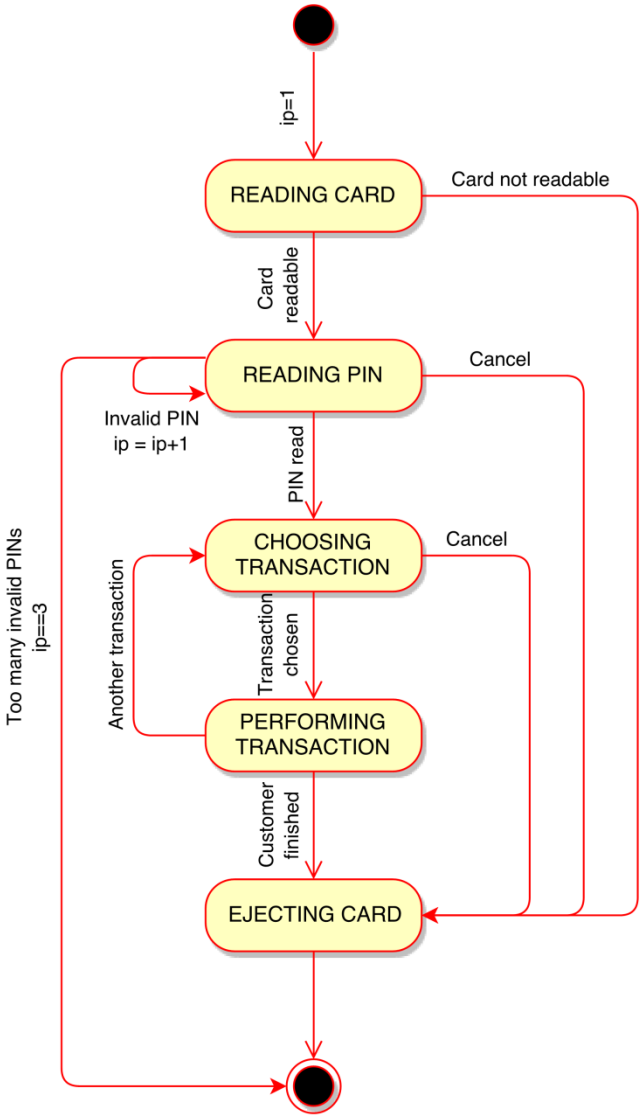
Simulirati rad bankomata čiji je rad opisan sledećim dijagramom stanja i tabelom prelaska iz stanje u stanje.



Stanje u / Stanje iz	READING CARD	READING PIN	CHOOSING TRANSACTION	PERFORMING TRANSACTION	EJECTING CARD	Final
Initial	*Start*, ip = 1 1.0	-	-	-	-	-
READING CARD	-	*Card readable* 0.95	-	-	*Card not readable* 0.05	-
READING PIN	-	*Invalid pin, ip++* 0.05	*PIN read* 0.70	-	Cancel 0.20	*Too many invalid PINs* ip==3
CHOOSING TRANSACTION	-	-	-	*Transaction chosen* 0.80	*Cancel* 0.20	-
PERFORMING TRANSACTION	-	-	*Another transaction* 0.40	-	*Customer finished* 0.60	-
EJECTING CARD	-	-	-	-	-	*Finish* 1.0

Rešenje:

Kod ovog rešenja stanja su deklarisana kao nabrojivi tip (enum). Događaji su predastavljeni kao klase. Promena stanja u klasi dijagrama stanja (statechart) se odigrava kroz preoprerećenu (overloaded) funkciju tranzicija koja za prosleđeni objekat događaja poziva odgovarajuću funkciju tranzicije (polimorfizam) i dovodi do promene stanja u kome se bankomat nalazi.

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

using namespace std;

// Funkcija vraca slucajan broj na intervalu od 0 do 1
double random() { return rand()/(1.0*RAND_MAX); }

// Stanja
enum state {
  reading_card,           // Citanje kartice
  reading_pin,            // Citanje PINa
  choosing_transaction,   // Izabiranje transakcije
  performing_transaction, // Transakcija
  ejecting_card           // Vracanje kartice
};

// Procitana kartica
class card_readable {};
// Ucitan pin kartice
class pin read {};
// Izabrana transakcija
class transaction_chosen {};
// Transakcija je zavrsena
class customer finished {};
// Kartica nije procitana
class card not_readable {};
// Otkazujem
class cancel {};
// Jos jedna transakcija
class another transaction {};
// Pogresan PIN
class invalid_pin {};


