Corso di laurea in Comunicazione digitale Fondamenti di architettura e programmazione

AA 2008/09 Prof. M.A. Alberti

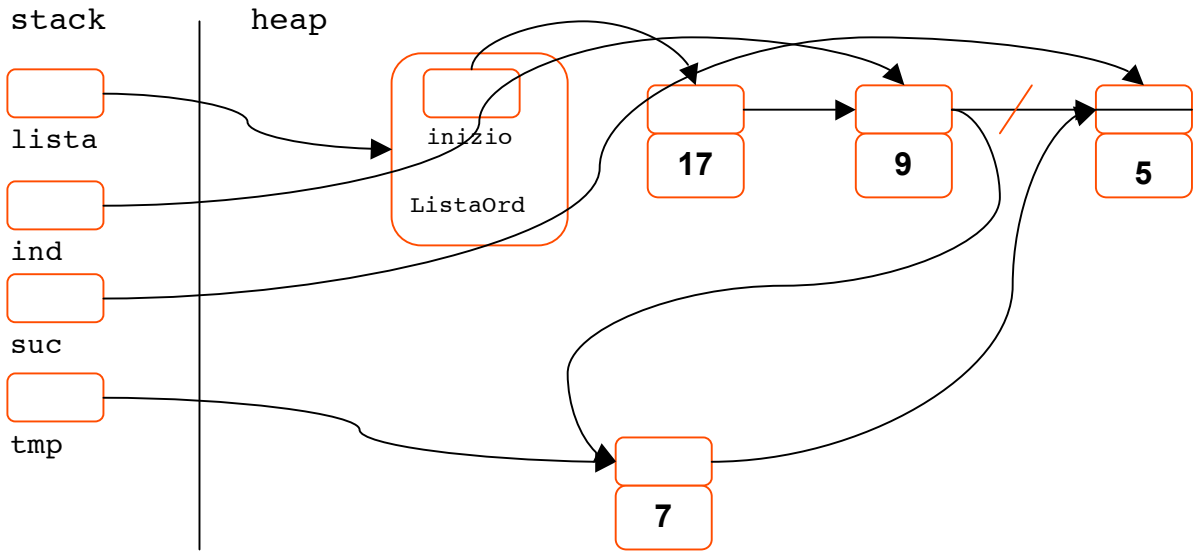
I Appello - 22 gennaio 2009

Cognome **Nome**

Matricola

---

1 (8 punti)



In figura, è schematizzato lo stato della memoria, ad un dato istante, durante il processo di costruzione di una lista ordinata decrescente di numeri, che usa la classe `ListaOrd` con la classe interna `NodoLista`:

```
public class ListaOrd {
    private NodoLista inizio;

    private static class NodoLista {
       Comparable dato;
       NodoLista pros;
    }    ……
}
```

La variabile `lista` sullo stack e' il riferimento alla lista di tipo `ListaOrd` dopo un certo numero d'inserimenti. Si noti che la figura mostra l'usuale semplificazione che mostra nel campo `dato` i valori interi, mentre in realtà dovrebbero esservi i riferimenti ad oggetti: ad esempio di tipo `Integer` o di qualunque classe che implementi l'interfaccia `Comparable`.

1. Definite e inizializzate le due variabili **`ind`** e **`suc`** usate come indice di scorrimento della lista - **`ind`** punta a un generico nodo e `suc` al suo successore.

```
NodoLista ind = null;
NodoLista succ = inizio;
```

2. Scrivete le istruzioni necessarie per far avanzare parallelamente gli indici sulla lista:

```
ind = suc;
suc = suc.pros;
```

3. Scrivete la condizione di uscita da un ciclo che serve a far avanzare i due puntatori **ind** e **suc** in modo da prepararsi ad inserire nella lista il nodo **tmp** nella posizione corretta. Si deve riflettere su come far fermare il ciclo in modo da conservare il riferimento al nodo che contiene un dato, inferiore a quello da inserire, e il riferimento al nodo dopo il quale inserire. Le variabili **ind** e **suc** servono allo scopo e consentono di individuare la posizione esatta in cui inserire il nodo **tmp**. Le istruzioni al punto 2. rappresentano il corpo del ciclo.

```
(suc != null && suc.dato.compareTo(tmp.dato) < 0)
```

4. Scrivete le istruzioni necessarie per inserire il nodo **tmp**, creando i legami corretti con i nodi pre-esistenti, una volta che gli indici di scorrimento sulla lista sono giunti alla posizione corretta:

```
tmp.pros = suc;
if (ind == null )
    inizio = tmp;   // caso inserimento nodo a inizio lista
else
    ind.pros = tmp;
```

5. Disegnate sulla figura la struttura che si ottiene: i legami che si formano e quelli che si perdono, dopo aver inserito il nodo **tmp** e lo stato delle variabili al termine del ciclo.

---

2 (5 punti)

Dato il metodo **`int f(int, int)`** seguente, disegnare la traccia dell'andamento dello stack delle chiamate dei metodi durante l'esecuzione di `f(6, 3)` (si ricodi che, nel record d'attivazione del metodo i campi `ind` e `val` indicano rispettivamente **l'indirizzo e il valore di rientro di un metodo chiamato**. Inoltre si noti che nella figura e' tracciata solo la parte di stack relativa alla chiamata del metodo `f` dall'inidirizzo `K` di qualche altro metodo, ad esempio `main`. La freccia indica la direzione di crescita dello stack

```
int f(int m , int n)    {
  if (m==0&&n==0)
    return 1;
  else if (n==0)
    return 1 + f(m-1, n);      // A      A, B, C sono indirizzi
  else if (m==0)
    return 1 + f(m, n-1);      // B
  else return 2 + f(m-1, n-1) // C
}
```

	n 3	2	1	0	0	0	0	
	m 6	5	4	3	2	1	0	
10	**val** ~~?~~ 8	~~?~~ 6	~~?~~ 4	~~?~~ 3	~~?~~ 2	~~?~~ 1	?	
K	**ind** C	C	C	A	A	A	?	

