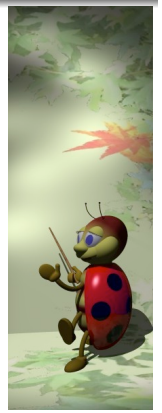
I Paradigmi del Giuoco delle Perle di Vetro

Claudio Mirolo

Dipartimento di Matematica e Informatica,
Università di Udine, via delle Scienze 206 – Udine
claudio.mirolo@uniud.it

Corso sulla Didattica dell'Informatica
Trieste, 3 Aprile 2012

Sommario



Il Giuoco è prerogativa dell'*Ordine di Castalia*?

- Qual è il ruolo della programmazione
  per capire la natura dell'informatica *oggi*?
- Programmare per sviluppare competenze professionali?
- Dibattito sulla centralità o meno dei *programming skill*
- Diffusione del modello *programming-first*
  per ragioni pratiche e storiche

Programmazione?

- Si può capire la geometria in mancanza
  di un'esperienza "corporea" dello spazio?
- Si può sviluppare il pensiero logico
  senza esercizio di uso del linguaggio?
  (p.es.: italiano)
- Si può capire la natura dell'informatica
  senza esercizio di programmazione?

Epistemologia procedurale

La rivoluzione informatica è una rivoluzione nel modo di pensare e di esprimere quello che si pensa. L'essenza di questo cambiamento è l'emergere [di un'] *epistemologia procedurale*

[. . .] i programmi devono essere scritti affinché li possano leggere le persone, e solo incidentalmente per farli eseguire dalle macchine. (*G.J. Sussman, 2004*)

. . . Ma allora programmare è molto di più che scrivere codice!

Computing Curricula 2001

Analisi della "Task Force" ACM/IEEE-CS:

- Non c'è evidenza di una strategia ideale
- Ogni approccio ha punti di forza e punti di debolezza
- La sperimentazione è incoraggiata:
  innovazione pedagogica come condizione
  per il successo in un campo in rapida evoluzione

Approccio "programming-first"

vantaggi	*svantaggi*
curricula costruiti a partire da corsi preesistenti	basi concettuali e teoriche differite e percepite come irrilevanti
raccomandato dai precedenti modelli	dettagli sintattici a spese di competenze algoritmiche
	esempi ipersemplificati: progetto? analisi? verifica?

Approccio "programming-first"

vantaggi	*svantaggi*
capacità di programmare essenziale	misconcezione: informatica = programmazione
programmare è gratificante per molti studenti	scoraggiamento vs. sopravalutazione capacità
competenze spendibili a breve termine ai fini dell'occupazione	sottovalutazione di altri strumenti per il problem-solving

Corsi introduttivi

impostazione	*annotazioni*
imperative-first	modello computazionale
functional-first	astrazione procedurale
object-first	astrazione sui dati
algorithms-first	linguaggio di progetto
hardware-first	bottom-up
breadth-first	approccio a spirale

Computer Science 2008

Evoluzioni recenti:

- Cruciale la tematica della *sicurezza*, anche nella programmazione
- Maggiore rilievo della *concorrenza*
- Pervasività del *net-centric computing*
- Altre competenze specifiche ai fini della professione. . .

Computer Science 2008

Aspetti importanti:

- Esperienza di diversi *paradigmi di programmazione*: in particolare ruolo della programmazione funzionale
- Ruolo formativo dello studio dei *compilatori*
- Progetti nell'ambito dell'*open source*
- Quali *ragionevoli* obiettivi di apprendimento?
- Proiezione nel futuro: *lifelong learning*

Paradigmi di programmazione

Diversi modi di "pensare" i programmi

- Linguaggi di programmazione *imperativa*
- Linguaggi di programmazione *funzionale*
- Linguaggi di programmazione *orientata agli oggetti*
- Linguaggi di programmazione *dichiarativa*

