Pokazivac.c

```c
#include <stdio.h>
int main () {
	int a = 4;
	int *b;

	b = &a;
	*b = 8;
	printf ("%d %d\n", a, *b);

	return 0;
}
```

AritmetikaPokazivaca.c

```c
#include <stdlib.h>

int main (void) {
  char *c; short *s; int *i; float *f; double *d;
  void *v;
	long dugi; double dupli;
	long *pdugi; double *pdupli;
	short veldugi, veldupli;
  c = 0; s = 0; i = 0; f = 0; d = 0; v= 0;
  // Inkrement za 1
  c++;
  s++;
  i++;
  f++;
  d++;
  // v++; Javlja pogresku jer se ne zna veličina


	veldugi = sizeof (dugi);
	pdugi = &dugi;
	++pdugi;

	veldupli = sizeof (dupli);
	pdupli = &dupli;
	pdupli = pdupli + 2;

	return 0;
}
```

KomunikacijaSFunkcijama.c

```c
void z0 (int tri, int sedam) { // call by value
  int pom;
  pom = tri;
  tri = sedam;
  sedam = pom;
}
void z1 (int *tri, int *sedam) { // call by reference
 int pom;
  pom = *tri;
  *tri = *sedam;
  *sedam = pom;
}
void z2 (int *tri, int *sedam) { // lokalna zamjena adresa
  int *pom;
  pom = tri;
  tri = sedam;
  sedam = pom;
}
int main () {
  int tri=3, sedam=7;

  z0 (tri, sedam);
  z1 (&tri, &sedam);
```

```
  z2 (&tri, &sedam);

  return 0;
}
```

PrimjerZaMalloc.c

```
#include <stdlib.h>
#include <malloc.h>
int main (void) {

  short *i;
  i = (short *) malloc (sizeof (short));
  *i = 7;

  free((void*) i);

  //*i = 7;

  //i = NULL;

  return 0;
}
```

MallocMatrica.c

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

#define r(i,j) r[(i)*n+(j)]

void fatal (char *poruka) {
  fputs (poruka, stderr); fputs ("\n", stderr);
  exit (1);
}

int main() {
  int *p, n, pom, i, j;
  long *r;
  FILE *d;
  d = fopen ("polje", "r");
  if (d == NULL) fatal("Ne mogu otvoriti datoteku");
  for (n = 0; fscanf(d, "%d", &pom) == 1; n++);
  fseek (d, 0L, SEEK_SET);

  p = (int *) malloc (n * sizeof (int));
  if (p == NULL)
    fatal ("Nema dovoljno memorije za ucitati polje");

  for (n = 0; fscanf(d, "%d", &p[n]) == 1; n++);
  fclose (d);

  if ((r = (long *) malloc (n*n*sizeof(long))) == NULL)
    fatal ("Nema dovoljno memorije za rezultat");
  for (j = 0; j < n; j++) {
    r(0,j) = p[j];
    for (i = 1; i < n; i++) {
      r(i,j) = r(i-1,j) * r(0,j);
    }
  }
  free (p);
  for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
      printf ("%10ld", r(i,j));
    }
    printf ("\n");
  }
  d = fopen ("npolje", "wb");
```

```
  fwrite (&n, sizeof (int), 1, d);
  fwrite (r, sizeof (long), n*n, d);

  fclose (d);
  free (r);
  return 0;
}
```

Realloc.c

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <stdlib.h>
#define MAXBUF 512

void fatal (char *poruka) {
  fputs (poruka, stderr); fputs ("\n", stderr);
  exit (1);
}

int main (int argc, char *argv[]) {
       FILE *du, *di;
       long *polje;
       char buf [MAXBUF+1];
       int i, n;

  if (argc != 3)
              fatal ("Poziv programa: realloc stara.txt nova.txt");
       if ((du = fopen (argv[1], "r")) == NULL)
              fatal ("Ne moze se otvoriti ulazna datoteka");

  n = 0; polje = NULL;
  do {
              polje = (long *) realloc (polje, (n+1)*sizeof (long));
              if (polje == NULL)  fatal ("Nedovoljno memorije");
              polje[n++] = ftell (du);
       } while (fgets (buf, MAXBUF, du) != NULL);

  if ((di = fopen (argv[2], "w")) == NULL)
              fatal ("Ne moze se stvoriti izlazna datoteka");

  for (i = n-2; i >= 0; i--) {
              fseek (du, (long) polje[i], SEEK_SET);
              fgets (buf, MAXBUF, du);
              fputs (buf, di);
       }
       free (polje);  fclose (du); fclose (di);
       return 0;
}
```

DatumJMBG.c

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

int Kontrola (char JMBG[]) {
  int i, kz;
  long suma;
  char tez [12+1] = "765432765432";
  suma = 0;
  for (i = 0; i < 12; i++) {
    suma += (JMBG[i] - '0') * (tez[i] - '0');
  }
  kz = 11 - (suma  % 11);
  if (kz == 10) kz = -1; // pogr. kontrolna znamenka
  if (kz == 11) kz = 0;
  return kz;
}
```

```c
char * datum (char *JMBG) {
	int d, m, g; // lokalne varijable – vrijede samo unutar funkcije
	char *p; // pokazivač je deklariran, ali nije inicijaliziran!
	// Citanje iz niza
	// JMBG ima oblik DDMMYYYXXXXXX
	sscanf (JMBG, "%2d%2d%3d", &d, &m, &g);
	// Ispis u niz treba biti oblika DD.MM.GGGG.
  p = (char *) malloc (11 * sizeof(char));
  //* pokazivač je inicijaliziran i naredbom malloc zauzeta je memorija potrebna za pohranu datuma u
formatu DD.MM.GGGG */
	// 21. stoljece?
	if (g < 100){
		g += 2000;
	} else {
		g += 1000;
	}
	sprintf (p, "%02d.%02d.%04d", d, m, g);
  // datum je zapisan na adresu na koju pokazuje pokazivač p
	return p;
}

int main () {
	int kz;
	char JMBG [13+1], *p;
	while (1) {
		printf ("\nUpisite JMBG >");
		scanf ("%13s", JMBG);
		kz = Kontrola (JMBG);
		if ((JMBG [12] - '0') == kz){
			p = datum (JMBG);
			printf ("\nDatum rodjenja je %s", p);
	       free(p);
		} else {
			printf ("\nNeispravan JMBG: %s:\n", JMBG);
			break;
		}
	}
	return 0;
}
```

DvodimenzionalnoPolje.c

```c
// DvodimenzionalnoPolje.c
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#define MAX_STU 8
#define MAX_RED 10

int polje(int a[], int max_s, int m, int n, int i, int j) {
	// a    : polje
	// max_s: deklarirani broj stupaca, potreban radi nalazenja pocetka retka
	// m, n : radne dimenzije matrice
	// i, j : indeksi retka i stupca clana

  printf("Ulaz: max_s=%d, m=%d, n=%d, i=%d, j=%d \n", max_s, m, n, i, j);
  if (i < 0 || i >= m || j < 0 || j >= n) {
    printf("\n\nNedopustena vrijednost indeksa\n");
    return 0;
  } else {
    printf ("Clan [ %d ][ %d ] polja s maksimalno %d stupaca \n",
                      i,    j,                    max_s);
    printf ("ekvivalentan je clanu jednodimenzionalnog polja [%d]=%d\n",
                                                i*max_s+j, a[i*max_s+j]);
    return 1;
  }
}
int main (void) {
  FILE *fi;
```

```
  int i, j, m, n, a[MAX_RED][MAX_STU];
  fi = fopen ("UlazZaDvodimenzionalnoPolje.txt", "r");
  if (!fi) {
    printf ("Nema ulazne datoteke");
       return 1;
  }
  fscanf (fi, "%d %d\n", &m, &n);
  printf ("m=%d, n=%d\n", m, n);
       printf ("Dvodimenzionalno Jednodimenzionalno\n");
  for (i = 0; i < m; i++) {
    for (j = 0; j < n; j++) {
      fscanf (fi, "%d", &a[i][j]);
      printf("a[%d][%d]=%3d \t a[%2d]\n",
                                                    i, j, a[i][j], i*MAX_STU+j);
    }
  }
  fclose (fi);
  while (1) {
    printf("\nUpisite i, j >");
    scanf("%d %d", &i, &j);
    if (polje (&a[0][0], MAX_STU, m, n, i, j) != 1) break;
         // Polje se predaje kao pokazivac na pocetak matrice da izostane upozorenje
  }
       return 0;
}
```

SumaUPolju.c

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

float zbroj_poz (int brred, int brstup, int maxstup, float *p ) {
/* brred, brstup, int maxstup – ulazni parametri funkcije; njihova eventualna promjena u funkciji
 neće biti sačuvana nakon povratka u glavni program.
 Budući da se polje uvijek prenosi u funkciju kao pokazivač,
 vrijednost članova polja moguće je u funkciji mijenjati */
       int i, j;
       float suma;
       suma = 0.0;
       for( i=0; i < brred; i++ )
              for( j = 0; j < brstup; j++ )
                     if (p [i*maxstup + j] > 0) suma += p[i*maxstup + j];
                     /* može i suma += *(p + i*maxstup + j)
                      nije dozvoljeno p(i,j), p[i][j] i slično */
       return suma;
}

#define MAXRED       100
#define MAXSTUP      20
int main() {
       int i, j, red, stup;
       float zbroj, mat[MAXRED][MAXSTUP];
       /* ovdje slijedi postavljanje stvarnog broja redova i stupaca
       (red<=MAXRED, stup<=MAXSTUP, punjenje polja */
  srand ((unsigned) time(NULL));
  red = rand() % MAXRED + 1;
  stup = rand() % MAXSTUP + 1 ;
  for (i = 0; i < red; ++i)
    for (j = 0; j < stup; ++j)
      mat[i][j] = (float) rand() - RAND_MAX;

       zbroj = zbroj_poz(red, stup, MAXSTUP, (float *) mat);
       return 0;
}
```

DohvatiBrisi.c

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

```c
struct zapis {
  int mbr;
  char ime [40+1];
  char spol [1+1];
};

struct zapis dohvati_brisi (FILE* direktna, int mbr) {  // funkcija vraća dohvaćeni zapis
  struct zapis z, z1; // z je varijabla tipa 'struct zapis'
  fseek (direktna,(mbr-1) * sizeof(struct zapis), SEEK_SET);
  /* direktan pristup zapisu jer je zadano da šifra odgovara rednom broju zapisa */
  fread (&z, sizeof(z), 1, direktna);
  if (z.mbr) {
        z1.mbr = 0;
        /* zapisi se iz direktne datoteke ne brišu fizički. Vrijednost 0 označava prazan zapis */
        fseek (direktna, (-1) * (int) sizeof(struct zapis), SEEK_CUR);
            /* nakon čitanja potrebno se vratiti na početak zapisa, da bi se prazni zapis upisao
na isto mjesto */
        fwrite (&z1, sizeof (z1), 1, direktna);
  }
  return z;
}
int main () {
  struct zapis z;
  int mbr, i;
  FILE *slijedna, * direktna;
  if (!(slijedna = fopen ("studenti.txt", "r"))){
        printf ("\nNema ulazne dateke studenti.txt\n");
        return 1;
  }
  // Stvaranje direktne neformatirane datoteke iz slijedne formatirane
  if (!(direktna = fopen ("studenti.dat", "w+b"))){
        printf ("\nNe mogu otvoriti direkltnu datoteku studenti.dat\n");
        fclose(slijedna);
        return 1;
  }
  // Pražnjenje direktne datoteke
  z.mbr = 0;
  for (i = 0; i <= 999; i++) fwrite (&z, sizeof (z), 1, direktna);
   // Prijepis iz slijedne formatirane u direktnu neformatiranu
  while (fscanf (slijedna, "%3d%s%s", &z.mbr, z.ime, z.spol) != EOF) {
      printf ("Procitan zapis %d %s %s\n", z.mbr, z.ime, z.spol);
        fseek (direktna,(z.mbr-1)*sizeof(struct zapis), SEEK_SET);
        fwrite (&z, sizeof (z), 1, direktna);
   }
 // Dohvat i brisanje
  while (1) {
        printf ("\nUpisite mbr >");
        scanf ("%d", &mbr);
        z = dohvati_brisi (direktna, mbr);
        if (z.mbr) {
              printf ("Izbrisan zapis: %d %s %s", z.mbr, z.ime, z.spol);
        } else {
              printf ("\nNema zapisa s mbr = %d\n", mbr);
              break;
        }
  }
  return 0;
}
```

Porez.c

```c
#include <stdio.h>
#include <stdlib.h>
struct element {
      char   JMGB[13+1];
      char   ImePrezime[40+1];
      float  Prihod;
      float  PlatitiPorez;
};
typedef struct element zapis;
```

```
zapis* Ucitaj (char *FileName, int *BrElem) {
	int		i; //lokalne varijable
	long	Br;
	FILE	*fp;
	zapis	*Polje = NULL;
	zapis	Elem;
	if ((fp = fopen(FileName, "rb")) == NULL) {
		printf("Ne mogu otvoriti datoteku");
		return NULL;
	}
	fread (&Br, sizeof(long), 1, fp );
	if( Br > 0 ) {
	Polje = (zapis *) malloc (Br * sizeof(zapis));
	// inicijalizacija varijable Polje – sadrzi adresu
	// na kojoj je slobodno kontinuirano područje od
	// Br * sizeof(zapis) byte-ova
	}
	else {
		printf ("Neispravan broj elemenata");
		return NULL;
	}
	i = 0;
	// pridruživanje sadržaja jedne strukture drugoj kopiranjem sadržaja memorije
	// moglo je i: fread (Polje, sizeof(zapis), Br, fp);
	while (fread (&Elem, sizeof(zapis), 1, fp) == 1)  Polje[i++] = Elem;
	*BrElem = Br; //uz pretpostavku da smo sve uspješno pročitali
	fclose(fp);
	return Polje;
	}
int	MaxPorez( zapis *Polje, int BrElem ) {
	int	i, MaxInd = 0;
	float Max;
	Max = Polje[0].PlatitiPorez;
	for(i = 1; i < BrElem; i++ ) {
		if( Polje[i].PlatitiPorez > Max ) {
			MaxInd = i;
			Max = Polje[i].PlatitiPorez;
		}
	}
	return MaxInd;
}

int main(){
	int	BrElem, MaxInd; //lokalne varijable
	char FileName[40];
	zapis *Polje;
	printf("Unesite ime datoteke : ");
	gets(FileName);
	Polje = Ucitaj(FileName, &BrElem);
	if( Polje != NULL ) {
		MaxInd = MaxPorez(Polje, BrElem);
		printf("Najviše poreza treba platiti %s, u iznosu od %f", Polje[MaxInd].ImePrezime,
		Polje[MaxInd].PlatitiPorez );
	}
	free (Polje);
}
```

IspisiTrazi.c

```
#include <stdio.h>
void ispisi(int A[], int n) {
	int i;
	for (i = 0; i < n; i++)
    printf("%d\n", A[i]);
}

// funkcija koja traži element u polju
// vraća 1 ako se element nalazi u polju, a 0 inače
int trazi(int A[], int n, int trazeni) {
  int i;
```

```
  for (i=0; i<n; i++) {
    if (trazeni == A[i])
      return 1;
  }
  return 0;
}

int main () {
       int A [] = {4, -3, 5, -2, -1, 2, 6, -2};
       int b, c;
       ispisi (A, 8);
       while (1) {
              printf("Upisite trazeni broj\n");
              scanf ("%d2", &b);
              c = trazi (A, 8, b);
              printf ("Trazeni broj %d se ", b);
              if (!c) printf ("ne ");
              printf ("nalazi u polju A\n");
       }
}
```

CitanjePoBlokovima.c

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

#define BLOK 60


typedef struct
{
       int pbr;
       char naziv[30];
       char opcina[30];

} zapmjesto;


int nadjiMjesto(char *naziv,zapmjesto *z)
{
       FILE *f;
       zapmjesto zm;
       int podatak_nadjen=0;
       int i;
       int brojacBlokova=0, brojacuBloku=0;

       if((f=fopen("mjesta.dat","rb")) == NULL)
       {
              printf("Greska: otvaranje datoteke!\n");
              exit (1);
       }

       //trazenje bloka
       while(fread(&zm,sizeof(zm),1,f))
       {
        brojacBlokova++;
              //debug ispis
              printf("Blok: %d; vodeci zapis: %s, %d %s\n", brojacBlokova, zm.naziv, zm.pbr,
zm.opcina);

              if(strcmp(naziv,zm.naziv)<0)
              {
                     break;
              }
              else if(strcmp(naziv,zm.naziv)==0)
              {
                     podatak_nadjen=1;
                     break;
              }
```

```
		else
			fseek(f, 1L * brojacBlokova * BLOK * sizeof(zm), SEEK_SET);

}
	if(!podatak_nadjen)
	{

	 if (!feof(f)) brojacBlokova-=1; // pronađen je zapravo u prethodnom bloku, osim ako smo na
kraju
	   //trazenje podatka u bloku
	 printf("Podatak se mora nalaziti u bloku %d\n", brojacBlokova);

		fseek(f, 1L * (brojacBlokova-1) * BLOK * sizeof(zm), SEEK_SET); //idemo na početak tog
prethodnog
		i=0;
		do
		{
			fread(&zm,sizeof(zm),1,f);

			//debug ispis
			printf("Blok %d, zapis %d: %s, %d %s\n", brojacBlokova, i, zm.naziv, zm.pbr,
zm.opcina);

			if(strcmp(naziv,zm.naziv)==0)
			{
				podatak_nadjen=1;
			}
			i++;
		}while(i<BLOK && !podatak_nadjen && !feof(f));
	}

	fclose(f);

	*z=zm;
	return (podatak_nadjen);

}

int main()
{
	zapmjesto zm;
	char trazi[30];

	do
	{
		printf("Upisi naziv mjesta (k - kraj): ");
		scanf("%s",trazi);
		if((strcmp(trazi,"k")!=0))
			if(nadjiMjesto(trazi,&zm))
			{
				printf("Trazeno mjesto:\n%d; %s; %s\n",zm.pbr,zm.naziv,zm.opcina);
			}
			else
				printf("Trazeno mjesto ne postoji!!\n");
	}while (strcmp(trazi,"k")!=0);

	return 0;
}
```

BinarnoPretrazivanje.c

```
---------------------------------------------------------------------
#include <stdio.h>
typedef int tip;
#define MAX 100

// binarno pretrazivanje polja a[] od n elemenata
// vraca indeks nadjenog elementa ili -1 ako trazeni ne postoji
int BinTraz (tip a[], tip x, int n, int *brojpokusaja) {
  int donji, srednji, gornji;     // granice podpolja koje se pretrazuje
	// inicijalizacija
```

```
  donji = 0; gornji = n - 1; *brojpokusaja = 0;

  while (donji <= gornji) {
              (*brojpokusaja)++;
    srednji = (donji + gornji) / 2; // "prepolovi" (pod)polje
              printf ("Trazim u a[%2d:%2d], a[%2d]=%2d\n",
                    donji, gornji, srednji, a[srednji]);
    if (a [srednji] < x)
      donji = srednji + 1;                              // trazeni u desnom dijelu
    else if (a [srednji] > x)
      gornji = srednji - 1;                             // trazeni u lijevom dijelu
    else
      return srednji;                    // nadjen
  }
  return -1;              // nije nadjen
}

int main (void) {
       tip a[100], x;
       int i, n, brojpokusaja;
       FILE *fi;

       // citanje ulaznih podataka
       fi = fopen ("UlazZaBinarnoPretrazivanje.txt", "r");
       for (n = 0; n < MAX && fscanf (fi, "%d", &a[n]) == 1; n++);
       printf ("Broj clanova polja: %d\nClanovi:\n", n);
       for (i = 0; i < n; i++) printf ("%d->%d ", i, a[i]);
       printf ("\n");
       fclose (fi);
       // binarno trazenje
       while (1) {
              printf ("Upisite trazenu vrijednost, -1 za kraj >");
              scanf ("%d", &x);
              if (x == -1) return 0;
              i = BinTraz (a, x, n, &brojpokusaja);
              if (i < 0 ) {
                     printf ("Vrijednost %d nije pronadjena!\n"
                                   "Broj pokusaja:%d\n",
                                                 x, brojpokusaja);
              } else {
                     printf ("Vrijednost %d pronadjena je na poziciji %d.\n"
                                   "Broj pokusaja:%d\n",
                                    x, i, brojpokusaja);
              }
       }
  return 0;
}
```

Hash.c

```
-------------------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define VELJMBG 13
#define VELIME 14
struct zapis{
  char JMBG[VELJMBG+1];
  char prezime[VELIME+1];
};

#define BLOK 512L          // Blok na disku
#define N 350                     // Ocekivani broj zapisa:
#define C ((int) (BLOK / sizeof (struct zapis)))      // Broj zapisa u pretincu
#define M ((int) (N / C *1.3))    // Broj pretinaca, kapacitet 30% veci od minimalnog:
/*
#define C 1
#define M 5
*/
```

```c
struct zapis ulaz, pretinac [C];

void Pogreska (char *poruka) {
  fputs (poruka, stderr); fputs ("\n", stderr);
  exit (1);
}

void Isprazni (FILE *ft) {
  int i;
  for (i = 0; i < C; i++) pretinac [i].JMBG[0] = '\0';
  for (i = 0; i < M; i++) {
    fseek (ft, i*BLOK, SEEK_SET);
    fwrite (pretinac, sizeof (pretinac), 1, ft);
  }
  printf ("Tablica ispraznjena N=%d, C=%d, M=%d\n", N, C, M);
  printf ("Velicina pretinca = %d\n", sizeof (pretinac));
}

int OdrediPrim (int m) {
  int i, prim, kraj;
       printf ("Trazenje najveceg prim broja <= %d\n", m);
  prim = m+1;
  do {
    prim--;
    kraj = (int) pow ((double) prim, (double) 0.5);
    for (i = kraj; i > 0; i--) {
      printf ("%d/%d ", prim, i);
      if (prim % i == 0) break;
    }
  } while (i != 1);
  return prim;
}

int Kontrola (char JMBG[]) {
  int i, kz;
  long suma;
  char tez [12+1] = "765432765432";
  suma = 0;
  for (i = 0; i < 12; i++) {
    suma += (JMBG[i] - '0') * (tez[i] - '0');
  }
  kz = 11 - (suma  % 11);
  if (kz == 10) kz = -1; // pogr. kontrolna znamenka
  if (kz == 11) kz = 0;
  return kz;
}

int Adresa (char JMBG[], int prim) {
  int i, a, b, c, adr;
  char pom [4+1];
  pom [4] = '\0';
  // Preklapanje
  for (i = 0; i < 4; i++) pom[i] = JMBG[i+4];
  a = atoi (pom);
  for (i = 0; i < 4; i++) pom[i] = JMBG[3-i];
  b = atoi (pom);
  for (i = 0; i < 4; i++) pom[i] = JMBG[11-i];
  c = atoi (pom);
  // Dijeljenje
  adr = (a + b + c) % prim;
  printf ("Izracunata adresa (%d) = %d\n", (a + b + c), adr);
  return adr;
}

int Upis (struct zapis ulaz, FILE *ft, int prim) {
  int i, j, poc;
  i = Adresa (ulaz.JMBG, prim);
  // Upamti izračunatu adresu kao početnu
  poc = i;
  do { // Ponavljaj dok ne upišeš ili ustanoviš da je datoteka puna
```

```
            // Čitaj iz pretinca sve upisane zapise
    fseek (ft, i*BLOK, SEEK_SET);
    fread (pretinac, sizeof (pretinac), 1, ft);
    for (j = 0; j < C; j++) {
      if (pretinac[j].JMBG[0] != '\0') {
                          // Ako zapis nije prazan
                          printf ("Vec upisani JMBG =");
                          printf ("%s\n", pretinac[j].JMBG);
            if (strncmp (pretinac[j].JMBG, ulaz.JMBG, VELJMBG) == 0) {
            // Ako je upisani JMBG identičan ulaznom
                                printf ("Vec postoji zapis s JMBG %s\n", ulaz.JMBG);
          return 1;
          }
      } else {
        // Upiši ulazni zapis na prazno mjesto
            pretinac[j] = ulaz;
            fseek (ft, i*BLOK, SEEK_SET);
            printf ("U pretinac %d upisujem %d. zapis\n", i, j);
            fwrite (pretinac, sizeof (pretinac), 1, ft);
        return 1;
      }
    }
    // U pretincu nema mjesta, prijeđi ciklički na sljedećega
    i = (i + 1) % M;
    printf ("Nema mjesta, slijedi pretinac = %d\n", i);
  } while (i != poc); // Dok se ne ne vratiš na početni
  return 0;  // Niti u jednom zapisu nema mjesta
}

void Ispis (FILE *ft) {
  // Ispis sadržaja tablice raspršenih adresa
  int i, j;
  printf("Ispis sadrzaja tablice \n");
  for (i = 0; i < M; i++) {
    fseek (ft, i*BLOK, SEEK_SET);
    fread (pretinac, sizeof (pretinac), 1, ft);
    for (j = 0; j < C; j++) {
      if (pretinac[j].JMBG[0] != '\0') {
                          // Ako zapis nije prazan
                          printf ("Zapis na adresi %d:", i);
                          printf ("%s %s \n", pretinac[j].JMBG, pretinac[j].prezime);
      }
    }
  }
}

int Trazi (char mbr[], FILE *ft, int prim, struct zapis *ulaz) {
  int i, j, poc;
  i = Adresa (mbr, prim);
  // Upamti izračunatu adresu kao početnu
  poc = i;
  do {  // Ponavljaj dok ne nadješ ili ustanoviš da ga nema
             printf ("Citam %d. zapis\n", i);
    fseek (ft, i*BLOK, SEEK_SET);
    fread (pretinac, sizeof (pretinac), 1, ft);
    for (j = 0; j < C; j++) {
      if (pretinac[j].JMBG[0] != '\0') {
             // Ako zapis nije prazan
                          printf ("%d. upisani JMBG =%s\n", j, pretinac[j].JMBG);
            if (strcmp (pretinac[j].JMBG, mbr) == 0) {
            // Ako je upisani JMBG identičan ulaznom
                                *ulaz = pretinac[j];
          return 1;
        }
      } else {
             // Nema zapisa
            return 0;
      }
    }
    // Pretinac je pun, prijeđi ciklički na sljedećega
```

