Linguaggi di programmazione

Sommario

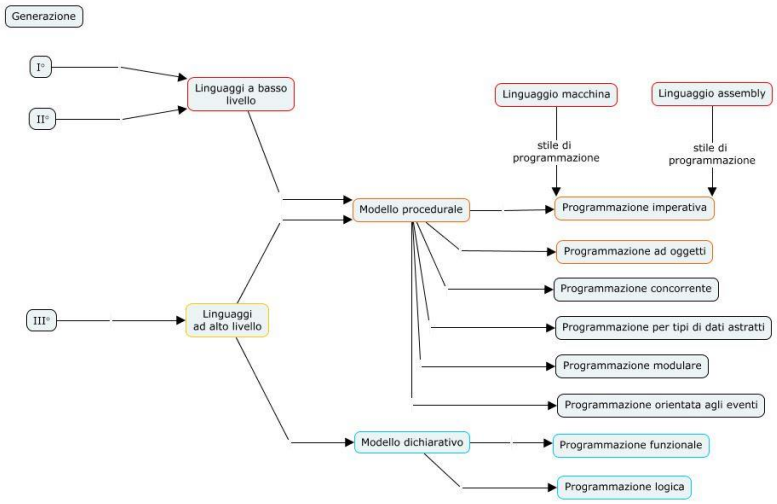
Cos'è un linguaggio di programmazione?

In informatica, insieme di parole e di regole, definite in modo formale, per consentire la programmazione di un elaboratore affinché esegua compiti predeterminati.

Treccani

Ricapitolando i linguaggi di programmazione sono i linguaggi formali di cui si serve il programmatore per impartire delle istruzioni al calcolatore, l'insieme di queste istruzioni si chiama programma e la logica che sottende le operazioni prende il nome di algoritmo.

Classificazione dei linguaggi di programmazione



I linguaggi di programmazione si suddividono principalmente in due categorie: i linguaggi a basso livello e i linguaggi ad alto livello, in queste pagine approfondiremo la seconda categoria.

Linguaggi di basso livello

Sono chiamati così perché sono i linguaggi più vicini alla macchina: le istruzioni impartite con questi linguaggi hanno effetto 1:1 sull'architettura della macchina in cui viene avviato il programma.

Si dividono ulteriormente in:

Linguaggi macchina

I programmi scritti con questi linguaggi sono costruiti direttamente con la base discreta con cui sono stati pensati che nella maggior parte dei casi è quella binaria.

```
Programma in linguaggio macchina

01000100100101001111011010101010010111010101010111110101011110101010101010101010011111111
```

Se vi chiedete cosa faccia questo programma sappiate che io non lo so, a dire il vero non so neanche se sia effettivamente un programma, tanto nessuno saprebbe dirvi che cosa sia esattamente quindi questo è un programma a tutti gli effetti.

Linguaggi assembly

I programmi scritti in linguaggi assembly sono degli insiemi di istruzioni scritti nella forma

```
ISTRUZIONE     VALORE
```

I linguaggi assembly sono considerati i linguaggi più veloci in assoluto (comprensibili) perché il calcolatore non fa alcuno sforzo nella loro conversione in linguaggio macchina perché le istruzioni corrispondono 1:1 a delle architetture fisiche nel centro di calcolo del calcolatore.

```
Programma assembly per INTEL 8086

START:          MOV     AX, BX  ; Carico AX con il contenuto di BX
                CMP     AX, 24  ; Confronto AX con il valore 24 decimale
                JZ      EQUAL   ; Se AX == 24 allora salta a EQUAL
                INT     21h     ; Chiama l'INTERRUPT numero 21 hex (esadecimale)
                RET             ; Ritorna alla procedura chiamante

EQUAL: …
```

In questa sezione ho parlato di linguaggi al plurale perché ogni dispositivo ha il proprio linguaggio macchina e il proprio linguaggio assembly rendendo i programmi non **portabili** su altri calcolatori a meno che non abbiano la stessa architettura hardware.

