Osnovne akademske studije

PREDMET: Objektno-orijentisana simulacija

TEMA: Strategija raspoređivanja događaja 1

Predmetni nastavnik: Doc. dr 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 odigrava jedan ili više događaja naziva se *vremenskim trenutkom*.

U slučaju koristi *mehanizam pomaka vremena na naredni događaj* *vreme simulacije (simulacioni časovnik)* se menja tako što se pomera iz vremena izvršenja prethodnog događaja u vreme izvršenja prvog narednog događaja .

Objekti čije se aktivnosti modeliraju nazivaju se *agentima*. Takvi objekti moraju biti identifikovani, a njihovo ponašanje određeno. Agenti mogu da komuniciraju međusobno. To , mogu biti: ljudi, vozila, oprema, proizvodi, preduzeća, itd.

***Događaj* je diskretna promena stanja agenta u određenom vremenskom trenutku. Takvo vreme se naziva vremenom narednog događaja. Između dva događaja stanje se ne menja. Razlikujemo dve vrste događaja:**

1. ***bezuslovni događaj* – događaj čije se vreme odigravanja može unapred predvideti. Primeri: dolasci korisnika u poštu, prosleđivanje paketa komunikacionim kanalom, pristizanje narudžbine u**

preduzeće, obrada zahteva od strane centralnog procesora.

2. ***uslovni događaj*** **– događaj čije izvršenje zavisi od ispunjenja određenog uslova. Događaji koji ne mogu da se izvrše ukoliko se određeni uslov ne ispuni. Primeri: pristupanje šalterima u pošti, zauzimanje komunikacionog kanala od strane paketa, zauzimanje centralnog procesora od strane programa.**

Resursi **su objekti simulacionog modela čija je osnovna uloga da ograniče aktivnosti agenata. Resursi imaju na raspolaganju jedno ili više mesta u koje primaju agente. Agenti zauzimaju i oslobađaju mesta u resursu. Primeri:**

radnici na šalterima, emiter koji šalje i obrađuje poruke, komunikacioni kanali koji posleđuju pakete, mašina koja sklapa delove, računar koji izvršava programe.

Redovi čekanja **su objekti simulacionog modela koji zadržavaju agente. Redovi čekanja uređuju agente u redu prema disciplini opsluživanja (FIFO, LIFO, prioritet). Red čekanja pokušava da oslobodi prvog agenta u redu što pre, ali njegov uspeh zavisi od ispunjenja uslova izlaska iz reda. Primeri: ljudi koji čekaju u redu, poruke u baferu koje čekaju da budu poslate, delovi koji čekaju da budu montirani, računarski programi koji treba da se izvrše.**

Strategija raspoređivanje događaja (dvofazna strategija)

Strategija raspoređivanja događaja je koncipirana tako da se događaji planiraju unapred i čuvaju u *listi budućih događaja (LBD)*. Događaji u listi su sortirani prema vremenu nastupanja.

Strategiju je moguće opisati na sledeći način:

1. **Sa liste budućih događaja se uzima prvi događaj i uklanja iz liste. Taj događaj postaje *tekući događaj*.**
2. ***Faza A*: Vreme simulacije se ažurira na vreme nastupanja tekućeg događaja.**
   ***Faza B*: Izvršava se tekući događaj.**

Raspoređivanje (smeštanje) događaja u LBD se vrši po sledećoj proceduri:

1. **Događaju se zadaju *agenti* kojima se menja stanje i *vreme nastupanja događaja*.**
2. **Događaj se postavlja u LBD.**
3. **LBD 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 korisnika dati su u tabeli 1.**

U tabeli 2 dati su dolasci korisnika, sadržaj zajedničkog reda čekanja, korisnici u sistemu, kao i vremenski trenuci odlaska korisnika. Tabela je dobijena ručnim rešavanjem simulacije.

