

Protocolli e Architetture di Rete

Appunti di teoria

Iacopo Ruzzier

Ultimo aggiornamento: 2 novembre 2024

Indice

1	Nozioni introduttive	2
1.1	Compilatori e interpreti	2
1.1.1	Compilazione con GCC	2
1.1.2	Compilatori vs Interpreti	2
1.2	Struttura del compilatore	3
1.2.1	Schema riassuntivo (moduli frontend)	4
1.3	Alcune nozioni di base	4
2	Linguaggi formali	5
3	Linguaggi regolari	5
4	Analizzatore lessicale (lexer)	5
5	Sintassi e grammatiche	5

1 Nozioni introduttive

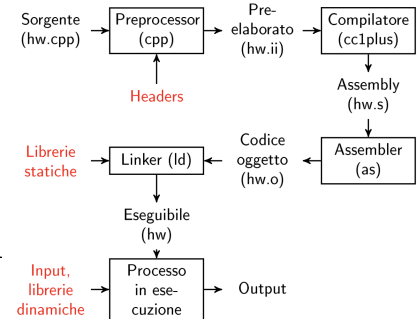
1.1 Compilatori e interpreti

Il **compilatore** è un componente della toolchain di programmi usati per **creare eseguibili** a partire da programmi scritti in un qualche **linguaggio di programmazione**

Altri componenti della toolchain sono

- precompilatore
- assemblatore
- linker (statico e dinamico)

Solitamente invoco i componenti mediante un unico **programma driver**



1.1.1 Compilazione con GCC

La **Gnu Compiler Collection** o **GCC** (originariamente acronimo di *GNU C Compiler*) è una suite per C/C++, Fortran e Ada. I componenti in relazione a C/C++ ed all'ambiente GNU/Linux sono:

```
cpp → cc1/cc1plus → as → ld
```

con `g++` programma driver.

I programmi in uso separato (spesso non in path, a settembre 2024 su distro Debian in `/usr/libexec/gcc/x86_64-linux-gnu/XX/`):

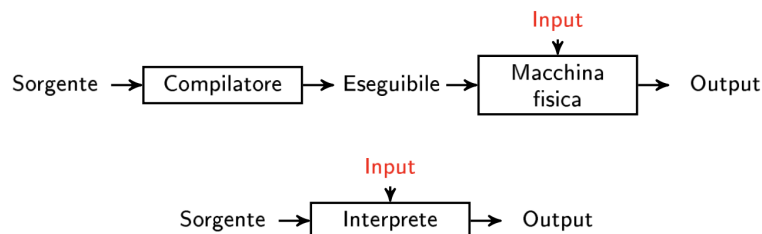
```
cpp -o hw.ii hw.cpp #preproc
cc1plus -o hw.s hw.ii 2>/dev/null #quasi sicuramente non in path
as -o hw.o hw.s
```

Per determinare i parametri del linker uso `g++` con opzione `-v` (verbose). Il driver (tra le altre cose) invoca il linker tramite comando `collect` (o `collect2`), ed elenca i parametri usati:

```
g++ -v -o hw hw.o
# esempio di output (alla fine):
# > [...] -build-id -eh-frame-hdr -m elf_x86_64 \
# > -hash-style=gnu -as-needed \
# > -dynamic-linker /lib64/ld-linux-x86-64.so.2 \
# > ...
# > /usr/lib/x86_64-linux-gnu/crti.o
ld -build-id -eh-frame-hdr [...] # invoco con tutti i parametri trovati,
# di norma escludendo i plugin
```

1.1.2 Compilatori vs Interpreti

Interpretazione: impressione di eseguire il programma direttamente in **linguaggio sorgente**



Un interprete *puro* legge il sorgente, lo analizza e lo esegue **mentre procede** → inefficiente (troppo tempo per **analisi testuale** e **riconoscimento di espressioni**), usato per pochi linguaggi (es. Lisp originale)

In generale, un'implementazione interpretata include un **traduttore** (identico al frontend di un compilatore) che fornisce un risultato \pm **vicino** alla macchina fisica - qui sta la differenza tra i vari interpreti

Modello Perl (Practical Extraction and Report Language)

- la parte iniziale di traduzione produce una rappr. **ad albero** del programma (AST - Abstract Syntax Tree)
- l'interpretazione del programma avviene mediante **visita post-order** dell'AST (con opportune str. dati di supporto)
Per maggiore efficienza, vengono prima eseguite svariate **ottimizzazioni** (es. porzioni da eseguire più volte vengono tradotte in codice macchina)

Modello Java (e Python)

Tipicamente, il traduttore produce codice **eseguibile da una VM \rightarrow bytecode**

- caso Perl: percezione **analogica all'int. pura** - trad. e int. appaiono come programma unico, l'esecuzione avviene in risposta al singolo comando
- caso Java: tr. in bytecode e int. in **momenti distinti** - ha i moduli distinti **javac** e il **JRE**

1.2 Struttura del compilatore

(da qui in poi inteso come modulo, non come toolchain completa)

