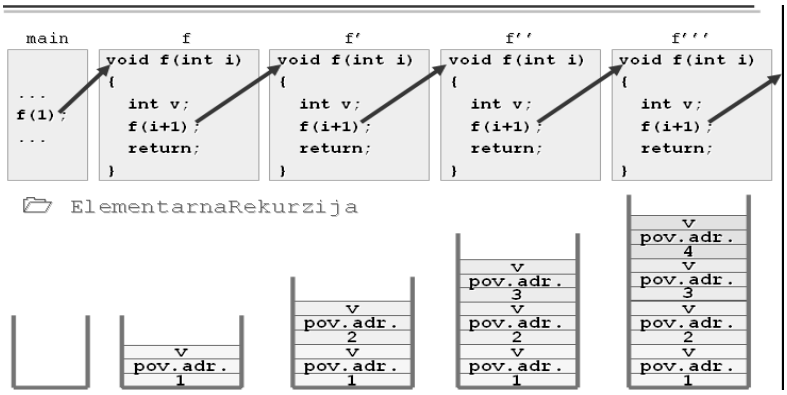
1. Rekurzija

Rekurzija je postupak rješavanja zadataka u kojem neka funkcija (direktno ili indirektno) poziva samu sebe. Za pohranjivanje rezultata i povratak iz rekurzije koristi se stuktura podataka stog (*eng. stack*). Rekurzivni programi su kratki, ali i vrlo spori jer su implementirani kao striktne matematičke definicije zadanih problema. Zbog toga ako je to moguće uvijek treba potražiti iterativno rješenje problema koje je znatno brže od rekurzivnog. Pri rješavanju rekurzivno zadanih problema važno je uočiti *„osnovni“* , najjednostavniji zadanog problema te da složenije slučajeve svedemo na jednostavnije. Osnovni slučaj moramo riješiti direktno, bez rekurzije. Svaki sljedeći rekurzivni poziv mora funkcionirati i mora težiti k osnovnom slučaju.



1.1. Jedan od jednostavnih rekurzivnih algoritama jest izračunavanje **n!** za **n >= 0**

```
int fakt(int n){
  if (n <= 1) {
    return 1;
  } else {
    return n * fakt(n-1);
  }
}
```

0! = 1; 1! = 1; n! = n* (n-1)!;

<u>Primjer: 4!</u>

k = fakt (4);

= 4 * fakt (3);

= 3 * fakt (2);

= 2 * fakt (1);

= 1

1.2. Jedan od najstarijih algoritama je Euklidov postupak za pronalaženje najveće zajedničke mjere (*nzm*) dva nenegativna cijela broja:

```
int nzm (int a, int b) {
  if(b == 0) return a;
  return nzm (b, a % b);
}
```

1.3. Rekurzivni postupak za traženje indeksa prvog člana jednodimenzionalnog polja od **n** članova koji ima vrijednost **x**. Ako takvoga nema, rezultat je **-1**.

```
int trazi (tip A[], tip x, int n, int i) {
   if(i >= n) return -1;
   if(A[i] == x) return i;
   return trazi (A, x, n, i+1);
}
```

Pretraživanje počinje pozivom funkcije **trazi(A, x, n, 0).**

1.4. Traženje najvećeg člana polja.

```
int maxclan (int A[], int i, int n) {
  int imax;
  if (i >= n-1) return n-1;
  imax = maxclan (A, i + 1, n);
  if (A[i] > A[imax]) return i;
  return imax;
}
```

Rekurzija – 1

Zadano je polje cijelih brojeva **int d[5]={1, 2, 3, 4, 5}.** Napisati prvu ***rekurzivnu*** funkciju prototipa: **void rekurzija1 (int d[ ], int n);** koja će ispisati elemente zadanog polja rastućim redoslijedom.
Također napisati i drugu ***rekurzivnu*** funkciju prototipa: **void rekurzija2 (int d[ ], int n);**
koja će ispisati elemente zadanog polja padajućim redoslijedom. Napisati i glavni program u kojem treba definirati zadano polje te pozvati obje rekurzivne funkcije.

```
#include <stdio.h>

void rekurzija1 (int d[ ], int n) {
        //za n==0 imamo osnovni slučaj, inače idemo „dublje“ u rekurziju
        if (n != 0) {
                //ispisuje brojeve od posljednjeg prema prvom
                printf("%d ", d[n-1]);
                rekurzija1(d, n-1);
        }
}

void rekurzija2 (int d[ ], int n) {
        if (n != 0) {
                printf("%d ", d[0]);
                rekurzija2(d+1, n-1);
        }
}

int main ( ) {
        int d[5]={1, 2, 3, 4, 5};
        printf("Prva rekurzivna funkcija: ");
        rekurzija1(d, 5);
        printf("\nDruga rekurzivna funkcija: ");
        rekurzija2(d, 5);
        printf("\n");
        return 0;
}
```

Rekurzija – 2

Napisati glavni program i ***rekurzivnu*** funkciju prototipa: **int suma_niza(int niz[], int n);**
koja će zbrojiti sve članove niza cijelih brojeva definiranog u glavnom programu. Npr. za zadani niz, **int niz[5]={1, 2, 3, 4, 5};** funkcija *suma_niza* mora vratiti 15.

```
#include <stdio.h>

int suma_niza (int niz[ ], int n) {
	if (n==1)
		return niz[0];
	else
		return suma_niza(niz, n-1)+niz[n-1];
}


int main( ) {
	int niz[5]={1, 2, 3, 4, 5};
	printf("Suma niza je: %d\n", suma_niza(niz, 5));
	return 0;
}
```

Rekurzija – 3

Napisati glavni program i ***rekurzivnu*** funkciju prototipa: **int provjeri_palindrom(char *rijec, int duzina);**
koja će za riječ koju korisnik unese sa tipkovnice provjeriti da li je palindrom (riječ koja se čita jednako s lijeva ili s desna) te ispisati odgovarajuću poruku.

```
#include <stdio.h>
#include <string.h>

int provjeri_palindrom (char *rijec, int duzina) {
	//osnovni slučaj nastupa kada je preostala dužina riječi 0 ili 1 te u tom slučaju vraćamo "logičku" jedinicu
	if (duzina<=1)
		return 1;
	else
	//ako nije nastupio osnovni slučaj rekurzija napreduje vraćamo rezultat logiče operacije I
		return((rijec[0] == rijec[duzina-1]) && provjeri_palindrom(rijec+1, duzina-2));
}

int main( ) {
	char rijec[20+1];
	int palindrom;
	printf("Zadajte rijec: ");
	scanf("%s", rijec);
	palindrom=provjeri_palindrom(rijec, strlen(rijec));
	printf("\nRijec %s %s palindrom!\n", rijec, (palindrom==1) ? "je": "nije");
	return 0;
}
```

Rekurzija – 4

Napisati ***rekurzivnu*** funkciju koja za zadani *n* računa aproksimaciju broja Pi (*na 10 decimala*) kao sumu prvih *n* članova reda:

