Fondamenti di Informatica A
Laboratorio 2
Funzioni e ricorsione

Danilo Pianini
danilo.pianini@unibo.it
Mirko Viroli
mirko.viroli@unibo.it
Andrea Santi
a.santi@unibo.it

Alma Mater Studiorum—Università di Bologna

20 marzo 2013

Operazioni preliminari

- Accendere il computer
- Effettuare il log-in con le proprie credenziali istituzionali
  - Normalmente sono simili a:
    `nome.cognome@studio.unibo.it`
- Aprire Google Chrome, o altro browser
- Collegarsi al sito `http://bit.ly/finfa2013-e2`
- Scaricare l'esercitazione odierna

Verifica del funzionamento del Framework F-F

- Sul desktop, troverete la cartella “framework”
- Fate doppio click sulla cartella
- Si aprirà una nuova finestra, all’interno della quale troverete tre software
- Fare doppio click sull’icona del FrameworkFF
- Inserire nel campo “Espressioni da valutare” il testo $1 + 2$
- Premere il tasto “Valutazione”
- Verificare che il risultato sia “3”
- In caso di risultato diverso dalle attese, chiamare subito il docente o il tutor

Come comportarsi di fronte agli errori

- Errare humanum est
  - Ed è anche molto meglio sbagliare in laboratorio che all'esame!
- È importante, di fronte agli errori, non scoraggiarsi, leggere **attentamente** il messaggio di errore, **capirlo** e studiare una contromisura.
- Java segnala gli errori in maniera piuttosto precisa...
- ...il framework, che ne usa una sottoparte molto ristretta, molto meno!
- In caso di errore, il framework vi mostrerà un messaggio che spiega cosa (il compilatore pensa che) sia successo, seguito dal codice Java reale che è stato invocato.

Come esercitarsi con il Framework

- Questo framework è una versione potenziata di FV
- Oltre a consentire di valutare espressioni, consente di definire funzioni
- Le funzioni definite potranno poi essere utilizzate all'interno dell'espressione da valutare

Struttura di una funzione

Partiamo da un semplice esempio:

```
int s(int i){ return i+1;}
```

- `int s(int i)` è detta "signature"
  - `La signature dà tutte le informazioni necessarie all'identificazione di una funzione:`
    - nome della funzione
    - numero e tipo dei parametri di ingresso
    - tipo del parametro di ritorno
  - Conoscendo la signature di una funzione, è possibile utilizzarla!
- `return i+1` è detto "corpo"
  - Il corpo contiene l'effettiva implementazione
  - Nel corpo si trovano le istruzioni che vengono effettivamente svolte quando si invoca la funzione

Invocazione delle funzioni

- Si inserisca la funzione `int s(int i)` della slide precedente nel box in alto
- Senza inserire nulla nel secondo box, si valuti. Perché questo risultato?
- Si provi ad intuire, e quindi si verifichi, il risultato della valutazione delle seguenti:
  - `s(0)`
  - `s(10)`
  - `s(2147483647)` — Perché?
  - `s(0)+s(0)`
  - `s(s(0))`

Correttezza semantica, singola funzione

Una funzione è semanticamente corretta se:

- I parametri sono usati in maniera compatibile con il loro tipo
- Il tipo dell'espressione corpo è uguale, oppure convertibile per coercizione, al tipo di ritorno della funzione
- Si provi ad intuire, e quindi si verifichi, quali delle seguenti funzioni sono semanticamente corrette. Ove presenti, si leggano e si capiscano i messaggi di errore.
  - `int f(int x, int y){ return x+y;}`
  - `int f(double x, int y){ return x+y;}`
  - `float f(double x, int y){ return x+y;}`
  - `int f(int x, int y){ return x+"y";}`
  - `double f(double x, long y){ return x+y;}`
  - `int f(boolean b, boolean b2){return (b&b2)?0:1f;}`
  - `int f(boolean b, boolean b2){return (b&b2)?-1:1;}`
  - `boolean f(int x, int y){ return x==y;}`

Correttezza dell'invocazione

Una funzione è correttamente invocata se:

