1.1. Pokazivači

Pokazivači (*eng. pointers*) su tipovi podataka u koje se pohranjuje adresa neke druge varijable u radnoj memoriji. Zbog toga kažemo da pokazivači „pokazuju na“ dotičnu varijablu. Dohvaćanje vrijednosti na koju pokazuje pokazivač naziva se dereferenciranje.
Prije uporabe pokazivač je potrebno inicijalizirati (b=&a), tj. pridružiti mu adresu. U protivnom bi izazvali grešku u programu jer je svim pokazivačima na početku pridružena neka pretpostavljena adresa (npr. MS Visual Studio 2008 inicijalizira sve pokazivače na 0xcccccccc).
Razmjena podataka s funkcijom se može obaviti na dva načina: *call by value* – prenoseći vrijednost same varijable, odnosno call by reference – prenoseći adresu varijable pomoću pokazivača.
Pokazivačima pridružujemo adrese koje su u računalu predstavljene brojem (*long int*) pa je zato pokazivač bilo kojeg tipa veličine 4 bajta. Pokazivači mogu pokazivati na bilo koji tip podatka, također postoji i pokazivač na tip void. Takav pokazivač se ne može dereferencirati sve dok se ne pretvori (*cast operator*) u pokazivač na neki od standardnih tipova podataka.

Pokazivaci_1 - Napisati program koji će deklarirati varijablu *a* znakovnog tipa (*char*) te joj pridružiti znak *'a'*. Također deklarirati pokazivač na znakovni tip podatka te mu pridružiti *adresu varijable a.* Ispisati sadržaj, adresu i veličinu u bajtovima od: *a, b, *b*.

```
#include <stdio.h>

int main(){
	//definicija varijable
	char a='a';
	//deklaracija pokazivača
	char *b;
	//obavezno uvijek inicijaliziramo pokazivač
	//tako da mu pridružimo adresu
	b=&a;

	printf("Sadrzaj od a: %c\n", a);
	printf("Adresa od a: %p\n", &a); //%p --> ispis adrese
	printf("Velicina od a: %d bajt\n\n", sizeof(a));

	printf("Sadrzaj od b: %p\n", b);
	printf("Adresa od b: %p\n", &b);
	printf("Velicina od b: %d bajta\n\n", sizeof(b));

	printf("Sadrzaj od *b: %c\n", *b);
	printf("Adresa od *b: %p\n", &(*b));
	printf("Velicina od *b: %d bajt\n\n", sizeof(*b));

	return 0;
}
```

Proučiti ispis programa te obratiti pažnju na sljedeće stvari:

a) sadržaj pokazivača b je jednak adresi varijable a
b) *b je „dereferencirani“ pokazivač on nam daje vrijednost varijable koja je pohranjena na adresi na koju pokazuje pokzivač b
c) pokazivačima pridružujemo adrese koje su u računalu predstavljene brojem (*long int*) pa je zato pokazivač bilo kojeg tipa veličine 4 bajta.

Pokazivaci_2 - Definirati pokazivač koji pokazuje na prvi znak niza znakova „EDQ“ te pomoću njega i jedne *for petlje* ispisati sve znakove niza, ali uvećane za 1. Npr. 'E' je ASCII znak predstavljen u računalu kao broj 69. Ako 69 uvećamo za 1 dobit ćemo ASCII znak 70 što predstavlja znak 'F'.

```
#include <stdio.h>

int main(){
	char *p;
	//p pokazuje na prvi znak niza znakova "EDQ"
	//petlja se vrti dok ne naiđe na NULL character, tj. kraj niza
	for(p = "EDQ"; *p; p++)
		//ispisuje vrijednost, koja je zapisana na adresi na koju pokazuje p,
		//uvećanu za 1 u obliku ASCII znaka
		printf("%c\n", *p + 1);
	return 0;
}
```

Pokazivaci_3 - Napisati dvije funkcije čiji su prototipovi:

```
int strlen_pokazivaci(char *str);
int strlen_polje(char str[]);
```

Funkicija *strlen_pokazivaci* mora preko imena vratiti duljinu niza znakova *str* koristeći aritmetiku s pokazivačima. Funkciju *strlen_polje* napisati bez korištenja pokazivača, ona također mora preko imena vratiti duljinu znakovnog niza *str* te će nam služiti za provjeru rezultata. Napisati također i glavni program u kojem definirajte znakovni niz, pozovite obje funkcije te usporedite rezultat.

```
#include <stdio.h>

//pokazivač str pokauje na prvi član niza znakova
//petlja se vrti dok vrijednost na adresi na koju pokazuje str nije NULL
int strlen_pokazivaci(char *str){
	int duljina=0;
	for(; *str; str++)
		duljina++;
	return duljina;
}

int strlen_polje(char str[]){
	int duljina=0, i;
	for(i=0; str[i]; i++)
		duljina++;
	return duljina;
}

int main(){
	char niz[100+1];
	int duljina;
	printf("Zadajte niz znakova: ");
	scanf("%s", niz);

	duljina=strlen_pokazivaci(niz);
	printf("Duljina niza (strlen_pokazivaci): %d\n", duljina);

	duljina=strlen_polje(niz);
	printf("Duljina niza (strlen_polje): %d\n", duljina);

	return 0;
}
```

Pokazivaci_4 - Napisati dvije funkcije čiji su prototipovi:

```
void zamijeni_varijabla(int prvi, int drugi);
void zamijeni_pokazivaci(int *prvi, int *drugi);
```

Funkciji *zamijeni_varijable* se predaju varijable *prvi* i *drugi*, dok se funkciji *zamijeni_pokazivaci* predaju adrese varijabla *prvi* i *drugi*. Napisati glavni program u kojem treba definirati varijable *prvi* i *drugi* te na odgovarajući način pozvati obje funkcije. Uvjeriti se da su promjene napravljene nad varijablama *prvi* i *drugi* u funkciji *zamijeni_varijable* samo lokalne te se ne vide u glavnom programu.

```
#include <stdio.h>

//call by value, promijene su lokalne i ne vide se u glavnom programu
void zamijeni_varijable(int prvi, int drugi){
	int privremena;
	privremena=prvi;
	prvi=drugi;
	drugi=privremena;
}
//call by reference, promjene su vidljive i u glavnom programu
void zamijeni_pokazivaci(int *prvi, int *drugi){
	int privremena;
	privremena=*prvi;
	*prvi=*drugi;
	*drugi=privremena;
}

int main(){
	int prvi=5, drugi=10;

	zamijeni_varijable(prvi, drugi);
	printf("Nakon poziva zamijeni_varijabla, prvi=%d, drugi=%d\n", prvi, drugi);

	zamijeni_pokazivaci(&prvi, &drugi);
	printf("Nakon poziva zamijeni_pokazivaci, prvi=%d, drugi=%d\n", prvi, drugi);

	return 0;
}
```

Pokazivaci_5 - Napisati funkciju prototipa:

```
int usporedi_memoriju(void *m1, void *m2, int n)
```

koja će počevši od adresa u memoriji *m1* i *m2* usporediti bajt po bajt, tj. provjeriti jednakost sadržaja na tim adresama. Usporediti *n* bajtova. Napisati i glavni program u kojem ćete deklarirati dvije varijable *a* i *b* tipa int i pridružiti im vrijednost 5. Deklarirati dva pokazivača i pridružiti im adrese varijabli *a* i *b*, pretvoriti ih u tip (*void **) i predati funkciji *usporedi_memoriju*. Funkciju pozvati dva puta, jednom za n=4 i jedno za n=32.

1.2. Prenošenje polja u funkciju

Jednodimenzionalno polje prenosimo u funkciju preko pokazivača – potrebno je prenjeti pokazivač na prvi element polja te broj elemenata polja (*n*), npr.:

```
int funkcija(int *polje, int n);
```

Pojedini element polja u funkciji dobivamo dereferenciranjem i uvećavanjem pokazivača za cijeli broj (*polje+broj preskočenih elemenata*), npr.:

`*(polje+i)` ili `polje[i]`;

a funkciju iz glavnog programa pozivamo:

```
funkcija(polje, n);
```

Bitno je uočiti način na koji se prenosi pokazivač na prvi element polja:

`&polje[0]` ili samo `polje` (*jer je ime polja ujedno i pokazivač na prvi element*)

Znakovni niz u memoriji je predstavljen jednodimenzionalnim polje pa vrijedi isti način prenošenja u funkciju i poziva funkcije iz glavnog programa. Razlika je jedino u tome što nije potrebno prenjeti duljinu niza jer je znakovni niz terminiran sa '\0' (*null characterom*). Duljinu niza (*bez '\0'*) tada lako možemo dobiti funkcijom:

```
size_t strlen(const char *s);
```

koja se nalazi u zaglavlju `<string.h>`.

Dvodimenzionalno polje u memoriji računala također je predstavljeno jednodimenzionalnim, pri čemu pojedini retci slijede jedan iza drugoga. Prototip funkcije je npr.:

```
int funkcija(int *polje, int m, int n, int MAXSTUP);
```

Funkciji je potrebno prenjeti pokazivač na prvi element polja, broj redaka, broj stupaca i maksimalni broj stupaca polja (kako bismo znali gdje počinje novi redak).

Pojedini element polja u funkciji dobivamo dereferenciranjem i uvećavanjem pokazivača za cijeli broj (polje+broj preskočenih redaka+broj preskočenih stupaca u trenutnom redu), npr.:

`*(polje+i*MAXSTUP+j)` ili `polje[i*MAXSTUP+j];`

a funkciju iz glavnog programa pozivamo:

```
funkcija(polje, m, n, MAXSTUP);
```

Bitno je uočiti način na koji se prenosi pokazivač na prvi element polja:

`&polje[0][0]` ili samo `polje` (*jer je ime polja ujedno i pokazivač na prvi element*)

Prenosenje_1 - Napisati funkciju koja će u zadanom cjelobrojnom jednodimenzionalnom polju zbrojiti sve neparne elemente, a sve parne postaviti na 0. Napisati funkciju koja će ispisati to polje te glavni program u kojem će se deklarirati polje i pozvati funkcije. Ispisati polje prije i nakon promjene.

```
#include <stdio.h>

//funkcija koja zbraja sve neparne elemente jednodimenzionalnog
//polja, a pozitivne postavlja na 0, prenosimo broj elemenata polja
//i pokazivač na prvi element polja
```

```c
int funkcija (int *polje, int n){
	int suma=0, i;
	for (i=0; i<n; i++){
		if((*(polje+i))%2)
			suma+=*(polje+i);
		else
			*(polje+i)=0;
	}
	return suma;
}
//funkcija koja ispisuje elemente polja jedan za drugim, prenosimo
//broj elemenata polja i pokazivač na prvi element polja
void ispisi_polje(int *polje, int n){
	int i;
	printf("\nIspis polja: ");
	for(i=0; i<n; i++)
		printf("%d ", polje[i]);
	printf("\n\n");
}

int main(){
	int suma, i, n;
	int polje[100]={0};
	printf("Unesite broj elemenata polja: ");
	scanf("%d", &n);
	printf("Unesite elemente polja [0,9]: ");
	for(i=0; i<n; i++)
		scanf("%d", &polje[i]);

	ispisi_polje(polje, n);
	suma=funkcija(polje,n);
	printf("Zbroj neparnih elemenata polja je: %d\n", suma);
	ispisi_polje(polje, n);
	return 0;
}
```

Prenosenje_2 - Napisati funkciju koja u zadanom nizu znakova mijenja velika slova u mala te vraća broj promijenjenih slova. Također, napisati i glavni program u kojem će se pozvati funkcija sa proizvoljnim nizom znakova te ispisati niz prije i poslije promjene.

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

//funkcija koja mijenja velika slova u mala, niz prenosimo
//preko pokazivača, nije potrebno prenjeti duljinu znakovnog
//niza, koristimo ugrađene funkcije definirane u zaglavlju
//string.h i ctype.h
int zamijeni_velike(char *niz){
	int brojac=0, i;

	for(i=0; i<strlen(niz); i++){
		if(isupper(niz[i])!=0){
			niz[i]=tolower(niz[i]);
			brojac++;
		}
	}
	return brojac;
}

int main(){
```

```
	char niz[30+1]="ABraKadBRa!";
	int promijenjeni;

	printf("Pocetni niz: %s\n", niz);
	promijenjeni=zamijeni_velike(niz);
	printf("Broj promijenjenih slova: %d\n", promijenjeni);
	printf("Promijenjeni niz: %s\n", niz);
	return 0;
}
```

Prenosenje_3 - Napisati funkciju *suma_parni* koja će u dvodimenzionalnom polju cijelih brojeva zbrojiti sve parne elemente te ih postaviti na nulu. Funkcija treba vratiti preko imena zbroj svih parnih članova. Napisati funkciju za ispis dvodimenzionalnog polja. U glavnom programu definirati dvodimenzionalno polje cijelih brojeva, pozvati funkciju *suma_parni* te ispisati matricu prije i poslije promjene.

```
#include <stdio.h>

//kad prenosimo 2D polje u funkciju moramo prenijeti:
//pokazivač na prvi član matrice, broj redak, broj stupaca i maksimalan broj
stupaca
//elementu u retku i i stupcu j pristupamo sa: *(matrica+i*maxstup+j);
int suma_parni(int *matrica, int m, int n, int maxstup){
	int suma=0, i, j;
	for(i=0; i<m; i++){
		for(j=0; j<n; j++){
			if(*(matrica+i*maxstup+j)%2==0){
				suma+=*(matrica+i*maxstup+j);
				*(matrica+i*maxstup+j)=0;
			}
		}
	}
	return suma;
}

void ispisi_matricu(int *matrica, int m, int n, int maxstup){
	int i, j;
	for(i=0; i<m; i++){
		for(j=0; j<n; j++)
			printf("%3d", *(matrica+i*maxstup+j));
		printf("\n");
	}
	printf("\n");
}

int main() {
	int suma;
	int matrica[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};

	ispisi_matricu(matrica, 3, 4, 4);
	suma=suma_parni(matrica, 3, 4, 4);
	printf("Suma parnih elemenata matrice: %d\n\n", suma);
	ispisi_matricu(matrica, 3, 4, 4);

	return 0;
}
```

1.3. Dinamička rezervacija memorije

Dinamička rezervacija memorije u programskom jeziku C se obavlja funkcijama:

*void *malloc(size_t size);*

*void *realloc(void *pointer, size_t size);*

a oslobađanje funkcijom:

*void free(void *pointer);*

Navedene funkcije nalaze se u zaglavlju *malloc.h*.

Ako malloc ili realloc ne uspiju zauzeti memoriju vraćaju NULL.

Realloc također možemo koristit kao malloc ako kao parametar funkcije (*void *pointer*) navedemo NULL. Kad se ukaže potreba za zauzimanjem dodatne memorije preporučuje se zauzeti dvostruko veći blok. Izlaskom iz funkcije, zauzeta memorija se ne oslobađa te ju je vrlo važno oslobodit pomoću funkcije *free*.

Dinamicka_1 - Napisati program koji će sa tipkovnice učitati početni broj elemenat cjelobrojnog polja kojeg treba popuniti slučajnim brojevima iz intervala [0-9]. Omogućiti korisniku da naknadno dinamički poveća polje te ga svaki put popuniti slučajnim brojevima iz istog intervala. Kad se unese 0 program treba završiti.

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

int main() {
	int *polje;
	int velicina, n, i;
	//inicijaliziramo generator pseudoslučajnih brojeva
	srand((unsigned)time(NULL));

	printf("Unesite zeljenu velicinu polja: ");
	scanf("%d", &velicina);
	//rezerviramo memorijski blok
	polje=(int *)malloc(velicina*sizeof(int));
	printf("Polje ima %d elemenata: ", velicina);
	for(i=0; i<velicina; i++) {
		//popunjavamo polje slučajnim vrijednostima iz intervala [0,9]
		polje[i]=rand()%10;
		printf("%d ", polje[i]);
	}
	printf("\n\n");

	do {
		printf("Za koliko elemenata zelite povecati polje(0 - prekid): ");
		scanf("%d", &n);
		velicina+=n;
		//povećavamo memorijski blok prethodno rezerviran mallocom
		polje=(int *)realloc(polje, velicina*sizeof(int));
		printf("Nakon povecanja polje ima %d elemenata: ", velicina);
		for(i=0; i<velicina; i++) {
```

```
                        //popunjavamo polje slučajnim vrijednostima iz intervala [0,9]
                        polje[i]=rand()%10;
                        printf("%d ", polje[i]);
                }
                printf("\n\n");
        } while(n!=0);

        //obavezno oslobađamo zauzetu memoriju
        free(polje);
        return 0;
}
```

Dinamicka_2 - Napisati program koji će s tipkovnice učitati niz znakova, pretpostaviti da zadani niz nije veći od 10 znakova. Dinamički zauzeti potrebnu memoriju te u nju spremiti palindrom koji je nastao od zadanog niza pa ga ispisati na zaslon (*obratiti pažnju na završni znak*). Npr. zadani niz: „banana“ → Ispis: „bananananab“

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

int main() {
        char niz[10+1];
        char *polje;
        int i;
        int velicina=0, nova_velicina;

        scanf("%s", niz);

        for(i=0; niz[i]!=0; i++) //ili velicina=strlen(niz);
                velicina++;

        nova_velicina=2*velicina; //velicina potreban za novi niz znakova

        polje=(char *)malloc(nova_velicina*sizeof(char)); //zauzimanje memorije

        //prolazimo kroz petlju toliko puta koliko početni niz ima elemenata
        //te istodobno upisujemo znakove u novi niz na početak i na kraj
        for (i=0; i<velicina; i++)
                polje[i]=polje[nova_velicina-2-i]=niz[i];

        polje[nova_velicina-1]='\0'; //terminiranje niza
        printf("Palindrom: %s\n", polje);
        free(polje); //obavezno osloboditi memoriju
        return 0;
}
```

Dinamicka_3 - Napišite funkciju čiji je prototip:

```
int *prosiri(int *polje, int *br_clanova)
```

koja će proširiti ulazno polje od br_clanova tako da na kraj nadopiše sve brojeve iz početnog polja koji su djeljivi s 3. Funkcija preko imena vraća pokazivač na novo polje te broje elemenata novonastalog polja. Napisati i glavni program u kojem će se stvoriti početno polje veličine 10 elemenata te ga popuniti slučajnim brojevima iz intervala [100, 200]. Glavni program mora pozvati funkciju, ispisati novonastalo polje te osloboditi dinamički stvorenu memoriju.

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

```c
int *prosiri(int *polje, int *br_clanova){
	int i, j=*br_clanova, novi_br_clanova=*br_clanova;
	//brojimo koliko ima elemenata početnog polja djeljivih s 3
	for(i=0; i<*br_clanova; i++)
		if(polje[i]%3==0)
			novi_br_clanova++;

	//zauzimamo dodatni prostor
	polje=(int*)realloc(polje, (novi_br_clanova)*sizeof(int));

	//dodajemo članove koji zadovoljavaju uvjet na kraj polja
	for(i=0; i<*br_clanova; i++)
		if (polje[i]%3==0)
			polje[j++]=polje[i];

	//ažuriramo broj članova polja te vraćamo pokazivač na polje
	*br_clanova=novi_br_clanova;
	return polje;
}

int main(){
	int *polje, i, br_clanova = 10, n=5;
	//zauzimamo memorije
	polje=(int*)malloc(br_clanova*sizeof(int));
	//inicijaliziramo generator pseudoslučajnih brojeva
	srand((unsigned) time(NULL));
	for(i=0; i<br_clanova; i++)
		//rand()%(gornji-donji+1)+donji
		polje[i]=rand()%(200-100+1)+100;

	printf("\nPrije prosirenja\n");
	for(i=0; i<br_clanova; i++)
		printf("%d ", polje[i]);

	polje=prosiri(polje, &br_clanova);

	printf("\nNakon prosirenja\n");
	for(i=0; i<br_clanova; i++)
		printf("%d ", polje[i]);

	printf("\n");
	//obavezno oslobađamo memoriju
	free(polje);
	return 0;
}
```

Dinamicka_4- Napišite funkciju čiji je prototip:

```
int *k
```

koja će stvoriti novo polje koje će sadržavati sve elemente početnog polja veće od 25. Broj elemenata u početnom polju je br1. Funkcija kao rezultat vraća pokazivač na navedeno polje. Ako u početnom polju nema takvih elemenata funkcija vraća NULL. Također, napisati glavni program u kojem će se, stvoriti polje veličine 10 elemenata iz intervala [0, 49], pozvati funkcija, ispisati polje te osloboditi dinamički stvorena memorija.

```c
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <time.h>

int *kopiraj(int *polje, int br1, int *br2){
	int i, j=0;
	int *novo;
	*br2=0;
	//brojimo koliko ima elemenata početnog polja većih od 25
```

```
	for(i=0; i<br1; i++){
		if (polje[i]>25)
		(*br2)++;
	}
	//punimo novo polje elementima početnog koji su veći od 25
	//ako takvi postoje te vraćamo pokazivač na novo polje, inače NULL
	if(*br2>0){
		//zauzimamo memoriju
		novo=(int*)malloc((*br2)*sizeof(int));
		for(i=0; i<br1; i++){
			if(polje[i]>25)
				novo[j++]=polje[i];
		}
		return novo;
	}
	else
		return NULL;
}

int main(){
	int polje[10], *novo, i, br2, br_clanova = 10;
	//inicijaliziramo generator pseudoslučajnih brojeva
	srand((unsigned)time(NULL));

	//inicijaliziramo polje slučajnim brojevima iz intervala [0,49]
	for(i=0; i<br_clanova; i++)
		polje[i]=rand()%50;

	printf("\nPrvo polje\n");
	for(i=0; i<br_clanova; i++)
		printf("%d ", polje[i]);
	printf("\n");

	novo=kopiraj(polje, br_clanova, &br2);

	if (novo!=NULL){
		printf("\nDrugo polje\n");
		for(i=0; i<br2 ; i++)
			printf("%d ", novo[i]);
		printf("\n");
		//obavezno oslobođamo memoriju
		free(novo);
	}

	return 0;
}
```

Dinamicka_5 - Na disku postoji formatirana datoteka *„studenti.txt“*. U prvom retku datoteke nalazi se cjelobrojni podatak o tome koliko ima zapisa u datoteci. Svaki zapis datoteke sastoji se od matičnog broj, imena i prezimena studenta te broja bodova osvojenih iz ASP-a na kraju semestra. Učitati sve zapise iz datoteke u dinamički alocirano polje, ispisati na zaslon koliko je ljudi položilo ovaj predmet (bodovi>=50) te ispisati podatke o najuspješnijem studentu. Obavezno provjeriti uspješnost otvaranje datoteke te uspješnost zauzeća memorije.

1.4. Pseudoslučajni brojevi

Generator pseudoslučajnih brojeva inicijalizira se funkcijom iz standardne biblioteke <stdlib.h>:

void srand (unsigned int seed);

Za istu vrijednost sjemena (*seed value*) generiraju se jednaki nizovi pseudoslučajnih brojeva. Zbog toga se kao sjeme najčešće uzima vrijednost koju vraća funkcija :

*time_t time(time_t *timer);*

deklarirana u zaglavlju *<time.h>*, npr:

srand((unsigned)time(NULL));

Kako funkcija `time` vraća broj sekundi proteklih od 00:00:00, 1. siječnja 1970. ta se vrijednost razlikuje svake sekunde.

Pseudoslučajne brojeve generira funkcija:

int rand(void);

također deklarirana u zaglavlju *<stdlib.h>*.

Ako trebamo generirati brojeve iz intervala [donja_vrijednost, gornja_vrijednost] to možemo napravit na sljedeći način:

rand()%(gornja_vrijednost-donja_vrijednost+1)+donja_vrijednost;

Pseudoslucajni_1 - Napraviti program koji će pomoću generatora pseudoslučajnih brojeva simulirati izvlačenje brojeva lota 7/39. Program mora izgenerirati 7 različitih brojeva te jedan dopunski broj iz intervala [1, 39].

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(){
	int broj, i;
	//polje u kojem će 0 označavati da se broj i na i-toj poziciji
	//još nije pojavio, a 1 označava da je taj broj već izvučen
	int pojavljivanja[40]={0};
	//inicijaliziramo generator pseudoslučajnih brojeva
	srand((unsigned)time(NULL));

	for(i=0; i<8; i++){
		//interval [1,39] → broj=rand()%(gornji-donji+1)+donji
		broj=rand()%(39-1+1)+1;
		//ako se broj još nije pojavio ispišemo ga i zapamtimo da se pojavio
		if(!pojavljivanja[broj]){
			if(i<7)
				printf("%d. izvuceni broj: %d\n", i+1, broj);
			else if(i==7)
				printf("\nDopunski broj: %d\n", broj);
			pojavljivanja[broj]=1;
		}
		//ako se taj broj već pojavio, moramo izvući neki drugi pa smanjujemo
		// brojač petlje za 1, tj. ponavljamo korak petlje
```

```
                else{
                        i--;
                        continue;
                }
        }
        return 0;
}
```

Pseudoslucajni_2 - Napisati funkciju koja unutar zadanog dvodimenzionalnog cjelobrojnog polja definiranog s proizvoljnim dimenzijama, generira matricu od zadanih m redaka i zadanih n stupaca. Elemente matrice funkcija postavlja na slučajne vrijednosti iz zatvorenog intervala [20,25] te preko imena vraća sumu elemenata na rubovima matrice. Napisati glavni program koji će definirati polje dimenzija 20×25, popuniti polje nulama i pomoću funkcije generirati matricu proizvoljnih dimenzija m×n. U glavnom programu ispisati tako dobivenu matricu i sumu elemenata na rubovima matrice.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int generiraj_matricu(int *polje, int maxstup, int m, int n){
        //dvodimenzionalno polje u memoriji je predstavljeno kao niz
        //jednodimenzionalnih, zbog toga, kada ga prenosimo u funkciju
        //obavezno moramo prenjeti pokazivač na prvi element polja, maksimalni
        //broj stupaca, broj redaka te broj stupacxa polja
        int i, j;
        int suma=0;
        //inicijaliziramo generator pseudoslučajnih brojeva
        srand((unsigned)time(NULL));
        //punimo elemente zadane matrice slučajnim brojevima iz intervala
        //[20,25] i zbrajamo elemente na rubu matrice (i=0 ili i=m-1 ili
        //j=0 ili j=n-1)
        for (i=0; i<m; i++){
                for (j=0; j<n; j++){
                        *(polje+i*maxstup+j)=rand()%(25-20+1)+20;
                        if((i==0)||(i==m-1)||(j==0)||(j==n-1)){
                                suma+=*(polje+i*maxstup+j);
                        }
                }
        }
        //preko imena funkcije vraćamo sumu elemenata na rubu matrice
        return suma;
}
int main() {
        int i,j,m,n,suma;
        //punimo polje nulama
        int polje[20][25]={0};
        printf("Unesite broj redaka m:");
        scanf("%d", &m);
        printf("Unesite broj stupaca n:");
        scanf("%d", &n);
        //pozivamo funkciju i prenosimo joj pokazivač na prvi element
        //polja, maksimalni broj stupaca, broj redaka i broj stupaca polja
        suma=generiraj_matricu(polje, 25, m, n);//ili &polje[0][0]
        //ispisujemo elemente matrice
        for(i=0; i<m; i++){
                for(j=0; j<n; j++){
                        printf("%d ", polje[i][j]);
                }
```

```
			printf("\n");
		}
		printf("Suma elemenata na rubu matrice je: %d\n", suma);
		return 0;
}
```

Pseudoslucajni_3 - Napisati funkciju prototipa:

```
int najcesci(int n, int *broj_pojavljivanja);
```

koja će generirati n brojeva iz intervala [0,99] te preko imena vratiti koji se broj najčešće pojavljivao, a preko adrese koliko puta. Također napisati i glavni program koji će pozvati funkciju.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int najcesci(int n, int *broj_pojavljivanja){
	int i, broj;
	int najcesci=0, pozicija;
	//matrica pojavljivanja, inicijaliziramo je nulama
	//svako pojavljivanje broja i povećava element na i-tom mjestu matrice za 1
	int pojavljivanja[100]={0};
	//inicijaliziramo generator pseudoslučajnih brojeva
	srand((unsigned)time(NULL));
	//generiramo brojeve iz intervala [0,99] i pamtimo njihova pojavljivanja
	for(i=0; i<n; i++){
		broj=rand()%100;
		pojavljivanja[broj]++;
	}
	//indeks(pozicija)najvećeg elementa u matrici pojavljivanja nam govori koji
	// broj se najčešće pojavljivao, a vrijednost tog elementa nam govori koliko
	 //puta se taj broj pojavio
	for(i=0; i<100; i++)
		if(pojavljivanja[i]>najcesci){
			najcesci=pojavljivanja[i];
			pozicija=i;
		}
	//preko pokazivača vraćamo broj pojavljivanja, a preko imena funkcije
	//vraćamo broj koji se najviše puta pojavio
	*broj_pojavljivanja=pojavljivanja[pozicija];
	return pozicija;
}

int main(){
	int n;
	int najcesci_broj, broj_pojavljivanja;
	printf("Koliko brojeva zelite generirati: ");
	scanf("%d", &n);
	najcesci_broj=najcesci(n, &broj_pojavljivanja);
	printf("Najvise puta se pojavio broj %d (%d puta)!\n", najcesci_broj,
	 broj_pojavljivanja);
	return 0;
}
```