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

Funkcija mora imati prototip **double izracunajPi(int n);**

```
#include <stdio.h>
#include <math.h>

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

int main( ) {
	int n;
	printf("Koliko clanova reda zelite zbrojiti: ");
	scanf("%d", &n);

	printf("\nBroj PI: %.10f\n", izracunajPi(n));
	return 0;
}
```

Rekurzija – 5

Napišite ***rekurzivnu*** funkciju prototipa: **int prebroji(int broj, int znamenka)**
koja će u zadanom broju prebrojiti sva pojavljivanja zadane znamenke. Također napisati i glavni program koji će sa tipkovnice učitati broj i znamenku te pozvati funkciju *prebroji* i ispisati rezultat. Npr. u broju 505 znamenka 5 se pojavljuje 5 puta.

```
#include <stdio.h>

int prebroji(int broj, int znamenka){
	if(broj==0) return 0;
	if (broj%10==znamenka)
	//ako znamenka u broju odgovara, sljedećem koraku rekurzije dodaj 1return 1 + prebroji(broj/10, znamenka);
	else
	//sljedeći korak rekurzije
		return prebroji(broj/10, znamenka);
}
int main(){
	int broj, znamenka;
	printf("Unesite broj: ");
	scanf("%d", &broj);
	printf("Unesite znamenku: ");
	scanf("%d", &znamenka);
	printf("U broju %d znamenka %d se pojavljuje %d puta!\n", broj, znamenka, prebroji(broj, znamenka));
	return 0;
}
```

2. Sortiranje

2.1. Bubble sort

- kreni od početka niza prema kraju
- zamijeni 2 elementa ako je prvi veći od drugog

Bubble sort – 1

Pomoću Bubble sorta sortirajte sljedeći niz brojeva: 6 4 1 8 7 5 3 2.

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

2.2. Selection sort

- pronađi najmanji element niza i zamijeni ga s prvim elementom niza
- ponavljaj s ostatkom niza, smanjujući nesortirani dio

Selection sort – 1

Pomoću Selection sorta sortirajte sljedeći niz brojeva: 4 8 2 1 5 7 3 9 6.

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

2.3. Insertion sort

- postoje dva dijela niza: sortirani i nesortirani
- u svakom koraku sortirani dio se proširuje tako da se u njega na ispravno mjesto ubaci prvi element iz nesortiranog dijela niza

Insertion sort – 1

Pomoću Insertion sorta sortirajte sljedeći niz brojeva: 7 2 9 4 8 5 3 6 1.

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

2.4. Merge sort

- nesortirani niz podijeli se na dva niza podjednake veličine
- svaki podniz sortira se rekurzivno, dok se ne dobije niz od 1 elementa
  - taj niz od jednog elementa je sortiran!
- spoje se dva sortirana podniza u sortirani niz
  - na temelju dva sortirana polja (A i B) puni se treće (C)

2.5. Shell sort

- za *k*-sortirano polje A vrijedi $A[i] \leq A[i+k], \forall i, i+k$ indeksi
- ako je polje *k*-sortirano i dodatno se *t*-sortira (*t*<*k*), ostaje i dalje *k*-sortirano
- potpuno sortirano polje je *1*-sortirano

Shell sort – 1

Pomoću Shell sortirajte sljedeći niz brojeva: 3 2 8 6 4 10 2 1 5 7 9, s koracima *k* = {4, 2, 1}. Rješenje mora sadržavati izgled polja nakon svake zamjene dvaju elemenata.

Korak 4:

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

Najprije gledamo polje 3 4 5, ali tako da prvo gledamo samo prva dva elementa, znači 3 i 4. Vidimo da su oni dobro sortirani: najprije ide 3 a zatim 4. Onda prelazimo na sljedeće polje, 2 10 7. Isto gledamo samo prva dva elementa te uočavamo da su i oni dobro sortirani: najprije ide 2 a zatim 10. Sada dolazimo do polja 8 2 9 i uočavamo da 8 i 2 nisu u najboljem redoslijedu pa im zamijenimo mjesta: najprije ide 2 pa 8.

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

I konačno, dolazimo do polja 6 1, jer je sljedeći element 4, član već anvedenog polja 3 4 5. Vidimo da 6 i 1 nisu dobro poredani pa ih zamijenimo.

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

Sada dolazimo na element 4 i ponovno smo u polju 3 4 5. Sada gledamo sva tri elementa! Vidimo da su 3, 4 i 5 u dobrom redoslijedu pa ih ne diramo. Doalzimo na element 10, koji je dio polja 2 10 7. Uočavamo da treba zamijeniti 10 i 7, pa to i napravimo.

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

Da je recimo gornje polje bilo poredano na sljedeći način: 7 10 2, najprije bismo zamijenili 10 i 2: 7 2 10, a zatim u sljedećem koraku 7 i 2: 2 7 10. Zatim dolazimo na 8 i uočavamo da je polje 2 8 9 dobro poredano. Slijedi 6, odnosno polje 1 6 koje je dobro poredano. Do kraja uočavamo da je sve dobro poredano pa prelazimo na sljedeći korak.

Korak 2:

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

Prešavši na korak 2, dobili smo dva polja: 3 2 4 8 5 9 i 2 1 7 6 10. Najprije gledamo prva dva elementa prvog polja, 3 2. Vidimo da su u krivom redoslijedu pa ga popravimo.

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

Zatim gledamo prva dva elementa drugog polja, 2 1. Isto su u krivom redoslijedu pa ih zamijenimo.

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

Sad dolazimo na 3 i gledamo 2 3 4. U dobrom su poretku pa idemo dalje. Dolazimo na 2 i gledamo 1 2 7. Također su i oni u dobrom poretku. Slijedi 4 pa gledamo 2 3 4 8 – dobar poredak. Zatim 7 pa gledamo 1 2 7 6. Uočavamo da su 7 i 6 u krivom redoslijedu pa ih zamijenimo.

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

Dolazimo na 8 i gledamo 2 3 4 8 5. 8 i 5 su u krivom poretku pa ih zamijenimo.

2 1 3 2 4 6 5 7 8 10 9

Slijedi 7 pa gledamo 1 2 6 7 10. Dobar je poredak. I preostaje nam samo 9 i gledamo 2 3 4 5 8 9 čiji je poredak dobar. Prelazimo na sljedeći korak.

Korak 1:

2 1 3 2 4 6 5 7 8 10 9

Ovdje je samo jedno polje. Ideom po redu: 2 i 1. Njih treba zamijeniti.

1 2 3 2 4 6 5 7 8 10 9

Onda dolazimo na 3 i zamijenimo ga s 2.

1 2 2 3 4 6 5 7 8 10 9

Slijedi 6 koji zamijenimo s 5.

1 2 2 3 4 5 6 7 8 10 9

I na kraju zamijenimo 10 i 9.

