa od predviđenog pretinca. Ukoliko pojedini zapis nije spremljen kao preljev njegova udaljenost je 0, a inače udaljenost se definira kao broj pretinaca koje je dodatno potrebno pročitati da bi se zapis pronašao. *Primjerice, ukoliko je $M=15$, a adresa nekog zapisa 13, a zapis se nalazi u pretincu broj 4, udaljenost je 6.*

```
float avg_udaljenost(FILE *f);
```

Napomena: Možete pretpostaviti da datoteka nije prazna.

Zadatak 6.

Napisati funkciju koja će u datoteci pronaći i vratiti zapis o artiklu sa zadanom šifrom. Funkcija preko imena mora vratiti broj dodatno pretraženih pretinaca prilikom traženja zapisa (0 ako se zapis nalazi u svom pretincu, 1 ako je odmah u sljedećem pretincu, ..., M ukoliko se zapis ne nalazi u datoteci). Funkcija mora imati prototip:

```
int trazi_zapis(FILE *f, int sifra, zapis *z);
```

Složenost

Zadatak 1.

Odredite apriornu složenost sljedeće funkcije i detaljno obrazložite svoje odgovore. Kada nastupaju najgori i najbolji slučaj?

```
a) int Funkcija1(int *mat, int m, int n, int maxstu)
{
    int i, j, suma;
    for (i = 0; i < n; i++)
        for (j = 0; j <= i; j++)
            if (mat[i*maxstu+j]%2 == 0) suma +=
mat[i*maxstu+j];
    return suma;
}
```

```
b) int Funkcija2(int *mat, int m, int n, int maxstu)
{
    int i, j, suma;
    for (i = 0; i < n; i++)
        for (j = 0; j <= i; j++)
        {
            if (mat[i*maxstu+j]%2 == 0) suma +=
mat[i*maxstu+j];
            else break;
        }
    return suma;
}
```

Zadatak 2.

Odredite apriornu složenost sljedećih programskih odsječaka i detaljno obrazložite svoje odgovore. Za koje i i j nastupaju najbolji i najgori slučaj?

```
a) nasao = 0;
   for (i=0; i<n; i++) {
       if (polje[i][i] == zadani) {
           nasao = 1;
           break;
       }
   }

b) nasao = 0;
   for (i=0; i<n; i++) {
       for (j=0; j<n; j++) {
           if (i==j && polje[i][j]==zadani)
               nasao = 1;
       }
   }
```

Zadatak 3.

Odredite apriornu složenost sljedeće funkcije. Detaljno obrazložite svoj odgovor.

```
int B(int a[], int x, int n) {
    int donji, srednji, gornji;
    donji = 0; gornji = n - 1;
    while (donji <= gornji) {
        srednji = (donji + gornji)/2;
        if (a[srednji] < x) donji = srednji + 1;
        else if (a[srednji] > x) gornji = srednji - 1;
        else return srednji;
    }
    return -1;
}
```

Zadatak 4.

Odredite apriornu složenost sljedeće funkcije. Detaljno obrazložite svoj odgovor.

```
int BrZnamenki(int n)
{
    int br = 0;
    while(n)
    {
        br++;
        n = n/10;
    }
    return br;
}
```

Sistemiški stog

Zadatak 1.

Koliko je elemenata dodano na sistemiški stog prilikom poziva i izvršavanja funkcije funkc:

```
int funkc (int a, int b, int c)
{
    int r1, r2;
    r1=a+b;
    r2=b-c;
    return r1/r2;
}
```

Skicirati izgled stoga.

Zadatak 2.

Skicirati sistemiški stog prilikom poziva i izvršavanja funkcije zbroji.

Funkcija je pozvana na sljedeći način:

```
zbroji (2, 3);
```

```
void Ispisi (int d) {
    printf ("Rezultat je %d: ", d);
}

void zbroji (int a, int b) {
    int c;
    c = a + b;
    Ispisi (c);
}
```

Rekurzija

Zadatak 1.