// Dijagram stanja
class statechart {
  // Zastavica prelaska u zavrsno stanje
  bool _final;
  // Tekuce stanje
  state _current;
public:
  // Konstruktor
  statechart() {}
  // Postavljamo pocetno stanje
  void initial(const state s) {
    _current = s;
  }
  // Tranzicija iz stanja u stanje za dogadjaj card_readable
  void transition(card_readable) {
    if(_current == reading_card) {
      _current = reading_pin;
      cout << "READING_PIN" << endl;
    }
    else {
      cerr << "Error" << endl;
      exit(1);
    }
  }
  // Tranzicija iz stanja u stanje za dogadjaj pin_read
  void transition(pin read) {
    if(_current == reading_pin) {
      _current = choosing_transaction;
      cout << "CHOOSING_TRANSACTION" << endl;
    }
    else {
```

```
      cerr << "Error" << endl;
      exit(1);
    }
  }
  // Tranzicija iz stanja u stanje za dogadjaj invalid_pin
  void transition(invalid pin) {
    if( current == reading pin) {
       current = reading pin;
      cout << "READING_PIN" << endl;
    }
    else {
      cerr << "Error" << endl;
      exit(1);
    }
  }
  // Tranzicija iz stanja u stanje za dogadjaj transaction_chosen
  void transition(transaction_chosen) {
    if( current == choosing transaction) {
       current = performing transaction;
      cout << "PERFORMING_TRANSACTION" << endl;
    }
    else {
      cerr << "Error" << endl;
      exit(1);
    }
  }
  // Tranzicija iz stanja u stanje za dogadjaj customer_finished
  void transition(customer finished) {
    if( current == performing transaction) {
       current = ejecting card;
      cout << "EJECTING_CARD" << endl;
    }
    else {
      cerr << "Error" << endl;
      exit(1);
    }
  }
  // Tranzicija iz stanja u stanje za dogadjaj card_not_readable
  void transition(card not readable) {
    if( current == reading card) {
       current = ejecting card;
      cout << "EJECTING_CARD" << endl;
    }
    else {
      cerr << "Error" << endl;
      exit(1);
    }
  }
  // Tranzicija iz stanja u stanje za dogadjaj cancel
  void transition(cancel) {
    if( current == reading pin ||
       current == choosing transaction) {
      _current = ejecting_card;
      cout << "EJECTING_CARD" << endl;
    }
    else {
      cerr << "Error" << endl;
      exit(1);
    }
  }
  // Tranzicija iz stanja u stanje za dogadjaj another_transaction
  void transition(another transaction) {
    if( current == performing transaction) {
       current = choosing transaction;
      cout << "CHOOSING_TRANSACTION" << endl;
    }
    else {
      cerr << "Error" << endl;
      exit(1);
    }
  }
public:
  // Postavi u finalno stanje i vrati da li je u finalnom stanju
  bool get final()                    { return  final;  }
  void set_final(const bool f)        { _final = f;     }
  // Postavi i vrati tekuce stanje
  void   set_current(const state s)   { _current = s;    }
  state  get_current() const          { return _current; }
```

```
};

int main() {
  // Brojac pogresno unetih PIN-ova
  size_t ip = 1;
  // Objekat dijagrama stanja
  statechart sc;
  // Seme generatora slucajnih brojeva iniciramo vremenom.
  // Na ovaj nacin sa svakim izvresenjem programa dobijamo druge vrednosti stanja.
  srand(time(NULL));
  // Postavljamo pocetno stanje
  sc.initial(reading card);
  cout << "READING CARD" << endl;
  // Prolazimo kroz sva stanja dijagrama stanja dok ne dodjemo do finalnog
  while(sc.get_current()!= ejecting_card) {
    if(sc.get_current()==reading_card) {
      double p = random();
      if(p<=0.05) {
        sc.transition(card not_readable());
        sc.set_final(true);
      }
      else {
        sc.transition(card_readable());
      }
    }
    else if(sc.get_current()==reading_pin) {
      double p = random();
      if(p<=0.20) {
        sc.transition(cancel());
        sc.set_final(true);
      }
      else if(p<=0.25) {
        if(ip<3) {
          sc.transition(invalid_pin());
          ++ip;
        }
        else {
          sc.set_final(true);
        }
      }
      else {
        sc.transition(pin_read());
      }
    }
    else if(sc.get current()==choosing_transaction) {
      double p = random();
      if(p<=0.20) {
        sc.transition(cancel());
        sc.set_final(true);
      }
      else {
        sc.transition(transaction_chosen());
      }
    }
    else if(sc.get_current()==performing_transaction) {
      double p = random();
      if(p<=0.40) {
        sc.transition(another_transaction());
      }
      else {
        sc.transition(customer_finished());
        sc.set_final(true);
      }
    }
  }
  return 0;
}
```