Linguaggi di alto livello

Sono linguaggi di programmazione più vicini al linguaggio comune, tramite dei simboli e parole chiave permettono ai programmatori trasporre il più naturalmente possibile i concetti al calcolatore, questo ci ha permesso di eliminare la difficoltà di leggere e mantenere il senso di pagine e pagine di codice scritto il linguaggio assembly che è un compito complesso e caotico. L'unico svantaggio di questo tipo di programmazione è il calo delle prestazioni del calcolatore: inevitabilmente il computer deve tradurre questo linguaggio in uno a lui comprensibile e quindi questo occupa tempo e potenza di computazione.

Nel prossimo paragrafo parleremo del processo di esecuzione di un linguaggio ad alto livello.

Dall'idea al file eseguibile

Possiamo dividere la creazione di un programma in linguaggio ad alto livello in 4 fasi principali che vanno dalla sua progettazione fino alla sua esecuzione da parte del calcolatore.

Fase 1° – L'Idea e la progettazione

Questa fase non è direttamente coinvolta processazione del codice ma rimane la parte fondamentale: è importantissimo avere un'idea generale dei limiti e delle necessità di un progetto. In questa fase il programmatore si DEVE occuparsi di scegliere il linguaggio giusto, fare un conto del tempo che dovrà dedicare alla costruzione del codice e di recuperare conoscenze necessarie alla sua realizzazione, perché c'è sempre qualcosa da imparare.

Date una forma al progetto, piccolo o grande che sia, vedrete che costruirlo vi risulterà molto più facile.

Fase 2° - Costruzione del codice

Abbiamo progettato la nostra idea considerando tutti fattori, non ci rimane che iniziare a scrivere il codice!

Per prima cosa dovremo creare uno, o più file, con l'estensione associata al linguaggio, o ai linguaggi, che vogliamo utilizzare, alcuni esempi possono essere

- `.c` per il linguaggio C.
- `.cpp` per il linguaggio C++.
- `.py` per il linguaggio Python .
- `.jar` per il linguaggio JAVA.

L'insieme di questi file viene chiamato codice sorgente e rappresenta il *codice sorgente* del nostro progetto.

Fase 3° - Trasformazione del codice in linguaggio macchina

Inevitabilmente per riuscire ad eseguire il codice dovremo convertire il codice sorgente in un codice comprensibile dalla macchina, il *file eseguibile*, la cui estensione dipende dalla macchina e dal sistema operativo utilizzato.

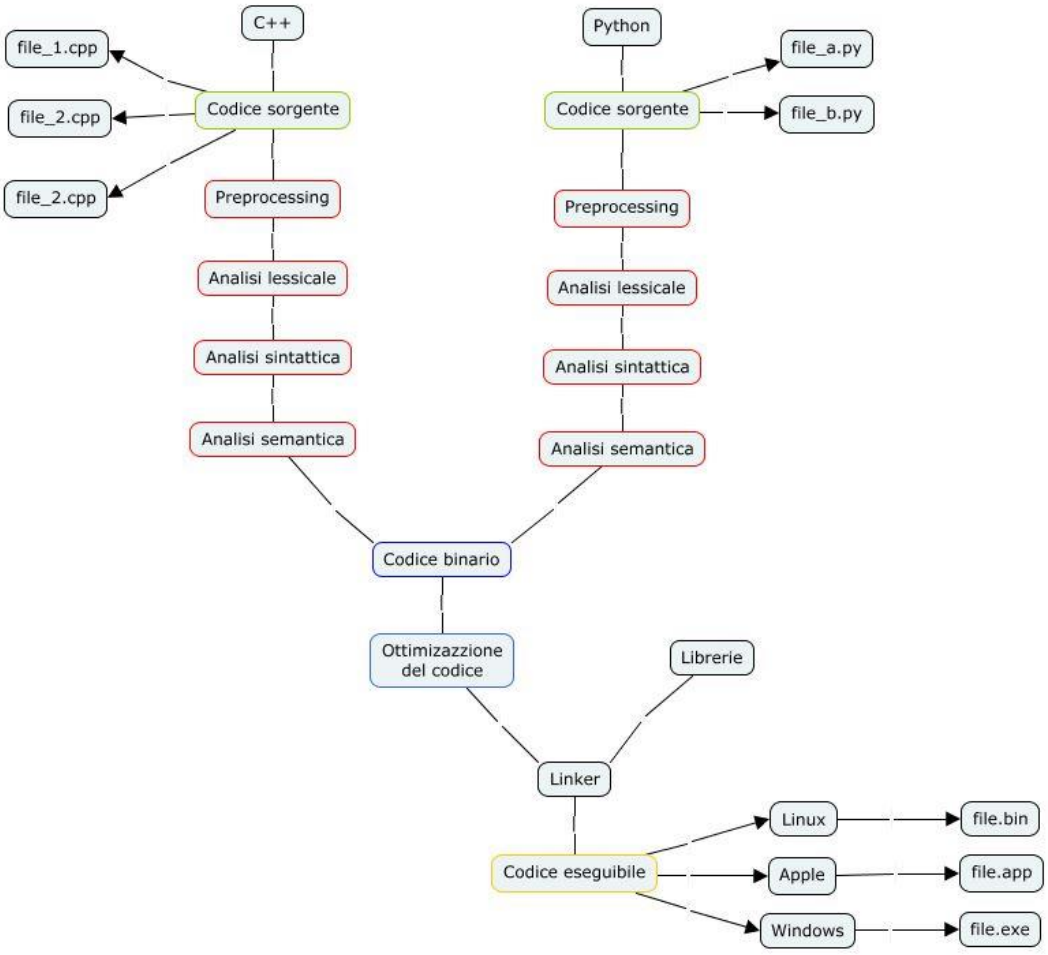
Per eseguire questa traduzione il calcolatore per prima cosa processa tutti i file del progetto tramite la *compilazione* o l'*interpretazione* ed infine utilizzando un programma chiamato *linker* genera il file eseguibile per il sistema operativo scelto.

