Osnovne akademske studije

PREDMET: Objektno-orijentisana simulacija

TEMA: Strategija raspoređivanja događaja 1

Predmetni nastavnik: Prof. dr Milorad Stanojević
Asistent: mr Marko Đogatović

Simulaciona metodologija

Stanje modela sistema zasnovanog na diskretnoj stohastičkoj simulaciji menjaju događaji koji se odigravaju u diskretnim vremenskim trenucima.

Svaki momenat u vremenu u kome se jedan ili više događaja odigravaju naziva se *vremenskim trenutkom*.

Razlikujemo dva vremena simulatora: relativno i apsolutno.

Relativno vreme simulacije je vreme od početka po završetka neke faze u toku simulacije (npr. radni dan) nakon čega se vreme resetuje (počinje od nule).

Apsolutno vreme simulacije je vreme od početka do kraja simulacije.

Vreme simulacije se uvek pomera iz vremena izvršenja prethodnog u vreme izvršenja prvog narednog događaja (mehanizam pomaka na naredni dogadjaj).

Objekti čije se aktivnosti modeliraju u simulacionom programu nazivaju se *entitetima*, koji su opisani *atributima* (svojstvima). Atributa može biti više, ali dva su osnovna: *vreme odigravanja narednog događa* i *naredni događaj*.

***Resursi* su objekti simulacionog modela čija je osnovna uloga da ograniče aktivnosti entiteta. Resursi imaju na**

raspolaganju jedno ili više mesta u koje primaju enititete. Entiteti zauzimaju i oslobađaju mesta u resursu.

Događaj **je diskretna promena stanja entiteta u određenom vremenskom trenutku. Između dva događaja stanje se ne menja. Razlikujemo dve vrste događaja:**

1. ***bezuslovni događaj*** **– događaj čije se vreme odigravanja može unapred predvideti.**

2. ***uslovni događaj*** **– događaj čije izvršenje zavisi od ispunjenja određenog uslova.**

Strategija raspoređivanje događaja (dvofazna strategija)

Strategija raspoređivanja događaja je tako koncipirana da se entiteti i njihovi događaji planiraju unapred i drže u *listi budućih događaja (LBD)*, sortirani prema vremenu.

Strategiju je moguće opisati na sledeći način:

1. Sa liste budućih događaja se uzima prvi entitet i uklanja iz liste. Taj entitet postaje *tekući enitet*.

2. Faza 1: Simulacioni sat se ažurira na vreme nastupanja događaja tekućeg entiteta.

3. Faza 2: Nakon toga se poziva odgovarajuća procedura koja izvršava događaj tekućeg entiteta.

Nakon izvršenja događaja tekućeg entiteta moguće je da se desi da tekući entitet bude ponovo *raspoređen* u LBD listu ili da novi entiteti budu raspoređeni u listu.

Raspoređivanje entiteta se vrši po sledećoj proceduri:

1. Entitetu se zada budući događaj i vreme njegovog izvršenja.

2. Entitet se postavlja u LBD listu.

3. LBD lista se sortira po vremenu nastupanja.

Primer 1. Posmatrajmo poštu u kojoj se nalazi četiri univerzalna šaltera ispred koga se formira zajednički red red čekanja. Vremenski trenuci dolaska kao i vreme opsluge prvih trinaest klijenata dati su u tabeli 1.

U tabeli 2 dati su dolasci klijenata, sadržaj zajedničkog reda čekanja, klijenti u sistemu, kao i vremenski trenuci odlaska klijenata. Ova tabela je dobijena ručnim rešavanjem simulacije.

Tabela 1: Vremenski trenutak dolaska klijenata u poštu i vreme opsluge na jednom od četiri šaltera

Klijent	Vreme dolaska (min)	Vreme opsluge (min)
1	0	4
2	1	3
3	1	4
4	1	3
5	2	1
6	4	4
7	4	2
8	4	1
9	4	2
10	5	1
11	5	1
12	6	1
13	8	3

Tabela 2: Vremenski trenutak dolaska klijenata u poštu i vreme opsluge na jednom od četiri šaltera

Trenutak (min)	Dolazak klijenata	Red čekanja	Klijenti u pošti	Odlasci klijenata
0	1	-	1	-
1	2,3,4	-	1,2,3,4	-
2	5	5	1,2,3,4	-
3	-	5	1,2,3,4	-
4	6,7,8,9	8,9	3,5,6,7	1,2,4
5	10,11	10,11	6,7,8,9	3,5
6	12	12	6,9,10,11	7,8
7	-	-	6,12	9,10,11
8	13	-	13	6,12
9	-	-	13	-
10	-	-	13	-
11	-	-	-	13

Bezuslovni događaji – Primer 1

```
procedure DolazakKlijenta
begin
    Postavi kljenta u red čekanja.
    if šalter je raspoloživ then
    begin
        Izvadi prvog klijenta iz reda čekanja.
        Zauzmi šalter.
        Rasporedi klijenta na događaj OdlazakKlijenta.
    end
    Napravi novog klijenta.
    Rasporedi klijenta na događaj DolazakKlijenta.
end
```

Kod ovog događaja kod svakog dolaska klijenta u poštu potrebno je rasporediti događaj dolaska narednog klijenta. Znači, svaki dolazak entiteta iziskuje i respoređivanje dolaska narednog entiteta.

```
procedure OdlazakKlijenta
begin
        Oslobodi šalter.
        Uništi klijenta.
        if red čekanja nije prazan then
        begin
                Izvadi prvog klijenta iz reda čekanja.
                Zauzmi šalter.
                Rasporedi klijenta na događaj OdlazakKlijenta.
        end
end
```

Kod ovog događaja prilikom oslobađanja šaltera i odlaska klijenta iz pošte potrebno je proveriti da li ima kljijenata u redu čekanja i ukoliko ima prvog klijenta iz reda poslati na opslugu na šalterima.

Tabela 3: Stanje LBD liste i reda čekanja nakon izvršenja događaja u 6 minutu.

