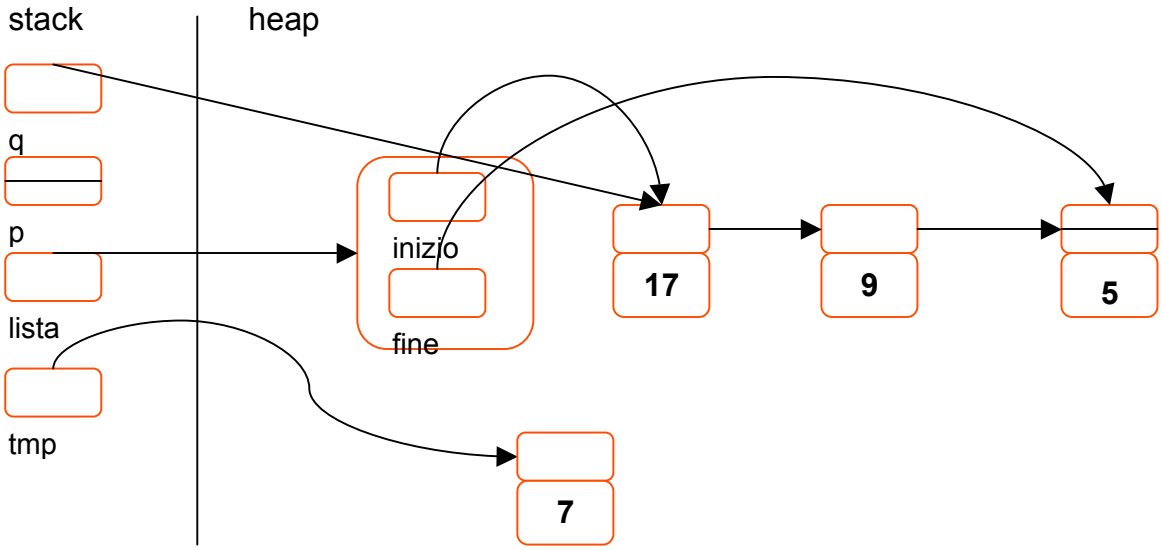
Pre-appello 2006/07 / VI Appello 2005/06 - 29 gennaio 2007

Cognome **Nome**

Matricola

1



Osservate la situazione dello stack delle chiamate dei metodi e dello heap schematizzato dalla figura, in cui si fa riferimento alla classe `ListaOrdinata` con la classe interna `NodoLista`:

```
public class ListaOrdinata {
    private NodoLista inizio, fine;
    private static class NodoLista {
       Comparable dato;
       NodoLista pros;
    }    ……
}
```

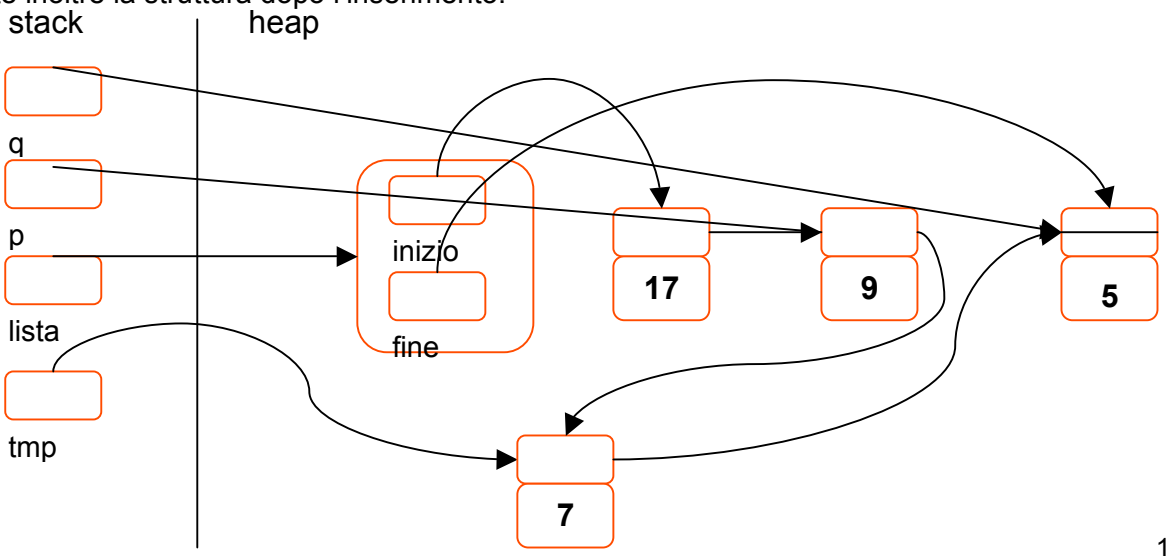
La variabile `lista` sullo stack e' il riferimento a una lista dopo un certo numero di inserimenti. Definite voi le due variabili `p` e `q` usate come indice di scorrimento della lista:

```
NodoLista p = null; NodoLista q = lista.inizio;
```

e le due istruzioni per far avanzare gli indici sulla lista:

```
p = q;     q = q.pros;
```

