

## TABELLA DEI SIMBOLI

Le tabelle dei simboli servono a due scopi: controllano la correttezza semantica (la parte dipendente dal contesto della grammatica) ed aiutano nella generazione del codice.

Tutte queste attività si ottengono mediante accesso alla tabella dei simboli per inserire o leggere alcuni attributi delle variabili del programma sorgente.

Questi attributi sono *tipo*, *nome*, *dimensione*, *indirizzo* e si determinano direttamente dalla dichiarazione o implicitamente dal contesto in cui le variabili compaiono.

Qui discuteremo l'organizzazione della tabella dei simboli e la modalità per creare ed accedere simboli.

La tabella dei simboli e' inserita in memoria centrale e viene normalmente modificata dinamicamente.

Ogni identificatore del programma sorgente richiede da parte del compilatore accessi alla tabella dei simboli. Questo spiega perche' l'accesso alla tabella dei simboli sia una delle parti piu' costose in tempo del processo di compilazione e si debba cercare di ottimizzare.

Quindi e' importane studiare metodi efficienti per l'accesso alla tabella dei simboli.

## Strutture dati: tavole o tabelle

Una tavola o tabella e' un tipo di dato astratto per rappresentare insiemi di coppie <*chiave*, *attributi*>. Ciascuna coppia rappresenta dati riferiti ad un'unica entità logica (ad es., persona, documento, etc.) identificata in modo univoco dalla *chiave*.

### Esempio:

Matricola, Cognome, Nome, DataDiNascita, ...

Operazioni tipiche sulle tavole:

- inserimento di un elemento <Chiave, Attributi>  
*inserisci: tavola x chiave x attributi → tavola*
- cancellazione di un elemento (nota la chiave)  
*cancella: tavola x chiave → tavola*
- verifica di appartenenza di un elemento  
*esiste: tavola x chiave → boolean*
- ricerca di un elemento nella tavola  
*ricerca: tavola x chiave → attributi*

L'operazione di ricerca e' la piu' importante (soprattutto nelle TS dei compilatori). Spesso la rappresentazione concreta viene scelta in modo da ottimizzare questa operazione.

In Pascal, ciascun elemento della tavola rappresenta una *aggregazione* di dati ---> **tipo record**.  
La chiave e ciascun attributo corrispondono ad un *campo del record*.

## Quando costruire ed interagire con la tabella dei simboli

In un compilatore a molti passi, la tabella dei simboli (TS) viene creata durante l'analisi lessicale.

Gli indici degli entries per le variabili nella TS formano parte della stringa di tokens prodotta dallo scanner.

