Dežurni nastavnik: _____

Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Testiranje softvera (SI3TS)
Nastavnik: doc. dr Dragan Bojić
Asistent: dipl. ing. Dražen Drašković
Ispitni rok: Jun 2013.
Datum: 16.06.2013.

Kandidat:* _____

Broj indeksa:* _____

Ispit traje 3 sata, prvih sat vremena nije dozvoljeno napuštanje ispita.
Upotreba literature nije dozvoljena.

Zadatak 1	_____/6	*Zadatak 5*	_____/9
Zadatak 2	_____/6	*Zadatak 6*	_____/9
Zadatak 3	_____/12	*Zadatak 7*	_____/9
Zadatak 4	_____/9		

Ukupno na ispitu: **_____/60** ***Ukupno na domaćem*:*** **_____/40**

Rok u kome je odbranjen domaći*: **_____** (primer: januar 2013)

Ukupno: _____/100

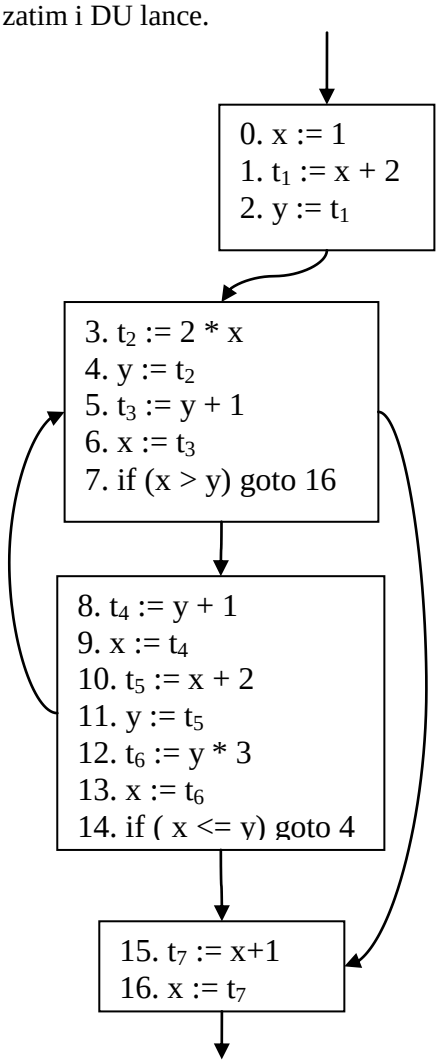
Ocena: _____ (_____)

Napomena: Ukoliko u zadatku nešto nije dovoljno precizno definisano, student treba da uvede razumnu pretpostavku, da je uokviri (da bi se lakše prepoznala prilikom ocenjivanja) i da nastavi da izgrađuje preostali deo svog odgovora na temeljima uvedene pretpostavke. Kod pitanja koja imaju ponuđene odgovore treba **samo zaokružiti** jedan odgovor. Na ostala pitanja odgovarati **čitko, kratko i precizno**. * popunjava student.

---

1. [6] a) Objasniti tehniku detekcije deadlock-a u programima sa više niti, kada ne može da se konstruiše *wait-for* graf.
   b) Koja su ograničenja pristupa opisanog pod a)?

2. [6] Globalnom iterativnom analizom toka podataka sračunati IN i OUT skupove svih osnovnih blokova, a zatim i DU lance.

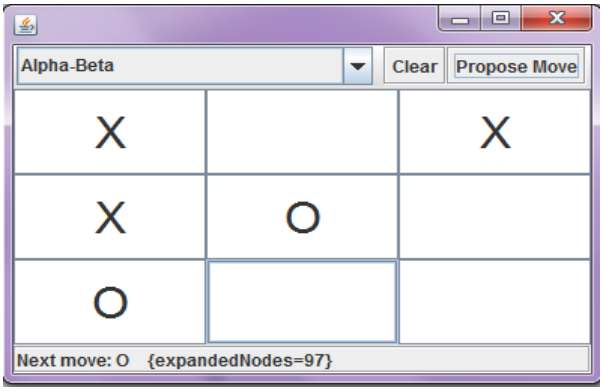


3. [12] Neka je data Java aplikacija koja realizuje igru *Tic-Tac-Toe* (iks-oks). Korisnik može iz padajuće liste da odabere jedan od dva tipa algoritma: *Alpha-Beta* ili *Minimax*, da obriše tablu i da pita računar za automatski potez (*Propose Move*).

