ALGORITMI LINGUAGGI E PROGRAMMI

1. Algoritmi e programmi

1.1 Il programma e l'algoritmo

Per arrivare a definire il concetto di algoritmo partiamo da quello di "**programma**". Il significato di questo termine legato al funzionamento di un computer. Essendo il computer una macchina non intelligente, ma un mero esecutore di ordini impartiti dall'esterno, necessita di una serie di istruzioni, scritte in un opportuno codice detto **linguaggio di programmazione**, che costituiscono, appunto, il programma. Prima però di poter scrivere un programma, il programmatore deve analizzare il problema da risolvere, definire gli obiettivi da raggiungere, individuare i dati iniziali e finali e descrivere tutti i passi necessari per ottenere il risultato voluto. Affinché il computer sia in grado di eseguire con successo i compiti ad esso assegnati, è necessario che la descrizione del procedimento sia accurata; è opportuno che alla macchina non venga indicato come risolvere un singolo problema, ma tutta una classe di problemi che differiscono per i dati iniziali. Tutto ciò sottende al concetto di **algoritmo.**

Il termine "**algoritmo**" prende la sua origine dal nome del matematico arabo Al-Khwarizmi, vissuto nell'800 d.C., ritenuto, storicamente, l'ideatore del procedimento che consente di effettuare il calcolo della moltiplicazione tra due numeri tramite l'incolonnamento delle cifre (lo stesso algoritmo che utilizziamo ancora oggi).

Nel senso più ampio della parola, un "algoritmo" è una sequenza finita di operazioni, come ad esempio una ricetta di cucina o le istruzioni di funzionamento di una lavatrice.

In informatica, con il termine algoritmo si intende: ***una sequenza ordinata e finita di passi, ripetibili e non ambigui, che se eseguita con determinati dati in ingresso (input), produce in uscita (output) un risultato ovvero la soluzione di una classe di problemi.***

Un algoritmo, per essere tale, deve avere le seguenti proprietà:

- essere **finito**: la sequenza di istruzioni deve essere finita e portare ad un risultato;

- essere **eseguibile**: le istruzioni devono poter essere eseguite materialmente dall'esecutore;
- essere **non ambiguo**: le istruzioni devono essere espresse in modo tale da essere interpretate da tutti allo stesso modo;
- essere **generale**: deve essere valido non solo per un particolare problema, ma per una classe di problemi;
- essere **deterministico**: partendo dagli stessi dati iniziali deve portare sempre allo stesso risultato finale indipendentemente dall'esecutore;
- essere **completo**: deve contemplare tutti i casi che si possono verificare durante l'esecuzione.

Gli algoritmi possono venire classificati in:

- algoritmi **deterministici**: se per ogni istruzione esiste, a parità di dati d'ingresso, un solo passo successivo; in pratica esiste uno e un solo possibile percorso dell'algoritmo e quindi a fronte degli stessi dati di partenza produrrà gli stessi risultati.
- algoritmi **non deterministici**: se contiene almeno un'istruzione che ammette più passi successivi che hanno la possibilità di essere scelti: l'algoritmo potrà produrre risultati diversi a partire da uno stesso insieme di dati compiendo percorsi diversi

Tra gli algoritmi non deterministici troviamo quelli **probabilistici** nei quali almeno un'istruzione ammette più passi successivi, ognuno dei quali ha una certa probabilità di essere scelto.

2. Rappresentazione degli Algoritmi

Affinché un algoritmo sia interpretato ed eseguito correttamente da un'altra persona o da una macchina, è necessario rappresentare i comandi (o **istruzioni**) che lo compongono in modo preciso e non ambiguo.

Considerando che l'algoritmo da dare al computer subirà un successivo passaggio (**codifica**) che lo convertirà da algoritmo a programma, in fase di progettazione del procedimento risolutivo, è importante utilizzare un linguaggio appropriato, che rappresenti in modo efficace l'algoritmo e ne faciliti la successiva traduzione.

