

## Osnovne akademske studije

**PREDMET:** Objektno-orijentisana simulacija

**TEMA:** Strategija raspoređivanja događaja 2

**Predmetni nastavnik:** Doc. dr Marko Đogatović

# Lehmerov generator slučajnih brojeva

Lehmerov GSB je takođe poznat kao multiplikativni linearni kongruentni GSB. To je najčešće korišćeni linearni kongruentni generator. Postoje tri parametra:

1.  $m$  – modulus,
2.  $a$  – multiplikator,
3.  $c$  – aditivna konstantna (najčešće je jednaka 0).

Takođe postoji i početna vrednost  $z_0$  (tzv. seme). Pseudoslučajne vrednosti se generišu primenom sledeće formule:

$$Z_i = (Z_{i-1}a+c) \bmod m. \text{ (mod je moduo - ostatak nakon deljenja dva cela broja)}$$

Ova formula generiše slučajnu sekvencu od  $m-1$  brojeva. Za kombinaciju parametara koja vodi ka sekvenci od  $m-1$  brojeva se kaže da ima celi period. Najčešće se vrednosti  $Z_i$  se normalizuju tako da se nalaze na jedničnom intervalu na sledeći način

$$U_i = Z_i / m \text{ za } i=1,2,\dots$$

Uspeh ovakvog generatora zavisi od izbora parametara. Npr. izborom  $m = 11$  i multiplikatorom  $a = 8$  dobija se sekvenca sa celim periodom, dok multiplikator  $a = 9$  daje sekvencu dužine 5.

$a = 8$  ... 1, 8, 9, 6, 4, 10, 3, 2, 5, 7, 1 ...

$a = 9$  ...1, 9, 4, 3, 5, 1, ...

Park i Miller daju odgovarajući izbor parametara  $m = 2147483647 = 2^{31}-1$  i  $a = 16807 = 7^5$  za generator sa celim periodom. U nastavku su dati programi realizovani u C jeziku kojima je implementiran Lehmerov GSB.

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

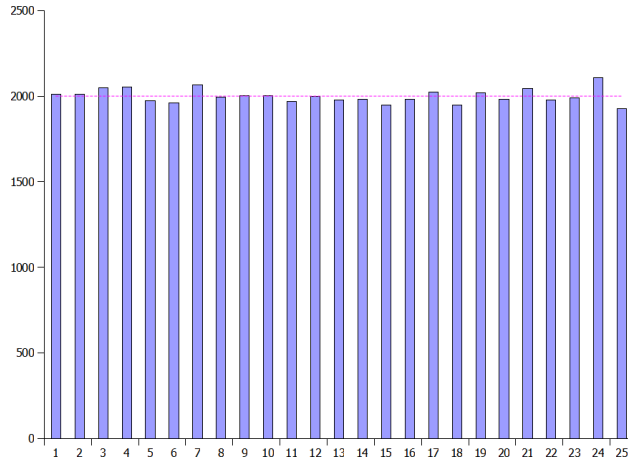
struct rand {
    int seed,mult;
    double val;
};

void randu(struct rand* rnd) {
    rnd->seed *= rnd->mult;
    if(rnd->seed<0)
        rnd->seed += LONG_MAX+1;
    rnd->val = rnd->seed/(1.0*LONG_MAX);
}
```

```

int main() {
    int i;
    struct rand rnd;
    rnd.seed = 1;
    rnd.mult = 16807;
    for(i=0; i<50000; ++i) {
        randu(&rnd);
        printf("%f\n", rnd.val);
    }
    return 0;
}

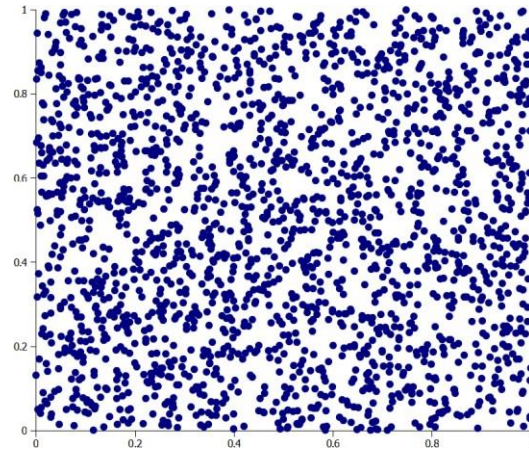
```



```

int main() {
    int i;
    struct rand rnd;
    double r1, r2;
    rnd.seed = 1;
    rnd.mult = 16807;
    for(i=0; i<2000; ++i) {
        randu(&rnd);
        r1 = rnd.val;
        randu(&rnd);
        r2 = rnd.val;
        printf("%f,%f\n", r1, r2);
    }
    return 0;
}

```



**Histogram od 50000 vrednosti generisanih `randu` funkcijom i dvodimenzionalna raspodela 2000 pseudoslučajnih parova generisanih `randu` funkcijom**

# rand funkcija

```
int rand(void);
```

vraća pseudoslučajni ceo broj na intervalu od 0 do `RAND_MAX`. Broj se generiše algoritmom koji vraća sekvencu slučajnih brojeva svaki put kada se funkcija pozove. Najčešće je algoritam neka varijanta linearnog kongruentnog generatora slučajnih brojeva.

Ovaj algoritam koristi seme da generiše seriju. Seme treba da bude inicijalizovano nekom vrednošću korišćenjem funkcije `srand`.

`RAND_MAX` je konstanta definisana u `<stdlib.h>` (`<cstdlib>`). Ova vrednost je najmanje 32767. `rand()` vraća broj na intervalu od 0 do 1 korišćenjem sledećeg dela koda

```
double r = rand() / (RAND_MAX + 1.0);
```

## srand funkcija

```
void srand(unsigned int seed) ;
```

Ova funkcija inicijalizuje generator slučajnih brojeva. GSB je inicijalizovan korišćenjem argumenta prosleđenog kao seme (seed).

Za različite vrednosti semena korišćene prilikom poziva `srand`, generator pseudoslučajnih brojeva će generisati različite sekvence brojeva.

Ukoliko se seme postavi na 1, generator se inicijalizuje na svoju početnu vrednost i vraća iste vrednosti kao i pre poziva `srand`.

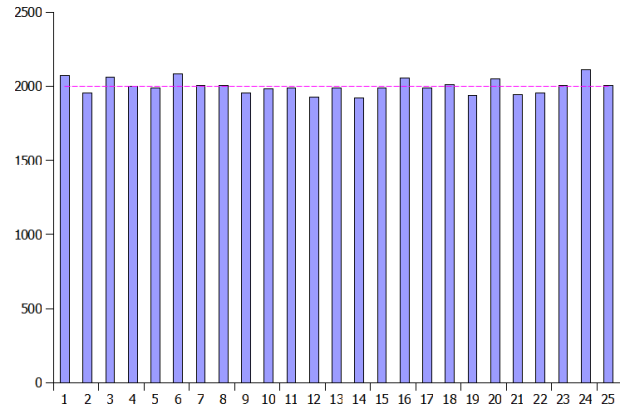
Moguće je seme inicijalizovati i vremenom na sledeći način

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

`time` funkcija se nalazi u zaglavlju `<time.h>` (`<ctime>`) .

Histogram od 50000 vrednosti generisanih `rand` funkcijom

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
    int i;
    double r;
    srand(time(NULL));
    for(i=0; i<50000; ++i) {
        r = rand()/(RAND_MAX+1.0);
        printf("%f\n", r);
    }
    return 0;
}
```