Faza	Događaj	LBD*	Entiteti u redu čekanja	Klijenti na šalterima	Komentar
Ažuriranje vremena	-	[1,12,6] [2,7,6] [2,8,6] [2,9,7] [2,6,8]	10,11	7,8,9,6	Vreme časovnika je 5. Na početku liste je entitet 12 sa vremenom nastupanja 6. Ažuriramo časovnik na vreme nastupanja tekućeg entiteta. Vreme časovnika je 6.
Izvršavanje događaja	DolazakKlijenta	[2,7,6] [2,8,6] [2,9,7] [2,6,8] [1,13,8]	10,11,12	7,8,9,6	Izvršavamo događaj DolazakKlijenata za entitet 12. Pošto je resurs neraspoloživ smeštamo entitet 12 u red čekanja. Stvaramo entitet 13 i raspoređujemo ga na DolazakKlijenata u trenutku 8.
	OdlazakKlijenta	[2,8,6] [2,9,7] [2,10,7] [2,6,8] [1,13,8]	11,12	10,8,9,6	Izvršavamo događaj OdlazakKlijenata za entitet 7. Oslobađamo šalter. Uništavamo entitet 7. Iz reda čekanja vadimo entitet 10, zauzimamo mesto u šalteru i raspoređujemo entitet 10 na OdlazakKlijenta u trenutku 7.
	OdlazakKlijenta	[2,9,7] [2,10,7] [2,11,7] [2,6,8] [1,13,8]	12	10,11,9,6	Izvršavamo događaj OdlazakKlijenata za entitet 8. Oslobađamo šalter. Uništavamo entitet 8. Iz reda čekanja vadimo entitet 11, zauzimamo mesto u šalteru i raspoređujemo entitet 11 na OdlazakKlijenta u trenutku 7.

*** [događaj (1 – DolazakKlijenta; 2 – OdlazakKlijenta), broj entiteta, vreme nastupanja događaja]**

Klasa entitet

```
// Klasa entiteta
class entitet {
  // Omogucavamo klasi simulacija da pristupi privatnim clanovima klase entitet
  friend class simulacija;
  // Omogucavamo klasi fifo_red da pristupi privatnim clanovima klase entitet
  friend class fifo_red;
  // Redni broj entiteta
  int id;
  // Buduci dogadjaj
  int naredni_dogadjaj;
  // Vreme nastupanja buduceg dogadjaja
  double vreme_nastupanja_dogadjaja;
public:
  // Konstruktor
  entitet(int eid):id(eid) { }
  // Vraca id entiteta
  int vrati_id() { return id; }
};
```

Klasa entitet sadrži tri privatna člana `id`, `naredni_dogadjaj` (tipa `int`) i `vreme_nastupanja_dogadjaja` (tipa `double`). `id` je redni broj entiteta, dok je `naredni_dogadjaj` broj budućeg događaja entiteta (za naš primer može biti 1 ili 2). `vreme_nastupanja_dogadjaja` odredjuje vremenski trenutak kada nastupa budući događaj. Klase `simulacija` i `fifo_red` deklarišemo kao prijateljske da bi mogle da pristupe privatnim članovima klase `entitet`.

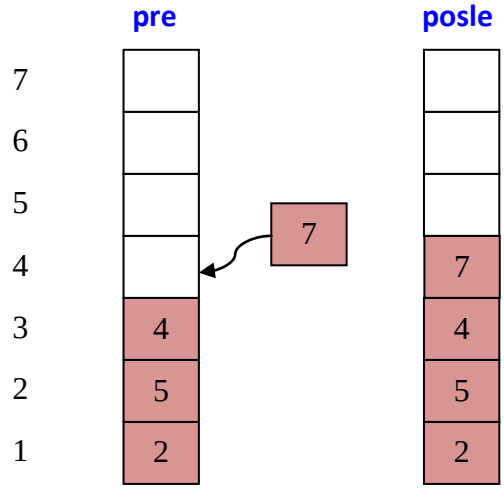
Klasa fifo_red

```
// Klasa FIFO red cekanja
class fifo_red {
  // Broj entiteta u redu cekanja
  int broj;
  // Polje pokazivaca na entitete
  entitet* red[max_broj_uredu];
public:
  // Konstruktor
  fifo_red(): broj(0) {}
  // Smestamo entitet u red cekanja
  void ured(entitet* e) {
    if( broj < max_broj_uredu) {
      // Smestamo entitet na kraj reda cekanja
      red[broj]=e;
      // Uvecavamo broj entiteta za 1
      broj++;
    }
    else {
      cerr << "Previse korisnika u redu cekanja" << endl;
      exit(1);
    }
  }
  // Vadimo entitet iz reda cekanja
  entitet* izred() {
    entitet *e;
    if(broj>0) {
      // Vadimo prvi entitet iz reda cekanja
      e = red[0];
      for(int i=1;i<broj;i++)
```

```
        red[i-1] = red[i];
      // Umanjujemo broj entiteta iz reda cekanja
      broj--;
    }
    else {
      cerr << "Nema entiteta u redu cekanja" << endl;
      exit(1);
    }
    // Vraca entitet
    return e;
  }
  // Vracamo velicinu reda cekanja
  int velicina() { return broj; };
};
```

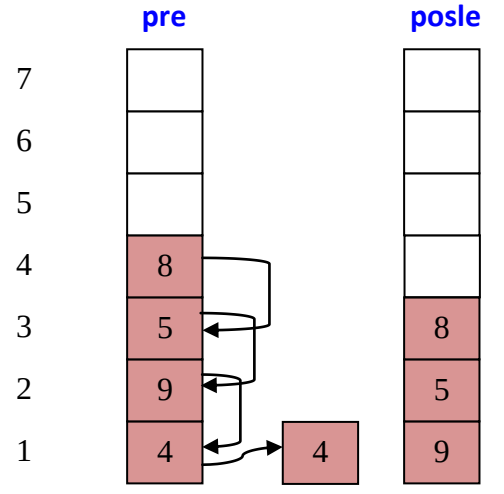
Klasa `fifo_red` sadrži dva privatna člana broj (tipa `int`) i polje pokazivača na entitete red od najviše `max_broj_uredu` članova (10 za naš primer). Broj predstavlja broj entiteta koji se u nekom vremenskom trenutku nalazi u redu čekanja. Polje red je takvo da se na kraj polja ubacuju entitet,i ukoliko je broj entiteta manji od `max_broj_uredu`, a sa početka polja se vade ukoliko ima entiteta u redu (FIFO princip).