Per soddisfare questo requisito, abbiamo due opzioni per descrivere un algoritmo:

- **Diagramma di Flusso** o **flow-chart**
- **Pseudolinguaggio** di programmazione o linguaggio di **pseudo codifica**

2.1 La pseudo codifica

Per descrivere algoritmi in modo chiaro, ordinato e corretto si fa uso del linguaggio di pseudo codifica, in cui si devono effettuare in ordine le seguenti fasi:

- **Intestazione:** riga iniziale che assegna un nome all'algoritmo. Si usa la parola ***Algoritmo*** seguita da un nome a piacere che lo identifica. Ad esempio, supponiamo di volere effettuare la somma di 2 numeri, un nome appropriato per l'algoritmo può essere ***Algoritmo Somma***
- **Fase dichiarativa:** prima di iniziare l'algoritmo è necessario dichiarare i dati che entrano in gioco specificando il nome e il tipo di dato. I dati si distinguono in dati **variabili** e dati **costanti**. Si usa la parola ***dichiara*** seguita dal nome o nomi del dato e dal loro tipo. Nel caso dell'algoritmo per la somma si ha:

  Variabili

  Dichiara X, Y somma come numeri interi

- **Fase esecutiva:** in cui si eseguono in ordine le istruzioni che partendo dai dati di input ci permettono di ottenere i risultati in output. Questa fase inizia con la parola chiave ***inizio*** e termina sempre con la parola chiave ***fine***.

Nella fase esecutiva si devono effettuare nell'ordine le seguenti azioni:

1. **Acquisizione dei dati:** si utilizza la parola chiave ***leggi*** seguita dal nome del dato che deve essere letto. Per il nostro esempio si ha:

   ***Leggi* X**

   ***Leggi* Y**

2. **Assegnazione o Calcolo**: si utilizza la parola chiave ***Calcola*** seguita dall'istruzione da eseguire. Per l'esempio in questione si ha:

   ***Calcola* Somma = X + Y**

3. **Scrittura dei risultati:** si utilizza la parola chiave ***Scrivi*** seguita del nome della variabile che deve essere scritta. Per l'esempio in questione si ha

   ***Scrivi* Somma**

Quindi l'algoritmo completo sarà:

***Algoritmo* Somma**

Variabili

Dichiara X, Y somma come numeri interi

Inizio

***Leggi* X**

***Leggi* Y**

Calcola Somma = X+Y

***Scrivi* Somma**

Fine

2.2 Diagramma di flusso

Un diagramma di flusso è una rappresentazione grafica di un algoritmo. In tale rappresentazione ogni passo è descritto attraverso l'utilizzo di una serie di simboli standard ciascuno avente un significato ben specifico.

In particolare il diagramma consente di visualizzare:

- Le operazioni da compiere: rappresentate da forme quali ovali, rettangoli etc…
- La sequenza in cui compiere tali operazioni, rappresentata da frecce.

Nella Tabella 1 sono elencati gli elementi propri di un diagramma di flusso.