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

Korisnici	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 korsnika u poštu i vreme opsluge na jednom od četiri šaltera

Trenutak (min)	Dolazak korisnika	Red čekanja	Korisnik u pošti	Odlasci korisnika
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 *DolazakKorisnika*
begin
Postavi korisnika u red čekanja.
if *šalter je raspoloživ* **then**
begin
Izvadi prvog korisnika iz reda čekanja.
Zauzmi šalter.
Rasporedi korisnika na događaj OdlazakKorisnika.
end
Napravi novog klorisnika.
Rasporedi korisnika na događaj DolazakKorisnika.
end

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

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

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

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

Faza	Događaj	LBD*	Korisnici u redu čekanja	Korisn. 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 događaja. Vreme časovnika je 6.
Izvršavanje događaja	DolazakKorisnika	[2,7,6] [2,8,6] [2,9,7] [2,6,8] [1,13,8]	10,11,12	7,8,9,6	Izvršavamo događaj DolazakKorisnika za korisnika 12. Pošto su šalteri zauzeti smeštamo korisnika 12 u red čekanja. Stvaramo korisnika 13 i raspoređujemo ga na DolazakKorisnika u trenutku 8.
	OdlazakKorisnika	[2,8,6] [2,9,7] [2,10,7] [2,6,8] [1,13,8]	11,12	10,8,9,6	Izvršavamo događaj OdlazakKorisnika za korisnika 7. Oslobađamo šalter. Uništavamo korisnika 7. Iz reda čekanja vadimo korisnika 10, zauzimamo mesto u šalteru i raspoređujemo korisnika 10 na OdlazakKorisnika u trenutku 7.
	OdlazakKorisnika	[2,9,7] [2,10,7] [2,11,7] [2,6,8] [1,13,8]	12	10,11,9,6	Izvršavamo događaj OdlazakKorisnika ta za korisnika 8. Oslobađamo šalter. Uništavamo korisnika 8. Iz reda čekanja vadimo korisnika 11, zauzimamo mesto u šalteru i raspoređujemo korisnika 11 na OdlazakKorisnika u trenutku 7.

*** [događaj (1 – DolazakKorisnika; 2 – OdlazakKorisnika), broj korisnika, vreme nastupanja događaja]**

Klasa agent

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

Klasa `agent` sadrži dve javne funkcije `posalji_poruku` i `primi_poruku` i destruktor. Destruktor i funkcija `primi_poruku` su virtuelne funkcije. `posalji_poruku` i `primi_poruku` se koriste za komunikaciju između agenata. Agenti ne mogaju da komuniciraju, međutim ukoliko to žele agenti moraju da implementiraju funkciju `primi_poruku`.

Klasa dogadjaj

```
// Apstrakna klasa dogadjaja
class dogadjaj {
protected:
  // Vreme nastupanja
  vreme    _vreme;
  // Agent
  agent** _agenti;
  // Broj agenata
  size_t  _broj;
public:
  //
  dogadjaj(const size_t n):_vreme(0.0),_broj(n) {
    _agenti = new agent*[_broj];
  }
  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 << "Pogresni indeks" << endl;
      exit(1);
    }
  }
```

```
    // Vrati agenta
    agent* vrati_agenta(const size_t n) const {
      if(n<_broj)
        return _agenti[n];
      else {
        cerr << "Pogresni indeks" << endl;
        exit(1);
      }
      return nullptr;
    }
    // Postavi vreme nastupanja
    void postavi_vreme(const vreme v) {
      _vreme = v;
    }
    // Vrati vreme nastupanja
    vreme vrati_vreme() const { return _vreme; }
  public:
    // Cista virtuelna funkcija dogadjaja
    virtual void akcija() = 0;
  };
```

Apstraktna klasa `dogadjaj` sadrži tri zaštićena člana: `_vreme`, koje odgovara vremenu nastupanju događaja, `_agenti`, koje predstavlja polje agenata pridruženih događaju i `_broj`, najveci broj agenata koje je moguće pridružiti događaju. Član `_vreme` je tipa `vreme` koji je predefinisano i odgovara tipu `double`. Član `_agenti` je polje pokazivača na agente, dok je član `_broj` tipa

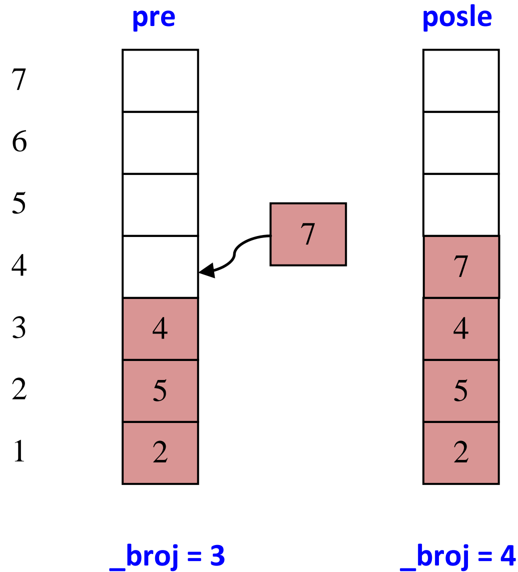
`size_t`. Klasa sadrži 5 funkcija koje sve javne i od kojih je funkcija `akcija` čista virtuelna funkcija. Klasa, takođe, sadrži jedan konstruktor i virtuelni destruktor. Konstruktorom se postavlja broj agenata koje je moguće pridružiti klasi. Ostale funkcije služe za postavljanje i vraćanje vrednosti vremena i agenata pridruženih događaju.

