**Tabella dei simboli**

Le tabelle dei simboli servono a due scopi:

* Controllo della certezza semantica (la parte dipendente dal contesto della grammatica)
* Aiuto 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. La tabella dei simboli è 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 perché l’accesso alla tabella dei simboli sia una delle parti più costose in tempo del processo di compilazione e si debba cercare di ottimizzare. Vi sono differenti metodi efficienti per l’accesso alla tabella dei simboli.

**Strutture dati: tavole o tabelle**

Una tavola o tabella è un tipo di dato astratto per rappresentare insiemi di coppie <chiave, attributi>. Ciascuna coppia rappresenta dati riferiti ad un’unica entità logica identificata in modo univoco dalla chiave. Vi sono diverse operazioni che si possono effettuare sulle tavole:

* Inserimento di un elemento
* Cancellazione di un elemento
* Verifica di appartenenza di un elemento
* Ricerca di un elemento nella tavola

L’operazione di ricerca è la più importante. Spesso la rappresentazione concreta viene scelta in modo da ottimizzare questa operazione. La chiave e ciascun attributo corrispondono ad un campo del record. In un compilatore a molti passi, la tabella dei simboli viene creata durante l’analisi lessicale. Gli indici delle entries per le variabili nella TS formano parte della stringa di tokens prodotta dallo scanner. 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 tabella dei simboli**

Una TS è 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. Si possono considerare i seguenti attributi:

* Nome della variabile
* Indirizzo nel codice oggetto
* Tipo
* Numero dei parametri di una procedura (o dimensione della variabile)
* Linea sorgente in cui la variabile è dichiarata
* Linee sorgenti in cui la variabile è referenziata
* Puntatori per listarli in ordine alfabetico

Il nome è inserito dallo scanner e potrebbe essere un problema la dimensione della stringa che lo compone. L’indirizzo indica la locazione relativa delle variabili a run-time. Questo indirizzo è inserito nella TS quando una variabile è dichiarata o incontrata per la prima volta. È richiamato dalla tabella quando la variabile è referenziata nel codice oggetto corrispondente. Per un linguaggio senza allocazione dinamica di memoria, gli indirizzi sono attribuiti in modo sequenziale da 1 a M dove M è la massima dimensione di memoria allocata per il programma. Invece per i linguaggi a blocchi, l’indirizzo è rappresentato da una coppia. A run-time on rappresenta l’offset rispetto al blocco. Il tipo può essere implicito, esplicito o non esistere. Il tipo della variabile è fondamentale per il controllo semantico. Il tipo serve anche sapere quanta memoria deve essere allocata per la variabile. Il tipo è inserito normalmente con una codifica. La dimensione è 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. Il pointer serve per costruire, ad esempio una lista per ordine alfabetico della tabella dei simboli o una cross reference.

**Operazioni sulla tabella dei simboli**

Le operazioni più 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, si deve fare anche una ricerca nella tabella e quindi diventa un procedimento costoso. Se invece la tabella è disordinata l’inserimento è 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 è già stata inserita in TS.

**Linguaggi a blocchi**

Per i 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 è necessario perché in un linguaggio a blocchi, una variabile con lo stesso nome può essere dichiarata in punti diversi del programma e assumere attributi diversi. In un linguaggio innestato è 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 è l’ultima che si è inserita ed è 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 unità di compilazione è 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 modulo più semplice per organizzare una tabella dei simboli è quello di aggiungere le entries alla tabella nell’ordine in cui le variabili sono dichiarate. In questo modo per un inserimento non è richiesto nessun confronto, mentre una ricerca richiede, nel caso peggiore, il confronto con gli N elementi all’interno della tabella. Poiché questo è inefficiente, questa organizzazione dovrebbe essere adottata solo se la dimensione della TS è piccola. Altrimenti si deve ricorrere ad altre organizzazioni.

*Tabelle dei simboli ordinate*

La posizione dell’elemento nella TS è determinata da nome della variabile (ordinamento lessicale). In questo caso un inserimento è sempre accompagnato ad una procedura di ricerca. L’inserimento può inoltre richiedere uno spostamento di altri elementi già inseriti nella tabella. Ottimizzazione della ricerca: si ottiene se gli elementi sono memorizzati in modo orinato nella tavola. Deve esistere un ordinamento sul campo chiave. 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, perché occorre mantenerla ordinata. Un miglioramento ulteriore si può effettuare utilizzando la ricerca binaria. Si accede all’elemento mediano della tavola. Se Chiave=k fine della ricerca. Altrimenti, se k<Chiave, ripeti la ricerca nella prima metà della tavola, invece se k>Chiave, ripeti la ricerca nella seconda metà della tavola. Il procedimento si arresta quando i=J. Ad ogni passo si eliminano dalla ricerca metà degli elementi della tavola corrente. Nel caso peggiore si eseguono log2N confronti.

