Test pitanja za vežbu u vezi bloka tema 8

1. Označiti tačne iskaze u vezi softverskih mašina stanja:
-služe za realizaciju programa (ili delova) koji predstavljaju niz uslovljenih pojedinačnih koraka
-realizuju se dobrim dekomponovanjem osnovnog algoritma
-nije ih moguće modelovati standardnim "bubble" dijagramima
-u sebi moraju sadržati bar jednu super-petlju

2. Ukoliko je petlja zadata u formi mašine stanja kao:

```
case x: if (test)
         telo_petlje;
        else
         state++;
        break;
```

u pitanju je:
-while petlja
-for petlja
-do-while petlja
-super-petja

3. Za datu funkciju realizovanu preko mašine stanja označiti elemente koji utiču na promenu stanja:

```
void merenje(void)
{
  switch(m_state)
  {
      case 0: if (isr_100)
                {
                  isr_100 = 0;
                  value = 0;
                  cnt = 10;
                  m_state++;
                }
                break;
      case 1: start_merenja();
                m_state++;
                break;
      case 2: if (kraj_merenja())
                {
                  value += read_value();
                  if (!(--cnt)
                      m_state++;
                  else
                      m_state--;
                }
                break;
      case 3: value /= 10;
                prikaz_OK = 1;
```

```
                    m_state = 0;
  }
}
```

-isr_100
-cnt
-m_state
-value
-prikaz_OK

4. Koji su najčešći razlozi da se neko stanje u neoptimizovanoj mašini stanja duže izvršava:
-poziv funkcije
-petlje
-složeniji matematički izraz
-prekid
-primena globalnih promenljivih

5. Kada se uporede rešenja na bazi softverskih mašina stanja (SFSM) i rešenja sa operativnim sistemima (OS) važe sledeći iskazi:
-SFSM rešenja su po pravilu manja od OS rešenja po pitanju zauzeća memorije
-SFSM rešenja mogu da koriste iste softverske mehanizme (indikatori, semafori i sl.) kao i u slučaju OS
-SFSM rešenja su po pravilu sporija od OS verzija
-kvalitet SFSM rešenja ne zavisi toliko od programera kao kod OS verzija

6. Poređati sledeće softverske mehanizme u rastućem redosledu od najmanjeg ka najvećem prema tome koliko memorije zauzimaju:
-indikator događaja
-semafor
-poruka
-"poštansko sanduče"

7. Zaokružiti tačne tvrdnje koje se odnose na FSM:

a)Osnovna ideja je podela algoritama na faze
b)Primena mašina stanja je ograničena samo na analizu sekvencijalnih podataka
c)Stanje FSM-a se menja u zavisnosti od jednog ili više uslova
d)Najveća prednost primene mašine stanja je u jednostavnosti pisanja programa

8. Zaokruzi tacnu tvrdnju/tvrdnje:

1) Indikator dogadjaja mora da bude velicine jednog bita
2) Semafor je promenljiva koja moze da ima samo dve vrednosti
3) Poruka moze biti sastavljena od jedne ili vise promenljivih standardnog C tipa
4) Postansko sanduce je promenljiva koja moze da sakupi vise poruka

9. Oznaciti netacnu tvrdnju vezanu za masine stanja:
a. Da bi se masina stanja mogla efikasno primeniti na program koji je slican radu sa multitasking operativnim sistemom, nije vazna brzina izvrsavanja koraka masine stanja.

b. Osnovni razlozi za sporo izvrsavanje unutar masine stanja mogu biti petlje, poziv neke funkcije ili slozeniji matematicki izraz.
c. Ako se iz masine stanja poziva neka funkcija cije izvrsavanje moze duze trajati, tada i ovakve funkcije treba realizovati u formi masine stanja, pri cemu se mora obezbediti indikacija da je pozvana funkcija u potpunosti zavrsila svoju sekvencu, odnosno postavljeni zadatak.
d. Promenljive svih masina stanja uvijek moraju biti globalne, jer moraju zadrzati vrednost i nakon napustanja funkcije u kojoj se koristi masina stanja.

10.Spojiti softverske mehanizme i objašnjenja za šta ona služe.

a)Event 1)Pomenljiva koja ima više od jedne vrednosti.

b)Semaphore 2)Indikator da se neki događaj dogodio.

c)Message 3)Bafer koji može da sakupi više poruka, koristeći princip FIFO bafera.

d)Mailbox 4)Razmena informacija između taskova.

11. Zaokruži netačne iskaze koji se odnose na mašinu stanja:
a)U mašini stanja, stanje (promenljiva state) može ostati nepromenjeno.
b)Petlju do-while ne treba dekomponovati jer poseduje samo telo petlje i test, za razliku od for i while petlji.
c) Mana programa realizovanog po principu mašine stanja je ta što softverski mehanizmi (npr. semafori i sl.) nisu dostupni.
d) Ukoliko je pozvana funkcija iz mašine stanja inicijalno u stanju 0, to stanje može, a i ne mora ponovo dostići nakon završetka definisanih koraka.
e)U poređenju sa OS, postoji zahtev za dodatnim kodom.