```
    i = (i + 1) % M;
  } while (i != poc); // Dok se ne ne vratiš na početni
       return 0; // Svi pretinci posjećeni, zapis nije pronađen
}

int main() {
  FILE *fi, *ft;
  char mbr [VELJMBG+1];
  int prim;
       if ((fi = fopen ("UlazZaHash.txt", "r")) == NULL)
              Pogreska ("Ne mogu otvoriti datoteku \"ulaz\"");
       if ((ft = fopen ("tablica","w+")) == NULL)
              Pogreska ("Ne mogu otvoriti datoteku \"tablica\"");
       printf ("Ulazni zapis je velicine %d\n", sizeof (ulaz));
  Isprazni (ft);
  Ispis (ft);
  // Odredi prim broj za dijeljenje
  prim = OdrediPrim (M);
  printf ("Prim broj za dijeljenje = %d\n", prim);
  // Čitaj slijedno JMBG, prezime, ime dok ima podataka
       getchar ();
  while (fscanf (fi, "%13s%s", ulaz.JMBG, ulaz.prezime) != EOF) {
    printf ("Procitan zapis %s %s \n", ulaz.JMBG, ulaz.prezime);
    if (Kontrola (ulaz.JMBG) == atoi (&ulaz.JMBG [12])) {
    // Ako je kontrolna znamenka ispravna
      if (!Upis (ulaz, ft, prim))
        Pogreska ("Tablica je puna\n");
    } else {
                    printf ("Neispravan JMBG %s", ulaz.JMBG);
                    printf(", kontrolna znamenka treba biti %d\n", Kontrola (ulaz.JMBG));
    }
              printf ("\n");
  }
  fclose (fi);
       getchar ();
  Ispis (ft);
  while (1) {
              printf ("Upisite JMBG >");
    scanf ("%s", mbr);
    if (Trazi (mbr, ft, prim, &ulaz)) {
                    printf ("%s %s\n", ulaz.JMBG, ulaz.prezime);
    } else {
      printf ("JMBG %s nije u tablici\n", mbr);
      break;
    }
  }
  fclose (ft);
  return 0;
}
```

ElementarnaRekurzija.c

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

void f (int i) {
       int p[40000];
       f (i+1);
       return;
}
int main (void) {
       f(1);
       return 0;
}
```

FaktorijeliRekurzijom.c

```
// FaktorijeliRekurzijom.c
#include <stdio.h>
#include <conio.h>
```

```c
#include <stdlib.h>
long fakt (int n) {
	long nfakt;
	if (n <= 1) { // 0! = 1! = 1
		nfakt = 1;
	} else {                    // n! = n* (n-1)!
		nfakt = n * fakt(n-1);
		if (nfakt < 0 ) {
			printf ("\nNe moze se izracunati %d! kao long (nfakt=%ld)\n", n, nfakt);
			(void) getch();
			exit (1);
		}
	}
	return nfakt;
}

int main (void) {
	int n;
	while (1) {
		printf ("\nUpisite n>"); // primjeri: n=15,16,17...
		scanf ("%d", &n);
		printf ("%d! = %ld",n, fakt (n));
	}
	return 0;
}
```

PotencijaRekurzijom.c

```c
#include <stdio.h>
int pot(int x, int y) {
  int ret;

  if (y <= 0) ret = 1;
  else ret = x * pot(x, y - 1);

  return ret;
}
int main () {
  int x, y, rez;
  while (1) {
    printf ("Upisite cijeli broj za bazu i nenegativni cijeli broj za eksponent>");
    scanf ("%d %d", &x, &y);
    if (y < 0) break;
         rez = pot (x, y);
         printf ("%d na potenciju %d = %d\n", x, y, rez);
  }
  printf ("\nNegativni eksponent\n");
  return 0;
}
```

RekurzivniIspisRedom.c

```c
#include <stdio.h>
// Rješenja s dva ulazna argumenta:

// Ispisuje uzlazno (od n1 do n2)
void ispis_u2a (int n1, int n2) {
   if (n1 <= n2) {
      printf("\n%d", n1);
      ispis_u2a (n1 + 1, n2);
   }
}
// Ispisuje uzlazno (od n1 do n2)
void ispis_u2b (int n1, int n2) {
   if (n1 <= n2) {
      ispis_u2b (n1, n2 - 1);
      printf("\n%d", n2);
   }
}
// Ispisuje silazno (od n2 do n1)
```

```c
void ispis_s2a (int n1, int n2) {
   if (n1 <= n2) {
      printf("\n%d", n2);
      ispis_s2a (n1, n2 - 1);
    }
}
// Ispisuje silazno (od n2 do n1)
void ispis_s2b( int n1, int n2) {
   if (n1 <= n2) {
      ispis_s2b (n1 + 1, n2);
      printf("\n%d", n1);
   }
}
// Rješenja s jednim ulaznim argumentom:

// Ispisuje uzlazno (od 1 do n2)
void ispis_u1 (int n) {
   if (n >= 1) {
      ispis_u1 (n - 1);
      printf("\n%d", n);
   }
}
// Ispisuje silazno (od n2 do 1)
void ispis_s1 (int n) {
   if (n >= 1) {
      printf("\n%d", n);
      ispis_s1 (n - 1);
   }
}

int main () {
  int n;
  printf ("Upisite najveci cijeli broj za ispis >");
  scanf ("%d", &n);
  ispis_u2a (1, n);
  getchar (); getchar (); // drugi char da bi progutao ENTER od gornjeg scanf-a
  ispis_u2b(1, n);
  getchar ();
  ispis_s2a (1, n);
  getchar ();
  ispis_s2b(1, n);
  getchar ();
  ispis_u1 (n);
  getchar ();
  ispis_s1 (n);
  return 0;
}
```

AritmetickiNiz.c

```c
#include <stdio.h>
int aniz(int a0, int d, int n) {
   if (n == 0) return a0;
   else return d + aniz(a0, d, n-1);
}
int main () {
       int a0, d, n, nclan;
       while (1) {
              printf ("\nUpisite nulti clan, diferenciju i indeks zadanog clana >");
              scanf ("%d %d %d", &a0, &d, &n);
              if (n < 0) break;
              nclan = aniz (a0, d, n);
    printf ("\n %d. clan aritmetickog niza, s nultim clanom %d i diferencijom %d: %d \n", n, a0, d,
nclan);
       }
       printf ("\nNegativni indeks clana %d\n", n);
       return 0;
}
```

```c
// Euklid.c
```

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

// NZM Rekurzivno
int nzm (int a, int b) {
	printf ("nzm(%d,%d) = ", a, b);
	if(b == 0) {
		printf ("%d\n", a);
		return a;
	}
  // ostali pozivi prosljedjuju isti rezultat
	return nzm (b, a % b);
}
// NZM Uklanjanjem rekurzije
int nzm1 (int a, int b) {
	int t;
L1:
	printf ("nzm(%d,%d) = ", a, b);
	if (b == 0) {
		printf ("%d\n", a);
		return a;
	}
	  // nzm (b, a % b)
	t = b;
		b = a%b;
		a = t;
	goto L1;
}
// NZM Iterativno, izbjegavanjem goto:
int nzm2 (int a, int b) {
	int t;
	while (b != 0) {
		printf ("nzm(%d,%d) = ", a, b);
		t = b;
		b = a%b;
		a = t;
	}
  printf ("nzm(%d,%d) = ", a, b);
	printf ("%d\n", a);
  return a;
}
int main (void) {
  int a, b;

  while (1) {
    printf ("Upisite 2 cijela nenegativna broja >");
    scanf ("%d %d", &a, &b); // primjer: 22 8
    if ((a < 0 || b < 0) || (a == 0 && b == 0)) {
      printf("Gotovo!\n");
			break;
    } else {
			printf("Najveca zajednicka mjera brojeva %d i %d je:\n\n", a, b);
			printf("Rekurzivno           : %d\n\n", nzm(a, b));
			printf("Nerekurzivno s goto  : %d\n\n", nzm1(a, b));
			printf("Nerekurzivno bez goto: %d\n\n", nzm2(a, b));
		}
  }
	return 0;
}
```

Fibonacci.c

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

// Fibonacci izravno
int FI(int n) {
  int i, fib, f1, f2;
  if (n <= 1) {
```

```
    fib = 1;
       } else {
       f1 = 1; f2 = 1;            // predzadnji i zadnji broj
       for (i = 2; i <= n; i++) {
              fib = f1 + f2;      // novi broj
              if (fib < 0) {
                     printf ("Ne moze se prikazati %d. Fibonaccijev broj!\n", i);
                     (void) getch();
                     exit (1);
              }
              f1 = f2;                          // zadnji postaje predzadnji
              f2 = fib;                         // novi postaje zadnji
       }
  }
       return fib;
}
// Fibonacci rekurzivno
int FR(int n, int *brojac) {
  int fib;
  if (n <= 1) {
    fib = 1;
  } else {
    fib = FR(n-2, brojac) + FR(n-1, brojac);
       if (fib < 0) {
              printf ("\nNe moze se prikazati %d. Fibonaccijev broj!", n);
              (void) getch();
              exit (1);
       }
  }
  (*brojac) ++;
  return fib;
}
// F(0) = F(1) = 1
// F(i) = F(i-2)+F(i-1); i>1
int main () {
  int n, brojac, fib;
  while (1) {
    brojac = 0;
    printf("Upisite broj >"); // Primjeri: n=5,40,50
    scanf("%d", &n);
    if (n < 0) {
              printf ("gotovo!\n");      break;
    } else {
              fib = FI (n);
              printf("%d. Fibonaccijev broj = %d , Izravno! \n", n, fib);
              fib = FR(n, &brojac);
              printf("%d. Fibonaccijev broj = %d , Rekurzivno u %d iteracija\n",
                                                                                n,
fib, brojac);
    }
  }
  return 0;
}
```

Rekurzija.c

```
// Rekurzija.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAXA 10

// ispis znaka c u zadanoj duljini n
void nznak (int c, int n) {
  while (--n >= 0) putchar(c);
}
// ispis polja
void ispisi(int A[], int n) {
       int i;
       printf("\n");
```

```
	for (i = 0; i < n; i++) printf(" A[%d]",i);
	printf("\n");
	for (i = 0; i < n; i++) printf("%5d", A[i]);
	printf("\n");
}
// Rekurzivno trazenje indeksa clana u polju
int trazi (int A[], int x, int n, int i) {
	// A-polje x-trazeni i-indeks od kojeg se trazi
	int ret;
	nznak(' ', i*5); printf("^^^^^\n");
	if (i >= n) ret = -1;
	else if (A[i] == x) ret = i;
	else ret = trazi (A, x, n, i+1);
	nznak(' ', i*5); printf("%5d\n", ret);
	return ret;
}
// Rekurzivno trazenje indeksa clana u polju s ogranicivacem
int trazi1 (int A[], int x, int i){
	int ret;
	nznak(' ', i*5); printf("^^^^^\n");
	if(A[i] == x) ret = i;
	else ret= trazi1 (A, x, i+1);
	nznak(' ', i*5); printf("%5d\n", ret);
	return ret;
}

// Drugi rekurzivni nacin bez ogranicivaca
int trazi2 (int A[], int x, int n){
    if (n < 1) return 0;    // Ako element ne postoji, vratiti će indeks n
    if (A[0] == x) return 0;
    return 1 + trazi2(&A[1], x , n-1);
}


// Rekurzivno trazenje najveceg clana polja
int maxclan (int A[], int i, int n) {
	int imax;
	if (i >= n-1) return n-1;
	imax = maxclan (A, i + 1, n);
	if (A[i] > A[imax]) return i;
	return imax;
}

// Rekurzivno trazenje najveceg clana polja - strukturirano
int maxclan1 (int A[], int i, int n) {
	int imax, ret;
	printf ("max(%d) -> ", i);
	if (i >= n-1) {
		printf ("\n");
		ret = n-1;
	} else {
		imax = maxclan1 (A, i + 1, n);
		if (A[i] > A[imax])
			ret = i;
		else
			ret = imax;
	}
	printf ("<- max(%d)=%d ", i, ret);
	return ret;
}
// macro naredba za vecu od dvije vrijednosti
#define maxof(a,b) ((a) > (b) ? (a) : (b))

// Funkcija s macro naredbom
// koja vraca vrijednost najveceg clana
int maxclan2 (int A[], int i, int n) {
	int m;
  if (i >= n-1) return A[i];
	m = maxclan2 (A, i + 1, n);
  return maxof(A[i], m);
```

```
}
// Primjer neispravne rekurzije
int los (int n, int *dubina) {
	int r;
	(*dubina)++;
	printf ("n = %d, dubina rekurzije = %d\n", n, *dubina);
	if (n == 0)
		r = 0;
	else
		r = los (n / 3 + 1, dubina) + n - 1;
  return r;
}
int main () {
	int A[MAXA], x, i, n, dubina;
	FILE *fi;
	fi = fopen ("UlazZaTrazenje.txt", "r");
	if (!fi) return 1;
	n = 0;
	while (n < MAXA - 1 && fscanf (fi, "%d", &A[n]) != EOF)	n++;
	fclose (fi);
	ispisi (A, n);
	printf ("Upisite vrijednost za x =");
	scanf ("%d", &x);
	printf ("\nRekurzivno trazenje indeksa clana\n");
	ispisi (A, n);
	if ((i = trazi (A, x, n, 0)) < 0)
		printf ("Vrijednost %d ne postoji u polju\n", x);
	else
		printf ("A [%d] = %d\n", i, A [i]);
	printf ("\nRekurzivno trazenje ... s ogranicivacem\n");
	A [n] = x;  // postavljanje ogranicivaca
	ispisi (A, n+1);
	if ((i = trazi1 (A, x, 0)) == n)
		printf ("Vrijednost %d ne postoji u polju", x);
	else
		printf ("A [%d] = %d\n", i, A [i]);

    printf("\nTraženje na treci nacin:\n");
    if ((i = trazi2(A, x, n)) == n) {
        printf ("Vrijednost %d ne postoji u polju", x);
    }
    else {
        printf ("A [%d] = %d\n", i, A [i]);
    }

	printf ("\nRekurzivno trazenje najveceg...\n");
	ispisi(A, n);
	if ((i = maxclan (A, 0, n)) != maxclan1 (A, 0, n)) {
		printf ("Pogreska: Strukturirana i nestrukturirana funkcija daju razlicite
rezultate!\n");
		return 1;
	}
	printf ("\nNajveci clan A [%d] = %d\n",i, A [i]);
	printf ("Funkcija s macro naredbom je nasla najveci clan %d\n", maxclan2 (A, 0, n));
	printf ("\nPozivam neispravnu rekurziju\n");
	while (1) {
		dubina = 0;
		printf ("Upisite vrijednost za n =");    // primjer: n=4
		scanf ("%d", &n);
		i = los (n, &dubina);
		printf ("\ni = %d", i);
	}
}
```

Kamate.c

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

float kamrac (float g, int n, float p) {
// g – glavnica
```

```
// n – trajanje oročenja u godinama
// p – kamatna stopa u postotcima
   if (n <= 0) return g;
   else return (1 + p / 100) * kamrac(g, n - 1, p);
}

// drugi nacin
float kamrac2 (float g, int n, float p) {
    if (n<=0) return g;
    else return kamrac2(g*(1+p/100), n-1, p);
}

int main () {
       float g, p, k1, k2;
       int n;
       while (1) {
              printf ("\nUpisite iznos glavnice, broj godina orocenja i kamatnu stopu>");
              scanf ("%f %d %f", &g, &n, &p);
              if (n < 0) break;
              k1 = kamrac (g, n, p);
    k2 = kamrac2 (g, n, p);
              printf ("\nGlavnica %10.2f orocena na %d godina uz kamatnu stopu %5.2f%% rezultira
iznosom %10.2f",
                                  g, n, p, k1);
    printf("\ndrugi nacin %10.2f", k2);
       }
       return 0;
}
```

Premetaljka.c

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

void rotiraj(size_t duljina, char *niz) {
  char sacuvaj;
  sacuvaj = *niz;
  while(--duljina) {
    *niz=*(niz+1);
    ++niz;
    }
  *niz = sacuvaj;
}

void permutiraj(size_t duljina, char *niz, unsigned dubina) {
  if (duljina == 0) printf("\n# %s\t",niz-dubina);
  else {
    size_t brojac;
    for (brojac = duljina ; brojac > 0; --brojac) {
    printf("%s ", niz);
      permutiraj(duljina-1,niz+1,dubina+1);
    printf("%s ", niz);
      rotiraj(duljina,niz);
    printf("%s ", niz);
    }
  }
}

int main() {
  char izvorno[30];
  printf("Upisite rijec:\n");
  gets(izvorno);
  printf("\nPermutiram rijec \"%s\"\n",izvorno);
  permutiraj(strlen(izvorno),izvorno,0);
  return EXIT_SUCCESS;
}
```

Obrtaljka.c

```
-------------------------------------------------------------------------
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define DULJINA_NIZA 120

void u_malo_slovo (char *niz) {
     char *kraj = niz;
     // ako je veliko slovo, pretvori ga u malo!
     while (*kraj) {
               if (((*kraj) >= 'A') && ((*kraj) <= 'Z'))
                           (*kraj) += 'a' - 'A';
               kraj++;
        }
}

int provjeri(char *niz) {
     char *kraj = niz + strlen (niz) - 1;
     // s pocetka ukloniti sve sto nije slovo
     while (*niz < 'a' || *niz > 'z') niz++;
     while (*kraj < 'a' || *kraj > 'z') kraj--;

     // osnovni slucaj: ako je niz dužine 0 ili 1 - kraj, palindrom je
     if ( kraj - niz < 1) return 1;

     // ako se znakovi na pocetku i kraju ne podudaraju, izlazi van, nije palindrom
     if ( *niz != *kraj) return 0;

     // novi niz je niz sa svake strane kraci za po jedan znak
     niz++;
     *kraj = 0;
     return provjeri(niz);
}

int main (void) {
    char izvor[DULJINA_NIZA];
       while (1) {
              printf("Upisite rijec ili recenicu:\n");
              gets(izvor);
              printf("\nGledam je li palindrom: \"%s\"\n", izvor);
              u_malo_slovo (izvor);
              provjeri (izvor) > 0 ? printf("Palindrom je!\n") : printf("Nije palindrom!\n");
       }
}
```

Hanoi.c

```c
#include <stdio.h>
void hanoi(char Izvor, char Odrediste, char Pomocni, int n) {
   if (n > 0) {
      hanoi (Izvor, Pomocni, Odrediste, n - 1);
      printf("\nPrebacujem element %d s tornja %c na toranj %c",
                                                          n, Izvor, Odrediste);
      hanoi (Pomocni, Odrediste, Izvor, n - 1);
   }
}
int main() {
  int n;
  while (1) {
              printf ("\n\nUpisite broj diskova>");
              scanf ("%d", &n);
              if (n <= 0) break;
              printf("\nHanojski tornjevi (%d elementa)", n);
              hanoi('I', 'O', 'P', n);
       }
  return 0;
}
```

Kraljice.c

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

```c
// ispis znaka c u zadanoj duljini n
void nznak (int c, int n) {
  while (--n >= 0) putchar(c);
}
int NeNapadaju(int x1, int y1, int x2, int y2) {
       // funkcija koja utvrdjuje da li se dvije kraljice,
       // postavljene na polja (x1, y1) i (x2, y2)
       // medjusobno ne napadaju
       int ne_napadaju = 1;
       if (x1 == x2) ne_napadaju = 0;
       if (y1 == y2) ne_napadaju = 0;
       if (x1 + y2 == x2 + y1) ne_napadaju = 0;
       if (x1 - y2 == x2 - y1) ne_napadaju = 0;
       return ne_napadaju;
}

int K8(int *k, int i, int n) {
  int a, b;
  int dobar;
  if (i == n) return 1; // rubni uvjet
  nznak(' ', i); printf ("Kraljica=%d\n", i+1);
  for (a = 0; a < n; a++) { // potencijalni retci
    dobar = 1;
    for (b = 0; b < i; b++) { // prethodne i-tu u a-tom
      if (!NeNapadaju(b + 1, k[b] + 1, i + 1, a + 1)) {
        dobar = 0;
        break;
      }
    }
    if (dobar) {
      k[i] = a; // redak i-te dobre
      nznak(' ', i); printf ("?? (%d %d)\n", i+1, a+1);
      if (K8(k,i+1,n) == 1) {
        nznak(' ', i); printf ("OK (%d %d)\n", i+1, a+1);
        return 1; // dobar do kraja
      } else {
        nznak(' ', i); printf ("-- (%d %d)\n", i+1, a+1);
      }
    }
  }
  nznak(' ', i); printf ("<< (%d, *)\n", i+1);
  return 0;
}
int main() {
  int k[8] = {0}, i;

  K8(k, 0, 8);
  for (i = 0; i < 8; i++)
    printf("(%d, %d)\n", i + 1, k[i] + 1);

  return 0;
}
```

Konj.c

```c
#include <stdio.h>
#include <conio.h>
#define MAXR 30
#define MAXS 30

int ploca[MAXR][MAXS] = {0};

int moze(int maxr, int maxs, int *tr, int *ts, int potez){
       int nr, ns;
       switch (potez){
              case 0: nr = *tr -2; ns = *ts - 1; break;
              case 1: nr = *tr -2; ns = *ts + 1; break;
              case 2: nr = *tr -1; ns = *ts + 2; break;
              case 3: nr = *tr +1; ns = *ts + 2; break;
```

```
                case 4: nr = *tr +2; ns = *ts + 1; break;
                case 5: nr = *tr +2; ns = *ts - 1; break;
                case 6: nr = *tr +1; ns = *ts - 2; break;
                case 7: nr = *tr -1; ns = *ts - 2; break;
        }
        if ( nr>=0 && nr<maxr && ns>=0 && ns<maxs && !ploca[nr][ns]){
                *tr = nr;
                *ts = ns;
                return 1;
        }
        return 0;
}
int fBrojOpcija(int maxr, int maxs, int tr, int ts){
        int rv=0;
        int nr, ns, potez;
        for (potez=0; potez<8; potez++){
        switch (potez){
                case 0: nr = tr -2; ns = ts - 1; break;
                case 1: nr = tr -2; ns = ts + 1; break;
                case 2: nr = tr -1; ns = ts + 2; break;
                case 3: nr = tr +1; ns = ts + 2; break;
                case 4: nr = tr +2; ns = ts + 1; break;
                case 5: nr = tr +2; ns = ts - 1; break;
                case 6: nr = tr +1; ns = ts - 2; break;
                case 7: nr = tr -1; ns = ts - 2; break;
        }
        if ( nr>=0 && nr<maxr && ns>=0 && ns<maxs && !ploca[nr][ns]){
                rv++;
        }
    }
        return rv;
}

/*
A technique known as  Warnsdorf's heuristic allows us to make much better choices for next move than
random selection.  The heuristic, discovered by H. C. von Warnsdorf in 1823 tells to select as our
next move  the one which has the fewest choices for moving on from there.*/
int WarnsdorfovKonj(int maxr, int maxs, int tr, int ts, int rbr){
        int potez, tr1, ts1, brojOpcija[8] = {0};
        //printf("\npokusavam %d %d %d",rbr, tr, ts);
    int min, i;
        ploca[tr][ts] = rbr;
        if (rbr == maxr * maxs){
                return 1;
        }
        for (potez = 0; potez < 8; potez++){
                tr1 = tr; ts1 = ts;
                if (moze(maxr, maxs, &tr1, &ts1, potez)){
                        brojOpcija[potez] = fBrojOpcija(maxr, maxs, tr1, ts1);
      }else{
         brojOpcija[potez] = 200;
      }
       }


   while (1){
     for (i = 0, min = 100; i < 8; i++){
        if (brojOpcija[i]<min){
           min = brojOpcija[i];
           potez = i;
        }
     }
   if (min==100) break;
   brojOpcija[potez] = 200;

   tr1 = tr; ts1 = ts;
   if (moze(maxr, maxs, &tr1, &ts1, potez)){
          if (WarnsdorfovKonj(maxr, maxs, tr1, ts1, rbr+1) == 1){
                return 1;
          }
```

```
   }


   }
       ploca[tr][ts] = 0;
       return 0;
}


int konj(int maxr, int maxs, int tr, int ts, int rbr){
       int potez, tr1, ts1;
       //printf("\npokusavam %d %d %d",rbr, tr, ts);
       ploca[tr][ts] = rbr;
       if (rbr == maxr * maxs){
              return 1;
       }
       for (potez = 0; potez < 8; potez++){
              tr1 = tr; ts1 = ts;
              if (moze(maxr, maxs, &tr1, &ts1, potez)){
                     if (konj(maxr, maxs, tr1, ts1, rbr+1) == 1){
                            return 1;
                     }
              }
       }
       ploca[tr][ts] = 0;
       return 0;
}

int main(){
       int r, s, i, j;
   printf("\nUpisite broj redaka i stupaca:");
       scanf("%d %d", &r, &s);




   for(i=0;i<MAXR;i++)
      for(j=0;j<MAXS;j++)
         ploca[i][j]=0;

   printf("\nWarnsdorfov konj...");
   if (WarnsdorfovKonj(r, s, 0, 0, 1) == 1){
      printf("\n\n\n");
              for (i=0; i<r; i++){
                     printf("\n");
                     for (j=0; j<s; j++){
                            printf(" %3d", ploca[i][j]);
                     }
              }
       }else{
              printf("\nNe moze :(\n");
       }


   for(i=0;i<MAXR;i++)
      for(j=0;j<MAXS;j++)
         ploca[i][j]=0;

   printf("\nObicni konj ('ne moze' preko 6x6)...");
       if (konj(r, s, 0, 0, 1) == 1){
      printf("\n\n\n");
              for (i=0; i<r; i++){
                     printf("\n");
                     for (j=0; j<s; j++){
                            printf(" %3d", ploca[i][j]);
                     }
              }
       }else{
              printf("\nNe moze :(\n");
       }
```

```
 system("PAUSE");
	return 0;
}
```

