STOG

ZI 2016

Zadatak 5. (8 bodova)

Za tip podatka `Stog` definirane su funkcije za inicijalizaciju stoga, dodavanje elementa na stog i skidanje elementa sa stoga:

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

Napisati funkciju

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

koja će izbaciti sa stoga sve vrijednosti koje se na stogu pojavljuju više od jedanput, pri čemu će u slučaju višestrukog pojavljivanja vrijednosti biti zadržana vrijednost koja se pojavljuje prvi put promatrano od vrha stoga.

Primjer:

Stog prije ulaska u fukciju:

1
1
4
3
2
7
2
3
1

Stog nakon izlaska iz fukcije:

1
4
3
2
7

Napomena: Rješenje mora koristiti isključivo funkcije za rad sa stogom i pomoćne stogove. Sva rješenja koja podrazumijevaju implementacijske detalje stoga (realizacija poljem ili listom) neće biti priznata.

ZI 2015

Zadatak 1. (8 bodova)

Za tip podatka `Stog` definirane su funkcije za inicijalizaciju stoga, dodavanje elementa na stog te skidanje elementa sa stoga. Elementi stoga su realni brojevi. Prototipovi navedenih funkcija su:

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

Funkcije `dodaj` i `skini` vraćaju 1, ako je operacija dodavanja ili skidanja uspjela, a 0 inače. Potrebno je napisati funkciju `otpadnici` čiji je prototip:

```
Stog otpadnici(Stog *S, double d);
```

Funkcija `otpadnici` iz stoga `S` uklanja sve elemente koji su od aritmetičke sredine elemenata stoga `S` udaljeni barem za d > 0 (ne treba provjeravati je li d > 0). Uklonjeni elementi se stavljaju na stog `T` i to tako da su prvo, ako postoje, stavljeni oni elementi koji su manji od aritmetičke sredine (poredak nije bitan), a onda, ako postoje, oni elementi koji su veći od aritmetičke sredine (poredak nije bitan). Na stogu `S`, poredak elemenata koji nisu uklonjeni treba ostati isti.

Funkcija treba biti neovisna o implementaciji tipa podataka `Stog`, a dozvoljeno je korištenje pomoćnih stogova.

Primjer:

Zadan je stog `S`: (dno stoga) 4 < 5 < 6 < 4 < 1 (vrh stoga). Aritmetička sredina elemenata na stogu je 4.

Ako je funkcija pozvana za d = 2, na izlazu iz funkcije stog `S` treba biti 4 < 5 < 4, a stog `T` 1 < 6. Ako je funkcija pozvana za d = 4, na izlazu iz funkcije, stog `S` se ne mijenja, a stog `T` je prazan.

ZI 2014

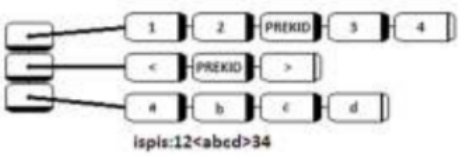
Zadatak 2. (6 bodova)

Definiran je tip podataka (varijabli) `Red` koji odgovara strukturi red čiji elementi su riječi (nizovi znakova). Za dohvat sljedećeg elementa reda koristi se funkcija:

```
char *redSkini(Red *red);
```

Također je definirana i struktura `Stog` koja se koristi za LIFO pohranu spomenutih redova, a za rad sa stogom na raspolaganju su funkcije:

```
void stogInit(Stog *stog);
void stogDodaj(Stog *stog, Red *element);
Red *stogSkini(Stog *stog);
```



ispis:12<abcd>34

Funkcija za dohvaćanje (skidanje) sa stoga vraća NULL ako je stog prazan.

Važno! Redovi riječi imaju strogo propisanu strukturu kao na slici, dakle svaki red osim posljednjeg (prvog stavljenog na stog) sadrži točno jednu riječ „PREKID“, dok posljednji red ne sadrži tu riječ. Također, može se pretpostaviti da stog nije prazan, tj. barem je jedan red riječi na stogu.

Potrebno je napisati funkciju `ispis` prototipa

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

koja ispisuje sve riječi svih redova na stogu na standardni izlaz tako da se trenutačni red ispisuje riječ po riječ sve do riječi „PREKID“. Riječ „PREKID“ se ne ispisuje, nego se nastavlja ispisivanjem sadržaja sljedećeg reda na stogu. Nakon što je sljedeći red na stogu ispisan, funkcija nastavlja ispisivati ostatak sadržaja prethodnog reda (čiji je ispis bio prekinut). Za stog i redove u primjeru na slici, ispis treba biti `12<abcd>34`.

Nakon povratka iz funkcije `ispis`, stog mora ostati sačuvan (iako redovi čije su adrese na stogu ne moraju).

ZI 2013

Zadatak 5. (6 bodova)

Zadan je tip podatka **Stog** za koji su definirane funkcije za inicijalizaciju stoga, dodavanje elementa na stog i brisanje elementa sa stoga. Elementi stoga su podaci **tipa Student** koji sadrže prezime studenta (40 znakova), ime studenta (30 znakova) te broj bodova ostvarenih na predmetu ASP. Prototipovi navedenih funkcija su:

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

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

a) **(1 bod)** Napišite **definicije** odgovarajućih struktura (tipovi podataka **Student, Stog** i **atom**) za stog realiziran vezanom listom.

b) **(5 bodova)** Napišite funkciju koja će sa stoga ukloniti sve studente koji imaju broj bodova manji od prosjeka bodova svih studenata na stogu. Redoslijed studenata koji su ostavljeni na stogu mora biti isti kao i prije poziva funkcije. Funkcija vraća broj studenata uklonjenih sa stoga. Prototip funkcije treba biti:

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

Napomene: Rješenja koja neće koristiti zadane funkcije za rad s stogom donijet će 0 bodova. Možete koristiti pomoćne stogove unutar funkcije **MakniIspodProsjeka**.

