

5 Febbraio 2009:

7) scope statico, passaggio per valore (per copia, i parametri attuali non vengono modificati) ed eccezioni.

int a = 6; 5 4 3

B void g() {

a--;

write(a); stampa 5 4

throw X;

}

stamperà 5 → 8 → 4 → 3

Se questo sollevamento non trova subito il suo gestore, si va a cercarlo nel blocco più esterno e, se non c'è ancora, un gestore di default fa scattare un errore.

C void f(int x) {

int a = 9; 8

if (x >= 0) g();

else try {g();} catch X {a--; write(a); f(0);}

}

②  
8

try {f(1);} catch X {a--;}

write(a); 3

④

8) La classe B estende (extends) A, quindi "aggiunge a quello che c'è già" quello che ha nella sua definizione, e il campo X

percui i) Da A, B eredita le funzioni f() ed s(int y); Di I implementa i metodi f() e s(); Di J implementa il metodo g()

ii) Visto che

A a = new B();

$\Rightarrow a.x = 10 \Rightarrow b.x = 20$

Assegna alla X  
della classe A il  
valore 10

Assegna alla X  
della classe A, da cui  
l'ha ereditato, il  
valore 20

a. f() applica la funzione f() della classe B

$\downarrow$   
 $x = 19$

b. f() applica la funzione f() della classe B

$\downarrow$   
 $x = 18$

↓  
stampa 18  
18

B b = (B) a;

CORSO DI PARADIGMI DI PROGRAMMAZIONE  
PROVA SCRITTA DEL 13 LUGLIO 2009.

Tempo a disposizione: ore 2.

1. Si consideri la grammatica  $G = (\{A, B\}, \{a, b\}, A, P)$  dove  $P$  è l'insieme seguente:

$$\begin{aligned} A &::= aAa \mid B \\ B &::= BbbB \mid \epsilon \end{aligned}$$

Si consideri la seguente affermazione: "Per ogni  $k \geq 0$ ,  $\mathcal{L}(G)$  contiene una stringa di lunghezza  $k$ ".  
Si dica se è vera o falsa. Se si ritiene che sia vera, si fornisca una giustificazione (informale o una dimostrazione); se si ritiene falsa, si mostri un  $k$  ed una stringa lunga  $k$  che non sta in  $\mathcal{L}(G)$ .

2. Si consideri la seguente definizione di funzione in un linguaggio con gestione della memoria con pila di RdA:

```
int f(int n,m){  
    if (n==0) return 1;  
    else return f(n-1,m+n); Ricorsione in codice e solo RdA
```

Quanti record di attivazione sono certamente necessari su una macchina astratta per calcolare  $f(3,0)$ ? Si dia una sintetica giustificazione.

3. In uno pseudolinguaggio con eccezioni (try/catch) si incontra il seguente blocco di codice:

```
public static void ecc() throws X {  
    throw new X();  
}  
public static void g (int para) throws X {  
    if (para == 0) {ecc();}  
    try {ecc();} catch (X) {write(3);} 3  
}  
public static void main () {  
    try {g(1);} catch (X) {write(1);} 0  
    try {g(0);} catch (X) {write(0);} 0  
}
```

Si dica cosa viene stampato all'esecuzione di `main()`.

4. Si dica cosa stampa il seguente frammento in uno pseudolinguaggio con passaggio per valore-risultato:

```
int X[10];  
int i = 1;  
X[0] = 10;  
X[1] = 10;  
X[2] = 10;  
void foo (value-result int Y,J){  
    X[J] = J-1;  
    write(Y);  
    J++;  
    X[J]=J;  
    write(Y);  
}  
foo(X[i],i);  
write(X[i]);
```

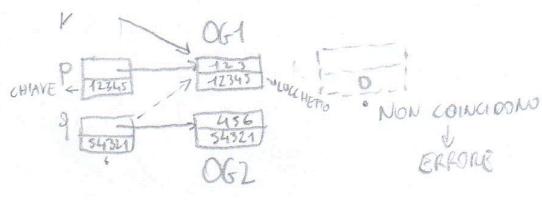
5. Si consideri la dichiarazione di array multidimensionale `int A[10][10][10]`. Sappiamo che: un intero è memorizzato su 4 byte; l'array è memorizzato in ordine di riga, con indirizzi di memoria crescenti (cioè se un elemento è all'indirizzo  $i$ , il successivo è a  $i+4$  ecc.) Qual è l'offset dell'elemento  $A[3][3][4]$  rispetto all'elemento  $A[0][0][0]$ ? (Si risponda in notazione decimale).

$$[(3 \cdot 10 \cdot 10) + (3 \cdot 10) + 4] \cdot \text{sizeof(int)}$$

```

int *p = (int*) malloc(sizeof(int))
int *q = (int*) malloc(sizeof(int))
*p = 123
*q = 456
q = p
free(p)

