

Zadaci za Stog

Zadatak 1.

Neka je zadan tip podatka `Stog` za koji su definirane funkcije za inicijalizaciju stoga, dodavanje elementa na stog te brisanje elementa sa stoga. Elementi stoga su podaci **tipa Stavke** (računa) koji sadrže ime stavke (40 znakova), količinu (cijeli broj) te cijenu jedinice stavke. Prototipovi navedenih funkcija su:

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

Funkcije `dodaj` i `skini` vraćaju 1 ako je operacija dodavanja ili skidanja uspjela, a 0 inače.

- a) Napišite nerekurzivnu i rekurzivnu funkciju koja će sa stoga maknuti sve stavke čija je količina 0, a onima koje ostaju promijeniti cijenu za zadani postotak. Funkcija vraća ukupni broj stavki na stogu. Prototip funkcije treba biti:

```
int AzurirajRacun (Stog *stog, float postotak);
```

```

int AzurirajRacun(Stog * stog, float postotak){
    int kolicina=0;
    tip element;
    Stog pom;
    init_stog(&pom);

    while (skini(&element, stog)){
        if (element.kolicina>0) {
            element.cijena=(1+postotak/100)*element.cijena;
            kolicina+=element.kolicina;
            dodaj(element, &pom);
        }
    }
    while (skini(&element, &pom)){
        dodaj(element, stog);
    }
    return kolicina;
}

int AzurirajRacun_Rek(Stog * stog, float postotak){
    int kolicina;
    tip element;
    if (skini(&element, stog)){
        kolicina= AzurirajRacun_Rek(stog, postotak);
        if (element.kolicina>0) {
            element.cijena=(1+postotak/100)*element.cijena;
            dodaj(element, stog);
        }

        return kolicina+element.kolicina;
    }
    else
        return 0;
}

```

- b) Napišite **definicije** odgovarajućih struktura (tipovi podataka **Stavka**, **Stog** i **atom**) za stog realiziran vezanom listom.

```

struct stavka{
    char naziv[40];
    int kolicina;
    float cijena;
};

typedef struct stavka tip;

struct at {
    tip element;
    struct at *sljed;
};

typedef struct at atom;

typedef struct{
    atom *vrh;
} Stog;

```

Zadatak 2.

Neka je zadan 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 čiji je prototip:

```
Stog KreirajStogPoZbroju(Stog *stog, int zbrojZnamenaka);
```

koja mora vratiti novi stog koji se sastoji od onih brojeva s ulaznog stoga čiji je zbroj znamenaka jednak parametru `zbrojZnamenaka`.

Nakon završetka funkcije `KreirajStogPoZbroju` sadržaj ulaznog stoga mora ostati nepromijenjen.

```

Stog KreirajStogPoZbroju(Stog *stog, int zbrojZnamenaka){
    Stog pom, stog2;
    int n, broj, zbroj;

    init_stog(&pom); init_stog(&stog2);

    while(skini(&n, stog)){
        dodaj(n, &pom);
    }

    while(skini(&n, &pom)){
        dodaj(n, stog);
        zbroj=0;
        broj=n;
        while(broj){
            zbroj+=broj%10;
            broj/=10;
        }
        if (zbroj==zbrojZnamenaka)
            dodaj(n, &stog2);
    }

    return stog2;
}

Stog KreirajStogPoZbroju_rek(Stog *stog, int zbrojZnamenaka){
    Stog stog2;
    int n, broj, zbroj;

    if (skini(&n, stog)){
        stog2= KreirajStogPoZbroju_rek(stog, zbrojZnamenaka);
    }
    else {
        init_stog(&stog2);
        return stog2;
    }

    dodaj(n, stog);
    zbroj=0;
    broj=n;
    while(broj){
        zbroj+=broj%10;
        broj/=10;
    }
    if (zbroj==zbrojZnamenaka) dodaj(n, &stog2);
    return stog2;
}

```

Zadatak 3.

Na stog realiziran jednostruko povezanom listom spremaju se realni brojevi (**float**). Napisati dodatnu funkciju koja će duplicirati zadani stog (napraviti kopiju stoga, tako da početni stog ostane sačuvan). Funkcija treba imati prototip:

```
Stog dupliciraj(Stog *stog);
```