RazneSlozenosti.c

```
// RazneSlozenosti.c
// Kubni, kvadraticni, NlogN ili linearni algoritam
#include <stdio.h>
#include <stdlib.h>

// vraca niz znakova c u zadanoj duljini n
char* nc (int c, int n) {
	static char s[80+1];
	s[n] = '\0';                                                    // prirubi
  while (--n >= 0) s[n] = c;      // popuni
	return s;
}
// ispis polja
void ispisi(int A[], int n) {
	int i;
	printf("\n");
	for (i = 0; i < n; i++) printf(" A[%d]",i);
	printf("\n");
	for (i = 0; i < n; i++) printf("%5d", A[i]);
	printf("\n");
}


// Kubna slozenost
int MaxPodSumaNiza3 (int A[], int N) {
	int OvaSuma, MaxSuma, i, j, k;
	int iteracija = 0;
	MaxSuma = 0;
	for (i = 0; i < N; i++) {
		printf ("i=%d\n", i);
		for (j = i; j < N; j++) {
			OvaSuma = 0;
			for (k = i; k <= j; k++) {
				OvaSuma += A [k];
				++iteracija;
			}
			if (OvaSuma > MaxSuma) MaxSuma = OvaSuma;
			printf ("Suma clanova [%d, %d] = %d, a najveca = %d\n",
                                                                    i, j, OvaSuma, MaxSuma);
		}
	}
	printf ("Broj iteracija: %d\n", iteracija);
	return MaxSuma;
}

// Kvadratna slozenost
int MaxPodSumaNiza2 (int A[ ], int N) {
	int OvaSuma, MaxSuma, i, j;
	int iteracija = 0;
	MaxSuma = 0;
	for (i = 0; i < N; i++) {
		printf ("i=%d\n", i);
		OvaSuma = 0;
		for (j = i; j < N; j++) {
			OvaSuma += A[ j ];
			++iteracija;
			if (OvaSuma > MaxSuma) MaxSuma = OvaSuma;
			printf ("Suma clanova [%d, %d] = %d, a najveca = %d\n",
                                                                    i, j, OvaSuma, MaxSuma);
		}
	}
	printf ("Broj iteracija: %d\n", iteracija);
	return MaxSuma;
}
```

```
// NlogN slozenost - koristi funkcije max3 i MaxPodSuma
// racuna najveci od 3 broja
int Max3 (int A, int B, int C) {
	return A > B ? A > C ? A : C : B > C ? B : C;
}
// trazi najvecu podsumu clanova od Lijeva do Desna
int MaxPodSuma (int A[], int Lijeva, int Desna, int dubina) {
	int MaxLijevaSuma, MaxDesnaSuma;
	int MaxLijevaRubnaSuma, MaxDesnaRubnaSuma;
	int LijevaRubnaSuma, DesnaRubnaSuma;
	int Sredina, i, ret;
	printf ("%s> MaxPodSuma(%d, %d) ...\n",
					nc(' ', dubina*2), Lijeva, Desna);
	if (Lijeva == Desna) { // Osnovni slucaj
	  if (A [Lijeva] > 0)
		ret = A [Lijeva]; // podniz od clana A[Lijeva]
	  else
		ret = 0; // suma je 0 ako su svi brojevi negativni
    printf ("%s< MaxPodSuma(%d, %d) = %d\n",
				nc(' ', dubina*2), Lijeva, Desna, ret);
	return ret;
	}
	// racun lijeve i desne podsume s obzirom na Sredina
	Sredina = (Lijeva + Desna) / 2;
	MaxLijevaSuma = MaxPodSuma (A, Lijeva, Sredina, dubina+1);
	MaxDesnaSuma = MaxPodSuma (A,Sredina + 1, Desna, dubina+1);
	// najveca gledano ulijevo od sredine
	MaxLijevaRubnaSuma = 0; LijevaRubnaSuma = 0;
	for (i = Sredina; i >= Lijeva; i--) {
		LijevaRubnaSuma += A [i];
		if (LijevaRubnaSuma > MaxLijevaRubnaSuma)
			MaxLijevaRubnaSuma = LijevaRubnaSuma;
	}
	// najveca gledano udesno od sredine
	MaxDesnaRubnaSuma = 0; DesnaRubnaSuma = 0;
	for (i = Sredina + 1; i <= Desna; i++) {
		DesnaRubnaSuma += A [i];
		if (DesnaRubnaSuma > MaxDesnaRubnaSuma)
		MaxDesnaRubnaSuma = DesnaRubnaSuma;
	}
	printf ("%s  Lijeva=%d Desna=%d Rubna=%d\n",
				nc (' ', dubina*2), MaxLijevaSuma, MaxDesnaSuma,
				MaxLijevaRubnaSuma + MaxDesnaRubnaSuma);
	// najveca od lijeva, desna, rubna
	ret = Max3 (MaxLijevaSuma, MaxDesnaSuma,
			MaxLijevaRubnaSuma + MaxDesnaRubnaSuma);
	printf ("%s< MaxPodSuma(%d, %d) = %d\n",
				nc(' ', dubina*2), Lijeva, Desna, ret);
	return ret;
}
// NlogN slozenost
int MaxPodSumaNizaLog (int A [], int N) {
	return MaxPodSuma (A, 0, N - 1, 0);
}
// Linearna složenost
int	MaxPodSumaNiza1 (int A[], int N) {
	int OvaSuma, MaxSuma, j;
	OvaSuma = MaxSuma = 0;
	for (j = 0; j < N; j++) {
		OvaSuma += A[ j ];
		if (OvaSuma > MaxSuma) MaxSuma = OvaSuma;
		else if (OvaSuma < 0) OvaSuma = 0;
		printf ("j=%d OvaSuma=%2d MaxSuma=%2d\n",
				j, OvaSuma, MaxSuma);
	}
	return MaxSuma;
}

int main (void) {
	int A [] = {4, -3, 5, -2, -1, 2, 6, -2};
```

```
	int rez;

	printf("\n\nKubna slozenost\n");
	ispisi(A, sizeof (A) / sizeof (A [0]));
	rez = MaxPodSumaNiza3 (A, sizeof (A) / sizeof (A [0]));
	printf("\nMaxSuma3 = %d", rez);

	printf("\n\nKvadratna slozenost\n");
	ispisi(A, sizeof (A) / sizeof (A [0]));
	rez = MaxPodSumaNiza2 (A, sizeof (A) / sizeof (A [0]));
	printf("\nMaxSuma2 = %d", rez);

	printf("\n\nLogaritamska slozenost\n");
	ispisi(A, sizeof (A) / sizeof (A [0]));
	rez = MaxPodSumaNizaLog (A, sizeof (A) / sizeof (A [0]));
	printf("\nMaxSumaLog = %d", rez);

	printf("\n\nLinearna slozenost\n");
	ispisi(A, sizeof (A) / sizeof (A [0]));
	rez = MaxPodSumaNiza1 (A, sizeof (A) / sizeof (A [0]));
	printf("\nMaxSuma1 = %d\n", rez);

	return 0;
}
```

ModPolja.c

```
// ModPolja.c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys\timeb.h>
#define MAXA 1000

// izravno pronalazi mod i ucestalost u a[n]
int mode0 (int a[], int n, int *f) {
  int mode, i, temp;// mod, trenutni, privremeni
  mode = a[0]; *f = 1; temp = 1;// prvi je mod, frekvencije 1
  for (i = 1; i < n; i++) {       // provjera ostalih
    if (a[i] != a[i - 1]) {       // trenutni razlicit od prethodnog
      temp = 1;                                 // nadjen je novi element
    } else {
      temp++;                           // povecaj ucestalost novog
      if (temp > *f) {                  // da li je trenutni novi mod ?
        *f = temp; mode = a[i];   // zapamti mod i ucestalost
      }
    }
  }
  return mode;      // vrati mod
      // frekvencija se vraca kroz *f
}
// rekurzivno pronalazi mod i ucestalost u a[0:i]
int rmode0 (int a[], int i, int *f) {
  int mode;
  if (i == 0) {                                 // osnovni slucaj
    mode = a[0]; *f = 1;          // prvi je mod, frekvencije 1
  } else {
    mode = rmode0 (a, i - 1, f);  // rekurzivni mod svih prethodnika
    if (a[i] == a[i - *f]) {            // da li trenutnome prethodi *f jednakih?
                    // novi mod, ili stari mod, ali s uvećanom učestalošću
      mode = a[i]; (*f)++;              // zapamti mod i ucestalost
    }
  }
  return mode;      // vrati mod
}
// rekurzivni postupak transformiran u iterativni
int rmode1(int a[], int n, int *f) {
  int mode, i;
  mode = a[0]; *f = 1;                                  // prvi je mod, frekvencije 1
  for (i = 1; i < n; i++) {
```

```
    if (a[i] == a[i - *f]) {                    // da li trenutnome prethodi *f jednakih?
      mode = a[i]; (*f)++;                      // zapamti mod i ucestalost
    }
  }
  return mode;       // vrati mod
}
// U sortiranom polju a pronalazi se mod i ucestalost.
int main (void) {
  int a[MAXA], n, m;// polje, broj clanova, najveci clan
  int i, j, pom;           // indeksi petlji, pomocna za sort
       int    broj, freq, p;       // broj ponavljanja, ucestalost, nadjeni mod
       struct timeb vrijeme1, vrijeme2; // poc. i zav. vrijeme
  long trajanje [3];// vremena izvodjenja u ms
       // unos parametara
       // 1. n= 10, m= 5, broj=1
       // 2. n=100, m=10, broj=100000
  do {
       printf ("Upisite broj clanova polja i maks. clan >");
       scanf ("%d %d", &n, &m);
  } while (n > MAXA);
  printf ("Upisite broj obavljanja programa >");
  scanf ("%d",&broj);
  printf("Izracuni ce se ponoviti %d puta\n", broj);
  // inicijalizacija generatora pseudoslucajnih brojeva
  srand ((unsigned) time (NULL));
  // popunjavanje polja
  for (i = 0; i < n; i++) {
       // slucajne vrijednosti skalirane na maks. clan
    a[i] = rand () % (m+1);
  }
  // sortiranje polja
  for (i = 0; i < n - 1; i++) {     // od prvog do predzadnjeg
    for (j = i; j < n; j++) {            // provjeri iza trenutnog
      if (a[i] > a[j]) {                        // onaj iza je manji
             // zamjena trenutnoga i manjega koji je iza
        pom = a[i]; a[i] = a[j]; a[j] = pom;
      }
    }
  }
  // ispis polja
  for (i = 0; i < n; i++) {
             printf ("%4d", a[i]);
  }
  // svaka od funkcija poziva se broj puta
  // mjeri se ukupno vrijeme izvrsenja za svaki algoritam
  // izravno
  ftime (&vrijeme1);
  for (i = 1; i <= broj; i++) {
    p = mode0 (a, n, &freq);
  }
  ftime (&vrijeme2);
  trajanje [0] = 1000 * (vrijeme2.time - vrijeme1.time) +
             vrijeme2.millitm - vrijeme1.millitm;

  // rekurzivno
  ftime (&vrijeme1);
  for (i = 1; i <= broj; i++) {
    p = rmode0 (a, n-1, &freq);
  }
  ftime (&vrijeme2);
  trajanje [1] = 1000 * (vrijeme2.time - vrijeme1.time) +
             vrijeme2.millitm - vrijeme1.millitm;

  // iterativna transformacija rekurzivnog
  ftime (&vrijeme1);
  for (i = 1; i <= broj; i++) {
    p = rmode1 (a, n, &freq);
  }
  ftime (&vrijeme2);
  trajanje [2] = 1000 * (vrijeme2.time - vrijeme1.time) +
```

```
            vrijeme2.millitm - vrijeme1.millitm;
  printf ("\nMod = %d, ucestalost = %3d\n", p, freq);
  printf ("\nBroj milisekundi za %d izvodjenja:\n"
                  " mode0: %d\nrmode0: %d\nrmode1: %d\n",
                                        broj, trajanje [0], trajanje [1], trajanje [2]);
  return 0;
}
```

PrimjeriRekurzije.c

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

void pisi1 (int broj, int n) {
  broj++;
  if (broj > n) return;
  pisi1 (broj, n);
  printf( "%d", broj);
}

void pisi2 (int broj, int n) {
  broj++;
  if (broj > n) return;
  printf( "%d", broj);
  pisi2 (broj, n);
}

void pisi3 (int *broj, int n) {
  (*broj)++;
  if (*broj > n) return;
  pisi3 (broj, n);
  printf( "%d", *broj);
}

void pisi4 (int *broj, int n) {
  (*broj)++;
  if (*broj > n) return;
  printf( "%d", *broj);
  pisi4 (broj, n);
}

void pisi5 (int *broj, int n) {
  (*broj)--;
  if (*broj < 0) return;
  pisi5 (broj, n);
  printf( "%d", *broj);
}

int main (void) {
  int nula;

       nula = 0; pisi1 (nula, 5);
       printf(" Nakon pisi1 nula = %d\n", nula);

       pisi2 (nula, 5);
       printf(" Nakon pisi2 nula = %d\n", nula);

  nula = 0; pisi3 (&nula, 5);
       printf(" Nakon pisi3 nula = %d\n", nula);

       nula = 0; pisi4 (&nula, 5);
       printf(" Nakon pisi4 nula = %d\n", nula);

       pisi5 (&nula, 5);
       printf(" Nakon pisi5 nula = %d\n", nula);

       return 0;
}
/*
a)  Kakav tip podatka sadrži *broj?
b)  Kakav tip podatka sadrži broj?
```

```
c)  Kakav tip podatka sadrži &nula?
d)  Koju vrijednost sadrži nula nakon povratka iz funkcije pisi1?
e)  Što ae program ispisati na zaslonu raeunala?
*/
```

BinomniKoeficijenti.c

```
--------------------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys\timeb.h>
#define MAXRED 100

// vraca niz znakova c u zadanoj duljini n
char* nc (int c, int n) {
       static char s[80+1];
       s[n] = '\0';                       // prirubi
  while (--n >= 0) s[n] = c;      // popuni
       return s;
}

// vraća faktorijela (n), broj iteracija, zastavicu pogreške
long FAKT (int n, long *freq, int *errorflag) {
  int i;
  long p;
  p = 1;
  for (i = 2; i <= n; i++) {
    p *= i;
              if (p <= 0) *errorflag = 1;
    *freq += 1;
  }
  return p;
}
// binomni koeficijenti s pomoću faktorijela
long BINOM (int n, int m, long *freq, int *errorflag) {
       long p;
  *freq += 1;
#if 1
       p = FAKT (n, freq, errorflag);
       p /= FAKT (m, freq, errorflag);
       p /= FAKT (n - m, freq, errorflag);
       return p;
#else
  return FAKT (n, freq, errorflag) /
                   FAKT (m, freq, errorflag) /
                            FAKT (n - m, freq, errorflag);
#endif
}

// binomni koeficijenti rekurzivno
long BINOMR (int n, int m, long *freq) {
  *freq += 1;
  if ((m == 0) || (m == n)) return 1;
  return BINOMR (n-1, m, freq) + BINOMR (n - 1, m - 1, freq);
}

// Pascalov trokut
void Blaise (int n) {
  int i, j;
  long stari[MAXRED], novi[MAXRED];
  if (n >= MAXRED) return;
       printf("\nIzracunavanje Pascalovog trokuta\n");
  novi[0] = 1;
  for (i = 0; i < n; i++) {
    novi[i+1] = 1;
    for (j = 1; j <= i; j++)
                novi[j] = stari[j-1] + stari[j];
         printf("%s", nc(' ', 2*(n-i)));
    for (j = 0; j <= i+1; j++) {
                     printf ("%3d ", novi[j]);
```

```
      if (novi[j] < 0) {
            printf ("\n za i=%d i j=%d broj postane prevelik\n", i, j);
            exit (1);
      }
      stari[j] = novi[j];
    }
    printf ("\n");
  }
}

int main (void) {
  int n, m, i, j;   // n povrh m, indeksi petlje
  int broj;                       // broj ponavljanja
  long k;                                 // pojedinacni rezultat
       int errorflag;       // zastavica pogreske
  float f[2][2];    // trajanje i broj iteracija
  long trajanje, freq;
  struct timeb vrijeme1, vrijeme2;

  while (1) {
             // citanje parametara
             printf ("Upisite broj obavljanja programa >");
    scanf ("%d",&broj); // npr: 1, 10000
    if (broj <= 0) {
      printf("Gotovo!\n");
      break;
    }
             do {
      printf ("Upisite n, m >"); // npr: 12 5, 13 5
      scanf ("%d %d", &n, &m);
    } while ((n < m) || (n < 0) || (m < 0) ||
                          ((m == 0) && (n == 0)));

    // inicijalizacija
         for (i = 0; i < 2; i++)
           for (j = 0; j < 2; j++)
                 f[i][j] = 0;
    printf ("Program ce se ponoviti %d puta\n", broj);
         errorflag = 0;
         // koristenjem faktorijela
         freq = 0;
    ftime (&vrijeme1);
    for (i = 1; i <= broj; i++)
              k = BINOM (n, m, &freq, &errorflag);
      ftime (&vrijeme2);
      trajanje = 1000 * (vrijeme2.time - vrijeme1.time) +
                    vrijeme2.millitm - vrijeme1.millitm;
      f[0][0] += trajanje;
      f[1][0] += freq;
      printf  (" BINOM : %d povrh %d = %ld %s\n",n, m, k,
                                                 errorflag ? "(pogresno)" : "");

         // rekurzivno
         freq = 0;
    ftime (&vrijeme1);
    for (i = 1; i <= broj; i++)
              k = BINOMR (n,m,&freq);
    ftime (&vrijeme2);
    trajanje = 1000 * (vrijeme2.time - vrijeme1.time) +
                   vrijeme2.millitm - vrijeme1.millitm;
    f[0][1] += trajanje;
    f[1][1] += freq;
    printf (    " BINOMR: %d povrh %d = %ld\n", n, m, k);

    // racun prosjecnih vremena i ispis rezultata
    for (i = 0; i < 2; i++) {
      f[0][i] = f[0][i] / (float) broj;
      f[1][i] = f[1][i] / (float) broj;
    }
    printf ("\nProsjecno vrijeme za %d izvodjenja:\n BINOM: %f\nBINOMR: %f\n",
```

```
                                                                broj, f[0][0], f[0][1]);
    printf ("\nBroj iteracija:\n BINOM: %ld BINOMR: %ld\n",
                                (long) f[1][0], (long) f[1][1]);
  }
       // Pascalov trokut
       while (1) {
              printf ("Unesite broj redaka Pascalovog trokuta >");
              scanf ("%d", &n);    // npr: 10
              if (n <= 0 || n >= MAXRED) return 0;
          Blaise (n);
       }
}
```

PascalovTrokutRekurzija.c

```
#include <stdio.h>


int p(int kat, int i){
       if (i==0 || i==(kat+1)) return 1;
       return p(kat-1, i-1) + p(kat-1, i);
}

void BlaisePascal(int kat){
       int i, k;
       for (k=0; k <= kat; k++){
              printf("\n");
              for (i=0; i<=k+1; i++)
                     printf(" %3d", p(k, i));
       }
}
int main (void) {
       int kat;
       scanf("%d", &kat);
       BlaisePascal(kat);
       system("PAUSE");
       return 0;
}
```

TSP.c

```
// Za razliku od TSPJednostavni, ovaj ispisuje i put koji daje najmanji trošak
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define MAXGRAD 15       // i to je previše s obzirom na n!
// c – matrica udaljenosti između gradova (i, j)
int c[MAXGRAD][MAXGRAD]; // da ne kompliciramo s prijenosom 2D polja

typedef struct s {
    int cijena;
    int put[MAXGRAD];   // moglo bi sa malloc, ali radi jednostavnosti
} zapis;

void minus (int skup[], int n, int element) {
// operacija (skup \ element)
       int i, j;
       for (i = 0; i < n; i ++)
              if (skup[i] == element) break;
       for (j = i; j < n - 1; j ++)
       skup[j] = skup[j + 1];
}


zapis TSP (int IzGrada, int *gradovi, int n) {
       int *lgradovi, i;
       zapis minTSP, pomTSP;
```

```
	lgradovi = malloc (n * sizeof(int));
	memcpy (lgradovi, gradovi, n * sizeof(int));
	minus (lgradovi, n, IzGrada);

	 -- n;
	if (n == 1) {
		minTSP.cijena = c[IzGrada][lgradovi[0]];
		minTSP.put[0] = lgradovi[0];
		minTSP.put[1] = IzGrada;
  } else {
		minTSP = TSP (lgradovi[0], lgradovi, n);
		minTSP.cijena += c[IzGrada][lgradovi[0]];
		minTSP.put[n] = IzGrada;
    for (i = 1; i < n; i++) {
		pomTSP = TSP (lgradovi[i], lgradovi, n);
		pomTSP.cijena += c[IzGrada][lgradovi[i]];
		pomTSP.put[n] = IzGrada;
		if (pomTSP.cijena < minTSP.cijena)
			minTSP = pomTSP;
		}
	}
	free (lgradovi);
	return minTSP;
}