Napisati rekurzivnu funkciju koja za zadani n računa aproksimaciju broja Pi kao sumu prvih n članova reda:

$$\pi = \sum_{i=0}^{\infty} (-1)^i \frac{4}{2^{*i+1}}$$

Prototip funkcije je:

```
double IzracunajPI_Rek(int n);
```

Zadatak 2.

Napisati rekurzivnu funkciju za binarno pretraživanje polja cijelih brojeva.

Funkcija treba imati prototip:

```
int BinTraz2 (int a[], int x, int n);
```

RJEŠENJA:

Dinamička alokacija memorije

Zadatak 1.

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

int *izbaci_parne(int *polje, int *br_clanova){
    int i, j;
    for(i=0 ; i<*br_clanova ; ){
        if (polje[i] % 2 == 0){1
            for(j=i+1 ; j<*br_clanova ; j++){
                polje[j-1] = polje[j];
                (*br_clanova)--;
            }
            else
                i++;
        }
        polje = (int*) realloc(polje, *br_clanova * sizeof(int));
        return polje;
    }
}

int main(){
    int *polje, i, br_clanova = 20;
    polje = (int*) malloc(br_clanova * sizeof(int));

    srand((unsigned) time(NULL));

    for(i=0; i<br_clanova ; i++){
        polje[i] = rand();
    }

    printf("\nPrije izbacivanja\n");
    for(i=0; i<br_clanova ; i++){
        printf("%d\n", polje[i]);
    }

    polje = izbaci_parne(polje, &br_clanova);

    printf("\nNakon izbacivanja\n");
    for(i=0; i<br_clanova ; i++){
        printf("%d\n", polje[i]);
    }
    free(polje);

    return 0;
}
```

Zadatak 2.

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

¹ Ako poredak elemenata nije bitan, onda se, umjesto petlje koja pomiče elemente za jedno mjesto ulijevo počevši od indeksa i+1, može zadnji element niza kopirati na mjesto onog broja koji se izbacuje.

```

int *stvari_polje(int a, int b, int n){
    int i, *polje;
    polje = (int*) malloc(n * sizeof(int));
    for(i=0; i<n ; i++)
        polje[i] = a + rand()%(b-a+1);
    return polje;
}

int main(){
    int *polje, i, a=15 ,b = 35 ,n = 20;

    srand((unsigned) time(NULL));

    polje = stvari_polje(a, b, n);
    for(i=0; i<n ; i++){
        printf("%d\n", polje[i]);
    }
    free(polje);

    return 0;
}

```

Zadatak 3.

```

int *stvari_polje(int a, int b, int *br_clanova){
    int i, *polje;
    *br_clanova = a + rand()%(b-a+1);
    polje = (int*) malloc(*br_clanova * sizeof(int));
    for(i=0; i<*br_clanova ; i++)
        polje[i] = rand();
    return polje;
}

int main(){
    int *polje, i, a=5 ,b = 15 ,n;

    srand((unsigned) time(NULL));
    polje = stvari_polje(a, b, &n);
    for(i=0; i<n ; i++){
        printf("%d\n", polje[i]);
    }
    free(polje);

    return 0;
}

```

Zadatak 4.

```

int *prosiri(int *polje, int br_clanova, int n){
    int i;
    polje = (int*) realloc(polje, (br_clanova+n) * sizeof(int));
    for(i= br_clanova; i < br_clanova + n ; i++)
        polje[i] = 50+rand()%(100-50+1);
    return polje;
}

int main(){
    int *polje, i, br_clanova = 10, n=5;
    polje = (int*) malloc(br_clanova * sizeof(int));

    srand((unsigned) time(NULL));
}

```

```

for(i=0; i<br_clanova ; i++){
    polje[i] = rand();
}

printf("\nPrije prosirenja\n");
for(i=0; i<br_clanova ; i++){
    printf("%d\n", polje[i]);
}

polje = prosiri(polje, br_clanova, n);
printf("\nNakon prosirenja\n");
for(i=0; i<br_clanova+n ; i++){
    printf("%d\n", polje[i]);
}
free(polje);

return 0;
}

```

Zadatak 5.

