Drugi primjer izvedbe hrpe

- Jednostavan programski kod koji od elemenata polja stvori hrpu, koristi se jednostavnija struktura podataka (samo polje)
- Ovdje se pretpostavlja da je prvi element hrpe u ćeliji polja indeksa 1 (prva ćelija polja indeksa 0 se ne koristi), pa je roditelj *i*-tog čvora na mjestu *i*//2
- Na "dno" (list) hrpe dodaje se član koji se onda uspoređuje i zamjenjuje ako je potrebno sa svojim roditeljem, praroditeljem, prapraroditeljem itd. dok ne postane manji ili jednak nekoj od tih vrijednosti.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>  //za funkciju pow()
#define MAXGOM 100
typedef int tip;

void ubaci (tip A[], int k) {  // ubacuje vrijednost iz A[k] na hrpu pohranjenu u A[1:k-1]
  int i, j;
  tip novi;
  j = k;
  i = k/2;
  novi = A[k];
  while ((i > 0) && (A[i] < novi)) {
    A[j] = A[i]; // povecaj razinu za roditelja
    j = i;
    i /= 2;    // roditelj od A[i] je na A[i/2]
  }
  A[j] = novi;
}

void main(void) {
  FILE *fi;
  int i, j, k;
  tip A[MAXGOM];
```

```
fi = fopen ("UlazZaHrpu.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 hrpu po retcima
            i = 1;
            k = 1;
            while (i < j) { // petlja do zadnjeg u hrpi
                        // pisi do maksimalnog u hrpi 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");
     }
     system("PAUSE");
     exit(0);
}
```

- Za analizu najgoreg slučaja algoritma uzmimo n elemenata. Na i-toj razini potpunog binarnog stabla ima najviše 2^{i-1} čvorova. Na svim nižim razinama do tada ima ukupno $2^{i-1} - 1$ čvorova, za $i > 1$. Stablo s k razina ima najviše 2^k-1 čvorova. Stablo s k-1 razinom ima najviše $2^{k-1}-1$ čvorova. Ako je stablo potpuno, započeta je posljednja razina, pa vrijedi $2^{k-1} - 1 < n \leq 2^k - 1$
- Iz ovoga slijedi:
  - $2^{k-1} < n + 1 \Rightarrow (k-1) \log 2 < \log (n+1) \Rightarrow k < \log_2 (n+1) + 1$

    $n + 1 \leq 2^k \Rightarrow \log (n+1) \leq k \log 2 \Rightarrow \log_2 (n+1) \leq k$

    $\log_2 (n+1) \leq k < \log_2 (n+1) + 1$ odnosno $k = \lceil log_2(n+1) \rceil$
- Na primjer:
  - za n = 14 treba $\lceil \log_2 15 \rceil = \lceil \ln 15/\ln 2 \rceil = \lceil 2.70805/0.693147 \rceil = \lceil 3.9 \rceil$ = 4 razine
  - za n = 15 treba $\lceil \log_2 16 \rceil = \lceil 4 \rceil$ = 4 razine
  - za n = 16 treba $\lceil \log_2 17 \rceil = \lceil 4.087 \rceil$ = 5 razina

- U najgorem slučaju, `while` petlja se izvršava proporcionalno broju razina u gomili. Skup podataka koji predstavlja najgori slučaj za ovaj algoritam je polje s rastućim podacima. Tada svaki novi element, onaj koji se ubacuje u gomilu pozivom funkcije `ubaci`, postaje korijen pa se kroz k razina obavlja zamjena. Vrijeme izvođenja je tada $O(n \log_2 n)$. Za prosječne podatke vrijeme za stvaranje gomile iz skupa podataka je $O(n)$, što je za red veličine bolje.

Još jedan primjer izvedbe hrpe

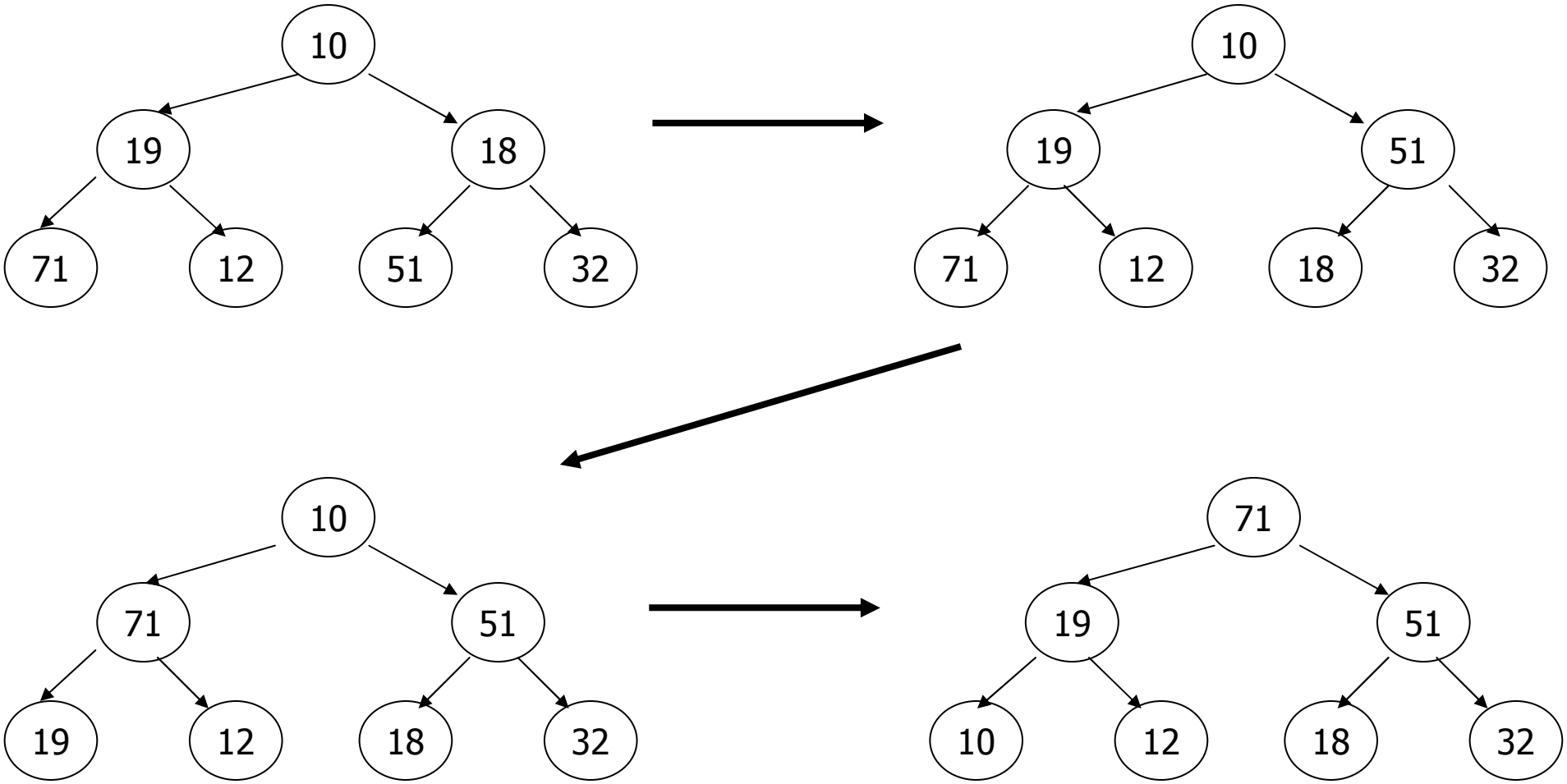
- Za poboljšanje brzine obavljanja zadanih operacija stvoren je algoritam koji kreće od krajnjih čvorova prema korijenu, razinu po razinu. Samo podatak u korijenu može narušavati svojstvo hrpe, dok podstabla zadržavaju to svojstvo. Tada je samo potrebno tu nepravilnost ispraviti i opet dobivamo željenu hrpu. To čini funkcija `podesi` u slijedećem primjeru. Za krajnje čvorove svojstvo hrpe je zadovoljeno, pa treba u `stvori_hrpu` funkciji provesti popravljanje svojstva hrpe samo za korijen stabla (identični postupak prvom primjeru za hrpu)
- Funkcija podesi(): potpuna binarna stabla s korijenima A[2*i] i A[2*i+1]kombiniraju se s A[i] formirajuci jedinstvenu hrpu