Ubacivanje entiteta u red



broj = 3 **broj = 4**

Izbacivanje entiteta iz reda



broj = 4 **broj = 3**

max_broj_uredu = 7

```
// Smestamo entitet na kraj reda cekanja
red[broj]=e;
// Uvecavamo broj entiteta za 1
broj++;
```

```
// Vadimo prvi entitet iz reda cekanja
e = red[0];
for(int i=1;i<broj;i++)
  red[i-1] = red[i];
// Umanjujemo broj entiteta iz reda cekanja
broj--;
```

Klasa resurs

```
// Klasa resursa
class resurs {
  // Tekuci broj zauzetih mesta u resursa
  int broj_zauzetih_mesta;
  // Maksimalni broj  mesta u resursu
  int ukupni_broj_mesta;
public:
  // Konstruktor
  resurs(int b):ukupni_broj_mesta(b),broj_zauzetih_mesta(0) {}
  // Zauzima mesto u resursu
  void zauzmi() {
    if(broj_zauzetih_mesta<ukupni_broj_mesta)
      // Uvecavamo broj zauzetih mesta
      broj_zauzetih_mesta++;
    else {
      cerr << "Sva mesta u resursu su zauzeta" << endl;
      exit(1);
    }
  }
  // Oslobadja mesto u resursu
  void oslobodi() {
    if(broj_zauzetih_mesta>0)
      // Umanjujemo broj zauzetih mesta
      broj_zauzetih_mesta--;
    else {
      cerr << "Nema entiteta u resursu" << endl;
      exit(1);
    }
  }
```

```
    // Raspolozivost resursa (ima makar jedno slobodno mesto u resursu)
    bool raspoloziv() { return (broj_zauzetih_mesta<ukupni_broj_mesta); }
};
```

Klasa resurs sadrži dva privatna člana broj_zauzetih_mesta i `ukupni_broj_mesta` (tipa `int`). `Broj_zauzetih_mesta` predstavlja broj zauzetih mesta u resursu u nekom vremenskom trenutku. `ukupni_broj_mesta` predstavlja maksimalni broj mesta u resursu. Funkcija `zauzmi` uvećava broj zauzetih mesta za 1 ukoliko sva mesta u resursu nisu zauzeta, dok funkcija `oslobodi` oslobađa jedno mesto u resursu. Funkcija `raspoloziv` vraća boolean vrednost true ukoliko nisu sva mesta u resursu zauzeta, u suprotnom vraća false.

Sortiranje izborom

Polje je podeljenjo u dva dela: sortirani deo i nesortirani deo. Na početku sortirani deo je prazan, dok nesortirani sadrži čitavo polje. U svakom koraku, algoritam nalazi najmanji element nesortiranog dela i dodaje ga na kraj sortiranog dela. Kada nesortirani deo postane prazan algoritam se zaustavlja.

Sortiranje izborom nije stabilan algoritam sortiranja. Algoritam je stabilan onda kada zadržava relativan raspored elemenata koji imaju istu vrednosti nakon sortiranja.

Primer: Potrebno je sortirati celobrojno polje 5, -1, 3, 2, 9, 7, 1 u rastućem redosledu koristeći sortiranje izborom. Plavom bojom su označeni nesortirani elementi liste, zelenom sortirani, a crvenom označen je najmanji element nesortiranog dela liste. Strelicom je označena zamena elemenata u listi.

5, -1, 3, 2, 9, 7, 1

5, -1, 3, 2, 9, 7, 1

Sortiranje celog polja algoritmom izbora

```
void sortiranje_izborom(int a[],int n) {
    int min,tmp;
    for(int i=0; i<n-1; i++) {
      // Kazemo da je prvi element najmanji
      min = i;
      // pa onda u petlji trazimo jos manji.
      for(int j=i+1; j<n; j++) {
        if( a[j] < a[min])
          min = j;
      }
       // Ukoliko nadjeni najmanji element nije
       // isti elementu sa pocetka nesortirane liste
       // vrsi se njihova medjusobna zamena.
      if(min!=i) {
        tmp = a[i];
        a[i] = a[min];
        a[min] = tmp;
      }
    }
  }
```

Klasa simulacija

```
class simulacija {
  // Brojac entiteta
  int eid;
  // Terminacioni brojac
  int tb;
  // Broj entiteta u LBD listi
  int broj;
  // Vreme simulacije
  double vreme_simulacije;
  // Polje entiteta u LBD listi
  entitet* lista_nastupanja_entiteta[max_broj_ulisti];
  // Sortiranje LBD - Sortiranje izborom
  void sortiraj_listu_nastupanja() {
    int min;
    entitet *tmp;
    for(int i=0; i<broj-1; i++) {
      min = i;
      for(int j=i+1; j<broj; j++) {
        if( lista_nastupanja_entiteta[j]->vreme_nastupanja_dogadjaja <
            lista_nastupanja_entiteta[min]->vreme_nastupanja_dogadjaja)
          min = j;
      }
      if(min!=i) {
        tmp = lista_nastupanja_entiteta[i];
        lista_nastupanja_entiteta[i] = lista_nastupanja_entiteta[min];
        lista_nastupanja_entiteta[min] = tmp;
      }
    }
  }
```

```
public:
  // Konstruktor
  simulacija():eid(1),tb(0),broj(0),vreme_simulacije(0.0) {}
  // Destruktor
  ~simulacija() {
    // Uklanjamo preostale entitete iz LBD na kraju simulacije.
    for(int i=0; i<broj; i++)
      delete lista_nastupanja_entiteta[i];
  }
  // Pravljenje entiteta
  entitet* napravi_entitet() {
    // Pravimo novi entitet
    return new entitet(eid++);
  }
  // Unistavanje entiteta
  void unisti_entitet(entitet *e, int b) {
    // Unistavamo entitet
    delete e;
    // Umanjujemo terminacioni brojac
    tb -= b;
  }
  // Rasporedjivanje dogadjaja
  void rasporedi(entitet* e,int dog,double vreme) {
    if( broj < max_broj_ulisti) {
      // Postavljamo dogadjaj
      e->naredni_dogadjaj = dog;
      // Postavljamo vreme nastupanja dogadjaja
      e->vreme_nastupanja_dogadjaja = vreme;
      // Smestamo entitet u listu
      lista_nastupanja_entiteta[broj] = e;
      broj++;
      // Sortiramo LBD
```

```
      sortiraj_listu_nastupanja();
    }
    else {
      cerr << "Previse entiteta u listi dogadjaja" << endl;
      exit(1);
    }
  }
  // Izvrsavanje simulacije
  void izvrsi(int b) {
    entitet * tekuci;
    // Postavljamo terminacioni brojac
    tb = b;
    // Izvrsavamo simulaciju. Simulacija se zavrsava ukoliko
    // nema entiteta u LBD ili ukoliko je terminacioni brojac 0.
    do {
      // Vadimo prvi entitet iz liste. Taj entitet postaje tekuci entitet.
      tekuci = lista_nastupanja_entiteta[0];
      for(int i=1;i<broj;i++)
        lista_nastupanja_entiteta[i-1] = lista_nastupanja_entiteta[i];
      broj--;
      // Faza 1: Azurirarmo vreme simulacije
      vreme_simulacije = tekuci->vreme_nastupanja_dogadjaja;
      // Faza 2: Izvrsavamo dogadjaj
      izvrsavanje_dogadjaja(tekuci->naredni_dogadjaj,tekuci);
    } while(broj && tb>0);
  }
  // Izvrsava dogadjaj
  void izvrsavanje_dogadjaja(int dog, entitet* e);
  // Vraca vreme simulacije
  double vrati_vreme_simulacije() { return vreme_simulacije; }
};
```