**Tavole: rappresentazione collegata e ad albero**

Il tempo per inserire un elemento in una TS ordinata si può ridurre utilizzando una struttura collegata o ad albero. Nel caso di rappresentazione collegata di una tavola, si utilizza una lista semplice. L’idea fondamentale: memorizzare gli elementi associando a ciascuno una particolare informazione (riferimento) che permetta di individuare la locazione in cui è inserito l’elemento successivo. La sequenzialità degli elementi della list non è rappresentata mediante l’adiacenza delle locazioni di memoria in cui sono memorizzati. Non c’è un limite massimo alla dimensione della tavola. Lo spazio di memoria occupato dipende solo dal numero di elementi della tavola. Se la lista è ordinata sulla chiave, si può applicare la ricerca ordinata.

*Tabelle dei simboli strutturate ad albero*

*Alberi binari*

Albero binario ordinato in cui ogni nodo ha al massimo due figli (figlio destro e figlio sinistro.

Definizione induttiva:

un albero binario è un grafo orientato aciclico che o è vuoto, oppure è formato da un nodo radice e da due sotto-alberi binari. Rappresentazione più efficiente direttamente attraverso record e pointer. Ogni record, detto nodo, ha tre campi: uno per l’informazione associata al nodo, uno per il puntatore al sottoalbero sinistro e uno per il puntatore al sottoalbero destro.

I vantaggi sono molteplici. Non è necessario che l’albero sia completo, si ha un’occupazione di memoria efficiente e operare modifiche è agevole

Vi sono diversi algoritmi di visita per gli alberi binari che consentono di analizzare tutti i nodi dell’albero in un determinato ordine. L’algoritmo di preordine o ordine anticipato analizza prima la radice, poi l’albero sinistro e infine l’albero destro. L’algoritmo di postordine o ordine ritardato analizza prima l’albero sinistro, poi l’albero destro e infine analizza la radice. L’algoritmo simmetrico analizza l’albero sinistro, la radice e infine l’albero destro.

*Alberi binari di ricerca*

Sono alberi binari ordinati utilizzati per memorizzare grosse quantità di dati su cui si esegue spesso un’operazione di ricerca di un dato. In un albero binario di ricerca, ogni nodo N ha la seguente proprietà:

* Tutti i nodi del sotto-albero sinistro di N hanno un valore minore o uguale a quello di N
* Tutti i nodi del sottoalbero destro di N hanno un valore maggiore di quello di N.

Ricerca binaria in un albero di ricerca:

il numero di confronti è proporzionale alla profondità dell’albero. Ad ogni passo si dimezza il problema, eliminando metà dei nodi.

*Alberi binari di ricerca: inserzione con bilanciamento*

Nodo perfettamente bilanciato: l’altezza dei suoi due sottoalberi è la stessa

Nodo bilanciato a sinistra: height(left(n)) =height(right(n)) +1

Nodo bilanciato a destra: height(left(n)) =height(right(n))-1

Albero bilanciato in altezza: quando l’altezza del sottoalbero sinistro e destro di ogni nodo differisce al più di 1.

**Tavole: rappresentazione a funzione di accesso**

Obbiettivo: permettere il recupero con pochi accessi di una registrazione, nota la sua chiave.

Idea-base: definire una qualche forma di corrispondenza fra i valori delle chiavi primarie e le posizioni delle registrazioni nell’archivio in modo da rintracciare velocemente la registrazione richiesta.

Ogni elemento è memorizzato in una posizione che dipende dal valore della chiave. Si utilizza una funzione di accesso biunivoca che data la chiave k restituisce il numero il corrispondente a k. Occorre una locazione di memoria per ciascuna possibile chiave. L’accesso è molto efficiente: data la chiave k, l’elemento cercato è all’indirizzo ind(f(k) nello spazio di memoria riservato alla tavola. Anche l’inserimento e la cancellazione sono realizzati in modo efficiente. L’utilizzo di una funzione di accesso biunivoca è possibile in pratica solo se il numero di elementi della tavola è circa uguale al numero di chiavi possibili. In alternativa la funzione può calcolare lo stesso indirizzo per chiavi distinte: si possono verificare collisioni. In questo caso occorre:

* Scegliere una funzione di accesso che riduca il più possibile le collisioni;
* In caso di collisione, determinare un metodo di scansione della tavola per ricerca o inserimento

**Tabelle Hash:**

Funzioni hash (tritare, mescolare), perché l’indirizzo calcolato dipende da tuta la chiave, eventualmente spezzata in parti rimescolate. Ad esempio, consideriamo una chiave di 15 caratteri. Sia k la sua rappresentazione binaria: si calcola il resto della divisione di k per il massimo numero di elementi N previsto per la tavola. Le chiavi sono partizionate in classi di equivalenza. Ogni classe contiene chiavi di ugual lunghezza che iniziano con la stessa lettera. Le chiavi che appartengono a una stessa classe di equivalenza collidono.

