Zadaci za vježbu za 2. međuispit

Rekurzija

Zadatak 1.

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

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

Funkcija ima prototip:

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

Zadatak 2.

Napisati **rekurzivnu** funkciju koja za zadano središte kruga (x i y koordinate), radijus i nivo crta krugove kao na slikama.

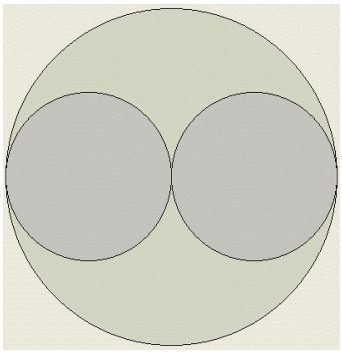
Prvi nivo sastoji se od jednog kruga, a svaki sljedeći od dva puta više krugova nego prethodni. U svaki krug K i-tog nivoa ucrtana su po dva kruga čiji su radijusi dva puta manji, a x koordinate središta pomaknute za pola radijusa lijevo, odnosno desno od središta kruga K. Iscrtavanje se zaustavlja kada su nacrtani svi krugovi do zadanog nivoa.

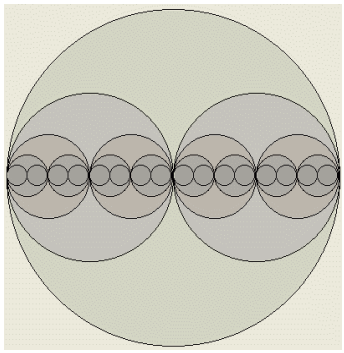
Za crtanje koristite funkciju DrawCircle koja iscrtava krug sa središtem u (x, y) i radijusom radius, koja ima prototip:

```
DrawCircle(int x, int y, int radius);
```

Funkciju DrawCircle **ne treba implementirati**!

Na slici su dani primjeri – na prvoj slici zadan je nivo 2, a na drugoj nivo 5.





Funkcija koju treba napisati ima prototip:

```
void NacrtajKrugove(int x, int y, int radijus, int nivo);
```

Zadatak 3.

Napisati **rekurzivnu** funkciju koja za zadano polje realnih brojeva (double) polje i zadanu vrijednost x, računa vrijednost polinoma u točki x:

$$\sum_{i=0}^{n} polje[i] \cdot x^i$$

Zadatak 4.

Napisati **rekurzivnu** funkciju za binarno pretraživanje polja cijelih brojeva.

Funkcija treba imati prototip:

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

Algoritmi sortiranja

Zadatak 1.
Zadan je niz brojeva: **5**, **4**, **9**, **1**, **7**, **10**, **8**, **2**, **3**, **6**. Ilustrirati sortiranje zadanog niza brojeva postupkom ***quicksort*** tako da se kao stože odabire:

a) prvi element u polju
b) zadnji element u polju

Obavezno ispisati sadržaj polja nakon svake zamjene dvaju elemenata. Nije dopušteno koristiti drugi algoritam za polja manja od neke zadane veličine (*cutoff*, npr. 3 ili 5).

Zadatak 2.
Ilustrirati korake **uzlaznog quicksorta** za podatke: **9, 10, 8, 6, 1, 4, 3, 5, 2, 7**. Stožer odabrati metodom **medijana**. Ispisati izgled polja nakon svake zamjene dvaju elemenata polja.

Zadatak 3.
Zadano je polje brojeva s elementima: **6, 1, 10, 3, 2, 8, 4, 7, 9, 5**. Ilustrirati (ispisati polje nakon svake promjene) sortiranje zadanog niza brojeva algoritmom **shellsort** za niz koraka

a) {4, 3, 1}
b) {3, 2, 1}

Zadatak 4.
Napisati funkciju koja će zadano polje cijelih brojeva (**int**) sortirati algoritmom **bubble sort** (bilo kojom varijantom algoritma). Funkcija treba imati prototip:

```
void bubble(int *polje, int N);
```

Zadatak 5.
Napisati funkciju koja će od dva zadana **sortirana** niza cijelih brojeva, napraviti treći **sortirani** niz koji sadrži sve elemente iz prva dva niza. Dva zadana niza moraju ostati nepromijenjena. Memoriju za treći niz potrebno je zauzeti u funkciji i preko imena (**return**) vratiti u glavni program. Funkcija treba imati prototip:

```
int *merge(int *niz1, int N1, int *niz2, int N2, int *NMerge);
```

Preko argumenta `NMerge`, u glavni program treba vratiti broj elemenata trećeg niza (N1 + N2).