int main () {
	int gradovi[MAXGRAD], n, i, j;
	zapis minTSP;
	srand (time(NULL));
	while (1) {
		do {
			printf ("\nUnesite broj gradova: ");
			scanf ("%d", &n);
		} while (n < 2 || n > MAXGRAD);
		// generiranje matrice
		for (i = 0; i < n; i++) {
			gradovi[i] = i;
			c[i][i] = 0;
			for (j = i + 1; j < n; j ++) {
#if 0
				c[i][j] = rand() + 1;
#else
				c[i][j] = i*10 + j;
#endif
				c[j][i] = c[i][j];
				printf ("c[%d][%d] = %d\n", i, j, c[i][j]);
			}
		}
		minTSP = TSP (0, gradovi, n);
		printf ("Najmanji trosak je: %d\n", minTSP.cijena);
		printf ("Put: ");
		for (i=n-1; i>=0; i--)
			printf("%d ", minTSP.put[i]);
	}
		return 0;
}
```

Sortovi.c

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

typedef int tip;

// vrijeme u ms
int Trajanje (struct timeb *vrijeme1) {
```

```
  struct timeb vrijeme2;
  ftime (&vrijeme2);
  return 1000 * (vrijeme2.time - vrijeme1->time) +
               vrijeme2.millitm - vrijeme1->millitm;
}

// ispis poruke i prekid programa
void Fatalno (char *niz) {
  printf ("\n %s \n", niz);
       exit (1);
}

// zamjena vrijednosti *lijevo i *desno
__inline void Zamijeni (tip *lijevo, tip *desno) {
  tip pom = *lijevo;
  *lijevo = *desno;
  *desno = pom;
}

// sort selekcijom
void SelectionSort (tip 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]);
       }
}

// mjehuricasti sort
void BubbleSort (tip A [], int N) {
       int i, j;
       for (i = 0; i < N-1; i++) {
    for (j = 0; j < N-1-i; j++) {
      if (A[j+1] < A[j]) Zamijeni (&A[j], &A[j+1]);
              }
       }
}

// mjehuricasti sort - poboljsani
void BubbleSortPoboljsani (tip 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;
                     }
              }
       }
}

// sort ubacivanjem (umetanjem)
void InsertionSort (tip A [], int N) {
  int i, j;
  tip 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;
  }
}

// Shell sort
void ShellSort (tip A [], int N) {
  int i, j, korak;
```

```
  tip pom;
  for (korak = N / 2; korak > 0; korak /= 2) {
             //printf("\nkorak=%d\n", korak);
    // Insertion sort s većim korakom
    for (i = korak; i < N; i++) {
                    //printf("\ni=%d:", i);
      pom = A [i];
      for (j = i; j >= korak && A[j-korak] > pom; j -= korak) {
                           //printf("%d:%d ", j, j-korak);
        A [j] = A [j - korak];
      }
      A [j] = pom;
    }
  }
}

// Heap sort - podesavanje gomile
void Podesi (tip A[], int i, int n) {
  int j;
  tip stavka;
  j = 2*i;
  stavka = A[i];
  while (j <= n ) {
    if ((j < n) && (A[j] < A[j+1])) j++;
    if (stavka >= A[j]) break;
    A[j/2] = A[j];
    j *=2;
  }
  A[j/2] = stavka;
}

// Heap sort - inicijalno stvaranje gomile
void StvoriGomilu (tip A[], int n) {
  int i;
  for (i = n/2; i >= 1; i--)
    Podesi (A, i, n);
}

// Heap sort
void HeapSort (tip A[], int n) {
  // A[1:n] sadrzi podatke koje treba sortirati
  int i;
  StvoriGomilu (A, n);
  for (i = n; i >= 2; i--) {
             // Zamijeni korijen i zadnji list, skrati polje za 1 i podesi gomilu
             Zamijeni (&A[1], &A[i]);
    Podesi (A, 1, i-1);
  }
}

// udruzivanje LPoz:LijeviKraj i DPoz:DesniKraj
void Merge (tip A [], tip 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)
    // Kopiraj ostatak prve polovice
    PomPolje [PomPoz++] = A [LPoz++];
  while (DPoz <= DesniKraj)
    // Kopiraj ostatak druge polovice
    PomPolje [PomPoz++] = A [DPoz++];
  for (i = 0; i < BrojClanova; i++, DesniKraj--)
```

```
    // Kopiraj PomPolje natrag
    A [DesniKraj] = PomPolje [DesniKraj];
}

// MergeSort - rekurzivno sortiranje podpolja
void MSort (tip A [], tip 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 (tip A [], int N) {
  tip *PomPolje;
  PomPolje = malloc (N * sizeof (tip));
  if (PomPolje != NULL) {
    MSort (A, PomPolje, 0, N - 1);
    free (PomPolje);
  } else
      Fatalno ("Nema mjesta za PomPolje!");
}

// QuickSort - medijan i stozer
// Vrati medijan od lijevo, sredina i desno,
// poredaj ih i sakrij stozer
tip medijan3 (tip 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]
  // Sakrij stozer
  Zamijeni (&A [sredina], &A [desno - 1]);
  // Vrati stozer
  return A [desno - 1];
}

// QuickSort - rekurzivno sortiranje podpolja
#define Cutoff (3)

void Qsort (tip A [], int lijevo, int desno) {
  int i, j;
  tip 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;
    }
    // Obnovi stozer
    Zamijeni (&A [i], &A [desno - 1]);
    Qsort (A, lijevo, i - 1);
    Qsort (A, i + 1, desno);
  } else {
    // Sortiraj podpolje
    InsertionSort (A + lijevo, desno - lijevo + 1);
  }
}
```

```
// QuickSort
void QuickSort (tip A [], int N) {
  Qsort (A, 0, N - 1);
}

// Quicksort, sožer je prvi element
void Qsort2(tip 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 (tip A [], int N) {
  Qsort2 (A, 0, N - 1);
}


// Testiranje sortova

// generira podatke za sort
void Generiraj (tip A [], int N) {
  int i;
  srand ((unsigned) time (NULL));
       // vrijednosti elemenata kao vrijednosti njihovih indeksa
  for( i = 0; i < N; i++ ) A [i] = i;
       // promijesaj vrijednosti
  for( i = 1; i < N; i++ )
    Zamijeni (&A [i], &A [rand () % (i + 1)]);
}

// provjeri da li svi elementi imaju vrijednost jednaku indeksu
void ProvjeriSort (tip A [], int N) {
  int i, flag = 0;
  for (i = 0; i < N; i++) {
    if (A[i] != i) {
      printf( "Sort ne radi: %d %d\n", i, A [i]);
           flag = 1;
         }
  }
  if (!flag) printf( "Provjera zavrsena: sort OK\n" );
}

// kopira polje desno[] u polje lijevo[]
void Kopiraj (tip lijevo [], tip desno [], int N) {
  int i;
  for (i = 0; i < N; i++) lijevo [i] = desno [i];
```

```
}

// ispis polja
void ispisi(int A[], int n) {
	int i;
	printf("\n");
	for (i = 0; i < n; i++) printf(" A[%d]",i);
	printf("\n");
	for (i = 0; i < n; i++) printf("%5d", A[i]);
	printf("\n");
}

// pokretanje potprograma za sort
void TestSorta (tip A[], tip B[], int N, char *ImeSorta, void (*Sort) (tip A[], int N)) {
	// A - polje koje se sortira
	// B - polje s podacima za sort
	// N - broj clanova polja
	// ImeSorta - naziv algoritma
	// Sort - pokazivac na funkciju koja obavlja sort
	struct timeb Vrijeme1;
	// kopiraj podatke iz B u A
	Kopiraj (A, B, N);
	// sortiraj i mjeri vrijeme
	printf ("%s...\n", ImeSorta);
	ftime (&Vrijeme1);
	if (strcmp(ImeSorta, "Heap Sort") == 0) {
		Sort (A-1, N); // da HeapSort "vidi" A[0] kao A[1]
	} else {
		Sort (A, N);  // standardni poziv
	}
	printf ("Trajanje: %d ms\n", Trajanje(&Vrijeme1));
	ProvjeriSort (A, N);

	// sortiraj prethodno sortirano polje A
	printf ("%s sortiranog polja...\n", ImeSorta);
	ftime (&Vrijeme1);
	if (strcmp(ImeSorta, "Heap Sort") == 0) {
		Sort (A-1, N);
	} else {
		Sort (A, N);
	}
	printf ("Trajanje: %d ms\n", Trajanje(&Vrijeme1));
	ProvjeriSort (A, N);
	printf ("Pritisni bilo koju tipku...\n\n");
	getch();
}

int main () {
#if 1
	int *Polje1, *Polje2, Duljina;
	// inicijalizacija
	printf ("Unesi broj clanova polja >");
	scanf ("%d", &Duljina);
	Polje1 = (int *) malloc (Duljina * sizeof (int));
	Polje2 = (int *) malloc (Duljina * sizeof (int));
	if (!Polje1 || !Polje2) Fatalno ("Nema dovoljno memorije!");
	// generiranje podataka
	Generiraj (Polje2, Duljina);
#else
  int Polje1[] = { 2, 6, 4, 5, 3, 7, 1, 0 };
  int Polje2[] = { 2, 6, 4, 5, 3, 7, 1, 0 };
  int Duljina;
  Duljina = sizeof(Polje1) / sizeof(Polje1[0]);
#endif
	// sortiranje
	TestSorta (Polje1, Polje2, Duljina, "Selection Sort", SelectionSort);
	TestSorta (Polje1, Polje2, Duljina, "Bubble Sort", BubbleSort);
	TestSorta (Polje1, Polje2, Duljina, "Bubble Sort poboljsani", BubbleSortPoboljsani);
	TestSorta (Polje1, Polje2, Duljina, "Insertion Sort", InsertionSort);
	TestSorta (Polje1, Polje2, Duljina, "Shell Sort", ShellSort);
```

```
	TestSorta (Polje1, Polje2, Duljina, "Heap Sort", HeapSort);
	TestSorta (Polje1, Polje2, Duljina, "Merge Sort", MergeSort);
	TestSorta (Polje1, Polje2, Duljina, "Quick Sort", QuickSort);
	TestSorta (Polje1, Polje2, Duljina, "Quick Sort 2", QuickSort2);
	return 0;
}
```

UpariDatoteke.c

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

void MergeF (FILE *f1, FILE *f2, FILE *fsort){
	char buf1[4096], buf2[4096], *pb1, *pb2;
	pb1 = fgets(buf1, 4096, f1);
	pb2 = fgets(buf2, 4096, f2);
	while (pb1 || pb2) {
		/* ako u obje datoteke još ima zapisa i
		   zapis iz prve datoteke manji je od zapisa iz druge ili
		   u prvoj datoteci još ima, a u drugoj više nema zapisa */
		if ((pb1 && pb2 && strcmp(pb1, pb2) <= 0) || (pb1 && !pb2)) {
			fputs(pb1, fsort);
			pb1 = fgets(buf1, 4096, f1);
		}
		/* ako u obje datoteke još ima zapisa i
		   zapis iz prve datoteke veći je od zapisa iz druge ili
		   u prvoj datoteci nema, a u drugoj još ima zapisa */
		if ((pb1 && pb2 && strcmp(pb1, pb2) > 0) || (!pb1 && pb2)) {
			fputs(pb2, fsort);
			pb2 = fgets(buf2, 4096, f2);
		}

	}
}
int main() {
	FILE *f1,*f2,*fsort;
	if ((f1 = fopen ("dat1.txt", "r"))==NULL){
		printf("Pogreska kod otvaranja datoteke dat1.txt");
		exit (1);
	}
	if ((f2 = fopen ("dat2.txt", "r"))==NULL){
		printf("Pogreska kod otvaranja datoteke dat2.txt");
		return 1;
	}
	if ((fsort = fopen ("sort.txt", "w"))==NULL){
		printf("Pogreska kod otvaranja datoteke sort.txt");
		return 1;
	}
   MergeF(f1, f2, fsort);
   fclose(f1); fclose(f2); fclose(fsort);
   return 0;
}

----------------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
/***** IMPLEMENTACIJA STOGA POLJEM **************/
#define MAXSTOG 5 /* maksimalna velicina stoga */

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

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

int dodaj(int stavka, Stog *stog){
```

```
	if (stog->vrh >= MAXSTOG-1) return 0;   /* dosegnut kapacitet stoga */
	stog->vrh++;
	stog->polje[stog->vrh] = stavka;
	return 1;
}


int skini(int *stavka, Stog *stog){
	if (stog->vrh < 0) return 0;
	*stavka = stog->polje[stog->vrh];
	stog->vrh--;
	return 1;
}
/* nije nuzna za funkcioniranje stoga, vec samo sluzi za provjeru ispravnosti rada programa*/
void ispis_stoga(Stog *stog){
	int i;
	if (stog->vrh == -1) {
			printf ("(prazan stog)");
	}
	else {
		printf ("Stog:");
		for (i=0; i <= stog->vrh; ++i)
			printf (" %d", stog->polje[i]);
	}
}

/************* KRAJ IMPLEMENTACIJE STOGA *****************/

int main () {
	int novi, stari;
	Stog stog;
	init_stog(&stog);
	printf ("Slucajno se generiraju nenegativni cijeli brojevi.\n");
	printf ("Neparni brojevi upisuju se na stog\n");
	printf ("Parni broj simulira skidanje sa stoga\n");
	printf ("Za obavljanje jednog koraka pritisnuti ENTER, za kraj CTRL-C\n\n");
	/* Inicijalizacija generatora pseudoslucajnih brojeva
	   na temelju sistemskog vremena */
	srand ((unsigned) time (NULL));
	while (1) {
		ispis_stoga(&stog);
		putchar ('\n');
		getchar ();
		novi = rand ();
		if (novi%2) {       /* Neparni se upisuju na stog */
			printf ("Dodaj %d\n", novi);
			if (!dodaj (novi, &stog))
				printf("Stog je pun!\n");
	} else {                    /* Parni broj simulira skidanje sa stoga */
			printf ("Skini...");
			if (skini (&stari, &stog))
				printf ("Skinut %d\n", stari);
			else
				printf("Stog je prazan!\n");
		}
	}
	return 0;
}
```

StogListom.c

```
#include <stdio.h>
#include <stdlib.h>
/***** IMPLEMENTACIJA STOGA LISTOM **************/

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

```c
typedef struct at atom;

typedef struct{
	atom *vrh;
} Stog;

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

int dodaj (tip element, Stog *stog) {
	atom *novi; // pokazivac na novi atom
	if ((novi = (atom *) malloc(sizeof(atom))) != NULL) {
		novi->element = element;
		novi->sljed = stog->vrh;
		printf("Na adresu %p dodao sam %d, a sljedeci je %p\n",
                                                                    novi, element,
stog->vrh);
		stog->vrh = novi;
		return 1;
	}
	else
		return 0;
}

int skini (tip *element, Stog *stog) {
	atom *pom;
    if (stog->vrh == NULL) return 0;
	*element = stog->vrh->element;
	printf ("\t Skidam s adrese %p ", stog->vrh);
	pom = stog->vrh->sljed; /* adresa novog vrha vrha */
	free(stog->vrh);           /* obriši stari vrh */
	stog->vrh = pom; /* postavi novi vrh */
	return 1;
}

/************* KRAJ IMPLEMENTACIJE STOGA ******************/

int  main () {
	FILE *fi;           /* ulazna datoteka */
	int j;              /* brojac ulaznih podataka */
	tip element;  /* element stoga */
	Stog stog;
	init_stog(&stog);
	/* Upis podataka na stog */
	fi = fopen ("UlazZaStogListom.txt", "r");
	if (fi) {
		/* inicijalizacija */
		j = 0;
		/* citanje podataka i stavljanje na stog */
		while (fscanf (fi, "%d", &element) == 1) {
			printf ("%d. ulazni podatak je %d \n\t", ++j, element);
			if (!dodaj (element, &stog)) {
				printf("Nema vise mjesta za stog!!\n");
				break;
			}
		}
		fclose (fi);

		/* Skidanje elemenata sa stoga */

		printf("\nSkidanje elemenata sa stoga: \n");
		while (skini(&element, &stog)) {
			printf ("%d\n", element);
		}
	} else {
		printf ("Nema ulazne datoteke\n");
		return 1;
	}
```

```
	return 0;
}
```

RedPoljem.c

```
-----------------------------------------------------------------------
#include <stdlib.h>
#include <stdio.h>
/***** IMPLEMENTACIJA REDA POLJEM **************/
#define MAXRED 10
typedef int tip;

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

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

// dodaje element u red
// vraca 1 ako ima mjesta u redu, inace 0
// mijenja ulaz, tj straznji kraj
int DodajURed (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;
}
// logicki uklanja element iz polja red od max n clanova
// vraca 1 ako ima clanova u redu, inace 0
// mijenja izlaz, tj prednji kraj
int SkiniIzReda (tip *element, Red *red) {
	if (red->ulaz == red->izlaz) return 0;
	red->izlaz++;
	red->izlaz %= MAXRED;
	*element = red->polje[red->izlaz];
	return 1;
}
// vraca broj elemenata u redu
int prebroji (Red *red) {
	if (red->ulaz >= red->izlaz) {
		return (red->ulaz - red->izlaz);		// standardno
	} else {
		return (red->ulaz - red->izlaz + MAXRED);// cirkularnost
	}
}


/************* KRAJ IMPLEMENTACIJE REDA *****************/

int main () {
	Red red;
	int element;  // element, krajevi reda
	int skini;
	FILE *fi;                                          // ulazna datoteka

	init_red(&red);
	fi = fopen ("UlazZaRed.txt", "r");
	if (fi) {
		while (fscanf (fi, "%d", &element) != EOF) {
			// stavljanje u red
			if ((DodajURed (element, &red))) {
				printf ("U red dodan element %d\n", element);
				printf ("\tBroj elemenata u redu je %d\n",
                                                                prebroji (&red));
			} else {
				printf ("Nema vise mjesta u redu. Koliko skinuti?\n");
				scanf ("%d", &skini);
```

```
                        // uklanjanje iz reda
                        while (--skini >= 0 && SkiniIzReda (&element, &red)) {
                                printf ("Iz reda skinut element %d\n", element);
                                printf ("\tBroj elemenata u redu je %d\n",
                                        prebroji (&red));
                        }
                        //break;
                }
        }
        fclose (fi);
        return 0;
    } else {
        printf ("Nema ulazne datoteke\n");
        return 1;
    }
}
```

RedListom.c

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
/***** IMPLEMENTACIJA REDA LISTOM **************/
struct at {
    int element;
    struct at *sljed;
};
typedef struct at atom;
typedef struct {
    atom *ulaz, *izlaz;
} Red;

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

// dodaje element u red, vraca 1 ako uspije, inace 0
int DodajURed (int element, Red *red) {
    atom *novi;
    if (novi = malloc (sizeof (atom))) {
        novi->element = element;
        novi->sljed = NULL;
        if (red->izlaz == NULL) {
            red->izlaz = novi;          // ako je red bio prazan
        } else {
            (red->ulaz)->sljed = novi; // inace, stavi na kraj
        }
        red->ulaz = novi;               // zapamti zadnjeg
        return 1;
    }
    return 0;
}
// uklanja element iz reda, vraca 1 ako uspije, inace 0
int SkiniIzReda (int *element, Red *red) {
    atom *stari;
    if (red->izlaz) {                                   // ako red nije prazan
        *element = (red->izlaz)->element; // element koji se skida
        stari = red->izlaz;                             // zapamti trenutni izlaz
        red->izlaz = (red->izlaz)->sljed;       // novi izlaz
        free (stari);                                   // oslobodi memoriju skinutog
        if (red->izlaz == NULL) red->ulaz = NULL; // prazan red
        return 1;
  }
    return 0;
}
// vraca broj elemenata u redu
int Prebroji (Red *red) {
    int n;
```

```
	atom *izlaz;
	izlaz = red->izlaz;
#if 1
	for (n = 0; izlaz; izlaz = izlaz->sljed) {
		printf ("%d -> ", izlaz->element);
		n++ ;
	}
	printf ("NULL\n");
#else
	// krace
	for (n = 0; izlaz; n++, izlaz = izlaz->sljed);
#endif
	return n;
}

/************* KRAJ IMPLEMENTACIJE REDA *****************/

int main () {
  int broj;			// podatak/kontrola
	Red red;
	init_red(&red);

	printf ("Slucajno se generiraju nenegativni cijeli brojevi.\n");
	printf ("Neparni brojevi upisuju se u red, a parni broj simulira skidanje iz reda\n");
	printf ("Za obavljanje jednog koraka pritisnuti ENTER, za kraj CTRL-C\n");

	// inicijalizacija generatora slucajnih brojeva
	srand ((unsigned) time (NULL));

	while (1) {
		getchar ();		// ENTER, Ctrl-C
		broj = rand ();
		if (broj%2) {
			// Neparne upisujemo u red
			printf ("U red se upisuje broj %d\n", broj);
			if (!DodajURed (broj, &red))
				printf("Nema vise memorije\n");
		} else {
			// Parni broj simulira skidanje iz reda
			if (SkiniIzReda (&broj, &red)) {
				printf ("Iz reda je skinut podatak %d\n", broj);
			} else {
				printf("Red je prazan\n");
			}
		}
		printf ("Broj elemenata u redu: %d\n", Prebroji (&red));
	}
}
```

Lista.c

```
// Lista.c
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>

typedef int tip;

struct at {
  tip element;
  struct at *sljed;
};
typedef struct at atom;
// Dodavanje u listu
// sortiranu po rastucoj vrijednosti elementa
// vraca 1 ako uspije, inace 0
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) {
    // Dodavanje na pocetak liste
    novi->sljed = *glavap;
    *glavap = novi;
  } else {
       // Dodavanje iza postojeceg elementa kad:
       // a) postojeći atom nema sljedećeg
       // b) element u sljedećem cvoru je veći ili jednak novome
       for (p = *glavap; p->sljed && (p->sljed)->element < element; p = p->sljed);
              novi->sljed = p->sljed;
              p->sljed = novi;
  }
  return 1;
}
// ispis elemenata liste
void ispisi (atom *glava) {
  atom *p;
  for (p = glava; p != NULL; p = p->sljed) {
    printf ("Na adresi %p je %d koji gleda na %p\n",
                    p, p->element, p->sljed);
  }
}
// trazenje elementa liste
// vraca pokazivac na trazeni element ili NULL ako ga ne nadje
atom *trazi1 (atom *glava, tip element) {
  atom *p;
  for (p = glava; p != NULL; p = p->sljed) {
       if (p ->element == element) return p;
  }
       return NULL;
}

// trazenje elementa liste - inacica 2
atom *trazi2 (atom *glava, tip element) {
  atom* p;
  for (p = glava; p && p->element != element; p = p->sljed);
  return p;
}

// trazenje elementa liste - inacica 3
atom *trazi3 (atom *glava, tip element) {
  for (; glava && glava->element != element; glava = glava->sljed);
  return glava;
}

// brisanje elementa liste po kljucu
// koristenjem funkcije trazi
int brisi (atom **glavap, tip element) {
  atom *p, *pp;
  if ((p = trazi1 (*glavap, element)) == NULL) //ili trazi2 ili trazi3
       return 0;
       if (p == *glavap) {  // Brisanje s pocetka liste
              pp = (*glavap)->sljed;
              free (*glavap);
              *glavap = pp;
       } else {                 // Brisanje iza clana liste
              // pronadji prethodni atom
              for (pp = *glavap; pp->sljed != p; pp = pp->sljed);
              // Povezi prethodni atom sa sljedbenikom izbrisanog cvora
              pp->sljed = p->sljed;
              // oslobodi memoriju zauzetu elementom koji se brise
              free (p);
       }
       return 1;
}

// Brisanje elementa liste po kljucu
// Objedinjuje traženje i brisanje
int brisi1 (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;
  }
}
int main (void) {
  int element, j; // element i brojac elemenata
  atom *glava;              // glava liste
  FILE *fi;                       // ulazna datoteka
  // inicijalizacija
  fi = fopen ("UlazZaListu.txt", "r");
  if (!fi) exit (1);
  glava = NULL;
  j = 0;
  // citanje i dodavanje elemenata
  while (fscanf (fi, "%d", &element) != EOF) {
    printf  ("%d. ulazni podatak je %d \n", ++j, element);
    if ((dodaj (&glava, element))) {
      ispisi (glava);
              } else {
      printf ("Nema vise mjesta\n");
           break;
    }
  }
  fclose (fi);
  printf ("\n");
       // trazenje i brisanje elemenata
  do {
       ispisi (glava);
    printf ("Upisite element koji se brise >");
    scanf ("%d", &element);
  } while (brisi (&glava, element));
  //} while (brisi1 (&glava, element));
  printf  ("Nema trazenog elementa!\n");
  return 0;
}
```

VisestrukaLista.c

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

struct tip {
  int mbr;
  char prezime[14+1];
};
struct at {
  struct tip element;
  struct at *smbr;
  struct at *sprez;
};
typedef struct at atom;

// Dodavanje u listu
// sortiranu po rastucoj vrijednosti maticnog broja
void DodajMBR (atom **glavap, atom *novi) {
#if 1
       // nadji pokazivac na atom s elementom vecim od elementa novog
       for (; *glavap && (*glavap)->element.mbr < novi->element.mbr;
                                                    glavap = &((*glavap)->smbr));
       // glavap sadrzi adresu trazenog pokazivaca
       // *glavap sadrzi vrijednost tog pokazivaca
```

```
        novi->smbr = *glavap; // novi gleda na veceg po kljucu
        *glavap = novi;                         // pokazivac smbr u cvoru koji prethodi novome
#else
        // Alternativno rješenje, s pomoćnim pokazivačem i while petljom:
        atom **pom;                 // pom sadrzi adresu pokazivaca cvora
        pom = glavap;       // adresa pokazivaca na 1.clan
        while (*pom) {      // adresa pokazivaca na trenutni clan
                if ((*pom)->element.mbr < novi->element.mbr) {
                        // Adresa pokazivaca na sljedeci atom
                        pom = &((*pom)->smbr);
                }       else {
                        break; // Skok iz petlje kad treba ubaciti novog
                }
        }
        // Izmjena pokazivaca smbr u cvoru koji prethodi novome
        novi->smbr = *pom;
        *pom = novi;
#endif
}

// Dodavanje u listu sortiranu po prezimenu
// analogno dodavanju po maticnom broju
void DodajPrezime (atom **glavap, atom *novi) {
#if 1
        for (;*glavap && strcmp((*glavap)->element.prezime,
                                novi->element.prezime) < 0; glavap = &((*glavap)->sprez))
                ;
  novi->sprez = *glavap;
        *glavap = novi;
#else
  // Alternativno rješenje, s pomoćnim pokazivačem i while petljom:
  atom **pom;
  pom = glavap;
  while (*pom) {
    if (strcmp((*pom)->element.prezime, novi->element.prezime) < 0) {
      pom = &((*pom)->sprez);
              } else {
      break;
    }
  }
  novi->sprez = *pom;
  *pom = novi;
#endif
}

// Trazenje clana za zadani maticni broj
int TraziMBR (atom *glava, int mbr, atom *trazeni) { // a) i b)
                        // c) ... atom **trazeni

  int nasao = 0;        // Podrazumijeva se da nije nasao

  while (glava) {       // Dok ima clanova liste
    if (glava->element.mbr < mbr) {
      // maticni broj clana u listi manji od trazenoga => trazi dalje
       glava = glava->smbr;

    } else if (glava->element.mbr == mbr) {
      // maticni broj clana u listi jednak trazenom => nasao
      *trazeni = *glava;    // a) i b) vrati atom na koji pokazuje glava
                        // *trazeni = glava; // c) vrati pokazivac na nadjenog
      nasao = 1;
                        break;

    } else {
      // maticni broj clana u listi vecu id trazenog => nema ga
      break;
    }
  }
  return nasao;
}
```

```
void main (void) {
  FILE *fi;                                          // ulazna datoteka
  int j, mbr;                                      // brojac elemenata, maticni broj za pretragu
  struct tip element;              // element koji se dodaje u listu
  atom *glavambr,                         // glava liste uredjene po mbr
                *glavaprez;                      // glava liste uredjene po prezimenu
       atom *p, *novi;                           // pomocne varijable

       // inicijalizacija
       fi = fopen ("UlazZaVisestrukuListu.txt", "r");
  if (!fi) exit (1);
  glavambr = NULL;
  j = 0;

       // citanje ulaznih podataka
       // i dodavanje u listu uredjenu po mbr
  while (fscanf (fi, "%d %s", &element.mbr,
                     &element.prezime) != EOF) {
    printf ("%d. ulazni podatak je %d %s\n",
                      ++j, element.mbr, element.prezime);
    if ((novi = (atom *) malloc(sizeof(atom))) != NULL) {
      novi->element = element;
      novi->smbr = NULL;
      novi->sprez = NULL;
      DodajMBR (&glavambr, novi);
    } else {
      printf("Nema vise mjesta\n");
      break;
    }
  }
  fclose (fi);

       // ispis po mbr
  p = glavambr;
  printf  ("\nIspis po maticnom broju \n");
  while (p) {
    printf  ("Na adresi %p je %d %s\n", p, p->element.mbr, p->element.prezime);
    p = p->smbr;
  }

  // prolazak kroz listu uredjenu po mbr
       // i povezivanje u listu uredjenu po prezimenu
  glavaprez = NULL;
  novi = glavambr;
  while (novi) {
    DodajPrezime (&glavaprez, novi);
    novi = novi->smbr;
  }

       // ispis po prezimenu
  p = glavaprez;
  printf  (    "\nIspis po prezimenu \n");
  while (p) {
    printf  ("Na adresi %p je %d %s\n",p, p->element.mbr, p->element.prezime);
    p = p->sprez;
  }

  // trazenje clana visestruke liste po MBR
       // varijante:
       // a) vraca se atom, za koji treba rezervirati memoriju
  p = (atom *) malloc (sizeof(atom));

       // b) vraca se atom u deklariranu strukturu, npr. atom c;
       // bez prethodne naredbe malloc

       // c) vraca se adresa cvora, koja se smijesta u p
       // bez prethodne naredbe malloc
```

```
  do {          // ciniti...
    printf ("Upisite maticni broj >");
    scanf ("%d", &mbr);
    if (TraziMBR (glavambr, mbr, p)) {
      // b) : TraziMBR (glavambr, mbr, &c)
                  // c) : TraziMBR (glavambr, mbr, &p)
      printf  ("Za maticni broj %d prezime je %s\n", mbr, p->element.prezime);
    } else {
      printf  ("Za maticni broj %d prezime nije nadjeno\n", mbr);
      break;    // Skok iz petlje i kraj
    }
  } while (1);  //... zauvijek

       exit (0);
}
```