# Uniformna raspodela

**Gustina**

$$f(x) = \frac{1}{b-a}, a \leq x \leq b$$

**Raspodela**

$$F(x) = \frac{x-a}{b-a}$$

**Sr. vred.**

$$E[X] = \frac{a+b}{2}$$

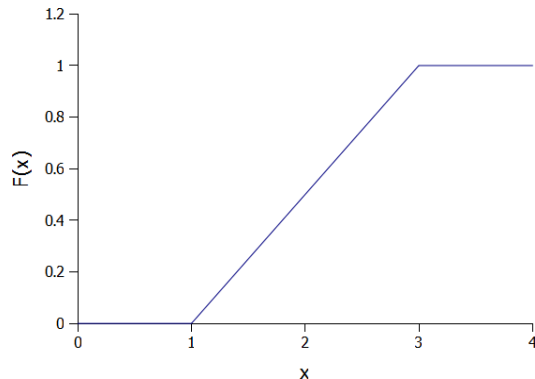
**Varijansa**

$$Var[X] = \frac{(b-a)^2}{12}$$

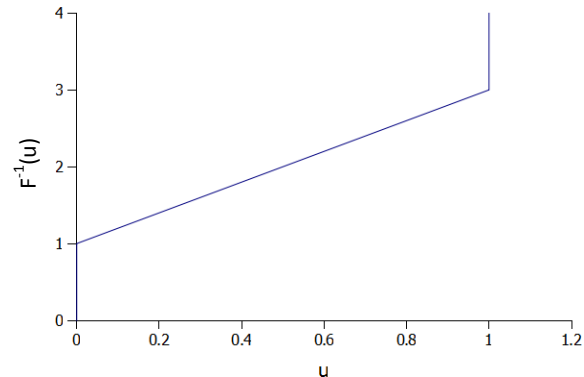
**Nalaženje inverzne raspodele**

$$u = F(x) \Rightarrow x = F^{-1}(u)$$

$$u = F(x) = \frac{x-a}{b-a} \Rightarrow x = a + (b-a)u$$



**Uniformna raspodela**



**Inverzna uniformna raspodela**



# Eksponeñcijalna raspodela

**Gustina**

$$f(x) = \lambda e^{-\lambda x}, x > 0,$$

**Raspodela**

$$F(x) = 1 - e^{-\lambda x},$$

**Sr. vred.**

$$E[X] = \lambda^{-1}$$

**Varijansa**

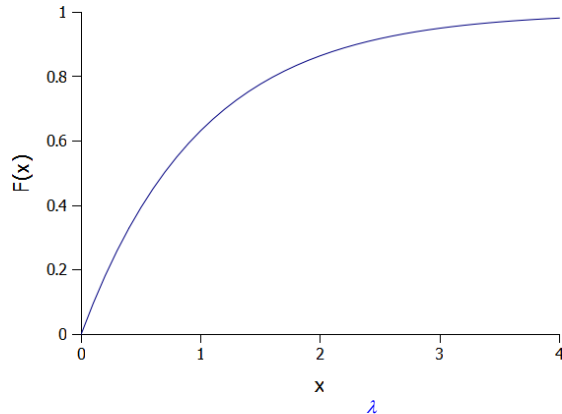
$$\text{Var}[X] = \lambda^{-2}$$

**Nalaženje inverzne raspodele**

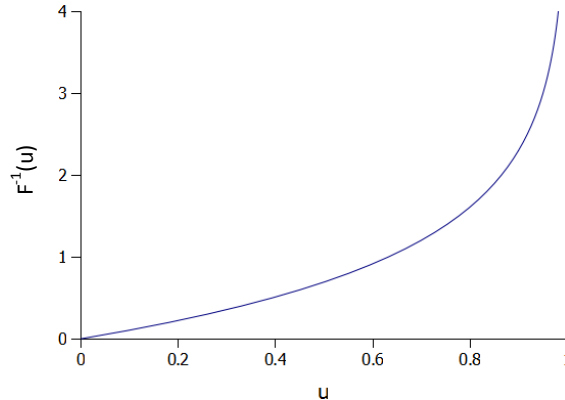
$$u = F(x) \Rightarrow x = F^{-1}(u)$$

$$u = F(x) = 1 - e^{-\lambda x} \Rightarrow x = -\lambda^{-1} \ln(1 - u)$$

$$x = -\lambda^{-1} \ln u$$



**Eksponeñcijalna raspodela**



**Inverzna eksponeñcijalna raspodela**

# Normalna raspodela

**Gustina**

$$f(x) = (2\pi\sigma^2)^{-\frac{1}{2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

**Raspodela**

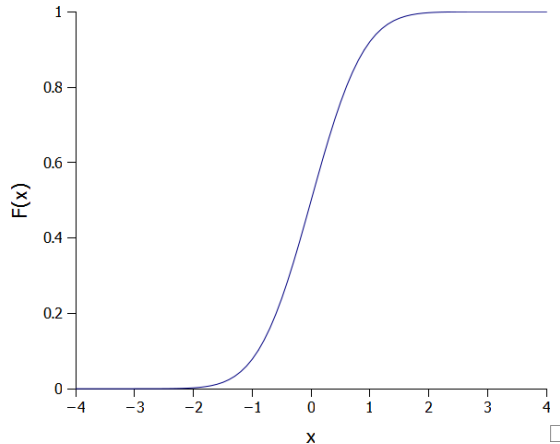
$$F(x) = \frac{1}{2} \left( 1 + \operatorname{erfc} \left( \frac{x-\mu}{\sqrt{2}\sigma} \right) \right)$$

**Sr. vred.**

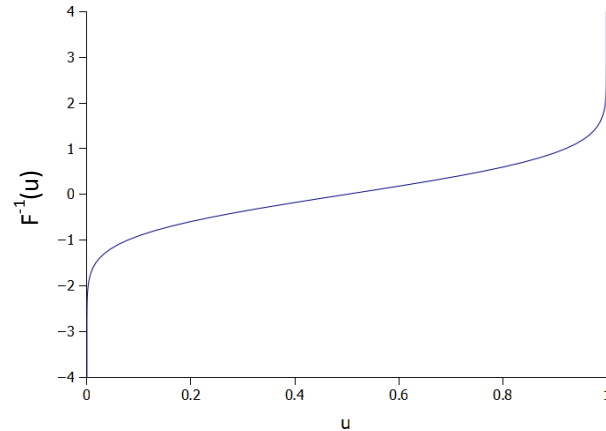
$$E[X] = \mu$$

**Varijansa**

$$\operatorname{Var}[X] = \sigma^2$$



**Normalna raspodela**



**Inverzna normalna raspodela**

Obzirom da vrednosti iz normalne raspodele nije moguće generisati korišćenjem analitičkog izraza za inverznu funkciju raspodele u GPSS-u smo koristili metodu inverzne transformacije za generisanje normalne raspodele. Ovde ćemo pokazati kako se generišu vrednosti iz normalne raspodele korišćenjem metode sumiranja.

### Metoda sumiranja

Ova metoda se koristi za generisanje normalne raspodele. Zasniva se na primeni centralne granične teoreme. Ona se formuliše na sledeći način. Neka je  $\tilde{X}_1, \tilde{X}_2, \dots, \tilde{X}_n$  niz nezavisnih slučajnih promenljivih sa jednakim verovatnoćama, svaka sa srednjom vrednošću  $\mu_x$  i konačnom varijansom  $\sigma_x^2$ .

Njihova srednja vrednost data je sledećim izrazom

$$X = \frac{1}{n} \sum_{i=1}^n \tilde{X}_i$$

Tada novo usvojena promenljiva

$$Y = \frac{X - \mu_x}{\sigma_x / \sqrt{n}}$$

konvergira ka standardnoj normalnoj raspodeli sa srednjom vrednošću  $\mu_x$  i standardnoj devijaciji  $\sigma_x / \sqrt{n}$  tj. za dovoljno veliko  $n$  razlika između promenljive i standardne normalne promenljive može se zanemariti iz praktičnih razloga.

Prema tome da bi generisali uzorak iz standardne normalne raspodele možemo uzeti  $n$  nezavisnih uniformno raspodeljenih slučajnih brojeva  $\tilde{r}_i \in [0, 1)$  sa srednjom vrednošću  $E(\tilde{r}_i) = \frac{1}{2}$  i varijansom  $E(\tilde{r}_i - \mu)^2 = \frac{1}{12}$ . Slučajna promenljiva

$$Z = \frac{\frac{1}{n} \sum_{i=1}^n \tilde{r}_i - \frac{1}{2}}{\sqrt{\frac{1}{12}} / \sqrt{n}} = \frac{\sum_{i=1}^n \tilde{r}_i - \frac{n}{2}}{\sqrt{\frac{n}{12}}}$$

ima normalnu raspodelu sa srednjom vrednošću 0 i varijansom 1 ( $E[Z] = 0, E[(Z - \mu)^2] = 1$ ), kada  $n \rightarrow \infty$ . Zadovoljavajući rezultati aproksimacije se dobijaju za  $n = 12$ . Tada gornja jednačina postaje

$$Z = \sum_{i=1}^{12} \tilde{r}_i - 6$$

Ukoliko je potrebno generisati uzorke iz nestandardizovane normalne raspodele  $\tilde{Z}$  koristiti se sledeći izraz.

$$\tilde{Z} = \mu + \sigma Z$$

Iako je ova metoda jednostavna za realizaciju ona pati od nekoliko nedostataka. Prvi, i najočigledniji nedostatak je da je za jednu vrednost normalne promenjive potrebno generisati dvanaest uniformnih raspodeljenih slučajnih brojeva. To znači da će korišćenjem metode sumiranja veoma brzo istrošiti sekvenca uniformnih brojeva čijim korišćenjem se generiše normalna raspodela. Drugi problem sa ovom metodom je da se na ovaj način standardna

normalna raspodela ograničava na intervalu od  $[-6,6]$  iako je po definiciji neograničena.

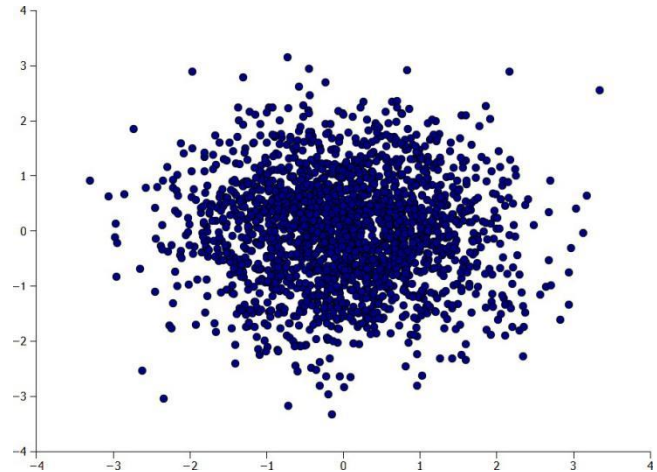
Pored metode inverzne transformacije i metode sumiranja postoje i druge metode za generisanje normalne raspodele. Ovde ćemo kao alternativu navesti Box-Muller-ovu, polarnu i Ziggurat metodu.