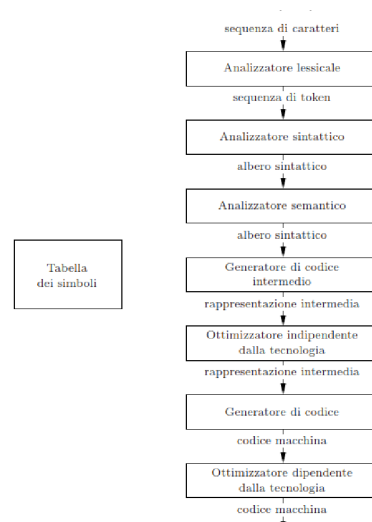
Strutturato in 3 moduli:

1. **front-end**: specializzato nel linguaggio, opera sul sorgente e produce una rappr. intermedia sia *machine* che *language-independent*
2. **middle-end**: ottimizza il codice intermedio (focus modulo 2)
3. **back-end**: produce il codice per l'architettura target (con specifiche ottimizzazioni)

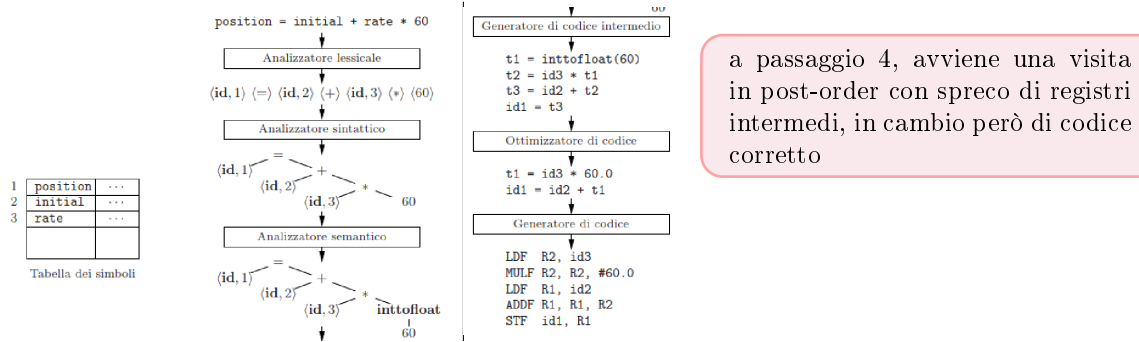
Passi della compilazione (4 + 1 + 2)

- **front-end**
 - **analizzatore lessicale**: raggruppa i caratteri in **token** (parentesi, op, id, ...)
 - ↓ **analizzatore sintattico**: controlla se i token formano strutture legali e restituisce un **albero sintattico** (dà struttura)
 - ↓ **analizzatore semantico**: dall'albero s. restituisce un a. *semantico*, ed esegue controlli più complessi (type check, check sul num. di argomenti passati ad una f. rispetto al num. di parametri formali, ...)
 - ↓ **generatore di codice intermedio**: dall'a. semantico produce la rappr. **intermedia** (codice corretto e funzionante, **non ottimizzato**)
- ↓ **middle-end**
 - ↓ **ottimizzatore** sulla rappr. intermedia
- ↓ **back-end**
 - **generatore + ottimizzatore** codice macchina

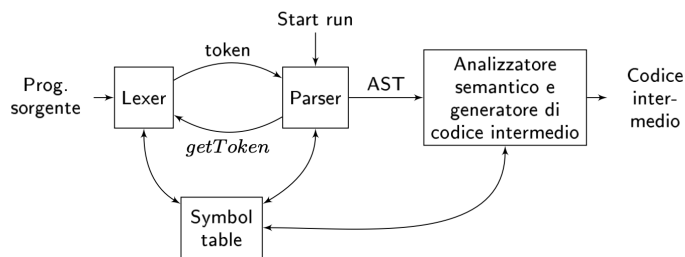
Tabella dei simboli: dizionario (tipicam. hash table) che memorizza i simboli **incontrati man mano** durante l'analisi del sorgente, assieme alle loro caratteristiche (posizione, tipo, ...)



Esempio: fasi applicate ad un'istruzione



1.2.1 Schema riassuntivo (moduli frontend)



Sia per lexer che parser esistono dei generatori, ma lasciano il compito di scrivere le **regex per identificare i token** (lexer) e le **regole di com'è fatto il linguaggio** (le grammatiche formali) (parser)

1.3 Alcune nozioni di base

(da ricordare)

- **type checking:** + o - forte; controllo sugli operandi; statico o dinamico
- **regole di scope:** definiscono la visibilità delle variabili
- **ambiente e memoria:**
 - ambiente: mapping tra nomi e locazioni di memoria (int a modifica l'a.)
 - memoria: mapping tra locazioni di memoria e valori (a = 1 modifica la m.)

Nei linguaggi formali la memoria **non si vede** (mapping diretto nomi-valori senza puntatori)

- **l-value e r-value:**
 - l-value: oggetti con posizione di memoria identificabile (es. id)
 - r-value: valori a destra di =

```

int x; int *p;
p = &x # legale
&x = p # illegale
l-value = r-value # in generale
    
```

- **implementazione di**
 - stack (pila - linked list)
 - dizionario (hash map, dict)
 - albero binario (struct con 2 puntatori, array paralleli; info, indice sx, indice dx)
 - albero n-ario (array paralleli non utilizzabili: struct con puntatori left e next-sibling)

- 2 Linguaggi formali
- 3 Linguaggi regolari
- 4 Analizzatore lessicale (lexer)
- 5 Sintassi e grammatiche