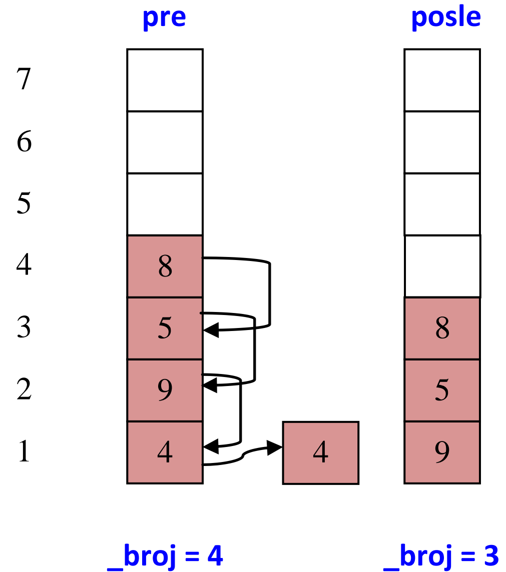
Klasa red

```
  // Klasa FIFO red cekanja
  class red {
#ifdef STAMPA
    friend void stampaj_red();
#endif
  protected:
    // Broj agenata u redu cekanja
    size_t _broj;
    // Polje pokazivaca na entitete
    agent* _red[max_broj_uredu];
  public:
    // Konstruktor
    red(): _broj(0) {}
    // Smestamo agenta u red cekanja
    void smesti(const agent* a) {
      if( _broj < max_broj_uredu) {
        // Smestamo entitet na kraj reda cekanja
        _red[_broj]=const_cast<agent*>(a);
        // Uvecavamo broj agenta za 1
        _broj++;
      }
      else {
        cerr << "Previse korisnika u redu cekanja" << endl;
        exit(1);
      }
    }
```

```
    // Vadimo agenta iz reda cekanja
    void vadi() {
      if(_broj>0) {
        for(size_t 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);
      }
    }
    // Prednji i zadnji agent u redu cekanja
    agent* prednji() const { return _red[0];        }
    agent* zadnji()  const { return _red[_broj-1]; }
    // Vracamo velicinu reda cekanja
    size_t vrati_velicinu() { return _broj; };
  };
```

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





max_broj_uredu = 7

```
// Smestamo entitet na kraj reda cekanja
_red[_broj]=const_cast<agent*>(a);
// Uvecavamo broj agenata za 1
_broj++;
```

```
// Uklanjamo prvog agenta iz reda cekanja
for(int i=1;i<_broj;i++)
   _red[i-1] = _red[i];
// Umanjujemo broj agenata u redu cekanja
_broj--;
```

Klasa resurs

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

```
            cerr << "Nema entiteta u resursu" << endl;
            exit(1);
        }
    }
    // Raspolozivost resursa (ima makar jedno slobodno mesto u resursu)
    bool raspoloziv() { return _broj<_kapacitet; }
};
```

Klasa `resurs` sadrži dva zaštićena člana `_broj` i `_kapacitet` (tipa `size_t`). `_broj` predstavlja broj zauzetih mesta u resursu u nekom vremenskom trenutku. `_kapacitet` 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 umetanjem

Algoritam

Izabira se član polja počevši od drugog člana. Izabrana vrednosti se menja sve do poslednjeg člana. Izabrani član se poredi sa svim članovima koji se nalaze ispred njega i ukoliko je neki manji vrši se zamena.

Sortiranje umetanjem je stabilan algoritam sortiranja.