Dodatna funkcija smije pristupati elementima stoga samo koristeći funkcije

```
int dodaj(float element, Stog *stog);
```

```
int skini(float *element, Stog *stog);
```

```
Stog dupliciraj(Stog * stog) {
    Stog stogDuplic;
    Stog pom;
    tip s;
    init_stog(&stogDuplic);
    init_stog(&pom);

    // prebacivanje u pomoćni stog
    while(skini(&s, stog)) {
        dodaj(s, &pom);
    }
    // vraćanje u početni i stvaranje duplikata
    while(skini(&s, &pom)) {
        dodaj(s, stog);
        dodaj(s, &stogDuplic);
    }

    return stogDuplic;
}

Stog dupliciraj_rek(Stog * stog) {
    Stog stogDuplic;
    tip s;

    if(skini(&s, stog)) {
        stogDuplic= dupliciraj_rek(stog);
    }
    else
    {
        init_stog(&stogDuplic);
        return stogDuplic;
    }
    // vraćanje u početni i stvaranje duplikata
    dodaj(s, stog);
    dodaj(s, &stogDuplic);

    return stogDuplic;
}
```

Zadaci sa redom

Zadatak 1.

Neka je zadan **tip** podatka **Red** za koji su definirane funkcije za inicijalizaciju reda, dodavanje elementa u red te za brisanje elementa iz reda. Prototipovi navedenih funkcija su:

```
void init_red(Red *red);
int DodajURed (int element, Red *red);
int SkiniIzReda (int *element, Red *red);
```

Funkcije DodajURed i SkiniIzReda vraćaju 1 ako je operacija dodavanja ili skidanja uspjela, a 0 inače.

Napisati funkciju čiji je prototip

```
void raspari (Red *red);
```

koja će red preurediti tako da se na prvom mjestu preuređenog reda nalazi prvi parni, na drugom mjestu prvi neparni i tako naizmjenice do kraja. Ako ponestane nekih elemenata, treba iskoristiti sve preostale, stavivši ih na kraj preuređenog reda. Primjerice, ako je red sadržavao sljedeće elemente:

8 7 7 4 5 5 1 2 3 6 11 4

nakon poziva funkcije red će izgledati ovako:

8 7 4 7 2 5 6 5 4 1 3 11.

```
void raspari (Red *red) {
    int n, n1, n2, prvi;
    Red parni, neparni;

    init_red(&parni);
    init_red(&neparni);

    while (SkiniIzReda (&n, red)) {
        if (n%2)
            DodajURed (n, &neparni);
        else
            DodajURed (n, &parni);
    }

    do{
        if (n1=SkiniIzReda (&n, &parni))
            DodajURed (n, red);
        if (n2=SkiniIzReda (&n, &neparni))
            DodajURed (n, red);
    }
    while (n1 || n2);
}
```

Zadatak 2.

Napisati funkciju koja će dva sortirana reda spojiti u jedan red koji će također biti sortiran. Funkcija treba imati prototip:

```
Red Sortiraj (Red *red1, Red *red2);
```

Redovi sadrže podatke tipa **int** i sortirani su uzlazno. Spojeni red također treba biti sortiran uzlazno.

```
Red Sortiraj (Red *red1, Red *red2){
    int n1, n2;
    tip elem1, elem2;
    Red redS;
    init_red(&redS);

    n1=SkiniIzReda(&elem1, red1);
    n2=SkiniIzReda(&elem2, red2);

    while(n1 || n2){
        if (!n1) {
            DodajURed (elem2, &redS);
            n2=SkiniIzReda(&elem2, red2);
        }
        else if (!n2) {
            DodajURed (elem1, &redS);
            n1=SkiniIzReda(&elem1, red1);
        }
        else
            if (elem1 <elem2){
                DodajURed (elem1, &redS);
                n1=SkiniIzReda(&elem1, red1);
            }
            else{
                DodajURed (elem2, &redS);
                n2=SkiniIzReda(&elem2, red2);
            }
    }
    return redS;
}
```