Dvodimenzionalna raspodela 2000 pseudoslučajnih parova generisanih  $\text{norm}(0,1)$  funkcijom

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

double norm(double m, double s) {
    int i;
    double z = 0.0;
    for(i=0; i<12; i++)
        z += rand()/(RAND_MAX+1.0);
    return z-6.0;
}

int main() {
    int i;
    double r;
    srand(time(NULL));
    for(i=0; i<2000; ++i)
        printf("%f,%f\n",norm(0,1),norm(0,1));
    return 0;
}
```

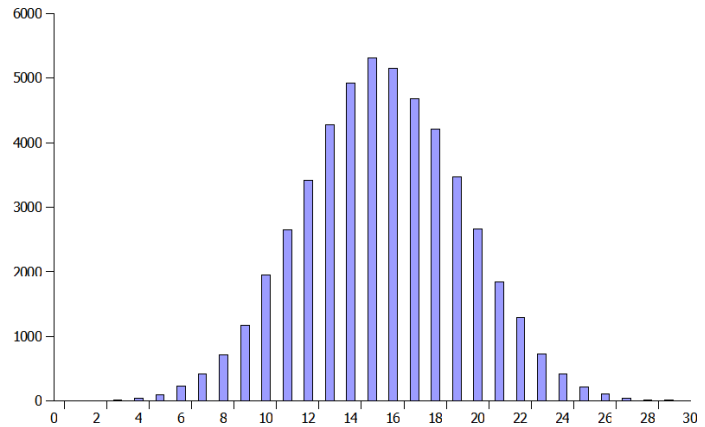


## Histogram od 50000 vrednosti generisanih $\text{norm}(0, 1)$ funkcijom

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

double norm(double m, double s) {
    int i;
    double z = 0.0;
    for(i=0; i<12; i++)
        z += rand()/(RAND_MAX+1.0);
    return z-6.0;
}

int main() {
    int i;
    double r;
    srand(time(NULL));
    for(i=0; i<50000; ++i)
        printf("%f\n", norm(0,1));
    return 0;
}
```



## Klasa raspodela

```
class raspodela {
    // Seme generatora slucajnih brojeva (GSB)
    uint32_t s;
    // Multiplikator
    uint32_t a;
    // Modulus
    uint32_t m;
public:
    // Konstruktor
    raspodela(const uint32_t s):s(s) {
        // Postavljamo vrednost multiplikatora
        a = 16807;
        // Postavljamo vrednost modulusa
        m = 2147483647;
    }
    // Lehmerov multiplikativni kongruentni generator
    double operator() () {
        s = static_cast<uint64_t>(a*s)%m;
        return s/(1.0*m);
    }
    // Uniformna raspodela
    double unif(const double l,const double u) {
        return l+(u-l)*(*this)();
    }
    // Eksponecijalna raspodela
    double expo(const double mi) {
        return -mi*log(*this());
    }
    // Normalna raspodela
    double norm(const double mi,const double sd) {
        double z = 0.0;
        for(int i=0; i<12; i++)
            z += (*this)();
        return mi+(z-6)*sd;
    }
};
```



```
}  
};
```

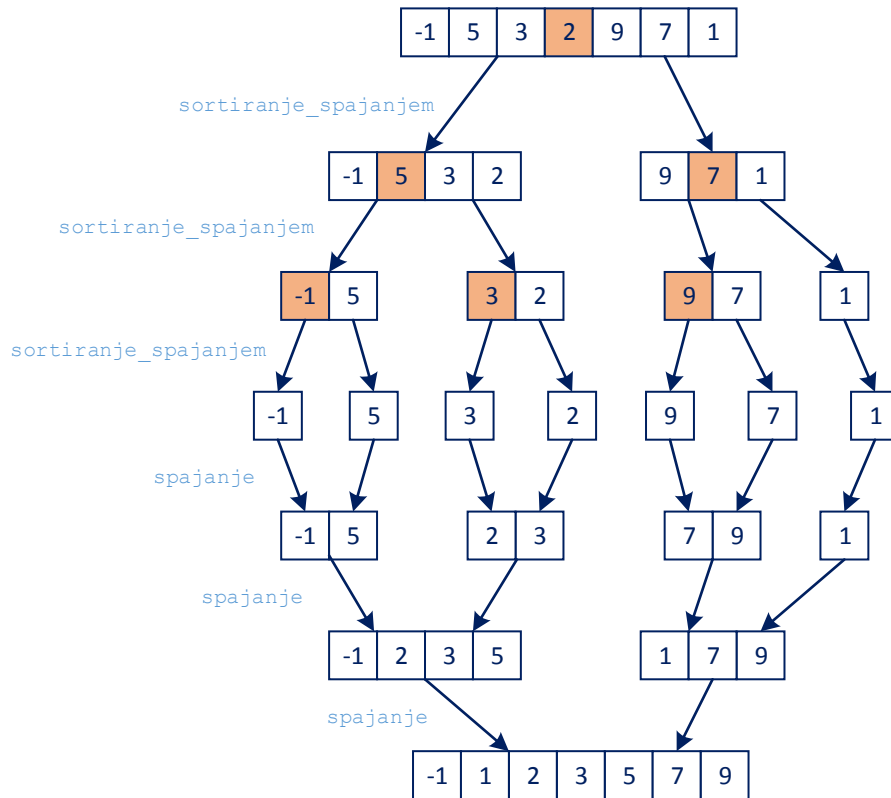
Klasa `raspodela` sadrži tri privatna člana `seme`, `a` i `m` (tipa `unsigned int`). `seme` je seme Lehmerovog generatora slučajnih brojeva. `a` je multiplikator Lehmerovog multiplikativnog GSB, dok je `m` modulus GSB-a. U konstruktoru se postavljaju vrednosti multiplikatora i modulusa i te vrednosti odgovaraju vrednostima Park-Millerovog minimalnog standardnog GSB-a ( $a = 16807$  i  $m = 2147483647$ ). Konstruktoru se prosleđuje vrednost početnog semena GSB-a. U klasi postoje još četiri javno vidljive funkcije. Jedna je operatorska funkcija `operator()` čijim korišćenjem se vraća vrednost slučajnog broja korišćenjem Lehmerovog generatora. Ostale tri funkcije odgovaraju uniformnoj, eksponencijalnoj i normalnoj raspodeli. Ove tri funkcije koriste izraze koji su prethodno prikazani i izvedeni.

`this` je ključna reč koja predstavlja pokazivač na tekuću instancu klase u funkciji članice klase u kojoj se `this` koristi. `(*this)()` poziva `operator()` i vraća vrednost slučajnog broja. `log()` je funkcija prirodnog logaritma (zaglavljuje `<cmath>`).

## „Merge sort“ algoritam

„Merge sort“ algoritam (algoritam sortiranja spajanjem) je rekurzivni algoritam sortiranja elemenata polja. Odabira se središnji element polja i nakon toga se polje deli na levu i desnu polovinu. Polje se rekurzivno deli na levu i desnu polovinu sve dok se ne dođe do polja veličine jednog elementa. Nakon toga tako podeljena polja se spajaju sve dok se ne dobije sortirano polje. Algoritam je složenosti  $O(n \log n)$  i stabilan je.

Primer: Sortirati celobrojno polje -1, 5, 3, 2, 9, 7, 1 korišćenjem „merge sort“ algoritma.