*Primjer: Neka je prvi niz: **1**, **3**, **5**, **6**, a drugi niz: **2**, **4**, **5**, **7**. Treći niz mora biti: **1**, **2**, **3**, **4**, **5**, **5**, **6**, 7 i preko varijable `NMerge` treba vratiti vrijednost 8.*

Redovi i stogovi

Zajednički dio za zadatke sa stogom i redom:

Neka je zadan tip podatka `Stog` za kojeg su definirane funkcije za inicijalizaciju stoga, dodavanje elementa na stog te za brisanje elementa sa stoga. Prototipovi navedenih funkcija su:

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

Neka je zadan tip podatka `Red` za kojeg 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 dodajR(int element, Red *red);` [1]
- `int skiniR(int *element, Stog *red);` [1]

Funkcije za dodavanje i skidanje vraćaju 1 ako je operacija dodavanja, odnosno skidanja uspjela, a 0 inače.

Zadatak 1.
Napišite funkciju, koristeći gore navedene funkcije, koja stvara novi red koji sadrži samo negativne elemente iz zadanog reda. Funkcija ima protopip:

```
Red RedNegativnih(Red *red);
```

Početni red mora ostati očuvan. Uputa: koristiti pomoćni red.

Zadatak 2.
Napišite **rekurzivnu** funkciju, koristeći gore navedene funkcije, koja prebacuje elemente sa stoga u red, protopipa:

```
void Prebaci (Stog *stog, Red *red);
```

Početni stog mora ostati očuvan, a redoslijed elemenata u redu mora biti isti kao i na stogu (onaj element koji prvi izlazi sa stoga treba ujedno biti prvi element koji izlazi iz reda). Možete pretpostaviti da je prije poziva funkcije red prazan i inicijaliziran.

Zadatak 3:
Napisati funkciju koja će napraviti i vratiti duplikat zadanog stoga (isti takav stog, isti elementi koji imaju isti redoslijed). Zadani stog mora na kraju ostati nepromijenjen. Funkcija mora imati prototip:

```
Stog duplikat(Stog *zadani);
```

[1] Ako zadatak sadrži samo red (a ne i stog), onda bi bilo razumno funkcije za dodavanje i skidanje elemenata iz reda nazvati dodaj i skini (bez sufiksa R na kraju, što je i učinjeno u rješenju zadatka 1), međutim, ako je u zadatku ujedno definiran i Stog, za kojeg su već definirane funkcije istog imena, u C-u nije moguće definirati nove funkcije istog imena (iako imaju različite argumenta).

Zadatak 4:
Napisati funkciju koja će iz stoga izbaciti sve parne brojeve. Međusobni redoslijed neparnih brojeva mora ostati nepromijenjen. Funkcija vraća broj izbačenih brojeva te ima sljedeći prototip:

```
int izbaci_parne(Stog *stog);
```

Zadatak 5:
Napisati funkciju koja će stvoriti novi stog koji će sadržavati sve one brojeve iz zadanog stoga za koje funkcija čiji je prototip `int provjeri(int n)` vrati istinu. Poredak brojeva u novom stogu nije bitan. Zadani stog mora na kraju ostati nepromijenjen. Funkcija mora imati prototip:

```
Stog kopiraj_provjerene(Stog *zadani);
```

Zadatak 6*:
Napisati funkciju koja će na osnovu ulaznih stogova `stog1` i `stog2` stvoriti novi stog koristeći pritom sljedeće pravilo da se za novi element u novom stogu uvijek odabere manji od elemenata na vrhu stogova `stog1` i `stog2`. Ukoliko se jedan od stogova isprazni, onda uzeti preostale elemente iz drugog.
Ulazni stogovi moraju nakon završetka funkcije ostati nepromijenjeni. Funkcija mora imati prototip:

```
Stog spoji(Stog *stog1, Stog *stog2);
```