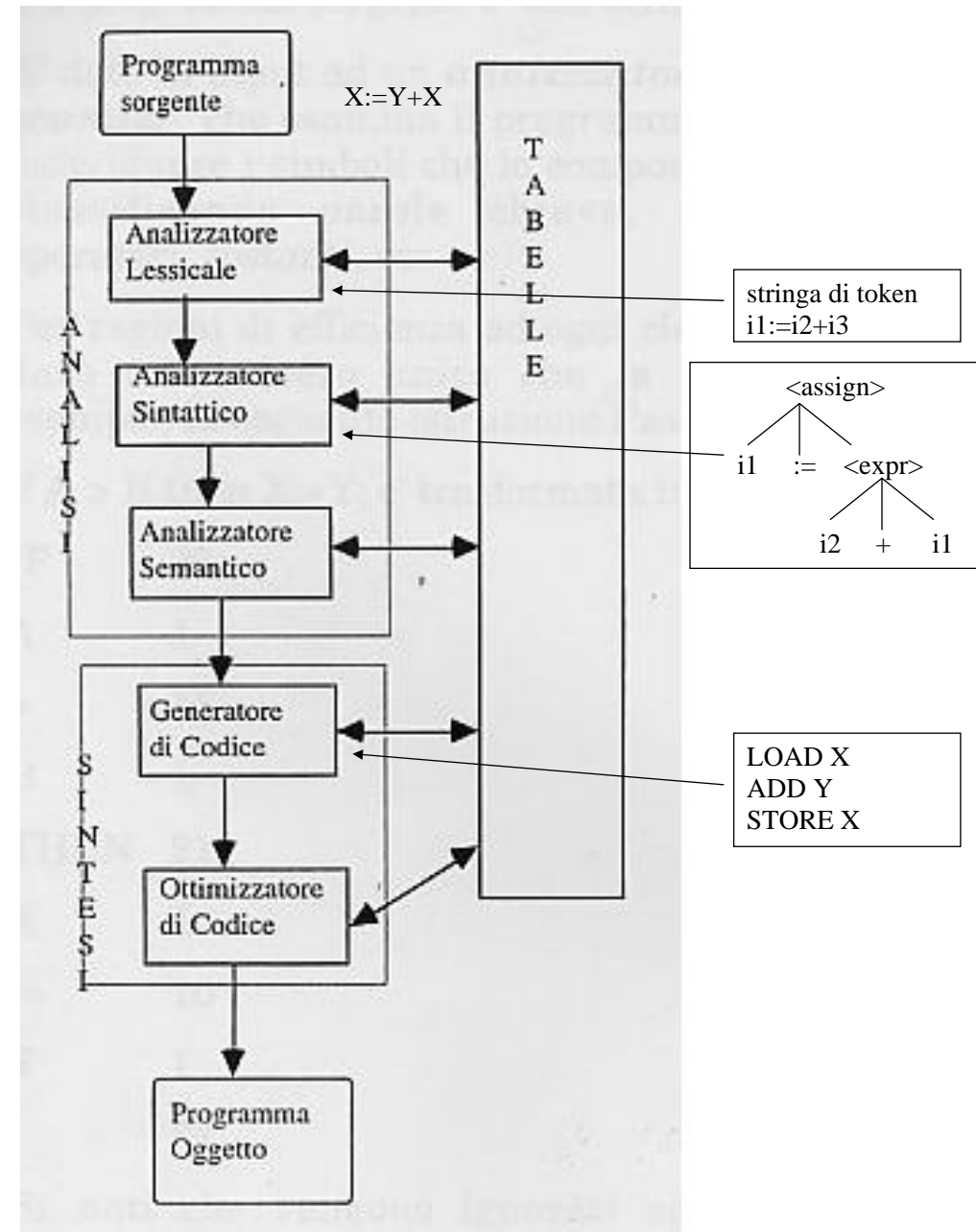
Nella figura seguente se X e Y occupano le posizioni 1 e 2 nella TS si generano i tokens `i1` e `i2`.

Quando il parser produce l'albero sintattico, le foglie hanno dei riferimenti alla tabella dei simboli.

L'analisi sintattica comunque non produce modifiche a TS.

Solo durante l'analisi semantica e la generazione di codice si possono assegnare valori agli attributi della variabile in TS.

Si pensi alla dichiarazione esplicita di tipo.



## Contenuti della TS

Una TS e' costituita da una serie di righe ognuna delle quali contiene una lista di valori di attributi associati con una particolare variabile.

Gli attributi dipendono dal linguaggio di programmazione che si compila (se non ha i tipi non comparirà tale attributo).

Variable Name	Address	Type	Dimension	Line Declared	Lines Referenced	Pointer
COMPANY	0	2	1	2	9,14,25	7
X3	4	1	0	3	12,14	0
FORM1	8	3	2	4	36,37,38	6
B	48	1	0	5	10,11,13,23	1
ANS	52	1	0	5	11,23,25	4
M	56	6	0	6	17,21	2
FIRST	64	1	0	7	28,29,30,38	3

Si possono considerare i seguenti attributi:

1. Nome della variabile;
2. Indirizzo nel codice oggetto;
3. Tipo;
4. Numero dei parametri di una procedura (o dimensione della variabile);
5. Linea sorgente in cui la variabile e' dichiarata;
6. Linee sorgenti in cui la variabile e' referenziata;
7. Puntatori per listarli in ordine alfabetico.

Ora commentiamo tali attributi.

1. Un problema può essere la dimensione della stringa che compone il nome. E' inserito dallo scanner.
2. L'indirizzo indica la locazione relativa delle variabili a run-time. Questo indirizzo e' inserito nella TS quando una variabile è dichiarata o incontrata per la prima volta. E' richiamato dalla tabella quando la variabile e' referenziata nel codice per generare il codice oggetto corrispondente.

Per un linguaggio senza allocazione dinamica di memoria (es. FORTRAN) gli indirizzi sono attribuiti in modo sequenziale da 1 a M dove M e' la massima dimensione di memoria allocata per il programma.

Per linguaggi a blocchi, l'indirizzo e' rappresentato da una coppia <BL (block level), ON (occurrence number)>. A run-time ON rappresenta l'offset rispetto al blocco.

3. Il tipo può essere implicito (es. FORTRAN), esplicito (es. PASCAL) o non esistere (es. LISP, PROLOG). Il tipo della variabile e' fondamentale per il controllo semantico. Ad esempio, S\*3 e' scorretto se S e' dichiarato come una variabile di tipo stringa.

