PROGRAMMAZIONE INFORMATICA

LINGUAGGI DI PROGRAMMAZIONE

LINGUAGGIO DI PROGRAMMAZIONE

Un **linguaggio di programmazione** è un insieme strutturato di regole e simboli che permettono ai programmatori di scrivere istruzioni comprensibili da una macchina, ovvero il computer. L'obiettivo di questi linguaggi è rendere più facile e intuitivo il processo di creazione di programmi che realizzano operazioni specifiche.

Le istruzioni sono tradotte in linguaggio macchina, il quale può essere eseguito direttamente dalla CPU. Tuttavia, senza un linguaggio di programmazione, i programmatori dovrebbero scrivere i programmi usando solo sequenze di 0 e 1, rendendo il compito estremamente complesso.

COMPONENTI FONDAMENTALI

- **Sintassi**: **regole** e **strutture** che determinano come devono essere scritte le istruzioni del linguaggio. In altre parole, la sintassi definisce la forma e la struttura dei programmi, specificando come i simboli, le parole chiave e le espressioni possono essere combinati per formare istruzioni valide.

- **Semantica**: **significato** delle istruzioni e delle espressioni. In altre parole, la semantica stabilisce cosa fa il codice quando viene eseguito e quale è il suo comportamento. Mentre la sintassi si occupa della forma, la semantica si occupa del contenuto.

- **Struttura del linguaggio**: Molti linguaggi di programmazione supportano **strutture di controllo** (come condizionali e cicli), funzioni, variabili, e tipi di dati, che permettono di scrivere programmi modulari e complessi.

CLASSIFICAZIONE

- **Linguaggi di alto livello**: Sono linguaggi progettati per essere comprensibili dagli esseri umani. La loro **sintassi** è **vicina al linguaggio naturale**, rendendo lo sviluppo di software più veloce ed efficiente. Offrono funzionalità come la gestione automatica della memoria e librerie standard per semplificare lo sviluppo. Sono **indipendenti dall'hardware** e necessitano di essere tradotti in linguaggio macchina tramite **un compilatore o un interprete**.

- **Linguaggi di basso livello**: Sono più **vicini al linguaggio macchina** e offrono un maggiore **controllo diretto sull'hardware**. Rispetto ai linguaggi di alto livello, sono più difficili da imparare e utilizzare, ma consentono ottimizzazioni più fini del codice. Sono specifici per ogni architettura e sono eseguiti più velocemente.

PARADIGMI DI PROGRAMMAZIONE

Modello di sviluppo software che definisce come i problemi dovrebbero essere risolti e come i programmi dovrebbero essere strutturati. Esistono diversi paradigmi che influenzano la sintassi e la semantica dei linguaggi di programmazione.

- **Programmazione imperativa**: i programmi consistono in una sequenza di istruzioni che il computer esegue passo dopo passo. Si basa sull'uso di variabili, cicli e strutture di controllo per modificare lo stato del programma (C, C++, Java, Python).

- **Programmazione dichiarativa**: invece di specificare esattamente come risolvere un problema, si descrive il risultato desiderato e il linguaggio determina come raggiungere tale risultato. Si concentra più sul "cosa fare" piuttosto che sul "come farlo" (SQL, Prolog).

- **Programmazione orientata agli oggetti**: i programmi sono organizzati attorno agli oggetti, che rappresentano entità reali o astratte e contengono dati e metodi (Java, C++, Python.).

- **Programmazione procedurale**: organizza il codice in una serie di istruzioni o procedure, spesso chiamate funzioni o subroutine, che vengono eseguite in modo sequenziale per raggiungere un obiettivo o risolvere un problema.

LINGUAGGIO MACCHINA

Il **linguaggio macchina** è la forma più elementare di linguaggio di programmazione. È composto esclusivamente da sequenze binarie di 0 e 1 (bit), che rappresentano istruzioni comprensibili ed eseguibili direttamente dalla CPU. Ogni processore ha il suo set di istruzioni, chiamato **Instruction Set Architecture (ISA)**, che specifica le operazioni che può eseguire, come sommare numeri, caricare dati in memoria o saltare a un'altra istruzione.

Il linguaggio macchina è strettamente legato all'hardware e ogni tipo di processore ha il suo specifico linguaggio macchina. Ad esempio, un programma scritto per un processore Intel non funzionerà direttamente su un processore ARM, poiché utilizzano ISA differenti.

CARATTERISTICHE

- **Codice binario**: Tutte le istruzioni sono rappresentate come combinazioni di 0 e 1.