   Nacrtati EFG (*Event Flow Graph*) i napisati sve moguće sekvence od trenutka prikazanog na slici, pri čemu su sledeći koraci izvršeni od početka igre:

   - igrač X igrao poziciju A1
   - igrač O igrao poziciju B2
   - igrač X igrao poziciju B1
   - igrač O igrao poziciju C1
   - igrač X igrao poziciju A3

   (redovi su označeni sa A, B, C, a kolone sa 1, 2, 3)



   Da li navedene sekvence garantuju da će se izvršiti sve varijante - pobeda igrača X, pobeda igrača O i nerešen ishod? Označiti koje su to sekvence.

4. [9] a) Neka je data sledeća funkcija *select*. Pretpostaviti da se funkciji dostavljaju ispravno svi argumenti. Razviti test primere za dati program metodom pokrivanja odluka i svih višestrukih uslova.

```
1. int select ( int a[], int n, int x)
2. {
3.    int i = 0;
4.    while ( i < n && a[i] < x )
5.    {
6.       if (a[i] < 0)
7.          a[i] = - a[i];
8.       i++;
9.    }
10.   return 1;
11.   }
```

b) Za sledeći program napisati sve test primere koji testiraju pozivanje funkcije *sample*.

```
 1. main ( )
 2. {
 3.  int a, b, i ;
 4.  printf ("Unesite sledece vrednosti: a, b, i");
 5.  scanf (" %d %d %d ", &a ,&b, &i);
 6.  if ( i < 10 ) {
 7.     sample(a, b);
 8.     i = i + 1;
 9.  }
10. }
11. sample( int x , int y ) {
12.   if ( x > 10 )
13.   x = x + y; break;
14.   if ( y > 10 )
15.   y = y + x; break;
16. }
```

5. [9] Dat je algoritam koji sortira niz od n elemenata. Testirati sve petlje u ovoj proceduri i detaljno obrazložiti:

```
public static void insertion_sort(int array[],
int n){
 for (int i = 1; i < n; i++){
  int j = i;
  int B = array[i];
  while ((j > 0) && (array[j-1] > B)) {
      array[j] = array[j-1];
      j--;
  }
  array[j] = B;
 }
}
```

6. [9] Neka je dat sledeći program.

a) Odrediti sve LCSAJ za dati program (poziv funkcije get_request smatrati kao sekvencijalni programski kod bez skokova)

b) Odrediti minimalan skup testova koji pokrivaju sve LCSAJ.

```
1. begin
2. int x,y;
3. int product, request;
4. #define exp=1
5. #define fact=2
6. #define exit=3
7. get_request(request); //vraca jedan od tri zahteva
8. product=1;
9. while(request!=exit){
10.      if(request==exp) {
11.         input(x,y);
12.         count=y;
13.         while(count>0){
14.            product=product*x;
15.            count=count-1;
16.         }
17.      } else if(request==fact){
18.            input(x);
19.            count=x;
20.            while(count>0){
21.               product=product*count;
22.               count=count-1;
23.            }
24.      } else if(request==exit){
25.            output("Hvala sto koristite program");
26.            break;
27.      }
28.      output(product);
29.      get_request(request);
30.      } //while
31.      end
```

7. [9] a) U programskom kodu u zadatku 5, zameniti FOR petlju sa WHILE petljom, a WHILE petlju sa FOR petljom, tako da ne narušite ispravnost programa. Zatim koristeći mutacioni operator SWDD (operator zamene WHILE-DO petlje sa DO-WHILE petljom) napisati novi program i ukoliko je to moguće napisati test primere koji rade, odnosno ne rade kao početni kod, pre primene mutacionog operatora.

b) Napisati funkciju u programskom jeziku Java, koja računa kvadratni koren pozitivnog realnog broja x (x se prosleđuje kao argument funkciji), a zatim primeniti mutacioni operator koji Javinu funkciju za korenovanje, zamenjuje Javinom funkcijom za stepenovanje *pow(double arg1, double arg2)*.

Navesti bar pet test primera koji generišu i žive i mrtve mutante i izračunati mutacioni rezultat. Od realizovanih test primera implementirati bar tri JUnit testa za dati program. Ukoliko je potrebno, definisati potrebne izmene nad programom kako bi testiranje bilo moguće.

c) Kada imamo FOR petlju, da li se izraz za inkrementiranje (ili dekrementiranje) petlje, i++ (ili i--) mutira, ako koristimo unarni operator mutacije? Obrazložiti odgovor.