1. Definicija rekurzije
   - kako ispisujete cifre broje – ispisete prvu cifru pa onda ostatak
   - ispisete poslednju cifru pa onda ostatak
   - kako se sabiraju elementi niza – saberemo poslednji (ili prvi) element niza sa sumom ostatka niza
   - kako studentska sluzba upisuje studente u bazu – upise prvog studenta u nizu, pa onda upise ostale studente
   - IZLAZ – jednocifren broj; niz sa jednim elementom (ili niz sa 0 elemenata i tada je suma 0), niz od jednog studenta (ili prazan niz, kada studentska sluzba moze da zavrsi sa radom --- ili kraj radnog vremena :))
2. Veza izmedju rekurzije i matematicke indukcije
   - bazni slucaj i korak indukcije
   - bazni slucaj – izlaz iz matematicke indukcije – moramo imati izlaz iz rekurzije
     - uslov, izlaz iz rekurzije
   - korak indukcije – rekurzivni korak, pozivamo se na nesto sto nam je vec poznato
     - ako uslov nije ispunjen, rekurzivni poziv
     - bez uslova – dobijamo beskonacnu rekurziju
   - obrnuto od indukcije
3. Prvi primer:
   - suma svih brojeva od 1 do n
   - primer rekurzija1.c
4. Prednost rekurzije je rad sa rekurzivno definisanim strukturama podataka: liste, stabla,…
   - nizove mozemo posmatrati kao rekurzivno definisane
   - [] niz je niz
   - a-niz, x-novi element → [a, x] tj. X dopisan na kraj niza a nam daje novi prosireni niz
   - rekurzivna fja nad nizom se pise:
     - nad praznim nizom
     - nad nizom koji ima bar jedan element
     - alternativa a-niz, x-novi el → [x,a] novi niz, x dopisan na pocetak niza
5. Drugi primer:
   - rekurzija nad nizom, sabiranje elemenata niza
   - primer rekurzija2.c
6. Kako se generisu stek okviri za rekurzivnu funkciju:
   - kada se pokrene program prvo se postavlja stek okvir za fju main
     - nakon toga se pravi stek okvir za fju suma([1,2,3], 3)
     - posto 3<>0 ponovo se poziva rekurzivno i dobijamo novi stek okvir
       - suma([1,2], 2)
       - i dodatno imamo promenljivu koju smo izvukli van rekurzivnog poziva i odgovara poslednjem elementu niza
       - posto 2<>0 imamo opet rekurzivni poziv i novi stek okvir
         - suma([1], 1)
         - posto 1<>0 imamo opet rekurzivni poziv i novi stek okvir
           - suma([], 0)
           - sada je n = 0, i ovde nema vise rekurzivnih poziva
           - vec ide grana koja odgovara izlazu iz rekurzije
           - vraca 0 prethodnoj fji
         - sada skidamo tu fju sa steka posto je zavrsila, vratila je 0
         - sabiramo tu povratnu vrednost sa onom vrednoscu koju je ona sacuvala kao poslednji element niza, tj. 1
         - 1 + 0 = 1 i ova fja vraca 1 prethodnoj fji
       - sada skidamo tu fju sa steka posto je zavrsila, vratila je 1

        - sabiramo tu povratnu vrednost sa onom vrednoscu koju je ona sacuvala kao poslednji element niza, tj. 2
        - 2 + 1 = 3 i ova fja vraca 3 prethodnoj fji
    - sada skidamo tu fju sa steka posto je zavrsila i vratila je 3
    - sabiramo tu povratnu vrednost sa njenim poslednjim elementom, tj. 3
    - 3 + 3 = 6 i ova fja vraca 6 prethodnoj fji, tj. Main-u koji dobija rezultat 1 + 2 + 3 = 6

				Suma, a = [], n = 0	Vraca 0 skidamo sa steka				
			Suma, a = [1], n = 1	Suma, a = [1], n = 1	1 + 0 = 1 Vraca 1				
		Suma, a = [1,2], n = 2	Suma, a = [1,2], n = 2	Suma, a = [1,2], n = 2	Suma, a = [1,2], n = 2	2 + 1 = 3, vraca 3			
	suma, a =[1,2,3], n = 3	suma, a =[1,2,3], n = 3	suma, a =[1,2,3], n = 3	suma, a =[1,2,3], n = 3	suma, a =[1,2,3], n = 3	suma, a =[1,2,3], n = 3	3 + 3 = 6, vraca 6		
main: argc, argv	main	main	main	main	main	main	main	main	

7. Ovakav izgled steka se gradi uvek u toku rada sa rekurzivnim fjama, treba voditi racuna da rekurzivni pozivi ne budu beskonacni ili da ne generisemo prevelike nizove u rekurzivnim fjama
    - primer, zaboravimo proveru n==0 u fji i onda se poziva f(0), pa f(-1), f(-2)…
    - rekurzivne fje moraju da imaju izlaz
8. Ovo je najstandardniji oblik rekurzije, slucaj za dimenziju n smo svodili na slucaj za dimenziju n-1
9. Ovo nije obavezno… rekurzijom se smatraju sve fje koje pozivaju samu sebe za vrednosti koje su manje od tekuce, tako da te vrednosti ne moraju biti samo n-1
    - u narednom primeru se pozivamo na prethodne dve vrednosti, n-1 i n-2
    - Primer: fibonaci.c
    - https://upload.wikimedia.org/wikipedia/commons/3/38/AureaFibonacci.jpg
    - https://upload.wikimedia.org/wikipedia/commons/thumb/0/08/NautilusCutawayLogarithmicSpiral.jpg/635px-NautilusCutawayLogarithmicSpiral.jpg
    -
10. U narednom primeru se ne pozivamo na prethodnu vec na ostatak pri deljenju, i tu koristimo pretpostavku da znamo rekurzivni poziv da izvedemo za sve brojeve manje od datog
    - Primer: euklid.c

