8 - Tabella dei simboli

Il compilatore traduce un **programma sorgente** in un **programma oggetto**.
Si articola in due fasi principali:

1. **Analisi**: trasforma il sorgente in una rappresentazione intermedia e comprende:
   - **Analisi lessicale (scanner)**: riconosce token (identificatori, parole chiave, operatori, costanti) e costruisce la tabella dei simboli.
   - **Analisi sintattica (parser):** verifica le regole grammaticali e costruisce l'albero sintattico.
   - **Analisi semantica**: controlla vincoli di contesto (tipi, dichiarazioni, compatibilità) e produce rappresentazione intermedia (IR).
2. **Sintesi**: genera e ottimizza il codice oggetto e comprende:
   - **Ottimizzazione intermedia**: riduce ridondanze (sotto-espressioni comuni o propagazioni di costanti) migliorando l'efficienza senza modificare la semantica
   - **Generazione del codice oggetto**: traduce la rappresentazione intermedia in linguaggio assembler o macchina, allocando registri e memoria.
   - **Ottimizzazione finale (opzionale)**: ottimizzazioni dipendenti/indipendenti dalla macchina.
3. **Programma oggeto**: Infine si ha il programma finale, che esegue le operazioni di:
   - **Linking:** unisce il codice oggetto con librerie e moduli esterni, risolvendo i riferimenti.
   - **Loading:** carica il programma eseguibile in memoria, trasformando gli indirizzi relativi in assoluti.

Ogni riga della TS contiene **attributi** legati a una variabile. Gli attributi possono variare in base al linguaggio, ma generalmente includono:

4. **Nome della variabile** – può essere di lunghezza variabile, spesso gestita dallo scanner.
5. **Indirizzo** – la posizione della variabile nella memoria a run-time. Nei linguaggi senza allocazione dinamica (es. FORTRAN), questo è sequenziale; nei linguaggi a blocchi può essere rappresentato come coppia `<livello di blocco, offset>`.
6. **Tipo** – può essere implicito (FORTRAN), esplicito (PASCAL), o assente (LISP). Determina il controllo semantico e la quantità di memoria necessaria.
7. **Dimensione** – serve per array, matrici, o numero di parametri di una procedura. Ad esempio, un array avrà dimensione 1, una matrice 2.
8. **Linea di dichiarazione**.
9. **Linee di riferimento** – dove la variabile viene utilizzata nel codice.
10. **Puntatore** – usato per ordinamenti (es. ordine alfabetico) o per generare cross-reference.

Le operazioni centrali sono **inserimento** e **ricerca**. Se il linguaggio richiede dichiarazioni esplicite, l'inserimento avviene durante l'elaborazione delle dichiarazioni. Se la tabella è ordinata (per esempio per nome), ogni inserimento implica una ricerca e possibile spostamento degli elementi per mantenere l'ordine. Se disordinata, l'inserimento è rapido ma la ricerca diventa costosa.

In linguaggi con **dichiarazioni implicite**, inserimento e ricerca si intrecciano: ogni riferimento a una variabile comporta prima una ricerca, seguita da inserimento se la variabile non è ancora presente.

Nei linguaggi a blocchi (come Pascal o C), variabili con lo stesso nome possono esistere in blocchi annidati. Servono quindi due operazioni:

- **Set**: entra in un nuovo blocco, inizializza una nuova sotto-tabella.
- **Reset**: esce da un blocco, rimuove la relativa sotto-tabella.
  La **ricerca** inizia dalla sotto-tabella più interna, risolvendo correttamente l'ambiguità con le regole di scope. Alla fine del blocco, le variabili locali non sono più visibili e vengono eliminate.