Informatica 3 **TE 19 Settembre 2005 Esercizio 1**

Dato il seguente codice, scrivere le istruzioni SIMPLESEM relative alle istruzioni etichettate con @@. Inoltre rispondere ai seguenti quesiti:

- dire se la dichiarazione di *t_coda* rappresenta un ADT,
- in caso di risposta affermativa, spiegare i vantaggi degli ADT a partire dalla dichiarazione,
- in caso di risposta negativa, implementare in C++ e Java il tipo di dato astratto equivalente.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
      int c[20];
      int n;
} t_coda;

t_coda coda;

void init(t_coda *coda) {
     coda->n = 0;                  /* @@@ */
}
void add(t_coda *coda, int x) {
   if (coda->n < 20)
     coda->c[(coda->n)++] = x;     /* @@ */
}
main() {
  init(&coda);
  coda->n=1;
  coda->c[0]=2;
  add(&coda,3);
  add(&coda,4);
}
```

Soluzione

```
/* @@@ */      set D[D[0]+3]+20, 0

/* @@  */      set D[D[0]+3]+D[D[D[0]+3]+20], D[D[0]+4]
               set D[D[0]+3]+20, D[D[D[0]+3]+20] + 1
```

t_coda non rappresenta un tipo dato astratto, perché, come evidenziato nel *main* la sua rappresentazione può essere manipolata direttamente senza ricorrere alle operazioni *init* e *add*.

```
public class TCoda { /* JAVA */
   int c[];
   int n;
   public void TCoda() {
                     c = new int[20];
                     n = 0;
   }
   public void add(int x) {
                     c[n++]=x;
   }
}

public class TCoda { /* C++ */
   int *c;
   int n;
   public void TCoda() { /* Costruttore */
                     c = static_cast<int*> new int[20];
                     n = 0;
   }
   public void add(int x) {
                     c[n++]=x;
   }
}

public class TCoda { /* C++ */
   int c[20];
   int n;
   public void TCoda() {  /* Costruttore */
                     n = 0;
   }
   public void add(int x) {
                     c[n++]=x;
   }
}
```

Informatica 3 **Prima prova in itinere 6 Maggio 2005**

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>
}
```

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

	0:	CURRENT	16
	1:	FREE	20
global	2:	x	2
	3:	y	1
Main()	4:	RP	##
	5:	DL	##
	6:	SL	2
	7:	RV	
Power()	8:	RP	##
	9:	DL	4
	10:	SL	2
	11:	RV	
CalcPower()	12:	RP	##
	13:	DL	8
	14:	SL	8
	15:	RV	
CalcPower()	16:	RP	##
	17:	DL	12
	18:	SL	8
	19:	RV	2

Informatica 3 **Prima prova in itinere 6 Maggio 2005 Esercizio 3.**

Si definiscano in Java due tipi di thread:

- Il tipo "Conta", che stampa interi da 1 a 10, aspettando 0.5 s ad ogni passo e, prima di terminare, segnala la fine del conteggio.
- Il tipo "Attende", che alla creazione si mette in attesa di un segnale da un thread, fornito alla creazione dell'oggetto, poi termina.

Si definisca poi un programma che faccia partire due thread C e A, rispettivamente di tipo "Conta" e di tipo "Attende", in modo che A si metta in attesa di C.

Soluzione

Per semplicità si sono omessi i blocchi try-catch

```
class Conta extends Thread {
            public void run() {
                        for (int i = 1; i <= 10; i++) {
                           System.out.println(i);
                           Thread.sleep(500);
                        }
                        synchronized (this) {
                           notify();
                        }
            }
}

class Attende extends Thread {
            Conta conta;
            public Attende(Conta conta) {
                  this.conta = conta;
            }
            public void run() {
                   System.out.println("Attende inizia e poi aspetta");
                   synchronized (conta) {
                         conta.wait();
                   }
                   System.out.println("Attende termina");
            }
}

class Esame {
            public static void main(String args[]) {
                  Object o = new Object();
                  Conta conta = new Conta();
                  Attende attende = new Attende(conta);
                  conta.start();
                  attende.start();
            }
}
```

Informatica 3 **22 luglio 2003**

Esercizio 1

Rispondere alle seguenti domande:

a) In che cosa consiste l'operazione di *garbage collection* che il supporto *run-time* di *Java* effettua? Perché è utile che l'operazione di *garbage collection* venga effettuata automaticamente dall'implementazione del linguaggio?

[R. Il *garbage collector* di *Java* si occupa di eliminare dallo *heap* le istanze degli oggetti o di altre strutture dati che non sono più utili al programma in corso di esecuzione.
Se si affidano le operazioni di *garbage collection* all'utente ci sono i seguenti rischi:

- l'utente rischia di dimenticare di deallocare oggetti dallo *heap*, come conseguenza con lo scorrere del tempo lo *heap* si satura.
- l'utente rischia di deallocare oggetti ancora in uso, causando dei *dangling reference*.

]

b) La possibilità che una variabile (**p**) abbia come valore l'indirizzo di un'altra variabile (**x**) può generare un *"dangling reference"*. Perché?

[R. C'è il rischio che attraverso **p** si possa cercare di accedere a **x** (per essere più precisi all'area di memoria che contiene l'*r-value* di **x**) quando **x** non esiste più.
I casi in cui la variabile **x** può non esistere più sono i seguenti:

- **x** era stata originalmente allocata sullo *heap* e poi è stata stata deallocata.
- **x** è stata originalmente allocata sullo stack e uscendo dallo scope di **x** la porzione corrispondente dello stack è stata deallocata.