## Sortiranje celobrojnog polja – „Merge sort“ algoritam

```
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <ctime>

void sortiranje_spajanjem(int x[],const size_t l,const size_t r) {
    void spajanje(int x[],const size_t l,const size_t m,const size_t r);
    // Proverava se da li ima elemenata u listi koja se sortira
    if(l<r) {
        // Pivot element
        size_t m = (l+r)/2;
        // Sortira se deo polja do pivota
        sortiranje_spajanjem(x,l,m);
        // Sortira se deo polja od pivota
        sortiranje_spajanjem(x,m+1,r);
        // Spajaju se polja
        spajanje(x,l,m,r);
    }
}

void spajanje(int x[],const size_t l,const size_t m,const size_t r) {
    // Velicina pomocnih polja
    size_t nl = m-l+1, nd = r-m;
    // Pomocna polja
    int *levo = new int[nl];
    int *desno = new int[nd];
    // Polja koja se spajaju se kopiraju u pomocna polja
    std::memcpy(levo ,x+l ,nl*sizeof(int));
    std::memcpy(desno,x+m+1,nd*sizeof(int));
    // Smestamo elemente iz pomocnih polja u novo, sortirano polje
    size_t i,j,k;
    for(i=0,j=0,k=l; i<nl && j<nd; ++k) {
        // Poredjenje
        if(levo[i]<=desno[j])
            x[k] = levo[i++];
        else
            x[k] = desno[j++];
    }
    // Preostali elementi se smestaju u novo polje
    if(i<nl) std::memcpy(x+k,levo+i ,(nl-i)*sizeof(int));
}
```

```
else      std::memcpy(x+k,desno+j,(nd-j)*sizeof(int));
// Brisemo pomocna polje
delete [] levo;
delete [] desno;
}
int main() {
    int x[] = {-1, 5, 3, 2, 9, 7, 1};
    size_t n = sizeof(x)/sizeof(int);
    int *x = new int[n];
    for(size_t i=0;i<n;++i)
        x[i] = 10+rand()%50;
    sortiranje_spajanjem(x,0,n-1);
    for(size_t i=0;i<n;++i)
        std::cout << x[i] << " ";
    std::cout << std::endl;
    delete [] x;
    return 0;
}
```

## Klasa simulacija

```
// Klasa simulacija
class simulacija {
private:
    // Isto kao u aes1
    // Sortiranje LBD - sortiranje spajanjem
    void sortiraj_lbd() {
        sortiranje_spajanjem(_lbd,0,_broj-1);
    }
    void sortiranje_spajanjem(dogadjaj* x[],const size_t l,const size_t r) {
        // Proverava se da li ima elemenata u listi koja se sortira
        if(l<r) {
            // Pivot element
            size_t m = (l+r)/2;
            // Sortira se deo polja do pivota
            sortiranje_spajanjem(x,l,m);
            // Sortira se deo polja od pivota
            sortiranje_spajanjem(x,m+1,r);
            // Spajaju se polja
            spajanje(x,l,m,r);
        }
    }
    void spajanje(dogadjaj* x[],const size_t l,const size_t m,const size_t r) {
        // Velicina pomocnih polja
        size_t nl = m-l+1, nd = r-m;
        // Pomocna polja
        dogadjaj **levo = new dogadjaj*[nl];
        dogadjaj **desno = new dogadjaj*[nd];
        // Polja koja se spajaju se kopiraju u pomocna polja
        std::memcpy(levo ,x+l ,nl*sizeof(dogadjaj*));
        std::memcpy(desno,x+m+1,nd*sizeof(dogadjaj*));
        // Smestamo elemente iz pomocnih polja u novo, sortirano polje
        size_t i,j,k;
        for(i=0,j=0,k=1; i<nl && j<nd; ++k) {
```

```
// Poredjenje
if(levo[i]->vrati_vreme()<=desno[j]->vrati_vreme())
    x[k] = levo[i++];
else
    x[k] = desno[j++];
}
// Preostali elementi se smestaju u novo polje
if(i<nl) std::memcpy(x+k,levo+i ,(nl-i)*sizeof(dogadjaj*));
else    std::memcpy(x+k,desno+j,(nd-j)*sizeof(dogadjaj*));
// Brisemo pomocna polje
delete [] levo;
delete [] desno;
}
// Isto kao u aes1
};
```

Klasa simulacija je uglavnom ista kao i klasa simulacija iz prethodne verzije programa (vidi Strategija raspoređivanja događaja 1). Ono što je promenjeno u odnosu na prethodnu verziju je to da su dodate privatne funkcije `void sortiranje_spajanjem(dogadjaj* x[], const size_t l, const size_t r)` i `void spajanje(dogadjaj* x[], const size_t l, const size_t m, const size_t r)` koje se pozivaju iz postojeće privatne funkcije `void sortiraj_lbd()`.



**Primer 2.** Posmatrajmo poštu u kojoj se nalazi četiri univerzalna šaltera ispred koga se formira zajednički red čekanja. Vreme opsluge je eksponencijalno raspodeljeno sa srednjim vremenom 200 sekundi.

Klijenti dolaze po puasonovom toku sa srednjim vremenom od 100 sekundi.

Simulirati rad pošte za period od 12 časova. Vremenska jedinica je 1 sekunda. Takođe, potrebno je izračunati statistike resursa i reda čekanja.

Bezuslovni događaji u ovom primeru su gotovo isti kao i u primeru 1. Ponovo postoje dva безусловna događaja *DolazakKlijenta* i *OdlazakKlijenta*. Pseudokodovi ova dva događaja su isti kao i u prethodnom primeru. Pored ova dva događaja uvodi se i treći događaj koji se aktivira nakon isteka vremena simulacije i služi da prekine simulaciju. Ovaj događaj nosi naziv *KrajSimulacije*. Pseudokod tog događaja je jednostavan i on glasi:

```
procedure KrajSimulacije  
begin  
    Zaustavi simulaciju.  
end
```

Za realizaciju primera koristićemo zaglavlje aes.h, aes\_proc.h i aes\_dist.h. Zaglavlje aes.h je jako slično prethodnoj verziji tog zagavlja, osim što je promenjena klasa simulacija (funkcija sortiranje\_spajanjem). Takođe se menja i početak datoteke, na taj način da se pored prethodno uvedenih zaglavlja uvozi i zaglavlje <cstring>.

```
// Zaglavlje aes.h
#ifndef __AES_H__
#define __AES_H__

#include <iostream>
#include <string>
#include <cstdlib>
#include <cstring>

// Uključujemo prostor imena std;
using namespace std;

// Prostor imena esa
namespace aes {
    // Realizacije klasa raspodela, entitet, fifo_red, resurs i simulacija
    // ...
}
#endif // __AES_H__
```

## zad1-2.cpp

U ovoj izvornoj datoteci neće biti mnogo izmena u odnosu na zad1.cpp. Kao i u zad1.cpp, na početku se učitavaju odgovarajuća zaglavlja i dozvoljava se korišćenje odgovarajućih prostora imena.

```
// Datoteka zad1.cpp
#include "aes.h"
#include "aes_proc.h"
#include "aes_dist.h"
#include <iostream>

// Uključujemo prostore imena esa i std
using namespace aes;
using namespace std;
```

Zatim realizujemo deklaracije klasa koje predstavljaju te bezuslovne događaje.

```
// Klasa dogadjaja dolazak korisnika
class dolazak_korisnika:public dogadjaj {
public:
    // Konstruktor
    dolazak_korisnika():dogadjaj(1) {}
    // Akcija
    void akcija();
    // Pomocna funkcija za pravljenje dogadjaja
    static dolazak_korisnika* napravi(korisnik* kor,vreme vn) {
        dolazak_korisnika* dk = new dolazak_korisnika;
        dk->postavi_agenta(0, kor);
        dk->postavi_vreme(simu.vrati_vreme()+vn);
        return dk;
    }
};
```

```
// Klasa dogadjaja odlazak korisnika
class odlazak_korisnika:public dogadjaj {
public:
    // Konstruktor
    odlazak_korisnika():dogadjaj(1) {}
    // Akcija dogadjaja
    void akcija();
    // Pomocna funkcija za pravljenje dogadjaja
    static odlazak_korisnika* napravi(korisnik* kor,vreme vn) {
        odlazak_korisnika* ok = new odlazak_korisnika;
        ok->postavi_agenta(0, kor);
        ok->postavi_vreme(simu.vrati_vreme()+vn);
        return ok;
    }
};
```

```
};

// Klasa dogadjaja zavrsetka simulacije
class kraj: public dogadjaj {
public:
    // Konstruktor dogadjaja kraj
    kraj():dogadjaj(0) {}
public:
    // Akcija dogadjaja
    void akcija();
    // Pomocna funkcija za pravljenje dogadjaja
    static kraj* napravi(vreme vn) {
        kraj* k = new kraj;
        k->postavi_vreme(simu.vrati_vreme()+vn);
        return k;
    }
};
```

**Dalje, kreiramo objekte raspodele, reda čekanja, četiri šatera i simulacije koji će kontrolisati izvršenje simulacije.**

```
// Objekat reda cekanja
red salt_red;
// Salteri
resurs salteri(4);
// Objekat simulacije
simulacija simu;
// Raspodele
raspodela rn1(11615), rn2(36517);
```

**Takođe, uvodimo i dve konstante koje će predstavljati srednje vreme dolaska i srednje vreme opsluge entiteta.**

```
// Srednje vreme dolaska entiteta
const double SRVRDOL = 100;
// Srednjevreme opsluge entiteta
const double SRVROPS = 200;
```

**Događaji *DolazakKlijenta* i *OdlazakKlijenta* su gotovo isti kao i u primeru 1. Jedino se menja raspoređivanje događaja obzirom da vremena ne uzimamo iz polja već vreme dobijamo korišćenjem objekata klase raspodela.**

## **U pozivima funkcije rasporedi**

```
// Postavljamo vreme narednog dolaska  
simu.rasporedi(dolazak_korisnika::napravi(novi_kor, rn1.expo(SRVRDOL)));
```

```
// Rasporedi klijenta za kraj opsluge  
simu.rasporedi(odlazak_korisnika::napravi(prvi_kor, rn2.expo(SRVROPS)));
```