Zadatak 7*:
Napisati funkciju koja će iz stoga izbaciti sve one brojeve koji se pojavljuju dva ili više puta. Funkcija ima sljedeći prototip:

```
void izbaci_duple(Stog *stog);
```

Zadatak 8:
Napisati funkciju koja će iz stoga izbaciti sva pojavljivanja broja n i vratiti novi stog koji će sadržavati brojeve koji predstavljaju koliko je izbačeni broj bio udaljen od originalnog vrha stoga. Funkcija ima sljedeći prototip:

```
Stog izbaci_broj(Stog *stog, int n);
```

Primjer: Ukoliko je ulazni stog bio (*vrh je naveden krajnje desno*):

```
8 1 4 9 9 2 5 9 6 7 9
```

i ukoliko je funkcija pozvana s `izbaci_broj(stog, 9)`
nakon završetka funkcije, početni stog će biti

```
8 1 4 2 5 6 7
```

a novi stog će imati vrijednosti `0 3 6 7` (može i `7 6 3 0`)

Zadatak 9:
Definirana je struktura:

```
typedef struct{
    int vrijednost, prioritet;
  }Element;
```

i pripadne funkcije za skidanje i dodavanje elemenata zadane strukture u red.
Napisati funkciju koja izbacuje iz reda elemente najvećeg prioriteta (prioritet najmanje vrijednosti)

Zadatak 10:
Napisati funkciju koja će rekurzivno prebrojiti (i vratiti) elemente reda.

RJEŠENJA

Rekurzija

Zadatak 1.

```
double IzracunajPI_Rek(int n)
{
      if (n == 0) return 4;
      else
         return pow(-1., (double)n)*(4./(2*n+1)) + IzracunajPI_Rek(n-1);
}
```

Zadatak 2.

```
void NacrtajKrugove(int x, int y, int radijus, int nivo)
{
      if(nivo >= 1) {
            DrawCircle(x, y, radijus);
            nivo = nivo - 1;
            NacrtajKrugove(x - radijus/2, y, radijus/2, nivo);
            NacrtajKrugove(x + radijus/2, y, radijus/2, nivo);
      }
}
```

Zadatak 3.

```
double Polinom(double *polje, int n, double x)
{
      if (n == 0) return polje[0];
      return polje[n]*pow(x,(double)n) + Polinom(polje, n-1, x);
}
```

Zadatak 4.

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

Algoritmi sortiranja

Zadatak 1

a) **5** – stožer
5 – element koji se zamjenjuje

5 4 **9** 1 7 10 8 2 **3** 6
5 4 3 1 **7** 10 8 **2** 9 6
5 4 3 1 **2** **10** 8 7 9 6
2 **4** 3 **1** 5 10 8 7 9 6
2 **1** **3** 4 5 10 8 7 9 6
1 2 3 4 5 **10** 8 7 9 **6**
1 2 3 4 5 6 **8** **7** **9** 10
1 2 3 4 5 6 7 8 9 10

b) **5** – stožer
5 – element koji se zamjenjuje

5 4 **9** 1 7 10 8 2 **3** **6**
5 4 3 1 **7** 10 8 **2** 9 **6**
5 4 3 1 **2** **10** 8 7 9 **6**
5 4 3 **1** **2** 6 8 7 9 10
1 **4** 3 5 **2** 6 8 7 9 10
1 2 **3** **5** **4** 6 8 7 9 10
1 2 3 4 5 **6** **8** **7** 9 10
1 2 3 4 5 6 7 8 9 10

Zadatak 2.