Pseudoslucajni_4 - Napisati funkciju prototipa:

```
int sasvim_slucajno(char rijec[], int duljina, char
zabranjeni[]);
```

koja će generirati riječ zadane duljine sastavljenu od slučajno odabranih malih slova engleske abecede [a, z]. Ako je generirano slovo iz skupa zabranjenih slova zamijeniti ga sa znakom 'X'. Napisati i glavni program u kojem će se zadati duljina riječi, zabranjeni niz te pozvati funkciju.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int sasvim_slucajno(char rijec[], int duljina, char zabranjeni[]){
	int i, j, zabranjen, ukupno=0;
	char slovo;
	//inicijaliziramo generator pseudoslučajnih brojeva
	srand((unsigned)time(NULL));
	for(i=0; i<duljina; i++){
		zabranjen=0;
		//generiramo slova iz intervala [a,z]
		//→ slovo=rand()%(gornji-donji+1)+donji
		slovo=rand()%('a'-'z'+1)+'a';
		for(j=0; zabranjeni[j]!='\0'; j++){ //ili zabranjeni[j]
			if(slovo==zabranjeni[j]) {
				zabranjen=1;
				ukupno++;
			}
		}
		if(zabranjen)
			rijec[i]='X';
		else
			rijec[i]=slovo;
	}
	//stavljamo oznaku za kraj niza
	rijec[duljina]='\0';
	return ukupno;
}

int main(){
	char rijec[10+1];
	char zabranjeni[10+1];
	int promasaja;

	printf("Unesite zabranjene znakove: ");
	scanf("%s", zabranjeni);

	promasaja=sasvim_slucajno(rijec, 10, zabranjeni);

	printf("Dobivena rijec: %s\n\n", rijec);
	printf("Promasaja: %d\n", promasaja);
	return 0;
}
```

Pseudoslucajni_5 - Napisati funkciju koja u zadano dvodimenzionalno cjelobrojno polje "ispaljuje zadani broj hitaca". Element polja koji funkcija gađa izabire se slučajno, a njegova se vrijednost kod pogotka uvećava za 1. Također, napisati i glavni program koji će zadati dimenzije polja, broj hitaca, postaviti vrijednosti elemenata polja na 0 te pozvati funkciju. Pomoću funkcije prototipa:

```
void ispisiPolje(int *polje, int m, int n, int maxstup);
```

koja će biti pozvana iz glavnog programa ispisati polje dobiveno nakon gađanja.

1.5. Datoteke

Razlikujemo dvije vrste datoteka: formatirane i binarne. Ovisno o tome koristimo određeni skup osnovnih fukcija za rad nad njima.

Svaku datoteku prije korištenja moramo otvoriti. Za otvaranje potreban nam je pokazivač na opisnik datoteke (*FILE *stream*). Koristimo funkciju:

```
FILE *fopen(const char *filename, const char *mode);
```

koja vraća pokazivač na opisnik otvorene datoteke ili NULL u slučaju pogreške. Drugi parametar funkcije (mode) određuje način na koji datoteku želimo otvoriti. Parametar može biti:

`"w"`	pisanje (ako datoteka ne postoji, stvara se; ako postoji, njen se sadržaj briše; nije dopušteno čitanje)
`"a"`	pisanje (ako datoteka ne postoji, stvara se; ako postoji podaci se dodaju na kraj; nije dopušteno čitanje)
`"r"`	čitanje (ako datoteka ne postoji, vraća NULL; nije dopušteno pisanje)
`"w+"`	čitanje i pisanje (ako datoteka ne postoji, stvara se)
`"a+"`	čitanje i pisanje (ako datoteka ne postoji, stvara se; podaci se dodaju na kraj)
`"r+"`	čitanje i pisanje (ako datoteka ne postoji, vraća NULL)

Ako se otvara binarna datoteka još se dodaje slovo b, npr. *"rb"* ili *"w+b"*.

Za rad s formatiranim datotekama koristimo sljedeće funkcije:

```
int fscanf(FILE *stream, const char *format,  arg1, arg2, ..., argn);
int fprintf(FILE * stream, const char *format, arg1, arg2, ..., argn);,
```

dok za binarne koristimo:

```
size_t fread(void *ptr, size_t size, size_t n, FILE *stream);
size_t fwrite(void *ptr, size_t size, size_t n, FILE *stream);
```

Ukoliko je binarna datoteka direktna, za pozicioniranje koristimo funkciju:

```
int fseek(FILE *stream, long offset, int whence);
```

Whence može biti:

SEEK_SET	pozicioniranje u odnosu na početak datoteke
SEEK_CUR	pozicioniranje u odnosu na trenutnu poziciju u datoteci
SEEK_END	pozicioniranje u odnosu na kraj datoteke

Želimo li saznati trenutnu poziciju u datoteci, upotrijebit ćemo:

```
long ftell(FILE *stream);
```

Po završetku rada s datotekom obavezno ju treba zatvoriti naredbom:

```
int fclose(FILE *stream);
```

Datoteke_1 - Korisnik sa tipkovnice unosi željeni broj zapisa o kobasicama. Svaki zapis sastoji se od: *šifra* (cijeli broj), *naziv* (20+1 znak), *broj_komada* (cijeli broj) i *cijena* (realan broj standardne preciznosti). Ukoliko kobasice pod određenom šifrom nema u trgovini, njenu šifru postaviti na 0. Iz unesenih podataka stvoriti formatiranu i binarnu datoteku. Zapise jedan za drugim ispisati na ekran.

```
#include <stdio.h>

int main() {
	FILE *binarna_datoteka, *formatirana_datoteka;
	//za čitanje i pisanje kod binarnih datoteka koristimo strukture
	struct k {
		int sifra;
		char naziv[20+1];
		int komada;
		float cijena;
	} kobasa;

	int n, i;

	printf("Koliko zapisa zelite unjeti: ");
	scanf("%d", &n);

	//otvaramo datoteke, one još ne postoje pa se stvaraju nove
	binarna_datoteka=fopen("kobasice.dat", "rb");
	formatirana_datoteka=fopen("kobasice.txt", "w");

	printf("***Sifra naziv komada cijena***\n\n");

	 //s tipkovnice unosimo zapise i upisujemo ih u binarnu datoteku (stvaramo
	 //binarnu datoteku)
	for(i=0; i<n; i++) {
		scanf("%d %s %d %f", &kobasa.sifra, kobasa.naziv, &kobasa.komada,
			&kobasa.cijena);
		fwrite(&kobasa, sizeof(kobasa), 1, binarna_datoteka);
	}

	//pozicioniramo se na početak binarne datoteke
	fseek(binarna_datoteka, 0, SEEK_SET);
	//u petlji čitamo zapise binarne datoteke, ispisujemo ih na ekran, te
	//upisujemo u formatiranu datoteku (stvaramo formatiranu datoteku)
	printf("\n\n\t***ISPIS***\n");
	for(i=0; i<n; i++) {
		fread(&kobasa, sizeof(kobasa), 1, binarna_datoteka);
		printf("%d %s %d %.2f\n", kobasa.sifra, kobasa.naziv, kobasa.komada,
			kobasa.cijena);
		fprintf(formatirana_datoteka, "%d %s %d %.2f\n", kobasa.sifra,
			kobasa.naziv, kobasa.komada, kobasa.cijena);
	}

	//obavezno zatvaramo obje datoteke
	fclose(binarna_datoteka);
	fclose(formatirana_datoteka);
	return 0;
}
```

Datoteke_2 - Na disku postoji direktna binarna datoteka „kobasice.dat“ koja sadrži podatke o kobasicama u lokalnoj trgovini: *šifra* (cijeli broj), *naziv* (20+1 znak), *broj_komada* (cijeli broj) i *cijena* (realan broj standardne preciznosti). Ukoliko zapis ne postoji njegova šifra je 0.

Prebrojiti sve kobasice u trgovini i pronaći njihovu ukupnu vrijednost.

```c
#include <stdio.h>

int main() {
	FILE *fp;
	//za čitanje i pisanje kod binarnih datoteka koristimo strukture
	struct k {
		int sifra;
		char naziv[20+1];
		int komada;
		float cijena;
	} kobasa;

	int ukupno_kobasica=0;
	float ukupna_cijena=0;

	//otvaramo binarnu datoteku("rb") i provjeravamo da li je uspješno otvorena
	if((fp=fopen("kobasice.dat", "rb"))==NULL) {
		printf("Ne mogu otvoriti datoteku!\n");
		exit(1);
	}
	//u petlji čitamo svaki zapis datoteke jedan po jedan
	//i povećavamo ukupan broj komada i ukupnu cijenu
	while(fread(&kobasa, sizeof(kobasa), 1, fp)==1) {
		ukupno_kobasica+=kobasa.komada;
		ukupna_cijena+=(kobasa.cijena*kobasa.komada);
	}

	printf("Ukupno komada kobasica: %d\n", ukupno_kobasica);
	printf("Ukupna cijena kobasica: %.2f\n", ukupna_cijena);

	//obavezno zatvaramo datoteku
	fclose(fp);
	return 0;
}
```

Datoteke_3 - Na disku postoji direktna binarna datoteka „kobasice.dat“ koja sadrži podatke o kobasicama u lokalnoj trgovini: *šifra* (cijeli broj), *naziv* (20+1 znak), *broj_komada* (cijeli broj) i *cijena* (realan broj standardne preciznosti). Ukoliko zapis ne postoji njegova šifra je 0. U svakom retku formatirane datoteke „nove_cijene.txt“ nalazi se šifra kobasice i nova cijena. Ažurirati podatke u binarnoj datoteci te ispisati podatke o kobasicama čija je cijena promijenjena.

```c
#include <stdio.h>

int main(){
	FILE *binarna_datoteka;
	FILE *formatirana_datoteka;
	//za čitanje i pisanje kod binarnih datoteka koristimo strukture
	struct k {
		int sifra;
		char naziv[20+1];
		int komada;
		float cijena;
	}kobasa;
	//a kod formatiranih pojedinačne varijable
	int broj;
	float nova_cijena;
	//otvaramo datoteke te za svaku provjerimo da li je uspješno otvorena
	if((binarna_datoteka=fopen("kobasice.dat","r+b"))==NULL){
```

```
		printf("Ne mogu otvoriti \"kobasice.dat\"!\n");
		exit(1);
	}
	if((formatirana_datoteka=fopen("nove_cijene.txt","r"))==NULL){
		printf("Ne mogu otvoriti \"nove_cijene.txt\"!\n");
		exit(1);
	}
	//u petlji čitamo zapise formatirane datoteke dok oni postoje, prema sifri
	//trazimo zapis u binarnoj datoteci i ažuriramo cijenu
	while(fscanf(formatirana_datoteka, "%d %f", &broj, &nova_cijena)==2){
		fseek(binarna_datoteka, (long)(broj-1)*sizeof(kobasa), SEEK_SET);
		fread(&kobasa, sizeof(kobasa), 1, binarna_datoteka);
		kobasa.cijena=nova_cijena;
		printf("Promjena cijene: %d %s %d %.2f\n", kobasa.sifra, kobasa.naziv,
			kobasa.komada, kobasa.cijena);
		fseek(binarna_datoteka, -1L*sizeof(kobasa), SEEK_CUR);
		fwrite(&kobasa, sizeof(kobasa), 1, binarna_datoteka);
	}
	//obavezno zatvaramo obje datoteke
	fclose(binarna_datoteka);
	fclose(formatirana_datoteka);
	return 0;
}
```

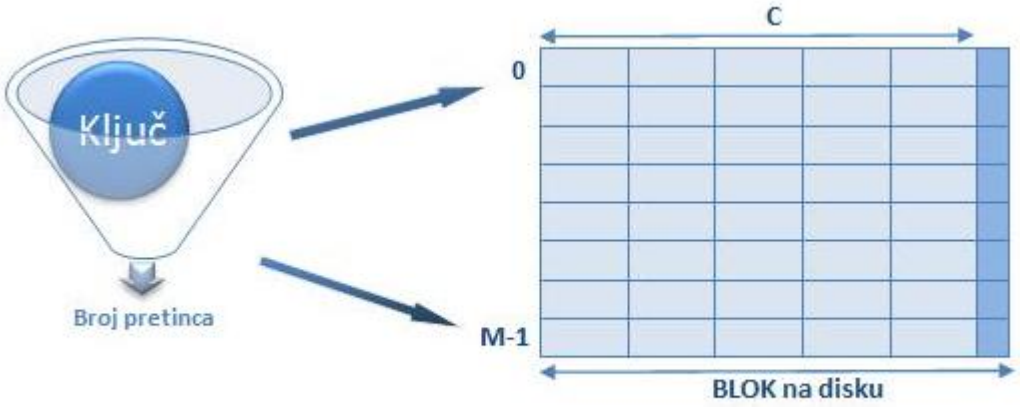
Datoteke_4 - Primljen je signal s Marsa te je pohranjen u datoteci *poruka.txt kao niz znakova*. Nadobudni ferovac je uspio dešifrirati njihov jezik. U direktnoj binarnoj datoteci *kodovi.dat*, se nalazi modificirana ASCII tablica. Svaki zapis sadrži: redni broj (*int, 0-128*), znak marsovske abecede (*char*) te njegov prijevod na naš jezik (*char*). Dešifrirajte poruku primljenu s Marsa!

Npr. ako je poslana poruka „/03132/“ → „abeceda“
Isječak iz tablice:

47	/	a
48	0	b
49	1	c
50	2	d
51	3	e

1.6. Raspršeno adresiranje

Raspršeno adresiranje (*eng. hashing*) omogućuje nam brzo pohranjivanje i brzo pristupanje podacima istog tipa. Svaki podatak predstavljen je jednim zapisom u datoteci. Datoteka je podijeljena na blokove i obično je njihova veličina usklađena s veličinom sektora na disku. Pretinac sadrži maksimalni broj zapisa iste veličine koji stanu u jedan takav blok. Transformacijom izvornog podatka (*šifra, matični broj i sl.*) nastaje pseudo-slučajni broj koji daje adresu pretinca. Ako je pretinac popunjen dogodio se preljev pa prelazimo u prvi sljedeći (*eng. bad neighbour policy*). Kako bi smanjili vjerojatnost da se dogodi preljev, broj pretinaca *M* obično računamo prema formuli (*N / C *1.3)*).



Zajednički dio za sve zadatke:

Jedan zapis datoteke organizirane po principu raspršenog adresiranja je definiran sljedećom strukturom:

```
struct zapis{
	int sifra;
	char tajni_agent[30+1];
	char tajno_ime[30+1];
	float placa;
	float ucinkovitost;
};
```

Zapis je prazan ako je na mjestu šifre vrijednost nula. Parametri za raspršeno adresiranje nalaze se u datoteci *parametri.h* i oni su:

- BLOK: veličina bloka na disku
- N: očekivani broj zapisa
- C: broj zapisa u jednom pretincu
- M: broj pretinaca

Ključ zapisa je šifra agenta, a transformacija ključa u adresu se obavlja zadanom funkcijom:

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

Hash_1 - Pridružite se tajnim agentima. Napravite funkciju prototipa:

```
int dodaj_agenta(FILE *fp,  zapis z);
```

koja će u datoteku upisati novi zapis. Funkcija treba vratiti 1 ako je zapis upisan u svoj pretinac, -1 ako je upisan kao preljev, a 0 ako zapis nije upisan u datoteku. *Napomena: možete pretpostaviti da zapis z ne postoji u datoteci prije poziva funkcije dodaj_agenta.*

```
int dodaj_agenta(FILE *fp, struct zapis z){
	int i, j, poc;
	//pretinac koji sadrži C zapisa
	struct zapis pretinac[C];
	//računamo u koji od M pretinaca ćemo smjestiti zapis, u slučaju da
	//taj pretinac popunjen (preljev), tražimo prvi sljedeći koji je slobodan
	i=adresa(z.sifra);
	poc=i;
	//ponavljamo dok se podatak ne upiše u neki od pretinaca ili dok
	//ne ustanovimo da je datoteka puna
	do {
		//pozicioniramo se na mjesto i-tog pretinca u datoteci
		fseek(fp, i*BLOK, SEEK_SET);
		//čitamo cijeli pretinac odjednom kako bi ga mogli pregledati u
		//radnoj memoriji
		fread (pretinac, sizeof(pretinac), 1, fp);
		//prolazimo kroz sve zapise u pretincu dok ne naiđemo na prazno mjesto
		for (j=0; j<C; j++) {
			if (pretinac[j].sifra==0) {
				pretinac[j]=z;
				fseek(fp, i*BLOK, SEEK_SET);
				fwrite(pretinac, sizeof(pretinac), 1, fp);
				//ako smo u početnom pretincu onda nema preljeva
				return poc==i ? 1 : -1;
			}
		}
		//pretinac je popunjen, ciklički prelazimo na sljedećeg
		i=(i+1)%M;
	} while(i!=poc);
	//ponavljamo dok se ne vratimo na početni pretinac što znači da u
	//cijeloj datoteci više nema mjesta pa vraćamo nulu
	return 0;
}
```

Hash _2 - Provjerite jeste li postali tajni agent. Napravite funkciju prototipa:

```
int pronadji_agenta(FILE *fp, int sifra);
```

koja će vratiti *„udaljenost„* zapisa od predviđenog pretinca. Ukoliko pojedini zapis nije spremljen kao preljev njegova udaljenost je 0, a inace udaljenost se definira kao broj pretinaca koje je dodatno potrebno procitati da bi se zapis pronašao.

Npr. ukoliko je M=10, a adresa nekog zapisa 7, a zapis se nalazi u pretincu broj 2, udaljenost je 5.

```
int pronadji_agenta(FILE *fp, int sifra){
	int i, j, poc, udaljenost;
	//pretinac koji sadrži C zapisa
	struct zapis pretinac[C];
	//računamo u koji od M pretinaca ćemo smjestiti zapis, u slučaju da
	//taj pretinac popunjen (preljev), tražimo prvi sljedeći koji je slobodan
	i=adresa(sifra);
	poc=i;
	//ponavljamo dok ne nađemo traženi zapis ili dok ne pregledamo sve pretince
	do {
		//pozicioniramo se na mjesto i-tog pretinca u datoteci
		fseek(fp, i*BLOK, SEEK_SET);
```

```
                //čitamo cijeli pretinac odjednom kako bi ga mogli pregledati u
                //radnoj memoriji
                fread(pretinac, sizeof(pretinac), 1, fp);
                //prolazimo kroz sve zapise u pretincu dok ne naiđemo na traženi zapis
                for(j=0; j<C; j++) {
                        //ako zapis nije prazan
                        if(pretinac[j].sifra!=0) {
                                //ako je to traženi zapis izračunaj udaljenost
                                if(pretinac[j].sifra==sifra){
                                        //ili udaljenost=(M+i-poc)%M;
                                        udaljenost=i-poc;
                                        if(udaljenost<0)
                                                udaljenost+=M;
                                        printf("%03d %s - %s\n\n", pretinac[j].sifra,
                                        pretinac[j].tajni_agent, pretinac[j].tajno_ime);
                                        printf("Agent pronadjen i spreman za akciju!\n");
                                        return udaljenost;
                                }
                        }
                }
                //pretinac je popunjen, ciklički prelazimo na sljedećeg
                i=(i+1)%M;
        }while(i!=poc);
        //ponavljamo dok se ne vratimo na početni pretinac što znači da u
        //cijeloj datoteci nema takvog zapisa
        return M;
}
```

Hash _3 - Priprema se jako zahtjevna tajna operacija. Odjel je odlučio poslati svog najboljeg čovjeka u akciju. Pomozite im da što prije dođu do svog najboljeg operativca. Napravite funkciju prototipa:

```
void najbolji_agent(FILE *fp, zapis *z);
```

koja će pronaći agenta s najvećom učinkovitošću.

```
void najbolji_agent(FILE *fp, struct zapis *z){
        int i, j;
        //pretinac koji sadrži C zapisa
        struct zapis pretinac[C];
        float naj=0;
        //svaki pretinac učitavamo u radnu memoriju, pregledavamo
        //zapise koji se nalaze u njemu i uspoređujemo učinkovitost
        //pojedinog agenta s do tada najvećom učinkovitošću
        for(i=0; i<M; i++){
                fseek(fp, i*BLOK, SEEK_SET);
                fread(pretinac, sizeof(pretinac), 1, fp);
                for(j=0; j<C; j++){
                        if(pretinac[j].sifra!=0){
                                if(pretinac[j].ucinkovitost>naj){
                                        naj=pretinac[j].ucinkovitost;
                                        //vraćamo zapis preko adrese
                                        *z=pretinac[j];
                                }
                        }
                }
        }
}
```

Hash _4 - Odjel se odlučio na reorganizaciju svog kadra. Svi agenti s učinkovitošću manjom od 25 bit će poslani u mirovinu, tj. treba ih izbrisati iz datoteke. Zapis u datoteci se smatra praznim ako mu je šifra jednaka 0. Također, svi agenti s učinkovitošću većom od 80 dobit će povišicu od 20%. Izvršiti navedene promjene funkcijom prototipa:

`int reorganizacija(FILE *fp);`

koja će preko imena vratiti broj umirovljenih agenata.

```
int reorganizacija(FILE *fp){
	int i, j;
	//pretinac koji sadrži C zapisa
	struct zapis pretinac[C];
	int umirovljeni=0;
	//svaki pretinac učitavamo u radnu memoriju, pregledavamo
	//zapise koji se nalaze u njemu i uspoređujemo učinkovitost
	for(i=0; i<M; i++){
		fseek(fp, i*BLOK, SEEK_SET);
		fread(pretinac, sizeof(pretinac), 1, fp);
		for(j=0; j<C; j++){
			if(pretinac[j].sifra!=0){
				if(pretinac[j].ucinkovitost<25) {
					//brišemo zapis
					pretinac[j].sifra=0;
					umirovljeni++;
					printf("Umirovljen: %s\n", pretinac[j].tajno_ime);
				}
				else if(pretinac[j].ucinkovitost>80) {
					//povećavamo plaću za 20%
					pretinac[j].placa*=1.2;
					printf("Povisica: %s\n", pretinac[j].tajno_ime);
				}
			}
		}
		//vraćamo se na početak trenutnog pretinca i spremamo
		//promjene na disk
		fseek(fp, i*BLOK, SEEK_SET);
		fwrite(pretinac, sizeof(pretinac), 1, fp);
	}
	return umirovljeni;
}
```

Hash _5 - Napisati funkciju koji će iz datoteke „ulaz.txt“ sa popisom tajnih agenata unjeti sve agente u novu datoteku po principu raspršenog adresiranja. Datoteka „ulaz.txt“ u svakom retku sadrži *sifru* (int), *ime* (char[30+1]), *tajno ime* (char[30+1]), *plaću* (float) te *učinkovitost agenta* (float). Transformacija ključa u adresu obavlja se prema formuli sifra%M. Također, napisati funkciju koja će ispisati cjelokupni sadržaj tablice, te funkciju koja će naći traženog agenta. U glavnom programu pozvati funkcije.

1.7. 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.

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){
	//za n==0 imamo osnovni slučaj
	//inače idemo "dublje" u rekurziju
	if (n != 0){
		//ispisuje brojeve od prvog prema posljednjem
		printf("%d ", d[0]);
		//predajući argument d+1, efektivno smanjujemo polje za 1 element
		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){
	//osnovni slučaj nastupa kad u polju preostane samo 1 element
	if (n==1)
		return niz[0];
	else
		//idemo korak dalje u rekurziju smanjujući broj elemenata za 1
		// približavamo se osnovnom slučaju, zbrajamo "izbačeni" član polja
		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, npr. radar) 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 racuna aproksimaciju broja Pi (*na 10 decimala*) kao sumu prvih *n* clanova reda:

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

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

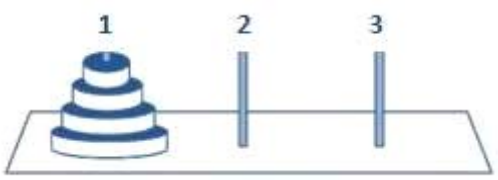
```
#include <stdio.h>
#include <math.h>

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){
	//osnovni slučaj nastupa kad dijeljenjem početnog broja s 10 dobijemo nulu
	if(broj==0) return 0;

	if (broj%10==znamenka)
		//ako znamenka u broju odgovara, sljedećem koraku rekurzije dodaj 1
		return 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;
}
```

Rekurzija_6 -Hanojski tornjevi su slagalica koju je 1883. godine osmislio E. Lucas. Danas je problem hanojskih tornjeva postao neizostavan primjer pri proučavanju rekurzije pa ga ni mi nećemo zaobići. Na prvom štapu ima n diskova različite veličine posloženih od najvećeg prema najmanjem, dok su preostala dva štapa prazna. Cilj je uz minimalan broj poteza premjestiti svih n diskova s jednog štapa na drugi. Dozvoljenim potezom se smatra ako se disk postavi na prazan štap ili ako je manji disk na većemu. Primjer za 4 diska:



Opis algoritma:

1) označimo štapove s brojevima **1**, **2** i **3** respektivno
2) zanemariti donji ujedno i najveći disk i riješiti problem za **n-1** disk, sa štapa **1** preseliti na štap **3** koristeći štap **2** kao pomoćni
3) sada će se najveći disk nalaziti na štapu **1**, dok će ostalih **n-1** diskova biti na štapu **3**
4) preseliti najveći disk s štapa **1** na štap **2**
5) preseliti **n-1** disk sa štapa **3** na štap **2** koristeći štap **1** kao pomoćni (sad smo riješili za **n-1** disk)

```
#include <stdio.h>

void hanojski_tornjevi(char prvi, char drugi, char treci, int broj_diskova){
	//osnovni je slučaj kad premjestimo sve diskove, tj. broj_diskova=0
	//inače rekurzija napreduje
	if(broj_diskova>0) {
		//n-1 disk selimo sa štapa 1 na štap 3, koristeći štap 2 kao pomoćni
		hanojski_tornjevi(prvi, treci, drugi, broj_diskova-1);
		printf("Prebacujem element %d s tornja %c na toranj %c\n",
			broj_diskova, prvi, drugi);
		//n-1 disk selimo sa štapa 3 na štap 2, koristeći štap 1 kao pomoćni
		hanojski_tornjevi(treci, drugi, prvi, broj_diskova-1);
	}
}
int main(){
	int broj_diskova;
	printf ("Upisite broj diskova: ");
	scanf ("%d", &broj_diskova);
	printf("\n\n\t**** Hanojski tornjevi (%d) ***\n\n", broj_diskova);
	hanojski_tornjevi('1', '2', '3', broj_diskova);
	return 0;
}
```

Rekurzija_7 - Napišite glavni program i rekurzivnu funkciju prototipa:

```
void permutiraj(int niz[], int velicina, int n)
```

koja će ispisati sve moguće rasporede brojeva iz niza cijelih brojeva zadanog u glavnom programu. Takvih kombinacija ima n!. Također napišite i pomoćnu funkciju kojom ćete ispisivati sadržaj niza.

Npr. za polje `int niz[3]={1, 2, 3};` veličine 3 program mora ispisati:

1 2 3
2 1 3
2 3 1
3 2 1
3 1 2

1.8. Analiza složenosti algoritama

Algoritam analiziramo kako bismo odredili količinu resursa (npr. vremena ili memorije) potrebnih za njegovo izvođenje te na osnovu tih podataka optimizirali početni algoritam ili pronašli učinkovitiji. Postoje dva osnovna tipa analize:

- *a priori:* trajanje izvođenja algoritma (u najgorem slučaju) kao vrijednost funkcije nekih relevantnih argumenata (npr. broja podataka)
- *a posteriori:* statistika dobivena mjerenjem na računalu

te nekoliko osnovnih tipova notacije:

- *O-notacija(ili apriorna složenost):* gornja granica za vrijeme izvođenja algoritma, samo red veličine;
- *Ω - notacija:* donja granica za vrijeme izvođenja algoritma;
- *Θ - notacija:* jednaka su trajanja za najbolji i najgori slučaj;
- *asimptotsko vrijeme izvođenja (~):* za razliku od *O-notacije*, osim reda veličine vodećeg člana u obzir uzimamo i konstantu koja ga množi;

U zadacima ćemo najčešće koristiti *O-notaciju* te *asimptotsko vrijeme izvođenja*.

Klasifikacija algoritama prema redu funkcije složenosti za najpoznatije klase algoritama prikazana je u sljedećoj tablici:

Tip algoritma	***f(n)***
konstantan	const.
logaritamski	log(n)
linearan	n
linearno-logaritamski	nlog(n)
kvadratni	n^2
stupanjski	n^k, (k>2)
eksponencijalni	k^n, (k>1)
faktorijelni	n!

Za dovoljno veliki n vrijedi:

$$O(1) < O(log\ n) < O(n) < O(nlog\ n) < O(n2) < O(n3) < \ldots < O(2n) < O(n!)$$

(1) Konstantni algoritmi:

- vrijeme izvođenja ograničeno je konstantom i ne ovisi o ulaznom parametru