Il tipo serve anche per sapere quante memoria deve essere allocata per la variabile. Il tipo e' inserito normalmente con una codifica (numero intero).

4. La dimensione e' importante per il controllo semantico. Se si tratta di un array ci dice le sue dimensioni e serve anche per calcolare l'indirizzo di un particolare elemento dell'array. In una procedura indichiamo il numero dei parametri per procedere al controllo durante le chiamate. Negli esempi dati qui gli scalari sono considerati di dimensione 0, vettori 1, matrici 2, ecc.
5. Il Pointer serve per costruire, ad esempio, una lista per ordine alfabetico della tabella dei simboli od una cross-reference (con i riferimenti di dove e' dichiarata e referenziata nel codice sorgente).

## **Operazioni sulla tabella dei simboli**

Le operazioni piu' comuni sono l'**inserimento** e la **ricerca**.

Se il linguaggio ha dichiarazioni esplicite di tutte le variabili allora l'inserimento avviene al momento della dichiarazione.

Se la tabella dei simboli è ordinata, ad esempio, alfabeticamente mediante il nome della variabile si deve fare anche una ricerca nella tabella e quindi diventa un procedimento costoso.

Se invece la tabella e' disordinata l'inserimento e' rapido, ma diventa poi più inefficiente la ricerca.

Le operazioni di ricerca vengono svolte per ogni riferimento a variabili in istruzioni non di dichiarazione.

L'accesso serve per il controllo semantico e la generazione di codice, nonché per la rilevazione di errori se le variabili non si trovano all'interno della tabella.

Quando un linguaggio ha la possibilità di dichiarare variabili implicitamente allora le operazioni di inserimento e ricerca sono legate strettamente.

Ogni riferimento ad una variabile genera una ricerca ed eventualmente un inserimento se non e' già stata inserita in TS.

## **Linguaggi a blocchi**

Per linguaggi a blocchi sono necessarie due operazioni addizionali che chiamiamo **set** e **reset**.

*Set* si invoca quando si entra in un blocco, *reset* quando si esce.

Questo e' necessario perché in un linguaggio a blocchi, una variabile con lo stesso nome può essere dichiarata in punti diversi del programma ed assumere attributi diversi.

In un linguaggio innestato e' importante assicurare che per ogni istanza di un nome di variabile sia associato un unico entry nella TS.

Quindi ad ogni inizio di un blocco l'operazione di set attribuisce una nuova sotto tabella per gli attributi delle variabili dichiarate nel nuovo blocco.

Supponiamo che le sottotavole siano create attribuendole numeri interi crescenti.

Se la ricerca comincia nella sottotavola N e poi procede fino alla sottotavola 1, la variabile che si trova e' l'ultima che si e' inserita (secondo le regole di scope di linguaggi a blocchi) ed e' eliminata ogni ambiguità.

All'uscita del blocco, l'operazione di reset rimuove l'ultima sottotabella allocata. Le variabili di quel blocco, infatti, non possono più essere referenziate.

## **Organizzazione delle tabelle dei simboli per linguaggi non a blocchi**

Intendiamo linguaggi in cui ogni singola unita' di compilazione e' un modulo senza alcun sottomodulo.

Tutte le variabili dichiarate nel modulo possono essere referenziate in ogni parte del modulo.

### **Tabelle dei simboli non ordinate**

Il modo piu' semplice di organizzare una tabella dei simboli e' quello di aggiungere gli entries alla tabella nell'ordine in cui le variabili sono dichiarate.

In questo modo per un inserimento non e' richiesto nessun confronto, mentre una ricerca richiede, nel caso peggiore, il confronto con gli N elementi all'interno della tabella.

Poiché ciò e' inefficiente, questa organizzazione dovrebbe essere adottata solo se la dimensione della TS e' piccola.

Altrimenti si deve ricorrere ad altre organizzazioni.

## **Tabelle dei simboli ordinate**

La posizione dell'elemento nella TS e' determinato dal nome della variabile (ordinamento lessicale).

In questo caso un inserimento e' sempre accompagnato ad una procedura di ricerca.

L'inserimento può inoltre richiedere uno spostamento di altri elementi già inseriti nella tabella (e' la maggiore fonte di inefficienza).