1 2 2 3 4 5 6 7 8 9 10

Prikazano samo nakon zamjene svaka dva elementa to izgleda ovako:

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

2.6. Quick sort

- ako je broj članova polja *S* jednak 0 ili 1, povratak u pozivni program
- odabrati bilo koji član *v* u polju *S.* To je stožer (*pivot*)
- podijeli preostale članove polja *S, S \ {v}* u dva odvojena skupa:
  - *S1 = { x ∈ S \ {v} / x ≤ v}* (sve što je manje od stožera, preseli lijevo)
  - *S2 = { x ∈ S \ {v} / x ≥ v}* (sve što je veće od stožera, preseli desno)
- vrati niz sastavljen od *{quicksort (S1), v, quicksort (S2)}*

Quick sort – 1

Pomoću Quick sorta sortirajte sljedeći niz brojeva: 9 10 8 6 1 4 3 5 2 7. Kao pivot odaberite medijan. Rješenje mora sadržavati izgled polja nakon svake zamjene dvaju elemenata.

Kod odabira medijana gleda se element na početku, na kraju i onaj koji se nalazi negdje u sredini. Ako se ne nalazi točno u sredini onda odaberite onaj element tako da tri odabrana element tvore neki niz koji „dočarava" poečtak, sredinu i kraj zadanog niza. Recimo u našem nizu odabiremo 9, 7 i 1. 9 i 7 smo odabrali jer se 9 nalazi na početku a 7 na kraju niza. Da odabiremo element koji se nalazi točno u sredini, taj element nalazio bi se između brojeva 1 i 4, a pošto moramo odabrati jedan od ta dva elementa, odabiremo 1 jer će on bolje „dočaravati" početak, kraj i sredinu niza.

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

Zatim te elemente poslažemo po redu: 1 7 9.

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

Sljedeći korak je taj da se medijan zamjenjuje s predzadnjim elementom niza: 7 <-> 2.

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

Sad krenemo sortirati niz lijevo od pivota, tj. lijevo od 7. Postavimo kazaljku ***i*** na 1 i kazaljku ***j*** na 5. Kazaljka ***i*** mora pokazivati na elemente koji su manji od 7, a kazaljka ***j*** elemente koji su veći od 7. Ako su i jedna i druga tvrdnja netočne, tj. kazaljka ***i*** pokazuje na element veći od 7 a kazaljka ***j*** na element manji od 7, onda elemente na koje pokazuju ove dvije kazaljke zamijenimo.

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

1 je manji od 7 pa uvećavamo kazaljku ***i***.

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

10 je veći od 7 pa prelazimo na kazaljku ***j***. 5 je manji od 7 pa smo dobili obadvije netočne tvrdnje i treba zamijeniti ova dva elementa.

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

Zatim opet krećemo od kazaljke ***i*** te juvećavamo. 8 je veći od 7 pa prelazimo na kazaljku ***j***. Uvećamo ju. 3 je manji od 7 pa opet moramo ta dva elementa zamijeniti.

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

6, 2 i 4 su redom manji od 7 te dolazimo na 8 koji je veći od 7 i tu zaustavljamo kazaljku ***i***. 4 je manji od 7 i tu zaustavljamo kazaljku ***j***.

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

Vidimo da su se kazaljke mimoišle. U tom slučaju pivot zamjenjujemo s onim elementom na koji pokazuje kazaljka ***i***, a to je u ovom slučaju 8.

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

Ovime smo postigli da nam se elementi koji su manji od pivota nalaze s lijeve strane (to su redom: 1 5 3 6 2 4), odnosno da su nam elementi veći od pivota s desne strane (redom: 10 8 9).

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

Sad dalje sortiramo tako da najprije sortiramo lijevi niz a zatim desni, s time da primjenjujemo isti postupak kao za početni niz. Odaberimo najprije medijan za lijevi niz. A je na početku, 4 na kraju, a između elemenata 3 i 6 odaberimo na primjer 6.

1	5	3	**6**	2	**4**

Nakon njihove zamjene dobijemo sljedeći niz:

1	5	3	**4**	2	**6**

Zamijenimo pivot (4) sa predzadnjim elementom (2).

1	5	3	2	**4**	6

Postavimo ponovno kazaljke ***i*** i ***j***.

1 ***i***	5	3	**2** ***j***	**4**	6
1	**5** ***i***	3	**2** ***j***	**4**	6

Zamijenimo:

1	**2** ***i***	3	**5** ***j***	**4**	6

Uvećamo kazaljke:

1	2	**3** ***j***	**5** ***i***	**4**	6

Kazaljke su se mimoišle pa pivot zamijenimo s elementom na mjestu s kazaljkom ***i***.

1	2	3 ***j***	**4** ***i***	**5**	6

Ovime smo sortirali lijevi niz. Sortirajmo sada desni istim postupkom.

10 8 9

Pri odabiru medijana kod ovog polja automatski ćemo ga i sortirati jer su samo tri elementa.

8 9 10

I time smo obavili naše sortiranje.

Prikazavši ovo sortiranje nakon svake zamjene dvaju elemenata, to izgleda ovako:

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

Quick sort – 2

Za zadani niz brojeva: 3 2 8 6 4 10 2 1 5 7 9, ilustrirati sortiranje postupkom *quicksort* tako da se kao stožer odabire prvi element u polju. Rješenje mora sadržavati izgled polja nakon svake zamjene dvaju elemenata.

3 2 8 6 4 10 2 1 5 7 9

Ovaj zadatak rješavamo slično kao i prethodni, samo što kao pivot odabiremo prvi element i ne zamjenjujemo ga s predzadnjim elementom.

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

Niz sortiramo tako da elemente s desne strane sortiramo po istom postupku sa kazaljkama ***i*** i ***j***.

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

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

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

Kazaljke su se mimoišle – pivot na mjesto broja s kazaljkom ***i***.

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

Zatim sortiramo lijevi niz: 2 2 1, na isti način kao i početni niz.

2	2 ***i***	1 ***j***
2	**2** ***j***	**1** ***i***
1	**2** ***j***	**2** ***i***

Lijevi niz smo sortirali. Još je potrebno desni.

Uočimo da je dio 1 2 2 3 4 već sortirani pa samo sortiramo dio 10 6 8 5 7 9.

10	**6** *i*	8	5	7	**9** ***j***
10	6	8	5	7	**9** ***i, j***
9	6	8	5	7	**10** ***i, j***

9	**6** *i*	8	5	**7** ***j***	**10**
9	6	8	5	**7** ***i, j***	**10**
7	6	8	5	**9** ***i, j***	**10**

7	**6** *i*	8	**5** ***j***	**9**	**10**

7	**6**	**8** *i*	**5** *j*	**9**	**10**

7	**6**	**5** *i*	**8** *j*	**9**	**10**

5	**6**	**7**	**8**	**9**	**10**

Prikazavši ovo sortiranje nakon svake zamjene dvaju elemenata, to izgleda ovako:

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

3. Stog i liste

3.1. Stog

Stog je struktura podataka kod koje se posljednji pohranjeni podatak prvi uzima u obradu. Da bismo mogli raditi sa stogom, potrebne su nam sljedeće operacije:

- dodavanje (*push*) elemenata na vrh stoga (*top*)
- brisanje (*pop*) elemenata s vrha stoga
- inicijalizacija praznog stoga

Pojedina operacija **dodaj** ili **brisi** zahtijeva jednako vremena bez obzira na broj pohranjenih podataka. Situacija da je stog pun može zahtijevati alociranje dodatne memorije i ponovno izvođenje programa. Prazan stog ne mora značiti pogrešku.