Klasa `simulacija` ima 4 privatna i 5 javnih članova i jednu privatnu metodu. `eid` je brojač entiteta i služi da postavi redni broj entitetu. Na početku simulacije se postavlja u 1. `tb` je terminacioni brojač koji kontroliše vreme trajanja simulacije. Kada vrednost terminacionog brojača padne na 0 simulacija se prekida. Broj je tekući broj elemenata u LBD listi. `vreme_simulacije` je tekuća vrednost simulacionog časovnika. `lista_nastupanja_entiteta` je polje pokazivača na entitete i predstavlja LBD listu. Velicina `liste_nastupanja_entiteta` je ograničena i iznosi `max_broj_ulisti` (za naš primer vrednost max_broj_ulisti je 50)

Funkcija `napravi_entitet` pravi novi entitet korišćenjem operatora `new`, dodeljuje mu redni broj i vraća pokazivač na novi entitet.

Funkcija `unisti_entitet` korišćenjem operatora `delete` briše prosleđeni entitet `e` iz memorije i umanjuje terminacioni brojač za prosleđenu vrednost `b`.

Funkcija `rasporedi` vrši raspoređivanje entiteta sa budućim događajem u LBD listu (`lista_nastupanja_entiteta`) prema vremenu nastupanja događaja. Prvo se u entitetu postavlja događaj i vreme nastupanja događaja,

```
// Postavljamo dogadjaj
e->naredni_dogadjaj = dog;
// Postavljamo vreme nastupanja dogadjaja
e->vreme_nastupanja_dogadjaja = vreme;
```

nakon toga se entitet ubacuje na kraj `lista_nastupanja_entiteta` na isti način kako se to radi kod reda čekanja,

```
// Smestamo entitet u listu
lista_nastupanja_entiteta[broj] = e;
broj++;
```

da bi se na kraju izvršilo sortiranje entiteta u listi prema vremenu nastupanja

```
// Sortiramo LBD
sortiraj_listu_nastupanja();
```

Ovom prilikom se poziva privatna funkcija `sortiraj_listu_nastupanja` koja sortira listu prema vremenu nastupanja koristeći algoritam sortiranja izborom.

Funkcija `izvrsi` izvršava simulaciju koristeći algoritam raspoređivanja događaja koji je dat na početku prezentacije. Prvo postavlja vrednost terminacionog brojača. Vrednost terminacionog brojača se umanjuje do 0 pozivima funkcije `unisti_entitet`. Nakon toga ulazimo u petlju koja se izvršava sve dok ima entiteta u `lista_nastupanja_entiteta` (tj. `broj` nije 0) i dok je terminacioni brojač veći od 0. Unutar petlje se prvo vadi prvi element LBD liste koji postaje tekući element. Element se vadi iz `lista_nastupanja_entiteta` na isti način kako se to radi kod klase `fifo_red`.

```
// Vadimo prvi entitet iz liste. Taj entitet postaje tekuci entitet.
tekuci = lista_nastupanja_entiteta[0];
for(int i=1;i<broj;i++)
  lista_nastupanja_entiteta[i-1] = lista_nastupanja_entiteta[i];
broj--;
```

Nakon toga se izvršava prva faza strategije raspoređivanja događaja u kojoj se ažurira simulacioni časovnik na vreme nastupanja tekućeg elementa.

```
// Faza 1: Azurirarmo vreme simulacije
vreme_simulacije = tekuci->vreme_nastupanja_dogadjaja;
```

Potom se realizuje druga faza strategije u kojoj se izvršava nastupajući događaj tekućeg entiteta.

```
// Faza 2: Izvrsavamo dogadjaj
izvrsavanje_dogadjaja(tekuci->naredni_dogadjaj,tekuci);
```

Ovom prilikom se poziva funkcija `izvrsavanje_dogadjaja`. `Izvrsavanje_dogadjaja` se realizuje posebno i zavisi od broja i vrste bezuslovnih događaja koji se pozivaju njom pozivaju.

Funkcija `vrati_vreme_simulacije` vraća vreme simulacije (vrednost simulacionog časovnika).

U slučaju da neki od uslova koji štite simulaciju od pojave greške (najčešće prekoračenja veličine polja) i nekontrolisanog prekida programa nije zadovoljen, na standardnom izlazu za greške (`cerr`) prikazuje se tekstualna poruka o grešci i program se prekida (`exit(1)`).

