Informatica 3 – Proff. Ghezzi e Morzenti – Secondo recupero – 16 settembre 2003

Cognome e Nome ..... Numero Matricola .....
(in stampatello)

Recupero della ☐ PRIMA ☐ SECONDA prova in itinere (barrare una o entrambe)

Risolvere i seguenti esercizi, scrivendo le risposte ed eventuali tracce di soluzione negli spazi disponibili.

NON CONSEGNARE ALTRI FOGLI. (spazio a uso dei docenti)

RECUPERO PRIMA PROVA

Esercizio 1

Si consideri il seguente frammento di programma in un ipotetico linguaggio di programmazione (che chiameremo LP), il quale adotta passaggio dei parametri per valore:

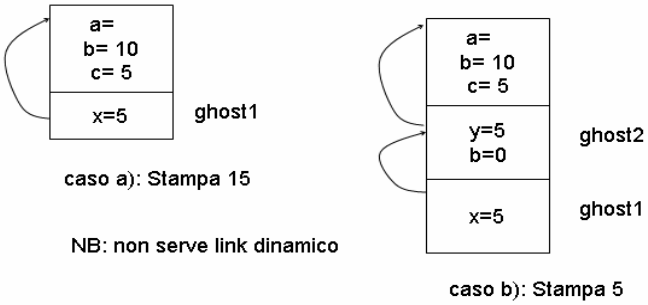
```
int a, b, c;
int function ghost1 (x) {
    a = x+b;
    return a;
}
int function ghost2 (y) {
    int b = 0;
    return (ghost1(y)); @@@
}
b = 10; c=5;
print (ghost1(c));  ###
print (ghost2(c));
```

1. Si ipotizzi che LP adotti delle regole di "scope" dinamiche. Disegnare lo stato della memoria della macchina astratta nei seguenti due casi: subito dopo che e' stata chiamata la funzione ghost1 nel punto ### e subito dopo che e' stata chiamata ghost1 nel punto @@@ per effetto della chiamata di ghost 2 che appare nell'ultima istruzione del programma. Spiegare anche quali valori stampano le due istruzioni print.

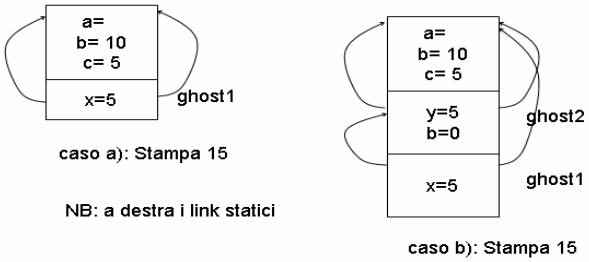
2. Si ipotizzi che LP adotti delle regole di "scope" statiche. Disegnare lo stato della memoria della macchina astratta nei seguenti due casi: subito dopo che e' stata chiamata la funzione ghost1 nel punto ### e subito dopo che e' stata chiamata ghost1 nel punto @@@ per effetto della chiamata di ghost 2 che appare nell'ultima istruzione del programma. Spiegare anche quali valori stampano le due istruzioni print.

SOLUZIONE

Domanda 1



Domanda 2



Esercizio 2

Che cosa calcolano le seguenti funzione LISP?

```
(DEFUN GUESS (ALFA BETA)
   (COND  ((NULL ALFA) NIL)
      ((MEMBER (CAR ALFA) BETA)  (GUESS (CDR ALFA) BETA))
      (T (CONS (CAR ALFA) (GUESS (CDR ALFA) BETA))) ))
```

(Si ipotizzi che ALFA e BETA siano due liste che contengono elementi atomici e che la funzione MEMBER dia risultato vero se un atomo e' membro di una lista)

SOLUZIONE: calcola una lista che contiene tutti gli atomi di BETA e tutti quelli di ALFA che non sono in BETA. Se ALFA e BETA memorizzano due insiemi, la funzione calcola l'unione.

```
(DEFUN WHAT (L)
   (COND ((NULL L) NIL)
      ((ATOM L) L)
      (T (CONS (WHAT (CAR L))
            (WHAT (CDR L))))))
```

SOLUZIONE: restituisce una lista uguale a L

Esercizio 3

Si consideri la seguente classe Java che definisce un monitor.

```
public class Example {
    private int n;
    private int total;
public synchronized void op1 (int item) {
        while (!(total < n))
            try { wait(); }
            catch (InterruptedException e) { }
        //parte di programma non specificata
        total++;
        notify();
    }
public synchronized void op2 () {
            while (!(total > 0))
            try { wait(); }
            catch (InterruptedException e) { }
        //parte di programma non specificata
        total--;
        notify();
    }
```

1. Si supponga che nel programma Java si generino due thread T1 e T2 che accedono concorrentemente a un oggetto O della classe Example. Si realizzi lo stesso sistema mediante task Ada, descrivendo in particolare il task che gestisce l'accesso all'oggetto condiviso. (Schizzare la soluzione senza preoccuparsi dei dettagli sintattici del linguaggio.)
2. Quale politica segue Ada nella gestione dei task sospesi? E quale politica segue Java?
3. Si supponga di voler imporre che in Java la soluzione segua la stessa politica di risveglio dei task che segue Ada. Si descriva a parole una possibile implementazione, senza scendere in dettagli implementativi.

SOLUZIONE

1) Lo schema del task che realizza il controllore dell'oggetto astratto è il seguente:

```
Integer N, TOTAL;
loop
    select
        when TOTAL<N
            accept op1() do....;
            end
    or
        when TOTAL>0 do...
        end:
    end select;
end loop;
```

2) Ada segue una politica fifo. Java lascia la scelta nondeterministica.

