Esercizi di Linguaggi di Programmazione

Luca Bedogni

April 16, 2015

1 - Eccezioni

```
{
  int y =0;
  void f( value-result int x ) {
    x = x +1;
    throw E;
    x = x +1;
  }
  try { f(y ); } catch E {};
  write (y );
}
```

2

```
int x=2
void foo (name int y){
  x = x + 1;
  y = y + 20;
  x = x + y;
  write (x);
}
{
  int x=50
  foo (x);
  write (x);
}
```

3 - riferimento

```
{
  int y =0;
  void f( int x ) {
    x = x +1;
    if y = 1 throw E;
    x = x +1;
  }
  void g( int x ){
    try { f(x ); } catch E { write (y +10)}
  }
```

```
  try { g(y ); } catch E { write (y +20)};
  write (y +30);
}
```

4 - nome

```
int x = 3;
void foo( name int y) {
  int x = 5;
  x = x + y;
  x = x + y;
  write (x );
  write (y );
}
foo(x++);
write(x);
```

5 - statico, dinamico con deep binding, dinamico con shallow binding

```
{
  void foo ( int f () , int n ){
    int m = 10;
    int fie (){
      write (n ,m );
    }
    if ( n == 0)
      f ();
    else {
      m = 30;
      foo ( fie ,0);
    }
  }
int g (){
  write (10);
}
foo (g , 1);
}
```

6 - statico, dinamico con deep binding, dinamico con shallow binding

```
int x = 3;
procedure stampa_x () {
  write_integer ( x );
}
procedure ass_x (n: int ) {
  x = n;
  if (n =1)
    stampa_x ();
}
procedure pippo ( function S , P ; int n ) {
```

```
  int x = 10;
  if (n = 1)
    ass_x (n)
  else {
    S(n);
    P ();
  }
}
pippo ( ass_x , stampa_x , 1);
pippo ( ass_x , stampa_x , 2);
```

7 - eccezioni

```
void f () throws X {
  throw new X ();
}
void g (int sw) throws X {
  if ( sw == 0) {
    f ();
  }
  try {f ();}
  catch (X e) {
    write (" in g" );}
  }

try {g (1);}
catch (X e) {
  write (" in main " );
}
```

8 - statico, dinamico con deep binding, dinamico con shallow binding

```
int x = 700;
int n = 30;
void g (){
  write (n+x)
}
void foo ( int f () , int n ) {
  if (n ==0)
    f();
  else
    foo (f ,0);
  g ();
}
{
  int x = 5;
  foo(g,1);
}
```

9 - dimensionamento e memorizzazione Si consideri la seguente definizione di tipo record:

```
type S = struct {
int x;
char y;
};
```

Si supponga che un int sia memorizzato su 2 byte, un char su 1 byte, su un'architettura a 16 bit con allineamento alla parola. In un blocco viene dichiarato un vettore:

```
S A [10];
```

Indicando con PRDA il puntatore all'RdA di tale blocco, e con ofst l'offset tra il valore di PRDA e l'indirizzo iniziale di memorizzazione di A, si dia l'espressione per il calcolo dell'indirizzo dell'elemento A[4].y.

10 - eccezioni

```
void ecc ( int para1 ) {
  try {
    throw new X ();
  } catch (X) {
    write (1);
  }
}
void f ( int para2 ) {
  if ( para2 == 1) {
    ecc (1);
  }
  try {
    ecc (2);
  } catch (X ) {
    write {2);
  }
}
void main () {
  try {f (1);} catch (X ) { write (3);}
  try {f (2);} catch (X ) { write (4);}
}
```

11 - L'esecuzione del seguente frammento di codice su una certa implementazione risulta nella stampa dei valori 4 e 1.

```
int W[10];
int x = 4;
for (int i=0, i<10, i++)
  W[i]=i;
void foo(int x; int y) {
  x = x+1;
  y=1;
```

```
}
foo (x, W[x])
write (W[4])
write (W[5])
```

Si fornisca una possibile spiegazione.

12 - dimensionamento e memorizzazione Si consideri la seguente dichiarazione di array multidimensionale

```
int A [10][10];
```

Come viene memorizzato l'array A? Inoltre sappiamo che: un intero é memorizzato su 4 byte; l'array é memorizzato in ordine di riga, con indirizzi di memoria crescenti (cioé se un elemento é all'indirizzo i, il successivo é a i + 4 ecc.). Qualé l'offset dell'elemento A[1][5] rispetto all'inizio dell'array?

13 - passaggio per valore-risultato e per nome

```
void foo {valuaresult/name int x, valuaresult/name int y) {
  x = x + 1
  y = 1
}
int i = 1
int[] A = new int[5]
A[1] = 4
foo(i,A[i])
```

14 - passaggio per nome

```
int i = 2
int foo(name int y) {
  return y + y
}

int a = foo(i++)
write(a)
write(i)
```

15 - garbage collector Si consideri il seguente frammento in uno pseudolinguaggio con modello delle variabili a riferimento e garbage collector mediante contatori dei riferimenti; se OGG é un generico oggetto nello heap, indichiamo con OGG.cont il suo contatore (nascosto).

```
class C { int n; C next ;}
C foo (){
  C p = new C ();     // oggetto OGG1
  p.next = new C (); // oggetto OGG2
  C q = new C ();     // oggetto OGG3
  q.next = p.next ;
  return p.next ;
}
C r = foo ();
```

Si dica quali sono i valori dei contatori dei riferimenti dei tre oggetti dopo l'esecuzione della linea 6 e della linea 9.