Per prima cosa presenterò la compilazione e l'interpretazione e successivamente parlerò del linker.

Compilazione

La compilazione si occupa di tradurre un programma da uno a più linguaggi (il codice sorgente) ad un altro specifico (il codice oggetto) solitamente il linguaggio macchina. Questa operazione è eseguita da un programma specifico chiamato compilatore.

La compilazione segue 4 fasi principali



a. Pre-processing

Il pre-processing è un'operazione eseguita dal preprocessore, anche questo un programma, che consiste nel inserire o sostituire delle parti del codice con del codice più complesso.

b. Analisi lessicale

Parte in cui il compilatore legge il codice e ne trova i lessemi, cioè riconosce le figure sintattiche del linguaggio come variabili, funzioni o classi. Nel caso in cui una parte di codice non venga riconosciuta come lessema valido il compilatore si adopererà di indicare l'errore al programmatore in modo che cos' possa risolverlo.

c. Analisi sintattica

Qui il compilatore si preoccupa di controllare che le relazioni fra i vari lessemi abbiano senso per il linguaggio sorgente dividendole in frasi e segnalando le sotto sequenze sbagliate. Tutto questo è fatto dal parser e si svolge senza sapere se queste relazioni hanno senso ad un livello di astrazione generale.

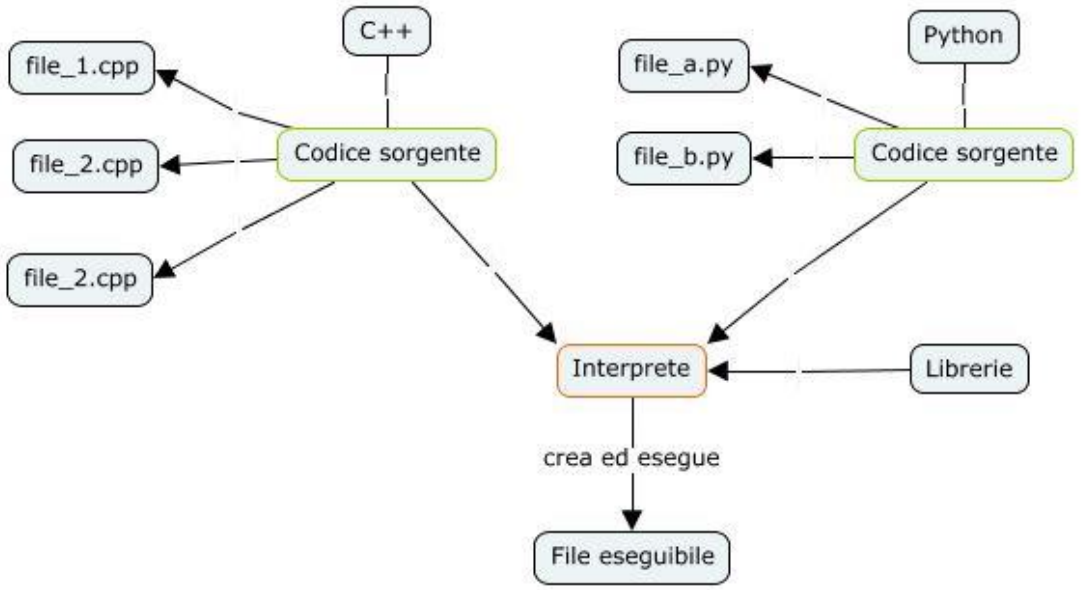
d. Analisi semantica

Controllo finale che si occupa di verificare il significato finale del programma, controllando la coerenza fra i tipi di dato.

Se il codice supera questo processo senza generare errori un programma chiamato linker si occuperà di recuperare le librerie statiche collegate al progetto integrandole insieme al codice sorgente (binario) nel file eseguibile.

Interpretazione

Al contrario della compilazione, che genera un file eseguibile da poter eseguire in un secondo momento, in questo tipo di traduzione il codice sorgente viene gestito per blocchi di codice: il calcolatore traduce il codice sorgente in binario, lo collega alle dovute librerie e lo esegue tutto in run-time generando il file eseguibile e avviandolo. Questa operazione da vita alla *fase di debug* che permette di verificare gli errori statici (analisi lessicale, analisi sintattica e analisi semantica) e quelli di run-time che accadono nel momento in cui si dà il via al flusso di dati.



Questa tecnica ha però anche delle note negative rispetto alla compilazione e sono date principalmente dal calo delle prestazioni: compiere tutte queste operazioni volta per volta richiede una quantità non indifferente di risorse.

