Appunti di Compilatori

Riccardo Lo Iacono

Dipartimento di Matematica & Informatica
Università degli studi di Palermo
Sicilia
a.a. 2023-2024

Indice.

– 1 – Introduzione: interpreti e compilatori.

La necessità di semplificare la scrittura di codice sorgente, porta alla nascita dei primi linguaggi di programmazione. Ciò preclude quindi un "meccanismo" che permetta di descrivere il come e il cosa, si possa fare con un dato linguaggio. E, segue banalmente, che tale meccanismo può essere definito solo con un linguaggio esistente. Per questa e altre ragioni, nascono i compilatori e gli interpreti.

Nota: sebbene non di interesse ai fini del corso, a seguito si fa una breve digressione sugli interpreti.

A partire dal codice sorgente, un interprete converte, istruzione per istruzione, il sorgente che è immediatamente eseguito. Linguaggi di questo tipo sono *python, perl, ecc.*

Parlando ora dei compilatori, questi convertono il source-code in un codice macchina ***equivalente***. Ulteriore compito dei compilatori è quello di segnalare eventuali errori.

Nota: esistono linguaggi (eg. JAVA) che fanno uso di compilatori ibridi: ossia compilatori che implementano sia la compilazione, sia l'interpretazione del sorgente.

– 1.1 – Struttura di un compilatore.

La struttura di un compilatore, può essere suddivisa in due parti: il "front-end" composto dalle fasi in *Figure 1.1*; e il "front-end" relativa la parte di generazione del codice.

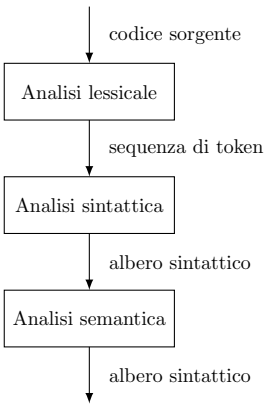

Figura 1.1: Struttura di un compilatore.

Nota: di interesse al corso risulta principalmente le fasi di front-end. Eventuali accenni alle altre fasi saranno discusse alla fine.

– 2 – Analisi lessicale.

Come mostrato in *Figura 1.1*, l'analisi lessicale è la prima fase della compilazione. I suoi compiti sono sintetizzati a seguire.

1. Il sorgente è scansionato e da questi si compongono i *lessemi*: sequenze di caratteri con un determinato significato.

   Esempio: un lessema per la gestione dei dati sarà del tipo: t_dataType.

2. Per ciascuno dei lessemi, un analizzatore sintattico genera dei token della forma (token_name, address), successivamente gestiti dall'analisi sintattica.

   Qui token_name identifica un lessema, mentre address è un puntatore alla cosiddetta *symbol table*. Quest'ultima, in breve, contiene le diverse proprietà di un istanza di un lessema.

Per quel che riguarda lo scanner questi ha essenzialmente due compiti:

- costruire la symbol table;
- semplificare il sorgente.

Prima che ciò possa essere fatto però, è necessario, a meno che non sia stata eseguita una fase di precompilazione, che:

- vengano rimossi i commenti: come ovvio sono utili al solo programmatore, dunque, per alleggeri l'eseguibile, si procede alla loro rimozione;
- si effettui una case conversion: se il linguaggio non distingue tra maiuscole e minuscole, allora si converte il sorgente interamente in minuscolo;
- si rimuovano gli spazi: per motivi analoghi ai commenti, si elimina gli spazi superflui;
- si deve tenere traccia del numero di linea: ciò è utile per la segnalazione di eventuali errori.

Nota: sebbene in *Figura 1.1* sia mostrata come fase precedente l'analisi sintattica, più correttamente, l'analisi lessicale è da intendere come una sub-routine di quella sintattica.

Nota: per quel che riguarda l'implementazione della symbol table, ciò è generalmente realizzato con una hash table.

Nota: per rappresentare la struttura dei token, si fa uso delle espressioni regolari. Poiché si suppongono conoscenze pregresse, non saranno trattate.

– 2.1 – Gestione degli input.

Lo scanner analizza il sorgente carattere per carattere, ma poiché un token potrebbe essere composto da più caratteri, è necessario un metodo di "backtracking": cioè un modo per poter tenere traccia di dove un token inizi. Ciò è generalmente realizzato con un doppio buffering. Con l'ausilio di due puntatori, forward e lexem_begin, si procede ad identificare i token. Nello specifico inizialmente i due puntatori coincidono, successivamente si fa avanzare forward fintantoché si riscontra un lessema. Fatto ciò si aggiorna la posizione di lexem_begin.