**Il modello di un compilatore**

La costruzione di un compilatore per un particolare linguaggio di programmazione è abbastanza complessa. La complessità dipende dal linguaggio sorgente. Il compilatore ha la funzione di tradurre il programma sorgente in programma oggetto. Esso esegue:

* L’analisi del programma sorgente
* La sintesi del programma oggetto

Il programma sorgente è una stringa di simboli. Viene dato come input a un analizzatore lessicale o scanner che esamina il programma per individuare i simboli, detti token, che lo compongono, classificando le parole chiave, gli identificatori, gli operatori, le costanti, ecc. Per ragioni di efficienza ad ogni classe di token è dato un numero unico che la identifica. L’output dell’analizzatore lessicale è una lista di token. Vengono ignorati gli spazi bianchi e i commenti. Inoltre alcuni scanner inseriscono label, costanti e variabili in tavole appropriate. Un elemento della tavola per una variabile, ad esempio, contiene il nome, il tipo, l’indirizzo, il valore e la linea in cui è dichiarata. Segue poi l’analizzatore sintattico o parser. Il quale ha in input una lista di token, individua la struttura sintattica della stringa in esame a partire dal programma sorgente sotto forma di token, per poi identificare espressioni, istruzioni e procedure. Il controllo sintattico si basa sulle regole grammaticali utilizzate per definire formalmente il linguaggio. Durante il controllo sintattico si genera l’albero di derivazione, detto albero sintattico.

Durante la generazione del codice oggetto si effettua anche l’analisi semantica. L’analizzatore semantico ha come ingresso l’albero sintattico generato dal parser. Si compone di due fasi principali:

* Controlli statici: vengono svolti vari controlli sui tipi, dichiarazioni ecc. deve determinare quali azioni sono specificate dagli operatori aritmetici di addizione e moltiplicazione.
* Generazione di una rappresentazione intermedia (IR). Spesso la parte di analisi semantica produce anche una forma intermedia di codice sorgente.

Spesso a valle dell’analizzatore semantico ci può essere un ottimizzatore del codice intermedio attraverso la propagazione di costanti ed eliminando sotto-espressioni comuni.

L’output dell’analizzatore semantico è passato al generatore di codice che trasla la forma intermedia in linguaggio assembler o macchina. Prima della generazione del codice oggetto ci sono delle fasi di preparazione:

* Allocazione della memoria: può essere allocata staticamente oppure è uno stack o heap la cui dimensione cambia durante l’esecuzione;
* Allocazione dei registri: poiché l’accesso ai registri è più rapido dell’accesso alle locazioni di memoria, i valori cui si accede più spesso andrebbero mantenuti nei registri.

L’output del generatore di codice è passato in input all’ottimizzatore di codice, presente nei compilatori più sofisticati. Esistono ottimizzazioni indipendenti dalla macchina sia ottimizzazioni dipendenti dalla macchina.

Molte fasi espresse sopra in maniera separate possono essere combinate tra di loro. Ad esempio lo scanner e il parser possono essere eseguiti in sequenza uno dopo l’altro, producendo prima tutti i token e poi l’analisi sintattica, oppure lo scanner è chiamato dal parser ogni volta che necessita un nuovo token. Nel primo caso lo scanner ha esaminato l’intero programma sorgente prima di passare il controllo al parser e quindi ha compiuto un intero passo separato. A volte, invece, il parser, l’analizzatore semantico e il generatore di codice sono combinati in un singolo passo.

Altri aspetti molto importanti della compilazione sono:

1. Error Detection e Recovery;
2. Le tabelle dei simboli prodotte dai vari moduli;
3. La gestione della memoria implicata da costrutti del linguaggio ad alto livello

Le fasi più semplici della progettazione, con un apparato formale ben sviluppato e quindi facilmente automatizzabile sono lo scanner e il parser, mentre maggiore difficoltà si trova nella progettazione di analizzatori semantici, generatori ed ottimizzatori di codice.

Il programma oggetto prodotto dal compilatore contiene una serie di riferimenti esterni che vengono risolti dal linker. Il programma diviene rilocabile: cioè può essere rilocabile, cioè può essere allocato in zone di memoria cambiando indirizzo ind (indirizzamento relativo). Segue poi la fase di caricamento compiuta dal loader che assegna un valore numerico all’indirizzo ind, trasformando gli indirizzi relativi in assoluti. Il linker ricever in ingresso uno o più moduli e genera un unico modulo con riferimento ad indirizzi contigui a partire da un indirizzo simbolico ind. Ogni riferimento a moduli esterni viene sostituito con l’indirizzo che viene calcolato.