Ibridazione dei due tipi di traduzione

Nei giorni nostri è difficile

Fase 4° - Esecuzione del programma

Ho tenuto da parte l'esecuzione del programma

Paradigmi di programmazione dei linguaggi ad alto livello

I linguaggi di alto livello moderni garantiscono varie tecniche di programmazione, detti *paradigmi*, che permettono di costruire programmi sempre più completi e performanti. Queste tecniche sono divise in due modelli fondamentali quello procedurale e quello dichiarativo.

Definizione di paradigma

[…]il termine è stato recentemente introdotto nella sociologia e filosofia della scienza per indicare quel complesso di regole metodologiche, modelli esplicativi, criterî di soluzione di problemi che caratterizza una comunità di scienziati in una fase determinata dell'evoluzione storica della loro disciplina: a mutamenti di paradigma sarebbero in tal senso riconducibili le cosiddette «rivoluzioni scientifiche».

Treccani

Modello procedurale

Il modello procedurale è la concezione classica della scrittura del codice: il calcolatore legge il programma in modo sequenziale "*Eseguo la somma => Moltiplico il risultato per 4 => Visualizzo il risultato in Output*, concepire il programma nello stesso modo (dall'alto verso il basso e da sinistra verso destra) ci permette di avere controllo su tutto il processo nei suoi più piccoli particolari rendendolo estremamente personalizzabile.

1P - Programmazione imperativa – “Qual è la logica migliore per risolvere questo problema? Come la traduco in codice?”

La programmazione imperativa è l’essenza del modello procedurale, una volta visualizzato il problema e costruito le logiche adeguate l programmazione imperativa permette di rappresentarle in codice pezzo per pezzo, riga per riga. Questa è la prima concezione della programmazione mai esistita e costituisce il fondamento di tutte le altre, negli anni si sono sviluppate sintassi sempre più comode ed efficienti che permettono di snellire i processi rendendoli sempre più eleganti.

2P -Programmazione ad oggetti – “Quali caratteristiche appartengono a questa entità? Come le codifico?”

l paradigma OOP (*Object Oriented Programming*) è basato su delle strutture di dati chiamate oggetti composte da parametri e funzioni specifiche chiamate *metodi*. Queste costruzioni interagiscono tra di loro cercando di mantenere nascosto il proprio stato interno e le logiche gestendo i dati in modo conscio e riducendo al massimo gli errori che i programmatori possono commettere mentre lavorano ai propri progetti contemporaneamente.