- I parametri passati sono uguali (o convertibili per coercizione) al tipo dichiarato
- Il tipo di ritorno è uguale (o convertibile per coercizione) al tipo da associare.
- Si considerino le seguenti funzioni corrette:
  - `boolean f3(int x,int y){ return x==y;}`
  - `boolean f4(double x,double y){ return x!=y;}`
- Per ciascuna delle invocazioni seguenti, predirre il risultato, quindi verificarlo nel framework:
  - `f3(1,2L)`
  - `f4(1,2L)`
  - `f3(1,1/2)`
  - `f3(1,1/2f)`
  - `f3(1,1/2)+3.4`
  - `""+f4(1,1/2f)`

Correttezza dell'invocazione

Dato il seguente gruppo di funzioni:

- `int f(int x, int y){ return x;}`
- `long f(int x, long y){ return y;}`
- `boolean f(double x, double y){ return x==y;}`
- `boolean f(float x, float y){ return x!=y;}`

Si preveda (e si verifichi) il risultato delle valutazioni seguenti:

- `f(0,1)`
- `f(0.0,1)`
- `f(0.0,1f)`
- `f(1/3,2l)`
- `f(0x01,0x10L)`
- `f(1/2,0.5)`
- `f(1/2,0.5f)`
- `f(2l,2l)`
- `f(010,07)`

Ricorsione

Le funzioni possono, al loro interno, chiamare altre funzioni o sé stesse. Si analizzi la funzione seguente:

- `int f(int n){ return n==0?  1 :  n*f(n-1); }`
- Che tipo di comportamento realizza?
- Per quali input funziona come dovrebbe?
- Come si potrebbe migliorare, per far sì che funzioni correttamente per un insieme di input più ampio?
- La funzione effettua una ricorsione tail o non tail? Perché?
- Si costruisca l'albero delle invocazioni relativo alla chiamata `f(3)`.

Ricorsione

Si analizzi la funzione `f(int, int)`:

```
int f(int x, int y) {
   return f(x, y, x, y);
}

int f(int x, int y, int z, int w) {
   return z==w ? z :
      z>w ? f(x, y, z, w+y) :
         f(x, y, z+x, w);
}
```

- Si costruisca l'albero delle invocazioni relativo alla chiamata `f(3,7)`.
- Che tipo di comportamento realizza? Si cerchi di intuirlo, in caso di difficoltà si usi il framework per ottenere dei suggerimenti.
- La funzione effettua una ricorsione tail o non tail? Perché?

Funzioni su array

Si analizzi la funzione `sum(int[])`:

```
int sum(int[] array){
   return sum(array, 0);
}

int sum(int[] array, int pos){
   return (pos==array.length) ? 0 :
      array[pos] + sum(array, pos + 1);
}
```

- Quale funziona realizza? Una volta intuito, si utilizzi il framework per verificare.
- Si costruisca l'albero delle invocazioni relativo alla chiamata `sum(new int[]{5,5,6,4})`.

Funzioni di test

Si analizzi la funzione `test_sum()`:

```
boolean test_sum(){
    return sum(new int[]{10,20,30})==60 &&
        sum(new int[]{10,20,30,40,50})==150 &&
        sum(new int[]{10})==10 &&
        sum(new int[]{})==0;
}
```

- Cosa realizza?

Modifica di funzioni

- Si modifichi la funzione `sum(int[])`, per fare in modo che, qualora l'array passato in ingresso abbia lunghezza superiore a 4, il risultato ritornato sia zero.
- Verificare tramite il framework il corretto funzionamento della nuova implementazione.
- Si modifichi `test_sum()`, in maniera tale che venga provato il nuovo funzionamento di `sum(int[])`.
- Si modifichi `test_sum()`, aggiungendo una verifica che provi il funzionamento di `sum(int[])` per array contenenti numeri negativi.

Si mostri al docente l'esercizio completato per ricevere commenti.

Modifica di funzioni

Si consideri `sum(int[])`, nella versione non modificata.

- Cosa succede se si sostituisce a "`pos + 1`" "`pos + 2`"?
- Si costruisca l'albero delle invocazioni relativo alla chiamata `sum(new int[]{10,2,30,4})`
- Quale nuova funzione si realizza?
- Si costruisca l'albero delle invocazioni relativo alla chiamata `sum(new int[]{10,2,30})`, e lo si verifichi nel framework.
- Perché si ha un errore?
- Si sostituisca == con >=. Si cerchi di capire se questa modifica risolve il problema, e perché. Si verifichi nel framework.

Costruzione di funzioni a partire da funzioni esistenti

Si consideri `sum(int[])`, nella versione non modificata.