3.2. Stog realiziran statičkim poljem

U jednodimenzionalno polje zadane strukture dodaju se ili brišu pojedine stavke prema načelu *Last In First Out* (*LIFO*). Ažurira se vrijednost varijable koja predstavlja vrh stoga.

Inicijalizacija praznog stoga - postavljanje vrha na početnu vrijednost.

```
typedef struct {
       int vrh, polje[MAXSTOG];
} Stog;

void init_stog(Stog *stog){
       stog->vrh = -1;
}
```

Dodavanje elemenata na stog

```
int dodaj (int element, Stog *stog) {
  if (stog->vrh >= MAXSTOG-1) return 0;
  stog->vrh++;
  stog->polje[stog->vrh] = element;
  return 1;
}
```

Skidanje elemenata sa stoga

```
int skini (int *element, Stog *stog) {
      if (stog->vrh < 0) return 0;
      *element = stog->polje[stog->vrh];
      stog->vrh--;
      return 1;
}
```

Pozivni program

```
Stog stog;
init_stog(&stog);
dodaj(5, &stog);
skini(&element, &stog);
```

3.3. Liste

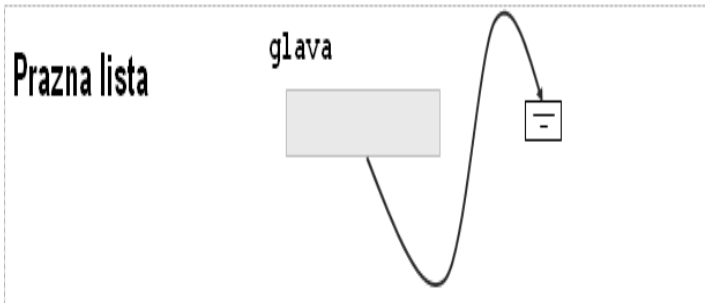
Linearna lista $A=(a_1,a_2,...a_n)$ struktura je podataka koja se sastoji od uređenog niza elemenata odabranih iz nekog skupa podataka. Za linearnu listu kažemo da je prazna ako ima **n=0** elemenata. Elementi liste a_i nazivaju se još i atomi.

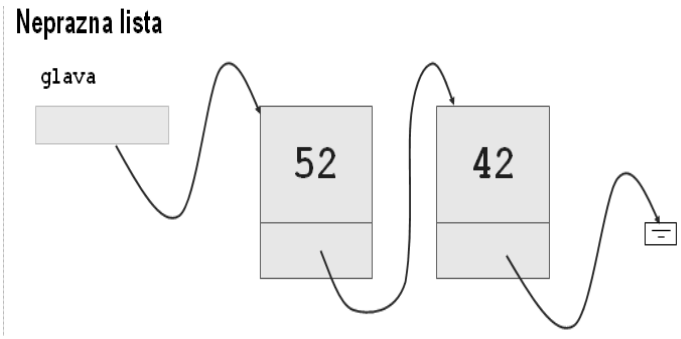
Linearna lista može se realizirati statičkom strukturom podataka – poljem.

Dinamička podatkovna struktura za realizaciju liste sastoji se od pokazivača na prvi element liste i od proizvoljnog broja atoma. Svaki se atom sastoji od podatkovnog dijela i pokazivača na sljedeći element liste. Memorija za svaki atom liste zauzme se u trenutku kad je potrebna za pohranu podataka, a oslobađa kad se podatak briše . Granulacija je veličine atoma.

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

Prazna I neprazna lista





3.4. Stog realiziran listom

Realizacija stoga statičkim poljem imala je veliki nedostatak jer je postojala mogućnost prepunjenja polja. Sada ćemo stog implementirati listom što će nam omogućiti da stavimo na stog gotovo neograničen broj elemenata. Jedino ograničenje predstavljat će veličina radnog spremnika u računalu.

Struktura podataka

```
struct at {
  tip element;
  struct at *sljed;
};
typedef struct at atom;
typedef struct{
      atom *vrh;
} Stog;
```

Inicijalizacija stoga

```
void init_stog(Stog *stog){
      stog->vrh = NULL;
}
```

Dodavanje elemenata na stog

```
int dodaj (tip element, Stog *stog) {
       atom *novi;
       if ((novi = (atom*) malloc(sizeof(atom))) != NULL) {
              novi->element = element;
              novi->sljed = stog->vrh;
              stog->vrh = novi;
              return 1;
       }
       else return 0;
}
```

Skidanje elemenata sa stoga

```
int skini (tip *element, Stog *stog) {
       atom *pom;
       if (stog->vrh == NULL) return 0;
       *element = stog->vrh->element;
       pom = stog->vrh->sljed;     /* adresa novog vrha */
       free(stog->vrh);            /* obriši stari vrh */
       stog->vrh = pom;            /* postavi novi vrh */
       return 1;
}
```

Pozivni program

```
Stog stog;
init_stog(&stog);
dodaj (5, &stog);
skini (&element, &stog);
```

ZADACI:

Zajednički dio za sve zadatke:

Na disku postoji formatirana datoteka *alatnica.txt.* U svakom retku datoteke nalazi se naziv alata (*niz znakova, najveće duljine 20+1*) te količina tog alata (*int*). Strukturu podataka za pohranu zapisa možete definirati na sljedeći način:

```
typedef struct {
        char alat[20+1];
        int kolicina;
} zapis;
```

Naš alatničar je pretrpan poslom pa nema vremena pospremiti svoje skladište već alat baca na stog. Kad alatničar posegne za nekim alatom on sa stoga prvo vadi posljednji alat kojeg je stavio na stog (*LIFO*).

StogListom – 1

Kako bi smanjio nered alatničar je odlučio na svoj stog baciti samo one alate kojih ima manje od 10 komada. Napišite funkcije za inicijalizaciju i dodavanje elemenata na stog realiziran jednostruko povezanom listom. Napišite i glavni program koji mora učitati sve zapise o alatima iz zadane datoteke te na stog dodati sve alate čija je količina manja od 10.

```
#include <stdio.h>
#include <malloc.h>

typedef struct{
	char alat[20+1];
	int kolicina;
}zapis;

typedef struct at{
  tip element;
  struct at *sljed;
} atom;
typedef struct {
	atom *vrh;
} Stog;

void init_stog(Stog *stog){
	stog->vrh=NULL;
}

int dodaj(tip element, Stog *stog){
	atom *novi;
	if((novi=(atom*)malloc(sizeof(atom)))!=NULL){
		novi->element=element;
		novi->sljed=stog->vrh;
		stog->vrh=novi;
		return 1;
	}
	else return 0;
}

int main( ) {
	FILE *alatnica;
	zapis z;
	Stog stog;

	init_stog(&stog);
	alatnica=fopen("alatnica.txt", "r");

	while(fscanf(alatnica, "%s %d", z.alat, &z.kolicina)!=EOF){
		if(z.kolicina<10){
			dodaj(z, &stog);
			printf("Dodan: %20s - %4d komada\n", z.alat, z.kolicina);
		}
	}
	fclose(alatnica);
	return 0;
}
```

