Tecniche di Programmazione Funzionale e Imperativa

Ivano Salvo

Evoluzione dei Linguaggi di Programmazione

Corso di Laurea in **Informatica**, III anno

SAPIENZA
UNIVERSITÀ DI ROMA

Lezione **1**, 1 marzo 2021

Lezione 1a:

Introduzione al corso

Programmazione vs Algoritmi

Qual è la differenza tra la **programmazione** e gli **algoritmi**?

Prendete un algoritmo di cui avete dimostrato la correttezza:

scrivete il programma e testatelo: usualmente non funziona.

Ma cosa può andare storto?

Tipicamente:

- **Operazioni astratte non** implementate correttamente
- **side-effects** e altri problemi legati alla **memoria**
- Anche l'**efficienza** del programma potrebbe essere non coerente con la complessità asintotica calcolata

Esempio: riflettete su quante sottigliezze possano nascondere soprattutto le liste e altre strutture dinamiche.

Esempio (linguaggio C)

Esercizio 2 (10 punti) Considerare il problema di verificare se una lista di caratteri è palindroma, cioè se letta da sinistra a destra o da destra a sinistra dà la stessa sequenza di caratteri (come le parole 'non', 'otto', 'radar', 'ingegni', 'onorarono', 'ossesso', ...).

1. (**3 punti**) Considerate il seguente programma (dove la funzione `eqList` verifica se due liste sono uguali e `reverse` restituisce il puntatore alla testa di una lista che contiene gli elementi di `L` in ordine rovesciato):

```
int palindroma(charList L){
    if (eqList(L, reverse(L)) return 1;
       else return 0;
   }
```

   Sotto quali ipotesi sulla funzione `reverse` la funzione dà risultati corretti?

`reverse` **deve lasciare** `L` **immutata**. Cioè deve essere una funziona che **crea una nuova lista come risultato**.

Assolutamente **imprevedibili i risultati** se L viene modificata.

Testare cosa accade con il vostro linguaggio preferito (Python?)

Computabilità e Pratica della Programmazione

Da un punto di vista **puramente teorico**, bastano modelli di calcolo (e linguaggi di programmazione) estremamente minimali.

In un corso di Calcolabilità e Complessità probabilmente vi hanno introdotto (o introdurranno?) modelli di calcolo minimali (Macchine di Turing, Macchine a Registri, …)

Ma vogliamo questo?

I linguaggi di programmazione devono fornire **astrazioni sul controllo** (ad esempio: **cicli**, **funzioni** etc.) e **astrazioni sui dati** (definizione dei **tipi di dato**, con operazioni e regole di visibilità)

"**Un linguaggio deve fornire un piccolo numero di metafore uniformi con chiara semantica**"
[Dan Ingalls "**Design Principles behind Smalltalk**"]

Evoluzione dei Linguaggi di Programmazione 1

- **Linguaggi Macchina**
  - **Istruzioni**: trasferimento di memoria/aritmetiche, **identificate da stringhe di bit**
  - completo controllo del processore e della memoria
  - **Tipo di dato**: celle di memoria (**stringhe di bit!**)
  - **Controllo**: salto (in)condizionato
- **Linguaggi Assembler**
  - Uso di **nomi e label simbolici** per istruzioni e dati (`add %d1 3` invece di `000123 010012 0000011`)
  - 1:1 con istruzioni macchina
  - Uso di label simboliche: **astrazioni utile?** [se aggiungo/tolgo un'istruzione non devo modificare tutti i salti]

Evoluzione dei Linguaggi di Programmazione 2

- **Linguaggi Imperativi**
  - **Istruzioni**: Assegnazioni, cicli
  - **Tipo di dato**: **astrazione** delle celle di memoria: controllo sui tipi
  - Definizioni di **nuovi tipi di dato**.
  - **Controllo**: cicli + **astrazione procedurale/funzionale**.
- **Linguaggi Funzionali**
  - Definizione di **funzioni ricorsive**.
  - Controllo: **riduzione di espressioni**.
  - **Niente memoria**, ma ambiente di valutazione.
  - No alias/side-effects: **ricca teoria algebrica**.
- **Linguaggi Logici**
  - Definizione logica della specifica genera un meccanismo computazionale.
  - Programmare **cosa** invece di programmare **come**.

Famiglie di Linguaggi di Programmazione

- **Imperativi**: un programma è essenzialmente una **sequenza di comandi** (sequenza, salti, cicli) che modificano la **memoria** di una macchina (**assegnazione**). L'esecuzione è una sequenza di trasformazioni della memoria: **Algol**, **Fortran**, **C**, **Pascal**, ...
- **Orientati agli Oggetti**: il programma consiste nella definizione di **classi** (≈ tipi) da cui creare **oggetti** (≈ dati). L'esecuzione di un programma è costituito dall'interazione di oggetti (**scambio di messaggi**). Gli oggetti possono essere visti come dati o piccoli automi. **Smalltalk**, **Java**, **C++**, …
- **Funzionali**: un programma è una sequenza di definizione di **funzioni ricorsive**. L'esecuzione consiste nella **riduzione di espressioni** (analoga alla riduzione di un'espressione aritmetica/algebrica): **Haskell**, **ML**, **LisP**, …
- **Logici**: specificano una **formula logica** di **cosa** il programma deve calcolare. Il controllo è implicito: **Prolog**, …

In questo corso…

Graffieremo la superficie del paradigma **funzionale** e a **oggetti**:

- Vedremo le basi di **Haskell** come esempio di linguaggio funzionale;
- Vedremo qualche come alcuni comportamenti si possono mimare con un linguaggio imperativo (C).
- Confronteremo soluzioni in paradigmi diversi.

Lezione 1b:

Informazioni Pratiche

Approposito di questo corso…

Lezioni:
Lunedì, 14 - 16 – Giovedì, 16 - 19
aula **G0** – modalità **BLENDED**.

Pagina web:
`https://twiki.di.uniroma1.it/twiki/view/TPFI/WebHome`

Trovate:

- diario delle lezioni
- materiali vari
- codice
- **queste slides**

Esame

Idealmente:

scritto + orale/progettino

Più probabilmente:

colloquio orale comprensivo di una breve discussione di un progettino

Dipende da numero di studenti/situazione pandemica

Lezione 1

That's all Folks…

…Domande?