Politecnico di Milano
Facoltà di Ingegneria dell'Informazione
Informatica 3
Prof. Ghezzi, Lanzi e Morzenti
Prova di recupero
16 Febbraio 2005

COGNOME E NOME (IN STAMPATELLO)

MATRICOLA

RECUPERO I PARTE ☐ Es. 1,2,3,4

RECUPERO II PARTE ☐ ES. 5,6,7

Risolvere i seguenti esercizi, scrivendo le risposte ed eventuali tracce di soluzione negli spazi disponibili.
Barrare le caselle relative alle parti recuperate. Non consegnare altri fogli.

Spazio riservato ai docenti

Esercizio 1. Parte I. Considerando il seguente programma:

```
int a = 1;        /* distanza di a= 0 */
int b = 0;        /* distanza di b= 0 */

int dec()
{
      int x=-20;        /* distanza di x= 0 */
      x = a++;          /* distanza di a=1 */
      if (x < 3) return ( inc() - x ); /* distanza di inc= 1 */
      else return 1;
}

int inc()
{
      int x=0;                      /* distanza di x= 0 */
      x = b--;  /*@@ */             /* distanza di b= 1 */
      return ( dec() * x );         /* distanza di dec= 1 */
}

main()
{
      int x = 12;       /* distanza di x= 0 */
      int a = 20;       /* distanza di a= 0 */
      b = 10;           /* distanza di b= 1 */
      print (2 * inc()); /* distanza di inc= 1 */
}
```

Quesito 1. A fianco di ogni istruzione, si scriva l'attributo di distanza per le variabili usate nell'istruzione o per la funzione ivi chiamata. Scrivere le istruzioni SIMPLESEM che implementano dell'istruzione "x = b--" identificata dall'etichetta @@ nel codice.

Soluzione SIMPLESEM:

set D[0]+3,D[fp(1)+1] con fp(1)=D[D[0]+2]

set fp(1)+1,D[fp(1)+1]-1

Esercizio 1. Parte I. (continua).

Quesito 2. Si descriva lo stato della memoria della macchina astratta subito dopo la TERZA chiamata di inc() e dopo l'assegnamento int x=0. Si mostrino in particolare la struttura dei record di attivazione, i link statici e dinamici e le variabili in essi allocate.

	0	Current	30
	1	Free	34
global	2	a	3
	3	b	8
main()	4	Return Pointer	##
	5	Dynamic Link	##
	6	Static Link	2
	7	x	12
	8	a	20
	9	Return Value	
inc()	10	Return Pointer	##
	11	Dynamic Link	4
	12	Static Link	2
	13	x	10
	14	Return Value	
dec()	15	Return Pointer	##
	16	Dynamic Link	10
	17	Static Link	2
	18	x	1
	19	Return Value	
inc()	20	Return Pointer	##
	21	Dynamic Link	15
	22	Static Link	2
	23	x	9
	24	Return Value	
dec()	25	Return Pointer	##
	26	Dynamic Link	20
	27	Static Link	2
	28	x	2
	29	Return Value	
inc()	30	Return Pointer	##
	31	Dynamic Link	25
	32	Static Link	2
	33	x	0

Esercizio 2. Parte I. Si consideri la seguente funzione

```
void func(int i, j) {
   int k, h;
   h = i++;
   k=++i;
   j=i+h+k;
}
```

Si considerino il seguente frammento:

```
int a=0; b=0;
int x[10] = {0,1,2,3,4,5,6,7,8,9};
func(a, b);
print(a, b);    /* 1 */
func(a, x[a])
print(a, x);    /* 2 */
```

Si descrivano i casi in cui il passaggio dei parametri avvenga per valore, per indirizzo e per valore-risultato.

Soluzione.

Passaggio parametri per valore:

1. stampa 0,0
2. stampa 0,0,1,2,3,…,9

Passaggio parametri per indirizzo:

1. stampa 2,4
2. stampa 4,0,1,10,3,…,9

Passaggio parametri per valore-risultato:

3. stampa 2,4
4. stampa 4,0,1,2,3,10,5,…,9

Esercizio 3. Parte I. Indicare l'output prodotto da questo programma Python, motivando la semantica delle istruzioni della funzione f. Risposte non motivate non verranno prese in considerazione.

```
def f(a, b, c, d):
      a.append(5)
      b = [3, 5]
      c[0] = 9
      d = 4

a = [0]
b = [1]
c = [2]
d = 3
f(a,b,c,d)
print a, b, c, d
```

Soluzione.

Il programma stampa [0, 5] [1] [9] 3

Le istruzioni:

1) aggiunge un elemento alla lista del chiamante, modificandola.
2) l'assegnamento annulla il binding con la lista passata come parametro e ne effettua uno nuovo, lasciando quindi inalterato l'oggetto originale.
3) L'assegnamento modifica la lista del chiamante, cambiando il binding di un suo elemento.
4) l'assegnamento annulla il binding con l'intero passato come parametro e ne effettua uno nuovo, lasciando quindi inalterato l'oggetto originale.

Esercizio 4. Parte I. Si consideri un programma concorrente in Java in cui più thread possono accedere a un oggetto ST di tipo stack condiviso (una pila). Per tale oggetto sono definiti due metodi "push" e "pop", di tipo "synchronized". Push ha per parametro una lista di elementi, che vengono inseriti in cima alla pila. Pop ha invece come parametro un intero K, e il suo effetto e' quello di prelevare una lista di K elementi dalla cima della pila, restituendola al chiamante. Si supponga che ST possa avere una capacità illimitata e quindi non esista un limite al numero di elementi che possono essere memorizzati. Occorre invece che i threads che cercano di eseguire l'operazione "pop" possano sospendersi nel caso ST non contenga il numero voluto di elementi.