```
double korijen(int n){
	//vrijeme izvođenja neovisno o ulaznom parametru
	//apriorna složenost: O(1)
	return sqrt(n);
}

void pisi(){
	int i;
	//petlja se izvodi konstantan broj puta
	//apriorna složenost: O(100000) = O(1)
	for(i=0; i<100000; i++)
		printf("\a");
}

int funkcija(int *polje, int n, int MAXSTUP){
	int i, j;
	for(i=0; i<n; i++)
		for(j=0; j<n; j++){
			printf("%d\n", polje[i*MAXSTUP+j]);
			//uvjet je prvi put zadovoljen za i=0 && j=2 i tada
			//izlazimo iz funkcije, unutarnja for petlja se do tog
			//trenutka izvrtila točno 2 puta, vrijeme izvođenja
			//dakle ne ovisi o n
			//apriorna složenost: O(1)
			if (i+j==2) return polje[i*MAXSTUP+j];
		}
	return 0;
}
```

(2) Logaritamski algoritmi:

- vrlo učinkoviti, veličina problema se svakom iteracijom smanjuje za red veličine baze, sve dok ne bude zadovoljen uvjet $n/a^k=1$ tj. $k=\log_a(n)$, npr. za binarno pretraživanje: $n/2^k=1 \rightarrow k=\log_2(n)$

```
int broj_znamenaka(int broj){
	int brojac=0;
	//pri svakom prolazu while petlje posao se smanjuje 10 puta
	//apriorna složenost: O(log10(n)) = O(log(n))
	while(broj){
		brojac++;
		broj/=10;
	}
	return brojac;
}


//algoritam binarnog pretraživanja - unutar while petlje problem
//se pri svakom prolazu smanjuje dva puta
//apriorna složenost: O(log2(n))
int B(int a[], int x, int n) {
	int donji, srednji, gornji;
	donji=0;
	gornji=n-1;
	while(donji<=gornji){
		srednji=(donji+gornji)/2;
		if(a[srednji]<x) donji=srednji+1;
		else if (a[srednji]>x) gornji=srednji-1;
		else return srednji;
	}
	return -1;
}
```

(3) Linearni algoritmi:

- obrađuje se n istovjetnih podataka, a udvostručenje količine radnji ima za posljedicu udvostručenje vremena obrade

```
int trazi(int *polje, int n, int zadani){
	int i;
	//vrijeme izvođenja ovisi o veličini polja
	//apriorna složenost: O(n)
	//najbolji slučaj: traženi element je prvi element polja - O(1)
	//najgori slučaj: traženi element nije nađen - O(n)
	for(i=0; i<n; i++)
		if (polje[i*n+i]==zadani)
			return i;
	return 0;
}

int funkcija(int *polje, int n, int MAXSTUP){
	int i, j;
	//dva puta ćemo proći kroz vanjsku petlju dok ne bude ispunjen
	//uvjet i==1 kada izlazimo iz funkcije, kako se za svaki prolaz
	//vanjske for petlje unutarnja petlja izvrti n puta, složenost je:
	//asimptotska složenost: ~2n
	//apriorna složenost: O(n)
	for(i=0; i<n; i++)
		for(j=0; j<n; j++){
			printf("%d\n", polje[i*MAXSTUP+j]);
			if(i==2) return polje[i*MAXSTUP+j];
		}
	return 0;
}

long potencija(int x, int y){
	//brojimo li samo pozive i povratke iz funkcije, složenost je:
	//asimptotska složenost: ~2y
	//apriorna složenost: O(y)
	if(y<=0) return 1;
```

```
        else return x*potencija(x, y-1);
}
```

(4) Linearno-logaritamski algoritmi:

- vrlo učinkoviti, složenost im za veliki n raste sporije od kvadratne funkcije, npr. *merge sort*, *qiuck sort*

```
//Merge Sort
//grananjem nastaje log2(n) razina, a u svakoj razini obavlja se O(n) posla
//apriorna složenost: O(log2(n))
void MSort(int A[], int PomPolje[], int lijevo, int desno) {
  int sredina;
  if (lijevo<desno){
    sredina=lijevo+(desno-lijevo)/2;
    MSort(A,PomPolje,lijevo,sredina);
    MSort(A,PomPolje,sredina+1,desno);
    Merge(A,PomPolje,lijevo,sredina+1,desno);
  }
}
```

(5) Kvadratni i stupanjski algoritmi:

- kvadratni algoritmi najčešće se dobivaju kada se koriste dvije for petlje, jedna unutar druge dok su stupanjski algoritmi ništa drugo nego poopćenje kvadratnih

```
int trazi(int *polje, int n, int zadani){
        int i, j, nasao=0;
        //vrijeme izvođenja ovisi o veličini polja, za svaki prolaz vanjske
        //petlje, unutarnja se izvrti n puta
        //apriorna složenost: O(n2)
        for(i=0; i<n; i++)
                for (j=0; j<n; j++) {
                        if (i==j && polje[i*n+j]==zadani)
```

```
                        nasao=1;
                }
        return nasao;
}

int funkcija(int *polje, int n, int MAXSTUP){
        int i, j, suma = 0;
        //kroz vanjsku petlju proći ćemo n puta, a za svaki taj prolaz
        //unutarnja će se izvrtiti također n puta
        //apriorna složenost: O(n^2)
        for (i=0; i<n; i++)
                for (j=0; j<n; j++){
                        printf("%d\n", polje[i*MAXSTUP+j]);
                        if(i+j==2) suma+=polje[i*MAXSTUP+j];
                }
        return suma;
}

int funkcija(int *polje, int m, int n, int MAXSTUP){
        int i, j, suma;
        //pri svakom prolazu vanjske petlje, unutarnja se izvrti n puta
        //asimptotska složenost: O(2n^2)
        //apriorna složenost: O(n^2)
        //najbolji slučaj: prvi element matrice je neparan - O(1)
        //najgori slučaj: svi su elementi matrice parni - O(n^2)
        //prosječni slučaj: O(n/2)
        for (i=0; i<n; i++)
                for (j=0; j<=i; j++){
                        if (polje[i*MAXSTUP+j]%2==0) suma+=polje[i*MAXSTUP+j];
                        else break;
                }
        return suma;
```

```
}
```

(6) Eksponencijalni algoritmi:

- spadaju u kategoriju problem za koje se suvremena računala ne mogu koristiti, izuzev u slučajevima kada su dimenzije takvog problema veoma male

```
//Fibonacci numbers
int fibonacci(int n){
	//osnovni slučaj za n=1 || n=0, inače rekurzija napreduje pri
	//čemu imamo dva rekurzivna poziva
	//apriorna složenost: O(2^n)
	if(n<=1) return 1;
	else return fibonacci(n-1)+fibonacci(n-1);
}

//Towers of Hanoi
void hanojski_tornjevi(char prvi, char drugi, char treci, int broj_diskova){
	//osnovni je slučaj kad premjestimo sve diskove, tj. broj_diskova=0
	//inače rekurzija napreduje pri čemu imamo 2 rekurzivna poziva
	//apriorna složenost: O(2^n)
	if(broj_diskova>0) {
		hanojski_tornjevi(prvi, treci, drugi, broj_diskova-1);
		printf("Prebacujem element %d s tornja %c na toranj %c\n",
			broj_diskova, prvi, drugi);
		hanojski_tornjevi(treci, drugi, prvi, broj_diskova-1);
	}
}
```

(7) Faktorijelni algoritmi:

```
//Trade salesman problem
```

Analiza „a posteriori“

```
#include <stdio.h>
#include <time.h>
#include <sys\timeb.h>

int fibonacci(int n){
	//osnovni slučaj za n=1 || n=0, inače rekurzija napreduje pri
	//čemu imamo dva rekurzivna poziva
	//apriorna složenost: O(2n)
	if(n<=1) return 1;
	else return fibonacci(n-1)+fibonacci(n-1);
}

int main(){
	int n;
	long fib, trajanje;
	//koristimo struct timeb deklariranu u zaglavlju <sys\timeb.h>
	//struct timeb{
	////time_t time; (broj sekundi od ponoći, 01.01.1970, UTC)
	////unsigned short millitm; (milisekunde)
	////short timezone; (razlika u minutama od UTC)
	////short dstflag; (<>0 ako je na snazi ljetno vrijeme)
	//};
	struct timeb vrijeme1, vrijeme2;
	printf("Koji Fibonacci broj zelite izracunati: ");
	scanf("%d", &n);
	//funkcija ftime također deklarirana u zaglavlju <sys\timeb.h>
	//određuje trenutno vrijeme te popunjava struct timeb
	//prototip: void ftime(struct timeb *timeptr);
	ftime(&vrijeme1);
	fib=fibonacci(n);
```

```
        ftime(&vrijeme2);

        //tražimo razliku konačnog i početnog vremena u milisekundama

        trajanje=1000*(vrijeme2.time-vrijeme1.time)+vrijeme2.millitm-
        vrijeme1.millitm;

        printf("Fibonacci(%d) = %d\n", n, fib);

        printf("Za izracunavanje Fibonacci(%d) bilo je potrebno: %d [ms]\n", n,
        trajanje);

        return 0;

}
```

1.9. Blic – pitanja

1) Pokazivači

1) Kolika je vrijednost varijabli i i j nakon izvođenja sljedećeg programskog odsječka:

```
int i=23, j=72;
int *p1, *p2;
p2=&i;p1=&j;
*p1=*p2;
```

   a. i=23, j=23
   b. Ništa, program će se srušiti.
   c. i=23, j=72
   d. i=72, j=23
   e. i=72, j=72

2) U slučaju da vrijedi:

```
int *p;
int i;
```

koji od sljedećih izraza ne možemo napisati?

   a. p=0;
   b. p=i;
   c. P=p+1;
   d. p=&i;
   e. p=NULL;

3) Što ispisuje sljedeći program:

```
void funkcija(int A[]) {
	printf("A[0]=%d, A[1]=%d, A[2]=%d\n", A[0], A[1], A[2]);
	A[1] = 0;
}
int main(){
	int A[3]={1, 2, 3};
	funkcija(A);
	printf("A[0]=%d, A[1]=%d, A[2]=%d\n", A[0], A[1], A[2]);
	return 0;
}
```

   a. A[0]=1, A[1]=2, A[2]=3
      A[0]=1, A[1]=2, A[2]=3
   b. Greška – elementi polja ne mogu se ispisivati na ovaj način.
   c. A[0]=1, A[1]=2, A[2]=3
      A[0]=1, A[1]=0, A[2]=3
   d. A[0]=1, A[1]=0, A[2]=3
      A[0]=1, A[1]=0, A[2]=3
   e. A[0]=0, A[1]=0, A[2]=0
      A[0]=0, A[1]=0, A[2]=0

4) Što će ispisati sljedeći program:

```
int main(){
	int i, j, *p, *q;
	p=&i;
	q=&j;
	*p=5;
```

```
        *q=*p+i;
        printf("i=%d, j=%d\n", i, j);
        return 0;
}
```

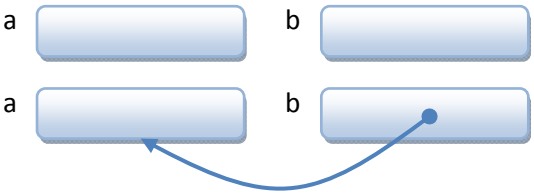
a. i=5, j=10;
b. i=5, j=5;
c. i=10, j=5;
d. i=10, j=10;
e. Ništa, program će se srušiti.

5) Kako treba glasiti naredba printf za kod programa da se ispise 2 3 1 ?

```
char a,b;
char *p1,*p2;
a='2'; b='3';
p1=&a; p2=&b;
```

a. printf("%c %c %c", p1, p2, p2-p1);
b. printf("%c %c %d", *p1, *p2, *p2-*p1);
c. printf("%c %c %d", *p1, *p2, p2-p1);
d. printf("%d %d %c", *p1, *p2, *p2-*p1);
e. printf("%c %c %d", p1, p2, *p2-*p1);

6) Uz pretpostavku da je veličina kućice proporcionalna zauzeću memorije, kojem programskom odsječku odgovara sljedeći slijed slika:



a.
```
int a, *b;
b = *a;
```
b.
```
int *a, b;
b = &a;
```
c.
```
int *a, b;
a = &b;
```
d.
```
int a, *b;
b = &a;
```
e.
```
int a, *b;
b = a;
```

7) Adresa pohranjena u pokazivaču deklariranom kao `int *p;` povećat će se za `1` bajt sljedećom naredbom:

a. p=(int *)(p+1);
b. p=(int *)((char *)p + 1);
c. p=*p+1;
d. p=*(p+1);
e. p=(char *)((int *)p + 1);

8) Što će se ispisati programom:

```
int fun(char *c, int i){
    (*c)++;
    i--;
    return (*c)*i;
}
```

```
int main(){
    char c='0'; int i=1, j;
    j=fun(&c, i);
    printf("%d %d %d", c, i, j);
}
```

a. 49 1 0
b. 48 1 0
c. Program je sintaktički neispravan.
d. 48 0 0
e. 49 0 0

9) Što će se ispisati sljedećim programom:

```
#include <stdio.h>
void uvecaj(int a, int *b){
    a=a+1; *b=*b+1;
}
void main(){
    int a=0, b=0;
    printf("a=%d b=%d  ", a, b);
    uvecaj(a, &b);
    printf("a=%d b=%d\n", a, b);
}
```

a. Greška – funkcija koja mijenja parametre ne smije biti tipa `void`.
b. Greška – varijable a i b imaju nedefiniranu vrijednost u funkciji `uvecaj`.
c. a=0 b=0 a=0 b=1
d. a=0 b=1 a=1 b=2
e. a=1 b=1 a=0 b=0

10) Što je krivo sa sljedećim programskim odsječkom?

```
void *p;
p=(float*)malloc(sizeof(float));
*p=7.89;
```

a. Ništa, sve radi dobro.
b. Float nije osnovni tip podatka pa ne možemo koristiti sizeof() nad njime.
c. P ne može biti dereferenciran zato što je void pokazivač.
d. Ne možemo deklarirati void pokazivače.
e. Pokazivaču ne možemo pridružiti brojčanu vrijednost.

11) Što će se pohraniti u varijablu a slijedećim programskim odsječkom:

```
short *a;
short a1=1;
a=&a1;
```

a. Programski odsječak je sintaktički neispravan.
b. Ne možemo deklarirati short pokazivače.
c. Adresa varijable a.
d. Nepoznata vrijednost.
e. Vrijednost varijable a.

12) Koja od sljedećih tvrdnji nije istinita, ako imamo naredbu: `*p=7;`:

a. Pokazivač p mora prethodno biti inicijaliziran da bi naredba bila logički ispravna.

b. Nova adresa na koju pokazuje pokazivač p nakon naredbe

```
p=p+broj;
```

može se dobiti na sljedeći način:

```
nova_adresa=stara_adresa+broj*sizeof(long);
```

c. Ako je deklarirano polje `int a[5]` sljedeće naredbe, kojima pristupamo trećem elementu polja, su ekvivalentne:

```
*(a+2)
a[2]
```

d. Pokazivače se može uspoređivati.

e. Pokazivaču se može oduzeti i dodati cijeli broj.

13) Ukoliko prva `printf` funkcija iz priloženog programskog odsječka ispiše:

```
1245032 2.710000
```

što će ispisati sljedeća `printf` funkcija ?

(pretpostavka je da se memorija adresira sa 4 bajta, dakle `unsigned long`)

```
void main(){
     float *pok, var[2]={2.71, 3.14};
     pok=var;
     printf("\n%lu %f", pok,*pok);
     pok++;
     printf("\n%lu %f", pok,*pok);
}
```

a. `1245033 2.710000`
b. `1245034 3.140000`
c. `1245036 2.710000`
d. `1245036 3.140000`
e. `1245034 2.710000`

2) Dinamička rezervacija memorije

1) Funkcija `sizeof(int)` vraća:

a. Najveći broj koji se može spremiti u varijablu tipa int.
b. Broj varijabli tipa int deklariranih u funkciji u kojoj se trenutno nalazimo.
c. Broj bajtova potrebnih da se pohrani varijabla tipa int.
d. Najveći dopušteni broj znakova u imenu verijable tipa int.
e. Najveći dopušteni broj varijabli tipa int u trenutnoj funkciji.

2) Koliko će se bajtova memorije dinamički zauzeti izvršavanjem naredbi:

```
long int *p=NULL, n=2;
p=(long int *)realloc(p, n*2*sizeof(long int));
```

a. 4
b. 16
c. 8
d. 2
e. 32

3) Što će na zaslon ispisati sljedeći program:

```
int main (void){
    long int *broj1, *broj2;
```

```
    int broj;
    broj1=broj2=(long int*)malloc(4*sizeof(long int));
    broj2+=8;
    broj=(int)((char*)broj2-(char*)broj1);
    printf("%d", broj);
    return 0;
}
```

a. 8
b. 32
c. 16
d. 0
e. 2

4) Koliko bajtova memorije će se zauzeti izvršavanjem sljedećih naredbi:

```
long int *broj;
broj=(long int*)malloc(4*sizeof(long int));
broj=(long int*)realloc(broj, 8*sizeof(long int));
```

a. 32
b. 64
c. 48
d. 8
e. 16

5) Koliko bajtova memorije će se zauzeti izvršavanjem sljedećih naredbi:

```
double *broj;
broj=(double*)malloc(4*sizeof(double));
broj=(double*)realloc(broj, 8*sizeof(double));
```

a. 96
b. 8
c. 64
d. 12
e. 16

6) Ugrađene funkcije za dinamičku alokaciju memorije nalaze se u zaglavlju:

a. realloc.h
b. mem.h
c. memory.h
d. malloc.h
e. stdio.h

7) Kako bismo rezervirali memorijski prostor za znakovni niz duljine 8?

a. (char*)malloc(8*sizeof(char));
b. (char*)malloc(9);
c. (int*)malloc(8)*sizeof(int));
d. (char*)realloc(9*sizeof(char));
e. realloc(p, 8*sizeof(char));

8) Prototip funkcije `realloc` jest:

a. *void realloc(size_t size);*
b. *void *realloc(void *pointer, size_t size);*
c. *void *realloc(void pointer, size_t size, size_t n);*
d. *void realloc(void *pointer, size_t size, int n);*

e. *void realloc(void pointer, size_t size);*

9) Prototip funkcije `malloc` jest:
   a. *void malloc(size_t size, size_t n);*
   b. *void *malloc(size_t *size);*
   c. *void *malloc(size_t size);*
   d. *void malloc(size_t *size, size_t n);*
   e. *void malloc(size_t size);*

10) Što od sljedećeg nije točno:
   a. Funkciju `realloc` također možemo koristiti kao `malloc`.
   b. I `malloc` i `realloc` u slučaju neuspjeha vraćaju NULL pokazivač.
   c. Funkciju `realloc` koristimo kada nam je potreban veći blok memorije od onog koji trenutno koristimo.
   d. Dinamički zauzeta memorija oslobađa se automatski pozivom funkcije `return`.
   e. Sve od navedenog.

3) Pseudoslučajni brojevi

1) Prototip funkcije rand() definiran je u zaglavlju:
   a. stdio.h
   b. stdlib.h
   c. time.h
   d. system.h
   e. math.h

2) Sljedeća funkcija:

```
int slucajni_broj(){
    return 10*rand()/(RAND_MAX+1)+1;
}
```

će vraćati vrijednosti iz intervala:
   a. [0,10]
   b. [0,11]
   c. [1,11]
   d. [1,9]
   e. [1,10]

3) Sljedeća funkcija:

```
int slucajni_broj(){
    return rand()%16+10;
}
```

će vraćati vrijednosti iz intervala:
   a. [10,16]
   b. [10,25]
   c. [10,24]
   d. [10,15]
   e. [10,26]

4) Kako inicijaliziramo generator pseudoslučajnih brojeva?
   a. `rand((unsigned)time(NULL));`

b. `srand((unsigned)time(NULL));`
c. `rand((unsigned)time(NULL, 0));`
d. `srand((unsigned)time(NULL, 0));`
e. `srand(unsigned (time(NULL)));`

5) Koja od sljedećih tvrdnji nije točna:
   a. Ugrađena funkcija `rand` deklarirana je u zaglavlju `stdlib.h`.
   b. Niti jedna od navedenih.
   c. Generirani nizovi pseudoslučajnih brojeva uvijek su različiti, bez obzira na vrijednost sjemena.
   d. Generirani pseudoslučajni brojevi su iz intervala [0, RAND_MAX].
   e. Generator pseudoslučajnih brojeva možemo inicijalizirati npr. ovako:
      `srand(1);`

4) Prenošenje polja u funkciju

1) Ukoliko funkcija f treba izračunati sumu svih elemenata u matrici koju naredbu treba umetnuti na mjesto označeno s ??? ?

```
int f(int *p, int m, int n, int maxstup){
        int i, j, suma=0;
        for(i=0; i<m ; i++){
                for(j=0; j<n; j++){
                        suma+=*p;
                        p++;
                }
                ???
        }
        return suma;
}
```

   a. `p+=maxstup;`
   b. `p+=n;`
   c. `p++;`
   d. `p+=n*maxstup`
   e. `p+=maxstup-n;`

2) Ukoliko funkcija suma treba izračunati sumu svih elemenata u matrici koju naredbu treba umetnuti na mjesto označeno s ??? ?

```
int suma(int *p, int m, int n, int maxstup){
        int i, j, suma=0;
        int *pom=p;
        for(i=0; i<m ; i++){
                for(j=0; j<n; j++){
                        suma+=*pom;
                        pom++;
                }
                ???
        }
        return suma;
}
```

   a. `pom=(i+1)*maxstup;`
   b. `pom=p+maxstup;`
   c. `pom=p+(i)*maxstup;`
   d. `pom=p+(i+1)*maxstup;`

e. `pom = (i)*maxstup;`

3) Ukoliko funkcija suma treba izračunati sumu svih elemenata u neparnim stupcima matrice koju naredbu treba umetnuti na mjesto označeno s ??? ?

```
int suma(int *p, int m, int n, int maxstup){
	int i, j, suma=0;
	int *pom;
	for(i=0; i<m ; i++){
		pom=p+i*maxstup;
		for(j=1; j<n; j+=2){
			???
		}
	}
	return suma;
}
```

a. `suma+=*(pom+j);`
b.
```
if(j%2==0){
	suma+=*(pom+j);
}
```
c. `suma+=*(j);`
d. `suma+=*(pom+j-1);`
e. `suma+=*(pom+j+1);`

4) Koji je od sljedećih prototipa funkcije koja prima dvodimenzionalno polje cijelih brojeva kao ulazni parametar točan:

a. `int funkcija(int polje, int m, int n);`
b. `int funkcija(int &polje[0][0], int m, int n, int MAXSTUP);`
c. `int funkcija(int &polje[0][0], int m, int n);`
d. `int funkcija(int *polje, int m, int n, int MAXSTUP);`
e. `int funkcija(int polje, int n);`

5) Koja je od sljedećih tvrdnji točna:

a. Kod prenošenja znakovnog niza u funkciju, koji je u memoriji računala predstavljen jednodimenzionalnim poljem, nije potrebno prenjeti duljinu niza.
b. Niti jedna.
c. Prenosimo li dvodimenzionalno polje u funkciju potrebno je prenjeti pokazivač na prvi element niza, broj redaka i broj stupaca.
d. U funkciji kojoj je predan pokazivač na prvi element jednodimenzionalnog polja, i-ti element polja možemo dobiti npr. ovako:
```
&polje[i];
```
e. Sve.

5) Datoteke

1) Koja od sljedećih naredbi ispravno provjerava da li je veza programa i datoteke ispravno prekinuta:

a. `if(fclose(*f)==0){printf("Uspješno zatvaranje");}`
b. `if(fclose(f)== NULL){printf("Uspješno zatvaranje");}`
c. `if(fclose(f)==1){printf("Uspješno zatvaranje");}`
d. `if(fclose(*f)==1){printf("Uspješno zatvaranje");}`
e. `if(fclose(f)==0){printf("Uspješno zatvaranje");}`

2) Ako je sadržaj formatirane datoteku `ulaz` 012012 što će ispisati sljedeći programski odsječak:

```
char c;
int s=0;
FILE *in=fopen("ulaz","r");
while(fscanf(in,"%c",&c)){
    s+=c-'1';
}
printf("%d\n",s);
```

a. 49
b. 0
c. 48
d. 288
e. 6

3) Što će biti ispisano sljedećim programom (underscore označava prazno mjesto):

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
int main() {
       FILE *fp; char c; int br=0;
       fp=fopen("c:\\tmp\\dat.txt", "w+");
       fputs("Nin Visegrad Drnis", fp);
       rewind(fp);
       while((c=fgetc(fp))!=EOF)
              if(isprint(c)) ++br;
       printf("%3d", br);
       return 0;
}
```

a. _15
b. 15
c. _18
d. 018
e. 18

4) Nakon izvršavanja naredbe `fprintf`, što će se nalaziti zapisano u datoteci (gledajući binarno):

```
fprintf(fp, "%c\n", '0');
```

a. 00110000
b. 00110000 00001010
c. 00110000 00000000
d. 00000000 00001010
e. 00000000

5) Što će biti zapisano u datoteku sljedećim programskim odsječkom:

```
int i = 4; FILE *datIzlaz;
...
fprintf(datIzlaz, "%3d", i);
```

a. 00000000 00000000 00110100
b. 00000100
c. 00000000 00000000 00000100
d. 00100000 00100000 00110100

e. 01100100

6) Što će se ispisati na zaslon sljedećim programskim odsječkom, ako datoteka `test.dat` ne postoji na magnetskom disku:

```
FILE *f;
f = fopen ("test.dat", "r");
if (f) {
    printf ("Datoteka već postoji");
} else {
    printf ("Datoteka ne postoji");
}
```

a. Program će dojaviti pogrešku jer varijabla `f` treba biti tipa `int`.
b. Varijabli tipa `FILE` * ne može se pridružiti vrijednost na opisani način.
c. Neće se ispisati ništa, jer će operacijski sustav dojaviti pogrešku.
d. Datoteka već postoji.
e. Datoteka ne postoji.

7) Što će biti zapisano u neformatiranu datoteku slijedećom naredbom:

```
char c='A';
FILE *datoteka;
datoteka=fopen("test.dat","wb");
fwrite(&c, sizeof(c), 1, datoteka);
```

a. 01000001
b. 00000000 00000000 00000000 01000001
c. 01000001 00000000 00000000 00000000
d. 00000000 01000001
e. 00000000 00000000 00000000 01000001

8) Što će biti zapisano u neformatiranu datoteku `dat.txt` sljedećim programskim odsječkom:

```
FILE *fp;
short int a=9;
fp=fopen("dat.txt", "w+");
fwrite(&a, sizeof(a), 1, fp);
```

a. 00001001 00000000
b. 00000000 00001001
c. 00001001 00000000 00000000 00000000
d. 00000000 00000000 00001001 00000000
e. 00000000 00000000 00000000 00001001

9) Na disku postoji datoteka `tu.sam` naredbama:

```
FILE *fp;
fp=fopen("tu.sam", "a");
```

otvorit će se datoteka :

a. za pisanje s tim da će se pokazivač postaviti na kraj datoteke.
b. za čitanje i pisanje s tim da se pokazivač postavi na kraj datoteke.
c. za pisanje s tim da će se prebrisati postojeća datoteka.
d. za čitanje i pisanje s tim da se pokazivač postavi na početak datoteke.
e. samo za čitanje.

10) Kojom od naredbi otvaramo datoteku iz koje nije dopušteno čitanje:

```
FILE *f;
```

a. `f=fopen("osobe.txt", "a+");`
b. `f=fopen("osobe.txt", "a");`
c. `f=fopen("osobe.txt", "w+");`
d. `f=fopen("osobe.txt", "r+");`
e. `f=fopen("osobe.txt", "r");`

11) Što će biti zapisano u datoteku sljedećim programskim odsječkom?

```
...
int i = 2; FILE *datIzlaz;
...
fprintf(datIzlaz, "%2d", i);
...
```

a. 00000000 00110010
b. 01100010
c. 00000000 00000010
d. 00000010
e. 00100000 00110010

6) Raspršeno adresiranje

1) Uzmimo da su podaci o studentima pohranjeni u datoteku. Podaci su sortirani prema JMBAG-u (kao na slici). Koliko će čitanja biti obavljeno za dohvaćanje podataka o studentu čiji je JMBAG 35 ako se koristi čitanje po blokovima, a veličina bloka je 4?

1
2
3
5
8
12
20
25
30
35
40
41
42
43

a. 4
b. 7
c. 5
d. 1
e. 6

2) Uzmimo da su podaci o studentima pohranjeni u datoteku. Podaci su sortirani prema JMBAG-u (kao na slici). Koliko će čitanja biti obavljeno za dohvaćanje podataka o studentu čiji je JMBAG 35 ako se koristi binarno adresiranje?