StogListom – 2

Završio je još jedan radni dan i naš alatničar mora pospremiti sav alat, tj. baciti ga na stog. A sutra ga mora izvaditi sa stoga i pripremiti za rad. Napišite funkcije za inicijalizaciju, dodavanje i skidanje elemenata sa stoga realiziranog jednostruko povezanom listom. Potom napišite glavni program koji će iz zadane datoteke učitati sve zapise o alatima te ih staviti na stog, a potom i skinuti sa stoga.

```
typedef struct{
        char alat[20+1];
        int kolicina;
} zapis;

typedef zapis tip;
typedef struct at{
  tip element;
  struct at *sljed;
}atom;
typedef struct{
        atom *vrh;
}Stog;

void init_stog(Stog *stog){
        stog->vrh=NULL;
}

int dodaj(tip element, Stog *stog); (kao i u prethodnom primjeru!!!)

int skini(tip *element, Stog *stog){
        atom *pom;
   if(stog->vrh==NULL) return 0;
        *element=stog->vrh->element;
        pom=stog->vrh->sljed;   //adresa novog vrha
        free(stog->vrh);                // obriši stari vrh
        stog->vrh = pom;       // postavi novi vrh
        return 1;
}
int main(){
        FILE *alatnica;
        zapis z;
        Stog stog;

        init_stog(&stog);
        alatnica=fopen("alatnica.txt", "r");

        printf("\nStavljam alat...\n\n");
        while(fscanf(alatnica, "%s %d", z.alat, &z.kolicina)!=EOF){
                dodaj(z, &stog);
                printf("Dodan: %20s - %4d komada\n", z.alat, z.kolicina);
        }
        printf("\nVadim alat...\n\n");
        while(skini(&z, &stog))
                printf("Skinut: %20s - %4d komada\n", z.alat, z.kolicina);

        fclose(alatnica);
        return 0;
}
```

StogListom – 3

Radnici su baš zahtjevni, naredili su našem alatničaru da sa stoga izvadi bušilice i brusilice, a da ne poremeti redoslijed ostalih alata na stogu. Napišite sve potrebne funkcije za rad sa stogom realiziranog jednostruko povezanom listom. Napišite funkciju prototipa:
void busilica_brusilica(Stog *stog);
koja će koristeći pomoćni stog sa početnog stoga izvaditi bušilice i brusilice na način da očuva redoslijed ostalih elemenata stoga. Također napišite glavni program u kojem ćete dodati na stog sve zapise o alatima iz zadane datoteke, pozvati funkciju *busilica_brusilica* te potom skinuti sve elemente sa stoga.

```
#include <stdio.h>
#include <malloc.h>

typedef struct{
        char alat[20+1];
        int kolicina;
}zapis;

typedef zapis tip;

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


typedef struct{
        atom *vrh;
}Stog;
//funkcija za inicijalizaciju
void init_stog(Stog *stog){
        stog->vrh=NULL;
}

int dodaj(tip element, Stog *stog){
        atom *novi;
        if((novi=(atom*)malloc(sizeof(atom)))!=NULL){
                novi->element=element;
                novi->sljed=stog->vrh;
                stog->vrh=novi;
                return 1;
        }
        else return 0;
}

int skini(tip *element, Stog *stog){
        atom *pom;
   if(stog->vrh==NULL) return 0;
        *element=stog->vrh->element;
        pom=stog->vrh->sljed;   //adresa novog vrha
        free(stog->vrh);                // obriši stari vrh
        stog->vrh = pom;       // postavi novi vrh
        return 1;
}


void busilica_brusilica(Stog *stog){
   Stog pomocni;
```

```
	zapis z;
	init_stog(&pomocni);

	//OPREZ funkciji skini predajemo stog, ne &stog, jer je u funkciju stog prenesen pomoću pokazivača
	while(skini(&z, stog))
		if(strcmp(z.alat, "Busilica")!=0 && strcmp(z.alat, "Brusilica")!=0)
			dodaj(z, &pomocni);
	while(skini(&z, &pomocni))
		dodaj(z, stog);
}
int main(){
	FILE *alatnica;
	zapis z;
	Stog stog;

	init_stog(&stog);
	alatnica=fopen("alatnica.txt", "r");

	printf("\nStavljam alat...\n\n");
	while(fscanf(alatnica, "%s %d", z.alat, &z.kolicina)!=EOF){
		dodaj(z, &stog);
		printf("Dodan: %20s - %4d komada\n", z.alat, z.kolicina);
	}

	printf("\nVadim busilice i brusilice...\n");
	busilica_brusilica(&stog);

	printf("\nVadim alat...\n\n");
	while(skini(&z, &stog))
		printf("Skinut: %20s - %4d komada\n", z.alat, z.kolicina);

	fclose(alatnica);
	return 0;
```

StogListom – 4

Riješite prethodni zadatak bez korištenja pomoćnog stoga u funkciji *busilica_brusilica*.
Napomena: koristiti **rekurziju**.

```
void busilica_brusilica(Stog *stog){
	zapis z;
	if(skini(&z, stog)){
		busilica_brusilica(stog);
		if(strcmp(z.alat, "Busilica")!=0 && strcmp(z.alat, "Brusilica")!=0)
			dodaj(z, stog);
	}
```

4. Redovi

Red je struktura podataka u kojoj se prvi pohranjeni podatak također i uzima prvi. U literaturi ćete često naići na oznaku FIFO, što je kratica od *First In First Out*.

Potrebne operacije koje treba ostvariti za rad s redom su:

1) Dodavanje elemenata na stog (*eng. Push*)
2) Skidanje elemenata sa stog (*eng. Pop*)
3) Inicijalizacija praznog stoga

Osim statičkim poljem red se može ostvariti i povezanom listom. Veliki nedostatak implementacije reda statičkim poljem je mogućnost prepunjenja jer imamo polje ograničene veličine (*MAXRED*).

Da bi omogućili razlikovanje punog i praznog reda, jedan element cirkularnog polja uvijek mora biti prazan. Red je prazan ako vrijedi ulaz==izlaz, a pun je ako vrijedi (ulaz+1)%MAXRED==izlaz.