3P -Programmazione concorrente – “Come posso eseguire più operazioni contemporaneamente?”

La programmazione concorrente si occupa di gestire nel modo più ottimizzato e coerente possibile la concorrenza dei processi, lo scopo è riuscire a rendere contemporanei più compiti possibili ottimizzando così i tempi e le prestazioni.

Per approfondimento vi lascio un esempio che descrive i due problemi maggiori della programmazione concorrente

Cinque filosofi a cena

Cinque filosofi siedono ad una tavola rotonda con un piatto di spaghetti davanti e una forchetta a sinistra. Immaginate che la vita di un filosofo consista di periodi alterni di mangiare e pensare, e che ciascun filosofo abbia bisogno di due forchette per mangiare, ma che le forchette vengano prese una per volta. Dopo essere riuscito a prendere due forchette il filosofo mangia per un po', poi lascia le forchette e ricomincia a pensare. I due problemi che si possono presentare in questo esempio sono lo stallo (*deadlock*), che si verifica se ciascuno dei filosofi tiene in mano una forchetta senza mai riuscire a prendere l'altra, o la morte d'inedia (*starvation*), cioè il filosofo non riesce mai ad accedere a due forchette per mangiare.

4P -Programmazione di tipi astratti – “Che cos’hanno in comune le entità?”

Lo scopo di questa tecnica è di creare delle sovrastrutture logiche da cui le classi “concrete” possano ereditarne le caratteristiche. Come esempio possiamo prendere il gatto ed il cane: entrambi camminano a quattro zampe ma non li identifichiamo per questa caratteristica perché molti animali sono accomunati da questa funzione ma non ne vengono rappresentati, la programmazione dei tipi astratti si occupa di trovare queste comunanze cercando di renderle il più generali possibili. Le strutture astratte possono essere sovrastrutture logiche degli oggetti di cui abbiamo parlato nella programmazione ad oggetti.

5P -Programmazione modulare – “Come organizzo il codice in modo che sia sempre chiaro?”

La programmazione modulare si occupa di organizzare il codice generato dagli altri tipi di programmazione all’interno del progetto. Lo scopo è la struttura: più il codice è organizzato e più la sua estensione sarà semplice perché si saprà sempre dov’è il suo posto creando familiarità al programmatore. Questa disciplina è dedicata a semplificare lo sviluppo, il test e la manutenzione di programmi di grosse dimensioni.

6P -Programmazione orientata agli eventi – “Quali eventi sono utili per il programma?”

Questo paradigma si basa su un unico ciclo fondamentale che si occupa di verificare tutte le variazioni dei codici e a cui a delle condizioni particolari (*eventi*) si associano delle funzioni specifiche. Un esempio di programmazione orientata agli eventi può essere lo sviluppo web: quando si clicca un bottone sia apre una finestra di dialogo (gestito da JavaScript e non dall’HTML).

Modello dichiarativo

I programmi sviluppati su questo modello (se così possiamo chiamarli) definiscono in modo esplicito soltanto lo scopo da raggiungere, lasciando che l'implementazione dell'algoritmo sia realizzata dal linguaggio, o software, che stiamo utilizzando.

Si divide ulteriormente in 2 tipi di programmazione:

1D - Programmazione logica

La programmazione logica, a differenza di quella imperativa, permette al programmatore di concentrarsi sulla struttura logica del problema piuttosto che della sua implementazione.

Per esempio, un'istruzione

```
SELECT * FROM Table
```

di SQL specifica le proprietà dei dati che devono essere estratti da un database, ma NON i dettagli del processo di estrazione vero e proprio che potremmo analizzare con l'occhio della programmazione imperativa.

2D - Programmazione funzionale

Nella programmazione funzionale non si ragiona utilizzando dei principi logici (*se, mentre, quando …*) ma piuttosto sui dei modelli matematici che trasformino il flusso delle informazioni, quindi non più delle variabili come "caselle" ma come flussi di informazioni.

Programmare diventa un concatenare diverse funzioni che per loro stessa definizione andranno a modellare i dati. Questo tipo di programmazione è anche chiamata programmazione a stati.