Linguaggi di programmazione imperativa

1945–55		Linguaggi macchina, linguaggi assemblatori
1953	FORTRAN	J.W. Backus: progetto del primo linguaggio di programmazione ad alto livello
1957		Primo compilatore FORTRAN completo
1958–68	Algol	F.L. Bauer, J.W. Backus, P. Naur, E.W. Dijkstra, C.A.R. Hoare e molti altri: eleganza e pulizia *"Un linguaggio così avanzato per il suo tempo da migliorare non solo quelli precedenti, ma anche quasi tutti i successivi."* (Hoare, 1973)
1959	COBOL	Linguaggio orientato al software gestionale
1969	Pascal	N. Wirth: linguaggio "Algol-like"
1969–73	C	D. Ritchie: legato al sistema operativo Unix

Esempio: Moltiplicazione del contadino russo

```
function mul( x, y: integer ) : integer;

  var  z: integer;

begin
  z := 0;
  while  y > 0  do
    begin
      if  odd( y )  then  z := z + x;
      x := 2 * x;  y := y div 2
    end;
  mul := z
end;
```


Possibili sviluppi: Algoritmo e rappresentazione

Quale interpretazione nel sistema binario?

x	*y*	*odd(y)*	*z + ...*
10100	1011	1	10100
101000	101	1	101000
1010000	10	0	
10100000	1	1	10100000
		0	
		mul(x,y) =	11011100

È esattamente quello che succede nell'hardware!

Programmazione imperativa

vantaggi	*svantaggi*
modello computazionale semplice	modelli relativi al problema e alla computazione inestricabili
linguaggi di progetto grafici (p.es. diagrammi di flusso)	linguaggio del programma distante da quello dell'analisi del problema
simulazioni con carta e penna facili	attenzione localizzata sui singoli passi della computazione
corrispondenza con l'architettura computer	sequenziazione predominante

Linguaggi di programmazione funzionale

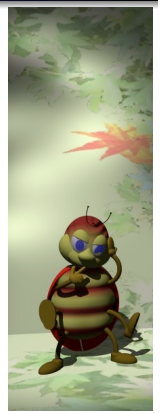
1958	LISP	J. McCarthy (progetto), S. Russel, T. Hart & M. Levin (realizzazione) — fra i linguaggi ad alto livello è preceduto solo dal FORTRAN
1962		Primo compilatore LISP completo
anni '70	Schemer	G.L. Steele Jr. & G.J. Sussman . . .
	Scheme	Sviluppi di LISP: progetto sperimentale di un linguaggio di nuova concezione
anni '70	ML	R. Milner: linguaggio funzionale con un sofisticato sistema di tipi (polimorfi)
1990	Haskell	Puramente funzionale con tipi polimorfi

Esempio: Moltiplicazione con la ricorsione

```
(define mul      ; valore: intero
  (lambda (x y)  ; x, y: interi non negativi
    (if (= y 0)
        0
        (let ((z (mul (* 2 x) (quotient y 2)))
              )
          (if (even? y)
              z
              (+ z x)
              ))
        )))
```

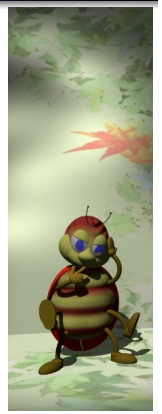

Peculiarità: Argomenti e valori procedurali!

```
(define punto-fisso?
  (lambda (f x)
    (= (f x) x)
    ))

(punto-fisso? sqrt 1)


(define comp
  (lambda (g f)
    (lambda (x) (g (f x)))
    ))

((comp sqrt abs) -9)
```



Peculiarità: Meta-livelli formalizzabili

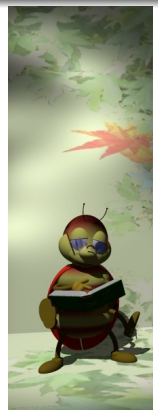
```
(define mcd
  (lambda (x y)
    (if (= y 0)
        x
        (mcd y (remainder x y))
        )))

(apply mcd (list 144 216))


(       ; sintassi o semantica?
 (eval
  '(lambda (x) (+ x 1))
  )
 5
 )
```



Peculiarità . . .