]

c) Nel caso precedente, sapresti individuare qualche controllo al tempo di compilazione che coinvolga lo scope delle variabili e che consenta di individuare potenziali sorgenti di *dangling reference* durante l'esecuzione?

[R. Si può controllare che lo scope di **p** sia strettamente incluso nello scope di **x**. In tal modo **p** viene deallocata prima di **x**. Tuttavia (in presenza di *routines*) questo controllo può essere compiuto integralmente solo a *runtime*.
Altri linguaggi inoltre ricorrono ai seguenti espedienti per cercare di minimizzare la presenza di *dangling references* (tutti controllabili a *compile-time*):

- Tipizzazione dei puntatori.
- Divieto di applicazione delle operazioni aritmetiche ai puntatori.

]

Esercizio 2: Si consideri un programma che opera sulle seguenti variabili:

```
int z, i;
int vect[10]
```

e che esegue il codice seguente contenente due chiamate alla routine *p*:

```
   z = 0;
   p(z);
3: write(z);
   for (i=0; i<10; i++) vect[i]=0;
   i=1;
   p(i);
4: write(i); write(vect[i]);
```

Ipotizziamo che la variabile *z* sia globale, visibile sia al programma che alla routine *p*. La routine *p* è così definita:

```
void p (int y) {
   1: write(z); write(y);
      z++;
      y := y + z;
   2: write(z); write(y);
};
```

Si scrivano i risultati che vengono visualizzati dall'esecuzione del programma considerando le diverse possibilità di passaggio dei parametri (a) per valore, (b) per indirizzo, (c) per valore-risultato.

[R. ***Passaggio per valore:***

1 z: 0, y: 0
2 z: 1, y: 1
3 z: 1
1 z: 1, y: 1
2 z: 2, y: 3
4 i: 1, vect[1]: 0

Passaggio per indirizzo:

1 z: 0, y : 0
2 z: 2, y : 2
3 z: 2
1 z: 2, y: 1
2 z: 3, y: 4
4 i: 4, vect[4]: 0

Passaggio per valore risultato

1 z: 0, y: 0
2 z :1, y: 1
3 z :1
1 z :1, y: 1
2 z: 2, y: 3
4 i: 3, vect[3]: 0

]

Informatica 3 **27 Aprile 2005**

Esercizo 2

Siano x, y due oggetti Java appartenenti alla classe ListadiSegmenti

```
class ListadiSegmenti{
   Segmento dato;
   ListadiSegmenti next;
……
}
```

dove Segmento è la classe definita nell'Esercizio1.

E' noto che in Java l'effetto dell'istruzione x = y è diverso dall'effetto della medesima istruzione in C. Come si potrebbe in Java simulare l'effetto di un tale assegnamento in modo che esso coincida con quello del C? E in C++?

Soluzione

In generale un assegnamento x=y in C comporta che il valore di y venga copiato nella variabile x, mentre lo stesso assegnamento in Java comporta che x vada a condividere l'area di memoria di y. Per ottenere lo stesso effetto dell'assegnamento in C, quindi, si definisce la classe di cui x e y sono istanze come implementazione dell'interfaccia Cloneable, implementando il metodo Clone.

Si noti che in questo specifico caso si considerano come x e y liste di segmenti, cioè un puntatore a un oggetto di tipo lista di Segmenti. L'assegnamento di puntatori in C e in C++ comporta alla condivisione della stessa aria di memoria, cioè si ha la stessa semantica dell'assegnamento di Java.

Qualora si volesse ottenere l'effetto della semantica dell'assegnamento di variabili in C, in questo caso, anche in C e C++ bisognerebbe creare una copia dell'oggetto y e poi assegnare a x tale copia.

Esercizo 3

Si consideri il seguente programma scritto in un ipotetico linguaggio con sintassi ispirata al C.

```
1  int *a[100];
2  int i;
3  void f(int *x)
4  {
5    int y,z;
6    x=a[0];
7    i++;
8    *a[*x]=*x+1;
9    y=*x;
10   z= *a[*x];
11  }
12 void main ()
13 {
14   for(i=0;i<100;i++)
15   {    new (a[i]);
          *a[i]=i+10;   }
16   i=30;
17   f(a[i]);
18 };
```

Assumendo che i parametri vengano passati per nome, simulare la chiamata di funzione (riga 17) mostrando il valore assunto dalle variabili locali y e z alle righe 9 e 10 rispettivamente.

Soluzione

Oss:

Le variabili *a[]* e *i* sono globali.

i valori numerici dereferenziati dai puntatori nel vettore a valgono nell'ordine:

**a[] = {10, 11, 12, …,109};* quindi ad esempio: **a[0] = 10; *a[50] = 60;*

	Codice Eseguito	***Valori Calcolati***
Riga 16	i = 30	
Riga 06	a[i] = a[0]	a[30]: a[0]
Riga 07	i=31	
Riga 08	*a[*a[i]] = *a[i] + 1	*a[*a[31]]: *a[31]+1 equivale *a[41]: *a[31]+1 equivale *a[41]:42
Riga 09	y = *a[i]	y : *a[31] equivale **y: 41**
Riga 10	z = *a[*a[i]]	z : *a[*a[31]] equivale z: *a[41] equivale **z: 42**