Linguaggi di Programmazione

Luca Abeni

March 10, 2014

Linguaggi di Programmazione

- Necessità di esprimere un algoritmo in modo formale
  - Abbiamo accennato diagrammi a blocchi...
  - ...Ma esistono anche molti altri modi!
- Per "dare in pasto" ad un computer un algoritmo, va descritto tramite un linguaggio di programmazione
  - Evitare ambiguità ed imprecisioni
  - Descrivere completamente ed in modo preciso un algoritmo
- Cos'è un linguaggio? Cercare sul vocabolario...

Definizione Formale di Linguaggio

- Per definire in modo formale un linguaggio, occorre specificare:
  - Quali parole fanno parte del linguaggio (Lessico)
  - Come costruire frasi "grammaticalmente corrette" a partire dai simboli del linguaggio (Sintassi)
  - Come assegnare un significato univoco e ben specificato a frasi sintatticamente corrette (Semantica)
- Non ha senso provare a creare frasi sintatticamente corrette con simboli che non fanno parte del linguaggio
- Non ha senso interrogarsi sul significato di una frase non sintatticamente corretta

Definizione di Linguaggio - 2

- Linguaggio di programmazione: costuito, come ogni altro linguaggio, da:
  - Un alfabeto di simboli
    - Simboli con cui vengono costruite parole e frasi
  - Un insieme di regole sintattiche per la costruzione di frasi corrette
  - Un insieme di regole lessicali per associare significato alle frasi
    - Regole per la costruzione e l'uso corretto delle parole e delle frasi del linguaggio
- In altre parole: insieme di parole chiave, costrutti e loro significati

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
  - La conversione fra due linguaggi può essere fatta in modo automatico
  - Programma "traduttore"
- Conversione da file testo (codifica ASCII del programma) a file binario (eseguibile contenente la codifica binaria di istruzioni macchina)
  - 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 direttamente file testo in file eseguibili
  - Interprete: trasforma il programma ad alto livello istruzione per istruzione, durante l'esecuzione
- Compilatore: processo in 2 fasi (prima compilazione, poi esecuzione)
  - Il programma generato è più efficiente
- Linguaggi interpretati: conversione ed esecuzione sono intercalate
  - Più flessibile
  - L'interprete è 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)

Linguaggi di Programmazione

- Linguaggio di programmazione:
  - Formalismo che permette di descrivere algoritmi in modo univoco e preciso
  - Evita ambiguità $\rightarrow$ più utile ai computer
- Caratterizzato da una sintassi ed una semantica ben definite
  - Sintassi: come scrivere "frasi" corrette
  - Semantica: come associare un significato preciso a "frasi" sintatticamente corrette
- Linguaggi di alto livello: vanno convertiti in linguaggio macchina per essere eseguiti da un computer

Come Scrivere ed Eseguire un Programma

- Programma: descritto da parole e simboli
  - Salvato su disco come file ASCII
  - Usare un editor di testo (`gedit` o simile)
  - Estensione `.c` per il linguaggio C
- I computer non sanno eseguire file di testo!!!
  - Convertito prima in linguaggio a basso livello (Assembly)
  - Poi l'Assembly è convertito in linguaggio macchina...
  - E finalmente pezzi di codice in linguaggio macchina sono messi assieme per formare un programma eseguibile

Di' Ciao...

- Insegnamo al computer a salutare (scriviamo un saluto)
- Primo programma, molto semplice
  - Algoritmo molto semplice: una sola istruzione
    - Scrivi "Hi, there!" sullo schermo

```
#include <stdio.h>

int main()
{
   printf("Hi, there!\n");

   return 0;
}
```

- Tutto chiaro, no?

Come Eseguo Questa Roba?

- Aprire una shell e usare `gedit` (o simile) per creare un file "hi.c" contenente il programma
  - E ora???
  - Il computer non può certo eseguire questa sequenza di caratteri
- Il programma va trasformato in un *eseguibile* in qualche modo...
  - Compilatore C!!!
  - Vedi slide su compilatori ed interpreti...
- Digitare: `gcc -Wall hi.c -o hi`
  - Uh... E cosa vuol dire???

Qualche Dettaglio su gcc

- gcc: comando che converte un file testo (contenente un programma C) in un file eseguibile...
- Gnu Compiler Collection
  - Può fare molto più che compilare un programma C...
- Vari passaggi:
  1. Pre-Processore (direttive "#")
  2. Converte da alto a basso livello (C $\rightarrow$ Assembly)
  3. Converte da Assembly a linguaggio macchina (sequenze di 0 e 1)
  4. Mette assieme codice in linguaggio macchina per generare un eseguibile