1
2
3
5
8
12
20
25
30
35
40
41
42
43

a. 3
b. 2
c. 4
d. 7
e. 6

3) Neka se ključevi zapisa nekom metodom transformacije u postupku raspršenog adresiranja transformiraju u *vrijednosti* iz intervala [0,999]. Koji će raspon *vrijednosti* smjestiti u pretinac s adresom 3, ako su adrese pretinaca iz intervala od [0,199]?

a. [20,24]
b. [15,20]
c. [15,19]
d. [20,25]
e. [16,20]

4) Neka se ključevi zapisa nekom metodom transformacije u postupku raspršenog adresiranja transformiraju u *vrijednosti* iz intervala [0,999]. Koji će raspon *vrijednosti* smjestiti u pretinac s adresom 3, ako su adrese pretinaca iz intervala od [0,199]? U koji će pretinac otići zapis s ključem koji se transformira u vrijednost 56?

a. 25
b. 56
c. 55
d. 10
e. 11

5) U datoteku organiziranu po načelu raspršenog adresiranja pohranjuju se zapisi koji sadrže šifru iz intervala [100000, 500000] i naziv (19 +1 znak). Fizički blok na disku je veličine 512 okteta. Koliki je maksimalni broj zapisa po pretincu?

a. 22
b. 25
c. 26
d. 21
e. 27

6) U datoteku organiziranu po načelu raspršenog adresiranja pohranjuju se zapisi koji sadrže šifru iz intervala [100000, 500000] i naziv (19 +1 znak). Fizički blok na disku je veličine 512 okteta. Broj zapisa koje treba pohraniti je 10000. Broj pretinaca je zbog očekivanog preljeva veći za 30%. Koliki je u tom slučaju broj pretinaca?

a. ~ 476
b. ~ 620
c. ~ 48
d. ~ 143
e. ~ 470

7) U datoteku organiziranu po načelu raspršenog adresiranja pohranjuju se zapis koji sadrže šifru iz intervala [100000, 500000] i naziv (19 +1 znak). Broj zapisa koje treba pohraniti je 10000. Ako je broj zapisa po pretincu 21, a ukupno ima 620 pretinaca, kolika je gustoća pakiranja?

a. 0.8
b. 0.08
c. 476
d. 16
e. 339

8) Ukoliko je N očekivani broj zapisa, C broj zapisa u jednom pretincu, a M broj pretinaca, gustoća pakiranja zapisa u datoteci G računa se prema formuli:

a. G=M/(N*C)
b. G=N/(M*C)
c. G=N/M*C
d. G=N/M
e. G=(N/M)*C

9) Ukoliko je K očekivani broj zapisa, L broj zapisa u jednom pretincu, a M broj pretinaca, gustoća pakiranja zapisa u datoteci G računa se prema formuli:

a. G=M/(K*L)
b. G=K/(M*L)
c. G=K/M*L
d. G=K/M
e. G=(K/M)*L

10) Nakon pohrane velike količine podataka, hash funkcija za transformaciju ključa u adresu slučajno je izgubljena. Što je od sljedećeg točno:
    a. Sve od navedenog.
    b. Traženom podatku više nikad nećemo moći pristupiti – potrebno je prepisati sve podatke u novu datoteku.
    c. Ništa od navedenog.
    d. Hash funkcija nam ionako nije potrebna za pronalazak traženog podatka – gubimo samo mogućnost upisa novih podataka u datoteku.
    e. Traženi podatak možemo pronaći slijedno, sa složenošću O(n).

7) Analiza složenosti algoritama

1) Koliko je prosječno asimptotsko vrijeme izvođenja funkcije pot, ako se broje samo pozivi funkcije i povratci iz funkcije:

```
long pot(long x, long y){
      if (y<=0) return 1;
      else return x*pot(x, y-1);
}
```

    a. y
    b. 2y
    c. y^2
    d. ylog(y)
    e. y/2

2) Koliko iznosi apriori složenost sljedećeg programskog odsječka:

```
int i;
...
while(i>0){
i/=5;
}
```

    a. $O(\log_5 i)$
    b. O(5)
    c. O(i)
    d. O(5i)
    e. O(i/5)

3) Kolika je složenost funkcije Fibonacci(3) u terminu O-notacije:

```
int Fibonacci(int n){
      if(n<=1) return 1;
      else return Fibonacci(n-1)+Fibonacci(n-2);
}
```

    a. $O(n^2)$
    b. $O(2^n)$
    c. O(n!)
    d. $O(3^n)$
    e. $O(n2^n)$

4) Koja od slijedećih tvrdnji je istinita za dovoljno velik broj n:

a. $O(n^2) < O(2n) < O(n^3)$
b. $O(n^2) < O(2n) < O(n^3)$
c. $O(n^3) < O(n^2) < O(n)$
d. $O(1) < O(n) < O(\log_2(n))$
e. $O(2n) < O(n^3) < O(3n)$

5) Koje je apriorno vrijeme f-je koja vraća najmanji element matrice:

```
int min(int *mat, int n){
    int i, min;
    min =*mat;
    for (i=1;i<n*n;i++){
        if (min>*(mat+1)){
            min=*(mat+i);
        }
    }
    return min;
}
```

a. O(n)
b. $O(n^2)$
c. O(log(n))
d. $O(2^n)$
e. O(2n)

6) Koja od sljedećih tvrdnji je istinita, za dovoljan velik n:

a. $O(1) < O(\log(n)) < O(n) < O(n\log(n)) < O(n^2) < O(n^3) < O(2n)$
b. $O(1) < O(\log(n)) < O(n\log(n)) < O(n) < O(n^2) < O(n^3) < O(2n)$
c. $O(\log(n)) < O(1) < O(n\log(n)) < O(n) < O(n^2) < O(n^3) < O(2n)$
d. $O(\log(n)) < O(1) < O(n) < O(n\log(n)) < O(n^2) < O(n^3) < O(2n)$
e. $O(1) < O(\log(n)) < O(n) < O(n\log(n)) < O(2n) < O(n^2) < O(n^3)$

7) Programski odsječak:

```
int n;
for (i=0; i<1; i++)
      for (j=0; j<1; j++){
            n+=n*2;
}
```

ima apriornu složenost:

a. O(i)
b. O(1)
c. O(n)
d. Ne može se odrediti bez izvođenja na računalu.
e. O(i*j)

8) Odredi O-notaciju:

```
int n;
...
while(n>0){
n/=3;
}
```

a. O(3)
b. $O(\log_3(n))$

c. $O(3\log_3(n))$
d. $O(n\log_3(n))$
e. $O(1)$

9) Odrediti vrijeme izvođenja sljedeće funkcije za najbolji slučaj:

```
int postoji(int polje[], int n, int br){
     int i;
     for (i=0;i<n;i++){
       if (polje[i]==br) return 1;
     }
     return 0;
}
```

a. $O(2)$
b. $O(\log(n))$
c. $O(2n)$
d. $O(1)$
e. $O(n)$

10) Što je od sljedećeg istinito:

a. $O(2n^2+n)=O(2n^2)$
b. $O(2n^2+n)=O(n)$
c. $O(2n^2+n)=O(2n)$
d. $O(2n^2+n)=O(n^2)$
e. $O(2n^2+n)=O(n)$

11) Odredite apriornu složenost sljedeće funkcije:

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

a. $O(n)$
b. $O((n-1)!)$
c. $O(n!)$
d. $O(n^2)$
e. $O(1)$

12) Apriorna složenost sljedećeg programskog odsječka je:

```
for(i=0; i<n; i++){
      for(j=0; j<m; j++){
            if(i%2){
                  polje[i][j]+=m;
            }else{
                  Polje[i][j]+=n;
            }
      }
}
```

a. $O(n^2)$
b. $O(n*m)$
c. $O(n)$
d. $O(n*m^2)$
e. $O(2m)$

7) Rekurzija

1) Koju vrijednost će vratiti funkcija `rek` ako ju pozovemo sa parametrom `5 - rek(5);`?

```
int rek(int i){
	int r;
	r=(i>1)?(i+rek(i-1)):(1);
	return r;
}
```

   a. 1
   b. 15
   c. 14
   d. 5
   e. 4

2) Što se ispisuje sljedećim programom:

```
int f(int n){
	if(n<=0) return 0;
	if(n%2) return 1+f(n-1);
	return n+f(n-1);
}
void main(){
	printf("%d", f(5));
}
```

   a. Greška - rekurzija se beskonačno poziva.
   b. 5
   c. 15
   d. 11
   e. 9

3) Koju će vrijednost vratiti funkcija `func` ako je se pozove s `func(polje,3,2);` a polje je deklarirano kao:

```
	polje[3][3]={1,2,3,1,2,3,1,2,3};
int func(int *p, int n, int i){
	int pom;
	pom=p[i]+p[n-i];
	if(i<=0) return pom;
	else return pom+func(p, n, i-1);
}
```

   a. 6
   b. 9
   c. 12
   d. 10
   e. 0

4) Što se ispisuje nakon izvođenja sljedećeg programskog odsječka:

```
void f(int *p, int n, int *max, int *i){
	if(n==-1) return;
	if(p[n]>*max){
		*max=p[n];
	}
	f(p, n-1, max, i);
	if(p[n]==*max){
		(*i)++;
	}
}
```

```
void main(){
	int p[]={1,10,3,4,10};
	int max=p[0], i=0;
	f(p, 4, &max, &i);
	printf("%d, ",i);
}...
```

a. 1
b. Greška - rekurzija nikad neće završiti.
c. 2
d. Greška - rekurzivna funkcija ne može biti tipa `void`.
e. 0

5) Zadane su međusobno rekurzivne funkcije:

```
ispisl(char *niz, int n){
	if((*niz!='\0') && (n>=0)){
		printf("%d", *niz);
		ispisl(niz+1, n-1)
	}
}
ispis2(char *niz, int n){
	if((*(niz+n)!='\0') && (n>=0)){
		printf("%d", *(niz+n));
		ispisl(niz, n-1);
	}
}
```

Što će se ispisati sljedećim pozivom funkcije:

```
ispisl("ABECEDA", 5);
```

a. ADBEEC
b. ABECED
c. ABECE
d. ABECEDA
e. ABECEDA

6) Koliko će biti poziva funkcije `func` ako se funkcija pozove s `func(10)`?

```
void func(int n){
	if(n!=1){
		func(n-1);
		func(n-1);
	}
}
```

a. 10^2-1=99
b. 2^{10}=1024
c. 2^{10}-1=1023
d. 10^2
e. Funkcija je neispravno napisana – rekurzija ulazi u beskonačnu petlju jer nema osnovnog slučaja.

7) Sljedeća funkcija implementira ugrađenu funkciju:

```
int mystery(char *s){
	if (*s=='\0') return 0;
	else return(1+mystery(s+1));
}
```

a. `strlen()`
b. `strcmp()`
c. `strncmp()`
d. `strstr()`

e. `strchr()`

8) Koja od sljedećih tvrdnji nije istinita:
   a. Svaka rekurzivna funkcija se može implementirati iterativno
   b. Svaka rekurzivna funkcija mora imati „osnovni“ slučaj.
   c. Rekurzive funkcije su spore.
   d. Rekurzivni programi su kratki.
   e. Svaki sljedeći rekurzivni poziv ne mora nužno težiti k „osnovnom slučaju“.

9) Što ispisuje sljedeći program:

```
void f(int n){
     if(n==2) return;
     if(n==5) return;
     f(n-2);
     printf("%d", n);
}
void main(){
      f(9);
      getch();
}
```

   a. 2579
   b. 97
   c. 9752
   d. 99
   e. 79

10) Što radi sljedeća funkcija:

```
long f(int poc, int kraj){
      if(poc<=kraj){
            if(poc%2==1) return poc+f(poc+1, kraj);
            else f(poc+1, kraj);
      }
      else return 0;
}
```

   a. Sumira sve neparne brojeve u intervalu `[poc,kraj]`.
   b. Sumira sve prim brojeve u intervalu `[poc,kraj]`.
   c. Sumira sve parne brojeve u intervalu `[poc,kraj]`.
   d. Sumira sve brojeve u intervalu `[poc,kraj]`.
   e. Uvijek vraća vrijednost 0 bez obzira na raspon intervala `[poc,kraj]`.

11) Što će ispisati sljedeći program:

```
void r(int n){
      if(n<=3) return;
      printf("%2d", n);
      r(n-1);
      r(n-2);
      printf("%2d", n);
      return;
}
int main(){
      r(6);
}
```

a. `6 5 6 5 5 4 4 6`
b. `6 5 4 4 5 4 4 6`
c. `6 6 5 5 4 4`
d. `6 5 4 5 4 4`
e. Neće ispisati ništa.

12) Pod pojmom rekurzije podrazumijeva se:
a. Potprogram mora pozvati sistemsku funkciju time.
b. U program se mora uključiti datoteka s zaglavljem `rec.h`.
c. Potprogram se mora zvati recursion.
d. Potprogram ne prima nikakve ulazne parametre.
e. Potprogram poziva sam sebe uz konačan broj poziva.

2.1. Bubble sort

```
#include <stdio.h>

void Zamijeni(int *prvi, int *drugi){
	int temp;
	temp=*prvi;
	*prvi=*drugi;
	*drugi=temp;
}

void Ispisi(int A[], int N){
	int i;
	for(i=0; i<N; i++)
		printf("%3d", A[i]);
	printf("\n\n");
}
//SLOŽENOST: O(n^2)
//NAJGORI SLUČAJ: naopako sortiran niz - O(n^2)
//NAJBOLJI SLUČAJ: već sortiran niz - O(n)
//STABILAN: Da
//zamijenimo susjedne elemente ako nisu u dobrom redoslijedu
void BubbleSort(int A[], int N) {
	int i, j;
	for (i=0; i<N-1; i++){
		printf("  %d. prolaz\n\n", i+1);
		for (j=0; j<N-1-i; j++){
			if (A[j+1] < A[j]) {
				Zamijeni(&A[j], &A[j+1]);
				Ispisi(A, N);
			}
		}
	}
}

int main(){
	int i, n;
	int niz[100];
	printf("Koliko brojeva zelite sortirati: ");
	scanf("%d", &n);
	printf("Unesite niz cijelih brojeva: ");

	for(i=0; i<n; i++)
		scanf("%d", &niz[i]);
	printf("\n");

	Ispisi(niz, n);
	BubbleSort(niz, n);

	return 0;
}
```

4	8	2	1	5	7	3	9	6
1. prolaz								
4	2	8	1	5	7	3	9	6
4	2	1	8	5	7	3	9	6
4	2	1	5	8	7	3	9	6
4	2	1	5	7	8	3	9	6
4	2	1	5	7	3	8	9	6
4	2	1	5	7	3	8	6	9
2. prolaz								
2	4	1	5	7	3	8	6	9
2	1	4	5	7	3	8	6	9
2	1	4	5	3	7	8	6	9
2	1	4	5	3	7	6	8	9
3. prolaz								
1	2	4	5	3	7	6	8	9
1	2	4	3	5	7	6	8	9
1	2	4	3	5	6	7	8	9
4. prolaz								
1	2	3	4	5	6	7	8	9

7	2	9	4	8	5	3	6	1
1. prolaz								
2	7	9	4	8	5	3	6	1
2	7	4	9	8	5	3	6	1
2	7	4	8	9	5	3	6	1
2	7	4	8	5	9	3	6	1
2	7	4	8	5	3	9	6	1
2	7	4	8	5	3	6	9	1
2	7	4	8	5	3	6	1	9
2. prolaz								
2	4	7	8	5	3	6	1	9
2	4	7	5	8	3	6	1	9
2	4	7	5	3	8	6	1	9
2	4	7	5	3	6	8	1	9
2	4	7	5	3	6	1	8	9
3. prolaz								
2	4	5	7	3	6	1	8	9
2	4	5	3	7	6	1	8	9
2	4	5	3	6	7	1	8	9
2	4	5	3	6	1	7	8	9
4. prolaz								
2	4	3	5	6	1	7	8	9
2	4	3	5	1	6	7	8	9
5. prolaz								
2	3	4	5	1	6	7	8	9
2	3	4	1	5	6	7	8	9
6. prolaz								
2	3	1	4	5	6	7	8	9
7. prolaz								
2	1	3	4	5	6	7	8	9
8. prolaz								
1	2	3	4	5	6	7	8	9

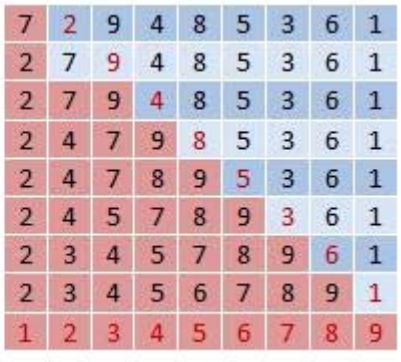
2.2. Poboljšani bubble sort

```
#include <stdio.h>

void Zamijeni(int *prvi, int *drugi){
	int temp;
	temp=*prvi;
	*prvi=*drugi;
	*drugi=temp;
}

void Ispisi(int A[], int N){
	int i;
	for(i=0; i<N; i++)
		printf("%3d", A[i]);
	printf("\n\n");
}
//SLOŽENOST: O(n^2)
//NAJGORI SLUČAJ: naopako sortiran niz - O(n^2)
//NAJBOLJI SLUČAJ: već sortiran niz - O(1)
//STABILAN: Da
//zamijeniti susjedne elemente ako nisu u dobrom redoslijedu
//POBOLJŠANJE: ako u nekom prolazu nije bilo zamjene, niz je sortiran
void BubbleSort (int A[], int N) {
  int i, j, BilaZamjena;
  for (i = 0, BilaZamjena = 1; BilaZamjena; i++) {
    BilaZamjena = 0;
    for (j = 0; j < N-1-i; j++) {
      if (A[j+1] < A[j]) {
        Zamijeni (&A[j], &A[j+1]);
        BilaZamjena = 1;
        Ispisi(A, N);
      }
    }
  }
}


int main(){
	int i, n;
	int niz[100];
	printf("Koliko brojeva zelite sortirati: ");
	scanf("%d", &n);
	printf("Unesite niz cijelih brojeva: ");

	for(i=0; i<n; i++)
		scanf("%d", &niz[i]);
	printf("\n");

	Ispisi(niz, n);
	BubbleSort(niz, n);

	return 0;
}
```

2.3. Insertion sort

```c
#include <stdio.h>

void Ispisi(int A[], int N){
	int i;
	for(i=0; i<N; i++)
		printf("%3d", A[i]);
	printf("\n\n");
}

//SLOŽENOST: O(n^2)
//NAJGORI SLUČAJ: naopako sortiran niz - O(n^2)
//NAJBOLJI SLUČAJ: već sortiran niz - O(n)
//STABILAN: Da
//niz podijelimo na sortirani i nesortirani dio, u svakom
//koraku sortirani dio proširujemo tako da u njega na ispravno
//mjesto ubacimo element iz nesortiranog dijela niza
void InsertionSort (int A[], int N){
	int i, j;
	int pom;
	for(i=1; i<N; i++){
		pom = A[i];
		for (j=i; j>=1 && A[j-1]>pom; j--)
			A[j]=A[j-1];
		A[j]=pom;
		Ispisi(A, N);
	}
}

int main(){
	int i, n;
	int niz[100];
	printf("Koliko brojeva zelite sortirati: ");
	scanf("%d", &n);
	printf("Unesite niz cijelih brojeva: ");

	for(i=0; i<n; i++)
		scanf("%d", &niz[i]);
	printf("\n");

	Ispisi(niz, n);
	InsertionSort(niz, n);

	return 0;
}
```

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



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

2.4. Shell sort

```
#include <stdio.h>

void Ispisi(int A[], int N){
	int i;
	for(i=0; i<N; i++)
		printf("%3d", A[i]);
	printf("\n\n");
}

//SLOŽENOST: O(n^1.25)
//NAJGORI SLUČAJ: naopako sortiran niz - O(n^1.5)
//NAJBOLJI SLUČAJ: već sortiran niz - O(n*log(n))
//STABILAN: Ne
//pronađemo najmanji element niza i zamijenimo ga s
//prvim elementom niza, ponavljamo s ostatkom niza
void ShellSort(int A[], int N){
	int i, j, korak, pom;
	for(korak=N/2; korak>0; korak/=2) {
		printf("  korak = %d\n\n", korak);
		for(i=korak; i<N; i++){
			pom=A[i];
			Ispisi(A, N);
			for(j=i; j>=korak && A[j-korak]>pom; j-=korak){
				A[j]=A[j-korak];
			}
			A[j]=pom;
		}
	}
}

int main(){
	int i, n;
	int niz[100];
	printf("Koliko brojeva zelite sortirati: ");
	scanf("%d", &n);
	printf("Unesite niz cijelih brojeva: ");

	for(i=0; i<n; i++)
		scanf("%d", &niz[i]);
	printf("\n");

	Ispisi(niz, n);
	ShellSort(niz, n);
	Ispisi(niz, n);

	return 0;
}
```

4	8	2	1	5	7	3	9	6
korak = 4								
4	8	2	1	5	7	3	9	6
4	8	2	1	5	7	3	9	6
4	7	2	1	5	8	3	9	6
4	7	2	1	5	8	3	9	6
4	7	2	1	5	8	3	9	6
korak = 2								
4	7	2	1	5	8	3	9	6
2	7	4	1	5	8	3	9	6
2	1	4	7	5	8	3	9	6
2	1	4	7	5	8	3	9	6
2	1	4	7	5	8	3	9	6
2	1	3	7	4	8	5	9	6
2	1	3	7	4	8	5	9	6
korak = 1								
2	1	3	7	4	8	5	9	6
1	2	3	7	4	8	5	9	6
1	2	3	7	4	8	5	9	6
1	2	3	7	4	8	5	9	6
1	2	3	4	7	8	5	9	6
1	2	3	4	7	8	5	9	6
1	2	3	4	5	7	8	9	6
1	2	3	4	5	7	8	9	6
1	2	3	4	5	6	7	8	9

7	2	9	4	8	5	3	6	1
korak = 4								
7	2	9	4	8	5	3	6	1
7	2	9	4	8	5	3	6	1
7	2	9	4	8	5	3	6	1
7	2	3	4	8	5	9	6	1
7	2	3	4	8	5	9	6	1
korak = 2								
1	2	3	4	7	5	9	6	8
1	2	3	4	7	5	9	6	8
1	2	3	4	7	5	9	6	8
1	2	3	4	7	5	9	6	8
1	2	3	4	7	5	9	6	8
1	2	3	4	7	5	9	6	8
1	2	3	4	7	5	9	6	8
korak = 1								
1	2	3	4	7	5	8	6	9
1	2	3	4	7	5	8	6	9
1	2	3	4	7	5	8	6	9
1	2	3	4	7	5	8	6	9
1	2	3	4	7	5	8	6	9
1	2	3	4	5	7	8	6	9
1	2	3	4	5	7	8	6	9
1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9

2.5. Merge sort

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

void Ispisi(int A[], int N){
	int i;
	for(i=0; i<N; i++)
		printf("%3d", A[i]);
	printf("\n\n");
}

//SLOŽENOST: O(n^2)
//NAJBOLJI SLUČAJ: već sortiran niz - O(n)
//NAJGORI SLUČAJ: naopako sortiran niz - O(n^2)
//nesortirani niz podijelimo na dva niza podjednake veličine,
//svaki podniz sortiramo rekurzivno, dok ne dobijemo niz od
//1 elementa koji je sortiran
//spoje se dva sortirana podniza u sortirani niz na temelju
//dva sortirana polja (A i B) puni se treće C
// udruzivanje LPoz:LijeviKraj i DPoz:DesniKraj
void Merge(int A[], int PomPolje[], int LPoz, int DPoz, int DesniKraj){
	int i, LijeviKraj, BrojClanova, PomPoz;
	LijeviKraj=DPoz-1;
	PomPoz=LPoz;
	BrojClanova=DesniKraj-LPoz+1;
	//glavna pelja
	while(LPoz<=LijeviKraj && DPoz<=DesniKraj){
		if (A[LPoz]<=A[DPoz])
		PomPolje[PomPoz++]=A[LPoz++];
    else
		PomPolje[PomPoz++]=A[DPoz++];
	}
	while(LPoz<=LijeviKraj)
		//kopiramo ostatak prve polovice
		PomPolje[PomPoz++]=A[LPoz++];
	while(DPoz<=DesniKraj)
		//kopiramo ostatak druge polovice
		PomPolje[PomPoz++]=A[DPoz++];
	for (i=0;i<BrojClanova; i++, DesniKraj--)
    //kopiramo PomPolje natrag
    A[DesniKraj]=PomPolje[DesniKraj];
}
//MergeSort - rekurzivno sortiranje podpolja
void MSort(int A[], int PomPolje[], int lijevo, int desno){
	int sredina;
	if (lijevo<desno){
	sredina=(lijevo+desno)/2;
    MSort(A,PomPolje,lijevo,sredina);
    MSort(A,PomPolje,sredina+1,desno);
    Merge(A,PomPolje,lijevo,sredina+1,desno);
  }
}

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

```
        exit(1);
    }
}

int main(){
        int i, n;
        int niz[100];
        printf("Koliko brojeva zelite sortirati: ");
        scanf("%d", &n);
        printf("Unesite niz cijelih brojeva: ");

        for(i=0; i<n; i++)
                scanf("%d", &niz[i]);
        printf("\n");

        Ispisi(niz, n);
        MergeSort(niz, n);
        Ispisi(niz, n);

        return 0;
}
```

2.6. Quick sort – medijan

```
#include <stdio.h>
#define Cutoff (3)

void Zamijeni(int *prvi, int *drugi){
	int temp;
	temp=*prvi;
	*prvi=*drugi;
	*drugi=temp;
}

void Ispisi(int A[], int N){
	int i;
	for(i=0; i<N; i++)
		printf("%3d", A[i]);
	printf("\n\n");
}

void InsertionSort (int A[], int N){
  int i, j;
  int pom;
  for(i=1; i<N; i++){
    pom=A[i];
    for(j=i; j>=1 && A[j-1]>pom; j--)
      A[j]=A[j-1];
    A[j]=pom;
  }
}
//SLOŽENOST:
//NAJGORI SLUČAJ:
//NAJBOLJI SLUČAJ:
//(1)ako je broj članova polja S jednak 0 ili 1, vraćamo se
//u glavni program
//(2)odabremo bilo koji član v u polju S - stožer (pivot)
//(3)podijelimo preostale članove polja S, S\{v} u dva odvojena
//skupa:
////S1={xeS\{v}/x<=v}(sve što je manje od stožera, preseli lijevo)
////S2={xeS\{v}/x>=v}(sve što je veće od stožera, preseli desno)
//(4)vratimo niz sastavljen od {quicksort(S1), v, quicksort S2)}

//QuickSort - medijan i stozer
//vratimo medijan od lijevo, sredina i desno, poredamo
//ih i sakrijemo stozer
int medijan3(int A[], int lijevo, int desno){
	int sredina=(lijevo+desno)/2;
	if(A[lijevo]>A[sredina])
		Zamijeni(&A[lijevo], &A[sredina]);
	if(A[lijevo]>A[desno])
		Zamijeni(&A[lijevo], &A[desno]);
	if(A[sredina]>A[desno])
		Zamijeni(&A[sredina], &A[desno]);
	//sada je: A[lijevo]<=A[sredina]<=A[desno]
	//sakrijemo stožer
	Zamijeni(&A[sredina], &A[desno-1]);
	//vraćamo stožer
	return A[desno-1];
}
//QuickSort - rekurzivno sortiramo podpolje
void Qsort1(int A[], int lijevo, int desno){
	int i, j;
	int stozer;
	if(lijevo+Cutoff<=desno){
```

```
                stozer=medijan3(A, lijevo, desno);
                i=lijevo;
                j=desno-1;
                while(1){
                        while(A[++i]<stozer);
                        while(A[--j]>stozer);
                        if(i<j){
                                Zamijeni(&A[i], &A[j]);
                        }else{
                                break;
                        }
                }
                //obnavljamo stozer
                Zamijeni(&A[i], &A[desno-1]);
                Qsort1 (A, lijevo, i-1);
                Qsort1 (A, i+1, desno);
        }else{
                //sortiramo podpolje
                InsertionSort(A+lijevo, desno-lijevo+1);
        }
}
//QuickSort
void QuickSort1(int A[], int N){
  Qsort1(A, 0, N-1);
}

int main(){
        int i, n;
        int niz[100];
        printf("Koliko brojeva zelite sortirati: ");
        scanf("%d", &n);
        printf("Unesite niz cijelih brojeva: ");

        for(i=0; i<n; i++)
                scanf("%d", &niz[i]);
        printf("\n");

        Ispisi(niz, n);
        QuickSort1(niz, n);
        Ispisi(niz, n);

        return 0;
}
```

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

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

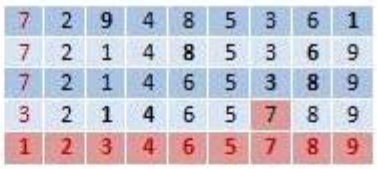
9	10	8	6	1	4	3	5	2	7
1	10	8	6	7	4	3	5	2	9
1	10	8	6	2	4	3	5	7	9
1	5	8	6	2	4	3	10	7	9
1	5	3	6	2	4	8	10	7	9
1	5	3	6	2	4	7	10	8	9
1	5	2	6	3	4	7	10	8	9
1	2	5	6	3	4	7	10	8	9
1	2	3	6	5	4	7	10	8	9
1	2	3	4	5	6	7	10	8	9
1	2	3	5	6	4	7	8	9	10

2.7. Quick sort – prvi element

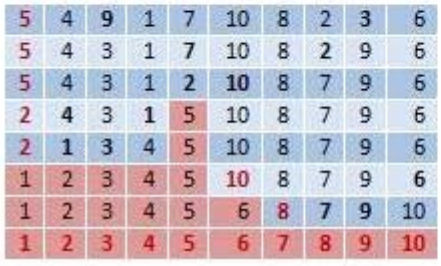
```
#include <stdio.h>

void Zamijeni(int *prvi, int *drugi){
	int temp;
	temp=*prvi;
	*prvi=*drugi;
	*drugi=temp;
}

void Ispisi(int A[], int N){
	int i;
	for(i=0; i<N; i++)
		printf("%3d", A[i]);
	printf("\n\n");
}
//SLOŽENOST: O(n*log(n))
//NAJGORI SLUČAJ: O(n^2)
//NAJBOLJI SLUČAJ: O(n)
//STABILAN: Ne
//(1)ako je broj članova polja S jednak 0 ili 1, vraćamo se
//u glavni program
//(2)odabremo bilo koji član v u polju S - stožer (pivot)
//(3)podijelimo preostale članove polja S, S\{v} u dva odvojena
//skupa:
////S1={xeS\{v}/x<=v}(sve što je manje od stožera, preseli lijevo)
////S2={xeS\{v}/x>=v}(sve što je veće od stožera, preseli desno)
//(4)vratimo niz sastavljen od {quicksort(S1), v, quicksort S2)}
//QuickSort - medijan i stozer
//vratimo medijan od lijevo, sredina i desno, poredamo
//ih i sakrijemo stozer
//Quicksort, stožer je prvi element
void Qsort2(int A[], int lijevo, int desno){
	int i, j;
    i=lijevo+1;
    j=desno;

    if(lijevo>=desno) return;

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

//QuickSort, stožer je prvi element
void QuickSort2(int A[], int N){
```

```
  Qsort2(A, 0, N-1);
}

int main(){
	int i, n;
	int niz[100];
	printf("Koliko brojeva zelite sortirati: ");
	scanf("%d", &n);
	printf("Unesite niz cijelih brojeva: ");

	for(i=0; i<n; i++)
		scanf("%d", &niz[i]);
	printf("\n");

	Ispisi(niz, n);
	QuickSort2(niz, n);
	Ispisi(niz, n);

	return 0;
}
```

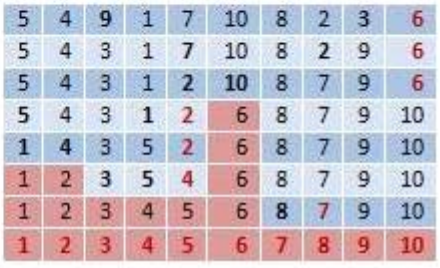
4	8	2	1	5	7	3	9	6
4	3	2	1	5	7	8	9	6
1	3	2	4	5	7	8	9	6
1	2	3	4	5	7	8	9	6
1	2	3	4	5	7	8	9	6
1	2	3	4	5	7	6	9	8
1	2	3	4	5	6	7	9	8
1	2	3	4	5	6	7	8	9



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



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



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

2.8. Selection sort

```
#include <stdio.h>

void Zamijeni(int *prvi, int *drugi){
	int temp;
	temp=*prvi;
	*prvi=*drugi;
	*drugi=temp;
}

void Ispisi(int A[], int N){
	int i;
	for(i=0; i<N; i++)
		printf("%3d", A[i]);
	printf("\n\n");
}
//SLOŽENOST: O(n^2)
//NAJGORI SLUČAJ: naopako sortiran niz - O(n^2)
//NAJBOLJI SLUČAJ: već sortiran niz - O(n^2)
//STABILAN: Ne
//pronađemo najmanji element niza i zamijenimo ga s
//prvim elementom niza, ponavljamo s ostatkom niza
void SelectionSort(int A[], int N){
	int i, j, min;
	for(i=0; i<N; i++){
		min=i;
		for(j=i+1; j<N; j++){
			if (A[j]<A[min])
				min = j;
		}
		Zamijeni(&A[i], &A[min]);
		Ispisi(A, N);
	}
}

int main(){
	int i, n;
	int niz[100];
	printf("Koliko brojeva zelite sortirati: ");
	scanf("%d", &n);
	printf("Unesite niz cijelih brojeva: ");

	for(i=0; i<n; i++)
		scanf("%d", &niz[i]);
	printf("\n");

	Ispisi(niz, n);
	SelectionSort(niz, n);

	return 0;
}
```

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	7	8	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
1	2	3	4	5	6	7	8	9

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

2.9. Objektno orijentirana izvedba stoga

Stog je struktura podataka u kojoj se posljednji pohranjeni podatak uzima prvi. U literaturi ćete često naići na oznaku LIFO, što je kratica od *Last In First Out*.
Potrebne operacije koje treba ostvariti za rad sa stogom su:

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

Osim statičkim poljem stog se može ostvariti i dinamičkim poljem, te povezanom listom. Veliki nedostatak implementacije stoga statičkim poljem je mogućnost prepunjenja jer imamo polje ograničene veličine (*MAXSTOG*).