**rn1.expo(SRVRDOL) i rn2.expo(SRVROPS) vraćaju vreme između dva dolaska entiteta i vreme trajanja opsluge. Vremenski trenuci nastupanja događaja dolaska novog entiteta i događaja završetka opsluge dobijaju se kada se na vreme između dolazaka i vreme opsluge doda vreme simulacionog časovnika `simu.vrati_vreme()`.**



Pored događaja *DolazakKlijenta* i *OdlazakKlijenta* potrebno realizovati i bezuslovni događaj *KrajSimulacije*. U akciji događaja `kraj` pozivamo funkciju klase `simulacija` `zaustavi()` kojom se zaustavlja simulacija. Prvo izvršenje događaja *KrajSimulacije* dovodi do završetka simulacije.

```
// Događaj kraj simulacije  
void kraj::akcija() {  
    simu.zaustavi();  
}
```

```
procedure KrajSimulacije  
begin  
    Zaustavi simulaciju.  
end
```

# main funkcija

Kao što smo i u prethodnom primeru rekli da bi simulacija uopšte mogla da započne neophodno je napraviti i rasporediti prvi entitet za događaj dolaska klijenta. Postoje dva načina raspoređivanja prvog entiteta. Prvi način je da se prvi entitet rasporedi na događaj dolaska u trenutku  $t=0$ , dok je drugi način, da se dolazak prvog entiteta rasporedi na vreme koje se dobija iz raspodele vremena dolaska (u našem slučaju to je eksponencijalna raspodela sa srednjim vremenom 100 sek). Mi ćemo za raspoređivanje dolaska prvog entiteta koristiti drugi način:

```
// Stvaramo prvog korisnika
korisnik *kor = new korisnik;
```

```
// Postavljamo prvi događaj u listu sto je odgovara dolasku prvog klijenta
simu.rasporedi(dolazak_korisnika::napravi(kor, rn1.expo(SRVRDOL)));
```

Takođe potrebno je na početku potrebno rasporediti i entitet koji će dovesti do završetka simulacije, odnosno koji će izvršiti događaj *KrajSimulacije*.

```
// Rasporedi događaj završetka simulacije
simu.rasporedi(kraj::napravi(43200));
```

## Funkcija main će glasiti

```
int main() {
    // Stvaramo prvog korisnika
    korisnik *kor = new korisnik;
    // Postavljamo prvi događaj u listu što je odgovara dolasku prvog klijenta
    simu.rasporedi(dolazak_korisnika::napravi(kor,rn1.expo(SRVRDOL)));
    // Rasporedi događaj zavrsetka simulacije
    simu.rasporedi(kraj::napravi(43200));
    // Izvršavamo simulaciju
    simu.izvrši();
#ifdef STAMPA
    // Stampamo vreme simulacije
    cout << "vsim" << "," << -1 << "," << simu.vrati_vreme() << endl;
#endif
    return 0;
}
```

## Štampanje izveštaja

Da bi smo mogli da izračunamo statistike neophodno je zabeležiti trenutke u kojima dolazi do promene stanja sistema. Dva stanja pratimo i to su dužina reda čekanja i broj zauzetih kanala opsluge u resursu. Promene koje se dešavaju su sledeće: ulazak u red čekanja (`ured`), izlazak iz reda čekanja (`izred`), ulazak u resurs (`usal`) i izlazak iz resursa (`izsal`). Takođe je potrebno zabeležiti i trenutak prekida simulacije, tj. vreme simulacije (`vsim`). Pored vrste promene i vremena nastupanja potrebno je zabeležiti i redni broj entiteta koji dovodi do promene stanja.

```
cout << "ured" << "," << e->vrati_id() << "," << simu.vrati_vreme_simulacije() << endl;
cout << "izred" << "," << fe->vrati_id() << "," << simu.vrati_vreme_simulacije() << endl;
cout << "usal" << "," << fe->vrati_id() << "," << simu.vrati_vreme_simulacije() << endl;
cout << "izsal" << "," << e->vrati_id() << "," << simu.vrati_vreme_simulacije() << endl;
cout << "vsim" << "," << -1 << "," << simu.vrati_vreme_simulacije() << endl;
```

## Kompajliranje i izvršenje programa

### Microsoft Visual C++

```
cl zad3.cpp -O2 -W4 -DSTAMPA /EHsc  
zad3
```

### MingW C++

```
g++ zad3.cpp -O3 -DSTAMPA -Wall -static -ozad3.exe
```

### Izvršenje

```
zad3 > izvestaj.txt
```

## Zaglavlje aes.h

```
// Zaglavlje aes.h
#ifndef __AES_H__
#define __AES_H__

#include <iostream>
#include <string>
#include <cstdlib>
#include <cstring>

// Uključujemo prostor imena std;
using namespace std;

// Prostor imena esa
namespace aes {
    // Konstante
    const int max_broj_uredu = 10;
    const int max_broj_ulisti = 50;

    // Predefiniramo tip double u vreme
    using vreme = double; // alternativno, typedef double vreme;

    // Klasa agenta
    class agent {
    public:
        virtual ~agent() {}
    public:
        void posalji_poruku(const agent* ap, const string& poruka) {
            const_cast<agent*>(ap)->primi_poruku(this, poruka);
        }
        virtual void primi_poruku(const agent*, const string&) {};
    };

    // Apstraktna klasa dogadjaja
    class dogadjaj {
    protected:
        // Vreme nastupanja
        vreme _vreme;
        // Agent
        agent** _agenti;
        // Broj agenata

```

```

    size_t _broj;
public:
    // Kontruktor
    dogadjaj(const size_t n):_vreme(0.0),_broj(n) {
        _agenti = new agent*[_broj];
    }
    // Destruktor
    virtual ~dogadjaj() {
        delete [] _agenti;
    }
    // Postavi agenta
    void postavi_agenta(const size_t n,const agent* a) {
        if(n<_broj)
            _agenti[n] = const_cast<agent*>(a);
        else {
            cerr << "dogadjaj::postavi_agenta - Pogresan indeks" << endl;
            exit(1);
        }
    }
    // Vрати agenta
    agent* vrati_agenta(const size_t n) const {
        if(n<_broj)
            return _agenti[n];
        else {
            cerr << "dogadjaj::vrati_agenta - Pogresan indeks" << endl;
            exit(1);
        }
        return nullptr;
    }
    // Postavi vreme nastupanja
    void postavi_vreme(const vreme v) {
        _vreme = v;
    }
    // Vрати vreme nastupanja
    vreme vrati_vreme() const { return _vreme; }
public:
    // Cista virtuelna funkcija dogadjaja
    virtual void akcija() = 0;
};
// Klasa simulacija
class simulacija {
private:
    // Flag zaustavljanja

```

```

bool _stop;
// Broj entiteta u LBD listi
size_t _broj;
// Vreme simulacije
vreme _vreme;
// Polje entiteta u LBD listi
dogadjaj* _lbd[max_broj_ulisti];
// Sortiranje LBD - sortiranje spajanjem
void sortiraj_lbd() {
    sortiranje_spajanjem(_lbd,0,_broj-1);
}
void sortiranje_spajanjem(dogadjaj* x[],const size_t l,const size_t r) {
    // Proverava se da li ima elemenata u listi koja se sortira
    if(l<r) {
        // Pivot element
        size_t m = (l+r)/2;
        // Sortira se deo polja do pivota
        sortiranje_spajanjem(x,l,m);
        // Sortira se deo polja od pivota
        sortiranje_spajanjem(x,m+1,r);
        // Spajaju se polja
        spajanje(x,l,m,r);
    }
}
void spajanje(dogadjaj* x[],const size_t l,const size_t m,const size_t r) {
    // Velicina pomocnih polja
    size_t nl = m-l+1, nd = r-m;
    // Pomocna polja
    dogadjaj **levo = new dogadjaj*[nl];
    dogadjaj **desno = new dogadjaj*[nd];
    // Polja koja se spajaju se kopiraju u pomocna polja
    std::memcpy(levo ,x+l ,nl*sizeof(dogadjaj*));
    std::memcpy(desno,x+m+1,nd*sizeof(dogadjaj*));
    // Smestamo elemente iz pomocnih polja u novo, sortirano polje
    size_t i,j,k;
    for(i=0,j=0,k=1; i<nl && j<nd; ++k) {
        // Poredjenje
        if(levo[i]->vrati_vreme()<=desno[j]->vrati_vreme())
            x[k] = levo[i++];
        else
            x[k] = desno[j++];
    }
    // Preostali elementi se smestaju u novo polje