3) Si potrebbe definire un oggetto, che possiamo chiamare CODA, di tipo coda FIFO (definito da una classe Java) da associare al monitor. Prima di ogni istruzione wait occorre aggiungere istruzioni per inserire in coda il riferimento al task che viene sospeso. Ad ogni risveglio del task (dopo quindi le istruzioni wait) occorre che il task risvegliato dalla NotifyAll verifichi se esso è il primo task di CODA. Se si, si toglie dalla coda e procede, altrimenti si rimette in wait.

RECUPERO SECONDA PROVA

Esercizio 1

Due alberi binari T e T' si dicono *isomorfi* se e solo se

- T e T' sono entrambi vuoti, **oppure**
- T e T' sono entrambi non vuoti e inoltre, detti T_s e T_d i sottoalberi sinistro e destro di T e T'_s e T'_d i sottoalberi sinistro e destro di T_s, T_s è isomorfo a T'_s e T_d è isomorfo a T'_d.

Due alberi binari T e T' si dicono invece *equimorfi* se e solo se

- T e T' sono entrambi vuoti, **oppure**
- T e T' sono entrambi non vuoti e inoltre, detti T_s e T_d i sottoalberi sinistro e destro di T e T'_s e T'_d i sottoalberi sinistro e destro di T_s
    - T_s è equimorfo a T'_s e T_d è equimorfo a T'_d, **oppure**
    - T_s è equimorfo a T'_d e T_d è equimorfo a T'_s.

Implementare i seguenti due metodi Java

boolean isomorfi(BinNode t1, BinNode t2)

boolean equimorfi(BinNode t1, BinNode t2)

che, utilizzando l'interfaccia definita a lezione per la classe BinNode, stabiliscano se tra i due alberi loro passati come parametro valgono le relazioni sopra definite. Valutare la complessità asintotica di calcolo di tali metodi nel caso pessimo, come funzione sia della profondità *h* degli alberi, sia del numero *n* dei loro nodi.

SOLUZIONE

```
boolean isomorfi(BinNode t1, BinNode t2) {
  return (t1 == NULL && t2 == NULL) ||
         (t1 != NULL && t2 != NULL &&
           isomorfi(t1.left(),t2.left()) && isomorfi(t1.right(),t2.right()) );
  }

boolean equimorfi(BinNode t1, BinNode t2) {
  return (t1 == NULL && t2 == NULL) ||
         (t1 != NULL && t2 != NULL) &&
           (equimorfi(t1.left(),t2.left()) && equimorfi(t1.right(),t2.right())
             ||
            equimorfi(t1.left(),t2.right()) && equimorfi(t1.right(),t2.left()) );

  }
```

Il caso pessimo è quello di un albero pieno e completo, in cui *n* è $\Theta(2^h)$

In *isomorfi* c'è una doppia ricorsione, quindi la complessità cresce in modo esponenziale con base 2 nella profondità dell'albero, quindi è $\Theta(2^h)$ e $\Theta(n)$.

In *equimorfi* ci sono 4 chiamate ricorsive, quindi si ha complessità $\Theta(4^h)$ e $\Theta(n^2)$.

Esercizio 2

Un algoritmo di ordinamento è detto *stabile* se, in caso di presenza di elementi duplicati (cioè aventi lo stesso valore di chiave) la loro posizione relativa tra nell'array di partenza è mantenuta invariata nell'array ordinato. L'algoritmo *bubblesort* è stabile? E quello di *quicksort*? Motivare accuratamente la risposta anche facendo riferimento al codice degli algoritmi.

SOLUZIONE

Bubblesort è stabile perché non può mai accadere che un valore venga scambiato con uno adiacente con chiave uguale, a causa della condizione di minore stretto della condizione dell'if nella riga 4 del codice.

Quicksort NON è stabile: in una chiamata del metodo *partition* due valori con la stessa chiave potrebbero essere oggetto di due successivi swap (istr. 13) in due iterazioni successive del ciclo do-while (istr.10-14), e quindi trovarsi, nella partizione risultante alla fine della chiamata di *partition*, in posizione relativa diversa da quella in cui si trovavano all'inizio.

Esercizio 3

Data una tabella hash di lunghezza m=11, si supponga di dover inserire (in ordine) le chiavi: 24, 42, 53, 163, 6, con la funzione di hash h(k) = k mod m e usando la tecnica di double hashing con $h_2(k) = 1+(k \bmod (m-1))$. Si illustrino i risultati dell'inserimento, riportando il dettaglio dei calcoli effettuati.

SOLUZIONE

24 mod 11 = 2 => 24 va in posizione 2

42 mod 11 = 9 => 42 va in posizione 9

53 mod 11 = 9 => posizione 9 occupata; p(53, 1) = 1*(1+(53 mod 10))=4; (9+4) mod 11 = 2, posizione 2 occupata; p(53, 2) = 2*4=8; (9+8) mod 11 = 6; va in posizione 6

163 mod 11 = 9 => posizione 9 occupata; p(163, 1) = 1*(1+(163 mod 10))=4; (9+4) mod 11 = 2, posizione 2 occupata; p(163, 2) = 2*4=8; (9+8) mod 11 = 6; posizione 6 occupata; p(153, 3) = 3*4=12; (9+12) mod 11 = 10; va in posizione 10

6 mod 11 = 6 => posizione 6 occupata; p(6, 1) = 1*(1+(6 mod 10))=7; (6+7) mod 11 = 2, posizione 2 occupata; p(6, 2) = 2*7=14; (6+14) mod 11 = 9; posizione 9 occupata; p(6, 3) = 3*7=21; (6+21) mod 11 = 5; va in posizione 5