4.1. Red realiziran cirkularnim poljem

Struktura podataka

```
typedef struct {
       tip polje[MAXRED];
       int ulaz, izlaz;
} Red;
```

Inicijalizacija praznog reda

```
void init_stog(Stog *stog){
      stog->vrh = -1;
}
```

Dodavanje elemenata u red

```
int dodaj (tip element, Red *red) {
 if ((red->ulaz+1) % n == red->izlaz)
return 0;
 red->ulaz++;
 red->ulaz %= n;
 red->polje[red->ulaz] = element;
 return 1;
}
```

Skidanje elemenata iz reda

```
int skini (tip *element, Red *red) {
  if (red->ulaz == red->izlaz)
return 0;
  red->izlaz ++;
  red->izlaz %= n;
  *element = red->polje[red->izlaz];
  return 1;
}
```

Pozivni program

```
Red red;
init_red(&red);
dodaj(5, &red);
skini(&broj, &red);
```

4.2. Red realiziran listom

Struktura podataka

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

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

Inicijalizacija praznog reda

```
void init_red(Red *red){
       red->ulaz = NULL;
       red->izlaz = NULL;
}
```

Dodavanje elemenata u red

```
int DodajURed (int element, Red *red) {
   atom *novi;
   if (novi = malloc (sizeof (atom))) {
      novi->element = element;
      novi->sljed = NULL;
      if (red->izlaz == NULL) red->izlaz = novi;     // ako je red
bio prazan
            else (red->ulaz)->sljed = novi;   // inace, stavi na
kraj
      red->ulaz = novi;                       // zapamti zadnjeg
      return 1;
}
   return 0;
}
```

Skidanje elemenata iz reda

```
int SkiniIzReda (int *element, Red *red) {
  atom *stari;
  if (red->izlaz) {                 // ako red nije prazan
    *element = red->izlaz->element;       // element koji se skida
    stari = red->izlaz;            // zapamti trenutni izlaz
    red->izlaz = red->izlaz->sljed;       // novi izlaz
    free (stari);                  // oslobodi memoriju skinutog
    if (red->izlaz == NULL) red->ulaz = NULL; // prazan red
   return 1;
  }
  return 0;
}
```

Pozivni program

```
Red red;
init_red(&red);
DodajURed(52, &red);
SkiniIzReda(&broj,&red);
```

ZADACI:

RedPoljem – 1

Napišite funkcije za inicijalizaciju, dodavanje i skidanje elemenata iz reda realiziranog cirkularnim poljem.Napišite glavni program koji će dodati u red 100 pseudoslučajnih cijelih brojeva iz intervala [0, 100] te ih potom skinuti iz reda i pronaći njihovu ukupnu sumu.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAXRED 128

typedef int tip;
typedef struct{
        tip polje[MAXRED];
        int ulaz, izlaz;
}Red;

void init_red(Red *red){
        red->ulaz = 0; red->izlaz = 0;
}

int dodaj(tip element, Red *red){
         if ((red->ulaz+1)%MAXRED==red->izlaz) return 0;
         red->ulaz++;
         red->ulaz%=MAXRED;
         red->polje[red->ulaz]=element;
 return 1;
}

int skini(tip *element, Red *red) {
         if(red->ulaz==red->izlaz) return 0;
         red->izlaz++;
         red->izlaz%=MAXRED;
         *element=red->polje[red->izlaz];
         return 1;
}
int main(){
        Red red;
        int element, ukupno=0, i;

        init_red(&red);
        srand((unsigned)time(NULL));

        for(i=0; i<100; i++){
                element=rand()%(100-0+1);
                dodaj(element, &red);
        }

        while(skini(&element, &red))
```

```
                ukupno+=element;

        printf("Zbroj brojeva u redu: %d\n", ukupno);
        return 0;
}
```

RedPoljem – 2

Napišite sve funkcije potrebne za rad s redom realiziranog cirkularnim poljem. Potom napišite glavni program koji mora izgenerirati 200 slučajnih cijelih brojeva iz intervala [0, 100] te staviti u red brojeve iz tog intervala koji se nisu nijednom generirali. Naposlijetku skinuti sve elemente iz reda.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAXRED 128

typedef int tip;
typedef struct{
        tip polje[MAXRED];
        int ulaz, izlaz;
}Red;

void init_red(Red *red){
        red->ulaz = 0; red->izlaz = 0;
}

int dodaj(tip element, Red *red){
         if ((red->ulaz+1)%MAXRED==red->izlaz) return 0;
         red->ulaz++;
         red->ulaz%=MAXRED;
         red->polje[red->ulaz]=element;
 return 1;
}

int skini(tip *element, Red *red) {
          if(red->ulaz==red->izlaz) return 0;
          red->izlaz++;
          red->izlaz%=MAXRED;
          *element=red->polje[red->izlaz];
          return 1;
}
int main(){
        Red red;
        int element, i;
        int pojavljivanja[101]={0};

        init_red(&red);
        srand((unsigned)time(NULL));

        for(i=0; i<200; i++){
                element=rand()%(100-0+1);
                pojavljivanja[element]++;
        }
        printf("\n\t*** Dodajem u red ***\n\n");
        for(i=0; i<=100; i++)
                if(!pojavljivanja[i]){
                        dodaj(i, &red);
```

```
                        printf("%3d\n", i);
                }
        printf("\n\t*** Skidam iz reda ***\n\n");
        while(skini(&element, &red))
                printf("%3d\n", element);

        return 0;
}
```

RedPoljem – 3

Napišite sve funkcije potrebne za rad s redom realiziranog cirkularnim poljem. Potom napišite glavni program koji mora izgenerirati 100 slučajnih cijelih brojeva iz intervala [0, 100] i staviti ih u prvi red. Zatim skinuti sve elemente iz prvog reda i prebaciti u drugi red brojeve veće od 75. Naposlijetku skinuti sve elemente iz drugog reda.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
//Najveća veličina polja
#define MAXRED 128

typedef int tip;
typedef struct{
        tip polje[MAXRED];
        int ulaz, izlaz;
}Red;

void init_red(Red *red){
        red->ulaz = 0; red->izlaz = 0;
}

int dodaj(tip element, Red *red){
         if ((red->ulaz+1)%MAXRED==red->izlaz) return 0;
         red->ulaz++;
         red->ulaz%=MAXRED;
         red->polje[red->ulaz]=element;
 return 1;
}

int skini(tip *element, Red *red) {
         if(red->ulaz==red->izlaz) return 0;
         red->izlaz++;
         red->izlaz%=MAXRED;
         *element=red->polje[red->izlaz];
         return 1;
}
int main(){
        Red red1, red2;
        int element, i;

        init_red(&red1); init_red(&red2);
        srand((unsigned)time(NULL));

        for(i=0; i<100; i++){
                element=rand()%(100-0+1);
                dodaj(element, &red1);
        }
```

```
	while(skini(&element, &red1))
		if(element>75)
			dodaj(element, &red2);

	printf("\n\t*** Sadrzaj drugog reda (veci od 75) ***\n\n");
	while(skini(&element, &red2))
		printf("%3d\n", element);
	return 0;
}
```

RedPoljem – 4

Napišite sve funkcije potrebne za rad s redom realiziranog cirkularnim poljem. Napišite funkciju prototipa:

void okreni(Red *red1, Red *red2);

koja će bez korištenja pomoćnog reda ili pomoćnog polja dodati sve elemente prvog reda u drugi, ali obrnutim redoslijedom. Napisati glavni program koji će izgenerirati 10 slučajnih cijelih brojeva iz intervala [0, 100] te ih dodati u prvi red. Glavni program mora pozvati funkciju *okreni* te po izvršenju funkcije skinuti sve elemente iz drugog reda i ispisati ih.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAXRED 128

typedef int tip;
typedef struct{
	tip polje[MAXRED];
	int ulaz, izlaz;
}Red;

void init_red(Red *red){
	red->ulaz = 0; red->izlaz = 0;
}

int dodaj(tip element, Red *red){
	 if ((red->ulaz+1)%MAXRED==red->izlaz) return 0;
	 red->ulaz++;
	 red->ulaz%=MAXRED;
	 red->polje[red->ulaz]=element;
 return 1;
}

int skini(tip *element, Red *red) {
	 if(red->ulaz==red->izlaz) return 0;
	 red->izlaz++;
	 red->izlaz%=MAXRED;
	 *element=red->polje[red->izlaz];
	 return 1;
}
void okreni(Red *red1, Red *red2){
	int element;
	if(skini(&element, red1)){
		okreni(red1, red2);
		dodaj(element, red2);
	}
}
int main(){
	Red red1, red2;
```

```
	int element, i;

	init_red(&red1); init_red(&red2);
	srand((unsigned)time(NULL));

	printf("\n\t*** Prvi red ***\n\n");
	for(i=0; i<10; i++){
		element=rand()%(100-0+1);
		dodaj(element, &red1);
		printf("%3d\n", element);
	}

	okreni(&red1, &red2);

	printf("\n\t*** Drugi red ***\n\n");
	while(skini(&element, &red2))
		printf("%3d\n", element);

	return 0;
}
```