Disegnate inoltre la struttura dopo l'inserimento:



2

Dato il metodo `int f(int, int, int)` seguente, tracciare lo stato dello stack delle chiamate dei metodi durante l'esecuzione di `f(6, 0, 1)` (si ricodi che `rit` indica l'indirizzo e `val` il valore di rientro)

```
int f(int n, int f0, int f1)  {
  if (n==0)
    return f0;
  else if (n==1)
    return f1;
  else
    return (f(n-1, f1, f0+f1)); // indirizzo di rientro A
}
```

	Record 1	Record 2	Record 3	Record 4	Record 5	Record 6
n	6	5	4	3	2	1
f0	0	1	1	2	3	5
f1	1	1	2	3	5	8
val	8	8	8	8	8	
rit	A	A	A	A	A	

Quale valore riporta la chiamata di `f(6, 0, 1)`? **8**
Quante chiamate ricorsive vengono effettuate? **6 chiamate del metodo f**

3

Scrivere un metodo per computare **iterativamente** la funzione di Fibonacci `f(n)`, dove `n` e' un numero intero non negativo, definita come segue:

$$f(n) = \begin{cases} 0 & \text{per } n = 0 \\ 1 & \text{per } n = 1 \\ f(n-1) + f(n-2)) & \text{per } n > 1 \end{cases}$$

```
public int f_it (int n)       {
   int prec=1, precprec=0, tmp=prec;
   if (n==0 || n==1) return n;
   else     {
      for (int i=2; i<=n; i++)  {
         tmp += precprec;
         precprec = prec;
         prec = tmp;
      }
      return tmp;
   }
}
```

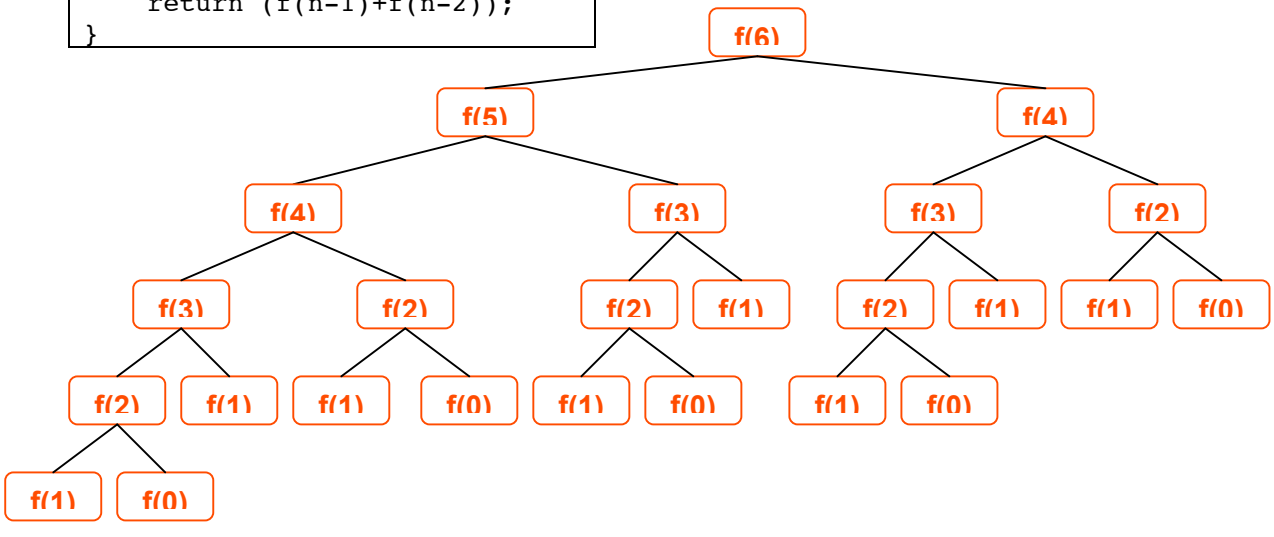
Calcolare `f(6)`: **8**

4

funzione ricorsiva numeri Fibonacci

```
int f(int n) {
  if (n==0 || n==1)
    return n;
  else
    return (f(n-1)+f(n-2));
}
```