Primer: Potrebno je sortirati celobrojno polje 5,-1, 3, 2, 9, 7, 1 u rastućem redosledu koristeći sortiranje umetanjem. Crvenom bojom je označen izabrani član polja. Magentom je označen član polja koji se poredi sa izabranim članom. Zelenom su označene članovi polja nakon razmene članova sa izabranim članova. 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 umetanja

```
void sortiranje_umetanjem(int x[],size_t n) {
  int t;
  for(size_t i=1;i<n;++i) {
    size_t j = i;
    while(j>0 && x[j-1]>x[j]) {
      t      = x[j-1];
      x[j-1] = [j];
      x[j--] = t;
    }
  }
}
```

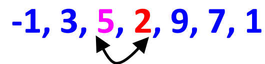
-1, 5, 3, 2, 9, 7, 1

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

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

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

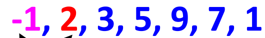
-1, 3, 5, 2, 9, 7, 1



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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Klasa simulacija

```
  // Klasa simulacija
  class simulacija {
#ifdef STAMPA
    friend void stampaj_lbd();
#endif
  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 umetanjem
    void sortiraj_lbd() {
      dogadjaj *tmp;
      for(size_t i=1;i<_broj;++i) {
        size_t j = i;
        while(j>0 && _lbd[j]->vrati_vreme()<_lbd[j-1]->vrati_vreme()) {
          tmp       = _lbd[j-1];
          _lbd[j-1] = _lbd[j];
          _lbd[j--] =  tmp;
        }
      }
    }
```

```
public:
  // Konstruktor
  simulacija():_stop(false),_broj(0),_vreme(0.0) {}
  // Destruktor
  ~simulacija() {
    // Uklanjamo 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 << "Previse entiteta u listi dogadjaja" << endl;
      exit(1);
    }
  }
  // Izvrsavanje simulacije
  void izvrsi() {
    // Flag zaustavljanja je netacan
    _stop = false;
    // Izvrsavamo simulaciju. Simulacija se zavrsava 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: Azurirarmo vreme simulacije
        _vreme = _tekuci->vrati_vreme();
        // Faza B: Izvrsavamo dogadjaj
        _tekuci->akcija();
        // Brisemo tekuci dogadjaj iz memorije
        delete _tekuci;
      } while(_broj && !_stop);
    }
    // Zaustavljamo izvrsenje simulacije
    void zaustavi() { _stop = true; }
    // Vraca vreme simulacije
    vreme vrati_vreme() { return _vreme; }
  };
```

Klasa simulacija ima 4 privatna člana i 4 javnih i jednu privatnu metodu. `_stop` je indikator koji se koristi da označi kraj simulacije. Na taj način se kontroliše vreme trajanja simulacije. Na početku simulacije je `false`. `_broj` je tekući broj elemenata u LBD listi. `_vreme` sadrži tekuću vrednost simulacionog časovnika. `_lbd` je polje događaja i predstavlja LBD listu. Veličina `_lbd` liste je

ograničena i iznosi `max_broj_ulisti` (za naš primer vrednost `max_broj_ulisti` je 50)

Funkcija `rasporedi` vrši raspoređivanje entiteta sa budućim događajem u LBD listu prema vremenu nastupanja događaja. Događaj se ubacuje na kraj `_lbd` liste na isti način kako se to radi kod reda čekanja,

```
// Smestamo entitet u listu
_lbd[_broj] = const_cast<dogadjaj*>(d);
++_broj;
```

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

```
    // Sortiranje LBD - sortiranje umetanjem
void sortiraj_lbd();
```

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

Funkcija `izvrši` izvršava simulaciju koristeći algoritam raspoređivanja događaja koji je dat na početku prezentacije. Unutar petlje se prvo vadi prvi

element LBD liste koji postaje tekući događaj. Element se vadi iz `_lbd` liste na isti način kako se to radi kod klase `red`.

```
// 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;
```

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

```
// Faza A: Azurirarmo vreme simulacije
_vreme = _tekuci->vrati_vreme();
```

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

```
// Faza B: Izvrsavamo dogadjaj
_tekuci->akcija();
```

Ovom prilikom se poziva funkcija `akcija` iz tekućeg događaja.

Funkcija `vrati_vreme` 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`) se prikazuje tekstualna poruka o grešci i program se prekida (`exit(1)`).