RedListom2.c

```
// RedListom2.c
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <time.h>

struct at2 {
       int element;
       struct at2 *sljed;
       struct at2 *preth;
};
typedef struct at2 atom2;

// dodavanje u red realiziran dvostruko povezanom listom
// funkcija vraca 1 ako uspije, inace 0
int DodajURed (int element, atom2 **glavap, atom2 **repp) {
       atom2 *novi;

       if (novi = malloc (sizeof (atom2))) {
              novi->element = element;
              novi->sljed = NULL;
              novi->preth = NULL;

              if (*glavap == NULL) {                                  // Ako je red bio prazan
                     *glavap = novi; *repp = novi;
              } else {                                                       // inace, stavi na kraj
                     (*repp)->sljed = novi;
                     novi->preth = *repp;
                     *repp = novi;
              }
              return 1;
       }
       return 0;
}

// skidanje iz reda
// funkcija vraca 1 ako uspije, inace 0
int SkiniIzReda (int *element, atom2 **glavap, atom2 **repp) {
       atom2 *stari;

       if (*repp) {  // neprazan red ?
              *element = (*glavap)->element; // vrati element

              if (*glavap == *repp) {    // Ako je samo jedan clan
                     stari = *glavap;
                     *glavap = NULL; *repp = NULL;
              } else {                                                       // inace, povezi ih
                     (*glavap)->sljed->preth = NULL;
                     stari = *glavap;
                     *glavap = stari->sljed;
              }
              free (stari);
```

```
                return 1;
        }
        return 0;
}

// ispis reda
void IspisiRed (atom2 *glava) {
        for (; glava; glava = glava->sljed)
                printf ("%d ", glava->element);
        printf ("\n");
}

// brisanje iz reda clana sa zadanim kljucem
int BrisiIzReda (atom2 **glavap, atom2 **repp, int element) {
        atom2 *pom;
        if (*glavap) { // neprazan red
                // trazi clan
                for (pom = *glavap; pom && (pom->element != element); pom = pom->sljed)
                        ;
                if (pom) {    // Ako je nadjen,
                        if (pom == *glavap) {       // ako je prvi
                                *glavap = pom->sljed;
                                if (pom->sljed)     {             // ako nije jedini
                                        pom->sljed->preth = NULL;
                                } else {                                        // ako jest jedini
                                        *glavap = NULL; *repp = NULL;
                                }
                        } else if (pom == *repp) { // ako je zadnji, ali ne i jedini
                                (*repp)->preth->sljed = NULL;
                                *repp = (*repp)->preth;
                        } else {      // nije ni prvi ni zadnji
                                pom->preth->sljed = pom->sljed;
                                pom->sljed->preth = pom->preth;
                        }
                        free (pom);
                        return 1;
                }
        }
        return 0;            // Nije nadjen ili lista prazna
}


// Red realiziran dvostruko povezanom listom
// omogucuje uklanjanje bilo kojeg clana iz reda
int main () {
        atom2 *glava = NULL; // glava reda
        atom2 *rep = NULL;   // rep reda
        int broj;            // pseudoslucajni broj

        printf ("Slucajno se generiraju nenegativni cijeli brojevi.\n");
        printf ("Neparni brojevi upisuju se u red, a parni broj simulira skidanje iz reda\n");
        printf ("Za obavljanje jednog koraka pritisnuti ENTER, za kraj K\n");

        // inicijalizacija generatora slucajnih brojeva
        srand ((unsigned) time (NULL));

        while (toupper(getch ()) != 'K') {
                broj = rand ();
                if (broj%2) { // Neparne upisujemo u red
                        printf ("U red se upisuje broj %d\n", broj);
                        if (!DodajURed (broj, &glava, &rep))
                                printf("Nema vise memorije\n");
                } else {                    // Parni broj simulira skidanje iz reda
                        if (SkiniIzReda (&broj, &glava, &rep)) {
                                printf ("Iz reda je skinut podatak %d\n", broj);
                        } else {
                                printf("Red je prazan\n");
                        }
                }
                IspisiRed (glava);
```

```
	}
	// brisanje iz reda
	while (1) {
		IspisiRed (glava);
		printf ("Upisite podatak koji se brise iz reda >");
		scanf ("%d", &broj);
		if (!BrisiIzReda (&glava, &rep, broj))
			return 0;
	}
}
```

SortiranoStablo.c BrisanjeCvoraStabla.c

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

struct cv {
  char element[15];
  struct cv *lijevo;
  struct cv *desno;
};
typedef struct cv cvor;

// upisuje u stablo podatke: lijevo manji, desno veci
cvor *upis (cvor *korijen, char element[]) {
	int smjer; // odluka o podstablu
	if (korijen == NULL) { // prazno (pod)stablo
		korijen = (cvor *) malloc (sizeof (cvor));
		if (korijen) {
			strcpy (korijen->element, element);
			korijen->lijevo = korijen->desno = NULL;
		} else {
			printf ("U memoriji mena mjesta za upisati '%s'\n", element);
		}
	} else if ((smjer = strcmp (element, korijen->element)) < 0) {
		korijen->lijevo = upis (korijen->lijevo, element);
	} else if (smjer > 0) {
		korijen->desno  = upis (korijen->desno, element);
	} else {
		printf ("Podatak '%s' vec postoji!\n", element);
	}
	return korijen; // pokazivac na zadnji element
}
// ispis stabla
void ispissta (cvor *korijen, int nivo) {
	int i;
	if (korijen != NULL) {
		ispissta (korijen->desno, nivo+1);
		for (i = 0; i < nivo; i++) printf("    ");
		printf ("%s \n", korijen->element);
		ispissta (korijen->lijevo, nivo+1);
	}
}
// brisnje uparivanjem
void BrisiUparivanjem (cvor **radni) {
	cvor *privremeni = *radni;
	if ((*radni) != NULL) {
		if (!(*radni)->desno)
			(*radni) = (*radni)->lijevo; //ako nema desno dijete, lijevo dijete (ako ga ima)
postaje radni
		else if (!(*radni)->lijevo)
			(*radni) = (*radni)->desno;  // nema lijevo dijete, desno dijete je radni
		else {
			privremeni = (*radni)->lijevo; //1. pomak lijevo
			while (privremeni->desno)   //2. do kraja desno
				privremeni = privremeni->desno;
			privremeni->desno = (*radni)->desno; //povezi najdesniji cvor lijevog podstabla
s desnim podtsblom
```

```
                        privremeni = *radni;
                        *radni = (*radni)->lijevo;
                }

                free (privremeni);
        }
}

// trazenje i brisanje cvora u binarnom stablu
void nadjiBrisi (cvor **korijen, char element[]) {
        cvor *radni = *korijen;
        cvor *preth = NULL;
        int smjer;
        while (radni != NULL) {
                if ((smjer = strcmp (element, radni->element)) == 0)
                        break;
                preth = radni;
                if (smjer < 0)
                        radni = radni->lijevo;
                else
                        radni = radni->desno;
        }
        if (radni != NULL && smjer == 0)
                if (radni == *korijen)
                        BrisiUparivanjem (korijen);
                else if (preth->lijevo == radni) {
                        BrisiUparivanjem (&(preth->lijevo));
                } else {
                        BrisiUparivanjem (&(preth->desno));
                }
        else if (korijen != NULL)
                printf ("%s nije u stablu\n", element);
        else ("Stablo je prazno\n");
}
int main() {
        FILE *fi;                                       // ulazna datoteka
        int j;                                          // brojac podataka
        cvor *korijen;      // pokazivac na korijen, pomocni pokazivac
        char ime[15];
        fi = fopen ("UlazZaSortiranoStablo.txt", "r");
        if (fi) {
                // inicijalizacija i citanje podataka
                j = 1;
                korijen = NULL;
                while (fscanf (fi, "%s", &ime) != EOF) {
                        printf ("%d. ulazni podatak je %s \n", j++, ime);
                        korijen = upis (korijen, ime);
                }
                fclose (fi);
                // obilazak i ispis stabla
         getchar ();
                printf ("Ispis stabla\n");
                ispissta (korijen, 0);
                // trazenje elementa
                while (1) {
                        printf ("Unesite element koji trazite, ili KRAJ >");
                        scanf ("%s", ime);
                        if (stricmp (ime, "KRAJ") == 0) break;
                        nadjiBrisi (&korijen, ime);
                        printf ("Ispis stabla nakon brisanja elementa %s\n", ime);
                        ispissta (korijen, 0);
                }
        } else {
                printf ("Nema ulaznih podataka\n");
                return 1;
        }
        return 0;
}
```

ProsjekUStablu.c

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

typedef struct {
  char artikl[15+1];
  float cijena;
} el;
typedef struct cv {
  el element;
  struct cv *lijevo;
  struct cv *desno;
} cvor;
typedef struct {
  float suma;
  int broj;
} pros;
// upisuje u stablo podatke: lijevo manji, desno veci
cvor *upis (cvor *korijen, el element) {
  int smjer;
  if (korijen == NULL) {
    korijen = (cvor *) malloc (sizeof (cvor));
       if (korijen) {
              korijen->element = element;
              //strcpy (korijen->element.artikl, element.artikl);
              //korijen->element.cijena = element.cijena;
              korijen->lijevo = korijen->desno = NULL;
       } else {
              printf ("Nema dovoljno memorije!\n");
       }
  } else if ((smjer = strcmp (element.artikl, korijen->element.artikl)) < 0) {
    korijen->lijevo = upis (korijen->lijevo, element);
  } else if (smjer > 0) {
    korijen->desno  = upis (korijen->desno, element);
  } else {
    printf ("Podatak '%s' vec postoji!\n", element.artikl);
  }
  return korijen;
}
// ispis inorder
void ispisin (cvor *korijen) {
  if (korijen != NULL) {
    ispisin (korijen->lijevo);
    printf ("%-15s %6.2f\n", korijen->element.artikl, korijen->element.cijena);
    ispisin (korijen->desno);
  }
}
// sumiranje cijena i brojanje elemenata element po element
void prosjek (cvor *korijen, pros *prs) {
  if (korijen != NULL) {
              prs->suma += korijen->element.cijena;
              prs->broj++;
    prosjek (korijen->lijevo, prs);
    prosjek (korijen->desno, prs);
  }
}
// brojanje elemenata stabla
int prebroji (cvor *korijen) {
  if (korijen != NULL) {
    return prebroji (korijen->lijevo) + 1 + prebroji (korijen->desno);
  } else {
    return 0;
  }
}
// sumiranje cijena i brojanje elemenata po podstablima
pros prosjek1 (cvor *korijen) {
  pros prs, prslijevo, prsdesno;
  if (korijen != NULL) {
```

```
    prslijevo = prosjek1 (korijen->lijevo);
    prsdesno = prosjek1 (korijen->desno);
    prs.broj = prslijevo.broj + 1 + prsdesno.broj;
    prs.suma = prslijevo.suma + korijen->element.cijena + prsdesno.suma;
  } else {
    prs.broj = 0; prs.suma = 0;
  }
  return prs;
}
int main(void) {
  FILE *fi;                // ulazna datoteka
  int j;                   // brojac elemenata
  cvor *korijen;           // pokazivac na korijen
  el element;        // sadrzaj cvora
  pros prs, prs1;    // broj elemenata i suma cijena
  // inicijalizacija
  prs.suma = 0.; prs.broj = 0;
  fi = fopen ("UlazZaProsjekUStablu.txt", "r");
  if (!fi) {
    printf ("Nema ulaznih podataka\n");
       return 1;
  }
  // citanje i upis
  j = 1;
  korijen = NULL;
  while (fscanf (fi, "%s %f", element.artikl, &element.cijena) != EOF) {
    printf ("%2d. ulazni podatak je %-15s %6.2f\n", j++, element.artikl, element.cijena);
    korijen = upis (korijen, element);
  }
  fclose (fi);

  // ispis, racun sume cijena i broja elemenata
  getchar ();
  ispisin (korijen);
  getchar ();
  prosjek (korijen, &prs);
  if (prs.broj) {
    printf ("Suma=%f, Broj cvorova=%d, Prosjek=%f\n",
                  prs.suma, prs.broj, prs.suma / prs.broj);
       printf ("Broj cvorova (bez argumenta funkcije) = %d\n", prebroji (korijen));
       printf ("Izracunato na drugi nacin:\n");
       prs1 = prosjek1(korijen);
       printf ("Suma=%f, Broj cvorova=%d, Prosjek=%f\n",
                  prs1.suma, prs1.broj, prs1.suma / prs1.broj);
       // varijante:
       printf ("Prosjek varijanta a) = %f\n",
                  prs1.suma / prebroji (korijen));
       printf ("Prosjek varijanta b) = %f\n",
                  (prosjek1 (korijen)).suma / (prosjek1 (korijen)).broj);
       getchar ();
  }
       return 0;
}
```

GomiluStvori.c

```
-------------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#define MAXGOM 100
typedef int tip;
// ubacuje vrijednost iz A[k] na gomilu pohranjenu u A[1:k-1]
void ubaci (tip A[], int j) {
  int i, k;
  tip novi;
  k = j;
  i = j/2;
  novi = A[j];
  while ((i > 0) && (A[i] < novi)) {
    A[k] = A[i]; // spusti roditelja na vecu razinu
    k = i;
```

```
    i /= 2;      // roditelj od A[i] je na A[i/2]
  }
  A[k] = novi;
}
int main () {
  FILE *fi;
  int i, j, k;
  tip A[MAXGOM];
  // citaj i ubacuj u gomilu
  fi = fopen ("UlazZaGomilu.txt", "r");
  if (fi) {
              j = 1;
              while (j < MAXGOM && fscanf(fi, "%d", &A[j]) != EOF) {
                     printf ("%d. ulazni podatak je %d\n", j, A [j]);
                     ubaci (A, j);
                     j++;
              }
              fclose (fi);
              // ispisi gomilu po retcima
              i = 1;
              k = 1;
              while (i < j) { // petlja do zadnjeg u gomili
                     // pisi do maksimalnog u gomili razine k
                     for (; i <= pow (2, k) - 1 && i < j; i++) {
                            printf(" %d ", A[i]);
                     }
                     k++; // spusti se na vecu razinu
                     printf ("\n");
              }
       } else {
              printf ("Nema ulazne datoteke\n");
       }
       getchar();
       return 0;
}
```

GomiluPodesi.c

```
-------------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#define MAXGOM 100
typedef int tip;
// potpuna binarna stabla s korijenima A[2*i]
// i A[2*i+1]kombiniraju se s A[i] formirajuci
// jedinstvenu gomilu
// 1 <= i <= n
void podesi (tip A[], int i, int n) {
  int j;
  tip stavka;
  j = 2*i;
  stavka = A[i];
  while (j <= n ) {
    // Usporedi lijevo i desno dijete (ako ga ima)
    if ((j < n) && (A[j] < A[j+1])) j++;
    // j pokazuje na vece dijete
    if (stavka >= A[j]) break;    // stavka je na dobrom mjestu
    A[j/2] = A[j];                       // vece dijete podigni za razinu
    j *=2;
  }
  A[j/2] = stavka; // pohrani stavku
}
// premjesti elemente A[1:n] da tvore gomilu
void StvoriGomilu (tip A[], int n) {
  int i;
  for (i = n/2; i >= 1; i--)
              podesi (A, i, n);
}
int main(void) {
       FILE *fi;
       int i, j, k, n;
```

```
	tip A[MAXGOM]; // gomila
	// citanje podataka
	fi = fopen ("UlazZaGomilu.txt", "r");
	if (fi) {
		j = 1;
		while (j < MAXGOM && fscanf (fi, "%d", &A[j]) != EOF) {
			printf ("%d. ulazni podatak je %d \n", j, A[j]);
			j++;
		}
		fclose (fi);
		// podesi broj elemenata i stvori gomilu
		n = j - 1;
		StvoriGomilu (A, n);
		// ispisi gomilu po retcima
		i = 1;
		k = 1;
		while (i < j) { // petlja do zadnjeg u gomili
			// pisi do maksimalnog u gomili razine k
			for (; i <= pow (2, k) - 1 && i < j; i++) {
				printf(" %d ", A[i]);
			}
			k++; // povecaj razinu
			printf ("\n");
		}
	} else {
		printf ("Nema ulazne datoteke\n");
	}
	getchar();
	return 0;
}
```

Aposteriori.c

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

int main()
{
  int i, j, n;
  struct timeb vrijeme1, vrijeme2; long trajanjems;

  while (scanf("%d", &n)==1 && n > 0) {
    ftime (&vrijeme1);

    for (i = 1; i < n; ++i) {
      if (i % 100 == 0) printf (".");

      for (j = 0; j < i; ++j)
        ;

    }

    ftime (&vrijeme2);
    trajanjems = 1000 * (vrijeme2.time - vrijeme1.time) +
                 vrijeme2.millitm - vrijeme1.millitm;
    printf("\n%ld ms\n", trajanjems);
  }
       return 0;
}
```

DvostrukiPokazivac.c

```
#include <stdio.h>
void f(int **ppa, int *novaAdresa){
       *ppa = novaAdresa;
}
int main(){
       int a=1, b=2;
       int *pa;
       pa = &a;
       printf("\n &pa=%p pa=%p *pa=%d", &pa, pa, *pa);
       f(&pa, &b);
       printf("\n &pa=%p pa=%p *pa=%d\n", &pa, pa, *pa);
       return 0;
}
```

glavni.c

```
#include <iostream>
#include <ctime>
#include "stog.h"
using namespace std;


int main () {
       int novi, stari;

       Stog *stog = new Stog();

       cout << "Slucajno se generiraju nenegativni cijeli brojevi.\n";
       cout << "Neparni brojevi upisuju se na stog\n";
       cout << "Parni broj simulira skidanje sa stoga\n";
       cout << "Za obavljanje jednog koraka pritisnuti ENTER, za kraj K\n\n";
       srand ((unsigned) time (NULL));
       while (1) {
              stog->Ispisi();
              putchar ('\n');
              if (toupper (getchar ()) == 'K') break;
              novi = rand ();
              if (novi%2) {       // Neparni se upisuju na stog
```

```
                    cout << "Dodaj " << novi;
                    try {
                            stog->Dodaj (novi);
                    }
                    catch (char *poruka)
                    {
                            cout << "Pogreska: " << poruka << '\n';
                    }
            } else {                    // Parni broj simulira skidanje sa stoga
                    try {
                            stog->Skini (stari);
                            cout << " skinut " << stari << '\n';
                    }
                    catch (char *poruka)
                    {
                            cout << "Pogreska: " << poruka << '\n';
                    }
            }

        }
        delete stog;
        return 0;
}
```

Imena.c

```
//citanje po blokovima

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

int hash(char *str, int tableSize) {
    int sum =0;

    if (!str) return -1;
    for ( ; *str; str++) sum+=*str;
        printf("%d - %d\n", sum, tableSize);
    return sum % tableSize;

}

int main()
{
    char ime[20];
    while (1) {
    printf("Unesi ime:");
    gets(ime);
    printf("Hash vrijednost imena %s je %d\n", ime, hash(ime, 11));
}
    system("PAUSE");
    return 0;
}
```

ListaNaDisku1.c

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

typedef struct {
  int kljuc;
} tip;

typedef struct {
  tip element;
  long smbr;
} cvor;
```

```c
int dodaj (tip element, FILE *fl) {
  // Dodavanje u listu sortiranu po rastucoj
  // vrijednosti maticnog broja
  long noviadr, prethodni, sljedeci;
  cvor c, novi;
  // Fizički upis i dodavanje po maticnom broju
  fseek(fl, 0L, SEEK_END);
  noviadr = ftell (fl);
  novi.smbr = 0;
  novi.element = element;
  prethodni = 0;
  fseek (fl, 0L, SEEK_SET);
  fread (&sljedeci, sizeof(sljedeci), 1, fl);
  while (sljedeci) {
    fseek (fl, sljedeci, SEEK_SET);
    fread (&c, sizeof (c), 1, fl);
    if (c.element.kljuc > element.kljuc) {
      // novi se ubacuje ispred
      novi.smbr = sljedeci;
      break;
    }
    prethodni = sljedeci + sizeof (c.element);
    sljedeci = c.smbr;
  }
      // Povezivanje prethodnog
  fseek (fl, prethodni, SEEK_SET);
  if (fwrite (&noviadr, sizeof (noviadr), 1, fl) != 1) return 0;
  fseek (fl, noviadr, SEEK_SET);
  if (fwrite (&novi, sizeof(novi), 1, fl) != 1) return 0;
  return 1;
}
int main () {
  FILE *fi, *fl;
  int j;
  tip element;
  cvor c;
  long glava, sljedeci;

  fi = fopen ("UlazZaListu1.txt", "r");
  fl = fopen ("Lista1", "w+b");
  if (!fi || !fl) exit (0);

  glava = 0;
  fwrite (&glava, sizeof (glava), 1, fl);

  j = 0;
  while (fscanf (fi, "%d", &element.kljuc) != EOF) {
    printf ("\n%d. ulazni podatak je %d\n", ++j, element.kljuc);
    if (!dodaj (element, fl)) {
      printf("Nema vise mjesta\n");
      break;
    }
  }
  fclose (fi);
  fseek (fl, 0L, SEEK_SET);
  fread (&sljedeci, sizeof (sljedeci), 1, fl);
  while (sljedeci) {
    fseek (fl, sljedeci, SEEK_SET);
    fread(&c, sizeof (c), 1, fl);
    printf ("Na adresi %ld je %d\n",sljedeci, c.element.kljuc);
    sljedeci = c.smbr;
  }
  return 0;
}
```

ListaNaDisku2.c

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

```c
typedef struct {
  int kljuc;
} tip;

typedef struct {
  tip element;
  long smbr;
} cvor;

int dodaj (tip element, FILE *fl) {
  // Dodavanje u listu sortiranu po rastucoj
  // vrijednosti maticnog broja
  long noviadr, prethodni, sljedeci;
  cvor c, novi;

  // Ima li u listi brisanih?
  fseek(fl, 4L, SEEK_SET);
  fread (&sljedeci, sizeof(sljedeci), 1, fl);
  if (sljedeci > 0) {
       // Ima, zapisat cemo na mjesto prvog iz liste brisanih
       noviadr = sljedeci;
       // Prespajanje glave brisanih
    fseek (fl, sljedeci, SEEK_SET);
    fread (&c, sizeof (c), 1, fl);
       fseek(fl, 4L, SEEK_SET);
    fwrite (&(c.smbr), sizeof(sljedeci), 1, fl);
  } else {
    // Nema, dodavanje na kraj datoteke
    fseek(fl, 0L, SEEK_END);
    noviadr = ftell (fl);
  }
  // Fizički upis i dodavanje po maticnom broju, jednako kao ListaNaDisku1
  novi.smbr = 0;
  novi.element = element;
  prethodni = 0;
  fseek (fl, 0L, SEEK_SET);
  fread (&sljedeci, sizeof(sljedeci), 1, fl);
  while (sljedeci) {
    fseek (fl, sljedeci, SEEK_SET);
    fread (&c, sizeof (c), 1, fl);
    if (c.element.kljuc > element.kljuc) {
      // novi se ubacuje ispred
      novi.smbr = sljedeci;
      break;
    }
    prethodni = sljedeci + sizeof (c.element);
    sljedeci = c.smbr;
  }
       // Povezivanje prethodnog
  fseek (fl, prethodni, SEEK_SET);
  if (fwrite (&noviadr, sizeof (noviadr), 1, fl) != 1) return 0;
  fseek (fl, noviadr, SEEK_SET);
  if (fwrite (&novi, sizeof(novi), 1, fl) != 1) return 0;
  return 1;
}
int brisi (tip element, FILE *fl) {
  // Brisanje iz uzlazno sortirane liste
  long prethodni, sljedeci;
  cvor c;
  prethodni = 0;
  fseek (fl, 0L, SEEK_SET);
  fread (&sljedeci, sizeof(sljedeci), 1, fl);
  while (sljedeci) {
    fseek (fl, sljedeci, SEEK_SET);
    fread (&c, sizeof (c), 1, fl);
    if (c.element.kljuc == element.kljuc) {
              // Povezivanje prethodnog sa sljedecim
              fseek (fl, prethodni, SEEK_SET);
              fwrite (&(c.smbr), sizeof(c.smbr), 1, fl);
```

```
              // Postavljanje na vrh liste brisanih
              fseek (fl, 4L, SEEK_SET);
              fread (&(c.smbr), sizeof(c.smbr), 1, fl);
              fseek (fl, 4L, SEEK_SET);
              fwrite (&sljedeci, sizeof(sljedeci), 1, fl);
              fseek (fl, sljedeci, SEEK_SET);
              fwrite (&c, sizeof(c), 1, fl);
              return 1;
    } else if (c.element.kljuc > element.kljuc) {
              // Nema smisla tražiti dalje, nije nasao
              return 0;
              }
    prethodni = sljedeci + sizeof (c.element);
    sljedeci = c.smbr;
  }
  return 0;
}

void lista (FILE *fl, long pozicijaglave, char *nazivliste) {
       // Ispisuje listu zadanu pozicijom glave: 0L lista, 4L lista brisanih
  long sljedeci;
  cvor c;
  printf ("%s\n", nazivliste);
  fseek (fl, pozicijaglave, SEEK_SET);
  fread (&sljedeci, sizeof (sljedeci), 1, fl);
  while (sljedeci) {
    fseek (fl, sljedeci, SEEK_SET);
    fread(&c, sizeof (c), 1, fl);
    printf ("Na adresi %ld je %d\n", sljedeci, c.element.kljuc);
    sljedeci = c.smbr;
  }
}
int main () {
  FILE *fi, *fl;
  int j;
  tip element;
  long glava, glavabrisanih;
  char op[80];
  fi = fopen ("UlazZaListu2.txt", "r");
  fl = fopen ("Lista2", "w+b");
  if (!fi || !fl) exit (0);
  glava = 0; glavabrisanih = 0;
  fwrite (&glava, sizeof (glava), 1, fl);
  fwrite (&glavabrisanih, sizeof (glavabrisanih), 1, fl);
  j = 0;
  while (fscanf (fi, "%d", &element.kljuc) != EOF) {
    printf ("\n%d. ulazni podatak je %d\n", ++j, element.kljuc);
    if (!dodaj (element, fl)) {
      printf("Nema vise mjesta\n");
      break;
    }
  }
  fclose (fi);
       lista (fl, 0L, "Lista: ");
       while (1) {
              printf ("Unesite oznaku operacije (Dodaj,Brisi,Kraj)>");
              scanf ("%s", op);
              switch (tolower(op[0])) {
              case 'd':
              printf ("Unesite kljuc elementa kojeg zelite dodati>");
                scanf ("%d", &element.kljuc);
          dodaj (element, fl);
                    break;
              case 'b':
              printf ("Unesite kljuc elementa kojeg zelite brisati>");
                    scanf ("%d", &element.kljuc);
          brisi (element, fl);
                    break;
              case 'k':
                    exit(0);
```

```
		}
		lista (fl, 0L, "Lista: ");
		lista (fl, 4L, "Lista brisanih: ");
	}
	return 0;
}
```

