Nome/cognome _____ N. di matricola (10 cifre) _____ Posizione: Riga _____ Col _____

UNIVERSITA' DI BOLOGNA - CORSO DI LAUREA IN INFORMATICA
CORSO DI SISTEMI OPERATIVI - ANNO ACCADEMICO 2003/2004
COMPITO CONCORRENZA - 18 Giugno 2004

Esercizio -1: essersi iscritti correttamente per svolgere questa prova.
Esercizio 0: Su entrambi i fogli, scrivere correttamente nome, cognome, matricola e posizione prima di svolgere ogni altro esercizio.

Esercizio 1:
a) Scrivere una coppia di processi che stampi la stringa
a.1 {AB|BA}C{DE|ED} (se il vostro numero di matricola è pari)
a.2 {AC|CA}{BA|AB}F (se il vostro numero di matricola è dispari)
facendo uso di **semafori**. (Da leggersi: stampa AB oppure BA, dopo di che stampa C, dopo di che stampa DE oppure ED)

Esercizio 2
Scrivere un meccanismo di "sincronizzazione di barriera" parametrizzabile per un numero N di processi. Lo schema è il seguente:

```
P[i]:
  // codice prima della barriera
  sincronizzazione di barriera
  // codice dopo della barriera
```

In altre parole, vogliamo che tutti i processi si sincronizzino in modo tale che un *qualsiasi* processo possa eseguire il codice *dopo* la barriera solo dopo che *tutti* i processi abbiano completato il codice *prima* della barriera. Scrivete il codice utilizzando **semafori**, fornendo il codice di sincronizzazione ed eventuali inizializzazioni dei semafori stessi. E' possibile utilizzare array di semafori.

Esercizio 3
Presso una spiaggia italiana ci sono un certo numero di campi da beach volley. Normalmente, i campi non vengono utilizzati per tornei "ufficiali"; si usa invece un approccio "auto-organizzante". La vita di un giocatore di beach é la seguente:

```
giocatore:
  while (ha voglia di giocare)
    s = campo.forma_squadra() ;
    do {
       campo.iniziaGioco(s);
       // Gioca partita: batti, ricevi, alza, schiaccia
       campo.finePartita(s);
     } while (la squadra s ha vinto);
 end while
```

Finché ha voglia di giocare, un giocatore resta al campo e cerca di formare una squadra tramite la p.e. **formaSquadra**. Una squadra è formata quando ha dimensione N; fino a quando la dimensione della squadra è inferiore a N, il giocatore aspetta a bordo campo. **formaSquadra** ritorna un valore intero che identifica univocamente la squadra.
Una volta che la squadra è formata, la squadra si accoda per giocare. Ogni giocatore della squadra chiama la p.e. **iniziaGioco**. Se il campo è libero (c'è solo un altra squadra oppure il campo è vuoto), la squadra può iniziare a giocare (anche da sola, intanto inizia a palleggiare). Altrimenti, la squadra deve attendere il proprio turno FIFO.
Una volta iniziato a giocare, la partita termina solo quando entrambe le squadre decidono di terminare, chiamando **finePartita**. In altre parole, un giocatore resta in attesa che tutti gli altri giocatori in campo abbiano deciso di smettere chiamando **finePartita**.
Al termine della partita, ne viene iniziata una nuova. La squadra che ha vinto resta in campo (squadra che vince non si cambia), mentre la squadra che ha perso si rompe e i suoi giocatori (se ne hanno ancora voglia) cercano di formare una squadra nuova.
Risolvere il problema tramite **monitor**. E' possibile utilizzare strutture dati (array associativi, code, etc) contenenti condizioni.
Nota: non è necessario determinare qual è la squadra vincente.

Esercizio 4
Si considerino le seguenti operazioni atomiche:
f(x) = < return x++; >
g(x) = < return ++x; >
E' possibile risolvere il problema della mutua esclusione con f(x)? E con g(x)? Se sì, come? Se no, perché?