Consente di affrontare su una base di *concretezza* alcuni argomenti fondazionali, per esempio:

- Macchina *universale*
- Problema della *terminazione* (algoritmicamente irrisovibile in generale)



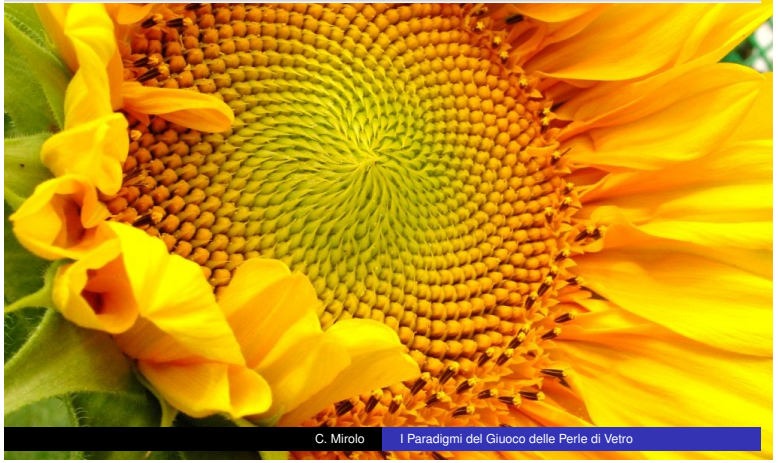
Programmazione funzionale

vantaggi	*svantaggi*
organizzazione funzionale: dai dati ai risultati	simulazioni con carta e penna più impegnative
linguaggio del programma vicino a quello dell'analisi del problema	modello computazionale meno intuitivo
astrazione e agganci con la matematica	difficoltà a orientarsi con la ricorsione
utente e programmatore si esprimono nello stesso linguaggio	linguaggi di progetto astratti
sintassi minimale e superamento dei dettagli di input/output	linguaggi poco "popolari"

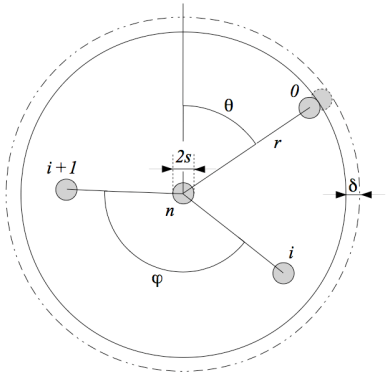
Linguaggi di programmazione orientata agli oggetti

anni '60	Simula	O.-J. Dahl & K. Nygaard: problemi di simulazione — introduzione dei principali concetti
anni '70	Smalltalk	A. Kay: intuizione della generalità ed espressività dell'approccio OOP
anni '80		estensioni OO di vari linguaggi (C, Pascal, LISP, . . .)
1985	C++	B. Stroustrup: linguaggio di ampia diffusione
1988	Eiffel	B. Meyer: formalizzazione nel linguaggio degli "invarianti di classe"
1991	Oak	J. Gosling: Java in embrione
1995	Java	Multipiattaforma, rete, sicurezza. . .

A proposito di simulazione. . .



Modello



$$\pi s^2 = 2\pi r \cdot \delta$$

$$\delta = \frac{s^2}{2} \cdot \frac{1}{r}$$

Esempio: Ciascun seme è un processo

```
public class Seed  extends Thread  implements Observer {

  private static final int S = 5;           // raggio seme
  private static final double A = 1.4 * S;  // area seme
    ...          ...

  // Modello relativo al seme

  public Seed( double theta ) { ... }       // creazione

  public void run() { ... }                 // avanzamento

  // Visualizzazione grafica del seme

  public void update( ... )  { ... }

}  // Seed
```


Esempio: Modello di sviluppo di un seme

```
private static final double K = A*A / 2;

private final double theta;     // direzione spostam.
private double r = A;           // distanza dall'apice

public Seed( double theta ) {  // creazione del seme
  this.theta = theta;
}

public void run() {            // avanzamento radiale
  while ( FOREVER ) {
    Sunflower.delay();
    r = r + K/r;               // spostamento increm.
  }
}
```


Esempio: Visualizzazione di un seme

```
// Visualizzazione grafica del seme

public void update( Observable obs, Object obj )  {

  Graphics g = (Graphics) obj;

  int x = (int) ( Simulation.X0 + r * Math.cos(theta) );
  int y = (int) ( Simulation.Y0 + r * Math.sin(theta) );

  g.setColor( Simulation.FG );
  g.fillOval( x-S, y-S, 2*S, 2*S );
}
```