Record 0 Record 1 →

Quale valore riporta la chiamata di `f(6, 3)`? 10

Quante chiamate ricorsive sono state effettuate, includendo nel conto la I chiamata? 7

Quale sarebbe invece il valore di `f(3, 6)`? 10

3 (2 punti)

Definire una struttura dati per archiviare le temperature medie dei diversi mesi degli anni tra il 1980 e il il 1999. La struttura si chiama **temperature**; durante la lettura dei dati archiviati si useranno gli indici numerici corrispondenti ai mesi (gennaio 1, febbraio 2 etc).

```
int [][] temperature = new int[20] [12];
```

4 (4 punti)

In una classe **Negozio** si definisce un campo **venduti** che contiene il numero di oggetti venduti in ogni giorno e mese dell'anno. Gli oggetti venduti sono classificati in due categorie, identificate come **cancelleria** e **regali**.

1. Si definisca la classe **Oggetti**, che costituisce il tipo dell'elemento di base della tabella **venduti**:

```
private class Oggetti {
    int cancelleria;
    int regali;
}
```

2. Si definisca la tabella **venduti** della classe **Negozio**

```
Oggetti [] [] venduti;
```

3. Si definisca il metodo void **init()** che inizializza opportunamente il campo **venduti**, usando le costanti ***GIORNI*** e ***MESE*** dal significato ovvio.

```
public void init() {
    venduti = new Oggetti [MESI][GIORNI];
    for (int m=0; m < MESI; m++)
        for (int g=0; g < GIORNI; g++)
            venduti[m][g] = new Oggetti();
}
```

4. Istanziate un oggetto di classe **Negozio**, e opportunamente richiamate il metodo **init()**

```
Negozio mioNegozio = new Negozio();
mioNegozio.init();
```

5. Definite ora un ciclo e le eventuali variabili necessarie per calcolare il valore medio degli oggetti di categoria **cancelleria** in un dato mese **mese**.

```
float totale = 0.0;
for (int g=0; g<GIORNI; g++) {
    totale += mioNegozio.venduti[mese-1][g].cancelleria;
}
float media = totale/GIORNI;
```

5 (5 punti)

1. Valutare l'espressione aritmetica: (5 + 3) * (4 / 2) + 4 * 3 = **28**

2. Scrivete l'espressione in modo postfisso: **5 3 + 4 2 / * 4 3 * +**

3. Valutate l'espressione mediante un metodo **postfissa()** che usa uno stack per la computazione. Disegnate l'andamento dello stack durante tale computazione nella griglia proposta, in cui ogni colonna rappresenta una fotografia dello stack a ciascun

passo della computazione; il fondo dello stack sia in basso e la cima in alto. La riga in basso nella griglia serve per evidenziare il token che viene prelevato dalla riga di input, l'espressione aritmetica in forma postscritta e in corrispondenza del quale si devono effettuare azioni diverse sullo stack.

				2				3		
	3		4	4	2		4	4	12	
5	5	8	8	8	8	16	16	16	16	28

5	3	+	4	2	/	*	4	3	*	+

---

6 (5 punti)

Scrivere un ciclo di acquisizione di input per stringhe di caratteri (nomi ad esempio), che dovranno popolare un array. Si supponga che l'input venga inserito su una sola riga in risposta al prompt "**input?** ". Si definisca la struttura che immagazzina le stringhe (array) e si usi la classe **StringTokenizer**.

```
String riga = in.readLine("input? ");
StringTokenizer st = new StringTokenizer(riga);
String [] struttura = new String[st.countTokens()];
for (int i=0; st.hasMoreTokens(); i++)
     struttura[i]=st.nextToken();
```

---

7 (4 punti)

Considerate le classi `A` e `B`, estesnsione della classe `A`:

```
class A {
   public int codice;
   protected String tipo;
   A(int c, String t) {
      codice = c;
      tipo = t;
   }
}
```

```
class B extends A {
   private int codice;
   B(int c) {
   super(c-1, "esteso");
   codice = c;
   }
}
```

1. Definite il costruttore della classe `B` tenedo conto delle seguenti indicazioni:
   a. ogni oggetto di classe `B` è caratterizzato da un numero di codice;
   b. il campo `codice` del suo antenato ha valore pari al valore – 1 del codice dell'oggetto
   c. il campo `tipo` è posto al valore predefinito "`estensione`".
2. Osservate i due membri `codice` definiti nelle due classi. C'è conflitto tra il campo `codice` ereditato e quello definito? SI **NO**
3. Si dice che il membro `codice` di classe `B` **oscura** quello di classe `A`
4. Il metodo `int getCodice() {return super.codice + codice;}` è corretto per la classe `B`? **SI** NO
5. Istanziati gli oggetti `A a = new A(21, "non esteso"); B b = new B(222);` valutate l'espressione `b.getCodice()` **443**