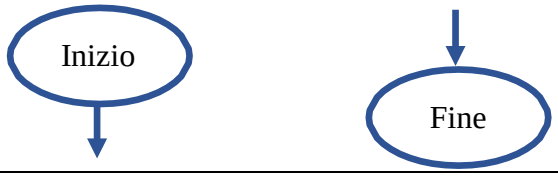
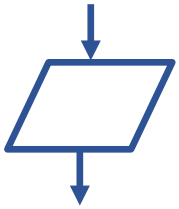
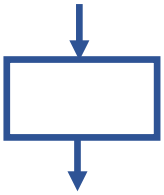
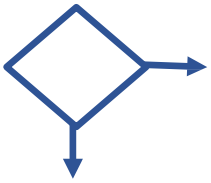
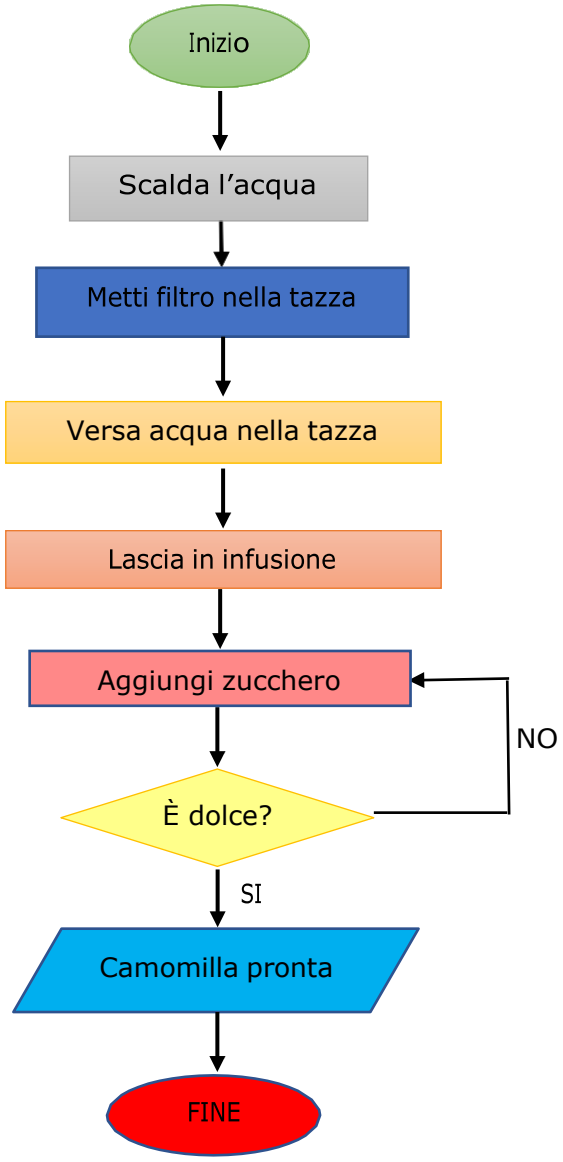
Simbolo	***Significato***
	**Inizio** e **fine** della sequenza di istruzioni
	**Inserimento** ed **emissione** dei dati Istruzioni di Lettura o Scrittura
	Istruzione da eseguire
	Istruzione che implica una scelta tra due possibili percorsi a seconda della valutazione di una certa condizione

Tabella 1: Simboli diagramma di flusso

- Facciamo un esempio: pensiamo ai passi da fare quando prepariamo una camomilla:
  1. scalda l'acqua;
  2. sistema un filtro in una tazza;
  3. versa nella tazza l'acqua calda;
  4. lascia in infusione per qualche minuto;
  5. aggiungi zucchero.

Proviamo a disegnare il relativo diagramma di flusso:



3. Linguaggi di programmazione

3.1 Introduzione

Ciò che da sempre spinge l'uomo alla progettazione di nuove macchine è l'esigenza di facilitare il proprio lavoro.

Il computer è stato progettato inizialmente per sostituire l'umo nella soluzione di problemi numerici complessi ma, con l'introduzione dei calcolatori *general-purpose* (capaci di risolvere un'ampia classe di problemi) nasce l'esigenza di "guidarli" nella risoluzione dei problemi per mezzo di istruzioni appartenenti ad un linguaggio a loro comprensibile: nasce la **programmazione**.

Con la nascita della programmazione, sorge l'esigenza di creare degli strumenti che consentano la comunicazione uomo-macchina: **i linguaggi di programmazione**.

Un linguaggio di programmazione è costituito da un alfabeto di simboli con cui vengono costruite le parole e le frasi e da un insieme di regole lessicali e sintattiche per l'uso corretto delle parole.

