Appunti sui compilatori

I compilatori sono degli strumenti che trasformano il codice sorgente scritto in un determinato linguaggio di programmazione in codice assembly/macchina eseguibile.

Il processo di compilazione è suddiviso in due macroaree:

- analisi;
- sintesi.

Analisi

La fase iniziale della compilazione è la fase di **analisi**, nella quale il compilatore esamina il codice sorgente e verifica la correttezza delle istruzioni. Questo processo è suddiviso in diverse fasi:

- analisi lessicale;
- analisi sintattica;
- analisi semantica.

Mentre il compilatore analizza il codice viene popolata la **tabella dei simboli**, che ricopre un ruolo fondamentale in tutto il processo di compilazione.

Analisi lessicale

In questo primo passo di analisi lo **scanner** (o analizzatore lessicale) scompone, leggendo carattere per carattere, il codice sorgente in unità elementari chiamate **token**. I token possiedono due caratteristiche:

- **categoria** (o tipo): che tipo di informazione rappresenta il token (identificatore, operatore, costante ecc...);
- **lexeme**: la sequenza vera e propria di caratteri che lo compone ("if", "x", "int" ecc...).

Ogni token individuato viene inserito all'interno di una **lista** utilizzata nelle fasi successive.
Poiché i pattern, ovvero le regole che descrivono l'insieme di stringhe che corrispondono al token, possono essere rappresentati attraverso **espressioni regolari**, lo scanner consiste nell'**implementazione di automi a stati finiti deterministici** in grado di riconoscere i vari tipi di token e classificarli. Nel caso in cui si individui una stringa consentita per diversi pattern, si utilizza un meccanismo di priorità nella

classificazione basato sull'ordine di definizione dei pattern. Esempio è il token "if", che corrisponde sia ad una parola chiave che ad un possibile identificatore, riconosciuto come parola chiave poiché il pattern per le parole chiave è definito prima di quello per gli identificatori. Se si invertisse l'ordine, "if" verrebbe riconosciuto come identificatore.

Analisi sintattica

Lo strumento che svolge questo compito è il **parser** (o analizzatore sintattico). Mentre l'obiettivo del passo di analisi lessicale era il riconoscimento dei singoli token, in questa fase vengono prese in considerazione sequenze di token per verificare che la **sintassi delle istruzioni e dei costrutti** sia valida. Le grammatiche dei linguaggi di programmazione sono di tipo 2 e la struttura di un programma può essere rappresentata attraverso un **albero sintattico**, nel quale il primo simbolo indica il punto di partenza, i nodi intermedi sono i simboli non terminali e le foglie sono i token.

Analisi semantica

L'ultima fase del processo di analisi è l'analisi semantica, nella quale si verifica la **consistenza semantica** del codice scritto (ovvero che esso abbia senso). Sono diversi i controlli che vengono svolti in questa fase:

- controlli di unicità: verificare che le variabili siano dichiarate una sola volta;
- controlli di flusso: verificare che le istruzioni di controllo di flusso vengano utilizzate correttamente;
- controllo dei tipi: verificare che le operazioni vengano applicate a operandi di tipo appropriato;
- controllo dei nomi: risolvere i riferimenti ai nomi.

In questa fase la tabella dei simboli ricopre un ruolo fondamentale, poiché l'analizzatore semantico recupera informazioni sulle caratteristiche delle varie entità del programma. L'output di questo processo è una rappresentazione intermedia, ovvero una rappresentazione del programma più astratta rispetto al codice sorgente ma meno astratta del codice macchina finale, utile nei processi di sintesi. La rappresentazione intermedia può assumere diverse forme, ad esempio quella di una **quadrupla** o di un**albero sintattico astratto**. Le quadruple sono utili per la rappresentazione delle **operazioni**. Esse sono nella forma `op arg1 arg2 result`, dove:

- `op` è il tipo di operazione;
- `arg1 arg2` sono operandi (spesso variabili temporanee create dal compilatore);
- `result` è la variabile in cui memorizzare il risultato.

Durante la fase di analisi lessicale viene inoltre trasformato l'albero sintattico prodotto dal processo di analisi sintattica in **albero sintattico astratto**, che rappresenta la struttura logica del programma, eliminando tutte le categorie sintattiche intermedie. Le foglie di questo albero rappresentano gli operandi e i nodi interni rappresentano le operazioni.
Il codice intermedio generato può essere sottoposto a ottimizzazione: attraverso l'eliminazione di codice ridondante, propagazione di costanti e ottimizzazione dei flussi di controllo si può migliorare l'efficienza del codice intermedio (e di conseguenza anche del codice macchina risultante).

Sintesi

In questa parte del processo di compilazione i risultati dei processi di analisi vengono utilizzati per produrre il codice target.

Generazione del codice

Utilizzando le informazioni sull'architettura del processore, le istruzioni disponibili e le convenzioni di chiamata, il generatore del codice prende in input le rappresentazioni intermedie e le **converte in codice oggetto**/macchina. Nella conversione vengono presi in considerazione diversi aspetti cruciali:

- **scelta delle istruzioni**: sulla base del codice intermedio, scegliere la sequenza di istruzioni macchina che implementa meglio le operazioni.
- **allocazione dei registri**: l'ottimizzazione dell'uso dei registri è cruciale, vista la loro velocità. Il generatore deve capire quali variabili conservare nei registri sulla base del loro utilizzo, riducendo così i tempi di accesso alle informazioni più utilizzate.
- **gestione della memoria**: determinare come allocare e gestire le variabili e le strutture dati in memoria.
- **ordinamento delle istruzioni**: in alcune architetture l'ordine delle istruzioni può influenzare le prestazioni.

Il risultato di questa fase è il **codice target**. Esso può essere:

- codice assembly (da assemblare);
- codice macchina rilocabile, contenente segnaposti per gli indirizzi di memoria;
- codice macchina assoluto, pronto a essere caricato in memoria ed eseguito.

Ottimizzazione del codice target

Il codice prodotto nel processo precedente può essere passato come input ad un ottimizzatore di codice, che si occupa (applicando ottimizzazioni indipendenti o dipendenti dalla macchina) di migliorare l'efficienza del codice.

Linking

È il processo che consente di **unire** più moduli oggetto in un unico file eseguibile. Il **linker**, lo strumento che si occupa di svolgere questa operazione, unisce i vari moduli, assegna indirizzi contigui partendo da indirizzi simbolici e risolve ogni riferimento esterno, calcolando l'indirizzo di memoria della risorsa richiesta.