Politecnico di Milano
Facoltà di Ingegneria dell'Informazione
Informatica 3
Prof. Ghezzi, Lanzi, Matera e Morzenti
Prima prova in itinere
6 Maggio 2005

COGNOME E NOME (IN STAMPATELLO)

MATRICOLA

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. Quesito 1. Specificare distanza e offset per tutte le variabili presenti nel codice.

```
int x, y;
power() {
   testForError()  {
      if ( y > 0)
         return 1;
      else {
         print("Error in Exponent");
         return -1;
      }
   }
   calcPower() {
      if ( y == 1)
         return x;
      else {
         y = y-1;
         return x*calcPower();
      }
   }
   if ( testForError() == -1)
      return -1;
   else
      return calcPower();
}
main() {
   x = 2;
   y = 2;
   print( power() );
}
```

Risposta:

```
power() {
   testForError()  {
      y: <2,1>
   }
   calcPower() {
      y: <2,1>
      x: <2,0>
   }
}
main() {
   x: <1,0>
   y: <1,1>
}
```

Esercizio 1. (continua).

Quesito 2. Descrivere lo stato della macchina astratta subito dopo l'ultima attivazione della funzione calcPower.

	0	Current	16
	1	Free	19
global	2	x	2
	3	y	1
main()	4	Return Pointer	##
	5	Dynamic Link	##
	6	Static Link	2
	7	Return Value	
power()	8	Return Pointer	##
	9	Dynamic Link	4
	10	Static Link	2
	11	Return Value	
calcPower()	12	Return Pointer	##
	13	Dynamic Link	8
	14	Static Link	8
	15	Return Value	
calcPower()	16	Return Pointer	##
	17	Dynamic Link	12
	18	Static Link	8

Esercizio 2. Quesito 1. Si consideri il seguente programma C.

```
int *p; int i=0;
void f(int* k) {
      p=k;
}
void g() {
      int i=10;
      f(&i);
}
main() {
      p=&i;
      g();
      printf("%d\n", *p)
}
```

Questo programma fa sorgere un serio errore concettuale.

1. Dire tecnicamente di quale errore si tratti e descriverlo schizzando lo stato della memoria a run-time con le relative celle e i relativi l-value e r-value. Con riferimento a tale schizzo della memoria, spiegare quale valore verrebbe comunque prodotto in uscita dal programma.
2. Spiegare se e perche` un analogo errore puo` sorgere o meno in Java.

Risposta:

1) Dangling reference: lo stato della memoria prima del termine di f è quella rappresentata in figura dopo di che f e g vengono deallocate dallo stack, ma p punta ancora a 10 (la variabile i locale a g) che è oltre il Free. In uscita dal programma, supponendo che non vi siano altri thread o interrupt il programma stamperebbe 10 cioè il contenuto della cella 10.

	0	Current	11
	1	Free	15
global	2	P	10
	3	I	11
Main()	4	Return Pointer	
	5	Dynamic Link	
	6	Static Link	
g()	7	Return Pointer	
	8	Dynamic Link	
	9	Static Link	
	10	I	10
f()	11	Return Pointer	
	12	Dynamic Link	
	13	Static Link	
	14	K	10

2) In java tale problema non può accadere perchè non è possibile usare i puntatori in modo esplicito. In particolare non è possibile avere puntatori a tipi base mentre gli handler agli oggetti mantengono in vita gli oggetti a cui puntano. Il garbage collector infatti elimina un oggetto solo quando non esiste più nessun handler per quell'oggetto.

Esercizio 2. Quesito 2.
Si consideri il seguente fragmento di programma in un dialetto inventato di C.

```
int i=10; int k=20; /* dichiarazioni globali */
int pippo (int m) {
  int * n = &m;
  *n = 30;
  n++;
  *n = 40;
  return m;
}
main () {
      print(pippo(i), i, k); /* istruzione di output dall'ovvio significato */
}
```

Si descriva che cosa produce il programma schizzando lo stato della memoria a run-time con le relative celle e i relativi l-value e r-value, nelle sguenti due ipotesi:

1. il passaggio del parametro avviene per valore;
2. il passaggio del parametro avviene per indirizzo.

Risposta

1. Stampa `30, 10, 20` (siccome i parametri sono passati per valore `i` e k non sono modificati)
2. Stampa `30, 30, 40` (con la semantica del passaggio per indirizzo, m è un alias di i quindi le modifiche su m si riflettono su i. Inoltre, in virtù dell'aliasing, quando n acquisice l'indirizzo di m, ottiene in realtà l'indirizzo di i. Conseguentemente, `n++` fa puntare n alla cella successiva sullo stack (ossia k). La modifica successiva avviene quindi su k)

Esercizio 3. Si considerino due thread Java X e Y e due oggetti di classi rispettivamente A e B. L'oggetto di classe A offre due metodi pubblici synchronized ma1 e ma2; l'oggetto di classe B offre due metodi pubblici synchronized mb1 e mb2.