ListaOO.c

```
--------------------------------------------------------------------------
#include <iostream>
#include <ctime>

using namespace std;

struct podatak {
	int kljuc;
	int element;
};

class Lista {
private:
	struct atom {
		podatak _stavka;
		atom *_sljed;
	};
	atom *_glava;
	atom *_rep;

public:
	Lista();
	Lista(const Lista &izvor);
	~Lista();

	void DodajNaPocetak(podatak stavka);
	void DodajNaKraj(podatak stavka);
	bool DodajSortirano(podatak stavka,bool uzlazno);
	bool NadjiPoKljucu(int kljuc,podatak &stavka);
	bool BrisiSPocetka(podatak &stavka);
	bool BrisiZadanog(int kljuc,podatak &stavka);
	void BrisiSve();
	void IspisiListu();
};


Lista::Lista() {
	_glava = NULL;
	_rep = NULL;
}

Lista::Lista(const Lista &izvor) {
	atom *pAtomPred,*pAtomIzvor;
	if ((izvor._glava==NULL) || (izvor._rep==NULL))
		_glava=_rep=NULL;
	else
	{
		_glava=new atom; // procisceni copy constructor
		_glava->_stavka=izvor._glava->_stavka;
		pAtomPred=_glava;
		for (pAtomIzvor=izvor._glava->_sljed; pAtomIzvor!=NULL; pAtomIzvor=pAtomIzvor->_sljed)
		{
			pAtomPred->_sljed=new atom;
			pAtomPred=pAtomPred->_sljed;
			pAtomPred->_stavka=pAtomIzvor->_stavka;
		}
		pAtomPred->_sljed=NULL;
		_rep=pAtomPred;
	}
}

Lista::~Lista() {
```

```
	BrisiSve ();
}
void Lista::DodajNaPocetak(podatak stavka) {
	atom *novi = new atom;

	novi->_stavka = stavka;
	novi->_sljed = _glava;
	_glava = novi;
	if (_rep == NULL)
		_rep = novi;
	cout << " na adresu " << novi;
}
void Lista::DodajNaKraj(podatak stavka) {
	atom *novi = new atom;

	novi->_stavka = stavka;
	novi->_sljed = NULL;
	if (_rep != NULL)
		_rep->_sljed = novi;
	else
		_glava = novi;
	_rep = novi;
	cout << " na adresu " << novi;
}
bool Lista::DodajSortirano(podatak stavka, bool uzlazno) {
	atom *novi = new atom;
	atom *p;

	novi->_stavka = stavka;
	if (_glava == NULL || (uzlazno ^ (_glava->_stavka.element < stavka.element))) {
		// Dodavanje na pocetak liste
		// koristi se "ekskluzivno ili" smjera i usporedbe
		DodajNaPocetak (stavka);
	} else {
		// Dodavanje iza postojeceg elementa kad:
		// nema sljedećeg ili element u sljedećem je (uzlazno ^ veci od novoga)
		for (p = _glava; (p->_sljed != NULL) &&
			(uzlazno ^ (p->_sljed->_stavka.element > stavka.element)); p = p->_sljed);
		// da li takav vec postoji
		if ((p->_sljed != NULL) && (p->_sljed->_stavka.element == stavka.element))
			return 0;
		novi->_sljed = p->_sljed;
		p->_sljed = novi;
		cout << " na adresu " << novi;
	}

	return 1;
}

bool Lista::NadjiPoKljucu(int kljuc, podatak& stavka) {
	atom *p;

	for (p = _glava; (p != NULL) && (p->_stavka.kljuc != kljuc); p = p->_sljed);
	if (p != NULL) {
		stavka = p->_stavka;
		return true;
	} else
		return false;
}

bool Lista::BrisiSPocetka(podatak& stavka) {
	if (_glava != NULL) {
		stavka = _glava->_stavka;
		_glava = _glava->_sljed;
		// ako je bio jedini
		if (_glava == NULL)
			_rep = NULL;
		return true;
	}
	return false;
```

```
}

bool Lista::BrisiZadanog(int kljuc, podatak& stavka) {
#if 0
	atom **glavap, *p;
	glavap = &_glava;
	for (; *glavap && (*glavap)->_stavka.kljuc != kljuc; glavap = &(*glavap)->_sljed);
	if (*glavap) {
		p = *glavap;
		*glavap = (*glavap)->_sljed;
		stavka = p->_stavka;
		if (p->_sljed == NULL )
			_rep = p;
		free (p);
		return true;
	} else
		return false;
#else
	atom *p, *preth = NULL;
	for (p = _glava; (p != NULL); p = p->_sljed){
		if (p->_stavka.kljuc == kljuc){
			if (_glava == p){  // brisanje 1. elementa - promjena glave
				_glava = p->_sljed;
			}else {   // brisanje elementa u tijelu liste
				preth->_sljed = p->_sljed;
			}
			if (_rep == p){  // da li se briše zadnji element?
				_rep = preth;
			}
			stavka = p->_stavka;
			free(p);
			return true;
		}
		preth = p;
	}
	return false;
#endif
}
void Lista::IspisiListu() {
	atom *pom;
	pom = _glava;
	if (pom == NULL)
		cout <<"Lista je prazna\n";
	else
		for (pom; pom !=NULL; pom = pom->_sljed)
			cout << "Kljuc=" << pom->_stavka.kljuc << " element="<< pom->_stavka.element <<
endl;
}

void Lista::BrisiSve() {
	atom *pom;
	while (_glava!=NULL)
	{
		pom = _glava;
		_glava = _glava->_sljed;
		free (pom);
	}
	_rep = NULL;
}

int main () {
	int kljuc = 0, kljucpriv;
	int kuda, sto;
	bool uzlazno;
	char dane;
	podatak stavka;
	Lista lista;
	Lista *pKopijaListe;

	char cSig;
```

```
	srand ((unsigned) time (NULL));

	do
	{
		cout << "Upisite oznaku zeljene akcije:\n\t1-dodaj, 2-nadji, 3-brisi, 4-ispisi, 5-
zavrsi\n>";
		cin >> sto;
		switch (sto)
		{
			case 1:
				cout << "Slucajni brojevi dodaju se u listu dok se ne upise K\n";
				do
				{
					cout << "Upisite oznaku nacina dodavanja:\n\t1-na pocetak, 2-na
kraj, 3-sortirano\n>";
					cin >> kuda;
					switch (kuda)
					{
						case 1:
							cout << "Na pocetak";
							break;
						case 2:
							cout << "Na kraj";
							break;
						case 3:
						 cout << "Sortirano - uzlazno (1) ili silazno (0)?\n";
							cin >> uzlazno;
							if (uzlazno)
								cout << "Uzlazno!";
							else
								cout << "Silazno!";
							break;
						default:
							cout << "Pogresna oznaka nacina dodavanja\n";
							break;
					}
				} while ((kuda<1) || (kuda>3));
				do {
					putchar ('\n');
					cSig=toupper(getchar());
					if (cSig!='K')
					{
						stavka.kljuc = ++kljuc;
						stavka.element = rand ();
						cout << "Dodaj kljuc=" << stavka.kljuc <<" element=" <<
stavka.element;
						switch (kuda)
						{
							case 1:
								lista.DodajNaPocetak (stavka);
								break;
							case 2:
								lista.DodajNaKraj (stavka);
								break;
							case 3:
								if (!(lista.DodajSortirano(stavka, uzlazno)))
									cout << "Kljuc " << stavka.kljuc <<"
vec postoji!";
								break;
						}
					}
				} while (cSig!='K');
				break;

			case 2:
				cout << "Upisite vrijednost kljuca\n";
				cin >> kljucpriv;
				if (lista.NadjiPoKljucu(kljucpriv, stavka))
```

```
                                cout << "Kljuc=" << stavka.kljuc <<" element=" << stavka.element
<< '\n';
                        else
                                cout << "Stavka s kljucem " << kljucpriv << " nije nadjena!\n";
                        break;

                case 3:
                        do
                        {
                                cout << "Upisite oznaku nacina brisanja:\n\t1-s pocetka, 2-za
zadani kljuc, 3-sve\n>";
                                cin >> kuda;
                                switch (kuda)
                                {
                                        case 1:
                                                if (lista.BrisiSPocetka(stavka))
                                                        cout << "Izbrisana stavka s pocetka\nKljuc ="
<< stavka.kljuc << " element =" << stavka.element << '\n';
                                                else
                                                        cout << "Lista je prazna\n";
                                                break;
                                        case 2:
                                                cout << "Upisite kljuc stavke koja se brise >";
                                                cin >> kljucpriv;
                                                if (lista.BrisiZadanog(kljucpriv, stavka))
                                                        cout << "Izbrisana stavka \n Kljuc =" <<
stavka.kljuc << " element =" << stavka.element << '\n';
                                                else
                                                        cout << "Stavka s kljucem " << kljucpriv << "
nije nadjena!\n";
                                                break;
                                        case 3:
                                                lista.BrisiSve ();
                                                break;
                                        default:
                                                cout << "Pogresna oznaka nacina brisanja\n";
                                                break;
                                }
                        } while ((kuda<1) || (kuda>3));
                        break;

                case 4:
                        lista.IspisiListu();
                        cout << "Zelite li kopirati listu? (D/N)>";
                        cin >> dane;
                        if (toupper (dane) == 'D')
                        {
                                cout << "\nKopiranje liste\n";
                                pKopijaListe = new Lista(lista);
                                cout << "Ispis kopije\n";
                                pKopijaListe->IspisiListu();
                                delete pKopijaListe;
                        }
                        break;

                default:
                        break;
                }
        } while (sto!=5);

        return 0;
}
```

MallocPrijepis.c

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <stdlib.h>
void fatal (char *poruka) {
```

```
  fputs (poruka, stderr); fputs ("\n", stderr);
  exit (1);
}
#define MAXREDAKA 1000
#define MAXBUF 512
int main (int argc, char *argv[]) {
  FILE *d;
  char *redak[MAXREDAKA];
  char buf[MAXBUF+1];
  int i, n;
  if (argc != 3)
    fatal ("Poziv programa: prijepis stara nova");
  if ((d = fopen (argv[1], "r")) == NULL)
    fatal ("Ne moze se otvoriti ulazna datoteka");
  n = 0;
  while (fgets (buf, MAXBUF, d) != NULL && n < MAXREDAKA) {
    redak [n] = (char *) malloc (strlen (buf) + 1);
    if (redak[n] == NULL)fatal ("Nedovoljno memorije!");
    strcpy (redak[n], buf);
    n++;
  }
  fclose (d);
  if ((d = fopen (argv[2], "w")) == NULL)
    fatal ("Ne moze se stvoriti izlazna datoteka");
  for (i = n-1; i >= 0; i--) {
    fputs (redak[i], d);  free (redak[i]);
  }
  fclose (d);
  return 0;
}
```

MaxClanStd.c

```
// MaxClanStd.c
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

#define MAX 10
typedef int tip; // primjer za viseoblicje

// trazenje najveceg clana u polju A od n clanova
tip maxclan (tip A[], int n) {
  int i, imax;
  imax = 0;   // uzmimo da je prvi najveci
  for (i = 1; i < n; i++) {       // provjerava ostale
       printf ("trenutni A[%d]=%3d ? najveci A[%d]=%3d",
                                          i, A[i], imax, A[imax]);
    if (A[i] > A[imax])    {      // i-ti je veci
              imax = i;           // zapamti mu indeks
              printf (" => novi imax=%d", imax);
       }
       printf ("\n");
   }
  return A[imax];    // vrati vrijednost najveceg clana
}

int main () {
  FILE *fi;
  int j, n;
  tip A[MAX];
  fi = fopen ("UlazZaMaxClanStd.txt", "r");
  if (!fi) exit (1);
  n = 0;
  while (n < MAX && fscanf (fi, "%d", &A[n]) != EOF) n++;
  for (j = 0; j < n; j++) printf("A[%d] = %3d\n", j, A[j]);
  printf ("Najveci clan je %d\n", maxclan(A, n));
  fclose (fi);

  return 0;
```

```
}
```

PodijeliUpariDatoteke.c

```
---------------------------------------------------------------------------
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAXREDAKA 100
#define MAXZNAKOVA 512

void MergeNFile (FILE **fpolje, FILE *fout, int brdat){
	int i, ind;
	char **plinija;
	plinija =(char**)malloc(brdat * sizeof(char*));
	for(i = 0; i < brdat; i++){
		rewind(fpolje[i]);  //premotati na pocetak
		plinija[i]=(char *)malloc(MAXZNAKOVA + 1);
		if (!fgets(plinija[i], MAXZNAKOVA, fpolje[i]))
			*plinija[i] = '\0';
	}
	while (1){
		ind = -1;
		for(i = 0; i < brdat; i++){
			if (*plinija[i]) {
				if (ind == -1 || strcmp(plinija[i], plinija[ind]) < 0)
					ind = i;
			}
		}
		if (ind == -1) break;
		fputs (plinija[ind], fout);
		if (!fgets(plinija[ind], MAXZNAKOVA, fpolje[ind])) *plinija[ind] = '\0';
	}
	for(i = 0; i < brdat; i++) free(plinija[i]);
	free(plinija);
}
int compare( const void *redak1, const void *redak2 ) {
  return strcmp( * (char **) redak1, * (char **) redak2 );
}
int main() {
	FILE *f, *fout, **fpolje = NULL;
	int brdat, brojaczapisa, i;
	char *flag, imedat[30], *redak[MAXREDAKA];
	char linija[MAXZNAKOVA + 1];
	if ((f = fopen ("dat1.txt", "r")) == NULL) {
		printf("Pogreska kod otvaranja datoteke dat1.txt");
		exit (1);
	}
	for (brdat = 0; ;) {
		// dok imamo manje od veličine bloka i imamo što čitati
		for (brojaczapisa = 0; brojaczapisa < MAXREDAKA &&
			(flag = fgets (linija, MAXZNAKOVA, f)); brojaczapisa ++) {
			redak [brojaczapisa]= (char *) malloc(strlen (linija) + 1);
			if (redak[brojaczapisa] == NULL){
				fprintf (stderr, "Nedovoljno memorije!");
				exit(1);
			}
			strcpy (redak[brojaczapisa], linija);
		}
		if (brojaczapisa > 0) {
			//treba sortirati podatke
			qsort((void *)redak, brojaczapisa, sizeof(char *), compare);
			//treba zapisati u novu datoteku
			fpolje = (FILE **)realloc(fpolje,(brdat + 1)* sizeof(FILE*));
			sprintf (imedat, "%03d.txt", brdat);
			if (( fpolje[brdat] = fopen (imedat, "w+")) == NULL){
				printf("Pogreška kod otvaranja datoteke %s", imedat);
				return 1;
			}
			for(i = 0; i < brojaczapisa; i++)
				fputs(redak[i], fpolje[brdat]);
```

```
                        brdat ++;
                }
                if (!flag)    break;
        }
        if ((fout = fopen ("out.txt", "w")) == NULL){
                fprintf (stderr, "Pogreška kod otvaranja datoteke out.txt");
                return 1;
        }
        MergeNFile (fpolje, fout, brdat);
        for (i = 0; i < brdat; i++)
                fclose(fpolje[i]);
        fclose(f);
        fclose(fout);
        free(fpolje);
        return 0;
}
```

PogadjanjeBroja.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>
#include <math.h>
#include <string.h>

int potez(int dg, int gg){
   int sredina;
   char odg;
   if (gg == dg) {
      printf("\nTrazeni broj je %d", dg);
      return 1;
   }else{
      sredina = (gg + dg)/2;
      printf("\nDa li je %d [D/V/M]?", sredina);
      odg = tolower(getche());
      if (odg == 'd'){
         printf("\n!! Trazeni broj je %d", sredina);
         return 1;
      }else if (odg == 'v'){
         return potez(sredina + 1, gg);
      }else if (odg == 'm'){
         return potez(dg, sredina - 1);
      }
   }
}
/*****************************************************************************/
int main(){
  potez(0, 100);
  system("pause");
}
/*****************************************************************************/
int slucajniBroj;

char student(int broj){
   /*
   static int slucajniBroj = 0;
   if (broj<0) slucajniBroj = rand()%101;
   */
   if (broj == slucajniBroj){
      return 'd';
   }else if (broj < slucajniBroj){
      return 'v';
   }else{
      return 'm';
   }
}
int potez2(int dg, int gg){
   int sredina;
   char odg;
```

```
   if (gg == dg) {
      printf("\nTrazeni broj je %d", dg);
      return 1;
   }else{
      sredina = (gg + dg)/2;
      odg = student(sredina);
      printf("\n   Da li je %d - odgovor je %c", sredina, odg);
      if (odg == 'd'){
         printf("\n!! Trazeni broj je %d", sredina);
         return 1;
      }else if (odg == 'v'){
         return potez2(sredina + 1, gg);
      }else if (odg == 'm'){
         return potez2(dg, sredina - 1);
      }
   }
}
int main2(){
   int i;

   for(i=1;i<=10;i++){
      slucajniBroj = rand()%101;
      printf("\n\nZamisljen je %d\n", slucajniBroj);
      potez2(0, 100);
   }
   system("pause");
}

/*****************************************************************************/
int potez3(int dg, int gg){
   int sredina;
   char odg;
   if (gg == dg) {
      // printf("\nTrazeni broj je %d", dg);
      return 1;
   }else{
      sredina = (gg + dg)/2;
      odg = student(sredina);
      // printf("\n   Da li je %d - odgovor je %c", sredina, odg);
      if (odg == 'd'){
         // printf("\n!! Trazeni broj je %d", sredina);
         return 1;
      }else if (odg == 'v'){
         return 1 + potez3(sredina + 1, gg);
      }else if (odg == 'm'){
         return 1 + potez3(dg, sredina - 1);
      }
   }

}
/*****************************************************************************/
int main3(){
   int i, j, suma = 0, gg;



   for (j=1; j<=6;j++){
      suma = 0;
      gg = (int) pow(10, j);
      for(i=1;i<=100;i++){
         slucajniBroj = rand()%(gg+1);
         // printf("\n\nZamisljen je %d\n", slucajniBroj);
         suma += potez3(0, gg);
      }
      printf("\nProsjecan broj pokusaja za gg=%d jest:%f", gg, suma / 100.);
   }

   system("pause");
   return 0;
}
```

```
/*****************************************************************************/
int main4(){
   int i, j, suma = 0, gg;



   for (j=1; j<=40;j++){
      suma = 0;
      gg = (int)pow(2, j);
      for(i=1;i<=100;i++){
         slucajniBroj = rand()%(gg+1);
         // printf("\n\nZamisljen je %d\n", slucajniBroj);
         suma += potez3(0, gg);
      }
      printf("\nProsjecan broj pokusaja za gg=%d jest:%f", gg, suma / 100.);
   }
   system("pause");
   return 0;
}
```

PoljeJePokazivac.c

```
-------------------------------------------------------------------------
#include <stdio.h>
void f(int *x) {                        // ili   void f (int x[]) {
  printf ("%d %d\n", *x, x[0]);
  ++x;                                     // prvi clan postaje nulti
  printf ("%d %d %d\n", *x, x[0], *(x-1));
}
int main (void) {
  int x[4] = {1,2,3,4};
  printf ("%d %d\n", *x, *(x+1));
  f(x);
  return 0;
}
```

Premetaljka.c

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

void rotiraj(size_t duljina, char *niz) {
  char sacuvaj;
  sacuvaj = *niz;
  while(--duljina) {
    *niz=*(niz+1);
    ++niz;
    }
  *niz = sacuvaj;
}

void permutiraj(size_t duljina, char *niz, unsigned dubina) {
  if (duljina == 0) printf("\n# %s\t",niz-dubina);
  else {
    size_t brojac;
    for (brojac = duljina ; brojac > 0; --brojac) {
    printf("%s ", niz);
      permutiraj(duljina-1,niz+1,dubina+1);
    printf("%s ", niz);
      rotiraj(duljina,niz);
    printf("%s ", niz);
    }
  }
}

int main() {
  char izvorno[30];
  printf("Upisite rijec:\n");
  gets(izvorno);
  printf("\nPermutiram rijec \"%s\"\n",izvorno);
```

```
  permutiraj(strlen(izvorno),izvorno,0);
  return EXIT_SUCCESS;
}
```

PremjestiStog.c

```
#include <stdio.h>

#define MAXSTOG1 10 // maksimalna velicina stoga
#define MAXSTOG2 10 // maksimalna velicina stoga
#define ISTO 1

int stavi (int stavka, int stog[], int n, int *vrh) {
  if (*vrh >= n-1) return 0;
  (*vrh)++;
  stog[*vrh] = stavka;
  return 1;
}
int skini (int *stavka, int Stog[], int *vrh) {
  if (*vrh < 0) return 0; // stog je prazan
  *stavka = Stog[*vrh];
  (*vrh)--;
  return 1;
}
void premjesti(int stog1[],int n1,int *vrh1,int stog2[],int n2, int *vrh2 ) {
  int element;
  if (skini(&element,stog1, vrh1)) {
#if ISTO
    premjesti (stog1,n1,vrh1, stog2,n2,vrh2);
    stavi(element,stog2,n2,vrh2);
#else
    stavi(element,stog2,n2,vrh2);
    premjesti (stog1,n1,vrh1, stog2,n2,vrh2);
#endif
  }
}

int main(){
  int stog1[MAXSTOG1];
  int stog2[MAXSTOG2];
  int vrh1=-1, vrh2=-1;
  int i;
  for (i=0; i<10; i++) stavi(i,stog1,MAXSTOG1, &vrh1);
  printf ("\nPrvi stog\n");
  for (i=0; i<=vrh1;i++) printf("%d\n", stog1[i]);

  premjesti(stog1, MAXSTOG1, &vrh1, stog2, MAXSTOG2, &vrh2);

  printf ("\nDrugi stog nakon premijestanja\n");
  for (i=0; i<=vrh2;i++) printf("%d\n", stog2[i]);
  return 0;}
```

PrimjeriRekurzijeOriginal.c

```
// PrimjeriRekurzije.c
#include <stdio.h>

void pisi1 (int broj, int n) {
  broj++;
  if (broj > n) return;
  pisi1 (broj, n);
  printf( "%d", broj);
}

void pisi2 (int broj, int n) {
  broj++;
  if (broj > n) return;
  printf( "%d", broj);
  pisi2 (broj, n);
}
```

```c
void pisi3 (int *broj, int n) {
  (*broj)++;
  if (*broj > n) return;
  pisi3 (broj, n);
  printf( "%d", *broj);
}

void pisi4 (int *broj, int n) {
  (*broj)++;
  if (*broj > n) return;
  printf( "%d", *broj);
  pisi4 (broj, n);
}

void pisi5 (int *broj, int n) {
  (*broj)--;
  if (*broj < 0) return;
  pisi5 (broj, n);
  printf( "%d", *broj);
}

int main (void) {
  int nula;

       nula = 0; pisi1 (nula, 5);
       printf(" Nakon pisi1 nula = %d\n", nula);

       pisi2 (nula, 5);
       printf(" Nakon pisi2 nula = %d\n", nula);

    nula = 0; pisi3 (&nula, 5);
       printf(" Nakon pisi3 nula = %d\n", nula);

       nula = 0; pisi4 (&nula, 5);
       printf(" Nakon pisi4 nula = %d\n", nula);

       pisi5 (&nula, 5);
       printf(" Nakon pisi5 nula = %d\n", nula);

       return 0;
}
/*
a)  Kakav tip podatka sadrži *broj?
b)  Kakav tip podatka sadrži broj?
c)  Kakav tip podatka sadrži &nula?
d)  Koju vrijednost sadrži nula nakon povratka iz funkcije pisi1?
e)  Što će program ispisati na zaslonu računala?
*/
```

PrimjerZaStruct.c