Sve do sada navedene klase realizovane su u dva zaglavlja: `esa.h` (`agent`, `dogadjaj`,`simulacija`) i `esaproc.h` (`red`, `resurs`), unutar prostora imena (`namespace`) `esa`.

```
// Zaglavlje esa.h
#ifndef _ESA_H_
#define _ESA_H_

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

// Ukljucujemo prostor imena std;
using namespace std;
```

```
// Prostor imena esa
namespace esa {
  // Realizacije klasa agent, dogadjaj, simulacija
}

#endif // _ESA_H_
```

`#ifndef`, `#define` i `#endif` preprocesorke direktive se koriste da onemoguće višestuko uključivanje zaglavlja. Definisani simbol (`_ESA_H_`) treba da bude takav da imenom određuje zaglavlje, ali da postoje male šanse da je u nekom drugom zaglavlju (neke eksterne biblioteke) takav isti simbol već definisan.

Program dalje realizujemo u izvornoj datoteci zad1.cpp. Na početku zad1.cpp učitavaju se odgovarajuća zaglavlja i dozvoljava se korišćenje odgovarajućih prostora imena.

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

// Ukljucujemo prostore imena esa i std
using namespace esa;
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_klijenata(e);
      break;
    case ODLAZAK_KLIJENTA:
      odlazak_klijenata(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;
```

Klasa `korisnik` nasleđuje klasu `agent`. Objekti ove klase služe da se kreću kroz simulacioni model. Klasa ima dva privatna člana `_idc` i `_id`. Prvi član je statički i predstavlja brojač korisnika. Uvećava se pri svakom stvaranja objekta klase `korisnik` i njegova vrednost se dodeljuje članu `_id`. Član `_id` predstavlja identifikacioni broj korisnika (indeksiran od 1). Identifikacionom broju se pristupa javnom metodom `vrati_id`.

```
// 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;
```

Dalje, deklarišemo klase događaja `dolazak_korisnika` i `odlazak_korisnika`. Klase nasleđuju apstraktnu klasu `događaj`.

Obe ove klase mogu da prime po jednog agenta. U ovom slučaju agent je objekat klase `korisnik`.

```
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(vn);
    return dk;
  }
};
```

```
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;
  }
};
```

Ove dve klase imaju i po jednu pomoćnu, statičku funkciju `napravi` koja se koristi za stvaranje objekata događaja. Argumenti ovih funkcija su pokazivač na klasu `korisnik` (tekući korisnik) i vreme nastupanja događaja (kod klase `dolazak_korisnika`) ili vreme do nastupanja događaja (kod klase `odlazak_korisnika`).

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

```
// Akcija
void dolazak_korisnika::akcija() {
  korisnik *kor,*prvi_kor,*novi_kor;
  // Korisnik koji dolazi u postu
  kor = dynamic_cast<korisnik*>(vrati_agenta(0));
  // 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();
    // Zauzmi jedno mesto u salteru
    salteri.zauzmi();
    // Rasporedi klijenta za kraj opsluge
    simu.rasporedi(odlazak_korisnika::napravi(prvi_kor,
                   vreme_opsluge[prvi_kor->vrati_id()-1]));
  }
  // Stvaramo narednog korisnika
  novi_kor = new korisnik;
  if(novi_kor->vrati_id()<=13)
    // Postavljamo vreme narednog dolaska
    simu.rasporedi(dolazak_korisnika::napravi(novi_kor,
                   vreme_dolaska[novi_kor->vrati_id()-1]));
  else
    delete novi_kor;
}
```

procedure *DolazakKorisnika*
begin
  Postavi korisnika u red čekanja.
  if *šalter je raspoloživ* **then**
  begin
    Izvadi prvog korisnika iz reda čekanja.
    Zauzmi šalter.
    Rasporedi korisnika na događaj OdlazakKorisnika.
  end
  Napravi novog korisnika.
  Rasporedi korisnika na događaj DolazakKorisnik.
end

Dalje, realizujemo virtuelnu funkciju `akcija` bezuslovnog događaja `odlazak_korisnika` po proceduri datoj na početku prezentacije

```
// Akcija dogadjaja
void odlazak_korisnika::akcija() {
  korisnik *kor, *prvi_kor;
  // Korisnik koji odlazi
  kor = dynamic_cast<korisnik*>(vrati_agenta(0));
  // Vrati salter
  salteri.oslobodi();
  // 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();
    // Zauzmi jedno mesto u salteru
    salteri.zauzmi();
    // Rasporedi klijenta za kraj opsluge
    simu.rasporedi(odlazak_korisnika::napravi(prvi_kor,
                  vreme_opsluge[prvi_kor->vrati_id()-1]));
  }
}
```

procedure *OdlazakKorisnika*
begin
    Oslobodi šalter.
    Uništi korisnika.
    if *red čekanja nije prazan* **then**
    begin
      Izvadi prvog korisnika iz reda čekanja.
      Zauzmi šalter.
      Rasporedi korisnika na događaj OdlazakKorisnika.
    end
end

main funkcija

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