Sve do sada navedene klase realizovane su u zaglavlju `simu1.h`, unutar prostora imena (`namespace`) `simu1`.

```
// Zaglavlje simu1.h
#include <iostream> // Treba nam zbog cout i cerr
#include <cstdlib>  // Treba nam zbog exit

// Koristimo prostor imena STL biblioteke
using namespace std;

// Prostor imena simu1
namespace simu1 {
  // Realizicije klasa entitet, fifo_red, resurs i simulacija
  // ...
}
```

Program dalje realizujemo u izvornoj datoteci simu1.cpp. Na početku simu1.cpp učitavaju se odgovarajuća zaglavlja i dozvoljava se korišćenje odgovarajućih prostora imena.

```
#include "simu1.h"  // Treba nam zbog entitet, fifo_red, resurs i simulacija
#include <iostream> // Treba nam zbog cout

using namespace simu1;
using namespace std;
```

Prvo postavljamo redne brojeve događaja i potpise funkcija koje realizuju te bezuslovne događaje. Potpisi su na nam neophodni da bi mogli da ih pozovemo u realizaciji funkcije `simulacija::izvrsavanje_dogadjaja`. Ova funkcija na osnovu prosleđenog rednog broja događaja bira odgovarajući bezuslovni događaj i izvršava ga.

```
// Redni broj dogadjaja DolazakKlijenta
const int DOLAZAK_KLIJENTA = 1;
// Redni broj dogadjaja OdlazakKlijenta
const int ODLAZAK_KLIJENTA = 2;

// Potpis funkcije bezuslovnog dogadjaja DolazakKlijenta
void dolazak_klijenta(entitet *e);
// Potpis funkcije bezuslovnog dogadjaja OdlazakKlijenta
void odlazak_klijenta(entitet *e);

// Izvrsavanje dogadjaja
void simulacija::izvrsavanje_dogadjaja(int dog, entitet* e) {
  switch(dog) {
    case DOLAZAK_KLIJENTA:
      dolazak_klijenta(e);
      break;
    case ODLAZAK_KLIJENTA:
      odlazak_klijenta(e);
      break;
  }
}
```

Formiramo polja `vreme_dolaska` i `vreme_osluge` na osnovu Tabele 1. Takođe, kreiramo objekat reda čekanja, četiri šatera i simulacije koji će kontrolisati izvršenje simulacije.

```
// Vremenski trenuci dolaska entiteta (Tabela 1)
double vreme_dolaska[] = { 0.0, 1.0, 1.0, 1.0, 2.0, 4.0,
                           4.0, 4.0, 4.0, 5.0, 5.0, 6.0, 8.0 };
// Vreme opsluge entiteta (Tabela 1)
double vreme_opsluge[] = { 4.0, 3.0, 4.0, 3.0, 1.0, 4.0,
                           2.0, 1.0, 2.0, 1.0, 1.0, 1.0, 3.0 };

// Objekat reda cekanja
fifo_red red;
// Salteri
resurs salteri(4);
// Objekat simulacije
simulacija simu;
```

Potom, korišćenjem kreiranih objekata realizujemo funkciju bezuslovnog događaja DolazakKlijenta po proceduri datoj na početku prezentacije.

```
void dolazak_klijenta(entitet *e) {
  entitet *fe, *ne;
  // Postavi klijenta u red cekanja
  red.ured(e);
  // Ukoliko je salter raspoloziv
  if(salteri.raspoloziv()) {
    // Izvadi prvog klijenta iz reda
    fe = red.izred();
    // Zauzmi jedno mesto u salteru
    salteri.zauzmi();
    // Rasporedi klijenta za kraj opsluge
    simu.rasporedi(fe,ODLAZAK_KLIJENTA,
        simu.vrati_vreme_simulacije()+
        vreme_opsluge[fe->vrati_id()-1]);
  }
  // Stvaramo narednog klijenta
  ne = simu.napravi_entitet();
  // Stvaramo najvise 13 entiteta zbog polja
  // vreme_dolaska koje ima vremenske trenutke
  // za najviše 13 entiteta
  if(ne->vrati_id()<=13)
    // Postavljamo vreme narednog dolaska
    simu.rasporedi(ne,DOLAZAK_KLIJENTA,
                vreme_dolaska[ne->vrati_id()-1]);
  else {
    // Unistavamo one novostvorene entitete
    // ciji je redni broj veci od 13
    simu.unisti_entitet(ne,0);
  }
}
```

```
procedure DolazakKlijenta
begin
        Postavi kljenta u red čekanja.
        if šalter je raspoloživ then
        begin
          Izvadi prvog klijenta iz reda čekanja.
          Zauzmi šalter.
          Rasporedi klijenta na događaj OdlazakKlijenta.
        end
        Napravi novog klijenta.
        Rasporedi klijenta na događaj DolazakKlijenta.
end
```

Dalje, realizujemo funkciju bezuslovnog događaja DolazakKlijenta po proceduri datoj na početku prezentacije

```
void odlazak_klijenta(entitet *e) {
  entitet *fe;
  // Vrati salter;
  salteri.oslobodi();
  // Klijent odlazi iz poste
  simu.unisti_entitet(e,1);
  // Ukoliko red nije prazan
  if(red.velicina()>0) {
    // Izvadi prvog klijenta iz reda
    fe = red.izred();
    // Zauzmi jedno mesto u salteru
    salteri.zauzmi();
    // Rasporedi klijenta za kraj opsluge
    simu.rasporedi(fe,ODLAZAK_KLIJENTA,
          simu.vrati_vreme_simulacije()+
          vreme_opsluge[fe->vrati_id()-1]);
  }
}
```

```
procedure OdlazakKlijenta
begin
        Oslobodi šalter.
        Uništi klijenta.
        if red čekanja nije prazan then
        begin
            Izvadi prvog klijenta iz reda čekanja.
            Zauzmi šalter.
            Rasporedi klijenta na događaj OdlazakKlijenta.
        end
end
```

main funkcija

U `main()` funkciji neophodno je napraviti i rasporediti prvi entitet za dogadjaj dolaska klijenta.

```
int main() {
  // Stvaramo prvog klijenta
  entitet *e = simu.napravi_entitet();
  // Postavljamo vreme dolaska prvog klijenta
  simu.rasporedi(e,DOLAZAK_KLIJENTA,vreme_dolaska[e->vrati_id()-1]);
  // Izvrsavamo simulaciju
  simu.izvrsi(13);
}
```