  10110100 00000001

- **Legato all'hardware**: Il linguaggio macchina dipende dall'architettura del processore. Ogni CPU ha il proprio set di istruzioni.

- **Non leggibile per gli esseri umani**: Il linguaggio macchina è praticamente incomprensibile senza strumenti di analisi specifici. Un programma scritto in linguaggio macchina è difficile da leggere, scrivere e mantenere.

- **Massima efficienza**: Poiché è direttamente eseguibile dalla CPU, non c'è alcun livello di astrazione tra il programma e l'hardware, rendendolo estremamente efficiente in termini di esecuzione.

ESEMPIO

Supponiamo di voler sommare due numeri interi (ad esempio 5 e 3). La CPU eseguirebbe una serie di istruzioni binarie come le seguenti:

- Caricamento del numero 5 in un registro: 10110000 00000101

Questo codice potrebbe significare "carica il valore 5 nel registro A".

- Caricamento del numero 3 in un altro registro: 10110001 00000011

Questo codice potrebbe significare "carica il valore 3 nel registro B".

- Somma dei due registri: 00000011 11000010

Questo codice potrebbe significare "somma i valori dei registri A e B".

LINGUAGGIO ASSEMBLY

Il **linguaggio assembly** è una **rappresentazione testuale** e mnemonica del linguaggio macchina. Permette ai programmatori di scrivere codice in una forma più leggibile rispetto al codice binario, ma rimane comunque molto vicino all'hardware. Ogni istruzione in assembly corrisponde direttamente a un'istruzione in linguaggio macchina.

In assembly, invece di lavorare con numeri binari, si utilizzano abbreviazioni chiamate **mnemonici** che rappresentano operazioni di base. Per esempio, l'istruzione "ADD" potrebbe rappresentare un'operazione di somma, e "MOV" potrebbe rappresentare lo spostamento di dati da una locazione a un'altra.

CARATTERISTICHE

- **Mnemonici**: Invece di scrivere in codice binario, il programmatore usa abbreviazioni leggibili per rappresentare le istruzioni.

- **Relativamente più leggibile**: Sebbene assembly sia più leggibile rispetto al linguaggio macchina, è ancora molto vicino all'hardware e richiede una buona conoscenza dell'architettura del processore.

- **Tradotto da un assemblatore**: Il codice assembly viene tradotto in linguaggio macchina da un programma chiamato assemblatore. Questo permette di trasformare il codice mnemonico in istruzioni binarie eseguibili dalla CPU.

- **Dipendente dall'architettura**: Come il linguaggio macchina, anche il codice assembly è specifico per il tipo di processore e non può essere eseguito su un'architettura diversa senza modifiche.

ESEMPIO

Supponiamo di voler sommare due numeri interi, come nel caso del linguaggio macchina:

- Caricamento del numero 5 in un registro:

```
MOV AX, 5
```

- Caricamento del numero 3 in un altro registro:

```
MOV BX, 3
```

- Somma dei valori nei registri AX e BX:

```
ADD AX, BX
```

- **Risultato**: Dopo l'istruzione ADD, il registro AX conterrà il valore 8, che è la somma di 5 e 3.

DIFFERENZE TRA LINGUAGIO MACCHINA E ASSEMBLY

Caratteristica	Linguaggio Macchina	Linguaggio Assembly
Rappresentazione	Codice binario (sequenze di 0 e 1)	Mnemonici (abbreviazioni testuali)
Leggibilità	Molto difficile per gli esseri umani	Relativamente più leggibile
Esecuzione diretta	Eseguibile direttamente dalla CPU	Deve essere tradotto in linguaggio macchina dall'assemblatore
Astrazione	Nessuna astrazione, massimo controllo sull'hardware	Bassa astrazione, ma più leggibile rispetto al linguaggio macchina
Mantenibilità del codice	Molto difficile da mantenere	Più semplice rispetto al linguaggio macchina, ma comunque complesso

COMPILATORI ED INTERPRETI

Per tradurre il codice sorgente (scritto dal programmatore) in codice eseguibile dalla CPU, sono necessari strumenti che eseguano questa traduzione. Qui entrano in gioco **compilatori e interpreti**, che sono due modi differenti di trasformare il codice sorgente in codice macchina.

Un **compilatore** è un programma che prende in ingresso il codice sorgente di un programma e lo traduce completamente in linguaggio macchina prima che il programma venga eseguito. Questa traduzione crea un file eseguibile, che può essere lanciato in un secondo momento senza la necessità di ritradurre il codice.