```
int main() {
  korisnik *kor = new korisnik;
  // Postavljamo prvi dogadja u listu sto je odgovara dolasku prvog klijenta
  simu.rasporedi(dolazak_korisnika::napravi(kor,
                                 vreme_dolaska[kor->vrati_id()-1]));
  // Izvrsavamo simulaciju
  simu.izvrsi();
}
```

U hip memoriji pravimo korisnika i pokazivač na novokreiranog korisnika dodeljujemo promenljivoj `kor`. Potom raspoređujemo događaj `dolazak_korisnika` u listu budućih događaja korišćenjem funkcije `simu.rasporedi()`. Događaj pravimo korišćenjem pomoćne, statičke funkcije `napravi()`. Ova funkcija ima dva argumenta: pokazivač na korisnika `kor` i vreme dolaska prvog korisnika. Kada je događaj `dolazak_korisnika` u pitanju vreme dolaska se uzima iz polja `vreme_dolaska`, dok kada

raspoređujemo korisnika na događaj `odlazak_korisnika` na trenutno vreme simulacije (`simu.vrati_vreme()`) dodajemo vreme iz vreme_opsluge. Vreme vadimo sa pozicije `kor->vrati_id()-1` obzirom da je redni broj korisnika (`_id`) indeksiran od 1, a elementi polja su indeksirani od 0. Funkcijom `simu.izvrsi()` započinjemo izvršenje simulacije. Simulacija se zaustavlja tako što u simulirani model dolazi samo 13 korisnika. Odlaskom poslednjeg, trinaestog korisnika lista budućih događaja postaje prazna što dovodi do završetka simulacije.

Štampanje izveštaja

Nakon svakog odigravanja bezuslovnog događaja ispisujemo koji je korisnik došao ili otišao iz sistema, korisnike u redu čekanja, kao i korisnike na opsluzi. Za štampu korisnika u redu čekanja i korisnika u listi budućih događaja koji se nalaze na opsluzi koriste se dve funkcije iz prostora imena `aes` (deklarisane u zalavlju aes.h): `stampaj_red()` i `stampaj_lbd()`. Ove dve funkcije su i prijateljske funkcije klasa `red` i `simulacija`, respektivno.

```
#ifdef STAMPA
void aes::stampaj_red() {
  for(size_t i=0; i<salt_red._broj; ++i) {
    cout << dynamic_cast<korisnik*>(salt_red._red[i])->vrati_id();
    if(i<salt_red._broj-1)
      cout << ",";
  }
}

void aes::stampaj_lbd() {
  for(size_t i=0;i<simu._broj;++i) {
    odlazak_korisnika* ok = dynamic_cast<odlazak_korisnika*>(simu._lbd[i]);
    if(ok) {
      cout << dynamic_cast<korisnik*>(simu._lbd[i]->vrati_agenta(0))->vrati_id();
      if(i<simu._broj-1)
        cout << ",";
    }
```

```
  }
}
#endif
```

Narednim delovima koda je data štampa u izveštaj funkcije `akcija` događaja `dolazak_korisinika` i `odlazak_korisnika`.

```
  #ifdef STAMPA
  // Stampa u izvestaj
  cout << simu.vrati_vreme() << ": Korisnik " << kor->vrati_id() << " odlazi iz
poste." << endl;
  #endif

  #ifdef STAMPA
  // Stampa u izvestaj
  cout << simu.vrati_vreme() << ": Korisnici u redu: "; stampaj_red(); cout << endl;
  cout << simu.vrati_vreme() << ": Korisnici u sistemu: "; stampaj_lbd(); cout <<
endl;
  #endif
```

Štampanje izveštaja je uslovno i zavisi od toga da li je prosleđen makro simbol `STAMPA`.

Kompajliranje i izvršenje programa

Microsoft Visual C++

```
cl zad1.cpp -O2 -W4 -DSTAMPA /EHsc
simu1
```

MingW C++

```
g++ zad1.cpp -O3 -DSTAMPA -Wall -static -ozad1.exe
```

Izvršenje

```
zad1 > izvestaj.txt
```

Izveštaj

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

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