Cilj ovog poglavlja nije implementirati vlastite klase za stog već izvježbati osnovne funkcije rada sa stogom te shvatiti način rada stoga i njegove primjene.

Zbog toga smo za vas pripremili datoteke **Stog.cpp** i **Stog.h** koje je potrebno uključiti u projekte koje ćete raditi. Stog je ovdje implementiran jednostruko povezanom listom pa gotovo da nema ograničenja na broj elemenata koje možemo staviti na njega (*osim fizičke količine memorije*). Programe morate pisati u programskom jeziku C++ jer C nije objektno orjentiran jezik.

U C++ programskom jeziku postoji biblioteka Standard Template Library (*STL*) u kojoj se nalazi zaglvalje `stack`. Uključivši to zaglavlje i odgovarajući imenički prostor naredbom `using namespace std;` dobijate već implementirane funkcije za rad sa stogom.

Zajednički dio za sve zadatke:
Za spremanje cijelih brojeva na stog definirana je klasa `Stog` koja ima jedan konstruktor i tri javne funkcije:

```
Stog::Stog();
void Stog::Stavi(int element);
void Stog::Skini(int *element);
int Stog::Prazan();
```

Funkcija `Prazan` vraća 1 ukoliko je stog prazan, a 0 inače. Za klasu `Stog` nije definiran *copy konstruktor*, a možete pretpostaviti da u svakom objektu klase Stog ima dovoljno mjesta za dodavanje novih elemenata za stog. Pokušaj skidanja elementa iz praznog stoga, neće dovesti do pogreške. Ukoliko je potrebno možete koristiti pomoćne stogove. Ukoliko drugačije nije navedeno, smijete koristiti pomoćne stogove, ali ne i dodatna polja.

StogOO_1 - Napisati funkciju koja će iz stoga izbaciti sve brojeve iz intervala [2,5] pri čemu redoslijed ostalih elemenata ostaje nepromijenjen. Funkcija vraća broj izbaćenih brojeva te ima sljedeći prototip:

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

Napisati glavni program koji će na stog staviti brojeve od 0 do 7 i pozvati funkciju *izbaci*. Za vježbu ispisati na zaslon svako stavljanje i skidanje sa stoga kako bi mogli pratiti redoslijed operacija.

```
#include <stdio.h>
#include "Stog.h"

int izbaci(Stog *stog){
	int element, br=0;
	//deklariramo pomoćni stog koji će nam služiti za
	//premještanje elemenata
	Stog pomocni;
	//dok je stog pun, skidamo elemente i u pomoćnom
	//stogu pamtimo one koji nisu iz zadanog intervala
	//redoslijed na pomoćnom stogu je obrnut od početnog
	while(!stog->Prazan()){
		stog->Skini(&element);
		printf("STOG - skidam: %d\n", element);
		if(element<2 || element>5){
			pomocni.Stavi(element);
			printf("POMOCNI - stavljam: %d\n", element);
		}
		else
			br++;
	}
	//vraćamo elemente koji nisu izbačeni na početni stog
	//uz očuvani  redoslijed
	while(!pomocni.Prazan()){
		pomocni.Skini(&element);
		printf("POMOCNI - skidam: %d\n", element);
		stog->Stavi(element);
		printf("STOG - stavljam: %d\n", element);
	}
	return br;
}

int main(){
	int i;
	Stog *stog = new Stog();

	for(i=0; i<7; i++){
		stog->Stavi(i);
		printf("STOG - stavljam: %d\n", i);
	}
	printf("\nIzbaceno elemenata: %d\n", izbaci(stog));
	delete stog;
	return 0;
}
```

StogOO_2 - Napisati funkciju koja će stvoriti novi stog koji će sadržavati sve one brojeve iz zadanog stoga za koje funkcija čiji je prototip `int prosti(int n)` vrati istinu. Poredak prostih brojeva u novom stogu nije bitan. Zadani stog mora na kraju ostati nepromijenjen. Funkcija mora imati prototip:

```
Stog *kopiraj_proste(Stog *stog);
```

Također napisati funkciju *prosti* te glavni program u kojem ćete na stog staviti brojeve od 1 do 25 te pozvati funkciju *kopiraj_proste*.

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

int prosti(int x){
	if (x==1 || x==2) return 1;
	//Prvo ispitamo djeljivost s 2
	if (x%2==0) return false;
	//Preostaje nam za ispitati djeljivost s neparnim brojevima vecim od 2
	//Osjetno ubrzanje mozemo postici ako se sjetimo da je iteraciju dovoljno
provesti do sqrt(x)
	for (int i=3; i<=(int)sqrt((double)(x)); i++)
		if (x%i==0) return 0;
	return 1;
}

Stog *kopiraj_proste(Stog *stog){
	//deklariramo novi stog kojeg ćemo vratiti preko imena funkcije
	Stog *novi = new Stog();
	//deklariramo pomoćni stog koji će nam služiti za
	//premještanje elemenata
	Stog pomocni;
	int element;
	//kako bi očuvali raspored elemenata na početnom stogu skidamo elemente s
njega
	//te ih stavljamo na pomoćni stog
	while(!stog->Prazan()){
		stog->Skini(&element);
		pomocni.Stavi(element);
	}
	//skidamo elemente s pomoćnog stoga te ih stavljamo na početni stog čime smo
	//sačuvali njihov prvotni raspored. Ako je broj koje skinemo prost stavljamo
	//ga na stog kojeg ćemo vratiti preko imena funkcije
	while(!pomocni.Prazan()){
		pomocni.Skini(&element);
		stog->Stavi(element);
		if (prosti(element)){
			novi->Stavi(element);
			printf("Kopiram: %d\n", element);
		}
	}
	return novi;
}

int main(){
	int i;
	Stog *stog = new Stog();
	Stog *kopirani = new Stog();

	for(i=1; i<25; i++)
		stog->Stavi(i);

	kopirani=kopiraj_proste(stog);
	//uništavamo dinamički stvorene stogove
	delete stog;
	delete kopirani;
	return 0;
}
```

StogOO_3 - Napišite funkciju koja će na osnovu ulaznih stogova *stog1* i *stog2* stvoriti novi stog koristeći pritom pravilo da se za novi element u novom stogu postavi zbroj od elemenata na vrhu stogova *stog1* i *stog2*. Ukoliko se jedan od stogova isprazni, onda preostale elemente iz drugog stoga treba nadodati na novi stog.

Ulazni stogovi moraju nakon završetka funkcije ostati nepromijenjeni. Funkcija mora imati prototip:

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

Napisati i glavni program u kojem na prvi stog treba staviti brojeve od 1 do 20, a na drugi brojeve od 15 do 30. Pozvati funkciju zbroji te s novog stoga skinuti i ispisati sve elemente.

```
#include <stdio.h>
#include "Stog.h"

Stog *zbroji(Stog *stog1, Stog *stog2){
	int element1, element2;
	//želimo očuvati redoslijed na prvom i drugom stogu pa nam zato trebaju
	//2 pomoća stoga. Novi stog vraćamo preko imena funkcije
	Stog pomocni1, pomocni2, *novi;
	novi = new Stog();
	//dok na prvom i drugom stogu ima elemenata skini gornji element iz svakog
	//te njihv zbroj stavi na novi stog
	while(!stog1->Prazan() && !stog2->Prazan()){
		stog1->Skini(&element1);
		stog2->Skini(&element2);
		pomocni1.Stavi(element1);
		pomocni2.Stavi(element2);
		novi->Stavi(element1+element2);
	}
	//kad se prvi ili drugi stog isprazni preostale
	//elemente dodajemo na novi stog
	while(!stog1->Prazan()){
		stog1->Skini(&element1);
		novi->Stavi(element1);
		pomocni1.Stavi(element1);
	}
	while(!stog2->Prazan()){
		stog2->Skini(&element2);
		novi->Stavi(element2);
		pomocni2.Stavi(element2);
	}
	//s pomoćnih stogova skidamo elemente i stavljamo ih na
	//početni odgovarajući stog, time osiguravamo nepromijenjen raspored
	while(!pomocni1.Prazan()){
		pomocni1.Skini(&element1);
		stog1->Stavi(element1);
	}
	while(!pomocni2.Prazan()){
		pomocni2.Skini(&element2);
		stog2->Stavi(element2);
	}
	return novi;
}

int main(){
	int i, element;
	Stog *stog1 = new Stog();
	Stog *stog2 = new Stog();
	Stog *zbrojeni = new Stog();

	for(i=1; i<20; i++) stog1->Stavi(i);
	for(i=15; i<30; i++) stog2->Stavi(i);

	zbrojeni=zbroji(stog1, stog2);
	while(!zbrojeni->Prazan()){
		zbrojeni->Skini(&element);
```

```
                printf("Skidam: %d\n", element);
        }
        //uništavamo dinamički stvorene stogove
        delete stog1;
        delete stog2;
        delete zbrojeni;
        return 0;
}
```

StogOO_4 - Napišite funkciju koja će iz stoga izbaciti sve one brojeve koji se pojavljuju dva ili više puta. Funkcija ima sljedeći prototip:

```
Stog *duplikati(Stog *stog);
```

Funkcija preko imena mora vratiti pokazivač na stog u kojem se nalaze svi duplikati. Napišite glavni program koji će na stog staviti 50 brojeva iz intervala [1,50], pozvati funkciju duplikati te potom skinuti i ispisati sve elemente sa stoga duplikati.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "Stog.h"
//skinuti prvi element sa stoga i spremiti ga u x, zatim skidati sve preostale
//elemente sa stoga i uspoređivati ih sa x. Ako skinuti element nije jednak x
//privremeno ga sačuvati na pomoćnom stogu pomocni1, inače označiti da x ima
duplikata
//ako x nije imao duplikata treba ga sačuvati na pomoćnom stogu pomocni2, inače ga
//staviti na novi stog u kojem ćemo pohraniti sve duplikate. Sve elemente
//iz pomocni1 vratiti na početni stog i ponoviti postupak.
Stog *duplikati(Stog *stog){
        Stog pomocni1, pomocni2;
        Stog *novi = new Stog();
        int element, duplikat, x;
        while(!stog->Prazan()){
                //uzimamo prvi element sa stoga
                stog->Skini(&x);
                duplikat=0;
                //uspoređujemo ga s preostalim elementima na stogu
                while(!stog->Prazan()){
                        stog->Skini(&element);
                        if(x==element)
                                duplikat=1;
                        else
                                pomocni1.Stavi(element);
                }
                if(!duplikat)
                        pomocni2.Stavi(x);
                else
                        novi->Stavi(x);
                while(!pomocni1.Prazan()){
                        pomocni1.Skini(&element);
                        stog->Stavi(element);
                }
        }
        while(!pomocni2.Prazan()){
                pomocni2.Skini(&element);
                stog->Stavi(element);
        }
        return novi;
}

int main(){
```

```
        int i, element;
        Stog *stog = new Stog();
        Stog *dupli = new Stog();

        srand((unsigned)time(NULL));

        for(i=0; i<50; i++){
                element=rand()%(50-1+1)+1;
                stog->Stavi(element);
        }

        dupli=duplikati(stog);

        printf("\n*** Stog bez duplikata ***\n\n");
        while(!stog->Prazan()){
                stog->Skini(&element);
                printf("Skidam: %d\n", element);
        }

        printf("\n*** Duplikati ***\n\n");
        while(!dupli->Prazan()){
                dupli->Skini(&element);
                printf("Skidam: %d\n", element);
        }
        //uništavamo dinamički stvorene stogove
        delete stog;
        delete dupli;
        return 0;
}
```

StogOO_5 - Postfix zapis je oblik zapisa matematičkih izraza poznat i pod imenom „Reverse Polish Notation“ (*RPN*). Kada koristimo ovaj način zapisa više nam ne trebaju zagrade za grupiranje izraza. Matematički operatori su zapisani poslije svojih argumenata i zbog toga je ovaj način vrlo pogodan za računanje stogom. Ovakva notacija i način računanja izraza koristi se u mnogim kalkulatorima i programskim jezicima. Vaš zadatak je pročitati matematički izraze iz formatirane datoteke *„postfix.txt“* koja se nalazi na disku te na zaslon računala ispisati sve međurezultate. Koristiti se već implemenitranim stogom (*uključiti zaglavlja Stog.cpp i Stog.h*). Npr.

Zadani niz učitan iz postfix.txt	**Međurezultati**
1 3 3 * +	9, 10
4 8 /	2
7 2 8 / +	4, 11
2 3 2 5 % * +	1, 3, 5

Napomena: Svi zadani nizovi će biti zadani ispravno, brojevi će biti isključivo pozitivni jednoznamenkasti (*zbog jednostavnosti*), a dozvoljene operacije su +, -, *, / i %.

StogOO_6 - Za one koji žele istražiti prednosti koje nudi C++ i pripadna STL biblioteka dajemo sljedeći ogledni primjer rada sa stogom.

```cpp
#include <iostream>
#include <stack>
using namespace std;

int main(){
	stack<int> stog;
	for(int i=1; i<=10; i++){
		stog.push(i);
		cout << "Dodan: " << i << endl;
	}
	cout << endl;
	cout << "Velicina stoga: " << stog.size() << endl;
	cout << "Na vrhu stoga:: " << stog.top() << endl << endl;
	while(!stog.empty()){
		cout << "Skinut: " << stog.top() << endl;
		stog.pop();
	}
	return 0;
}
```

2.10. Stog implementiran statičkim poljem

Stog je struktura podataka u kojoj se posljednji pohranjeni podatak uzima prvi. U literaturi ćete često naići na oznaku LIFO, što je kratica od *Last In First Out*.
Potrebne operacije koje treba ostvariti za rad sa stogom su:

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

Osim statičkim poljem stog se može ostvariti i dinamičkim poljem, te povezanom listom. Veliki nedostatak implementacije stoga statičkim poljem je mogućnost prepunjenja jer imamo polje ograničene veličine (*MAXSTOG*).

Struktura podataka koja opisuje stog realiziran poljem:

```
#define MAXSTOG 100
typedef struct{
	int vrh;
	tip polje[MAXSTOG];
}Stog;
```

Implementacije potrebnih funkcija su:

1) Dodavanje elementa na stog

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

2)Skidanje elementa sa stoga

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

3)Inicijalizacija praznog stoga

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

Zajednički dio za sve zadatke:
Na disku postoji binarna datoteka *automobili.dat* u kojoj su slijedno pohranjeni zapisi o automobilima. Svi zapisi sadrže matični broj (*int*), naziv proizvođača (*niz znakova duljine 20+1*), vrstu

(*niz znakova duljine 20+1*) te godinu proizvodnje automobila(*int*). Strukturu podataka za pohranu zapisa možete definirati na sljedeći način:

```
typedef struct{
	int mbr;
	char proizvodjac[20+1];
	char vrsta[20+1];
	int godina;
}zapis;
```

Šime se bavi transportom automobila za razne proizvođače. Ukrcaj i iskrcaj automobila s njegovog kamiona radi po principu stoga (*LIFO*). Automobil koji je prvi ukrcan iskrcava se posljednji.

StogPoljem – 1 Šime je dobio naredbu od svog poslodavca da isporuči sve automobile proizvedene poslije 2003. godine. Pomozite Šimi da ukrca na svoj kamion odgovarajuće automobile. Napište funkcije za inicijalizaciju stoga te za dodavanje elemenata na stog (*elementi tipa zapis*). Napišite i glavni program koji će koristeći navedene funkcije učitati sve zapise o automobilima iz binarne datoteke *automobili.dat* te one koji su proizvedeni poslije 2003. dodati na stog.
Napomena: Ne zaboravite inicijalizirati stog.

```
#include <stdio.h>
//Najveća moguća veličina polja
#define MAXSTOG 100
//struktura podataka za pojedini automobil
typedef struct{
	int mbr;
	char proizvodjac[20+1];
	char vrsta[20+1];
	int godina;
}zapis;
typedef zapis tip;
//opis strukture podataka stoga
typedef struct{
	int vrh;
	tip polje[MAXSTOG];
}Stog;
//funkcija za inicijalizaciju
void init_stog(Stog *stog){
	stog->vrh = -1;
}
//funkcija za dodavanje elementa na stog
//ako je stog prepunjen vratiti 0, inače povećati vrh za 1
//i na mjesto koje vrh sada indeksira pohraniti novi element
int dodaj(tip element, Stog *stog){
	if (stog->vrh >= MAXSTOG-1) return 0;
	stog->vrh++;
	stog->polje[stog->vrh] = element;
	return 1;
}

int main(){
	FILE *binarna;
	//deklaracija stoga
	Stog stog;
	zapis z;
	//obavezno inicijalizirati stog prije rada s njime
	init_stog(&stog);
	binarna=fopen("automobili.dat", "rb");

	printf("\nUkrcavam u kamion...\n\n");
	while(fread(&z, sizeof(zapis), 1, binarna)){
		if(z.godina>2003){
			dodaj(z, &stog);
```

```
                    printf("%02d %s %s %d\n", z.mbr, z.proizvodjac, z.vrsta,
z.godina);
                }
        }
        fclose(binarna);
        return 0;
}
```

StogPoljem – 2 Šime je danas zatrpan poslom. Sve automobile koje ima mora ukrcati na kamion, dovesti ih do autokuće i iskrcati ih. Napišite funkcije za inicijalizaciju stoga realiziranog poljem, funkciju za dodavanje zapisa na stog te funkciju za skidanje sa stoga . Napisati i glavni program u kojem će se učitati svi zapisi o automobilima u skladištu (*automobili.dat*) te ih ukrcati na kamion, a poslije i iskrcati.

```
#include <stdio.h>
//Najveća moguća veličina polja
#define MAXSTOG 100
//struktura podataka za pojedini automobil
typedef struct{
        int mbr;
        char proizvodjac[20+1];
        char vrsta[20+1];
        int godina;
}zapis;
typedef zapis tip;
//opis strukture podataka stoga
typedef struct{
        int vrh;
        tip polje[MAXSTOG];
}Stog;
//funkcija za inicijalizaciju
void init_stog(Stog *stog){
        stog->vrh = -1;
}
//funkcija za dodavanje elementa na stog
//ako je stog prepunjen vratiti 0, inače povećati vrh za 1
//i na mjesto koje vrh sada indeksira pohraniti novi element
int dodaj(tip element, Stog *stog){
        if (stog->vrh >= MAXSTOG-1) return 0;
        stog->vrh++;
        stog->polje[stog->vrh] = element;
        return 1;
}
//funkcija za skidanje elementa sa stoga
//ako na stogu nema više elemenata funkcija vraća 0
//skinuti element se vraća preko adrese, a vrh stoga se umanjuje za 1
int skini(tip *element, Stog *stog){
        if (stog->vrh < 0) return 0;
        *element = stog->polje[stog->vrh];
        stog->vrh--;
        return 1;
}

int main(){
        FILE *binarna;
        //deklaracija stoga
        Stog stog;
        zapis z;
        //obavezno inicijalizirati stog prije rada s njime
        init_stog(&stog);
        binarna=fopen("automobili.dat", "rb");
```

```
	printf("\nUkrcavam u kamion...\n\n");
	while(fread(&z, sizeof(zapis), 1, binarna)){
		dodaj(z, &stog);
		printf("%02d %s %s %d\n", z.mbr, z.proizvodjac, z.vrsta, z.godina);
	}
	printf("\nIskrcavam iz kamiona...\n\n");
	while(skini(&z, &stog))
		printf("%02d %s %s %d\n", z.mbr, z.proizvodjac, z.vrsta, z.godina);

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

StogPoljem – 3 Poduzetni Šime mora ponovno ukrcati sve automobile koje ima, no ovog puta mora u Renault centru iskrcati sve automobile marke Renault na način da raspored automobila u kamionu ostane nepromijenjen. Nakon toga Šime odvozi preostale automobile u sljedeću auto kuću gdje ih sve iskrcava. Napisati sve potrebne funkcije za rad sa stogom realiziranog statičkim poljem. Napisati i funkciju prototipa:

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

koja će korištenjem **pomoćnog stoga** iskrcati sve automobile iz kamiona te na kamion vratiti samo one čiji proizvođac nije Renault. U glavnom programu dodati sve zapise o automobilima iz binarne datoteke *automobili.dat* na stog, pozvati funkciju *renault*, a nakon toga skinuti sve zapise sa stoga.

```
#include <stdio.h>
#include <string.h>
//Najveća moguća veličina polja
#define MAXSTOG 100
//struktura podataka za pojedini automobil
typedef struct{
	int mbr;
	char proizvodjac[20+1];
	char vrsta[20+1];
	int godina;
}zapis;
typedef zapis tip;
//opis strukture podataka stoga
typedef struct{
	int vrh;
	tip polje[MAXSTOG];
}Stog;
//funkcija za inicijalizaciju
void init_stog(Stog *stog){
	stog->vrh = -1;
}
//funkcija za dodavanje elementa na stog
//ako je stog prepunjen vratiti 0, inače povećati vrh za 1
//i na mjesto koje vrh sada indeksira pohraniti novi element
int dodaj(tip element, Stog *stog){
	if (stog->vrh >= MAXSTOG-1) return 0;
	stog->vrh++;
	stog->polje[stog->vrh] = element;
	return 1;
}
//funkcija za skidanje elementa sa stoga
//ako na stogu nema više elemenata funkcija vraća 0
//skinuti element se vraća preko adrese, a vrh stoga se umanjuje za 1
int skini(tip *element, Stog *stog){
	if (stog->vrh < 0) return 0;
	*element = stog->polje[stog->vrh];
	stog->vrh--;
```

```
	return 1;
}
//funkcija skida sve elemente sa stoga te na pomocni stog stavlja samo
//one elemente čiji proizvođač nije "Renault". Sa pomoćnog stoga skidamo
//sve elemente i stavljamo ih na početni stog
void renault(Stog *stog){
    Stog pomocni;
	zapis z;
	init_stog(&pomocni);
	//OPREZ funkciji skini predajemo stog, ne &stog
	//jer je u funkciju renault stog prenesen pomoću pokazivača
	while(skini(&z, stog))
		if(strcmp(z.proizvodjac, "Renault")!=0)
			dodaj(z, &pomocni);

	//OPREZ funkciji dodaj predajemo stog, ne &stog
	while(skini(&z, &pomocni))
		dodaj(z, stog);
}
int main(){
	FILE *binarna;
	//deklaracija stoga
	Stog stog;
	zapis z;
	//obavezno inicijalizirati stog prije rada s njime
	init_stog(&stog);
	binarna=fopen("automobili.dat", "rb");

	printf("\nUkrcavam u kamion...\n\n");
	while(fread(&z, sizeof(zapis), 1, binarna)){
		dodaj(z, &stog);
		printf("%02d %s %s %d\n", z.mbr, z.proizvodjac, z.vrsta, z.godina);
	}

	printf("\nIskrcavam sve i vracam natrag sve koji nisu Renault...\n");
	renault(&stog);

	printf("\nIskrcavam iz kamiona...\n\n");
	while(skini(&z, &stog))
		printf("%02d %s %s %d\n", z.mbr, z.proizvodjac, z.vrsta, z.godina);

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

StogPoljem – 4 Riješiti prethodni zadatak, ali bez korištenja pomoćnog stoga. U rješenju prethodnog programa treba promijeniti funkciju prototipa:

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

Napomena: Koristiti **rekurziju**.

```
void renault(Stog *stog){
	zapis z;

	if(skini(&z, stog)){
		renault(stog);
		if(strcmp(z.proizvodjac, "Renault")!=0)
			dodaj(z, stog);
	}
}
```

3.1. Stog realiziran listom

U prethodnom poglavlju stog smo realizirali statičkim poljem. Ta realizacija je imala 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. Podsjetimo se koje operacije moramo ostvariti da bi stog ispravno radio.

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

Struktura podataka koja opisuje stog realiziran listom:

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

Implementacije potrebnih funkcija su:

1) Dodavanje elementa 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;
}
```

2)Skidanje elementa sa stoga

```
int skini(zapis *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;
}
```

3)Inicijalizacija praznog stoga

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

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>
//struktura podataka za pojedinu vrstu alata
typedef struct{
	char alat[20+1];
	int kolicina;
}zapis;
//ako u red želimo pohraniti elemente drukčijeg tipa dovoljno je
//zapis zamijeniti s drugim tipom podataka, npr.
//typedef int tip; --> za red cjelobrojnih elemenata
typedef zapis tip;
//opis strukture podataka reda
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;
}
//funkcija za dodavanje elementa u red
//ako se ne može zauzeti dovoljno memorije za novi element vratimo 0
//inače dodajemo novi element u red i vraćamo 1
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.