```
#include <stdio.h>
#include <stdlib.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 hrpu, 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;
}

// premjesti elemente A[1:n] da tvore gomilu
void StvoriHrpu (tip A[], int n) {
  int i;
  for (i = n/2; i >= 1; i--)
              podesi (A, i, n);
}
```

```
void main(void) {
    FILE *fi;
    int i, j, k, n;
    tip A[MAXGOM];
    fi = fopen ("UlazZaHrpu.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 hrpu
        n = j - 1;
        StvoriHrpu (A, n);
        // pisi hrpu po retcima
        i = 1;  k = 1;
        while (i < j) { // petlja do zadnjeg u hrpi
            // pisi do maksimalnog u hrpi 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"); }
    system("PAUSE"); exit(0); }
```

- Stvaranje hrpe za ulazni niz podataka: 10,19,18,71,12,51,32



- Za n podataka, $2^{k-1} \leq n < 2^k$, broj razina je $k = \lceil log_2(n+1) \rceil$. Za najgori slučaj broj iteracija u `podesi` iznosi k-i za čvor na razini i gdje ima najviše 2^{i-1} čvorova. Slijedi da je, vrijeme izvođenja za `StvoriHrpu`:

  $\sum_{i=1}^{k} 2^{i-1} (k-i)$. Uočimo da se eksponent mijenja od 0 do $k-1$, a faktor od $k-1$ do 0.

  Slijedi ekvivalentni izraz kad se izbaci faktor 0 i obrne redosljed sumacije:

  $$= \sum_{i=1}^{k-1} i\, 2^{k-i-1} = \sum_{i=1}^{k-1} 2^{k-1}\, i\, 2^{-i}$$

  S obzirom da je $2^{k-1} \leq n$,

  $\leq n \sum_{i=1}^{k-1} i/2^i \leq 2n = O(n)$, jer suma reda teži prema 2.

- Vrijeme izvođenja za najgori slučaj algoritma `StvoriHrpu` je $O(n)$, što je za red veličine bolje od $O(n \log_2 n)$ za uzastopno korištenje `ubaci` funkcije.
- Funkcija `StvoriHrpu` traži da su svi elementi za stvaranje hrpe već prisutni, dok ubaci može ubaciti novi element u hrpu bilo kada. Funkcije koje hrpa treba brzo obaviti i radi kojih je napravljena ta struktura podataka su ubacivanje novih i brisanje najvećeg elementa iz skupa podataka. Brisanje najvećeg podatka se obavlja izbacivanjem korijena i pozivanjem funkcije `podesi`, a ubacivanje novih se radi funkcijom `ubaci`. Tako se postiže da se obje željene funkcije obavljaju u $O(\log_2 n)$ vremenu.

Prioritetni red i hrpa

- Napisati program za ubacivanje i skidanje elemenata (prioritet određuje cijeli broj) iz prioritetnog reda realiziranog pomoću hrpe. Za prioritet elementa i odabir operacije (ubacivanje ili skidanje) koristi se generator slučajnih brojeva. Ako generator načini parni broj, element najvećeg prioriteta vadi se iz prioritetnog reda. Ako generator načini neparni broj, stvara se novi element sa slučajno generiranim prioritetom i stavlja u prioritetni red.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <ctype.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] stvarajuci jedinstvenu hrpu, 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 hrpu pohranjenu u A[1 : k - 1]
    int i, j;
    tip novi;
    j = k;
    i = k / 2;
    novi = A[k];
    while ((i > 0) && (A[i] < novi)) {
            A[j] = A[i]; // smanji razinu za roditelja
            j = i;
            i /= 2;        // roditelj od A[i] je na A[i/2]
    }
    A[j] = 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));
     printf("Za obavljanje jednog koraka pritisni ENTER, za kraj bilo koji znak\n”);
    while(isspace(getchar())) {
            if (rand() % 2) {
                    if (k >= MAXPRIOR)
                            printf(“Prioritetni red je pun!\n");
                    else {
     printf("Dodavanje u prioritetni red: %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(“Prioritetni red je prazan!\n");
                    else
                            printf("Skidanje iz prioritetnog reda: %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"); } }
  system(“PAUSE”);
}
```