Numero delle chiamate ricorsive per calcolare f(6): **25**
Disegnare l'albero delle chiamate:



5

Esiste una relazione tra la funzione ricorsiva dell'esercizio **4** e quella dell'esercizio **2**?
I due metodi calcolano la stessa funzione.
Cosa si può concludere? **Il metodo dell'esercizio 2 è più efficiente. Infatti per calcolare il numero di Fibonacci `f(6)` il metodo dell'esercizio 2 effettua 6 chiamate ricorsive, mentre il metodo dell'esercizio 4 ne effettua 25.**

6

Data l'espressione aritmetica `(2 + 6) / ( 2 * 2) + 5 * 3` trascriverla in forma postfissa:
`2 6 + 2 2 * / 5 3 * +`
Quindi valutare l'espressione facendo uso di una struttura a stack.
Disegnare l'andamento dello stack durante la valutazione dell'espressione nella griglia proposta (ogni colonna rappresenta una fotografia dello stack ad un passo della computazione, il fondo dello stack sia in basso)

				2				3		
	6		2	2	4		5	5	15	
2	2	8	8	8	8	2	2	2	2	17
token letto	2	6	+	2	2	*	/	5	3	*	+

7

Dato il seguente spezzone di codice:

```
public class Giochi {
   private int pedine;
   public int muovi() {
      return pedine--;
   }
}
```

```
public class Scacchi extends Giochi {
   int scacchiera;

}
```

Dire se le seguenti istruzioni sono lecite o no:

a. `Giochi g = new Scacchi();` **SI** NO
b. `Giochi f = new Giochi(); int p = f.muovi();` **SI** NO
c. `Scacchi s; int p = s.muovi();` SI **NO**
d. `g.scacchiera = 8;` SI **NO**

In caso di risposta NO, correggete, se possible, le istruzioni per renderle lecite:

a.

b.

c. va istanziato `s`, **`s = new Scacchi();`** quindi `s.muovi()` è lecito

d. `g` è definita di tipo `Giochi`, quindi occorre effettuare un cast esplicito pere accedere al campo `scacchiera`: **`((Scacchi) g).scacchiera = 8;`**

8

In una implementazione dinamica della struttura dati stack, definita come segue

```
public class Stack  {
   private NodoStack cima;
   private class NodoStack  {
      Object dato;
      NodoStack pros;
   }

   public Stack()  {
      cima = null;
   }
```

```
public void push(Object o)  {
      NodoStack t = new NodoStack();
      t.dato = o;
      t.pros = cima;
      cima = t;
}
```

Implementare la funzione `top()` che accede al valore del nodo in cima allo stack e lo riporta all'ambiente senza modificare la struttura e un'eccezione non controllata `EmptyStackException()`.

```
public class EmptyStackException extends RuntimeException {
}

public Object top() {
   if (cima == null)
      throw new EmptyStackException();
   else
      return cima.dato;
}
```

Si scriva una istruzione `try-catch` in cui si cattura l'eventuale eccezione lanciata dal metodo `top()` nel contesto di un programma in cui sia stata introdotta la variabile `pila`, onde poter recuperare e inserire almeno un nodo. Gli oggetti da inserire nella pila siano di classe `Integer`.

```
Stack pila = new Stack();
try {
   if (((Integer)pila.top()).intValue()==x)
      pila.pop();
} catch (EmptyStackException e) {
      pila.push(Integer(x));
}
```

In questo modo si guarda alla cima della pila per vedere se il primo nodo della pila contiene il dato informativo `x`; in questo caso il nodo viene eliminato dalla pila; altrimenti recupera il flusso d'esecuzione del programma inserendo un nodo contenete il dato informativo `x`

9

Avendo definito `class Settimana implements java.util.Iterator { };` dire se sono lecite le seguenti istruzioni:

a. `Settimana s = new Settimana();`

   `while(s.hasNext()) <computa s.next()>` **SI** NO

b. `Iterator i = new Settimana();` **SI** NO

c. Qual'è il concetto della programmazione a oggetti per cui diversi oggetti che hanno una interfaccia comune rispondono in modo diverso quando un metodo di quell'interfaccia è invocato: **polimorfismo**

d. Casting è appropriato quando si riceve un riferimento a una classe antenato e si sappia che l'oggetto è una particolare sotto-classe e si voglia usare l'oggetto nella sua funzionalità completa. **SI** NO

   Si veda infatti l'esercizio precedente 7 al punto d.