```
#include <stdio.h>
#include <malloc.h>
//struktura podataka za pojedinu vrstu alata
typedef struct{
	char alat[20+1];
	int kolicina;
}zapis;
//ako u red želimo pohraniti elemente drukčijeg tipa dovoljno je
//zapis zamijeniti s drugim tipom podataka, npr.
//typedef int tip; --> za red cjelobrojnih elemenata
typedef zapis tip;
//opis strukture podataka reda
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;
}
//funkcija za dodavanje elementa u red
//ako se ne može zauzeti dovoljno memorije za novi element vratimo 0
//inače dodajemo novi element u red i vraćamo 1
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;
}
//funkcija za skidanje elementa iz reda
//ako u redu više nema elemenata funkcija vraća 0
//skinuti element se vraća preko adrese
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>
//struktura podataka za pojedinu vrstu alata
typedef struct{
	char alat[20+1];
	int kolicina;
}zapis;
//ako u red želimo pohraniti elemente drukčijeg tipa dovoljno je
//zapis zamijeniti s drugim tipom podataka, npr.
//typedef int tip; --> za red cjelobrojnih elemenata
typedef zapis tip;
//opis strukture podataka reda
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;
}
//funkcija za dodavanje elementa u red
```

```c
//ako se ne može zauzeti dovoljno memorije za novi element vratimo 0
//inače dodajemo novi element u red i vraćamo 1
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;
}
//funkcija za skidanje elementa iz reda
//ako u redu više nema elemenata funkcija vraća 0
//skinuti element se vraća preko adrese
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);

	//OPREZ funkciji dodaj predajemo stog, ne &stog
	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);
	}
}
```

StogListom– 5 Krajnje je vrijeme da se alatnica dovede u red. Napravilo se novo skladište u koje će se prebaciti sav alat čija je količina veća ili jednaka 20. Napišite sve potrebne funkcije za rad sa stogom realiziranog jednostruko povezanom listom. Napišite funkciju prototipa:

```
void razvrstaj(Stog *stog, Stog *novi);
```

koja će sa početnog stoga skinuti sve elemente čija je količina veća ili jednaka 20 te ih dodati na novi stog. Redoslijed elemenata na početnom stogu mora ostati očuvan. Napišite i glavni program u kojem ćete iz zadane datoteke učitati sve zapise o alatima, staviti ih na stog i pozvati funkciju *razvrstaj*. Potom skinuti sve elemente sa starog i sa novog stoga te ih ispisati.

```
#include <stdio.h>
#include <malloc.h>
//struktura podataka za pojedinu vrstu alata
typedef struct{
	char alat[20+1];
	int kolicina;
}zapis;
//ako u red želimo pohraniti elemente drukčijeg tipa dovoljno je
//zapis zamijeniti s drugim tipom podataka, npr.
//typedef int tip; --> za red cjelobrojnih elemenata
typedef zapis tip;
//opis strukture podataka reda
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;
}
//funkcija za dodavanje elementa u red
//ako se ne može zauzeti dovoljno memorije za novi element vratimo 0
//inače dodajemo novi element u red i vraćamo 1
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;
}
```

```
//funkcija za skidanje elementa iz reda
//ako u redu više nema elemenata funkcija vraća 0
//skinuti element se vraća preko adrese
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 razvrstaj(Stog *stog, Stog *novi){
    Stog pomocni1, pomocni2;
	zapis z;

	init_stog(&pomocni1);
	init_stog(&pomocni2);

	//OPREZ funkciji skini predajemo stog, ne &stog
	//jer je u funkciju renault stog prenesen pomoću pokazivača
	while(skini(&z, stog)){
		if(z.kolicina<20)
			dodaj(z, &pomocni1);
		else
			dodaj(z, &pomocni2);
	}
	//OPREZ funkciji dodaj predajemo stog, ne &stog
	while(skini(&z, &pomocni1))
		dodaj(z, stog);
	while(skini(&z, &pomocni2))
		dodaj(z, novi);
}

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

	init_stog(&stog); init_stog(&novi);
	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);
	}

	razvrstaj(&stog, &novi);

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

	printf("\n\t*** Novi stog (vece od 20) ***\n\n");
	while(skini(&z, &novi))
		printf("Skinut: %20s - %4d komada\n", z.alat, z.kolicina);

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

3.2. Red realiziran cirkularnim poljem

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.

Struktura podataka koja opisuje red realiziran cirkularnim poljem:

```
#define MAXSTOG 100
typedef struct{
	int vrh;
	tip polje[MAXSTOG];
}Stog;
```

Implementacije potrebnih funkcija su:

1) Dodavanje elementa u red

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

2)Skidanje elementa iz reda

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

3)Inicijalizacija praznog reda

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

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>
//Najveća veličina polja
#define MAXRED 128

typedef int tip;
//opis strukture podataka reda
typedef struct{
	tip polje[MAXRED];
	int ulaz, izlaz;
}Red;
//funkcija za inicijalizaciju
void init_red(Red *red){
	red->ulaz = 0; red->izlaz = 0;
}
//funkcija za dodavanje elementa u red
//ako je red prepunjen vratiti 0 dodati element u red i vratiti 1
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;
}
//funkcija za skidanje elementa iz reda
//ako u redu više nema elemenata funkcija vraća 0
//skinuti element se vraća preko adrese
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>
//Najveća veličina polja
#define MAXRED 128

typedef int tip;
//opis strukture podataka reda
typedef struct{
	tip polje[MAXRED];
	int ulaz, izlaz;
}Red;
//funkcija za inicijalizaciju
void init_red(Red *red){
	red->ulaz = 0; red->izlaz = 0;
}
//funkcija za dodavanje elementa u red
//ako je red prepunjen vratiti 0 dodati element u red i vratiti 1
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;
}
//funkcija za skidanje elementa iz reda
//ako u redu više nema elemenata funkcija vraća 0
//skinuti element se vraća preko adrese
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;
//opis strukture podataka reda
typedef struct{
	tip polje[MAXRED];
	int ulaz, izlaz;
}Red;
//funkcija za inicijalizaciju
void init_red(Red *red){
	red->ulaz = 0; red->izlaz = 0;
}
//funkcija za dodavanje elementa u red
//ako je red prepunjen vratiti 0 dodati element u red i vratiti 1
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;
}
//funkcija za skidanje elementa iz reda
//ako u redu više nema elemenata funkcija vraća 0
//skinuti element se vraća preko adrese
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>
//Najveća veličina polja
#define MAXRED 128

typedef int tip;
//opis strukture podataka reda
typedef struct{
	tip polje[MAXRED];
	int ulaz, izlaz;
}Red;
//funkcija za inicijalizaciju
void init_red(Red *red){
	red->ulaz = 0; red->izlaz = 0;
}
//funkcija za dodavanje elementa u red
//ako je red prepunjen vratiti 0 dodati element u red i vratiti 1
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;
}
//funkcija za skidanje elementa iz reda
//ako u redu više nema elemenata funkcija vraća 0
//skinuti element se vraća preko adrese
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>
//Najveća veličina polja
#define MAXRED 128

typedef int tip;
//opis strukture podataka reda
typedef struct{
	tip polje[MAXRED];
	int ulaz, izlaz;
}Red;
//funkcija za inicijalizaciju
void init_red(Red *red){
	red->ulaz = 0; red->izlaz = 0;
}
//funkcija za dodavanje elementa u red
//ako je red prepunjen vratiti 0 dodati element u red i vratiti 1
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;
}
//funkcija za skidanje elementa iz reda
//ako u redu više nema elemenata funkcija vraća 0
//skinuti element se vraća preko adrese
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;
}
```

3.3. Red realiziran listom

U prethodnom poglavlju red smo realizirali statičkim poljem. Ta realizacija je imala veliki nedostatak jer je postojala mogućnost prepunjenja polja. Sada ćemo red implementirati listom što će nam omogućiti da stavimo u red gotovo neograničen broj elemenata. Jedino ograničenje predstavljat će veličina radnog spremnika u računalu. Podsjetimo se koje operacije moramo ostvariti da bi red ispravno radio.

1) Dodavanje elemenata u red (*eng. Push*)
2) Skidanje elemenata iz reda (*eng. Pop*)
3) Inicijalizacija praznog reda

Struktura podataka koja opisuje red realiziran listom:

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

Implementacije potrebnih funkcija su:

1) Dodavanje elementa u red

```
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;//ako je bio prazan
		else red->ulaz->sljed=novi;	//inace, stavi na kraj
		red->ulaz=novi;			//zapamti zadnjeg
		return 1;
	}
	return 0;
}
```

2)Skidanje elementa iz reda

```
int SkiniIzReda(tip *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;
}
```

3)Inicijalizacija praznog reda

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

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>

//potrebne strukture podataka
typedef struct{
	char ime[20+1];
	char prezime[20+1];
	char fakultet[20+1];
	int godina_rodjenja;
	int stanje_na_xici;
}student;
//ako želimo u red staviti cijele brojeve promijenimo
//typedef int tip
typedef student tip;
typedef struct at{
	tip element;
	struct at *sljed;
}atom;
typedef struct {
	atom *ulaz, *izlaz;
}Red;
//kraj potrebnih struktura podataka

//implementacija reda listom
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; //ako je red bio prazan
		else red->ulaz->sljed=novi;          //inace, stavi na kraj
		red->ulaz=novi;                      //zapamti zadnjeg
		return 1;
		}
	return 0;
}
int SkiniIzReda(tip *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;
}
//kraj implementacije reda listom
void prednost_starijima(Red *red){
	Red pomocni1, pomocni2;
	student s;
	//prije rada red je potrebno inicijalizirati
	init_red(&pomocni1); init_red(&pomocni2);
	//skidamo sve studente iz početnog reda, one koji zadovoljavaju uvjet
	//stavljamo u red pomocni1, a ostale u red pomocni2
	while(SkiniIzReda(&s, red)){
		if(s.godina_rodjenja<1987)
			DodajURed(s, &pomocni1);
		else
			DodajURed(s, &pomocni2);
	}
	//vraćamo sve studente iz reda pomocni1, zatim iz 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;
	//prije rada red je potrebno inicijalizirati
	init_red(&pomocni);
	//skidamo sve studente iz reda i u pomcni red dodajemo samo
	//one koji zadovoljavaju zadani uvjet
	while(SkiniIzReda(&s, red))
		if(s.stanje_na_xici>=200)
			DodajURed(s, &pomocni);
	//skidamo sve studente iz pomoćnog reda i vraćamo ih u početni red
	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;
	//prije rada red je potrebno inicijalizirati
```

```
	init_red(&pomocni);
	//skidamo sve studente iz početnog reda. One koji nisu s FER-a dodajemo
	//u pomoćni red, a ferovce u ekskluzivni red
	while(SkiniIzReda(&s, red)){
		if(strcmp(s.fakultet, "FER")!=0)
			DodajURed(s, &pomocni);
		else
			DodajURed(s, ekskluzivni);
	}
	//vraćamo sve studente iz pomoćnog reda u početni
	while(SkiniIzReda(&s, &pomocni))
		DodajURed(s, red);
}

int main(){
	FILE *binarna;
	student s;
	Red red, ekskluzivni;
	//prije rada red je potrebno inicijalizirati
	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.

```
//rekuzivna funkcija
void okreni(Red *red1, Red *red2){
	student s;
	//ako u redu još uvijek ima studenata, skinuti prvog iz reda
	//te ga dodati u red2 tek nakon što rekurzija dođe do kraja.
	//Time osiguravamo obrnut redoslijed studenata
	if(SkiniIzReda(&s, red1)){
		okreni(red1, red2);
		DodajURed(s, red2);
	}
}

int main(){
	FILE *binarna;
	student s;
	Red red1, red2;
```

```
	//prije rada red je potrebno inicijalizirati
	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;
	//prije rada red je potrebno inicijalizirati
	init_red(&pomocni);
	//skidati jednog po jednog studenta iz reda i dodavati ih u pomoćni red.
	//Ako je taj student traženi Fero Ferovcic dodati skupinu pridošlica u
	//pomoćni red
	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);
		}
	}
	//vratiti sve studente (zajedno s pridošlicama) iz pomoćnog u početni red
	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");
```

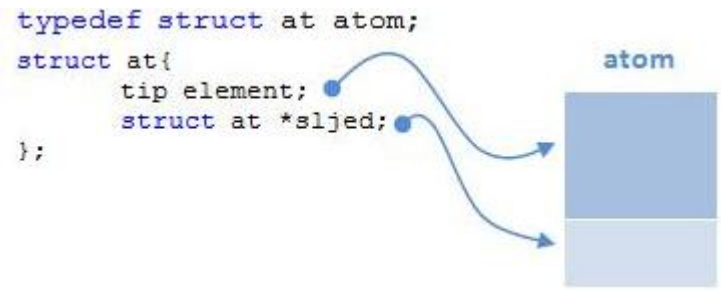
```
	printf("\n\t*** Pocetni red ***\n\n");
	while(fread(&s, sizeof(student), 1, binarna)){
		DodajURed(s, &red);
		printf("%s %s, %s, %d, %d\n", s.ime, s.prezime, s.fakultet,
			s.godina_rodjenja, s.stanje_na_xici);
	}

	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;
}
```

RedListom_6 - Za one koji žele istražiti prednosti koje nudi C++ i pripadna STL biblioteka dajemo sljedeći ogledni primjer rada sa redom.

```
#include <iostream>
#include <queue>

using namespace std;

int main(){
	queue<int> red;
	for(int i=1; i<=10; i++){
		red.push(i);
		cout << "Dodan: " << i << endl;
	}
	cout << endl;
	cout << "Velicina reda: " << red.size() << endl;
	cout << "Prvi u redu: " << red.front() << endl;
	cout << "Posljednji u redu: " << red.back() << endl << endl;
	while(!red.empty()){
		cout << "Skinut: " << red.front() << endl;
		red.pop();
	}
	return 0;
}
```

3.4. Linearna jednostruko povezana lista

Linearna lista je struktura podataka koja se sastoji od uređenog niza elemenata odabranih iz nekog skupa podataka. Pojedini elemenat liste zvat ćemo atom, a smatrat ćemo da je lista prazna ukoliko je njen broj elemenata 0. Osim statičkim poljem listu možemo realizirati i dinamičkom podatkovnom strukturom koja se sastoji od pokazivača na prvi element liste i od proizvoljnog broja atoma.
Atom jednostruko povezane liste sastoji se od podatkovnog dijela i od pokazivača na sljedeći element liste:



Za rad s jednostrukom listom potrebne su nam funkcije za:

1) Dodavanje elementa u listu

```
int dodaj(atom **glavap, tip element){
	atom *novi, *p;
	if((novi=(atom *)malloc(sizeof(atom)))==NULL)
		return 0;
	novi->element=element;
	if(*glavap==NULL || (*glavap)->element >= element){
		novi->sljed=*glavap;
		*glavap=novi;
	}else{
		for(p=*glavap; p->sljed && (p->sljed)->element < element;
		p=p->sljed);
		novi->sljed=p->sljed;
		p->sljed=novi;
	}
	return 1;
}
```

2) Brisanje elementa iz liste

```
int brisi(atom **glavap, tip element){
	atom *p;
	for(; *glavap && (*glavap)->element!=element;
	glavap=&((*glavap)->sljed));
	if(*glavap){
		p=*glavap;
		*glavap=(*glavap)->sljed;
		free(p);
		return 1;
	}else{
		return 0;
	}
```

```
}
```

3) Traženje elementa u listi

```
atom *trazi(atom *glava, tip element){
	atom *p;
	for(p=glava; p!=NULL; p=p->sljed){
		if(p->element==element) return p;
	}
	return NULL;
}
```

Zajednički dio za sve zadatke:

Na disku postoji formatirana tekstualna datoteka "psi.txt" u kojoj su sadržani zapisi o psima. Svaki zapis sadrži matični broj psa (int), ime psa (*niz znakova duljine 20+1*), pasminu (*niz znakova duljine 20+1*), godinu rođenja (*int*) te matični broj vlasnika psa (*int*). Strukturu podataka za pohranu zapisa možete definirati na sljedeći način:

```
typedef struct{
	int mbr;
	char ime[20+1];
	char pasmina[20+1];
	int godina_rodenja;
	int mbr_vlasnika;
}zapis;
```

Veterinarki Lidiji pala je na pamet odlična ideja - odlučila je organizirati zapise o psima u jednostruko povezanu listu kako bi si olakšala posao. Međutim, Lidija se ne razumije u liste i zato joj je potrebna vaša pomoć.

Koristeći se datotekom "psi.txt" koja sadrži popis pasa registriranih u Lidijinoj veterinarskoj stanici, organizirajte zapise u jednostruko povezanu listu prema rastućem matičnom broju psa. Napišite funkciju koja će dodavati zapise u jednostruko povezanu listu i funkciju koja će ispisati sadržaj liste počevši od najmanjeg matičnog broja. Funkcija za dodavanje zapisa u listu vraća 1 ako je zapis uspješno dodan, a 0 u slučaju neuspjeha. Također, napišite i glavni program koji će čitati zapise iz datoteke i dodavati ih u listu te na kraju ispisati sadržaj tako dobivene liste.

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

//definiramo strukturu koja će nam služiti za dodavanje
//zapisa u listu
typedef struct{
	int mbr;
	char ime[20+1];
	char pasmina[20+1];
	int godina_rodenja;
	int mbr_vlasnika;
}zapis;
typedef struct at atom;
//atom liste sastoji se od zapisa i pokazivača na sljedeći
//element liste
struct at{
	zapis element;
	struct at *sljed;
};

//funkcija koja dodaje atom u listu organiziranu prema rastućem matičnom broju
int dodaj(atom **glavap, zapis element){
```

```
		atom *novi, *p;
		//stvaramo novi atom liste
		if((novi=(atom *)malloc(sizeof(atom)))==NULL)
			return 0;
		novi->element=element;
		//u slučaju da je lista prazna, ili je matični broj psa kojeg želimo
		//dodati manji od svih ostalih u listi, zapis dodajemo na početak
		if(*glavap==NULL || (*glavap)->element.mbr >= element.mbr){
			novi->sljed=*glavap;
			*glavap=novi;
		//inače, zapis dodajemo negdje u sredinu
		}else{
			for(p=*glavap; p->sljed && (p->sljed)->element.mbr <
			element.mbr; p=p->sljed);
			novi->sljed=p->sljed;
			p->sljed=novi;
		}
		return 1;
}

//funkcija koja ispisuje sadržaj liste
void ispisi(atom *glava){
	atom *p;
	for(p=glava; p!=NULL; p=p->sljed){
		printf("%03d %s %s\n", p->element.mbr, p->element.ime,
		p->element.pasmina);
	}
}

int main(){
	zapis pas;
	atom *glava=NULL;
	FILE *ulaz;
	ulaz=fopen("psi.txt","r");
	if(!ulaz){
		printf("Ne mogu otvoriti datoteku!\n");
	}
	//citamo podatke iz formatirane datoteke i upisujemo ih u listu
	while(fscanf(ulaz, "%d %s %s %d %d", &pas.mbr, pas.ime, pas.pasmina,
		&pas.godina_rodenja, &pas.mbr_vlasnika)==5){
		printf("Dodajem psa: %s, mbr %03d\n", pas.ime, pas.mbr);
		dodaj(&glava, pas);
	}
	//obavezno zatvaramo datoteku
	fclose(ulaz);
	//ispisujemo sadržaj liste
	printf("\nIspisujem listu organiziranu prema rastucem maticnom broju
	psa:\n");
	ispisi(glava);
	return 0;
}
```

Napišite funkciju prototipa:

```
atom *trazi (atom *glava, char *ime);
```

koja će pomoći Lidiji da u jednostruko povezanoj listi pronađe traženi zapis. Funkcija vraća adresu tog zapisa, ili NULL pokazivač u slučaju da takav zapis ne postoji u listi.

```
//funkcija koja traži zadani zapis
//pretražujemo listu, sve dok postoji zapis i dok on nije traženi zapis
atom *trazi(atom *glava, char *ime){
	atom* p;
	for (p=glava; p && strcmp(p->element.ime,ime); p=p->sljed);
	//vraćamo adresu traženog zapisa, a ukoliko takav zapis ne postoji, p je NULL
	return p;
```

```
}
```

Lidijina veterinarska stanica je relativno nova, pa tako niti jedan štenac registriran u proteklih godinu dana još nije bio cijepljen. Napišite funkciju koja će pretražiti listu i ispisati sve pse mlađe od godinu dana i njihov ukupan broj kako bi Lidija znala koliko cjepiva treba naručiti.
Funkcija koja mora imati prototip:
`int ispisi_stence(atom *glava);`
preko imena vraća broj štenaca.

```
//funkcija koja ispisuje sve pse mlađe od godinu dana
int ispisi_stence(atom *glava){
	int br=0;
	atom *p;
	//prolazimo kroz listu, kada naiđemo na zapis koji zadovoljava uvjete,
	//ispisujemo ga
	for(p=glava; p != NULL; p=p->sljed){
		if((2008-p->element.godina_rodenja)<=1){
			br++;
			printf("%03d %s %s, roden %d. godine\n", p->element.mbr,
			p->element.ime, p->element.pasmina, p->element.godina_rodenja);
		}
	}
	return br;
}
```

Jedan vlasnik je odselio i Lidija više ne brine o njegovim psima. Napisati funkciju protipa:
`int brisi(atom **glavap, int mbr_vlasnika);`
koja će iz liste izbrisati zapis koji sadrži podatke psa dotičnog vlasnika. Funkcija vraća jedan ako je zapis uspješno izbrisan, a 0 u slučaju neuspjeha.

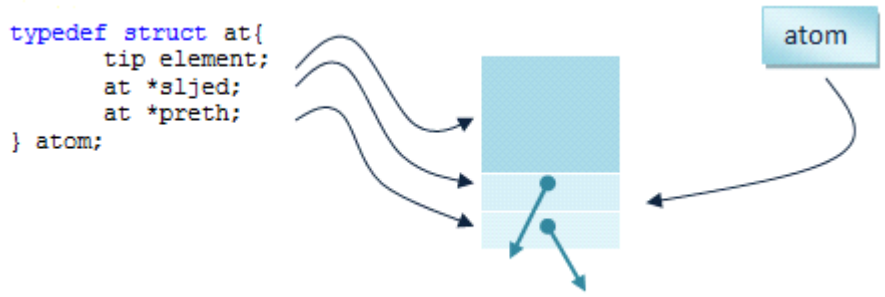
```
//funkcija koja briše zapis sa zadanim matičnim brojem vlasnika
int brisi(atom **glavap, int mbr_vlasnika){
	atom *p;
	//pretražujemo listu dok ne nađemo zadani matični broj ili dođemo do kraja
	for(; *glavap && (*glavap)->element.mbr_vlasnika!=mbr_vlasnika;
	glavap=&((*glavap)->sljed));
		//ako nismo došli do kraja liste, izbriši zapis i vrati 1
		if(*glavap){
			p=*glavap;
			printf("Brisem: %s, mbr %d, mbr_vlasnika %d\n", p->element.ime,
			p->element.mbr, p->element.mbr_vlasnika);
			*glavap=(*glavap)->sljed;
			free(p);
			return 1;
		//inače, vrati 0
		}else{
			printf("Vlasnik ne postoji!\n\n1");
			return 0;
		}
}
```

Lidija ima prigovor na prethodno rješenje – funkcija `brisi` briše samo jedan zapis. A što ako vlasnik ima više pasa? Napišite funkciju prototipa:
`void brisi(atom **glavap, int mbr_vlasnika);`
koja će skratiti Lidijine muke sa uzastopnim brisanjem. Naime, funkcija prima matični broj vlasnika koji je odselio, i briše iz liste **sve** zapise koji predstavljaju njegove pse.

```
//funkcija koja briše sve zapise sa zadanim matičnim brojem vlasnika
void brisi(atom **glavap, int mbr_vlasnika){
	//inicijaliziramo varijablu koja će nam poslužiti u slučaju da
	//vlasnik nije pronađen
	int nadeno=0;
	atom *stari;
	atom *pomocni=*glavap;
	//while petlja koja briše atome sa početka liste
	//dok postoji atom, i dok je taj atom baš onaj koji tražimo
	while((pomocni!=NULL) && (pomocni->element.mbr_vlasnika==mbr_vlasnika)){
		pomocni=(*glavap)->sljed;
		nadeno=1;
		printf("Brisem: %s, %s, maticni broj vlasnika: %d\n",
		(*glavap)->element.ime, (*glavap)->element.pasmina,
		(*glavap)->element.mbr_vlasnika);
		free(*glavap);
		*glavap=pomocni;
	}
	//nakon što smo provjerili da li se zatomi koje moramo obrisati nalaze
	//na početku liste, i iste obrisali, brišemo atome negdje u sredini liste
	//dok ne dođemo do kraja
	while(pomocni!=NULL){
		//preskačemo sve atome koji ne spadaju u kategoriju onih koje brišemo
		while(pomocni->sljed!=NULL &&
		(pomocni->sljed)->element.mbr_vlasnika!=mbr_vlasnika){
			pomocni=pomocni->sljed;
		}
		//ako se nalazimo iznad atoma koji treba obrisati - obrišemo ga
		if(pomocni->sljed!=NULL){
			stari=pomocni->sljed;
			pomocni->sljed=stari->sljed;
			nadeno=1;
			printf("Brisem: %s, %s, maticni broj vlasnika: %d\n",
			stari->element.ime, stari->element.pasmina,
			stari->element.mbr_vlasnika);
			free(stari);
		}
		//inače, došli smo do kraja liste, pomoćni postavljamo na NULL
		//i time izlazimo iz petlje
		else{
			pomocni=pomocni->sljed;
		}
	}
	//ako nismo našli vlasnika, ispisujemo poruku
	if(!nadeno){
		printf("Vlasnik sa zadanim maticnim brojem nije pronaden!\n");
	}
}
```

3.5. Linearna dvostruko povezana lista

Linearna lista je struktura podataka koja se sastoji od uređenog niza elemenata odabranih iz nekog skupa podataka. Pojedini elemenat liste zvat ćemo atom, a smatrat ćemo da je lista prazna ukoliko je njen broj elemenata 0. Osim statičkim poljem listu možemo realizirati i dinamičkom podatkovnom strukturom koja se sastoji od pokazivača na prvi element liste i od proizvoljnog broja atoma. Radi bržeg traženja elementa u listi, ona može biti i dvostruko povezana. Takva lista ima glavu i rep. Atom dvostruko povezane liste sastoji se od podatkovnog dijela te od pokazivača na sljedeći i na prethodni element liste.