5 – element kandidat za stožer
5 – stožer
5 – element koji se zamjenjuje

9 10 8 6 **1** 4 3 5 2 **7**
1 10 8 6 **7** 4 3 5 2 **9**
1 **10** 8 6 2 4 3 **5** **7** 9
1 5 **8** 6 2 4 **3** 10 **7** 9
1 5 3 6 2 **4** **8** 10 **7** 9
1 5 **3** 6 2 **4** 7 10 8 9
1 **5** **2** 6 **3** 4 7 10 8 9
1 **2** **5** 6 **3** 4 7 10 8 9
1 2 3 **6** **5** **4** 7 10 8 9
1 2 3 4 5 6 7 **10** **8** **9**
1 2 3 4 5 6 7 8 9 10

Zadatak 3.

a) **6** – element koji se zamjenjuje

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

b) **6** – element koji se zamjenjuje

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

Zadatak 4.

```
void bubble(int *polje, int N){
	int i, j;
	int pom;
	for (i = 0; i < N-1; i++) {
		for (j = 0; j < N-1-i; j++) {
			if (polje[j+1] < polje[j]){
				pom = polje[j];
				polje[j] = polje[j+1];
				polje[j+1] = pom;
			}
		}
	}
}
```

Zadatak 5.

```
int *merge(int *niz1, int N1, int *niz2, int N2, int *NMerge) {
	int i,j,k;
	int *niz3;

	i = j = k = 0;
	*NMerge = N1 + N2;

	niz3 = (int *)malloc(*NMerge * sizeof(int));
	if (niz3 == NULL) return NULL;

	while (i<N1 && j<N2) {
		if (niz1[i] < niz2[j]) {
			niz3[k] = niz1[i];
			i++;
			k++;
		}
		else {
			niz3[k] = niz2[j];
			j++;
			k++;
		}
	}
	while (i<N1) {
		niz3[k] = niz1[i];
		i++;
		k++;
	}
	while (j<N2) {
		niz3[k] = niz2[j];
		j++;
		k++;

	}

	return N1+N2;
}
```

Redovi i stogovi

Zadatak 1.

```
Red RedNegativnih(Red *red)
{
	Red pom, novi;
	int element;
	init_red(&pom); init_red(&novi);

	while(skini(&element, red))
	{
		if (element < 0)
			dodaj(element, &novi);
		dodaj(element, &pom);
	}
	while(skini(&element, &pom))
		dodaj(element, red);

	return novi;
}
```

Zadatak 2.

```
void Prebaci (Stog *stog, Red *red)
{
	int element;
	if (skini(&element, stog))
	{
		dodajR(element, red);
		Prebaci(stog, red);
		dodaj(element, stog);
	}
}
```

Zadatak 3:

```
Stog duplikat(Stog *zadani){
	Stog novi, pomocni; int element;
	init_stog(&novi); init_stog(&pomocni);
	while(skini(&element, zadani)){
		dodaj(element, &pomocni);
	}
	while(skini(&element, &pomocni)){
		dodaj(element, zadani);
		dodaj(element, &novi);
	}
	return novi;
}
```

Zadatak 4:

```
int izbaci_parne(Stog *stog){
	Stog pomocni;int element, br=0;
```

```
	init_stog(&pomocni);
	while(skini(&element, stog)){
		if (element % 2)
			dodaj(element, &pomocni);
		else
			br++;
	}
	while(skini(&element, &pomocni)){
		dodaj(element, stog);
	}
	return br;
}
```

Zadatak 5:

```
Stog kopiraj_provjerene(Stog *zadani){
	Stog pomocni, novi; int element;
	init_stog(&novi); init_stog(&pomocni);
	while(skini(&element, zadani)){
		dodaj(element, &pomocni);
	}
	while(skini(&element, &pomocni)){
		dodaj(element, zadani);
		if (provjeri(element))
			dodaj(element, &novi);
	}
	return novi;
}
```

Zadatak 6*:

```
Stog spoji(Stog *stog1, Stog *stog2){
	int el1, el2;
	Stog pom1, pom2, novi;
	init_stog(&novi); init_stog(&pom1); init_stog(&pom2);
	while(1){
		if (!skini(&el1, stog1)){
			break;
		}
		if (!skini(&el2, stog2)){
			dodaj(el1, &novi);
			dodaj(el1, &pom1);
			break;
		}
		if (el1 < el2){
			dodaj(el1, &novi);
			dodaj(el1, &pom1);
			/*vrati većeg(neiskorištenog) na njegov stog,
			  tako da opet bude na vrhu */
			dodaj(el2, stog2);
		}
		else{
			dodaj(el2, &novi);
			dodaj(el2, &pom2);
			dodaj(el1, stog1);
```

```
            }
        }
        while(skini(&el1, stog1)){
            dodaj(el1, &novi);
            dodaj(el1, &pom1);
        }
        while(skini(&el2, stog2)){
            dodaj(el2, &novi);
            dodaj(el2, &pom2);
        }
        while(skini(&el1, &pom1)){
            dodaj(el1, stog1);
        }
        while(skini(&el2, &pom2)){
            dodaj(el2, stog2);
        }
        return novi;
}
```

