Octave

Luca Abeni

Linguaggi di Programmazione di Alto Livello

- Parole chiave: possibilmente con significato in inglese
  - Ma agli informatici piacciono molto contrazioni e acronimi...
  - Esempio: Integer $\rightarrow$ `int`
- Costrutti: da programmazione strutturata
  - Sequenza
  - Ciclo
  - Selezione
- Significati: il più possibile intuitivi

Linguaggi di Programmazione di Basso Livello

- Parole chiave: codici mnemonici
  - Direttamente associabili ad istruzioni macchina
  - Ancora: contrazioni e acronimi!
    - `addl %eax, %eax`
    - `cmpl $9, -4(%ebp)`, `jle .L3`
- Costrutti: solo sequenze e salti
  - Salti incondizionati (`jmp`) o condizionali (`jle .L3`)
  - No cicli e selezioni: implementabili tramite salti!
- Spaghetti programming???
  - Il codice di basso livello deve essere facilmente interpretabile dal computer, non da esseri umani!!!

Conversione fra Alto e Basso Livello

- Linguaggio di programmazione: lessico, sintassi e semantica ben definiti
  - Conversione alto/basso può essere automatica
  - Programma "traduttore"
- Conversione da file testo (codifica ASCII del programma) a file binario (eseguibile)
  - Passaggio attraverso linguaggio di basso livello "intermedio" (Assembly)

```
c = a + b;        ⇒    movl    12(%ebp), %eax
                       movl    8(%ebp), %edx
                       addl    %edx, %eax
                       movl    %eax, c
```

Il "Traduttore"

- Traduzione da linguaggio ad alto livello ad assembly e poi codice macchina
- Può avvenire in vari modi:
  - Compilatore: trasforma file testo in file eseguibili
  - Interprete: trasforma il programma ad alto livello istruzione per istruzione, durante l'esecuzione
- Compilatore: 2 fasi (compilazione poi esecuzione)
  - Il programma generato è più efficiente
- Interprete: conversione ed esecuzione intercalate
  - Più flessibile
  - Sempre necessario per eseguire un programma

Compilatori vs Interpreti

- Conversione di programmi da linguaggio ad alto livello a linguaggio a basso livello $\rightarrow$ equivalente a traduzione di linguaggi naturali
- Se devo comunicare con persone che parlano un'altra lingua...
  - Posso scrivere un testo in italiano, farlo tradurre, spedirlo all'altra persona ed attendere una risposta (tradotta)
  - Posso ricorrere ad un traduttore simultaneo, che traduce per l'altra persona quel che dico man mano che parlo
- Nel primo caso usiamo un compilatore (genera una versione tradotta del testo), nel secondo caso un interprete (traduzione intercalata con la comunicazione)

L'Ambiente Octave

- Abbiamo visto il C come esempio di linguaggio compilato
- Vediamo ora un esempio di linguaggio interpretato
  - Ricordi? Più flessibile, ma meno efficiente...
- GNU Octave: linguaggio di programmazione ad alto livello orientato al calcolo scientifico
  - Clone libero (open source) di mathlab
  - Interprete con interfaccia a linea di comando
- Vedremo un utilizzo di base
  - Focus su differenze / similitudini con C

Perché Octave?

- Software libero
  - Non è reato copiarlo
  - Non una questione di soldi, ma di liberà / filosofia
  - Free as in “free speech”, not as in “free beer”
  - Codice sorgente disponibile a tutti
  - `http://www.gnu.org/philosophy/philosophy.html` per maggiori informazioni
- Discreta compatibilità con mathlab...
  - ...Che non è software libero
  - Provare a leggere la licenza d’uso

Il Linguaggio Octave

- Linguaggio interpretato
- Linguaggio di alto livello, strutturato
- Sintassi simile al linguaggio C per molti aspetti
- Alcune differenze
  - Non occorre definire una funzione main
  - Non occorre dichiarare le variabili prima di usarle
  - Non occorre specificare il tipo delle variabili
- Supporto “avanzato” per array e matrici

Variabili in Octave

- Concetto di variabile simile ad altri linguaggi di programmazione
- Non occorre definire / dichiarare variabili
  - L'interprete crea una variabile quando è usata per la prima volta
  - Vantaggio: semplicità
  - Svantaggio: più facile commettere errori...
- Il tipo di una variabile è deciso dall'interprete in base al valore assegnato
- Il tipo può variare durante l'esecuzione di un programma!!!
  - Tipi di variabili: numero, stringa o array/matrice (vettori)

