Politecnico di Milano
Facoltà di Ingegneria dell'Informazione
Informatica 3. Prof. Ghezzi, Lanzi e Morzenti
Prima Prova in Itinere
29 Aprile 2004

COGNOME E NOME (IN STAMPATELLO)

MATRICOLA

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

(spazio per i docenti)

Esercizio 1.

Si supponga di voler definire un nuovo linguaggio di programmazione object-oriented LANG. LANG ha un costrutto di classe e di sottoclasse. Si vuole che gli oggetti siano allocabili sia nello stack che nello heap, secondo una sintassi del tipo

```
MyClass pippo;   //alloca nello stack un oggetto di classe MyClass
MyClass* pluto;  //alloca nello heap un oggetto di classe MyClass, al quale fa
                 //riferimento la variabile Pluto
```

I progettisti di LANG discutono se sia per le variabili allocate nello stack che quelle allocate nello heap si possano eseguire assegnamenti polimorfi e si possa realizzare binding dinamico. In tal caso, l'assegnamento

```
pippo = paperino;
```

sarebbe possible se **paperino** fosse oggetto di una sottoclasse di **MyClass**, e l'operazione

```
pippo.op(…);
```

eseguirebbe l'operazione `op` eventualmente ridefinita per tale sottoclasse.

1. Qual è la vostra opinione in merito a questa discussione? In particolare, ritenete che la scelta in merito all'adozione delle regole di binding dinamico e polimorfismo sia compatibile con la scelta di allocare gli oggetti nello stack? Perchè?
2. In C++ e` possibile allocare le istanze di una classe sia nello stack che nello heap? Se si, quale scelta fa il linguaggio in merito alla possibilità di assegnamanto polimorfo e di binding dinamico?
3. In Java e` possibile allocare le istanze di una classe sia nello stack che nello heap? Se si, quale scelta fa il linguaggio in merito alla possibilità di assegnamanto polimorfo e di binding dinamico?

Risposta:

1. scelta sbagliata: quale spazio riservare per gli oggetti nello stack se la sottoclasse puo` aggiungere attributi?
2. In C++ e` possible allocare sia nello stack che nello heap, ma l'assegnazione per variabili nello stack non e` polimorfa. Viene fatta coercion.
3. In Java tutte le variabili istanze di class e sono allocate nello heap.

Esercizio 2.
Si consideri il seguente frammento di programma in un ipotetico linguaggio:

```
int i =0;
int f(int k) {
      if (i  == 0 ) i++;
      return (k+i);
}
main () {
      int i=1;
      print f(10);
}
```

Indicare il risultato prodotto dall'esecuzione del programma nei seguenti due casi, fornendo anche una sintetica spiegazione:
1. il linguaggio adotta regole di "scope" statiche;
2. il linguaggio adotta regole di "scope" dinamiche.

Risposta:

Se lo scoping e' statico il programma stampa 11 perche' dentro la funzione f la variabile i e' quella globale quindi si entra nell'if, i viene incrementata e il valore restituito (e poi stampato) e' quindi 11

Se lo scoping e' dinamico il programma stampa ancora 11, nella funzione f la variabile i e' quella del main quindi vale gia' 1, non si entra nel ramo if e il valore restituito e' quindi ancora 11

Esercizio 3. (primo quesito).
Si consideri il seguente frammento di programma in un ipotetico linguaggio:
Si consideri il seguente frammento di programma:

```
int i=0;
void f1 (int h) {
      struct {
            int a [10];
            int b;
      } k;
      int f2 (int j) {
     . . .
            k.b = j + i;    (*)
            k.a[h] = i;       (**)
            . . .
      }
      . . .
}
main ( ) {
      . . .
}
```

1. Si calcoli la coppia <distanza, offset> per ciascun elemento delle istruzioni (*) e (**), ipotizzando che nel record di attivazione siano allocati, nell'ordine, il link statico, il link dinamico, l'indirizzo dell'istruzione di ritorno, i parametri, le variabili locali e infine l'eventuale spazio per il risultato restituito dalle funzioni chiamate

Risposta

(*) j = <0,3>
i = <2,0>
k.b = <1,14>

(**) i = <2,0>
h = <1,3>
k.a[h] = <1,4+h>

Esercizio 3. (secondo quesito).