- a = q0*b + r0, r0 = a%b
- za euklidov algoritam se kaze da je nastariji netrivijalni algoritam koji je preziveo do danas
- Prvi poznati sačuvani opis Euklidovog algoritma se nalazi u *Elementima* (oko 300. p. n. e.), što ga čini najstarijim numeričkim algoritmom koji se još uvek aktivno koristi. U originalu, objašnjen je samo za prirodne brojeve i geometrijske dužine (realne brojeve), ali je u 19. veku uopšten na polinome jedne promenljive i na Gausove cele brojeve, što je dovelo do razvoja novih pojmova apstraktne algebre kao što je Euklidski domen. Euklidov algoritam je dalje uopštavan na drugim matematičkim strukturama, poput čvorova i polinoma više promenljivih.
- Euklidov algoritam ima široku terijsku i praktičnu primenu. Predstavlja ključan element RSA algoritma, metode asimetrične kriptografije koja se u značajnoj meri primenjuje u elektronskom poslovanju. Koristi se pri rešavanju diofantskih jednačina, na primer kod određivanja brojeva koji zadovoljavaju višestruke kongruencije (Kineska teorema o ostacima) ili pri određivanju multiplikativnog inverza konačnog polja. Može se upotrebiti za kostruisanje verižnih razlomaka, u Šturmovoj metodi za određivanje realnih nula polinoma, i još nekoliko savremenih algoritama za faktorizaciju prirodnih brojeva. Na kraju, Euklidov algoritam je osnovno sredstvo za dokazivanje teorema moderne teorije brojeva, kao što su Lagranžova teorema o četiri kvadrata i osnovna teorema aritmetike o jedinstvenoj faktorizaciji prirodnih brojeva.
- zasto se zavrsava? Zato sto se oba argumenta konstantno smanjuju
- pokazi poziv za 15,9 i za 9,15
  - 15 = 1*9 + 6
  - 9 = 1*6 + 3
  - 6 = 2*3 + 0
- Ukoliko je *a* manje od *b*, u prvom koraku algoritma obrcu im se vrednosti

11. Rekurzija mora da ima bazni slucaj i smanjivanje argumenata koje obezbedjuje da cemo stici do baznog slucaja
12. U knjizi ima jos rekurzivnih primera, koje mozete pogledati
13. Osim direktne rekurzije koja poziva samu sebe, postoji i uzajamna rekurzija ili indirektna rekurzija
    - dve funkcije koje se uzajamno pozivaju
    - vestacki primer parni i neparni brojevi
    - bolji primer je definicija aritmetickog izraza nad prirodnim brojevima i operatorima + i *
    - Izraz = niz Sabiraka razdvojenih + (ukljucujuci i mogucnost da postoji samo jedan sabirak)
    - Sabirak = niz Cinilaca razdvojenih operatorom * (ukljucujuci i mogucnost da postoji samo jedan cinilac)
    - Cinilac = prirodan_broj ILI (izraz)
    - tj.izraz u zagradama

14. Sta su dobre a sta lose strane rekurzije i da li je treba koristiti uvek kada je to moguce
    - kod je najcesce kratak i lako razumljiv
    - lakse je dokazati korektnost koda ako je implementirana rekurzivno
    - MANE:
    - treba biti pazljiv, voditi racuna o stek okviru i o potencijalnim greskama
    - potencijalni problem je cena poziva
    - umesto rekurzivne fje zbira elemenata u nizu mogla je da se implementira samo jedna petlja i sve bi bilo uradjeno u jednoj fji umesto u n poziva fja koje zauzimaju memoriju i oduzimaju procesorsko vreme
      - stalna ponavljanja istih izracunavanja – fib…

15. Svaka rekurzivna fja se moze napisati tako da ne koristi rekurziju, kod rekurzivno definisanih podataka (stabla, liste…) nije tako lako.
16. Ne postoji opsti postupak
    - osim u specijalnim slucajevima: repna rekurzija
    - situacija kada je poslednja naredba u fji upravo rekurzivni poziv
    - nije repna: suma brojeva od 1 do n: return n + suma(n-1); zato sto je poslednja naredba sabiranje
    - ako zamenimo redosled, opet je poslednja operacija sabiranje
    - jeste repna rekurzija: euklid.c
    -
17. Pitanje: zasto je moguce elimnisati repnu rekurziju a nije moguce u nekom opstem slucaju?
    - Primer euclid.c
    - originalne vrednosti sa kojima je fja inicijalno pozvana se vise ne koristi i nema razloga da se pamti
    - stek okviri sluze da nama ostanu stare vrednosti fje
    - kompajler moze – umesto da generise rekurzivne pozive svaki put:
        - detektuje da se koristi repna rekurzija i da se a i b ne koristi nakon sledeceg rekurzivnog poziva
        - kompajler moze da odluci da ne otvara novi stek okvir vec ponovo da izvrsava fju nzd sa novim vrednostima promenljivim a i b
    - ovo mozemo mi da uradimo sami za nase fje
    -

				Nzd(3,0)	
			Nzd(6,3)		
		Nzd(9,6)			
	Nzd(15,9) a = 15 b = 9				
main	main	main			

18. ...
19. ...
    -