ZI 2012

Zadatak 4. (8 bodova)

Za tip podatka `Stog` koji je realiziran jednostruko povezanom listom definirane su funkcije za inicijalizaciju stoga, dodavanje elementa na stog te skidanje elementa sa stoga. Zadana su dva stoga `stog1` i `stog2`, a elementi stogova su podaci tipa `Osoba` koji sadrže informaciju o šifri osobe (cijeli broj) i visini (cijeli broj). Stog `stog1` sadrži podatke samo za muškarce, a `stog2` samo za žene i na oba stoga su podaci sortirani po visini tako da je najviša osoba na vrhu stoga. Prototipovi navedenih funkcija su:

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

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

a)*(1 bod)* Napišite **definicije** odgovarajućih struktura (tipovi podataka **`Osoba`**, **`Stog`** i **`atom`**) za stog realiziran jednostruko povezanom listom.

b)*(7 bodova)* Napišite **rekurzivnu** funkciju **`spoji`** koja će sa stoga `stog2` skinuti sve podatke i staviti ih na stog `stog1`, ali tako da ostane očuvan poredak po visini. Poredak osoba s istom visinom nije bitan.

Primjer:

	stog1		**stog2**		**stog1 (nakon spoji)**
vrh stoga->	345, 202	vrh stoga->	348, 203	vrh stoga->	348, 203
	161, 200		234, 200		345, 202
	654, 197		111, 189		234, 200
	122, 182		981, 169		161, 200
	234, 170				654, 197
					111, 189
					122, 182
					234, 170
					981, 169

LJIR 2017

Zadatak 4. (15 bodova)

Zadani su tipovi podatka **`Red`** i **`Stog`** za koje su definirane funkcije za inicijalizaciju, te za dodavanje i skidanje elementa podatkovnog tipa `char`. Prototipovi navedenih funkcija su:

```
void init_red(Red *red);
int dodajURed(char element, Red *red);
int skiniIzReda(char *element, Red *red);
```

```
void init_stog(Stog *stog);
int dodajNaStog(char element, Stog *stog);
int skiniSaStoga(char *element, Stog *stog);
```

Funkcije **`dodajURed`**, **`dodajNaStog`**, **`skiniIzReda`** i **`skiniSaStoga`** vraćaju 1 ako je operacija uspjela, a 0 ako nije.

Napišite funkciju koja će sve znamenke ('0'-'9') iz zadanog reda izdvojiti na stog na način da se prva znamenka koja se pojavljuje u redu nalazi na vrhu stoga, a zadnja na dnu. U redu ostaju svi ostali znakovi u nepromijenjenom redoslijedu. Prototip funkcije je:

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

Napišite i dio glavnog programa u kojem ćete definirati red i stog i ispravno pozvati napisanu funkciju. Dio koda u kojem se puni red nije potrebno pisati.

Primjer: Za ulazni red sadržaja **1, a, b, 2, c, 3, 4, d** poziv funkcije **izdvojiZnamenke** puni stog sadržajem (gledano od dna prema vrhu) **4, 3, 2, 1** dok u starom redu ostaju **a, b, c, d**.

JIR 2017

Zadatak 2. (15 bodova)

Zadan je tip podatka **Stog** za koji su definirane funkcije za inicijalizaciju, te za dodavanje i skidanje elementa podatkovnog tipa `int`. Prototipovi navedenih funkcija su:

```
void init_stog(Stog *stog);
int dodajNaStog(int element, Stog *stog);
int skiniSaStoga(int *element, Stog *stog);
```

Funkcije **dodajNaStog** i **skiniSaStoga** vraćaju 1 ako je operacija uspjela, a 0 ako nije.

Napišite rekurzivnu funkciju koja na temelju polaznog stoga tipa `Stog` stvori novi stog koji sadrži identične elemente u istom redoslijedu. Funkcija mora očuvati početni stog.

DR 2017

Zadatak 4 (17 bodova)

Zadan je tip podatka **Stog** za kojeg su definirane funkcije za inicijalizaciju, te za dodavanje i skidanje elementa podatkovnog tipa **double**. Prototipovi navedenih funkcija su:

```
void init_stog(Stog *stog);
int dodajNaStog(double element, Stog *stog);
int skiniSaStoga(double *element, Stog *stog);
```

Funkcije **dodajNaStog** i **skiniSaStoga** vraćaju 1 ako je operacija uspjela, a 0 ako nije.

Napišite funkciju **iznadProsjeka** koja će vratiti stog koji sadrži sve elemente zadanog stoga koji su veći od aritmetičke sredine elemenata stoga i to tako da njihov poredak ostane očuvan. Originalni stog treba ostati nepromijenjen. Prototip funkcije je:

```
Stog* iznadProsjeka(Stog *S);
```

Rješenje ne smije ovisiti o implementaciji tipa **Stog**. Također, rješenja koja u sebi uključuju inicijalizaciju dodatnih polja donijet će znatno manje bodova.
Primjer: Neka je zadan stog **S** s elementima (od vrha prema dnu) **3.1, 2.7, 4.5, 4.8, 4.5, 3.1, 4.8, 4.1**.
Aritmetička sredina elemenata stoga je 3.95, pa funkcija treba vratiti stog s elementima **4.5, 4.8, 4.5, 4.8, 4.1** (elementi su poredani od vrha prema dnu).