```

int *nadodaj(int *polje, int *br_clanova){
    int i, br;
    br = *br_clanova;
    for(i=0 ; i < br ; i++){
        if (polje[i] % 2 == 0){
            (*br_clanova)++;
            polje = (int*) realloc(polje, (*br_clanova) *
                sizeof(int));
            polje[*br_clanova - 1] = polje[i];
        }
    }
    return polje;
}

int main(){
    int *polje, i, br_clanova = 10;
    polje = (int*) malloc(br_clanova * sizeof(int));

    srand((unsigned) time(NULL));

    for(i=0; i<br_clanova ; i++){
        polje[i] = rand()%20;
    }

    printf("\nPrije prosirenja\n");
    for(i=0; i<br_clanova ; i++){
        printf("%d\n", polje[i]);
    }

    polje = nadodaj(polje, &br_clanova);
    printf("\nNakon prosirenja\n");
    for(i=0; i<br_clanova ; i++){
        printf("%d\n", polje[i]);
    }
    free(polje);

    return 0;
}

```

Zadatak 6.

```
int *kopiraj(int *polje, int br1, int *br2){
    int i,j=0, *novo;
    *br2 = 0;
    for(i=0; i < br1 ; i++){
        if (polje[i] % 2)
            (*br2)++;
    }
    if (*br2 > 0){
        novo = (int*) malloc(*br2 * sizeof(int));
        for(i=0; i < br1 ; i++){
            if (polje[i] % 2)
                novo[j++] = polje[i];
        }
        return novo;
    }
    else
        return NULL;
}

int main(){
    int polje[15], *novo, i, br2, br_clanova = 15;

    srand((unsigned) time(NULL));

    for(i=0; i<br_clanova ; i++){
        polje[i] = rand()%20;
    }

    printf("\nPrvo polje\n");
    for(i=0; i<br_clanova ; i++){
        printf("%d\n", polje[i]);
    }

    novo = kopiraj(polje, br_clanova, &br2);
    if (novo != NULL){
        printf("\nDrugo polje\n");
        for(i=0; i<br2 ; i++){
            printf("%d\n", novo[i]);
        }
        free(novo);
    }

    return 0;
}
```

Raspršeno adresiranje

Zadatak 1.

```
int broji(FILE *f, int n){
    zapis pretinac[C];
    int i, j, broj_preljeva_u_pretincu;
    int br = 0;

    for (i = 0; i < M; i++) {
        broj_preljeva_u_pretincu = 0;
        fseek (f, i*BLOK, SEEK_SET);
        fread (pretinac, sizeof (pretinac), 1, f);
        for (j = 0; j < C; j++) {
            if (pretinac[j].sifra != 0) {
                /* Ako zapis nije prazan */
                if (adresa(pretinac[j].sifra) != i)
                    broj_preljeva_u_pretincu++;
            }
        }
        if (broj_preljeva_u_pretincu > n)
            br++;
    }
    return br;
}
```

Zadatak 2.

```
int pronadji(FILE *f, int sifra, zapis *z){
    zapis pretinac[C];
    int i, j, poc;
    i = adresa (sifra);
    /* Upamtiti izračunatu adresu kao početnu */
    poc = i;
    do {
        /* Ponavljati dok se podatak ne pronađe ili dok se ne
           ustanovi da ga nema */
        fseek (f, i*BLOK, SEEK_SET);
        fread (pretinac, sizeof (pretinac), 1, f);
        for (j = 0; j < C; j++) {
            if (pretinac[j].sifra != 0) {
                /* Ako zapis nije prazan */
                if (pretinac[j].sifra == sifra){
                    *z = pretinac[j];
                    return poc == i ? 1 : -1; /* da li je pronađeni
                                               zapis bio preljev ili ne?*/
                }
            }
        }
        /* Pretinac je pun, prijeđi ciklički na sljedećega */
        i = (i + 1) % M;
    }
    while (i != poc); /*Ponavljati dok se ne vratimo na početni
    pretinac*/
    return 0; /* Svi pretinci posjećeni, zapis nije pronađen */
}
```

Zadatak 3.