Računanje sume članova polja

- Zadatak: napisati funkciju koja računa sumu članova polja.
- Najjednostavnije rješenje:

```
int suma1 (int polje[], int n) {
  int i, suma = 0;
  for (i = 0; i < n; i ++) {
        suma += polje[i]; }
  return suma;
}
```

složenost O(n)

- Rekurzivni postupak: sumiranje prvog člana s ostatkom polja:

```
int suma2 (int polje[], int n) {
if (n <= 0) return 0;
return polje[n - 1] + suma2(polje, n - 1);
}
```

složenost O(n), prosječno asimptotsko vrijeme izvođenja je 2n

- Podijeli-pa-vladaj algoritam: polje se dijeli u polovice koje se sumiraju

```
int suma3 (int polje[], int l, int d) {    // l (d) – indeks lijevog (desnog) ruba polja
    int p;
    if (d < l) return 0;
    if (d == l) return polje[d];
    p = (d + l) / 2;
        return suma3 (polje, l, p) + suma3 (polje, p + 1, d);
   }
```

- Složenost ovog algoritma je također O(n)

- Glavna procedura:

```
void main() {
    int N,i,*a,s1,s2,s3;

    printf("Unesi duljinu polja:\n");
    scanf("%d",&N);
    if ((a = (int *)malloc(N*sizeof(int))) == NULL)
            {printf("Greska u rezerviranju memorije !");
        exit(1); }
```

```
    srand(time(NULL));
    printf(" Elementi polja su:\n");
    for(i=0; i < N; i++) {
            a[i]= 100 * ((float) rand()/(RAND_MAX + 1));
            printf("%d  ",a[i]); }
            printf("\n\n");

    s1 = suma1(a,N);
    printf(" Rezultat funkcije suma1:%d\n",s1);
    s2 = suma2(a,N);
    printf(" Rezultat funkcije suma2:%d\n",s2);
    s3 = suma3(a,0,N-1);
    printf(" Rezultat funkcije suma3:%d\n",s3);
    system("PAUSE");
}
```

Računanje binomnih koeficijenata – vrijeme izvršavanja

- Po definiciji binomni koeficijenti su:

$$\binom{n}{k} = \frac{n!}{(n-k)!\,k!}$$

- Rekurzija za računanje binomnih koeficijenata:

$$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$$

$$\binom{n}{0} = \binom{n}{n} = 1$$

Ovakva rekurzija mnogo puta računa iste vrijednosti, pa je neefikasna

- Preurediti račun tako da se jednom izračunata vrijednost pospremi i sačuva za daljnja računanja – računanje Pascalovog trokuta
- Algoritam:

  - u prvom redu se upiše samo broj 1

  - za računanje elemenata u sljedećim redovima se uzimaju brojevi iz prethodnog reda koji su lijevo i desno od novog broja, ako broj lijevo ili desno ne postoji uzima se 0

				1					
			1		1				
		1		2		1			
	1		3		3		1		
1		4		6		4		1	
	5		10		10		5		1

```
#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;}

// vraca faktorijela (n), broj iteracija, zastavicu pogreske, radi do n=20
long long FAKT (int n, long *freq, int *errorflag) {
  int i;
  long long p;
  p = 1;
  for (i = 2; i <= n; i++) {
    p *= i;
              if (p <= 0)  *errorflag = 1;
    *freq += 1;  }
  return p;}
```

```
// binomni koeficijenti s pomocu faktorijela
long BINOM (int n, int m, long *freq, int *errorflag) {
     long long p;
  *freq += 1;
     p = FAKT (n, freq, errorflag);
     p /= FAKT (m, freq, errorflag);
     p /= FAKT (n - m, freq, errorflag);
     return (long) p;
}

// 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");
  }
}
```

```
void main (void) {
  int n, m, i, j;
  int broj;
  long k;
  int errorflag;
  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 >");
      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) break;
     Blaise (n);
   }
   system("PAUSE");
   exit (0);
}
```