Funkcijom `simu.napravi_entitet()` pravimo entitet i pokazivač na novokreirani entitet dodeljujemo promenljivoj `e`. Nakon toga za taj isti entitet raspoređujemo vreme dolaska korišćenjem funkcije `simu.rasporedi()`. Funkciji rasporedi se prosledjuje pokazivač na entitet koji se raspoređuje, događaj na koji se raspoređuje i vreme kada će se taj događaj odigrati. Kada je događaj DolazakKlijenta u pitanju vreme dolaska se uzima iz polja `vreme_dolaska`, dok kada raspoređujemo klijenta na događaj OdlazakKlijenta na trenutno vreme simulacije (`simu.vrati_vreme_simulacije()`) dodajemo

vreme iz `vreme_opsluge`. Vreme vadimo sa pozicije `e->vrati_id()-1` obzirom da je redni broj entiteta (id) indeksiran od 1, elementi polja su indeksirani od 0. Funkcijom `simu.izvrsi(13)` izvršavamo simulaciju i postavljamo terminacioni brojač na 13. Na taj omogućavamo da simulacija traje dok se ne uništi 13 entiteta u događaju odlazak entiteta, primenom funkcije `simu.unisti_entitet(e,1)`.

Štampanje izveštaja

Nakon svakog odigravanja bezuslovnog događaja ispisujemo koji je entitet došao ili otišao iz sistema, klijente u redu čekanja, kao i klijente na opsluzi.

```
// Stampa u izvestaj
cout << simu.vrati_vreme_simulacije() << ": Klijent " <<
                  e->vrati_id() << " dolazi u postu." << endl;

// Stampa u izvestaj
cout << simu.vrati_vreme_simulacije() << ": Klijent " <<
                  e->vrati_id() << " odlazi iz poste." << endl;

// Stampa u izvestaj
cout << simu.vrati_vreme_simulacije() << ": Klijenti u redu: ";
red.stampa(); // Stampamo klijente u redu.
cout << "."  << endl;
cout << simu.vrati_vreme_simulacije() << ": Klijenti u sistemu: ";
simu.stampaj_listu(); // Stampamo klijente iz LBD.
cout << "."  << endl;
```

Izveštaj

```
0: Klijent 1 dolazi u postu.
0: Klijenti u redu: .
0: Klijenti u sistemu: 1,.
1: Klijent 2 dolazi u postu.
1: Klijenti u redu: .
1: Klijenti u sistemu: 1,2,.
1: Klijent 3 dolazi u postu.
1: Klijenti u redu: .
1: Klijenti u sistemu: 2,1,3,.
1: Klijent 4 dolazi u postu.
1: Klijenti u redu: .
1: Klijenti u sistemu: 1,2,4,3,.
2: Klijent 5 dolazi u postu.
2: Klijenti u redu: 5,.
2: Klijenti u sistemu: 2,4,1,3,.
4: Klijent 2 odlazi iz poste.
4: Klijenti u redu: .
4: Klijenti u sistemu: 4,1,3,5,.
4: Klijent 4 odlazi iz poste.
4: Klijenti u redu: .
4: Klijenti u sistemu: 1,3,5,.
4: Klijent 1 odlazi iz poste.
4: Klijenti u redu: .
4: Klijenti u sistemu: 3,5,.
4: Klijent 6 dolazi u postu.
4: Klijenti u redu: .
4: Klijenti u sistemu: 3,5,6,.
4: Klijent 7 dolazi u postu.
4: Klijenti u redu: .
4: Klijenti u sistemu: 5,3,7,6,.
4: Klijent 8 dolazi u postu.
4: Klijenti u redu: 8,.
4: Klijenti u sistemu: 3,5,7,6,.
```

```
4: Klijent 9 dolazi u postu.
4: Klijenti u redu: 8,9,.
4: Klijenti u sistemu: 5,3,7,6,.
5: Klijent 5 odlazi iz poste.
5: Klijenti u redu: 9,.
5: Klijenti u sistemu: 3,7,8,6,.
5: Klijent 3 odlazi iz poste.
5: Klijenti u redu: .
5: Klijenti u sistemu: 7,8,9,6,.
5: Klijent 10 dolazi u postu.
5: Klijenti u redu: 10,.
5: Klijenti u sistemu: 7,8,9,6,.
5: Klijent 11 dolazi u postu.
5: Klijenti u redu: 10,11,.
5: Klijenti u sistemu: 8,7,9,6,.
6: Klijent 8 odlazi iz poste.
6: Klijenti u redu: 11,.
6: Klijenti u sistemu: 7,9,10,6,.
6: Klijent 7 odlazi iz poste.
6: Klijenti u redu: .
6: Klijenti u sistemu: 9,10,11,6,.
6: Klijent 12 dolazi u postu.
6: Klijenti u redu: 12,.
6: Klijenti u sistemu: 9,10,11,6,.
7: Klijent 9 odlazi iz poste.
7: Klijenti u redu: .
7: Klijenti u sistemu: 10,11,6,12,.
7: Klijent 10 odlazi iz poste.
7: Klijenti u redu: .
7: Klijenti u sistemu: 11,6,12,.
7: Klijent 11 odlazi iz poste.
7: Klijenti u redu: .
7: Klijenti u sistemu: 6,12,.
8: Klijent 6 odlazi iz poste.
8: Klijenti u redu: .
8: Klijenti u sistemu: 12,.
8: Klijent 13 dolazi u postu.
```

```
8: Klijenti u redu: .
8: Klijenti u sistemu: 12,13,.
8: Klijent 12 odlazi iz poste.
8: Klijenti u redu: .
8: Klijenti u sistemu: 13,.
11: Klijent 13 odlazi iz poste.
11: Klijenti u redu: .
11: Klijenti u sistemu: .
```