```



```

    if(i<nl) std::memcpy(x+k,levo+i ,(nl-i)*sizeof(dogadjaj*));
    else    std::memcpy(x+k,desno+j,(nd-j)*sizeof(dogadjaj*));
    // Brisemo pomocna polje
    delete [] levo;
    delete [] desno;
}
public:
// Konstruktor
simulacija():_stop(false),_broj(0),_vreme(0.0) {}
// Destruktor
~simulacija() {
    // Uklanjam preostale entitete iz LBD na kraju simulacije.
    for(size_t i=0; i<_broj; ++i)
        delete _lbd[i];
}
// Rasporedjivanje dogadjaja
void rasporedi(const dogadjaj* d) {
    if(_broj < max_broj_ulisti) {
        // Smestamo entitet u listu
        _lbd[_broj] = const_cast<dogadjaj*>(d);
        ++_broj;
        // Sortiramo LBD
        sortiraj_lbd();
    }
    else {
        cerr << "simulacija::rasporedi - Previše entiteta u listi dogadjaja" << endl;
        exit(1);
    }
}
// Izvršavanje simulacije
void izvrsi() {
    // Flag zaustavljanja je netacan
    _stop = false;
    // Izvršavamo simulaciju. Simulacija se završava ukoliko
    // nema dogadjaja u LBD ili ukoliko je flag zaustavljanja tacan.
    do {
        // Vadimo prvi dogadjaj iz liste. Taj dogadjaj postaje tekuci dogadjaj.
        dogadjaj*_tekuci = _lbd[0];
        for(size_t i=1;i<_broj;++i)
            _lbd[i-1] = _lbd[i];
        --_broj;
        // Faza A: Azuriramo vreme simulacije
        _vreme = _tekuci->vrati_vreme();
    }
}

```

```
    // Faza B: Izvrsavamo dogadjaj
    _tekuci->akcija();
    // Brisemo tekuci dogadjaj iz memorije
    delete _tekuci;
} while(_broj && !_stop);
}
// Zaustavljamo izvršenje simulacije
void zaustavi() { _stop = true; }
// Vraca vreme simulacije
vreme vrati_vreme() { return _vreme; }
};
}

#endif // __AES_H__
```

## Zaglavlje aes\_dist.h

```
// Zaglavlje aes_dist.h
#ifndef __AES_DIST_H__
#define __AES_DIST_H__

#include "aes.h"
#include <cmath>

// Uključujemo prostor imena std;
using namespace std;

namespace aes {
    // Klasa raspodela
    class raspodela {
        // Seme generatora slučajnih brojeva (GSB)
        uint32_t s;
        // Multiplikator
        uint32_t a;
        // Modulus
        uint32_t m;
    public:
        // Konstruktor
        raspodela(const uint32_t s):s(s) {
            // Postavljamo vrednost multiplikatora
            a = 16807;
            // Postavljamo vrednost modulusa
            m = 2147483647;
        }
        // Lehmerov multiplikativni kongruentni generator
        double operator() () {
            s = static_cast<uint64_t>(a*s)%m;
            return s/(1.0*m);
        }
        // Uniformna raspodela
        double unif(const double l,const double u) {
            return l+(u-l)*(*this)();
        }
        // Eksponencijalna raspodela
        double expo(const double mi) {
            return -mi*log(*this());
        }
    };
}
```

```
    }  
    // Normalna raspodela  
    double norm(const double mi,const double sd) {  
        double z = 0.0;  
        for(int i=0; i<12; i++)  
            z += (*this)();  
        return mi+(z-6)*sd;  
    }  
};  
}  
#endif // __AES_DIST_H__
```

## Izvorna datoteka zad3.cpp

```
// Datoteka zad3.cpp
#include "aes.h"
#include "aes_proc.h"
#include "aes_dist.h"
#include <iostream>

// Uključujemo prostore imena esa i std
using namespace aes;
using namespace std;

// Objekat reda cekanja
red salt_red;
// Salteri
resurs salteri(4);
// Objekat simulacije
simulacija simu;
raspodela rn1(11615), rn2(36517);
// Srednje vreme dolaska entiteta
const double SRVDOL = 100;
// Srednjevreme opsluge entiteta
const double SRVOPS = 200;

// Klasa korisnika
class korisnik:public agent {
    // Brojac korisnika
    static size_t _idc;
    // Redni broj korisnika
    size_t _id;
public:
    // Konstruktor
    korisnik():agent() { _id = ++_idc; }
public:
    // Vraca id korisnika
    size_t vrati_id() { return _id; }
};
size_t korisnik::_idc = 0;

// Klasa dogadjaja dolazak korisnika
class dolazak_korisnika:public dogadjaj {
```

```

public:
    // Konstruktor
    dolazak_korisnika():dogadjaj(1) {}
    // Akcija
    void akcija();
    // Pomocna funkcija za pravljenje dogadjaja
    static dolazak_korisnika* napravi(korisnik* kor,vreme vn) {
        dolazak_korisnika* dk = new dolazak_korisnika;
        dk->postavi_agenta(0,kor);
        dk->postavi_vreme(simu.vrati_vreme()+vn);
        return dk;
    }
};

// Klasa dogadjaja odlazak korisnika
class odlazak_korisnika:public dogadjaj {
public:
    // Konstruktor
    odlazak_korisnika():dogadjaj(1) {}
    // Akcija dogadjaja
    void akcija();
    // Pomocna funkcija za pravljenje dogadjaja
    static odlazak_korisnika* napravi(korisnik* kor,vreme vn) {
        odlazak_korisnika* ok = new odlazak_korisnika;
        ok->postavi_agenta(0,kor);
        ok->postavi_vreme(simu.vrati_vreme()+vn);
        return ok;
    }
};

// Klasa dogadjaja zavrsetka simulacije
class kraj: public dogadjaj {
public:
    // Konstruktor dogadjaja kraj
    kraj():dogadjaj(0) {}
public:
    // Akcija dogadjaja
    void akcija() { simu.zaustavi(); }
    // Pomocna funkcija za pravljenje dogadjaja
    static kraj* napravi(vreme vn) {
        kraj* k = new kraj;
        k->postavi_vreme(simu.vrati_vreme()+vn);
        return k;
    }
};

```

```

}
};

// Akcija
void dolazak_korisnika::akcija() {
    korisnik *kor,*prvi_kor,*novi_kor;
    // Korisnik koji dolazi u postu
    kor = dynamic_cast<korisnik*>(vrati_agenta(0));
#ifdef STAMPA
    cout << "ured" << ", " << kor->vrati_id() << ", " << simu.vrati_vreme() << endl;
#endif
    // Postavi klijenta u red cekanja
    salt_red.smesti(kor);
    // Ukoliko je salter raspoloziv
    if(salteri.raspoloziv()) {
        // Izvadi prvog klijenta iz reda
        prvi_kor = dynamic_cast<korisnik*>(salt_red.prednji());
        // Prvi korisnik izlazi iz reda
        salt_red.vadi();
#ifdef STAMPA
        cout << "izred" << ", " << kor->vrati_id() << ", " << simu.vrati_vreme() << endl;
#endif
        // Zauzmi jedno mesto u salteru
        salteri.zauzmi();
#ifdef STAMPA
        cout << "usal" << ", " << kor->vrati_id() << ", " << simu.vrati_vreme() << endl;
#endif
        // Rasporedi klijenta za kraj usluge
        simu.rasporedi(odlazak_korisnika::napravi(prvi_kor,rn2.expo(SRVROPS)));
    }
    // Stvaramo narednog korisnika
    novi_kor = new korisnik;
    // Postavljamo vreme narednog dolaska
    simu.rasporedi(dolazak_korisnika::napravi(novi_kor,rn1.expo(SRVRDOL)));
}

// Akcija dogadjaja
void odlazak_korisnika::akcija() {
    korisnik *kor, *prvi_kor;
    // Korisnik koji odlazi
    kor = dynamic_cast<korisnik*>(vrati_agenta(0));
    // Vрати salter
    salteri.oslobodi();
}

```

```

#ifdef STAMPA
cout << "izsal" << ", " << kor->vrati_id() << ", " << simu.vrati_vreme() << endl;
#endif
// Klijent odlazi iz poste - unistavamo tekuceg klijenta
delete kor;
// Ukoliko red nije prazan
if(salt_red.vrati_velicinu(>0) {
    // Izvadi prvog korisnika iz reda
    prvi_kor = dynamic_cast<korisnik*>(salt_red.prednji());
    salt_red.vadi();
#ifdef STAMPA
    cout << "izred" << ", " << prvi_kor->vrati_id() << ", " << simu.vrati_vreme() << endl;
#endif
    // Zauzmi jedno mesto u salteru
    salteri.zauzmi();
#ifdef STAMPA
    cout << "usal" << ", " << prvi_kor->vrati_id() << ", " << simu.vrati_vreme() << endl;
#endif
    // Rasporedi klijenta za kraj opsluge
    simu.rasporedi(odlazak_korisnika::napravi(prvi_kor,rn2.expo(SRVROPS)));
}
}

int main() {
    // Stvaramo prvog korisnika
    korisnik *kor = new korisnik;
    // Postavljamo prvi dogadja u listu sto je odgovara dolasku prvog klijenta
    simu.rasporedi(dolazak_korisnika::napravi(kor,rn1.expo(SRVRDOL)));
    // Rasporedi dogadja zavrssetka simulacije
    simu.rasporedi(kraj::napravi(43200));
    // Izvrsavamo simulaciju
    simu.izvrsi();
#ifdef STAMPA
    // Stampamo vreme simulacije
    cout << "vsim" << ", " << -1 << ", " << simu.vrati_vreme() << endl;
#endif
    return 0;
}