```



6. Si assuma di avere uno pseudolinguaggio che adotti la tecnica dei *locks and keys*. Se *OGG* è un generico oggetto nello heap, indichiamo con *OGG.lock* il suo lock (nascosto); se *PTR* è un generico puntatore (sulla pila o nello heap), indichiamo con *PTR.key* la sua key (nascosta). Si consideri il seguente frammento di codice:

```

P 12345
C foo = new C(); // oggetto OG1
C bar = new C(); // oggetto OG2
C fie = foo;    fie.key = og1.lock
bar = fie;      bar.key = og1.lock

```

$$\text{OG1.lock} = \text{foo.Key}$$

$$\text{bar.lock} = \text{OG1.lock} = 12345$$

$$\text{OG2.lock} = \text{bar.Key} = 54321$$

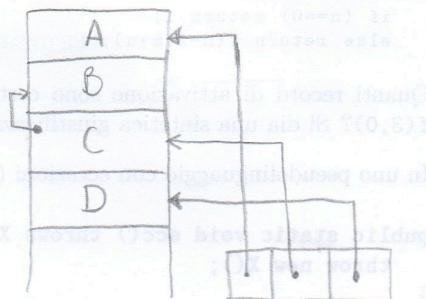
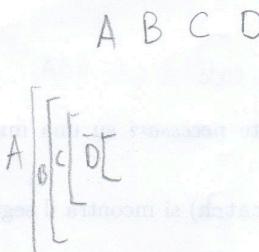
Si diano possibili valori di *OG1.lock*, *OG2.lock*, *foo.key*, *fie.key* e *bar.key* dopo l'esecuzione del frammento.

7. È dato il seguente frammento di codice in uno pseudolinguaggio con scope statico gestito con display:

```

int x = 5;
int y = 4;
void B(){
    int x = 4;
    int z = 3;
    C();
}
void C(){
    int x = 3;
}
void D(){
    int x = 2;
}
D();
B();
1 2 3

```



Si rappresenti graficamente il display subito dopo che il controllo è entrato nella funzione D.

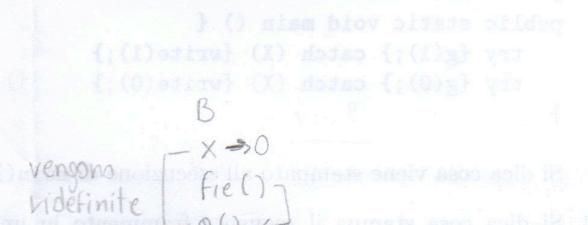
8. Si considerino le seguenti classi Java:

```

public class A {
    int x = 5;
    int fie () {return g();}
    int g() {return x;}
}

public class B extends A{
    int x = 0;
    → int g() {return x;}
}

```



undefined

Si consideri adesso il seguente frammento di codice:

```

B b = new B();
A a = b;
int zz = a.fie() + a.x ;

```

Si dica qual è il valore di *zz* al termine dell'esecuzione del frammento.

Si veda come già [01] [01] [01] A mai si nota cambiamenti verso il progresso si troveranno [01] [01] [01] B mentre le scritte non segnali di scritte di ottenere il tutto. Però se si esegue un'operazione di ordinamento delle scritte si troverà [01] [01] [01] B, così si ottiene una struttura di memoria (esempio: memoria (01) [01] [01] B, a cui si aggiunge [01] [01] B).

29 Giugno 2009;

$$\begin{aligned}
 A &::= aA \mid bA \mid B \\
 B &::= abBb \mid C \\
 C &::= C \mid \epsilon
 \end{aligned}$$

- 1) a) Falso, perché la stringa "acb" che posso formare con la grammatica G è dispari  
b) Falso, perché la stringa "ab" è un "pari"  
c) Vero, il simbolo  $\epsilon$  è appunto la stringa vuota  
d) Falso, perché applicando le regole  $1 \rightarrow 1 \rightarrow 2 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 7$  ottengo proprio la stringa "aabbb"  
e) Vero, perché applicando, a piacimento, le regole  $1 \rightarrow 2$  ottengo proprio le palindrome su a, b e  
con  $3 \rightarrow 5 \rightarrow 7$  chiudo la stringa formato con una stringa vuota.

F) Falso, il linguaggio L generato dalla grammatica G è:  $L = \{a^n b^m a^q c^o b^q \mid n, m, q, o \geq 0\}$

le lettere con esponente diverso possono essere ripetute indipendentemente dalle altre, mentre quelle con esponente 0 vengono ripetute assieme o non ci sono.

- 8)  $I^L_{L_1} (C^L_{L_2, L_3}, C^L_{L_2, L_3})$  questo darà come output  $C^L_{L_2, L_3} (C^L_{L_2, L_3})$  che a sua volta restituirà un programma compilato scritto in  $L_3$

*Progr. in  $L_1$       input al Progr.*

Questi devono essere uguali  $\Rightarrow$  OK!

Devono essere uguali  $\Rightarrow$  OK!