Kompajliranje i izvršenje programa

```
cl simu1.cpp /EHsc

simu1
```

ili

```
simu1 > izvestaj.txt
```

Kod datotetke zaglavlja simu1.h

```
// Zaglavlje simu1.h
#pragma once
#include <iostream>
#include <cstdlib>

// Ukljucujemo prostor imena std;
using namespace std;

// Prostor imena simu1
namespace simu1 {
  // Konstante
  const int max_broj_uredu  = 10;
  const int max_broj_ulisti = 50;
  // Klasa entiteta
  class entitet {
    // Omogucavamo klasi simulacija da pristupi privatnim clanovima klase entitet
    friend class simulacija;
    // Omogucavamo klasi fifo_red da pristupi privatnim clanovima klase entitet
    friend class fifo_red;
    // Redni broj entiteta
    int id;
    // Buduci dogadjaj
    int naredni_dogadjaj;
    // Vreme nastupanja buduceg dogadjaja
    double vreme_nastupanja_dogadjaja;
  public:
    // Konstruktor
    entitet(int eid):id(eid) { }
    // Vraca id entiteta
    int vrati_id() { return id; }
```

```
};
// Klasa FIFO red cekanja
class fifo_red {
  // Broj entiteta u redu cekanja
  int broj;
  // Polje pokazivaca na entitete
  entitet* red[max_broj_uredu];
public:
  // Konstruktor
  fifo_red(): broj(0) {}
  // Smestamo entitet u red cekanja
  void ured(entitet* e) {
    if( broj < max_broj_uredu) {
      // Smestamo entitet na kraj reda cekanja
      red[broj]=e;
      // Uvecavamo broj entiteta za 1
      broj++;
    }
    else {
      cerr << "Previse korisnika u redu cekanja" << endl;
      exit(1);
    }
  }
  // Vadimo entitet iz reda cekanja
  entitet* izred() {
    entitet *e;
    if(broj>0) {
      // Vadimo prvi entitet iz reda cekanja
      e = red[0];
      for(int i=1;i<broj;i++)
        red[i-1] = red[i];
      // Umanjujemo broj entiteta iz reda cekanja
```

```
      broj--;
    }
    else {
      cerr << "Nema entiteta u redu cekanja" << endl;
      exit(1);
    }
    // Vraca entitet
    return e;
  }
  // Vracamo velicinu reda cekanja
  int velicina() { return broj; };
  void stampa() {
    for(int i=0; i<broj; i++)
      cout << red[i]->id << ",";
  }
};
// Klasa resursa
class resurs {
  // Tekuci broj zauzetih mesta u resursa
  int broj_zauzetih_mesta;
  // Maksimalni broj  mesta u resursu
  int ukupni_broj_mesta;
public:
  // Konstruktor
  resurs(int b):ukupni_broj_mesta(b),broj_zauzetih_mesta(0) {}
  // Zauzima mesto u resursu
  void zauzmi() {
    if(broj_zauzetih_mesta<ukupni_broj_mesta)
      // Uvecavamo broj zauzetih mesta
      broj_zauzetih_mesta++;
    else {
      cerr << "Sva mesta u resursu su zauzeta" << endl;
```

```
      exit(1);
    }
  }
  // Oslobadja mesto u resursu
  void oslobodi() {
    if(broj_zauzetih_mesta>0)
      // Umanjujemo broj zauzetih mesta
      broj_zauzetih_mesta--;
    else {
      cerr << "Nema entiteta u resursu" << endl;
      exit(1);
    }
  }
  // Raspolozivost resursa (ima makar jedno slobodno mesto u resursu)
  bool raspoloziv() { return (broj_zauzetih_mesta<ukupni_broj_mesta); }
};
// Klasa simulacija
class simulacija {
  // Brojac entiteta
  int eid;
  // Terminacioni brojac
  int tb;
  // Broj entiteta u LBD listi
  int broj;
  // Vreme simulacije
  double vreme_simulacije;
  // Polje entiteta u LBD listi
  entitet* lista_nastupanja_entiteta[max_broj_ulisti];
  // Sortiranje LBD - Sortiranje izborom
  void sortiraj_listu_nastupanja() {
    int min;
    entitet *tmp;
```

```
    for(int i=0; i<broj-1; i++) {
      min = i;
      for(int j=i+1; j<broj; j++) {
        if( lista_nastupanja_entiteta[j]->vreme_nastupanja_dogadjaja <
            lista_nastupanja_entiteta[min]->vreme_nastupanja_dogadjaja)
          min = j;
      }
      if(min!=i) {
        tmp = lista_nastupanja_entiteta[i];
        lista_nastupanja_entiteta[i] = lista_nastupanja_entiteta[min];
        lista_nastupanja_entiteta[min] = tmp;
      }
    }
  }
public:
  // Konstruktor
  simulacija():eid(1),tb(0),broj(0),vreme_simulacije(0.0) {}
  // Destruktor
  ~simulacija() {
    // Uklanjamo preostale entitete iz LBD na kraju simulacije.
    for(int i=0; i<broj; i++)
      delete lista_nastupanja_entiteta[i];
  }
  // Pravljenje entiteta
  entitet* napravi_entitet() {
    // Pravimo novi entitet
    return new entitet(eid++);
  }
  // Unistavanje entiteta
  void unisti_entitet(entitet *e, int b) {
    // Unistavamo entitet
    delete e;
```

```
  // Umanjujemo terminacioni brojac
  tb -= b;
}
// Rasporedjivanje dogadjaja
void rasporedi(entitet* e,int dog,double vreme) {
  if( broj < max_broj_ulisti) {
    // Postavljamo dogadjaj
    e->naredni_dogadjaj = dog;
    // Postavljamo vreme nastupanja dogadjaja
    e->vreme_nastupanja_dogadjaja = vreme;
    // Smestamo entitet u listu
    lista_nastupanja_entiteta[broj] = e;
    broj++;
    // Sortiramo LBD
    sortiraj_listu_nastupanja();
  }
  else {
    cerr << "Previse entiteta u listi dogadjaja" << endl;
    exit(1);
  }
}
// Izvrsavanje simulacije
void izvrsi(int b) {
  entitet * tekuci;
  // Postavljamo terminacioni brojac
  tb = b;
  // Izvrsavamo simulaciju. Simulacija se zavrsava ukoliko
  // nema entiteta u LBD ili ukoliko je terminacioni brojac 0.
  do {
    // Vadimo prvi entitet iz liste. Taj entitet postaje tekuci entitet.
    tekuci = lista_nastupanja_entiteta[0];
    for(int i=1;i<broj;i++)
```

```
          lista_nastupanja_entiteta[i-1] = lista_nastupanja_entiteta[i];
        broj--;
        // Faza 1: Azurirarmo vreme simulacije
        vreme_simulacije = tekuci->vreme_nastupanja_dogadjaja;
        // Faza 2: Izvrsavamo dogadjaj
        izvrsavanje_dogadjaja(tekuci->naredni_dogadjaj,tekuci);
      } while(broj && tb>0);
    }
    // Izvrsava dogadjaj
    void izvrsavanje_dogadjaja(int dog, entitet* e);
    // Vraca vreme simulacije
    double vrati_vreme_simulacije() { return vreme_simulacije; }
    // Stampamo redne brojeve onih entiteta koji su rasporedjeni
    void stampaj_listu() {
      for(int i=0; i<broj; i++)
        if(lista_nastupanja_entiteta[i]->naredni_dogadjaj==2)
          cout << lista_nastupanja_entiteta[i]->id << ",";
    }
  };
}
```