Esempio: Modello di sviluppo del girasole

```
public class Sunflower  extends Observable
                        implements Runnable {
    ...          ...

  // Modello di fillotassi del girasole

  public void run() { ... }              // primordi

  public void drawHead( Graphics g )  {  // visualizz.
    setChanged();
    notifyObservers( g );
  }

  public static void delay() { ... }     // temporizz.

}  // Sunflower
```


Esempio: Fillotassi del girasole

```
private static final double
  PHI = Math.PI*(Math.sqrt(5)-1), // rapporto aureo
  DELTA = PHI;                    // +/- 0.1%

private double theta = 0;         // direzione iniz.

public void run() {               // generazione
  for ( int k=0; k<N; k=k+1 ) {   // primordi
    delay();
    Seed seed = new Seed(theta);  // nuovo primordio
    theta = theta + DELTA;        // direzione succ.
    addObserver( seed );          // per visualizz.
    seed.start();                 // proc. autonomo
  }
}
```


Esempio: Avvio della simulazione

```
public class Simulation  extends JPanel
                         implements Runnable {

  private final Sunflower sunflower = new Sunflower();

  public Simulation() {                 // simulazione
    ( new Thread(sunflower) ).start();  // sviluppo
    ( new Thread(this) ).start();       // per visual.
  }

  public void run() {                   // refresh
    while ( true ) {
      Sunflower.delay();  repaint();
    }
  }
    ...          ...
}  // Simulation
```

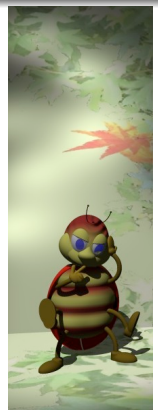

Esempio: Interfaccia grafica

```
public static final Color FG = new Color(0,127,0);
public static final Color BG = Color.ORANGE;

public void paint( Graphics g ) {          // visual.
  g.setColor( BG );
  g.fillRect( 0, 0, getWidth(), getHeight() );
  sunflower.drawHead( g );
}

public static void main(String[] args) {  // avvio
  JFrame fr = new JFrame( HEADER );
  fr.setSize( W, H+HEAD );
  ( fr.getContentPane() ).add( new Simulation() );
  fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  fr.setVisible( true );
}
```

Peculiarità . . .

Fra gli aspetti notevoli della programmazione orientata agli oggetti si possono menzionare:

- Analisi basata su *modelli*
- Trattamento di computazioni parallele



Programmazione orientata agli oggetti

vantaggi	*svantaggi*
organizzazione: entità astratte e relative interazioni	rischio di perdere di vista il concetto di algoritmo
linguaggio in parte definito dal programmatore in base al problema	modello computazionale estremamente complesso
sviluppo di competenze di modellazione (astrazione sui dati)	difficoltà a orientarsi fra diversi livelli di astrazione
diffusione e attualità degli strumenti	numerosi diversivi (librerie sterminate, I/O, GUI. . .)

Linguaggi di programmazione logica/dichiarativa

anni ’60		Precursori “question-answering” ispirati da “advice taker”, ipotizzato da J. McCarthy (1958)
1960–70		Sviluppi nell’ambito dell’intelligenza artificiale
1969	Absys	J.M. Foster & E.W. Elcock: primo linguaggio asserzionale (dichiarativo)
1972	Prolog	R. Kowalski (modello di calcolo: risoluzione SL), A. Colmerauer & P. Russel (realizzazione): programmazione logica “general-purpose”
anni ’90	Curry	M. Hanus e altri: linguaggio logico-funzionale

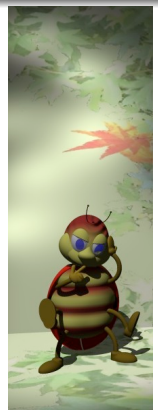
Esempio: Longest Common Subsequence

```
lcs( [], V, [] ).

lcs( U, [], [] ).

lcs( [X|U], [X|V], [X|S] )  :-  lcs( U, V, S ).

lcs( [X|U], [Y|V], S )  :-  X =\= Y,
                            lcs( U, [Y|V], P ),
                            lcs( [X|U], V, Q ),
                            longest( P, Q, S ).


?- lcs( [1,2,3], [1,3,2], Z ).
```

Peculiarità . . .