Un **interprete** è un programma che legge e interpreta il codice sorgente linea per linea o istruzione per istruzione, traducendo ed eseguendo il codice in tempo reale senza produrre un file eseguibile separato.

DIFFERENZE

Caratteristica	Compilatore	Interprete
Modalità di traduzione	Traduce l'intero programma in linguaggio macchina **prima** dell'esecuzione.	Traduce ed esegue il programma **istruzione per istruzione**.
Tempo di esecuzione	Il codice tradotto viene eseguito in modo indipendente dal compilatore, quindi l'esecuzione è molto veloce.	L'esecuzione è generalmente più lenta, perché ogni istruzione deve essere tradotta al momento dell'esecuzione.
File eseguibile	Genera un file eseguibile	Non genera un file eseguibile. L'interprete deve essere presente ogni volta che si esegue il programma.
Diagnosi di errori	Gli errori vengono rilevati **prima dell'esecuzione**, durante la fase di compilazione.	Gli errori vengono rilevati **durante l'esecuzione**, istruzione per istruzione.
Efficienza	Poiché tutto il codice è tradotto in un'unica volta, i programmi compilati tendono a essere più efficienti in fase di esecuzione.	L'esecuzione può essere più lenta, poiché ogni istruzione viene interpretata al volo.
Esempi di linguaggi	C, C++, Java	Python, Ruby, JavaScript, R

PROCESSO DI COMPILAZIONE

1. **Analisi lessicale (Lexer)**: il compilatore legge il codice sorgente e lo scompone in unità più piccole chiamate **token**. Ogni token rappresenta un simbolo di base del linguaggio, come una parola chiave (ad esempio if, while), un identificatore (come il nome di una variabile), un operatore o un delimitatore (ad esempio ;).

2. **Analisi sintattica (Parser)**: il compilatore verifica se la sequenza di token generata dall'analisi lessicale segue le regole grammaticali (sintassi) del linguaggio di programmazione. Costruisce una struttura ad albero chiamata **albero sintattico o albero di derivazione** che rappresenta la struttura gerarchica del codice.

3. **Analisi semantica**: il compilatore verifica il significato del codice, ovvero se ogni istruzione ha un senso nel contesto del programma. Ad esempio, verifica che le variabili siano dichiarate prima di essere utilizzate, che i tipi di dati siano compatibili e che le operazioni aritmetiche siano applicate correttamente.

PROCESSO DI COMPILAZIONE

4. **Ottimizzazione**: è una fase facoltativa in cui il compilatore cerca di migliorare il codice generato, riducendone la dimensione o migliorandone le prestazioni. L'ottimizzazione può essere applicata sia a livello di codice intermedio (durante la traduzione) sia a livello di codice macchina finale.

5. **Generazione del codice**: il compilatore genera il codice oggetto o codice macchina. Questo è il codice che sarà eseguito direttamente dalla CPU. In questa fase, vengono assegnati indirizzi di memoria alle variabili e le istruzioni vengono tradotte in sequenze binarie.

6. **Linking**: il compilatore collega insieme vari pezzi di codice (come librerie o funzioni definite in file separati) per formare il file eseguibile finale. Questa fase è particolarmente importante per i linguaggi come C e C++, dove il programma può essere suddiviso in più file sorgente.

INTERPRETI

Un **interprete**, a differenza di un compilatore, traduce il codice sorgente linea per linea o istruzione per istruzione, ed esegue immediatamente il codice tradotto senza produrre un file eseguibile. Di conseguenza, un programma interpretato viene eseguito più lentamente rispetto a uno compilato, poiché ogni istruzione deve essere tradotta durante l'esecuzione. Tuttavia, gli interpreti offrono alcuni vantaggi significativi.

Il codice sorgente interpretato può essere eseguito su **qualsiasi piattaforma** che abbia l'interprete appropriato. Non è necessario ricompilare il programma per ogni architettura hardware.

Poiché il codice viene eseguito linea per linea, è possibile interrompere l'esecuzione in qualsiasi momento e correggere eventuali errori senza dover ricompilare tutto il programma. Questo rende l'interprete uno strumento ideale per il debug e la sperimentazione.

INTERPRETI

Non è necessario attendere la fase di compilazione. Gli interpreti sono spesso utilizzati in linguaggi di scripting (come Python e JavaScript) per eseguire rapidamente piccoli programmi o script.

L'esecuzione dei programmi interpretati è più lenta rispetto a quella dei programmi compilati, poiché ogni istruzione viene tradotta ed eseguita al momento.

Per eseguire un programma interpretato, è necessario che l'interprete sia presente sulla macchina. Il codice non può essere eseguito autonomamente.