RedPoljem – 5

Napišite sve funkcije potrebne za rad s redom realiziranog cirkularnim poljem. Napišite funkciju prototipa:

void spoji(Red *red1, Red *red2, Red *novi);

koja će bez korištenja pomoćnog reda ili pomoćnog polja skidati elemente iz prvog i drugog polja istodobno te zbroj takva dva elementa dodati u novi red, ali obrnutim redoslijedom. Napisati glavni program koji će u prvi red dodati 10 slučajnih cijelih brojeva iz intervala [0, 100], a u drugi red 10 slučajnih cijelih brojeva iz intervala [100, 200] te pozvati funkciju *spoji*. Npr.

Prvi red	Drugi red	Novi red
53	176	**44+162=206**
30	120	**46+102=148**
46	102	**30+120=150**
44	162	**53+176=229**

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define MAXRED 128

typedef int tip;
typedef struct{
	tip polje[MAXRED];
	int ulaz, izlaz;
}Red;

void init_red(Red *red){
	red->ulaz = 0; red->izlaz = 0;
}
int dodaj(tip element, Red *red){
	 if ((red->ulaz+1)%MAXRED==red->izlaz) return 0;
	 red->ulaz++;
	 red->ulaz%=MAXRED;
	 red->polje[red->ulaz]=element;
 return 1;
}
```

```
int skini(tip *element, Red *red) {
	 if(red->ulaz==red->izlaz) return 0;
	 red->izlaz++;
	 red->izlaz%=MAXRED;
	 *element=red->polje[red->izlaz];
	 return 1;
}
void spoji(Red *red1, Red *red2, Red *novi){
	int element1, element2;
	if(skini(&element1, red1) && skini(&element2, red2)){
		spoji(red1, red2, novi);
		dodaj(element1+element2, novi);
		printf("%3d + %3d = %3d\n", element1, element2, element1+element2);
	}
}
int main(){
	Red red1, red2, novi;
	int element1, element2, i;

	init_red(&red1); init_red(&red2); init_red(&novi);
	srand((unsigned)time(NULL));

	for(i=0; i<10; i++){
		element1=rand()%(100-0+1);
		element2=rand()%(200-100+1);
		dodaj(element1, &red1);
		dodaj(element2, &red2);
		printf("Prvi red: %3d  Drugi red: %3d\n", element1, element2);
	}

	printf("\t\n*** Novi red ***\n\n");
	spoji(&red1, &red2, &novi);

	return 0;
}
```

Zajednički dio za sve zadatke:

Na disku postoji slijedna binarna datoteka *studenti.dat.* Svaki zapis u datoteci je jednake veličine i sadrži ime (*niz znakova duljine do 20+1*), prezime (*niz znakova duljine do 20+1*) fakultet (*niz znakova duljine do 20+1*) te godinu rođenja (*int*) i stanje na x-ici (*int*) pojedinog studenta . Strukturu podataka za pohranu zapisa možete definirati na sljedeći način:

```
typedef struct{
        char ime[20+1];
        char prezime[20+1];
        char fakultet[20+1];
        int godina_rodjenja;
        int stanje_na_xici;
}student;
```

Studenti svakodnevno čekaju u menzi u redu za ručak i pritom se često nađu u čudnim i malo manje čudnim situacijama. Za rješavanje dozvoljeno je koristiti pomoćne redove, ako nije navedeno drukčije. Korištenje dodatnih polja nije dopušteno.

RedListom – 1

Studenti su odjednom nahrupili u red za ručak, no umiješao se zaštitar i odlučio pustiti u red prvo one koji su se rodili prije 1987. Napisati sve funkcije potrebne za rad s redom realiziranim povezanom listom. Napisati funkciju prototipa:

void prednost_starijima(Red *red);

koja će korištenjem pomoćnog reda u početni red prvo pustiti studente rođene prije 1987, a zatim preostale. Napisati i glavni program koji mora učitati sve zapise o studentima iz zadane datoteke, pozvati funkciju *prednost_starijima* te skinuti sve studente iz tako nastalog reda i ispisati ih.

```
#include <stdio.h>
#include <string.h>

typedef struct{
        char ime[20+1];
        char prezime[20+1];
        char fakultet[20+1];
        int godina_rodjenja;
        int stanje_na_xici;
}student;

typedef student tip;
typedef struct at{
        tip element;
        struct at *sljed;
}atom;
typedef struct {
        atom *ulaz, *izlaz;
}Red;

void init_red(Red *red){
        red->ulaz = NULL;
        red->izlaz = NULL;
}
int DodajURed(tip element, Red *red){
        atom *novi;
        if(novi=malloc(sizeof(atom))) {
                novi->element=element;
                novi->sljed=NULL;
                if (red->izlaz==NULL) red->izlaz=novi;
                else red->ulaz->sljed=novi;
```

```
                red->ulaz=novi;
                return 1;
                }
        return 0;
}
int SkiniIzReda(tip *element, Red *red) {
  atom *stari;
  if (red->izlaz) {
        *element = red->izlaz->element;
        red->izlaz = red->izlaz->sljed;
         free (stari);
        if (red->izlaz == NULL) red->ulaz = NULL;
        return 1;
        }
return 0;
}

void prednost_starijima(Red *red){
        Red pomocni1, pomocni2;
        student s;
        init_red(&pomocni1); init_red(&pomocni2);
        while(SkiniIzReda(&s, red)){
                if(s.godina_rodjenja<1987)
                        DodajURed(s, &pomocni1);
                else
                        DodajURed(s, &pomocni2);
        }
        while(SkiniIzReda(&s, &pomocni1))
                DodajURed(s, red);
        while(SkiniIzReda(&s, &pomocni2))
                DodajURed(s, red);
}
int main(){
        FILE *binarna;
        student s;
        Red red;

        init_red(&red);
        binarna=fopen("studenti.dat", "rb");

        while(fread(&s, sizeof(student), 1, binarna))
                DodajURed(s, &red);
        prednost_starijima(&red);
        while(SkiniIzReda(&s, &red))
                printf("%s %s, %s, %d, %d\n", s.ime, s.prezime, s.fakultet, s.godina_rodjenja, s.stanje_na_xici);
        fclose(binarna);
        return 0;
}
```

RedListom – 2

Pristigli val poskupljenja nije zaobišao ni studentske menze. Napisati funkciju prototipa:
void izbaci(Red *red);
koja će iz rad izbaciti sve studente čije je stanje na x-ici manje od 200 kuna. Napisati i glavni program koji će iz zadane datoteke učitati sve zapise o studentima, dodati ih u red te pozvati funkciju *izbaci*. Naposlijetku skinuti sve studente iz reda i ispisati podatke o njima.