In particolare, i linguaggi di programmazione sono analoghi ai linguaggi naturali, come questi infatti sono caratterizzati dall'avere:

- **Lessico:** insieme di regole formali per la scrittura di parole in un linguaggio;
- **Sintassi:** insieme di regole formali per la scrittura di frasi in un linguaggio che, stabiliscono la grammatica del linguaggio stesso;
- **Semantica:** insieme di significati da attribuire alle frasi, sintatticamente corrette, costruite nel linguaggio.

3.2 Classificazione dei linguaggi di programmazione

La classificazione dei linguaggi di programmazione è utile al fine di evidenziare le caratteristiche di ciascun linguaggio così da capire se un linguaggio è più idoneo a risolvere un problema rispetto ad un altro.

Le classificazioni più utilizzate sono:

- Classificazione per paradigma (approccio alla soluzione del problema)
- Classificazione per formato del codice
- Classificazione per livello

Uno stesso linguaggio può essere inquadrato contemporaneamente in più classificazioni

In questa discussione analizzeremo la classificazione per livello.

Nella classificazione per livello, i linguaggi vengono suddivisi in:

- Linguaggio macchina
- Linguaggi di basso livello
- Linguaggi di alto livello

3.2.1 Il linguaggio macchina

Il **Linguaggio macchina** è quel linguaggio di programmazione strettamente dipendente dall'hardware, utilizzato per eseguire le istruzioni direttamente da un **processore** o **CPU** di un computer.

Ogni istruzione compilata seguendo le regole di questo codice potrà eseguire solamente **un'operazione ben precisa** come ad esempio:

- Scrivere dati in una cella di memoria;

- Saltare una cella di memoria e passare alla successiva;
- Svolgere un'operazione logico-aritmetica, etc…

Essendo un linguaggio strettamente legato all'architettura del calcolatore, ha come grande svantaggio, quello che un programma scritto in linguaggio macchina per un'architettura, non funziona in un'altra architettura.

3.2.2 Il linguaggio di basso livello

Per ovviare alle problematiche presentate dal linguaggio macchina, sono stati introdotti i linguaggi di basso livello che risultano essere comunque linguaggi orientati alla macchina ma con un linguaggio più elementare.

Un esempio di linguaggio a basso livello è il **linguaggio assembly**.

Tale linguaggio consente al programmatore di ignorare il formato binario del linguaggio macchina, sostituendo ogni codice operativo del linguaggio macchina con una sequenza di caratteri che lo rappresenta in forma mnemonica; per esempio, il codice operativo per la somma potrebbe essere trascritto come ADD e quello per il salto come JMP.

Dal momento che le istruzioni sono molto semplici, per raggiungere un buon grado di astrazione è necessaria una grossa mole di istruzioni, che rendono il programma molto lungo e di difficile comprensione per un programmatore; per questo motivo, in un programma a basso livello è molto più difficile effettuare i controlli sugli errori e la manutenzione del codice.

3.2.3 Il linguaggio di alto livello

Un linguaggio di alto livello è un linguaggio più vicino al linguaggio umano che permette al programmatore di scrivere programmi che sono più o meno indipendenti da un particolare tipo di computer.

Poiché un programma scritto in un qualsiasi linguaggio ad alto livello è molto più vicino all'uomo che alla macchina, per essere compreso da quest'ultima deve essere "tradotto" in linguaggio macchina.

Per fare la "traduzione" si utilizzano i seguenti programmi:

- **Compilatore:** traduce l'intero codice sorgente di un programma nel linguaggio macchina prima che sia eseguito.
- **Interprete:** elabora il codice sorgente di un programma mentre questo è in esecuzione, e funge da interfaccia tra quel programma e il processore. A differenza del compilatore, l'interprete esamina un'istruzione alla volta, realizzandone la traduzione e l'esecuzione.

Alcuni linguaggi ad altro livello più conosciuti sono Java, PHP, C, C++ e così via.