```



# Statistike simulacionog programa

## Statistike resursa

$R(t)$ ,	trenutni broj klijenata u resursu
$J_R$ ,	broj promena stanja u resursu u toku simulacije
$t_R^i$ ,	vremenski trenuci u kojima se menja stanje resursa
$N_R$ ,	broj entiteta koji je ušao u resurs u toku simulacije
$n_R$ ,	broj kanala opsluge
$W_R^i$ ,	vreme opsluge $i$ -tog entiteta
$T$ ,	vreme simulacije

1. Srednji broj zauzetih kanala:  $\bar{R} = \frac{1}{T} \int_0^T R(\tau) d\tau = \frac{1}{T} [\sum_{i=1}^{J_R-1} R(t_R^i)(t_R^{i+1} - t_R^i) + R(t_R^J)(T - t_R^J)]$

2. Srednje vreme opsluge:  $\bar{W}_R = \frac{\sum_{i=1}^{N_R-R(T)} W_R^i}{N_R - R(T)}$

3. Iskorišćenost:  $\rho = \frac{\bar{R}}{n_R}$

## Statistike reda čekanja

$Q(t)$ ,	trenutni broj klijenata u redu čekanja
$J_Q$ ,	broj promena stanja u redu čekanja u toku simulacije
$t_Q^i$ ,	vremenski trenuci u kojima se menja stanje reda čekanja
$N_Q$ ,	broj entiteta koji je ušao u red čekanja u toku simulacije
$N'_Q$ ,	broj entiteta koji je ušao u red čekanja, a nije se zadržavao u redu
$W_Q^i$ ,	vreme čekanja $i$ -tog entiteta
$T$ ,	vreme simulacije

1. Srednja dužina reda čekanja:  $\bar{Q} = \frac{1}{T} \int_0^T Q(\tau) d\tau = \frac{1}{T} \left[ \sum_{i=1}^{J_Q-1} Q(t_Q^i)(t_Q^{i+1} - t_Q^i) + Q(t_Q^J)(T - t_Q^J) \right]$

2. Srednje vreme čekanja:  $\bar{W}_Q = \frac{\sum_{i=1}^{N_Q - Q(T)} W_Q^i}{N_Q - Q(T)}$

3. Srednje vreme čekanja u redu isključujući one entitete koji se nisu zadržavali u redu

$$\bar{W}_Q = \frac{\sum_{i=1}^{N_Q - N'_Q - Q(T)} W_Q^i}{N_Q - N'_Q - Q(T)}$$

## Parametri SMO M/M/n/∞

$\lambda$ , intenzitet dolazaka  
 $\mu$ , intenzitet opsluživanja

1. Verovatnoća da je sistem prazan:  $P_0 = \left[ \sum_{j=1}^{n-1} \left(\frac{\lambda}{\mu}\right)^j \frac{1}{j!} + \frac{1}{n!} \left(\frac{\lambda}{\mu}\right)^n \frac{n\lambda}{n\mu-\lambda} \right]^{-1}$
2. Srednja dužina reda čekanja:  $\bar{k}_R = \frac{P_0 \lambda \mu (\lambda/\mu)^n}{(n-1)!(n\mu-\lambda)^2}$
3. Srednje vreme čekanja u redu:  $\bar{t}_R = \frac{\bar{k}_R}{\lambda}$
4. Srednje vreme opsluge:  $\bar{t}_o = \frac{1}{\mu}$
5. Iskorišćenost:  $\rho = \frac{\lambda}{n\mu}$

## Listing simulacionog programa realizovanog u GPSS/H

Student GPSS/H Release 3.70 (UN214) 22 Dec 2017 08:28:38 File: zad1.gps

Line#	Stmt#	If Do	Block#	*Loc	Operation	A,B,C,D,E,F,G	Comments
1	1				SIMULATE		
2	2				STORAGE S(1),4		
3	3		1		GENERATE RVEXPO(3,100)		
4	4		2		QUEUE 1		
5	5		3		ENTER 1		
6	6		4		DEPART 1		
7	7		5		ADVANCE RVEXPO(7,200)		
8	8		6		LEAVE 1		
9	9		7		TERMINATE		
10	10			*			
11	11		8		GENERATE 43200		
12	12		9		TERMINATE 1		
13	13				START 1		
14	14				END		

Entity Dictionary (in ascending order by entity number; "\*" => value conflict.)

Queues: 1

Storages: 1

Random Numbers: 3 7

Symbol	Value	EQU Defns	Context	References by	Statement Number
1	1		Queue	4	6
1	1	2	Storage	5	8
3	3		Random Nnbr	3	
7	7		Random Nnbr	7	

Storage Requirements (Bytes)

Compiled Code: 336  
Compiled Data: 80  
Miscellaneous: 0

```

Entities:      896
Common:       10000
-----
Total:        11312

```

GPSS/H Model Size:

```

Control Statements  4
Blocks             9

```

Simulation begins.

Relative Clock: 43200.0000    Absolute Clock: 43200.0000

```

Block Current  Total
1              409
2              409
3              409
4              409
5              1   409
6              408
7              408
8              1
9              1

```

```

--Avg-Util-During--
Storage Total Avail Unavl  Entries  Average  Current  Percent  Capacity  Average  Current
Maximum  Time  Time  Time          Time/Unit  Status  Avail          Contents  Contents
Contents
1  0.478          409  202.096  AVAIL  100.0          4          1.913          1
4

```

```

Queue Maximum Average Total Zero Percent Average $Average Qtable
Current Contents Contents Entries Entries Zeros Time/Unit Time/Unit Number
Contents
1 5 0.109 409 354 86.6 11.496 85.491
0

```

Random Stream	Antithetic Variates	Initial Position	Current Position	Sample Count	Chi-Square Uniformity
3	OFF	300000	300410	410	0.69
7	OFF	700000	700409	409	0.84

Status of Common Storage

9448 bytes available  
552 in use  
1528 used (max)

Simulation complete. Absolute Clock: 43200.0000

Total Block Executions: 2863

Blocks / second: 8559557

Microseconds / Block: 0.12

Elapsed Time Used (Sec)

Pass1:	0.00
Sym/Xref	0.00
Pass2:	0.00
Load/Ctrl:	0.00
Execution:	0.00
Output:	0.00
-----	
Total:	0.00

## Rezultati dobijeni u programskom jeziku C++

Vreme simulacije: 43200.000

### STATISTIKE REDA CEKANJA

Broj entiteta koji je usao u red cekanja: 428

Preostali broj entiteta u redu cekanja: 0

Maksimalni broj entiteta u redu: 5

Srednje vreme cekanja u redu: 18.254

Srednji broj entiteta u redu: 0.252

Broj entiteta koji je prosao kroz red bez zadrzavanja: 348

Procenat ulaza bez zadrzavanja: 81.308

Srednje vreme cekanja u redu (iskljuc. ent. koji se nisu zadrzali): 97.661

### STATISTIKE RESURSA

Broj entiteta koji je usao u resurs: 428

Preostali broj entiteta u resursu: 1

Maksimalni broj zauzetih kanala opsluge: 4

Srednje vreme opsluge: 214.647

Srednji broj zauzetih kanala: 2.128

Iskoriscenost: 0.532

## Uporedne statistike reda čekanja i resursa (skladišta) dobijene simulacionim programom realizovanim u GPSS/H i C++

### Uporedne statistike reda čekanja

Statistike reda čekanja	GPSS/H	C++	Analitičke vrednosti
Broj entiteta koji je ušao u red čekanja	409	431	432
Preostali broj entiteta u redu čekanja	0	0	-
Maksimalni broj entiteta u redu	5	5	-
Srednje vreme čekanja u redu	11.496	15.049	16.667
Srednji broj entiteta u redu	0.109	0.220	0.167
Broj entiteta koji je prošao kroz red bez zadržavanja	354	350	-
Procenat ulaza bez zadržavanja	86.6	81.206	-
Srednje vreme čekanja u redu (isključ. ent. koji se nisu zadržali)	85.491	80.075	-

### Uporedne statistike resursa (skladišta)

Statistike resursa (skladišta)	GPSS/H	C++	Analitičke vrednosti
Broj entiteta koji je ušao u resurs	409	431	432
Preostali broj entiteta u resursu	1	4	-
Maksimalni broj zauzetih kanala opsluge	4	4	-
Srednje vreme opsluge	202.096	194.199	200.000
Srednji broj zauzetih kanala	1.913	1.938	2.167
Iskorišćenost	0.478	0.485	0.500