Si supponga che ma1 invochi mb2 e che mb1 invochi ma2.

1. Si tratteggi il codice Java relativo alle classi A e B.
2. Quale grave situazione di errore puo` capitare se X chiama ma1 sull'oggetto di classe A e Y chiama mb1 sull'oggetto di classe B? Mostrare con un esempio l'insorgere della situazione di errore.
3. Per eliminare la suddetta situazione di errore si potrebbe ricorrere alla seguente soluzione, che viene solo tratteggiata sommariamente; eliminare le clausole synchronized dai metodi delle classi A e B e introdurre un oggetto di una nuova classe C che funge da monitor. La classe C esporta due metodi mc1 e mc2 che si incaricano di chiamare rispettivamente ma1 e mb1. Si chiede di descrivere questa soluzione in maniera sufficientemente precisa, in modo tale da poter dimostrare con un semplice ragionamento l'eliminazione della sopra analizzata situazione di errore.

1.

```
class A {

   ...
   synchronized public void ma1(B b) {
      ...
      b.mb2();
   }

   synchronized public void ma2() {
      ...
   }
 }
```

```
class B {

   ...
   synchronized public void mb1(A a) {
      ...
      a.ma2();
   }

   synchronized public void mb2() {
      ...
   }
 }
```

2. Il codice esposto al punto 1 può creare un *deadlock*. Una possibile situazione in cui può verificarsi è la seguente. Supponiamo che inizialmente siano stati creati due oggetti x e y rispettivamente di tipo A e B e che poi siano stati eseguite le istruzioni `x.setB(y)` e `y.setA(x)`. Successivamente il thread `X` esegue l'invocazione di ma1 sull'oggetto di classe A (acquisendone quindi il lock) ma prima di terminare l'esecuzione del metodo il controllo passa al thread Y. Questi, a sua volta, invoca mb1 sull'oggetto di classe `B` ma si blocca quando tenta di invocare `ma2` sull'oggetto a perchè quest'ultimo è bloccato dal thread `X`. Il controllo passa ritorna quindi a X ma anche questo thread non può procedere con l'esecuzione perchè l'oggetto `b` è bloccato da Y. Siccome entrambi i thread non posso continuare l'esecuzione il sistema è bloccato permanentemente (*deadlock*).
3. Con riferimento alla soluzione tratteggiata al punto 1 e supponendo di aver rimosso synchronized dalla signature da tutti i metodi, si consideri la seguente classe aggiuntiva:

```
class C {
  A a;
  B b;

  public C(A a, B b) {
    this.a = a;
    this.b = b;
  }

  synchronized void mc1() {
    a.ma1();
  }
```

```
    synchronized void mc2() {
        b.mb1();
    }
}
```

Si supponga inoltre che sia stata creato un oggetto di tipo C, accessibile sia da X che da Y che invocano rispettivamente mc1 e mc2. Questa soluzione elimina i problemi evidenziati al punto 2 in quanto se per esempio X è il primo thread ad essere eseguito, si ha la garanzia che Y non potrà invocare mc2 fin quando X non ha rilasciato il lock, permettendo così a X di invocare (indirettamente tramite C) prima ma1 e poi mb2 senza che Y riesca ad invocare mb1. Chiaramente lo stesso discorso vale anche per Y qualora venga selezionato per primo.

Esercizio 4. Dato il seguente codice Python: calcolare l'output del programma, per ogni istruzione di output giustificare il valore delle variabili stampate (output non giustificati non verranno presi in considerazione).

```
def f(a,b):
    a=a+1
    if (a==1):
        b.append(2)
    elif (a==2 or a==3):
        b()
    else:
        b=b+1
def g():
    print a
a=0
b=[1]
f(a,b)
print a,b
a=a+1;
b=g
f(a,b)
a=a+2
f(a,b)
print b
```

Risposta:

0 [1,2]
1
eccezione

1) Il passagio parametri è per reference ma a=a+1 non è un operazione sull'oggetto ma invece provoca un nuovo binding, quindi questa modifica non è visibile al chiamante. Append viceversa è un operazione sulla lista quindi la modifica è visibile al chiamante
2) La visibilità in python è statica quindi all'atto della chiamata di g() la a che viene stampata è quella globale che vale ancora 1 mentre quella locale a f vale 2
3) Alla terza chiamata di f il parametro b è una funzione e a vale 3 quindi in f a vale 4 e si entra nel ramo else. Dato che b è una funzione non si può "incrementare" di uno. Python segnala quindi un errore di tipo (Typechecking Dinamico)