Napišite funkciju prototipa:

```
int dodaj(atom **glavap, atom **repp, tip element);
```

koja će dodati element u dvostruko povezanu listu.

```
int dodaj(atom **glavap, atom **repp, tip element){
		atom *novi, *p;
		//zauzimamo memoriju za novi element, u slučaju neuspjeha vraćamo 0
		if((novi=(atom *)malloc(sizeof(atom)))==NULL)
			return 0;
		novi->element=element;
		novi->sljed=NULL;
		novi->preth=NULL;
		//dodajemo novi element na početak (lista je prazna)
		if(*glavap==NULL){
			printf("Lista je prazna - dodajem %d na pocetak!\n",element);
			*repp=novi;
			*glavap=novi;
		}
		//dodajemo novi element na početak
		else if((*glavap)->element >= element){
			printf("Dodajem %d na pocetak!\n",element);
			novi->sljed=*glavap;
			(*glavap)->preth=novi;
			*glavap=novi;
		}
		//dodajemo novi element na kraj
		else if((*repp)->element <= element){
			printf("Dodajem %d na kraj!\n",element);
			novi->preth=*repp;
			(*repp)->sljed=novi;
			*repp=novi;
		}
		//dodajemo novi element negdje u sredinu liste
		else{
			for(p=*glavap; p->sljed && (p->sljed)->element < element;
			p=p->sljed);
			printf("Dodajem %d negdje u sredinu!\n",element);
```

```
                novi->sljed=p->sljed;
                novi->preth=p;
                (p->sljed)->preth=novi;
                p->sljed=novi;
        }
        //ako je sve uspjelo, vraćamo 1
        return 1;
}
```

Napišite funkciju prototipa:

```
int brisi_od_glave(atom **glavap, atom **repp, tip element);
```

koja će brisati zadani element počevši s traženjem od glave. Također, napišite inačicu prethodne funkcije koja će brisati zadani element počevši od repa.

```
//od glave
int brisi_od_glave(atom **glavap, atom **repp, tip element){
        atom *p;
        //prolazimo kroz listu dok ne dođemo do zadanog elementa ili do kraja
        liste
        for(; *glavap && (*glavap)->element!=element;
        glavap=&((*glavap)->sljed));
        //ako smo ga našli
        if(*glavap){
                p=*glavap;
                //ako se traženi element nalazi na kraju liste
                if((*glavap)->sljed==NULL){
                        *repp=(*repp)->preth;
                        *glavap=(*glavap)->sljed;
                }
                //ako je negdje u sredini ili na početku
                else{
                        ((*glavap)->sljed)->preth=(*glavap)->preth;
                        *glavap=(*glavap)->sljed;
                }
                printf("Brisem element: %d, adresa: %p\n", p->element, p);
                free(p);
                return 1;
        }
        //u slučaju neuspjeha vraćamo 0
        else{
                return 0;
        }
}

//od repa
int brisi_od_repa(atom **glavap, atom **repp, tip element){
        atom *p;
        //prolazimo kroz listu dok ne dođemo do zadanog elementa ili do kraja
        liste
        for(; *repp && (*repp)->element!=element; repp=&((*repp)->preth));
        //ako smo ga našli
        if(*repp){
                p=*repp;
                //ako se traženi element nalazi na početku liste
                if((*repp)->preth==NULL){
                        *glavap=(*glavap)->sljed;
                        *repp=(*repp)->preth;
                }
                //ako je negdje u sredini ili na kraju
                else{
                        ((*repp)->preth)->sljed=(*repp)->sljed;
                        *repp=(*repp)->preth;
                }
                printf("Brisem element: %d, adresa: %p\n", p->element, p);
```

```
                free(p);
                return 1;
        }
        //u slučaju neuspjeha vraćamo 0
        else{
                return 0;
        }
}
```

Spojite prethodne dvije funkcije za brisanje elementa iz liste u jednu – napišite optimiziranu funkciju koja će istovremeno tražiti i od glave i od repa.

```
int brisi(atom **glavap, atom **repp, tip element){
        atom *p;
        //dok se pokazivači ne susretnu
        while((*glavap)!=(*repp)){
                //ako glava pokazuje na traženi element, brišemo element
                if((*glavap)->element==element){
                        printf("Brisem element: %d, adresa: %p\n",
                        (*glavap)->element, *glavap);
                        p=*glavap;
                        if((*glavap)->sljed==NULL){
                                *repp=(*repp)->preth;
                                *glavap=(*glavap)->sljed;
                        }
                        else{
                                ((*glavap)->sljed)->preth=(*glavap)->preth;
                                *glavap=(*glavap)->sljed;
                        }
                        free(p);
                        return 1;
                }
                //pomaknemo pokazivač za jedno mjesto naprijed
                glavap=&((*glavap)->sljed);
                //ako rep pokazuje na traženi element, brišemo element
                if((*repp)->element==element){
                        printf("Brisem element: %d, adresa: %p\n",
                        (*repp)->element, *repp);
                        p=*repp;
                        if((*repp)->preth==NULL){
                                *glavap=(*glavap)->sljed;
                                *repp=(*repp)->preth;
                        }
                        else{
                                ((*repp)->preth)->sljed=(*repp)->sljed;
                                *repp=(*repp)->preth;
                        }
                        free(p);
                        return 1;
                }
                //ako su pokazivači različiti (da se ne bi mimoišli i nesmetano
                //nastavili) pomaknemo pokazivač za jedno mjesto unazad
                if((*glavap)!=(*repp)){
                        repp=&((*repp)->preth);
                }
        }
        //ako ga nismo našli, a lista nije prazna, vraćamo 0
        printf("Element %d ne postoji u listi!\n", element);
        return 0;
}
```

Napišite funkciju prototipa:

```
atom *trazi_od_glave(atom *glava, tip element);
```

koja će tražiti zadani element počevši od glave. Također, napišite inačicu prethodne funkcije koja će tražiti zadani element počevši od repa.

```
//od glave
atom *trazi_od_glave(atom *glava, tip element){
		atom *p;
		//prolazimo kroz listu dok ne dođemo do kraja liste
		for(p=glava; p!=NULL; p=p->sljed){
			if(p->element==element){
				printf("Trazeni element pronaden: %d, adresa: %p\n",
				p->element, p);
				if(p->preth!=NULL) printf("\t- prethodni: %d, adresa:
				%p\n", (p->preth)->element, p->preth);
				if(p->sljed!=NULL) printf("\t- sljedeci:  %d, adresa:
				%p\n", (p->sljed)->element, p->sljed);
				return p;
			}
		}
		printf("Trazeni element nije pronaden!\n");
		return NULL;
}

//od repa
atom *trazi_od_repa(atom *rep, tip element){
		atom *p;
		//prolazimo kroz listu dok ne dođemo do kraja liste
		for(p=rep; p!=NULL; p=p->preth){
			if(p->element==element){
				printf("Trazeni element pronaden: %d, adresa: %p\n",
				p->element, p);
				if(p->preth!=NULL) printf("\t- prethodni: %d, adresa:
				%p\n", (p->preth)->element, p->preth);
				if(p->sljed!=NULL) printf("\t- sljedeci:  %d, adresa:
				%p\n", (p->sljed)->element, p->sljed);
				return p;
			}
		}
		printf("Trazeni element nije pronaden!\n");
		return NULL;
}
```

Dodatno, napišite modificiranu funkciju koja će tražiti istovremeno od i glave i od repa.

```
atom *trazi(atom *glava, atom *rep, tip element){
		atom *p_glava=glava;
		atom *p_rep=rep;
		//dok se pokazivači ne susretnu
		while(p_glava!=p_rep){
			//ako p_glava pokazuje na traženi element, vraćamo pokazivač -
			//našli smo ga
			if(p_glava->element==element){
				printf("Trazeni element pronaden: %d, adresa: %p\n",
				p_glava->element, p_glava);
				if(p_glava->preth!=NULL) printf("\t- prethodni: %d,
				adresa: %p\n", (p_glava->preth)->element,
				p_glava->preth);
				if(p_glava->sljed!=NULL) printf("\t- sljedeci:  %d,
				adresa: %p\n", (p_glava->sljed)->element,
				p_glava->sljed);
				return p_glava;
			}
			//pomaknemo p_glava za jedno mjesto naprijed
			p_glava=p_glava->sljed;
```

```
                        //ako p_rep pokazuje na traženi element, vraćamo pokazivač -
                        //našli smo ga
                        if(p_rep->element==element){
                                printf("Trazeni element pronaden: %d, adresa: %p\n",
                                p_rep->element, p_rep);
                                if(p_rep->preth!=NULL) printf("\t- prethodni: %d, adresa:
                                %p\n", (p_rep->preth)->element, p_rep->preth);
                                if(p_rep->sljed!=NULL) printf("\t- sljedeci:  %d, adresa:
                                %p\n", (p_rep->sljed)->element, p_rep->sljed);
                                return p_rep;
                        }
                        //ako su pokazivači različiti (da se ne bi mimoišli i nesmetano
                        //nastavili) pomaknemo p_rep za jedno mjesto unazad
                        if(p_glava!=p_rep){
                                p_rep=p_rep->preth;
                        }
                }
                //ako ga nismo našli, a lista nije prazna, vraćamo 0
                printf("Trazeni element nije pronaden!\n");
                return NULL;
}
```

Napišite funkcije koje će ispisati dvostruko povezanu listu poševši od glave te počevši od repa.

```
//od glave
void ispisi_od_glave(atom *glava){
        atom *p;
        for(p=glava; p != NULL; p=p->sljed){
                printf("%d ", p->element);
        }
        printf("\n\n");
}
//od repa
void ispisi_od_repa(atom *rep){
        atom *p;
        for(p=rep; p != NULL; p=p->preth){
                        printf("%d ", p->element);
        }
        printf("\n\n");
}
```

3.6. Red realiziran dvostruko povezanom listom

Napišite funkciju prototipa:

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

koja će dodati novi element u red realiziran dvostruko povezanom listom.

```
int dodaj(atom **ulaz, atom **izlaz, tip element){
	//zauzimamo memoriju za novi element i u slučlaju neuspjeha vraćamo 0
	atom *novi=(atom*)malloc(sizeof(atom));
	if(!novi) return 0;

	printf("Dodajem element %d!\n", element);
	novi->element=element;
	novi->sljed=NULL;

	//ubacujemo novi element u red
	//ako je red bio prazan, mijenjamo i ulaz, i izlaz
	if(*ulaz==NULL){
		*ulaz=*izlaz=novi;
		novi->preth=NULL;
	}
	//inače, mijenjamo samo ulaz
	else{
		(*ulaz)->sljed=novi;
		novi->preth=*ulaz;
		*ulaz=novi;
	}
	//ako je sve bilo uspješno, vraćamo 1
	return 1;
}
```

Napišite funkciju prototipa:

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

koja će izbrisati zadani element iz reda realiziranog dvostruko povezanom listom.

```
int brisi(atom **ulaz, atom **izlaz, int *element){
	atom *p;
	//ako je red prazan
	if(*ulaz==NULL){
		printf("Red je prazan!\n\n");
		return 0;
	}
	//zapamtimo element koji ćemo izbrisati i spajamo one koji ostaju
	p=*izlaz;
	*izlaz=(*izlaz)->sljed;
	//ao je to bio jedini element u redu, mijenjamo i ulaz
	if(*izlaz==NULL) *ulaz=NULL;
	else (*izlaz)->preth=NULL;

	//vraćamo izbrisani element preko adrese
	*element=p->element;
	printf("Brisem element: %d\n", p->element);
	//obavezno oslobađamo zauzetu memoriju
	free(p);
	return 1;
}
```

Napišite funkciju prototipa:

```
atom *trazi(atom *izlaz, tip element);
```

koja će tražiti zadani element u redu realiziranom dvostruko povezanom listom. Funkcija mora vratiti pokazivač na traženi element ili NULL u slučaju da takav element ne postoji.

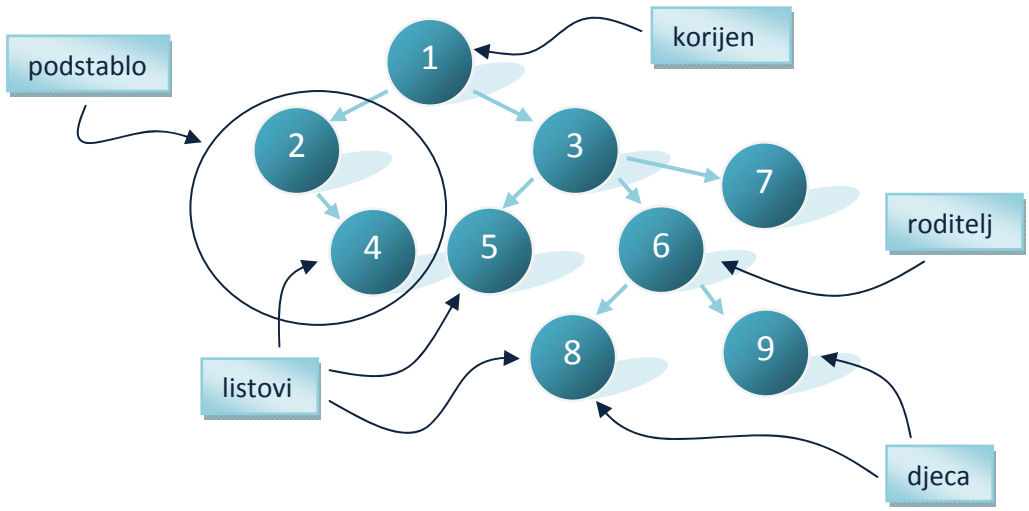
```
atom *trazi(atom *izlaz, tip element){
	int br=0;
	atom *p;
	//prolazimo kroz red dok ne dođemo do kraja
	for(p=izlaz; p!=NULL; p=p->sljed){
		if(p->element==element){
			printf("Pronaden zadani element: %d!\n", p->element);
			printf("Ispred njega nalazi se %d elemenata.\n\n", br);
			return p;
		}
		//brojimo element koji se nalaze ispred traženog elementa
		br++;
	}
	printf("Trazeni element ne postoji!\n\n");
	return NULL;
}
```

Napišite funkciju koja će ispisati red realiziran dvostruko povezanom listom.

```
void ispisi(atom *izlaz){
	atom *p;
	printf("Ispisujem red implementiran dvostrukom listom:\n");
	printf("    **izlaz se nalazi na lijevoj strani**\n");
	printf("    ");
	//prolazimo kroz red i ispisujemo element jedan za drugim
	for(p=izlaz; p!=NULL; p=p->sljed){
		printf("%d ", p->element);
	}
	printf("\n\n");
}
```

3.7. Binarno stablo

Stablo je konačan skup čvorova pri čemu je jedan čvor *korijen*, a ostali čvorovi podijeljeni su u disjunktne podskupove od kojih je svaki stablo - *podstabla*.

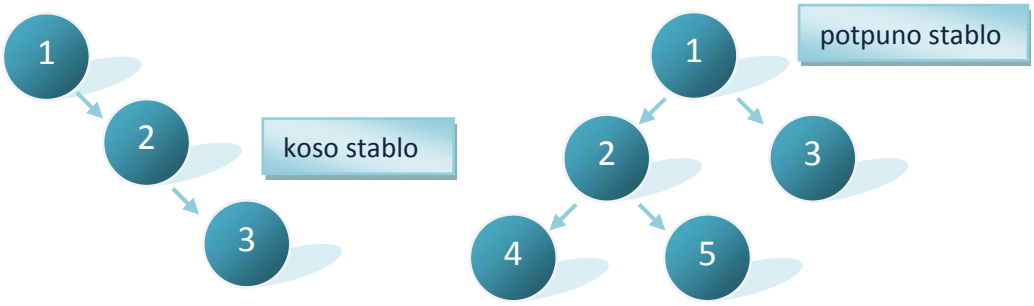


Neki pojmovi vezani uz stabla:

- *stupanj čvora:* broj podstabala nekog čvora (*stupanj čvora 1=2*)
- *stupanj stabla:* maksimalni stupanj svih čvorova tog stabla (*stupanj stabla=3*)
- *razina(level):* krijen je razine *1*, a razine djece nekog čvora razine *k* jednake su *k+1*
- *dubina(depth):* maksimalna razina nekog čvora u stablu (*dubina=4*)

Binarno stablo

Binarno stablo je stablo koje se sastoji od nijednog, jednog ili više čvorova drugog stupnja pa tako razlikujemo *lijevo* i *desno podstablo*.

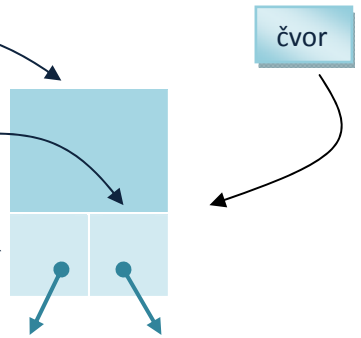


Maksimalni broj čvorova na k-toj razini binarnog stabla jednak je 2^{k-1}, doj je maksimalni broj čvorova binarnog stabla dubine k jednak $2^k - 1$ i takvo stablo nazivamo punim (*full*) binarnim stablom. Binarno stablo s *n* čvorova dubine *k* je potpuno (*complete*) binarno stablo ako i samo ako njegovi čvorovi odgovaraju čvorovima punog binarnog stabla dubine k koji su numerirani od 1 do n.
Binarno stablo moguće je prikazati statičkom strukturom polje kakav prikaz ćemo koristiti kod *heap sorta*. Kod potpunog binarnog stabla s *n* elemenata realiziranog poljem, za *i-ti čvor* vrijedi:

- *roditelj(i)=* ⌊i/2⌋ za i≠ 1; kada je i=1, čvor i je korijen, nema roditelja
- *lijevo_dijete(i)=* 2*i ako je 2*i≤ n; kad je 2*i>n čvor i nema lijevog djeteta
- *desno_dijete(i)=* 2*i+1 ako je 2*i+1≤n; kad je 2*i+1>n čvor i nema desnog djeteta

Binarno stablo možemo realizirati i i dinamičkom podatkovnom strukturom koja se sastoji od pokazivača na korijen stabla i od proizvoljnog broja čvorova. Čvor stabla sastoji se od podatkovnog dijela te od dva pokazivača – na lijevo i na desno dijete:

```
typedef struct cv{
        tip element;
        struct cv *lijevo_dijete;
        struct cv *desno_dijete;
}cvor;
```



Sortirano binarno stablo(stablo za traženje)

Novi element u sortirano binarno stablo dodajemo na sljedeći način (počevši od korijena):

- ako je element novog čvora manji od elementa upisanog čvora usporedbe, nastavljamo usporedbu u lijevom podstablu;
- inače, nastavljamo usporedbu u desnom podstablu;
- ako upisani čvor u traženom smjeru nema podstablo, novi čvor postaje dijete upisanog čvora;

Za rad sa sortiranim binarnim stablom potrebne su nam funkcije za:

1) Dodavanje čvora u stablo

```
//dodavanje novog čvora u stablo: lijevo manji, desno veci
cvor *upis(cvor *korijen, tip element){
        //ako čvor nema dijete u traženom smjeru, dodaj novi čvor
        //(to uključuje i slučaj kad je cijelo stablo prazno)
        if(korijen==NULL){
                korijen=(cvor*)malloc(sizeof(cvor));
                if(korijen){
                        korijen->element=element;
                        korijen->lijevo=korijen->desno=NULL;
                }else{
                        printf("Ne mogu zauzeti memoriju za novi element!\n",
                        element);
                }
        //inače ako je novi element manji od elementa čvora usporedbe,
        //idemo lijevo
        }else if(element < korijen->element){
                korijen->lijevo=upis(korijen->lijevo, element);
        //inače ako je novi element veći od elementa čvora usporedbe,
        //idemo desno
        }else if(element > korijen->element){
                korijen->desno=upis(korijen->desno, element);
        //inače takav element već postoji
        }else{
                printf("Podatak %d vec postoji!\n", element);
        }
        //vrati pokazivač na dodani element, ili NULL u slučaju neuspjeha
        return korijen;
}
```

2) Brisanje čvora iz stabla

```
//brisanje uparivanjem
void BrisiUparivanjem(cvor **radni){
```

```
	cvor *privremeni=*radni;
	if((*radni)!=NULL){
		//ako čvor kojeg brišemo nema desno dijete, lijevo dijete
		//(ako ga ima) stavljamo na njegovo mjesto
		if(!(*radni)->desno)
			(*radni)=(*radni)->lijevo;
		//inače ako nema lijevo dijete, desno dijete (ako ga ima)
		//stavljamo na njegovo mjesto
		else if(!(*radni)->lijevo)
			(*radni)=(*radni)->desno;
		else{
			//pomaknemo čvorove za 1 lijevo
			privremeni=(*radni)->lijevo;
			//a zatim do kraja desno
			while(privremeni->desno)
				privremeni=privremeni->desno;
			//povezujemo najdesniji čvor lijevog podstabla s desnim
			//podstablom
			privremeni->desno=(*radni)->desno;
			privremeni=*radni;
			*radni=(*radni)->lijevo;
		}
		free(privremeni);
	}
}
//trazenje i brisanje cvora u binarnom stablu
void NadjiBrisi(cvor **korijen, tip element){
	cvor *radni=*korijen;
	cvor *preth=NULL;
	//tražimo čvor dok ne pretražimo cijelo stablo
	while(radni!=NULL){
		//ako je čvor na koji pokazuje radni traženi čvor, izlazimo
		if(element==radni->element)
			break;
		//zapamtimo prethodni (služi nam kasnije za brisanje)
		preth=radni;
		//ako je element manji od elementa čvora usporedbe, idemo lijevo
		if(element<radni->element)
			radni=radni->lijevo;
		//inače, idemo desno
		else
			radni=radni->desno;
	}
	//ako smo našli traženi čvor
	if(radni!=NULL && radni->element==element){
		printf("Brisem element %d!\n\n", radni->element);
		//ako je zadani čvor korijen
		if (radni==*korijen)
			BrisiUparivanjem(korijen);
		//inače, ako je lijevo dijete
		else if(preth->lijevo==radni){
			BrisiUparivanjem(&(preth->lijevo));
		//inače, zadani čvor je desno dijete
		}else{
			BrisiUparivanjem(&(preth->desno));
		}
	}
	//inače ako je korijen != NULL, nismo našli zadani čvor
	else if(korijen!=NULL)
		printf("Element %d nije u stablu!\n\n", element);
	//inače (korijen = NULL), stablo je prazno
	else("Stablo je prazno!\n\n");
}
```

3) Traženje čvora u stablu

```
//trazenje cvora u binarnom stablu
cvor *trazi(cvor *korijen, tip element){
	//dok ne dođemo do kraja stabla (što uključuje i slučaj kada
```

```
        //je stablo prazno) i dok nismo našli zadani čvor
        while(korijen && korijen->element != element){
              //ako je element manji od elementa čvora usporedbe, idemo lijevo
              if(element < korijen->element) korijen=korijen->lijevo;
              //inače, idemo desno
              else korijen=korijen->desno;
        }
        //vraćamo pokazivač na traženi čvor, ili NULL
        return korijen;
  }
```

Zajednički dio za sve zadatke:
U binarnom stablu pohranjeni su cjelobrojni elementi i stablo je sortirano (lijevo manji, desno veći).

Napišite **rekurzivnu** funkciju koja će izračunati dubinu stabla.

```
int dubina_stabla(cvor *korijen){
int dubina_lijevo, dubina_desno;
if(korijen){
      //dubina stabla = 1 + max(dubina_lijevo, dubina_desno)
      dubina_lijevo=dubina_stabla(korijen->lijevo);
      dubina_desno=dubina_stabla(korijen->desno);
      return 1 + (dubina_lijevo > dubina_desno?dubina_lijevo:dubina_desno);
}
//osnovni slučaj
return 0;
}
```

Napišite **rekurzivnu** funkciju koja će prebrojati sve listove zadanog stabla.

```
int broji_listove(cvor *korijen){
if(korijen){
      //ako je trenutni čvor list, vraćamo 1
      if(!korijen->lijevo && !korijen->desno) return 1;
      //inače vraćamo broj listova u lijevom podstablu + broj listova u
      //desnom podstablu
      else return (broji_listove(korijen->lijevo)
      + broji_listove(korijen->desno));
}
//osnovni slučaj
return 0;
}
```

Napišite **rekurzivnu** funkciju koja će vratiti najveći element u stablu.

```
int max_element(cvor *korijen){
int max, m;
//nađemo najveći element u desnom podstablu (stablo je sortirano,
//znači svi veći elementi nalaze se desno), te ga usporedimo s trenutnim
//elementom i vratimo veći
if(korijen){
      m=max_element(korijen->desno);
      return max=korijen->element > m? korijen->element:m;
}
//osnovni slučaj
return 0;
}
```

Promijenite funkciju iz prethodnog zadatka tako da ona vrijedi i u slučaju kada stablo **nije sortirano**.

```
int max_element(cvor *korijen){
	if(korijen){
		int max, max_lijevo, max_desno;
		//nađemo max u lijevom podstablu
		max_lijevo=max_element(korijen->lijevo);
		//pa zatim u desnom
		max_desno=max_element(korijen->desno);
		//usporedimo trenutni element s max lijevog i max desnog stabla, te
		//vratimo najveći
		max=max_lijevo > max_desno?max_lijevo:max_desno;
		max=korijen->element > max?korijen->element:max;
		return max;
	}
	//osnovni slučaj
	return 0;
}
```

Napišite **rekurzivnu** funkciju koja će zbrojiti sve neparne elemente zadanog stabla.

```
	int zbroji_neparne(cvor *korijen){
	if(korijen)
		//vrati: element(ako je neparan), 0 inače + zbroj svih elemenata
		//lijevog podstabla + zbroj svih elemenata desnog podstabla
		return ((korijen->element%2?korijen->element:0) +
		zbroji_neparne(korijen->lijevo) + zbroji_neparne(korijen->desno));
	//osnovni slučaj
	return 0;
}
```

Napišite **rekurzivnu** funkciju koja će zrcaliti stablo (svakom čvoru međusobno zamijeniti lijevo i desno dijete).

```
cvor *zrcali_stablo(cvor *korijen){
	//zamijenimo lijevo i desno dijete trenutnog čvora
	if(korijen){
		cvor *temp;
		temp=korijen->lijevo;
		korijen->lijevo=zrcali_stablo(korijen->desno);
		korijen->desno=zrcali_stablo(temp);
	}
	//osnovni slučaj
	return korijen;
}
```

Napišite **rekurzivnu** funkciju koja će vratiti razinu u kojoj se nalazi najveći element zadanog stabla. Također, napisati **nerekurzivnu** inačicu prethodne funkcije.

```
//rekurzivno
int razina(cvor *korijen){
```

```
	//vraćamo 1 (za trenutnu razinu) + broj svih ostalih razina
	//do najvećeg elementa (krajnji desno)
	if(korijen) return (1+razina(korijen->desno));
	//osnovni slučaj
	return 0;
}

//nerekurzivno
int razina(cvor *korijen){
	int razina=0;
	//brojimo razine do najvećeg elementa (krajnji desno)
	while(korijen){
		korijen=korijen->desno;
		razina++;
	}
	return razina;
}
```

Napišite **rekurzivnu** funkciju koja će ispisati sve čvorove nekog stabla na zadanoj razini.

```
	void ispisi_na_razini(cvor *korijen, int razina){
	if(korijen){
		razina--;
		//ako se nalazimo na zadanoj razini, ispisujemo element
		if(razina==0) printf("%d ", korijen->element);
		//inace, spustamo se nize
		else if(razina>0){
			ispisi_na_razini(korijen->lijevo, razina);
			ispisi_na_razini(korijen->desno, razina);
		}
	}
}
```

Napišite **rekurzivnu** funkciju koja će vratiti broj elemenata stabla na zadanoj razini. Funkcija ne smije pretraživati cijelo stablo, već samo do zadane razine.

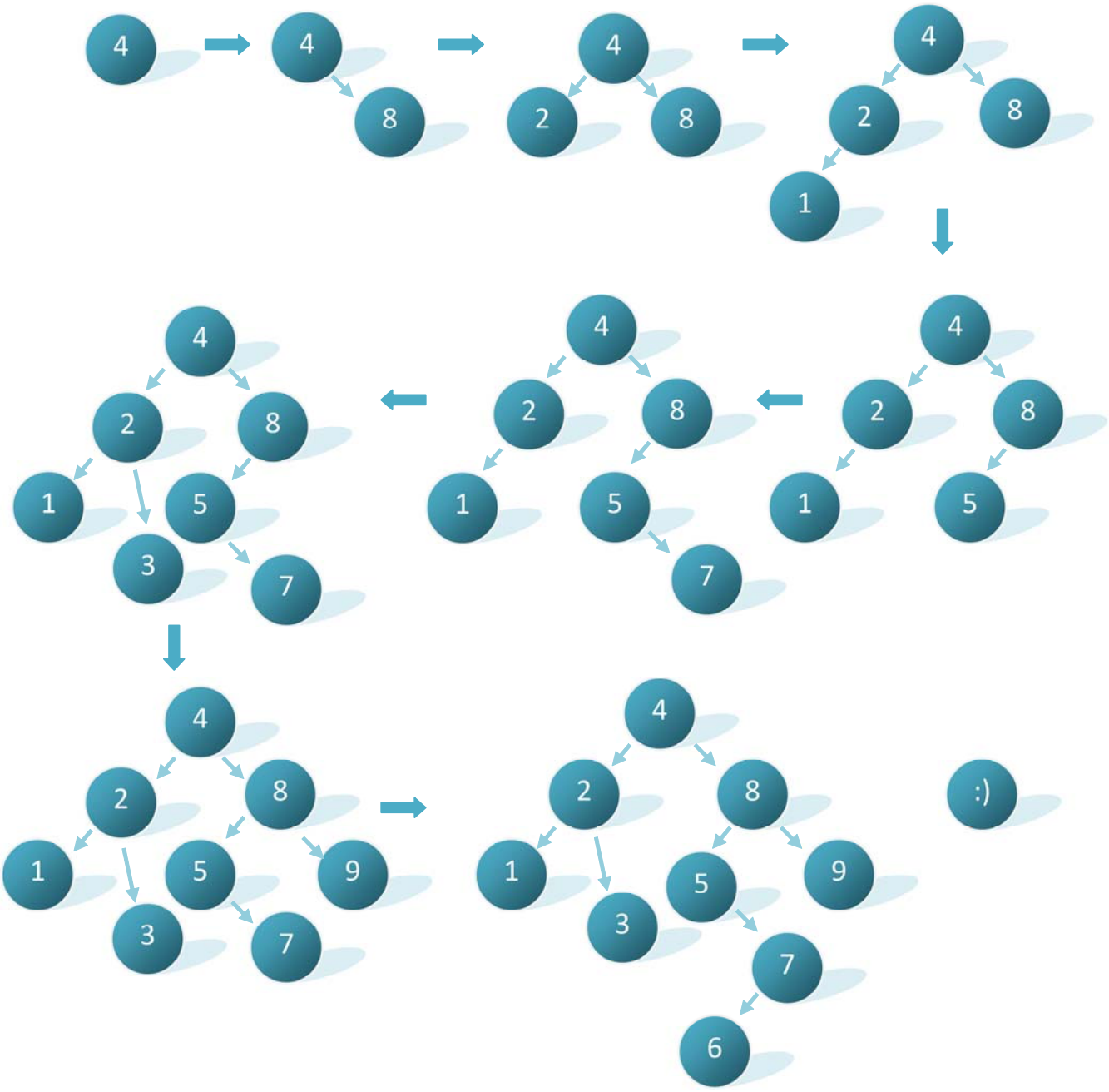
```
int prebroji_na_razini(cvor *korijen, int razina){
	//osnovni slučaj
	if(!korijen) return 0;
	//ako smo prešli zadanu razinu, isto vraćamo 0
	if(razina<1) return 0;
	//ako smo na zadanoj razini, vraćamo 1
	if(razina==1) return 1;
	//inače, prebrajamo dalje lijevo i desno
	return (prebroji_na_razini(korijen->lijevo, razina-1) +
	prebroji_na_razini(korijen->desno, razina-1));
}
```

Napišite **rekurzivnu** funkciju koja će obrisati zadano stablo (osloboditi memoriju za sve čvorove i korijen postaviti na NULL). Funkcija mora preko imena vratiti broj čvorova zadanog stabla.

```
int brisi_sve(cvor **korijen){
	int br;
	if(*korijen){
		//broj čvorova je 1 (za trenutni čvor) + broj čvorova u lijevom
```

```
            //i desnom podstablu
            br=1+brisi_sve(&((*korijen)->lijevo))+brisi_sve(&((*korijen)->desno));
            //oslobodi memoriju trenutnog čvora
            free(*korijen);
            //postavi pokazivač na njega u NULL
            *korijen=NULL;
            return br;
        }
        //osnovni slučaj
        return 0;
}
```

Ilustrirajte stvaranje sortiranog binarnog stabla za zadani niz brojeva: 4, 8, 2, 1, 5, 7, 3, 9, 6. Nacrtajte izgled stabla nakon svakog dodavanja novog broja.