```c
#include <stdlib.h>
#include <malloc.h>
#include <string.h>

#define MAXBROJ 13
#define MAXNAZIV 40

typedef struct s_osoba {   // krace: typedef struct { ... } osoba;
  char jmbg [MAXBROJ + 1];
  char prezime [MAXNAZIV + 1];
  char ime [MAXNAZIV + 1];
  int visina;
  float tezina;
} osoba;

typedef struct s_adresa {
       char ulica [MAXNAZIV + 1];
       int kbr;
```

```
	int post_broj;
	char mjesto [MAXNAZIV + 1];
} adresa;

typedef struct s_student {
	int maticni_broj;
	osoba osobni_podaci;
	adresa adresa_roditelja;
	adresa adresa_u_mjestu_studiranja;
	osoba otac;
	osoba majka;
} student;


int main (void) {
	student pero, *p;

	// referenciranje elemenata strukture
	strcpy (pero.osobni_podaci.ime, "Pero");
	pero.otac.visina = 182;
	pero.majka.tezina = 92.5;

	// referenciranje preko pokazivaca
	p = (student *) malloc (sizeof (student));
	strcpy (p->osobni_podaci.ime, "Ivo");
	p->otac.tezina = 82.8;            // p je pokazivac na strukturu
	p->majka.visina = 152;
	(*p).majka.tezina = 55.5;  // (*p) je struktura

	return 0;
}
```

PrioritetniRed.c

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

#define MAXPRIOR 100
typedef int tip;

void podesi (tip A[], int i, int n) {
	// potpuna binarna stabla s korijenima A[2 * i] i A[2 * i + 1]
	// kombiniraju se s A[i] formirajuci jedinstvenu gomilu
	// 1 <= i <= n
	int j;
	tip stavka;
	j = 2 * i; stavka = A[i];
	while (j <= n) {
		// Usporedi lijevo i desno dijete (ako ga ima)
		if ((j < n) && (A[j] < A[j + 1])) j++;
		// j pokazuje na vece dijete
		if (stavka >= A[j]) break; // stavka je na dobrom mjestu
		A[j / 2] = A[j]; // vece dijete podigni za razinu
		j *= 2;
	}
	A[j / 2] = stavka;
}
void ubaci (tip A[], int k) {
	// ubacuje vrijednost iz A[k] na gomilu pohranjenu u A[1 : k - 1]
	int i;
	tip novi;
	i = k / 2;
	novi = A[k];
	while ((i > 0) && (A[i] < novi)) {
		A[k] = A[i]; // spusti roditelja na vecu razinu
		k = i;
		i /= 2;          // roditelj od A[i] je na A[i/2]
	}
```

```
	A[k] = novi;
}
tip skini (tip A[], int *k) {
	// izbacuje vrijednost iz A[k] sa prvog mjesta
	// ako je red prazan vraca -1
	tip retVal = -1;
	if (*k <= 1) return retVal;
	retVal = A[1];
	(*k) --;
	A[1] = A[*k];
	podesi (A, 1, *k);
	return retVal;
}
int main() {
	int prior, i, j, k = 1;
	tip A[MAXPRIOR];
	srand((unsigned) time(NULL));
	while(1) {
		if (rand() % 2) {
			if (k >= MAXPRIOR)
				printf("Red prioriteta pun!\n");
			else {
				printf("Dodavanje u red prioriteta: %d\n",
				prior=(int)(rand()/(RAND_MAX + 1.) * 99 + 1));
				A[k] = prior;
				ubaci(A, k);
				k++;
			}
		} else {
			if ((prior = skini(A, &k)) == -1)
				printf("Red prioriteta prazan!\n");
			else
				printf("Skidanje iz reda prioriteta: %d\n", prior);
		}
		for (i = 1, j = 1; i < k; j++) {
			for (; i <= pow (2, j) - 1 && i < k; i++) {
				printf(" %d ", A[i]);
			}
			printf ("\n");
		}
		printf("Jos (d/n)? ");
		if (getchar() == 'n') return 0;
	}
}
```

ProvjeriXML.c

```
-------------------------------------------------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>
#include <math.h>
#include <string.h>
/* Ispisuje zadani tag i izbacuje atribute iz taga, ako ih je bilo. */
void ispisiSkratiTag(char *tag, unsigned int dubina){
   unsigned int i;
   printf("\n");
   for(i=0; i<dubina; i++)
      printf(" ");
   printf("<");
   for(i=0; i<strlen(tag); i++){
      if (tag[i] == ' '){
         printf(" %s", &tag[i]);
         tag[i] = 0;
         break;
      }else{
         putch(tag[i]);
      }
   }
   printf(">");
```

```
}

/* Provjerava da li je zatvoren zadani tag. */
int xmlOKrek(FILE *f, char *tag, unsigned int dubina){
   char ctag[500] = {0};

   do {
      if (fscanf(f, "%*[^<]%*c%[^>]", ctag) != 1)
         return 0;
      if (ctag[strlen(ctag)-1] == '/'){
         ispisiSkratiTag(ctag, dubina);
      }else if (*ctag  != '/'){
         ispisiSkratiTag(ctag, dubina);
         if (xmlOKrek(f, ctag, dubina+1) == 0)
            return 0;
      }else{
         break;
      }
   } while (1);


   if (*ctag  == '/' && (strcmp(tag, ctag+1) == 0)){
      ispisiSkratiTag(ctag, dubina);
      return 1;
   }else{
      printf("\nNe valja zavrsni tag %s", ctag);
      return 0;
   }
}


/* Cita korjenski tag i poziva rekurzivnu funkciju */
int xmlOK(FILE *f){
   char tag[500] = {0};
   char *t;
   if (fscanf(f, "%100[^>]", tag) != 1)
      return 0;
   else{
      if ((t = strchr(tag, '<')) == NULL) return 0;
      ispisiSkratiTag(++t, 1);
      return xmlOKrek(f,t,1);
   }

}

int main(){
   FILE *f;
   f = fopen("d:\\temp\\recipee.xml", "r");
   if (f == NULL) {
      printf("\nPogreska kod otvaranja datoteke");
   } else {
      printf("\nXML je %s \n", (xmlOK(f) == 1) ? "OK" : "!OK" );
   }
   return 0;
}
```

RedListomPogresan.c

```
// RedListom.c
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
struct at {
       int element;
       struct at *sljed;
};
typedef struct at atom;

// dodaje element u red, vraca 1 ako uspije, inace 0
int DodajURed (int element, atom *ulaz, atom *izlaz) {
```

```
	atom *novi;
	if (novi = malloc (sizeof (atom))) {
		novi->element = element;
		novi->sljed = NULL;
		if (izlaz == NULL) {
			izlaz = novi;		// ako je red bio prazan
		} else {
			(ulaz)->sljed = novi;	// inace, stavi na kraj
		}
		ulaz = novi;			// zapamti zadnjeg
		return 1;
	}
	return 0;
}
// uklanja element iz reda, vraca 1 ako uspije, inace 0
int SkiniIzReda (int *element, atom *ulaz, atom *izlaz) {
	atom *stari;
	if (izlaz) {					// ako red nije prazan
		*element = (izlaz)->element;	// element koji se skida
		stari = izlaz;				// zapamti trenutni izlaz
		izlaz = (izlaz)->sljed;		// novi izlaz
		free (stari);				// oslobodi memoriju skinutog
		if (izlaz == NULL) ulaz = NULL; // prazan red
		return 1;
  }
	return 0;
}
// vraca broj elemenata u redu
int Prebroji (atom *izlaz) {
	int n;
#if 1
	for (n = 0; izlaz; izlaz = izlaz->sljed) {
		printf ("%d -> ", izlaz->element);
		n++ ;
	}
	printf ("NULL\n");
#else
	// krace
	for (n = 0; izlaz; n++, izlaz = izlaz->sljed);
#endif
	return n;
}
int main () {
  int broj;			// podatak/kontrola
	atom *ulaz = NULL;  // krajevi
	atom *izlaz = NULL;

	printf ("Slucajno se generiraju nenegativni cijeli brojevi.\n");
	printf ("Neparni brojevi upisuju se u red, a parni broj simulira skidanje iz reda\n");
	printf ("Za obavljanje jednog koraka pritisnuti ENTER, za kraj CTRL-C\n");

	// inicijalizacija generatora slucajnih brojeva
	srand ((unsigned) time (NULL));

	while (1) {
		getchar ();			// ENTER, Ctrl-C
		broj = rand ();
		if (broj%2) {
			// Neparne upisujemo u red
			printf ("U red se upisuje broj %d\n", broj);
			if (!DodajURed (broj, ulaz, izlaz))
				printf("Nema vise memorije\n");
		} else {
			// Parni broj simulira skidanje iz reda
			if (SkiniIzReda (&broj, ulaz, izlaz)) {
				printf ("Iz reda je skinut podatak %d\n", broj);
			} else {
				printf("Red je prazan\n");
			}
		}
```

```
                printf ("Broj elemenata u redu: %d\n", Prebroji (izlaz));
        }
}
```

RedOboje.c

```
-------------------------------------------------------------------------
#include <iostream>
#include <ctime>
using namespace std;

// Red poljem
#include <stdlib.h>
#include <stdio.h>

typedef int tip;

class RedP {
private:
        tip *_red;
        int _MAXRED;
        int _ulaz;
        int _izlaz;
public:
        RedP () {
                _red = new tip [10];
                _MAXRED = 10;
                _ulaz = _izlaz = 0;
        }
        RedP (int n) {
                _red = new tip [n];
                _MAXRED = n;
                _ulaz = _izlaz = 0;
        }
        void DodajURed (tip element) {
                if (((_ulaz+1) % _MAXRED) == _izlaz)
              throw "Red je pun!";
                _ulaz++;
                _ulaz %= _MAXRED;
                _red [_ulaz] = element;
        }
        tip SkiniIzReda () {
                if (_ulaz == _izlaz)
           throw "Red je prazan!";
                _izlaz ++;
                _izlaz %= _MAXRED;
                return _red[_izlaz];
        }
        int Prebroji () {
                if (_ulaz >= _izlaz) {
                        return (_ulaz - _izlaz);          // standardno
                } else {
                        return (_ulaz - _izlaz + _MAXRED);// cirkularnost
                }
        }
        ~RedP () {
                delete []_red;
                _ulaz = _izlaz = 0;
        }
};
// Red listom
class RedL {
        private:
                struct atom {
                        tip _element;
                        struct atom *_sljed;
                };
                atom *_ulaz;
                atom *_izlaz;
public:
        RedL () {
```

```
		_ulaz = _izlaz = NULL;
	}
	void DodajURed (int element) {
		atom *novi = new atom;
		if (!novi) throw "Nema memorije";
		novi->_element = element;
		novi->_sljed = NULL;
		if (_izlaz == NULL) {
			_izlaz = novi;			// ako je red bio prazan
		} else {
			_ulaz->_sljed = novi;		// inace, stavi na kraj
		}
		_ulaz = novi;			// zapamti zadnjeg
	}
	tip SkiniIzReda () {
		atom *stari;
		tip element;
		if (_izlaz != NULL) {						// ako red nije prazan
			element = _izlaz->_element;	// element koji se skida
			stari = _izlaz;					// zapamti trenutni izlaz
			_izlaz = _izlaz->_sljed;			// novi izlaz
			delete stari;					// oslobodi memoriju skinutog
			if (_izlaz == NULL) _ulaz = NULL; // ispraznjeni red
			return element;
		} else {
			throw "Red je prazan";
		}
	}
	int Prebroji () {
		int n;
		atom *izlaz = _izlaz;
		for (n = 0; izlaz != NULL; n++, izlaz = izlaz->_sljed);
		return n;
	}
	~RedL () {
		Brisi (_izlaz);
}
	void Brisi (atom *lokalniIzlaz) {
		if (lokalniIzlaz != NULL) {
			Brisi (lokalniIzlaz->_sljed);
			cout << "Brisem " << lokalniIzlaz->_element << '\n';
			free (lokalniIzlaz);
		}
		_ulaz = NULL;
	}
};
int main () {
#if 0
	RedP *red = new RedP;
#else
	RedL *red = new RedL;
#endif
	tip element;  // element, krajevi reda
	FILE *fi;					// ulazna datoteka
	// inicijalizacija
	fi = fopen ("UlazZaRed.txt", "r");
	if (fi) {
		while (fscanf (fi, "%d", &element) != EOF) {
			// stavljanje u red
			try {
				red->DodajURed (element);
				cout << "U red je dodan element " << element << '\n';
				cout << "Broj elemenata u redu je " << red->Prebroji () << '\n';
			}
			catch (char *poruka) {
				cout << poruka << '\n';
			}
		}
		// uklanjanje iz reda
	/*	while (red->Prebroji () > 0) {
```

```
                        try {
                                cout << "Iz reda skinut element " << red->SkiniIzReda () << '\n';
                                cout << "Broj elemenata u redu je " << red->Prebroji () << '\n';
                        }
                        catch (char *poruka) {
                                cout << poruka << '\n';
                        }
                }
                try {
                        red->SkiniIzReda (); // testiranje javljanja pogreske za prazni red
                }
                catch (char *poruka) {
                                cout << poruka << '\n';
                }
                fclose (fi);*/
                delete red;
                return 0;
        } else {
                printf ("Nema ulazne datoteke\n");
                return 1;
        }
}
```

SlozenoStablo.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct podaci {
  long matBroj;
  char ime[25];
  char prezime[25];
  short godRod;
};
typedef struct podaci Podaci;

struct cvor {
  Podaci * student;
  struct cvor * lijevo;
  struct cvor * desno;
};
typedef struct cvor Cvor;

// funkcija dodaje u binarno sortirano stablo
// kljuc je prezime
Cvor * dodajPrezime (Cvor * korijen, Podaci * student) {
  int smjer;
  if (korijen == NULL) {
    korijen = (Cvor *) malloc (sizeof(Cvor));
    if (korijen) {
      korijen->student = student;
      korijen->lijevo = korijen->desno = NULL;
    }
  } else if ((smjer = strcmp(student->prezime, korijen->student->prezime)) < 0)
            korijen->lijevo = dodajPrezime (korijen->lijevo, student);
    else if (smjer > 0)
            korijen->desno = dodajPrezime (korijen->desno, student);
    else
            printf("Podatak %s vec postoji!\n", student->prezime);
  return korijen;
}
// funkcija dodaje u binarno sortirano stablo
// kljuc je ime
Cvor * dodajIme (Cvor * korijen, Podaci * student) {
  int smjer;
  if (korijen == NULL) {
    korijen = (Cvor *) malloc (sizeof(Cvor));
    if (korijen) {
      korijen->student = student;
      korijen->lijevo = korijen->desno = NULL;
```

```
    }
  } else if ((smjer = strcmp(student->ime, korijen->student->ime)) < 0)
            korijen->lijevo = dodajIme (korijen->lijevo, student);
          else if (smjer > 0)
                  korijen->desno = dodajIme (korijen->desno, student);
               else
                  printf("Podatak %s vec postoji!\n", student->ime);
  return korijen;
}

// funkcija pretrazuje binarno sortirano stablo po kljucu prezime
Cvor * potraziPrezime (Cvor * korijen, char * prezime) {
  int smjer;
  if (korijen) {
    if ((smjer = strcmp(prezime, korijen->student->prezime)) < 0)
       return potraziPrezime (korijen->lijevo, prezime);
    else if (smjer > 0)
       return potraziPrezime (korijen->desno, prezime);
  }
  return korijen;
}

// funkcija inorder ispisuje zadano binarno stablo
void inOrder (Cvor * korijen) {
  if (korijen) {
    inOrder (korijen->lijevo);
    printf("%s, %s;", korijen->student->prezime, korijen->student->ime);
    inOrder (korijen->desno);
  }
}

// funkcija ispisuje sve cvorove zadane dubine
// poziv ispis(korijen, 1, n)
// gdje je n trazena dubina
void ispisi (Cvor * korijen, int trenutnaDubina, int trazenaDubina) {
  if (korijen) {
    if (trenutnaDubina == trazenaDubina)
      printf("%s %s\n", korijen->student->ime, korijen->student->prezime);
    ispisi (korijen->lijevo, trenutnaDubina + 1, trazenaDubina);
    ispisi (korijen->desno, trenutnaDubina + 1, trazenaDubina);
  }
}

// funkcija ispisuje sve cvorove na zadanoj dubini a ispred
// podataka stavlja odgovarajuci broj praznina
// poziv ispis2(korijen, 1, n)
// gdje je n trazena dubina
void ispisi2 (Cvor * korijen, int trenutnaDubina, int trazenaDubina) {
  int i;
  if (korijen) {
    if (trenutnaDubina == trazenaDubina) {
      for (i=0; i<trazenaDubina; i++) printf(" ");
      printf("%s %s\n", korijen->student->ime, korijen->student->prezime);
    }
    ispisi2 (korijen->lijevo, trenutnaDubina + 1, trazenaDubina);
    ispisi2 (korijen->desno, trenutnaDubina + 1, trazenaDubina);
  }
}

// funkcija trazi listove s najvecom i najmanjom razinom
// poziv balansirano (korijen, 1, &maxDub, &minDub);
void balansirano (Cvor * korijen, int trenDub,
                                    int * maxDub, int * minDub) {
  if (korijen) {
    if (!korijen->lijevo & !korijen->desno) {
      if (*maxDub == 0 || trenDub > *maxDub)
        *maxDub = trenDub;
      if (*minDub == 0 || trenDub < *minDub)
        *minDub = trenDub;
       } else {
```

```
      balansirano (korijen->lijevo, trenDub + 1, maxDub, minDub);
      balansirano (korijen->desno, trenDub + 1, maxDub, minDub);
    }
  }
}

// funkcija za zadano binarno stablo ispisuje listove
void ispisiListove(Cvor * korijen) {
  if (korijen) {
    if (!korijen->lijevo & !korijen->desno)
      printf("%s %s; ", korijen->student->ime, korijen->student->prezime);
    ispisiListove(korijen->lijevo);
    ispisiListove(korijen->desno);
  }
}
// funkcija u zadanom binarnom stablu zbraja broj cvorova (argument
// funkcije) i ukupne godine starosti podataka u stablu (podatak se
// prenosi preko imena funkcije)
int zbroj(Cvor * korijen, int * broj) {
  if (korijen) {
    (*broj)++;
    return 2007 - korijen->student->godRod +
         zbroj(korijen->lijevo, broj) +
         zbroj(korijen->desno, broj);
  }
  return 0;
}
// funkcija prebroji sve cvorove stabla
int prebroji(Cvor * korijen) {
  if (korijen)
    return 1 + prebroji(korijen->lijevo) + prebroji(korijen->desno);
  else
    return 0;
}
int main () {
  FILE * fUl;
  char buf[256];
  Podaci * student;
  Cvor * korijenPrezime = NULL;
  Cvor * korijenIme = NULL;
  Cvor * trazeni;
  int ukupno, broj = 0;
  int minDubina = 0, maxDubina = 0;
  int i;
  if ((fUl = fopen("stud.txt", "r")) == NULL) {
    fprintf(stderr, "Ne mogu otvoriti 'stud.txt'\n");
    return 1;
  }
  while (fgets(buf, 256, fUl)) {
    student = (Podaci *) malloc(sizeof(Podaci));
    sscanf(buf, "%ld;%[^;];%[^;];%d", &(student->matBroj), student->prezime,
      student->ime, &(student->godRod));
    korijenPrezime = dodajPrezime(korijenPrezime, student);
    korijenIme = dodajIme(korijenIme, student);
  }
  fclose(fUl);
  inOrder(korijenPrezime);
  printf("\n");
  if(trazeni = potraziPrezime(korijenPrezime, "Maric"))
    printf("Pronasao: %s, %s %ld %d\n", trazeni->student->prezime,
      trazeni->student->ime, trazeni->student->matBroj, trazeni->student->godRod);
  inOrder(korijenIme);
  printf("\nPrezime, listovi:\n");
  ispisiListove(korijenPrezime);
  printf("\nIme, listovi:\n");
  ispisiListove(korijenIme);
  ukupno = zbroj(korijenPrezime, &broj);
  printf ("\nZbroj godina=%d Podataka u listi=%d prosjek=%5.2f\n",
    ukupno, broj, (float)ukupno / broj);
  broj = 0;
```

```
  ukupno = zbroj(korijenIme, &broj);
  printf ("\nZbroj godina=%d Podataka u listi=%d prosjek=%5.2f\n",
    ukupno, broj, (float)ukupno / broj);
  printf("Prebroji po prezimenu=%d\n", prebroji(korijenPrezime));
  printf("Prebroji po imenu=%d\n", prebroji(korijenIme));
  balansirano (korijenPrezime, 1, &maxDubina, &minDubina);
  printf("MaxDubina=%d  MinDubina=%d Balansirano=%s\n", maxDubina, minDubina,
    maxDubina - minDubina > 1 ? "false":"true");
  for (i=1; i<=maxDubina; i++) {
    printf("%d razina:\n", i);
    ispisi2(korijenPrezime, 1, i);
  }
  maxDubina = minDubina = 0;
  balansirano (korijenIme, 1, &maxDubina, &minDubina);
  printf("MaxDubina=%d  MinDubina=%d Balansirano=%s\n", maxDubina, minDubina,
    maxDubina - minDubina > 1 ? "false":"true");
  for (i=1; i<=maxDubina; i++) {
    printf("%d razina:\n", i);
    ispisi2(korijenIme, 1, i);
  }
}
```

SortiranoStablo.c

```
// SortiranoStablo.c
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string.h>

struct cv {
  char element[15];
  struct cv *lijevo;
  struct cv *desno;
};
typedef struct cv cvor;

// upisuje u stablo podatke: lijevo manji, desno veci
cvor *upis (cvor *korijen, char element[]) {
       int smjer; // odluka o podstablu
       if (korijen == NULL) { // prazno (pod)stablo
              korijen = (cvor *) malloc (sizeof (cvor));
              if (korijen) {
                     strcpy (korijen->element, element);
                     korijen->lijevo = korijen->desno = NULL;
              } else {
                     printf ("U memoriji mena mjesta za upisati '%s'\n", element);
              }
       } else if ((smjer = strcmp (element, korijen->element)) < 0) {
              korijen->lijevo = upis (korijen->lijevo, element);
       } else if (smjer > 0) {
              korijen->desno  = upis (korijen->desno, element);
       } else {
              printf ("Podatak '%s' vec postoji!\n", element);
       }
       return korijen; // pokazivac na zadnji element
}
// obilazak inorder lijevo-desno
void ispisinld (cvor *korijen) {
       if (korijen != NULL) {
              ispisinld (korijen->lijevo);
              printf ("%s \n", korijen->element);
              ispisinld (korijen->desno);
       }
}
// obilazak inorder desno-lijevo
void ispisindl (cvor *korijen) {
       if (korijen != NULL) {
```

```
                ispisindl (korijen->desno);
                printf ("%s \n", korijen->element);
                ispisindl (korijen->lijevo);
        }
}
// obilazak preorder
void ispispre (cvor *korijen) {
        if (korijen != NULL) {
                printf ("%s \n", korijen->element);
                ispispre (korijen->lijevo);
                ispispre (korijen->desno);
        }
}
// obilazak postorder
void ispispost (cvor *korijen) {
        if (korijen != NULL) {
                ispispost (korijen->lijevo);
                ispispost (korijen->desno);
                printf ("%s \n", korijen->element);
        }
}
// ispis stabla
void ispissta (cvor *korijen, int nivo) {
        int i;
        if (korijen != NULL) {
                ispissta (korijen->desno, nivo+1);
                for (i = 0; i < nivo; i++) printf("    ");
                printf ("%s \n", korijen->element);
                ispissta (korijen->lijevo, nivo+1);
        }
}
// trazenje cvora u binarnom stablu
cvor *trazi (cvor *korijen, char element[]) {
        int smjer;
        if (korijen) {
                if ((smjer = strcmp (element, korijen->element)) < 0) {
                        return trazi (korijen->lijevo, element);
                } else if (smjer > 0) {
                        return trazi (korijen->desno, element);
                }
        }
        return korijen; // ili je pronadjen ili NULL;
}
int main() {
        FILE *fi;                                   // ulazna datoteka
        int j;                                      // brojac podataka
        cvor *korijen, *p;  // pokazivac na korijen, pomocni pokazivac
        char ime[15];
        fi = fopen ("UlazZaSortiranoStablo.txt", "r");
        if (fi) {
                // inicijalizacija i citanje podataka
                j = 1;
                korijen = NULL;
                while (fscanf (fi, "%s", &ime) != EOF) {
                        printf ("%d. ulazni podatak je %s \n", j++, ime);
                        korijen = upis (korijen, ime);
                }
                fclose (fi);
                // obilazak i ispis stabla
         getchar ();
                printf ("Ispis inorder lijevo-desno\n");
                ispisinld (korijen);
                getchar ();
                printf ("Ispis inorder desno-lijevo\n");
                ispisindl (korijen);
                getchar ();
                printf ("Ispis preorder\n");
                ispispre (korijen);
                getchar ();
                printf ("Ispis postorder\n");
```

```
                ispispost (korijen);
                getchar ();
                printf ("Ispis stabla\n");
                ispissta (korijen, 0);
                // trazenje elementa
                while (1) {
                        printf ("Unesite element koji trazite, ili KRAJ >");
                        scanf ("%s", ime);
                        if (stricmp (ime, "KRAJ") == 0) break;
                        p = trazi (korijen, ime);
                        if (p) {
                                printf ("Pronadjen je element: %s\n", p->element);
                        } else {
                                printf ("Nije pronadjen element: %s\n", ime);
                        }
                }
        } else {
                printf ("Nema ulaznih podataka\n");
                return 1;
        }
        return 0;
}
```

StabloOO.c