```
int upisi_zapis (FILE *f, zapis z) {
    int i, j, poc;
    zapis pretinac[C];
    i = adresa (z.sifra);
    /* Upamtiti izračunatu adresu kao početnu */
    poc = i;
    do {
        /* Ponavljati dok se podatak ne upiše u neki od pretinaca ili
        dok se ne ustanovi da je datoteka puna
        */
        fseek (f, i*BLOK, SEEK_SET);
        /*Pročitati cijeli pretinac odjednom, pa ga 'pregledati' u
        memoriji*/
        fread (pretinac, sizeof (pretinac), 1, f);
        for (j = 0; j < C; j++) {
            if (pretinac[j].sifra == 0) {
                pretinac[j] = z;
                fseek (f, i*BLOK, SEEK_SET);
                fwrite (pretinac, sizeof (pretinac), 1, f);
                fflush(f);
                return poc == i ? 1 : -1 ;
            }
        }
        /* U pretincu nema mjesta, treba ciklički prijeći na sljedećega
        */
        i = (i + 1) % M;
    }
    while (i != poc); /*Ponavljati dok se ne vratimo na početni
    pretinac*/

    /* Ni u jednom pretincu nije bilo mjesta */
    return 0;
}
```

Zadatak 4.

```
int broji_preljeve(FILE *f){
    zapis pretinac[C];
    int i, j;
    int br = 0;

    for (i = 0; i < M; i++) {
        fseek (f, i*BLOK, SEEK_SET);
        fread (pretinac, sizeof (pretinac), 1, f);
        for (j = 0; j < C; j++) {
            if (pretinac[j].sifra != 0){/*Ako zapis nije
prazan*/

                /*Da li je zapis bio preljev*/
                if (adresa(pretinac[j].sifra) != i)
                    br++;
            }
        }
    }
    return br;
}
```

Zadatak 5.

```
float avg_udaljenost(FILE *f){
    zapis pretinac[C];
    int i, j;
    int udaljenost, br = 0, sum=0;

    for (i = 0; i < M; i++) {
        fseek (f, i*BLOK, SEEK_SET);
        fread (pretinac, sizeof (pretinac), 1, f);
        for (j = 0; j < C; j++) {
            if (pretinac[j].sifra != 0) {
                udaljenost = i - adresa(pretinac[j].sifra);
                if (udaljenost < 0)
                    udaljenost += M;
                sum += udaljenost; br++;
            }
        }
    }
    return (float) sum/br;
}
```

Zadatak 6.

```
int trazi_zapis(FILE *f, int sifra, zapis *z){
    zapis pretinac[C];
    int i, j, poc;
    int udaljenost;
    i = adresa (sifra);

    poc = i;
    do {

        fseek (f, i*BLOK, SEEK_SET);
        fread (pretinac, sizeof (pretinac), 1, f);
        for (j = 0; j < C; j++) {
            if (pretinac[j].sifra != 0) {

                if (pretinac[j].sifra == sifra){
                    *z = pretinac[j];
                    udaljenost = i - poc;
                    if (udaljenost < 0)
                        udaljenost += M;
                    return udaljenost;
                }
            }
        }