Array - 1

- Usati per rappresentare vettori o matrici (nel senso matematico del termine)
- Sequenze di numeri
- Rappresentati in Octave da numeri separati da virgole o spazi e racchiusi fra parentesi quadre (esempio: `[1 2 3 4 5]`)
  - Esempio: `v = [2 4 5 0 23]` assegna un array contenente i numeri 2, 4, 5, 0 e 23 alla variabile `v`
  - Sequenze di numeri possono essere create tramite il simbolo ":". Esempio: `v = 1:5` equivale a `v = [1 2 3 4 5]`

Array - 2

- Si possono usare array per definire altri array
  - Esempio: `v1 = [1.1 2.2 3]; v2 = [v1 5]` assegna `[1.1 2.2 3 5]` a `v2`
- I singoli elementi di un array si accedono tramite "`()`" e non "`[]`" come in C
  - *i*-esimo elemento dell'array `v`: "`v(`*i*`)`"
- I valori dell'indice cominciano da 1 (e non da 0 come in C)

Costrutti Strutturati

- Octave: linguaggio di alto livello: supporta programmazione strutturata
- Servono costrutti sintattici per selezione e ciclo
  - Selezione: `if ... else ... end`
  - Cicli: `for ... end` e while ... end

```
if condizione
  ...
else
  ...
end
```

Simile al linguaggio C

```
for var = vettore
  ...
end
```

Ripete il ciclo per ogni elemento del `vettore` (assegnato a `var`)

```
while condizione
  ...
end
```

Simile al linguaggio C

Funzioni

- Le funzioni in octave non sono fondamentali come in C...
- ...Ma è comunque possibile definire funzioni!
  - Keyword `function`
  - `function` *risultato* = *nome*(*parametri*)
  - `end` (o `endfunction`) per terminare il corpo della funzione
- Similitudini e differenze con linguaggio C
  - Possibile avere più di un valore di ritorno!

```
function [somma, differenza] = esempio(a, b)
    somma     = a + b;
    differenza = a - b;
end
```

Funzioni in File .m

- È possibile definire una funzione in un file `.m`
- Al solito, utilizzabile con "`source "file.m"`"
  - Il comando rende usabile la definizione di funzione
- Possibile alternativa: scrivere la definizione di funzione in un file che ha nome "*nomefunzione*`.m`"
  - Funzione disponibile direttamente, senza usare `source`

Calcolo di Massimo e Minimo

```
if (a >= b )
    max = a
    min = b
else
    max = b
    min = a
endif
```

- Simile a C
- Non ci sono “{” e “}”
  - L’inizio e la fine dei blocchi sono marcati dalle keyword
- Non si sono “;”
  - “;” ha solo l’effetto di rendere silenziosi gli assegnamenti

Calcolo della Media

```
x = [1.2, 6.3, 7.8, 3.6];

somma = 0;
for elemento = x
    somma = somma + elemento;
end
x_mean = somma / length(x)
```

- Costrutto `for` diverso da quello del C
  - Scorre elementi di un array
- Alcuni statement con “;”, altri senza!
- `length()`: lunghezza di un array

Fattoriale di un Numero

```
risultato = 1;
if( n != 0 )
    numeri = 1:n;
    for i = numeri
        risultato = risultato * i;
    end
end
fattoriale = risultato
```

```
if( n == 0 )
    fattoriale = 1;
else
    fattoriale = prod( 1:n )
end
```

Massimo e Minimo - Funzione

```
function [max,min] = minmax(a, b)
    if(a >= b )
        max = a;
        min = b;
        return;
    else
        max = b;
        min = a;
        return;
    end
end
```

Media - Funzione

```
function res = mean(x)
  sum = 0;
  for entry = x,
    sum = sum + entry;
  end
  res = sum / length(x);
endf
```

Fattoriale - Funzione

```
function result = fattoriale(n)
    if (n == 0)
        result = 1;
        return;
    else
        result = prod(1:n);
    end
end
```

```
function result = fattoriale(n)
    if (n == 0)
        result = 1;
        return;
    else
        result = n * fattoriale(n - 1);
    end
end
```