2. Si descriva lo stato della memoria dopo la seguente sequenza di chiamate: main chiama f1 che chiama f2 che chiama f1.

Risposta:

	0	Current	###
	1	Free	###
global	2	**i** in 0	
Main()	3	Static Link	2
	4	Dynamic Link	###
	5	Return Pointer	
f1()	6	Static Link	2
	7	Dynamic Link	3
	8	Return Pointer	
	9	h	
	10	k.a[0]	
	11	k.a[1]	
	12	k.a[2]	
	13	k.a[3]	
	14	k.a[4]	
	15	k.a[5]	
	16	k.a[6]	
	17	k.a[7]	
	18	k.a[8]	
	19	k.a[9]	
	20	k.b	
	21	Return Value	
f2()	22	Static Link	6
	23	Dynamic Link	6
	24	Return Pointer	
	25	j	
f1()	26	Static Link	2
	27	Dynamic Link	22
	28	Return Pointer	
	29	h	
	30	k.a[0]	
	31	k.a[1]	
	32	k.a[2]	
	33	k.a[3]	
	34	k.a[4]	
	35	k.a[5]	
	36	k.a[6]	
	37	k.a[7]	
	38	k.a[8]	
	39	k.a[9]	
	40	k.b	

Esercizio 4.

Si consideri il seguente programma scritto in una versione semplificata di Java.

```
class Panic {. . .};
class MyClass {
      ...
      private void f ( ) throws Panic {
            ...
            throw Panic ( ); //non esiste catch di Panic nella funzione f
            ...
      }
      void g ( ) {
            ...
            try {...
                 f ( );
                 ...
            } catch Panic ( ) { ... }
            ...
      }

public static void Main ( ) {
      ...
      try {...
           f ( );
           ...
      } catch Panic ( ) { ... }
}
```

1. Basandosi su quanto già noto (da altri corsi) sul meccanismo delle eccezioni in Java, si indichi se il binding tra l'eccezione sollevata (nell'esempio, l'eccezione Panic sollevata nella funzione f) e il corrispondente gestore (la catch corrispondente) è statico o dinamico. Si motivi sinteticamente la risposta considerando in particolare il caso in cui l'eccezione viene sollevata dopo la chiamata di f da g oppure dopo la chiamata di f da main.

Risposta

Dinamico. Infatti ogni volta che in quella riga viene sollevata un eccezione il catcher invocato puo' essere diverso a seconda di chi ha chiamato la funzione che solleva l'eccezione. Ad esempio nel caso in cui f venga chiamata da g il catcher che la raccoglie e' quello di g. Nel caso che f sia chiamata dal main e' quello del del main.

2. Quando può essere risolto il binding: al tempo di compilazione o di esecuzione?

Risposta

Dato che la propagazione dell'eccezione segue la catena dinamica e quindi dipende da un informazione che e' disponibile solo a tempo di esecuzione il binding puo' essere risolto solo a tempo di esecuzione.

Esercizio 5.

1. Definire nel linguaggio Python la funzione map(f, lis) che, a partire da una funzione f() (che si suppone abbia un argomento) e da una lista seq = [e1 ... en] restituisca la lista [f(e1)...f(en)].

```
def map(f, lis):
      s=[]
      for i in lis
            s.append(f(i))
      return s
```

2. Definire inoltre una funzione accumulate(f, lis, zero) che a partire da una funzione f() (che si suppone abbia due argomenti) e da una lista seq = [e1 e2 ... en] e dal valore zero restituisca il valore f(...f(f (zero, e1),e2) ... , en).

```
def accumulate(f, lis, zero):
      res = zero
      for i in lis :
            res = f(res, i)
      return res
```

3. Facendo uso della funzione add (definita qui sotto), della funzione ithOdd(i) (pure definita qui sotto, restituisce l'i-simo numero dispari), delle funzioni map e accumulate, ed eventualmente di altri noti costrutti del linguaggio Python, definire la funziona squareBySum(n) che calcola il quadrato del numero n, assunto intero positivo, come somma dei primi n numeri dispari (Es. 5 => [0 1 2 3 4] => [1 3 5 7 9] => 1+3+5+7+9 => 25).

```
def add(a, b):
     return a+b
```

```
def ithOdd(i):
     return 2*i+1
```

```
def squareBySum(n):
      return accumulate(add, map(ithOdd, range(n) ), 0)
```