Fra gli aspetti notevoli della programmazione dichiarativa/logica si possono menzionare:

- Proprietà della soluzione in termini relazionali
- In generale molteplici soluzioni



Programmazione logica

vantaggi	*svantaggi*
organizzazione: relazioni fra dati e risultati	prerequisiti di logica formale
linguaggio del programma vicino a quello di specifica	modello computazionale difficile
agganci con la matematica e con la logica	formalizzazione “densa” ed elevato livello di astrazione
utente e programmatore si esprimono nello stesso linguaggio	linguaggio per molti versi innaturale
sintassi minimale e superamento dei dettagli di input/output	linguaggi poco “popolari”

Un paio di note su Logo

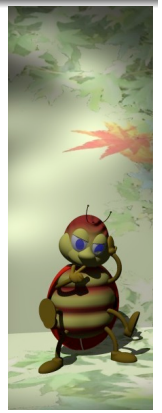
- Realizzato nel 1967 per scopi didattici da W. Feurzeig & S. Papert
- Ambito della pedagogia costruttivista
- È più ricco di quanto normalmente si immagini: utilizzabile anche a livello universitario
- Interessante compromesso fra caratteristiche funzionali e imperative

Esempio: Imperativo o funzionale?

```
to  iteration_step  :x

    output  ( 1 + 1 / :x )
end


to  golden_section  :n

    output  ( cascade :n [iteration_step ?] 1 )
end


? show  golden_section  20
```

Meta-livelli anche con Logo

```
to  phi  :x
    output  ( 1 + 1 / :x )
end

to  iterate  :f  :n  :x
    output  ( cascade :n :f :x )
end


? show ( iterate "phi 50 1 )

? show ( arity "iterate )

? show ( invoke "iterate "phi 3 1 )
```



Programmazione con Logo

vantaggi	*svantaggi*
espressivi sia i costrutti imperativi che funzionali	rischio di confusione fra approccio imperativo e funzionale
feedback immediato con certe modalità d'uso	(ingiustamente) considerato adatto ai bambini
immediata operatività con le funzionalità grafiche di base	può indurre a lavorare senza un progetto
dipende dall'uso...	dipende dall'uso...

Materiale didattico disponibile liberamente

- How to Think Like a Computer Scientist (Java)
  http://www.greenteapress.com/thinkapjava/
- How to Design Programs (Scheme)
  http://www.htdp.org/
- Computer Science Logo Style (Logo)
  http://www.cs.berkeley.edu/ bh/v1-toc2.html
- Logo Foundation (Logo)
  http://el.media.mit.edu/logo-foundation/

Strumenti didattici disponibili liberamente

- Dev + GNU Pascal IDE
  http://www.gnu-pascal.de/binary/mingw32/
- Racket / DrScheme
  http://racket-lang.org/
- SWI Prolog
  http://www.swi-prolog.org/
- FMS Logo
  http://fmslogo.sourceforge.net/

Strumenti didattici disponibili liberamente: Java

- BlueJ
  http://www.bluej.org/
- DrJava
  http://drjava.org/
- Jeliot
  http://cs.joensuu.fi/jeliot/downloads/bluej.php
- Greenfoot
  http://www.greenfoot.org/door

Altri strumenti per apprendere a programmare

- Alice
  http://www.alice.org/
- Scratch
  http://scratch.mit.edu/
- StarLogo
  http://education.mit.edu/starlogo/

Modelli curriculari

C. Stephenson, Chair, 2008
The New Educational Imperative:
Improving High School Computer Science Education
ACM & CSTA

A. Tucker, Chair, 2003
A Model Curriculum for K-12 Computer Science
ACM & CSTA

CC Joint Task Force, 2005
Computing Curricula 2005: The Overview Report
ACM & IEEE-CS

Pausa di riflessione . . .

Grazie
per la vostra
pazienza