        i = (i + 1) % M;
    }
    while (i != poc);
    return M;
}
```

Složenost

Zadatak 1.

- Složenost je $O(n^2)$. Kroz petlje prolazimo uvijek, bez obzira je li uvjet unutar petlje zadovoljen ili ne. Unutarnja petlja (petlja po j) ide do i , tako da je točan broj naredbi koje će se izvršiti jednak $2(1+2+3+\dots+n) = 2n(n+1) = 2n^2+2n$. Najbolji i najgori slučaj su dakle isti.
- Složenost je $O(n^2)$. Najbolji slučaj nastupa kada je prvi element svakog retka neparan broj, te se unutarnja petlja svaki put izvršava samo jednom, a vanjska petlja se izvršava n puta. Najgori slučaj nastupa kada su svi elementi parni, pa je broj naredbi jednak $2(n^2+n)$.

Zadatak 2.

- Složenost je $O(n)$. Ako uvjet nikad ne bude zadovoljen, petlju će se izvršiti n puta.
Najbolji slučaj nastupa kada je zadani element jednak vrijednosti elementa polje[0][0], pa se nakon samo jednog prolaza izlazi iz petlje. Broje li se naredbe `if (...)` i `nasao = 1`, broj naredbi koje će se izvršiti je 2.
Najgori slučaj nastupa kada se zadani element pronađe u zadnjem koraku petlje; kroz petlju se tada prolazi n puta. Broje li se naredbe `if (...)` i `nasao = 1`, broj naredbi koje će se izvršiti je $n+1$.
- Složenost je $O(n^2)$. Bez obzira na ispitivanje uvjeta, nikad se ne izlazi iz petlje (nema `break` naredbe). Najbolji i najgori slučaj su dakle isti – broj naredbi koje će se izvršiti je $2n^2$.

Zadatak 3.

Funkcija je algoritam binarnog pretraživanja. Složenost je $O(\log n)$.

Do te složenosti dolazimo na sljedeći način:

- Naredbe izvan `while` petlje zahtjevaju konstantno vrijeme izvođenja.
- Naredbe unutar `while` petlje također zahtjevaju konstantno vrijeme izvođenja (točan broj naredbi).
- Složenost ovisi o tome koliko će se puta obaviti sama petlja. Pitanje možemo preformulirati u: *Za dati n , koliko koraka treba da bi dijeleći n s 2 došli do 1?* Naime, ono što u binarnom pretraživanju i radimo jest da u svakom koraku dijelimo polje od n elemenata na dva veličine $n/2$ te odabiremo samo jedno od ta dva polja.

Dakle u prvom koraku imamo niz duljine n , u drugom $n/2$, pa $n/4$, ..., u k -tom koraku $n/2^{k-1}$. Tražimo k za koji je $n/2^{k-1} = 1$. Rješenje je $k = \lfloor \log_2 n \rfloor + 1$.

Zadatak 4.

Složenost je $O(\log n)$.

Do te složenosti dolazimo na sljedeći način:

- Naredbe izvan `while` petlje zahtjevaju konstantno vrijeme izvođenja.
- Naredbe unutar `while` petlje također zahtjevaju konstantno vrijeme.
- Složenost ovisi o tome koliko će se puta obaviti sama petlja.

Petlja će stati izvršavati kada n postane 0. U petlji n svaki put cjelobrojno dijelimo s 10 te tako dobivamo broj koji je za jednu znamenku manji od n . Petlja će se dakle, izvršiti onoliko puta koliko n ima znamenki, a to je $\lfloor \log n \rfloor + 1$.

Sistemiški stog

Rješenje 1:

Na sistemiški stog je dodano 6 elemenata

r2
r1
povratna adresa iz funkc
a
b
c

Rješenje 2:

Povratna adresa iz Ispisi
d
c
povratna adresa iz zbroji
a
b

Rekurzija

Zadatak 1.

```
float IzracunajPI_Rek(int n)
{
    if (n<0)
    {
        return 0;
    }
    else
        return IzracunajPI_Rek(n-1) +
            (n%2 == 0 ? 4. : -4.)/(2*n+1);
}
```

Zadatak 2.

```
int BinTraz2 (int a[], int x, int n)
{
    int srednji,pom;
    if(n<=0) return -1;
    srednji=(n-1)/2;
    if(a[srednji]>x)
    {
        return BinTraz2(a,x,srednji);
    }
    else if(a[srednji]<x)
    {
        pom = BinTraz2(a+srednji+1,x,n-srednji-1);
        if(pom==-1)
            return -1;
        else
            return pom+srednji+1;
    }
    else
        return srednji;
}
```