Anatomia di un Programma C

- In C, gli algoritmi sono modellati come *funzioni*
  - Algoritmo: blocchi in sequenza o annidati...
  - Blocco: annidamento di blocchi o sequenza di operazioni o blocchi...
  - In C, non solo operazioni, ma anche chiamate a funzioni
  - Algoritmo come composto da "sottoalgoritmi"
- Programma $\rightarrow$ funzione principale (`main`)
  - Nome e parametri stabiliti dallo standard C
- Altre funzioni definite dal compilatore
  - Esempio: I/O - no operazioni ma funzioni

La Funzione main()

- Eseguendo un programma, il sistema comincia sempre dalla funzione chiamata `main()`
  - Ogni programma deve sempre contenere almeno una funzione, chiamata `main()`
- Valore di ritorno: deve essere sempre di tipo `int` (insieme di definizione: numeri interi. 16, 32 o 64 bit per intero, dipende dalla CPU)
  - Valore di ritorno 0: il programma è terminato senza errori
- Parametri: permettono di capire i parametri passati al programma a linea di comando.
  - Per ora non ci interessano: usiamo `int main()`

Le Funzioni di I/O

- In un generico algoritmo si considerano operazioni di I/O...
  - ...Ma il linguaggio C non prevede istruzioni/operazioni di I/O
  - Ingresso e uscita sono implementati tramite funzioni
- Esempio: `printf()`
  - `int printf("stringa", ...)`
  - Stampa *"stringa"* sullo schermo, ma può fare di più
    - Stampare il valore di variabili
  - In C, stringhe racchiuse fra virgolette
    - Esempio: `"Hi, there!"`

Blocchi di Codice

- Un algoritmo strutturato è "scomponibile" in blocchi
  - Blocchi concatenati o annidati
- In C, il simbolo "{" segna l'inizio di un blocco ed il simbolo "}" ne segnala la fine
  - **Ricorda:** dopo un "{", le righe che seguono vanno scritte più a destra (4 spazi, 1 tab, ...)
  - Col simbolo "}" si ritorna all'allineamento del "{" corrispondente
  - *Indentazione:* permette di identificare visivamente i vari blocchi di codice in modo semplice!
- Blocchi usati per funzioni o cicli o selezioni

Variabili

- Variabile `(n,v)`: `n` è il nome, `v` è il valore attuale
  - Valore: elemento di un insieme detto insieme di definizione della variabile
  - I valori delle variabili sono specificati solo durante l'esecuzione dell'algoritmo
- Linguaggio di programmazione: nome variabile associato ad una o più celle di memoria
  - Quando è fatta / disfatta tale associazione?
    - Tempo di vita della variabile...
  - Cosa contengono tali celle di memoria?
    - Insieme di definizione / Tipo della variabile...
  - Inoltre: dove è visibile la variabile?

Variabili in C

- In C, le variabili vanno *dichiarate* (meglio: *definite*) prima di usarle
  - Fuori da ogni blocco
  - All'inizio di un blocco
- *tipo* nome;
  - Esempio `int i;`
  - Il tipo di una variabile ne specifica l'insieme di definizione...
  - ...Vale a dire, come interpretare la sequenza di bit contenuta nella memoria associata alla variabile
- Posizione della definizione (fuori da tutti i blocchi o all'inizio di un blocco) $\rightarrow$ tempo di vita e visibilità

Funzioni in C

- Caratterizzate da parametri (e loro insiemi di definizione) e insieme di definizione del risultato
  - Possono avere effetti collaterali (esempio: `printf()`)
- *tipo* nome(*tipo* parametro, ...)
- seguito da un blocco (corpo della funzione)
  - Esempio: `int fattoriale(int n)`
  - `double discriminante(double a, double b, double c)`
  - Parametri non specificati: `int main()`
  - Nessun parametro: `int esempio(void)`
  - No valore di ritorno: `void esempio2(int a)`

Definizione e Dichiarazione di Identificatori

- Variabili e funzioni hanno un nome: Identificatore
  - Case sensitive (`int variabile` è diverso da `int Variabile`)
  - Composti da lettere, cifra e "_"
  - Non possono cominciare con una cifra
- Dichiarazione: associa un identificatore ad una entità (variabile o funzione)
  - Non "crea" realmente l'entità (quindi, non occorre descriverla appieno)
- Definizione: descrive come è fatta l'entità e la "crea"
  - Per le variabili, spesso dichiarazione e definizione coincidono

Prototipi di Funzioni

- Dichiarazione di funzione: prototipo
  - Specifica che ad un identificatore corrisponde una funzione
  - Specifica i parametri (con relativo insieme di definizione) e l'insieme di definizione del valore di ritorno della funzione
  - Non descrive come la funzione è fatta
- Definizione di una funzione: include prototipo e corpo della funzione
- Per poter invocare una funzione, devo prima averne dichiarato il prototipo
  - Ma... E `printf("Hi there!!!\n");`? Dove è dichiarata `printf()`?
  - Nel file `stdio.h`!!! Ecco a cosa serve `#include <stdio.h>`!!!