- 3) Con il passaggio per nome, per risolvere un frammento di codice, devo sostituire a tutte le occorrenze del parametro formale della procedura/funzione, con il parametro attuale presente nella chiamata alla funzione. Bisogna stare attenti però; I parametri sostituiti mantengono il valore che avevano ~~dal~~ al momento della chiamata, mentre le variabili interne alla procedura/funzione useranno il loro valore locate alla funzione.

```

void foo (name int X[], i) {
    X[i] = i - 1;
    write (X[i]);
    i++ = 2;
    X[i] = 1;
    write (X[i]);
}

```

foo(x[i], i);  
write (x[i]);

$x[0]$	$x[1]$	$x[2]$	$i$
10	10	10	1

Stampa  $0 \rightarrow 8 \rightarrow 2$

- 1) Caso 1: f(x) si riconosce Parberry così:  $A[5] = \{0, 1, 2, 3, 4\}$

Con il for si riempie l'array così: A[0] = {0, 1, 2, 3, 4}, applicando l'assegnamento A[x++] = A[x++]+x si avrà A[5] = {0, 2, 2, 3, 4}

$$X = X + 1$$

valutazione  
prima dell'assegnamento

$$\text{Se } x=1 \rightarrow A[1] = A[1]+1 \Rightarrow A[1] = 1+1 = 2$$

- 5) Scambiare gli l-value significa, per una variabile, cambiare il riferimento alla locazione di memoria in cui è memorizzato il suo valore. Quindi ...

$X \rightarrow 1 \Rightarrow X \rightarrow 2 \Rightarrow X \rightarrow 3 \Rightarrow X \rightarrow 3 \Rightarrow X \rightarrow 4 \Rightarrow$  STAMPA 4, 3  
 $Y \rightarrow 2 \quad Y \rightarrow 1 \quad Y \rightarrow 1 \quad Y \rightarrow 4 \quad Y \rightarrow 3$   
 1            2            3            4            5            6

6) Scope Statico, con Display (n° di celle dell'Array = n° livelli di annidamento), con identificatori noti staticamente.  
Ogni nome usato è usato in 1 solo blocco.

NL = Accesso ad una variabile NON Locale X al blocco ]- Queste sono le 2 operazioni da prendere in esame.  
L = Accesso ad una variabile Locale Y

i) NL = Non dipende dal numero di variabili, ma, sfruttando un array per i livelli di annidamento, la tecnica del display permette di ridurre gli accessi a 2.

L = idem come sopra

ii) NL = idem come sopra

L = idem come sopra

iii) NL = Non dipende dai RdA, ma gli accessi sono ridotti a 2

L = idem come sopra.

iv) NL = Non dipende dai blocchi

L = Non dipende dai blocchi

v) Si, il tempo è costante perché il numero degli accessi in memoria a 2

vi) No, la v) è la risposta giusta

Come impostarsi le domande per rispondere correttamente: iniziare a leggere i vari punti anteponendo la frase:

"Il tempo necessario all'esecuzione dell'operazione di (NL, L) dipende da ---"

7) goto, scope dinamico, blocchi annidati etichettati.

Illustrare il CRT con coda esplicita al momento (\*\*)

Il CRT o Central Referencing Table raccoglie in una tabella tutti nomi usati nel programma dai vari blocchi. Per ogni nome è presente un flag che indica se l'associazione per quel nome è attiva o no. e un puntatore al blocco in cui la variabile è dichiarata in quel momento. Quando un blocco termina, vengono levate dalla lista anche le informazioni relative a quel blocco.

1) Quali sono i nomi coinvolti nel frammento di codice? (Variabili, procedure, ...)

X, Y, Z,  $\Rightarrow$  Tabella di 3 celle

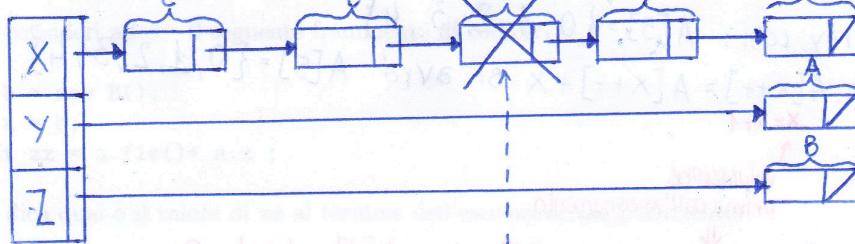
X
Y
Z

2) Contare quante volte il nome viene ripetuto nel codice e preparare i blocchi procedere al "linkaggio":

X x5

Y x1

Z x1



Il blocco D termina e

tutti i suoi nomi, in questo caso solo X,  
vengono tolti dalla lista aggiornandola opportunamente.

8) In esercizi come questi bisogna sempre tenere conto questo:

Quando ci si riferisce a un campo (es. a.X) ci si riferisce al campo x della classe/tipo di a ( $A.a = b$ )

Quando ci si riferisce a una procedura (es. a.fie()), dopo che ad a, è stato assegnato il riferimento di un altro oggetto b di tipo/classe B, la procedura da tenere in considerazione è quella della classe B.

Quindi  $\text{int}zz = a.\text{fie}() + a.X \quad zz = 5$