```
void izbaci(Red *red){
	Red pomocni;
	student s;
	init_red(&pomocni);
	while(SkiniIzReda(&s, red))
		if(s.stanje_na_xici>=200)
			DodajURed(s, &pomocni);
		while(SkiniIzReda(&s, &pomocni))
		DodajURed(s, red);
}
int main(){
	FILE *binarna;
	student s;
	Red red;

	init_red(&red);
	binarna=fopen("studenti.dat", "rb");

	while(fread(&s, sizeof(student), 1, binarna))
		DodajURed(s, &red);

	izbaci(&red);

	while(SkiniIzReda(&s, &red))
		printf("%s %s, %s, %d, %d\n", s.ime, s.prezime, s.fakultet, s.godina_rodjenja, s.stanje_na_xici);

	fclose(binarna);
	return 0;
}
```

RedListom – 3

Fakultetsko vijeće je donijelo odluku da se oni Fer-ovci koji su se danas zatekli u redu za menzu izvuku iz reda i posluže u ekskluzivnom restoranu. Redoslijed studenata u starom redu mora ostati isti kako se studenti ne bi međusobno posvađali. Odluku vijeća ostvarite funkcijom prototipa:

void ferovci(Red *red, Red *ekskluzivni);

Također napišite glavni program koji će iz zadane datoteke učitati sve zapise o studentima, dodati ih u red te pozvati funkciju *ferovci*. Naposlijetku skinuti sve studente iz ekskuzivnog reda i ispisati podatke o njima.

```
void ferovci(Red *red, Red *ekskluzivni){
	Red pomocni;
	student s;
	init_red(&pomocni);
	while(SkiniIzReda(&s, red)){
		if(strcmp(s.fakultet, "FER")!=0)
			DodajURed(s, &pomocni);
		else
			DodajURed(s, ekskluzivni);
	}
	while(SkiniIzReda(&s, &pomocni))
	DodajURed(s, red);
}

int main(){
	FILE *binarna;
	student s;
	Red red, ekskluzivni;
	init_red(&red); init_red(&ekskluzivni);
	binarna=fopen("studenti.dat", "rb");

	while(fread(&s, sizeof(student), 1, binarna))
		DodajURed(s, &red);

	ferovci(&red, &ekskluzivni);

	while(SkiniIzReda(&s, &ekskluzivni))
		printf("%s %s, %s, %d, %d\n", s.ime, s.prezime, s.fakultet, s.godina_rodjenja, s.stanje_na_xici);

	fclose(binarna);
	return 0;
}
```

RedListom – 4

Djelatnici menze su se odlučili našaliti sa studentima. Iznenada su zatvorili liniju u restoranu te otvorili susjednu. Naravno svi studenti su potrčali u novi red. Međutim u novom redu su prvi oni studenti koji su u starom redu bili posljednji. Premjestiti studente iz starog reda u novi na opisani način funkcijom prototipa:

void okreni(Red *red1, Red *red2);

koja ne smije koristiti pomoćen redove niti pomoćna polja (koristiti rekurziju). Napisati i glavni program koji će iz zadane datoteke učitati sve zapise o studentima, dodati ih u red te pozvati funkciju *okreni*. Naposlijetku skinuti studente iz novog reda te ispisati njihove podatke.

```
void okreni(Red *red1, Red *red2){
	student s;
	if(SkiniIzReda(&s, red1)){
		okreni(red1, red2);
		DodajURed(s, red2);
	}
}

int main(){
	FILE *binarna;
	student s;
	Red red1, red2;
	init_red(&red1); init_red(&red2);
	binarna=fopen("studenti.dat", "rb");

	printf("\n\t*** Prvi red ***\n\n");
	while(fread(&s, sizeof(student), 1, binarna)){
		DodajURed(s, &red1);
		printf("%s %s, %s, %d, %d\n", s.ime, s.prezime, s.fakultet, s.godina_rodjenja, s.stanje_na_xici);
	}

	okreni(&red1, &red2);

	printf("\n\t*** Drugi red ***\n\n");
	while(SkiniIzReda(&s, &red2))
		printf("%s %s, %s, %d, %d\n", s.ime, s.prezime, s.fakultet, s.godina_rodjenja, s.stanje_na_xici);

	fclose(binarna);
	return 0;
}
```

RedListom – 5

Nakon predavanja u menzu je došla skupina gladnih ferovaca čiji se podaci nalaze u svakom retku formatirane datoteke *ferovci.txt*. Zapisi su istog oblika kao i zapisi o studentima iz datoteke *studenti.dat*. Vesela skupina je ugledala svog kolega imenom i prezimenom: Fero Ferovcic. Napisati funkciju prototipa:

void preko_reda(FILE *ferovci, Red *red);

koja će ubaciti studente novopridošlu skupinu studenat u red iza kolege Ferovcica. Napisati i glavni program u kojem ćete pročitati sve zapise o studentima iz binarne datoteke *studenti.dat*, dodati ih u red te pozvati funkciju *preko_reda*.

```
void preko_reda(FILE *ferovci, Red *red){
	Red pomocni;
	student s1, s2;
	init_red(&pomocni);
	while(SkiniIzReda(&s1, red)){
		DodajURed(s1, &pomocni);
		if(strcmp(s1.ime, "Fero")==0 && strcmp(s1.prezime, "Ferovcic")==0){
			while(fscanf(ferovci, "%s %s %s %d %d", s2.ime, s2.prezime,
			s2.fakultet, &s2.godina_rodjenja, &s2.stanje_na_xici)!=EOF)
				DodajURed(s2, &pomocni);
		}
	}
	while(SkiniIzReda(&s1, &pomocni))
	DodajURed(s1, red);
}
int main(){
	FILE *binarna, *formatirana;
	student s;
	Red red;

	init_red(&red);
	binarna=fopen("studenti.dat", "rb");
	formatirana=fopen("ferovci.txt", "r");

	printf("\n\t*** Pocetni red ***\n\n");
	while(fread(&s, sizeof(student), 1, binarna)){
		DodajURed(s, &red);
		printf("%s %s, %s, %d, %d\n", s.ime, s.prezime, s.fakultet,
			s.godina_rodjenja, s.stanje_na_xici);
	}

	preko_reda(formatirana, &red);

	printf("\n\t*** Red nakon ubacivanja ***\n\n");
	while(SkiniIzReda(&s, &red))
		printf("%s %s, %s, %d, %d\n", s.ime, s.prezime, s.fakultet,
			s.godina_rodjenja, s.stanje_na_xici);

	fclose(binarna);
	fclose(formatirana);
	return 0;
}
```