Zadatak 3. (30 bodova)

a) Potrebno je napisati rekurzivnu funkciju **hanoiStog**, koja sve poteze u igri Hanojski tornjevi pohranjuje na stog (na vrhu stoga je uvijek potez koji je odigran posljednji):

```
void hanoiStog (char izvor, char pomocni, char odrediste, int n, Stog *stog);
```

Argumenti **izvor**, **pomocni** i **odrediste** predstavljaju izvorni, pomoćni i odredišni toranj (oznake 'I', 'O' ili 'P'). Kod inicijalnog poziva funkcije stog je prazan, a argument **n** predstavlja broj diskova na tornju **izvor** (**n** ≥ 1).

Definirana je struktura **potez** i pomoćne funkcije za rad sa stogom:

```
typedef struct s {
  char izvor, odrediste; /* disk se prebacuje s izvora na odrediste */
  int disk; /* redni broj diska koji se prebacuje; najmanji disk ima redni broj 1 */
} potez;

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

Napomena: nerekurzivno rješenje se neće priznati.

b) Potrebno je napisati funkciju **ispisPoteza** koja ispisuje sve poteze u igri Hanojski tornjevi redoslijedom kojim su odigrani (prvi potez treba biti ispisan prvi). Funkcija se poziva s argumentom **stog**, čiji vrh, u trenutku poziva funkcije, pokazuje na potez koji je odigran posljednji. Prototip funkcije je:

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

Primjer: za n = 2 funkcija `ispisPoteza` treba ispisati:

```
Prebacujem element 1 s tornja I na toranj P
Prebacujem element 2 s tornja I na toranj O
Prebacujem element 1 s tornja P na toranj O
```

c) Odredite vremena izvođenja u O, Ω i Θ notaciji za funkcije **hanoiStog** i **ispisPoteza** iz a) i b) dijela zadatka (u ovisnosti o broju diskova **n**):

Funkcija **hanoiStog**: O _____ Ω _____ Θ_____

Funkcija **ispisPoteza**: O _____ Ω _____ Θ_____

RED

ZI 2017

Zadatak 4. (10 bodova)

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

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

Funkcije **dodaj** i **skini** vraćaju 1 ako je operacija uspjela, a 0 ako nije.

Napišite funkciju koja će sva pojavljivanja višekratnika zadanog broja iz zadanog reda izdvojiti u novi red koji treba vratiti pozivatelju. Prototip funkcije je:

```
Red *izdvojiVisekratnike(Red *red, int broj);
```

U ulaznom redu moraju ostati svi preostali elementi.

Napišite i dio glavnog programa u kojem ćete definirati redove i ispravno pozvati napisanu funkciju. Dio koda u kojem se puni red nije potrebno pisati.

Primjer: Za ulazni red sadržaja **3, 8, 2, 5, 35, 2, 24, 15** poziv funkcije **izdvojiVisekratnike** za **broj=3** stvara novi red sadržaja **3, 24, 15** dok u starom redu ostaju **8, 2, 5, 35, 2**.

ZI 2013

Zadatak 1. (6 bodova)

Za tip podatka Red definirane su funkcije (nije ih potrebno pisati) za inicijalizaciju reda, dodavanje elementa u red te za skidanje elementa iz reda. Prototipovi navedenih funkcija su:

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

Funkcije **dodaj** i **skini** vraćaju 1 ako je operacija dodavanja ili skidanja uspjela, a 0 inače. Elementi reda su podaci tipa **Element**. Tipovi **Element** i **Red** definirani su redom sljedećim programskim odsječkom:

```
#define MAXNIZ 20
typedef struct Element{
    char niz[MAXNIZ + 1];
} Element;
```

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

```
typedef struct {
      atom *ulaz;
      atom *izlaz;
} Red;
```

Napišite funkciju koja će iz zadanog reda u novi red kopirati sve elemente čija varijabla `niz` sadrži podniz koji funkcija primi kao ulazni argument. Funkcija pozivatelju vraća taj novi red, a prototip joj je:

```
Red* trazi(Red *red, char* podniz)
```

Poredak elemenata u novom redu mora biti isti kao i poredak u ulaznom redu. Po završetku izvođenja funkcije, ulazni red mora sadržavati iste podatke u istom redoslijedu kao i prije poziva funkcije `trazi`.

Ako npr. zadani red sadrži elemente sa sljedećim nizovima znakova: "234dd6", "aBc569", "1234d", "ab45de", a traženi podniz je "34d", funkcija mora vratiti novi red čiji elementi sadrže nizove "234dd6" i "1234d".

Napomene: Rješenja koja neće koristiti zadane funkcije za rad s redom donijet će 0 bodova. Možete koristiti pomoćne redove unutar funkcije **trazi**. *Skrećemo pozornost da bi funkcije* `strcpy` *i* `strstr` *iz standardne programske knjižnice C-a mogle biti od koristi.*

ZI 2012

Zadatak 1. (6 bodova)

Za red realiziran jednostruko povezanom listom napišite funkciju **dodaj** za dodavanje elementa u red.
Prototip funkcije je:

```
int dodaj(int element, atom **ulaz, atom **izlaz);
```

Funkcija treba vratiti 1 ukoliko je element uspješno dodan u red, a 0 inače.
Tip `atom` definiran je sljedećim programskim odsječkom:

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

TEHNIKE ADRESIRANJA

ZI 2017

Zadatak 3. (10 bodova)

Podatke o studentima potrebno je organizirati u memorijski rezidentnu tablicu raspršenog adresiranja veličine 10000 zapisa. Memorija za tablicu raspršenog adresiranja rezervira se u glavnom programu (nije ga potrebno pisati). Kolizija se rješava linearnim ispitivanjem.

Potrebno je napisati:

a) Pretprocesorsku direktivu kojom se definira veličina tablice te ostali potrebni parametri u strukturi. Potrebno je definirati strukturu `zapis` za pohranu podataka o studentu. Svaki student ima šifru (cijeli broj), ime (maksimalna veličina 20 znakova) i prezime (maksimalna veličina 30 znakova).
b) Funkciju koja će odrediti na koju adresu je bilo upućeno najviše zapisa. Funkcija treba vratiti tu adresu i broj zapisa upućenih na nju (lokacija u tablici), a koji su završili u preljevu. Šifra nula (`0`) označava prazan zapis. Ako ima više takvih adresa vratiti rezultat za bilo koju od njih.
   ```
   int Provjera(zapis* hash, int *brojPreljeva);
   ```

Prilikom realizacije funkcije za upis koristiti predefiniranu hash-funkciju:

```
int Adresa(int sifra);
```

ZI 2016

Zadatak 4. (8 bodova)

Podatke o djelatnicima potrebno je organizirati u memorijski rezidentnu tablicu raspršenog adresiranja veličine 5000 zapisa. Memorija za tablicu raspršenog adresiranja rezervira se u glavnom programu (nije ga potrebno pisati). Kolizija se rješava dvostrukim raspršenim adresiranjem.

Potrebno je napisati:

a) Pretprocesorsku direktivu kojom se definira veličina tablice i ostale potrebne konstante za potrebe dvostrukog raspršenog adresiranja. Potrebno je napisati i strukturu za pohranu podataka o djelatniku. Svaki djelatnik ima šifru (cijeli broj), ime (maksimalna veličina 20 znakova) i prezime (maksimalna veličina 30 znakova).
b) Funkciju za upis u tablicu raspršenog adresiranja koja pojavu kolizija rješava dvostrukim raspršenim adresiranjem. Funkcija vraća 1 ako je upis uspio, a 0 ako je hash-tablica puna. Prototip funkcije je:
`int Upis(struct zapis* hash, struct zapis element);`

Prilikom realizacije funkcije za upis koristiti predefinirane hash-funkcije:

```
int Adresa1(int sifra) {
	return sifra%M;
}

int Adresa2(int sifra) {
	return 1 + sifra%(M - 1);
}
```

JIR 2017

Zadatak 1. (18 bodova)

Svaki zapis memorijski rezidentne tablice raspršenog adresiranja sadrži podatke o jednom proizvodu:

šifru proizvoda (int), naziv proizvoda (duljine do 50 znakova) te cijenu proizvoda (double).

Šifra nula (0) označava prazan zapis. Očekuje se najviše 150000 zapisa, a kapacitet tablice je 15% veći. Prilikom upisa primjenjuje se metoda kvadratnog ispitivanja (c1=1, c2=3). Ključ zapisa je *šifra*, a pretvorba ključa u adresu se obavlja već pripremljenom funkcijom prototipa:

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

Pretpostavite da je tablica već prethodno popunjena i nalazi se u varijabli **`podatci`** u funkciji `main` (tip varijable **`podatci`** odredite sami).

Napišite funkciju koja će dinamički alocirati jednodimenzionalno polje i u njega upisati šifre i nazive proizvoda onih zapisa koji su završili u preljevu, a čija je cijena proizvoda manja ili jednaka prosječnoj cijeni proizvoda čiji su zapisi završili u preljevu. Funkcija treba vratiti pokazivač na alocirano polje i broj elemenata. Prototip funkcije je:

```
povratni_tip* fun(ulazni_tip *tab, int *brZapisa);
```

Definirajte sve potrebne strukture i tipove podataka, te napišite glavni program koji poziva vašu funkciju.

DR 2016

Zadatak 2. (14 bodova)

Podatke o zračnim lukama potrebno je organizirati u memorijski rezidentnu tablicu raspršenog adresiranja veličine 10000 mjesta. Memorija za tablicu raspršenog adresiranja rezervira se u glavnom programu (nije ga potrebno pisati). Kolizija se rješava *kvadratnim ispitivanjem*. Podrazumijeva se da je mjesto u tablici slobodno, ako je šifra upisanoga zapisa jednaka 0.

a) Potrebno je napisati pretprocesorsku naredbu kojom se definira veličina tablice i ostale potrebne konstante za potrebe kvadratnog ispitivanja ($c_1 = 0$ *i* $c_2 = 1$). Potrebno je definirati i strukturu **zapis** za pohranu podataka o zračnim lukama, koja sadrži sljedeće podatke: šifru (cijeli broj), naziv (najviše 50+1 znak) i oznaku (najviše 5+1 znak).

b) Potrebno je napisati funkciju **prebroji** koja izračunava i vraća broj zapisa u tablici raspršenog adresiranja koji se *ne nalaze* na „pravom" mjestu, a njihovo je „pravo" mjesto slobodno. Pod pojmom „pravo" mjesto podrazumijeva se mjesto u tablici na kojemu bi zapis bio smješten da nema kolizije. Napomena: slobodna mjesta mogu nastati brisanjem elemenata.

Prototip funkcije je:

```
int prebroji(zapis *hash);
```

Na raspolaganju imate unaprijed definiranu funkciju raspršenog adresiranja **adresa** čiji je prototip:

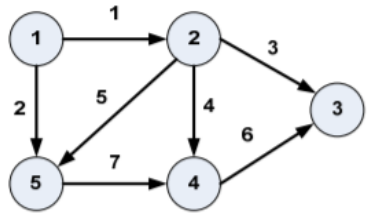
```
int adresa(int sifra);
```

GRAFOVI

ZI 2016

Zadatak 1. (6 bodova)

Za zadani graf nacrtajte matrice susjedstva i incidencije te listu susjedstva.



LJIR 2016

Zadatak 1. (10 bodova)

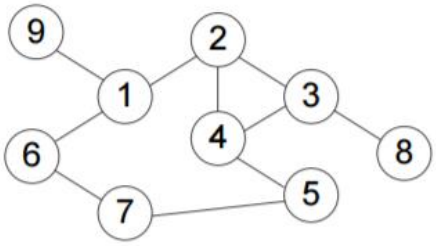
Za zadanu matricu incidencije nacrtajte usmjereni graf.

$$Inc = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & -1 \\ 0 & -1 & -1 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

LJIR 2017

Zadatak 5. (15 bodova)

Zadan je graf prikazan slikom:



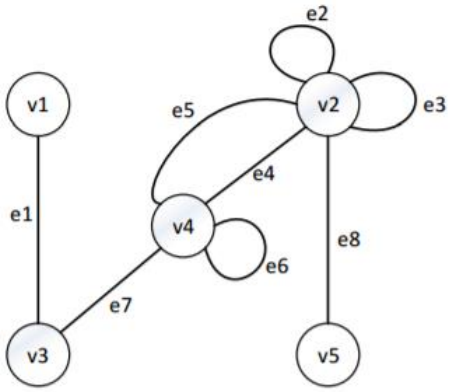
Prikažite kako izgleda ispis vrhova grafa korištenjem DFS i BFS načina obilaska grafa, uz pretpostavku da se kreće od čvora s vrijednošću 4. Prikažite korake u ispisu vrijednosti vrhova grafa, redoslijed posjećivanja vrhova te sadržaj pratećih pomoćnih struktura stog i red u svakom koraku.

Napomena: prilikom odabira neobiđenih susjednih vrhova za oba načina obilaska grafa vrhove odabirati prema **rastućoj** vrijednosti vrha.

JIR 2017

Zadatak 5. (12 bodova)

Prikažite za zadani neusmjereni graf:



a) **(4 boda)** listu susjedstva

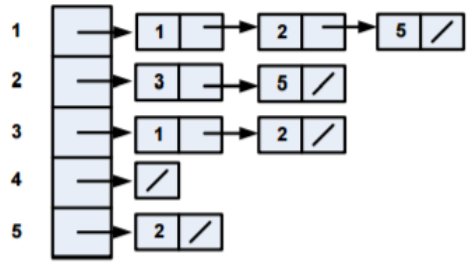
b) **(4 boda)** matricu susjedstva

c) **(4 boda)** matricu incidencije.

DR 2016

Zadatak 6. (7 bodova)

Za zadanu listu susjedstva, nacrtajte usmjereni graf:



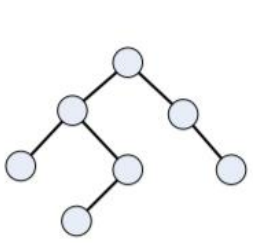
STABLA

ZI 2017

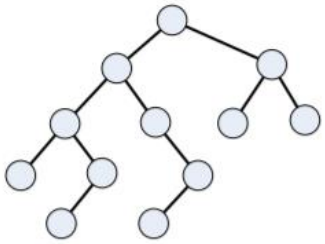
Zadatak 1. (10 bodova)

Potrebno je provjeriti je li binarno stablo balansirano s obzirom na visinu. Prazno stablo je balansirano, a neprazno binarno stablo T je balansirano ako vrijedi:

1) Lijevo i desno podstablo od T je balansirano
2) Razlika u visini lijevog i desnog podstabla nije više od 1



Stablo balansirano po visini



Stablo koje nije balansirano po visini

a) Napisati funkciju koja će izračunati visinu stabla. Ako je stablo prazno vraća 0. Prototip funkcije:

```
int visina(cvor *korijen);
```

b) Napisati funkciju koja će provjeriti da li je stablo balansirano. Ako je balansirano vraća 1, inače 0. Dopušteno je koristiti funkciju `visina`. Prototip funkcije:

```
int balans(cvor *korijen);
```

Čvorovi stabla imaju sljedeću strukturu:

```
typedef struct cv{
      int podatak;
      struct cv *lijevo_dijete;
      struct cv *desno_dijete;
} cvor;
```

ZI 2016

Zadatak 2. (10 bodova)

Potrebno je sortirati zadano polje cijelih brojeva na način da ga se prvo prepiše u binarno stablo za pretraživanje, a zatim ga se, inorder postupkom, prepiše natrag u polje. Možete pretpostaviti da već postoji funkcija

```
cvor *dodaj(int br, cvor *korijen)
```

koja zadani element dodaje u binarno stablo te vraća pokazivač na novonastalo stablo. Čvorovi stabla imaju sljedeću strukturu:

```
typedef struct cv{
      int podatak;
      struct cvor *lijevo_dijete;
      struct cvor *desno_dijete;
} cvor;
```

a) Napisati funkciju koja će elemente zadanog binarnog stabla za pretraživanje prepisati u jednodimenzionalno polje, tako da to polje bude sortirano uzlazno. Primjer prototipa funkcije:

```
void prepisi(cvor *korijen, int *i, int *polje);
```

Varijabla *i* određuje na kojem mjestu u polju počinje zapisivanje elemenata stabla. Funkcija treba osloboditi svu memoriju koju su elementi stabla zauzeli.

b) Napisati funkciju koja će sortirati zadano polje cijelih brojeva pomoću binarnog stabla za pretraživanje.

Funkcija treba imati prototip:

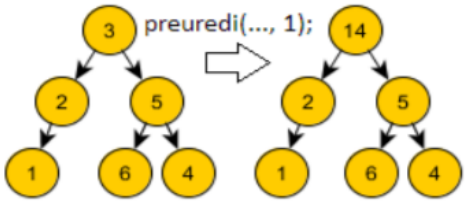
```
void mojsort(int *polje, int n);
```

ZI 2015

Zadatak 4. (6 bodova)

Čvor stabla je zadan je strukturom:

```
typedef struct cv {
      int vrijednost;
      struct cv * l, *d;
} cvor;
```



a) Napisati rekurzivnu funkciju, koja će za svaki čvor stabla na razinama čiji je paritet `paritet`, izračunati novu vrijednost kao zbroj inicijalne vrijednosti tog čvora i vrijednosti svih njegovih potomaka na razini `razina+2`. Prototip funkcije jest:

```
int preuredi (cvor* korijen, int razina, int paritet);
```

(napomena: argumentom paritet iz skupa {1,0} se funkciji zadaje uvećava li vrijednosti na neparnim ili na parnim razinama stabla; npr. ako je paritet 1, onda za zadanu razina koja je neparan broj, treba uvećati razinu korijena te svaku drugu od nje, a ako je paritet 0,onda ostale. Crtež pokazuje primjer transformacije za poziv s argumentom 1.)

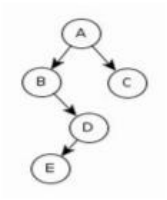
b) Napisati dio koda koji postiže da je cijelo stablo promijenjeno na način naveden u a) dijelu zadatka.

ZI 2014

Zadatak 4. (6 bodova)

Čvor stabla definiran je odsječkom:

```
struct cv {
  int vrijednost;
  struct cv *lijevo, *desno;
};
```



Zadani su prototipovi dviju funkcija:

```
int sirinaRazine(cvor* korijen, int razina);
int sirinaStabla(cvor* korijen);
```

Funkcija `sirinaRazine` treba vratiti broj čvorova na razini `razina`. Funkcija `sirinaStabla` treba vratiti najveću širinu svih razina u binarnom stablu, dakle broj čvorova na najširoj razini. Po dogovoru, korijen stabla je na razini 1.

U primjeru na slici širina razine dva je 2, što je najveća širina u ovom stablu pa time i vrijednost koju funkcija `sirinaStabla` mora vratiti.

Napišite funkcije `sirinaRazine` i `sirinaStabla`.

ZI 2013

Zadatak 4. (6 bodova)

Binarno stablo za pretraživanje organizirano je po pravilu da su brojevi u lijevom podstablu manji od brojeva u desnom. **Postorder** ispis binarnog stabla za pretraživanje je:

3, 5, 4, 1, 8, 15, 16, 11, 7, 6

a) **(2 boda) Rekonstruirajte** binarno stablo za pretraživanje (nacrtajte izgled) u prostor s desna

Izgled stabla:

b) **(4 boda)** Binarno stablo za pretraživanje, zadano postorder ispisom, ostvareno je statičkom strukturom polje, pri čemu se prvi element polja (indeks nula) ne koristi, nego je korijen stabla na indeksu 1. Napišite, u odgovarajućem redoslijedu, niz indeksa **(i_1, i_2, i_3, ... i_{10})** u polju na kojima se nalaze elementi ispisani postorder obilaskom stabla. Na primjer, i_3 je indeks na kojem se u polju nalazi 3. po redu ispisani element stabla. Odgovor napišite na crtu ispod.

_____

ZI 2012

Zadatak 2. (12 bodova)

Zadan je niz brojeva 2, 5, 7, 8, 11, 1, 4, 2, 3, 7 koji su spremljeni u čvorove binarnog stabla. Čvor stabla definiran je sljedećim programskim odsječkom:

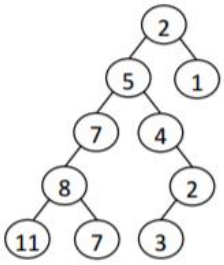
```
typedef struct cv {
  int el;
  struct cv *lijevo;
  struct cv *desno;
} cvor;
```

a)*(6 bodova)* Napišite funkciju **dodaj** koja dodaje element u stablo tako da se pozivima te funkcije za sve elemente zadanog niza brojeva stvori stablo kao na slici Slika 1. Funkcija vraća pokazivač na korijen stabla. Prototip funkcije je:

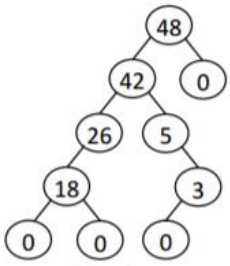
```
cvor *dodaj (cvor *korijen, int broj);
```

b)*(3 boda)* Napišite funkciju **zamijeni** koja će svaki element stabla zamijeniti sumom elemenata u njegovom lijevom i desnom podstablu (tj. sumom njegovih potomaka **prije zamjene** vrijednosti u tim potomcima). Npr. stablo sa slike Slika 1 transformirat će se u stablo na slici Slika 2.

c)*(3 boda)* Napišite glavni program koji će definirati cjelobrojno polje i inicijalizirati ga na vrijednosti iz zadanog niza brojeva. Korištenjem funkcije iz a) dijela zadatka stvoriti stablo sa slike Slika 1, te pozivom funkcije iz b) dijela zadatka transformirati ga u stablo na slici Slika 2.



Slika 1



Slika 2

DR 2017

Zadatak 1. (15 bodova)

Binarno stablo sadrži cjelobrojne elemente, a čvor je definiran strukturom

```
typedef struct s {
        int vrijednost;
        struct s *lijevo, *desno;
} cvor;
```

Napišite funkciju **jeLiUzlazan**, čiji je prototip:

```
int jeLiUzlazan(cvor *korijen);
```

kojom ćete utvrditi čine li vrijednosti čvorova binarnog stabla nakon inorder obilaska stabla uzlazni niz ili ne. Ako čine uzlazni niz, funkcija treba vratiti 1, a inače vraća 0.

Za rješenje zadatka možda će trebati koristiti funkcije **najmanji** i najveći, čiji su prototipovi

```
int najmanji(cvor *korijen);
int najveci(cvor *korijen);
```

koje vraćaju najmanju, odnosno najveću vrijednost u stablu čiji je korijen argument funkcije. Ako ćete ih koristiti, dovoljno je napisati jednu od njih, a ako riješite zadatak bez njihova korištenja, ne trebate ih pisati.

LJIR 2016

Zadatak 2. (20 bodova)

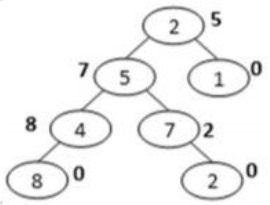
Čvorovi binarnog stabla sadrže binarne brojeve i definirani su strukturom **cvor** :

```
typedef struct cv {
  int vrijednost;
  struct cv *lijevo, *desno;
} cvor;
```

Harmanov zbroj stabla je definiran kao zbroj *cMax* (child maximal) vrijednosti svakog čvora. *cMax* vrijednost za čvor je definirana kao veća od vrijednosti lijevog i desnog djeteta. Ako čvor nema niti jedno dijete, podrazumijeva se da je njegova cMax vrijednost 0. Napisati funkciju koja za zadano stablo računa Harmanov zbroj (ulazno stablo mora ostati nepromijenjeno). Prototip funkcije je

```
int HarmanovZbroj(cvor *korijen);
```

Na slici je prikazan primjer stabla, a pored svakog čvora dana je njegova *cMax* vrijednost. Harmanov zbroj prikazanog stabla je 22.



DR 2016

Zadatak 1. (15 bodova)

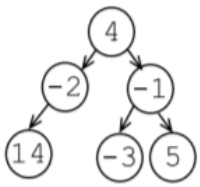
Zadano je binarno stablo čiji su čvorovi definirani strukturom:

```
typedef struct s {
      int element; // vrijednost čvora (cijeli broj različit od 0)
      struct s *lijevo, *desno;
} cvor;
```

Napišite funkciju **najList** koja vraća najveću vrijednost među listovima zadanog binarnog stabla. Funkcija vraća 0, ako je stablo prazno. Prototip funkcije je:

```
int najList(cvor *korijen);
```

Primjer: za stablo na donjoj slici, funkcija **najList** treba vratiti 14.



JIR 2016

Zadatak 1. (20 bodova)

a) Potrebno je definirati strukturu **zOsoba** koja će sadržavati šifru osobe (cijeli broj), ime osobe (30 + 1 znakova) i prezime osobe (40 + 1 znakova).

b) Binarno stablo sadrži zapise o osobama (koristi se struktura **zOsoba**), a čvorovi su definirani strukturom **cvorBS**:

```
typedef struct cvBS {
  zOsoba osoba;
  struct cvBS *lijevo, *desno;
} cvorBS;
```

Čvorovi u stablu uređeni su prema šifri osobe tako da lijevo dijete sadrži šifru koja je manja od ili jednaka šifri u čvoru-roditelju, a desno dijete sadrži šifru osobe koja je veća od šifre u čvoru-roditelju.

Potrebno je napisati funkciju **stvoriListu** koja će stvoriti jednostruko povezanu listu koja će sadržavati sve zapise o osobama iz binarnog stabla tako da zapisi u listi budu uzlazno poredani prema šifri osobe. Čvorovi u jednostruko povezanoj listi definirani su strukturom **cvorL**:

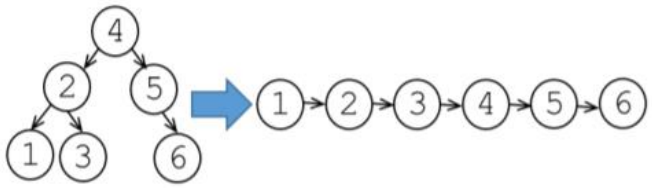
```
typedef struct cvL {
  zOsoba osoba;
  struct cvL *sljed;
} cvorL;
```

Prototip funkcije je:

```
int stvoriListu(cvorBS *korijen, cvorL **glava, cvorL **zadnji);
```

Po izlasku iz funkcije pokazivač **glava** treba pokazivati na početni čvor u listi (koji sadrži najmanju šifru), a pokazivač **zadnji** na zadnji čvor u listi. Funkcija vraća vrijednost 1, ako su svi zapisi o osobama uspješno prepisani u listu, a 0 inače.

Na slici je prikazan primjer binarnog stabla te liste koja se dobije prepisivanjem zapisa o osobama korištenjem funkcije **stvoriListu**. Zbog jednostavnosti, u čvorovima stabla i liste prikazane su samo šifre osoba.



c) Pretpostavite da binarno stablo ima **n** čvorova. Odredite vrijeme izvođenja funkcije **stvoriListu** iz b) dijela zadatka u O, Ω i Θ notaciji:

O _____ Ω _____ Θ_____

GOMILA

ZI 2017

Zadatak 2. (10 bodova)

Zadan je niz brojeva: **12, 48, 54, 22, 81, 62, 71, 32.**

a) **(5 bodova)** Ilustrirajte (nacrtajte stablo nakon svake promjene) stvaranje gomile (max heap) od zadanog polja brojeva algoritmom čija je složenost za najgori slučaj *O(n)*.

b) **(5 bodova)** Ilustrirajte (nacrtajte stablo nakon svake promjene) stvaranje gomile (max heap) od zadanog polja brojeva algoritmom čija je složenost za najgori slučaj *O(nlogn)*.

ZI 2016

Zadatak 3. (8 bodova)

Napišite funkciju koja će za zadano polje prirodnih brojeva provjeriti je li gomila. U polju su s -1 označeni elementi koji se ne koriste. Funkcija treba vratiti 1 ako je zadano polje gomila, a -1 ako nije. Funkcija treba imati prototip:

```
int jeliGomila(int *polje, int n, int vrstaGomile);
```

Pomoću varijable `vrstaGomile`, određuje se za koju vrstu gomile se obavlja ispitivanje. Ako je `vrstaGomile = 1` roditelj treba biti veći od djece, inače roditelj treba biti manji od djece. Možete pretpostaviti da gomila kreće od indeksa 1.

ZI 2015

Zadatak 5. (4 boda)

Prioritetni red `P` realiziran je gomilom s relacijom manji od (*min. heap*). Gomila je implementirana poljem cjelobrojnih elemenata. Polje može imati najviše 15 članova, a sadržaj polja na početku izvođenja je:

6	10	9	11	20	25	19	24	16						

Funkcija zadana prototipom:

```
void DodajURed( PRed *P, int e);
```

realizira ubacivanje elementa u gomilu algoritmom čije je vrijeme izvođenja $O(\log_2 n)$.

Funkcija zadana prototipom:

```
void SkiniIzReda(PRed *P, int *e);
```

vraća vršni element iz gomile te nakon toga podešava gomilu algoritmom čije je vrijeme izvođenja $O(\log_2 n)$.

Ilustrirati sadržaj prioritetnog reda `P` nakon svake zamjene dvaju elemenata, ako su dodavani i skidani elementi sljedećim nizom naredbi (za gore zadani prioritetni red P):

```
DodajURed(&P, 5);
SkiniIzReda(&P, &e);
SkiniIzReda(&P, &e);
DodajURed(&P, 21);
```

ZI 2014

Zadatak 5. (6 bodova)

Zadan je niz brojeva: **36, 58, 44, 28, 96, 62, 76, 38**.

a) Ilustrirajte (nacrtajte stablo nakon svake promjene) stvaranje gomile s relacijom ***veći od*** (max heap) od zadanog polja brojeva algoritmom čija je složenost za najgori slučaj *O(n)*.

b) Počevši od gomile iz podzadatka a), prikažite sortiranje zadanog niza *heapsortom*, jasno prikazujući svaki korak sortiranja (nacrtajte stablo nakon svakog koraka algoritma).

ZI 2013

Zadatak 3. (6 bodova)

Zadan je niz brojeva: **30, 45, 31, 60, 87, 95, 12, 69**

a) **(3 boda)** Ilustrirajte (nacrtajte stablo nakon svake promjene) stvaranje gomile od zadanog polja brojeva algoritmom čija je složenost za najgori slučaj *O(n)*.

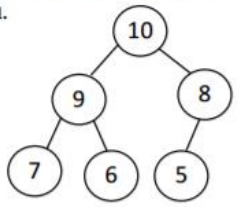
b) **(3 boda)** Ilustrirajte (nacrtajte stablo nakon svake promjene) stvaranje gomile od zadanog polja brojeva algoritmom čija je složenost za najgori slučaj *O(nlogn)*.

ZI 2012

Zadatak 3. (4 boda)

a)*(2 boda)* Zadan je **inorder** (lijevo, korijen, desno) ispis gomile u kojoj je roditelj veći od svoje djece: 75, 78, 74, 80, 11, 32, 92, 50, 84, 58. Nacrtajte gomilu.

b)*(2 boda)* Zadana je gomila kao na slici. Prikažite sve korake uzlaznog heap sorta



LJIR 2017

Zadatak 2. (14 bodova)

Zadan je niz brojeva: **3, 68, 40, 87, 29, 33, 26, 51, 28, 60.**

a) **(6 bodova)** Sortirajte niz silazno shell sortom za niz koraka {4,3,1}.

b) **(8 bodova)** Sortirajte niz silazno heapsortom. Prikažite postupak do trenutka kada ste sortirali barem 4 elementa.

Potrebno je prikazati polje nakon svake zamjene dvaju elemenata u oba dijela zadatka.

DR 2017

Zadatak 3. (13 bodova)

Zadan je niz brojeva: **11, 28, 39, 25, 78, 55, 77, 32**.

a) **(7 bodova)** Ilustrirajte (nacrtajte stablo nakon svake promjene) stvaranje gomile s relacijom ***veći od*** (max heap) od zadanog polja brojeva algoritmom čija je složenost za najgori slučaj *O(n log(n))*.

b) **(6 bodova)** Za gomilu iz a) zadatka prikažite postupak uzlaznog *heapsorta*. Prikažite svaki korak sortiranja (nacrtajte stablo nakon svake zamjene dva elementa). Potrebno je prikazati polje nakon svake zamjene dvaju elemenata u oba dijela zadatka.

DR 2016

Zadatak 3. (10 bodova)

U polju cijelih brojeva pohranjen je sljedeći niz brojeva:
1, 2, 5, 6, 8, 9, 7, 10
Polje predstavlja **gomilu** s relacijom *manji od* (*min heap*).

Potrebno je **silazno** sortirati zadano polje metodom *heapsort* uz prikazivanje svakog koraka sortiranja (potrebno je napisati sadržaj polja nakon svake promjene).