```
7: Korisnici u redu:
7: Korisnici u sistemu: 11,6,12
7: Korisnik 11 odlazi iz poste.
7: Korisnici u redu:
7: Korisnici u sistemu: 6,12
8: Korisnik 6 odlazi iz poste.
8: Korisnici u redu:
8: Korisnici u sistemu: 12
8: Korisnik 13 dolazi u postu.
8: Korisnici u redu:
8: Korisnici u sistemu: 12,13
8: Korisnik 12 odlazi iz poste.
8: Korisnici u redu:
8: Korisnici u sistemu: 13
11: Korisnik 13 odlazi iz poste.
11: Korisnici u redu:
11: Korisnici u sistemu:
```

Kod datotetke zaglavlja `aes.h`

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

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

// Ukljucujemo prostor imena std;
using namespace std;

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

  // Predefinisacemo 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* ap,const string& poruka) {};
  };
  // 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);
      }
    }
    // Vrati 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;
```

```
    }
    // Vrati vreme nastupanja
    vreme vrati_vreme() const { return _vreme; }
  public:
    // Cista virtuelna funkcija dogadjaja
    virtual void akcija() = 0;
  };
#ifdef STAMPA
  // Pomocna funkcija - potrebna je zbog zadataka 1
  void stampaj_lbd();
#endif
  // Klasa simulacija
  class simulacija {
#ifdef STAMPA
    friend void stampaj_lbd();
#endif
  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 umetanjem
    void sortiraj_lbd() {
      dogadjaj *tmp;
      for(size_t i=1;i<_broj;++i) {
        size_t j = i;
        while(j>0 && _lbd[j]->vrati_vreme()<_lbd[j-1]->vrati_vreme()) {
          tmp       = _lbd[j-1];
          _lbd[j-1] = _lbd[j];
          _lbd[j--] =  tmp;
        }
      }
    }
```

```
public:
  // Konstruktor
  simulacija():_stop(false),_broj(0),_vreme(0.0) {}
  // Destruktor
  ~simulacija() {
    // Uklanjamo 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 - Previse entiteta u listi dogadjaja" << endl;
      exit(1);
    }
  }
  // Izvrsavanje simulacije
  void izvrsi() {
    // Flag zaustavljanja je netacan
    _stop = false;
    // Izvrsavamo simulaciju. Simulacija se zavrsava 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: Azurirarmo vreme simulacije
      _vreme = _tekuci->vrati_vreme();
```

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

#endif // __AES_H__
```

Kod datotetke zaglavlja `aes_proc.h`

```
// Zaglavlje aes_proc.h
#ifndef __AES_PROC_H__
#define __AES_PROC_H__

#include "aes.h"

namespace aes {
#ifdef STAMPA
  // Pomocna funkcije - potrebna je zbog zadataka 2
  void stampaj_red();
#endif

  // Klasa FIFO red cekanja
  class red : public agent {
#ifdef STAMPA
    friend void stampaj_red();
#endif
  protected:
    // Broj agenata u redu cekanja
    size_t _broj;
    // Polje pokazivaca na entitete
    agent* _red[max_broj_uredu];
  public:
    // Konstruktor
    red(): _broj(0) {}
    // Smestamo agenta u red cekanja
    void smesti(const agent* a) {
      if( _broj < max_broj_uredu) {
        // Smestamo entitet na kraj reda cekanja
        _red[_broj]=const_cast<agent*>(a);
        // Uvecavamo broj agenta za 1
        _broj++;
      }
```

```
    else {
      cerr << "red::smesti - Previse agenata u redu cekanja" << endl;
      exit(1);
    }
  }
  // Vadimo agenta iz reda cekanja
  void vadi() {
    if(_broj>0) {
      for(size_t i=1;i<_broj;++i)
        _red[i-1] = _red[i];
      // Umanjujemo broj agenata iz reda cekanja
      --_broj;
    }
    else {
      cerr << "red::vadi - Nema agenata u redu cekanja" << endl;
      exit(1);
    }
  }
  // Prednji i zadnji agent u redu cekanja
  agent* prednji() const { return _red[0];       }
  agent* zadnji()  const { return _red[_broj-1]; }
  // Vracamo velicinu reda cekanja
  size_t vrati_velicinu() { return _broj; };
};
// Klasa resursa
class resurs : public agent {
protected:
  // Tekuci broj zauzetih mesta u resursa
  size_t _broj;
  // Maksimalni broj  mesta u resursu
  size_t _kapacitet;
public:
  // Konstruktor
  resurs(size_t b):_broj(0),_kapacitet(b) {}
  // Zauzima mesto u resursu
  void zauzmi() {
    if(_broj<_kapacitet)
```

