# Programmazione funzionale

## Presentazione 2013-2014

Camillo Fiorentini
Alberto Momigliano

*Dimenticate (quasi) tutto quello che sapete di programmazione*

# Organizzazione

- Sito del corso: cooml.di.unimi.it/fp
- Orari (parliamone):
  - Mercoledì, 14.00 -16.30, laboratorio sigma.
  - Giovedì, 14.45-17.00, laboratorio tau,
- Modalità di esame: 2 compitini per chi segue, altrimenti esame scritto finale
  - Possibilità di un progetto per alzare il voto
  - Chi non passa il primo compitino non può fare il secondo.

# Testi e software

- *Functional Programming using F#* , Michael R. Hansen and Hans Rischel,CUP
  - sito del libro.
- Altri riferimenti su pagina web del corso, es:
  - Pagina wiki sulla programmazione in F#
  - Bob Harper. Programming in Standard ML
- F# 2.0 su Visual studio 2010
  - win@di.unimi.it da account studentesco
- Mono su Linux, ma ...

# Corsi collegati/propedeutici

- Ovviamente *programmazione*. Auspicabile *algoritmi*, meglio se avete anche qualche esperienza di Prolog, come da *Intelligenza Artificiale*
- Il corso di Cazzola *Linguaggi di Programmazione"* ha un certo overlap per le prime 3-4 lezioni
- Pubblicità: nuovo curriculum magistrale "*Metodi e modelli per la progettazione e sviluppo del software*"
-

# IMP vs FP

- Un programma imperativo consiste essenzialmente di una sequenza di assegnamenti a variabili
- La sequenza contiene  loops, jumps, branches, astratti come for/while, switch etc
- Vi sono altra astrazioni come procedure, oggetti etc., ma alla fine si programma modificando variabili in memoria
- Es **(linked) lists** in Java (LinkedList.java)

# IMP: problemi

- Semantica imprecisa (es non determinismo in ANSI C)
- Ordine di esecuzione fisso (parallelismo?)
- Poca astrazione (es aritmetica dei puntatori)
- Visione esplicite della memoria (aliasing, null pointers etc.), prona all'errore
- Difficile applicare tecniche formali per la correttezza del software

# FP

- Modello di computazione basato su *valutazione di espressioni* non *esecuzione di comandi* – computazione è passare argomenti a funzioni
- Programmare è dichiarare tipi di dati (la cui rappresentazione è lasciata al compilatore) e delle funzioni che li manipolano
- Invece di loop, funzioni *ricorsive e higher order* (map)
- Es Lz1.fs

# FP

- Focus: costruire *funzioni.*
  - Il programmatore dichiara *cosa* (**non come)** fa il programma definendo una funzione che mappa input a output.
- Si costruiscono funzioni complesse a partire da più semplici *componendole* nel senso matematico del termine
- Nozione matematica di variabile *immutabile*

# FP: tipi

- Tipo = insieme di valori + operazioni
  - int, 'a list, 'a stack red & black tree …
  - stack ops: push, pop, is_empty
- Tipo è una “predizione” del valore che ha un'espressione (se converge)
- Static vs dynamic vs uni-typed
  - *Phase distinction*
- Type checking vs type inference

# FP: vantaggi

- Semantica precisa e semplice
- Ordine flessibile di esecuzione
  - Le funzioni operano isolatamente
  - Parallelismo naturale
- Ricchi meccanismi di astrazione
  - Valori non sono la memoria
  - Astraggo su funzioni, strutture infinite...
- Garbage collection

# FP: svantaggi

- Minor controllo della CPU e della memoria:
  - Programmi occasionalmente poco efficienti
  - Uso non ottimale della memoria (raro in linguaggi *strict* come F#
- Certi algoritmo sono essenzialmente procedurali
  - Come in Haskell, in F# la programmazione monadica simula ciò, e comunque posso usare oggetti .NET

# Perché F#

- Sviluppato da Microsoft Research Cambridge
- Pienamente compatibile con .NET e i suoi linguaggi, in primis C#
- Noi usiamo la versione 2.0, ma esiste la 3.0, che però richiede il pesantissimo VS2012
- Non è un linguaggio accademico, anzi è il competitor di **Scala** senza tradire la filosofia FP, mentre invece Scala …

# Caratteristiche di F# (ML)

- E' *strict* o *call-by-value*
  - Argomenti di una funzione valutati prima del corpo della funzione
- *Impuri*: funzioni possono avere *side effects*
  - Modificare una variable in memoria, etc.
  - Effects usati sporadicamente e solo dove servono.
- *Strongly-typed:*
  - Tipi controllati ed inferiti **staticamente**