- A partire dalla soluzione precedente, si costruisca la funzione `sum_odd(int[])`, che somma solo gli elementi in posizione dispari.
- Si scriva una funzione `test_sum_odd()`, che verifichi il funzionamento della precedente. In particolare, testare che funzioni con array di lunghezza zero, pari e dispari.
- Si costruisca la funzione `sum_abs(int[])` che somma i valori assouti degli elementi dell'array.
- Si costruisca la funzione `test_sum_abs()` che verifica il corretto funzionamento di `sum_abs(int[])` per array di varia lunghezza (fra cui zero), contenenti numeri positivi e negativi

Si mostri al docente l'esercizio completato per ricevere commenti.

Costruzione di funzioni a partire da funzioni esistenti: riconoscitori EBNF

- `ric123(int[])` che riconosce la grammatica $1\{2|3\}$. Come esempio, si consideri il riconoscitore per $1\{2\}$ visto a lezione. Si scriva un test per verificarne il comportamento.
- Considerando il riconoscitore per la grammatica $\{2\}1$ sotto riportato, si realizzi `ric1234(int[])` che riconosce la grammatica $\{1|2\}34$. Si scriva un test.

```
boolean ric21(int[] a) {
   return ric2(a, 0);
}
boolean ric2(int[] a, int pos) {
   return pos == a.length ? false :
      a[pos] != 2 ? ric1(a, pos) :
      ric2(a, pos + 1);
}
boolean ric1(int[] a, int pos) {
   return a[pos] == 1 && a.length == pos + 1;
}
```

Si mostri al docente l'esercizio completato per ricevere commenti.

Esercizi aggiuntivi

Da questo punto, vengono proposti esercizi aggiuntivi. Sono consigliati per coloro che hanno completato **tutti** i precedenti con successo e desiderano esercitarsi ulteriormente.

Correttezza semantica, gruppi di funzioni

Si noti che più funzioni possono avere lo stesso nome, a patto che la loro signature (senza considerare il tipo di ritorno) sia diversa. Si preveda quali delle seguenti coppie di funzioni possono essere definite contemporaneamente senza generare errori:

- `int f(int x, int y){ return x+y;}`
- `int f(long x, int y){ return x-y;}`

- `int f(int x, int y){ return x+y;}`
- `int f(int x, int y){ return x-y;}`

- `long f(int x, int y){ return x+y;}`
- `int f(int x, int y){ return x-y;}`

- `boolean f(double x, double y){ return x==y;}`
- `boolean f(int x, int y){ return x==y;}`

Ricorsione

Si analizzi la funzione `f(int, int)`:

```
int f(int x) { return x == 0 ? 0 : f2(x-1); }
int f2(int x) { return x == 1 ? 1 : f(x-1); }
```

- Si costruisca l'albero delle invocazioni relativo alla chiamata `f(6)`, si verifichi il risultato con il framework.
- Si costruisca l'albero delle invocazioni relativo alla chiamata `f(5)`, si verifichi il risultato con il framework.

Progettare nuove funzioni

Si progettino le seguenti funzioni, e per ognuna si scriva un test appropriato che ne verifichi il funzionamento.

- `int length(int[])`, che conta quanti elementi vi sono in un array.
- `int evens(int[])`, che conta quanti elementi pari vi sono in un array.
- `boolean arrayEquals(int[], int[])`, che ritorna true se i due array sono uguali (ossia hanno la stessa lunghezza, e presentano gli stessi elementi nello stesso ordine).

Si mostri al docente l'esercizio completato per ricevere commenti.

Progettare nuove funzioni: riconoscitori EBNF

Si progettino le seguenti funzioni, e per ognuna si scriva un test appropriato che ne verifichi il funzionamento.

- `ric111(int[])`, riconoscitore per la grammatica $1\{1\}1$.
- `ric12(int[])`, riconoscitore per la grammatica $\{1|2\}$.
- `ric123(int[])`, riconoscitore per la grammatica $1\{2\}3$.
- `ric1234(int[])`, riconoscitore per la grammatica $\{1|2\}\{3|4\}$.

Si mostri al docente l'esercizio completato per ricevere commenti.

Al termine dell'esercitazione

- Ricordarsi di effettuare il logout
  - Dal menu Start, selezionare "Disconnetti"
- Una volta disconnessi:
  - Se siete del primo turno, lasciate le macchine accese, di modo che i vostri colleghi del turno successivo siano agevolati
  - Se siete del secondo turno, spegnete le macchine