*Collisioni*

La funzione di hash si dice perfetta se e solo se non produce collisioni. Evitare le collisioni non è facile. Tuttavia l’overflow può essere evitato quando si riesce a stabilire una corrispondenza biunivoca fra l’insieme delle possibili chiavi e l’insieme delle posizioni disponibili nell’area di memorizzazione. In generale, però, le chiavi possibili sono molto più di quelle realmente usate, e ciò rende questo approccio impossibile da adottare perché drammaticamente inefficiente. Per densità di chiavi attive si intende il rapporto tra N/M, dove N è il numero di registrazioni da archiviare e M la cardinalità dell’insieme delle chiavi. Se le chiavi sono stringhe lunghe L definite su un alfabeto di V simboli, la cardinalità M vale VL.

*Gestioni delle collisioni*

Vi sono diversi metodi di scansione nel caso in cui si verificano delle collisioni. Si può procedere attraverso una scansione lineare. Dato un h prefissato, si passa a considerare per k2 l’indirizzo i+h, poi i+2\*h, i+3\*h, …

In alternativa si può utilizzare una scansione con aree di trabocco rappresentate attraverso una lista. Altri metodi che si possono utilizzare sono gli alberi e le tabelle Hash. Può esserci anche un’unica area di trabocco per tutta la tavola.

**La funzione di Hash**

Una buona funzione di hash deve assicurare: una distribuzione uniforme delle chiavi nello spazio degli indirizzi, una distribuzione casuale delle chiavi (cioè tale che valori vicini di chiave non finiscano in indirizzi vicini). Per creare una funzione di hash si possono utilizzare diversi metodi:

1. Metodo del modulo: H(k)= k mod R, con R= massimo numero primo minore o uguale a P (o anche semplicemente numero non primo minore o uguale a P, purché non abbia fattori primi minori di 20)
2. Metodo della somma: H(k)= (p1+p2+…pS) mod 10h, dove p1…pS sono pezzi della chiave k stessa, che viene suddivisa in parti fatte da un egual numero h di caratteri (esclusa eventualmente l’ultima), pari all’ampiezza dell’indirizzo da generare. Fatta la somma, si ignora la cifra più alta (riporto).
3. Metodo dell’EX-OR: come il precedente metodo, con l’0perazione di EX-OR (fatta sulle corrispondenti rappresentazioni binarie) al posto della somma.
4. Metodo del quadrato: H(k)= parte\_centrale(k2), in pratica, si moltiplica k per sé stessa, e si prendono le h cifre centrali del numero ottenuto.

Vantaggi e svantaggi dell’approccio hash:

* Basso costo medio di ricerca, con elevate prestazioni, specialmente con un’organizzazione dinamica
* Semplicità di realizzazione
* Non si prestano a una visita dei dati in ordine diverso da quello fisico
* Non sono adatte a reperire un sottoinsieme dei dati con chiave primaria che soddisfatti una relazione specificata
* Costo medio delle principali operazioni basso, ma costo di caso peggiore nettamente più alto

Linguaggi strutturati a blocchi: organizzazione della tabella dei simboli

Il linguaggio può contenere moduli innestati ciascuno contenente delle variabili locali

Struttura a blocchi, regole di visibilità degli identificatori e tempo di vita: tutti gli identificatori devono essere dichiarati prima del loro utilizzo. Il significato della dichiarazione è quello di associare, durante l’attivazione di un’unità di programma, varie informazioni all’identificatore. Ad esempio, tipo di variabili, l’indirizzo del codice eseguibile per il nome di una procedura, etc.

Il tempo di vita di una di queste associazioni p 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 promma in cui tale dichiarazione si trova.

**Regole di visiblità degli identificatori**

Le regole di visibilità vengono definite dal programma e variano da programma a programma. Esistono due ambiti di visibilità (scope) di un identificatore comuni a tutti i linguaggi di programmazione object-oriented:

A livello di blocco. Caratterizza le variabili locali; sono gli identificatori dichiarati in un blocco. Lo scope inizia dalla dichiarazione dell'identificatore e termina con la fine del blocco stesso. Le variabili locali dichiarate all'interno di una funzione, così come i parametri di una funzione, hanno visibilità a livello di blocco. Nel caso di blocchi nidificati, se un identificatore del blocco esterno ha lo stesso nome di quello del blocco interno, l'identificatore del blocco esterno viene occultato fino alla fine del blocco più interno secondo la regola dello [shadowing](https://it.wikipedia.org/wiki/Shadowing). Il blocco interno vede solamente il proprio identificatore locale.

A livello di file. Caratterizza le variabili globali; un identificatore dichiarato all'esterno di una qualsiasi funzione ha visibilità a livello di file. È noto a tutte le funzioni che si trovano dopo la sua dichiarazione.

TS a Stack

È l’organizzazione più semplice per un linguaggio a blocchi. I record che contengono gli attributi delle variabili di un blocco vengono messi nello stack quando si incontrano le corrispondenti dichiarazioni ed eliminati al termine del blocco. L’operazione di ricerca per un simbolo parte dal top dello stack, e quindi garantisce che i simboli più innesti 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 ad albero o con funzione di accesso hash.