Zadaci sa sortovima

sortiranje biranjem (<i>selection sort</i>)	pronađi najmanji element niza i zamijeni ga s prvim elementom niza, ponavljaj s ostatkom niza, smanjujući nesortirani dio	$O(n^2)$
bubble sort	zamjena susjednih elemenata ako nisu u dobrom redoslijedu u svakom koraku najveći element dolazi na zadnji-korak mjesto	$O(n^2)$
sortiranje umetanjem (<i>insertion sort</i>)	u svakom koraku sortirani dio se proširuje tako da se u njega na ispravno mjesto ubaci prvi element iz nesortiranog dijela niza	$O(n^2)$
Shellov sort	modificirani sort umetanjem	prosječno $O(n^{5/4})$, najgori slučaj $O(n^2)$
mergesort	koristi se strategija "podijeli pa vladaj" uz rekurziju nesortirani niz podijeli se na dva niza podjednake veličine svaki podniz sortira se rekurzivno, dok se ne dobije niz od 1 elementa	trajanje je $O(n \log^2 n)$
quick sort	odaberi bilo koji član v u polju S . To je stožer (pivot) podijeli preostale članove polja S , $S \setminus \{v\}$ u dva odvojena skupa: $S_1 = \{ x \in S \setminus \{v\} \mid x \leq v \}$ (sve što je manje od stožera, preseli lijevo) $S_2 = \{ x \in S \setminus \{v\} \mid x \geq v \}$ (sve što je veće od stožera, preseli desno) vrati niz sastavljen od {quicksort (S_1), v , quicksort (S_2)}	prosječno vrijeme izvođenja je $O(n \log n)$ najgori slučaj je $O(n^2)$ uz krivi odabir stožera (min ili max član), dobiva se n particija i za svaku je vrijeme izvođenja $O(n)$

```
typedef int tip;
```

```
// zamjena vrijednosti *lijevo i *desno
inline void Zamijeni (tip *lijevo, tip *desno) {
    tip pom = *lijevo;
    *lijevo = *desno;
    *desno = pom;
}
```

```
// sort selekcijom
void SelectionSort (tip A [], int N) {
    int i, j, min;
    for (i = 0; i < N; i++) {
        min = i;
        for (j = i+1; j < N; j++) {
            if (A[j] < A[min]) min = j;
        }
    }
}
```

```

    }
    Zamijeni(&A[i], &A[min]);
}
}

void BubbleSortPoboljsani (tip A [], int N) {
    int i, j, BilaZamjena;
    for (i = 0, BilaZamjena = 1; BilaZamjena; i++) {
        BilaZamjena = 0;
        for (j = 0; j < N-1-i; j++) {
            if (A[j+1] < A[j]) {
                Zamijeni (&A[j], &A[j+1]);
                BilaZamjena = 1;
            }
        }
    }
}

void InsertionSort (tip A [], int N) {
    int i, j;
    tip 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;
    }
}

// Shell sort
void ShellSort (tip A [], int N) {
    int i, j, korak;
    tip pom;
    for (korak = N / 2; korak > 0; korak /= 2) {
        //printf("\nkorak=%d\n", korak);
        // Insertion sort s većim korakom
        for (i = korak; i < N; i++) {
            //printf("\ni=%d:", i);
            pom = A [i];
            for (j = i; j >= korak && A[j-korak] > pom; j -= korak) {
                //printf("%d:%d ", j, j-korak);
                A [j] = A [j - korak];
            }
            A [j] = pom;
        }
    }
}

void Merge (tip A [], tip PomPolje [], int LPoz, int DPoz, int DesniKraj) {
    int i, LijeviKraj, BrojClanova, PomPoz;
    LijeviKraj = DPoz - 1;
    PomPoz = LPoz;
    BrojClanova = DesniKraj - LPoz + 1;
    // glavna pelja
    while (LPoz <= LijeviKraj && DPoz <= DesniKraj) {
        if (A [LPoz] <= A [DPoz])
            PomPolje [PomPoz++] = A [LPoz++];
        else
            PomPolje [PomPoz++] = A [DPoz++];
    }
    while (LPoz <= LijeviKraj)
        // Kopiraj ostatak prve polovice
        PomPolje [PomPoz++] = A [LPoz++];
    while (DPoz <= DesniKraj)

```

```

    // Kopiraj ostatak druge polovice
    PomPolje [PomPoz++] = A [DPoz++];
for (i = 0; i < BrojClanova; i++, DesniKraj--)
    // Kopiraj PomPolje natrag
    A [DesniKraj] = PomPolje [DesniKraj];
}

// MergeSort - rekurzivno sortiranje podpolja
void MSort (tip A [], tip PomPolje[], int lijevo, int desno ) {
    int sredina;
    if (lijevo < desno) {
        sredina = (lijevo + desno) / 2;
        MSort (A, PomPolje, lijevo, sredina);
        MSort (A, PomPolje, sredina + 1, desno);
        Merge (A, PomPolje, lijevo, sredina + 1, desno);
    }
}

// MergeSort - sort udruživanjem

void MergeSort (tip A [], int N) {
    tip *PomPolje;
    PomPolje = malloc (N * sizeof (tip));
    if (PomPolje != NULL) {
        MSort (A, PomPolje, 0, N - 1);
        free (PomPolje);
    } else
        Fatalno ("Nema mjesta za PomPolje!");
}

void Qsort (tip A [], int lijevo, int desno) {
    int i, j;
    tip stozer;
    if (lijevo + Cutoff <= desno) {
        stozer = medijan3 (A, lijevo, desno);
        i = lijevo; j = desno - 1;
        while (1) {
            while (A [++i] < stozer);
            while (A [--j] > stozer);
            if (i < j)
                Zamijeni (&A [i], &A [j]);
            else
                break;
        }
        // Obnovi stozer
        Zamijeni (&A [i], &A [desno - 1]);
        Qsort (A, lijevo, i - 1);
        Qsort (A, i + 1, desno);
    } else {
        // Sortiraj podpolje
        InsertionSort (A + lijevo, desno - lijevo + 1);
    }
}

// QuickSort
void QuickSort (tip A [], int N) {
    Qsort (A, 0, N - 1);
}

// Quicksort, sožer je prvi element
void Qsort2 (tip A[], int lijevo, int desno) {
    int i, j;