3.8. Korištenje dinamičkih struktura podataka

Cilj ovog poglavlja je utvrditi princip rada dinamičkih struktura podataka: stoga, reda, jednostruko i dvostruko povezana liste te stabla. Bitno je svladati rad s pokazivačima i pozivanje funkcija ovih dinamičkih struktura. U tu svrhu pripremili smo vam već implementirane strukture u pripadnim zaglavljima. Slijedi njihov pregled:

Zaglavlje: stog.h → Stog realiziran jednostrukom listom
`void init_stog(Stog *stog);`
`int DodajNaStog(tip element, Stog *stog);`
`int SkiniSaStoga(tip *element, Stog *stog);`

Zaglavlje: red.h → Red realiziran jednostrukom listom
`void init_red(Red *red);`
`int DodajURed(tip element, Red *red);`
`int SkiniIzReda(tip *element, Red *red);`

Zaglavlje: lista.h → Jednostruko povezana lista
`void init_lista(Lista *lista);`
`int DodajUListu(atoml **glavap, tip element);`
`void IspisiListu(atoml *glava);`
`int BrisiIzListe(atoml **glavap, tip element);`
`atoml *TraziUListi(atoml *glava, tip element);`

Zaglavlje: dvostruka.h → Dvostruko povezana lista
`void init_dvostruka(Dvostruka *dvostruka);`
`int DodajUDvostruku(atomd **glavap, atomd **repp, tip element);`
`void ispisi_od_glave(atomd *glava);`
`void ispisi_od_repa(atomd *rep);`
`atomd *trazi_od_glave(atomd *glava, tip element);`
`atomd *trazi_od_repa(atomd *rep, tip element);`
`atomd *TraziDvostruku(atomd *glava, atomd *rep, tip element);`
`int brisi_od_glave(atomd **glavap, atomd **repp, tip element);`
`int brisi_od_repa(atomd **glavap, atomd **repp, tip element);`
`int BrisiIzDvostruke(atomd **glavap, atomd **repp, tip element);`

Zaglavlje: stablo.h → Binarno stablo
`void init_stablo(Stablo *stablo);`
`cvor *UpisiUStablo(cvor *korijen, tip element);`
`void IspisiInordeLD(cvor *korijen); //inorder lijevo-desno`
`void IspisiInordeDL(cvor *korijen); //inorder desno-lijevo`

```
void IspisiPreorder(cvor *korijen);
void IspisiPostorder(cvor *korijen);
void IspisiStablo(cvor *korijen, int nivo);
cvor *TraziUStablu(cvor *korijen, tip element);
void BrisiUparivanjem(cvor **radni);
void NadjiBrisi(cvor **korijen, tip element);
```

DinamickeStrukture_1 - Napisati funkciju prototipa:

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

koja će dodati na stog sve cijele brojeve iz intervala [1, 10]. Zatim skinuti sve elemente sa stoga i staviti ih u red. Napisati glavni program u kojem ćete inicijalizirati stog i red te pozvati funkciju.

```
#include "stog.h"
#include "red.h"

void funkcija(Stog *stog, Red *red){
	int element, i;
	for(i=1; i<=10; i++){
		//OPREZ predaje se stog, ne &stog
		DodajNaStog(i, stog);
		printf("Dodan na stog: %d\n", i);
	}
	printf("\n");

	while(SkiniSaStoga(&element, stog)){
		DodajURed(element, red);
		printf("Skidam sa stoga i dodajem u red: %d\n", element);
	}
}

int main(){
	Stog stog; Red red;
	//inicijalizacija stoga i reda
	init_stog(&stog); init_red(&red);

	funkcija(&stog, &red);

	return 0;
}
```

DinamickeStrukture_1 - Napisati funkciju prototipa:

```
void funkcija(Stog *stog, Lista *lista);
```

koja će dodati na stog sve cijele brojeve iz intervala [1, 10]. Zatim skinuti sve elemente sa stog i staviti ih u jednostruko povezanu listu. Iz liste izbrisati sve parne elemente te je ispisati.

```
#include "stog.h"
#include "lista.h"

void funkcija(Stog *stog, Lista *lista){
	int element, i;
	for(i=1; i<=10; i++){
```

```
			DodajNaStog(i, stog);
			printf("Dodan na stog: %d\n", i);
		}
		printf("\n");

		while(SkiniSaStoga(&element, stog)){
			DodajUListu(&(lista->glava), element);
			printf("Skidam sa stoga i dodajem u listu: %d\n", element);
		}
		printf("\n");

		printf("\nBrisem parne elemente iz liste!\n\n");
		for(i=1; i<=10; i++){
			if(i%2==0)
				BrisiIzListe(&(lista->glava), i);
		}

		printf("Ispis liste: ");
		IspisiListu(lista->glava);
		printf("\n\n");
}

int main(){
		Stog stog; Lista lista;
		//inicijalizacija stoga i liste
		init_stog(&stog); init_lista(&lista);

		funkcija(&stog, &lista);

		return 0;
}
```

Zadane su funkcije za dodavanje i brisanje elemenata iz binarnog sortiranog stabla, i dodavanje i brisanje elemenata iz jednostruko povezane liste. Koristeći potrebna zaglavlja i funkcije deklarirane u njima, napišite **rekurzivnu** funkciju prototipa:

```
Stablo *ListaUStablo(atom* lista);
```

koja će prepisati jednostruko povezanu listu u sortirano binarno stablo. Također, napišite i glavni program u koji treba stvoriti listu realiziranu pokazivačima čiji atom sadrži cijele brojeve (*int*) . Listu popunite sa 6 proizvoljnih elemenata, pozovite funkciju te ispišite stablo preorder postupkom.

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <time.h>
#include "lista.h"
#include "stablo.h"

Stablo *ListaUStablo(atom *glava){
		if(lista==NULL){
			Stablo *st//=(Stablo*)malloc(sizeof(Stablo));
			init_stablo(st);
			return st;
		}
		Stablo *st=ListaUStablo(lista->sljed);
		st=UpisiUStablo(st->korijen, lista->element);
		return st;
```

```
}

int main(){
	int i;
	Lista *lista;
	Stablo *st;
	init_lista(lista);
	init_stablo(stablo);
	srand((unsigned)time(NULL));

	for(i=0; i<6; i++)
		DodajUListu(&(lista->glava), rand()%10);

	IspisiListu(lista->glava);
	printf("\n\n");
	ListaUStablo(stablo->korijen, lista->glava);

	printf("Preorder ispis stabla:\n");
	IspisiPre(st->korijen);

	return 0;
}
```

Napišite **rekurzivnu** funkciju prototipa:

```
void StabloUListu(cvor *stablo, atom *glava);
```

koja prepisuje binarno sortirano stablo u jednostruku listu. Koristite potrebna zaglavlja.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "lista.h"
#include "stablo.h"



void StabloUListu(cvor *stablo, atoml *glava){
	if(stablo){
		StabloUListu(stablo->lijevo, glava);
		DodajUListu(&glava, stablo->element);
		StabloUListu(stablo->desno, glava);
	}
}

int main(){
	int i;
	Stablo *stablo;
	Lista *lista;
	init_stablo(stablo);
	init_lista(lista);
	srand((unsigned)time(NULL));

	for(i=0; i<10; i++)
		UpisiUStablo(stablo->korijen, i);

	StabloUListu(stablo->korijen, lista->glava);

	printf("Ispis liste:\n");
	IspisiListu(lista->glava);

	return 0;
}
```

3.9. Objektno orijentirano programiranje

C++ je objektno orijentiran programski jezik. Objektno orijentirano programiranje nam omogućuje ponovnu iskoristivost koda (*eng. reusability*). Svaka klasa se ponaša kao „crna kutija“. Programer ne mora nužno znati kako je klasa implementirana, samo mora znati javne funkcije klase ili metode (*enkapsulacija*).

Razlika između struktura i klasa ima više. Struktura je samo nakupina podataka, dok je klasa skup podataka i metoda koje vrše operacija nad njima. Varijable u strukturi su javne, a u klasi su privatne. Ako je varijabla privatna, to znači da njoj mogu pristupiti samo metode te klase. Metode obično imaju javni pristup (*public*) te na taj način tvore sučelje klase prema korisniku.

Kada napišemo razred ili klasu mi smo ju deklarirali, a kad je instanciramo stvorili smo objekt. Dakle, klasa je samo jedna, a objekata iste klase ima onoliko koliko ih mi stvorimo.

Konstruktor se automatski poziva pri stvaranju objekta. Svaki razred može imati najviše jedan podrazumijevani konstruktor, dok ostalih konstruktora može biti i više. Konstruktor ima isto ime kao i sam razred te ne vraća nikakav tip podatka. Destruktor razreda se poziva automatski pri uništenju instanciranog objekta. Razred može imati najviše jedan destruktor i on mora biti istog imena kao i sam razred, ali ispred imena mora imati predmetak `~`, npr. `~Razred`. Destruktor ne vraća nikakav tip podatka.

Način rada s klasama te prednosti koje on pruža najbolje ćete naučiti na primjerima. Zato slijedi nekoliko lakših primjera klasa s pripadnim glavnim programima u kojima su pokazane osnovne funkcionalnosti objektno orijentiranog programiranja.

OOP_1 - Implementirati razred Slucajni koji će nam koristiti za rad sa pseudoslučajnim brojevima. Razred mora sadržavati tri varijable. Prva varijabla predstavlja generirani pseudoslučajni broj, a druge dvije su granice zatvorenog intervala. Napisati podrazumijevani konstruktor , konstruktor koji će inicijalizirati generator pseudoslučajnih brojeva zadanim sjemenom (*seed*) i destruktor. Implementirati postavljanje zatvorenog intervala pomoću metode `void postaviInterval(int a, int b)`. Također implementirati metodu koja će vratitii generirani broj.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

class Slucajni{
private:
	//privatne varijable kojima se može pristupiti
	//samo pomoću metoda razreda Slucajni
	int _broj;
	int _donjaGranica;
```

```
	int _gornjaGranica;
public:
	//javne metode razreda, one predstavljaju sučelje razreda
	//pomoću njih korisnik radi s privatnim varijablama

	//podrazumijevani konstruktor
	Slucajni(){
		srand((unsigned)time(NULL));
		_donjaGranica=_gornjaGranica=0;
	}
	//konstruktor
	void postaviSjeme(int sjeme){
		srand((unsigned)sjeme);
		_donjaGranica=_gornjaGranica=0;
	}
	//metoda za postavljanje zatvorenog intervala
	void postaviInterval(int a, int b){
		_donjaGranica=a;
		_gornjaGranica=b;
	}
	//metoda koja vraća generirani broj iz zadanog intervala
	int dodijeliBroj(){
		return rand()%(_gornjaGranica-_donjaGranica+1)+_donjaGranica;
	}
	//destruktor razreda
	~Slucajni(){
	}
};

int main(){
	int a, b;
	Slucajni slucajni;

	printf("\nUnesite granice zatvorenog intervala: ");
	scanf("%d %d", &a, &b);
	slucajni.postaviInterval(a, b);

	printf("\nSlucajan broj iz zadanog intervala: %d\n\n",
slucajni.dodijeliBroj());
	return 0;
}
```

OOP_2 - Implementirati razred Trokut koji će sadržavati tri varijable jednostruke preciznosti koje predstavljaju pojedine stranice trokuta. Razred mora imati podrazumijevani konstruktor (*sve stranice trokuta postavlja na 0*) te konstruktor koji inicijalizira stranice trokuta zadanim veličinama. Implementirati dvije metode koje će računati opseg i površinu trokuta.

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

class Trokut{
private:
	//privatni članovi razreda Trokut
	float _a, _b, _c;
public:
	//podrazumijevani konstruktor
	Trokut(){
	_a=_b=_c=0;
	printf("Poziv podrazumijevanog konstruktora za trokut(%.2f, %.2f, %.2f)\n",
		_a, _b, _c);
	}
	//konstruktor --> postavlja stranice trokuta na zadane veličine
	Trokut(float a, float b, float c){
```

```
		_a = a; _b = b; _c = c;
		printf("Poziv konstruktora za trokut(%.2f, %.2f, %.2f)\n", _a, _b,
			_c);
	}
	//metoda koja vraća opseg trokuta
	float Opseg(){
		return _a+_b+_c;
	}
	//metoda koja vraća površinu trokuta
	float Povrsina(){
		float s=Opseg()/2.; //srednjica trokuta s=(a+b+b)/2
		return sqrt(s*(s-_a)*(s-_b)*(s-_c)); //Heronova formula za površinu
trokuta
	}
	//destruktor razreda
	~Trokut(){
		printf("Poziv destruktora za trokut(%.2f, %.2f, %.2f)\n", _a, _b, _c);
	}
};

int main(){
	Trokut t1; //varijabla se stvara na stogu
	Trokut *t2 = new Trokut (3, 4, 5); //dinamički stvaramo varijablu na heapu
	Trokut &t3 = Trokut (6, 8, 10);//REFERENCA: "poboljšani" pokazivači u C++

	printf("\nOpseg: %5.2f Povrsina: %5.2f\n", t1.Opseg(), t1.Povrsina());
	printf("Opseg: %5.2f Povrsina: %5.2f\n", t2->Opseg(), t2->Povrsina());
	printf("Opseg: %5.2f Povrsina: %5.2f\n\n", t3.Opseg(), t3.Povrsina());

	delete t2; //oslobađamo dinamički zauzetu memoriju na heapu
	return 0;
}
```

OOP_3 - Implementirati razred Automobil koji mora sadržavati tri varijable. Prva varijabla predstavlja ime vozača (*niz znakova duljine 20+1*), druga predstavlja broj litara benzina u rezervoaru (*float*) dok treća predstavlja potrošnju automobila na 100 kilometara (*float*). Razred mora imati podrazumijevani konstruktor koji će vozaču pridjeliti ime „Default“, potrošnju postaviti na 10 litara na 100 km te količinu benzina na 0. Također napraviti i konstruktor koji će postaviti zadano ime vozača i zadanu potrošnju, a stanje benzina postaviti na 0. Implementirati metode NapuniGorivo, PomijeniVozaca i Vozi. Metoda Vozi mora omogućiti da automobil prijeđe zadani broj kilometara ako ima dovoljno benzina inače automobil vozi onoliko kilometara dok ne potroši sav beznin.

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

class Automobil{
private:
	//privatni članovi razreda Automobil
	char _ime[20+1];
	float _benzin;
	float _potrosnja;
public:
	//podrazumijevani konstruktor
	Automobil(){
		strcpy(_ime, "Default"); _potrosnja=10;
		_benzin=0;
		printf("\nDodijeljen podrazumijevani vozac\n");
		printf("Automobil potrosnje %.2f na 100 km\n", _potrosnja);
```

```
	}
	//konstruktor
	Automobil(char *vozac, float potrosnja){
		strcpy(_ime, vozac); _potrosnja=potrosnja;
		_benzin=0;
		printf("\nDodijeljeni vozac: %s\n", _ime);
		printf("Automobil potrosnje %.2f na 100 km\n", _potrosnja);
	}
	//destruktor razreda
	~Automobil(){
		printf("\nKraj voznje, unisten sam!\n\n");
	}
	void NapuniGorivo(float litara){
		_benzin+=litara;
		printf("\nDoliveno %.2f litara benzina\n", litara);
		printf("U rezervoaru ukupno: %.2f litara\n", _benzin);
	}
	void PromijeniVozaca(char *vozac){
		strcpy(_ime, vozac);
		printf("\nPromijenjen vozac: %s\n", _ime);
	}
	//javna metoda razreda Automobil
	//vozi dok ne prijeđe zadani broj kilometara
	//ili dok ne potroši sav benzin iz rezervoara
	void Vozi(float kilometri){
		float temp=_benzin;
		_benzin-=kilometri*_potrosnja/100;
		if(_benzin>0){
			printf("\nPresao %.2f kiometara\n", kilometri);
			printf("Ostalo %.2f litara benzina\n", _benzin);
		}
		else{
			float prosao=100*temp/_potrosnja;
			_benzin=0;
			printf("\nPresao %.2f kilometara i ostao bez benzina!\n",
prosao);
		}
	}
};

int main(){
	Automobil automobil("Michael Schumacher", 9.3);
	automobil.NapuniGorivo(15.7);
	automobil.Vozi(70.);
	automobil.Vozi(50.);
	automobil.PromijeniVozaca("Aerton Senna");
	automobil.NapuniGorivo(5.2);
	automobil.Vozi(200.);

	return 0;
}
```

3.10. Objektno orijentirano programiranje (naprednije tehnike)

Članske funkcije razreda moraju biti deklarirane unutar, ali njhova definicija može biti i izvan razreda. U tom slučaju moramo koristiti operator određivanja dosega (*:: scope resolution operator*). Korištenjem ovog operatora naglašavamo pripadnost članske funkcije određenom razredu.

Dodatna pogodnost koju nudi C++ je podrazumijevani argument. Primjerice ukoliko je prototip članske funkcije razreda `int Dvd::Play(int track=1);`, onda pri pozivu takve funkcije programer može navesti neki cjelobrojni argument, ali ako ne preda nikakav argument, funkcija će se koristiti podrazumijevanim koji je u ovom primjeru 1.

Kod klasa osim članskih funkcija (*metoda*) možemo implementirati i operatore (*operator overloading*). Operatori nam omogućuju kraći i prirodniji zapis operacija vezanih za pripadnu klasu. Također pomoću operatora možemo napisati složene operacije odjednom.

OOPnaprednije_1 - Implementirati razred Dvd koji će simulirati osnovne funkcije DVD playera. Klasa treba sadržavati sljdeće privatne članove:

```
FILE *filmovi;
int power;
struct record{
	int track;
	char title[30+1];
	int duration;
}r;
```

Podrazumijevani konstruktor mora uključiti DVD player (*power=1*) te ispisati odgovarajuću poruku. Destruktor mora zatvoriti datoteku ako je ona otvorena te isključiti DVD player (*power=0*). Klasa mora imati sljedeće javne metode koje moraju vratiti 1 u slučaju uspješnog obavljanja odnosno 0 u slučaju neuspjeha:

```
int Load(char *disc);
int Play(int track);
int Stop();
int Eject();
int Next();
int Previous();
```

Za potrebe ovog primjera na disku se nalazi slijedna binarna datoteka filmovi.dat u kojoj su pohranjeni zapisi o filmovima. Svaki zapis je tipa struct record.

```
#include <stdio.h>

class Dvd{
private://privatni članovi klase
	FILE *filmovi;
	int power;
	struct record{
		int track;
		char title[30+1];
		int duration;
```

```
        }r;
public://javne metode (sučelje klase)
        //u ovom primjeru unutar klase smo napisali samo deklaracije
        //metoda, a njihove definicije ćemo napisati izvan klase
        Dvd();
        int Load(char *disc);
        int Play(int track);
        int Stop();
        int Eject();
        int Next();
        int Previous();
        ~Dvd();
};
//kako definicije metoda u ovom primjeru pišemo izvan klase
//moramo koristiti operator dosega :: (scope resolution)

//podrazumijevani konstruktor
Dvd::Dvd(){
        power=1;
        printf("\nDVD UKLJUCEN\n");
}
//metoda koja učita naziv datoteke te provjerava uspješnost otvaranja
int Dvd::Load(char *disc){
        if((filmovi=fopen(disc, "rb"))!=NULL){
                printf("\nUCITAN DVD: %s\n\n", disc);
                return 1;
        }
    return 0;
}
//kod metoda u argumentima funkcije možemo koristiti podrazumijevane vrijednosti
//tako u slučaju da korisnik pozove funkiju dvd.Play() bez argumenata, klasa radi
//s pretpostavljenim argumentom 1.
int Dvd::Play(int track=1){
        if(filmovi){
                fseek(filmovi, (track-1)*sizeof(record), SEEK_SET);
                fread(&r, sizeof(r), 1, filmovi);
                printf("PLAYING(play): %02d %15s - %d min\n", r.track, r.title,
r.duration);
                return 1;
        }
        return 0;
}
int Dvd::Stop(){
        if(filmovi){
                fseek(filmovi, (r.track-1)*sizeof(record), SEEK_SET);
                printf("STOPPED: %02d %15s - %d min\n", r.track, r.title, r.duration);
                return 1;
        }
        return 0;
}
//metoda zatvara datoteku ako je ona otvorena
int Dvd::Eject(){
        if(filmovi){
                fclose(filmovi);
                printf("\nDISC EJECTED\n");
                return 1;
        }
        return 0;
}
//metoda čita slijedeći zapis iz datoteke
int Dvd::Next(){
        if(filmovi){
                fread(&r, sizeof(r), 1, filmovi);
                printf("PLAYING(next): %02d %15s - %d min\n", r.track, r.title,
                        r.duration);
                return 1;
        }
        return 0;
```

```
}
int Dvd::Previous(){
	if(filmovi){
		fseek(filmovi, -2*sizeof(record), SEEK_CUR);
		fread(&r, sizeof(r), 1, filmovi);
		printf("PLAYING(prev): %02d %15s - %d min\n", r.track, r.title,
			r.duration);
		return 1;
	}
	return 0;
}
//destruktor klase
Dvd::~Dvd(){
	if(filmovi)
		Eject();
	power=0;
	printf("\nDVD ISKLJUCEN\n\n");
}

int main(){
	Dvd dvd;
	dvd.Load("filmovi.dat");
	dvd.Play();
	for(int i=0; i<15; i++) dvd.Next();
	dvd.Previous(); dvd.Previous();
	return 0;
}
```

OOPnaprednije_2 - Implementirati razred Kompleksni koji mora sadržavati dvije varijable tipa double koje će predstavljati realni i imaginarni dio kompleksnog broja. Razred mora sadržavti podrazumijevani konstruktor, konstruktor kojem će se predati dva argumenta tipa double i copy konstruktor. Klasa mora imati i destruktor. Metode koje treba implementirati su:

`void Unesi();`	**omogućuje korisniku da unese kompl. broj sa tipkovnice**
`Kompleksni Konjugiraj();`	**vraća konjugirani kompleksni broj**
`double Modul();`	**vraća modul kompleksnog broja**
`double RealniDio();`	**vraća realni dio kompleksnog broja**
`double ImaginarniDio();`	**vraća imaginarni dio kompleksnog broja**
`void Ispisi();`	**ispisuje kompleksni broj na zaslon**

Također treba implementirati sljedeće operatore za rad s objektima tipa Kompleksni: +, ++, -, --, *, / i logički operator ==.

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

class Kompleksni{
private:
		double realni;
		double imaginarni;
public:
	//podrazumijevani konstruktor
	 Kompleksni(){
	realni=0; imaginarni=0;
    }
	//konstruktor
	Kompleksni(double re, double im){
		realni=re; imaginarni=im;
```

```
   }
   //copy konstruktor
   Kompleksni(const Kompleksni &other){
          realni=other.realni; imaginarni=other.imaginarni;
   }
   //destruktor
~Kompleksni(){
          //printf("Poziv destruktora\n");
}
   //konjugiranje kompl. broja x+yi-->x-yi
Kompleksni Konjugiraj(){
          Kompleksni temp;
          temp.realni=realni;
          temp.imaginarni=imaginarni*-1.0;
          return temp;
}
   //dopušta korisniku da zada kompl. broj preko tipkovnice
   void Unesi(){
          printf("Unesite realni i imaginarni dio kompleksnog broja: ");
          scanf("%lf %lf", &(this->realni), &(this->imaginarni));
   }
   //vraća modul kompleksnog broja sqrt(x^2+y^2)
double Modul(){
          return sqrt((realni*realni)+(imaginarni*imaginarni));
}
   double RealniDio(){
          return realni;
}
   double ImaginarniDio(){
          return imaginarni;
}
   //overload binarnog operatora +
   Kompleksni Kompleksni::operator +(Kompleksni other){
          Kompleksni temp;
          temp.realni=realni+other.realni;
          temp.imaginarni=imaginarni+other.imaginarni;
          return temp;
   }
   //overload unarnog operatora ++
   Kompleksni operator ++(){
          return Kompleksni(realni++, imaginarni++);
   }
   //overload binarnog operatora *
   Kompleksni operator *(Kompleksni other){
          return Kompleksni(realni*other.realni-imaginarni*other.imaginarni,
                 realni*other.imaginarni+imaginarni*other.realni);
   }
   //overload binarnog operatora -
   Kompleksni operator -(Kompleksni other){
          Kompleksni temp;
          temp.realni=realni-other.realni;
          temp.imaginarni=imaginarni-other.imaginarni;
          return temp;
   }
   //overload unarnog operatora --
   Kompleksni operator --(){
          return Kompleksni(realni--, imaginarni--);
   }
   //overload binarnog operatora /
Kompleksni operator /(Kompleksni other){
          double div;
          div=(other.realni*other.realni)+(other.imaginarni*other.imaginarni);
          if(!div) return Kompleksni(); //Dijeljenje s nulom
          Kompleksni temp;
          temp.realni=(realni*other.realni)+(imaginarni*other.imaginarni);
          temp.realni/=div;
          temp.imaginarni=(imaginarni*other.realni)-(realni*other.imaginarni);
          temp.imaginarni/=div;
```

```
            return temp;
    }
      //overload logičkog operatora ==
    bool operator ==(Kompleksni other){
            if((realni==other.realni)&&(imaginarni==other.imaginarni)) return 1;
            else return 0;
    }
    void Ispisi(){
            printf("%.2lf%+.2lfi\n", realni, imaginarni);
    }
};

int main(){
      Kompleksni a; //poziva se podrazumijevani konstruktor
      printf("a="); a.Ispisi();
      Kompleksni b(1,-5); //poziva se odgovarajući konstruktor
      printf("b="); b.Ispisi();
      Kompleksni c = b; //poziva se copy konstruktor
      printf("c="); c.Ispisi();
      Kompleksni *d = new Kompleksni();   //dinamički stvara objekt na heapu
      printf("d="); d->Ispisi();

      a.Unesi(); a++;
      printf("a="); a.Ispisi();
      printf("\nRealni dio kompl. broja a: %.2lf\n", a.RealniDio());
      printf("Imaginarni dio kompl. broja a: %.2lf\n", a.ImaginarniDio());

      b--; b--; b.Konjugiraj();
      printf("\nb = "); b.Ispisi();
      if(a==b) printf("a i b SU jednaki\n");
      else printf("a i b NISU jednaki\n");

      printf("\nModul kompl. broja c je %.2lf\n\n", c.Modul());

      printf("Zbrajanje: a+b=");
      *d=a+b;
      d->Ispisi();

      printf("\nOduzimanje: a-b=");
      *d=a-b;
      d->Ispisi();

      printf("\nMnozenje: a*b=");
      *d=a*b;
      d->Ispisi();

      printf("\nDjeljenje: a/b=");
      *d=a/b;
      d->Ispisi();
      //overloading operatora nam omogućuje kompaktniji zapis operacija nad
      //objektima
      printf("\nVise operacija odjednom: a+2*b-a*c=");
      *d=a+Kompleksni(2, 0)*b-a*c;
      d->Ispisi();


  delete d; //brisemo dinamički zauzetu memoriju
  return 0;
}
```

OOPnaprednije_3- Implementirati razred Stog koji mora imati punu funkcionalnost stoga. Stog implementirati jednostruko povezanom listom. Razred mora imati konstruktor i

destruktor. Implementirati funkcije stavi, skini i prazan.

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

class Stog{
private: //privatni članovi razreda
	struct atom{
		int element;
		struct atom *sljed;
	};
	atom *vrh;
public:
	//deklaracije javnih metoda
	Stog();
	void Stavi(int element);
	void Skini(int *element);
	int Prazan();
	void Unisti(atom *lokalni);
	~Stog();
};
//definicije javnih metoda izvan klase, koristimo :: scope operator

//podrazumijevani konstruktor, inicijaliziramo listu
Stog::Stog(){
	vrh=NULL;
	printf("\n*** Stvoren stog ***\n\n");
}
//metoda za stavljanje elementa na stog
void Stog::Stavi(int element){
	atom *novi;
	novi=new atom;
	if(novi!=NULL){
		novi->element=element;
		novi->sljed=vrh;
		vrh=novi;
		printf("Dodan element: %d\n", element);
	}
}
//metoda za skidanje elementa iz liste
void Stog::Skini(int *element){
	atom *pom;
    if(vrh==NULL) return;
	*element=vrh->element;
	pom=vrh->sljed;  //adresa novog vrha
	free(vrh); //obriši stari vrh
	vrh=pom; //postavi novi vrh
	printf("Skinut element: %d\n", *element);
}
//ako je stog prazan vraća 1, inače 0
int Stog::Prazan(){
	if(vrh==NULL)return 1;
	else return 0;
}
//rekurzivno uništavamo svaki atom
void Stog::Unisti(atom *lokalni){
	if(lokalni) {
		Unisti(lokalni->sljed);
		free(lokalni);
	}
}
//destruktor razreda poziva metodu Unisti()
Stog::~Stog(){
	Unisti(vrh);
	printf("\n*** Unisten stog ***\n\n");
}
```

```
int main(){
	int element;
	Stog stog;//stog je instanca klase Stog

	for(int i=1; i<=10; i++)
		stog.Stavi(i);

	printf("\n\n");
	while(!stog.Prazan())
		stog.Skini(&element);
	return 0;
}
```