Zadatak 7*:

Ideja: Uzeti prvi element na stogu i spremiti ga u x. Zatim ga uspoređivati sa preostalim elementima u stogu. Ukoliko pojedini element nije jednak x sačuvati ga privremeno u pomoćnom stogu `pom1`*. Ukoliko je taj element jednak x, onda označiti da x ima duplikate i taj broj odbaciti (ne čuvati ga u pomoćnom stogu). Nakon prolaska kroz sve elemente stoga, vrijednost duplikat će označavati da li x ima duplikata ili ne. Ako nije imao duplikata, onda x sačuvati premještanje u pomoćni stog* `pom2`*. Sve elemente iz* `pom1` *vratiti u originalni stog i ponoviti postupak. Nakon što postupak bude gotov, tj. na originalnom stogu više ne bude nikakvih elemenata, na stogu* `pom2` *će se nalaziti samo oni elementi koji nisu imali duplikata u stogu te ih treba vratiti na originalni stog.*

```
void izbaci_duple(Stog *stog){
        Stog pom1, pom2;int duplikat=1, x, element;
        init_stog(&pom1); init_stog(&pom2);
        while(skini(&x, stog)){
                duplikat = 0;
                while(skini(&element, stog)){
                        if (x == element)
                                duplikat = 1;
                        else
                                dodaj(element, &pom1);
                }
                if (!duplikat)
                        dodaj (x, &pom2);
                while(skini(&element, &pom1)){
                        dodaj(element, stog);
                }
        }
        while(skini(&element, &pom2)){
                dodaj(element, stog);
        }
}
```

Zadatak 8:

```
Stog izbaci_broj(Stog *stog, int n){
	Stog pom, novi;int element, udaljenost=0;
	init_stog(&pom); init_stog(&novi);
	while(skini(&element, stog)){
		if (element == n)
			dodaj(udaljenost, &novi);
		else
			dodaj(element, &pom);
		udaljenost++;
	}
	while(skini(&element, &pom)){
		dodaj(element, stog);
	}
	return novi;
}
```

Zadatak 9:

```
void Izbaci(Red *red)
{
	Red pomocni;
	Element element;
	int najveciPrioritet;
	init_Red(&pomocni);

	skini(&element, red);
	najveciPrioritet = element.prioritet;
	dodaj(element, &pomocni);

	while (skini(&element, red))
	{
		if(element.prioritet < najveciPrioritet)
			najveciPrioritet = element.prioritet;
		dodaj(element, &pomocni);
	}

	while (skini(&element, &pomocni))
	{
		if (element.prioritet != najveciPrioritet)
			dodaj(element, red);
	}
}
```

Zadatak 10:

a) Bez očuvanja reda

```
int PrebrojiRekurzivno(Red *red)
{
  tip element;

if (skini(&element, red))
{
```

```
        return 1 + PrebrojiRekurzivno(red);
    }
    return 0;
    }
```

b) S očuvanjem reda

```
int PrebrojiRekurzivno(Red *red)
{
      Red pomocni;
      init_Red(&pomocni);

      return PrebrojiRekurzivnoPomocna(red, &pomocni);

}

int PrebrojiRekurzivnoPomocna(Red *red, Red *pomocni)
{
      tip element;
      int brojElemenata;

      if (skini(&element, red))
      {
            dodaj(element, pomocni);
            brojElemenata = 1 + PrebrojiRekurzivnoPomocna(red,
      pomocni);
            skini(&element, pomocni);
            dodaj(element, red);
            return brojElemenata;
      }
      return 0;
}
```