```

```
i = lijevo+1;
j = desno;

if (lijevo >= desno) return;

while ((i <= j) && (i<=desno) && (j>lijevo)) {
    while ((A[i] < A[lijevo]) && (i<=desno)) i++;
    while ((A[j] > A[lijevo]) && (j>lijevo)) j--;
    if (i<j) {
        Zamijeni (&A [i], &A [j]);
    }
}
if (i > desno) { // stožer je najveći u polju
    Zamijeni (&A [lijevo], &A [desno]);
    Qsort2(A, lijevo, desno-1);
}
else if (j<=lijevo) { // stožer je najmanji u polju
    Qsort2(A, lijevo+1, desno);
}
else { // stožer je negdje u sredini
    Zamijeni (&A [lijevo], &A [j]);
    Qsort2(A, lijevo, j-1);
    Qsort2(A, j+1, desno);
}
}
```

Zadaci sa sortovima

Zadatak 1.

Zadano je polje brojeva s elementima: 7, 1, 3, 8, 4, 10, 5, 2, 9, 6. Ilustrirajte sortiranje zadanog niza brojeva algoritmom **shellsort** za niz koraka {3, 2, 1}.

```

7 1 3 8 4 10 5 2 9 6      k=3
5 1 3 7 4 10 8 2 9 6
5 1 3 7 2 10 8 4 9 6
5 1 3 7 2 9 8 4 10 6
5 1 3 6 2 9 7 4 10 8      k=2
5 1 3 6 2 9 7 4 10 8
3 1 5 6 2 9 7 4 10 8
2 1 3 6 5 9 7 4 10 8
2 1 3 4 5 6 7 9 10 8
2 1 3 4 5 6 7 8 10 9      k=1
1 2 3 4 5 6 7 8 10 9
1 2 3 4 5 6 7 8 9 10      gotovo
  
```

4. Ilustrirati sortiranje zadanog niza:

30, 60, 50, 40, 70, 80, 20, 90

postupkom **quicksort** u kojem se stožerni element određuje metodom aproksimacije medijana korištenjem 3 člana niza.

U svakom koraku je potrebno označiti stožer i elemente koji se uspoređuju. Na svakoj razini rekurzije označiti stožer te prikazati sadržaj polja prije i poslije prebacivanja elemenata u odnosu na stožer.

```

30    60    50    40    70    80    20    90
// stožer je 40, a dobije se metodom medijana između 30, 40 i 90
30    60    50    40    70    80    20    90
  
```

// skrivanje stožera ili zamjena stožera s predzadnjim (zamjena 40 i 20)

```

30    60    50    40    70    80    20    90
  
```

// traže se elementi koji će se zamijeniti (to će biti 60 i 20)

```

30    60    50    20    70    80    40    90
  
```

// nakon zamjene 60 i 20, treba obnoviti stožer → mijenjaju se 50 i 40

```

30    20    50    60    70    80    40    90
  
```

// podjela u potpolja

```

30    20    40    60    70    80    50    90
  
```

// potpolja koja imaju manje od 3 elementa se sortiraju bez daljnjeg dijeljenja → sortira

// se lijevo potpolje

```

20    30    40    60    70    80    50    90
  
```

// sortira se desno potpolje → traži se medijan između 60, 80 i 90. Stožer je 80.

```

20    30    40    60    70    80    50    90
  
```

// skrivanje stožera ili zamjena stožera s predzadnjim (zamjena 80 i 50)

20 30 40 60 70 **80** **50** 90

// skrivanje stožera ili zamjena stožera s predzadnjim (zamjena 80 i 50)

20 30 40 60 70 50 **80** 90

// nema nikakvih zamjena, pa ide podjela u potpolja

20 30 40 60 70 50 **80** 90

// sortiraju se potpolja

20 30 40 50 60 70 80 90

// sortiraju se potpolja

20 30 40 50 60 70 80 90