12. Spojiti standardne petlje (1, 2, 3) sa primerima njihovih dekomponovanih oblika u masini stanja (a, b, c):

Standradne petlje: Petlje u masini stanja:

1. for (inicijalizacija; test; finalizacija)
telo_petlje;
 	
2. while (test)
telo_petlje;
 	
3. do
telo_petlje;
 	
while (test);
	a) case x: telo_petlje;
	if (!test)
	state++;
	break;
	
	b) case x: if (test)
	telo_petlje;
	else
	state++;
	break;
	
	c) case x : inicijalizacija;
	state++;
	break;
	
	case x+1: if (test)

	{
	telo_petlje;
	finalizacija;
	}
	else state++;
	break;

13. Koja od navedenih funkcija realizovanih metodom mašine stanja odgovara sledećoj funkciji realizovanoj u beskonačnoj petlji:

```
void funkcija()
{
     while(1)
    {
         char taster = 0;
         while(ocitaj_taster() == 0)
              printf("Pritisni taster");

         taster = ocitaj_taster();

         while(taster == ocitaj_taster());

         for(int i=1; i<5; i++)
         {
             if(taster == i) printf("%d", i);
         }
    }
}
```

a)

```
 void funkcija()
 {
       switch(stanje)
      {
            case 0:  char taster = 0;
                     stanje++;
                     break;

            case 1: if(ocitaj_taster() != 0)
                             {stanje++;}
                    else
                             {printf("Pritisni taster");}
                    break;

           case 2: taster = ocitaj_taster();
                   stanje++;
                   break;

           case 3: if(taster != ocitaj_taster()) stanje++;
                   break;

           case 4: int i=1;
                   stanje++;
                   break;
```

```
                case 5: if(i<5)
                        {
                            if(taster == i)
                            {
                                printf("%d", i);
                                stanje++;
                            }
                        }
                        else
                            {stanje = 0;}
                         break;
            }
       }
```

b)

```
void funkcija()
     {
           switch(stanje)
          {
                case 0: if(ocitaj_taster() != 0)
                                {stanje++;}
                        else
                                {printf("Pritisni taster");}
                        break;

               case 1: taster = ocitaj_taster();
                       stanje++;
                       break;

               case 2: if(taster == ocitaj_taster()) stanje++;
                       break;

               case 3: int i=1;
                       stanje++;
                       break;

                case 4: if(i<5)
                        {
                            if(taster == i)
                            {
                              printf("%d", i);
                              stanje++;
                            }
                         }
                         else
                            {stanje = 0;}
                         break;
            }
       }
```

c)

```
void funkcija()
{
           switch(stanje)
           {
```

```
                case 0:  char taster = 0;
                         stanje++;
                         break;

                case 1: if(ocitaj_taster() != 0)
                             {printf("Pritisni taster");}
                        else
                             {stanje++;}
                        break;

               case 2: taster = ocitaj_taster();
                       stanje++;
                       break;

               case 3: if(taster != ocitaj_taster()) stanje++;
                       break;

               case 4: int i=1;
                       stanje++;
                       break;

                case 5: if(i<5)
                        {
                            if(taster == i)
                            {
                                  printf("%d", i);
                                  stanje++;
                            }
                        }
                        else
                           {stanje = 0;}
                        break;
          }
     }
```

d)

```
 void funkcija()
 {
           char taster = ocitaj_taster();
           switch(stanje)
           {
                case 0: if(taster == 0)
                          {stanje++;}
                        else
                          {printf("Pritisni taster");}
                        break;

               case 1: if(taster == ocitaj_taster()) stanje++;
                       break;

               case 2: int i=1;
                       stanje++;
                       break;

               case 3: if(i<5)
                       {
```

```
                        if(taster == i)
                        {
                             printf("%d", i);
                             stanje++;
                        }

                    }
                    else
                      {stanje = 0;}
                    break;
          }
     }
```

14. U sledecem primeru funkcije za rad sa dogadjajima popuniti oznacena mesta (__) odgovarajucim bitwise logickim operacijama kako bi se obezbedila zeljena funkcionalnost setovanja Event-a i testiranja Event-a

```
char events;

void SetEvent( char ev )
{
    events __ = ev;
}

char TestEvent( char ev )
{
   char tmp = events __ ev;
   events __ = ~ev;
   return ( tmp != 0 );
}
```

a) I I &

b) I & I

c) I & &

d) & I &

15. Od ponuđenih iskaza obeležiti one koju su tačni:

a) Iz mašine stanja se mogu pozivati funkcije koje nisu realizovane u formi mašine stanja.

b) Ukoliko je pozivana funkcija realizovana u vidu mašine stanja, mora se obezbediti indikacija o završetku njene sekvence.

c) Ukoliko je pozivana funkcija realizovana u vidu mašine stanja, njen poziv je uvek potrebno izvršiti u bar dva koraka.

d) Ukoliko koristimo promenljive stanja u realizaciji mašina stanja, one moraju biti globalne.

16. Za standardnu FOR petlju

```
for (int i=0; i<10 ; i++)
     telo_petlje;
```

ispraviti grešku u dekomponovanoj formi, tako da ona radi korektno:

```
case x : int i=0;
            state++;
            break;
case x+1: if (i<10)
              { telo_petlje; }
          else
              state++;
          break;
```

a) Suvišan je break, jer ćemo svakako preći na naredno stanje
b) Nedostaje finalizacija iza tela petlje u IF naredbi
c) Nedostaje finalizacija u else delu IF naredbe
d) FOR petlja se mora inicijalizovati van case-a