Kod datoteke zaglavlja simu1.cpp

```
// Datoteka simu1.cpp
#include "simu1.h"
#include <iostream>

// Ukljucujemo prostore imena simu1 i std
using namespace simu1;
using namespace std;

// Redni broj dogadjaja DolazakKlijenta
const int DOLAZAK_KLIJENTA = 1;
// Redni broj dogadjaja OdlazakKlijenta
const int ODLAZAK_KLIJENTA = 2;

// Potpis funkcije bezuslovnog dogadjaja DolazakKlijenta
void dolazak_klijenta(entitet *e);
// Potpis funkcije bezuslovnog dogadjaja OdlazakKlijenta
void odlazak_klijenta(entitet *e);

// Izvrsavanje dogadjaja
void simulacija::izvrsavanje_dogadjaja(int dog, entitet* e) {
  switch(dog) {
    case DOLAZAK_KLIJENTA:
      dolazak_klijenta(e);
      break;
    case ODLAZAK_KLIJENTA:
      odlazak_klijenta(e);
      break;
  }
}

// Vremenski trenuci dolaska entiteta (Tabela 1)
double vreme_dolaska[] = { 0.0, 1.0, 1.0, 1.0, 2.0, 4.0,
                           4.0, 4.0, 4.0, 5.0, 5.0, 6.0, 8.0 };
```

```
// Vreme opsluge entiteta (Tabela 1)
double vreme_opsluge[] = { 4.0, 3.0, 4.0, 3.0, 1.0, 4.0,
                           2.0, 1.0, 2.0, 1.0, 1.0, 1.0, 3.0 };

// Objekat reda cekanja
fifo_red red;
// Salteri
resurs salteri(4);
// Objekat simulacije
simulacija simu;

void dolazak_klijenta(entitet *e) {
  entitet *fe, *ne;
  // Stampa u izvestaj
  cout << simu.vrati_vreme_simulacije() << ": Klijent " << e->vrati_id() << " dolazi u postu."
<< endl;
  // Postavi klijenta u red cekanja
  red.ured(e);
  // Ukoliko je salter raspoloziv
  if(salteri.raspoloziv()) {
    // Izvadi prvog klijenta iz reda
    fe = red.izred();
    // Zauzmi jedno mesto u salteru
    salteri.zauzmi();
    // Rasporedi klijenta za kraj opsluge
    simu.rasporedi(fe,ODLAZAK_KLIJENTA,simu.vrati_vreme_simulacije()+vreme_opsluge[fe-
>vrati_id()-1]);
  }
  // Stampa u izvestaj
  cout << simu.vrati_vreme_simulacije() << ": Klijenti u redu: "; red.stampa(); cout << "."  <<
endl;
  cout << simu.vrati_vreme_simulacije() << ": Klijenti u sistemu: "; simu.stampaj_listu(); cout
<< "."  << endl;
  // Stvaramo narednog klijenta
  ne = simu.napravi_entitet();
  if(ne->vrati_id()<=13)
    // Postavljamo vreme narednog dolaska
```

```
    simu.rasporedi(ne,DOLAZAK_KLIJENTA,vreme_dolaska[ne->vrati_id()-1]);
  else
    simu.unisti_entitet(ne,0);
}

void odlazak_klijenta(entitet *e) {
  entitet *fe;
  // Vrati salter;
  salteri.oslobodi();
  // Stampa u izvestaj
  cout << simu.vrati_vreme_simulacije() << ": Klijent " << e->vrati_id() << " odlazi iz poste."
<< endl;
  // Klijent odlazi iz poste - unistavamo tekuceg klijenta
  simu.unisti_entitet(e,1);
  // Ukoliko red nije prazan
  if(red.velicina()>0) {
    // Izvadi prvog klijenta iz reda
    fe = red.izred();
    // Zauzmi jedno mesto u salteru
    salteri.zauzmi();
    // Rasporedi klijenta za kraj opsluge
    simu.rasporedi(fe,ODLAZAK_KLIJENTA,simu.vrati_vreme_simulacije()+vreme_opsluge[fe-
>vrati_id()-1]);
  }
  // Stampa u izvestaj
  cout << simu.vrati_vreme_simulacije() << ": Klijenti u redu: "; red.stampa(); cout << "."  <<
endl;
  cout << simu.vrati_vreme_simulacije() << ": Klijenti u sistemu: "; simu.stampaj_listu(); cout
<< "."  << endl;
}

int main() {
  // Stvaramo prvog klijenta
  entitet *e = simu.napravi_entitet();
  // Postavljamo vreme dolaska prvog klijenta
  simu.rasporedi(e,DOLAZAK_KLIJENTA,vreme_dolaska[e->vrati_id()-1]);
  // Izvrsavamo simulaciju
```

```
  simu.izvrsi(13);
}
```