Quesito 1. Tratteggiare la soluzione in Java.

```
public synchronized void push(List l) {
   Iterator i = l.iterator();

    while(l.hasNext()) {
        pila.addFirst(l.getNext());
   }

   notifyAll();
}

public synchronized List pop(int k) {
  while(pila.size() < k) {
      wait();
   }

    List l = new LinkedList();

    while(l.size() < k) {
       l.add(pila.removeFirst());
    }
 }
```

Quesito 2. Tratteggiare la soluzione nel caso di Ada, che utilizza come schema per la concorrenza il rendez-vous.

In Ada non esiste una semplice soluzione perché nella clausola when di una select non è possibile valutare il parametro della entry citata nella corrispondente accept; il valore del parametro risulta noto solo DOPO l'esecuzione della accept e quindi non può essere utilizzato per decidere l esecuzione dell'accept stessa. Una possibilita' consiste nel fare in modo che la chiamata di pop sia sempre accettata dal task guardiano, ma che venga restituito un esito di "estrazione non avvenuta" se il numero di dati memorizzati e' inferiore a quello richiesto. Il task chiamante dovrebbe dunque inserire la chiamata al rendezvous all'interno di un ciclo, fino a che avviene l'estrazione.

Esercizio 5. Parte II. Si consideri un albero binario di ricerca (BST) che goda anche della proprieta' di essere completamente equilibrato (un albero completamente equilibrato è un albero pieno in cui tutte le foglie hanno la stessa distanza dalla radice).

1. qual e' la complessita' della ricerca di un elemento nel caso pessimo? Giustificare la risposta.
2. si tratteggi un algoritmo che elenchi tutti i valori minori o uguali a un certo valore X. Si valuti la sua complessita' nel caso ottimo, pessimo e medio.
3. si supponga nota l'altezza H dell'albero. Si tratteggi un algoritmo che calcola quanti sono i valori memorizzati minori di un certo valore X (senza necessariamente visitare tutti i nodi che contengono tali valori). Si valuti la complessita' dell'operazione nel caso ottimo, pessimo e medio.

Soluzione.

1. lg n
2. Un possibile algoritmo potrebbe essere il seguente:

```
/* T è il albero, v è il valore */

MinoreUguali(T,v) {
if (T==NULL) return;

if (T.valore>v) {
  /* discende il sottoalbero sinistro T.left */
  MinoreUguali(T.left,v);
} else if (T.valore==v) {
  print(v);
  MinoreUguali(T.left,v);
} else {
  MinoreUguali(T.left,v);
  print(v);
  MinoreUguali(T.right,v);
}
}
```

   La complessità è data dalla somma della complessità per cercare l'elemento X più la complessità dell'algoritmo sopra esposto.
   Il caso ottimo si ha quando l'elemento X è minore uguale del più piccolo tra gli elementi dell'albero. In tal caso la complessità totale sarà lg n (costo per cercare l'elemento più piccolo) + 1 (costo dell'algoritmo MinoreUguali) = lg n.
   Il caso pessimo si verifica invece quando X è maggiore uguale dell più grande degli elementi dell'albero quindi lg n(costo per cercare il più grande elemento) + n (costo per elencare tutti i valori) = n.
   In media vengono stampati n/2 elementi, quindi lg (n) (costo per cercare X) + n/2 (costo medio per elencare gli elementi) = n/2

3. Il numero dei nodi inferiori a un nodo X a profondità d è dato dalla formula $2^{h-d-1} - 1$. Calcolare tale valore ha costo costante (O(1)) e quindi la complessità dipende esclusivamente dall'algoritmo di ricerca.

   Quindi, il caso ottimo si verificherà quando l'elemento cercato coincide con la radice (O(1)).
   Il caso medio e il caso pessimo coincidono perchè saranno sempre O(lg(n)).

Esercizio 6. Parte II. Si consideri una tabella di hash con 13 slot, numerati da 0 a 12, a cui si accede con un hashing doppio definito dalle due funzioni:

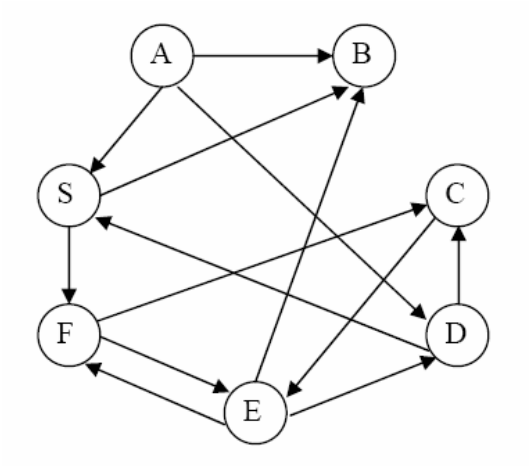
- H1(k) = k mod 13
- H2(k) = (Rev(k+1) mod 11)

In cui la funzione Rev(k) restituisce il numero ottenuto rovesciando le cifre del numero k, ad es. Rev(37)=73; Rev(7)=7. Si mostri lo stato della tabella di hash dopo che sono state inserite le seguenti chiavi: 2, 8, 31, 20, 19, 18, 53, 27.

Soluzione:

0	1	2	3	4	5	6	7	8	9	10	11	12
	53	2	27		31	19	20	8			18	

Esercizio 7. Parte II. Si consideri il seguente grafo:



supponendo che il grafo venga visitato a partire dal nodo S, e che la scelta del nodo da visitare prima sia basata sull'ordinamento alfabetico delle etichette dei nodi, si dica,

- in quale ordine i nodi del grafo verranno visitati se si utilizza una visita depth-first.
- in quale ordine i nodi del grafo verranno visitati se si utilizza una vista breadth-first.

Soluzione.

In entrambe i casi sono SBFCED