**Ottimizzazione della ricerca:** si ottiene se gli elementi sono memorizzati in modo ordinato nella tavola.

Deve esistere un ordinamento sul campo Chiave.  
Questo induce un ordinamento sugli elementi della Tavola come segue:

$el_1 = \langle k_1, Attr_1 \rangle$

$el_2 = \langle k_2, Attr_2 \rangle$

$el_1 < el_2$  *se e solo se*  $k_1 < k_2$

La ricerca può arrestarsi appena si incontra un elemento con chiave maggiore di quella cercata (caso medio  $N/2$  confronti, caso peggiore  $N$  confronti).

Si complica l'operazione di inserimento di un nuovo elemento nella tavola → occorre mantenerla ordinata.

Miglioramento ulteriore: *ricerca binaria*.

Si accede all'elemento mediano  $\langle Chiave, Attr \rangle$  della tavola.

Se Chiave= $k$ , fine della ricerca

altrimenti,

se  $k < Chiave$ , ripeti la ricerca nella prima metà della tavola;

se  $k > Chiave$ , ripeti la ricerca nella seconda metà della tavola.

Ad ogni passo si eliminano dalla ricerca metà degli elementi della tavola corrente.

Nel caso peggiore si eseguono  $\log_2 N$  confronti.

### **Tavole: rappresentazione collegata (liste linkate)**

Il tempo per inserire un elemento in una TS ordinata si può ridurre utilizzando una struttura collegata od ad albero.

Nel caso di *rappresentazione collegata* di una tavola, si utilizza una lista semplice:

Idea fondamentale: memorizzare gli elementi associando a ciascuno una particolare informazione (*riferimento*) che permetta di individuare la locazione in cui e' inserito l'elemento successivo.

La sequenzialità degli elementi della lista non e' rappresentata mediante l'adiacenza delle locazioni di memoria in cui sono memorizzati.

Notazione grafica per rappresentazione collegata: elementi della lista come nodi e riferimenti come archi.

Non c'è un limite massimo alla dimensione della tavola.

Lo spazio di memoria occupato dipende solo dal numero di elementi della tavola.

### **Altre rappresentazioni per le tabelle dei simboli:**

- Alberi binari
- Alberi binari di ricerca
- A funzioni di accesso

### **Linguaggi strutturati a blocchi: organizzazione della tabella dei simboli**

Il linguaggio può contenere moduli (blocchi) innestati ciascuno contenente delle variabili locali.

### **Struttura a blocchi, regole di visibilità degli identificatori e tempo di vita:**

Abbiamo già detto che tutti gli identificatori devono essere dichiarati prima del loro uso.

Identificatori, nomi per indicare costanti, variabili, tipi, unità di programma definiti dall'utente.

Il significato della dichiarazione e' quello di associare - durante l'attivazione di un'unità di programma (main o sottoprogramma) - varie informazioni all'identificatore (**ambiente**). Ad esempio, tipo della variabile, l'indirizzo del codice eseguibile per il nome di una procedura, etc.

Il tempo di vita di una di queste associazioni e' la durata dell'attivazione dell'unità di programma in cui compare la dichiarazione dell'identificatore.

L'effetto di una dichiarazione perdura per tutto il tempo di attivazione dell'unità di programma in cui tale dichiarazione si trova.

## Regole di visibilita' degli identificatori

In Pascal, il campo di azione per gli identificatori segue le seguenti regole:

1. il campo di azione della dichiarazione di un identificatore è il blocco (unità di programma) in cui essa compare e tutti i blocchi in esso contenuti, a meno della regola 2;
2. quando un identificatore dichiarato in un blocco P è ridichiarato in un blocco Q, racchiuso da P, allora il blocco Q, e tutti i blocchi innestati in Q, sono esclusi dal campo di azione della dichiarazione dell'identificatore in P.

Il campo di azione è determinato staticamente, dalla struttura del testo del programma (**regole di visibilità lessicali**).

## TS a Stack

E' l'organizzazione più semplice per un linguaggio a blocchi.

I record che contengono gli attributi delle variabili di un blocco B1 vengono messi nello stack quando si incontrano le corrispondenti dichiarazioni (push) ed eliminati al termine del blocco B1 (pop) .

L'operazione di ricerca per un simbolo parte dal top dello stack e quindi garantisce che i simboli più innestati siano trovati per primi (anche se ne esistessero più occorrenze).

L'operazione di set, salva il contenuto di top nello stack degli indici, mentre l'operazione di reset lo ripristina, cancellando implicitamente tutti i valori non più referenziati.

L'organizzazione si può rendere più efficiente introducendo nello stack rappresentazioni più sofisticate quali quelle ad albero o con funzione di accesso hash.