# Caratteristiche (cont.)

- *Polimorfismo*: funzioni si applicano a valori di più tipi
- *Higher order*: funzioni sono *first-class* e possono essere passati come ogni altro dato
- *Gestione automatica della memoria* – no puntatori!
- *ADT, eccezioni*
- Specifico di F#: *workflows/monads* (Haskell) e oggetti (.NET)

# Si, tutto bello, ma che mi serve?

- "But let's be honest about this: you'll probably never see an employer advertising for someone with ML skills" Webber, *Modern programming languages*, 2003
- 10 anni dopo: ICFP
- In particolare: CUPS
- Un esempio: Jane Street
- Un esempio italiano (stage disponibile): workinvoice.it

# Si, tutto bello, ma che mi serve?

- **Financial marketplace.** Abbiamo realizzato la versione alfa di una web application per compravendita di fatture commerciali di aziende da parte di investitori qualificati. Il funzionamento della piattaforma e` basato su un meccanismo di asta che mette in concorrenza diversi investitori per acquistare un credito commerciale. La piattaforma e` implementata in Clojure (un linguaggio funzionale della famiglia Lisp, basato su Java Virtual Machine) e database Postgresql, e utilizza un ampio spettro di librerie Clojure (compojure, lib-noir, friend, korma, etc.). Il front end e` Html/Javascript. L'implementazione sta evolvendo rapidamente ed e` previsto il rilascio incrementale di nuove funzionalita` mano a mano che si raccoglie il feedback degli utenti. Cerchiamo stagisti che abbiano gia` una conoscenza di base delle tecnologie web e di Clojure/Common Lisp/linguaggi funzionali e siano interessati a contribuire all'evoluzione della piattaforma.

# Outcome didattico

- Esposizione a un paradigma di prog. alternativo e in grande espansione
- Approfondimento di argomenti trattati poco nel corso di laurea come ADT, lexing & parsing, interpreti, compilatori, type-checkers …
- Cenni su presupposti teorici della programmazione funzionale, in particolare il lambda calcolo e l'inferenza di tipo.
- (Ullmann): it's fun, it's functional!

# Un po' di storia 1

- FP è il paradigma più antico in programmazione; dimenticatevi APL, FORTRAN, etc.
- Lambda-calculus, Church [1932-41]
  - computational notion of a function vs Turing machines [1937],
  - fun vs imp, Church's thesis
  - TM = Lambda

# Un po' di storia 2

- **LISP** [late 50's] (McCarthy)
  - recursion and conditionals,
  - lists and ho functions (map)
  - garbage collection
  - S-expressions for both programs and data
- **ISWIM** (Landin [60's]):
  - first fp coming explicitly from l-calc,
  - first abstract machine
  - emphasis on equational reasoning
  - *let* clauses

# Un po' di storia 3

- Backus's **FP** [77] "Can Programming Be Liberated From the vonNeumann Style? A Functional Style and its Algebra of Programs"
- **ML** (Gordon Milner Wadsworth [79]):
  - first full functional language
  - static types/type inference/ADT/poly
- Lazy: **Miranda** (Turner [85])
  - Guards
  - list comprehension

# Un po' di storia 4

- **Erlang** (Ericsson)  late 1980s, fault-tolerant systems based on async message passing, no shared memory. Started to model telecommunication systems.
- **Haskell** (Hudak, Wadler etc. [88]): first full *lazy* language
- Etc.
- **F#** [Syme 2005]

# SPOT: Costruisci il tuo futuro nell'informatica

Terza edizione dell'iniziativa AICA-CINI-PROSPERA sugli standard europei di competenze e professionalità ICT **per studenti e neo-laureati dei corsi di laurea triennale e magistrale in area informatica**

- Primo incontro su mercato del lavoro e profili professionali più richiesti
  - **Milano**: 19 **marzo ore 15 (aula Alfa)**

- Prossimi incontri (date da definire)

–**Self assessment**: usare gli standard europei per auto valutarsi

–**Colloqui individuali** di indirizzo al self-empowerment

–**Fai la tua start-up (1)**: possibili iniziative imprenditoriali nel settore digitale

–**Fai la tua start-up (2)**: costruzione di un business plan per una start-up nel settore digitale

AICA Associazione Italiana per l'Informatica ed il Calcolo Automatico

prospera Progetto Speranza

cini Consorzio Interuniversitario Nazionale per l'Informatica

UNIVERSITÀ DEGLI STUDI DI MILANO