```
        // Uvecavamo broj zauzetih mesta
        ++_broj;
      else {
        cerr << "resurs::zauzmi - Sva mesta u resursu su zauzeta" << endl;
        exit(1);
      }
    }
    // Oslobadja mesto u resursu
    void oslobodi() {
      if(_broj>0)
        // Umanjujemo broj zauzetih mesta
        --_broj;
      else {
        cerr << "resurs::oslobodi - Nema entiteta u resursu" << endl;
        exit(1);
      }
    }
    // Raspolozivost resursa (ima makar jedno slobodno mesto u resursu)
    bool raspoloziv() { return _broj<_kapacitet; }
  };
}
#endif // __AES_PROC_H__
```

Kod datotetke izvornog koda `zad1.cpp`

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

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

// Vremenski trenuci dolaska korisnika (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 korisnika (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
red salt_red;
// Salteri
resurs salteri(4);
// Objekat simulacije
simulacija simu;

// 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;

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(vn);
    return dk;
  }
};

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;
  }
};

#ifdef STAMPA
void esa::stampaj_red() {
```

```
  for(size_t i=0; i<salt_red._broj; ++i) {
    cout << dynamic_cast<korisnik*>(salt_red._red[i])->vrati_id();
    if(i<salt_red._broj-1)
      cout << ",";
  }
}

void esa::stampaj_lbd() {
  for(size_t i=0;i<simu._broj;++i) {
    odlazak_korisnika* ok = dynamic_cast<odlazak_korisnika*>(simu._lbd[i]);
    if(ok) {
      cout << dynamic_cast<korisnik*>(simu._lbd[i]->vrati_agenta(0))->vrati_id();
      if(i<simu._broj-1)
        cout << ",";
    }
  }
}
#endif

// 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
  // Stampa u izvestaj
  cout << simu.vrati_vreme() << ": Korisnik " << kor->vrati_id() << " dolazi u postu." << 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();
    // Zauzmi jedno mesto u salteru
```

```
    salteri.zauzmi();
    // Rasporedi klijenta za kraj opsluge
    simu.rasporedi(odlazak_korisnika::napravi(prvi_kor,vreme_opsluge[prvi_kor->vrati_id()-1]));
  }
  #ifdef STAMPA
  // Stampa u izvestaj
  cout << simu.vrati_vreme() << ": Korisnici u redu: "; stampaj_red(); cout << endl;
  cout << simu.vrati_vreme() << ": Klijenti u sistemu: "; stampaj_lbd(); cout << endl;
  #endif
  // Stvaramo narednog korisnika
  novi_kor = new korisnik;
  if(novi_kor->vrati_id()<=13)
    // Postavljamo vreme narednog dolaska
    simu.rasporedi(dolazak_korisnika::napravi(novi_kor,vreme_dolaska[novi_kor->vrati_id()-1]));
  else
    delete novi_kor;
}

// Akcija dogadjaja
void odlazak_korisnika::akcija() {
  korisnik *kor, *prvi_kor;
  // Korisnik koji odlazi
  kor = dynamic_cast<korisnik*>(vrati_agenta(0));
  // Vrati salter
  salteri.oslobodi();
  #ifdef STAMPA
  // Stampa u izvestaj
  cout << simu.vrati_vreme() << ": Korisnik " << kor->vrati_id() << " odlazi iz poste." <<
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();
```

```
    // Zauzmi jedno mesto u salteru
    salteri.zauzmi();
    // Rasporedi klijenta za kraj opsluge
    simu.rasporedi(odlazak_korisnika::napravi(prvi_kor,vreme_opsluge[prvi_kor->vrati_id()-1]));
  }
  #ifdef STAMPA
  // Stampa u izvestaj
  cout << simu.vrati_vreme() << ": Korisnici u redu: "; stampaj_red(); cout << endl;
  cout << simu.vrati_vreme() << ": Korisnici u sistemu: "; stampaj_lbd(); cout << endl;
  #endif
}

int main() {
  korisnik *kor = new korisnik;
  // Postavljamo prvi dogadja u listu sto je odgovara dolasku prvog klijenta
  simu.rasporedi(dolazak_korisnika::napravi(kor,vreme_dolaska[kor->vrati_id()-1]));
  // Izvrsavamo simulaciju
  simu.izvrsi();
}
```