```
-------------------------------------------------------------------------
#include <iostream>
#include <ctime>
using namespace std;

class Stablo {

private:
  struct cvor {
    char element[15];
    cvor *lijevo;
    cvor *desno;
  };
  cvor *_glava;

  void Dodaj(cvor **cv, char *element);
  bool Trazi(cvor *cv, char *element);
  void Inorder(cvor *cv);
  void OslobodiMemoriju(cvor **cv);

public:
  Stablo();
  ~Stablo();
  void Dodaj(char *element);
  bool Trazi(char *element);
  void Inorder();

};

Stablo::Stablo()
{
  _glava = NULL;
}
Stablo::~Stablo()
{
  OslobodiMemoriju(&_glava);
}
void Stablo::OslobodiMemoriju(cvor **cv)
{
  if (*cv != NULL)
  {
    if ((*cv)->lijevo != NULL) OslobodiMemoriju(&(*cv)->lijevo);
    if ((*cv)->desno != NULL) OslobodiMemoriju(&(*cv)->desno);
    free (*cv);
  }
}
```

```
void Stablo::Dodaj(char* element) {
  Dodaj(&_glava, element);
}
void Stablo::Dodaj(cvor **cv, char* element) {
  if (*cv == NULL) {
    *cv = (cvor *) malloc (sizeof (cvor));
    if (*cv == NULL) throw "Nema dovoljno memorije!";
    else
    {
      strcpy((*cv)->element, element);
      (*cv)->lijevo = NULL;
      (*cv)->desno = NULL;
    }
  }
  else
  {
    if (strcmp(element, (*cv)->element) <= 0)
      Dodaj(&(*cv)->lijevo, element);
    else
      Dodaj(&(*cv)->desno, element);
  }
}
bool Stablo::Trazi(char* element) {
  return Trazi(_glava, element);
}
bool Stablo::Trazi(cvor *cv, char *element)
{
  if (cv == NULL) return false;
  else
  {
    if (strcmp(element, cv->element) < 0)
      return Trazi(cv->lijevo, element);
    else if (strcmp(element, cv->element) > 0)
      return Trazi(cv->desno, element);
  }
  return true;
}
void Stablo::Inorder()
{
  Inorder(_glava);
}
void Stablo::Inorder(cvor *cv)
{
  if (cv != NULL) {
    Inorder(cv->lijevo);
    printf("%s \n", cv->element);
    Inorder(cv->desno);
  }
}
int main()
{

  try
  {
    Stablo St;
    St.Dodaj("Ivana");
    St.Dodaj("Marin");
    St.Dodaj("Tomislav");
    St.Dodaj("Sonja");

    printf("Inorder:\n");
    St.Inorder();
    printf("Trazi element Ivana: %d\n", St.Trazi("Ivana"));
    printf("Trazi element Ana: %d\n", St.Trazi("Ana"));

  }
  catch(char *ex)
  {
    printf("%s\n", ex);
  }
```

```
  return 0;
}
```

StogIspiti.c

```
#include <stdio.h>
#define MAXSTOG 100
struct s {
  int  sifraIspit;
  long sifraStudent;
  int  ocjena;
};
typedef struct s zapis;

int dodaj(zapis stavka, zapis stog[], int n,
int *vrh) {
       // ako je stog pun
       if (*vrh >= n-1) return 0;
       // ako ima mjesta na stogu
       (*vrh)++;
       stog [*vrh] = stavka;
return 1;
}
int skini (zapis *stavka, zapis stog[], int *vrh) {
   // ako je stog prazan
   if (*vrh < 0) return 0;
   // ako ima zapisa na stogu
   *stavka = stog[*vrh];
       (*vrh)--;
       return 1;
}
int main() {
   zapis z, stog[MAXSTOG], pomStog[MAXSTOG];
   int i = 0, vrh = -1, pomVrh = -1;
   // ucitavanje elemenata stoga
   do {
         printf("\nUpisite sifru ispita, sifru studenta i ocjenu>");
      scanf("%d %ld %d", &z.sifraIspit,
                         &z.sifraStudent, &z.ocjena);
         // prekini ucitavanje ako je za ocjenu ucitana 0 ili ako je stog pun
   } while (z.ocjena && dodaj (z, stog, MAXSTOG, &vrh));
   // brisanje zapisa s neprolaznim ocjenama koristenjem pomocnog stoga:
   // 1. premjesti sve zapise s prolaznom ocjenom na pomocni stog
   while (skini(&z, stog, &vrh)) {
      if (z.ocjena > 1)
         dodaj(z, pomStog, MAXSTOG, &pomVrh);
   }
   // 2. premjesti nazad na pocetni stog zapise s pomocnog stoga
   while (skini(&z, pomStog, &pomVrh)) {
      dodaj(z, stog, MAXSTOG, &vrh);
      // kontrolni ispis
      printf("Sifra ispit, z.sifraStudent, z.ocjena = %d\n", z.sifraIspit, z.sifraStudent, z.ocjena);
   }
   return 0;
}
```

stogl.c

```
#include <iostream>
#include <ctime>
using namespace std;

// Stog listom
class Stog {
private:
       struct atom {
```

```
            int element;
            struct atom *sljed;
      };
      atom *_vrh;
      void Brisi (atom *lokalniVrh);
public:
      Stog();
      void Dodaj (int stavka);
      void Skini (int& stavka);
  void Ispisi ();
      ~Stog ();
};
Stog::Stog () {
      _vrh = NULL;
}
void Stog::Dodaj (int stavka) {
      atom *novi; // pokazivac na novi atom
      novi = new atom;
      if (novi == NULL)throw "Nema memorije!";
      novi->element = stavka;
      novi->sljed = _vrh;
      _vrh = novi;  // spremi pokazivac na novi cvor
      cout << " na adresu " << _vrh;
}
void Stog::Skini (int& stavka) {
      atom *pom;                        // pomocni pokazivac
      if (!_vrh) throw "Stog je prazan!";
      stavka = _vrh->element;
      cout << "S adrese " << _vrh;
      pom = _vrh->sljed;      // sacuvaj novi vrh
      free (_vrh);               // oslobodi vrh
      _vrh = pom;                       // vrati novi vrh
}
Stog::~Stog () {
  Brisi (_vrh);
}
void Stog::Brisi (atom *lokalniVrh) {
      if (lokalniVrh != NULL) {
            Brisi (lokalniVrh->sljed);
            cout << "Brisem s " << lokalniVrh << '\n';
            free (lokalniVrh);
      }
}
void Stog::Ispisi () {
  atom *a;
  if (_vrh == NULL) cout << "(prazan)";
  cout << '\n';
  for (a = _vrh; a != NULL; a = a->sljed)
    cout << a->element << " ";
}
```

StogMax.c

```
#include <stdio.h>
#define MAXSTOG 100
int dodaj(int stavka, int stog[], int n, int *vrh) {
      // ako je stog pun
      if (*vrh >= n-1) return 0;
      // ako ima mjesta na stogu
      (*vrh)++;
      stog [*vrh] = stavka;
      return 1;
}
int skini (int *stavka, int stog[], int *vrh) {
      // ako je stog prazan
      if (*vrh < 0) return 0;
      // ako ima zapisa na stogu
      *stavka = stog [*vrh];
      (*vrh)--;
      return 1;
```

```
}
int izbaciNajveci(int stog[],int pomStog[], int n, int *vrh) {
	int pomVrh = -1, elemMax, stavka;
	// prebaci element s vrha stoga na pomocni stog,
	// a ujedno ga oznaci kao inicijalno najveci element na stogu
	skini(&elemMax, stog, vrh);
	dodaj(elemMax, pomStog, n, &pomVrh);
	// nadji najveci element na stogu,
	// a pritom prebacuj sve stavke na pomocni stog
	while (skini(&stavka, stog, vrh)) {
		dodaj(stavka, pomStog, n, &pomVrh);
		if (stavka > elemMax) elemMax = stavka;
	}
	// skidaj s pomocnog stoga sve elemente
	// i sve, osim najveceg, premjestaj nazad na pocetni stog
	// (Ako ima vise najvecih, sve ce ih izbaciti!)
	while (skini(&stavka, pomStog, &pomVrh)) {
		if (stavka < elemMax) dodaj(stavka, stog, n, vrh);
	}
	return elemMax;
}

int main() {
  return 0;
}
```

StogNaDisku.c

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

struct s {
  long  maticniBroj;
  char  imePrezime[24+1];
  int   sifraPredmet;
  short ocjena;
};
typedef struct s zapis;

int fatal(char *poruka) {
  puts(poruka);
  exit(1);
}
int stavi(zapis *stavka, long *vrh, FILE *f) {
  // zapisi na kraj
  if (*vrh <= 0) {
    if (fseek(f, sizeof(*vrh), SEEK_SET)) return 0;
  } else {
    if (fseek(f, *vrh+sizeof(*stavka), SEEK_SET)) return 0;
    // da nema skini ...
    // if (fseek(f, 0l, SEEK_END)) return 0;
  }
  fwrite(stavka, sizeof(*stavka), 1, f);
  *vrh = ftell(f) - sizeof(zapis);
  return 1;
}

int skini(zapis *stavka, long *vrh, FILE *f) {
  // ako se ne moze pozicionirati (vrh < 0)
  // u datoteci, znaci da je stog prazan
  if (fseek(f, *vrh, SEEK_SET)) return 0;

  // procitaj zapis s kraja datoteke, tj. s vrha stoga
  fread(stavka, sizeof(*stavka), 1, f);
  *vrh -= sizeof(zapis);
  return 1;
}
```

```c
void stog () {
  FILE *f;
  long vrh = -1L;
  int i;
  zapis z;

  srand((unsigned) time(NULL));

  if ((f = fopen("stog.dat", "r+b")) == NULL)
    if ((f = fopen("stog.dat", "w+b")) == NULL)
      fatal("Datoteka se ne moze otvoriti!");
  // prvo procitaj adresu zadnjeg
  fread(&vrh, sizeof(vrh), 1, f);
  // isprazni stog i ispisi sve podatke
  while (skini(&z, &vrh, f)) {
    printf("%ld %s %d %d\n",
               z.maticniBroj, z.imePrezime,
               z.sifraPredmet, z.ocjena);
  }
  // nakon sto procitas sve zapise,
  // malo stavi malo skini
  strcpy (z.imePrezime, "Hrvoje Horvat");
  for (i = 1; i <= 20; ++i) {
    if (rand() % 2) {
      z.maticniBroj = rand() % 1024;
      z.ocjena = rand() % 5 + 1;
      z.sifraPredmet = rand() % 100;
      if (!stavi(&z, &vrh, f))
        printf("stavi %ld\n", z.maticniBroj);
    } else if (skini(&z, &vrh, f)) {
        printf("skini %ld\n", z.maticniBroj);
    } else {
        printf("stavi/skini neuspješan\n");
    }
  }

  // zapisi na pocetak datoteke novu adresu zadnjeg zapisa
  fseek(f, 0L, SEEK_SET); // ILI: rewind(f);
  fwrite(&vrh, sizeof(vrh), 1, f);
  fclose(f);
  return;
}

int main() {
  stog();
  return 0;
}
```

StogOboje.c

```c
#include <iostream>
#include <ctime>
using namespace std;

// Stog poljem
class StogP {
public:
       StogP ();
       StogP (int n);
       void Dodaj (int stavka);
       void Skini (int& stavka);
       void Isprazni ();
       int Prazan();
  void Ispisi();
       ~StogP ();
private:
       int *_stog;
       int _MAXSTOG;
       int _vrh;
```

```
};
StogP::StogP () {
//  StogP (5);
       _MAXSTOG = 5;
  _stog = new int [_MAXSTOG];
       _vrh = -1;
}
StogP::StogP (int n) {
       _MAXSTOG = n;
       _stog = new int [_MAXSTOG];
       _vrh = -1;
}
void StogP::Dodaj (int stavka) {
       if (_vrh >= _MAXSTOG - 1) throw "Stog je pun!";
       _vrh++;
       cout << " na adresu " << _vrh;
       _stog [_vrh] = stavka;
}
void StogP::Skini (int& stavka) {
       if (_vrh < 0) throw "Stog je prazan!";
       cout << "S adrese " << _vrh;
       stavka = _stog [_vrh];
       _vrh--;
}
void StogP::Isprazni () {
       _vrh = -1;
}
int StogP::Prazan() {
       return _vrh == -1;
}
StogP::~StogP () {
       delete [] _stog;
}
void StogP::Ispisi() {
  int a;
  if (_vrh == -1) cout << "(prazan)";
  cout << '\n';
  for (a = 0; a <= _vrh; ++a)
    cout << _stog[a] << " ";
}
// Stog listom
class StogL {
private:
       struct atom {
              int element;
              struct atom *sljed;
       };
       atom *_vrh;
       void Brisi (atom *lokalniVrh);
public:
       StogL();
       void Dodaj (int stavka);
       void Skini (int& stavka);
  void Ispisi ();
       ~StogL ();
};
StogL::StogL () {
       _vrh = NULL;
}
void StogL::Dodaj (int stavka) {
       atom *novi; // pokazivac na novi atom
       novi = new atom;
       if (novi == NULL)throw "Nema memorije!";
       novi->element = stavka;
       novi->sljed = _vrh;
       _vrh = novi;  // spremi pokazivac na novi cvor
       cout << " na adresu " << _vrh;
}
void StogL::Skini (int& stavka) {
       atom *pom;                        // pomocni pokazivac
```

```
        if (!_vrh) throw "Stog je prazan!";
        stavka = _vrh->element;
        cout << "S adrese " << _vrh;
        pom = _vrh->sljed;      // sacuvaj novi vrh
        free (_vrh);               // oslobodi vrh
        _vrh = pom;                        // vrati novi vrh
}
StogL::~StogL () {
  Brisi (_vrh);
}
void StogL::Brisi (atom *lokalniVrh) {
        if (lokalniVrh != NULL) {
                Brisi (lokalniVrh->sljed);
                cout << "Brisem s " << lokalniVrh << '\n';
                free (lokalniVrh);
        }
}
void StogL::Ispisi () {
  atom *a;
  if (_vrh == NULL) cout << "(prazan)";
  cout << '\n';
  for (a = _vrh; a != NULL; a = a->sljed)
    cout << a->element << " ";
}
int main () {
        int novi, stari;
#if 1                                   // Stog listom
        StogL *stog = new StogL ();
#else                                   // Stog poljem
        StogP *stog = new StogP();
#endif

        cout << "Slucajno se generiraju nenegativni cijeli brojevi.\n";
        cout << "Neparni brojevi upisuju se na stog\n";
        cout << "Parni broj simulira skidanje sa stoga\n";
        cout << "Za obavljanje jednog koraka pritisnuti ENTER, za kraj K\n\n";
        srand ((unsigned) time (NULL));
        while (1) {
    stog->Ispisi();
                putchar ('\n');
                if (toupper (getchar ()) == 'K') break;
                novi = rand ();
                if (novi%2) {       // Neparni se upisuju na stog
                        cout << "Dodaj " << novi;
                        try {
                                stog->Dodaj (novi);
                        }
                        catch (char *poruka)
                        {
                                cout << "Pogreska: " << poruka << '\n';
                        }
                } else {                    // Parni broj simulira skidanje sa stoga
                        try {
                                stog->Skini (stari);
                                cout << " skinut " << stari << '\n';
                        }
                        catch (char *poruka)
                        {
                                cout << "Pogreska: " << poruka << '\n';
                        }
                }

        }
        delete stog;
        return 0;
}
```

stogp.c

```
-------------------------------------------------------------------------
#include <iostream>
```

```
#include <ctime>
using namespace std;

// Stog poljem
class Stog {
public:
       Stog ();
       Stog (int n);
       void Dodaj (int stavka);
       void Skini (int& stavka);
       void Isprazni ();
  void Ispisi();
       ~Stog ();
private:
       int *_stog;
       int _MAXSTOG;
       int _vrh;
};
Stog::Stog () {
//  StogP (5);
       _MAXSTOG = 5;
  _stog = new int [_MAXSTOG];
       _vrh = -1;
}
Stog::Stog (int n) {
       _MAXSTOG = n;
       _stog = new int [_MAXSTOG];
       _vrh = -1;
}
void Stog::Dodaj (int stavka) {
       if (_vrh >= _MAXSTOG - 1) throw "Stog je pun!";
       _vrh++;
       cout << " na adresu " << _vrh;
       _stog [_vrh] = stavka;
}
void Stog::Skini (int& stavka) {
       if (_vrh < 0) throw "Stog je prazan!";
       cout << "S adrese " << _vrh;
       stavka = _stog [_vrh];
       _vrh--;
}
void Stog::Isprazni () {
       _vrh = 0;
}
Stog::~Stog () {
       delete [] _stog;
       _vrh = -1;
}
void Stog::Ispisi() {
  int a;
  if (_vrh == -1) cout << "(prazan)";
  cout << '\n';
  for (a = 0; a <= _vrh; ++a)
    cout << _stog[a] << " ";
}
```

TSPjednostavni.c

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

#define MAXGRAD 15       // i to je previše s obzirom na n!
// c – matrica udaljenosti između gradova (i, j)
int c[MAXGRAD][MAXGRAD]; // da ne kompliciramo s prijenosom 2D polja

// ispis znaka c u zadanoj duljini n
void nznak (int c, int n) {
  while (--n >= 0) putchar(c);
```

```
}

// operacija \ (skup \ element)
void minus (int skup[], int n, int element){
       int i, j;
       for (i = 0; i < n; i ++)
    if (skup[i] == element) break;
  for (j = i; j < n - 1; j ++)
       skup[j] = skup[j + 1];
}

// ispisuje tekst pa vrijednosti clanova
void PisiPolje(char *s, int *p, int n) {
  int i;
  printf("%s ", s);
  for (i=0; i < n; ++i)
    printf ("%d ", p[i]);
  printf("\n");
}

int TSP (int IzGrada, int *gradovi, int n) {
       int *lgradovi, minTSP, pomTSP, i;
       lgradovi = malloc (n * sizeof(int));
  memcpy (lgradovi, gradovi, n * sizeof(int));
       minus (lgradovi, n, IzGrada);


  -- n;
  nznak(' ', 5-n);
  if (n == 1) {
       minTSP = c[IzGrada][lgradovi[0]];
    printf(" %d>%d=%d\n", IzGrada, lgradovi[0],c[IzGrada][lgradovi[0]]);
  } else {
    printf(" %d>%d=%d\n", IzGrada, lgradovi[0],c[IzGrada][lgradovi[0]]);
       minTSP = c[IzGrada][lgradovi[0]]
                     + TSP (lgradovi[0], lgradovi, n);

       for (i = 1; i < n; i++) {
    nznak(' ', 5-n);
    printf(" %d>%d=%d\n", IzGrada, lgradovi[i],c[IzGrada][lgradovi[i]]);
              pomTSP = c[IzGrada][lgradovi[i]]
                     + TSP (lgradovi[i], lgradovi, n);
      if (pomTSP < minTSP) {
        minTSP = pomTSP;
      }
              }
       }

  //nznak(' ', 5-n); printf("< %d (%d)\n", IzGrada, minTSP);
       free (lgradovi);
       return minTSP;
}

int main () {
  int gradovi[MAXGRAD], n, i, j, minTSP;
       srand (time(NULL));
       do {
              printf ("Unesite broj gradova: ");
              scanf ("%d", &n);
  } while (n < 2 || n > MAXGRAD);

  // generiranje matrice
       for (i = 0; i < n; i++) {
              gradovi[i] = i;
              c[i][i] = 0;
              for (j = i + 1; j < n; j ++) {
#if 1
                     c[i][j] = rand()%9 + 1;
#else
      c[i][j] = i*10 + j;
```

```
#endif
                    c[j][i] = c[i][j];
                    printf ("c[%d][%d] = %d\n", i, j, c[i][j]);
                }
        }
        minTSP = TSP (0, gradovi, n);
        printf ("Najmanji trosak je: %d\n", minTSP);

        return 0;
}
```

TSPjednostvni.c

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

#define MAXGRAD 15        // i to je previše s obzirom na n!
// c – matrica udaljenosti između gradova (i, j)
int c[MAXGRAD][MAXGRAD]; // da ne kompliciramo s prijenosom 2D polja

// ispis znaka c u zadanoj duljini n
void nznak (int c, int n) {
  while (--n >= 0) putchar(c);
}

// operacija \ (skup \ element)
void minus (int skup[], int n, int element){
        int i, j;
        for (i = 0; i < n; i ++)
    if (skup[i] == element) break;
  for (j = i; j < n - 1; j ++)
        skup[j] = skup[j + 1];
}

// ispisuje tekst pa vrijednosti clanova
void PisiPolje(char *s, int *p, int n) {
  int i;
  printf("%s ", s);
  for (i=0; i < n; ++i)
    printf ("%d ", p[i]);
  printf("\n");
}

int TSP (int IzGrada, int *gradovi, int n) {
        int *lgradovi, minTSP, pomTSP, i;
        lgradovi = malloc (n * sizeof(int));
  memcpy (lgradovi, gradovi, n * sizeof(int));
        minus (lgradovi, n, IzGrada);

  //PisiPolje("gradovi", gradovi, n);
  //PisiPolje("lgradovi", lgradovi, n);

  -- n;
  nznak(' ', 5-n); printf("> %d\n", IzGrada);
  if (n == 1) {
        minTSP = c[IzGrada][lgradovi[0]];
  } else {
        minTSP = c[IzGrada][lgradovi[0]]
                    + TSP (lgradovi[0], lgradovi, n);
        for (i = 1; i < n; i++) {
              pomTSP = c[IzGrada][lgradovi[i]]
                    + TSP (lgradovi[i], lgradovi, n);
       if (pomTSP < minTSP) {
         minTSP = pomTSP;
       }
```

```
                }
        }

  nznak(' ', 5-n); printf("< %d (%d)\n", IzGrada, minTSP);
        free (lgradovi);
        return minTSP;
}

int main () {
  int gradovi[MAXGRAD], n, i, j, minTSP;
        srand (time(NULL));
        do {
                printf ("Unesite broj gradova: ");
                scanf ("%d", &n);
  } while (n < 2 || n > MAXGRAD);

  // generiranje matrice
        for (i = 0; i < n; i++) {
                gradovi[i] = i;
                c[i][i] = 0;
                for (j = i + 1; j < n; j ++) {
#if 0
                        c[i][j] = rand() + 1;
#else
      c[i][j] = i*10 + j;
#endif
                        c[j][i] = c[i][j];
                        printf ("c[%d][%d] = %d\n", i, j, c[i][j]);
                }
        }
        minTSP = TSP (0, gradovi, n);
        printf ("Najmanji trosak je: %d\n", minTSP);
        return 0;
}
```

UklanjanjeRekurzije.c

```
// UklanjanjeRekurzije.c
#include <stdio.h>
#include <stdlib.h>
#define MAXA 10
// uklonjena rekurzija iz maxclan() u Rekurzija.c
/* originalni algoritam
  if (i >= n-1) return n-1;
  imax = maxclan (A, i + 1, n);
  if (A[i] > A[imax]) return i;
  return imax;

 * pri tom je promijenjen u
  if (i < n-1) {
    imax = maxclan (A, i + 1, n);
    if (A[i] > A[imax])
                return i;
              else
                return imax;
        } else {
          return n-1;
        }
*/
int maxclan2 (int A[], int i, int n) {
  int imax, k, adresa, vrh, *stog;
  vrh = -1;                         //Pravilo  1
  stog = (int *) malloc (2 * n * sizeof(int));
L1:                                 //Pravilo  2
  if (i < n-1) {
        vrh++; stog[vrh] = i;       //Pravilo  3
    vrh++; stog[vrh] = 2;           //Pravilo  4
    i++;                            //Pravilo  5
    goto L1;                        //Pravilo  6
L2:                                 //Pravilo  7
```

```
    imax = stog[vrh]; vrh--;
    if (A[i] > A[imax]) {
      k = i;
    } else {
      k = imax;
    }
  } else {
    k = n-1;
  }
  if (vrh == -1) {                      //Pravilo  8
    free (stog);
    return k;
  } else {                              //Pravilo  9
    adresa = stog[vrh]; vrh--;          //Pravilo 10
    i = stog[vrh]; vrh--;               //Pravilo 11
    vrh++; stog[vrh] = k;               //Pravilo 12
    if (adresa == 2) goto L2;           //Pravilo 13
  }
}
// Trazenje najveceg clana iterativno
// Rekurzija je uklonjena intuitivno znajuci redoslijed poziva
// max(0) -> max(1) -> max(2) -> max(3) -> max(4)
// i pitalice if (A[i] > A[imax]) po povratku iz rekurzije
int maxclan3 (int A[], int n) {
       int i, imax;
       i = n-1;
       imax = n-1;
       while (i > 0) {
              i--;
              if (A[i] > A[imax]) imax = i;
       }
       return imax;
}
// Trazenje najveceg clana iterativno
int maxclan4 (int A[], int n) {
       // Vrijedi samo za n > 0
       int i, imax = 0;                 // pretpostavimo da je prvi najveci
       for (i = 1; i < n; i++)          // trazimo veci u ostatku polja
              if (A[i] > A[imax])       // da li je trenutni veci?
                     imax = i;                  // postaje najveci
       return imax;
}
int main () {
  int A[MAXA], n;
       FILE *fi;
       int i;
       fi = fopen ("UlazZaUklanjanjeRekurzije.txt", "r");
       if (!fi) exit (1);
       n = 0;
       while(n < MAXA && fscanf(fi,"%d",&A[n])!= EOF) n++;
       fclose (fi);
       for (i = 0; i < n; i++) printf("\nA [%d] = %d", i, A [i]);
       i = maxclan2 (A, 0, n);
       printf("\nNajveci clan je A [%d] = %d",i, A[i]);
       i = maxclan3 (A, n);
       printf("\nNakon pojednostavnjenja:\nNajveci clan je A [%d] = %d",i, A[i]);
       i = maxclan4 (A, n);
       printf("\nZa n > 0:\nNajveci clan je A [%d] = %d\n",i, A[i]);
       return 0;
}
```