## Kompajliranje i izvršenje programa

```
cl stat_zad3.cpp /EHsc
```

```
stat_zad3
```

## Program stat\_zad3.cpp

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

const int max_id_num = 20;

class resurs_stat {
public:
    // Ukupni, tekuci i maksimalni broj entiteta u resursu
    int broj, tek_broj, max_broj;
    // Ukupno vreme za koje je resur bio zauzet
    double ukupno_vreme;
    // Vreme za koje se entitet zadrzava u resursu
    double površina_čekanja;
    // Poslednji trenutak promene stanja entiteta u resurs
    double vreme_promene;
    // Ukupno vreme simulacije
    double vreme_sim;
    // Polje rednih brojeva entiteta koji se nalaze u resursu
    int id[max_id_num];
    // Vremenski trenutak promene stanja resursa od strane entiteta
    double vreme[max_id_num];
private:
    // Broj entiteta u polju id i polju vreme
    int broj_id;
public:
    resurs_stat(): broj(0), tek_broj(0), max_broj(0),
        ukupno_vreme(0), površina_čekanja(0),
        vreme_promene(0), broj_id(0) {}
    // Dodajemo redni broj entiteta u polje id
    void dodaj_id(int novi_id, double novo_vreme) {
        if (broj_id < max_id_num) {
            id[broj_id] = novi_id;
            vreme[broj_id] = novo_vreme;
            broj_id++;
        }
        else {
            cerr << "Greska dodaj id" << endl;
        }
    }
};
```

```

        exit(1);
    }
}
// Nalazimo redni broj entiteta u polju id
int nadji_id(int trazi_id) {
    int i;
    for(i=0; i<broj_id; i++)
        if(id[i]==trazi_id)
            break;
    return i;
}
// Uklanjammo redni broj i tekuće vreme entiteta iz polja id i vreme sa pozicije pos
void ukloni_id(int pos) {
    for(int i=pos+1; i<broj_id; i++) {
        id[i-1] = id[i];
        vreme[i-1] = vreme[i];
    }
    broj_id--;
}
void vreme_simulacije(double vs) { vreme_sim = vs; }
inline double srednje_vreme() {
    return ukupno_vreme/(broj-tek_broj);
}
inline double srednji_broj_entiteta() {
    return površina_čekanja/vreme_sim;
}
inline double iskoriscenost(int broj_kanala) {
    return srednji_broj_entiteta()/broj_kanala;
}
};
// Klasa statistika reda čekanja
class fifo_red_stat: public resurs_stat {
public:
    // Broj klijenata koji nisu čekali
    int broj_bezčekanja;
public:
    fifo_red_stat():resurs_stat(),broj_bezčekanja(0) {}
    inline double procenat_entiteta_bez_zadržavanja() {
        return (100.0*broj_bezčekanja)/broj;
    }
    inline double srednji_broj_entiteta_bez_zadržavanja() {
        return ukupno_vreme/(broj-tek_broj-broj_bezčekanja);
    }
};

```

```

    }
};
int main() {
    int pos, posn, idv;
    string str, tip, id, vreme;
    char line[1024];
    resurs_stat salt;
    fifo_red_stat red;
    double vreme_v, vreme_sim,
           vreme_u_redu, vreme_u_res;
    // Ucitavamo izvestaj
    ifstream in("zad3.txt", ifstream::in);
    // Ucitavamo podatke iz izvestaja sve dok ne dodjemo do kraja datoteke
    while(in.good()) {
        // Ucitavamo liniju iz izvestaja
        in.getline(line, 1024);
        // Dodeljujemo liniju stringu
        str = line;
        // Izdvajamo tip obavestenja
        posn = str.find_first_of(',');
        tip = str.substr(0, posn);
        pos = posn;
        // Izdvajamo vrednost rednog broja entiteta
        posn = str.find_first_of(',', pos+1);
        id = str.substr(pos+1, posn-pos-1);
        pos = posn;
        // Izdvajamo vremenski trenutak
        vreme = str.substr(pos+1);
        // Prevodimo id i vremenski trenutak u broj
        idv = atoi(id.c_str());
        vreme_v = atof(vreme.c_str());
        // Proveravamo tip obavestenja
        if(!tip.compare("ured")) {
            // Broj entiteta koji su usli u red cekanja
            red.broj++;
            // Uvecavamo ukupno cekanje u redu
            red.povrsina_cekanja += red.tek_broj*(vreme_v-red.vreme_promene);
            // Pamtimo trenutak dolaska entiteta u red
            red.vreme_promene = vreme_v;
            // Uvecavamo broj entiteta u redu
            red.tek_broj++;
            // Ukoliko je tekuci broj clijenata u redu veci od maksimalnog

```

```

// tekuci broj postaje maksimalni broj
if(red.tek_broj>red.max_broj)
    red.max_broj = red.tek_broj;
// Pamtim redni broj i trenutak ulaska entiteta u redu
red.dodaj_id(idv,vremev);
}
else if(!tip.compare("izred")) {
// Nalazimo poziciju entiteta u redu
int n = red.nadji_id(idv);
// Izracunavamo vreme cekanja u redu
vreme_u_redu = vremev-red.vreme[n];
// Ukupno vreme koje su entiteti proveli u redu
red.ukupno_vreme += vreme_u_redu;
//
red.povrsina_cekanja += red.tek_broj*(vremev-red.vreme_promene);
// Ukoliko je vreme koje je entitet proveo u redu 0
if(vreme_u_redu==0.0)
// Uvecavamo broj entiteta koji nisu cekali u redu
red.broj_bezcekanja++;
// Uklanjam redni broj entiteta sa pozicije n
red.ukloni_id(n);
// Umanjujemo tekuci broj entiteta u redu
red.tek_broj--;
}
else if(!tip.compare("usal")) {
// Uvecamo broj entiteta koji su usli resurs
salt.broj++;
//
salt.povrsina_cekanja += salt.tek_broj*(vremev-salt.vreme_promene);
// Pamtim trenutak poslednje promene broja zauzetih mesta
salt.vreme_promene = vremev;
// Uvecavamo tekuci broj zauzetih mesta u salteru
salt.tek_broj++;
// Ukoliko je tekuci broj zauzetih mesta veci od maksimalnog
// tekuci broj postaje maksimalni broj
if(salt.tek_broj>salt.max_broj)
    salt.max_broj = salt.tek_broj;
// Pamtim redni broj entiteta u resursu i trenutak ulaska entiteta u resurs
salt.dodaj_id(idv,vremev);
}
else if(!tip.compare("izsal")) {
// Nalazimo poziciju entiteta u resursu

```

```

int n = salt.nadji_id(idv);
// Vreme koje entitet provodi u resursu
vreme_u_res = vreme-v-salt.vreme[n];
// Ukupno vreme koje su entiteti proveli u salteru
salt.ukupno_vreme += vreme_u_res;
//
salt.povrsina_cekanja += salt.tek_broj*(vreme-v-salt.vreme_promene);
// Pamtimo vreme ulaska entiteta u resurs
salt.vreme_promene = vreme;
// Uklanjammo redni broj entiteta sa pozicije n
salt.ukloni_id(n);
// Umanjujemo tekuci broj entiteta u resursu
salt.tek_broj--;
}
else if(!tip.compare("vsim")) {
// Pamtimo vreme simulacije
vreme_sim = vreme;
// Postavljamo vreme simulacije
red.vreme_simulacije(vreme);
salt.vreme_simulacije(vreme);
//
salt.povrsina_cekanja += salt.tek_broj*(vreme-v-salt.vreme_promene);
red.povrsina_cekanja += red.tek_broj*(vreme-red.vreme_promene);
}
}
// Zatvaramo datoteku
in.close();
// Podesavamo preciznost izlaznih rezultata
cout.precision(3);
cout.setf(ios_base::fixed);
// ISPISUJEMO STATISTIKE
// Ispisujemo vreme simulacije
cout << "Vreme simulacije: " << vreme_sim << endl;
cout << endl;
// Statistike reda cekanja
cout << "STATISTIKE REDA CEKANJA" << endl;
cout << "Broj entiteta koji je usao u red cekanja: " << red.broj << endl;
cout << "Preostali broj entiteta u redu cekanja: " << red.tek_broj << endl;
cout << "Maksimalni broj entiteta u redu: " << red.max_broj << endl;
cout << "Srednje vreme cekanja u redu: " << red.srednje_vreme() << endl;
cout << "Srednji broj entiteta u redu: " << red.srednji_broj_entiteta() << endl;
cout << "Broj entiteta koji je prosao kroz red bez zadržavanja: " << red.broj_bezcekanja << endl;

```

```
cout << "Procenat ulaza bez zadržavanja: " << red.procenat_entiteta_bez_zadržavanja() << endl;
cout << "Srednje vreme cekanja u redu (iskljuc. ent. koji se nisu zadržali): "
    << red.srednji_broj_entiteta_bez_zadržavanja() << endl;
cout << endl;
// Statistike resursa
cout << "STATISTIKE RESURSA" << endl;
cout << "Broj entiteta koji je usao u resurs: " << salt.broj << endl;
cout << "Preostali broj entiteta u resursu: " << salt.tek_broj << endl;
cout << "Maksimalni broj zauzetih kanala opsluge: " << salt.max_broj << endl;
cout << "Srednje vreme opsluge: " << salt.srednje_vreme() << endl;
cout << "Srednji broj zauzetih kanala: " << salt.srednji_broj_entiteta() << endl;
cout << "Iskoriscenost: " << salt.iskoriscenost(4) << endl;
return 0;
}
```